Distributed Shared Memory Architectures and Global Performance State Estimation

A Thesis
Submitted to the Office of Graduate Studies
of
The University of Dublin
Trinity College
in Candidacy for the Degree of
Doctor of Philosophy

by
Michael Manzke

June, 2006

Declaration:

This thesis has not been submitted as an exercise for a degree at this or any other University. Furthermore this thesis is entirely my own work and I agree that the Library may lend or copy the thesis upon request. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

Michael Manzke

Abstract

Invasive and non-invasive methods may be applied to measure and analyse the performance of hardware Distributed Shared Memory (DSM) systems. This thesis presents novel solutions for both methods and discusses the architectural organisation of loosely and tightly coupled systems. The work begins with a discussion of the design and implementation of a non-invasive deep-trace instrument for high-speed interconnects and also deals with the analysis of the trace-data. Analysis results are used to tune interconnect simulations.

This thesis then presents an innovative invasive approach to estimate and predict the system-wide utilisation of computational resources in real-time. An algorithm that implements a discrete minimum mean-square error filter is applied to fuse concurrent and sequential observations of system event counts into a state-vector. Contemporary computer components and subsystems make these event counts available through hardware Performance Monitoring Counter (PMC) registers. The registers may be accessed by the system's software quasi-concurrently but the number of registers in individual components is usually smaller than the number of events that can be monitored. This approach overcomes the problem by modelling individual PMC readings as vector random processes and recursively processes them one PMC set at a time into a common state-vector, thereby making larger PMC sets observable than would otherwise be possible.

Finally this work looks at loosely and tightly coupled hardware DSM systems as targets for the estimation algorithm. Particular attention is paid to the conceptual design of a tightly coupled hybrid reconfigurable DSM graphics cluster.

Acknowledgments

My sincerest thanks and gratitude goes to Dr. Brian Coghlan and Prof. John Byrne. I thank Brian for taking me on as research assistant and postgraduate student on the SCI Europe project. At that time, I had spent a decade as an engineer in various positions in industry, subsequent to the completion of my engineering degree. Brian introduced me to many aspects of computer architecture and provided guidance for my PhD. He also gave me the freedom to pursue my own research interest. Prof. Byrne, who was at the time Head of Department, offered me the opportunity to prove myself as a lecturer. The income arising from this position allowed me the continue with my PhD. I could not have achieved this without them.

My children Denis, Rachel and Oscar and my wife Carol deserve a special mention here. I thank my children for coping with their parent's very busy life style and Carol for looking after the family when I was not available. Carol is not only an outstanding researcher; she also enables me to pursue my academic career. I appreciate very much what Denis, Rachel, Oscar and Carol have done for me to make this possible and to Carol a special thanks for the many extra hours and weekends that I could dedicate to the completion of my PhD. I could not have done this without her support.

I would also like to thank my postgraduate students Ross Brennan, Eoin Creedon and Muiris Woulfe for all the support with the teaching, particularly the demonstrating and the marking of assignments.

The reviewers of my PhD-related publications deserve a thank you for their valuable feedback.

Furthermore I would like to thank the Head of Department Prof. Jane Grimson, the Head of School, Dr. David Abrahamson, and all my colleagues (there are too many to mention them all) for making our department such a nice and supportive environment. I should specifically mention the chief technician Tom Kearney. Tom and his group are an incredible help with my teaching and research activities.

I should also mention the many organisations I have worked for either directly or indirectly. I have learned so much from my colleagues during my time in the following organisations: Navy, Philips (Germany), Dow Chemical (Germany), Saronix (Ireland), Advance (USA), Siemens/Nixdorf (Ireland), Schering-Plough (Ireland), Industrial Design Corpora-

tion (Ireland, USA), Intel (Ireland), Motorola (USA). I would like to mention in particular Detlev Bock and Hilger Walter from Dow Chemical who introduced me to advanced process control and Kalman filtering during my time at Dow.

Initial funding for my PhD came from the SCI Europe Esprit project 25257. It was during this project that I developed a collaboration with Dolphin Interconnect Solutions Inc. that has lasted many years. Dolphin has not only donated hardware but also provided me with significant engineering support for the graphics cluster nodes. I'd like to thank Kåre Løchsen , Hugo Kohmann , Nils Jørgen Kjærnet, Tor Undheim, Roy Nordstrøm and Svein Erik Johansen for their support. I am also grateful to Patrick Lysaght, Phil Roxby and Brendan Cremen from Xilinx Inc. for their Field Programmeable Gate Array (FPGA) donation and to Mike Doggett from ATI for his help with the Graphics Processing Unit (GPU) driver software.

Contents

List of Figures

List of Tables

Nomenclature

β_i Time constant for a particular PMC process, see equation (5.20), page 70

Δt Sample interval, see equation (2.1), page 35

Δt_{min} Minimum sample interval, see equation (2.1), page 35

$\frac{1}{\beta}$ Time constant in exponential autocorrelation function for Gauss-Markov process, see equation (5.15), page 69

$\hat{R}_X(\tau)$ Estimated autocorrelation function, see equation (5.25), page 71

$\hat{R}_X(n\Delta t)$ Discrete estimated autocorrelation function, see equation (5.25), page 71

σ^2 Variance in exponential autocorrelation function for Gauss-Markov process, see equation (5.15), page 69

σ_i^2 Variance for a particular PMC process, see equation (5.20), page 70

τ Autocorrelation time difference variable, see equation (5.25), page 71

$e(t)$ Mean of the counted events $e(t)$ over the sample interval Δt, see equation (2.1), page 35

$R_X(\tau)$ Autocorrelation, see equation (5.15), page 69

T Autocorrelation time interval, see equation (5.25), page 71

t_k Sample time, see equation (5.2), page 65

$u(t)$ Unity white noise in continuous state space model, see equation (5.16), page 69

$X(t)$ Stationary Gaussian process, see equation (5.14), page 69

x_1 Integrated Gauss-Markov process in continuous state space model, see equation (5.16), page 69

x_2 Gauss-Markov process in continuous state space model, see equation (5.16), page 69

z_k Performance Monitoring Counter register readings, see equation (2.1), page 35

$\hat{\boldsymbol{x}}_k^-$ A Priori state vector, see equation (5.14), page 68

$\hat{\boldsymbol{x}}_k$ Estimated state vector, see equation (5.2), page 65

$\boldsymbol{\phi}_k$ State transitions matrix, see equation (5.2), page 65

$\boldsymbol{e}_k^- =$ Estimation error, see equation (5.5), page 67

$\boldsymbol{H}$ Measurement sensitivity matrix, see equation (5.2), page 65

$\boldsymbol{K}_k$ Kalman Gain, see equation (5.2), page 65

$\boldsymbol{P}_k$ A Posteriori error covariance matrix, see equation (5.8), page 67

$\boldsymbol{P}_k^-$ A Priori error covariance matrix of the estimated state vector $\hat{\boldsymbol{x}}_k^-$, see equation (5.8), page 67

$\boldsymbol{Q}_k$ Process noise covariants matrix, see equation (5.2), page 65

$\boldsymbol{R}_k$ Measurement noise covariance matrix, see equation (5.5), page 67

$\boldsymbol{v}_k$ Measurement noise is described by the covariance matrix $\boldsymbol{R}_k$, see equation (5.3), page 66

$\boldsymbol{w}_k$ Sequence with a covariance determined by the covariants matrix $\boldsymbol{Q}_k$ of the process noise associated with the system's state dynamics, see equation (5.2), page 65

$\boldsymbol{x}_k$ State vector of the linear dynamic system at sample time t_k, see equation (5.2), page 65

List of Abbreviations

AGP Accelerated Graphics Port

AKF Adaptive Kalman Filter

AMBA Advanced Microcontroller Bus Architecture

API Application Programming Interface

ASIC Application Specific Integrated Circuits

ATM Asynchronous Transfer Mode

Blink Backside Link

C^{2}STATE Compute Cluster State Estimation

CCA Common Component Architecture

ccNUMA cache-coherent None-Uniform Memory Access

CPU Central Processing Unit

DCU Data Cache Unit

DDR Double Data Rate

DIRA Divided-Interval Rectangular Area

DMA Direct Memory Access

DSM Distributed Shared Memory

DSP Digital Signal Processing

DVI Digital Visual Interface

DVS Dynamic Voltage Scaling

EBL External Bus Logic

EISA Extended Industry Standard Architecture

EKF Extended Kalman Filter

ESA European Space Agency

FDDI Fiber Distributed Data Interface

FIFO First In First Out

FPGA Field Programmeable Gate Array

FSB Front-Side Bus

GALS Globally Asynchronous Locally Synchronous

gcc GNU Compiler Collection

GIDS Grid-wide Intrusion Detection System

GNU GNU's Not Unix

GPS Global Positioning System

GPU Graphics Processing Unit

GUI Graphical User Interface

HDL Hardware Description Languages

I/O Input/Output

IC Integrated Circuits

IEEE Institute of Electrical and Electronics Engineers

IFU Instruction Fetch Unit

IKF Interval Kalman Filter

ILP Instruction Level Parallelism

IPC Instructions Per Cycle

IQ Issue Queue

KVM Keyboard Video Mouse

L1 Level 1 Cache

L2 Level 2 Cache

LC2 Link Controller 2

LC3 Link Controller 3

LC Link Controller

LSQ Load/Store Queue

LVDS Low Voltage Differential Signalling

MCD Multiple Clock Domain

MDL Metric Description Language

MESI Modified Exclusive Shared Invalid

MIT Massachusetts Institute of Technology

MLR Multiple Linear Regression

MMSE Minimum Mean-Square Error

MPI Message Passing Interface

MPP Massively Parallel Processing

NFS Network File System

NUMA Non-Uniform Memory Access

ODBC Open DataBase Connectivity

OS Operating System

PAL Programmable Array Logic

PCB Printed Circuit Board

PCI Peripheral Component Interconnect

PCL Performance Counter Library

PC Personal Computer

PDF Probability Density Function

PDT Program Database Toolkit

PIO Programmed Input Output

PI Principal Investigator

PLB Pipeline Balancing

PMC Performance Monitoring Counter

PME Positional Mean Error

PVM Parallel Virtual Machine

ROB Reorder Buffer

SANTA System Area Network Trace Analysis

SCAAT Single-Constraint-At-A-Time

SCI Scalable Coherent Interface

SFI Science Foundation Ireland

SISCI Software Infrastructure for Scalable Coherent Interface

SMP Symmetric Multiprocessor

SQL Structured Query Language

SRAM Static Random Access Memory

TAM Trapezoid-area Method

TCD Trinity College Dublin

TSC Time-Stamp Counter

VHDL (VHSIC) Hardware Description Language

VHSIC Very High Speed Integrated Circuit

VRAM Video Random Access Memory

Chapter 1

Introduction

This thesis is concerned with the architectural organisation, the performance measurement and the real-time performance estimation of hardware DSM clusters. These machines may incorporate various subsystems, e.g. Central Processing Unit (CPU)s, GPUs, Chip-sets, FPGAs and interconnect interfaces. It is the objective of the performance estimation to provide a global view of the computational state of all these subsystems throughout the cluster. In order to facilitate real-time performance estimation research and investigations into novel high performance hybrid graphics cluster architectures, a general-purpose testbed cluster was built from commodity components and a second special-purpose hybrid system was designed. Both systems employ the Institute of Electrical and Electronics Engineers (IEEE) 1596-1992 Scalable Coherent Interface (SCI) [Ins93] as interconnect. This technology provides DSM in hardware and allows for the configuration of Non-Uniform Memory Access (NUMA) and cache-coherent None-Uniform Memory Access (ccNUMA) clusters.

1.1 Non-Invasive Trace Data Acquisition of SCI Hardware DSM Interconnect Traffic

Initial investigations with Dr. Brian Coghlan into the non-intrusive acquisition of SCI interconnect traffic [MC99a] provided an understanding of their true communication statistics. This work required the design and construction of hardware and software for the non-invasive acquisition of interconnect traffic at real-time and the subsequent off-line trace data analysis. This infrastructure is shown in Fig. 2.2.

A collaboration with Dr. Brian Coghlan and Stuart Kenny from Trinity College Dublin, and Dr. Olav Lysne from the University of Oslo, investigated how these trace data statistics could be used to tune and verify high speed interconnects such as SCI [MKCL01]. The results of the trace data analysis were used to successfully tune an OPNET [Opn06] simulation model for SCI interconnect topologies.

1.2 Global Real-time Estimation of Incomplete Performance Measurements

Subsequent work focused on a novel approach to estimating and predicting the DSM system-wide utilisation of computational resources in real-time [MC05a]. An algorithm that implements a Discrete Minimum Mean-Square Error (MMSE) Filter is applied to fuse concurrent and sequential observations of system event counts into the filter's state vector. Contemporary computer components and subsystems make these event counts available through hardware PMC registers. The registers may be accessed by the system's software quasi-concurrently [1] but the number of registers in individual components is usually smaller than the number of events that can be monitored. This approach overcomes this problem by modelling individual hardware PMC readings as vector random processes and recursively processes them one (or a group) at a time into a common state vector, thereby making larger performance counter sets observable than would otherwise be possible.

This algorithm can be applied to fuse various PMCs from a single node or a set of compute cluster nodes. The memory hierarchy of a hardware DSM cluster is in this context of particular interest because load and store operations trigger PMC events that indicate cache and memory activity. Furthermore, a memory reference made to a remote memory causes counter events on the DSM's interconnect interface cards. The algorithm allows us to merge all this information into a common state vector. Consequently this approach allows us to observe the entire system state at real-time from any node in the system.

1.3 Special Purpose High Performance Graphics DSM Cluster

Work on the non-invasive measurement and analysis of high speed interconnect traffic and investigations into the global state estimation of compute clusters led to the conceptual design of a special-purpose high-performance graphics cluster [MBO+06]. Section 7.2 of this thesis describes the design of this scalable tightly coupled cluster of custom-built boards that provide an Accelerated Graphics Port (AGP) interface for commodity graphics accelerators. These boards are supplied with rendering instructions by a cluster of commodity Personal Computer (PC)s that execute OpenGL graphics applications. All the commodity PCs and custom-built boards are interconnected with an implementation of the SCI standard. This technology provides the system with a high bandwidth, low latency, point to point interconnect. The design allows for the implementation of a 2D torus topology with good scalability properties and excellent suitability for parallel rendering. Most importantly the interconnect implements a DSM architecture in hardware. Figure 7.7 shows how local

[1] The reading of several PMC registers is considered as a concurrent operation because the amount of time consumed by this operation is much smaller than the *counter sample time* Δt ($\sum_{k=1}^{n} t_{load\ k} \ll \Delta t_{sample}$).

memories on the custom-built boards and the PCs become part of the system wide DSM. Figure 7.7 also depicts FPGAs on the custom-built boards. These reconfigurable components assist the SCI implementation and provide substantial additional computational resources that may be used to control the commodity graphics accelerators and to perform operations associated with a parallel rendering infrastructure, or even ray tracing and physics simulations. These reconfigurable components are an integral part of the scalable shared-memory graphics cluster and consequently, increase the programmability of the parallel rendering system just like vertex and pixel shaders increased programmability of graphics pipelines. The implementation and investigation into the opportunities provided by this design will be investigated as future work. The global performance estimation algorithm will be integrated into the graphics cluster to investigate its suitability for load balancing.

1.4 Contributions

The main contribution is this work is the global real-time performance state estimation of DSM clusters. To the best of my knowledge nobody has investigated the suitability of MMSE Filters to observe a large number of PMC readings that would otherwise be unobservable. Further contributions are the non-invasive acquisition and analysis of SCI interconnect traffic and the application of interconnect traffic statistics to tune network simulations. All this work was published in international peer reviewed conferences [MC05a, MKCL01, MC99a] and a list is provided in section 1.4.2.

My conceptual design of the graphics cluster is the predominant part of a successful Science Foundation Ireland (SFI) Basic Research Grant proposal [OMK04] written in collaboration with the Principal Investigator (PI) Dr. Carol O'Sullivan and Dr. Anil Kokaram, both of Trinity College Dublin (TCD). The following three quotations are from the rigorous peer review:

> A well written and well thought through proposal. The research is very far-sighted, and considerable ingenuity has been applied to obtain a cost-effective solution to the graphics speed problem.
>
> If successful, the project outcomes would make an important contribution to the field.
>
> Overall though this is a very strong, well thought out proposal, based on a very good and novel idea.

This work was accepted for publication as a SIGGRAPH work in progress sketch [MBO+06].

Some work that is not directly related to this thesis has influenced the design of the special purpose high performance DSM graphics cluster. This applies mostly to FPGAs and the SCI technology [MB04, BM03]. These publications are shown in section 1.4.3.

1.4.1 Thesis Statement

Data fusion of sequential and concurrent Performance Monitoring Counter (PMC) readings with Minimum Mean-square Error (MMSE) algorithms allows for the estimation of the computational state of Distributed Shared Memory (DSM) clusters. These PMC readings could otherwise not be observed with a comparable accuracy.

1.4.2 Directly Relevant Peer Reviewed Publications

[MBO+06] Michael Manzke, Ross Brennan, Keith O'Conor, John Dingliana, and Carol O'Sullivan. A scalable and reconfigurable shared-memory graphics architecture. In *Proceedings of the SIGGRAPH 2006 Conference on Sketches & Applications*, 2006

[MC05a] Michael Manzke and Brian A. Coghlan. Optimal performance state estimation of compute systems. In *the Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005)*, pages 511–516, September 2005

[MKCL01] Michael Manzke, Stuart Kenny, Brian Coghlan, and Olav Lysne. Tuning and verification of simulation models for high speed interconnection. In *PDPTA'2001*, June 2001

[MC99a] Michael Manzke and Brian Coghlan. Non-intrusive deep tracing of sci interconnect traffic. In Wolfgang Karl and Geir Horn, editors, *SCI Europe '99*, pages 53–58. SINTEF Electronics and Cybernetics, September 1999. ISBN 82-14-00014-9

1.4.3 Indirectly Relevant Peer Reviewed Publications

[MB04] Michael Manzke and Ross Brennan. Extending fpga based teaching boards into the area of distributed memory multiprocessors. In *Workshop on Computer Architecture Education*, pages 15–21, June 2004

[BM03] Ross Brennan and Michael Manzke. On the introduction of reconfigurable hardware into computer architecture education. In *Workshop on Computer Architecture Education*, pages 96–102, June 2003

1.5 Thesis Organisation

This thesis is organised as follows:

Chapter 1 Introduction provides a general introduction to the main research questions that were investigated as part of this thesis. This includes a description of the contribution in section 1.4, the thesis statement and lists of directly and indirectly relevant publications in section 1.4.2 and section 1.4.3 respectively.

Chapter 2 Motivation discusses the motivation for the main investigations that were conducted. In section 2.2 an argument to model PMC readings as random processes and to apply a discrete MMSE filter to fuse these PMC readings into common state vector is presented. At the beginning of this chapter in section 2.1 and section 2.1.1 a non-intrusive SCI trace instrument and its associated analysis software is introduced. Section 2.1.2 then elaborates on how trace data from the SCI Deep Trace instrument may be used to tune and validate SCI network models. It is pointed out that the non-intrusive acquisition of SCI network traffic and the tuning of SCI network models based on these trace data led to the idea to model PMC readings as random processes. Section 2.3 makes an argument for the advantages of Kalman filters to fuse many sequential PMC readings. This notion is expanded in section 2.4 to a distributed implementation that allows for the global observation throughout the nodes in hardware DSM clusters. The last section 2.5 in the motivation chapter 2 looks at the design of a special purpose graphics cluster. This design was strongly influenced by the previously mentioned investigations.

Chapter 3 Background and Related Work provides an introduction to the SCI technology in section 3.4.2. This is followed by related work descriptions for the main research questions. Section 3.2 presents related work for the "Trace Data Acquisition", section 3.3.1 for the "MMSE Filter", section 3.1.1 for "Performance Monitoring Counters", section 3.1 for "Performance Analysis" and section 3.5 for the "Special Purpose Graphics Cluster".

Chapter 4 High Speed Interconnect Trace Data Aquisition and Analysis provide a detailed discussion of the non-invasive high speed interconnect trace data acquisition and analysis. Section 4.4 describes how a SCI network model may be tuned and validated with interconnect statistics acquired with the trace instrument from section 4.1, section 4.2 and section 4.3.

Chapter 5 State Estimation of a Single Compute Node is dedicated to the main research topic, the state estimation of compute systems, and deals with all aspects of the estimation algorithm in section 5.1. The following section 5.2 deals with the acquisition and analysis of PMC event counts. This is necessary to tune the PMC process models in the

estimation algorithm. The last two sections of the chapter deal with the fusing of PMC reading sets into a larger state vector and with the implementation of the algorithm. This is discussed in section 5.3 and section 5.4 respectively.

Chapter 6 Optimisation and Re-evaluation of the Estimation Algorithm discusses some of the possible optimisations for the estimation algorithm in section 6.1 and analyses these in section 6.2. This is followed by an extended evaluation of the estimation algorithm for a single-node system in section 6.3 and section 6.4.

Chapter 7 Hardware DSM Testbeds provides a detailed description of two multi-node testbeds. One is a loosely coupled DSM cluster that is the testbed for all the implementations and evaluations in this thesis. This cluster is discussed in section 7.1. The second cluster is a conceptual design of a tightly coupled special purpose high performance graphics cluster. This cluster will eventually take full advantage of the estimation algorithm by using it for load-balancing. This tightly coupled cluster is described in section 7.2

Chapter 8 Compute Cluster State Estimation Algorithm (C^2STATE) finally applies the estimation algorithm to hardware DSM systems. Section 8.1 describes the necessary alterations to the single-node estimation algorithm and section 8.2 presents experiments and evaluation results of the global state estimation algorithm (C^2STATE).

Chapter 9 Conclusions and Future Work section 9.1 summarises the work presented in previous chapters and section 9.2 discusses limitations and future work.

Chapter 2

Motivation

Clusters of commodity PCs have become a dominant alternative to Massively Parallel Processing (MPP) systems. The *Top 500 Supercomputer list* included 28 clusters and 346 MPP systems in November 2000. In November 2005 this had changed to 360 clusters and 104 MPP systems [Sit05]. The majority of these clusters are loosely coupled systems that exhibit reasonably high bandwidth but relatively long latencies. They are not really suitable for fast real-time applications.

Interconnect technologies enable us to connect uniprocessors or Symmetric Multiprocessor (SMP) machines into clusters. This technology determines how tightly or loosely coupled a cluster is. This thesis focuses on a particular subset of clusters that implement DSM in hardware. Within this scope, the work investigates hardware DSM systems that are completely assembled from commodity components as well as systems that employ some custom-built components, thereby filling the void in the design space between fully custom-built high performance systems and systems that are entirely constructed from off-the-self components.

This class of clusters is of particular interest if the computational demands cannot be meet by a uniprocessors or a SMP machine and fast real-time constraints must be met. A good example application is a large scale interactive visualisation system e.g. a CAVE. These systems can have high computational demands. Furthermore the cluster must meet real-time constraints, typically at the frame-rate, and must also process significant Input/Output (I/O) within these constraints. The I/O operations may arise from motion detection equipment. This puts the motion detection, processing and visualisation system into a closed loop between the user's motion and the user's perception of the motion through the visualisation system. The overall latency through the motion detection and visualisation may not exceed the human-vision-system's ability to register changes, otherwise the system loses its ability to immerse the user and may cause simulation sickness. In comparison to high performance systems for scientific or engineering computation these immersive CAVE systems tend to have a modest amount of compute nodes but they are less tolerant to long latencies.

In this thesis I have chosen the SCI interconnect technology because it allows for the design of hardware DSM clusters that explicitly reduce latencies. In order the take full advantage of various computational resources provided by a hardware DSM cluster and to reduce the power dissipation of these clusters, one can employ various optimisation techniques. All these optimisations require run-time performance measurements.

This thesis describes two novel performance measurement and analysis methodologies that can be used to exploit any of these optimisations. The first method is concerned with the non-intrusive acquisition of interconnect traffic, which can assist the design and the verification of SCI based hardware DSM clusters. It allows the analysis of whether the interconnect topology can meet bandwidth and real-time constraints. It also helps to pinpoint bottlenecks in the interconnect topologies. The second method, the global state estimation of hardware DSMs, can control runtime adaption whether in hardware or software. The bulk of this thesis concentrates on this second method.

This thesis also discusses the conceptual design of a special-purpose tightly coupled hardware DSM system that can drive large scale interactive visualisation systems. This design is intended as a future target for the non-intrusive acquisition of SCI interconnect traffic and execution of the global state estimation algorithm.

2.1 Trace Data Acquisition and Analysis

The observation of high speed interconnect traffic, such as SCI, requires either to take measurements on the interconnect cable or to perform these measurements on the interface-adapter. Both methods have advantages and disadvantages that will be discussed later. In section 4.1 and section 4.2 an instrument and the associated software infrastructure is described that acquires interconnect traces and deposits these time-stamped and decoded traces into a database for subsequent analysis. Section 4.4 discusses how the instrument and software infrastructure may be used to tune a SCI fabric simulation model.

2.1.1 The SCI Non-Intrusive Deep Trace Instrument and Analysis Infrastructure

The SCI is one of the enabling interconnect technologies for high performance computing on PC clusters. Between 1997 and 2001, a trace instrument was constructed by Dr. Brian Coghlan of TCD that allows deep traces of SCI interconnect traffic. I designed and implemented the bulk of its software infrastructure. Such an instrument is essential for a detailed spatial and temporal analysis of parallel executed algorithms on loosely coupled clusters, as in the case of the general purpose SCI testbed cluster discussed in section 7.1, and tightly coupled clusters, as in case of the special-purpose high performance graphics cluster presented in chapter 7.2. At the time, there were no commercial instruments available to non-invasively sample and store very deep ($\gg$10Mbyte) interconnect traces per target node.

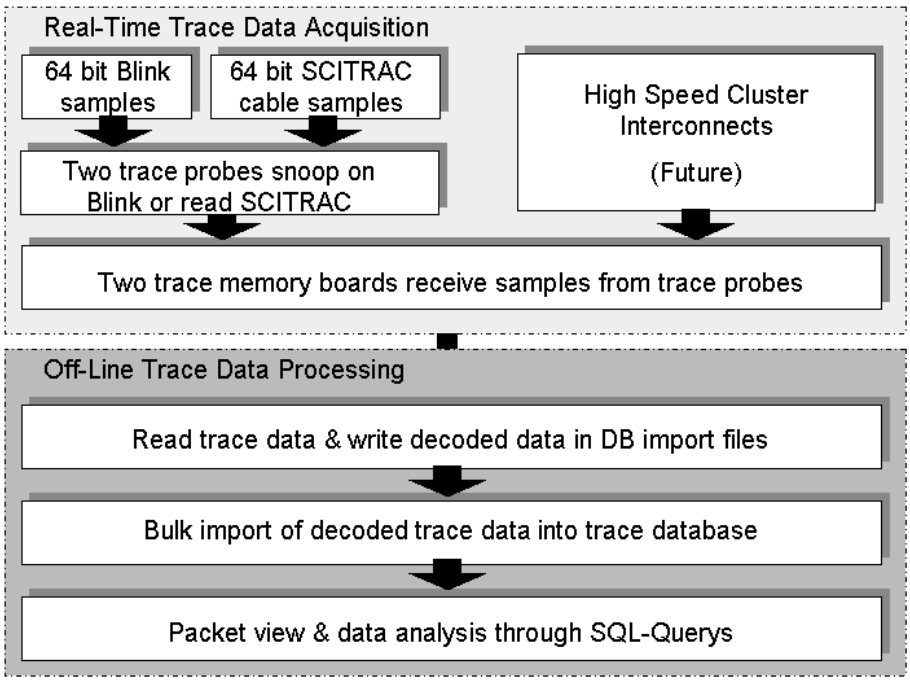

Figure 2.1: Trace data flow overview

The primary observation of interface traffic is accomplished through snooping on the Back-side Link (Blink) [Dol96], Dolphin's implementation of the SCI transfer cloud. Snooping on SCI cable traffic, via SCILAB's SCITRAC [SBNW98], is also supported. The tracer's configuration consists of three modules, a trace probe, a deep trace memory and a trace database. The database provides a powerful means for a fine-grained analysis of a large quantity of trace data.

The instrument provides hardware designers and software developers with a tool that allows a deeper understanding of the temporal behaviour on any given target system. Unlike other systems, e.g. see [KL97, KLS99], this trace instrument is targeted to commercially available interconnect hardware and therefore provides the user with information about the true temporal behaviour of clusters made up of standard components. Fig. 2.1 shows how the trace instrument's hardware and software components are related to each other during trace data acquisition and subsequent off-line data analysis.

The instrument is designed to fulfil the following main objectives:

- Non-intrusive monitoring of SCI interconnect traffic
- Very deep ($\gg$ 10Mbyte) interconnect traces per node
- Acquisition of all the interconnect traffic
- Synchronous trace acquisition on multiple nodes through a shared trigger mechanism

- Allowance for various probes to accommodate SCI cable traffic and Blink traffic
- Straightforward adaptation to various SCI interface implementations.
- Trace data storage in commercial relational database
- Ability to analyse causal relationships in synchronously acquired traces from different targets

The utilisation of a relational database provides the user with an easy means to extend and to adapt the predefined database queries to their specific needs.

2.1.2 Tuning and Validation of SCI Network Models through Non-Intrusive Deep Traces

The non-intrusive acquisition of interconnect trace data and a subsequent data analysis can be used to accurately tune the definition of interconnect loads and the parameterisation of interconnect simulation models. High speed interconnects or system area networks are the principal components of a compute cluster that transform stand alone computers into a cluster. The design of such interconnect fabrics may be assisted through performance prediction. This prediction is accomplished through simulation if the model's parameterisation reflects the physical fabric and realistic load descriptions are provided.

A simulation is only as accurate as its simulation model. A simulation model must be tuned and verified in order to guarantee that the model reflects the real physical system behaviour, but this requires information about the true temporal behaviour of the physical interconnect. This information can be extracted from interconnect trace data. In particular, the trace data must be acquired non-invasively for it to be accurate.

In section 4.4.1 the parameterised SCI node model developed at the University of Oslo is presented. This model has been used to evaluate SCI topologies consisting of 20 ringlets, and up to 96 nodes. The model development was done within the framework provided by the OPNET Modeler [MIL97]. A standard distribution of OPNET contains models of most standard communication technologies like Ethernet, Fiber Distributed Data Interface (FDDI), Asynchronous Transfer Mode (ATM), etc. This facilitates easy integration of our node with models of other technologies, enabling simulation of heterogeneous systems with very limited additional development. Section 4.4.2 then presents mechanisms that allow the extraction of statistical information from a trace-database for the generation of realistic system loads. These loads are used to tune the simulation model described in section 4.4.1. Fig. 2.2 shows the framework for this. The individual components of the simulation, measurement and analysis system are based on work on non-intrusive deep tracing of SCI interconnect traffic [MC99a, SBNW98, SNBW99] and simulation of high-speed-interconnect traffic [RL99] at the Department of Physics and the Department of Informatics at the University of Oslo and at TCD.

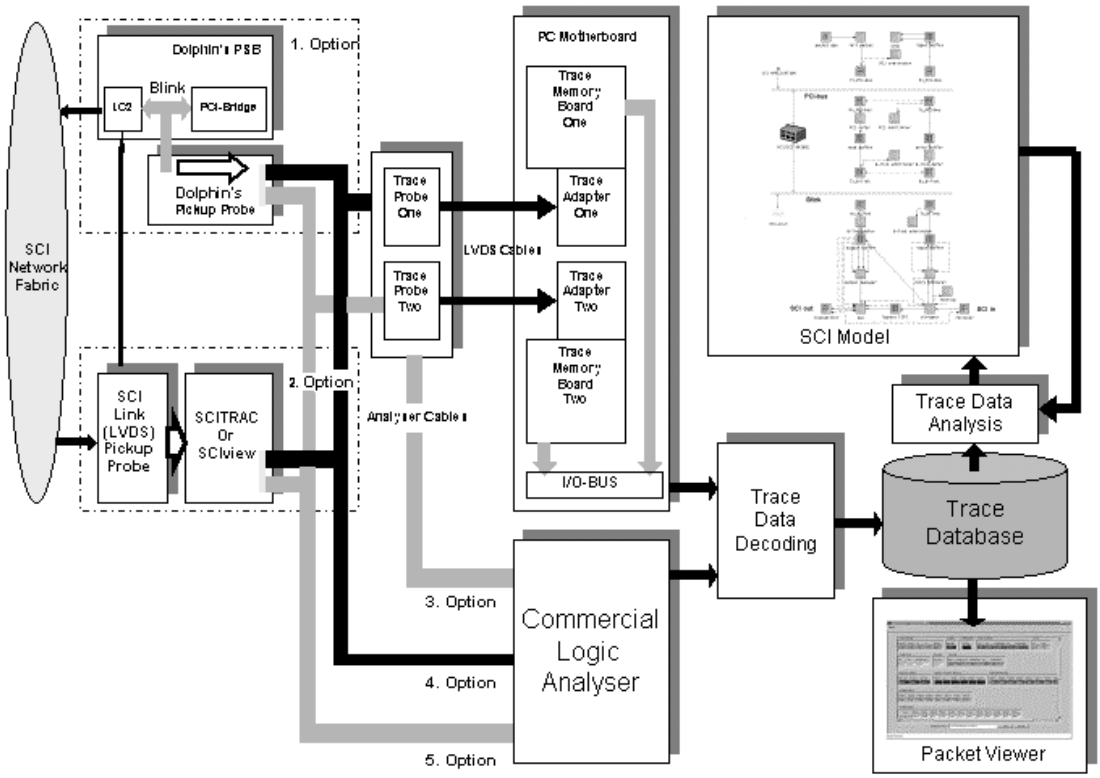

Figure 2.2: Trace Data Acquisition and analysis framework

2.2 Global State Estimation of Hardware Distributed Shared Memory (DSM) Systems

Today's high performance computers, whether uniprocessor, multiprocessor or hardware DSM systems, are madeup of concurrently operating subsystems. Run-time knowledge of the system-wide utilisation state of these subsystems including CPUs and their interconnects can assist efficient scheduling of tasks onto these resources. E. Duesterwald et al. [DCS03] argue for a prediction that would allow the system to adapt more efficiently to the time-varying behaviour of the programs. Their work observes performance metrics that are based on CPU PMC event readings. An analysis of these observations showed that programs exhibit strong behaviour variations at PMC sample interval granularity but that the various metrics shared periodicity. E. Duesterwald et al. [DCS03] exploit this characteristic to perform resource-aware scheduling.

Looking more specifically at the CPUs, there has been an increase in research activities concerned with dynamic optimisations of the processors' operations through hardware or software adaption. Bahar and Manne [BM01] minimise the power dissipation of a gen-

eral purpose processor through Pipeline Balancing (PLB). In this case performance monitoring controls the dynamic adaption of resources. Dynamic program phase detection is applied in Balasubramonian et al. [BABD00] to optimise the memory hierarchy configuration. This leads to a lower power dissipation and improved performance. Balasubramonian et al. [BDA03] investigate optimisations of clustered micro-architectures [FCJV97, PJS97]. This is accomplished by taking advantage of Instruction Level Parallelism (ILP). For clustered micro-architectures the optimal performance is achieved by finding the best trade-off between communication and parallelism. Again program phases are detected to adapt the cluster configuration to the current workload. In Dhodapkar and Smith [DS02] a reconfigurable instruction set cache is adapted by matching working set signatures with current program phases. Folegnani and Gonzales [FG01] lower the energy consumption of the CPU's issue logic by dynamically reducing the effective size of the instruction queues. Huang et al. [HRT03] propose an alternative to the temporal approach to adaption. They use subroutines at the granularity of program phases to determine the correct adaption of the system. Dynamic Voltage Scaling (DVS) is used by Hughes et al. [HSA01] to adapt general-purpose processors to multimedia workloads that operate on per frame time constraints. Magklis et al. [MSS+03] investigate DVS techniques in Multiple Clock Domain (MCD) micro-architectures. These target systems operate with Globally Asynchronous Locally Synchronous (GALS) clocks that are dynamically scaled relative to queue utilisation. These queues supply clock domains with data or instructions. Similar to Folegnani and Gonzales [FG01], Ponomarev et al. [PKG01] adapt the length of the Issue Queue (IQ), the Reorder Buffer (ROB) and the Load/Store Queue (LSQ) depending on their occupancies. Finally, Wu et al. [WMC+05] explore a dynamic compilation environment to control DVS.

In addition to the dynamic adaption work, researchers have successfully investigated PMC events to predict the run-time CPU and memory power consumption [IM03, Bel00, Mar01, KCK+01, LJ03]. Contreras and Martonosi [CM05b] used a first-order, linear power estimation model to observe CPU and memory power consumption based on PMC readings.

All these research activities are good examples of the wealth of optimisation opportunities that are available if dynamic adaption or scheduling is employed. It also illustrates that the ability to indirectly measure power consumption in parts of the CPUs micro-architecture is useful if energy is to be saved. Furthermore all the work cited here depends on accurate PMC event readings. This emphasises the importance of accurate counter readings of a variety of events.

This thesis is not concerned with the scheduling of resources, the dynamic optimisations or the measurement of power consumption, but introduces an algorithm that generates a global view of the system's utilisation or computational state. It is for other to use these state information for optimisations. The global view is derived from hardware PMC readings that count the number of occurrences of a selected event.

The PMC register reading z_k in Eq. (2.1) represents the *mean* of the counted events

$e(t)$ over the *sample interval* Δt. The choice of duration of the sample interval Δt is a trade-off between counter accuracy and computational overhead. Reducing Δt will increase the accuracy until the computational overhead (caused by instructions that read and process the PMC registers) contributes a significant amount of counter events to the reading. Section 5.4 provides a detailed discussion of this topic. Furthermore if the system's software is responsible for the acquisition of counter readings then the Operating System (OS) ability to schedule such register readings determines the minimum sample interval Δt_{min}.

$$z_k = \frac{1}{\Delta t} \sum_{k\Delta t}^{(k\Delta t)+\Delta t} e(t) \tag{2.1}$$

System events, such as the *Number of Instruction Fetch Misses*, that are counted over a time period provide a measurement of the degree of availability or utilisation of particular system resources. These observations are easily obtained through registers that implement performance counters. In general counter registers can be instructed to count a particular event by means of a selection register. This selection is required because in most cases the number of PMC registers does not match the number of events that may be counted. For example the *Intel*^{TM}*PIII* processor has only two registers but each of these may be used to count any one of more than 100 events ranging from the *Number of Bus Transactions* to the *Number of Floating-point Operations* [Ord01]. Please see Table A.1 for a complete list of all available PMCs.

This selection approach does not constitute a problem as long as the number of required PMC readings does not exceed the number of available counter registers. If the number of distinct event counts exceeds the number of available registers then this approach fails. One remedy is if the system uses a single register to count different events in a nested sequence. Fig. 2.3 depicts this multiplexing technique. A *set* in Fig. 2.3 refers to the number of PMC registers in a given subsystem. The shaded areas in this figure highlight the intervals $[\Delta t_2, \Delta t_n]$ and $[\Delta t_{n+2}, \Delta t_{n+n}]$, with n as the total number of event sets required to observe the current state of resource utilisation. No events are counted for the first counter set, and consequently the accuracy decreases for these multiplexed readings [DLM+01].

This thesis proposes to model PMC readings as random processes and to apply a Kalman filter to fuse PMC readings into a state-vector that holds an optimal estimate of the system-wide PMC event counts. The inspiration for this approach was developed during the analysis of SCI interconnect traffic and the subsequent tuning of SCI network models based on statistics derived from SCI trace data.

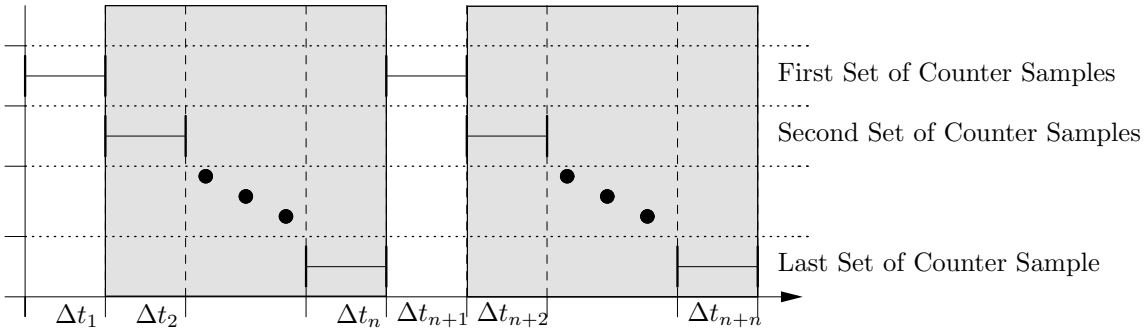

Figure 2.3: Multiplexed sets of performance counter readings

2.3 MMSE Filter Algorithm

The discrete MMSE filter was originally formulated by R. Kalman in 1960 [Kal60] and can process multiple time-variable inputs through the use of state space methods. The application of stochastic system models for the PMC event readings is not only beneficial for the optimal estimation of a noise-corrupted and incomplete scalar reading but also for multiple readings. This thesis demonstrates how the well known Kalman filter algorithm can be used to make a larger variety of PMC information accessible to the system's software than would otherwise be observable with a restricted number of counter registers.

Fundamental to the Kalman filter is the presence of noise. In this particular case, the noise originates from the non-deterministic execution of the PMC acquisition software. The histogram in Fig. 2.4 shows how the sample time changes around the mean of 3.9532e+07 clock cycles. The CPU operates at 1 GHz, consequently the mean sample time of 3.9532e+07 clock cycles is equivalent to 39.532 ms. The samples vary with a standard deviation of $\sigma = 1.1074e + 5$ around the mean. Section 5.1.2 shows that the state transition matrix Φ and the covariance matrix Q of the Kalman filter algorithm both depend on the sample interval Δt but the filter operates with a fixed sample time. Therefore the filter will process randomly too many or too few PMC events. This is seen from the filter's perspective as noise. A further contributing factor is the quantisation of the PMC event readings.

2.4 Distributed MMSE Filter Algorithm

The implementation of a MMSE filter algorithm can help to observe more PMC events than otherwise possible. If the node is a SMP machine it is possible monitor all the CPUs in the system, but also remember that this thesis is particularly concerned with the design and performance measurements of hardware DSM clusters, especially those using SCI. In order to provide a global view, in terms of performance, of such clusters, it is necessary to extend the scope of the MMSE filter algorithm over the entire cluster or a subset of cluster nodes that

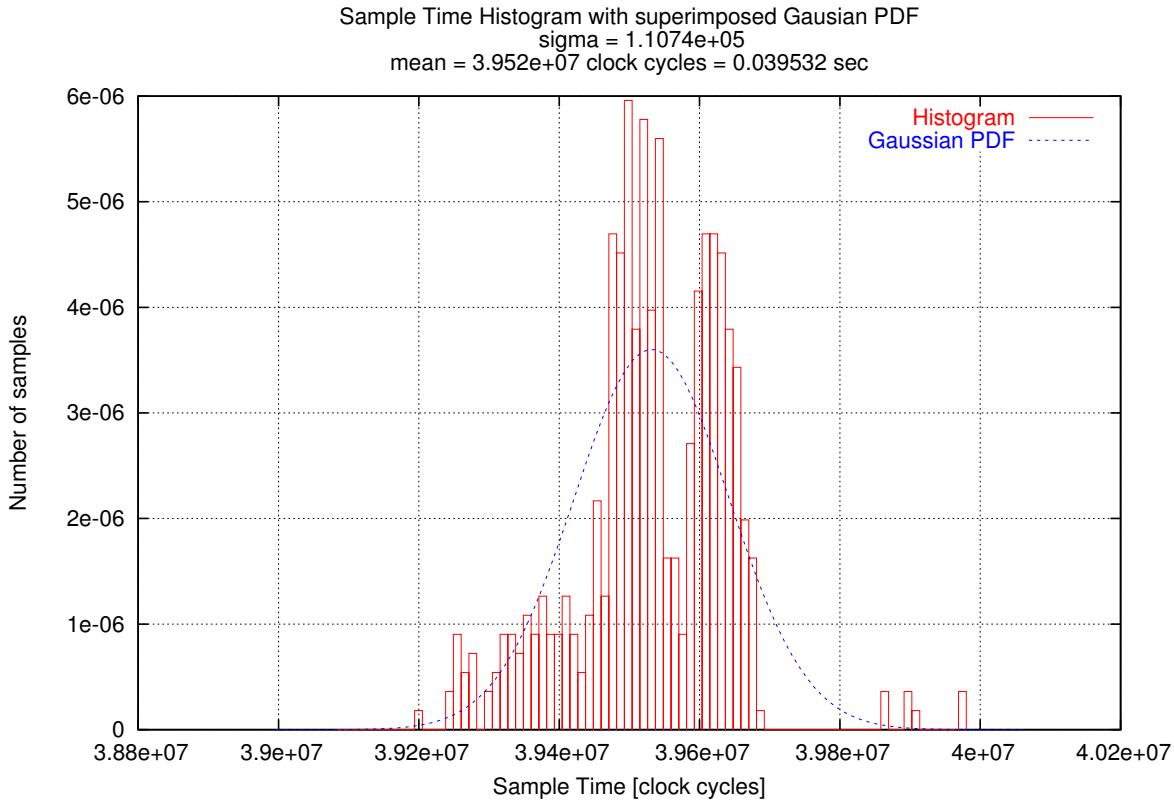

Figure 2.4: Sample Time measured on CPU 1 in one of the two SMP systems specified in table 7.1

are suitable for a run-time performance analysis. I have named this algorithm the "Compute Cluster STATE estimation" (C^2STATE) algorithm. Every node in the cluster that requires a global performance view runs a MMSE filter algorithm that fuses local and remote PMC readings into its state-vector. From the perspective of each node, remote PMC readings are made available through the shared memory. Consequently all participating nodes hold an estimate of the cluster wide performance in state space representation. This state-vector may be used for performance measurements, load balancing and other optimisations.

2.5 Special-Purpose Graphics Cluster

When looking for a good example where advantage could be taken of a global performance estimation algorithm and the interconnect technology, an interactive parallel graphics system appeared to be a good match. General purpose OSes, e.g. Linux, can sample PMC readings at 50Hz. This is approximately the frame rate that one would expect from graphics systems and it would also be advantageous to make load balancing decisions in a parallel rendering systems for every new frame. Furthermore, interactive visualisation systems frequently use

motion tracking and other I/Os to provide feedback to the application. To guarantee a seamless integration of this information into the interactive application it is necessary to limit the latencies involved. One advantage of the SCI interconnect is that it operates at extreme low latencies especially if an I/O bus is avoided.

If the computational demands of an interactive graphics rendering application cannot be met by a single commodity GPU, multiple graphics accelerators (provided by a cluster of PCs in conjunction with a software infrastructure) may be employed to provide the required additional resources. Typically these systems allow the application programmer to use a standard OpenGL Application Programming Interface (API). This thesis describes a hardware architecture that accelerates graphics applications with a tightly coupled hybrid system of parallel commodity GPUs and reconfigurable hardware. The GPUs share a single address space with a PC cluster, implemented through dedicated commodity high-speed interconnect. Applications may take advantage of resources provided by the shared-memory GPUs, the reconfigurable hardware and the PC cluster. When complete, this system will be an explicit tightly-coupled target for the C^2STATE algorithm.

Chapter 3

Background and Related Work

This thesis is concerned with the global state estimation of hardware DSM clusters. The work also presents a novel special purpose hardware DSM architecture that is intended to employ the global state estimation algorithm to optimise its performance. This chapter provides the background to this work and discusses related work.

3.1 The Performance Analysis

In general performance analysis comprises the measurement of performance data, the evaluation of these data and a subsequent optimisation. The evaluation and optimisation may be performed off-line or at run-time. Performance analysis can also be divided into: performance measurement, performance modelling and performance simulation. This thesis focuses on the measurement and estimation of DSM performance but also discusses possible applications.

There are several commercial and research tools for the performance analysis of parallel applications. They are suitable for message passing and shared memory programming paradigms. The performance measurements are derived from system event counts. These may be based on hardware or software event counts. Hardware events are implemented within PMC registers. Software events require the instrumentation of the source or object code. These event counts are then used for profiling or tracing. In the case of profiling, statistics for several event-counts are generated. Tracing requires the acquisition of the event history. This approach generates large quantities of trace data that are usually analysed after the execution of a parallel program.

Some manufacturer of parallel high performance systems provide performance analysis tools for their systems. IBM's XProfiler allows for the profiling of serial and parallel applications [Xpo06]. Similar features are included in SGI's ProDev WorkShop [Pro06] and Cray's PAT [GM98, PAT06].

Pallas' Vampir and Vampirtrace third party tools are now part of Intel's Trace Ana-

lyzer Collector [Tra06]. Other third party tools such as ETNUS' Totalview include Message Passing Interface (MPI) [Mes03] and OpenMP [Ope05] debugging tools [Tot06]. As with Totalview, Crescent Bay Software's Deep [Dee06] supports the debugging of message passing and shared memory machines, but this tool also uses PAPI [PAP02] to gain access to PMC readings. The "European Center for Parallelism of Barcellona" has developed the Paraver tools with similar functionality [JJLG03, GTAB01, JJL+03, FCL02].

The TAU project [Tau06] provides profiling and tracing for performance analysis of parallel programs. Performance measurements are achieved through the instrumentation of source code and use of a Program Database Toolkit (PDT), see Lindlan et al. [LCM+00]. The framework can also dynamically instrument the binaries by using Paradyn's DyninstAPI (see later). The PDT database holds compile-time information for source-level feedback. Malony et al. [MST+05] discuss the integrated measurement, monitoring, and optimisation of Common Component Architecture (CCA) component-based applications.

Paradyn [Par06] is suitable for parallel and distributed applications [MCC+95]. The tool uses dynamic instrumentation for performance measurements. This is achieved by instrumenting unmodified executable files. The program may be modified during the execution. Paradyn monitors its overhead during the execution and corrects the instrumentation if necessary. This tool was subject to research for more than a decade: Hollingworth and Miller [HM93] demonstrate how the performance analysis of large scale parallel computers can be made manageable by combining the dynamic selection of performance data with decision support. Hollingworth et al. [HMC94] argue in their paper for dynamic instrumentation. Irvin and Miller [IM94] demonstrate the NV model that links system level activities with the high level abstraction of the source code for debugging purpose. Miller et al. [MCC+95] report a Paradyn implementation for Parallel Virtual Machine (PVM). Irvin et al. [HM96] discuss a cost model for perturbation caused by the software instrumentation. Irvin et al. [IM96b] suggest necessary cooperations between compilers and performance tools. Irvin et al. [IM96a] are also concerned with the interpretation of low level events from the perspective of a high level language. The work suggests certain mappings as a solution. Hollingworth et al. [HMG+97] discuss the Metric Description Language (MDL) and a compiler that allows a machine independent instrumentation definition. Zhichen et al. [XLM97] demonstrate how Pardyn exploits a shared-memory systems' cache coherence protocol to identify performance bottlenecks. Karavanic et al. [KMLM97] report on the performance data visualisation tool that can integrate various performance data including those generated by Pardyn dynamic instrumentation. Zhichen et al. [XMN99] introduce an extension to Pardyn's dynamic instrumentation for threaded sequential and parallel programs. Cain et al. [CMW00] discuss the introduction of call-graph-based search strategies into the Pardyn tool. Miller et al. [MCI+01] discuss how application security vulnerabilities can be exploited through dynamic instrumentation. Roth and Miller [RM02] present a new performance bottleneck search strategy that uses stack sampling. Mirgorodskiy and Miller [MM05] report a self pro-

pelled dynamic instrumentation agent that searches for intermittent performance problems. Harris and Miller [HM05] present content and structure analysis methods that can deal with stripped binary code. Collins and Miller [CM05a] discuss a search techniques for fine-grained program structures in order to efficiently instrument loops. Roth and Miller [RM06] present a bottleneck detection strategy for systems that run a large number of processes. They achieve this by applying a novel multicast and data aggregation infrastructure.

The TAU and Paradyn projects are a good examples of very productive long-term research activities.

3.1.1 Performance Counter

Most contemporary CPUs have hardware PMC registers that may be sampled to derive performance information about the CPU's micro-architecture. Access to these registers may be provided through high level APIs, e.g. Performance Counter Library (PCL) [PCL06] and PAPI [PAP02]. These high level APIs are also used by many of the previously mentioned performance analysis tools, e.g. TAU. A number of publications relate to the widely used PAPI API [MDK+04, AM05, DMM+04, DBL, DMM+03, DLM+03, WM03]. These two APIs use Mikael Pettersson's Linux 2.x.x kernel patch [Pet02] to access Intel Pentium, Pentium MMX, PPro, Pentium II, Pentium III, Pentium 4 and AMD Athlon, Duron PMCs in kernel space.

Ojha [Ojh01] compares hardware PMC measurement with software and hybrid performance counters. Moore [Moo02] compares the PMC counting and sampling modes. Cérin and Fkaier [CFJ03] investigate sorting algorithms by observing the ratio of L1 data cache misses and retired instructions through PMC readings. This work uses Mikael Pettersson's "perfctr" [Pet02].

3.1.2 Multiplexed Performance Counter Readings

Many performance measurement situations require the reading of more PMC events than can be simultaneously sampled with the number of available PMC registers. One solution is to multiplex PMC events onto the PMC registers at the expense of accuracy. For example PAPI uses the MPX [May01] library to implement multiplexing. Mathur and Cook [MC05b] increase the accuracy of multiplexed PMC readings by applying Positional Mean Error (PME), Multiple Linear Regression (MLR), Trapezoid-area Method (TAM) and Divided-Interval Rectangular Area (DIRA) estimation methods. Azimi et al.[ASW05] improves multiplexing accuracy through high frequency PMC sampling. This is possible with the K42 research OS [SKW+06].

3.1.3 Cluster wide PMC Collection

Cluster wide PMC reading collections are presented in [DBL, WSS+04].

3.2 Trace Data Acquisition and Analysis Related Work

Hollingsworth et al. [HLM95] discuss various techniques for performance measurement of parallel systems. The authors distinguish between program instrumentation and hardware instrumentation. They point out that the instrumentation of executables may perturb the execution of a parallel program but that hardware instrumentation is non-intrusive. Despite this obvious advantage the hardware instrumentation makes it difficult to associate hardware based performance information with the source code of the executed program. The authors suggest that a hybrid solution can provide a trade-off between the non-intrusion nature of the hardware instrumentation and the software instrumentation ability to provide performance data that are easily associated with the source code. They also argue that a hardware instrumentation must be distributed over all the nodes in the system.

Martonosi et al. [MCM96] presents Princeton's SHRIMP performance monitor and provides a detailed discussion of the hardware monitor. An FPGA-based monitor is attached to each node in the SHRIMP system and accumulates interconnect events. The monitor cards also support multiplexing.

An alternative solution is provided by Liao et al. [LJI+98]. The authors utilise the programmable network interface card of a Myrinet-based cluster to implement a performance measurement mechanism. This approach uses a software solution but implements it on an independent platform.

Karl et al. [KST00] reports about a hardware monitor similar to Princeton's SHRIMP performance monitor. The monitor hardware is attached to SCI interface cards on every node in the cluster and snoops on the local bus of the interface card. Interconnect events are stored through a caching mechanism that increases the utilisation of the available event memory [HJK+00, TGS+01, GST+02, KLS99].

More recently Kenny et al. [KCB+05] investigated Ethernet traffic performance analysis. The authors used features of the interface card to collect and analyse Ethernet traffic that is specific to grid operations.

3.3 Kalman Filtering

In 1960, R. E. Kalman published his now well cited journal article [Kal60]. In this article he describes an alternative to the Wiener filter solution. He proposed to apply state-space methods for the MMSE filtering problem. This approach makes it possible to process complex time-variable, multi-input/output problems. The filter allows modelling of the random processes and recursive processing of them in a digital computer. Since Kalman's original article many papers have been published in this area. A search with CiteSeer [cit06] provides references to approximately 5000 scientific publications and the Compendex and Inspec [Com06] database hold approximately 30000 papers that are concerned with Kalman

filtering.

The filter has been applied in many areas: Chen et al. [CWS97] propose an Interval Kalman Filter (IKF) for the estimation of interval linear systems. Pearson et al. [PGEM97] investigate the stability of combat aircraft Extended Kalman Filter (EKF)s for high system integration. Hong et al. [HCC98] use discrete wavelet transforms that are implemented on a Kalman filter bank to decompose multiresolutional random signals. McMillan et al. [McM94] compares Kalman filter estimates that are based on Global Positioning System (GPS) receivers with estimates that are based on inertial equipment. The measurements were conducted during sea tails. Kuo et al. [KHJ+96] propose a block-based motion estimation method that uses a Kalman filter to assist the compression of video signals. Murty and Smolinski [MS88] apply a five-state Kalman filter to estimate the fundamental and second harmonic of a power system in order to implement a digital differential relay for a three phase power transformer. Sinopoli et al. [SLF+04] use a Kalman filter to model the arrival of observations as a random process. These observation originate from unreliable communication channels in large, wireless, multi-hop sensor networks. These example demonstrate the diversity of applications of the Kalman filter.

3.3.1 MMSE Filter Related Work

In the Computer Science domain there are also many examples of Kalman filtering. Discrete MMSE filters were investigated for network traffic flow control [Kes91] and motion tracking applications [Bac00][Gre96] but, to my knowledge, they were not investigated for the acquisition of PMC readings. Gregory Welsh developed the Single-Constraint-At-A-Time (SCAAT) solution for motion tracking with an EKF[Gre96].

3.4 Compute Cluster

According to the *Top 500 Supercomputer list* [Sit05] clusters have became the most used systems for high performance computations. The list included 28 clusters and 346 MPP systems in November 2000. In November 2005 this had changed to 360 clusters and 104 MPP systems [Sit05].

3.4.1 Interconnect Technologies

The majority of these clusters are loosely coupled systems that exhibit reasonably high bandwidth but relatively long latencies. The main interconnect alternatives are: Myrinet [Myr06], Gigabit Ethernet [Gig06], Infiniband [Inf06], Quadrics [Qua06] and SCI [SCI06]. Out of this list SCI is the only technology that allows assembly of hardware DSM clusters.

3.4.2 Scalable Coherent Interface (SCI)

Defined in 1992, SCI is a well established technology and many high performance cluster implementations employ this interconnect (e.g, PC2 University of Paderborn Germany, University Of Delaware - The Bartol Research Institute USA and National Supercomputer Centre in Sweden) [Top04]. Subsets of the SCI standards have been implemented and are available as commodity components. In particular, Dolphin [Dol04] has implemented Peripheral Component Interconnect (PCI) cards that bridge PCI bus transactions to SCI transactions. Hellwagner et al. [HR99] cover much of the related hardware and software. Compute nodes with PCI slots may be interconnected through PCI-SCI bridges together with a suitable SCI fabric topology, thus bridging their PCI buses. Memory references made by one of these nodes into its own PCI address space are translated into an SCI transaction and transported to the correct remote node. The remote node translates this transaction into a memory access, thus providing a hardware DSM implementation. Programmed Input Output (PIO) and Direct Memory Access (DMA) may be performed without the need for system calls.

