Summary of the

1998 SIGMETRICS Symposium on Parallel and Distributed Processing

Editors:
Jeffrey K. Hollingsworth and Barton P. Miller

Student Note Takers:

Bryan Buck	Cheng Liao	Ahmet Ozmen
Lynn Daley	Kathleen Lindlan	HeeDong Park
Delbert Hart	Joao Lourenco	Sergio Torres-Varela
Chu Jong	Tia Newhall	Wanging Wu

The 2nd SIGMETRICS Symposium on Parallel and Distributed Tools (SPDT'98) was held on August 3-4, 1998. The goal of this conference was to bring together researchers, developers, and users in the area of tools for parallel and distributed programming. Technical areas include performance profiling, debugging, visualization, program steering, checkpointing, and experience reports.

The conference included two invited talks, one on the Department of Energy's ASCI program and one on the new IA-64 processor architecture. The conference also included technical paper sessions and a panel of independent software vendors (ISV's) in the area of parallel tools.

The invited talks lasted approximated one hour each, followed by an extended question period. The panel included 10-minute position statements by each panelist, followed by a question period. The technical sessions included 25-minute talks, each followed a short question period. At the end of each session, there was a "mini-panel" where the audience was could ask additional questions of the session speakers.

This document contains summaries of each presentation, plus the questions and answers that. The summaries are meant as a companion to the conference proceedings published by ACM Press (acmpubs@acm.org, +1-800-342-6626), ACM Order Number 488981.

Session 1: Invited Talk

ASCI Simulations of Thousands of Processors

Speaker: Dr. John Reynders (LANL)

The speaker presented an overview of the DOE ASCI project. ASCI is responsible for testing, safety, reliability, and performance of the US nuclear stockpile via simulation. He explained that for these applications they must achieve higher-resolution (three-dimensional, full-physics, and full-system) that requires high-performance computing far beyond current levels of performance. For example, the ASCI White computer (to be installed in 1999) will have a peak performance of 10Tflops, 4TB of memory, occupy 17,000 sq. feet, and consume 3.6 Megawatts of power. The ASCI systems are geographically distributed, so high-speed networking, managing distributed objects, and security issues are important. In the context of application development, tools are required to debug and analyze multi-process, multi-processor, and multi-thread applications. The tools used for ASCI applications include TAU for profiling; Vampir and AIMS for performance analysis of message passing applications; View, Prism, MUTT, and Vampir for debugging.

Jeff Hollinsworth (University of Maryland): Does the system include redundancy for hardware?

John R.: We have not looked at that issue yet. We have just tried to implement and get multiple processor performance, but checkpointing and fault tolerance are future research directions.

Jeff H: In a large systems, failures are routine not an exception

John R: One thing we are really interested in fast IO to permit frequent checkpointing of applications.

Cherry Pancake (Oregon State): How do we insure the accuracy of simulation output? For ASCI this can potentially be very dangerous for all of us.

John R: There is no high level support for verification of the accuracy of the simulations.

Bart Miler (Wisconsin): What is the start up time for an application?

John R: It has a low start up time (practically instantaneous).

Mary Zosel (LLNL): Start up time depends on the number of shared memory images. May be slow if there are lots of nodes.

Jeff Brown (LLNL) Startup time is difficult to predict and a complex issue.

Session 2: Middleware

OCM –A Monitoring System for Interoperable Tools

Speaker: Thomas Ludwig (TU Munich)

OMIS is an On-line Monitoring Interface Specification between tools and low level monitoring modules. The main goal is to provide an inter-operable, uniform and flexible environment to the overall monitoring system, so that the monitoring system could be ported across the systems with little effort. Another advantage of OMIS is that existing tools (e.g., debuggers or performance analyzers) can easily be attached to the low-level monitoring layer. The OMIS interface layer converts tool requests to the monitoring layer by splitting the requests into individual event-action pairs. This obviates the restrictions of conventional monitoring systems and thus allows different classes of tools to attach and deattach. A set of global arguments defines all possible re-

quests in the form of event-action pair. Debugging and checkpointing tools are the first targets for the interface.

M Ranganathan (NIST)): Can your system support composite, distributed breakpoints (i.e. test conditions for breakpoints on multiple processes)?

Thomas L: Breakpoints can be set so that the condition is fully distributed.

Wagner Meira, Jr (UFMG-Brazil) Do you have an idea of what sort of intrusion measurements you get; (i.e. what effects are introduced by this kind of monitoring)?

Thomas L: If you are doing debugging, the intrusion is high, however, if you are doing performance evaluation, the intrusion is quite low.

An Object Based Infrastructure for Program Monitoring and Steering

Speaker: Greg Eisenhauer

The speaker described MOSS, an infrastructure for program monitoring and steering. The primary abstraction of the system is a set of objects with associated state and methods that mirrors objects in the application. Changes in application object state cause UPDATE methods to be called in the mirror objects. Program steering is accomplished by remote invocation to the application objects. An advantage of the authors approach is that if the application is written in an object-oriented style, steering is available without requiring the application programmer to write specific code since the existing application methods can be called from the monitoring system. It also allows the Mirror Object Model to leverage any synchronization code already written by the application developer to arbitrate access to application objects. If the application is not object-oriented, or if its existing synchronization mechanisms will not permit safe steering, then the application will have to be modified to include object-oriented wrappers around key procedures and possibly augmented with additional synchronization. Further, the Mirror Object Model is meant to be applied to itself to enable meta-level monitoring and steering. The mirror objects in the monitoring system have many of the same features and responsibilities as objects in applications.

The MOSS was compared with Audobon and Autopilot. A key advantage of MOSS is that the overhead per sensor is low and does not increase with when the sensor size (amount of data extracted) increases due to the MOSS system avoiding copies that the other two tools need. A second comparison showed that the round trip time of monitoring and steering functions of MOSS do not increase as much as Autopilot when the data size increases. However, for smallest data set Autopilot is 30% faster than MOSS.

Another important contribution of MOSS is the use of a subscribe model rather than a broadcast based event-stream, which allows data to be delivered only to interested parties. This reduces the network bandwidth requirements approximately 37 times comparing to event-stream based systems.

John Reynders (Los Alamos): What are the effects on communication of coupling multiple applications? What kind of data vs. controls channel is used?

Greg E: The controls and data used with Mirrors are for monitoring and steering. You would still want some shared memory or something to link (or couple) different applications.

