First USENIX Workshop on Hot Topics in Parallelism (HotPar '09)

Berkeley, CA
March 30–31, 2009

CHALLENGES AND OPPORTUNITIES OF HETEROGENEOUS HARDWARE

Summarized by Rik Farrow (rik@usenix.org)

- ***A Case for Machine Learning to Optimize Multicore Performance***
  Archana Ganapathi, Kaushik Datta, Armando Fox, and David Patterson, University of California at Berkeley

Kaushik Datta explained that compilers produce poorly performing code on multicore CPUs without manual tuning. Their approach involves machine learning that tries particular motif-specific optimizations, generates code, and tests it. It is possible to do this for the entire problem space, but doing so would take many months to compute their example problems.

Jim Larus asked why compilers don't do this, and Datta responded that compilers do not do domain-specific modifications or change data structures to adjust for best memory access performance on a particular architecture. Rik Farrow asked if they had accounted for the difference in memory architecture between Intel Clovertown and AMD Barcelona, and Datta answered that they did, through pinning the memory to each Barcelona chip. Paul Emming of IBM asked whether the performance issues were related to memory bandwidth or latency, and Datta responded that it was effectively latency issues.

Archana Ganapathi took over the presentation and explained how they used machine learning to dramatically shorten the tuning time. Their model chooses a sample set of 1500 datapoints, runs the code, compares feature vectors, then adjusts the parameters and tries again. Someone asked why they chose 1500 for the sample size, and Ganapathi answered that this was a sweet spot in a process where the runtime can grow geometrically. Steve Johnson of Mathworks asked if there was some assumption about monotonic trend in the analysis of correlation, and Ganapathi answered that there are assumptions about relationships.

Ganapathi talked more about how they chose the point that expressed best performance, picked two neighboring points, and used these to find matching points in configuration space. They then used a genetic algorithm to permute optimizations. Their method takes about two hours to reach a performance level in the optimized result similar to what a domain expert could do with manual tuning in two weeks. An exhaustive automated search through the configuration space could take 180 days, so their learning approach shows real promise.

- ***Hardware Parallelism vs. Software Parallelism***
  John A. Chandy and Janardhan Singaraju, University of Connecticut

John Chandy said that processor clock scaling had stopped, but transistor scaling will continue for a while yet. Multicore processors are the current answer to what to do with billions of transistors, but there are serious problems with this approach. First, software that can use multiple cores has not been written, and it would be difficult to write and debug. Then there is the problem of memory bandwidth, which cannot supply more than a handful of cores at once. Their solution is a reconfigurable hybrid multicore architecture (RHyMA) that puts the reconfigurable portion of the processor on the "other side" of memory.

Chandy displayed a table (Table 1 in the paper) that compares performance of specialized hardware to software implementations; it shows that hardware, even running at slower clock speeds, outperforms software implementations of specific tasks like intrusion detection, numeric simula-

tions, and genome sequencing. Vikram Adve of the University of Illinois pointed out that they were comparing FPGA (Field Programmable Gate Arrays) to CPUs, but saying nothing about memory. Chandy said that this depends on the application—IDS, for example, which is basically string matching, ran 27.8 times faster in the FPGA. Adve asked if using FPGA helps with the memory access, and Chandy said that using FPGAs *can* make this better, but will not solve the data access problem.

Chandy pointed out that the use of heterogeneous processors is not a new idea. What they want to add is the ability to create new "cores" on the fly, using libraries of hardware. Steve Johnson pointed out that most operating systems are extremely allergic to special-purpose hardware, as most has state and is thus difficult to share. Chandy responded that they do need OS support but are not as pessimistic as Johnson.

Dave Patterson agreed that transistors are plentiful, but not power, and asked if reconfiguration was power-efficient. Chandy again pointed to Table 1, where FPGA versions are many times more efficient. Hans Boehm asked about security, if hardware is to be shared, and Chandy said that in their current version there is no way to leak information unless you create a routing path between two parts.

- ***Embracing Heterogeneity—Parallel Programming for Changing Hardware***
  Michael D. Linderman, James Balfour, Teresa H. Meng, and William J. Dally, Stanford University

Michael Linderman explained how their pragmatic approach to supporting heterogeneity in processors helps solve some of the issues brought up about the previous paper. He pointed out that the software ecosystem relies on stability and that running software where there may be hardware resources for some functions but not others, depending on the platform, is a problem with a solution.

Their own solution is to wrap implementations for particular algorithms with a common API so that the program has the same interface, regardless of whether the algorithm is done in software or by a specialized processor. Armando Fox asked if they separated policy from mechanism, and Linderman replied that they do via metawrappers based on policy. Jim Demmel asked about runtime resources and Linderman said that their software makes runtime choices depending on hardware availability.

Steve Johnson wondered how they handle the difference between passing arguments, as an ordinary CPU can use pointers but a GPU requires an array of values. Linderman said that the layer they propose handles copy of data when needed. Jim Demmel asked if data structures would need to be changed on the fly, and Linderman said he would get to this.

Linderman described this wrapper as sophisticated enough to support both programmer notations and the ability to group resources and to merge functions that should be combined for best performance. María Garzarán wondered whether they intuit the programmer's intent, and Linderman replied that they don't try to extract parallelism. Demmel expressed concern about determinism, and Linderman suggested that this concern could be expressed within metawrappers. Clem Cole speculated that Boeing would want the same answer every time. Linderman said that floating point includes some degree of non-determinism, depending on the implementation used.

MODELS AND PARADIGMS I

Summarized by Micah Best (mbest@sfu.ca)

- ***Parallel Programming Must Be Deterministic by Default***
  Robert L. Bocchino Jr., Vikram S. Adve, Sarita V. Adve, and Marc Snir, University of Illinois at Urbana-Champaign

Parallel programming is too hard, Robert Bocchino began, with too many non-deterministic interleavings making it difficult to reason about correctness. Most programs are intended to be deterministic and so parallel languages should be deterministic by default, non-determinism occurring only when explicitly requested. Some languages do guarantee determinism, but mainstream general-purpose languages do not. Martin Rinard brought up the point that even sequential programming is sometimes not deterministic, so why make parallel programming deterministic? Bocchino responded that non-determinism is limited in the sequential model and programmers tend to understand this, generally introducing it on purpose.