Figure 3.1 illustrates the design of a Symmetric Multiprocessor SMP node with a commodity SCI card in one of its PCI slots and also shows the main components on this SCI card. The PCI-SCI bridge translates between PCI transactions and SCI transactions and forwards them onto the PCI bus or the Blink bus. The SCI Blink bus interconnects the PCI-SCI bridge with up to seven SCI Link Controller (LC)s or alternative components. The SCI cards shown have two SCI LCs attached to the Blink and consequently are suitable for the construction of a 2-dimensional torus. Systems with more than one LC route packets between them over the Blink to the corrected LC according to a routing table. This enables distributed routing of SCI packets between individual SCI rings without an expensive central SCI switch. Routing is configured during SCI fabric initialisation. Every LC has an input and output port, and the output port of one is connected via a cable to the input port of another. These links are 16 bit parallel and unidirectional with a bandwidth of 667Mbytes/s.

SCI and Cache Coherency

The connection of SCI LCs to the I/O bus via a bridge was not intended during initial specification of the SCI standard but it allows commodity component manufacturers to offer SCI subsystems that may be attached to a diverse set of computer architectures as long as they provide a standard I/O bus. This therefore enables the construction of NUMA machines from commodity components. This approach prohibits the implementation of cache coherency as defined in the SCI IEEE 1596-1992 standard [Ins93] for the commodity PC cluster. On the other hand ccNUMA machines can be built if the I/O bus is avoided.

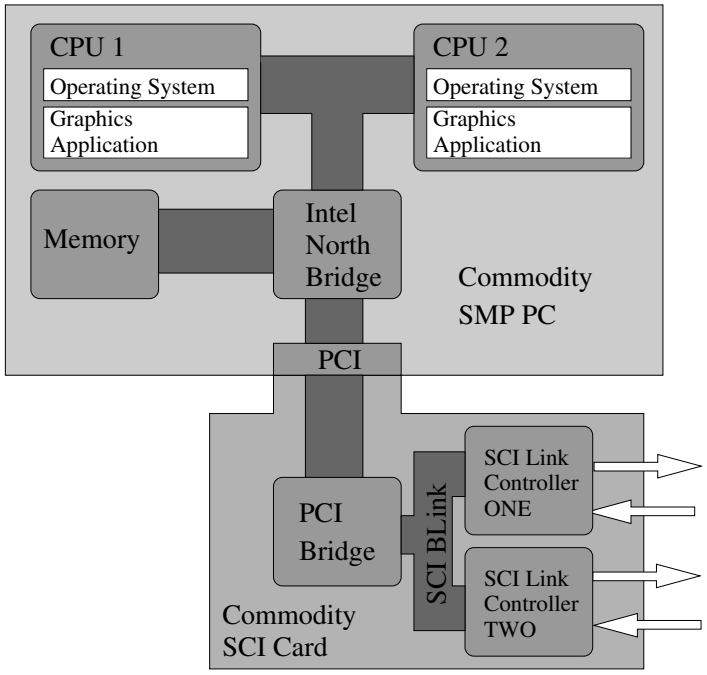

Figure 3.1: SMP Desktop Node with 2D SCI-PCI interface card.

SCI Commodity System Software

SCI driver software and a Software Infrastructure for Scalable Coherent Interface (SISCI) API for the Dolphin SCI cards force the OS to reserve a particular section of the main memory for SCI, thus also preventing paging of this set of memory pages [Dol04]. The software then makes this part of the memory available to other nodes in the SCI cluster. These nodes' processes may map this remote memory into their processes' virtual address space. Subsequent references to virtual addresses that are located on a remote node will be executed in hardware without expensive system calls. Although the SCI driver is not really a driver in the traditional sense, as it does not avail of OS services during SCI transactions, its definition as such facilitates the system initialisation and set-up.

3.5 Special Purpose Graphics Cluster

This work is mostly motivated by Standford's research around the WireGL project [HEB+01] and the Chromium project [HHN+02]. My parallel rendering solution integrates reconfigurable hardware in form of FPGAs into the graphics pipeline as a distinct computational stage between the application stage that executed on the host system's instruction set processor and the geometry stage of the commodity graphics accelerator. Recent work at the Saarland University demonstrated an implementation of a real-time ray tracer on a single FPGA that would otherwise require a cluster of commodity PCs [SWW+04]. Other recent work at the University of North Carolina at Chapel Hill implemented an FPGA based

view-independent graphics rendering architecture [SBM04] and earlier research at the University of Tübinger was concerned with the development of a real-time volume renderer implemented through Digital Signal Processing (DSP) and FPGA components [MKS98]. These are just three examples that demonstrate the suitability and computational ability of reconfigurable hardware for graphics applications. My architecture further improves the performance potential of these components by making them part of the hardware DSM. A recent commercial shared memory graphics cluster solution, the Onyx4, was evaluated by the University of Utah [GPH04]. Other investigations at University of Utah were concerned with the implementation of interactive ray tracing on clusters but they applied software DSM for their ray tracer [DGBP05, DGP04, DPH+03].

Chapter 4

High Speed Interconnect Trace Data Acquisition and Analysis

The remainder of this chapter describes an SCI trace instrument created in the Department of Computer Science, TCD. It also shows how this instrument is used to validate SCI simulations. The author contributed significantly to the software infrastructure.

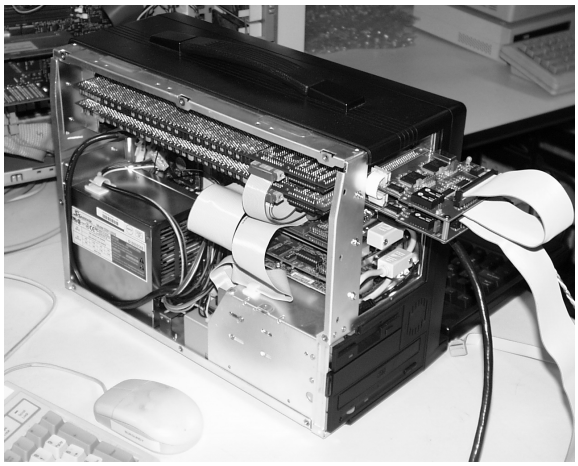

Figure 4.1: SCI Deep Trace Instrument Front

Figure 4.2: SCI Deep Trace Instrument Back

4.1 SCI Trace Instrument Hardware

The non-intrusive measurement of interconnect traffic can be achieved with two stages of trace instrumentation. The following hardware options represent the first stage of instrumentation: an SCITRAC [SBNW98] link tracer, an SCIview [SNBW99] field programmable tracer instrument or an adapter card that observes interface traffic through snooping on the Blink [Dol96], Dolphin's implementation of the IEEE standard SCI transfer cloud [Ins93].

The second stage collects trace data from this instrumentation. There are two options

available for this purpose: a standard commercially available logic analyser or a trace instrument developed at TCD [MC99a, CMB+98, CM99, MC99b].

The hardware of the latter trace instrument [CMB+98] comprises a portable PC, two deep trace memory boards, two probe adapters [CM99] and two trace probes (Fig. 4.3). The latter two are medium complexity 7-layer Printed Circuit Board (PCB)s, but the deep trace boards are very complex Multiwire PCBs equivalent to > 12 layers.

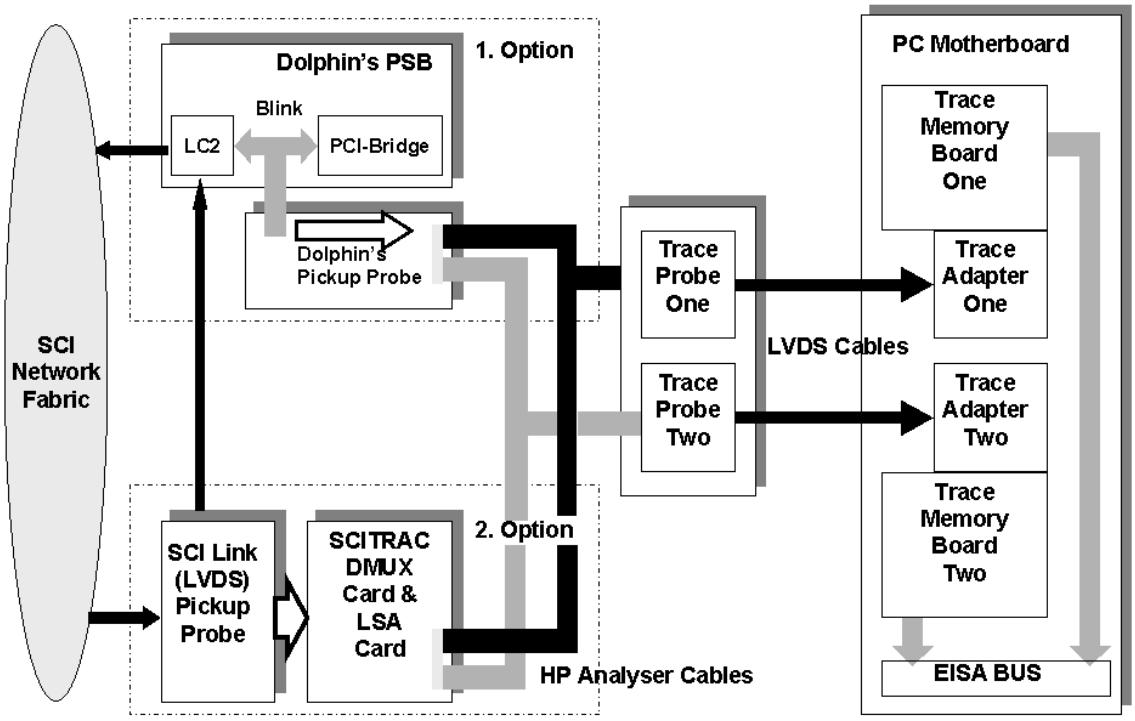

Figure 4.3: Trace hardware overview including three possible trace targets

4.1.1 Trace Probes

Blink traces from Dolphin's SCI-PCI bridge can be acquired via a probe card supplied by Dolphin that attaches to their SCI interface cards via elastomeric connectors. This card breaks out the Blink signals to a number of connectors that will accept cables for a HP16500 series logic analyser (see Fig. 4.3, Option 1). The same pin-out and connectors are used in a proprietary avionics SCI-PCI Bridge implementation. Furthermore SCILAB's SCITRAC cable tracer provides broadly similar connectivity (see Fig. 4.3, Option 2).

The instrument requires two trace probes [CM99] that attach to the trace target via

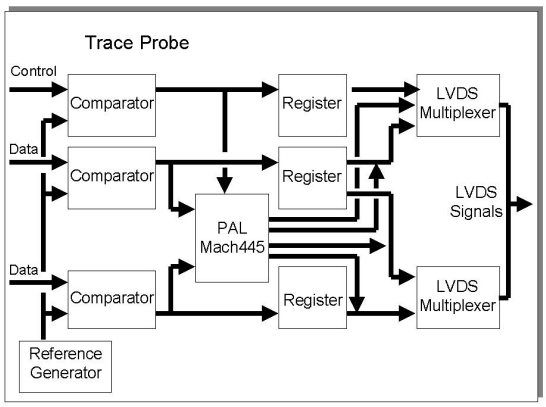

Figure 4.4: Trace probe block diagram

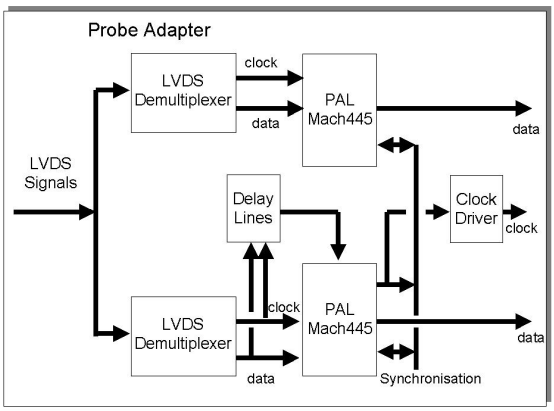

Figure 4.5: Probe adapter block diagram

HP16500 series compatible cables and are synchronised by an inter-probe cable. Each trace probe attaches to 48bits of the 96bit-sample data path. A block diagram for a trace probe is shown in Fig. 4.4. The trace probe multiplexes the trace samples onto Low Voltage Differential Signalling (LVDS) cables, which connect the probes to the trace instrument's adapter cards. They allow a maximum sample rate of 66 MHz, and consequently they are suitable for Link Controller 2 (LC2) applications. The Mach445 Programmable Array Logic (PAL) can be used as a test pattern generator, deriving its clock from a local crystal oscillator. The PAL can further be employed to match input patterns.

4.1.2 Probe Adapter

The probe adapters demultiplex and resynchronise the trace probes' LVDS signals. Each adapter attaches to one of the deep trace memory boards and provides the memory board with 48bits of a 96bit data path. Again the PAL may be used as a test pattern generator.

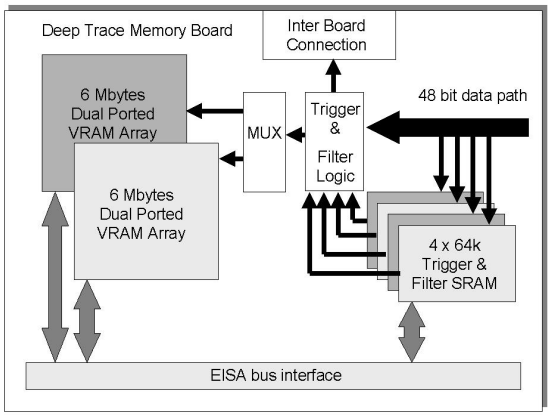

Figure 4.6: Trace memory board block diagram

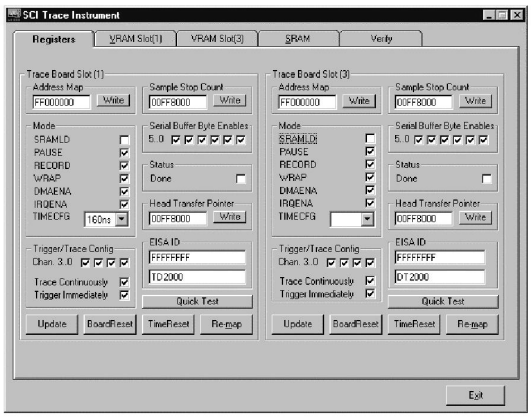

Figure 4.7: Trace instrument control GUI

4.1.3 Trace Memory Boards

The deep trace memory boards are inserted into the PC's Extended Industry Standard Architecture (EISA) slots. Each trace board contains 12 Mbytes of dual ported Video Random Access Memory (VRAM); one port receives trace data from the probe adapter while the second connects the trace memory to the EISA I/O bus of the trace instrument. The first trace board inserts absolute time stamps following each packet into the trace memory while the second board inserts relative time stamps of a finer time resolution.

4.1.4 Control Software

The trace board's operation is controlled by a suite of driver, API and Graphical User Interface (GUI) application software through the EISA bus (see Fig. 4.7). The trace tool API may be employed by the user to adapt the instrument to their specific needs.

The trace boards implement a trigger and filter mechanism via additional Static Random Access Memory (SRAM) that is used to store associative match patterns (see Fig. 4.6). Both boards are interconnected to enable triggering over the full 96-bit sample width. Furthermore, it is intended that a number of instruments could be interconnected for a synchronised trace data acquisition on two or more target nodes. The trigger mechanism provides four-level triggering. The filter and trigger patterns are configured through the instrument's API. A trigger and filter GUI implementation is shown in Fig. 4.8. A view of the trace board memory contents is provided through the instrument's control software (see Fig. 4.9). Fig. 4.1 and Fig. 4.2 show the instrument, which is packaged within a portable (Lunchbox) PC. Two of these instruments were constructed.

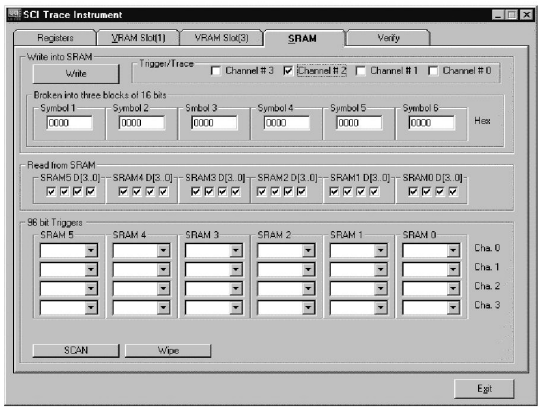

Figure 4.8: Trace instrument trigger and filter GUI

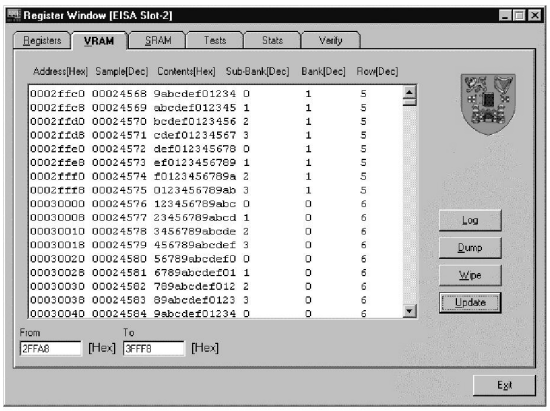

Figure 4.9: Trace memory viewer

4.2 SCI Trace Database

The trace instrument employs a relational database to store and analyse trace data [CM99]. The trace database is designed to accommodate all SCI packet types encountered on SCI cable links and Blinks. The following packet classification satisfies the Blink specification [Dol96] and the SCI IEEE standard [Ins93]. This categorisation is used for the decoding, the trace database storage and the retrieval of SCI and Blink packets.

4.2.1 SCI Cable-link Tables

Type 1 Request-send-packet with extended header and 0 byte data

Type 2 Request-send-packet with extended header and 16 byte data

Type 3 Request-send-packet with extended header and 64 byte data

⋮

Type 17 Response-send-packet with 256 byte data

Type 18 Response-echo-packet

Type 19 Idle Symbols

Type 20 Sync packets

4.2.2 Blink Tables

Type 21 Encapsulated request-send-packet with extended header and 0 byte data

Type 22 Encapsulated request-send-packet with extended header and 16 byte data

Type 23 Encapsulated request-send-packet withextended header and 64 byte data

⋮

Type 34 Encapsulated response-send-packet with 16 byte data

Type 35 Encapsulated response-send-packet with 64 byte data

Type 36 Encapsulated response-send-packet with 56 byte data

Subsequent to a trace acquisition, the instrument's control software writes the trace memory contents into two trace files. A Java decoding application reads the trace-data from those trace files and reunites the two 48 bit fractions into a full 96 bit-sample.

The software also detects the packet types as categorised above and decodes the packets. The trace database is broken up into a number of tables to accommodate the various

types of SCI packets. The database design provides space-optimised storage. The decoded SCI-packets are written into trace-database-table-files according to their packet type specification. These trace-database-table-files are used for a subsequent bulk import into the trace database.

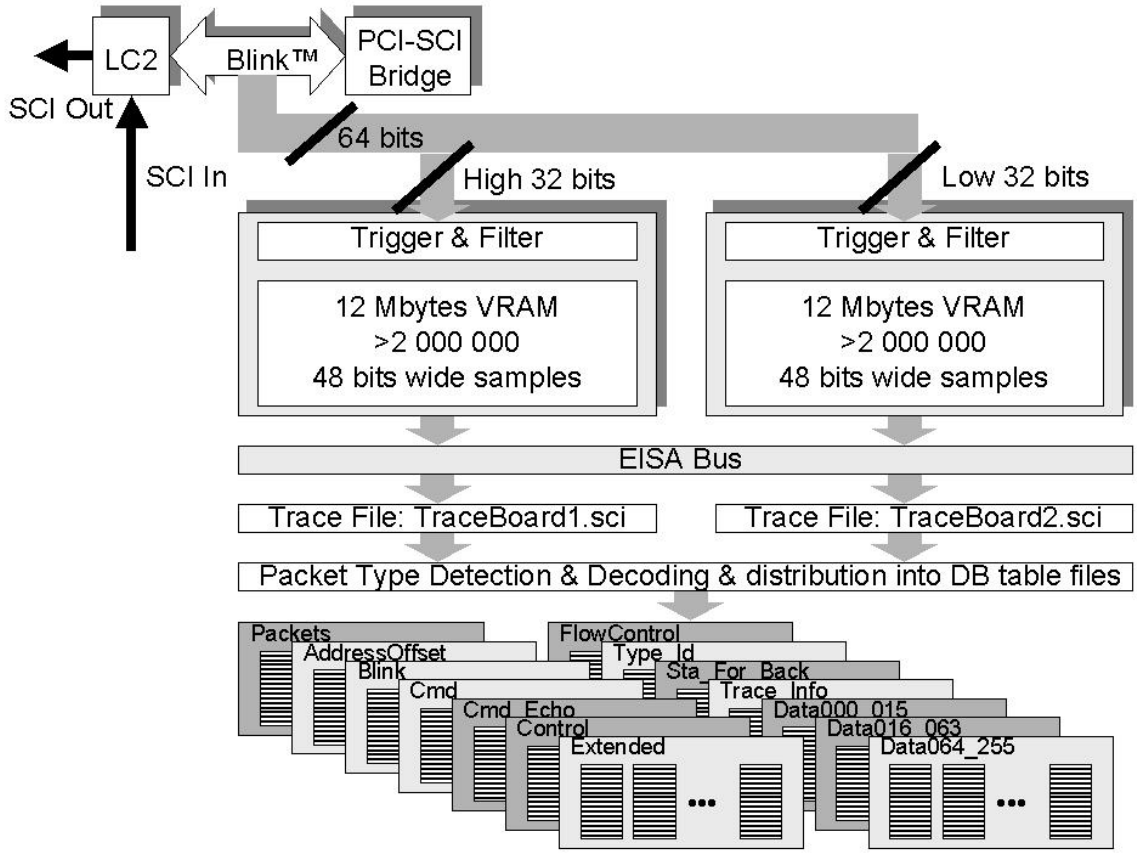

Figure 4.10: Trace data flow from Blink into DB-table-files

Each trace-database-table-file is associated with a table in the trace database. The file format reflects the database table design to accommodate bulk imports. Fig. 4.10 demonstrates how trace data flows from a target node's Blink into the trace-database-table-files. Fig. 4.11 gives an example of how a specific packet type, in this case a Response-send-packet with 64 bytes data, Type 16, is distributed into the appropriate trace-database-table-files.

A Trace-ID and a Packet-ID uniquely identify every SCI packet in every trace. Every trace-data-table contains these two IDs as primary keys. A main table is shared by all packets and contains a packet-type-ID but all packets occupy only a subset of the available tables. Fig. 4.12 shows the relations between the trace database tables.

The fields in the trace-database table exhaustively enumerate SCI-packet information, preserving the maximum level of detail, e.g. targetID, command type, sourceID, etc. This

allows for very detailed queries, e.g. all request-send packets with targetId = X, sourceId = Y and addressOffset between A and B.

The user may give meaningful interpretation to trace data fields through the implementation of additional tables and additional one-to-many relations. The design allows the analysis of subsets of the packet's data while maintaining a relation to the full packet information e.g. a query result-set is easily associated with the full packet information.

4.2.3 Trace Database Performance

A preliminary investigation has shown that direct Structured Query Language (SQL) insertions of individual packets subsequent to the packet's decoding are too expensive. The estimated execution time exceeds 1 hour for a full trace while a bulk import into the trace instrument's MS Access database can be achieved in less than 10 minutes. MS SQL-Server imports are even less time consuming.

```
CREATE PROCEDURE [SCI_Packet_Type_01] AS SELECT
    SCI_Packets.TraceId,
    SCI_Packets.PacketId,
    SCI_Packets.Packet_Type_Id,
    SCI_Packet_Type_Id.Packet_Type_Description,

    ... (place holder 33 fields in 6 tables)

    SCI_Packets.relative_Time_2,
    SCI_Packets.relative_Time_3
FROM SCI_Packets
  INNER JOIN SCI_FlowControl ON
    SCI_Packets.TraceId = SCI_FlowControl.TraceId AND
    SCI_Packets.PacketId = SCI_FlowControl.PacketId
  INNER JOIN SCI_Cmd ON
    SCI_Packets.TraceId = SCI_Cmd.TraceId AND
    SCI_Packets.PacketId = SCI_Cmd.PacketId
  INNER JOIN SCI_Control ON
    SCI_Packets.TraceId = SCI_Control.TraceId AND
    SCI_Packets.PacketId = SCI_Control.PacketId
  INNER JOIN SCI_AddressOffset ON
    SCI_Packets.TraceId = SCI_AddressOffset.TraceId AND
    SCI_Packets.PacketId = SCI_AddressOffset.PacketId
  INNER JOIN SCI_Extended ON
    SCI_Packets.TraceId = SCI_Extended.TraceId AND
    SCI_Packets.PacketId = SCI_Extended.PacketId
  INNER JOIN SCI_Trace_Information ON
    SCI_Packets.TraceId = SCI_Trace_Information.TraceId
  INNER JOIN SCI_Packet_Type_Id ON
```

```
30      SCI_Packets.Packet_Type_Id = SCI_Packet_Type_Id.Packet_Type_Id
    WHERE (SCI_Packets.TraceId = 3) AND
32        (SCI_Packets.PacketId = 40200)
```

Listing 4.1: SQL-query for a Type 1 Request-send-packet with extended header and 0 bytes data

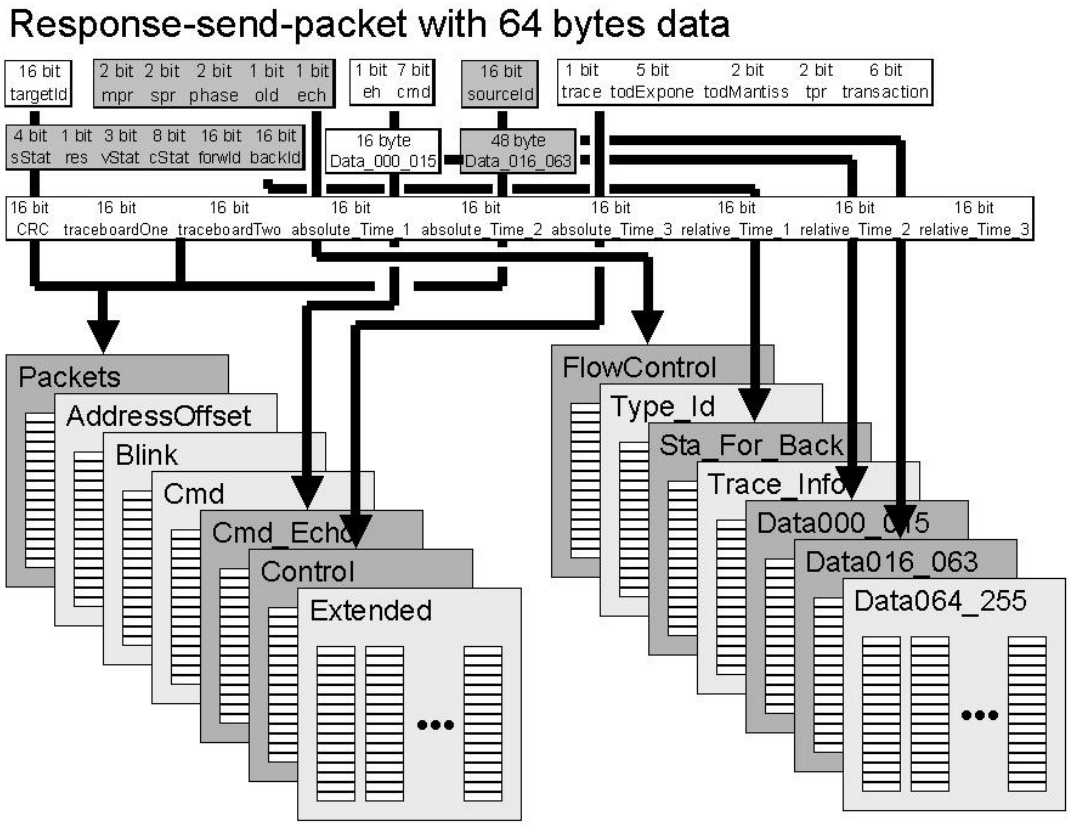

Figure 4.11: Packet trace database distribution

The SQL-query in Listing 4.1 reconstructs a specific Type 1 Request-send-packet with extended header and 0 bytes data. The packet has a TraceID = 3 and a PacketID = 40200 and is retrieved from a trace database with 100,000 packets. The query must retrieve 39 fields in 6 tables in order to reassemble this packet and it's associated trace information.

Even using a Microsoft SQL-Server 7.0 on a 450 MHz Intel Pentium II with 128 MB memory required less than 1 second. The same query into an MS Access database required about 15 seconds, so a Microsoft SQL-Server is a preferable database engine.

4.3 SCI Trace Data Presentation and Analysis

The primary trace data acquisition, the decoding and the trace bulk import are associated with the trace instrument itself. But trace data will in all likelihood be transferred to a remote node for performance and accessibility reasons. Fig. 4.14 provides a system overview. Trace data is easily transferred from one trace database to another. A remote node hosts a web server and a Java trace database server. Client nodes may load trace viewer and analysis applets into their web browser. The trace instrument can behave as a client in this scenario. The applet establishes a socket connection to the trace database server.

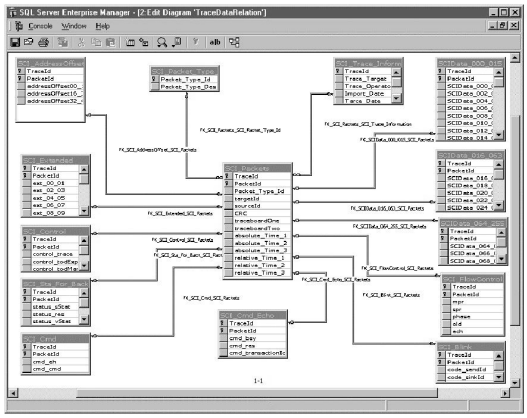

Figure 4.12: Trace database relations

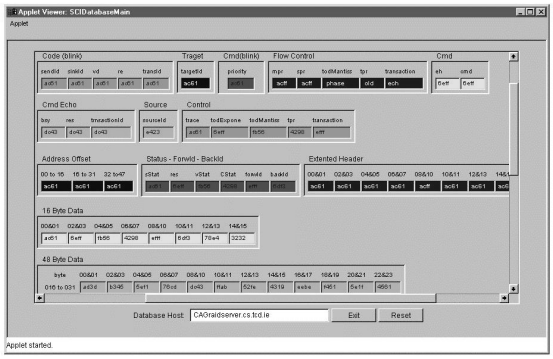

Figure 4.13: Java Packet Viewer

4.3.1 Java Trace Database Server

The Java database server creates a new thread for every connecting client, thereby allowing concurrent access from multiple clients. The client applet initiates the server to connect to a particular trace database either on the local node or a remote node. The Java server establishes the trace database connection through an Open DataBase Connectivity (ODBC) server. The trace database server holds a set of prepared SQL statements. A client may invoke a specific prepared SQL statements and forward parameters to the server. The server then invokes the statement with the inserted client parameters and returns the query result-set to the applet.

4.3.2 Java Packet Viewer Applet

Fig. 4.13 shows an SCI packet viewer applet. The user provides the applet with a TraceID and PacketID. The software then initially queries the packet type and adjusts its layout accordingly. A subsequent type-specific query for the full set of trace-data provides the applet with the required data.

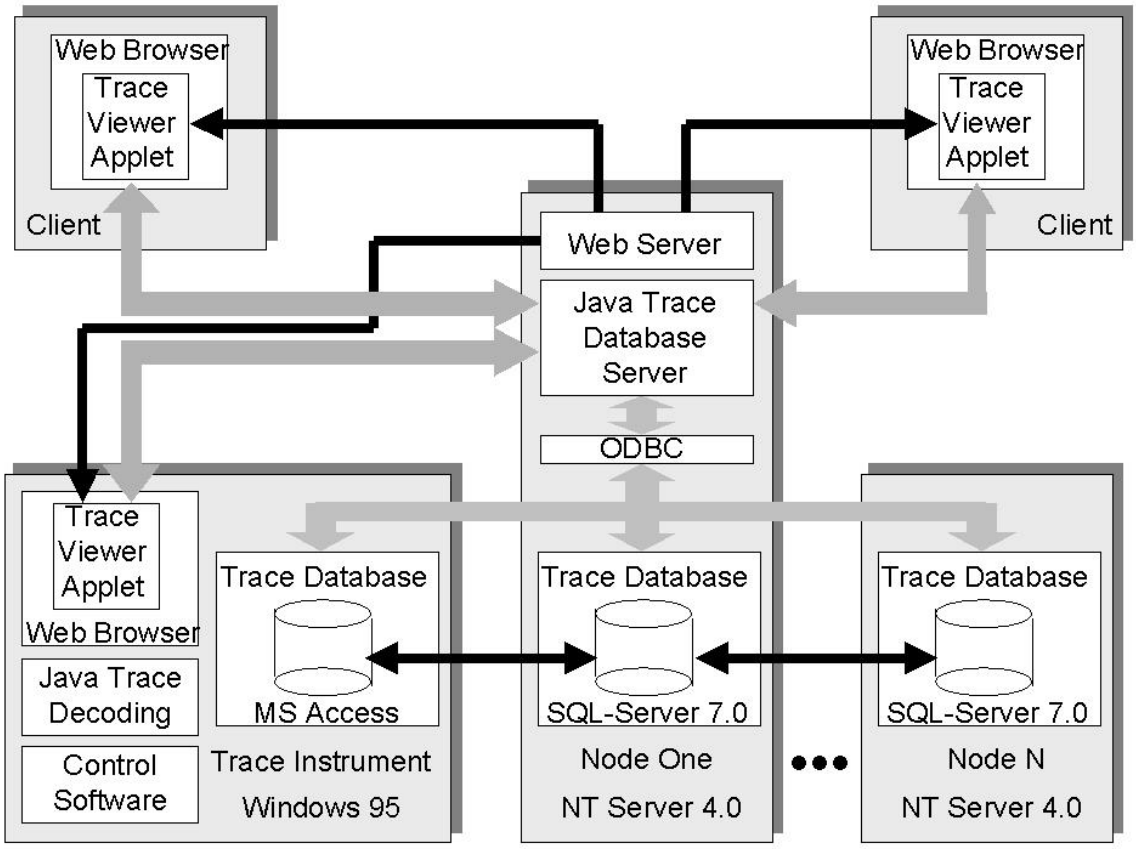

Figure 4.14: Trace system software

The above software is collectively called System Area Network Trace Analysis (SANTA). It has been distributed to SCILAB in Norway and the Department of Physics and Informatics at the University of Oslo. It has also been grid-enabled (as SANTA-G) [BK+05] and deployed on both the EU CrossGrid (23 sites) [Cro06] and the Irish Grid-Ireland (18 sites) [Gri06] grid infrastructures within Stuart Kenny's PhD thesis [Ken06]. SANTA-G includes SCI and Ethernet tracing, and is also the basis for the Grid-wide Intrusion Detection System (GIDS) deployed on Grid-Ireland. GIDS [KC04, KC05] will be further developed for active security within the recently approved int.eu.grid EU project [int06]. In recent times SCI traces tend to be acquired using SCILAB's SCITRAC cable tracer and a modern logic analyser (e.g. Tektronix 7014) rather than the trace instrument described above, and then analysed using SANTA-G.

4.4 Tuning and Verification of High Speed Interconnect Fabric Simulation Models

SCI trace data acquired and stored as described above can then be used to tune and verify SCI simulation models, which then may be applied to explore other SCI configurations and topologies. The framework for this is shown in Fig. 2.2.

4.4.1 SCI Simulation Model

A model of an SCI-node has been implemented in OPNET-modeller. OPNET was originally developed at Massachusetts Institute of Technology (MIT), and was introduced in 1987 as the first commercial network simulator. The node model is designed to simulate SCI at the packet level for increased simulation efficiency. Still, it achieves symbol level accuracy by exploiting the fact that the time interval between the arrival of the first and the last symbol of a packet is a function of its length. The model is object oriented in the sense that there are separate modules (objects) for units like the input and output buffers, the bypass First In First Out (FIFO), the stripper, the multiplexer at the output end, etc.. It contains a full implementation of the go-bit based flow control, as well as the A-B ageing scheme of the retry protocol. The formation of ringlets or bigger topologies consisting of both rings and switches can, to a large extent, be done through drag and drop functionality.

The usefulness of a simulation model is dependent on the statistics it assumes for the physical system. This is related to the statistics it is able to generate when it runs. This SCI model includes a comprehensive set of "points of measurements". Fig. 4.16 shows where these points are located in the logical model. The points of measurements are:

1. The number of packets held in the output buffer. A new sample is generated at each packet insertion or removal from the output buffer.

2. The number of packets held in the active buffer. A new sample is generated at each packet insertion or removal from the active buffer.

3. Output throughput, which is measured in symbols per clock cycle. The output throughput is collected when ever a new packet arrives at the output buffer.

4. Bypass throughput, which is measured in symbols per clock cycle. The bypass throughput is collected whenever a new packet arrives at the output buffer.

5. Node throughput, which is measured in symbols per clock cycle; Node throughput = Output throughput + Bypass throughput.

6. Idles between arriving packets. The time between packets arriving at the node's input link, which indicates the link's load. The link is run at maximum capacity if there is only one idle between arriving packets. In this case the load factor equals 1.

7. Number of send packets held in the input buffer. A new sample is generated at each packet insertion or removal from the input buffer.

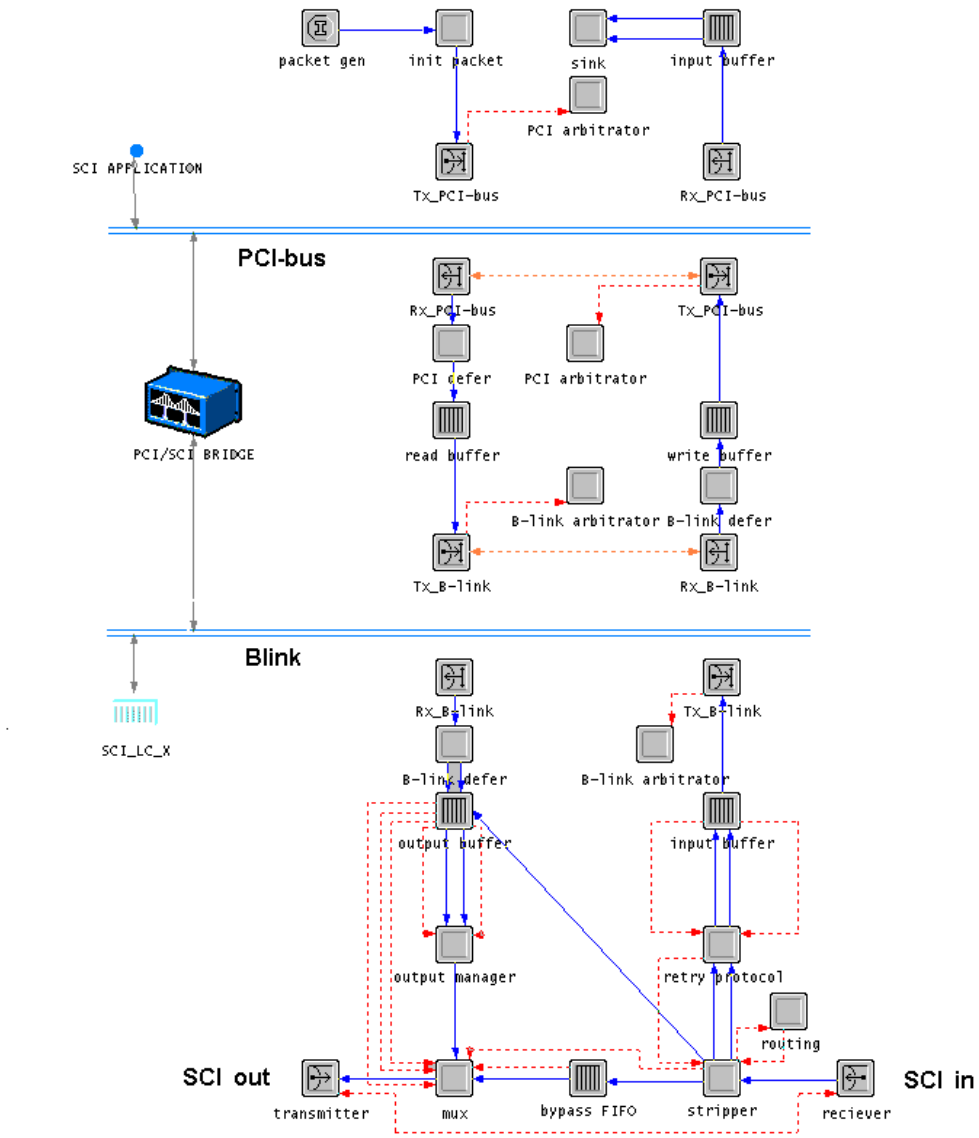

Figure 4.15: SCI node OPNET model including PCI-bridge and Blink

The SCI interface simulation generates SCI packets in the packet_genmodule. Fig. 4.15 shows the individual simulation components. The packet_gen module is shown above the PCI-bus. The Model assigns the packet type randomly and the inter-arrival time of the packets is dictated by a Probability Density Function (PDF).

4.4.2 SCI Simulation Model Tuning

The relational trace database stores SCI trace data including time stamps. These time stamps are associated with individual packets. Timing details are appended to the packet during the non-intrusive acquisition of trace data. The trace database makes these absolute timestamps available for further analysis. Consequently trace data provides accurate information about the temporal behavior of the system under test. Time stamped packets are essential for a statistical description of the system's behavior.

The system provides the means to monitor interconnect traffic by snooping on the link cable at three different locations SCI IN and/or SCI OUT and/or the Blink. Fig. 4.15 shows the three snoop targets in an OPNET representation of the SCI interface. Traces acquired from these three locations complement each other by providing different subsets of the full set of ringlet and Blink traffic that passes through the LC. The LC receives ringlet packets on the SCI IN port. These packets may be addressed to this node and routed into the LC or forwarded to the SCI OUT port if addressed to a different node. The LC processes the packet further by decoding the packet type and processing it accordingly, e.g. an echo packet acknowledges a previous transaction and will be absorbed in the LC but send-request or send-response packets are encapsulated and forwarded onto the Blink because they require the assistance of the PCI/SCI Bridge. Send-requests or send-responses also require the generation of echo packets to acknowledge the transaction to the source node. This echo packet is transmitted on the SCI OUT port.

The LC also receives encapsulated send-requests or send-responses on its Blink port from the PCI/SCI Bridge and transmits them on the SCI OUT port. This description of LC transactions is by no means exhaustive but should demonstrate that monitoring on the Blink is restricted to the subset of send-request and send-response packets whereas snooping on the link provides information about link-level related transactions. See [Ins93] for full details. Due to the nature of unidirectional link traffic, monitoring on a single link cable limits the visibility of node transactions, e.g. snooping on the SCI OUT cable enables the observation of a send-request packet but will neither see the associated echo packet nor a send-response packet from the responding node. It will see the echo packet that acknowledges the reception of a send-response packet and completes the transaction. On the other hand monitoring the Blink would allow the observation of both the send-request and the send-response packets but not the echo packets nor any passing traffic.

The above implies that the most comprehensive tracing would require the use of three trace data acquisition channels and a synchronized trigger mechanism. The trace database actually allows one to relate 2+n traces that are monitored at different locations whether at the same node or remote notes in the SCI fabric. Queries for particular trace data attributes, e.g. transaction ID in conjunction with time constraints, can extract packets related to a specific transaction.

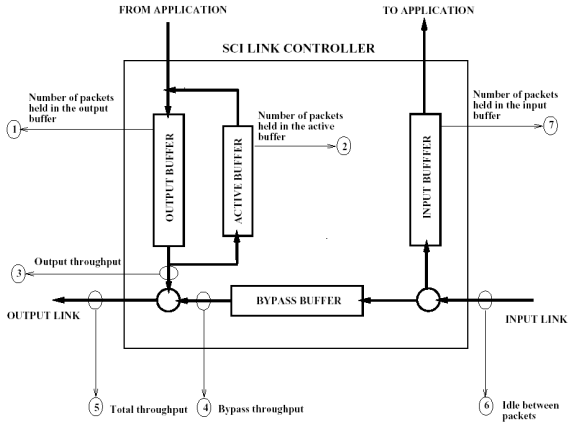

Figure 4.16: The points of measurement where the statistic is collected for an SCI node. For convenience the source and the destination device for the input and the output buffer are called "TO APPLICATION" and "FROM APPLICATION". In a real SCI node the buffers are physically connected to a bridge.

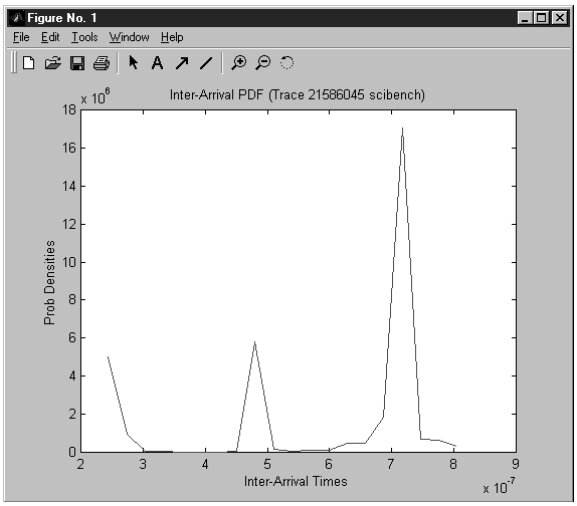

Figure 4.17: Probability density function

A trace data analysis can take advantage of the completeness of the information held within the trace database and the relational operations that can be performed upon it. First it is necessary to extract specific subsets of trace data in order to generate PDFs that can be used in the SCI simulation model described in Section 4.4.2.

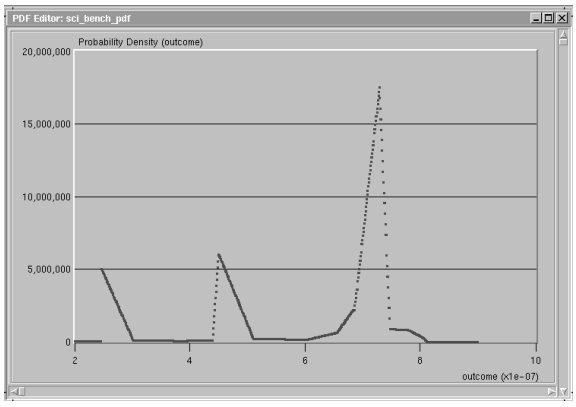

Figure 4.18: Load definition

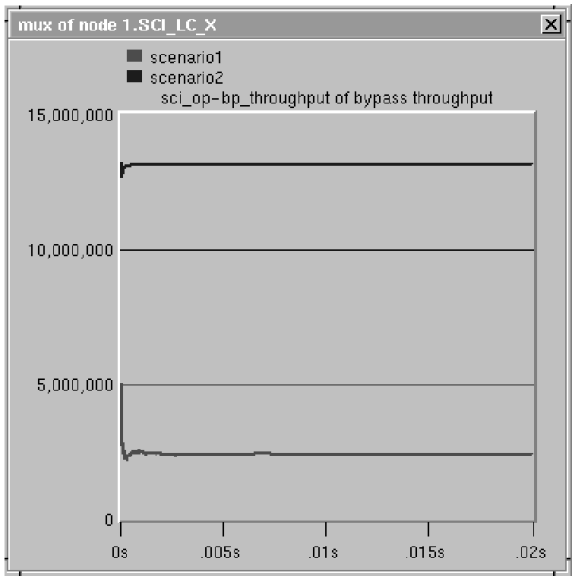

Figure 4.19: Model output

The SCI interface simulation generates SCI packets in the packet_gen module. Fig. 4.15 shows the individual simulation components. The packet_gen module is shown above the PCI-bus. As previously indicated the model assigns the packet type randomly and the inter-arrival time of the packets is dictated by a PDF. Currently the SCI model uses PDFs provided by the OPNET package but the software allows the definition of PDFs by the user. For this purpose, one can extract through an SQL query a relevant subset from the trace database and compute the PDFs from timestamps that are associated with each packet. The default model uses a uniform PDF with limits set to 6.0E-7 and 6.0E-6 seconds. The computed PDF, however, shows a strong deviation from this assumption, see Fig. 4.17. The computed PDF function represents the probability density of packets generated by real system loads. When the default PDF is replaced by the computed PDF the model is then stimulated with realistic system loads. Fig. 4.18 shows the load definition derived from measurements shown in Fig. 4.17. In a further step the SCI model can then be stimulated with both a realistic system load and the default uniform PDF. Fig. 4.19 compares the resulting predicted system throughput. The significance of this has yet to be established.

4.5 Summary

The trace instrument seen in Fig. 4.1 and Fig. 4.2 provides a non-intrusive method of measuring SCI interconnect traffic and consequently will not influence the temporal behaviour of the system. It enables researchers and developers to analyse the true temporal behaviour of clusters made up of standard components. The employment of a relational database for trace-data storage provides the user with well understood and easy-to use tools to extend and to adapt the predefined database queries to their specific needs. The use of Java and SQL makes the software platform independent. There is significant potential for enhancement through the implementation of important methods for the analysis and visualisation of the dynamic behaviour of parallel processes [KDH+95], and software continues to be developed by others.

The non-invasive acquisition of interconnect traffic allows analyses of the true temporal behaviour of compute clusters. The advantage of offline query-based filtering of traces, as opposed to real-time filtering within the trace acquisition instrument, is the ability to analyse numerous different aspects of the same traces without the need to re-perform the trace acquisition. Repeated acquisitions may not lead to similar results, thereby creating uncertainty about the correctness of the analysis. The method described above generates realistic system load statistics for the SCI model. Furthermore, it enables the tuning and the verification of the SCI model's parameterisation. This framework has the potential to support similar interconnects, in particular InfiniBand [Inf00a, Inf00b].

Chapter 5

State Estimation of a Single Compute Node

The tuning of SCI topology simulations requires interconnect event statistics that were gained from the analysis of SCI trace data. The notion of viewing interconnect events in statistical terms inspired the idea of modeling PMC readings as random processes. A Kalman filter could then be used to improve the accuracy of PMC readings and merge multiple readings into a common state vector. This in turn would allow the calculation of derived (unobservable) measurements across an entire DSM cluster.

Modern CPU and other subsystems provide PMC registers that can be configured to measure various aspects of the CPU or subsystem behaviour. In this chapter I propose that an algorithm based on Kalman filtering can implement a discrete MMSE filter to fuse concurrent and sequential observations of PMC event readings into the filter's state-vector. Experimental results then test this hypothesis.

5.1 The Estimation Algorithm

The algorithm can provide a system-wide optimal estimate of the PMC readings that may be merged into a filter's state vector as subsets of the total set of counter observations. This may include all processors and subsystems in the architecture. The algorithm can also use prediction to compensate for latencies in the counter reading acquisition process.

Fig. 5.1 and Fig. 5.2 are two examples of sampled PMC readings with a sample time of $\Delta t = 0.04sec$. They show the histograms and samples over time of the *Number of lines allocated in the L2* on a two way SMP machine.

The system software that implements the acquisition and processing of PMC register readings introduces additional system dynamics and distortions. This software includes firstly a driver that provides access to the registers, secondly a low-level user-space API that implements the control of the registers and finally the filter algorithm itself [Pet02].

The computer's architecture and operating system design can result in non-determinism in the execution time of the above mentioned software; this leads to a noise corruption of the reading that is otherwise only corrupted by quantisation noise.

5.1.1 The Filter

The filter's objective is to provide an optimal to sub-optimal estimation and prediction of n performance counter readings. This estimate or prediction is based on imperfect and incomplete counter data; imperfect if corrupted by quantisation noise and variations in the counter acquisition process, and incomplete if counter readings are multiplexed rather than read in parallel. A Kalman filter uses stochastic models to accommodate these properties. Stochastic models can overcome problems arising from imperfect system models and noise that cannot be modeled deterministically.

The filter incorporates the system dynamics into the model. In its fundamental form the model is assumed to be linear and driven by *white Gaussian noise* and the PMC readings that are taken from this process are also assumed to be corrupted by white Gaussian noise. These measurements are presumed to have a linear relationship with the system model.

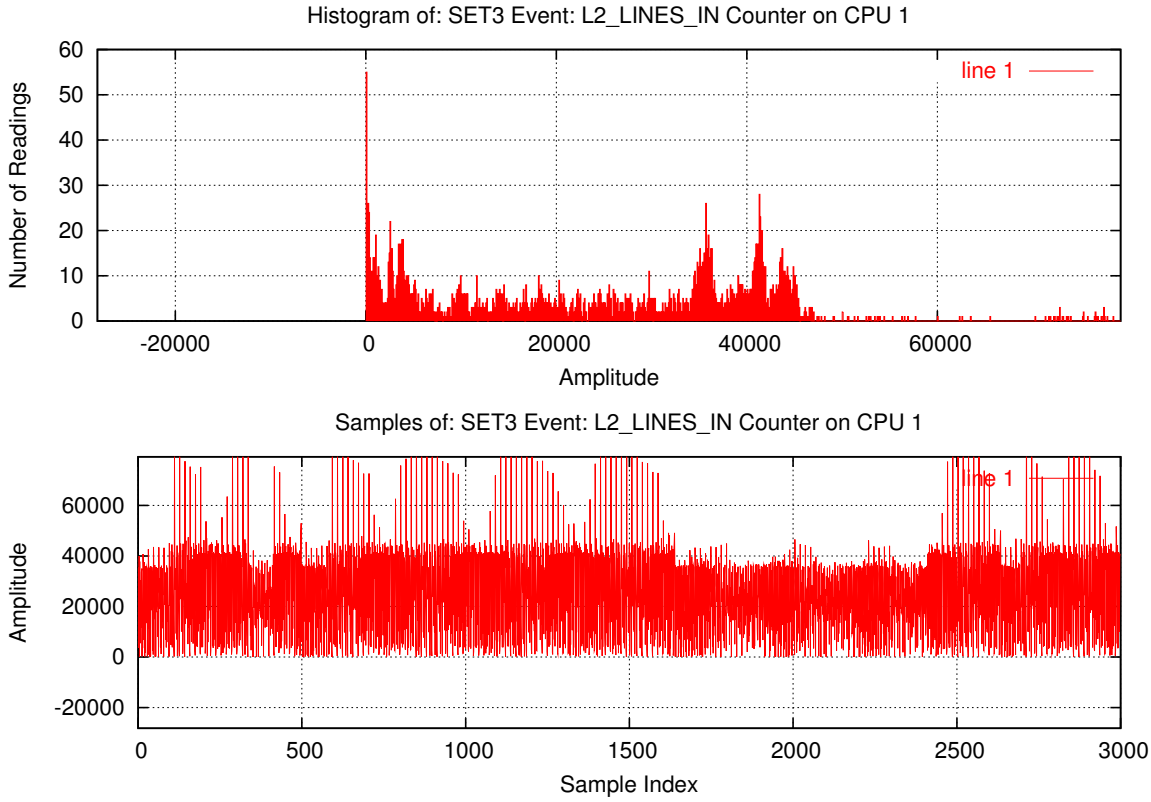

Figure 5.1: Histogram and samples for event number 26 symbol L2_LINES_IN on CPU 1

Note that an EKF [GA01, BH97] can be used if either the system model is nonlinear or the measurements are not linearly related with the system model. However, a linear Kalman Filter proves adequate for the observation of PMC readings.

The filter is a recursive algorithm that processes measurements (in our case PMC readings) of any quality in order to derive an optimal estimate from these data. The algorithm must be fed with information about the process and measurement dynamics. Furthermore, statistics about the measurement noise and the process model's uncertainty must be supplied to the filter algorithm along with an initialisation. Fig. 5.3 demonstrates the recursive nature of the algorithm. Fig. 5.3 shows also one possible implementation, but there are algebraically equivalent forms that have computational advantages and disadvantages.

The filter provides an optimal estimate by combining the measurements and the knowledge of the system to minimise the error between the model and the measurements statistically. To this end the algorithm propagates the *conditional probability density* of the system's state. The *probability density* is conditioned by the discrete time PMC samples and represents the uncertainty of the current state of the system.

Consider the Kalman filter algorithm presented in Fig. 5.3. The discrete Kalman Filter matrix block diagram in Fig. 5.4 will help to discuss the operations of the filter.