Margaret Martonosi (Princeton): What is the cost of eliminating the data copies? What do you loose? Is there some functionality loss?

Greg E: No functionality was lost, this was extra work.

Mini-Panel

Diane Rover (Michigan State) Compare and contrast publish-subscribe vs. request based monitoring.

Greg E: For long running applications, for example several days, you want to pop in and out and monitor different things, publish-subscribe supports this. Publish-subscribe also eases the task of building re-usable clients.

Thomas L: By making requests on a regular basis, you can build what is basically a publish-subscribe system.

Ranga M: What if the system / language does not support the object model?

Greg E: Not all the applications need to be objectified; only the parts needed to instrument the application must be object oriented. The basic object-oriented primitives required are similar to CORBA get and set mechanisms.

Allan Malony (Oregon): What are the effects of different language and object systems?

Greg E: We have few assumptions about inheritance, only a generic CORBA-type model is assumed.

Thomas L: We're at a lower level of abstraction. The tools that are built on top of OCM, not OCM itself, should handle the issue of blocking vs. non-blocking.

Jeff H: If we have two external things trying to control the application, i.e. trying to steer, what are the implications.

Greg E: The result is unpredictable, I don't think it's a system problem, it's more of an application collaboration problem.

Thomas L: The semantics of interpretability are not complete yet. Our basic strategy is to find what is of high interest, let one tool manipulate, and the rest observe. If two tools wish to manipulate the application at once, the tool builders have to negotiate the semantics so the tools are non-interfering.

Bruce Irvin (Oracle): What are the problems with objects being created and deleted during program execution?

Greg E: Objects send events on invocation, whatever is mirrored in our system is registered then, the same as CORBA object references. Naming objects is a problem, so we choose to assume the same naming scheme as the application.

Thomas L: Naming is easier as we have node abstractions. The node abstraction means you can access each process on a node by just referring to a node.

Session 3: Network Instrumentation

Performance Monitoring in Myrinet-Connected Shrimp Cluster

Speaker: Cheng Liao (Princeton)

The goal for this research was to understand the performance issues present in NOWS. Simulation is too slow for understanding performance issues. Software monitoring allows measuring quantities tied to the program model, it loses low

level data. Hardware monitoring provides low level information, but it is not portable and flexible. Their approach is to build the monitoring into firmware on LANai board and to maintain a global clock in software. The architecture was 8 PCI-based PCs using Virtual Memory Mapped Communication library with Myrinet network interface cards. The monitoring code and data gathered resided with the 1 MB of memory on the card.

The monitoring tool ran as LANai code. The tool focused on the latency incurred at the different stages of communication. Latency can be split up into Source Latency (the time taken from when the host initiates the send to when the LANai finishes the DMA transfer), LANai Latency (from when the LANai finishes the DMA transfer to when it finishes sending the data), Net Latency (from the end of the DMA transfer to when the data has finished arriving at the destination), Destination Latency (from when the data finishes arriving until it has all been transferred into memory by the DMA), and Tail Latency which is Net Latency – LANai Latency.

There wasn't a global clock available at the hardware level, so a global clock was implemented in the firmware using Christian's algorithm. The problem is that LANai polls for information. The solution was to keep dedicated operations as short as possible and to check for events regularly inside of long operations. This allowed the clock to have less than 3 microseconds of error. The execution time overhead of monitoring with this approach was less than 3%. The system can also keep separate statistics for different programs on the same machine.

Jim Larus (Microsoft): You talked about perturbation of bandwidth, what about latency?

Cheng L.: The monitoring overhead on latency is a constant 5 micro-seconds.

Jeff H.: You have lots of numbers to show where the time was going -- have you been able to use them to improve performance?

Cheng L.: We've been able to improve the performance for Barnes by 50%, and for another application by 150%.

Margaret (Princeton): We made some SVM protocol changes based on the results at different layers.

Experimental Evaluation of On-line Techniques for Removing Monitoring Intrusion

Speaker: Wanqing Wu (University of Pittsburgh)

Existing techniques for removing monitoring intrusion have included off-line techniques, in which traces of program execution are examined and an attempt is made to correct for intrusive effects; static techniques; and the use of hardware. This paper presents a new, on-line technique. The on-line intrusion removal is based on several principles. First, extend the notion of time to keep information about how much the computation and communication have been delayed by monitoring. Second, synchronize the relative progress of communication and computation, by attempting to equalize these delays. Third, accommodate delayed and out-of-order messages. This is done by control at the source; intrusion is removed when the message is sent. A token-ring network was used, and when computation lags, the token is set to an "abnormal" state. A token in an abnormal state can only be used to send monitoring messages. When the computation has caught up and the token passes the node that set it to the abnormal state, it is returned to normal and can again be used to send application messages.

Although they can remove many types of intrusion, there will always be some intrinsic overhead that is the cost of providing intrusion removal. To evaluate the effectiveness of the intrusion removal, they looked at the accumulated number of events in the system and the order of events. (See graphs in the paper for results showing that most intrusion was removed.) They also examined the difference between the original execution time and what the intrusion removal system estimated as the original execution time. The percentage difference was very low for the applications tested 1.7-2% for our Parallel Matrix Multiplication, and about 2% for our Concurrent FFT program.

Normam Matloff (U.C. Davis): In some applications even one scheduling change could have a ripple effect. It seems like you can't get rid of it all. Have you thought about applications where a small difference makes a big difference?

Wanqing: Some scheduling points are inappropriate for using this technology. We can get very close, but there are overheads we can't remove (very small ones, for instance).

Jung Choi (IBM): The two networks you presented were Token Ring and Point to Point. Would your methods work on Ethernet?

Wanqing: Because of the random delay, our technology is not suited for analyzing Ethernet.

Choi: You presented the token as arriving at P1 then P2, what if it arrived at P2 then P1?

Wanqing: It would be delayed more. If P1 has changed the token to abnormal, then P2 can't change it back, only P1 can.

Mini-Panel

Jeff H.: This is more of a question for the first speaker. The slow clock on the network co-processor is a constant problem. Is the fact that dedicated communication processors are slower than the latest general purpose processor a fundamental problem, and will it get better?

Cheng: It is a fundamental problem that the dedicated processors are slower, there is no cache, and the overhead is longer. You might be able to solve it with an extra card that takes over clock synchronization, for instance. This could offload some work onto one dedicated node. A lot of things can't be offloaded in this way, but some things can. Basically the problem is very fundamental, a co-processor is much slower. Another possibility is to move some functions back to the main processor.

