Wolfgang Gentzsch, Wolfgang J. Paul (editors)

Architecture and Performance

Dagstuhl-Seminar-Report; 1
18.6.1990 - 21.6.1990 (9025)

Workshop on Architecture and Performance

Organizers: W. Gentzsch and W. J. Paul

A good computer architecture delivers high performance at reasonable price. The field of computer architecture provides many techniques for constructing computers in order to achieve this goal. But computer architects rarely make their model of price and workload explicit; in case their performance figures come from simulations or measurements and not from analytical proofs.
Explicit models of workload are contructed in the field of benchmarking. Sophisticated methods to analyze the runtime even of very involved algorithms exist in the field of theoretical computer sience.
The organizers felt that insights into the effects of architectural changes on performance and into benchmarking could be gained, if the models of workload were explicit and if certain performance figures could be obtained in an analytical way. Therefore scientists from the following three fields had been invited: architecture, benchmarking and analysis of algorithms.
This meeting was supposed to be the very first meeting to be held in the newly opened research institute IBFI in Dagstuhl, beautifully located in the middle of nowhere in the northern Saar country. Because of delays in the renovation of the building in Dagstuhl, the meeting had to be moved to a resort Hotel nearby. Funding was provided mostly by the IBFI institute. Additional funds came from IBM.
The remote location and the apparently (but not really) absent organization of the meeting in the style of Oberwolfach was a new experience for some participants but everyone quickly learned to enjoy this.
The meeting was held in a relaxed atmosphere with plenty of time after talks. This time was filled with extraordinarily lively discussions, which went across the boundaries of the three invited fractions. Stimulating this discussion was of cource the main purpose of the meeting.
The organizers would like to express their thanks to everybody who helped to contribute to the success of this meeting.

Monday, 18 June 1990

W. J. Paul:	Welcome and Introduction
W. Gentzsch:	Overview of Benchmarks and Related Problems
R. Weicker:	A Detailed Look into the 'Stone Age': Dhrystone, Whetstone, 'Lhinstone'
N. Doduc:	SSBA, SPEC, PERFECT
W. Schönauer:	Micro–Measurements of Supercomputers
A. Bode:	Performance Measurements for Parallel Systems with Dynamic Load Balancing

Tuesday, 20 June 1990

A. v.d.Steen:	Structure of the European Standard Benchmark
D. Schmidt:	Efficiency Considerations on CPU Architectures
A. Formella:	Performance, Efficiency and Quality of Vectorprocessors
R. Klar:	Architectural Requirements for Efficient Monitoring
R. Hockney:	Benchmark Parameters and Measurements

Wednesday, 21 June 1990

D. Müller–Wichards:	Problem Size Scaling in the Presence of Parallel Overhead
H. Mierendorf:	Performance Modelling of Grid–Oriented Problems on Message Passing Systems
O. Kolp:	Performance Evaluations for Parallel Systems — A Workload Analysis
F. Meier auf der Heide:	Shared Memory Emulations and Distributed Dictionaries
J. Cuellar:	Lock Performance

Benchmarks and Related Problems

Wolfgang Gentzsch
FH Regensburg and GENIAS Software GmbH
Roentgenstr. 13
D-8402 Neutraubling

This presentation consists mainly of two parts: In the first part we give an overview of existing benchmarks to evaluate the performance of various computers, among them the Whetstone and Dhrystone, the program kernels from Lawrence Livermore, Los Alamos and NASA Laboratories, Linpack, SPEC and Perfect, ACCU and SSBA, and the GPC graphics benchmark.
The second part of the talk deals with the main problems arising with benchmarking and resulting from computer architectures, compilers and algorithms. Benchmarking detects the bottlenecks of the machine under consideration, such as memory bank conflicts, paths to memory cache misses, bus saturation, small register size, long start-up time, task creation, message passing overhead, memory contention and others. Problems with the software/algorithms are e. g. Amdahl's law, data dependencies, non-contigues data, memory access pattern, rate of flops/memory references, load balancing, synchronization and communication overhead, etc.
Finally, selected benchmark results for super- and mini-supercomputers are discussed.