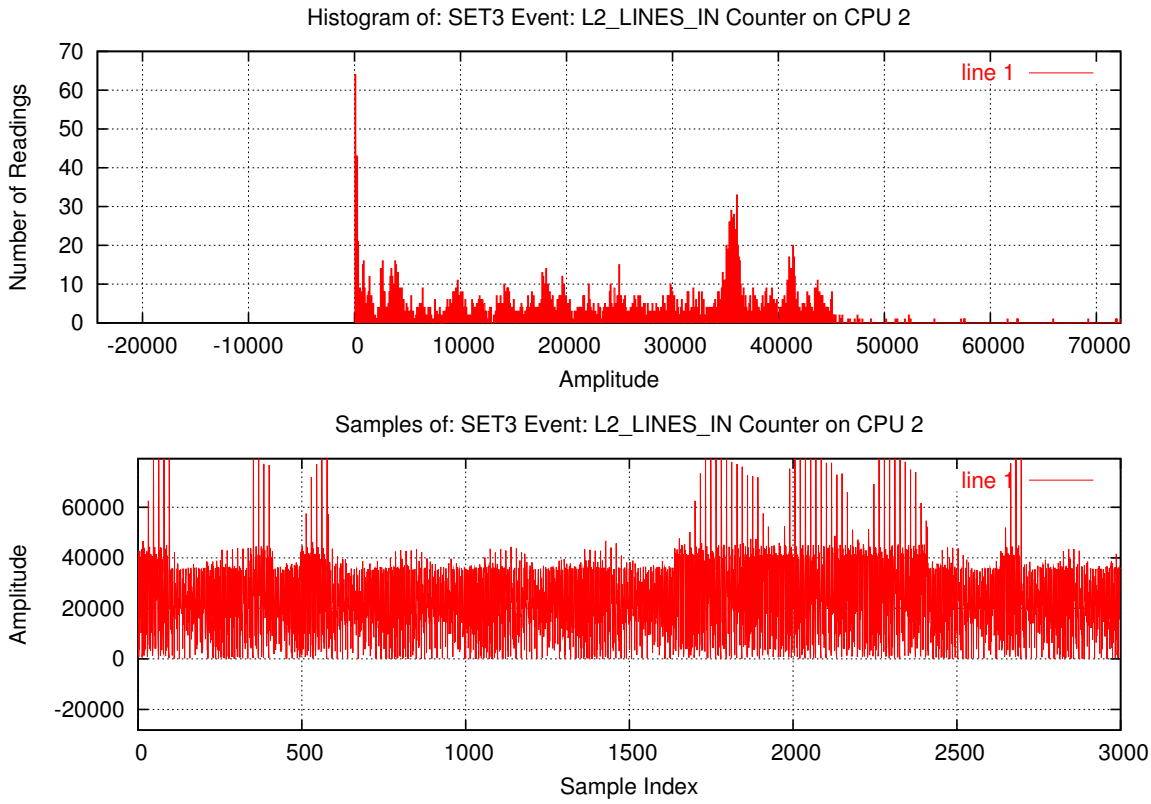

Figure 5.2: Histogram and samples for event number 26 symbol L2_LINES_IN on CPU 2

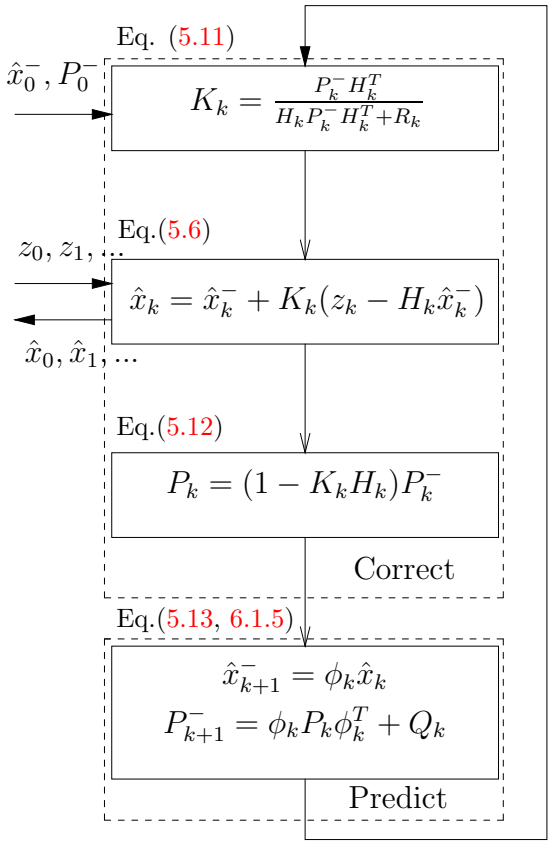

Figure 5.3: Discrete Kalman Filter Algorithm

The random processes that represent the current state of activities in measured parts of a component's micro-architecture are modeled with Eq.(5.1), where $\boldsymbol{x}_k$ is the *state vector* of the linear dynamic system at sample time t_k, $\boldsymbol{\phi}_{k-1}$ is the *state transitions matrix* and $\boldsymbol{w}_k$ is a sequence with a covariance determined by the *covariance matrix* $\boldsymbol{Q}_k$ of the process noise associated with the system's state dynamics. The state transitions matrix $\boldsymbol{\phi}_{k-1}$ defines how the current state vector $\boldsymbol{x}_{k-1}$ influences the state vector $\boldsymbol{x}_k$ at time t_k.

$$\boldsymbol{x}_k = \boldsymbol{\phi}_{k-1}\boldsymbol{x}_{k-1} + \boldsymbol{w}_k \tag{5.1}$$

$$\hat{\boldsymbol{x}}_k = \boldsymbol{\phi}_{k-1}\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{K}_k(\boldsymbol{z}_k - \hat{\boldsymbol{x}}_k\boldsymbol{H}_k) \tag{5.2}$$

The *PMC Random Process* section of the discrete matrix block diagram in Fig. 5.4 shows a graphical representation of Eq. (5.1). That section represents the real physical process, for example an instruction-set-processor that executes a program and consequently generates counter events that can be sampled with PMC registers according to Eq. (2.1). This random

process is then modeled in the Kalman filter Eq.(5.2). This is also shown in the *Kalman Filter* section of Fig. 5.4. The process dynamics terms $\boldsymbol{\phi}_{k-1}\boldsymbol{x}_{k-1}$ and $\boldsymbol{\phi}_{k-1}\hat{\boldsymbol{x}}_{k-1}$ in Eq.(5.1) and Eq.(5.2) respectively are equivalent but the Kalman filter substitutes the state vector $\boldsymbol{x}$ with the estimated state vector $\hat{\boldsymbol{x}}$. Furthermore the process noise term $\boldsymbol{w}_k$ in Eq.(5.1) that accounts for the uncertainty of the process model is replaced in the Kalman filter Eq.(5.2) with a blending factor $\boldsymbol{K}_k$ that weights the error between the measurement vector $\boldsymbol{z}_k$ (in this case performance counter readings) and the estimated measurement vector $\hat{\boldsymbol{z}}_k{=}\hat{\boldsymbol{x}}_k^-\boldsymbol{H}_k$ (the *super minus*$^-$ is explained later).

This blend factor matrix $\boldsymbol{K}_k$ is also known as *Kalman Gain* and its calculation is part of the recursive Kalman filter algorithm shown in Fig. 5.3. This gain minimises the error in the estimated state vector $\hat{\boldsymbol{x}}$.

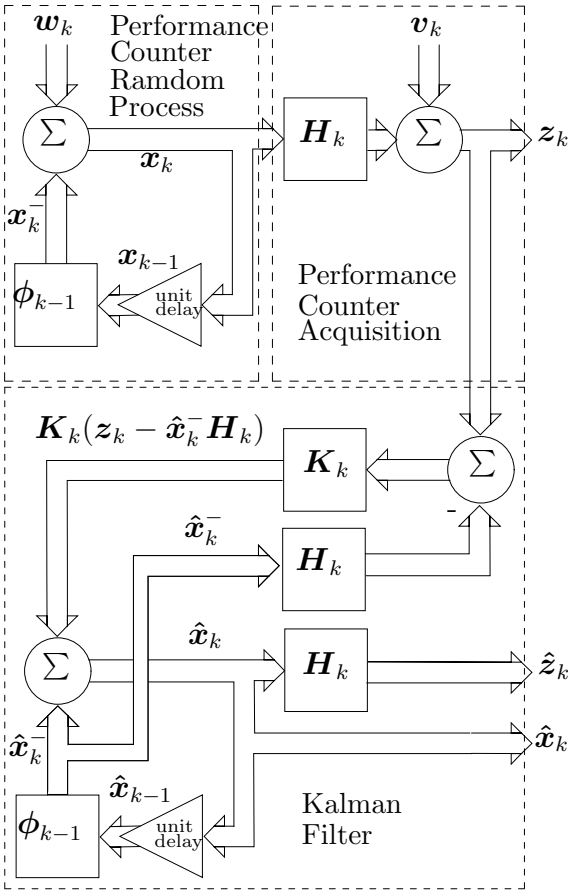

Figure 5.4: Discrete Kalman Filter Matrix Block Diagram

$$\boldsymbol{z}_{k+1} = \boldsymbol{H}_k\boldsymbol{x}_k + \boldsymbol{v}_k \tag{5.3}$$

The discrete measurement model of Eq.(5.3) represents a linear relationship between the

state vector $\boldsymbol{x}_k$ and the measurement vector $\boldsymbol{z}_k$ at the time t_k. The *measurement sensitivity matrix* $\boldsymbol{H}$ defines this relationship for the noiseless case. The measurement noise $\boldsymbol{v}_k$ is added to this. The covariance of the vector $\boldsymbol{v}_k$ is able to be described by the measurement noise covariance matrix $\boldsymbol{R}_k$.

Eq.(5.4) and Eq.(5.5) assume that the process noise $\boldsymbol{w}_k$ and measurement noise $\boldsymbol{v}_k$ are zero-mean Gaussian processes. A further assumption is that $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are uncorrelated.

$$p(\boldsymbol{w}_k) \sim N(0, \boldsymbol{Q}_k) \tag{5.4}$$
$$p(\boldsymbol{v}_k) \sim N(0, \boldsymbol{R}_k) \tag{5.5}$$

An error covariance matrix $\boldsymbol{P}_k^-$ of the estimated state vector $\hat{\boldsymbol{x}}_k^-$ must be calculated prior to blending the performance counter reading into the state vector. $\boldsymbol{P}_k^-$ in Eq.(5.7) represents the expectation of the estimation error $\boldsymbol{e}_k^- = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^-$.The *a priori* nature (prior to blending the $\boldsymbol{z}_k$ vector) of $\boldsymbol{P}_k^-$ and $\hat{\boldsymbol{x}}_k^-$ is indicated by the *super minus*$^-$.

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{z}_k - \hat{\boldsymbol{x}}_k^- \boldsymbol{H}_k) \tag{5.6}$$
$$\boldsymbol{P}_k^- = E[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^-)(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^-)^T] \tag{5.7}$$
$$\boldsymbol{P}_k = E[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)^T] \tag{5.8}$$

The a posteriori state estimate $\hat{\boldsymbol{x}}_k$ was defined by Eq.(5.2) and can be rewritten as Eq. (5.6).

In order to derive a *Kalman Gain* $\boldsymbol{K}_k$ that minimises the error in $\hat{\boldsymbol{x}}_k$, the a posteriori error covariance matrix $\boldsymbol{P}_k$ must also be defined as in Eq.(5.8). The optimal Kalman gain $\boldsymbol{K}_k$ minimises the a posteriori error covariance matrix $\boldsymbol{P}_k$. To this end Eq.(5.3) can be substituted into Eq.(5.6) and the resulting equation for the *a posteriori state estimate* $\hat{\boldsymbol{x}}_k$ can then substitute for $\hat{\boldsymbol{x}}_k$ in Eq.(5.8). Subsequently the expectation of this new expression for $\boldsymbol{P}_k$ can be calculated. This leads to Eq.(5.9), a generic description of the a posteriori error covariance matrix for any Kalman gain matrix values.

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_k^-(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)^T\boldsymbol{K}_k\boldsymbol{R}_k\boldsymbol{K}_k^T \tag{5.9}$$

If the derivative of the trace of $\boldsymbol{P_k}$ in Eq. (5.9) with respect to $\boldsymbol{K_k}$ is set equal to zero, it becomes possible to solve for the Kalman Gain. This way the trace of $\boldsymbol{P_k}$ is minimised

$$\boldsymbol{Q}_k = \begin{bmatrix} \frac{2\sigma^2}{\beta}\left[\Delta t - \frac{2}{\beta}(1-e^{-\beta\Delta t}) + \frac{1}{2\beta}(1-e^{-2\beta\Delta t})\right] & & \\ 2\sigma^2\left[\frac{1}{\beta}(1-e^{-\beta\Delta t}) + \frac{1}{2\beta}(1-e^{-2\beta\Delta t})\right] & & \\ & 2\sigma^2\left[\frac{1}{\beta}(1-e^{-\beta\Delta t}) + \frac{1}{2\beta}(1-e^{-2\beta\Delta t})\right] & \\ & & 2\sigma^2(1-e^{-2\beta\Delta t}) \end{bmatrix} \tag{5.10}$$

because the major diagonal of the a posteriori error covariance matrix $\boldsymbol{P_k}$ holds the sum of the mean-square error in the estimate of all the state variables in the state vector $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$. The result of the operation is presented in Eq.(5.11) and can also be seen in Fig. 5.3 as part of recursive Kalman filter algorithm.

$$\boldsymbol{K_k} = \frac{\boldsymbol{P_k^- H_k^T}}{\boldsymbol{H_k P_k^- H_k^T + R_k}} \tag{5.11}$$

Eq. (5.11) for the optimal Kalman gain matrix can be used to compute the error covariance matrix that is associated with the updated optimal estimate by substituting the Kalman gain $\boldsymbol{K_k}$ in the generic a posteriori error covariance matrix Eq.(5.9) by the optimal Kalman gain of Eq. (5.11). Subsequent to this substitution a number of alternative equations can be derived that have certain computational advantages. Eq. (5.12) presents one of the possible equations. This equation gives the error covariance for the update state vector $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$ and is also part of the recursive Kalman filter algorithm shown in Fig. 5.3.

$$\boldsymbol{P_k} = (\boldsymbol{I} - \boldsymbol{K_k H_k})\boldsymbol{P_k^-} \tag{5.12}$$

Eq. (5.6), Eq. (5.11) and Eq. (5.12) in the recursive Kalman filter algorithm of Fig. 5.3 requires either the a priori state vector $\hat{\boldsymbol{x}}_{\boldsymbol{k}}^-$ or the a priori covariance matrix $\boldsymbol{P_k^-}$. These are calculated in Eq. (5.13) and Eq. (6.1.5) respectively. Both equations use the state transition matrix $\boldsymbol{\phi_k}$ to propagate or predict $\hat{\boldsymbol{x}}_{\boldsymbol{k+1}}^-$ and $\boldsymbol{P_{k+1}^-}$ for the next time step $\boldsymbol{k+1}$. The state transition matrix $\boldsymbol{\phi_k}$ represents the process dynamics but the calculation of the a priori covariance matrix $\boldsymbol{P_k^-}$ requires the addition of uncertainty about the state estimate represented by the process noise covariance matrix $\boldsymbol{Q_k}$ as in Eq.(5.10).

$$\hat{\boldsymbol{x}}_{\boldsymbol{k+1}}^- = \boldsymbol{\phi_k}\hat{\boldsymbol{x}}_{\boldsymbol{k}} \tag{5.13}$$

$$\boldsymbol{P_{k+1}^-} = \boldsymbol{\phi_k P_k \phi_k^T + Q_k} \tag{5.14}$$

The equations that make up the Kalman filter algorithm as shown in Fig. 5.3 may be classified as equations that *Predict* and those that *Correct*. Eq. (5.13) and Eq. (6.1.5)

participate in the prediction process whereas Eq. (5.6), Eq. (5.11) and Eq. (5.12) handle the correction. This is also shown in Fig. 5.3.

5.1.2 PMC Process Models

The *Kalman Filter* implementation requires the PMC readings to be modeled as random processes. Equipped with an understanding of the basic operations of the filter we can now discuss the PMC process model. Both a *Gauss-Markov Process* and an *Integrated Gauss-Markov Process* have been investigated as a suitable random process model. Limited investigation indicated that an integrated Gauss-Markov process represents a PMC random process more accurately.

A stationary Gaussian process $\boldsymbol{X(t)}$ that has an exponential autocorrelation function is classified as a *Gauss-Markov* process. Eq.(5.15) defines the autocorrelation function for a Gauss-Markov process.

$$\boldsymbol{R_X(\tau) = \sigma^2 e^{-\beta|\tau|}} \tag{5.15}$$

This autocorrelation function Eq.(5.15) provides the full statistical description of the random process. The two parameters, time constant $\frac{1}{\beta}$ and variance $\boldsymbol{\sigma^2}$, can be calculated from an experimentally determined autocorrelation function. Section 5.2 deals with the determination of these parameters through an offline analysis of performance counter readings.

Fig. 5.5 shows the transfer function for an integrated Gauss-Markov process, with $\boldsymbol{u(t)}$ as unity white noise, $\boldsymbol{x_2}$ as Gauss-Markov process and $\boldsymbol{x_1}$ as integrated Gauss-Markov process. Eq.(5.16) then provides a continuous state space model for this random process.

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\beta \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \sqrt{2\sigma^2\beta} \end{bmatrix} u(t) \tag{5.16}$$

Eq.(5.17) and Eq.(5.10) define the discrete state transition matrix $\boldsymbol{\Phi}$ and the discrete covariance matrix $\boldsymbol{Q}$ respectively. These two equations are derived from the continuous state space model of Eq.(5.16).

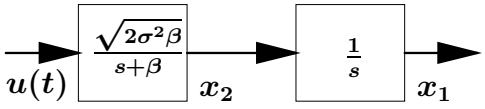

Figure 5.5: Integrated Gauss-Markov Block Diagram

$$\mathbf{\Phi_k} = \begin{bmatrix} 1 & \frac{1}{\beta_1}(1-e^{-\beta_1 \Delta t}) & 0 & 0 & \dots & 0 & 0 \\ 0 & e^{-\beta_1 \Delta t} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \frac{1}{\beta_2}(1-e^{-\beta_2 \Delta t}) & \dots & 0 & 0 \\ 0 & 0 & 0 & e^{-\beta_2 \Delta t} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & \frac{1}{\beta_n}(1-e^{-\beta_n \Delta t}) \\ 0 & 0 & 0 & 0 & \dots & 0 & e^{-\beta_n \Delta t} \end{bmatrix} \tag{5.18}$$

$$\mathbf{Q_k} = \begin{bmatrix} E[x_1x_1] & E[x_1x_2] & 0 & 0 & \dots & 0 & 0 \\ E[x_1x_2] & E[x_2x_2] & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & E[x_3x_3] & E[x_3x_4] & \dots & 0 & 0 \\ 0 & 0 & E[x_3x_4] & E[x_4x_4] & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & E[x_{2n-1}x_{2n-1}] & E[x_{2n-1}x_{2n}] \\ 0 & 0 & 0 & 0 & \dots & E[x_{2n-1}x_n] & E[x_{2n}x_{2n}] \end{bmatrix} =$$

$$= \begin{bmatrix} Eq.(5.21) & Eq.(5.22) & 0 & 0 & \dots & 0 & 0 \\ Eq.(5.22) & Eq.(5.23) & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & Eq.(5.21) & Eq.(5.22) & \dots & 0 & 0 \\ 0 & 0 & Eq.(5.22) & Eq.(5.23) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & Eq.(5.21) & Eq.(5.22) \\ 0 & 0 & 0 & 0 & \dots & Eq.(5.22) & Eq.(5.23) \end{bmatrix} \tag{5.19}$$

$$\mathbf{\Phi_k} = \begin{bmatrix} 1 & \frac{1}{\beta}(1-e^{-\beta \Delta t}) \\ 0 & e^{-\beta \Delta t} \end{bmatrix} \tag{5.17}$$

5.1.3 Integrated Gauss-Markov Process Model for n Counter Processes

Assume that every PMC reading is taken from an independent random counter process i with a specific autocorrelation function Eq.(5.20) below. This allows definition of the discrete state transition matrix ϕ and the discrete covariance matrix Q for n counters as in Eq. (5.19). What remains is the definition of all β_i and σ_i for every counter process in the state transition matrix ϕ and covariance matrix Q.

$$R_X(\tau) = \sigma_i^2 e^{-\beta_i |\tau|} \tag{5.20}$$

Eq.(5.21), Eq.(5.22) and Eq.(5.23) are the elements of Q_k. These elements are identical to those in Eq.(5.10) but they include the index i for the time constant $1/\beta_i$ and the variance σ_i^2.

$$E[x_{2i-1}x_{2i-1}]=\frac{2\sigma_i^2}{\beta_i}\left[\Delta t-\frac{2}{\beta_i}(1-e^{-\beta_i\Delta t})+\frac{1}{2\beta_i}(1-e^{-2\beta_i\Delta t})\right] \tag{5.21}$$

$$E[x_{2i-1}x_{2i}]=2\sigma_i^2\left[\frac{1}{\beta_i}(1-e^{-\beta_i\Delta t})+\frac{1}{2\beta_i}(1-e^{-2\beta_i\Delta t})\right] \tag{5.22}$$

$$E[x_{2i}x_{2i}]=2\sigma_i^2(1-e^{-2\beta_i\Delta t}) \tag{5.23}$$

5.2 PMC Acquisition and Offline Analysis

With the filter algorithm and the process model in place we can now continue to determine or estimate the time constants $1/\beta_i$ and the variances σ_i^2 for all available PMCs. Table A.1 on page 147 in appendix A provides a list of all the PMCs in an Intel PIII processor. Mikael Pettersson's [Pet02] *Linux x86 Performance-Monitoring Counters Driver* is used to sample PMC readings for a subsequent offline analysis. The same driver is used for the eventual real-time filter algorithm. If function $X(t)$ is assumed continuous over time T, an estimated autocorrelation function $\hat{R}_X(\tau)$ may be calculated according to Eq.(5.24) with $0 \leq \tau \ll T$. However, the sampled PMC readings are discrete, so the discrete Eq.(5.25) must be applied to calculate the estimated autocorrelation function $\hat{R}_X(n\Delta t)$ with N as the total number of samples. Remembering that $X(t)$ has an exponential autocorrelation function, the accuracy of $\hat{R}_X(n\Delta t)$ may be increased by calculating the mean of n estimated autocorrelation functions and fitting an exponential curve after subtracting the sample $mean^2$ from $\hat{R}_X(n\Delta t)$. The curve fitting computation provides the time constants $1/\beta_i$ and the variances σ_i^2.

$$\hat{R}_X(\tau)=\frac{1}{T-\tau}\int_0^{T-\tau}X(t)X(t+\tau)dt \tag{5.24}$$

$$\hat{R}_X(n\Delta t)=\frac{1}{N-n+1}\sum_{k=0}^{N-n}X(t)X(k+n) \tag{5.25}$$

It is now possible to look at the PMC data acquisition and analysis procedure in more detail by providing a list of steps that are required to estimate the time constants $1/\beta_i$ and the variances σ_i^2 for all available PMCs:

1. Acquisition of PMC readings (see section 5.2.1).
2. Visual inspection of sampled PMC readings and their histograms (see sections 5.2.2).
3. Autocorrelation calculation for sampled PMC readings (see sections 5.2.4).

4. Calculation of the mean autocorrelation for sampled PMC readings (see sections 5.2.6).
5. Estimation of $\mathbf{1/\beta_i}$ and $\boldsymbol{\sigma_i^2}$ for sampled PMC readings (see sections 5.2.8).
6. Visual inspection of histograms with superimposed Gaussian PDFs for sampled PMC readings (see sections 5.2.10).

The estimated time constants $\mathbf{1/\beta_i}$ and variances $\boldsymbol{\sigma_i^2}$ from the six step procedure are used to run Monte Carlo simulations for these PMC random processes. The output of the PMC Monte Carlo simulation is processed again with the same six step procedure. Section 5.2.3, section 5.2.5, section 5.2.7, section 5.2.9 and section 5.2.11 provide figures that show the output from the six step procedure that processes simulated PMCs. These sections follow the relevant sections that deal with the processing of sampled PMC readings and therefore allow for a comparison between simulated and sampled data.

See section 8.3 on page 141 for a description of the work loads.

5.2.1 Acquisition of PMC readings

As mentioned in section 5.2 it is desirable to increase the accuracy of the estimated autocorrelation function $\hat{\boldsymbol{R}}_{\boldsymbol{X}}(\boldsymbol{n\Delta t})$ by calculating the mean of an ensemble of these functions. To this end it is necessary to collect 10 sets with 3000 samples each and subsequently calculate 10 $\hat{\boldsymbol{R}}_{\boldsymbol{X}}(\boldsymbol{n\Delta t})$ from this ensemble. Mikael Pettersson's [Pet02] *Linux x86 Performance-Monitoring Counters Driver* is applied by the PMC acquisition program in global mode to sample selected PMCs at a sample rate of $\boldsymbol{\Delta t = 0.04 sec}$. Global mode means that the driver continuously samples the PMC readings unrelated to particular processes that may run. The autocorrelation analysis presented in this thesis is performed on data that were sampled with $\boldsymbol{\Delta t = 0.04 sec}$; more recent kernels allow this to be reduced to less than $\boldsymbol{\Delta t = 0.02 sec}$. The acquisition software writes the readings initially into preallocated memory and saves these data to disk after the sampling is completed. The two available PMCs are sampled alongside the Time-Stamp Counter (TSC). In SMP systems the acquisition software samples both CPUs concurrently. This procedure takes a significant amount of time if it is necessary to sample all available 132 PMCs.

Execution time = Samples * $\boldsymbol{\Delta t}$ * Number of Sets * Number of PMCs

Execution time = 3000 * 0.04 sec * 10 * 132 = 44 hours

If both PMC registers are used this execution time can be reduced to 22 hours. The acquisition software's procedure is as follows: the program selects the next two PMC events and samples 3000 readings before writing the data to disk and performs the task 10 times. Subsequently the software selects the next two events and continues this until all PMC events have been sampled. The acquisition software is written in *C* and the analysis software that is used for the next 6 steps is written in Octave code, a Matlab-like open source implementation.

5.2.2 Visual Inspection of sample PMC Readings and their Histograms

This is the first step in the off-line analysis. The octave program opens the previously saved PMC sample files and calculates histograms for the 10 sets with their 3000 samples. Fig. 5.1 on page 63 and Fig. 5.2 on page 64 provide an example for event number 26 (L2_LINES_IN) set 3 on CPU 1 and CPU 2 respectively. Please see table A.1 on page 147 in appendix A for a description of the event. Section B.1.1 on page 157 in appendix B shows the histograms for all 10 sets in both CPUs for this event. A visual inspection guarantees that all sets contain valid PMC data.

5.2.3 Visual Inspection of simulated PMC Readings and their Histograms

It is possible to compare histograms from section 5.2.2 with histograms in this section that are based on simulated PMC random processes. These simulations apply $\mathbf{1/\beta_i}$ and $\boldsymbol{\sigma_i^2}$ parameters that resulted from the analysis of sampled PMCs show in section 5.2.2. A full list of an ensemble of 10 sets can be found section B.1.2 on page 161 in appendix B.

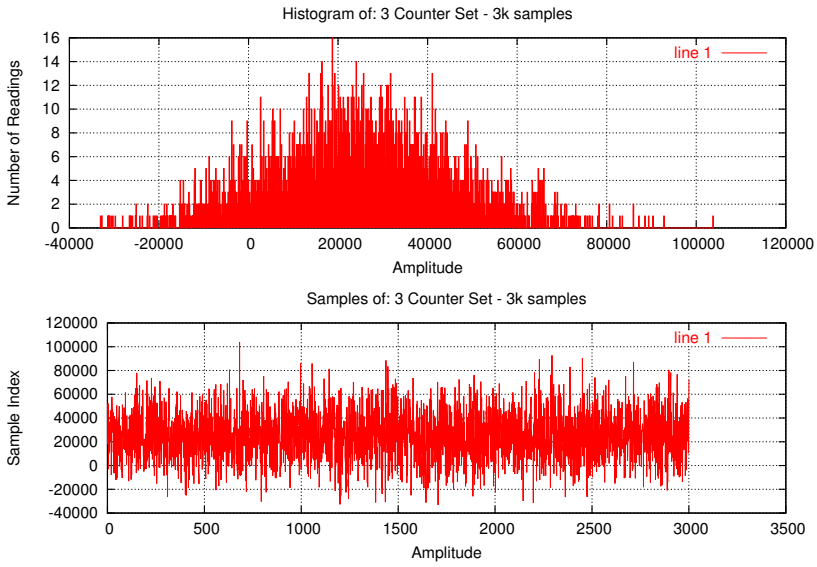

Figure 5.6: Histogram and samples for a simulated event based on statistics for event number 26 symbol L2_LINES_IN

5.2.4 Autocorrelation Calculation for Sampled PMC Readings

Fig. 5.7 and Fig. 5.8 show the histogram and the estimated autocorrelation functions $\hat{\boldsymbol{R}}_{\boldsymbol{X}}(\boldsymbol{n\Delta t})$ that are calculated with Eq.(5.25) in the Octave analysis program for the same set and event that was used in the previous sections. Section B.1.3 on page 163 in appendix B provides figures for the estimated autocorrelation functions of all sets in both CPUs.

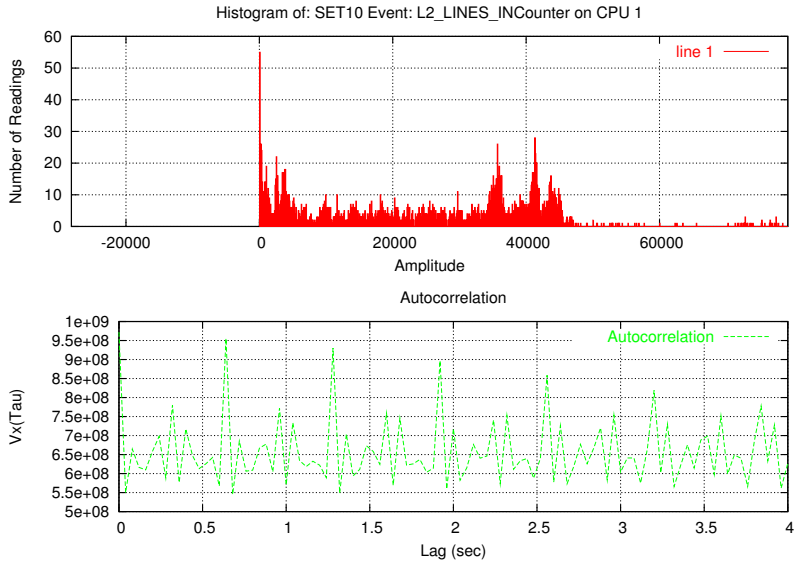

Figure 5.7: Histogram and autocorrelation for event number 26 symbol L2_LINES_IN on CPU 1

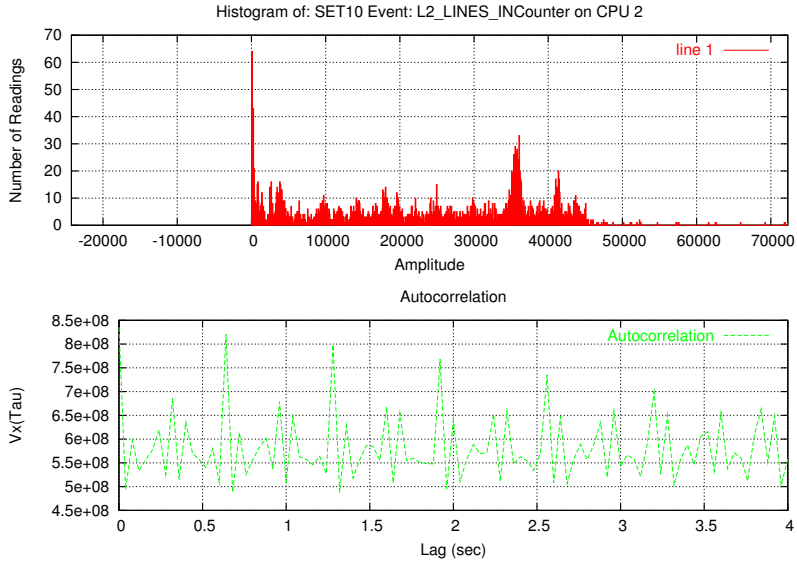

Figure 5.8: Histogram and autocorrelation for event number 26 symbol L2_LINES_IN on CPU 2

5.2.5 Autocorrelation Calculation for Simulated PMC Readings

Again it is possible to compare the estimated autocorrelation functions $\hat{R}_X(n\Delta t)$ from section 5.2.4 with autocorrelation functions in this section that are based on simulated PMC random processes. These simulations apply statistics that resulted from the analysis of sampled PMCs show in section 5.2.2. Furthermore the estimated autocorrelation functions for all sets can be found in section B.1.4 on page 166 in appendix B.

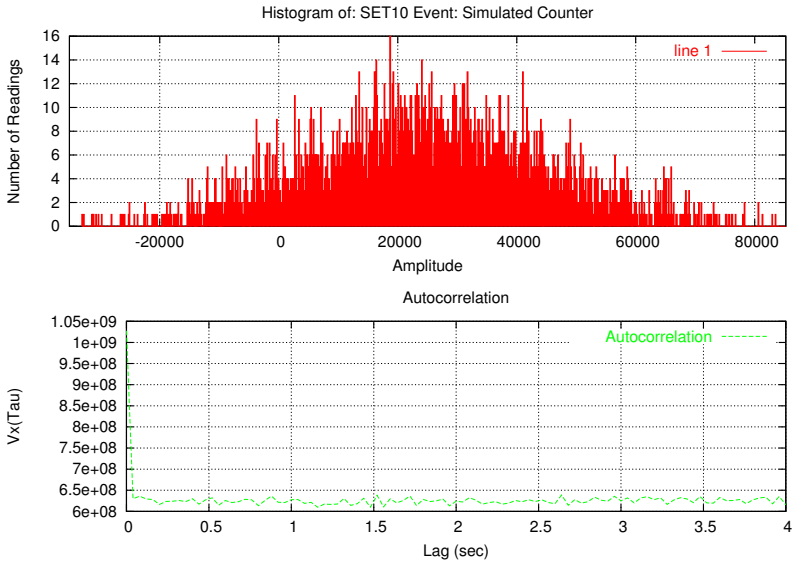

Figure 5.9: Histogram and autocorrelation for a simulated event based on statistics for event number 26 symbol L2_LINES_IN

5.2.6 Calculation of the Sampled PMC Readings' Mean Autocorrelation

Subsequent to the calculation of the 10 estimated autocorrelation functions $\hat{R}_X(n\Delta t)$ this step computes the mean of these 10 functions. The result of this operation can be seen in Fig. 5.10 and Fig. 5.11 for both CPUs. The top graph shows the 10 autocorrelation functions and the bottom graph depicts the mean of these functions.

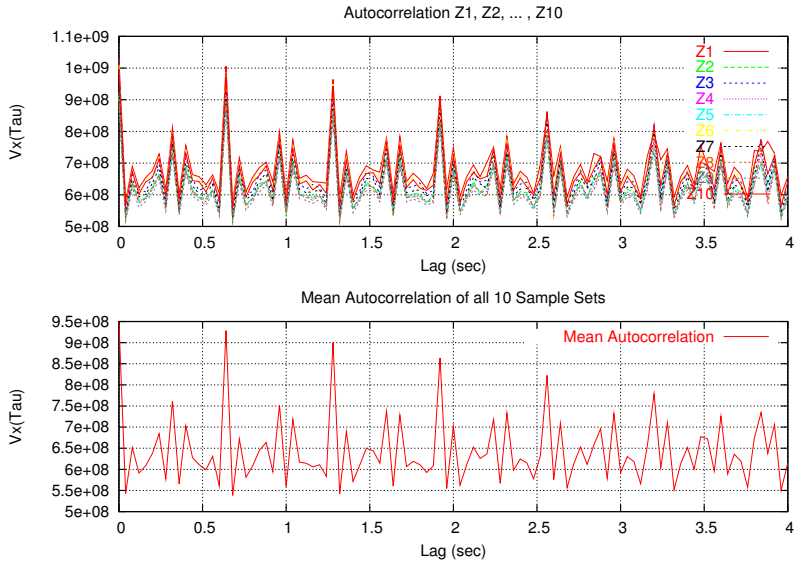

Figure 5.10: Autocorrelation for all 10 sets and the mean autocorrelation for event number 26 symbol L2_LINES_IN on CPU 1

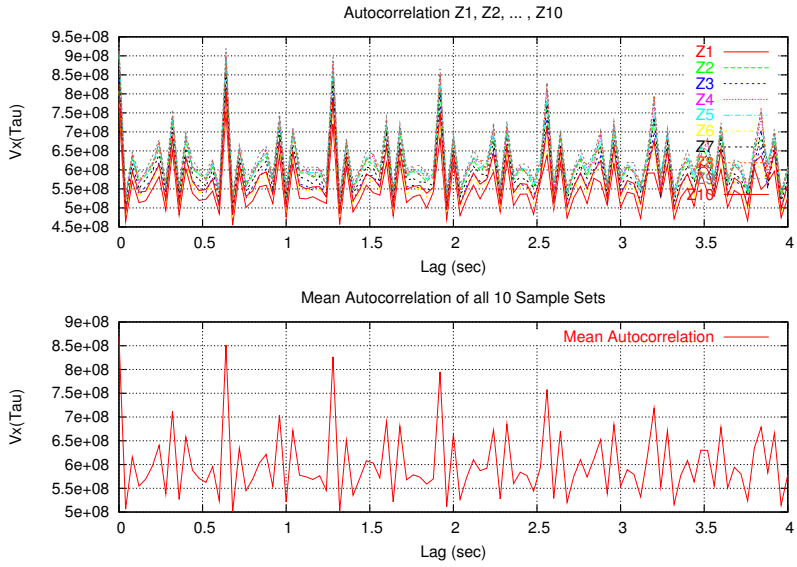

Figure 5.11: Autocorrelation for all 10 sets and the mean autocorrelation for event number 26 symbol L2_LINES_IN on CPU 2

5.2.7 Calculation of the Simulated PMC Readings' Mean Autocorrelation

As in the previous simulation sections the calculations are based on simulated PMC random processes that apply statistics from sampled PMCs. Fig. 5.12 show the mean estimated autocorrelation functions $\hat{R}_X(n\Delta t)$ and the 10 autocorrelation functions for the simulated PMC random processes.

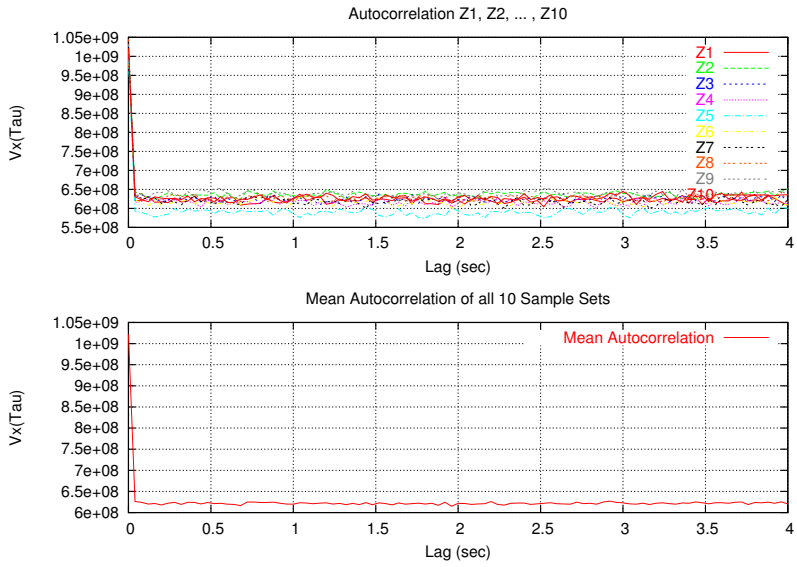

Figure 5.12: Autocorrelation for all 10 sets and the mean autocorrelation for a simulated event based on statistics for event number 26 symbol L2_LINES_IN

5.2.8 Estimation of β and σ^2 for Sampled PMC Readings

After calculating the estimated autocorrelation functions $\hat{R}_X(n\Delta t)$ for 10 sets of samples in section 5.2.4 and then computing the mean autocorrelation function from these ten functions in section 5.2.6 it is now possible to derive $1/\beta_i$ and σ_i^2 from the mean autocorrelation function. Fig. 5.13 shows the autocorrelation function of the Gauss-Markov process described by Eq.(5.15). There are two observations to notice. The experimental determination of the autocorrelation function provides only the right side of the function. This has no consequences since the function is symmetric. Furthermore the $\hat{R}_X(n\Delta t)$ converges in the mean^2. Therefore as a first step the mean^2 must be subtracted from $\hat{R}_X(n\Delta t)$. In order to derive $1/\beta$ and σ^2 from the result of this operation it is necessary to take the $\ln(\hat{R}_X(n\Delta t) - mean^2)$. Linear regression can then be applied to this outcome to determine a and b in $y = ax + b$ with $a = \beta$ and $b = \sigma^2$.

Table 5.1 provides some example results from the off-line autocorrelation analysis. Fig. 5.14 and Fig. 5.15 shows the mean of 10 estimated autocorrelation functions as $\boldsymbol{R_x}$ and the fitted curve as $(\boldsymbol{\sigma^2 e^{(-\beta \Delta t)}}) + \boldsymbol{mean^2}$ for CPU 1 and CPU 2 respectively.

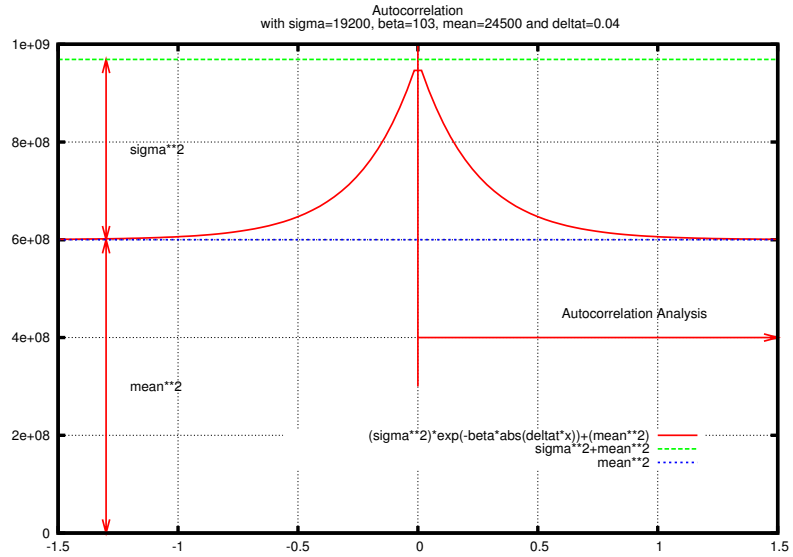

Figure 5.13: Autocorrelation Function

Event	CPU	σ_i	σ_i^2	β_i	Δt	Mean
INST_RETIRED	1	7.248e+06	5.253e+13	40.02	0.04	8.705e+06
	2	6.434e+06	4.139e+13	32.91	0.04	7.038e+06
UOPS_RETIRED	1	1.027e+07	1.055e+14	41.93	0.04	1.24e+07
	2	9.83e+06	9.664e+13	40.2	0.04	1.161e+07
INST_DECODED	1	7.095e+06	5.034e+13	39.41	0.04	8.45e+06
	2	7.059e+06	4.983e+13	36.6	0.04	8.149e+06
L2_LINES_IN	1	2.023e+04	4.091e+08	112.7	0.04	2.529e+04
	2	1.919e+04	3.682e+08	103	0.04	2.445e+04
BR_BOGUS	1	1170	1.368e+06	30.66	0.04	1288
	2	1170	1.368e+06	29.73	0.04	1275

Table 5.1: Offline autocorrelation analysis

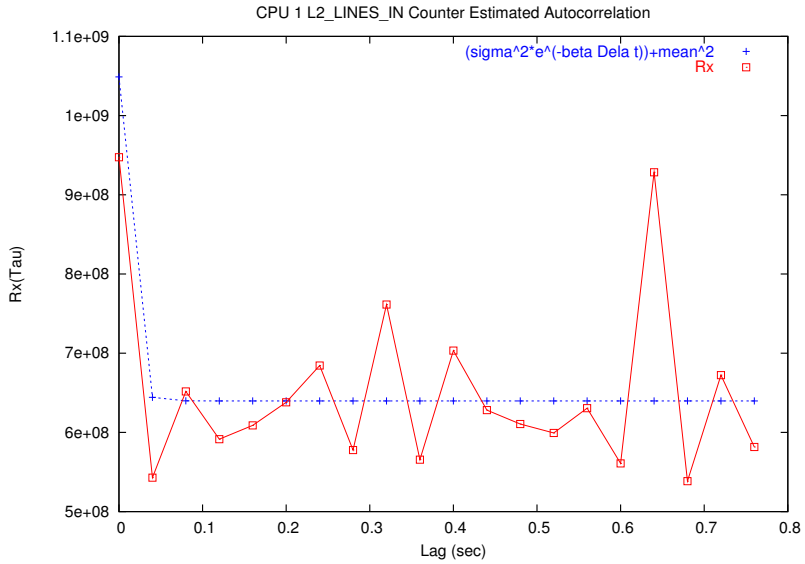

Figure 5.14: Curve Fitted Autocorrelation for event number 26 symbol L2_LINES_IN on CPU 1

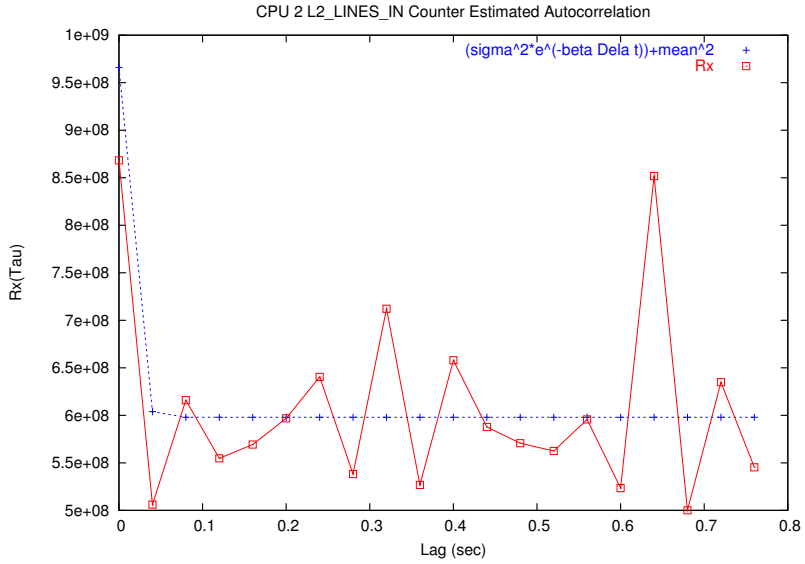

Figure 5.15: Curve Fitted Autocorrelation for event number 26 symbol L2_LINES_IN on CPU 2

5.2.9 Estimation of β and σ^2 for Simulated PMC Readings

As in the previous simulation sections the calculations are based on simulated PMC random processes that apply statistics from sampled PMCs. Fig. 5.16 show the mean of 10 estimated autocorrelation functions as $\boldsymbol{R_x}$ and the fitted curve as $(\boldsymbol{\sigma^2 e^{(-\beta \Delta t)}}) + \boldsymbol{mean^2}$ for the simulated PMC random processes.

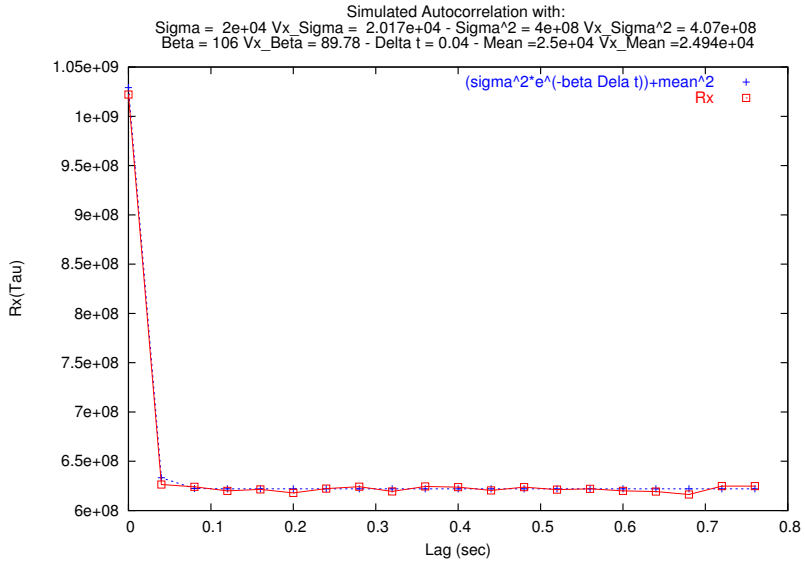

Figure 5.16: Curve Fitted Autocorrelation for a simulated event based on statistics for event number 26 symbol L2_LINES_IN

5.2.10 Visual Inspection of Histograms with Superimposed Gaussian PDFs for Sampled PMC Readings

The last of the PMC sample analysis steps uses statistics derived in the previous step to verify some of the findings by overlaying the PDF over the histograms. The figures for all the 10 sets can be found in section B.1.5 on page 168 in appendix B.

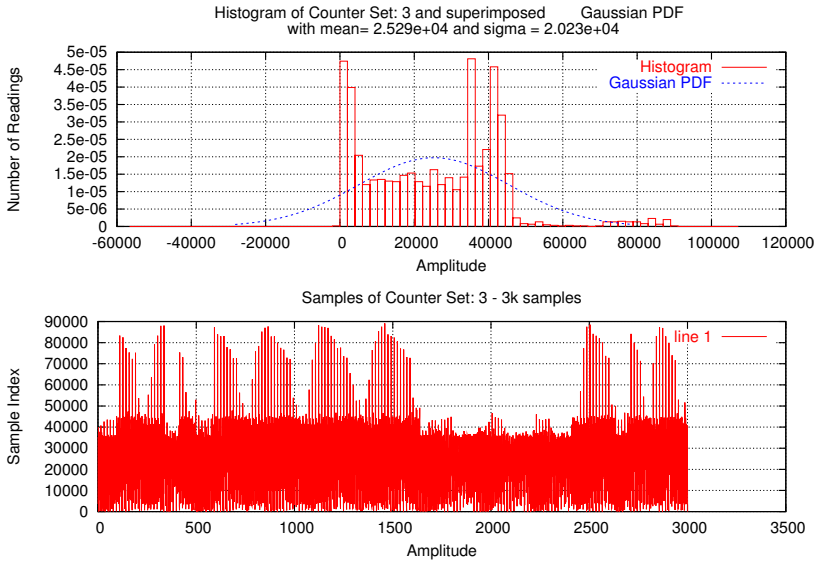

Figure 5.17: Histogram of Real PMC Readings with superimposed Gaussian PDF for event number 26 symbol L2_LINES_IN on CPU 1

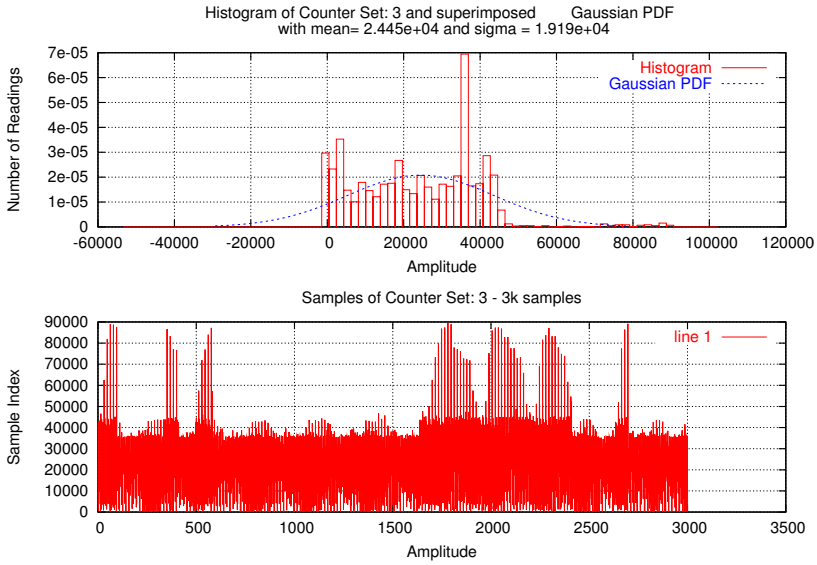

Figure 5.18: Histogram of Real PMC Readings with superimposed Gaussian PDF for event number 26 symbol L2_LINES_IN on CPU 2

5.2.11 Visual Inspection of Histograms with Superimposed Gaussian PDFs for Simulated PMC Readings

Fig. 5.19 overlays a PDF over the histogram for a simulated PMC random process. The full list of figures for all 10 set is presented in section B.1.6 on page 171 in appendix B.

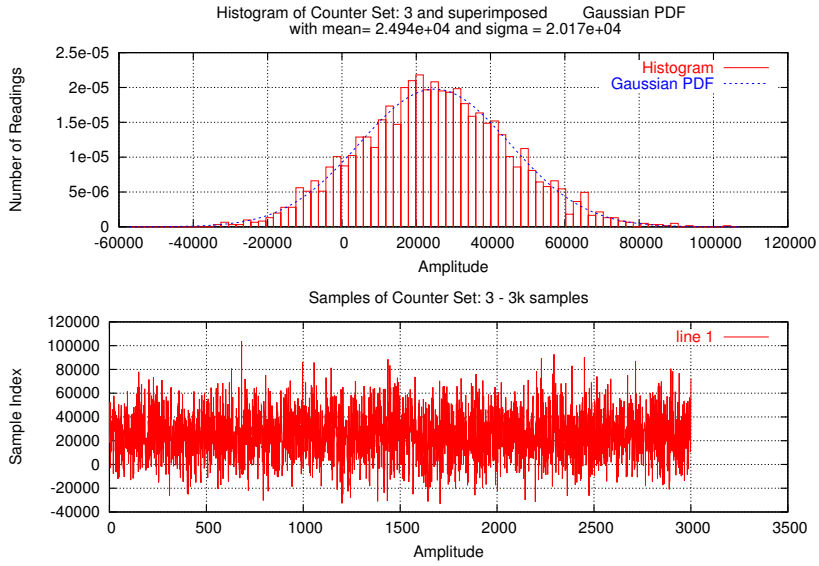

Figure 5.19: Histogram of Real PMC Readings with superimposed Gaussian PDF for a simulated event based on statistics for event number 26 symbol L2_LINES_IN

5.2.12 Autocorrelation Analysis Results

The section presents the PMC sample analysis results for all the 132 PMC events.

A short list of analysis results is given in Table 5.1 for a comprehensive list please see table B.1 in appendix B on page 78. Fig. 5.20, Fig. 5.21, Fig. 5.22 Fig. 5.23, Fig. 5.24 and Fig. 5.25 present the same results sorted by their β value and also show the differences between CPU 1 and CPU 2.