The benefits of achieving this goal would be almost sequential reasoning, the avoidance of subtle bugs, and simplified testing. Jim Demmel asked if floating-point operations were included in the "almost" part of the first point. Bocchino agreed that floating point leads to an increase in non-determinism in parallel, but reiterated that programmers understand this. David Patterson asked whether this proposed model allowed floating point to be non-deterministic. The response, including an example with reduction, clarified that the programmer would be able to specify the level of non-determinism.

After Bocchino described default determinism guarantees, support for controlled non-determinism, and methods for simplifying development and porting, Rajesh Nishtala asked about performance. Bocchino admitted that in some cases determinism will have performance consequences by nature, but they believe that in many cases that can be alleviated. Checks can also introduce overhead, but they were focusing on doing checks statically. Nishtala followed up by asking how well this would scale. Bocchino answered that, hopefully, one won't do this globally and in fact this may help with reasoning about performance.

After describing the strengths and weaknesses of approaches based on language, compiler, and runtime components, the speaker concluded that strong language mechanisms

are essential. Brandon Lucia brought up Kendo, a compiler-based auto-optimization. Bocchino responded that indeed compiler support can help make guarantees possible. The talk continued with a description of the effect system, which uses annotation of memory, called regions, as parameters in order to track what areas are being read and written during a particular operation. Nishtala asked if these regions are dynamically created. Bocchino responded that, yes, they are, but the reasoning is static.

Deterministic parallel Java with an explicit type and effect system was then introduced and its limitations were discussed. Jim Larus asked about the connection between determinism and type effect. Bocchino responded that if disjoint parts had disjoint regions, you could use that to ensure that all computations are deterministic. Larus asked whether all computations were independent and was told they were, with every pair of memory operations either commutative or disjoint. Larus then asked if this wasn't very restrictive. Bocchino responded that it was restrictive but fundamental and that they are working on more complex patterns. Rob Schreiber asked about the model of temporal epochs separated by barriers. Bocchino responded that the barrier model was supported.

The talk then shifted to the topic of hidden non-determinism. Bocchino outlined the use of programmer-provided trusted annotations with which the compiler can prove determinism. An example of this was the commutative operator, which was completely trusted by the compiler. Maurice Herlihy asked about operations that commutate with other operations. Bocchino responded that the support was not this fine-grained, but could be. The talk turned to visible non-determinism, which is sometimes necessary for high performance. This needed to be carefully controlled and explicitly requested by the programmer, with the non-deterministic code and the deterministic code isolated from each other. In terms of supporting this in the language, the conclusion was that the benefits outweigh the costs and that technical solutions, not necessarily specific to Java, can reduce these costs.

- ***Opportunistic Computing: A New Paradigm for Scalable Realism on Many-Cores***
  Romain Cledat, Tushar Kumar, Jaswanth Sreeram, and Santosh Pande, Georgia Institute of Technology

Santosh Pande explained that in opportunistic computing and scalable realism on many-cores, speedup is not always the end-goal. Immersive applications, such as gaming, multimedia, and interactive visualization, are designed to provide the richest and most engrossing experience possible to the user. Focusing on realism provides avenues to utilize multi- and many-cores over and above traditional task and data parallelism techniques.

This domain calls for algorithms with the highest sophistication possible so that a probabilistic achievement of realism is sufficient. The first approach for maximizing realism was a technique referred to as N-version parallelism. This technique involved speeding up hard-to-parallelize algorithms that made random choices by running multiple versions in isolation using different random choices and choosing the fastest one. This increases the probability of getting a faster result. Someone asked how it was known that this converged on the fastest result. Santosh replied that theoretic results support it. Someone else asked how the 2x speedup was justified. This was specific to the example; in general, it depends on the asymptotic complexity; many algorithms show a great deal of variance.

Next was discussed a probability density function (PDF) that described the speedup of the algorithm and using this to determine the potential results when running N copies of the algorithm. To support this technique, programming language abstractions were required to render each instance of the algorithm so as to be side-effect free.

Pande discussed the quality of the results and enhancements. This involved taking advantage of additional cores, scaling algorithms, and data sets with available resources. The runtime component of the system is based on offline profiling via machine-learning techniques. The profiling infers the structure of the application and learns the cause-effect relationship across the application.

An audience member said that similar techniques were used in circuit simulation, where multiple solvers were begun with the hopes of getting a fast convergence to results. Santosh responded that, absolutely, this technique had been inspired by others, specifically multi-scale physics simulation. Another audience member asserted that N-version parallelism works for randomized algorithms, but not for statistical sampling algorithms. Santosh replied that one could express computation by accuracy constraint on sampling.

- ***A Case for System Support for Concurrency Exceptions***
  Luis Ceze, Joseph Devietti, and Brandon Lucia, University of Washington; Shaz Qadeer, Microsoft Research

Brandon Lucia discussed what makes concurrency bugs such a challenge: they are difficult to reproduce and crashes may occur far from bugs during execution. Concurrency errors are not "fail-stop," but their effects may be, obscuring the original illegal behavior. Lucia asserted that an error should be delivered, an exception should be thrown, where the state changed to wrong. He then talked about how to specify exception conditions in terms of determining what behavior is illegal, which addressed an earlier question from Jim Larus.