A Detailed Look into the 'Stone Age' (An Overview of Common Benchmarks)

Reinhold Weiker
Siemens AG AUT E 51
Postfach 3220
D-8520 Erlangen

The three most often used benchmarks for processors performance are analyzed in detail, and comparative tables of language features are given:

- **Whetstone:** Floating-point benchmark with heavy emphasis on mathematical library subroutines, high locality (short loops), mostly global variables.
- **Linpack:** Floating-point benchmark spending most of its time in a vector add/multiply subroutine, very high code locality, data locality depending on the array size.
- **Dhrystone:** Integer benchmark modeling system programs, string operations somewhat overrepresented, balanced data types and locality.

All three benchmarks are too small to model memory access behaviour in the presence of caches, and compiler writers can be tempted to twist their compilers towards these benchmarks. More recent benchmarks suites, most notably the SPEC benchmarks, try to avoid these problems. Non-CPU influences on benchmark performance are discussed: Programming language, compiler optimization, runtime library, cache size. For 32-bit microprocessors, benchmark results obtained with the tree small benchmarks, and where available, with the SPEC benchmarks are presented and discussed.

Micro–measurements of Supercomputers

Willi Schönauer, Hartmut Häfner
Rechenzentrum der Universität Karlsruhe
Postfach 6980
D–7500 Karlsruhe 1

We are interested in the 'lost cycles' of supercomputers. They explain the often very poor performance (compared to the theoretical peak performance). We measured the IBM 3090 VF (Vector Facility) for the addition with stride 1, stride $k \geq 3$, with gather/scatter and under mask. There are significant differences for data in cache and in main storage. Only the 'micro model' makes visible all sources of the losses. The 'full model' compresses the information. The 'mean model' then replaces the step functions by an interpolating linear function. The parameters influencing the operation are visible. We define a 'waste factor' $W = \tau_{eff,mean}/\tau$ the ratio of effective mean and hardware cycle time. The reciprocal is the 'architectural efficiency' $\eta_a = 1/W$. The most important operation is the vector triad $a_i = b_i + c_i * d_i$. For this operation we obtain $\eta_a = 0.147$ for all data in main storage. Then measurements of the Siemens/Fujitsu VP 4000–EX are presented in a similar way. This is work in progress and therefore only provisional data is presented. In contrast to benchmarks the micro–measurements make visible the sources of the bottlenecks and losses. Such losses can be avoided only by increased internal parallelism in the computer, i. e. by a better architecture. As long as $\eta_a < 1$ programming of super computers is mere 'bottleneckology'.

Performance Measurement for Load Balancing Strategies in Multiprocessor Systems

Arndt Bode
Lehrstuhl für Rechnertechnik und Rechnerorganisation
Institut für Informatik
Technische Universität München
Postfach 20 24 20, Arcisstr. 21
D–8000 München 2, FRG

The paper presents performance measurement with different monitoring techniques (software monitoring, hardware monitoring, hybrid monitoring, simulation) in multiprocessor systems to be used in three areas of interest:

- Use of collected runtime data for interactive development tools (debugger, performance analyzer, visualizer) and manual performance debugging.
- Use of collected runtime data for a dynamic load balancing mechanism.
- Use for information collected when monitoring multiprocessor architectures and when implementing monitoring tools to enhance the design of components, architecture and basic software of parallel systems that will support runtime monitoring.

Scalable multiprocessor systems will only be successful, if they can be used in the sense of an universally programmable general purpose machine. This implies the virtualization of concurrency in hardware. Several schemes for virtualization, including virtual shared memory

and dynamic load balancing have been proposed for this purpose. Future microprocessor components (e. g. iWarp or Inmos H1) will support virtually completely interconnected systems by offering communication processors with line switching procedures on the same integrated circuit as the processor.
The paper presents TOPSYS (TOols for Parallel SYStems), an integrated and hierarchical tool environment for multiprocessor systems, that will be support all of the above features. TOPSYS has been implemented on an iPSC and iPSC/2 multiprocessor system. It is based on hardware, software and hybrid monitoring techniques. A number of tools have been implemented for manual performance debugging such as visualizers, performance analyzers, debuggers, synthetic load generators etc. An operating system MMK, offering a virtual common object space has been implemented on these machines. A progress migration component migrating processes on a demand based strategy has been implemented. The system moduls to evaluate runtime data for the purpose of deciding upon process migration and different algorithms for process migration are currently being in development. Analogies to operating system with load balancing for distributed systems are reviewed. First results on the overhead incurred when virtualizing parallel architectures are presented.