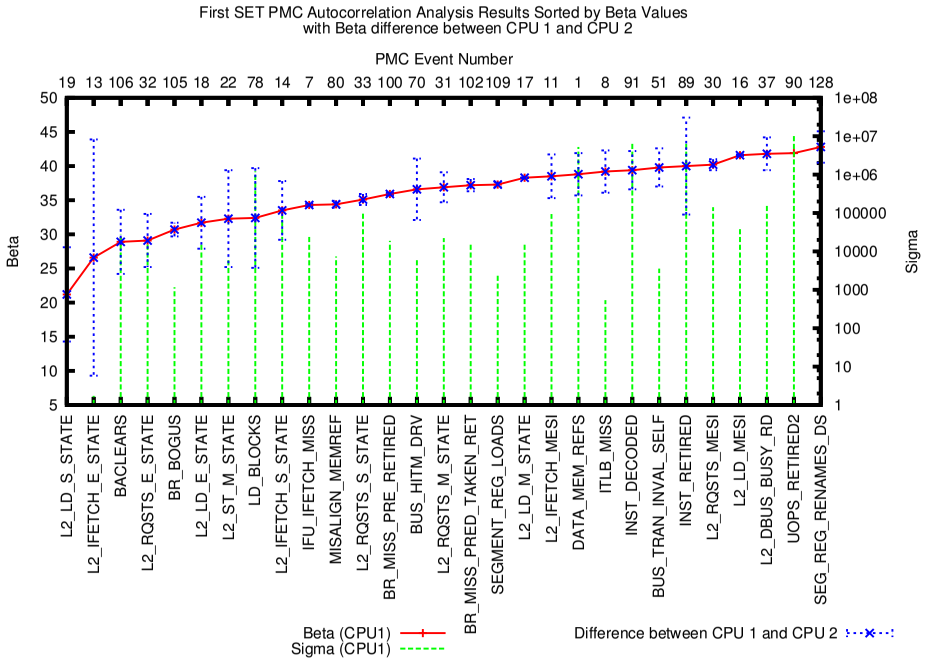

Figure 5.20: First set of autocorrelation analysis results with beta error

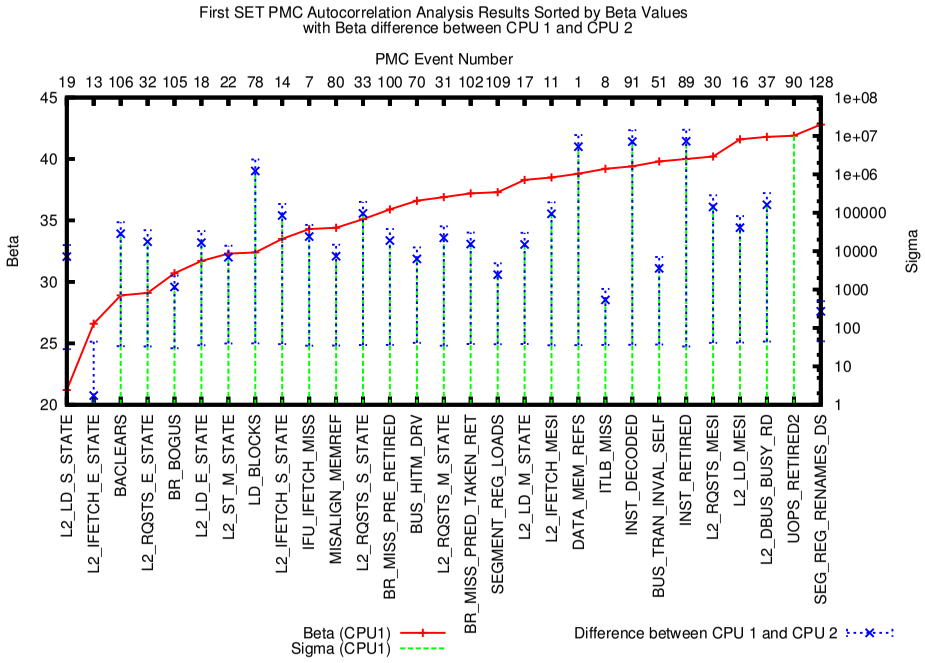

Figure 5.21: First set of autocorrelation analysis results with sigma error

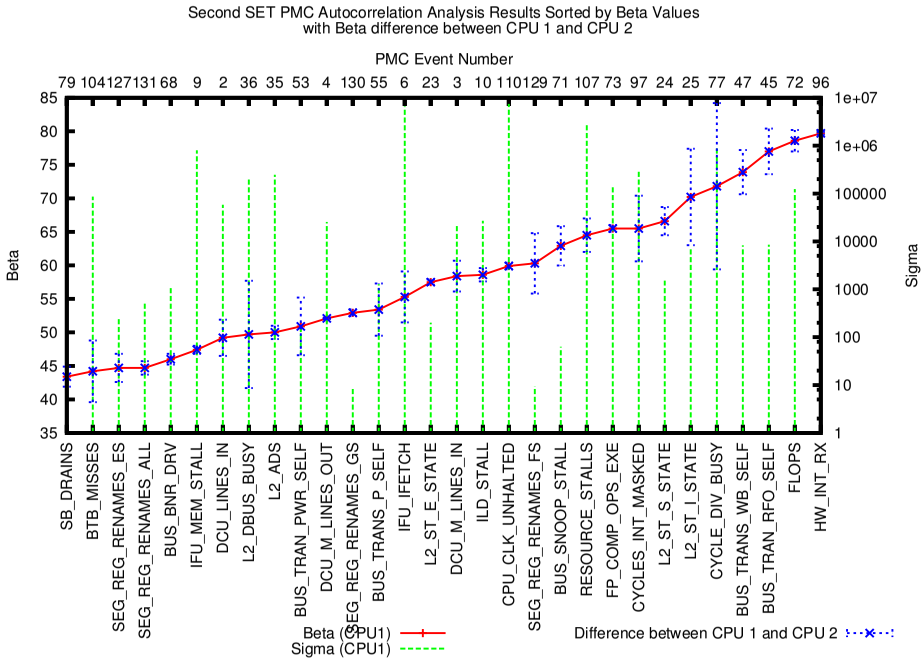

Figure 5.22: Second set of autocorrelation analysis results with beta error

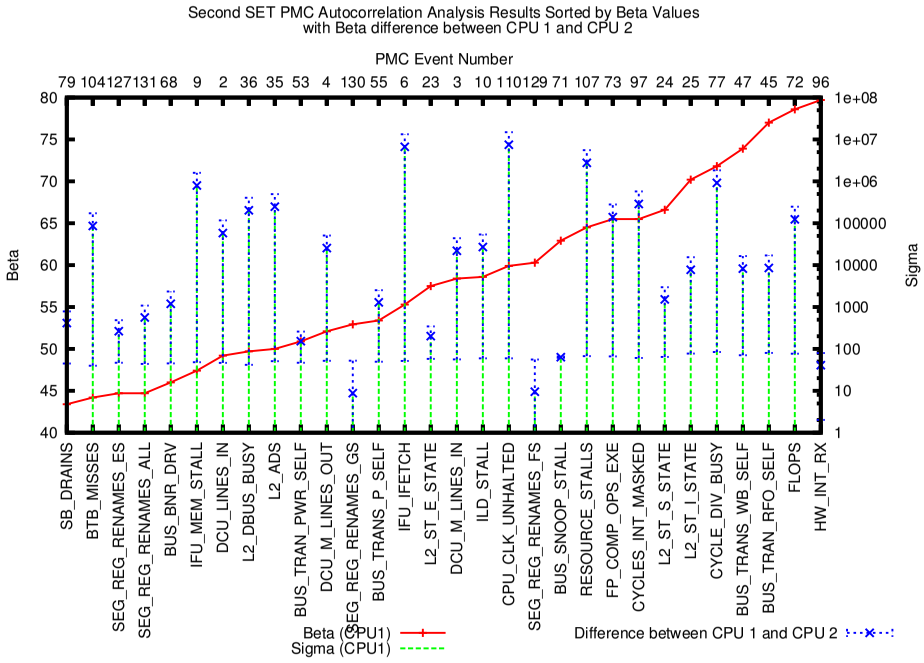

Figure 5.23: Second set of autocorrelation analysis results with sigma error

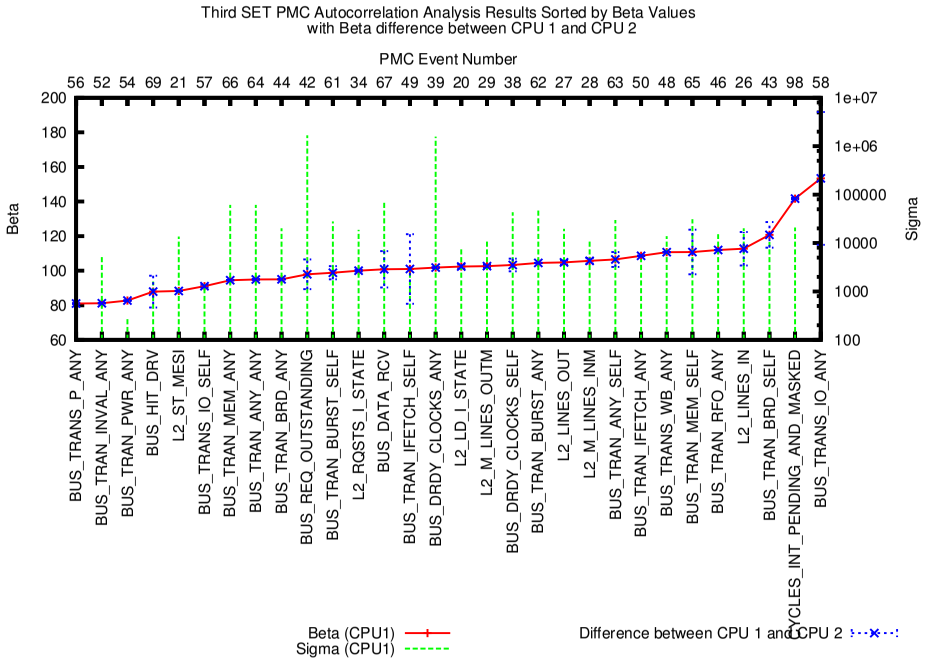

Figure 5.24: Third set of autocorrelation analysis results with beta error

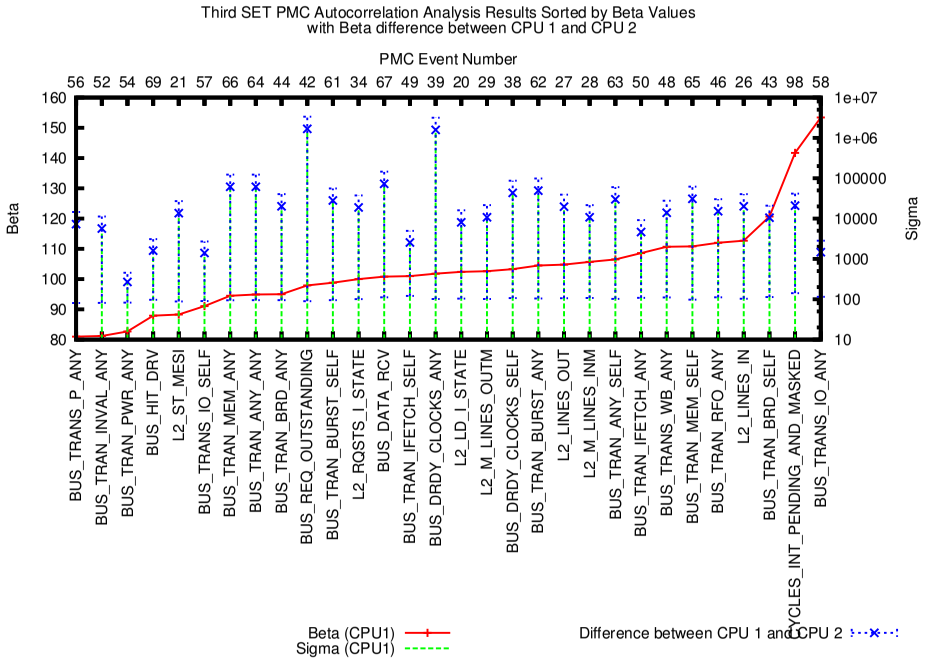

Figure 5.25: Third set of autocorrelation analysis results with sigma error

5.3 One Performance Monitoring Counter (PMC) Set-at-a-Time

So far we have developed the notion of a Kalman filter including a process model suitable for PMC processes and the tuning of the model. This filter algorithm can process $\boldsymbol{n}$ performance counter readings, but if the number of observed counters exceeds the number of available PMC registers then we must multiplex sets of counter readings as discussed in section 2.2. Assuming that, Fig. 2.3 shows us that counter data for a particular set are only available during certain intervals. If a simple sample-and-hold approach is applied in conjunction with this multiplexing then counter data will be held until new data becomes available, but the sampling of these data will cause a discrete step if a counter's value differs from a previous sample. No consideration is given to the uncertainty of this data or how it varies during the interval. The use of a Kalman filter can resolve these issues, as described below.

In order to process sets of PMCs one-at-a-time modifications must be made to the recursive filter algorithm presented in Fig. 5.3 section 5.1.1. The three equations for the Kalman gain $\boldsymbol{K_k}$ Eq. (5.11), the a posteriori state estimate $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$ Eq. (5.6) and the error covariance $\boldsymbol{P_k}$ Eq. (5.12) are all part of the filter's *correction* process. The equations are repeated here for convenience. In the modified algorithm the equation performs the correction only on submatrices that are associated with the set of selected performance counters for the current iteration of the loop. This process allows for the merging of available counter readings into the state vector $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$.

$$
\begin{aligned}
\boldsymbol{K_k} &= \frac{\boldsymbol{P_k^-}\boldsymbol{H_k}}{\boldsymbol{H_k}\boldsymbol{P_k^-}\boldsymbol{H_k^T} + \boldsymbol{R_k}} \\
\hat{\boldsymbol{x}}_{\boldsymbol{k}} &= \hat{\boldsymbol{x}}_{\boldsymbol{k}}^- + \boldsymbol{K_k}(\boldsymbol{z_k} - \hat{\boldsymbol{x}}_{\boldsymbol{k}}^-\boldsymbol{H_k}) \\
\boldsymbol{P_k} &= (\boldsymbol{I} - \boldsymbol{K_k}\boldsymbol{H_k})\boldsymbol{P_k^-}
\end{aligned}
$$

The remaining two equations of the filter algorithm, the a priori state vector $\hat{\boldsymbol{x}}_{\boldsymbol{k+1}}^-$ Eq. (5.13) and the a priori covariance matrix $\boldsymbol{P_{k+1}^-}$ Eq. (6.1.5) implement the *prediction* aspect of the filter. The equations are also repeated below for convenience. This part of the algorithm remains unchanged. Consequently the entire state vector $\hat{\boldsymbol{x}}$ for all the performances counters is propagated according to the state transition matrix $\boldsymbol{\phi}$. This is an important detail because the state vector $\hat{\boldsymbol{x}}$ and (through the measurement matrix $\boldsymbol{H}$) the estimated measurement $\hat{\boldsymbol{z}}$ changes over the time intervals without data according to the system dynamics defined in the state transition matrix $\boldsymbol{\phi}$. Furthermore the a priori covariance matrix $\boldsymbol{P_{k+1}^-}$ propagates the uncertainty for counter processes during every iteration of the algorithm. Therefore the measure of uncertainty for unavailable counters will increase with every iteration until new data are available. Moreover, an optimal blending operation can be

performed once data become available. This is in contrast to a sample-and-hold operation.

Fig. 5.26 and Fig. 5.27 shows how the filter algorithm increases the two a priori error covariance matrix $\boldsymbol{P^-}$ elements on the major diagonal that represent the the algorithm uncertainty about the current estimate of the two state-variables associated with the PMC event DATA_MEM_REFS. These increments to the two $\boldsymbol{P^-}$ matrix elements will continue until a new reading is sampled. In this example the filter take a new sample of the PMC event DATA_MEM_REFS every third iteration.

$$\begin{aligned} \hat{\boldsymbol{x}}_{\boldsymbol{k+1}}^{-} &= \boldsymbol{\phi_k}\hat{\boldsymbol{x}}_{\boldsymbol{k}} \\ \boldsymbol{P}_{\boldsymbol{k+1}}^{-} &= \boldsymbol{\phi_k P_k \phi_k^T + Q_k} \end{aligned}$$

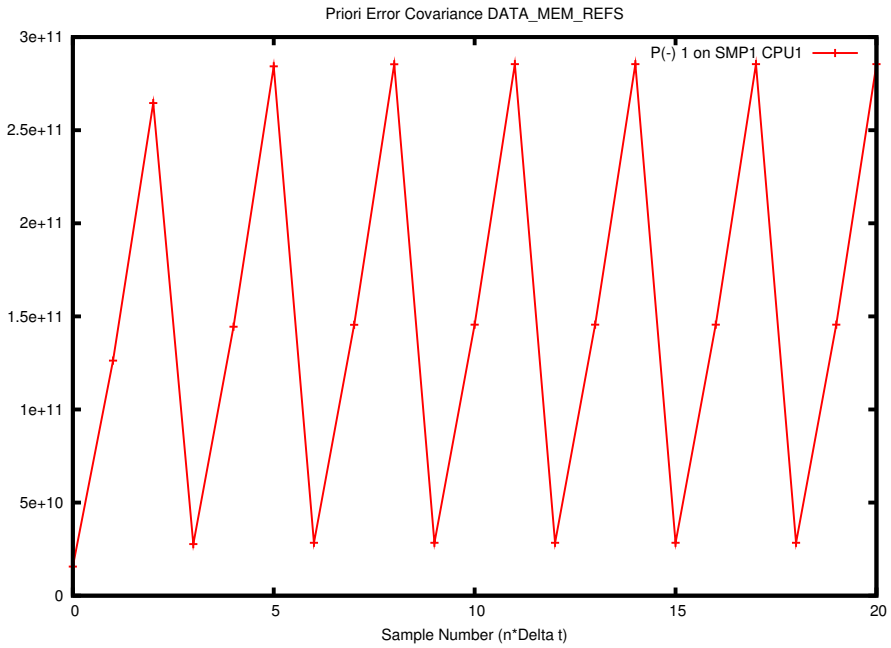

Figure 5.26: A Priori Error Covariance matrix element one of the major diagonal.

5.4 Implementation of the Estimation Algorithm

The Kalman Filter Algorithm as shown in Fig. 5.3 requires 11 multiplications, 3 additions, 2 subtractions and 1 inversion. These operations are either matrix-matrix or matrix-vector operations. Table 5.4 provides a list of all the 17 operations including a specification of the matrices or vectors that receive the result. The table also gives references to equations in this thesis where the implementation of these individual operation is discussed in more detail.

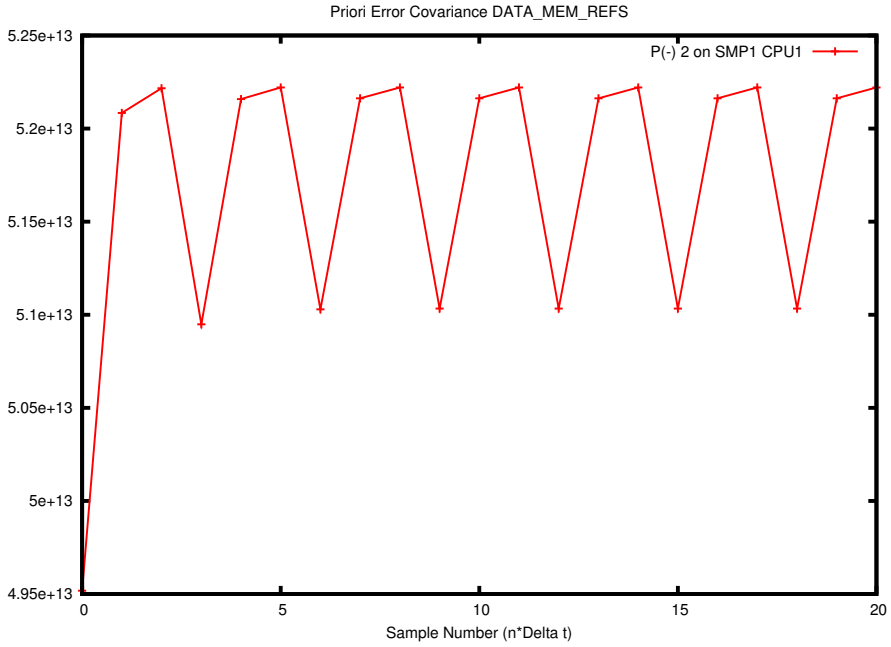

Figure 5.27: A Priori Error Covariance matrix element two of the major diagonal.

The predominant operation is the multiplication: 11 out of 17 operations are multiplications that are necessary for one iteration of the algorithm. Together with the matrix inversion these are the more computational expensive calculations. A matrix multiplication has a complexity of $\boldsymbol{O(n^3)}$ [PTVF99]. The Strassen algorithm [Str69] and the Coppersmith-Winograd algorithm [CW90] can improve this up to a complexity of $\boldsymbol{O(n^{2.376})}$. This section demonstrates that despite these potential improvements the algorithm is unsuitable for real time application with large state vectors and small sample intervals $\boldsymbol{\Delta t}$. Fig. 5.28, Fig. 5.29 and Fig. 5.30 show the execution time of the full Kalman filter algorithm as shown in Ta-

Filter Initialisation					
#	Equation	Matrix	Description	Rows	Columns
1		$\boldsymbol{H}$	Measurement sensitivity matrix	n	2n
2		$\boldsymbol{H^T}$	Transpose $\boldsymbol{H}$	2n	n
3	Eq.(5.18)	$\boldsymbol{\phi}$	State transitions matrix	2n	2n
4		$\boldsymbol{\phi^T}$	Transpose $\boldsymbol{\phi}$	2n	2n
5		$\boldsymbol{I}$	Identitity matrix	2n	2n
6	Eq.(6.1.5)	$\boldsymbol{P^-}$	A Priori error covariance matrix	2n	2n
7		$\boldsymbol{R}$	Measurement noise covariance matrix	n	2n
8	Eq.(5.19)	$\boldsymbol{Q}$	Process noise covariants matrix	2n	2n

Table 5.2: This table presents all the Kalman filter matrix initialisations. The equation numbers indicate the vicinity of a more detailed discussions of the individual operations.

ble 5.4 and Fig. 5.3 on a Hyperthreaded Intel P4, a two way Intel PIII SMP and a Intel PIII CompactPCI system respectively. These systems are specified in Table 7.1 on page 123. *Execution time of the full Kalman filter algorithm* means that all the elements of all matrices and vectors involved must be calculated. Furthermore Fig. 5.31, Fig. 5.32 and Fig. 5.33 provide the same information with a higher execution time resolution.

All three systems employ Linux as the OS, with 2.6.9 or 2.6.10 kernels. The graphs shown in Fig. 5.28 to Fig. 5.33 are generated with the `bash time` program. The `time` program runs the compiled `c` program that implements the Kalman filter algorithm. On completion the `time` program provides information about:

- Real time (the elapsed real time between invocation and termination)
- User time (CPU time consumed by the program in user mode)
- System time (time consumed by the CPU in system mode)

The Kalman filter algorithm requires a certain start-up time to build all the matrices and vectors for the algorithm. In order to reduce the measured CPU time that is caused by the initialisation, the filter algorithm was executed 100 times for every measurement, rendering the start-up time insignificant in comparison to the 100 filter iterations. The real time, user time and system time values shown in Fig. 5.28 to Fig. 5.33 are the average execution times for a single iteration.

For the initial measurement the Kalman filter is configured for a single PMC reading and then incremented by one PMC after every measurement. Since the PMCs are modeled as integrated Gauss-Markov processes (see section 5.1.2) the state vector $\boldsymbol{x_k}$ has twice the number of elements. This procedure allows us to plot the execution time in terms of real time, user time and system time over the number of estimated PMCs.

Fig. 5.28, Fig. 5.29 and Fig. 5.30 depicts the real, user and system execution time over the number of PMCs estimated or predicted by the Kalman filter for the three machines specified in Table 7.1. It is obvious from the graphs that the system execution time is insignificant when compared to the user execution time. When analysing these figures we should assume that the PMCs are potentially sampled every 20 ms before they are fed into the filter. This means that if the filter's execution time exceeds 20 ms then the algorithm fails. The 20 ms threshold is labelled *PMC Sample Time*. The filter algorithm was compiled with and without compiler optimisation using the GNU Compiler Collection (gcc) with the `-O0` option for no optimisation or with the -O3 option to turn on most of the available optimisations [StGDC05].

The experiments shown in Fig. 5.28, Fig. 5.29 and Fig. 5.30 tell us that the filter execution time exceeds the sample time $\boldsymbol{\Delta t}$ even for a modest number of PMCs. This conclusion can be observed in more detail by looking at Fig. 5.31, Fig. 5.32 and Fig. 5.33.

System	Compiler Optimised	No Compiler Optimisation
Intel P4 Hyperthreaded	8	6
Intel PIII 2 Way SMP	7	7
Intel PIII CompactPCI System	7	7

Table 5.3: This table shows the maximum number of PMC readings that can be processed by the Kalman filter algorithm on the various machines with exceeding 1% of the available CPU execution time (Maximum Time for Filter Algorithm threshold) assuming a sample time $\boldsymbol{\Delta t}$ of 20 ms.

If the algorithm's execution time exceeds the sample time $\boldsymbol{\Delta t}$ this leaves no execution time for the application. Furthermore, if we allow the filter algorithm to consume a generous 1% of the available CPU execution time then we can define a new threshold, the *Maximum Time for Filter Algorithm* $= \frac{\boldsymbol{\Delta t}}{\mathbf{100}}$. Fig. 5.31, Fig. 5.32 and Fig. 5.33 show this threshold assuming a sample time $\boldsymbol{\Delta t}$ of 20 ms. Table 5.3 provides the maximum number of PMCs for three machines from Table 7.1.

Thus a very limited number of PMCs can be estimated and merged into a state vector. In the case of the DSM Testbed (see section 7.1) all machines use either the Intel PIII 2 Way SMP or the Intel PIII CompactPCI System from Table 7.1. These PIII systems provide only 2 PMC registers and on the 2 Way SMP system, assuming only one estimation algorithm runs, it is not possible to even achieve a twofold improvement without increasing the Maximum Time for Filter Algorithm threshold because two registers can be sampled on each of the two CPUs. Clearly this represents a major performance problem.

Kalman Gain Eq.(5.11)						
$\boldsymbol{K_k = (P_k^- H_k^T)(H_k P_k^- H_k^T + R_k)^{-1}}$						
Step	Equation	Matrix	Rows	Columns	Operation	#
1	Eq.(6.1)	$\boldsymbol{P_k^- H_k^T}$	2n	n	MUL 1	1
2	Eq.(6.3)	$\boldsymbol{H_k P_k^-}$	n	2n	MUL 2	2
3	Eq.(6.5)	$\boldsymbol{H_k P_k^- H_k^T}$	n	n	MUL 3	3
4	Eq.(6.6)	$\boldsymbol{H_k P_k^- H_k^T + R_k}$	n	n	ADD 1	4
5	Eq.(6.8)	$\boldsymbol{(H_k P_k^- H_k^T + R_k)^{-1}}$	n	n	INVERSE	5
6	Eq.(6.9)	$\boldsymbol{K_k}$	2n	n	MUL 4	6
A Posteriori State Estimate Eq.(5.6)						
$\boldsymbol{\hat{x}_k = \hat{x}_k^- + K_k(z_k - \hat{x}_k^- H_k)}$						
Step	Equation	Matrix	Rows	Columns	Operation	#
1	Eq.(6.10)	$\boldsymbol{H_k \hat{x}_k^-}$	n	1	MUL 5	7
2	Eq.(6.11)	$\boldsymbol{Z_k - H_k \hat{x}_k^-}$	n	1	SUB 1	8
3	Eq.(6.13)	$\boldsymbol{K_k(Z_k - H_k \hat{x}_k^-)}$	2n	1	MUL 6	9
4	Eq.(6.15)	$\boldsymbol{\hat{x}_k}$	2n	1	ADD 2	10
A Posteriori Error Covariance Eq.(5.12)						
$\boldsymbol{P_k = (I - K_k H_k) P_k^-}$						
Step	Equation	Matrix	Rows	Columns	Operation	#
1	Eq.(6.16)	$\boldsymbol{K_k H_K}$	2n	2n	MUL 7	11
2	Eq.(6.17)	$\boldsymbol{I K_k H_k}$	2n	2n	SUB 2	12
3	Eq.(6.19)	$\boldsymbol{P_k}$	2n	2n	MUL 8	13
A Priori State Estimation Eq.(5.13)						
$\boldsymbol{\hat{x}_{k+1}^- = \phi_k \hat{x}_k}$						
Step	Equation	Matrix	Rows	Columns	Operation	#
1	Eq.(6.20)	$\boldsymbol{\hat{x}_{k+1}^-}$	2n	1	MUL 9	14
A Priori Error Covariance Eq.(6.1.5)						
$\boldsymbol{P_{k+1}^- = \phi_k P_k \phi_k^T + Q_k}$						
Step	Equation	Matrix	Rows	Columns	Operation	#
1	Eq.(6.22)	$\boldsymbol{\phi_k P_k}$	2n	2n	MUL 10	15
2	Eq.(6.23)	$\boldsymbol{\phi_k P_k \phi_k^T}$	2n	2n	MUL 11	16
3	Eq.(6.25)	$\boldsymbol{P_{k+1}^-}$	2n	2n	ADD 3	17

Table 5.4: This table presents all 17 Kalman filter matrix operations and the associated matrices for the interim solutions. The equation numbers indicate the vicinity of more detailed discussions of the individual operations.

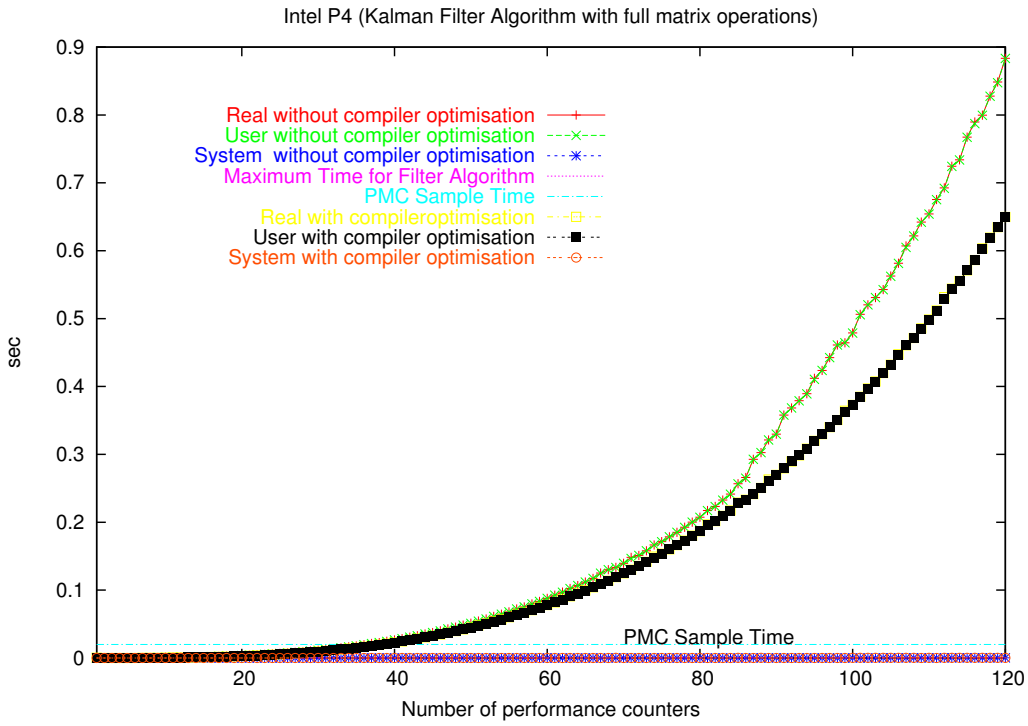

Figure 5.28: Full Kalman Operations on a Intel P4

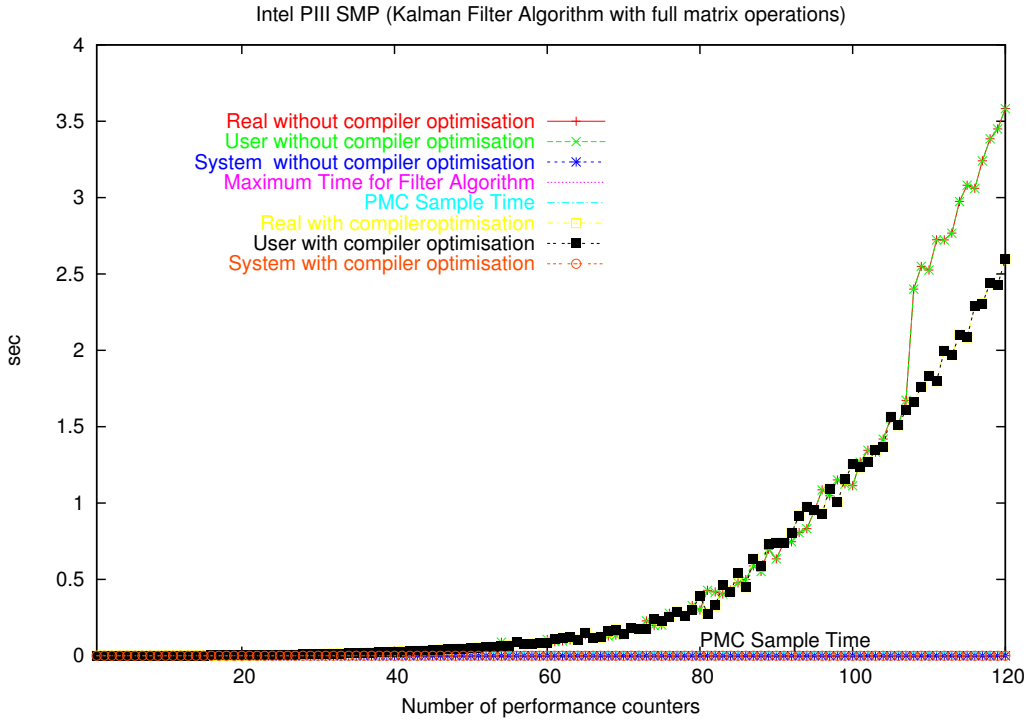

Figure 5.29: Full Kalman Operations on a 2 way Intel PIII SMP

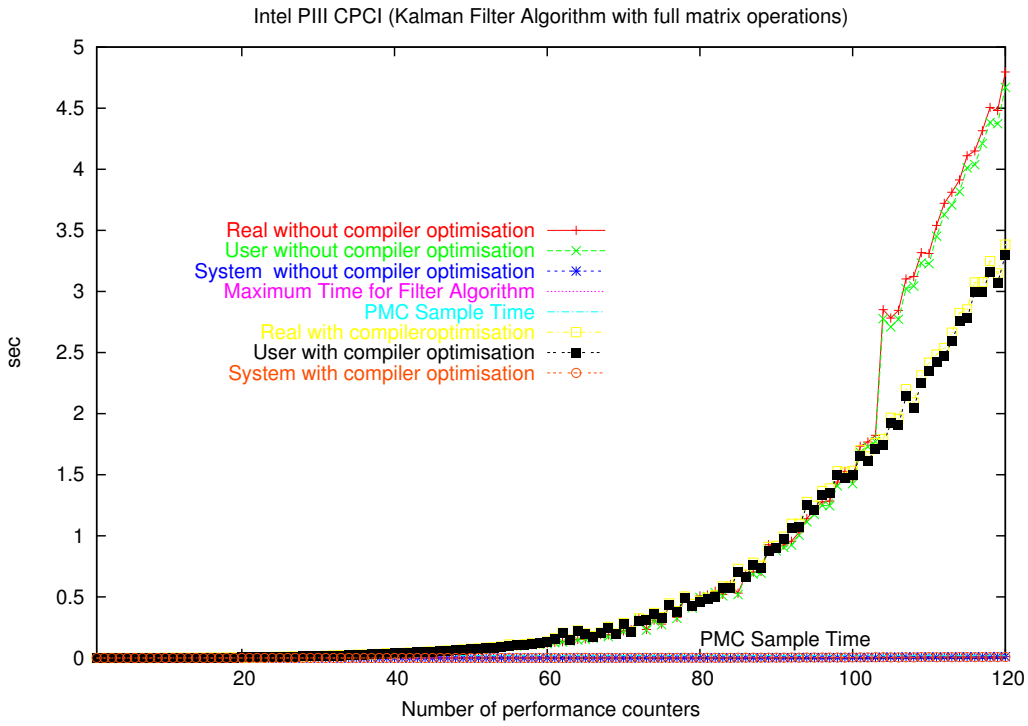

Figure 5.30: Full Kalman Operations on a Intel PIII CompactPCI system

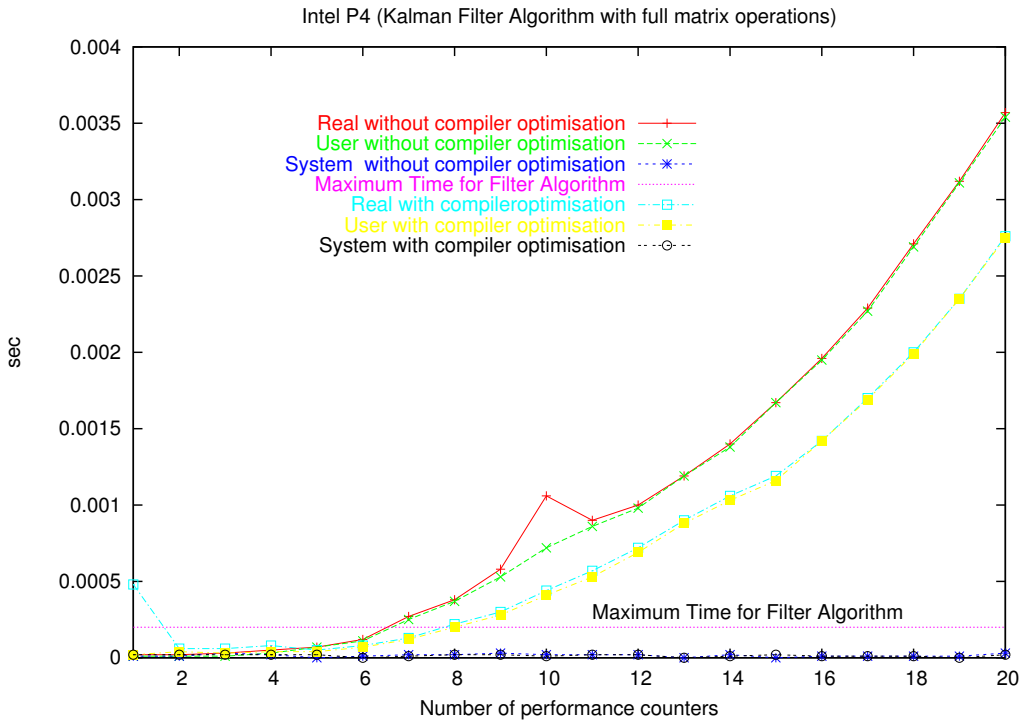

Figure 5.31: Full Kalman Operations on a Intel P4 high resolution

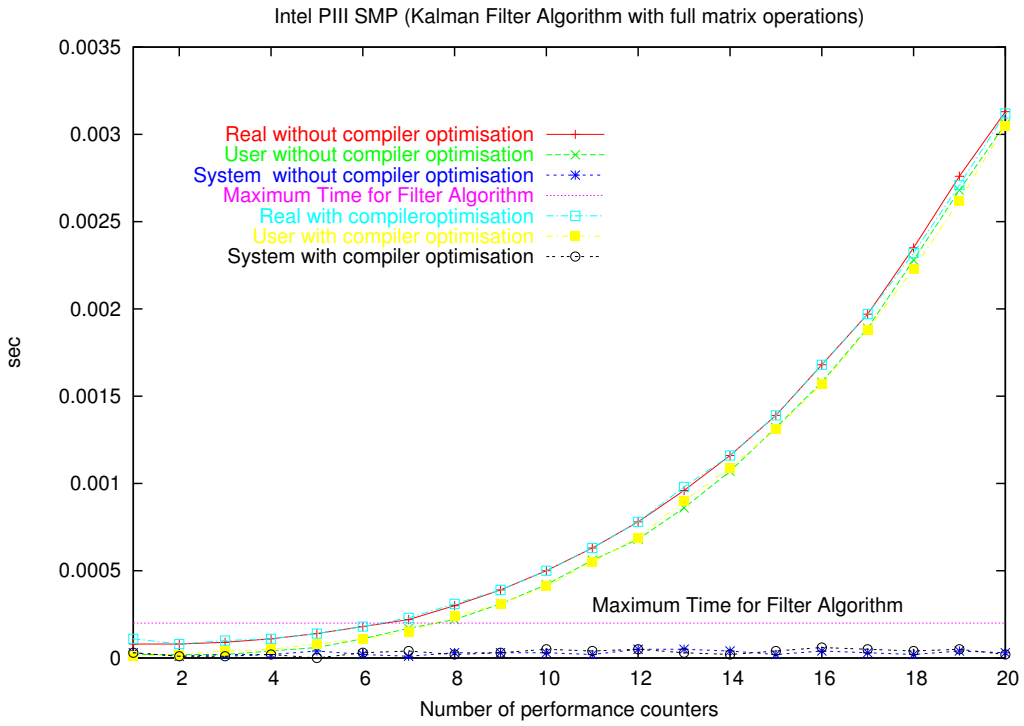

Figure 5.32: Full Kalman Operations on a 2 way Intel PIII SMP high resolution

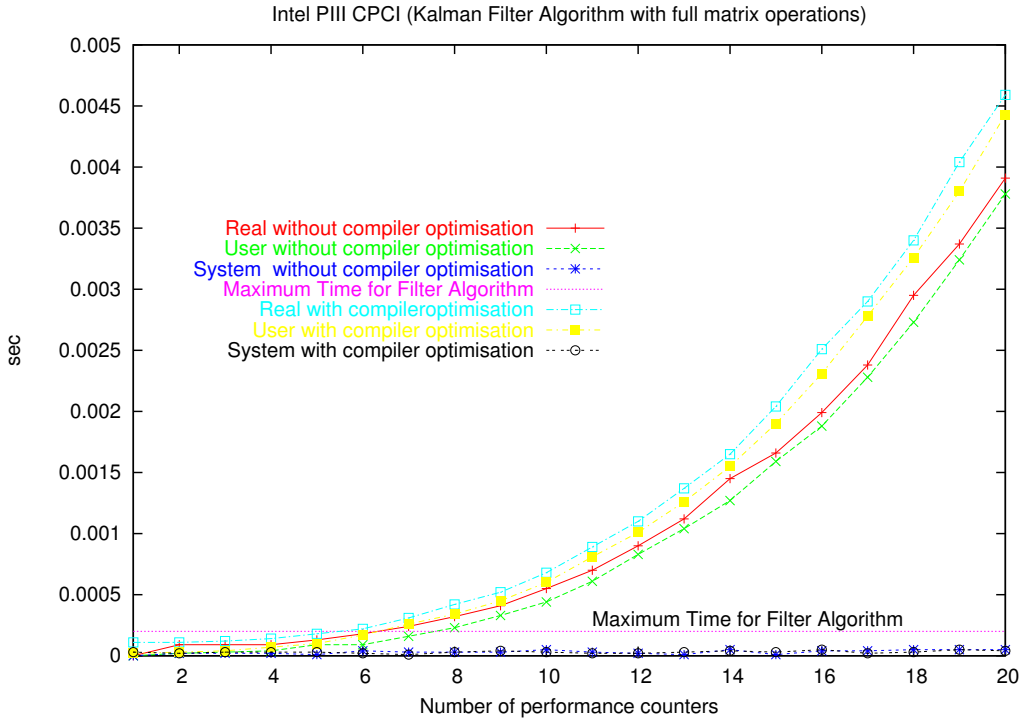

Figure 5.33: Full Kalman Operations on a Intel PIII CompactPCI system high resolution

Chapter 6

Optimisation and Re-evaluation of the Estimation Algorithm

A solution to this performance problem might be obtained by analysing the individual matrix operations that implement the Kalman filter. An implementation that models individual PMCs as independent random processes leads to a sparse linear system. Initial optimisation focused on a reduction in execution time and not on the memory consumed. This chapter looks at these matrix operations in detail and suggests optimisations that might improve the performance sufficiently to make it feasible to observe a reasonable number of PMCs in a DSM system.

6.1 Sparse Matrix Optimisation

Assume that for all the Kalman filter matrix operations in this section that the PMC readings are modeled as independent integrated Gauss-Markov random processes as described in section 5.1.2. These models lead to sparse linear systems and this section analyses the individual matrix operations for opportunities to optimise the execution speed. This should make the overall algorithm suitable to estimate and predict a reasonable number of PMCs. All these filter operations are summarised in Table 5.4.

6.1.1 The Kalman Gain

Eq.(5.11) is here shown again for convenience.

$$\boldsymbol{K_k} = \frac{\boldsymbol{P_k^- H_k^T}}{\boldsymbol{H_k P_k^- H_k^T} + \boldsymbol{R_k}}$$

This equation is implemented in six steps that require ether a matrix multiplication, addition or inverse.

The **first step** (see Table 5.4) multiplies the a priori error covariance matrix $\boldsymbol{P_k^-}$ Eq.(6.25) with the transpose of the measurement matrix $\boldsymbol{H_k^T}$ Eq.(6.2). The result is a new matrix $\boldsymbol{P_k^- H_k^T}$. Eq.(6.1) shows the elements of the three matrices that are populated and that are consequently involved in the computation of the $\boldsymbol{P_k^- H_k^T}$ matrix. All three matrices are sparse matrices. Advantage can be taken of this sparseness by writing out the equation explicitly and therefore calculating only the elements of the matrices that require this computation. A slightly more elegant solution is to express this in a loop since it is likely that this loop is unrolled by a compiler optimisation. This reduces the matrix multiplication to a single loop.

$$\boldsymbol{P_k^- H_k^T} = \begin{bmatrix} c_{11} & 0 & \dots & 0 \\ c_{21} & 0 & \dots & 0 \\ 0 & c_{32} & \dots & 0 \\ 0 & c_{42} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & c_{(m-1)n} \\ 0 & 0 & \dots & c_{mn} \end{bmatrix} = \tag{6.1}$$

$$= \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \dots & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & \dots & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & a_{(m-1)(n-1)} & a_{(m-1)n} \\ 0 & 0 & 0 & 0 & \dots & a_{m(n-1)} & a_{mn} \end{bmatrix} *$$

$$* \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ 0 & 0 & \dots & 0 \end{bmatrix} \tag{6.2}$$

The **second step** (see Table 5.4) multiplies the measurement matrix $\boldsymbol{H_k}$ Eq.(6.4)with the a priori error covariance matrix $\boldsymbol{P_k^-}$ Eq.(6.25). The result is a new matrix $\boldsymbol{H_k P_k^-}$. Eq.(6.3) shows the elements of the three matrices that are populated and that are consequently involved in the computation of the $\boldsymbol{H_k P_k^-}$ matrix. As in step one all three matrices are sparse matrices. Again, advantage can be taken of this sparseness by writing out the

equation explicitly and therefore calculating only the elements of the matrices that require this computation. Again it can be expressed in a loop since it is likely that this loop is unrolled by a compiler optimisation, and hence reduced to a single loop.

$$\boldsymbol{H_k P_k^-} = \begin{bmatrix} \boldsymbol{c_{11}} & \boldsymbol{c_{12}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{c_{23}} & \boldsymbol{c_{24}} & \boldsymbol{\dots} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{0} & \boldsymbol{\ddots} & \boldsymbol{\vdots} & \boldsymbol{\vdots} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{c_{m(n-1)}} & \boldsymbol{c_{mn}} \end{bmatrix} = \tag{6.3}$$

$$= \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{0} & \boldsymbol{\ddots} & \boldsymbol{\vdots} & \boldsymbol{\vdots} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{1} & \boldsymbol{0} \end{bmatrix} * \tag{6.4}$$

$$* \begin{bmatrix} \boldsymbol{b_{11}} & \boldsymbol{b_{12}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{b_{21}} & \boldsymbol{b_{22}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{b_{33}} & \boldsymbol{b_{34}} & \boldsymbol{\dots} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{b_{43}} & \boldsymbol{b_{44}} & \boldsymbol{\dots} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{0} & \boldsymbol{\ddots} & \boldsymbol{\vdots} & \boldsymbol{\vdots} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{b_{(m-1)(n-1)}} & \boldsymbol{b_{(m-1)n}} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{b_{m(n-1)}} & \boldsymbol{b_{mn}} \end{bmatrix}$$

The **third step** (see Table 5.4) multiplies the result from step 2 matrix $\boldsymbol{H_k P_k^-}$ Eq.(6.3) with the transpose of the measurement sensitivity matrix $\boldsymbol{H_k^T}$ Eq.(6.2). This calculation results in a new matrix $\boldsymbol{H_k P_k^- H_k^T}$. Eq.(6.5) shows the elements of the three matrices that are populated and that are consequently involved in the computation of the $\boldsymbol{H_k P_k^- H_k^T}$ matrix. As in the previous steps all three matrices are sparse matrices, where the equation can be written explicitly and expressed as a loop that is likely to be reduced by compiler optimisation to a single loop.

$$\boldsymbol{H_k P_k^- H_k^T} = \begin{bmatrix} \boldsymbol{c_{11}} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{c_{22}} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{\ddots} & \boldsymbol{\vdots} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{c_{mn}} \end{bmatrix} = \tag{6.5}$$

$$= \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & a_{23} & a_{24} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & a_{m(n-1)} & a_{mn} \end{bmatrix} \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

The **fourth step** (see Table 5.4) adds the measurement noise covariance matrix $\boldsymbol{R_k}$ Eq.(6.7) to the result from step 3 matrix $\boldsymbol{H_k P_k^- H_k^T}$ Eq.(6.5). The result is a new matrix $\boldsymbol{H_k P_k^- H_k^T + R_k}$. Eq.(6.6) shows the elements of the three matrices that are populated and that are consequently involved in the computation of the $\boldsymbol{H_k P_k^- H_k^T + R_k}$ matrix. As in the previous steps all three matrices are sparse matrices, where the equation is likely to be reduced to a single loop.

$$\boldsymbol{H_k P_k^- H_k^T + R_k} = \begin{bmatrix} c_{11} & 0 & \dots & 0 \\ 0 & c_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & c_{mn} \end{bmatrix} = \tag{6.6}$$

$$= \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & 0 & \dots & 0 \\ 0 & b_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & b_{mn} \end{bmatrix} \tag{6.7}$$

The **fifth step** (see Table 5.4) inverts the result from step 4 matrix $\boldsymbol{H_k P_k^- H_k^T + R_k}$ Eq.(6.6). This operation results in a new matrix $\boldsymbol{(H_k P_k^- H_k^T + R_k)^{-1}}$. Eq.(6.8) shows the elements of the two matrices that are populated and that are consequently involved in the computation of the $\boldsymbol{(H_k P_k^- H_k^T + R_k)^{-1}}$ matrix. As in the previous steps both matrices are sparse matrices, where the equation is likely to be reduced to a single loop.

$$(\boldsymbol{H_k}\ \boldsymbol{P_k^-}\ \boldsymbol{H_k^T} + \ \boldsymbol{R_k})^{-1} \ = \ \begin{bmatrix} \boldsymbol{c_{11}} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{c_{22}} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{\ddots} & \boldsymbol{\vdots} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{c_{mn}} \end{bmatrix} = \tag{6.8}$$

$$= \begin{bmatrix} \boldsymbol{a_{11}} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{a_{22}} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{\ddots} & \boldsymbol{\vdots} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{a_{mn}} \end{bmatrix}^{-1}$$

The **sixth and last step** (see Table 5.4) computes the Kalman gain $\boldsymbol{K_k}$ Eq.(6.9) by multiplying the result from step 1 matrix $\boldsymbol{P_k^- H_k^T}$ Eq.(6.1) with the inverse matrix from the previous step $\boldsymbol{(H_k P_k^- H_k^T + R_k)^{-1}}$. Eq.(6.8) shows the elements of the three matrices that are populated and that are consequently involved in the computation of the $\boldsymbol{(H_k P_k^- H_k^T + R_k)^{-1}}$ matrix. As in the previous steps all three matrices are sparse matrices, where the equation is likely to be reduced to a single loop.

$$\boldsymbol{K_k} \ = \ \begin{bmatrix} \boldsymbol{c_{11}} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{c_{21}} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{c_{32}} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{c_{42}} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{\ddots} & \boldsymbol{\vdots} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{c_{(m-1)n}} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{c_{mn}} \end{bmatrix} = \tag{6.9}$$

$$= \begin{bmatrix} \boldsymbol{a_{11}} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{a_{21}} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{a_{32}} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{a_{42}} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{\ddots} & \boldsymbol{\vdots} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{a_{(m-1)n}} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{a_{mn}} \end{bmatrix} * \begin{bmatrix} \boldsymbol{b_{11}} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{b_{22}} & \boldsymbol{\dots} & \boldsymbol{0} \\ \boldsymbol{\vdots} & \boldsymbol{\vdots} & \boldsymbol{\ddots} & \boldsymbol{\vdots} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\dots} & \boldsymbol{b_{mn}} \end{bmatrix}$$

6.1.2 The A Posteriori State Estimate

Eq.(5.6) is here shown again for convenience.

$$\boldsymbol{\hat{x}_k = \hat{x}_k^- + K_k(z_k - \hat{x}_k^- H_k)}$$

This calculation is implemented in four steps that require either a matrix multiplication, addition or subtraction.

The **first step** (see Table 5.4) multiplies the measurement sensitivity matrix $\boldsymbol{H_k}$ Eq.(6.4) with a priori state vector $\hat{\boldsymbol{x}}_{\boldsymbol{k+1}}^{-}$ Eq.(5.13). The result is a new vector $\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}\boldsymbol{H_k}$. Eq.(6.10) shows the elements of the one matrix that are populated and that are consequently involved in the computation of the $\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}\boldsymbol{H_k}$ vector. Only one matrix is a sparse matrix; the rest are vectors. Again, this sparseness can be taken advantage of to reduce the equation towards a single loop.

$$\boldsymbol{H_k}\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-} = \begin{bmatrix} \boldsymbol{c_1} \\ \boldsymbol{c_2} \\ \vdots \\ \boldsymbol{c_m} \end{bmatrix} = \tag{6.10}$$

$$= \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \mathbf{0} & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{1} & \mathbf{0} \end{bmatrix} * \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \\ \boldsymbol{b_4} \\ \boldsymbol{b_5} \\ \vdots \\ \boldsymbol{b_m} \end{bmatrix}$$

In the **second step** (see Table 5.4) the result is subtracted from the previous calculation $\boldsymbol{H_k}\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}$ Eq.(6.10) from the measurement vector $\boldsymbol{Z_k}$ Eq.(6.12). This calculation results in a new vector $\boldsymbol{Z_k} - \boldsymbol{H_k}\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}$ Eq.(6.11). This subtraction involves only vectors that offer no optimisation.

$$\boldsymbol{Z_k} - \boldsymbol{H_k}\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-} = \begin{bmatrix} \boldsymbol{c_1} \\ \boldsymbol{c_2} \\ \vdots \\ \boldsymbol{c_m} \end{bmatrix} = \tag{6.11}$$

$$= \begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \vdots \\ \boldsymbol{a_m} \end{bmatrix} - \tag{6.12}$$

$$- \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

In the **third step** (see Table 5.4) the Kalman gain matrix $\boldsymbol{K_k}$ Eq.(6.9) is multiplied with the result from the previous calculation $\boldsymbol{Z_k - H_k\hat{x}_k^-}$ Eq.(6.11). This calculation presents us a new vector $\boldsymbol{K_k(Z_k - H_k\hat{x}_k^-)}$. Eq.(6.13) shows the elements of the one matrix that are populated and that are consequently involved in the computation of the $\boldsymbol{K_k(Z_k - H_k\hat{x}_k^-)}$ vector. Only one matrix is a sparse matrix; the rest are vectors. This sparseness can be taken advantage of by calculating only the elements of the matrix and vectors that require this computation, as before, and expressing this in a loop that is likely to reduce the matrix vector multiplication to a single loop.

$$\boldsymbol{K_k(Z_k - H_k\hat{x}_k^-)} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ \vdots \\ c_m \end{bmatrix} = \tag{6.13}$$

$$= \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & 0 & \dots & 0 \\ 0 & a_{32} & \dots & 0 \\ 0 & a_{42} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{(m-1)n} \\ 0 & 0 & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{6.14}$$

The **fourth and last step** (see Table 5.4) adds the a priori state vector $\boldsymbol{\hat{x}_{k+1}^-}$ Eq.(5.13) to the result from the previous calculation $\boldsymbol{K_k(Z_k - H_k\hat{x}_k^-)}$ Eq.(6.13). This calculation results in the a posteriori state estimate vector $\boldsymbol{\hat{x}_k}$ Eq.(6.15). This addition involves only vectors that offer no optimisation.

$$\hat{x}_k = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ \vdots \\ c_m \end{bmatrix} = \tag{6.15}$$

$$= \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ \vdots \\ a_m \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ \vdots \\ b_m \end{bmatrix}$$

6.1.3 The A Posteriori Error Covariance

Eq.(5.12) is here shown again for convenience.

$$P_k = (I - K_k H_k) P_k^-$$

This calculation is implemented in three steps that require ether a matrix multiplication or subtraction.

The **first step** (see Table 5.4) multiplies the Kalman gain matrix K_k Eq.(6.9) with the measurement sensitivity matrix H_k Eq.(6.4). The result is a new matrix $K_k H_K$. Eq.(6.16) shows the elements of the three matrices that are populated and that are consequently involved in the computation of the $K_k H_K$ matrix. All three matrices are sparse matrices. Again, the matrix multiplication is likely to be reduced to a single loop.

$$K_k H_k = \begin{bmatrix} c_{11} & 0 & 0 & 0 & \dots & 0 & 0 \\ c_{21} & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & c_{33} & 0 & \dots & 0 & 0 \\ 0 & 0 & c_{43} & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & c_{(m-1)(n-1)} & 0 \\ 0 & 0 & 0 & 0 & \dots & c_{m(n-1)} & 0 \end{bmatrix} = \tag{6.16}$$

$$= \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & 0 & \dots & 0 \\ 0 & a_{32} & \dots & 0 \\ 0 & a_{42} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{(m-1)n} \\ 0 & 0 & \dots & a_{mn} \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \end{bmatrix}$$

The **second step** (see Table 5.4) subtracts the result from the previous calculation $\boldsymbol{K_k H_K}$ Eq.(6.16) from the identity matrix $\boldsymbol{I}$ Eq.(6.18). This calculation results in a new matrix $\boldsymbol{I - K_k H_k}$. Eq.(6.17) shows the elements of the three matrices that are populated and that are consequently involved in the computation of the $\boldsymbol{I - K_k H_k}$ matrix. As in the previous steps all three matrices are sparse matrices. Again, the matrix subtraction is likely to be reduced to a single loop.

$$\boldsymbol{I - K_k H_k} = \begin{bmatrix} c_{11} & 0 & 0 & 0 & \dots & 0 & 0 \\ c_{21} & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & c_{33} & 0 & \dots & 0 & 0 \\ 0 & 0 & c_{43} & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & c_{(m-1)(n-1)} & 0 \\ 0 & 0 & 0 & 0 & \dots & c_{m(n-1)} & 1 \end{bmatrix} = \tag{6.17}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} - \tag{6.18}$$

$$- \begin{bmatrix} b_{11} & 0 & 0 & 0 & \dots & 0 & 0 \\ b_{21} & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & b_{33} & 0 & \dots & 0 & 0 \\ 0 & 0 & b_{43} & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & b_{(m-1)(n-1)} & 0 \\ 0 & 0 & 0 & 0 & \dots & b_{m(n-1)} & 0 \end{bmatrix}$$

In the **third and last step** (see Table 5.4) the result from the previous step $\boldsymbol{I - K_k H_k}$ Eq.(6.17) is multiplied with the $\boldsymbol{P_k^-}$ Eq.(6.25). The result is the a posteriori error covariance matrix $\boldsymbol{P_k}$. Eq.(6.19) shows the elements of the three matrices that are populated and that are consequently involved in the computation of the $\boldsymbol{P_k}$ matrix. As in the previous steps all three matrices are sparse matrices. Again, the matrix multiplication is likely to be reduced to a single loop.

$$
\begin{aligned}
\boldsymbol{P_k} &= \begin{bmatrix}
c_{11} & c_{12} & 0 & 0 & \dots & 0 & 0 \\
c_{21} & c_{22} & 0 & 0 & \dots & 0 & 0 \\
0 & 0 & c_{33} & c_{34} & \dots & 0 & 0 \\
0 & 0 & c_{43} & c_{44} & \dots & 0 & 0 \\
\vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \dots & c_{(m-1)(n-1)} & c_{(m-1)n} \\
0 & 0 & 0 & 0 & \dots & c_{m(n-1)} & c_{mn}
\end{bmatrix} = \\
&= \begin{bmatrix}
a_{11} & 0 & 0 & 0 & \dots & 0 & 0 \\
a_{21} & 1 & 0 & 0 & \dots & 0 & 0 \\
0 & 0 & a_{33} & 0 & \dots & 0 & 0 \\
0 & 0 & a_{43} & 1 & \dots & 0 & 0 \\
\vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \dots & a_{(m-1)(n-1)} & 0 \\
0 & 0 & 0 & 0 & \dots & a_{m(n-1)} & 1
\end{bmatrix} * \\
&* \begin{bmatrix}
b_{11} & b_{12} & 0 & 0 & \dots & 0 & 0 \\
b_{21} & b_{22} & 0 & 0 & \dots & 0 & 0 \\
0 & 0 & b_{33} & b_{34} & \dots & 0 & 0 \\
0 & 0 & b_{43} & b_{44} & \dots & 0 & 0 \\
\vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \dots & b_{(m-1)(n-1)} & b_{(m-1)n} \\
0 & 0 & 0 & 0 & \dots & b_{m(n-1)} & b_{mn}
\end{bmatrix}
\end{aligned}
\tag{6.19}
$$

6.1.4 The A Priori State Vector

Eq.(5.13) is here shown again for convenience.

$$\hat{\boldsymbol{x}}_{\boldsymbol{k+1}}^- = \boldsymbol{\phi_k}\hat{\boldsymbol{x}}_{\boldsymbol{k}}$$

This calculation is implemented in one step and requires one multiplication only.

In the **first step and only step** (see Table 5.4) the discrete state transition matrix $\boldsymbol{\phi_k}$ Eq.(5.18) is multiplied with the a posteriori state vector $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$ Eq.(6.15). The result is the a

priori state vector $\hat{\boldsymbol{x}}_{\boldsymbol{k+1}}^{-}$. Eq.(6.20) shows the elements of the one matrix that are populated and that are consequently involved in the computation of the $\hat{\boldsymbol{x}}_{\boldsymbol{k+1}}^{-}$ vector. Only one matrix is a sparse matrix the rest are vectors. Again, the matrix vector multiplication is likely to be reduced to a single loop.

$$\hat{\boldsymbol{x}}_{\boldsymbol{k+1}}^{-} = \begin{bmatrix} \boldsymbol{c_1} \\ \boldsymbol{c_2} \\ \boldsymbol{c_3} \\ \boldsymbol{c_4} \\ \boldsymbol{c_5} \\ \vdots \\ \boldsymbol{c_m} \end{bmatrix} = \tag{6.20}$$

$$= \begin{bmatrix} \mathbf{1} & \boldsymbol{a_{12}} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{a_{22}} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \boldsymbol{a_{34}} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{a_{44}} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \mathbf{0} & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{1} & \boldsymbol{a_{(m-1)n}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \boldsymbol{a_{mn}} \end{bmatrix} *$$

$$* \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \\ \boldsymbol{b_4} \\ \boldsymbol{b_5} \\ \vdots \\ \boldsymbol{b_m} \end{bmatrix} \tag{6.21}$$

6.1.5 The A Priori Error Covariance Matrix

Eq.(6.1.5) is here shown again for convenience.

$$\boldsymbol{P}_{\boldsymbol{k+1}}^{-} = \boldsymbol{\phi_k} \boldsymbol{P_k} \boldsymbol{\phi_k^T} + \boldsymbol{Q_k}$$

This calculation is implemented in three steps that require two matrix multiplications and one subtraction.