Lucia outlined the three basic questions of concurrency exceptions: when should exceptions be delivered, to which threads are they delivered, and what is the system state at delivery? The burden is on the language, and it is desirable for programmers to be able to encode what behavior is illegal and embed their synchronization protocol. He identified three types of illegal behavior: locking discipline violation, atomicity violation, and sequential consistency violation. An

audience member asked if any bugs were left out, to which Lucia replied that ordering-constraint bugs were excluded for brevity. Another attendee asked if this implied sequentially consistent behavior. Lucia replied that a programmer needed to specify what regions of code should be atomic.

Lucia then said that locking discipline exception should occur when locks protecting data are not acquired before the data is accessed. The exception should be delivered immediately, before the access that violates the condition is given. An atomicity violation exception should be thrown when code was expected to execute atomically but didn't. Language support for defining expected atomic code is needed, as is monitoring of memory access interleaving. Mark Moir commented that this places a burden on the programmer, compile writer, and architecture designer. Isn't it better to change things so these problems are not possible? Lucia replied that they felt that this was not an excessive burden and not the only solution to concurrency errors. In response to another question about concurrent thread access he replied that this mechanism doesn't create atomicity, it enforces atomicity. As for when to deliver the exception, the violating thread was a good candidate, but the originating thread was also a good target for receiving the exception.

Data-race is a heavily overloaded term, and various memory models may define it differently. What is really wanted is a guarantee on sequential consistency. A sequential consistency exception occurs when it is impossible to guarantee that memory access reordering wasn't observed remotely. This exception should be delivered immediately before the reordered instructions execute. Tim Harris asked about detecting compiler reorderings. Lucia responded that what is needed is a way to communicate this to the lower levels of the system. Was support needed to see if reordering was observed? Yes, based on the work of Gharachorloo and Gibbons.

Multi-threaded state is the sum of the state of all threads, and concurrency and non-determinism make precise state tricky. Lucia offered two options: offer precise state to the offending thread only, and deterministic exception replay. An attendee asked how the state recovery mechanism interacted with I/O. Lucia said this was a difficult unsolved problem with replay. How much simpler was this than transactional memory? To achieve what they want they don't need to buffer values but only monitor. There is no need to keep an arbitrary number of versions.

APPLICATIONS AND TOOLS

Summarized by Eric M. Hielscher (hielscher@gmail.com)

- ***Parallelizing the Web Browser***
  Christopher Grant Jones, Rose Liu, Leo Meyerovich, Krste Asanović, and Rastislav Bodik, University of California, Berkeley

Leo Meyerovich pointed out that in order for handheld mobile devices such as smartphones to take over the space currently filled by laptops, the software that runs on them must run as fast as it now does on laptops. Bell's Law indicates that this shift to handhelds should take place due to shrinking transistors, but we've hit a power wall preventing handhelds from reusing the software of their laptop ancestors in the way laptops reused desktop software. Meyerovich focused on the parallelization of mobile Web browsers. Browsers are important because they are the dominant application platform, easy to deploy, Javascript is portable, etc. They also present an interesting challenge since writing programs for handheld browsers is difficult, as witnessed by the specialized versions of Web pages for phones and pages loading around seven times more slowly than on laptops.

The anatomy of the Web browser workflow is as follows: download pages, decompress them, lex, parse and build the DOM layout, render, and run scripts. Vikram Adve asked where the bottleneck is, and Leo responded that on handhelds it's truly everywhere—everything is slow. Ras Bodik said that compared with IE, layout takes twice as long. The project's status is as follows: work-efficient algorithms for various aspects of the browser have been developed, and work has been done on a programming model for scripting. While, on the surface, lexing may seem inherently sequential, a parallel algorithm for lexing was outlined that involves splitting the input text into blocks with some overlap. The scans can then proceed in parallel with care taken that the DFAs start in tolerant states. This results in an algorithm that wastes only a little work and scales very well (4.5x speedup on five processors). A parallel algorithm for page layout was also given that scales well up to three cores.

Here the talk turned to the problem of developing a parallel programming model for scripting. The extant browser programming model is a non-preemptive event-driven model where handlers respond to events and execute atomically. To parallelize this, we must understand how a document is shared, including document-carried and layout-carried dependencies. Concurrency bugs can crop up in many places: GUI animations and interactions, server interactions, eager script loading, JavaScript gotos, etc. Preliminary design on a new parallel scripting language has been done that focuses on making program structure clearer by making data and control explicit. Programmer productivity, targeting the 99% of programmers who aren't concurrency experts, will be enhanced by providing callbacks to actos, and performance will be improved by adding structure to detect dependences. Rik Farrow asked whether security was addressed by the work, and Leo responded that security is a concern but that it's orthogonal to the work at hand.

- ***Exploring the Limits of Disjoint Access Parallelism***
  Amitabha Roy and Steven Hand, University of Cambridge; Tim Harris, Microsoft Research

Harris pointed out the important traditional distinction between abstractions (programming language constructs) and implementations (e.g., transactional memory versus locks). What we would like is to be able to talk about the semantics of our abstractions without discussing their implementa-

tions. We then ask the question, when are TM-style implementation techniques useful? Harris showed a graph, with one axis representing contention for critical sections and the other the likelihood of conflicting memory accesses. The quadrant of the graph where there is high contention but low likelihood of conflicting accesses seems to be the region that is just right for TM techniques. Slower TM implementations make this sweet spot smaller, and faster ones make it larger. A formula characterizing the bound on possible speedup was given, using terms such as the probability of conflict, the fraction of time waiting to enter critical sections, and the fraction of time in critical sections.

The focus of this work was to develop a tool that uses binary instrumentation and models of thread timing and memory access to allow profiling of programs for locating synchronization bottlenecks. An assumption in the models is that conflict probability is a property of a given critical section. Pairwise conflict probabilities are generated for each critical section. Comparing the tool's predictions to serialized versions of a red-black tree and of Apache gave a fairly good match between the curves of the prediction and the actual data. The conclusion is that for these workloads, the assumptions of the model work well enough for the tool to be useful. The instrumentation is lightweight enough to allow large apps to be run at a reasonable speed and thus to provide good feedback. Further work includes addressing the questions of whether more complex timing models are needed for other workloads, and what the tradeoffs are between different conflict detection strategies.