The Structure of the European Standard Benchmark

Aad J. van der Steen
Academic Computing Center Utrecht
Budapestlan 6
NL–3584 CD Utrecht

The European Standard Benchmark is a synthetic benchmark set which is aimed at the performance evaluation of (super)computers for scientific computation. We give the nationale for the structure of this benchmark. It constist of modules which contain programs of increasing complexity in such a way that the results obtained from programs in the lower moduls help to explain the outcomes of programs in higher modules. The complexity of the programs range from simple operations in module 1 to full application programs in module 4. By running this benchmark, it is hoped that enough insight in the character of a machine is obtained that general statements about its performance on general scientific programs can be done. Apart from discussing the structure we also present examples of recently obtained results.

Efficiency Considerations on CPU-Architectures

Dietmar Schmidt
Lehrstuhl Prof. Paul, Rechnerarchitektur und Parallele Rechner
Universität des Saarlandes
D–6600 Saarbrücken

Given a computer A, one is interested in evaluating and comparing it to other computers. From the architect's point of view, it is not interesting to know that Computer A is faster or cheaper than computer B. If B was implemented in the same technology as A, it could be as fast or cheap as A. To evaluate the architecture you need another time–cost model. We reduce the problem by considering CPU–architectures. The hardware of such architectures

may be modelled by switching circuits and evaluated by analyzing place and propagation delays of that circuits. We consider the compiler as a part of an architecture. Our aim is to say for example: Under certain technology parameters T and workload W the VAX–11 architecture is better or worse than the IBM–370 one.

Performance, Efficiency and Quality of Vectorprocessors

Arno Formella
Lehrstuhl Prof. Paul, Rechnerarchitektur und Parallele Rechner
Universität des Saarlandes
D–6600 Saarbrücken

Vectorprocessors are used to achieve high performance solving numerical problems. Performance is measured in MFlop/s (Million Floating Point Operations per Second) on the implemented algorithm. Given two different architectures (e.g. CRAY I or SPARK 2.0) how can you decide which one is more efficient (in any sense) ?
We introduce a formal model trying to answer this question. The model takes the technologie and the workload as parameters, so comparisons of architectures can be made. An architecture is discribed by its hardware, its machine language, the used compiler and the high level language. The quality (time–depending–cost–function) is defined as the quotient of performance and costs. The performance is modelled by implementing the 'livermore loops' in VectorPASCAL and measuring the computing time in gate delays. The costs are obtained by counting the gate equivalents of the hardware.
Some first results of the theory are presented, e. g. the optimal register file length for SPARK 2.0 and CRAY I. Further questions are formulated, especially such questions concerning modifications (improvements) of real architectures.

Architectural Requirements for Efficient Monitoring

Rainer Klar
Institut für Mathematische Maschinen und Datenverarbeitung VII
Universität Erlangen
Martensstr. 3
D–8520 Erlangen

As a result of long term experience in monitoring the advantages and drawbacks of hardware, software and hybrid monitoring are presented and the relevance of event–driven monitoring is described. Event–driven monitoring provides more than just one performance index. It helps to understand the how and why of the dynamic behaviour. A short introduction of the distributed hardware and hybrid monitor system ZM4 shows how applying hybrid monitoring to many distributed objects computers or to large parallel systems can lead to a comprehensive, intelligible and quantifying view. The method is recording event tokens from each processor in a local event trace adding a globally valid time stamp to each recorded event token. Eventually all local event traces are merged to a global event trace. The ZM4 evaluation environmemt SIMPLE provides a set of tools for comfortable evaluation of measured traces based on source level identifiers. SIMPLE is independent of the monitor system and evaluates arbitrarily formatted event traces.