Karsten Schwan (Georgia Tech): Are there things you wouldn't want to put on a card (to Cheng Liao), or (to Wanqing) that aren't applicable to being removed?

Cheng: Tracing is one thing we don't have. Tracing is important, but we are unable to include it due to memory restrictions. One thing that we could look into is to make use of the memory in the host. Also, we don't have an adaptive function to do on the fly migration or adaptation of moving the monitoring between the host and the firmware. We want to add migration to the system. But there's the question "Can the cost be justified?"

Wanqing: Our assumption is that we can't remove intrusion at the level of cache misses or the pipeline – we can't remove that completely. And our work is not fine-grained enough for some applications, e.g. aircraft flight control systems.

Session 4: Debugging

Protocol Based Data-Race Detection

Speaker: Brad Richards (Vassar)

The authors developed a data race detection technique that uses distributed shared memory (DSM) coherence protocols to detect data races at runtime. A data race is a read and write to the same shared data where there is not enough synchronization to prevent conflicting events from executing concurrently. The typical approach to detecting data races consists of annotating the application code between synchronization operations to maintain an access history to shared data, and then checking the access history either at runtime or postmortem to detect data conflicts and races. The authors' approach requires no code annotations and thus requires no special compilers or tools to generate the access histories.

Their implementation uses Blizzard-S, an all software implementation of a DSM coherence protocol running on the CM5, and is implemented for shared memory programs that use barrier synchronization. For barrier synchronization any data conflict is also a data race. Access histories are kept for one reader and one writer to each data item between synchronization points. The access histories are tacked onto the CM5 messages that pass the data around from node to node. The authors extended the coherence protocol to monitor each access to shared data to maintain the access history. When a shared block is acquired it is kept in an invalid state until each data item on the block has been read and written. The coherence protocol is triggered on the first read and write to a data item. The protocol detects if this is a real fault, and if not, only the access history for the block will be updated. The access histories are checked on-the-fly to detect data races.

The authors present results from using their technique to detect races in five SPLASH benchmarks. Of the five, they found data races in two of the benchmarks. They also found that their on-line race detection techniques cause an average slowdown of about 3.

Robert Hood (NASA): Why do you only have to keep one reader in the access history? How do you handle the case of a read by P_1, then a read by P_2 followed by a write to P_2?

Brad: Each host has a local version and it is merged with the global version.

Ranga: If you tried to make the analysis off-line using traces would you get better performance?

Brad: I don't know for sure, however, with traces you would have to keep more information, not just the last reader and writer.

Jeff H: How would you extend your technique for locks?

Brad: To do this, requires vector time stamps. This implies a larger memory overhead and more information would need to be sent. More network traffic would be generated, so network performance would be the biggest problem.

Deterministic Replay of Java Multithreaded Applications

Speaker: Jong-Deok Choi (IBM TJ Watson)

The authors developed a record/replay tool for JAVA threaded applications, called DejaVu that provides a deterministic execution. DejaVu was implemented by modifying the Sun's JVM. They demonstrated how DejaVu can be used to deterministically replay non-deterministic execution behavior due to threads and related concurrent constructs such as synchronization primitives in JAVA. DejaVu can also deterministically replay I/O and windowing events in JAVA thread applications.

DejaVu uses a portable approach, the logical thread scheduler, which is independent of the underlying thread scheduler. The logical thread scheduler contains enough thread schedule information, which is captured based on a global clock for the entire application and one local clock for each thread, to reproduce the execution behavior of the program during the replay. Their experiments demonstrated that DejaVu runs efficiently on a uniprocessor system. The authors claimed that DejaVu can also be used on a multiprocessor system.

Ranga: Will you miss a data race if you use your tracing and synchronization techniques?

Choi: No, you can record the access to shared variables and then detect the data race. However, the performance depends on what kind of information you want to see in the replay. You could build a replay tool that can detect the data race in the recorded execution. The tool can slow down when replaying the interesting parts of the execution, in this case a data race, and speed up during the replay of the uninteresting parts.

Efficient and Flexible Fault Tolerance and Migration of Scientific Simulations Using CUMULVS

Speaker: Jim Kohl (ORNL)

Large-scale, long-running applications need fault-tolerance and load balancing. CUMULVS is a software infrastructure for collaboration. It provides online visualization of distributed simulations, coordinated steering, and user-directed checkpointing. The approach of CUMULVS is that the user must instrument the application and mark each item for access. CUMULVS will handle the rest: it will periodically pass control, and checkpoint as desired. The user must be involved because distributed data decompositions are not always obvious or deducible and fault tolerance is hard. It is the user's responsibility to identify program state, and determine checkpoint consistency. CUMULVS handles all the tedious "stuff": organizing checkpoint data and coordinating the commitment of checkpoints. There are two types of recovery, rollback or restart. Rollback recovery only replaces the failed tasks and rolls back the rest. Restart recovery kills everything and restarts all tasks. The runtime system architecture provides one checkpointing daemon per host. Checkpointing is gruesome, but CUMULVS assists with the te-

dium. CUMULVS has minimal cost for added flexibility. Project data is at the following URL:

http://www.epm.ornl.gov/cs/cumulvs.html

Bart: For small programs you have about a 5% overhead, but what about very large programs, like one million lines of code? If you have 10,000 variables how do you know which ones to checkpoint?

Jim K: Techniques do not depend on the number of lines of code, but on the number of variables you want to checkpoint. If you have a lot of variables then you need a technique to automatically annotate the source code with instrumentation. Also, if you restart from a checkpoint and it doesn't work, then you know you missed a variable you should have checkpointed.

Cherri Pancake (OSU): What are your plans for assisting the instrumentation process?

Jim K.: No, I have no specific plans. I need to investigate HPF. HPF accessor functions could possibly be used to automatically instrument the code.

Bob H.: Is there just one application state for all the processes when you checkpoint?

Jim K: Yes. When you do the decomposition you have the information about the distribution of the data, you keep the local state, and when you do the checkpoint you merge them to get one global state. This is how you can migrate to a different number of processors than the number from which the checkpoint was taken.

Jeff B.: Is it a non-trivial time to write checkpoint data to disk? Are you doing parallel I/O?

Jim K.: No, we are writing one big checkpoint file. Currently the daemon is sent the data and it writes to disk, but it doesn't have to be done that way. You can have the individual tasks write to disk.