Someone asked how stable the results were, and Harris replied that he wasn't sure. Jim Larus asked whether the researchers felt they had a good a priori intuition about which locks would be good ones. Harris said they didn't check ahead of time, but the results seemed very reasonable after the fact. María Garzarán asked whether the programs needed to be run with every possible number of threads. The profiling is done with a single thread running in order to get traces. Someone asked whether the tool might affect the computations. It was carefully validated. Paul McKenney asked how this tool compares with the ad-hoc feedback mechanisms used by the groups who develop various large systems. Harris wasn't sure, but his group was having discussions with such teams. Mark Moir asked about more refined models based on the size of transactions and on contention. This should be easy to plug in and would be interesting. Moir then asked how much profiling we could get for free from STM implementations. Perhaps it would be possible to add something like a Bloom filter to record access sets.

- ***Parallel Search on Video Cards***
  Tim Kaldewey, Jeff Hagen, Andrea Di Blas, and Eric Sedlar, Oracle Server Technologies—Special Projects

From a database perspective, search is sped up by the addition of indexes. The bottleneck in this domain lies with memory. The growth rates of the size of memory have outstripped those of structured data, and so the memory wall is increasingly an issue. Larger caches and specialized processors are the current approaches to alleviating this problem. One way to tolerate memory latency is through parallel memory accesses, increasing the throughput of computation. GPUs are a good example of high-performance architectures, with massive parallelism, high memory throughput, and high performance/watt. The goal of this work is to improve the response time of search by using GPUs.

Kaldewey described an algorithm for parallel binary search. Divide the data sets, find which set contains the search query, and then redistribute the subsets of this set to the processors since it is the only set worth searching. The runtime of this algorithm is log_p(n), where p is the number of processors, as opposed to log_2(n), assuming that redistribution and lookup are free. The GPU architecture in question has up to 16 independent streaming multiprocessors (MP), each with eight processing elements. The execution model is SIMT, or single-instruction multiple-thread, with each thread on an SM executing the same code. The problem with the approach as given thus far is that we need the number of queries to be equal to the number of processors or we'll have poor hardware utilization, memory access collisions will slow things down, and the number of memory accesses is log_2(n). More processors lead to more results, but a running time likely to be the worst-case expected running time. The number of memory accesses in the p-ary search algorithm is (p-1)log_p(n), as opposed to log_2(n), but the expected throughput is lower. In practice, however, with large data sets, p-ary search gets 30% performance improvement over binary search because GPUs parallelize memory accesses; this in turn leads to fewer memory conflicts, and p-ary search has a smaller code footprint. Parallelism does have its costs, however, in that there are more memory accesses, but the algorithm scales on the number of GPUs.

The conclusion is that there is a tradeoff between response time and throughput, but p-ary actually improves both. Future work includes targeting other parallel architectures, evaluating more complex functions, optimizing data structures, and integrating with the rest of resource management in the system (when to parallelize, how much to parallelize, which architecture to use). Rajesh from Berkeley asked at this point how much it costs to do insertions using this scheme, and Kaldewey said he wasn't sure but that he envisions just using the GPU as a consistent cache of the data. Rajesh then asked how to partition the index over processors, and the response was that it's data dependent. A number of database people feel that more cores are simply a waste due to the memory wall problem. Paul McKenney asked whether Kaldewey would expect better or worse results for other things like pattern matching, and the response was that they saw good speedups on parallel scan. Hans Boehm asked why they didn't use interpolation search, and Kaldewey said they haven't looked at it. María Garzarán asked whether there was anything missing on the GPU he'd like to have, and Kaldewey responded that he re-

ally misses dynamic memory allocation and would like better synchronization primitives. It's not currently possible to make hash tables, and a better programming environment would be useful. He didn't have any preliminary results to share with different data structures.

PANEL: PARALLEL COMPUTING IN REAL-TIME INTERACTIVE MUSIC AND MEDIA COMPUTING

Summarized by Rik Farrow (rik@usenix.org)

David Wessel, Center for New Music & Audio Technologies, University of California, Berkeley; David Zicarelli, Cycling 74; Miller Puckette, Department of Music, University of California, San Diego; Amar Chaudhary, Digidesign; Dinesh Manocha, Department of Computer Science, University of North Carolina

David Wessel explained that live sound produced by computers has extreme real-time requirements. Video, even at 30 frames/second, can drop frames without a person noticing, but a much shorter audio lapse gets perceived as a click or pop. Wessel designs and plays instruments that use a computer for sound "rendering." One of his designs, the SLABS, consists of many multi-touch sensitive pads. A 20" by 20" array of pads consists of 100 taxels/inch with 12 bits of sampling data per taxel, a sampling rate of 10 kilohertz, for a total of 4.6 gigabits per second. You can hear an example of Wessel explaining and play the SLABS here: http://www.youtube.com/watch?v=q_mtCZqN0Ms.

Wessel mentioned that real-time sound has applications other than performance, including many channel audio systems and hearing aids.

Amar Chaudhary of Digidesign (the makers of ProTools studio sound software) showed off Open Sound World (http://osw.sourceforge.net/html/note/PlayScore.html). Like Max/MSP (described later), OSW allows composers to put together executable objects (shared objects) that transform their inputs. The inputs can be chained together as well as work in parallel. There are state variables as well as activation expressions triggered by variable changes. Activation expressions can be functions or code similar to C++, and there are 250 transforms on OSW.

Chaudhary was the first person to demonstrate Max/MSP, a GUI that looks a little like a digital representation of a soundboard, with the addition of "patches," objects that process sound.

OSW includes implicit parallelism, making this and other audio software natural users of future multicore processors. Chaudhary pointed out that the difference between using a single and two cores on his MacBook Pro was only 3–4% less CPU usage. David Wessel mentioned that his Mac usually runs at 80% CPU during performances and has as many as 16 different patches (for guitar players, think parallel effects) going at once.