The **first step** (see Table 5.4) multiplies the discrete state transition matrix $\boldsymbol{\phi_k}$ Eq.(5.18) with the a posteriori error covariance matrix $\boldsymbol{P_k}$ Eq.(6.19). The result is a new matrix

$\boldsymbol{\phi_k P_k}$. Eq.(6.22) shows the elements of the three matrices that are populated and that are consequently involved in the computation of the $\boldsymbol{\phi_k P_k}$ matrix. All three matrices are sparse matrices. Again, the matrix multiplication is likely to be reduced to a single loop.

$$\boldsymbol{\phi_k P_k} = \begin{bmatrix} c_{11} & c_{12} & 0 & 0 & \dots & 0 & 0 \\ c_{21} & c_{22} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & c_{33} & c_{34} & \dots & 0 & 0 \\ 0 & 0 & c_{43} & c_{44} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & c_{(m-1)(n-1)} & c_{(m-1)n} \\ 0 & 0 & 0 & 0 & \dots & c_{m(n-1)} & c_{mn} \end{bmatrix} = \tag{6.22}$$

$$= \begin{bmatrix} 1 & a_{12} & 0 & 0 & \dots & 0 & 0 \\ 0 & a_{22} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & a_{34} & \dots & 0 & 0 \\ 0 & 0 & 0 & a_{44} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & a_{(m-1)n} \\ 0 & 0 & 0 & 0 & \dots & 0 & a_{mn} \end{bmatrix} *$$

$$* \begin{bmatrix} c_{11} & c_{12} & 0 & 0 & \dots & 0 & 0 \\ c_{21} & c_{22} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & c_{33} & c_{34} & \dots & 0 & 0 \\ 0 & 0 & c_{43} & c_{44} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & c_{(m-1)(n-1)} & c_{(m-1)n} \\ 0 & 0 & 0 & 0 & \dots & c_{m(n-1)} & c_{mn} \end{bmatrix}$$

In the **second step** (see Table 5.4) the result from the previous step $\boldsymbol{\phi_k P_k}$ Eq.(6.22) is multiplied with the transposed discrete state transition matrix $\boldsymbol{\phi_k^T}$ Eq.(6.24). The result is a new matrix $\boldsymbol{\phi_k P_k \phi_k^T}$. Eq.(6.23) shows the elements of the three matrices that are populated and that are consequently involved in the computation of the $\boldsymbol{\phi_k P_k \phi_k^T}$ matrix. As in step one all three matrices are sparse matrices. Again, the matrix multiplication is likely to be reduced to a single loop.

$$\phi_k P_k \phi_k^T = \begin{bmatrix} c_{11} & c_{12} & 0 & 0 & \dots & 0 & 0 \\ c_{21} & c_{22} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & c_{33} & c_{34} & \dots & 0 & 0 \\ 0 & 0 & c_{43} & c_{44} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & c_{(m-1)(n-1)} & c_{(m-1)n} \\ 0 & 0 & 0 & 0 & \dots & c_{m(n-1)} & c_{mn} \end{bmatrix} = \tag{6.23}$$

$$= \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \dots & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & \dots & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & a_{(m-1)(n-1)} & a_{(m-1)n} \\ 0 & 0 & 0 & 0 & \dots & a_{m(n-1)} & a_{mn} \end{bmatrix} *$$

$$* \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ b_{21} & b_{22} & & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & b_{43} & b_{44} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & b_{(m-1)n} \\ 0 & 0 & 0 & 0 & \dots & b_{m(n-1)} & a_{mn} \end{bmatrix} \tag{6.24}$$

The **third and last step** (see Table 5.4) adds the result from the previous calculation $\phi_k P_k \phi_k^T$ Eq.(6.23) to the process noise covariance matrix Q_k Eq.(5.10) and Eq.(5.16). This calculation yields the a posteriori state estimate vector P_{k+1}^-. Eq.(6.25) shows the elements of the three matrices that are populated and that are consequently involved in the computation of the P_{k+1}^- matrix. As in the previous steps all three matrices are sparse matrices. Again, the matrix addition is likely to be reduced to a single loop.

$$P_{k+1}^- = \begin{bmatrix} c_{11} & c_{12} & 0 & 0 & \dots & 0 & 0 \\ c_{21} & c_{22} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & c_{33} & c_{34} & \dots & 0 & 0 \\ 0 & 0 & c_{43} & c_{44} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & c_{(m-1)(n-1)} & c_{(m-1)n} \\ 0 & 0 & 0 & 0 & \dots & c_{m(n-1)} & c_{mn} \end{bmatrix} = \tag{6.25}$$

$$= \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \dots & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & \dots & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & a_{(m-1)(n-1)} & a_{(m-1)n} \\ 0 & 0 & 0 & 0 & \dots & a_{m(n-1)} & a_{mn} \end{bmatrix} +$$

$$+ \begin{bmatrix} b_{11} & b_{12} & 0 & 0 & \dots & 0 & 0 \\ b_{21} & b_{22} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & b_{33} & b_{34} & \dots & 0 & 0 \\ 0 & 0 & b_{43} & b_{44} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & b_{(m-1)(n-1)} & b_{(m-1)n} \\ 0 & 0 & 0 & 0 & \dots & b_{m(n-1)} & b_{mn} \end{bmatrix}$$

6.2 Optimisation Analysis

Fig. 6.1, Fig. 6.2 and Fig. 6.3 show the real, user and system execution time over the number of PMCs estimated or predicted by the Kalman filter for the three machines specified in Table 7.1. This is identical to graphs seen in Fig. 5.28, Fig. 5.29 and Fig. 5.30 but with all the amendments to the filter algorithm that were suggested in section 6.1. The modified algorithm now exploits the sparseness of the matrices involved in the algorithm's execution. The execution time is reduced to such an extent that even the *Maximum Time for Filter Algorithm* threshold $= \frac{\Delta t}{100}$ is no longer exceeded for most measurements. For consistency Fig. 6.4, Fig. 6.5 and Fig. 6.6 depict the same information with a higher execution time resolution. Unlike in the case of the full matrix operations the *Maximum Time for Filter Algorithm* is now off-scale.

These are promising results that show that it is feasible to run the sparse version of the algorithm for up to 120 and more PMCs while sampling these PMC readings at 20 ms without consuming more than 1% of the available CPU time. A comparison for both algorithm versions is given in Table 6.1. The maximum number of 120 PMCs for these experiments was chosen because this is approximately the number of counter events on a PIII system.

Furthermore Fig. 6.7, Fig. 6.8 and Fig. 6.9 allow us to compare the three systems. The figures show the user execution time over the number of PMCs estimated by the algorithm for both the full matrix operations and the sparse operation. Fig. 6.8 is particular interesting because it shows how quickly the algorithm with the full matrix operation reaches the sampling time (20 ms). Also Fig. 6.9 depicts when the full matrix operation exceeds 1% of the

sample time (0.0002 sec). User execution time was measured for a gcc compiler-optimised (-O3) algorithm.

An alternative view is provided by Fig. 6.10, Fig. 6.11 and Fig. 6.12. These three figures show the ratio between the sample time (assuming sampling at 20 ms) and the User plus System execution time. Particularly Fig. 6.11 demonstrates how the sparse algorithm is $\frac{1}{100}$ of the sampling time for the CompactPCI system and $\frac{1}{250}$ of the sampling time for the P4 system while estimating 120 PMCs.

See section 8.3 on page 141 for a description of the work loads.

Full Algorithm		
System	Compiler Optimised	No Compiler Optimisation
Intel P4 Hyperthreaded	8	6
Intel PIII 2 Way SMP	7	7
Intel PIII CompactPCI System	7	7
Sparse Algorithm		
System	Compiler Optimised	No Compiler Optimisation
Intel P4 Hyperthreaded	$\gg 120$	$\gg 120$
Intel PIII 2 Way SMP	> 120	≈ 120
Intel PIII CompactPCI System	≈ 120	≈ 100

Table 6.1: This table shows the maximum number of PMC readings that can be processed by the Kalman filter algorithm on the various machines with exceeding 1% of the available CPU execution time (Maximum Time for Filter Algorithm threshold) assuming a sample time $\boldsymbol{\Delta t}$ of 20 ms. The maximum number of PMC readings is provided for the algorithm that implements the full matrix operations and the algorithm the exploits the sparseness of the matrices.

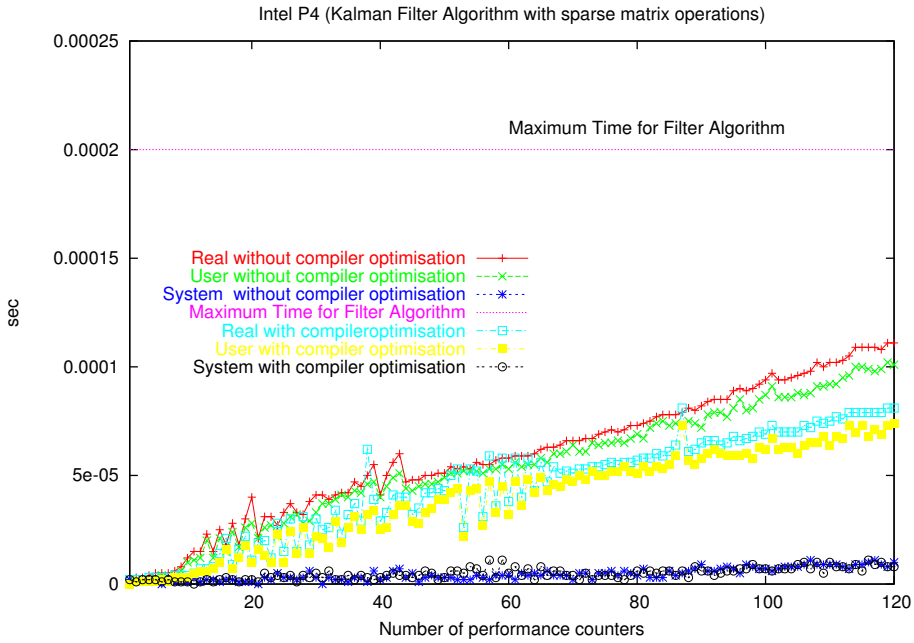

Figure 6.1: Sparse Kalman Operations on a Intel P4

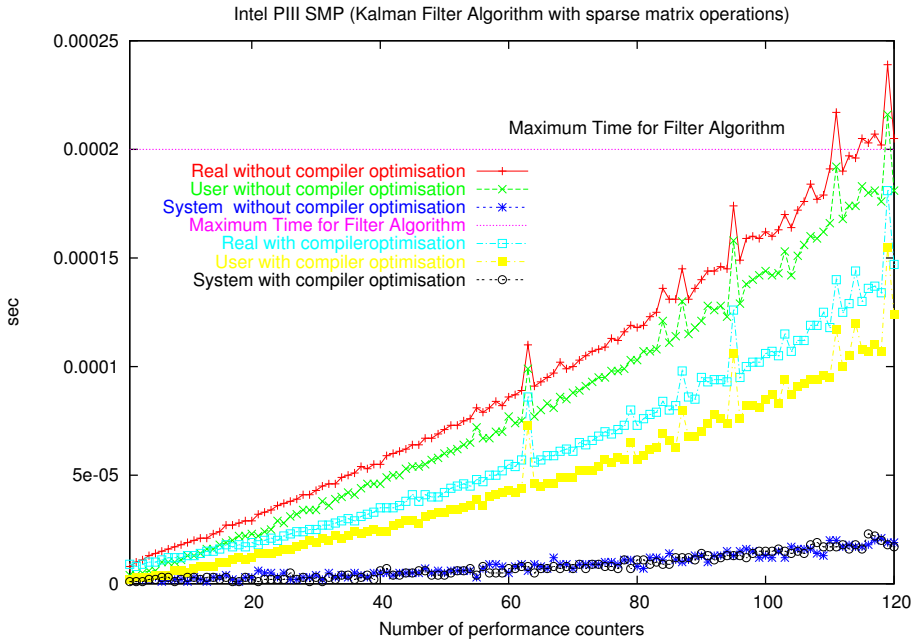

Figure 6.2: Sparse Kalman Operations on a 2 way Intel PIII SMP

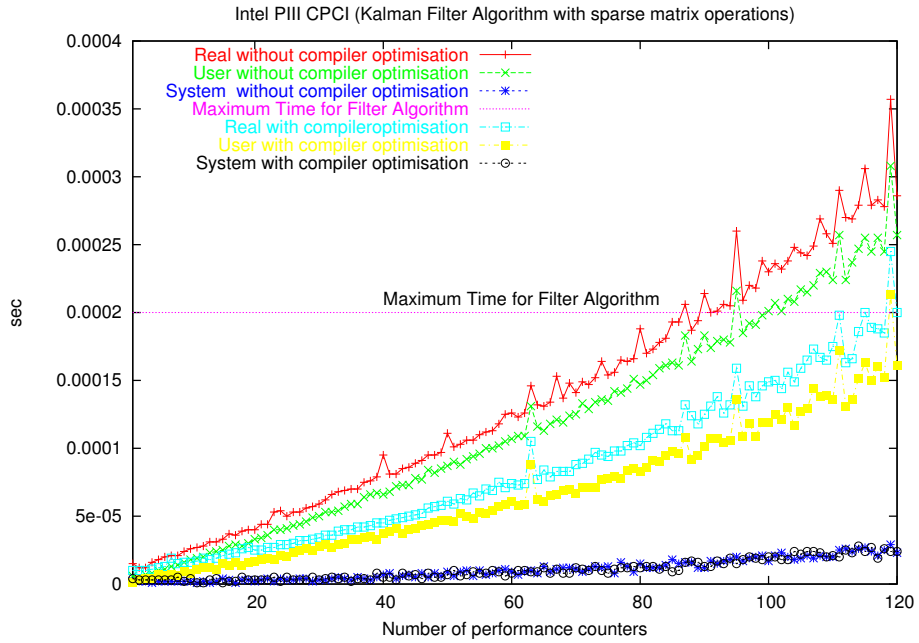

Figure 6.3: Sparse Kalman Operations on a Intel PIII CompactPCI system

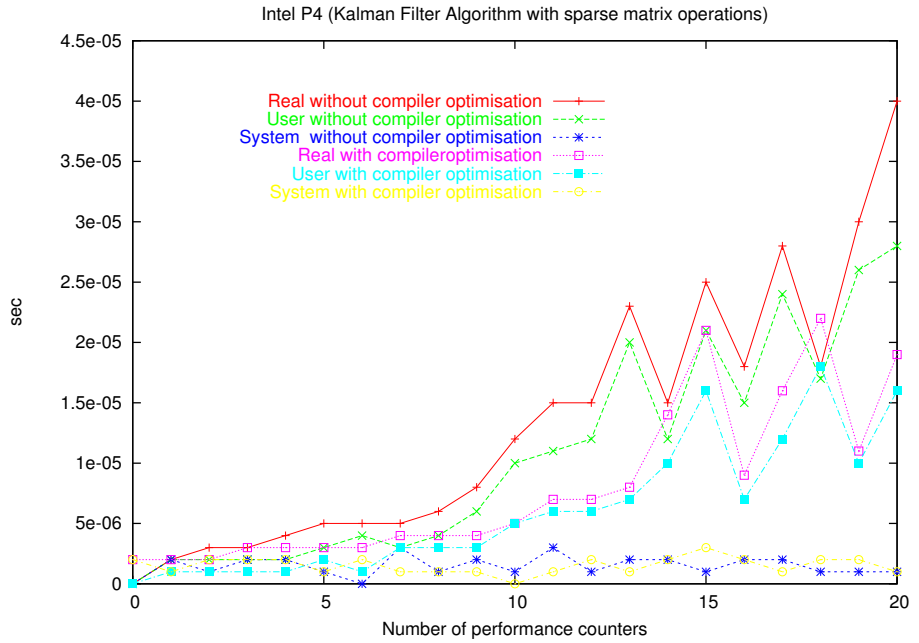

Figure 6.4: Sparse Kalman Operations on a Intel P4 high resolution

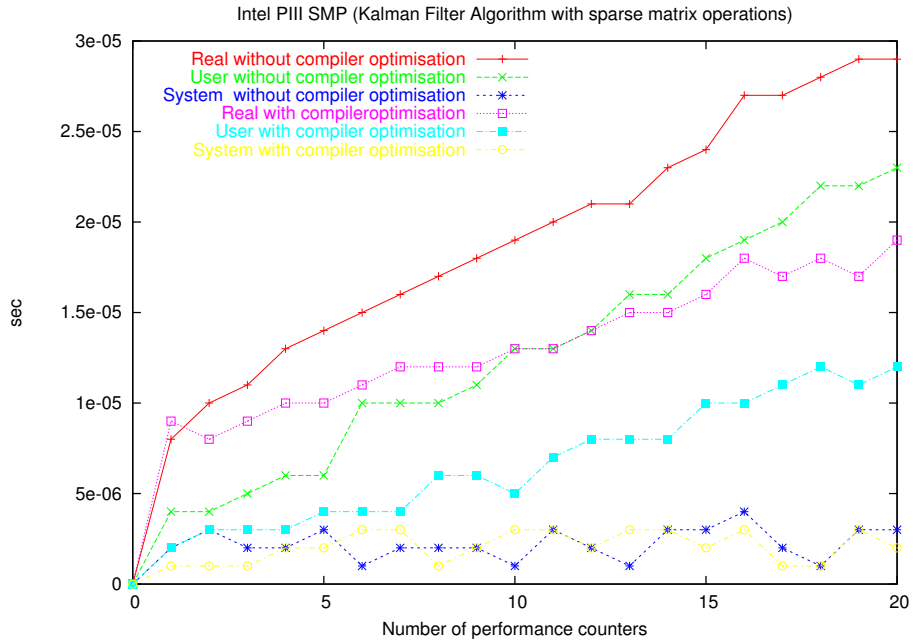

Figure 6.5: Sparse Kalman Operations on a 2 way Intel PIII SMP high resolution

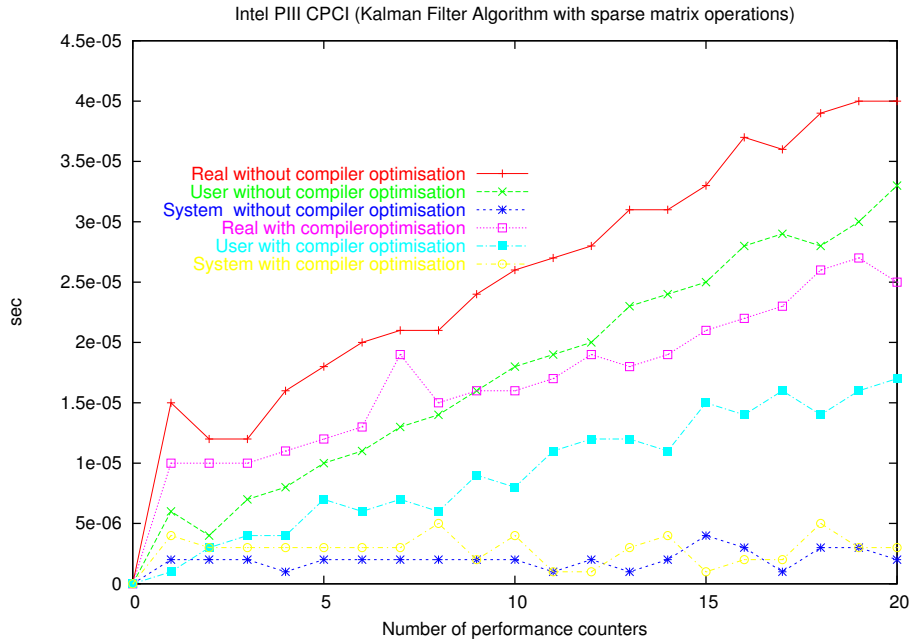

Figure 6.6: Sparse Kalman Operations on a Intel PIII CompactPCI system high resolution

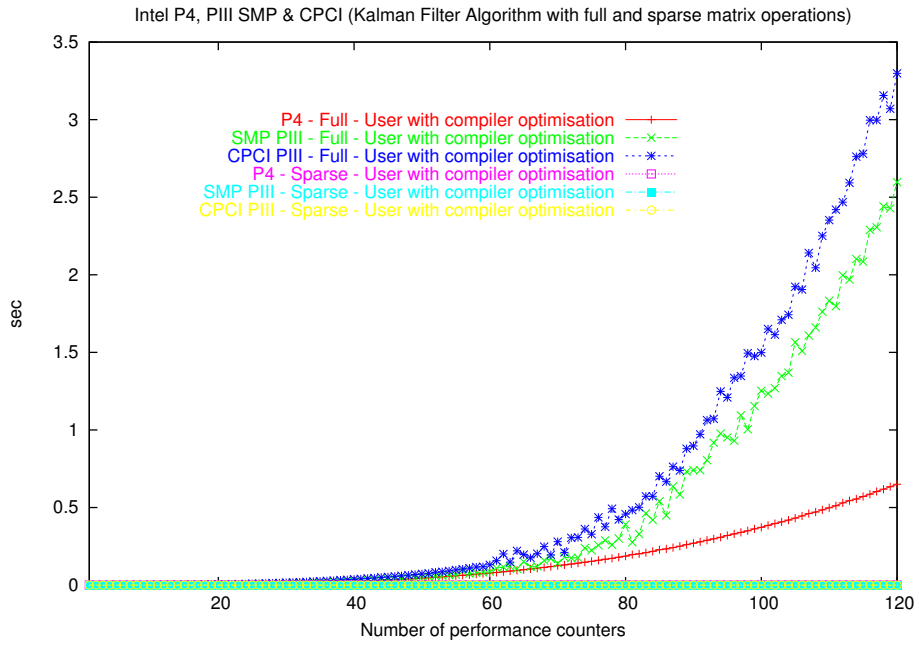

Figure 6.7: User Time Sparse and Full Filter Operations on Intel P4, 2 way Intel PIII SMP and Intel PIII CompactPCI system. User execution time was measured for a gcc compiler optimised (-O3) algorithm.

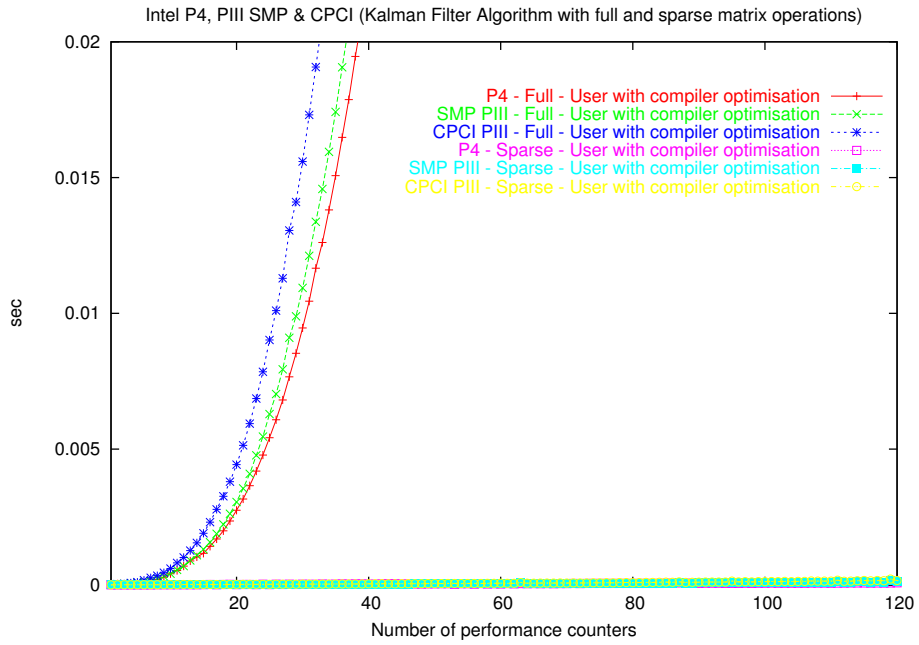

Figure 6.8: User Time Sparse and Full Filter Operations on Intel P4, 2 way Intel PIII SMP and Intel PIII CompactPCI system at high resolution. User execution time was measured for a gcc compiler optimised (-O3) algorithm.

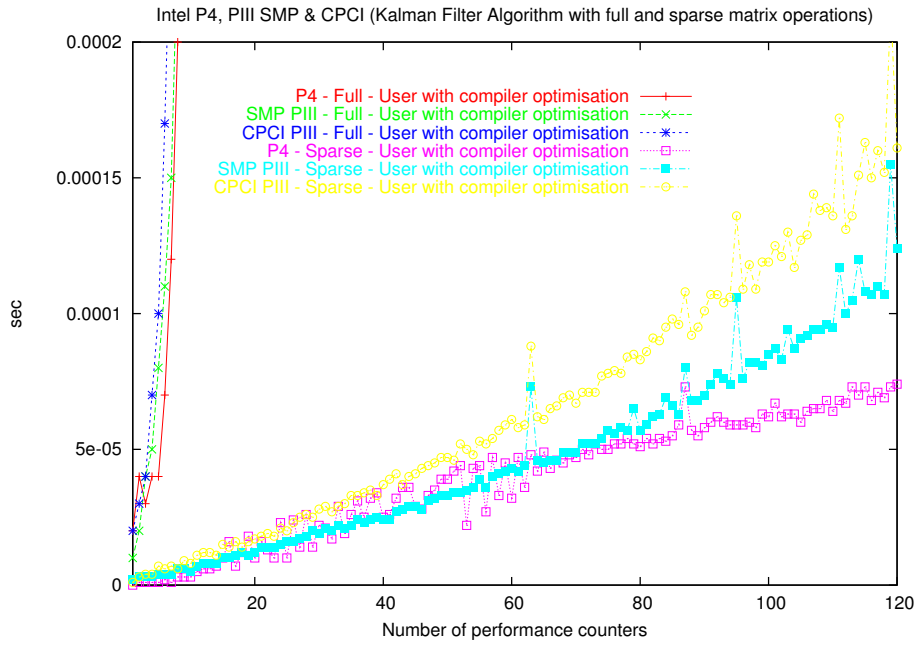

Figure 6.9: User Time Sparse and Full Filter Operations on Intel P4, 2 way Intel PIII SMP and Intel PIII CompactPCI system at very high resolution. User execution time was measured for a gcc compiler optimised (-O3) algorithm.

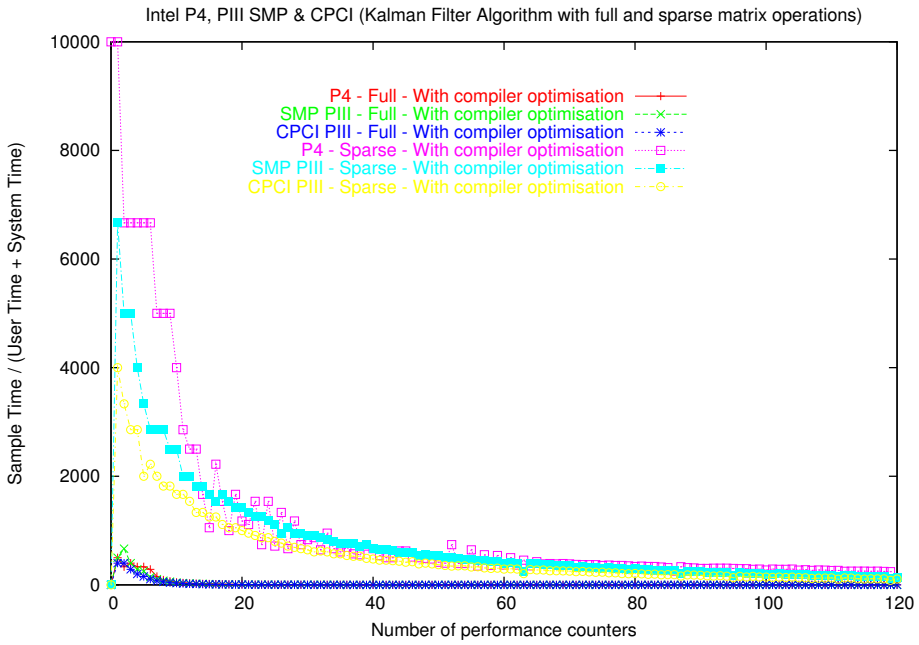

Figure 6.10: Sample time to user and system time ratio on Intel P4, 2 way Intel PIII SMP and Intel PIII CompactPCI system. User execution time was measured for a gcc compiler optimised (-O3) algorithm.

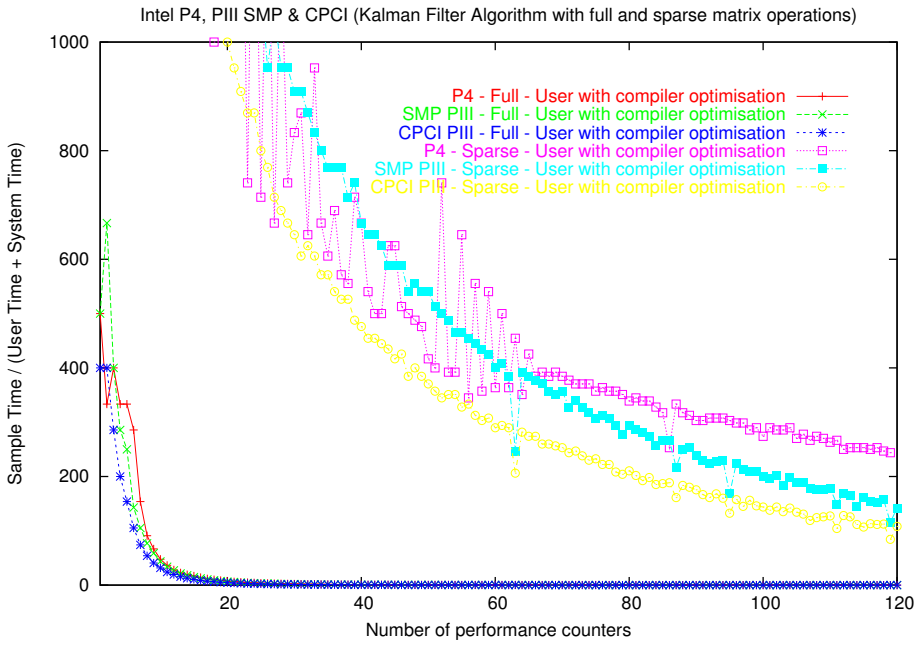

Figure 6.11: Sample time to user and system time ratio on Intel P4, 2 way Intel PIII SMP and Intel PIII CompactPCI system at high resolution. User execution time was measured for a gcc compiler optimised (-O3) algorithm.

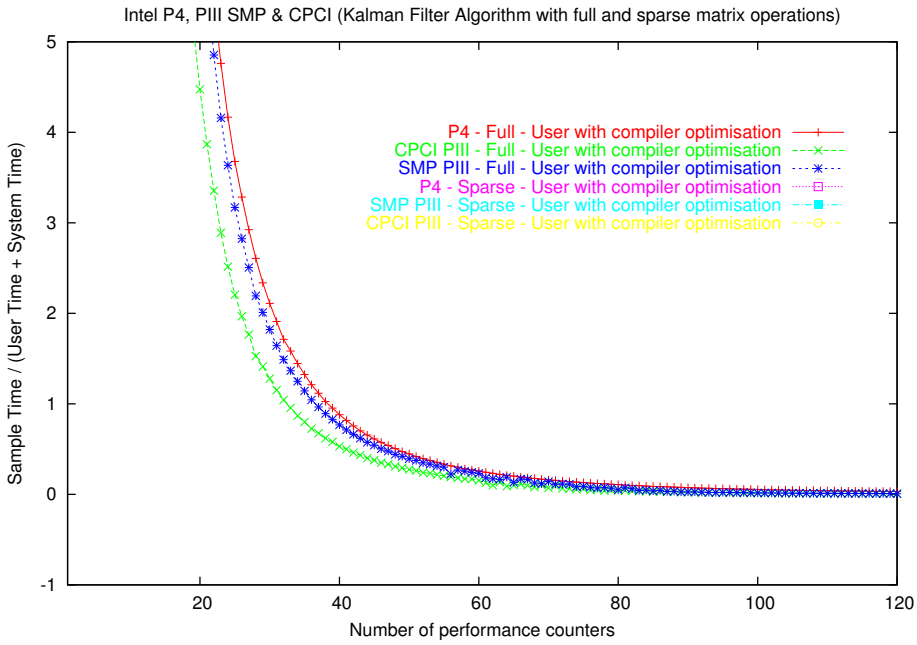

Figure 6.12: Sample time to user and system time ratio on Intel P4, 2 way Intel PIII SMP and Intel PIII CompactPCI system at very high resolution. User execution time was measured for a gcc compiler optimised (-O3) algorithm.

6.3 Memory Hierarchy Performance Measurements Evaluation for Uniprocessor Systems

It is now feasible to extend the evaluation of the filter. Fig. 6.13, Fig. 6.14 and Fig. 6.15 demonstrate the performance of the one-at-a-time filter solution. The algorithm fuses three sets of performance counters into the state vector. The three figures show only one counter and the associated state variables. In Fig. 6.13 the state variable $\boldsymbol{x_1}$ estimates the events/sec from the noisy counter readings despite the fact that only every third sample interval provides data for the filter's blending operation. It is obvious that the filter does not trust the counter readings entirely hence the difference between state variable $\boldsymbol{x_1}$ and the counter readings. The two graphs Fig. 6.14 and Fig. 6.15 demonstrate the algorithm's ability to follow rapid changes.

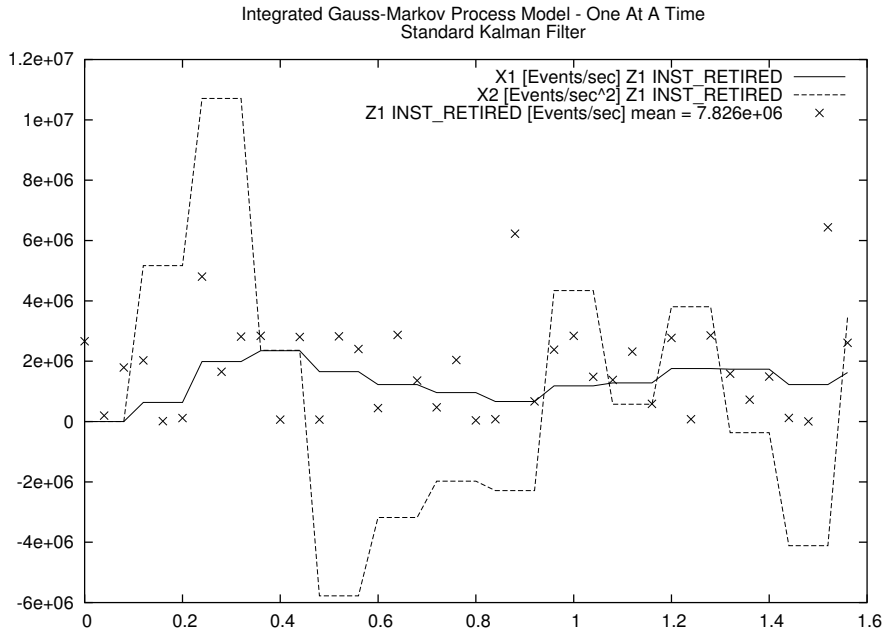

Figure 6.13: One-at-time Filter with integrated Gauss-Markov process model shows two state variables and a performance counter reading.

In order to demonstrate the one-at-a-time filter with a larger number of counters, for example with six PMC events, selected events are presented in Table 6.2. These events and their associated process model parameters are fed into the filter that was introduced in section 5.1 on page 62. Table 6.2 holds the selected PMC events and parameters. This table is a subset of table B.2. The parameters for these PMC events were measured and calculated with techniques described in section 5.2 on page 71.

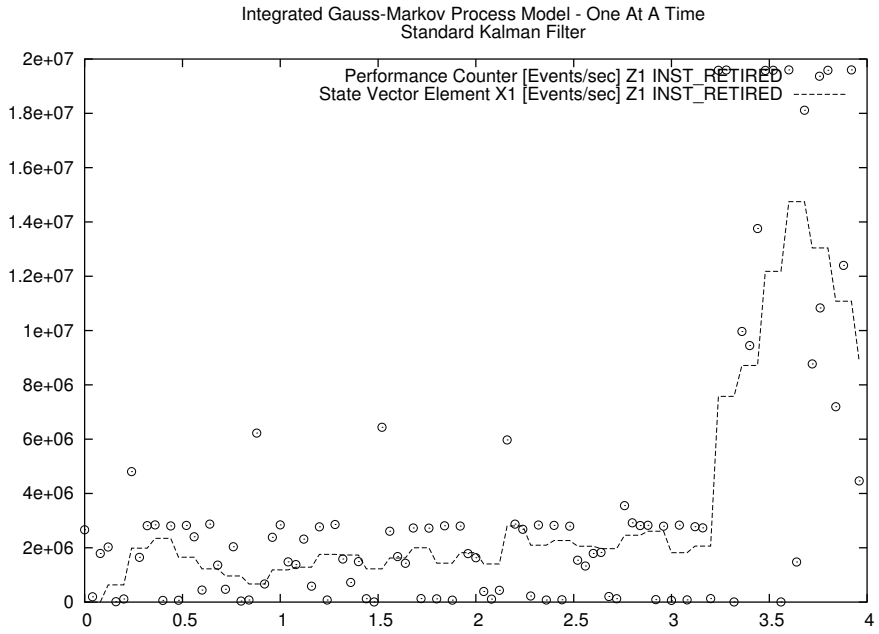

Figure 6.14: One-at-time Filter with integrated Gauss-Markov process model shows one state variable and a performance counter reading over a longer time.

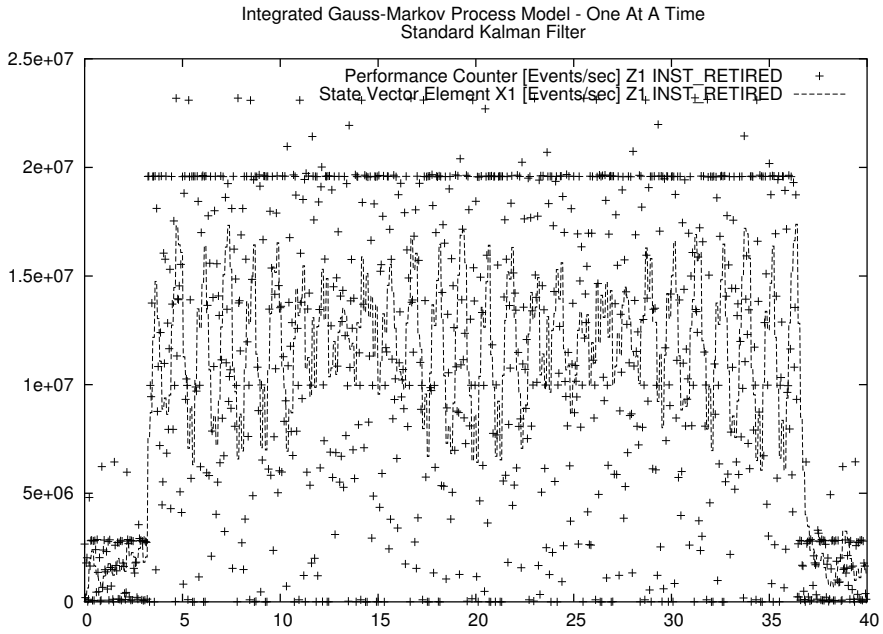

Figure 6.15: One-at-time Filter with integrated Gauss-Markov process model shows one state variable and a performance counter reading over an even longer time.

#	Symbol	CPU	σ_i	β_i	Mean
1	DATA_MEM_REFS	1	5.26e+06	38.8	6.19e+06
		2	5.03e+06	35.7	5.73e+06
2	DCU_LINES_IN	1	5.83e+04	49.2	7.40e+04
		2	5.92e+04	46.5	7.40e+04
6	IFU_IFETCH	1	6.65e+06	55.3	8.40e+06
		2	6.59e+06	51.5	8.13e+06
7	IFU_IFETCH_MISS	1	2.41e+04	34.3	2.77e+04
		2	2.48e+04	34.5	2.86e+04
26	L2_LINES_IN	1	2.02e+04	112.7	2.53e+04
		2	1.92e+04	103.0	2.45e+04
89	INST_RETIRED	1	7.25e+06	40.0	8.71e+06
		2	6.43e+06	32.9	7.04e+06

Table 6.2: Selected PIII Performance Monitoring Counter (PMC) events for Memory Hierarchy Experiment

The uniprocessor filter implementation estimates the six selected PMC events counts by fusing the readings into its state-vector. Fig. 6.16 shows graphs for the six estimated event counts. Remember that only two of these events are sampled at any sample interval. The remaining four are estimated. The PMC count estimates actually displayed are computed from the state variables associated with a particular counter. This is a very simple use of the filter, but nicely demonstrates how the information from the state vector can be used to observe variables that could not be measured with the available PMC registers.

6.3.1 Derived Performance Measurements

For some variables, measurements require the simultaneous use of multiple PMC events. L1 Instructions Fetch Unit (IFU) Hit Rate Eq.(6.26), L2 Cache Hit Rate Eq.(6.27), L1 to L2 Bandwidth MB/s Eq.(6.28) and L2 to Memory Bandwidth MB/s Eq.(6.29) are four simple examples of performance measurements that cannot be directly read. These variables must be calculated from sampled PMC readings. For example Eq.(6.27) requires three PMC readings L2_LINES_IN, DCU_LINES_IN and IFU_IFETCH_MISS. These readings need to be sampled at three different sample intervals because of the one PMC set-at-a-time approach. The calculation may use the state variables that relate to the three PMC readings to calculate the Level 2 Cache (L2) Cache Hit Rate base on the current estimate of all three PMCs. Fig. 6.17 presents the four derived measurements Eq.(6.26), Eq.(6.27), Eq.(6.28) and Eq.(6.29). The calculation has computed these performance measurements based on state vector information. Again this is a nice example of the utility of the estimation algorithm.

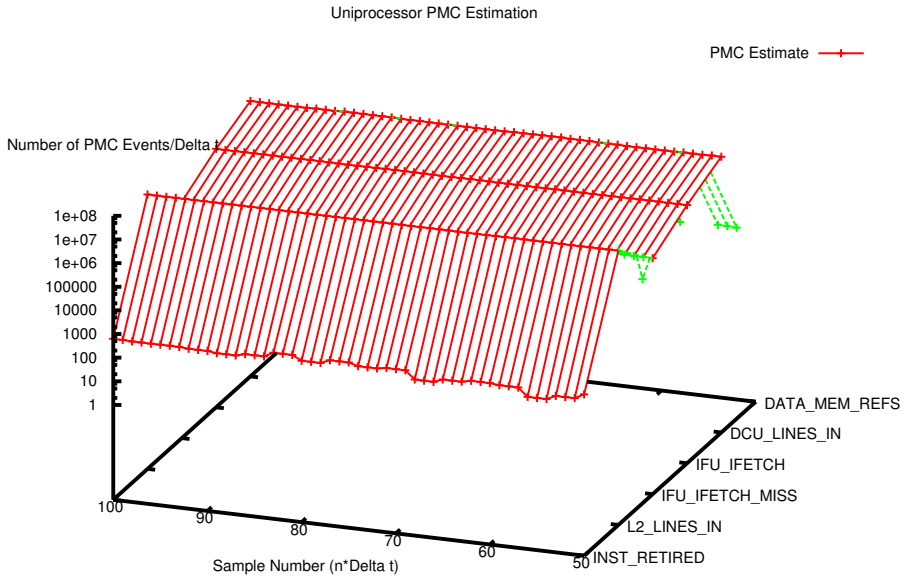

Figure 6.16: Uniprocessor PMC Estimation

$$\text{L1 IFC Hit Rate} = \mathbf{1} - \frac{\text{L1 Instruction Misses}}{\text{Loads}} = \mathbf{1} - \frac{\text{IFU\_IFETCH\_MISS}}{\text{IFU\_IFETCH}} \tag{6.26}$$

$$\text{L2 Cache Hit Rate} = \mathbf{1} - \frac{\text{L2 Misses}}{\text{L1 Misses}} = \mathbf{1} - \frac{\text{L2\_LINES\_IN}}{\text{DCU\_LINES\_IN} + \text{IFU\_IFETCH\_MISS}} \tag{6.27}$$

$$\begin{aligned} \text{L1 to L2 Bandwidth MB/s} &= \frac{\text{L1 Misses * L1 Line size bytes * Clock MHz}}{\text{Cyles}} \\ &= \frac{(\text{DCU\_LINES\_IN} + \text{IFU\_IFETCH\_MISS}) * 32 * \text{Clock}}{\text{TSC}} \end{aligned} \tag{6.28}$$

$$\begin{aligned} \text{L2 to MEM Bandwidth MB/s} &= \frac{\text{L2 Misses * L2 Line size bytes * Clock MHz}}{\text{Cyles}} \\ &= \frac{(\text{L2\_LINES\_IN} * 32 * \text{Clock}}{\text{TSC}} \end{aligned} \tag{6.29}$$

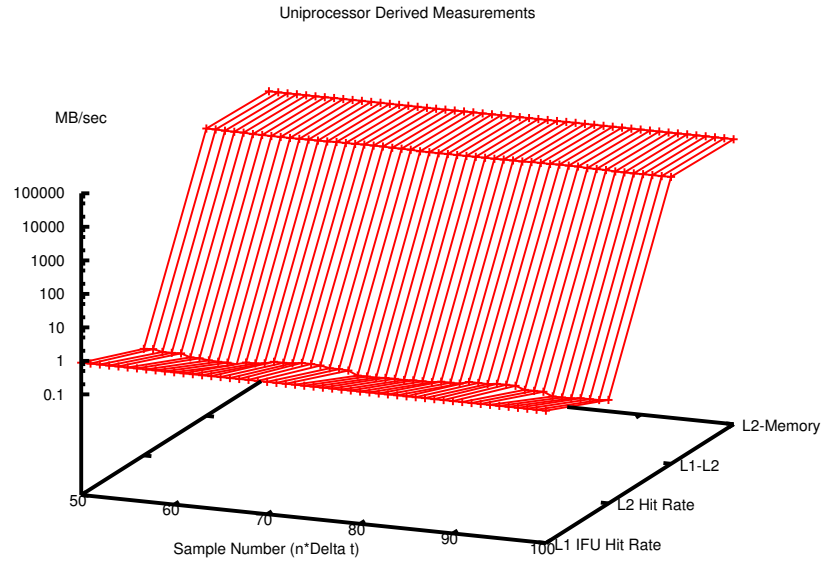

Figure 6.17: Derived Measurements for a Uniprocessor

6.4 Memory Hierarchy Performance Measurements Evaluation for SMP Systems

Similar measurements can be performed on a SMP system. The same algorithm may be applied to sample and estimate PMC readings from all the CPUs in the system. Table 6.3 provides a list of Modified Exclusive Shared Invalid (MESI) related PMC events. This table is a subset of table B.1. These events are of particular interest in the context of SMP machines because MESI or alternative protocols are used to maintain cache coherency throughout the system. The filter's state-vector holds the state variables for all the CPU's PMCs.

#	Symbol	CPU	σ_i	β_i	Mean
11	L2_IFETCH_MESI	1	9.37e+04	38.5	1.09e+05
		2	8.70e+04	35.3	9.85e+04
12	L2_IFETCH_M_STATE	1	No Data		
		2			
13	L2_IFETCH_E_STATE	1	1.73	26.6	0.2438
		2	1.24	43.9	0.2646
14	L2_IFETCH_S_STATE	1	8.49e+04	33.5	9.31e+04
$\Delta t = 0.04sec$ for every PMC reading.					continued on next page

$\Delta t = 0.04sec$ for every PMC reading.					Continued from previous page
#	**Symbol**	**CPU**	σ_i	β_i	**Mean**
		2	9.35e+04	37.8	1.06e+05
15	L2_IFETCH_I_STATE	1	No Data		
		2	2524	112.6	3372
16	L2_LD_MESI	1	4.09e+04	41.6	4.81e+04
		2	4.13e+04	41.7	4.89e+04
17	L2_LD_M_STATE	1	1.50e+04	38.3	1.71e+04
		2	1.50e+04	38.3	1.71e+04
18	L2_LD_E_STATE	1	1.679e+04	31.7	1.779e+04
		2	1.785e+04	35.5	1.983e+04
19	L2_LD_S_STATE	1	7232.0	21.2	5382.0
		2	6393.0	28.1	5471.0
20	L2_LD_I_STATE	1	8063	102.4	1.04e+04
		2	8159	103.6	1.04e+04
21	L2_ST_MESI	1	1.36e+04	88.3	1.76e+04
		2	1.38e+04	88.2	1.79e+04
22	L2_ST_M_STATE	1	6934.0	32.3	7447.0
		1	7637.0	39.4	8838.0
23	L2_ST_E_STATE	1	201.5	57.5	143.4
		2	203.4	57.9	145.4
24	L2_ST_S_STATE	1	1518.0	66.6	2049.0
		2	1442.0	64.5	1934.0
25	L2_ST_I_STATE	1	7734.0	70.2	8568.0
		2	9144.0	77.4	1.00e+04
28	L2_M_LINES_INM	1	1.08e+04	105.7	1.37e+04
		2	1.12e+04	107.3	1.38e+04
29	L2_M_LINES_OUTM	1	1.08e+04	102.6	1.38e+04
		2	1.06e+04	101.7	1.37e+04
30	L2_RQSTS_MESI	1	1.42e+05	40.2	1.67e+05
		2	1.46e+05	41.0	1.72e+05
31	L2_RQSTS_M_STATE	1	2.22e+04	36.9	2.53e+04
		2	2.23e+04	34.7	2.51e+04
32	L2_RQSTS_E_STATE	1	1.76e+04	29.1	1.83e+04
		2	1.81e+04	33.0	1.95e+04
33	L2_RQSTS_S_STATE	1	9.60e+04	35.1	1.08e+05
$\Delta t = 0.04sec$ for every PMC reading.					continued on next page

$\Delta t = 0.04sec$ for every PMC reading.					Continued from previous page
#	**Symbol**	**CPU**	σ_i	β_i	**Mean**
		2	9.56e+04	35.9	1.07e+05
34	L2_RQSTS_I_STATE	1	1.88e+04	100.0	2.32e+04
		2	1.88e+04	100.5	2.33e+04

Table 6.3: Examples of MESI related PMC events specified in table 7.1.

6.5 Distributed State Estimation

Conceivably the single node state estimation algorithm could be extended to multiple nodes, i.e. to a cluster. All the available PMC and TSC registers in a cluster could be sampled and processed at discrete time steps Δt. Accurate sample intervals would obviously be essential for the correctness of the algorithm. The single node algorithm would be executed on every node to fuse PMC event readings into an state vector that provides state variables for all the monitored PMC events in the system. Every node would compute a cluster-wide state estimate. The sampling would require synchronisation. Consequently, all sampled PMC readings must be made available to all the nodes, and must represent the same sample interval. The synchronisation and PMC data communication must be implemented with hard-real-time capable interconnect technology. A high speed interconnect with hardware DSM support, such as SCI, ideally fits these requirements. Before this hypotheses can be tested, it is necessary to assemble and conceptually design experimental hardware DSM platforms. This is discussed in the next chapter before returning to the subject of distributed state estimation in chapter 8.

Chapter 7

Hardware DSM Testbeds

Ideally experimental measurements of the distributed state estimation algorithm, discussed in section 8.1, should be conducted on both a loosely and tightly coupled hardware DSM cluster. This chapter describes an example of each. Unfortunately the implementation of my conceptual design of the tightly coupled special purpose graphics cluster is incomplete, and is also the subject of another PhD, for which I am the supervisor. Therefore the experimental measurements, which are described in section 5.2 on page 71, section 6.2 on page 108, section 6.3 on page 116, section 6.4 on page 120, section 8.2 on page 136, and appendix B *PMC Offline Autocorrelation Analysis* on page 157 are only conducted on the loosely coupled cluster. Equivalent experiments will take place on the graphics cluster once it is completed. It is also intended to use the distributed state estimation algorithm to load-balance the graphics cluster. Note that the terms "loosely coupled" and "tightly coupled" are relative, since many people would consider any hardware DSM system to be tightly coupled. Section 7.1 discusses the configuration of the loosely coupled DSM testbed and section 7.2 provides information concerning the conceptual design of the tightly coupled special purpose graphics cluster.

Intel P4 Hyperthreaded					
Model	CPU Hz	Cache Size	Manufacturer	Chipset	Memory
Intel Pentium 4	3 GHz	1024 KB	Dell	Intel	1 GByte
Intel PIII 2 Way SMP					
Model	CPU Hz	Cache Size	Manufacturer	Chipset	Memory
Intel Pentium III (Coppermine)	1 GHz	256 KB	Super Micro	ServerWorks HE-SL	500 MByte
Intel PIII CompactPCI System					
Model	CPU Hz	Cache Size	Manufacturer	Chipset	Memory
Intel Pentium III (Coppermine)	700 MHz	256 KB	Force	ServerWorks LE-III	500 MByte

Table 7.1: Testbed machines for the Kalman filter evaluation. The Intel P4 system is not part of the Testbed cluster but was used to evaluate the filter algorithm.

7.1 Loosely Coupled Distributed Shared Memory Testbed

Figure 7.1: Front view of the Hardware Distributed Shared Memory Cluster

Figure 7.2: Rear view of the Hardware Distributed Shared Memory Cluster

The implementation of a distributed version of the Kalman filter required the construction of a general purpose hardware DSM cluster from commodity components. Pictures of a small testbed cluster are shown in Fig. 7.1 and Fig. 7.2. The system includes a CompactPCI node, two SMP nodes and a file-server. In addition to these compute nodes the system incorporates a Keyboard Video Mouse (KVM) switch and an Ethernet switch. The Ethernet switch is required by the two SMP nodes and the CompactPCI node to access a common Network File System (NFS) on the file-server. The CompactPCI system can be seen at the top of the cluster in Fig. 7.1 and in Fig. 7.2, right under the Ethernet switch. The two SMP systems are below the the CompactPCI node (one chassis is empty). The file-server is mounted underneath the black KVM switch. The two SMP nodes and the CompactPCI system are interconnected through SCI. This interconnect transforms the three systems into a hardware DSM system.

Fig. 7.5 shows the CompactPCI system's CPU board including a PMC-SCI adapter card that is inserted into the CPU board's 64 bit/66 MHz PMC interface. The CompactPCI CPU is a Force PENT/CPCI-735 board [For01] and the two SMP systems are assembled with Super Micro P3TDE6 main boards [Sup01], see Fig. 7.4. These boards also provide

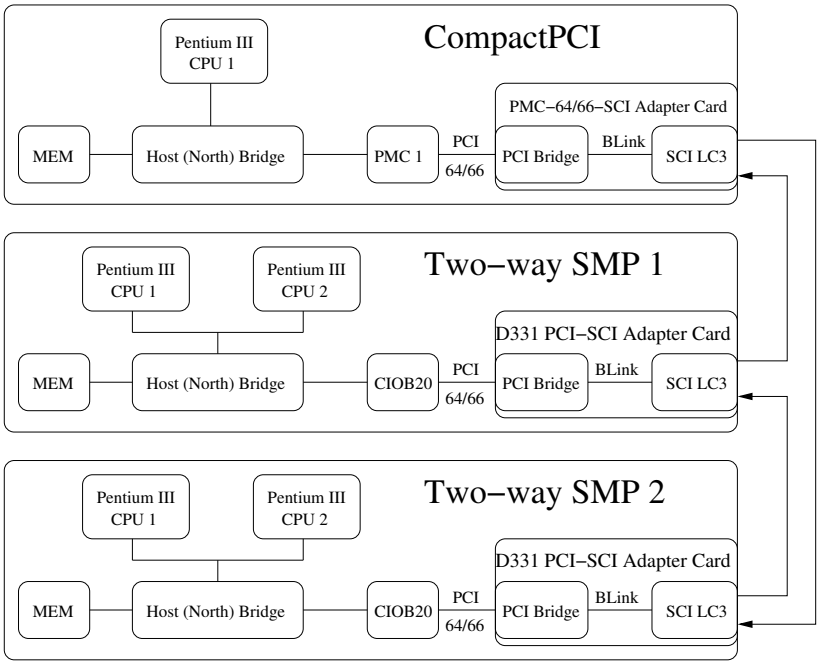

Figure 7.3: Hardware Distributed Shared Memory Testbed

a 64 bit/66 MHz PCI interface. More details on these systems can be found in table 7.1 on page 123. At the time of the design of the cluster in 2001 it was difficult to source main boards with a 64 bit/66 MHz PCI interface. This fast PCI was required by the SCI technology from Dolphin.

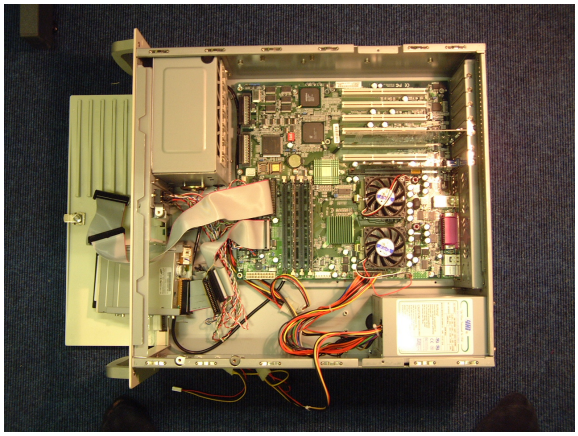

Figure 7.4: One of the Cluster's PIII SMP Nodes

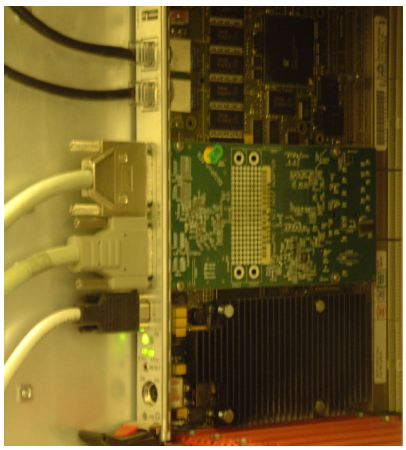

Figure 7.5: CompactPCI System with PMC-SCI Adapter Card

Fig. 7.3 provides a schematic view of the three SCI interconnected systems. The local memories that are attached to the Host (North) bridges become part of the global address space. This may be mapped into an application's virtual address space through a SISCI low-

level API [GAB+99]. The SISCI API can be used to write shared memory applications and it plays a vital role in the implementation of the distributed Kalman filter that is discussed in section 8.1. Fig. 7.6 provides an outline of the Intel PIII micro-architecture. These CPUs are used in all the cluster nodes. An understanding of the processor's micro-architecture will help with the PMC event descriptions in table A.1 on page 147 [Ord01].

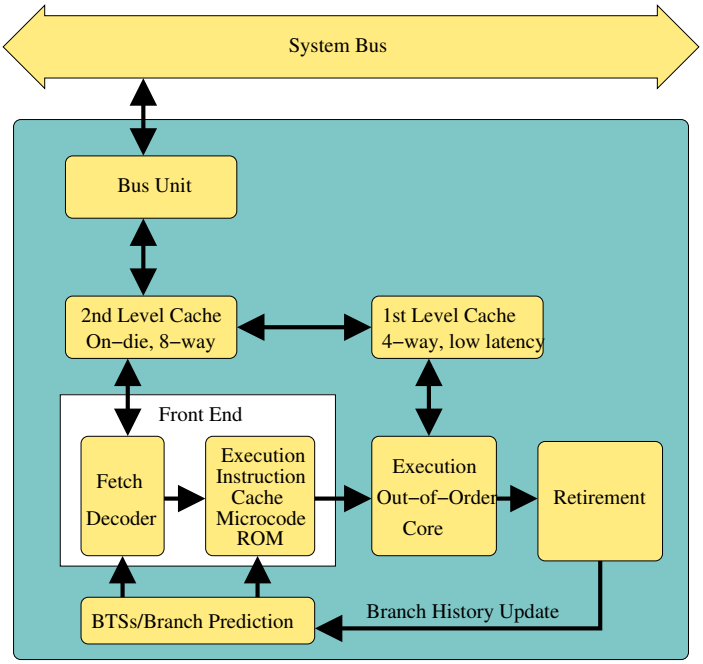

Figure 7.6: P6 Processor Microarchitecture

7.2 Tightly Coupled Special Purpose High Performance Graphics DSM Cluster

As stated before, the conceptional design of this tightly coupled graphics DSM cluster is intended to investigate novel architectural organisations that are suitable for interactive scalable high performance visualisation systems. The detailed design is implemented by my PhD student. When it is complete, performance improvements will be investigated by applying the distributed state estimation algorithm to load-balancing.