Jeff B.: Are your measurements for checkpoints with small data structures, and are they similar for large data structures?

Jim K.: Yes they are for small data structures, they are not good for large data structures.

Jeff B.: Do you have the capability to do graceful shutdown, and do you have the capability to handle signals such as termination or interrupts?

Jim K: There is no good way to reliably catch signals. You can guarantee that you can catch it at each checkpoint

Doug P: It would be nice to be able to do the check point using a timer rather than on some number of iterations.

Mini-Panel

Jeff H: These are the only 3 talks in debugging in the past 2 years, is this the entire state of debugging? Is debugging research dead?

Jim K.: There is no reason to build a new debugger from scratch, when I can use something like TotalView and modify to do the things I want for my research.

Cherri: Users are still demanding checkpointing, why isn't this available?

Doug P: Venders aren't providing it because it is a hard problem to solve.

Bart: Why was CRAY able to do it?

Doug P: It depends on the environment; how much control you have over the OS; all kinds of stuff gets in your way.

Jeff B: ASCI codes do data checkpointing, but this doesn't help with debugging so you still need system level checkpointing.

Choi: Maybe. 10-20 years ago academic and research labs where leading research effort in debuggers/ checkpointing for big scientific applications like the space program. Now, Java can effect tool building. We have the opportunity to use what we know to build tools that may have a big impact on the commercial world.

Margaret M: If you had the freedom to design a programming language and programming model that would be ideal for debugging tools, what would it be?

Bart: sequential programs. ☺

Jeff H.: no memory hierarchy. ☺

Brad: A realistic programming model will have nasty points to deal with like threads and locks. You could look at compositional models, when you add a task you don't have to worry about how it interacts with other tasks.

Jeff B.: Condor can checkpoint data and state in user space; could you the condor routines?

James K: Maybe Condor's model could work, but Condor can't restart on another platform.

Bart: Condor is able to checkpoint PVM applications.

Cherri: The biggest problem for all tools is that most information that would help is not visible in the program. Is anyone looking at program annotations?

Choi: But then you have to debug the annotations.

Jim K.: The best programming environment is the one you are using now...Java. The problem is how to deal with Java calling Fortran, or more difficult Fortran calling Java.

Choi: There is a problem with Java semantics requiring array bounds testing on every access.

Jeff H: What is wrong with checking array bounds on each access?

Session 5: Invited talk

Parallelism: The IA-64 Advantage

Speaker: Michael Mahon (Hewlett-Packard)

The Goal for this architecture is to be optimized for cost/performance. It's mostly like RISC, but different in some ways. IA-64 will support several levels of Parallelism: Fine-grained (instruction-level parallelism, pipelining, superscalar, and VLIW) It also supports Medium-grained parallelism via symmetric multiprocessing using real shared memory and multi-threaded apps. Coarse-grained is supported via multi-computer systems such as client-server, distributed apps and virtual shared memory apps.

Several trends in uniprocessor Performance Growth: 1) impressive growth staves off parallelism, 2) Improvement of 25%-30% per year gained by faster transistors, and 3) Improvement of 60% per year: gained by more transistors (exploit parallelism). Pipelined RISC machines are direct descendants of vertical microcode, and can produce 4x or 5x concurrency. In superscalar Parallelism (multiple pipelines), the HW evaluates potential conflicts between instructions each cycle. Scheduling superscalar processors involves fair amount of logic. What do we do for an encore? One idea is that compilers get additional responsibility for scheduling to minimize hardware delays. This could gain us a factor of 2 compared with a hardware pipeline. *Superscalar complexity*

is growing rapidly: functional unit area grows linearly with the number of units, but the scheduler area grows as the square of the number of units!

The new model we are moving to is Explicitly Parallel Instruction Computing. EPIC (in Merced and beyond) features a large number of functional units and the compiler schedules instructions and guarantees independence. There are several limits to parallelism: large memory latency, conditional/unpredictable branches, and sequential machine-language model. Processor speed and memory speed have a 100:1 gap and growing, and thus loads can incur high latencies, cause exceptions, and often they head a dependency chain. One strategy is to hide this latency via control speculation, where we separate a load from its exception behavior. For example, we could overlap iterations of a WHILE loop.

Several factors limit parallelism: unpredictable branches disrupt the fetch and execution pipelines. Predication creates larger basic blocks to schedule avoids pipeline disruption

The IA-64 instruction format packs instructions into 128-bit bundle that includes a template. The template has inter- and intra-bundle routing information. It specifies explicitly how many instructions can be dispatched in the same cycle. This permits the bits in a bundle define which instructions need to complete before the current instruction.

Ranga: If there are 2 instruction streams in parallel, one will be discarded. If these streams are executed in parallel, where is the win?

Michael M: We would otherwise be sitting idle.

Mary Z: How difficult will it be to support dynamic modification on IA-64?

Michael M: Dynamic techniques are interesting. They used to be mainstream. They give flexibility that we don't have with a static stream.

Bart: We do lots of code patching. We would like to be able to replace instructions dynamically. We have 128-bit bundles, but most architectures have 64-bit bundles. So we can't replace 128-bit bundle atomically. Is there a way to atomically update?

Michael M: Yes, but I'm not willing to explain.

Bart: What does the PC address?

Michael M: A Bundle.

Bart: If an interrupt occurs in middle of a bundle, how is this handled?

Michael M: I can't describe the interrupt architecture yet. A number of architecture features haven't been announced yet.

Bart: Intel has been hiring compiler writers at a rapid rate (presumably for the IA-64 projects), why not trade hiring compiler grads, for a faster memory chip?

Michael M: You simply can't build bigger AND faster memory chips. Large is enemy of fast! Memory hierarchies are here to stay.

Jim Larus: How will the wide bundles affect performance?

Michael M: In general, i-cache hit rates are pretty good. We need memory bandwidth to fetch multiple bundles simultaneously. Man makes bandwidth, but God makes latency.

Jim L: Can you capture locality of machine?

Michael M: Instruction density is not all it's cracked up to be.

Arnt B.: You noted that the size of scheduler is growing. How big and complex is a compiler for EPIC going to be?

Michael M: Compiler complexity goes up rapidly. The complexity of the computations for scheduling grows also quite rapidly. It is an NP-complete problem. However, we can restrict the scope of attention. If we assume that by profiling, we find the important regions to concentrate our optimization effort. The algorithms are not as complex as their execution -- the algorithms are comprehensible by humans. Putting things into hardware tends to limit aspirations, so pushing things to the compiler creates more opportunities.