Puckette Miller, the author of Max/MSP and later of the open source pD (Pure Data), used Max/MSP to demonstrate how you could have 15 oscillators and 64 channels at the same time. Sasha Fedorova asked about algorithms and data. Miller responded that there are two worlds, the outside world and the CPU world. Steve Johnson wondered, since most OSes are not real-time, how significant would it be to get an email during a performance. Wessel answered that you disconnect your network during performances and don't use software that does garbage collection. Another panelist said that things should sound exactly the same way every time, leading Vakrim Adve to ask if there can be some slippage, some non-determinism. Chaudhary answered that things should be bit-accurate every time.

Miller mentioned that the UCB ParLab people present understand what happens when you try to parallelize multiple streams. In both Max/MSP and pD, you might have an array of floating-point numbers representing a stream you'd like to add to, as you are using it to create sound. But this implies sharing the data between two processors, which you can't do in a general programming language.

David Zicarelli, the current support person behind Max/SMP (see http://www.cycling74.com/products/), gave a quick demo of Max/SMP.

Dinesh Manocha, of the University of North Carolina, went last. Unlike the other presenters, he is not a musician or a music software designer. Manocha explained that his work involves rendering sounds in virtual environments. Sound reflects off surfaces as well as diffracting around edges, making any accurate rendering very much like 3D image rendering. Applications of this work include modulating, for example, cabin noise in airliner design, as well as in games. Game consoles allot no more than 5% of CPU for sound rendering, which means that most games have primitive sound.

Manocha played several demonstrations of moving through virtual environments. As the virtual position changed, so did the quality of the sound. In a cathedral demo, he dramatically changed the apparent size of the room using altered sound absorption. His work cannot be done in real time, as it involves petaflop computation that handles only mid-range frequencies. He also showed demos of a ball dropping into water and raindrops, without then with sound to demonstrate how much sound adds to human perception.

Wessel said that one can make beautiful sounds but you need to be able to control them in order to perform. Chaudhary agreed and said that real time and control were the biggest challenges faced at this point. Miller suggested that audio programmers should not use threads but different address spaces. Zicarelli said that the control algorithms are very simple, but they are really the bottleneck as everything goes through them, and part of the challenge is trying to apply all these techniques. Manocha mentioned using GPUs to process sound, but said that we have no idea of the latency of GPUs. Latency, which must be less than 5ms, is always going to be the challenge.

OPERATING SYSTEMS AND MIDDLEWARE

- ***Tessellation: Space-Time Partitioning in a Manycore Client OS***

  Rose Liu, Kevin Klues, and Sarah Bird, University of California at Berkeley; Steven Hofmeyr, Lawrence Berkeley National Laboratory; Krste Asanović and John Kubiatowicz, University of California at Berkeley

  Summarized by Ben Hindman (benh@cs.berkeley.edu)

Rose Liu argued that space-time partitions should replace processes as the main abstraction for new "client" operating systems. She defined "client" operating systems as those that are single-user; run a heterogeneous mix of interactive, real-time, and batch applications; and are battery (power) constrained. A new client operating system was needed because existing operating systems were not designed for parallel applications. Furthermore, those operating systems that were designed for parallel execution mainly address server and HPC workloads, not client workloads.

Rose proposed that cores, memory, and even network bandwidth can be partitioned. Alexandra Fedorova asked how spatial partitions differ from Solaris zones. Rose believed that zones were more of a logical partitioning than a physical one. She spelled out the benefits of spatial partitioning, including that it was a natural unit for fault containment and a natural unit for energy management, and it allows two-level scheduling, i.e., partitions can schedule themselves. Alexandra Fedorova asked what happens when one partition uses a library that wants to schedule itself. Rose deferred to the next talk (Lithe) for a solution.

Rose went on to explain how partitions allow operating system services to be put into partitions, similar to microkernels. The space partitioning is probably not enough, so the authors propose space-time partitioning. The time multiplexing is done at a much coarser granularity, which alleviates some of the overhead of context-switching an entire partition. Rik Farrow asked if partitions were created by pinning threads to resources. Rose replied that threads are not the abstraction used within partitions (or at least not the default abstraction), and suggested looking at the abstractions discussed in the upcoming talk (Lithe). Farrow followed up by asking how data in the cache suffers when you do the space-time partitioning discussed. Alexandra Fedorova wanted to know how we can even attempt to partition a cache. Rose proposed hardware support for cache partitioning. Vikram Adve asked what happens if we don't get such hardware support. Alexandra Fedorova proposed some form of software partitioning (e.g., page coloring).

Rose then explained that the fundamental communication primitive across partitions is a form of message passing. Stephen Johnson asked what happens when a message is sent to a partition that is not scheduled. The speaker said they are investigating mechanisms (such as priority inversion) to wake up partitions that have pending messages. Alexandra Fedorova suggested an operating system that could observe communication patterns and then gang-scheduling those partitions that communicate with one another. Rose agreed that this might be a promising idea. Michael Linderman asked if the authors planned to support legacy applications. Rose responded that they are considering running VMs for legacy OSes and applications, but that was not their immediate goal. Stephen Johnson suggested that if they could get the software to perform fairly well, the hardware community would follow suit and produce the hardware needed for a parallel operating system like this.

Alexandra Fedorova asked how the system would respond to changing demands of applications. John Kubitowitz suggested that client devices are fairly bursty, so requirements might change between 1000 cores and two cores. Krste Asanović said that sometimes it might make sense to just keep execution resources within the partition until they are needed again rather than changing partition sizes as frequently. An unidentified audience member suggested Rose look into cluster-aware managers like SLURM. Rob Schreiber asked what happens when the operating system can't figure out a good way to schedule the partitions (because, for example, the constraints are unsatisfiable). Krste Asanović suggested that the system would have to perform some conservative approximation to handle those cases.