Current scalable high-performance graphics systems are either constructed using special purpose graphics acceleration hardware or built as a cluster of commodity components with a software infrastructure that exploits multiple graphics cards [HEB+01] [HHN+02]. Both these solutions are used in application domains where the computational demand cannot be met by a single commodity graphics card, e.g. large-scale scientific visualisation. The former approach tends to provide the highest performance but is expensive because it requires

frequent redesign of the special purpose graphics acceleration hardware in order to maintain a performance advantage over the commodity graphics hardware used in the latter cluster approach. The latter approach, while more affordable and scalable, has intrinsic performance drawbacks due to computationally expensive communication between the individual graphics pipelines. The hybrid solution described here aims to bridge the gap between both of these solutions, offering a minimal custom-built hardware component together with a novel and efficient shared memory infrastructure that exploits cutting-edge consumer graphics hardware.

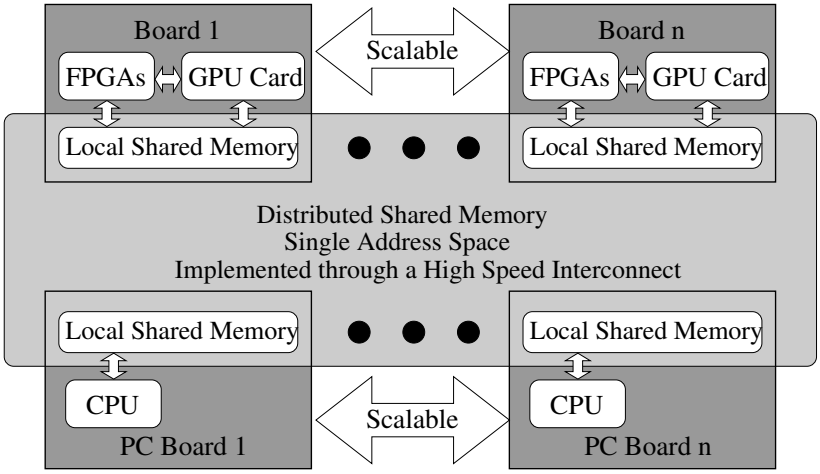

Figure 7.7: Shared Memory.

7.2.1 Cluster Architecture

The scalable and reconfigurable shared-memory graphics cluster is predominantly designed with commodity, off-the-shelf, components with a limited amount of custom-built hardware. The boards allow graphics accelerator cards to interface distributed shared memory that is also shared by a number of PCs in the graphics cluster. Figure 7.7 and 7.8 show the overall design of the architecture. One of the main design objectives is to keep the custom-built hardware part of the system as small and simple as possible while still enabling high performance computations. Adaptability to the latest generation of desktop graphics acceleration hardware was a further important issue, as the highly competitive nature of this market ensures that the performance of these cards increases dramatically with each new version. Also an ability to change parts of the hardware design after the PCB for the GPU/FPGA nodes are manufactured and populated with Integrated Circuits (IC) is very desirable in order to conduct research into different implementation alternatives. Reconfigurable hardware in the form of FPGAs is an ideal solution to meet all these design objectives, with the added advantage of providing substantial additional compute resources for the application stages of the parallel graphics pipelines. These additional resources can be used to implement al-

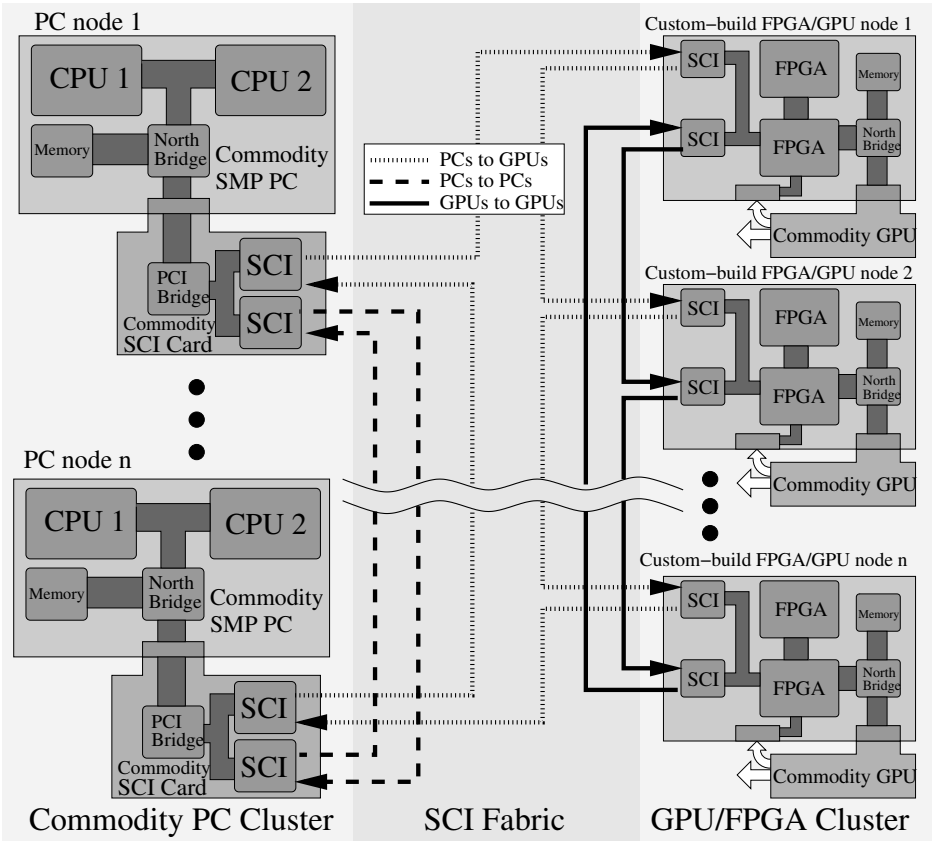

Figure 7.8: Hybrid Parallel Graphic Cluster.

gorithms usually executed on the CPU or GPU of a desktop machine. These algorithms may be defined at compile time by the application developer and loaded into reconfigurable hardware just as a traditional graphics application is loaded into the main memory.

Parallel Rendering and Sorting

The hybrid scalable DSM system of PCs, GPUs and Reconfigurable Hardware is a parallel rendering architecture and as such can be classified according to Molnar et al's taxonomy [MCEF94]. This classification defines parallel rendering as a sorting problem and divides the graphics pipelines into two main pipeline stages: geometry and rasterisation. The geometry processing stage is concerned with transformation, clipping and lighting and is parallelised by distributing subsets of the primitives in the scene over the available concurrent geometry stages. The rasterisation deals with scan-conversion, shading and visibility determination and may take advantage of parallel rasterisation stages by assigning each stage a share of the pixel calculations. The rasterisation is followed by image composition. In order to increase the utilisation of the different parallel pipeline stages, a sorting or redistribution of data between the main stages may be performed. Molnar et al's taxonomy specifies redistribution during the geometry processing as "sort-first", sorting between the geome-

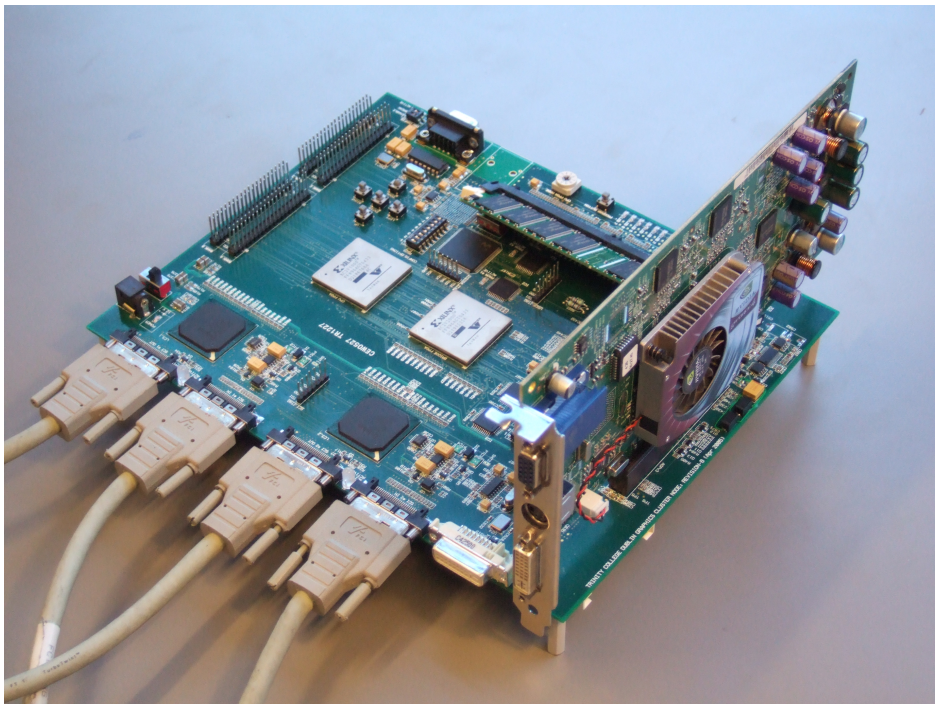

Figure 7.9: The first prototype of the custom-built high-performance graphics cluster node. The figure shows how a commodity graphics card interfaces the cluster node. It also depicts the four SCI cables that should interconnect the custom-built GPU interface boards and the PC cluster via a 2D torus topology. Detailed design by Ross Brennan.

try processing and rasterisation as "sort-middle" and a distribution during rasterisation as "sort-last".

Some compute clusters utilise a software infrastructure such as Stanford's Chromium [HHN+02] to convert a cluster of commodity, off-the-shelf desktop machines with graphics acceleration cards into a parallel rendering system. Such systems can implement sort-first and sort-last alternatives but sort-last mechanisms can only be achieved through expensive read-backs of colour and depth buffers since this is the only form of access to data in the graphics pipeline of a commodity GPU card. The sorting of data before it enters the different stages of the parallel graphics pipelines allows load-balancing and therefore increases the utilisation of the overall system. An efficient solution for load-balancing, a sort-first algorithm, is provided through pre-transformation in order to determine the most suitable data distribution over the GPUs [HEB+01]. These pre-transformations are part of the geometry processing stage and calculate the screen space position of primitives in order to allocate them to screen regions that are served by a particular GPU. This computation is one of the overheads that must be carried by a scalable parallel rendering system in order to most efficiently exploit parallelism.

Sorting Acceleration

Clusters that provide parallel rendering facilities through a software infrastructure must perform these pre-transformations with the assistance of the system's CPUs as part of the graphics pipeline's application stages. In contrast, the architecture described here provides the application stages with reconfigurable hardware resources on every GPU/FPGA node. These FPGAs interface to the distributed shared memory in the same way as the GPUs and CPUs are connected to the shared address space. This allows individual FPGAs to communicate with each other at extremely low latencies and high bandwidth (>500Mbytes/s). This additional infrastructure has the potential to implement complex sort-first load-balancing mechanisms without any additional computational overhead for the CPUs and at superior performance. Furthermore, a sort-last implementation on the tightly coupled cluster of GPU/FPGA nodes will also be feasible. It is intended that these operations will be implemented with limited or no CPU involvement.

Additional Features

In addition, the architecture provides features heretofore only available with high performance, special purpose graphics acceleration hardware such as those in the Pomegranate hardware [EIH00], which provides low-latency, high-bandwidth shared texture memory. The commodity graphics cards have access to textures through the AGP aperture that is implemented in the DSM and accessible to all GPUs and PCs in the cluster. Furthermore, copying the GPUs' frame buffer contents via the Digital Visual Interface (DVI) ports into the DSM can be used to provide a shared distributed frame buffer. Alternatively, a copy can be generated with frame buffer read-backs through the AGP interface. In cases where a single display is driven by the hybrid graphics cluster, the final tile reassembly can be implemented by composing the frame buffer copies from the DSM. Tile reassembly is in this configuration required because every graphics pipelines' frame buffer holds a particular screen region or tile. This aproach provide functions similar to those provided by Stanford's Lightning-2 system [SEP+01] that reassembles multiple graphics card DVI outputs' from commodity rendering clusters in order to generate one or more outputs.

7.2.2 Interconnect Technology

The interconnect, in conjunction with the reconfigurable hardware, fulfils a vital function in the design. The distributed FPGAs allow the pipeline's application stages to migrate subsets of their computations from the CPUs onto the FPGAs, while an SCI interconnect implements the DSM in hardware with some additional logic in the FPGAs. The SCI standard defines a high bandwidth and low latency interconnect and is scalable to a large number of nodes while providing bus-like services and flexible fabric configuration, i.e., nodes may be interconnected using a variety of configurations such as two and three dimensional

torii or as rings. Furthermore, crossbar switches, which allow various switch ports to be directly connected, are also a possible solution and all of these configuration options are available as commodity hardware. The SCI fabric flexibility guarantees the scalability of the parallel rendering architecture.

SCI and Real-Time Constraints

Concurrent graphics operations require synchronisations that must meet real time constraints. Furthermore, interactive immersive scientific visualisation frequently involves complex user input such as the tracking of the users' motions. This motion-tracking data must be processed rapidly to influence the animation in real time, as large latencies between motion detection and animation can cause simulator sickness [PCC92]. SCI technology is already being used in mission critical real-time applications. For example, Thales Airborne System employs SCI for backplane communication in their EMTI unit (Data Processing Modular Equipment). This scalable unit is integrated into the Mirage F1, 2000 and Rafale combat aircraft, NH-90 helicopters, Leclerc tanks, sub-marines, the Charles de Gaulle aircraft carrier and strategic missiles [New00]. The author, with colleagues, previously conducted research on SCI fabric's suitability for real-time application [MC99a, MKCL01].

7.2.3 Commodity and Custom-built GPU/FPGA Cluster Nodes

Figure 7.8 shows how a PC cluster is interconnected with commodity SCI cards. The SCI fabric then connects the Commodity PC cluster to the custom-built GPU/FPGA cluster nodes.

Cluster's Commodity Desktop Machines

This part of the scalable parallel rendering cluster is completely assembled from commodity components and provides the services of a non-cache-coherent hardware DSM. Increasing the number of nodes and possibly changing the SCI fabric topology allows scalability to be achieved. These nodes execute that part of the graphics pipeline's application stages that are not migrated onto the FPGAs on the GPU/FPGA nodes shown in Figure 7.10.

Cluster's Custom-built GPU/FPGA Boards

The GPU/FPGA cluster nodes are custom-built boards that provide an AGP slot for commodity graphics accelerator cards (See Figure 7.10). Two SCI LCs interface to a Xilinx XC2V2000-5FF896C FPGA (Bridge-FPGA) via the 64 bit Blink-bus at 800 Mbytes/s. These LCs can be purchased from the manufacturer of the SCI cards [Dol05] as Application Specific Integrated Circuits (ASIC)s and solve many of the SCI link-level implementation challenges. The Bridge-FPGA is connected to a second Xilinx XC2V1000-4FF896C FPGA (Control-FPGA) via an Advanced Microcontroller Bus Architecture (AMBA) bus at 400

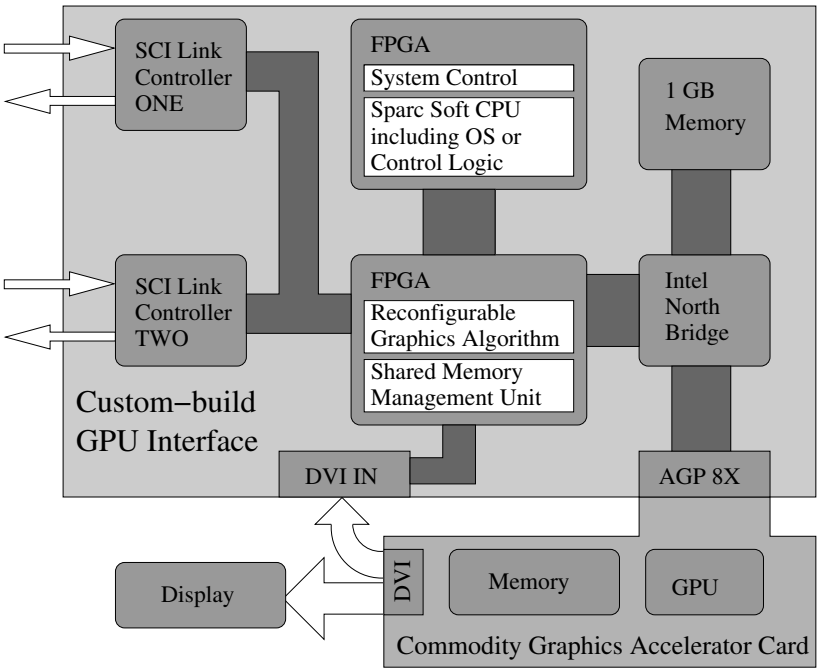

Figure 7.10: GPU Cluster Node with commodity graphics card in AGP slot

Mbytes/s [Lim99], and to an Intel i865G chip (Northbridge). The Front-Side Bus (FSB) between the Bridge-FPGA and the Northbridge operates at 3.2 Gbytes/s. The Northbridge then provides a 2.7 Gbyte/s interface to 1 Gbyte of Double Data Rate (DDR) memory and a 2.1 Gbytes/s link to the AGP slot. The DDR memory forms part of the global address space. It is the Bridge-FPGA's task to translate global address space memory references into SCI transactions and vice versa. Figure 7.11 shows the layout of the printed circuit board (Detailed design by Ross Brennan). This PCB has 10 layers and 5908 pins and vias in order to mount 1277 components. Ten prototype PCBs have been manufactured and two have been assembeled at a cost of EUR3000 per board. The boards are currently been debugged and outstanding problems are being corrected for the next revision. It can be expected that this price will be significantly lower if manufactured in larger quantities.

Some of the services provided by a commodity SCI card must also be implemented in the Bridge-FPGA in order to allow the PCs to function with the GPU/FPGA cluster nodes. The shared memory management is one of the main functions of this Bridge-FPGA. The second and equally important objective is to execute application-specific graphics algorithms as outlined previously. In this design the memory and the graphics subsystem are directly connected via the Northbridge and the Bridge-FPGA is connected to the SCI Blink bus. This approach avoids bandwidth restrictions and additional latencies that are introduced in commodity SCI cards by the I/O bus.

The direct SCI connection of the GPU/FPGA nodes classifies these nodes as tightly coupled, relative to the less tightly coupled commodity SCI cards. Nevertheless, the commod-

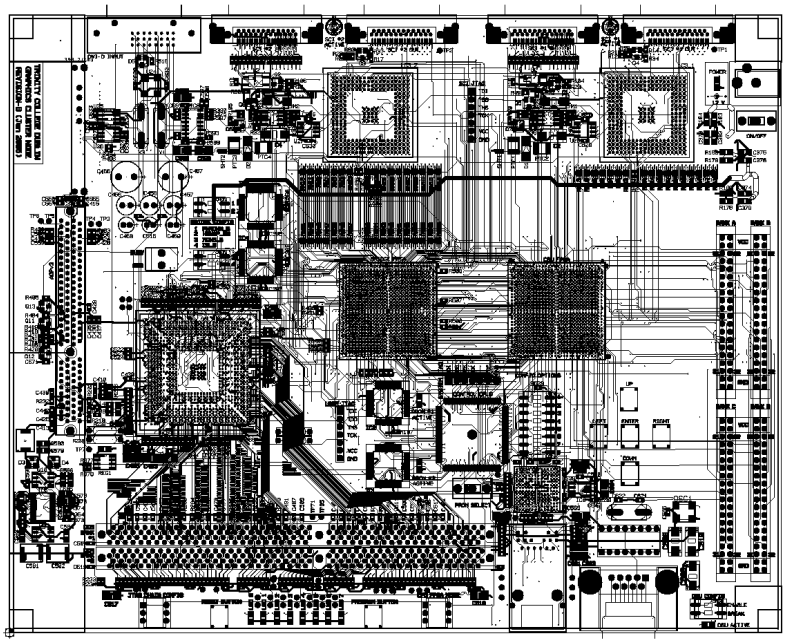

Figure 7.11: PCB.

ity SCI cards achieve approximately 300 Mbytes/s with 1.46 μs application-to-application latency. The maximum bandwidth of the commodity SCI cards is dependent on the motherboard's chipset [Dol04]. Measurements on two-node and four-node fabrics have demonstrated that FPGAs directly connected to the Blink may exchange data at more than 500 Mbytes/s over the SCI fabric [NT01]. Latency measurements for this configuration are not available but it is reasonable to assume that transactions that do not have to pass through the PCI bus will exhibit a low latency ($\ll$ 1.5 μs).

Control of the Cluster's Custom-built GPU/FPGA Boards

The second "Control-FPGA" on the GPU/FPGA cluster nodes implements the control of the board through logic or holds a LEON soft CPU [AGE05]. This SPARC compatible processor may be clocked at 100MHz. The LEON core was developed by the European Space Agency (ESA) for space missions and is available as open source Hardware Description Languages (HDL). The soft CPU is intended to execute RTEMS, an open source hard real-time OS for embedded systems [RTE05]. It is not intended that the control logic or the soft CPU and its real-time OS to participate in computation of the graphics rendering pipeline. It should merely function as a control processor that provides services that are required for the operation of the parallel rendering cluster.

Chapter 8

Compute Cluster State Estimation Algorithm (C^2STATE)

Section 2.1 “Trace Data Acquisition and Analysis” on page 30, section 6.2 “Optimisation Analysis” on page 108, section 6.3 “Uniprocessor Systems Evaluation” on page 116 and section 6.4 “SMP Systems Evaluation” on page 120 provide extensive experimental results and a discussion of the results for single-node systems. Furthermore some of the experimental results are provided an appendix B “PMC Offline Autocorrelation Analysis” on page 157. This chapter extends the filter to a hardware DSM cluster implementation. I have named the resulting global DSM state estimation algorithm the “Compute Cluster State Estimation Algorithm (C^2STATE)”.

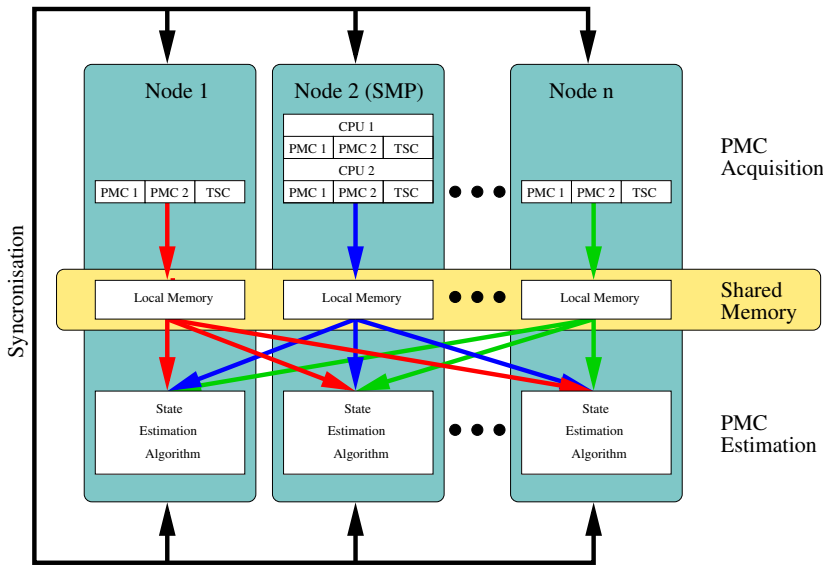

Figure 8.1: Compute Cluster State Estimation Algorithm (C^2STATE)

8.1 The C^2STATE Algorithm

As with the single node state estimation implementations, whether dealing with a uniprocessor or a SMP system, all the available PMC and TSC registers in a cluster can be sampled and processed at discrete time steps $\boldsymbol{\Delta t}$. Accurate sample intervals are esential for the correctness of the estimation algorithm.

In a hardware DSM implementation, a single node algorithm, as introduced in section 5.1, can be executed on every node to fuse PMC readings from all the nodes into its state vector $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$. Therefore every node computes a state estimate of the PMCs registers from all the systems that participate in this algorithm. The sampling of PMC readings at discrete time steps $\boldsymbol{\Delta t}$ and subsequent processing in all the systems requires a synchronisation. Furthermore the sampled PMC readings must be made available to the other nodes and must represent the same time interval. The synchronisation and PMC data communication can be implemented through a hardware DSM system, if available. In this case, the testbed described in section 7.1 is able, via the SISCI low-level API [GAB+99], to read and write remote and local memory locations that hold PMC readings and is also able to synchronise the state estimation algorithms through remote SCI interrupts.

Fig. 8.1 shows how this compute cluster state estimation algorithm (C^2STATE) is distributed over $\boldsymbol{n}$ nodes. It depicts how local PMC readings are written into shared memory and read by the single node algorithms in all the nodes. It also indicates the synchronisation of all the PMC acquisitions and subsequent processing in the Kalman filters.

Once the number of monitored PMC events exceeds the number of available PMC registers it becomes necessary to employ the *one performance monitoring counter (PMC) set-at-a-time* solution that was discussed in section 5.3. As a consequence the measurement vector $\boldsymbol{z}_{\boldsymbol{k}}$ increases in size. Table 8.1 provides an example that uses the testbed configuration described in section 7.1. For the first sample in this table, the algorithm selects two PMC events (DATA_MEM_REFS and DCU_LINES_IN) in all of the CPUs. Since every local estimation algorithm fuses all of the PMC registers in all nodes the measurement vector $\boldsymbol{z}_{\boldsymbol{k}}$ must allocate elements for these readings. For the second sample, the PMC acquisition selects the next two PMC events (DATA_MEM_REFS and DCU_LINES_IN) and processes the readings accordingly. Again the measurement vector $\boldsymbol{z}_{\boldsymbol{k}}$ must be adjusted. The number of elements in the measurement vector $\boldsymbol{z}_{\boldsymbol{k}}$ also determines the size of the vectors and matrices in the Kalman filters (see table 5.4 on page 91).

This procedure continues until the last two PMC events (FP_COMP_OPS_EXE and FP_ASSIST) have been selected. During the next discrete time step $\boldsymbol{\Delta t}$ the algorithm again selects the two PMC events (DATA_MEM_REFS and DCU_LINES_IN) from the first sample in order to fuse the readings in the state vectors $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$. This round-robin scheduling of PMC events and their fusion into the state vectors $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$ continues until the algorithm is stopped. A coarse outline of the C^2STATE algorithm is as follows:

1. Start the C²STATE algorithm on all nodes
2. Initialise the SISCI library and map remote memories from all the nodes into the application virtual memory.
3. Setup SCI remote interrupts for synchronisation.
4. Allocate memory for vectors and matrices listed in:
   - Table 5.2 on page 88.
   - Table 5.4 on page 91.
5. Load parameters for the selected PMC events:
   - For the integrated Gauss-Markov process models $\boldsymbol{\beta}$, $\boldsymbol{\sigma^2}$ and $\boldsymbol{t}$
   - For the elements of the measurement noise covariance matrix $\boldsymbol{R_k}$
6. Initialise the state transition matrix $\boldsymbol{\phi_k}$ Eq.(5.18) on page 70 with parameters for the selected PMC events from step 4.
7. Initialise the process noise covariance matrix $\boldsymbol{Q_k}$ Eq.(5.19) on page 70 with parameters for the selected PMC events from step 4.
8. Select two PMC events on every CPU from the range of selected PMC events.
9. Sample the PMC and TSC register at sample time $\boldsymbol{t}$.
10. All nodes copy PMC and TSC readings into shared memory.
11. All nodes read data into the correct location in the measurement vector $\boldsymbol{z_k}$
12. Execute Kalman filter algorithm Fig. 5.3 on page 65 with *one performance monitoring counter PMC set at a time* modification that was discussed in section 5.3.
13. Return to step 8 if the algorithm is not stopped.
14. Stop the execution.

8.2 Memory Hierarchy Performance Measurements on Shared Memory Clusters

The proposition that a valid global hardware DSM state estimation is possible (the hypothesis) can be tested in the same fashion as was done in the single-node case. This section provides some acquisition and estimation examples using the C²STATE algorithm. The

Z_k	PMC	CPU	Node	Sample	PMC Event	PMC #
z_{1k}	PMC 1	CPU 1	SMP1	**1**	DATA_MEM_REFS	1
z_{2k}	PMC 2				DCU_LINES_IN	2
z_{3k}	PMC 1	CPU 2			DATA_MEM_REFS	1
z_{4k}	PMC 2				DCU_LINES_IN	2
z_{5k}	PMC 1	CPU 1	SMP2		DATA_MEM_REFS	1
z_{6k}	PMC 2				DCU_LINES_IN	2
z_{7k}	PMC 1	CPU 2			DATA_MEM_REFS	1
z_{8k}	PMC 2				DCU_LINES_IN	2
z_{9k}	PMC 1	CPU 1	CPCI1		DATA_MEM_REFS	1
z_{10k}	PMC 2				DCU_LINES_IN	2
$z_{11(k+1)}$	PMC 1	CPU 1	SMP1	**2**	L2_DBUS_BUSY	36
$z_{12(k+1)}$	PMC 2				L2_DBUS_BUSY_RD	37
$z_{13(k+1)}$	PMC 1	CPU 2			L2_DBUS_BUSY	36
$z_{14(k+1)}$	PMC 2				L2_DBUS_BUSY_RD	37
$z_{15(k+1)}$	PMC 1	CPU 1	SMP2		L2_DBUS_BUSY	36
$z_{16(k+1)}$	PMC 2				L2_DBUS_BUSY_RD	37
$z_{17(k+1)}$	PMC 1	CPU 2			L2_DBUS_BUSY	36
$z_{18(k+1)}$	PMC 2				L2_DBUS_BUSY_RD	37
$z_{19(k+1)}$	PMC 1	CPU 1	CPCI1		L2_DBUS_BUSY	36
$z_{20(k+1)}$	PMC 2				L2_DBUS_BUSY_RD	37
⋮	⋮	⋮	⋮	**3** to **m-1**	⋮	⋮
$z_{(n-9)(k+m)}$	PMC 1	CPU 1	SMP1	**m**	FP_COMP_OPS_EXE	73
$z_{(n-8)(k+m)}$	PMC 2				FP_ASSIST	74
$z_{(n-7)(k+m)}$	PMC 1	CPU 2			FP_COMP_OPS_EXE	73
$z_{(n-6)(k+m)}$	PMC 2				FP_ASSIST	74
$z_{(n-5)(k+m)}$	PMC 1	CPU 1	SMP2		FP_COMP_OPS_EXE	73
$z_{(n-4)(k+m)}$	PMC 2				FP_ASSIST	74
$z_{(n-3)(k+m)}$	PMC 1	CPU 2			FP_COMP_OPS_EXE	73
$z_{(n-2)(k+m)}$	PMC 2				FP_ASSIST	74
$z_{(n-1)(k+m)}$	PMC 1	CPU 1	CPCI1		FP_COMP_OPS_EXE	73
$z_{n(k+m)}$	PMC 2				FP_ASSIST	74

Table 8.1: The Kalman Filter's Measurement Vector Structure for the C²STATE Algorithm. Please see table A.1 for more details on the selected PMC events.

algorithm was implemented on the testbed cluster described in section 7.1 on page 124. Furthermore the filter was configured with same list of PMC events as used in section 6.3 (please see table 6.2 for details). Consequently every CPU in the cluster must sample the six specified PMC events. The layout of the measurement vector $\boldsymbol{z_k}$ can be determined from table 8.1 on page 137 in conjunction with the list of selected PMC events. With the 5 CPUs in the system (two SMPs and the CompactPCI uniprocessor) the total number of sampled

PMC events comes to 30. Fig. 8.2 and Fig. 8.3 show the first and the last 15 estimated PMC event counts respectively. It should be pointed out that every node in the system can have a global view of the system's state in real-time at a granularity of 20ms for Linux systems. This is a considerable feat in itself.

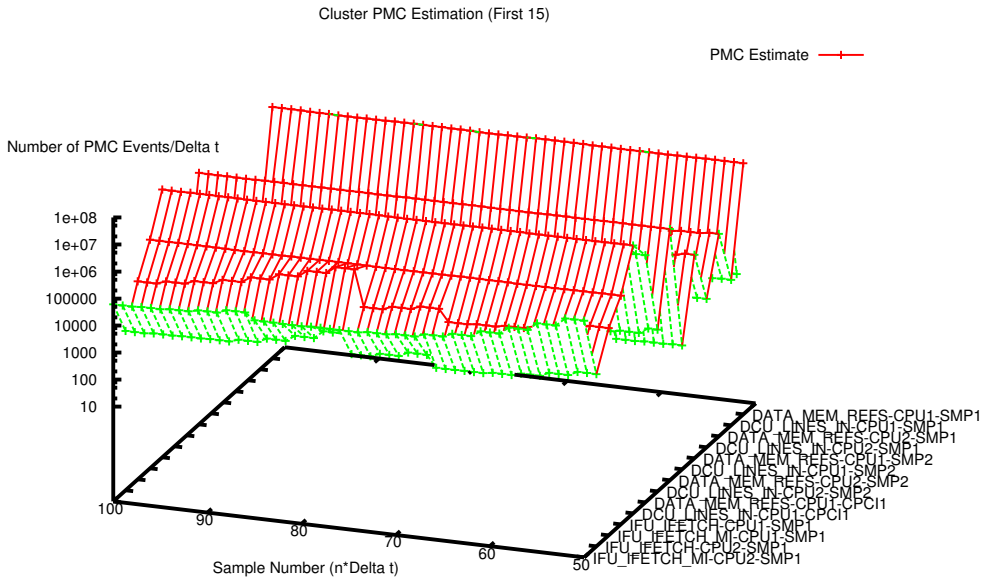

Figure 8.2: Cluster PMC Estimation (First 15)

As in the case of the uniprocessor it is interesting to look at derived performance measurements since these cannot be measured without the data fusion provided by the filter. Again the PMC events specified in table 6.2 are used. Fig. 8.4 shows how the filter applies Eq.(6.26) to calculate the L1 Instructions Fetch Unit (IFU) Hit Rate, and very clearly shows the caching behaviour. Similarly Fig. 8.5 presents the L1 to L2 Bandwidth in MB/s for every CPU in the cluster. The Filter achieves this by using Eq.(6.28) to compute the L1 to L2 Bandwidth for information provided in the state vector.

Fig. 8.4 and Fig. 8.5 demonstrate how the C^2STATE algorithm may be applied to monitor parallel applications at runtime. Fig. 8.6 is another example that shows for every CPU in the cluster their L1-L2 bandwidth minus the average L1-L2 bandwidth. Again the global view of the system's performance allows observation of this information from every node in the system. The global DSM state estimation algorithm works, and has clear potential use for cluster optimisations.

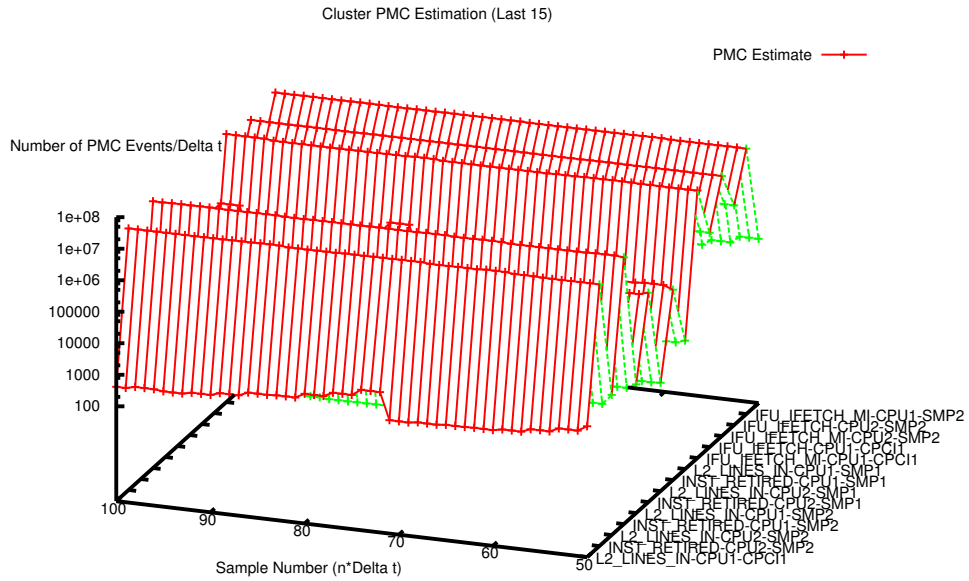

Figure 8.3: Cluster PMC Estimation (Last 15)

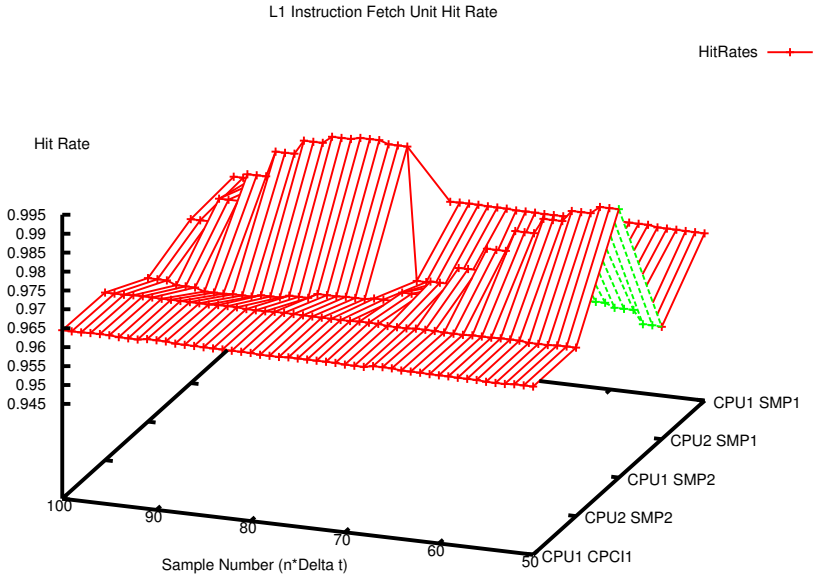

Figure 8.4: Cluster L1 Instruction Fetch Unit Hit Rate

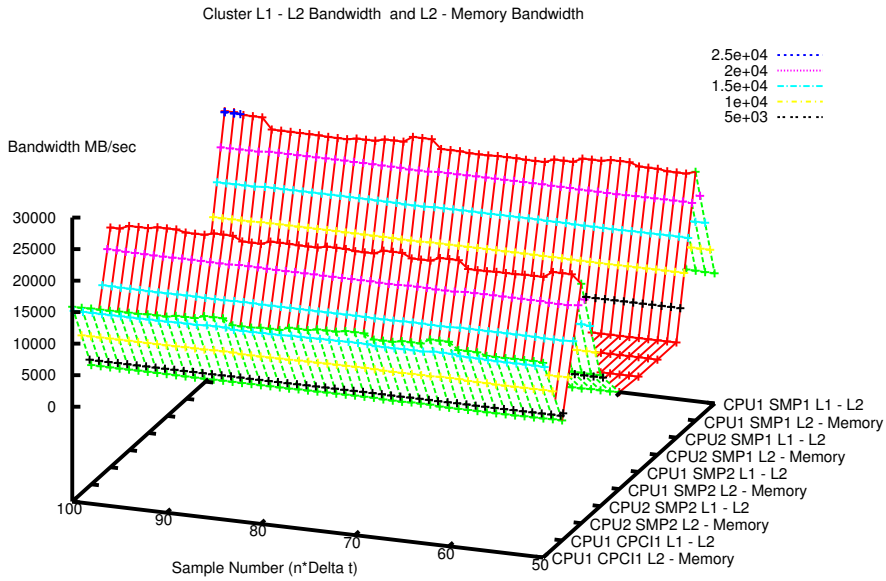

Figure 8.5: Cluster L1 - L2 Bandwidth and L2 - Memory Bandwidth

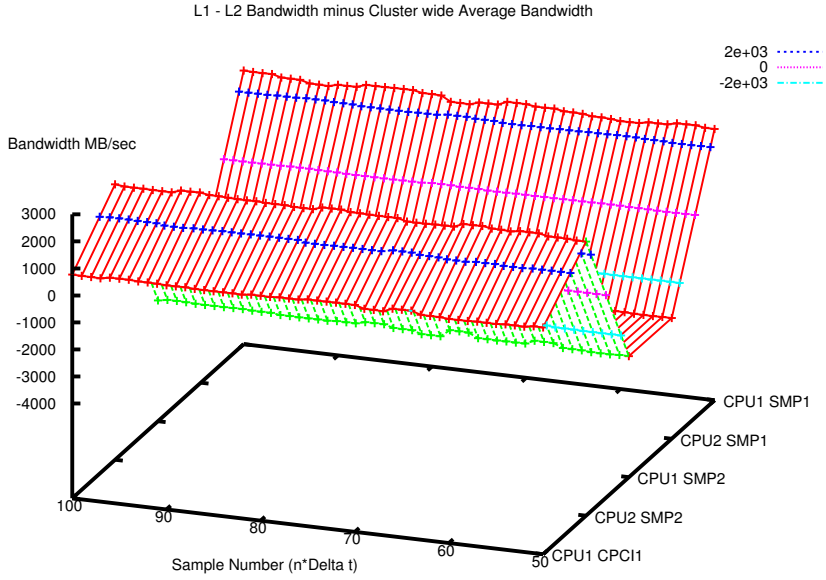

Figure 8.6: L1 - L2 Bandwidth minus Cluster wide Average Bandwidth

8.3 Work Loads

As a final note, all examples presented in this chapter, chapter 5 and chapter 6 were generated while either a BYTEmark [BYT06] benchmark program or Celestia [Cel06] application was executed. According to the Linux "top" utility the benchmark consumed almost 99% of the CPU and the Celestia application consumed around 50% of the CPU depending on the configuration. It is interesting to note that the C^2STATE algorithm consumed only 1% of the CPU and sampled at a very high accuracy. The histogram in Fig. 2.4 on page 37 shows how the sample time changes around the mean of 3.9532e+07 clock cycles. The CPU operates at 1 GHz, consequently the mean $\boldsymbol{\Delta t}$ of 3.9532e+07 clock cycles is equivalent to 39.532 ms. The samples have a standard deviation of $\boldsymbol{\sigma = 1.1074e + 5}$.

Fig. 6.13, Fig. 6.14 and Fig. 6.15 were plotted based on PMC readings that were caused by a Celestia application. The PMC acquisition in chapter 5 took place while the same application generated PMC events.

The BYTEmark Assignment algorithm is associated with Fig. 6.16 and Fig. 6.17 and data for Fig. 8.2, Fig. 8.3, Fig. 8.4, Fig. 8.5 and Fig. 8.6 were generated while the BYTEmark String sort benchmark was executed.

The following is a description from the BYTEmark web page:

> "**Assignment algorithm** - A well-known task allocation algorithm. The test moves through large integer arrays in both row-wise and column-wise fashion. Cache/memory with good sequential performance should see a boost (memory is altered in place – no moving as in a sort operation). Processing is done in 32-bit chunks – no advantage given to 64-bit processors. **String sort** - Sorts an array of strings of arbitrary length. Tests memory-move performance. Should exercise non-sequential performance of cache, with added burden that moves are byte-wide and can occur on odd address boundaries. May tax the performance of cell-based processors that must perform additional shift operations to deal with bytes" [BYT06].

Chapter 9

Conclusions and Future Work

All aspects of this thesis are either concerned with the architectural organisation or the performance analysis of hardware DSM clusters. The target systems utilise implementations of the IEEE SCI interconnect standard to make loosely or tightly coupled NUMA services available to the OSs and the applications that may run on them. It is this SCI technology that can transform a collection of uniprocessor and SMP machines into a system that allows individual CPUs to access remote memories in hardware without OS intervention.

9.1 Performance Analysis

Two aspects of performance analysis were covered in this thesis. Firstly this work looked at the non-invasive acquisition of real-time SCI interconnect traffic. To this end a SCI deep trace acquisition instrument was designed and built. It was also necessary to develop a software infrastructure that enabled the conduct of detailed off-line analyses of SCI traces. At the time of the instrument design and assembly no technology was available to perform deep tracing. This has since changed. The analysis software infrastructure is built around a relational database that accepts decoded and time-stamped SCI packets and allows for a detailed spatial and temporal analysis at SCI packet granularity. SCI packets hold encoded routing and flow-control information in addition to their payload. This enables sophisticated queries into a relational database of these SCI packet fields.

The tuning of a SCI topology simulation directly applied the deep trace instrument and its associated software infrastructure. SCI database queries were executed to filter relevant packets for a subsequent statistical analysis. The packet's routing and flow-control database fields allow for this filter process. The simulation requires statistics about packet producers and consumers, in the form of PDFs, to stimulate interconnect traffic in the simulation. Tuned simulation results for particular SCI interconnect topologies were compared against real systems and demonstrated that the method is able to predict the performance of a proposed topology. It should also be pointed out that the non-invasive and time-stamped

acquisition of interconnect traffic captures the true behaviour of the system. This is something that cannot be achieved with invasive methods such as software instrumentation.

Secondly the implementation of the C^{2}STATE algorithm demonstrates that a discrete minimum mean-square error filter may be successfully applied to fuse concurrent and sequential observations of system event counts into a state vector. This technique allows the observation of an increased variety of PMC events without sacrificing accuracy. The optimal estimate of large sets of performance counter processes provides us with the means to observe complex systems by combining the elements of the state vector to infer system metrics that are unobservable with a restricted number of counter readings.

Fundamental to this approach is the modelling of PMC readings as random processes. This idea was conceived while analysing SCI interconnect traffic in order to infer PDFs for the parameterisation of the SCI interconnect model. Similarly a Kalman filter requires parameters for the random process models. This requires the repeated acquisition of long sample series for subsequent autocorrelation analyses. Every available PMC event must be processed in this fashion in order to derive process model parameters for the Kalman filter. This is a time consuming undertaking and is further discussed in section 9.2 *Limitation and Future Work*.

An important observation is that if the filter has to perform the full matrix and vector operations then it is not feasible to fuse a larger number of PMC readings into the state vector. As the filter quickly consumes a large percentage of the available CPU time. If we further increase the number of PMC events then it will not be able to schedule PMC samples at appropriate sample intervals. Fortunately the assumption that all of the PMC random processes are independent leads to sparsely populated matrices that can be far more efficiently executed. This is also further discussed in section 9.2 *Limitation and Future Work*.

9.1.1 Hardware DSM Testbeds

A loosely coupled SCI based cluster had to be assembled. This general-purpose cluster was entirely constructed with commodity components and was used to implement the C^{2}STATE algorithm.

A tightly coupled SCI cluster was also designed and is currently being implemented. The design of this scalable graphics cluster was motivated by the desire to take full advantage of the hardware DSM functionality. This special-purpose cluster could incorporate features that would provide room for novel research. One design objective was to allow for the integration of the C^{2}STATE algorithm into the cluster to investigate its suitability for load balancing of interactive parallel graphics applications. Unlike scientific high performance computations that ideally require load balancing at a granularity $\ll$ 20ms, parallel hardware-accelerated polygonal renderers could take advantage of a load balancing algorithm that operates at frame rates. On Linux systems it is the OS's ability to schedule tasks that determines the PMC-register sampling rate, currently at 20ms. Consequently a load balancing algorithm

that is assisted by feedback from the C^{2}STATE PMC state estimation algorithm could operate at 50 frames/second.

Very loosely coupled graphics clusters of commodity PCs that exploit multiple GPUs have been investigated. In these systems the graphics processor are distributed over the cluster nodes and the cluster uses a software infrastructure to schedule graphics instructions onto the various GPUs in the system. Typically the interconnect limits the performance of these systems.

The tightly coupled SCI cluster design aims to provide superior performance for interactive graphics applications. The design is novel in that all computational units such as the CPUs and the GPUs interface to the same DSM and also connect the FPGAs on every GPU-node to this DSM. Logic in these reconfigurable units can control the GPU and allows them to communicate directly with each other. The FPGA could also perform graphics related computation such as implementing specific physics operations. The design requires a limited amount of custom-built hardware and provides a tightly coupled scalable NUMA architecture of distributed FPGAs, GPUs and local memory. It is expected that this hardware DSM will communicate data at 500Mbytes/s with low latencies ($\ll 1.5\ \mu$s). This hard real-time capable parallel rendering cluster also interfaces with the same high speed interconnect, to a commodity PC cluster that execute the graphics application. Based on the arguments presented, this solution is expected to outperform pure commodity implementations without increased hardware cost and yet maintain its adaptability to the most recent generation of commodity graphics accelerators and target applications. The next prototype version will incorporate a PCI-Express interface to be compatible with the latest commodity graphics accelerators. The conceptual design is part of this thesis but the detailed design and its implementation is my PhD student's work.

9.2 Limitations and Future Work

There are of course limitations to the research presented in this thesis, and it is appropriate to discuss potential research that could build on the work presented here.

9.2.1 SCI Trace Acquisition and Analysis

The current acquisition of trace-data at a single point in the SCI fabric (Blink or cable) has certain restrictions e.g. it is not possible to analyse latencies between two or more measurement points in the interconnect. Consequently it would be advantageous to extend the software and hardware infrastructure to allow for a synchronised collection of trace-data with two or more instruments that deposit their trace-data in a common trace database for a subsequent analysis.

9.2.2 C²STATE Algorithm

The determination of the process model's parameters through the acquisition of several long PMC sample series and a subsequent autocorrelation analysis is very time consuming and provides only a suboptimal process model because parameters change with the system load. The parameters represent an approximation. An alternative solution would be to investigate Adaptive Kalman Filter (AKF)s that estimate the process model's parameters at run time. A substantial amount of research has been conducted in this area for various application [YWA05, MZ02, TK92, SW87, AW06]. This would be the most important improvement and would allow the C²STATE algorithm to run on various systems without the need to tune the filter.

It would also be desirable to identify dependencies between various PMC events in order to formalise new random process models that reflect these dependencies. These more complex process models could sustain longer unsampled periods of their contributing PMC events if individual PMC events lower the uncertainty in the estimation by providing sample readings to the filter. This would result in less sparse matrix operations, which are computationally more expensive. Therefore one has to investigate the extent to which this would be feasible.

A number of researchers successfully investigated PMC events to predict the run-time CPU and memory power consumption [IM03, Bel00, Mar01, KCK+01, LJ03]. Contreras and Martonosi [CM05b] used a first-order, linear power estimation model to observe CPU and memory power consumption based on PMC readings. The C²STATE algorithm has the potential to estimate a DSM system-wide power consumption of the CPU's micro-architectures and other subsystems observable.

Duesterwald et al. [DCS03] investigated the time-varying behaviour of a number of benchmark programs through the observation of metrics that were derived from PMC event readings. They pointed out that the behaviour across metrics, such as Instructions Per Cycle (IPC) and Level 1 Cache (L1) misses, differs, but that the periodicity in the behaviour is shared across metrics. Duesterwald et al. proposed to exploit this correlation for the prediction of computational phases in order to optimise adaptive micro-architectures or OSs. The C²STATE algorithms could incorporate the correlation between the metrics in its process model in order to more accurately predict the further behaviour (one sample interval ahead) of the system and therefore identify computational phases.

As previously mentioned the estimation algorithm could provide feedback for run-time load balancing of the tightly coupled scalable graphics cluster at frame rate granularity.

9.2.3 Interconnect Measurements

The SCI interconnect cards also provide PMC register. In order to monitor parallel applications on these hardware DSM system it would be interesting to merge interconnect performance information into the state-vector. This must be left for future work.

9.2.4 Tightly Coupled Scalable Graphics Cluster

Beyond the design and implementation of the hardware architecture, the main thrust of this research is to design a software infrastructure that best exploits the performance potential of this scalable tightly-coupled cluster of commodity graphics cards and FPGAs. From the programmer's perspective, the architecture provides programmability of the concurrent graphics rendering pipelines at multiple stages: The commodity PCs and the distributed FPGAs may be programmed. The programming of the reconfigurable hardware could either be achieved through HDL such as (VHSIC) Hardware Description Language (VHDL) or high level languages such SystemC. Depending on the target application the programmability of the different pipeline stages must be exploited with a specific approach. Interactive real time immersive scientific visualisation may require the reconfigurable hardware to implement efficient sort-first and sort-last mechanisms. For interactive graphics applications, it is intended to implement Chromium-like services by providing a full abstraction of the underlying parallel graphics architecture. A sequential or parallel OpenGL application may be executed on one or more of the cluster's commodity PCs and the resulting OpenGL commands may be distributed over the available resources. As much as possible of the Chromium [HEB+01] infrastructure should be migrated onto the distributed FPGAs.

9.2.5 Implementation of the C²STATE Algorithm on the Graphics Cluster

Clearly the C²STATE algorithm can be executed on the loosely coupled hardware DSM cluster. The challenge will be to execute the algorithm in conjunction with the custom-built GPU/FPGA nodes, but this is future work.

9.2.6 Contributions

The following enumerated list encapsulates the contributions of this thesis:

1. Collaboration in the design of a SCI interconnect trace instrument and the full design and implementation of the associated software infrastructure.

2. The application of the SCI interconnect trace instrument to the tuning of SCI interconnect topology simulations.

3. Design, implementation and evaluation on a loosely coupled cluster of a performance state estimation algorithm, C²STATE, that provides a global view of hardware DSM systems.

4. The design of the tightly coupled scalable graphics cluster as an explicit tightly coupled target for the C²STATE algorithm.

Appendix A

Appendix:PIII Performance Monitoring Counters (PMC) Description

The following provides information about the Intel PIII PMCs.