Monitoring of already existing software is accompanied by program models, by implementation models and by monitoring models. These types of models are used to describe inherent parallelism, to predict the performance of implementations on different architectures/configurations and to define interesting points of a program as being an event in therms of monitoring.
Three examples illustrate typical problems with measurement interfaces and lead to the following requirements for efficient monitoring:

- The hardware interface should either be dedicated only to monitoring or it should be a general interface with special qualifiers for monitoring which enables fast output of event tokens.
- The hardware interface should offer problemoriented event tokens.
- Software monitoring statements which refer to the hardware interfaces should not be handled like general I/O statements but like internal register transfers.
- Software monitoring statements should be available for system and user code.

Performance Parameters and Measurements

Roger Hockney
23 Hillside Hardwick Road
Whitchurch–on–Thames
Readirg, RG 87 HL England

The observed performance of supercomputers is found to vary between 1 % to 80 % of the theoretical peak performance of their arithmetic pipelines. This can be seen in the performance achieved, for example, on the 24 Livermore Fortran Kernels. Parameters are introduced to explain this degradation of performance.
First the variation of performance with vector length is characterised by the parameter $n_{1/2}$ which gives the vector length needed to achieve half of the asymptotic performance. The concept of computational intensity (the amount of arithmetic performed per memory reference or data communication) was introduced, and the parameter $f_{1/2}$ defined to quantify the performance degradation arising from memory or communication bottlenecks. Similarly the parameter $s_{1/2}$ was introduced to quantify the degradation in parallel computer systems due to insufficient grain size, s, and the need for global synchronisation.
The techniques for measuring these parameters were discussed and measured values were given for the Cray–2, Cray X–MP, IBM 3090, IBM LCAP, Sequent Symmetry, and T 800 Transputer.
The above characterisations of performance, rely on approximately linear relation between time and a suitably defined program variable (n, f, ω_s), and each effect is defined by a pair of parameters such as $(r_\infty, n_{1/2})$. Such a two parameter characterisation of performance is also successful in representing the message transfer performance on distributed memory networks. The 'ping–pong' experiment was described for measuring these parameters on a network. Also, frequently, the performance of a complex algorithm can be approximately represented in the same way. The example was given of matrix multiplication.

Problem Size Scaling in the Presence of Parallel Overhead

Dieter Müller–Wichards
IBM Scientific Center Heidelberg
Tiergartenstr. 15
D–6900 Heidelberg

In this talk we study the performance of applications on multiprocessor systems. In particular we investigate the effect of synchronization and parallelization overhead where the fact that part of the application may be inherently sequential is taken into account. By relating our assumptions to an earlier work by Flatt and Kennedy we establish that the overhead function can be characterized using the concept of convex functions.
In order to observe a satisfactory payoff for increased processing power it is essential to increace the problem size accordingly. We discuss linear and nonlinear scaling schemes and compare the corresponding asymptotic performance behaviour. Throughout this investigation we profit from the well developed mathematical apparatus of convex functions.

Performance modelling of grid oriented algorithms on message passing systems

Hermann Mierendorf
Gesellschaft für Mathematik und Datenverarbeitung mbH
Schloß Birlinghofen
D–5205 St. Augustin 1

Prior to the realization of a computer, performance prediction by a theoretic model is of special interest. Homogeneous iterative grid oriented algorithms are considered as process systems. Fore message passing systems, a method is investigated, where the system is represented by the structure and and a relatively small number of performance parameters. Using these parameters, the time cost of basic activities can be evaluated. For an estimation of the overall runtime, sequences of basic activities are considered. If there is an isomorphic mapping of the process structure into the system structure, the set of these sequences is partially ordered with respect to the parameters and the sequence length. Only a small number of maximum elements must be combined in an model for estimating the time cost of the algorithm. The method can be extended to systems showing resource sharing by contention analysis for the last process.

Performance Evaluation for Parallel Computer Systems —A Workload Analysis

Otto Kolp
GMD
Schloß Birlinghofen
D–5205 Sankt Augustin 1

For parallel computers, performance prediction is an important and difficult task. Different methods are available. A workload analysis tool has been presented to evaluate large

parallel systems for appropriate applications. The method involves hardware, software and application aspects. Descriptions of the hardware including routing, of the application including a parallel process structure and of the mapping of the parallel process structure to the parallel processor system have to be given to the evaluation tool. The method was illustrated by examples. The hardware aspects are discussed by considering a two dimensional crossbar network. For an eight color relaxation scheme of a 3-dimensional multigrid algorithm the description of the application was shown. For a 2-dimensional process structure the mapping of the algorithm to the system has been discussed. Some results concerning performance in the terms of efficiency or 'megaflops' are given for some parallel systems as the SUPRENUM system.