- ***Lithe: Enabling Efficient Composition of Parallel Libraries***

  Heidi Pan, Massachusetts Institute of Technology; Benjamin Hindman and Krste Asanović, University of California, Berkeley

  Summarized by Leo Meyerovich (lmeyerov@eecs.berkeley.edu)

Heidi Pan said that Lithe is meant to address the performance problem of composing parallel applications. Various parallel frameworks are well suited for various parallel problems, but many applications consist of heterogeneous problems for which different libraries are suited. Furthermore, this composition is increasingly hierarchical, such as a machine learning library splitting off tasks where each task might be a BLAS (Basic Linear Algebra Subprogram) routine. Naively, these libraries assume full control of the machine to do many of their optimizations. Previously, developers could often also assume full control and knowledge of a machine at design time; the expert could successfully tune the partitioning of resources through multiple layers. However, this is not abstracted well enough for mainstream development, bigger projects, or when there is limited design-time knowledge of the deployment environment. Worse, there is a composition problem: a developer calling into a library must tune resource allocation all the way down the stack.

Lithe is an ABI for cooperative resource allocation within large programs that use different libraries (that, in turn, may also be large, etc.). It is envisioned as sitting on top of the Tessellation OS, moving allocation (if desired) into the application. The proposal is three-part. First, it asserts that hardware threads (HARTs) should be reified as a resource that applications should be able to manipulate. For example, a core with two threads would have two HARTs active at any time step, and each HART is owned by only one com-

ponent. Second, frameworks should be able to cooperatively exchange HARTs (and, potentially, other resources). Unlike other proposals (e.g., Charm), the integration is low at the ABI level, so the team is already able to support systems like TBB and OpenMP. Third, not everything needs to be scheduled cooperatively—but this pushes the decision to framework writers (who might implement such alternatives). For their results, the team showed that an untuned application struggled without cooperative allocation, but a tuned one did much better. The Lithe version ran a little faster than a manually tuned version.

Jim Larus asked if interference says something about the design of libraries (e.g., hidden parameters of number of threads). Pan answered that today, you can typically assume control and expert programmers want to do this tuning. We're seeing interference now that the scenario is changing. Someone asked about the Charm++ abstraction of virtual processors. Pan answered that they only build Charm on it, but we're also concerned with supporting other codes—we have a similar philosophy but a different route. Another person asked what the difference is between a HART and a processor. Ben Hindman answered that by making the HART an abstraction, we can do space-time partitioning. Someone wondered what happens when an agent wants a resource and doesn't get it. Pan answered that it will have to keep asking. Someone else wondered how many apps in the consumer space require this type of support. Pan replied that they have a white paper that shows that a lot of gaming, etc., domains exhibit these properties. Another person asked whether Lithe introduces composability issues, e.g., makes deadlocks more likely. Pan responded that in terms of synchronization, the runtime systems get to handle it (or you can use our own), decreasing the risk.

- ***Energy-efficient Parallel Software for Mobile Hand-held Devices***
  Antti P. Miettinen, Nokia Research Center; Vesa Hirvisalo, Helsinki University of Technology

  Summarized by Leo Meyerovich (lmeyerov@eecs.berkeley.edu)

Miettinen is interested in providing performance and energy simulations for heterogeneous mobile devices for developers. Such devices have many components, such as GPUs, CPUs, and radios, and some optimizations for one component (e.g., slowing down the CPU) might affect another (e.g., running the wireless card longer than desired). An example was presented of running various naive multi-threaded sorting algorithms where one or two didn't scale, showing that it's important to tune.

The proposal is to build a software simulator, parameterized by a machine model, that can run a mobile application and show speed and energy performance. It is still in the motivation and planning stage, and Miettinen asked for input from the workshop participants, both now and later.

Someone agreed about the existence of the problem and suggested looking at various groups interested in it, such as the RAMP project and various projects at Microsoft and Samsung. Another person suggested that scratchpads and alternative architectures are important. Finally, someone wondered if they considered components singly or together in performance and whether there is monotonicity. Miettinen said that there can be nasty interactions: you might lower voltage/frequency to lower energy, but if you're doing data transfer you don't need this, which might have an effect on the wireless interface, losing the benefits from the CPU. They try to find problems like this early on.

TRANSACTIONAL MEMORY

Summarized by Ben Hindman (benh@cs.berkeley.edu)

- ***Lightweight Software Transactions for Games***
  Alexandro Baldassin, State University of Campinas, Brazil; Sebastian Burckhardt, Microsoft Research, Redmond

Alexandro Baldassin discussed the desire to exploit multicore/manycore hardware for better performance without sacrificing software engineering principles, and he hypothesized that software transactional memory (STM) might be a means to achieve this. To test this hypothesis, Alexandro proposed applying STM to a multi-threaded game. STM applies well to games because of the complicated interactions of threads with lots of shared data structures that make locking rather difficult.

In their first attempt at using STM they simply turned critical sections into atomic blocks. They claimed that this still made code too difficult to maintain, because they had to remember which functions inside versus outside transactions, and they had to perform careful copying of private versus shared data in and out of critical sections. Moreover, they claimed that it was still difficult to guarantee atomicity of what they called "tasks," because a task may have multiple critical sections. In their second attempt, they made entire "tasks" be transactions. This avoided tricky code maintenance issues, but it resulted in horrible performance (too many conflicts).

Alexandro suggested that most programmers want coarse-grained transactions that can perform I/O and provide strong atomicity. He recognized, however, that it may be very difficult to get performance given the above requirements. Alexandro next described their STM-like framework. Unlike STM, tasks in their framework are never rolled back, which means they can freely do I/O. He explained that the execution of tasks is atomic and isolated, but there are no serializability or linearizability guarantees.