	Description	**Symbol**
Data Cache Unit (DCU)		
1	All loads from any memory Type	DATA_MEM_REFS
2	Total lines allowed in the DCU	DCU_LINES_IN
3	Number of M state lines allowed in the DCU	DCU_M_LINES_IN
4	Number of M state lines evicted from the DCU	DCU_M_LINES_OUT
5	Weighted number of cycles while a DCU miss is outstanding	DCU_MISS_OUTSTANDING
Instruction Fetch Unit (IFU)		
6	Number of instruction fetches, both cacheable and noncacheable	IFU_IFETCH
7	Number of instruction fetch misses	IFU_IFETCH_MISS
8	Number of ITLB misses	ITLB_MISS
9	Number of cycles instruction fetch is stalled, for any reason	IFU_MEM_STALL

continued on next page

continued from previous page

#	Description	Symbol
10	Number of cycles that the Instruction Length Decoder (ILD) is stalled	ILD_STALL
L2 Cache		
11	Number of L2 instruction fetches (MESI)	L2_IFETCH_MESI
12	Number of L2 instruction fetches (M STATE)	L2_IFETCH_M_STATE
13	Number of L2 instruction fetches (E STATE)	L2_IFETCH_E_STATE
14	Number of L2 instruction fetches (S STATE)	L2_IFETCH_S_STATE
15	Number of L2 instruction fetches (I STATE)	L2_IFETCH_I_STATE
16	Number of L2 data loads (MESI)	L2_LD_MESI
17	Number of L2 data loads (M STATE)	L2_LD_M_STATE
18	Number of L2 data loads (E STATE)	L2_LD_E_STATE
19	Number of L2 data loads (S STATE)	L2_LD_S_STATE
20	Number of L2 data loads (I STATE)	L2_LD_I_STATE
21	Number of L2 data stores (MESI)	L2_ST_MESI
22	Number of L2 data stores (M STATE)	L2_ST_M_STATE
23	Number of L2 data stores (E STATE)	L2_ST_E_STATE
24	Number of L2 data stores (S STATE)	L2_ST_S_STATE
25	Number of L2 data stores (I STATE)	L2_ST_I_STATE
26	Number of lines allocated in the L2	L2_LINES_IN

continued on next page

continued from previous page

#	Description	Symbol
27	Number of lines removed from the L2 for any reason	L2_LINES_OUT
28	Number of modified lines allocated in the L2	L2_M_LINES_INM
29	Number of modified lines removed from the L2 for any reason	L2_M_LINES_OUTM
30	Total number of L2 requests (MESI)	L2_RQSTS_MESI
31	Total number of L2 requests (M STATE)	L2_RQSTS_M_STATE
32	Total number of L2 requests (E STATE)	L2_RQSTS_E_STATE
33	Total number of L2 requests (S STATE)	L2_RQSTS_S_STATE
34	Total number of L2 requests (I STATE)	L2_RQSTS_I_STATE
35	Number of L2 address strobes	L2_ADS
36	Number of cycles during which the L2 caches data bus was busy	L2_DBUS_BUSY
37	Number of cycles during which the data bus was busy transferring read data from L2 to the processor	L2_DBUS_BUSY_RD
External Bus Logic (EBL)		
38	Number of clocks during which DRDY is asserted (SELF)	BUS_DRDY_CLOCKS_SELF
39	Number of clocks during which DRDY is asserted (ANY)	BUS_DRDY_CLOCKS_ANY
40	Number of clocks during which LOCK is asserted on the external system bus (SELF)	BUS_LOCK_CLOCKS_SELF
41	Number of clocks during which LOCK is asserted on the external system bus (ANY)	BUS_LOCK_CLOCKS_ANY

continued on next page

continued from previous page

#	Description	Symbol
42	Number of bus request outstanding	BUS_REQ_OUTSTANDING
43	Number of burst read transactions (SELF)	BUS_TRAN_BRD_SELF
44	Number of burst read transactions (ANY)	BUS_TRAN_BRD_ANY
45	Number of completed read for ownership transactions (SELF)	BUS_TRAN_RFO_SELF
46	Number of completed read for ownership transactions (ANY)	BUS_TRAN_RFO_ANY
47	Number of completed write back transactions (SELF)	BUS_TRANS_WB_SELF
48	Number of completed write back transactions (ANY)	BUS_TRANS_WB_ANY
49	Number of completed instruction fetch transactions (SELF)	BUS_TRAN_IFETCH_SELF
50	Number of completed instruction fetch transactions (ANY)	BUS_TRAN_IFETCH_ANY
51	Number of completed invalidate transactions (SELF)	BUS_TRAN_INVAL_SELF
52	Number of completed invalidate transactions (ANYY)	BUS_TRAN_INVAL_ANY
53	Number of completed partial write transactions (SELF)	BUS_TRAN_PWR_SELF
54	Number of completed partial write transactions (ANY)	BUS_TRAN_PWR_ANY
55	Number of completed partial transactions (SELF)	BUS_TRANS_P_SELF
56	Number of completed partial transactions (ANY)	BUS_TRANS_P_ANY
57	Number of completed I/O transactions (SELF)	BUS_TRANS_IO_SELF
58	Number of completed I/O transactions (ANY)	BUS_TRANS_IO_ANY
59	Number of completed deferred transactions (SELF)	BUS_TRAN_DEF_SELF

continued on next page

continued from previous page

#	Description	Symbol
60	Number of completed deferred transactions (ANY)	BUS_TRAN_DEF_ANY
61	Number of completed burst transactions (SELF)	BUS_TRAN_BURST_SELF
62	Number of completed burst transactions (ANY)	BUS_TRAN_BURST_ANY
63	Number of all completed bus transactions (SELF)	BUS_TRAN_ANY_SELF
64	Number of all completed bus transactions (ANY)	BUS_TRAN_ANY_ANY
65	Number of completed memory transactions (SELF)	BUS_TRAN_MEM_SELF
66	Number of completed memory transactions (ANY)	BUS_TRAN_MEM_ANY
67	Number of bus clock cycles during which this processor is receiving data	BUS_DATA_RCV
68	Number of bus clock cycles during which this processor is driving the BNR pin	BUS_BNR_DRV
69	Number of bus clock cycles during which this prcessor is driving the HIT pin	BUS_HIT_DRV
70	Number of bus clock cycles during which this processor is driving the HITM pin	BUS_HITM_DRV
71	Number of clock cycles during which the bus is snoop stalled	BUS_SNOOP_STALL
Floting Point Unit		
72	Number of computational floating-point operations retired	FLOPS
73	Number of computetional floating-point operations executed	FP_COMP_OPS_EXE

continued on next page

continued from previous page

#	Description	Symbol
74	Number of floating-point exception cases handled by microcode	FP_ASSIST
75	Number of multiplies	MUL
76	Number of divides	DVI
77	Number of cycles during which the divider is busy, and cannot accept new divides	CYCLE_DIV_BUSY
Memory Ordering		
78	Number of load operations delayed due to store buffer blocks	LD_BLOCKS
79	Number of store buffer drain cycles	SB_DRAINS
80	Number of misaligned data memory references	MISALIGN_MEMREF
81	Number of streaming SIMD extensions prefetch/weakly ordered instruction dispached (prefetch NTA)	EMON_KNI_PREF_DISPATCHED_NTA
82	Number of streaming SIMD extensions prefetch/weakly ordered instruction dispached (prefetch T1)	EMON_KNI_PREF_DISPATCHED_T1
83	Number of streaming SIMD extensions prefetch/weakly ordered instruction dispached (prefetch T2)	EMON_KNI_PREF_DISPATCHED_T2
84	Number of streaming SIMD extensions prefetch/weakly ordered instruction dispached (weakly ordered stores)	EMON_KNI_PREF_DISPATCHED_WEAKLY
85	Number of prefetch/weakly ordered instruction that miss all caches (prefetch NTA)	EMON_KNI_PREF_MISS_NTA

continued on next page

continued from previous page

#	Description	Symbol
86	Number of prefetch/weakly ordered instruction that miss all caches (prefetch T1)	EMON_KNI_PREF_MISS_T1
87	Number of prefetch/weakly ordered instruction that miss all caches (prefetch T2)	EMON_KNI_PREF_MISS_T2
88	Number of prefetch/weakly ordered instruction that miss all caches (weakly ordered stores)	EMON_KNI_PREF_MISS_WEAKLY
Instruction Decoding and Retirement		
89	Number of instructions retired	INST_RETIRED
90	Number of micro-ops retired	UOPS_RETIRED
91	Number of instructions decoded	INST_DECODED
92	Number of streaming SIMD extentions retired (packed & scalar)	EMON_KNI_INST_RETIRED_PACK_SCA
93	Number of streaming SIMD extentions retired (scalar)	EMON_KNI_INST_RETIRED_SCA
94	Number of streaming SIMD extensions computation instructions retired (packed & scalar)	EMON_KNI_COMP_INST_RET_PACK_SCA
95	Number of streaming SIMD extensions computation instructions retired (scalar)	EMON_KNI_COMP_INST_RET_SCA
Interrupts		
96	Number of hardware interrupts received	HW_INT_RX
97	Number of processor cycles for which interrupts are disabled	CYCLES_INT_MASKED
98	Number of processor cycles for which interrupts are disabled and interrupt are pending	CYCLES_INT_PENDING_AND_MASKED
Branches		
99	Number of branch instructions retired	BR_INST_RETIRED

continued on next page

continued from previous page

#	Description	Symbol
100	Number of mispredicted branches retired	BR_MISS_PRE_RETIRED
101	Number of taken branches retired	BR_TAKEN_RETIRED
102	Number of taken mispredicted branches retired	BR_MISS_PRED_TAKEN_RET
103	Number of branch instructions decoded	BR_INST_DECODED
104	Number of branches for which the BTB did not produce a prediction	BTB_MISSES
105	Number of bogus branches	BR_BOGUS
106	Number of times BACLEAR is asserted	BACLEARS
Stalls		
107	Incremented by 1 during every cycle for which there is a resource related stall	RESOURCE_STALLS
108	Number of cycles or events for partial stalls	PARTIAL_RAT_STALLS
Segment Register Loads		
109	Number of segment register loads	SEGMENT_REG_LOADS
Clocks		
110	Number of cycles during which the processor is not halted	CPU_CLK_UNHALTED
MMX UNIT		
111	Number of MMX saturating instructions executed	MMX_SAT_INSTR_EXEC
112	Number of MMX micro-ops executed	MMX_UOPS_EXEC
113	MMX instructions executed (packed multiply)	MMX_INSTR_TYPE_EXEC_MULTIPLY
114	MMX instructions executed (packed shift)	MMX_INSTR_TYPE_EXEC_SHIFT
115	MMX instructions executed (pack operation)	MMX_INSTR_TYPE_EXEC_PACK

continued on next page

continued from previous page

#	Description	Symbol
116	MMX instructions executed (unpack operation)	MMX_INSTR_TYPE_EXEC_UNPACK
117	MMX instructions executed (packed logical)	MMX_INSTR_TYPE_EXEC_LOGICAL
118	MMX instructions executed (packed arithmetic)	MMX_INSTR_TYPE_EXEC_ARITHMETIC
119	Transition between floating-point and MMX instructions (MMX instruction to floting-point)	FP_MMX_TRANS_MMX_TO_FP
120	Transition between floating-point and MMX instructions (floting-point instruction to MMX)	FP_MMX_TRANS_FP_TO_MMX
121	Number of MMX assits (number of EMMS instruction executed)	MMX_ASSIST
Segment Register Renaming		
122	Number of segment register renaming stalls (Segment Register ES)	SEG_RENAME_STALLS_ES
123	Number of segment register renaming stalls (Segment Register DS)	SEG_RENAME_STALLS_DS
124	Number of segment register renaming stalls (Segment Register FS)	SEG_RENAME_STALLS_FS
125	Number of segment register renaming stalls (Segment Register GS)	SEG_RENAME_STALLS_GS
126	Number of segment register renaming stalls (Segment Register ALL)	SEG_RENAME_STALLS_ALL
127	Number of segment register renames (Segment Register ES)	SEG_REG_RENAMES_ES
128	Number of segment register renames (Segment Register DS)	SEG_REG_RENAMES_DS

continued on next page

continued from previous page

#	**Description**	**Symbol**
129	Number of segment register renames (Segment Register FS)	SEG_REG_RENAMES_FS
130	Number of segment register renames (Segment Register GS)	SEG_REG_RENAMES_GS
131	Number of segment register renames (Segment Register ALL)	SEG_REG_RENAMES_ALL
132	Number of segment register rename events retired	RET_SEG_RENAMES

Table A.1: PIII Performance Monitoring Counters Description

Appendix B

Appendix: PMC Offline Autocorrelation Analysis

B.1 PMC Offline Autocorrelation Analysis Procedure

This section presents the full details for the offline autocorrelation analysis procedure described in section 5.2 on page 71.