Karsten: I don't see the low-end in your design space

Michael M: What is introduced at high-end ends up in low-end in 2-3 years.

Mary: With the template as part of instruction, how many bits is it, how fully defined is it?

Michael M: We haven't revealed this yet. If you think about it, you can probably figure it out. Also, it's not a good practice to fully exploit all bit patterns in an initial release.

Keith Shields (Tera Computer Company): What about performance counters?

Michael M: There are lots of performance counters, but I can't talk about them.

Margaret M: With all the x86 code out there, it seems a big problem to translate them into IA-64 code.

Michael M: It's Intel's problem to run with running pre-existing code. With HPUX, software-based methods are appropriate.

Session 6: Performance Tool Infrastructure

An Enhanced 3-D Animation Tool for Performance tuning of Parallel Programs Based on Dynamic Models

Speaker: Noritaka Osawa (University of Electro-Communications)

Efficient parallel programs are hard to develop. The programmer has to consider both utilization of locality (communication overheads) and exploitation of parallelism. It is necessary to distribute suitably computation and communication in a parallel environment as well as to use an appropriate parallel algorithm. The author proposed a 3-D animation performance-tuning tool "Kanoko" to help find the imbalance of computation and communication in programs. This tool maps the physical structures to a dynamic system model, simulates the system, and then shows the user a visualization and sonification of the program behavior. It also exploits the human ability to distinguish slight changes of familiar objects.

The user first determines a dynamic system to be used, the tool will map elements and states in the parallel computer system to bodies and forces in the dynamic system. Solving the equation of motion then simulates the motion in the system. The results can be made visible and audible. Kanoko is implemented using VRML97 and the Java programming language. It is composed of VRML definitions to specify the basic shapes of objects, a Java program to solve equation

of motions, and a VRML browser. To use the tool, the user first collects trace data by executing a target parallel program, then uses Kanoko to animate the trace data. By looking at the animation, users can search for imbalances that cause performance problems. Then the user can go over the source code and trace data to determine if it is a real problem. Finally, the user can confirm the improvement by repeating the process again. The author also presented some examples of using their tool for performance debugging.

Margaret: Have you experimented with larger parallel programs?
Noritaka O: No

The Performance of a Service for Network-Aware Applications

Speaker: Katia Obraczka (USC ISI)

Some distributed applications need to be network-aware, which means they need to have the knowledge about the dynamics of the underlying communication and computing infrastructure. The author presented evaluation of the performance of topology-d, a service that estimates the state of networked resources by periodically computing the end-to-end latency and available bandwidth among them. The advantage of end-to-end measurements is that they can capture both network and server load. Topology-d then computes a low-delay, high-bandwidth spanning tree with additional edges for fault-tolerance. The estimates of topology-d can be used in many different ways.

For performance and robustness, topology-d was implemented as a UNIX daemon using non-blocking I/O for all communications. The daemon can be queried from a Web browser or a command-line utility. Each machine running topology-d periodically computes RTT between itself and all the other nodes by using a round-trip time stamped UDP packet, and computes available bandwidth by sending a block of data by TCP. There is also an aging algorithm involved to take previous history into account to dampen the effects of transients. A group master process collects the estimates reported by group members into a cost matrix, in which each entry is communication cost between corresponding machines. The cost is computed by dividing bandwidth by RTT. Then an algorithm based on minimum spanning tree, which also takes a parameter k as connectivity of each node for fault-tolerance, generates the logical topology.

The author also showed results from experiments running topology-d at 27 Internet sites over a period of two and a half months. The bandwidth and RTT data were validated by comparing with the data got from NETPERF and PING, respectively. After that, the author then validated that the logical topologies generated by topology-d were consistent with the estimates of the current network state, and the logical topologies adjusted to changes in network and server load, as well as in group membership.

Rangana: Why use bandwidth/RTT as cost metric?
Katia: We use bandwidth/RTT because we want to favor high bandwidth and low latency/delay links, but it doesn't mean we have to do that.
Michael M.: The cost seems to be inverse to what you talked about, if you want to use a minimal spanning tree.
Katia: Yeah, for us, low cost is high bandwidth.

Mini-panel

Jeff H: Is storage space a problem when your visualization is converted into VRML representation?
Noritaka: Our system generates VRML on-the-fly, so there is no storage requirement.
Rangana: What do you mean "collaborate tuning"?
Noritaka: It means multiple people see the same animation.
Doug: Katia, what technique do you use to validate your tools?
Katia: We compare case to case with other tools
Ranga: Do you use multicast protocols?
Katia: No, we don't use multicast, the problem is multicast is not reliable.
Rangana: How to measure server load?
Katia: It's compound, taken into account in our bandwidth & RTT estimate.
Doug: It seems that you are comparing apples with oranges. You compare RTT measured with UDP and bandwidth measured suing by TCP
Katia: To compute bandwidth we need to send data. We used actual data to compute bandwidth, and for historical reasons we also needed reliability.
Doug: My point is that you measured bandwidth and RTT with different protocols. Also, in practice, traffic on networks is almost all TCP.
Katia: We compare ping with RTT & netperf with bandwidth to validate them. This is not a big problem because UDP uses the same links as TCP, and so it takes into account the contention for the same resources (routers and links).

Session 7: Panel Independent Software Vendors Perspective

Moderator Bart Miller (University of Wisconsin)

The moderator posed the following questions:

- What is the role of ISV in parallel computing?
- What are the big guys doing right in tools?
- What are thy missing?
- Who are your customers?
- What are you doing?
- What are your biggest obstacles to success?

Hans-Christian Hoppe (Pallas GmbH)

Pallas was started in 1991 and has 20 employees. It is a leading software company in the European HPC community. It has a worldwide market. The types of software they work on is benchmarking and porting, tools (which make up about 40% of their business), project management, and application development. Pallas targets application software developers, such as those at BMW or Volkswagen. Customers are often research centers, and they often collaborate with universities to commercialize their software. The hardware vendors that Pallas deals with are three Japanese vendors. The application areas that Pallas is involved with are meteorology, CFD, chemistry, and structural mechanics.