Dhruva Chakrabarti asked how this system can guarantee the absence of deadlock without rollback. Alexandro explained that rollback is only necessary for handling conflicts, not deadlock, and he described the mechanisms for resolving conflicts without rollback. Micah Best asked how exactly a programmer might decide how to handle many updated conflicts. Alexandro explained that the programmer only gets to resolve pair-wise conflicts. In the event

of many conflicts all at once the programmer will only be presented with two at a time, and the programmer will have to decide which one to propagate only based on those two.

- ***Exceptions and Transactions in C++***
  Ali-Reza Adl-Tabatabai, Intel Corporation; Victor Luchangco, Virendra J. Marathe, and Mark Moir, Sun Microsystems Laboratories; Ravi Narayanaswamy and Yang Ni, Intel Corporation; Dan Nussbaum, Sun Microsystems Laboratories; Xinmin Tian and Adam Welc, Intel Corporation; Peng Wu, IBM Research

Ali-Reza Adl-Tabatabai described the current state of the world regarding software transactional memory (STM). He limited the scope of his discussion to exception handling within a software transaction. He presented the following example:

```
atomic {
        x++;
        if (cond)
                throw MyException();
}
```

and posited the question: should the update to x be committed? Ali then discussed both sides of the argument: commit-on-exception vs. abort-on-exception (rollback).

The commit-on-exception has the benefit of being simpler to implement as well as having more sequential-like semantics (or even global lock-like semantics). However, if you commit rather than abort, you might actually break an invariant that some critical section is supposed to maintain, especially if it is because an exception is thrown that the programmer wasn't expecting.

The abort-on-exception handles the broken invariant issue, but it raises another weird issue involving the propagating exception. Specifically, what if you capture some state in the exception that gets propagated, yet you rollback that state before the exception propagates?

Ali proposed that the right solution is to have both and let the programmer decide what they need, and he suggested that the only point of contention between the commit-on-exception and abort-on-exception camps now is what the default should be. An audience member said that there should be no default, and every programmer must specify what they want. Ali decided to hold a vote. A majority of the audience agreed that there should be no default.

Leo Meyerovich asked how prepared the community is for STM standards like this and how close STM is to being an actual product where the standards will be really important. Ali suggested that it was still very much a work in progress and he hopes that lots of programmers will attempt to use their STM implementation (with these standards) so they can learn from their mistakes and make them better. Dave Patterson asked if the problems regarding exceptions and STM were specific to C++. Ali explained that the problems were not C++ specific, and applied just as well to languages like Java.

- ***Transactional Memory Should Be an Implementation Technique, Not a Programming Interface***
  Hans-J. Boehm, HP Laboratories

Hans Boehm reminded the audience why locks are hard to use. Specifically, he targeted deadlocks as being a major downfall to the use of locks. Hans suggested that an obvious, although strawman, solution is to use a single (reentrant) global lock. He argued this eliminated lock-based deadlocks as well as the need to distinguish between strong and weak isolation and the need to worry about irreversible I/O actions.

Robert Bocchino asked how a global lock actually provides strong atomicity (strong isolation). Hans explained that a key assumption is the absence of data races and, therefore, sequential consistency of the possible interleavings.

Hans went on to ponder whether a global lock-like model will ever get good performance or scale. He suggested that one can use software transactional memory to attempt to implement this global lock-like semantics, but some transactional memory-like constructs might not be admissible with such semantics. For example, implementing something like the retry construct will be difficult, if not impossible. He suggested relying on locks and condition variables for this type of construct instead.

Rob Schrieber asked how exception handling might be done with the global lock semantics. Hans said that the right thing is the commit-on-exception model, where the programmer will have to deal with fixing any broken invariants manually. Jim Larus asked how valuable something like atomic blocks really is for programmers. Hans reiterated that they relieve the burden on programmers to have to avoid deadlocks, but he felt only time and experience will show how valuable they really are.

MODELS AND PARADIGMS II

Summarized by Micah Best (mbest@sfu.ca)

- ***New Abstractions for Data Parallel Programming***
  James C. Brodman, University of Illinois at Urbana-Champaign; Basilio B. Fraguela, Universidade da Coruña, Spain; María J. Garzarán and David Padua, University of Illinois at Urbana-Champaign

After James Brodman introduced the topic of the talk, that of extensions to and new techniques for data parallelism, an audience member asked whether task parallel programs were scalable. Brodman replied that task parallel programs may be redefined as data parallel programs. He outlined the advantages of data parallelism in terms of programs with data parallel operators. These programs will be a sequence of data and there is an extensive collection of data parallel operators that allow expression of parallelism but are not designed explicitly for control.

Brodman then began a detailed description of an instance of the suggested techniques, a method to explicitly parti-

tion array data called the hierarchically tiled array (HTA). Their approach was to make tiles first-class objects in the language to recognize the importance of tiling in terms of control. Someone asked about the uniformity of tile sizes. Brodman responded that tiles could be non-uniform.

Higher-level HTA operations include element-by-element operations such as reduction, circular shift, replicate, transpose, MapReduce, etc. Additionally, programmers can create new complex parallel operators through the primitive hmap. The operators in their library were sufficient naturally to implement several programs from a number of benchmark suites. The results compared favorably in terms of efficiency, and Brodman noted that HTA notation also produces code that is compact and more readable. Someone asked about the methods of communication for the library. Brodman answered that communications were done in MPI, which was hidden from the programmer.

Brodman pointed out that although HTA worked well for numerical programs, many programs are not numerical. There was a need to identify the data parallel operators and data structures needed for other data structures. Sets were identified as a target for this inquiry and Brodman outlined what would be needed to support this. Sets would require operators such as map, union, and reduce. Their research had extended to studying several applications, including search, data-mining, and mesh refinement.