B.1.1 Histogram and Samples for real PMC Readings

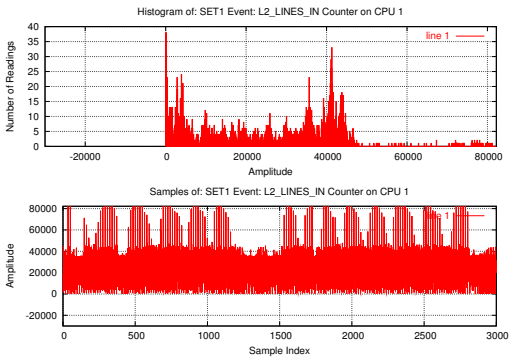

Figure B.1: Histogram and samples for sample set 1 on CPU 1

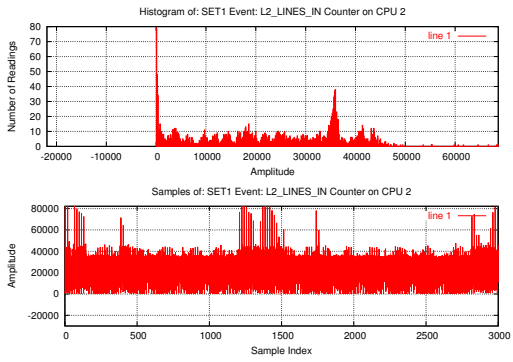

Figure B.2: Histogram and samples for sample set 1 on CPU 2

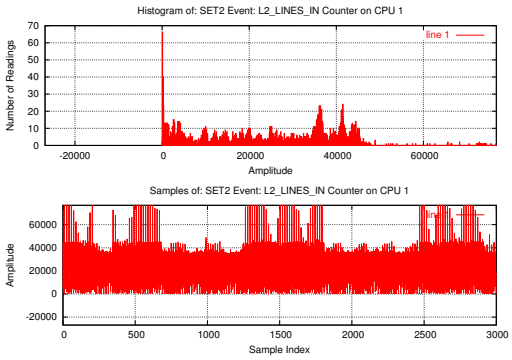

Figure B.3: Histogram and samples for sample set 2 on CPU 1

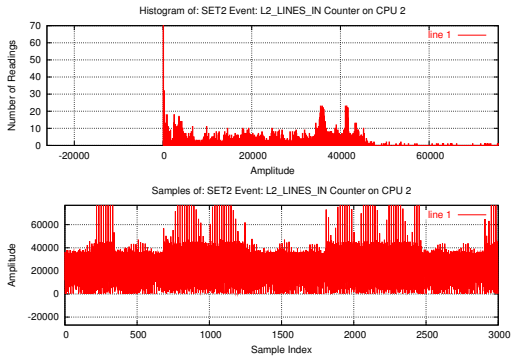

Figure B.4: Histogram and samples for sample set 2 on CPU 2

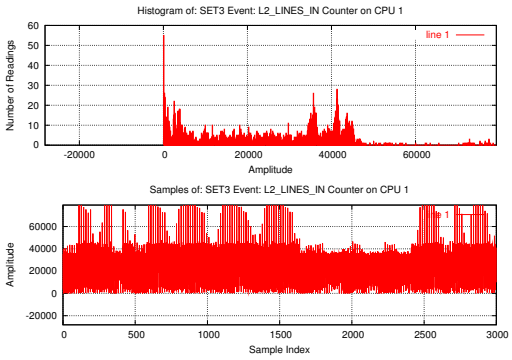

Figure B.5: Histogram and samples for sample set 3 on CPU 1

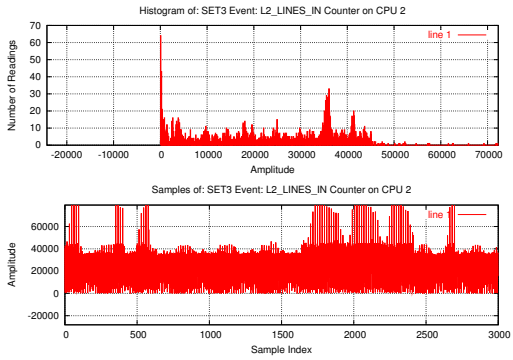

Figure B.6: Histogram and samples for sample set 3 on CPU 2

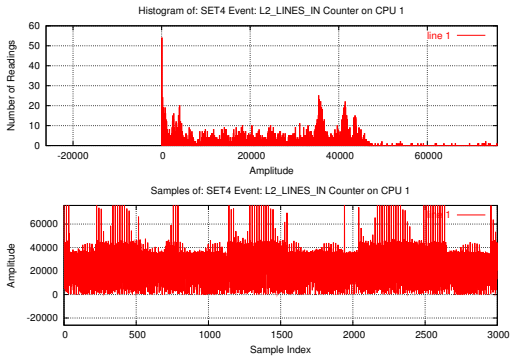

Figure B.7: Histogram and samples for sample set 4 on CPU 1

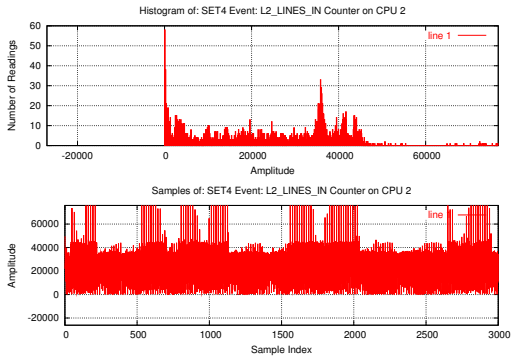

Figure B.8: Histogram and samples for sample set 4 on CPU 2

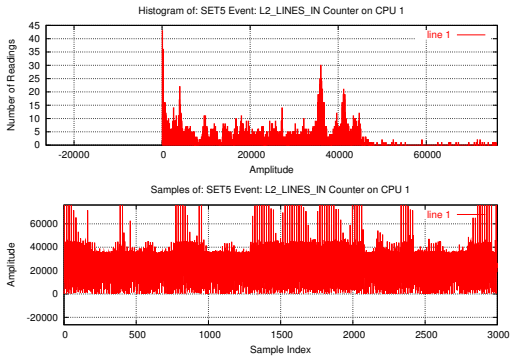

Figure B.9: Histogram and samples for sample set 5 on CPU 1

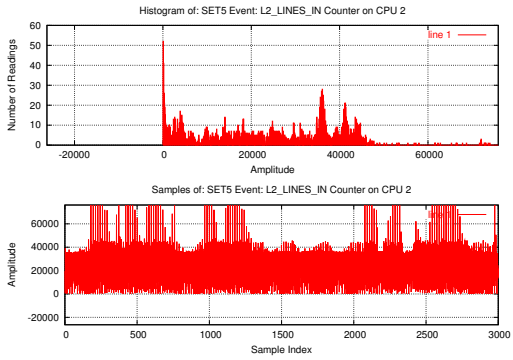

Figure B.10: Histogram and samples for sample set 5 on CPU 2

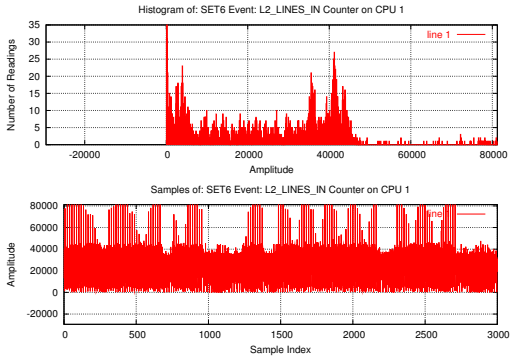

Figure B.11: Histogram and samples for sample set 6 on CPU 1

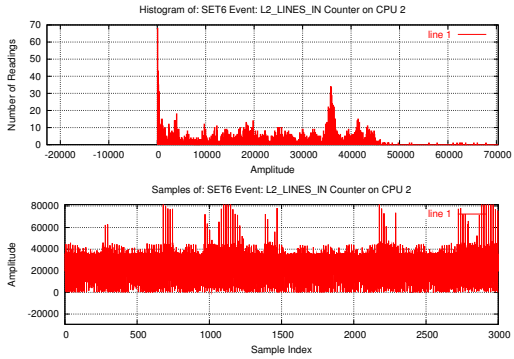

Figure B.12: Histogram and samples for sample set 6 on CPU 2

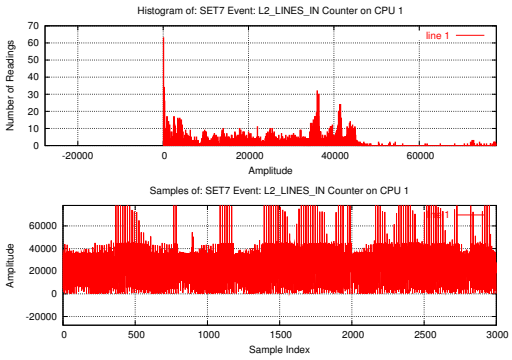

Figure B.13: Histogram and samples for sample set 7 on CPU 1

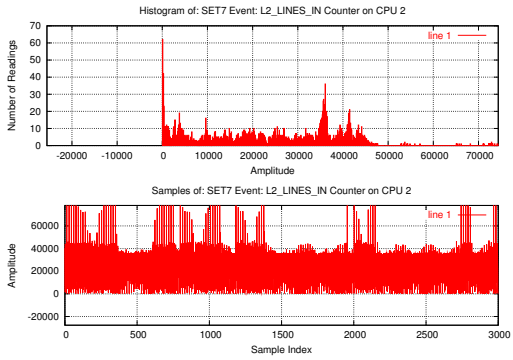

Figure B.14: Histogram and samples for sample set 7 on CPU 2

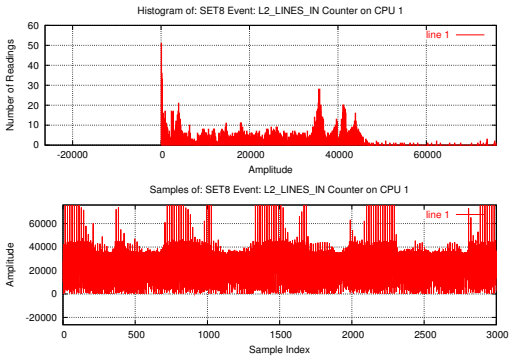

Figure B.15: Histogram and samples for sample set 8 on CPU 1

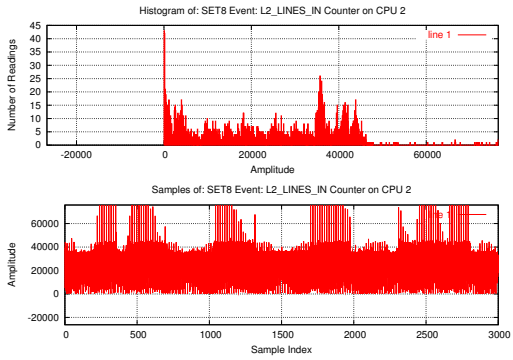

Figure B.16: Histogram and samples for sample set 8 on CPU 2

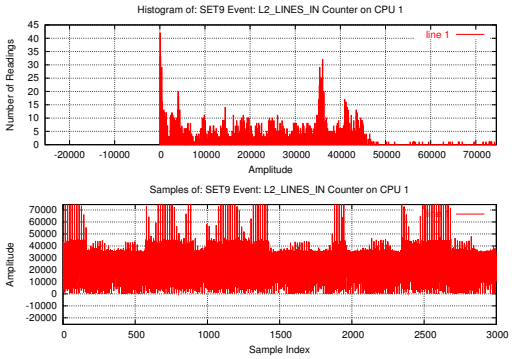

Figure B.17: Histogram and samples for sample set 9 on CPU 1

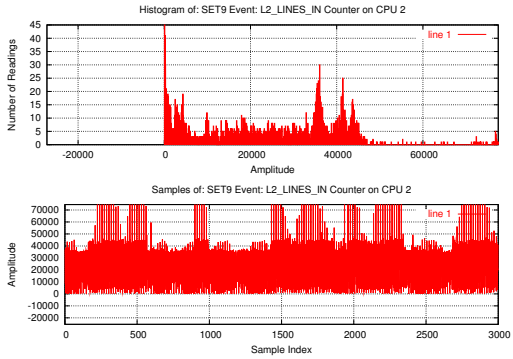

Figure B.18: Histogram and samples for sample set 9 on CPU 2

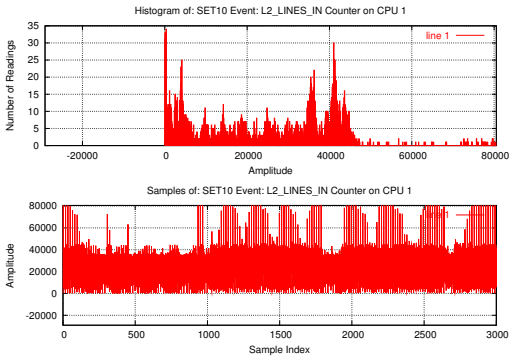

Figure B.19: Histogram and samples for sample set 10 on CPU 1

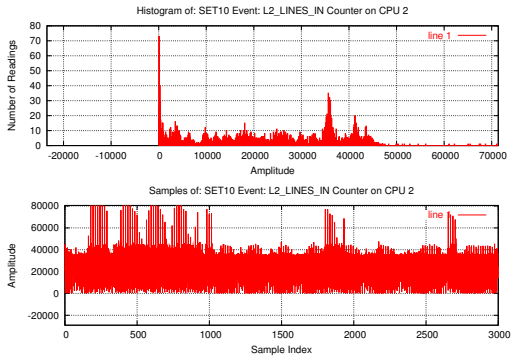

Figure B.20: Histogram and samples for sample set 10 on CPU 2

B.1.2 Histogram and Samples for Simulated Readings

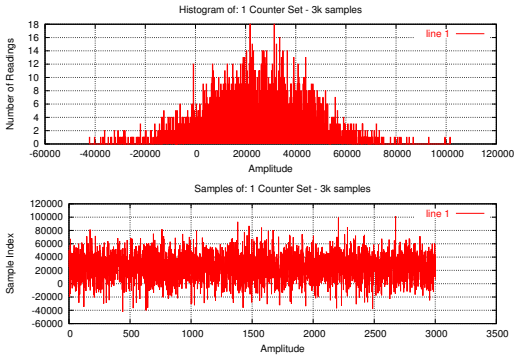

Figure B.21: Histogram and samples for simulation set 1

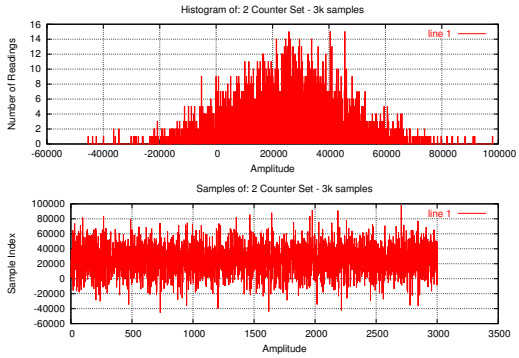

Figure B.22: Histogram and samples for simulation set 2

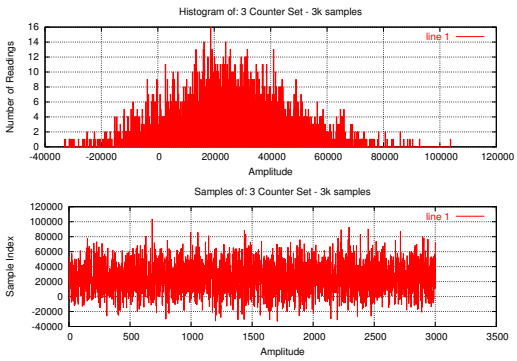

Figure B.23: Histogram and samples for simulation set 3

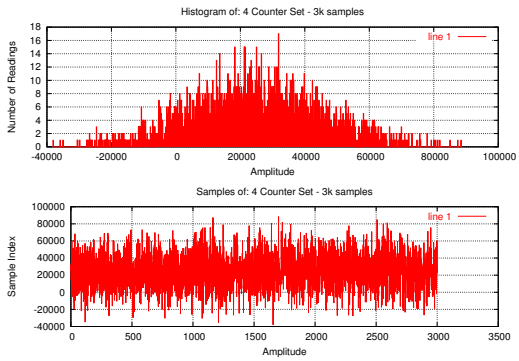

Figure B.24: Histogram and samples for simulation set 4

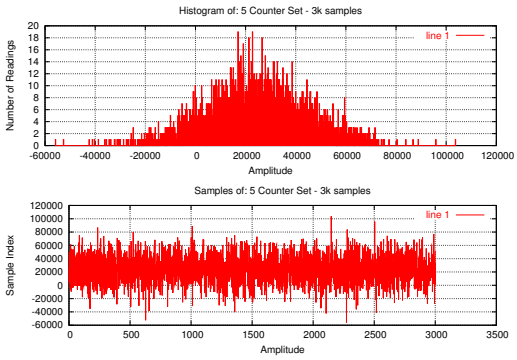

Figure B.25: Histogram and samples for simulation set 5

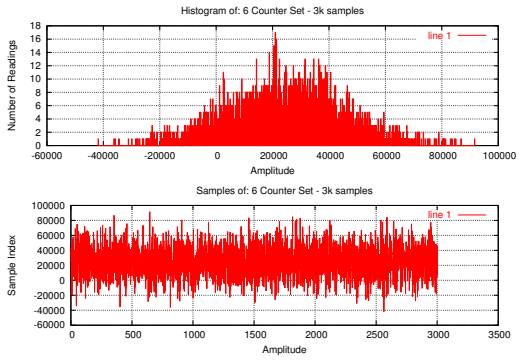

Figure B.26: Histogram and samples for simulation set 6

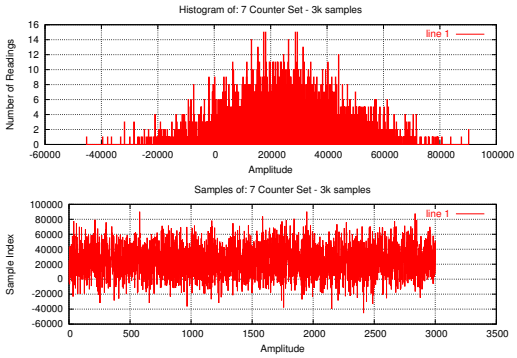

Figure B.27: Histogram and samples for simulation set 7

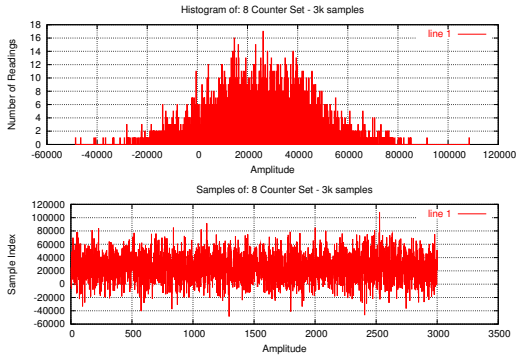

Figure B.28: Histogram and samples for simulation set 8

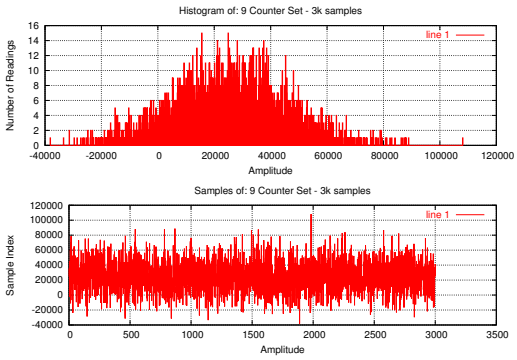

Figure B.29: Histogram and samples for simulation set 9

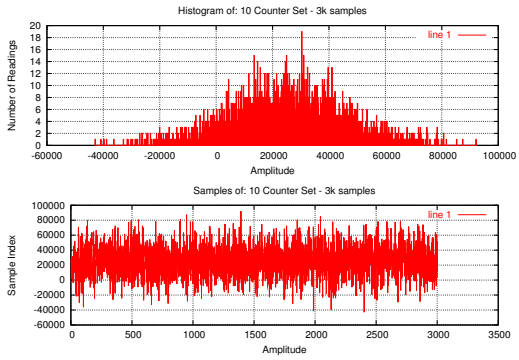

Figure B.30: Histogram and samples for simulation set 10

B.1.3 Histogram and Autocorrelation for Real PMC Readings

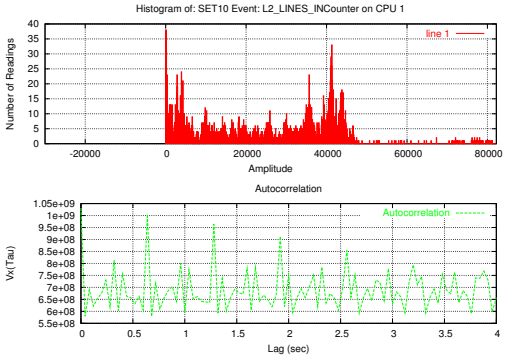

Figure B.31: Histogram and autocorrelation for sample set 1 on CPU 1

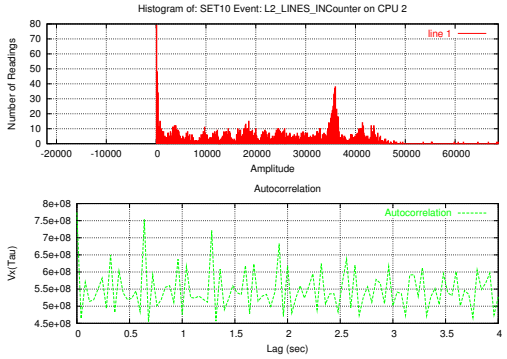

Figure B.32: Histogram and autocorrelation for sample set 1 on CPU 2

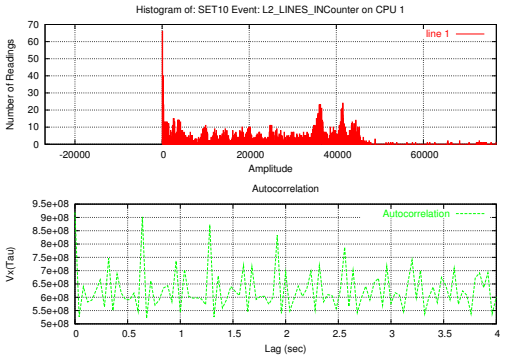

Figure B.33: Histogram and autocorrelation for sample set 2 on CPU 1

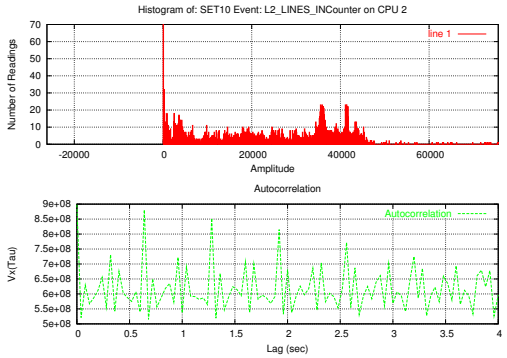

Figure B.34: Histogram and autocorrelation for sample set 2 on CPU 2

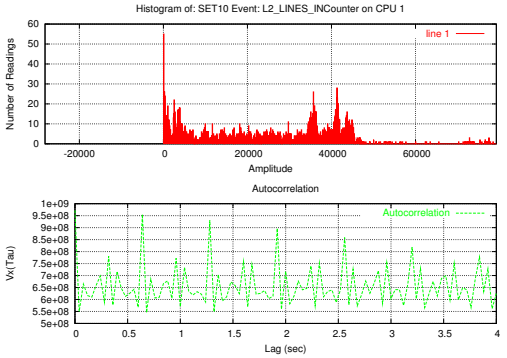

Figure B.35: Histogram and autocorrelation for sample set 3 on CPU 1

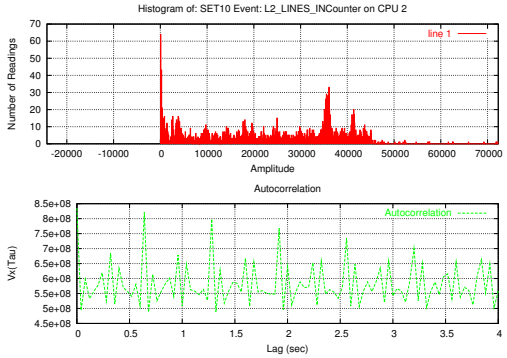

Figure B.36: Histogram and autocorrelation for sample set 3 on CPU 2

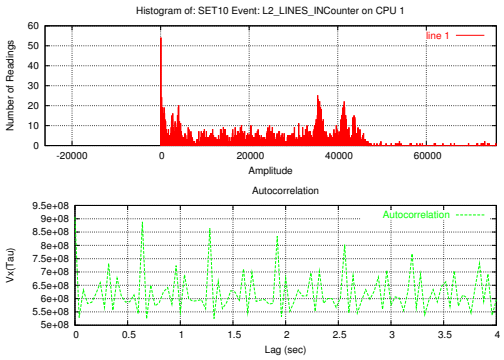

Figure B.37: Histogram and autocorrelation for sample set 4 on CPU 1

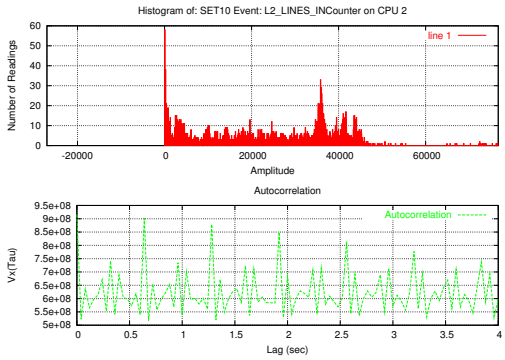

Figure B.38: Histogram and autocorrelation for sample set 4 on CPU 2

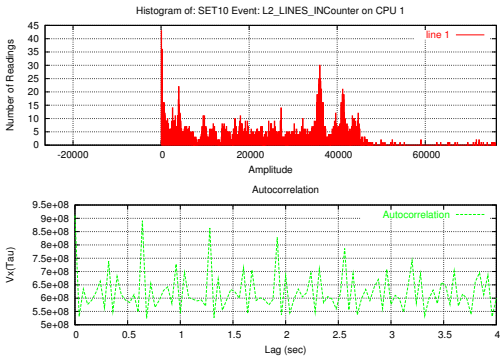

Figure B.39: Histogram and autocorrelation for sample set 5 on CPU 1

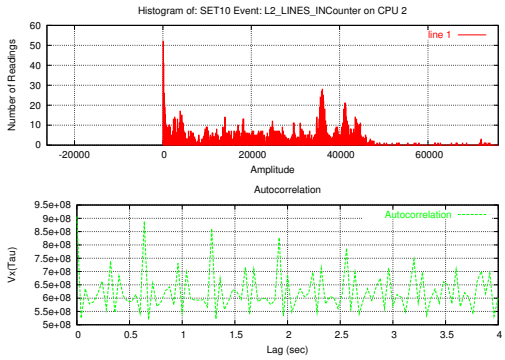

Figure B.40: Histogram and autocorrelation for sample set 5 on CPU 2

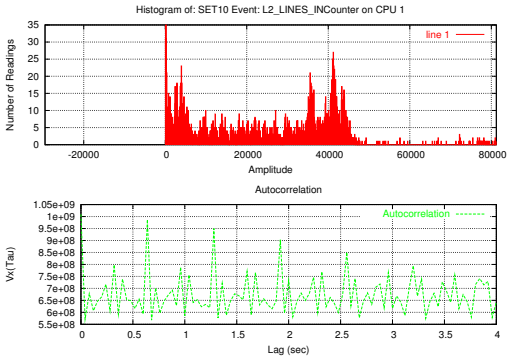

Figure B.41: Histogram and autocorrelation for sample set 6 on CPU 1

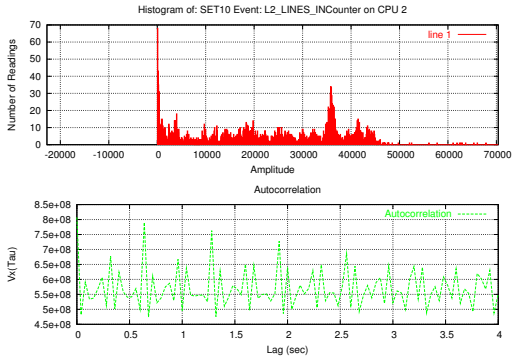

Figure B.42: Histogram and autocorrelation for sample set 6 on CPU 2

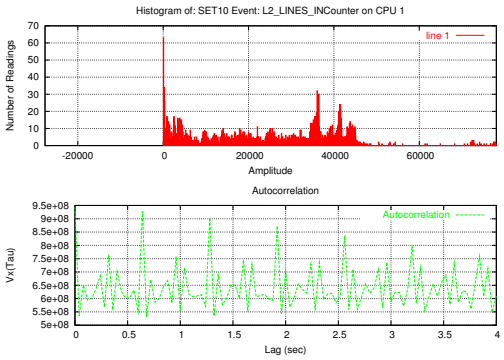

Figure B.43: Histogram and autocorrelation for sample set 7 on CPU 1

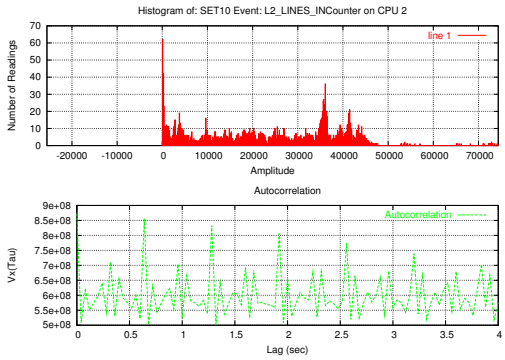

Figure B.44: Histogram and autocorrelation for sample set 7 on CPU 2

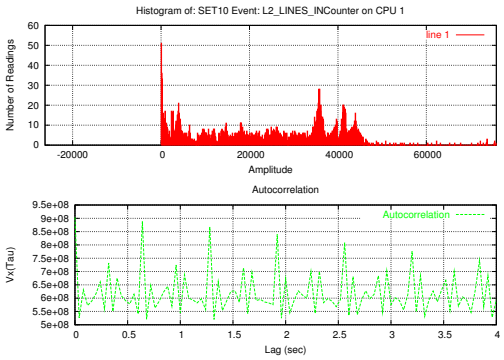

Figure B.45: Histogram and autocorrelation for sample set 8 on CPU 1

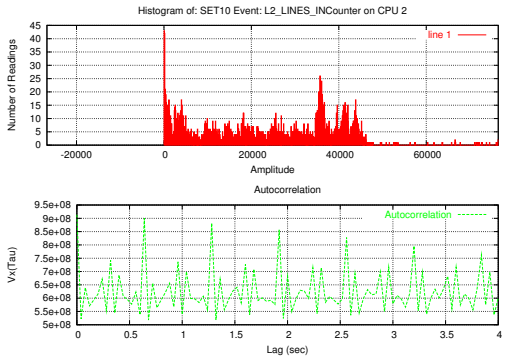

Figure B.46: Histogram and autocorrelation for sample set 8 on CPU 2

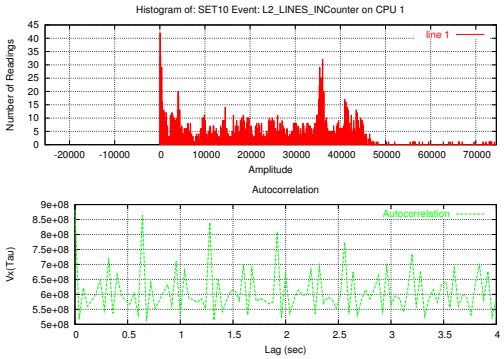

Figure B.47: Histogram and autocorrelation for sample set 9 on CPU 1

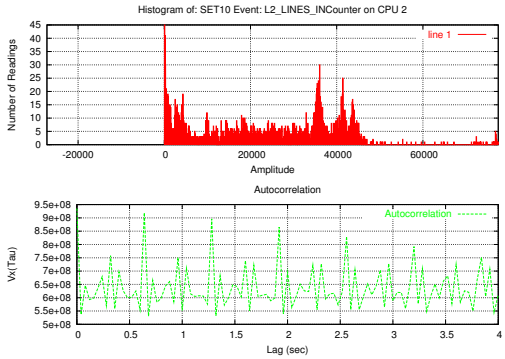

Figure B.48: Histogram and autocorrelation for sample set 9 on CPU 2

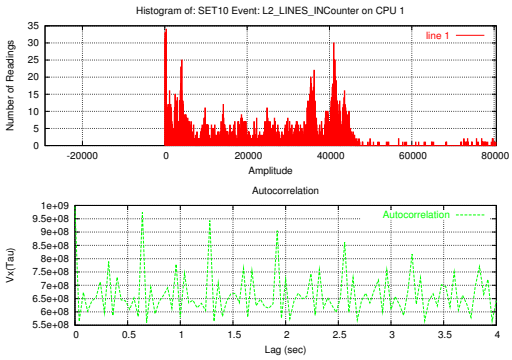

Figure B.49: Histogram and autocorrelation for sample set 10 on CPU 1

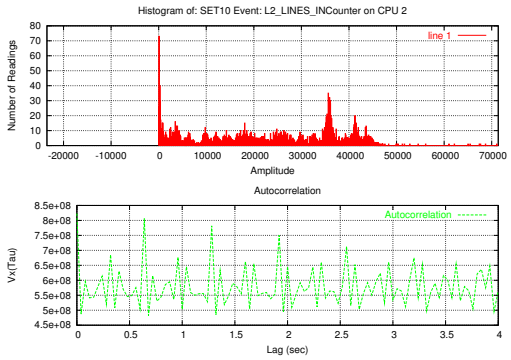

Figure B.50: Histogram and autocorrelation for sample set 10 on CPU 2

B.1.4 Histogram and Autocorrelation for Simulated Readings

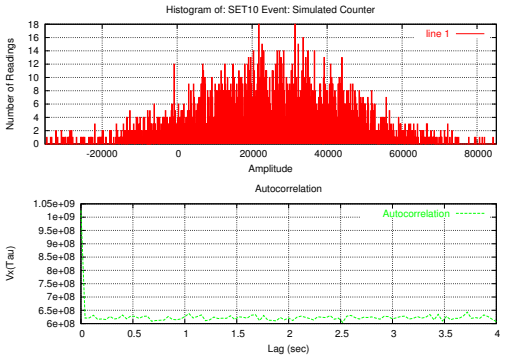

Figure B.51: Histogram and autocorrelation for simulation set 1

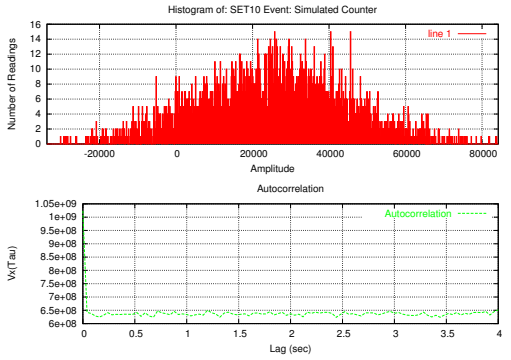

Figure B.52: Histogram and autocorrelation for simulation set 2

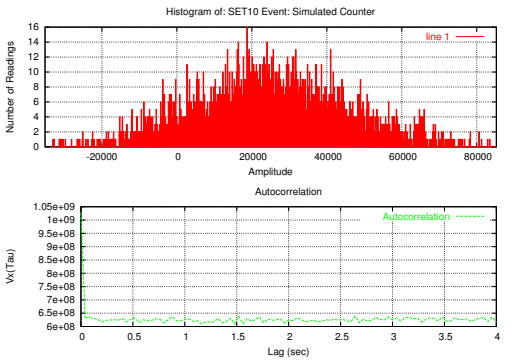

Figure B.53: Histogram and autocorrelation for simulation set 3

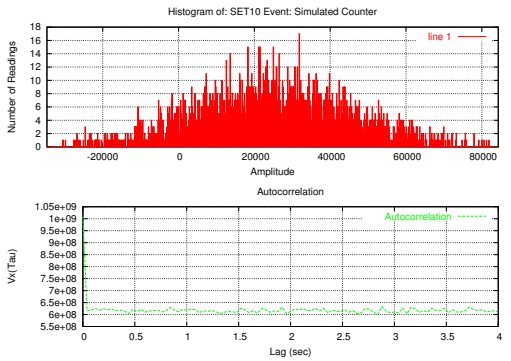

Figure B.54: Histogram and autocorrelation for simulation set 4

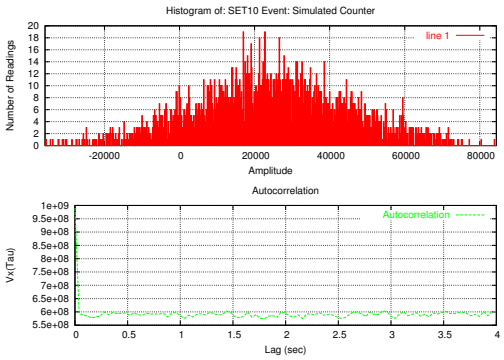

Figure B.55: Histogram and autocorrelation for simulation set 5

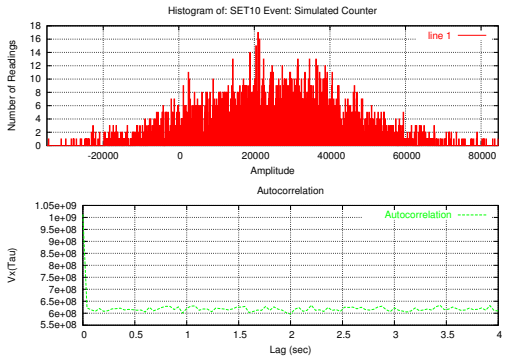

Figure B.56: Histogram and autocorrelation for simulation set 6

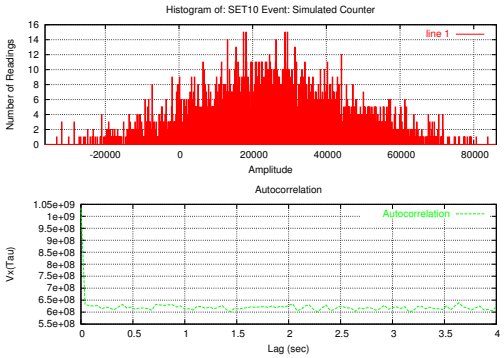

Figure B.57: Histogram and autocorrelation for simulation set 7

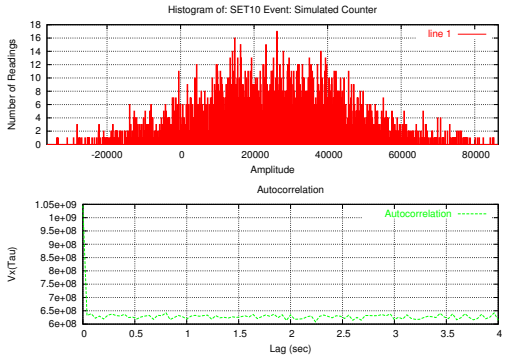

Figure B.58: Histogram and autocorrelation for simulation set 8

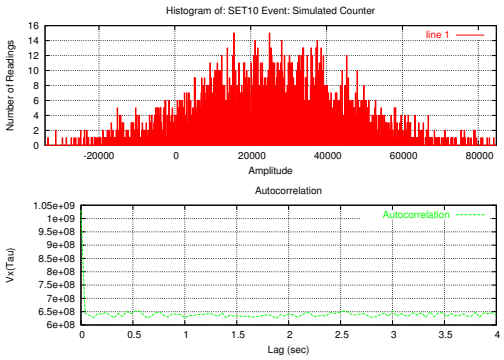

Figure B.59: Histogram and autocorrelation for simulation set 9

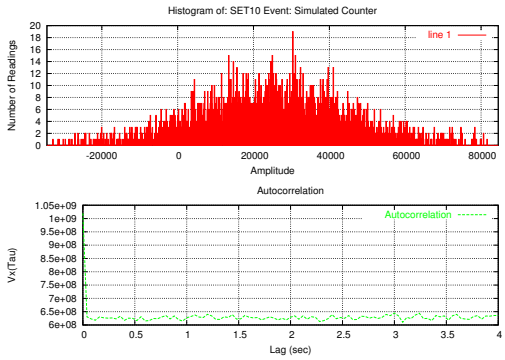

Figure B.60: Histogram and autocorrelation for simulation set 10

B.1.5 Histogram of Real PMC Readings with superimposed Gaussian PDF

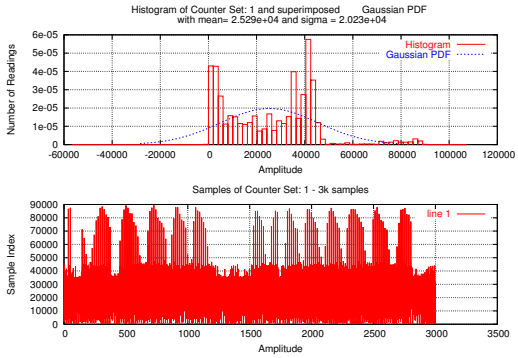

Figure B.61: Histogram with superimposed Gaussian PDF for sample set 1 on CPU 1

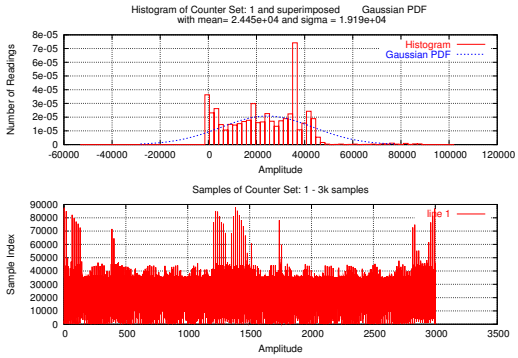

Figure B.62: Histogram with superimposed Gaussian PDF for sample set 1 on CPU 2

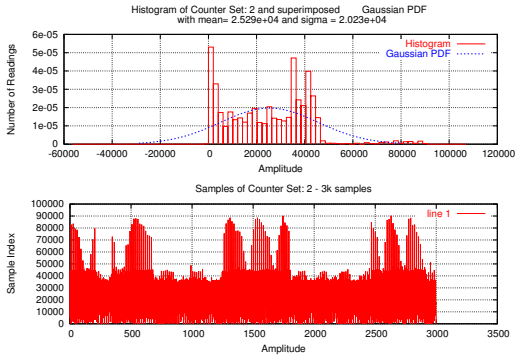

Figure B.63: Histogram with superimposed Gaussian PDF for sample set 2 on CPU 1

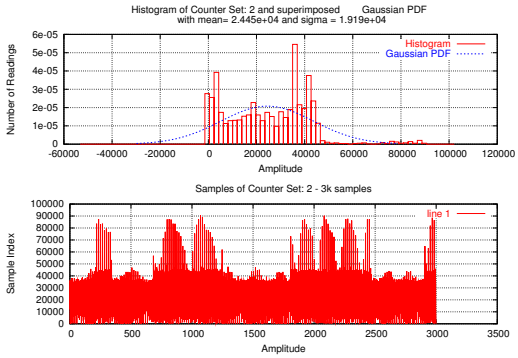

Figure B.64: Histogram with superimposed Gaussian PDF for sample set 2 on CPU 2

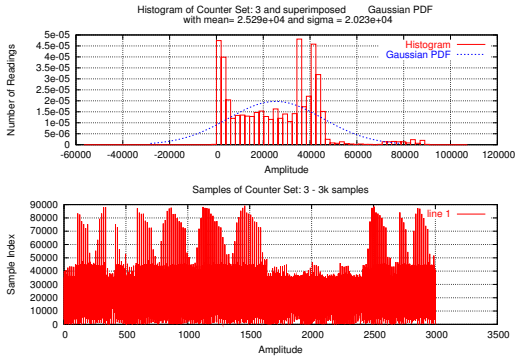

Figure B.65: Histogram with superimposed Gaussian PDF for sample set 3 on CPU 1

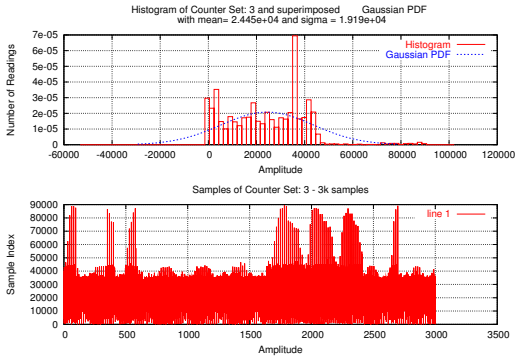

Figure B.66: Histogram with superimposed Gaussian PDF for sample set 3 on CPU 2

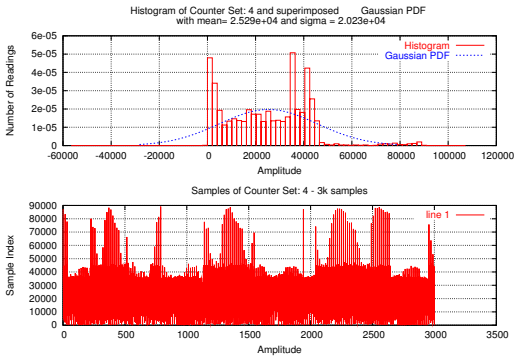

Figure B.67: Histogram with superimposed Gaussian PDF for sample set 4 on CPU 1

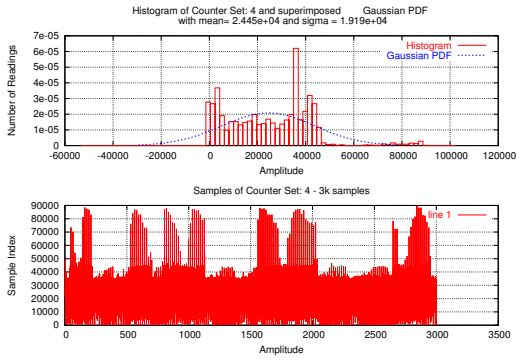

Figure B.68: Histogram with superimposed Gaussian PDF for sample set 4 on CPU 2

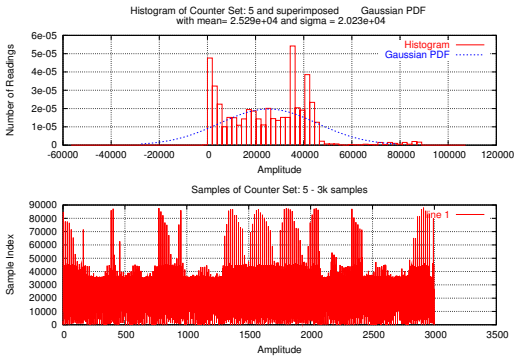

Figure B.69: Histogram with superimposed Gaussian PDF for sample set 5 on CPU 1

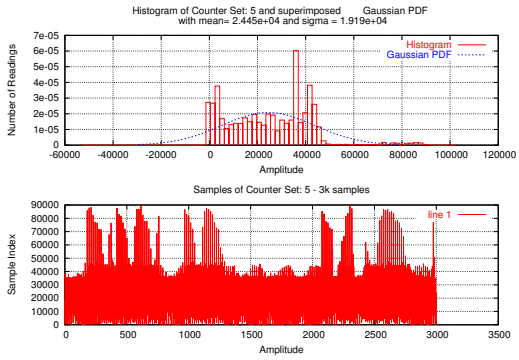

Figure B.70: Histogram with superimposed Gaussian PDF for sample set 5 on CPU 2

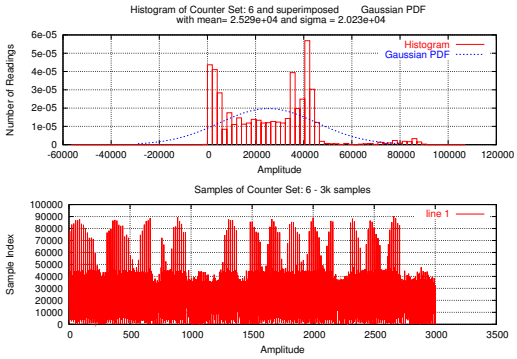

Figure B.71: Histogram with superimposed Gaussian PDF for sample set 6 on CPU 1

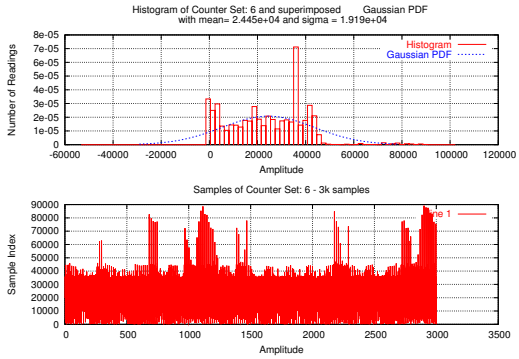

Figure B.72: Histogram with superimposed Gaussian PDF for sample set 6 on CPU 2

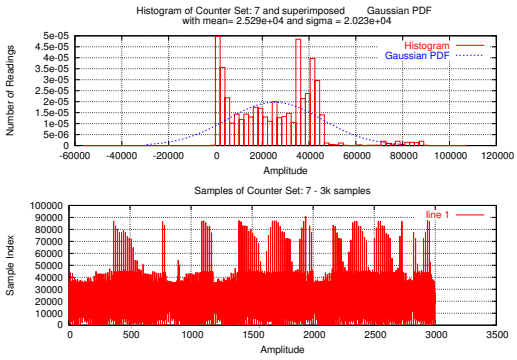

Figure B.73: Histogram with superimposed Gaussian PDF for sample set 7 on CPU 1

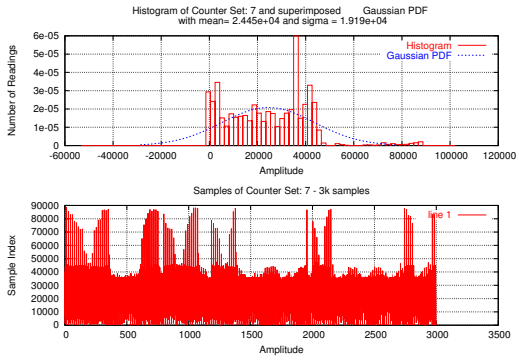

Figure B.74: Histogram with superimposed Gaussian PDF for sample set 7 on CPU 2

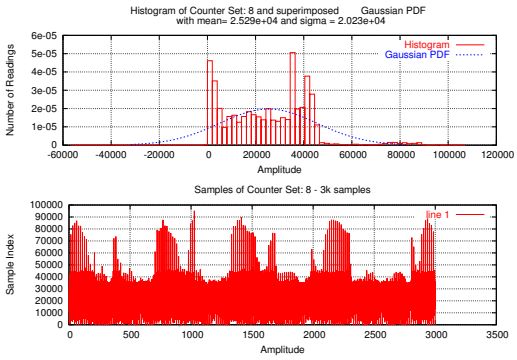

Figure B.75: Histogram with superimposed Gaussian PDF for sample set 8 on CPU 1

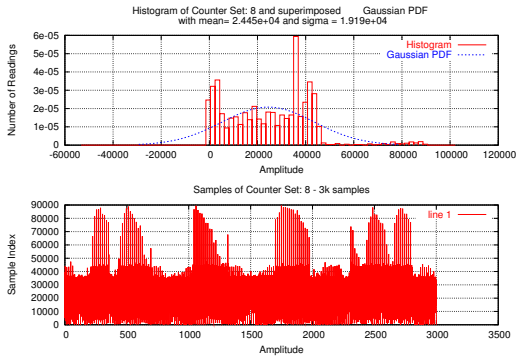

Figure B.76: Histogram with superimposed Gaussian PDF for sample set 8 on CPU 2

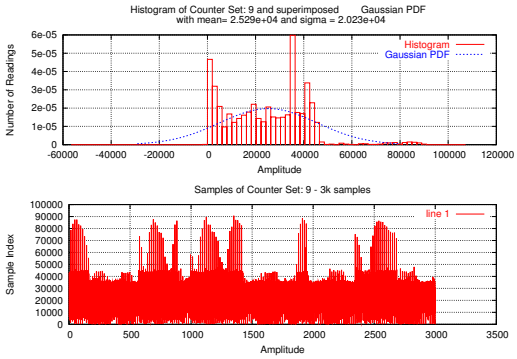

Figure B.77: Histogram with superimposed Gaussian PDF for sample set 9 on CPU 1

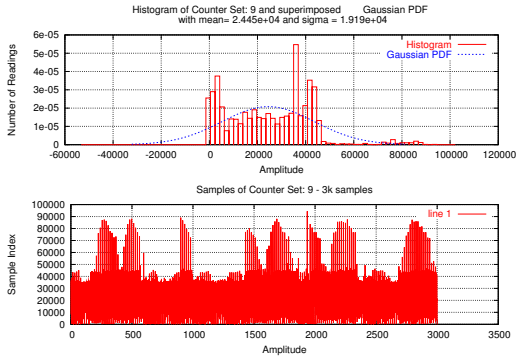

Figure B.78: Histogram with superimposed Gaussian PDF for sample set 9 on CPU 2

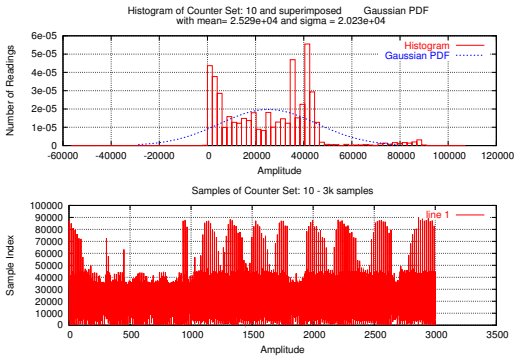

Figure B.79: Histogram with superimposed Gaussian PDF for sample set 10 on CPU 1

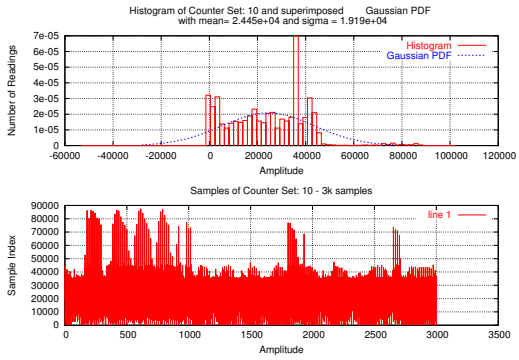

Figure B.80: Histogram with superimposed Gaussian PDF for sample set 10 on CPU 2

B.1.6 Histogram of Simulated Readings with superimposed Gaussian PDF

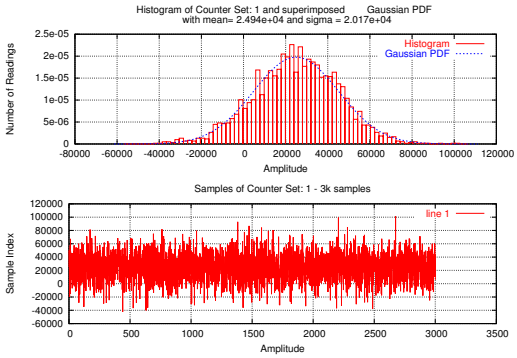

Figure B.81: Histogram with superimposed Gaussian PDF for simulation set 1

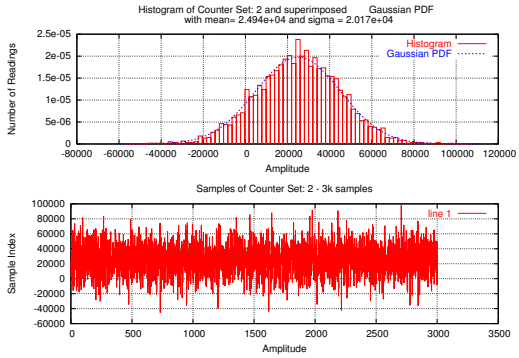

Figure B.82: Histogram with superimposed Gaussian PDF for simulation set 2

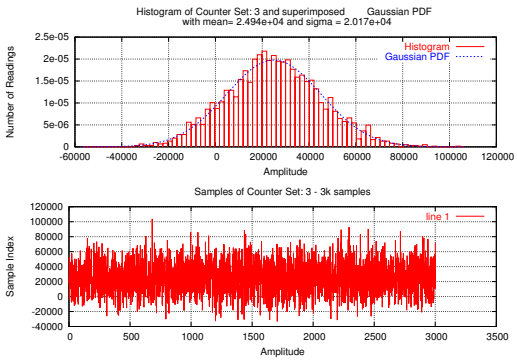

Figure B.83: Histogram with superimposed Gaussian PDF for simulation set 3

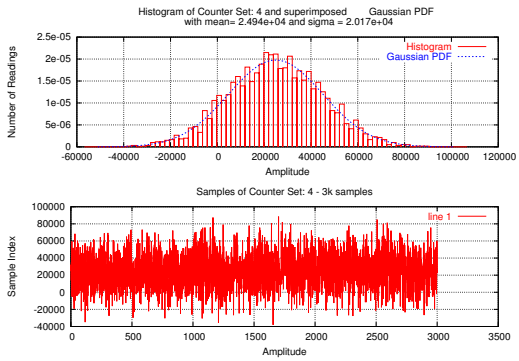

Figure B.84: Histogram with superimposed Gaussian PDF for simulation set 4

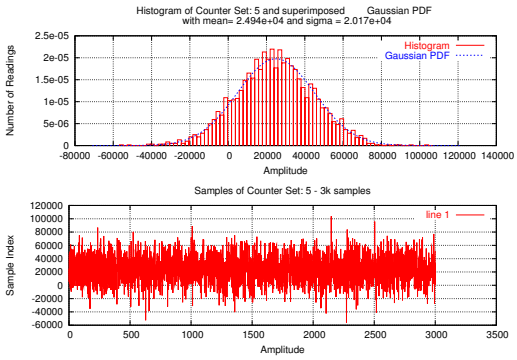

Figure B.85: Histogram with superimposed Gaussian PDF for simulation set 5

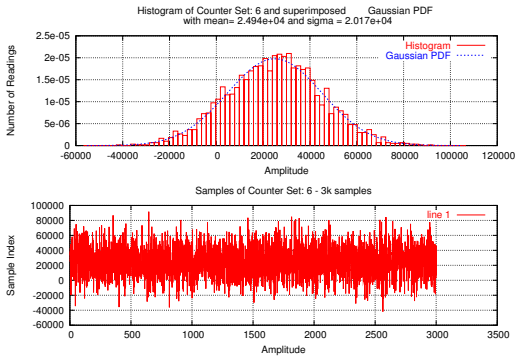

Figure B.86: Histogram with superimposed Gaussian PDF for simulation set 6

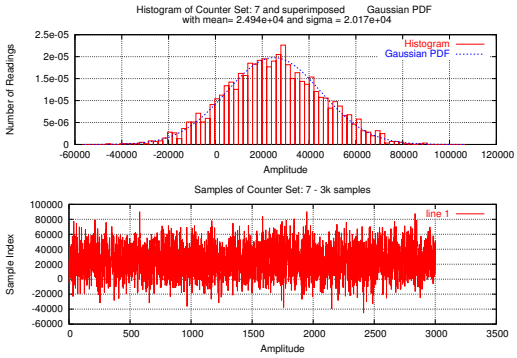

Figure B.87: Histogram with superimposed Gaussian PDF for simulation set 7

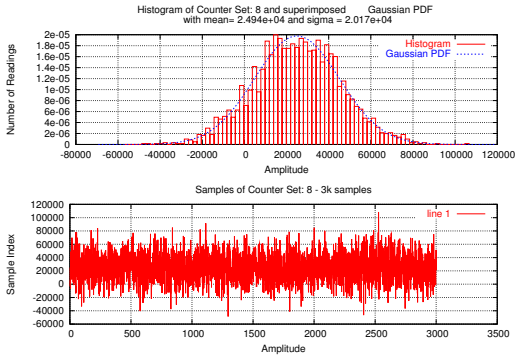

Figure B.88: Histogram with superimposed Gaussian PDF for simulation set 8

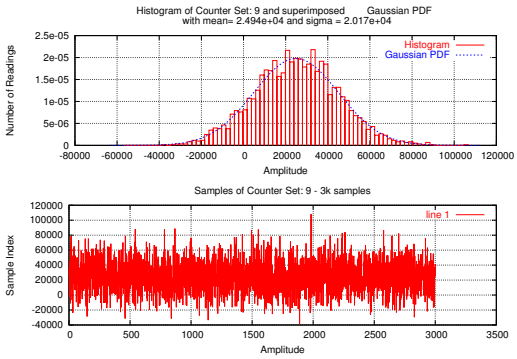

Figure B.89: Histogram with superimposed Gaussian PDF for simulation set 9

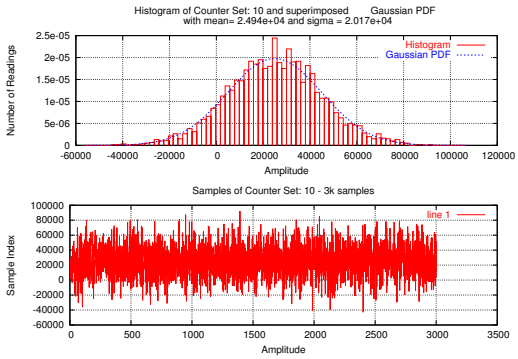

Figure B.90: Histogram with superimposed Gaussian PDF for simulation set 10

B.2 PIII Performance Monitoring Counter (PMC) off-line autocorrelation analysis results.

The section presents the PMC sample analysis results for all the 132 PMC events as described in section 5.2.12 on page 82.

#	Symbol	CPU	σ_i	β_i	Mean
Data Cache Unit (DCU)					
1	DATA_MEM_REFS	1	5.26e+06	38.8	6.19e+06
		2	5.03e+06	35.7	5.73e+06
2	DCU_LINES_IN	1	5.83e+04	49.2	7.40e+04
		2	5.92e+04	46.5	7.40e+04
3	DCU_M_LINES_IN	1	2.20e+04	58.4	2.88e+04
		2	1.91e+04	56.1	2.39e+04
4	DCU_M_LINES_OUT	1	2.55e+04	52.1	3.249e+04
		2	2.36e+04	52.0	2.98e+04
5	DCU_MISS_OUTSTANDING	1	No Data		
		2	1.66e+06	146.1	2.23e+06
Instruction Fetch Unit (IFU)					
6	IFU_IFETCH	1	6.65e+06	55.3	8.40e+06
		2	6.59e+06	51.5	8.13e+06
7	IFU_IFETCH_MISS	1	2.41e+04	34.3	2.77e+04
		2	2.48e+04	34.5	2.86e+04
8	ITLB_MISS	1	539.4	39.2	613.8
		2	495.0	36.1	518.3
9	IFU_MEM_STALL	1	7.99e+05	47.4	1.01e+06
		2	8.26e+05	48.0	1.05e+06
10	ILD_STALL	1	2.70e+04	58.6	2.29e+04
		2	2.90e+04	59.6	2.457e+04
L2 Cache					
11	L2_IFETCH_MESI	1	9.37e+04	38.5	1.09e+05
		2	8.70e+04	35.3	9.85e+04
12	L2_IFETCH_M_STATE	1	No Data		
		2			
13	L2_IFETCH_E_STATE	1	1.73	26.6	0.2438

$\Delta t = 0.04sec$ for every PMC reading. continued on next page

$\Delta t = 0.04sec$ for every PMC reading.					Continued from previous page
#	**Symbol**	**CPU**	σ_i	β_i	**Mean**
		2	1.24	43.9	0.2646
14	L2_IFETCH_S_STATE	1	8.49e+04	33.5	9.31e+04
		2	9.35e+04	37.8	1.06e+05
15	L2_IFETCH_I_STATE	1	No Data		
		2	2524	112.6	3372
16	L2_LD_MESI	1	4.09e+04	41.6	4.81e+04
		2	4.13e+04	41.7	4.89e+04
17	L2_LD_M_STATE	1	1.50e+04	38.3	1.71e+04
		2	1.50e+04	38.3	1.71e+04
18	L2_LD_E_STATE	1	1.679e+04	31.7	1.779e+04
		2	1.785e+04	35.5	1.983e+04
19	L2_LD_S_STATE	1	7232.0	21.2	5382.0
		2	6393.0	28.1	5471.0
20	L2_LD_I_STATE	1	8063	102.4	1.04e+04
		2	8159	103.6	1.04e+04
21	L2_ST_MESI	1	1.36e+04	88.3	1.76e+04
		2	1.38e+04	88.2	1.79e+04
22	L2_ST_M_STATE	1	6934.0	32.3	7447.0
		1	7637.0	39.4	8838.0
23	L2_ST_E_STATE	1	201.5	57.5	143.4
		2	203.4	57.9	145.4
24	L2_ST_S_STATE	1	1518.0	66.6	2049.0
		2	1442.0	64.5	1934.0
25	L2_ST_I_STATE	1	7734.0	70.2	8568.0
		2	9144.0	77.4	1.00e+04
26	L2_LINES_IN	1	2.02e+04	112.7	2.53e+04
		2	1.92e+04	103.0	2.45e+04
27	L2_LINES_OUT	1	1.97e+04	104.8	2.49e+04
		2	1.96e+04	106.1	2.49e+04
28	L2_M_LINES_INM	1	1.08e+04	105.7	1.37e+04
		2	1.12e+04	107.3	1.38e+04
29	L2_M_LINES_OUTM	1	1.08e+04	102.6	1.38e+04
		2	1.06e+04	101.7	1.37e+04
30	L2_RQSTS_MESI	1	1.42e+05	40.2	1.67e+05
$\Delta t = 0.04sec$ for every PMC reading.					continued on next page

$\Delta t = 0.04sec$ for every PMC reading.					Continued from previous page
#	**Symbol**	**CPU**	σ_i	β_i	**Mean**
		2	1.46e+05	41.0	1.72e+05
31	L2_RQSTS_M_STATE	1	2.22e+04	36.9	2.53e+04
		2	2.23e+04	34.7	2.51e+04
32	L2_RQSTS_E_STATE	1	1.76e+04	29.1	1.83e+04
		2	1.81e+04	33.0	1.95e+04
33	L2_RQSTS_S_STATE	1	9.60e+04	35.1	1.08e+05
		2	9.56e+04	35.9	1.07e+05
34	L2_RQSTS_I_STATE	1	1.88e+04	100.0	2.32e+04
		2	1.88e+04	100.5	2.33e+04
35	L2_ADS	1	2.48e+05	50.0	3.07e+05
		2	2.54e+05	51.0	3.22e+05
36	L2_DBUS_BUSY	1	2.03e+05	49.7	2.54e+05
		2	1.80e+05	41.7	2.14e+05
37	L2_DBUS_BUSY_RD	1	1.62e+05	41.8	1.99e+05
		2	1.56e+05	44.2	1.88e+05
External Bus Logic (EBL)					
38	BUS_DRDY_CLOCKS_SELF	1	4.34e+04	103.3	5.60e+04
		2	4.26e+04	107.0	5.54e+04
39	BUS_DRDY_CLOCKS_ANY	1	1.58e+06	101.8	2.09e+06
		2	1.57e+06	101.7	2.09e+06
40	BUS_LOCK_CLOCKS_SELF	1	No Data		
		2			
41	BUS_LOCK_CLOCKS_ANY	1	No Data		
		2			
42	BUS_REQ_OUTSTANDING	1	1.68e+06	97.9	2.14e+06
		2	1.65e+06	89.3	2.11e+06
43	BUS_TRAN_BRD_SELF	1	1.05e+04	120.7	1.38e+04
		2	1.05e+04	113.3	1.38e+04
44	BUS_TRAN_BRD_ANY	1	2.02e+04	95.0	2.77e+04
		2	2.02e+04	95.0	2.77e+04
45	BUS_TRAN_RFO_SELF	1	8572	77.0	9759
		2	9004	80.4	1.02e+04
46	BUS_TRAN_RFO_ANY	1	1.52e+04	112.0	1.92e+04
		2	1.52e+04	112.0	1.92e+04
$\Delta t = 0.04sec$ for every PMC reading.					continued on next page

$\Delta t = 0.04sec$ for every PMC reading.					Continued from previous page
#	**Symbol**	**CPU**	σ_i	β_i	**Mean**
47	BUS_TRANS_WB_SELF	1	8276	73.9	9399
		2	8246	70.6	9290
48	BUS_TRANS_WB_ANY	1	1.39e+04	110.7	1.76e+04
		2	1.39e+04	110.7	1.76e+04
49	BUS_TRAN_IFETCH_SELF	1	2544	101.0	3374
		2	2528	121.1	3354
50	BUS_TRAN_IFETCH_ANY	1	4645	108.6	6645
		2	4644	108.7	6645
51	BUS_TRAN_INVAL_SELF	1	3553	39.8	4095
		2	3542	37.0	4052
52	BUS_TRAN_INVAL_ANY	1	5657	81.2	7838
		2	5657	81.2	7838
53	BUS_TRAN_PWR_SELF	1	153.0	50.9	197.4
		2	151.5	46.6	194.2
54	BUS_TRAN_PWR_ANY	1	268.1	82.7	375.8
		2	268.1	82.8	375.8
55	BUS_TRANS_P_SELF	1	1287	53.4	1679
		2	1176	49.5	1548
56	BUS_TRANS_P_ANY	1	7301	81.0	1.03e+04
		2	7301	81.0	1.03e+04
57	BUS_TRANS_IO_SELF	1	1404	91.07	299.7
		2	2054	90.24	318.8
58	BUS_TRANS_IO_ANY	1	1468	153.4	376.2
		2	1488	115.0	375.1
59	BUS_TRAN_DEF_SELF	1	No Data		
		2			
60	BUS_TRAN_DEF_ANY	1	nan	nan	nan
		2	nan	nan	nan
61	BUS_TRAN_BURST_SELF	1	2.80e+04	98.8	3.39e+04
		2	2.58e+04	95.0	3.16e+04
62	BUS_TRAN_BURST_ANY	1	4.93e+04	104.5	6.45e+04
		2	4.93e+04	104.5	6.45e+04
63	BUS_TRAN_ANY_SELF	1	3.00e+04	106.5	3.81e+04
		2	2.95e+04	102.2	3.76e+04
$\Delta t = 0.04sec$ for every PMC reading.					continued on next page

$\Delta t = 0.04sec$ for every PMC reading.					Continued from previous page
#	**Symbol**	**CPU**	σ_i	β_i	**Mean**
64	BUS_TRAN_ANY_ANY	1	6.13e+04	94.9	8.31e+04
		2	6.13e+04	94.9	8.31e+04
65	BUS_TRAN_MEM_SELF	1	3.10e+04	110.8	3.87e+04
		2	2.82e+04	97.9	3.63e+04
66	BUS_TRAN_MEM_ANY	1	6.14e+04	94.5	8.24e+04
		2	6.14e+04	94.5	8.24e+04
67	BUS_DATA_RCV	1	7.29e+04	100.8	9.17e+04
		2	7.84e+04	111.3	9.72e+04
68	BUS_BNR_DRV	1	1196	46.0	1377
		2	1185	45.2	1356
69	BUS_HIT_DRV	1	1599	87.9	2157
		2	1713	97.1	2297
70	BUS_HITM_DRV	1	6277	36.6	7167
		2	6545	41.1	7759
71	BUS_SNOOP_STALL	1	63.45	62.92	65.33
		2	62.56	65.85	64.39
Floating Point Unit					
72	FLOPS	1	1.24e+05	78.6	8.07e+04
		2	1.27e+05	77.0	8.25e+04
73	FP_COMP_OPS_EXE	1	1.40e+05	65.5	1.09e+05
		2	1.39e+05	66.0	1.08e+05
74	FP_ASSIST	1	No Data		
		2			
75	MUL	1	No Data		
		2			
76	DVI	1	No Data		
		2			
77	CYCLE_DIV_BUSY	1	9.18e+05	71.8	1.04e+06
		2	9.52e+05	84.2	1.17e+06
Memory Ordering					
78	LD_BLOCKS	1	1.22e+06	32.4	1.29e+06
		2	1.42e+06	39.7	1.64e+06
79	SB_DRAINS	1	414.5	43.4	482.9
		2	426.6	44.9	500.0
$\Delta t = 0.04sec$ for every PMC reading.					continued on next page

$\Delta t = 0.04sec$ for every PMC reading.					Continued from previous page
#	**Symbol**	**CPU**	σ_i	β_i	**Mean**
80	MISALIGN_MEMREF	1	7382	34.4	8216
		2	7142	34.9	7973
81	EMON_KNI_PREF_DISPATCHED_NTA	1	No Data		
		2			
82	EMON_KNI_PREF_DISPATCHED_T1	1	No Data		
		2			
83	EMON_KNI_PREF_DISPATCHED_T2	1	No Data		
		2			
84	EMON_KNI_PREF_DISPATCHED_WEAKLY	1	No Data		
		2			
85	EMON_KNI_PREF_MISS_NTA	1	No Data		
		2			
86	EMON_KNI_PREF_MISS_T1	1	No Data		
		2			
87	EMON_KNI_PREF_MISS_T2	1	No Data		
		2			
88	EMON_KNI_PREF_MISS_WEAKLY	1	No Data		
		2			
Instruction Decoding and Retirement					
89	INST_RETIRED	1	7.25e+06	40.0	8.71e+06
		2	6.43e+06	32.9	7.04e+06
90	UOPS_RETIRED	1	1.03e+07	41.9	1.24e+07
		2	9.83e+06	40.2	1.16e+07
91	INST_DECODED	1	7.10e+06	39.4	8.45e+06
		2	7.06e+06	36.6	8.15e+06
92	EMON_KNI_INST_RETIRED_PACK_SCA	1	1.503	17.98	1.862
		2	1.527	20.18	2.01
93	EMON_KNI_INST_RETIRED_SCA	1	No Data		
		2			
94	EMON_KNI_COMP_INST_RET_PACK_SCA	1	No Data		
		2			
95	EMON_KNI_COMP_INST_RET_SCA	1	No Data		
		2			
Interrupts					
$\Delta t = 0.04sec$ for every PMC reading.					continued on next page

$\Delta t = 0.04sec$ for every PMC reading.					Continued from previous page
#	**Symbol**	**CPU**	σ_i	β_i	**Mean**
96	HW_INT_RX	1	40.96	79.7	66.3
		2	40.89	79.9	66.25
97	CYCLES_INT_MASKED	1	2.88e+05	65.5	4.09e+05
		2	2.69e+05	60.6	3.88e+05
98	CYCLES_INT_PENDING_AND_MASKED	1	2.09e+04	141.7	4609
		2	2.22e+04	141.3	3959
Branches					
99	BR_INST_RETIRED	1	No Data		
		2			
100	BR_MISS_PRE_RETIRED	1	1.89e+04	35.9	2.20e+04
		2	1.90e+04	36.1	2.22e+04
101	BR_TAKEN_RETIRED	1	No Data		
		2			
102	BR_MISS_PRED_TAKEN_RET	1	1.53e+04	37.2	1.80e+04
		2	1.55e+04	38.1	1.86e+04
103	BR_INST_DECODED	1	nan	inf	3.103e+05
		2	nan	inf	3.096e+05
104	BTB_MISSES	1	8.64e+04	44.2	1.08e+05
		2	7.65e+04	39.6	9.18e+04
105	BR_BOGUS	1	1170	30.66	1288
		2	1170	29.73	1275
106	BACLEARS	1	2.84e+04	28.9	2.98e+04
		2	3.13e+04	33.6	3.57e+04
Stalls					
107	RESOURCE_STALLS	1	2.767e+06	64.45	3.569e+06
		2	2.791e+06	66.99	3.671e+06
108	PARTIAL_RAT_STALLS	1	No Data		
		2			
Segment Register Loads					
109	SEGMENT_REG_LOADS	1	2433	37.26	2964
		2	2387	37.74	2937
Clocks					
110	CPU_CLK_UNHALTED	1	7.45e+06	59.9	9.85e+06
		2	8.17e+06	59.4	1.10e+07
$\Delta t = 0.04sec$ for every PMC reading.					continued on next page

$\Delta t = 0.04sec$ for every PMC reading.					Continued from previous page
#	**Symbol**	**CPU**	σ_i	β_i	**Mean**
MMX UNIT					
111	MMX_SAT_INSTR_EXEC	1 2	No Data		
112	MMX_UOPS_EXEC	1 2	No Data		
113	MMX_INSTR_TYPE_EXEC_MULTIPLY	1 2	No Data		
114	MMX_INSTR_TYPE_EXEC_SHIFT	1 2	No Data		
115	MMX_INSTR_TYPE_EXEC_PACK	1 2	No Data		
116	MMX_INSTR_TYPE_EXEC_UNPACK	1 2	No Data		
117	MMX_INSTR_TYPE_EXEC_LOGICAL	1 2	No Data		
118	MMX_INSTR_TYPE_EXEC_ARITHMETIC	1 2	No Data		
119	FP_MMX_TRANS_MMX_TO_FP	1 2	No Data		
120	FP_MMX_TRANS_FP_TO_MMX	1 2	No Data		
121	MMX_ASSIST	1 2	No Data		
Segment Register Renaming					
122	SEG_RENAME_STALLS_ES	1 2	No Data		
123	SEG_RENAME_STALLS_DS	1 2	No Data		
124	SEG_RENAME_STALLS_FS	1 2	No Data		
125	SEG_RENAME_STALLS_GS	1 2	No Data		
126	SEG_RENAME_STALLS_ALL	1 2	No Data		
$\Delta t = 0.04sec$ for every PMC reading.					continued on next page

$\Delta t = 0.04sec$ for every PMC reading.					Continued from previous page
#	**Symbol**	**CPU**	σ_i	β_i	**Mean**
127	SEG_REG_RENAMES_ES	1	265.3	44.7	351.9
		2	274.8	46.8	368.9
128	SEG_REG_RENAMES_DS	1	271	42.8	354.6
		2	275	45.1	363.4
129	SEG_REG_RENAMES_FS	1	9.516	60.3	13.72
		2	8.712	55.8	12.59
130	SEG_REG_RENAMES_GS	1	8.826	52.93	12.41
		2	9.641	52.41	13.32
131	SEG_REG_RENAMES_ALL	1	564.2	44.7	744.4
		2	563.4	43.7	739.5
132	RET_SEG_RENAMES	1	No Data		
		2			

Table B.1: PIII Performance Monitoring Counter (PMC) off-line autocorrelation analysis results. These measurements were taken on the SMP system specified in table 7.1. Some of the PMC off-line autocorrelation results are labeled as *NO Data* this may have one of the following reasons: 1. The application that run during PMC acquisition did not generate event for the selected event. For example SIMD and MMX related events. 2. The event was not read on the correct counter. For example, floating point unit multiply events (event MUL) can only be counted with the second counter. 3. The autocorrelation results were not suitable for the automatic analysis (Octave script). 4. Event data were not taken for a specific event during the automated data acquisition process.

Bibliography

[AGE05] EUROPEAN SPACE AGENCY. //www.estec.esa.nl/wsmwww/leon/, 2005.

[AM05] Ulf Andersson and Philip Mucci. Analysis and optimization of yee_bench using hardware performance counters. In *Proceedings of Parallel Computing 2005 (ParCo)*, September 2005.

[ASW05] Reza Azimi, Michael Stumm, and Robert W. Wisniewski. Online performance analysis by statistical sampling of microprocessor performance counters. In *ICS*, pages 101–110, 2005.

[AW06] Hyo-Sung Ahn and Chang-Hee Won. Fast alignment using rotation vector and adaptive kalman filter. *IEEE Transactions on Aerospace and Electronic Systems*, 42(1):70–83, January 2006.

[BABD00] Rajeev Balasubramonian, David H. Albonesi, Alper Buyuktosunoglu, and Sandhya Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *MICRO*, pages 245–257, 2000.

[Bac00] Eric Robert Bachmann. *Inertial and Magnetic Tracking of Limb Segment Orientation for Inserting Humans into Synthetic Environments.* PhD thesis, Naval Postgraduate School, Monterrey, California, dec 2000.

[BDA03] Rajeev Balasubramonian, Sandhya Dwarkadas, and David H. Albonesi. Dynamically managing the communication-parallelism trade-off in future clustered processors. In *ISCA*, pages 275–286, 2003.

[Bel00] Frank Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop*, Kolding, Denmark, September 17–20 2000.

[BH97] Robert Grover Brown and Patrick Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering, Third Edition.* John Wiley Sons, 1997. ISBN 0-471-12839-2.

[BK+05] Rob Byrom, Stuart Kenny, et al. *SANTA-G: and instrument monitoring framework using the Relational Grid Monitoring Architecture (R-GMA)*, chapter CrossGrid. LNCS. 2005.

[BM01] R. Iris Bahar and Srilatha Manne. Power and energy reduction via pipeline balancing. In *ISCA*, pages 218–229, 2001.

[BM03] Ross Brennan and Michael Manzke. On the introduction of reconfigurable hardware into computer architecture education. In *Workshop on Computer Architecture Education*, pages 96–102, June 2003.

[BYT06] BYTE. Bytemark. //www.byte.com/bmark/bdoc.htm, 2006.

[Cel06] Celestia. //www.shatters.net/celestia/, 2006.

[CFJ03] Christophe Cérin, Hazem Fkaier, and Mohamed Jemni. Accessing hardware performance counters in order to measure the influence of cache on the performance of integer sorting. In *IPDPS*, page 274, 2003.

[cit06] Citeseer. //citeseer.ist.psu.edu/cs, 2006.

[CM99] Brian Coghlan and Michael Manzke. Prototype trace probe and probe adapter. Technical Manual TCD-CS-2000-32, Trinity College Dublin, Department of Computer Science, O'Reilly Institude, Trinty College, Dublin 2, Ireland, apr 1999. http://www.cs.tcd.ie/Brian.Coghlan/scieuro.htm.

[CM05a] Eli D. Collins and Barton P. Miller. A loop-aware search strategy for automated performance analysis. In *HPCC*, pages 573–584, 2005.

[CM05b] G. Contreras and M. Martonosi. Power prediction for intel xscale processors using performance monitoring unit events. In *ISLPED '05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, pages 221–226. IEEE, August 2005.

[CMB+98] Brian Coghlan, Michael Manzke, Erich Barnstedt, Ronon Cunniffe, and Jonathan Dukes. Deep trace dt200.1, prototype tracer. Technical Manual TCD-CS-2000-34, Trinity College Dublin, Department of Computer Science, O'Reilly Institude, Trinty College, Dublin 2, Ireland, oct 1998. http://www.cs.tcd.ie/Brian.Coghlan/scieuro.htm.

[CMW00] Harold W. Cain, Barton P. Miller, and Brian J. N. Wylie. A callgraph-based search strategy for automated performance diagnosis (distinguished paper). In *Euro-Par*, pages 108–122, 2000.

[Com06] Compendex and inspec database. //www.engineeringvillage2.org/controller/servlet/Controll 2006.

[Cro06] Crossgrid. //www.eu-crossgrid.org, 2006.

[CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetric progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[CWS97] Guanrong Chen, Jianrong Wang, and Leang S.Shieh. Interval kalman filtering. *IEEE Transactions on Aerospace and Electronic Systems*, 33(1):250–259, January 1997.

[DBL]

[DCS03] E. Duesterwald, C. Cascaval, and D. Sandhya. Characterizing and predicting program behavior and its variability. In *Proceedings 12th International Conference on Parallel Architectures and Compilation Techniques - PACT 2003*, pages 220–231. IEEE Comput. Soc, 2003.

[Dee06] Crescent bay software's deep. //www.crescentbaysoftware.com/deep.html, 2006.

[DGBP05] David E. DeMarle, Chistiaan Gribble, Solomon Boulos, and Steven Parker. Memory sharing for interactive ray tracing on clusters. *Journal of Parallel and Distributed Computing*, 2005. to appear.

[DGP04] David E. DeMarle, Christiaan Gribble, and Steven Parker. Memory-savvy distributed interactive ray tracing. In *Eurographics Symposium on Parallel Graphics and Vsualization*, 2004.

[DLM+01] Jack Dongarra, Kevin London, Shirley Moore, Phil Mucci, and Dan Terpstra. Using papi for hardware performance monitoring on linux systems. In *Conference Proceedings of Linux Clusters: The HPC Revolution*, Urbana, Illinois, jun 2001.

[DLM+03] Jack Dongarra, Kevin S. London, Shirley Moore, Philip Mucci, Daniel Terpstra, Haihang You, and Min Zhou. Experiences and lessons learned with a portable interface to hardware performance counters. In *IPDPS*, page 289, 2003.

[DMM+03] Jack Dongarra, Allen D. Malony, Shirley Moore, Philip Mucci, and Sameer Shende. Performance instrumentation and measurement for terascale systems. In *International Conference on Computational Science*, pages 53–62, 2003.

[DMM+04] Jack Dongarra, Shirley Moore, Philip Mucci, Keith Seymour, and Haihang You. Accurate cache and tlb characterization using hardware counters. In *International Conference on Computational Science*, pages 432–439, 2004.

[Dol96] Dolphin Interconnect Solutions AS, Olaf Helsets vei 6, Bogerud, N-0621 Oslo, Norway. *A Backside Link (Blink) for Scalable Coherent Interface (SCI) nodes*, draft 2.41 edition, may 1996.

[Dol04] Dolphin. //www.dolphinics.com, 2004.

[Dol05] Dolphin. //www.dolphinics.com/products/hardware/lc3.html, 2005.

[DPH+03] David E. DeMarle, Steven Parker, Mark Hartner, Christiaan Gribble, and Charles D. Hansen. Distributed interactive ray tracing for large volume visualization. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 87–94, 2003.

[DS02] Ashutosh Dhodapkar and James E. Smith. Managing multi-configuration hardware via dynamic working set analysis. In *ISCA*, pages 233–, 2002.

[EIH00] Matthew Eldridge, Homan Igehy, and Pat Hanrahan. Pomegranate: a fully scalable graphics architecture. In *SIGGRAPH*, pages 443–454, 2000.

[FCJV97] Keith I. Farkas, Paul Chow, Norman P. Jouppi, and Zvonko G. Vranesic. The multicluster architecture: Reducing cycle time through partitioning. In *MICRO*, pages 149–159, 1997.

[FCL02] Felix Freitag, Jordi Caubet, and Jesús Labarta. On the scalability of tracing mechanisms. In *Euro-Par*, pages 97–104, 2002.

[FG01] Daniele Folegnani and Antonio González. Energy-effective issue logic. In *ISCA*, pages 230–239, 2001.

[For01] Force Computers. *PENT/CPCI-735/736 Family Installation Guide*, p/n 215889 revission aa edition, October 2001.

[GA01] Mohinder S. Grewal and Angus P. Andrews. *Kalman Filterring Theory and Proctice using MATLAB*. John Wiley Sons, second edition edition, 2001. ISBN 0-471-39254-5.

[GAB+99] F. Giacomini, T. Amundsen, A. Bogaerts, R. Hauser, B.D. Johnsen, H. Kohmann, R. Nordstrom, and P. Werner. *Low-level SCI software functional specification*. Esprit Project 23174, version 2.1.1 edition, March 1999.

[Gig06] Gigabit ethernet. //www.gigabit-ethernet.org/, 2006.

[GM98] Jim Galarowicz and Bernd Mohr. Analyzing message passing programs on the cray t3e with pat and vampir, 1998.

[GPH04] Chistiaan Gribble, Steven Parker, and Charles Hansen. A preliminary evaluation of the silicon graphics onyx4 ultimatevision visualization system for large-scale parallel volmue rendering. Technical report, University of Utah, School of Computing, January 2004.

[Gre96] Welch Gregory, Francis. *SCAAT: Incremental Tracking with Incomplete Information*. PhD thesis, Department of Computer Science, CB 3175, Sitterson Hall, UNC-Chapel Hill, oct 1996.

[Gri06] Grid-ireland. //www.grid.ie, 2006.

[GST+02] G. Torralbaand V. Gonzlez, E. Sanchis, J. Tao, M. Schulz, and W. Karl. Data monitoring in high-performance clusters for computing applications. *IEEE Transactions on Nuclear Science*, 49(2), April 2002.

[GTAB01] Jordi Guitart, Jordi Torres, Eduard Ayguadé, and J. Mark Bull. Performance analysis tools for parallel java applications on shared-memory systems. In *ICPP*, pages 357–364, 2001.

[HCC98] L. Hong, G. Cheng, and C. K. Chui. A filter-bank-based kalman filtering technique for wavelet estimation and decomposition of random signals. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(2):237–241, February 1998.

[HEB+01] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordon Stoll, Matthew Everett, and Pat Hanrahan. Wiregl: a scalable graphics system for clusters. In *SIGGRAPH*, pages 129–140, 2001.

[HHN+02] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *SIGGRAPH*, pages 693–702, 2002.

[HJK+00] J. Hockauf, J. Jeitner, W. Karl, R. Lindhof, M. Schulz, V. Gonzales, E. Sanquis, and G. Torralba. Design and implementation aspects for the smile hardware monitor. In *Scalable Coherent Interface - Conference Proceedings of SCI Europe 2000 3rd International Conference on SCI-based Technology and Research*. SINTEF Electronics and Cybernetics, August 2000.

[HLM95] J. K. Hollingsworth, J. E. Lumpp, and B. P. Miller. Techniques for performance measurement of parallel programs. *arallel Computers: Theory and Practice*, pages 225–240, 1995.

[HM93] Jeffrey K. Hollingsworth and Barton P. Miller. Dynamic control of performance monitoring on large scale parallel systems. In *International Conference on Supercomputing*, pages 185–194, 1993.

[HM96] Jeffrey K. Hollingsworth and Barton P. Miller. An adaptive cost system for parallel program instrumentation. In *Euro-Par, Vol. I*, pages 88–97, 1996.

[HM05] Laune C. Harris and Barton P. Miller. Practical analysis of stripped binary code. In *Workshop on Binary Instrumentation and Applications (WBIA-05)*, September 2005.

[HMC94] Jeffrey K. Hollingsworth, Barton P. Miller, and Jon Cargille. Dynamic program instrumentation for scalable performance tools. In *Scalable High-performance Conputing Conference (SHPCC)*, May 1994.

[HMG+97] Jeffrey K. Hollingsworth, Barton P. Miller, M. J. R. Goncalves, Oscar Naim, Zhichen Xu, and Ling Zheng. Mdl: A language and compiler for dynamic program instrumentation. In *IEEE PACT*, pages 201–, 1997.

[HR99] Hermann Hellwagner and Alexander Reinefeld, editors. *SCI: Scalable Coherent Interface, Architecture and Software for High-Performance Compute Clusters*, volume 1734 of *Lecture Notes in Computer Science*. Springer, 1999.

[HRT03] Michael C. Huang, Jose Renau, and Josep Torrellas. Positional adaptation of processors: Application to energy reduction. In *ISCA*, pages 157–168, 2003.

[HSA01] Christopher J. Hughes, Jayanth Srinivasan, and Sarita V. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *MICRO*, pages 250–261, 2001.

[IM94] R. Bruce Irvin and Barton P. Miller. A performance tool for high-level parallel programming languages. In *IFIP WG10.3 Working Conference on Programming Environments for Massively Parallel Distributed Systems*, April 1994.

[IM96a] R. Bruce Irvin and Barton P. Miller. Mapping performance data for high-level and data views of parallel program performance. In *International Conference on Supercomputing*, pages 69–77, 1996.

[IM96b] R. Bruce Irvin and Barton P. Miller. Mechanisms for mapping high-level parallel performance data. In *ICPP Workshop*, pages 10–19, 1996.

[IM03] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: methodology and empirical data. In *6th International Symposium on Microarchitecture*, pages 93–104. IEEE Comput. Soc, December 2003.

[Inf00a] InfiniBand Trade Association. *InfiniBand ™Architecture Specification - General Specification*, oct 2000. Volume 1.

[Inf00b] InfiniBand Trade Association. *InfiniBand^{TM} Architecture Specification - Physical Specification*, oct 2000. Volume 2.

[Inf06] Infiniband. //www.infinibandta.org/home, 2006.

[Ins93] The Institude of Electrical and Electronics Engineers, Inc. *IEEE Standard for Scalable Coherent Interface (SCI) 1596-1992*, 345 east 47th street, new york, ny 10017-2394, usa edition, 1993. ISBN 1-55937-222-2.

[int06] int.eu.grid. //grid.ifca.unican.es/int.eu.grid, 2006.

[JJL+03] Gabriele Jost, Haoqiang Jin, Jesús Labarta, Judit Gimenez, and Jordi Caubet. Performance analysis of multilevel parallel applications on shared memory architectures. In *IPDPS*, page 80, 2003.

[JJLG03] Gabriele Jost, Haoqiang Jin, Jesús Labarta, and Judit Gimenez. Interfacing computer aided parallelization and performance analysis. In *International Conference on Computational Science*, pages 181–190, 2003.

[Kal60] Emil Kalman, Rudolph. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[KC04] Stuart Kenny and Brian A. Coghlan. Grid-wide intrusion detection system. In Marian Bubak, Michal Turala, and Kazimierz Wiatr, editors, *Proc. Cracow Grid Workshop (CGW04)*, Cracow, Poland, December 2004. Academic Computer Centre CYFRONET AGH.

[KC05] S. Kenny and B.A. Coghlan. Towards a grid-wide intrusion detection system. In Peter M.A. Sloot, Alfons G. Hoekstra, Thierry Priol, Alexander Reinefeld, and Marian Bubak, editors, *Advances in Grid Computing - EGC 2005*, LNCS3470, Amsterdam, The Netherlands, February 2005. Springer.

[KCB+05] Stuart Kenny, Brian Coghlan, Rob Byrom, Andrew Cooke, Roney Cordensoni, Linda Cornwall, Ari Datta, Abdeslem Djaoui, Laurence Field, Steve Fisher, Stuart Kenny, James Magowan, Werner Nutt, Manfred Oevers, David O'Callaghan, Norbert Podhorski, John Ryan, Manish Soni, Paul Taylor, Antony Wilson, and Xiaomei Zhu. The CanonicalProducer: an instrument monitoring component of the Relational Grid Monitoring Architecture. *Scientific Programming*, 13(2):151–158, 2005.

[KCK+01] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, and A. Sivasubramaniam. vec: Virtual energy counters. In *2001 ACM SIGPLAN -*

SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, pages 28–31. ACM, June 2001.

[KDH+95] Rainer Klar, Peter Dauphin, Franz Hartleb, Richard Hofmann, Bernd Mohr, Andreas Quick, and Markus Siegle. *Messung und Modellierung paraller und verteilter Rechnersysteme*. B.G Teubner Stuttgard, 1995. ISBN 3-519-02144-7.

[Ken06] Stuart Kenny. *A Framework for Instrument Monitoring on the Grid*. PhD thesis, Department of Computer Science, Tinity College Dublin, Ireland, 2006.

[Kes91] Srinivasan Keshav. A control-theoretic approach to flow control. In *SIGCOMM*, pages 3–15, 1991.

[KHJ+96] Chung-Ming Kuo, Chaur-Heh Hsieh, Yue-Dar Jou, Hsieh-Cheng Lin, and Po-Chiang Lu. Motion estimation for video compression using kalman filtering. *IEEE Transactions on Broadcasting*, 42(2):110–116, June 1996.

[KL97] W. Karl and M. Leberecht. Ein monitorkonzept fr systeme mit verteiltem gemeinsamen speichern. In *ARCS'97: Architektur von Rechensystemen*, sep 1997.

[KLS99] Wolfgang Karl, Markus Leberecht, and Martin Schulz. Optimizing data locality for sci-based pc-clusters with the smile monitoring approach. In *IEEE PACT*, pages 169–176, 1999.

[KMLM97] Karen L. Karavanic, Jussi Myllymaki, Miron Livny, and Barton P. Miller. Integrated visualization of parallel program performance data. *Parallel Computing*, 23(1-2):181–198, 1997.

[KST00] Wolfgang Karl, Martin Schulz, and Jörg Trinitis. Multilayer online-monitoring for hybrid dsm systems on top of pc clusters with a smile. In *Computer Performance Evaluation / TOOLS*, pages 294–308, 2000.

[LCM+00] Kathleen A. Lindlan, Janice E. Cuny, Allen D. Malony, Sameer Shende, Bernd Mohr, Reid Rivenburgh, and Craig Edward Rasmussen. A tool framework for static and dynamic analysis of object-oriented software with templates. In *SC*, 2000.

[Lim99] ARM Limited. *AMBA Specification, Rev. 2.0*, May 1999.

[LJ03] T. Li and L.K. John. Run-time modeling and estimation of operating system power consumption. In *International Conference on Measurement and Modeling of Computer Systems ACM SIGMETRICS 2003*, pages 160–171. ACM, June 2003.

[LJI+98] Cheng Liao, Dongming Jiang, Liviu Iftode, Margaret Martonosi, and Douglas W. Clark. Monitoring shared virtual memory performance on a myrinet-based pc cluster. In *International Conference on Supercomputing*, pages 251–258, 1998.

[Mar01] R. Josephand M. Martonosi. Run-time power estimation in high performance microprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design, Digest of Technical Papers*, pages 135–140, August 2001.

[May01] John M. May. Mpx: Software for multiplexing hardware performance counters in multithreaded programs. In *IPDPS*, page 22, 2001.

[MB04] Michael Manzke and Ross Brennan. Extending fpga based teaching boards into the area of distributed memory multiprocessors. In *Workshop on Computer Architecture Education*, pages 15–21, June 2004.

[MBO+06] Michael Manzke, Ross Brennan, Keith O'Conor, John Dingliana, and Carol O'Sullivan. A scalable and reconfigurable shared-memory graphics architecture. In *Proceedings of the SIGGRAPH 2006 Conference on Sketches & Applications*, 2006.

[MC99a] Michael Manzke and Brian Coghlan. Non-intrusive deep tracing of sci interconnect traffic. In Wolfgang Karl and Geir Horn, editors, *SCI Europe '99*, pages 53–58. SINTEF Electronics and Cybernetics, September 1999. ISBN 82-14-00014-9.

[MC99b] Michael Manzke and Brian Coghlan. Prototype trace software. Technical Manual TCD-CS-2000-28, Trinity College Dublin, Department of Computer Science, O'Reilly Institude, Trinty College, Dublin 2, Ireland, apr 1999. http://www.cs.tcd.ie/Brian.Coghlan/scieuro.htm.

[MC05a] Michael Manzke and Brian A. Coghlan. Optimal performance state estimation of compute systems. In *the Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005)*, pages 511–516, September 2005.

[MC05b] Wiplove Mathur and Jeanine Cook. Improved estimation for software multiplexing of performance counters. In *the Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005)*, pages 23–34, 2005.

[MCC+95] Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce Irvin, Karen L. Karavanic, Krishna Kunchithapadam,

and Tia Newhall. The paradyn parallel performance measurement tool. *IEEE Computer*, 28(11):37–46, 1995.

[MCEF94] Steven Molner, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. *Computer Graphics and Applications, IEEE*, 14(4):23–32, July 1994.

[MCI+01] Barton P. Miller, Mihai Christodorescu, Robert Iverson, Tevfik Kosar, Alexander Mirgorodskii, and Florentina I. Popovici. Playing inside the black box: Using dynamic instrumentation to create security holes. *Parallel Processing Letters*, 11(2/3):267–280, 2001.

[McM94] J.Chris McMillan. A gps attitude error model for kalman filtering. In *IEEE 1994 Position Location and Navigation Symposium*, pages 329–336, 1994.

[MCM96] M. Martonosi, D. Clark, and M. Mesarina. The shrimp performance monitor: Design and applications. In *ACM SIGMETRICS Symposium on Parallel and Distributed Tools*, May 1996.

[MDK+04] P. Mucci, J. Dongarra, R. Kufrin, S. Moore, F. Song, and F. Wolf. Automating the large-scale collection and analysis of performance. In *Proceedings of the 5th LCI International Conference on Linux Clusters: The HPC Revolution*, May 2004.

[Mes03] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interfacean*, November 2003.

[MIL97] MIL3,Inc.3400 International Drive NW,Washington DC20008 USA. *OPNET Modeler*, 1997.

[MKCL01] Michael Manzke, Stuart Kenny, Brian Coghlan, and Olav Lysne. Tuning and verification of simulation models for high speed interconnection. In *PDPTA'2001*, June 2001.

[MKS98] M. Meišner, U. Kanus, and W. Straßer. A pci-card for real-time volmue rendering. In *Eurographics Workshop on Graphics Hardware,*, pages 61–67, 1998.

[MM05] Alexander V. Mirgorodskiy and Barton P. Miller. Autonomous analysis of interactive systems with self-propelled instrumentation. In *12th Multimedia Computing and Networking (MMCN 2005)*, January 2005.

[Moo02] Shirley V. Moore. A comparison of counting and sampling modes of using performance monitoring hardware. In *International Conference on Computational Science (2)*, pages 904–912, 2002.

[MS88] Y. V. V. S. Murty and W. J. Smolinski. Design and implementation of a digital differential relay for a 3-phase power transformer based on kalman filtering theory. *IEEE Transactions on Power Delivery*, 3(2):525–533, April 1988.

[MSS+03] Grigorios Magklis, Michael L. Scott, Greg Semeraro, David H. Albonesi, and Steve Dropsho. Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. In *ISCA*, pages 14–25, 2003.

[MST+05] Allen D. Malony, Sameer Shende, N. Trebon, Jaideep Ray, Robert C. Armstrong, Craig Edward Rasmussen, and Matthew J. Sottile. Performance technology for parallel and distributed component software. *Concurrency - Practice and Experience*, 17(2-4):117–141, 2005.

[Myr06] //www.myri.com/. , 2006.

[MZ02] Ruiping Ma and Minglian Zhang. A new adaptive kalman filter for gps/ins integrated system. In *Proceedings of Asian Simulation Conference; System Simulation and Scientific Computing (Shanghai)*, pages 197–201, November 2002.

[New00] Dolphin New. //www.dolphinics.com/news/2000/june020-2000.html, 2000.

[NT01] Jorgen Norendal and Kurt Tjemsland. Tle version 2 description and test report. Deliverable 25257, SINTEF, March 2001. Work package: WP2.

[Ojh01] A.K. Ojha. Techniques in least-intrusive computer system performance monitoring. In *SoutheastCon 2001.*, pages 150 – 154. IEEE, April 2001.

[OMK04] Carol O'Sullivan, Michael Manzke, and Anil Kokaram. A shared-memory hybrid graphics cluster for visualisation and video processing, July 2004.

[Ope05] OpenMP Architecture Review Board. *OpenMP Application Program Interface*, May 2005.

[Opn06] Opnet. //www.opnet.com/, 2006.

[Ord01] Intel. *IA-32 Intel Architecture Software Developer's Manual*, 2001.

[PAP02] PAPI. Papi performance application programming interface. //icl.cs.utk.edu/papi/, 2002.

[Par06] Paradyn. //www.paradyn.org/, 2006.

[PAT06] Cray's pat. //www.cray.com/index.html, 2006.

[PCC92] Randy Pausch, Thomas Crea, and Matthew Conway. A literature survey for virtual environments: military flight simulator visual systems and simulator sickness. *Presence: Teleoperators and Virtual Environments*, 1(3):344–363, 1992.

[PCL06] Performance counter library (pcl). //www.fz-juelich.de/zam/PCL/, 2006.

[Pet02] Mikael Pettersson. Linux x86 performance-monitoring counters driver. //www.csd.uu.se/ mikpe/linux/perfctr/, jan 2002.

[PGEM97] J. Pearson, R. Goodall, M. Eastham, and C. MacLeod. Investigation of kalman filter divergence using robust stability techniques. In *Proceedings of the 36th IEEE Conference on Decision and Control*, volume 5, pages 4892–4893, 1997.

[PJS97] Subbarao Palacharla, Norman P. Jouppi, and James E. Smith. Complexity-effective superscalar processors. In *ISCA*, pages 206–218, 1997.

[PKG01] Dmitry Ponomarev, Gurhan Kucuk, and Kanad Ghose. Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources. In *MICRO*, pages 90–101, 2001.

[Pro06] Sgi's prodev workshop. //www.sgi.com/products/software/irix/tools/prodev.html, 2006.

[PTVF99] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*, chapter 2. Solution of Linear Algebraic Equations. Cambrige University Press, second edition edition, 1999.

[Qua06] Quadrics. //www.quadrics.com/Quadrics/QuadricsHome.nsf/DisplayPages/Homepage, 2006.

[RL99] G. Rnneberg and O. Lysne. An opnet-based simulation model of sci-nodes. In *Conference Proceedings of SCI Europe'99*, pages 101–112, Toulouse (France),, 1999.

[RM02] Philip C. Roth and Barton P. Miller. Deep start: A hybrid strategy for automated performance problem searches. In *Euro-Par*, pages 86–96, 2002.

[RM06] Philip C. Roth and Barton P. Miller. On-line automated performance diagnosis on thousands of processes. In *2006 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'06)*, March 2006.

[RTE05] RTEMS. //www.rtems.com/, 2005.

[SBM04] J. Stewart, E.P. Bennett, and L. McMillen. Pixelview: A view-independet graphics rendering architecture. In M. McCool T. Akenine-Möller, editor, *Graphics Hardware 2004*, 2004.

[SBNW98] Bernhard Skaali, Inge Birkeli, Baard Nossum, and David Wormald. Scitrac - an lsa preprocessor for sci link tracing. In Hermann Hellwagner and Alexander Reinefeld, editors, *Scaleble Coherent Interface: Technology and Application*, pages 131–136. SINTEF Electronics and Cybernetics and ESPRIT Working Group 'SCIWG' (EP22582), Cheshire Henbury, September 1998. ISBN 1-901864-02-2.

[SCI06] Sci. //www.dolphinics.com/, 2006.

[SEP+01] Gordon Stoll, Matthew Eldridge, Dan Patterson, Art Webb, Steven Berman, Richard Levy, Chris Caywood, Milton Taveira, Stephen Hunt, and Pat Hanrahan. Lightning-2: a high-performance display subsystem for pc clusters. In *SIGGRAPH*, pages 141–148, 2001.

[Sit05] Top500 Supercomputer Sites. //www.top500.org/, 2005.

[SKW+06] Dilma Da Silva, Orran Krieger, Robert W. Wisniewski, Amos Waterland, David Tam, and Andrew Baumann. K42: an infrastructure for operating system research. *ACM SIGOPS Operating Systems Review*, Volume 40(Issue 2):34 – 42, 2006.

[SLF+04] Bruno Sinopoli, LucaSchenato, Massimo Franceschetti, Kameshwar Poolla, Michael I. Jordan, and Shankar S. Sastry. Kalman filtering with intermittent observations. *IEEE Transactions on Automatic Control*, 49(9):1453–1464, Septembe 2004.

[SNBW99] Bernhard Skaali, Baard Nossum, Inge Birkeli, and David Wormald. Sciviewsci test, verification and monitoring instrument. In Wolfgang Karl and Geir Horn, editors, *Conference Proceedings of SCI Europe '99*, pages 47–52. ESPRIT Project 'SCI-Europe' (EP25257) and ESPRIT Working Group 'SCI-WG'(EP22582), SINTEF Electronics and Cybernetics, sep 1999. ISBN 82-14-00014-9.

[StGDC05] Richard M. Stallman and the GCC Developer Community. *Using the GNU Compiler Collection*. Free Software Foundation, 51 Franklin Street, Fifth Floor Boston, MA 02110-1301 USA, 2005.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[Sup01] Super Micro Computer Inc. *Super P3TDE6 User's Manual*, revision 1.0 edition, 2001.

[SW87] Jerzy Z. Sasiadek and Piotr J. Wojcik. On the application of an adaptive kalman filter for sensor signals. In *Proceedings of the 1987 American Control Conference.*, pages 859–1864, 1987.

[SWW+04] Jörgen Schmittler, Sven Woop, Daniel Wagner, Wolfgang Paul, and Philipp Slusallek. Realtime ray tracing of dynamic scence on an fpga chip. In M. McCool T. Akenine-Möller, editor, *Graphics Hardware 2004*, 2004.

[Tau06] University of oregon, los alamos national laboratory, and research centre jlich, zam's tau. //www.cs.uoregon.edu/research/tau/home.php, 2006.

[TGS+01] G. Torralba, V. Gonzles, E. Sanchis, J. Tao, M. Schulz, and W. Karl. Data monitoring in high performance clusters. In *Proceedings of the 12th IEEE International Congress on Real Time for Nuclear and Plama Sciences*, pages 90–95, June 2001.

[TK92] Edison T.S. Tse and Kenichiro Kataoka. Adaptive kalman filter approach to identifying the weights of a multi-layer neural network. In *Proceedings of the 1992 Artificial Neural Networks in Engineering, ANNIE'92*, volume 2, pages 325–330. ASME, Fairfield, NJ, USA, November 1992.

[Top04] Top500. //clusters.top500.org, 2004.

[Tot06] Etnus' totalview. //www.etnus.com/TotalView/MPI.html, 2006.

[Tra06] Intel's trace analyzer collector formaly pallas' vampir and vampirtrace. //www.intel.com/cd/software/products/asmona/eng/cluster/clustertoolkit/index.htm, 2006.

[WM03] Felix Wolf and Bernd Mohr. Hardware-counter based automatic performance analysis of parallel programs. In *PARCO*, pages 753–760, 2003.

[WMC+05] Qiang Wu, Margaret Martonosi, Douglas W. Clark, V. J. Reddi, Dan Connors, Youfeng Wu, Jin Lee, and David Brooks. A dynamic compilation framework for controlling microprocessor energy and performance. In *MICRO*, pages 271–282, 2005.

[WSS+04] Robert W. Wisniewski, Peter F. Sweeney, Kartik Sudeep, Matthias Hauswirth, Evelyn Duesterwald, Calin Cascaval, and Reza Azimi. Performance and environment monitoring for whole-system characterization and optimization. In *PAC2 (Conference on Power/Performance interaction with Architecture, Circuits, and Compilers)*, October 2004.

[XLM97] Zhichen Xu, James R. Larus, and Barton P. Miller. Shared memory performance profiling. In *PPOPP*, pages 240–251, 1997.

[XMN99] Zhichen Xu, Barton P. Miller, and Oscar Naim. Dynamic instrumentation of threaded applications. In *PPOPP*, pages 49–59, 1999.

[Xpo06] Ibm's xpofiler. //www.research.ibm.com/actc/projects/xprofiler.shtml, 2006.

[YWA05] Kent K.C. Yu, N.R. Watson, and J. Arrillaga. An adaptive kalman filter for dynamic harmonic state estimation and harmonic injection tracking. *IEEE Transactions on Power Delivery*, 20(2 II):1577–1584, April 2005.

Index