The use of open interfaces is beneficial to independent software vendors. MPI is very important, HPF's support is dwin-

dling and OpenMP looks promising, but its future is questionable. Open interfaces allow ISVs to amortize the development effort over many platforms. Another opportunity is that there are generally less effort being made for vendor-specific tool development. An example is Dolphin's success with TotalView.

The things that hardware vendor development does well are the integration of the software with the rest of the platform, its ability to exploit low level interface and the amount of support allows them to enter tougher markets. The things that aren't so good about hardware vendor software development is that it is a platform specific solution, i.e. not portable, a tendency to 'lock in' the customer, and the lack of domain specific tools.

Some trends in parallel computing are the decreasing number of platforms and a decreasing number of vendors. There is a split between high and low-end parallelism. This split will widen such that ISVs will have to choose which market they want to support. There will be a trickle down of HPC into SMPs, there is a reluctance to change though. Vendor specific unixes will decrease in importance, while Windows NT will dominate the US market before long and become important to the HPC/server platforms. Linux is a grass root movement that might become important, but it will need dependable support and development.

Obstacles for ISVs are the alliance between users and hardware vendors. Users will choose the hardware vendor if possible. The 'caveman syndrome', of why use tools, is also an obstacle. ISVs face challenges in information dissemination and marketing. And while vendor cooperation is improving it is still a problem

Wolfgang Gentzsch (Genias)

Genias is a small independent software vendor (ISV) founded in the 1980's consisting of approximately 22 employees. Its focus is on software tools for parallel and distributed computing. Genias started out selling tools (Express, Forge, Tecplot, ESIS), and today has developed CODINE a distributed and parallel resource management tool, PaTENT (parallel tools for NT), ESIS and PRIMEUR. PaTENT is part of the government funded WINDPAR project.

Genias' customers started out with university users and developers. Genias also provides consulting with public domain software like PVM and MPI to industrial end users. Genias acts as a distributor for other ISV's products (APR, PARASOFT), and they do their own development (CODINE/GRD, PaTENT, ESIS, PRIMEUR).

Some predictions for future trends in parallel computing are that the use of tools will increase, that UNIX will stay as OS on the mid to high end machines, that Linux will become more popular on low end machines and universities, and that NT growth will slow down. Also, SM vs. DM vs. VSM will need to be resolved.

Some obstacles to success for ISVs are that parallel computing is still too small a market, that there are too many operating systems (plus, it is very difficult porting tools from UNIX to NT). There are hundreds of public domain tools to compete with, that the cost of production is very high (ISVs need some type of government or big HW company funding), and that the time to market is very long. Also, customers still have too high a priority on hardware, and end users typically have high expectations, or else have the idea that they don't need a tool ("the tool? that is me").

Computer Manufacturers today are concentrating more on hardware and less, or not at all, on software tools. Computer Manufacturers often bundle software with their hardware, so it is hard for ISVs to get into this market; the big manufacturers have the machines and support for porting tools. For ISVs to compete manufacturers should provide more in-depth information about hardware and system software, and cooperate more with ISVs.

Leonid Shtilman (Mutek)

Mutek is an ISV that has over 60 employees. One of their major products is BugTrapper, a tool for remote trouble-shooting of Windows NT programs in C++. Another is AutoPar, a tool for automatically porting sequential programs to parallel machines.

The "big guys" do some things right. The reliability of both their hardware and software is good, and so is the quality of their application programs. However, their prices are too high for junior decision makers. Low- and mid-level managers cannot authorize the kind of spending necessary, and those higher-up tend to be conservative (and therefore not willing to experiment with parallel computing). The existing tools from these vendors are hard to use, and they are poor at the management of really large amounts of data.

There are several trends in the field. One is application oriented computing, with companies building their own supercomputers for specific applications. Another is that because message passing is difficult to program, software solutions are needed to make development easier.. Finally, companies are willing to pay a high price for parallel computing, but increasingly they will use C++ on Windows NT rather than Unix.

One of the major difficulties in parallel computing is the high development cost of applications versus the small number of customers. Also, the fast pace of progress in hardware makes it difficult for software vendors to keep up. There is also a lack of good solutions for parallelizing C++.

Mutek sees a need for automatic parallelization. Source code can be analyzed using both traditional compiler techniques and new techniques, and its behavior can be analyzed at runtime. The information gained can be used to create new, parallelized, source code. Mutek's tool, AutoPar, allows a user to work on ordinary sequential code in FORTRAN 77 or FORTRAN 90 and then parallelize it. The tool also provides feedback, with a graphical display showing different colors indicating where the program can be parallelized and other information. This allows the programmer to alter the program and which parts of it are to be parallelized (normally parts to be parallelized are chosen automatically based on a model of the cost of message passing) until they are satisfied with the level of parallelization. AutoPar also allows the user to profile a program, so that the results can be used in parallelization. Currently, a beta version of AutoPar for FORTRAN 77 is ready, which supports the Solaris and Windows NT platforms.

Jim L.: I hear the ISV complaints about how small the market is, etc. What about looking at business and industrial markets rather than the smaller "technical markets" such as universities? Why limit yourselves? There are big parallel applications in databases and business applications.

Wolfgang G: Mostly it's simply that our history has been in the technical markets. We started in the technical market, and don't really know much about databases, etc. Originally our goals were not focused on making money, we were more interested in the fun technical applications. We would focus strongly on the commercial market today, based on financial considerations.

Hans-Christian: Also historically we have been in the technical market, that started because we know what we are good at. We are currently moving into tools for performance analysis, getting good results, hoping to complete the shift to more business fields and business applications in 2-3 years.

Leonoid: Also historical for us, for all the reasons mentioned. We do have a static debugger in process, which will start our move into the commercial market.

Bart: TotalView was initially funded by DARPA, and several other products were initially started with either venture capital or government funding of some sort. Can we do development of this kind without initial money dump? Can small ISVs do this without government support, or is that possible only for big venders?

Wolfgang: Rather impossible for ISV's. We need 2-3 million dollars of venture capital for 2 years of development, then you need big sales forces before you can go anywhere. Since scientific computing market is small, no venture capital will fund us, so we must rely on the governmental support.

Leonoid: It is important to use government money. We started with venture capital of ~14 million dollars partly (indirectly) from the Russian government.

Hans-Christian: I agree that we need the influx of government support. This might change as parallel computing becomes more popular in industry, otherwise ISVs always need startup help. Sometimes it is possible to develop new tools as part of another project, then spin them off into products from there.

Margaret M: The aerospace industry has money, do you have aerospace companies as clients?