The next segment of the talk detailed an example of data parallel search in the form of the "15 puzzle," a 4 by 4 grid with a single hole. A model for search and a process were then detailed. The effectiveness of tiling was the same as for arrays by emphasizing locality and parallelism; however, tiled sets are not created as easily as tiled arrays. The talk concluded with ideas for enforcing determinacy through map primitives or annotations for atomicity. The benefits of data parallelism for portability and parallelism were reiterated. Finally, sets were discussed again as a promising data type for further research.

Someone asked about the size of tiles and the depth of hierarchy. Brodman responded that these parameters would be set by the programmer. Who was the target audience, in terms of programmer expertise? The "average programmer" would receive the data types that would have been implemented in turn by experts who would produce highly tuned code. Could tiles from different data structures be tied together? They hadn't looked into that yet, but he could see it as a possibility as long as the data structures were amenable. A final question concerned encapsulating atomic sections. Brodman said they were looking into it.

- ***Ease of Use with Concurrent Collections (CnC)***
  Kathleen Knobe, Intel

Knobe's research goal was to create a separation of concerns between the domain expert and the tuning expert. She admitted that this had not been completely achieved, but there was positive movement in that direction. The problem was that most serial languages over-constrain orderings, while most parallel programming languages are embedded within serial languages. The solution is to isolate roles and to raise the level of the programming model just enough to avoid over-constraints. Two ordering constraints were identified: producer/consumer constraints for dataflow dependencies, and controller/controllee for control dependencies.

The design of Concurrent Collections (CnC) was informed by streaming and tuple spaces. From streaming came the concept of associating data items with computational steps, labeled with control tags. Tuple spaces inspired the tagging of each instance for independent scheduling. To illustrate these concepts she provided a simple example of filtering strings. This system of tagging relies only on application knowledge and does not require considering parallelism. Despite this, the results are still parallel, deterministic (with respect to results), and race-free. She then described the execution model of how tags were used to schedule instances.

Knobe introduced dataflow as the third influence. An audience member asked her to compare CnC and the Linda language and the relative restrictiveness of the two. Knobe answered that CnC does not require streams and they were careful not to make that constraint. Linda produced a result where, in Knobe's words, a computation just "sits there," whereas CnC is dynamically scheduled and also allows specification of control flow. She did note that there was a slight constraint in terms of syntax in only allowing deterministic programs and having single assignment.

She then offered another example, a "cell tracker," presenting a CnC graph that fully captured all the information needed to parallelize the application. The system supports not only different schedules but a wide range of runtime systems. There are many options in the back-end for tuning, since the only thing provided by the program is the constraint. John Kubiatowicz pointed out that there are no data-ordering constraints. Knobe responded that there are the two kinds of constraints already specified and that the domain expert has to know the producer-consumer relationships in the program. Another audience member asked about allowed data types such as arrays. Knobe responded that any serial code was a candidate for CnC and that data items can be of any type. This was followed with an inquiry into the feasibility of handling trees. Knobe answered that they used them all the time.

The discussion of the CnC implementation continued with a description of the various back-ends available. CnC performance results were roughly equivalent on multicore systems to those obtainable with Intel TBB (Thread Building Blocks) or OpenMP. Someone asked about the gains in performance by CnC over p-threads in a dedup, one of the benchmarks tested. Knobe was not sure, as she didn't write the application. To another similar question comparing performance results to TBB, Knobe pointed out that the overheads were unknown, applications tend to vary, and there are differences in scheduling. How does developer time vary between TBB and CnC? Anecdotally, developers have far preferred

CnC to TBB. In response to questions about code reuse she added that both code and frameworks were amenable to reuse. Additionally, reuse could be accomplished by linking graphs.

Motohiro Takayama asked about a development environment (IDE) for CnC. Knobe said that they hadn't yet looked into it, but it needed to be addressed. She would like to see it merged it with a GUI, including both a debugger and visualizer. Romain Cledat asked what issues still remained between the domain and tuning expert. Knobe responded that issues such as grain size, support for tiling, and similar facets still needed to be exposed. She would like to see those made a little easier.

- ***Optimizing Collective Communication on Multicores***
  Rajesh Nishtala and Katherine A. Yelick, University of California, Berkeley

Rajesh Nishtala noted that as core counts continue to grow and application scalability takes the center stage, it is quickly becoming infeasible to support uniform access to shared memory. An audience member wondered whether there was a limit, as sometimes applications simply don't need to go faster. Rajesh agreed, but this research was focused on high-performance applications. The discussion then focused on a product of the research, the Partitioned Global Address Space Language. The central concept is to expose the idea of locality to programmers, a technique that has proven successful in distributed memory.

Nishtala discussed collective communications, which involves an operation called by many threads to perform globally coordinated communication. Interfaces to the collectives, used as parallel communication building blocks, are typically delivered through a software library and exposed in modern programming languages. Two categories of communication were defined: one-to-many and many-to-many. The focus of the work was given as reducing one-to-many and optimizing the many-to-many pattern with barriers. Example trees were given with barrier performance results. Fast barrier enables finer-grained synchronous programs. Optimizing collectives for shared memory allows the programmer to do finer-grained synchronous programs.

Potential synchronization problems were then discussed, to highlight the need for strictly synchronized collectives. These may be alleviated by using synchronization before and after the collective and enforcing a global ordering of the operations. The collective is considered complete once all threads have the data.

In conclusion Rajesh reminded the audience that future systems will certainly rely on NUMA, underscoring the need for this type of research. Application scalability will take center stage. Tuning collectives for latency of throughput can lead to significantly different algorithmic choices, necessitating passing the requirements to the collective library.

Someone asked whether the type of communication was to be specified by the user, if this was a "tuning issue." Rajesh responded that the collective library is designed to be part of the runtime library, capable of detecting a situation where loosely synchronized collectives are applicable. Another question involved a particular comparison with p-threads in the given results. Barriers using p-threads had taken 3ms on the Niagra. As a possible explanation, Rajesh noted that p-threads assumes more threads than cores. When the resources are not over-subscribed, the overhead becomes detrimental.