Shared memory emulations and distributed dictionary

Friedhelm Meyer auf der Heide
Universität-GH Paderborn FB 17
Warburger Str. 100
D-4790 Paderborn

The aim of the talk is to give an overview of efforts in theoretical computer science to design and measure performance of shared memory emulations on networks of processors.
We assume an idealized shared memory machine with some number p of processors, where a parallel shared memory access executed synchronously by all processors needs one time unit. We want to show how to simulate such memory accesses on a network with $q \leq p$ processors, where the shared memory is distributed among the processors in some clever way. We present emulations with time delay $O(\log(p)/\log\log(p))$, if $q = p$ and $O(\log(p))$, if $q = p/\log(p)$. Both results are special cases of a design and performance analysis of a distributed dictionary on a network. Both delays become $O(\log(p))$, if a communication network is assumed and its delay is taken into account.
The $\log(p)/\log\log(p)$ delay for $q = p$ is unavoidable. Thus scalable realistic parallel machines can offer a virtual shared memory to be accessible by all processors only to the cost of a significant delay.
The second result shows how (if possible at all) a virtual shared memory can be offered to a user: She has to design algorithms using (at least) as much as $\log(p)$ times p virtual processors that means she has to put more effort in parallelizing her problem in order to use an efficient virtual shared memory.
This a joint work with Martin Dietzfelbinger, Paderborn.

Lockperformance in Betriebssystemen

Jorge Cuellar
Siemens AG
Otto-Hahn-Ring 6
8000 München 83

Im Multitasking-Betrieb eines Betriebssystems werden Betriebsmittel wie Prozessoren, Arbeitsspeicher oder auch Verwaltungsdaten nach Anforderung verteilt. Dies erfordert Koordinierung und Serialisierung in kritischen Pfaden. Die Realisierung erfolgt über Lockmechanismen.

Um den Einfluß der Locks auf das Systemverhalten besser zu verstehen und die Lockverluste quantitativ zu bestimmen, wird Modellierung eingesetzt. Die neuen Erkenntnisse über das Lockverhalten wurden am BS 2000 verifiziert und entsprechende Verbesserungen implementiert.
Im Nukleus eines Betriebssystems kommt es häufig vor, daß ein Prozessor bei Anforderung eines Betriebsmittels (z. B. eine zentrale Tabelle für Task- oder Speicherverwaltung) dieses auch unbedingt braucht, um weitere produktive Arbeit leisten zu können. Aus diesem Grunde, oder einfach weil die Kosten eines Kontextswitches teuerer als das Warten auf die Betriebsmittelfreigabe wären, sind in vielen Multiprozessorsystemen die meisten Nukleuslocks Spinlocks.
Es wird gezeigt, daß die Varianz neben der Lockwahrscheinlichkeit einen deutlichen Einfluß auf die Lockverluste hat.
Während bei Spinlocks in einer Schleife auf Lockfreigabe gewartet wird, verliert bei Suspendlocks der Aufrufer die Kontrolle der CPU, sein gesamter Kontext wird gesichert. Der Aufrufer ist im BS 2000 eine Task, im Falle eines Lockmisses findet ein Taskwechsel statt. Weitere den Lock anfordernde Tasks werden nach FCSFS eingeordnet. Der Vorteil der Suspendlocks liegt darin, daß die CPU nicht blockiert wird, als Nachteil schlagen die direkten Kosten des Taskwechsels und eventuell indirekte, wie partieller Verlust des Working Sets während der Wartezeit, zu Buche.
Bei beiden Lockarten wachsen die Lockverluste quadratisch mit der Lockwahrscheinlichkeit p. Die durch Suspendlocks verursachten Verluste reagieren jedoch auf die Varianz der Lockstrecken kaum. Einige Formeln zur Berechnung der Lockverluste werden vorgestellt und diskutiert.