Hans-Christian: Yes, we have European aerospace companies. Mostly they now use vector machines, as they move to more parallel systems we hope for growth into the industry.

Leonoid: I'm a professor in an aerospace engineering department! Most AE applications currently use packages with MPI, our tool is already useful for these applications.

Diane: Who do you see as competitors: each other, big hardware vendors, or perhaps public domain software? How does collaboration fit into the picture?

Wolfgang: Yes, yes, and yes. Public domain and collaboration are competitors. We have moved to distributed computing, thus no longer compete with Pallas in the parallel market. We want to have fun as well as do computing and focus more on industry. We try to avoid direct competition. Public domain tools allow us consulting opportunities.

Leonoid: We don't have competitors, we have friends. Porting is our strategy. Together with the HW producers we want to be part of their environment rather than sell independently. We don't really want to sell independently.

Hans-Christian: In the public domain, we are competing somewhat. PVM has opened the market for some areas to grow. Public domain software helps get the word out that such things exist. Users move to new tools by using the public domain tools, but then they find they need company support, and they start looking for products.

Bart: So our challenge is to create new ideas for more spin offs of new companies.

Session 8: Performance Analysis

Using GODIVA for Data Flow Analysis

Speaker: Terrence W. Pratt (NASA Goddard)

GODIVA (GODard Instrumentation Visualizer and Analyzer) is a tool to analyze the performance characteristics of large science simulation codes running on scalable parallel computers. The speaker focused on the unique aspect of GODIVA that allows the user to probe the performance characteristics of large arrays and the flow of data into and out of arrays in large science codes. The goals of GODIVA tool are (1) the applicability to large codes and machines, (2) comprehensive measurements, (3) control of cost and granularity, and (4) flexibility. It is organized into input (user-annotated code), output (measurement table and graphs) and system(preprocessor, run-time library and post-processor). Godiva cannot check the correctness of the annotations. Godiva provides the live-dead data flow analysis that gives unique insights that particular storage areas are actually used during execution. The current version of Godiva lacks a graphical user interface (GUI) because they are in a period of early evolution of the system, with limited resources available for development. Pratt reported the sample-annotated code of matrix multiplication, MPI send operation, and cache use. GODIVA status is has been in use for two years in the study of Fortran and C codes written to address "Grand Challenge" problems in earth and space science. This tool is portable and runs on several scalable parallel systems, including the 1088-processor SGI/Cray T3E at Goddard and two Beowulf-class cluster computers.

Cherri Pancake: How large was the array, and how much of the trace file was required for the live-dead storage histogram?

Terrence: The array is about half a megabyte, and the trace file is under 30 kilobytes, it's small.

Bart: What's the granularity from the analysis.

Terrence: We use an X-axis granularity of 500 points, but that's just constant, so we could change that.

Bart: What is the runtime cost of your analysis?

Terrence: The overhead for this is fairly severe if we track fine-grained slices of the arrays. If the code is using large contiguous blocks and the arrays are fairly small, the overhead is low. We can also turn-on and off data collection for particular arrays.

Jeff H: How do you debug the annotations?

Terrence: The assumption is that you can make a lot of errors. We do it very carefully. The real debugging comes

when we have got some results that do something. Also, the preprocessor doesn't even look at the code, it only looks at the annotations, so we can use the annotations to lie about the code, and sometimes it is useful for abstracting the rest of the code.

Margaret M: What hardware is used get the caching data?

Terrence: We do not use any hardware counters for the caching data. We take an address and map it into the cache lines. We assume a direct mapped cache so it's easy.

Karsten: Can the annotations be generated automatically?

Terrence: Of course. However, we don't know what we want or need from the system yet, and you can't automate something that you don't fully understand. We would like to dig more deeply than the traditional analysis, and data flow analysis is a good example of this, but it's hard to know exactly what will be useful.

Karsten: Is advanced visualization needed?

Terrence: We have had some discussion about that, but have not explored because of lack of time.

Using Cause-Effect Analysis to Understand the Performance of Distributed Programs

Speaker: Wagner Meira Jr. (UFMG, Brasil)

CEA (Cause-Effect Analysis), is an approach to performance understanding of distributed program and to the development of explanation techniques for non-deterministic execution. CEA differs from the previous work because it produces explanations for the duration of performance events, not for their occurrence. CEA is based on Weighted Causality Graphs (WCG) which abstract the execution of distributed programs and are DAGs where nodes are execution events and the edges are causal relationships among those events.

He showed the result of tuning Squid (a software DSM) hierarchies, and said that, by changing the hierarchy, the average response time decreased to 2.3 seconds, a 22% improvement over the initial flat configuration. In this case contention among requests increased, while both sibling intrusion and contention for parents decreased to 7% and 23%, respectively. The experiences with CEA are in the area of contention vs. throughput, load imbalance, lock granularity, multiprogramming effects, false sharing, and data replication vs. communication.

Karsten: What analysis can the system give now?

Wagner: It depends on the environment. For DSM, we have shared addresses, and for message passing we export the message sender and receiver.

Jeff H: We have estimated that for the type applications used in the ASCI program, a typical run would generate several terabytes of trace data.

Wagner: Tracing is never reasonable. That's exactly the point that by using the WCG and exploiting their properties to limit the trace size.

Jeff H: What is the bound of the length of a trace?

Wagner: The bound is application dependent. Because the amount of information we have to keep is bounded by barrier events. A worst case scenario is a program with only one barrier at the beginning and one barrier at the end.

Bart: Non-trace based techniques, such as sampling, can greatly reduce the volume of data collected.

Wagner: The price for having such detail is increased data collection, and you can do things like contiguous profiling but that's not our goal. We are more worried, in Carnival, about the intrusion and the perturbation of the application execution.

Jeff H: For the ASCI program, if Meira were to run his performance debugging on a production sized data set, it would take a machine almost as large as the measured configuration to process the trace data.

Wagner: I agree, but don't have the answer for that. Detailed analysis requires overhead, so that is basic trade-off.

Jeff H: The goal of performance profiling problem to find out the source of a bottleneck. One way to approach this is have narrowly focused traces, and thus reduce the complications of storing, and moving large trace files. Have you thought about what a minimal trace to meet your needs would be?

Wagner: What we really have tried to do is put something in between profiles and traces. The problem with traces is to analyze one instance and then the next and so on. What we do is to generate the explanation for an entire phase of the program's execution. For example, we may abstract 500,000 iterations of a loop by a single (representative) iteration.

Session 9: User Experience

An Empirically Derived Framework for Classifying Parallel Program Performance Tuning

Speaker: Rob Procter (Edimburgh University)

This work is the result of analyzing problems presented to the authors and reviewing files of queries to a help desk at a production supercomputer center (Edimburg Parallel Computing Centre). The queries received can be classified using the following scheme: *Specific Advice*: (the user has identified something they need to know, compiling options, use of a compiler, flags, etc), *Why is that happening?* (the user noticed changes in behavior, but has only a vague idea of the cause), *General Advice* (a user asked for advice on performance improvement. -- the most common type of question), *Inadequate Tools* (usually about how to use the information provided by the tool, that sometimes is contrary to what is expected)

Based on this data, they divided the questions into four causes: *Cause /Effect Chasm* (the manifestation of a bug is removed from its cause -- 25 occurrences), *Inadequate Tools* (the tools interfered or didn't contribute to solve the bug: i.e. lack of memory, probe effect - 11 occurrences), *Faulty Assumption/Model* (the programmer had a mental model that didn't match what the code actually did - 17 occurrences), *Spaghetti Code* (5 occurrences). They also discovered that programmers use four techniques to find these bugs: *Gather Data* (mostly via of 'printf's and sometimes breakpoints), *Inspeculation* (meaning inspection, simulation and speculation), *Expert help* (ask someone else), *Controlled Experiments* (programmer develops a hypothesis about the source of a problem, and then conducts tests to confirm or refine the hypothesis).

Users were asked about the process of tuning and its duration, responses ranged from "less than a day" to "months" (94% of responded at least "several days" and 50 percent indicated "weeks or longer").

Keith S: Which tools were used by the tuners involved? Do you keep track of the tools they used?
Rob P.: Mainly Paragraph PRISIM (CM-5), and MPP Apprentice (T3D) and some locallly created tools.
Doug P.: Did they like or dislike them?
Rob P.: There were mixed opinions. Some users even said it was good for some situations and not for others.

Searching for the Sorting Record: Experiences in Tuning NOW-Sort.

Speaker: Remzi Arpaci-Dusseau (UC Berkeley)

The work described here is the development of the NOW-sort, a disk-to-disk sorting algorithm. During this process, performance tools were useful for several purposes; setting expectations, visualization of performance, and searching for anomalies; configuration tools helped to define expectations; the visualization tools helped to identify performance problems, the search tools were useful at the end of the process.

The cluster used was the Berkeley NOW, consisting of 105 Ultra1 workstations, 16 KB cache (instructions and data), 512 KB L2 cache, 128 MB DRAM. Each workstation has two Seagate Hawk disks (10 MB/s), and a Myrinet communications card. Three 35-nodes clusters form the NOW, each node has 13 Myrinet switches, connected in a 3-ary tree-like structure. Each machine runs Solaris 2.5.1, GLUnix (a local set of OS extensions) is used for load balancing, scheduling, controlling jobs, and managing console I/O. The parallel NOW-sort was written using the Split-C library.

Before they proceed to tune parallel sorting, they started by trying to maximize the single node sort by minimizing total I/O required, avoiding paging, ensuring the system was I/O bound, and getting the disks to transfer data at nearly peak rates. An early tuning step was to use the mmap interface and madvise to inform the file-system that accesses were sequential. For the parallel version, an additional tuning step was to overlap communication with other phases of the sort.

Once the system was scaled, the next step was to do a clear, unperturbed run of the sort. To do this, they had to find out why runs were not meeting expectations. Four common causes were identified: faulty hardware, CPU stealing (someone had a process running on the machine), Memory hog (existing processes using large amount of the swap space), and Disk layout (disks are faster at the outer tracks). In the end, 95 of the initial 105 machines were considered usable. The results obtained with NOW-sort were 8.41 GB sorted in one minute using 95 nodes.

Mary: You were using a cluster with a large number of machines. On average, how many nodes were dead or unavailable?
Remzi: Around 10%.
Bart: How many months/years did you invest in this project?
Remzi: We started in 1996, as a joint project with my wife Andrea Arpaci-Dusseau. It started as a Database course project and has evolved, but overall it was quick, approximately one year elapsed time.

Portable Profiling and Tracing for Parallel, Scientific Applications using C++.

Speaker: Sameer Shende (University of Oregon)

The use of C++ and the language itself is evolving, and creates holes in tool support. Tools are important for scientific application developers, their project is building tools to trace and profile scientific applications that were written using C++. The model in which the work is based is the HPC++Lib. This library supports node abstraction, contexts and threads. A node is a shared-memory multiprocessor. The library supports two modes of execution multithreaded-shared memory and SPMD. The TAU tool uses timing via instrumentation to profile programs.

TAU is a portable set of measurement and visualization tools, and has been used in the ASCI Blue Mountain, the ASCI Red, Cray T3E, and PC's clusters under Linux. Currently, TAU can produce a large variety of statistics and they are currently investigating how to present all this information in understandable ways. They are investigating using 3D graphics and virtual reality.

MC++ is a Monte Carlo neutron transport code written in C++ using the POOMA framework. Since TAU profiling macros were built into POOMA, the profiling was available for use, but a few additional profiling macros were added to the MC++ code. The use of profiling macros at the top-level of the application allowed TAU to relate performance data back to high-level application events.

Another example is the Conejo code. This code was written right on the top of POOMA. Conejo is used to calculate mach numbers in non-mixing materials, and is written as a mix of data-parallel (POOMA) and SPMD styles. Another use of the TAU environment was with Blitz++. Bliz++ is a library that uses templates. The advantage of TAU for all of these applications is that the user chooses the level of profiling, and that information can be represented in a form relative to the nested types.

Mini-Panel

Jeff B.: Does VAMPIR use data from your instrumentation?
Sameer: Yes! VAMPIR can get data not only from the MPI communication information but also the information we provide.
Jeff H: Several of you created new tools, why? Is it because "my tool is the best tool"?
R. Arpaci-Dusseau: We needed very simple tools. We didn't have the time to invest in learning how to operate with the existing ones.
R. Procter: The tools available were not flexible enough for their needs. Many of the users in our survey were experiencing problems and they couldn't identify clearly their problems, and couldn't choose the right tool. We looked for flexibility and adaptability, mixing and matching the tools to different problems.