Process Coordination for Commodity Systems

Research Proposal
Eitan Frachtenberg

December, 2004

Abstract

Parallel and distributed processing are no longer the exclusive realm of supercomputers. The growing prevalence of systems with multiple processing units brings parallel hardware to commodity computers. Parallel hardware cannot be fully utilized unless running parallel software, which in turn depends on the operating system's ability to support various, often conflicting scheduling requirements. This proposal describes research toward achieving a fully flexible autonomous operating system (OS), that seamlessly supports the entire range of current and future applications: serial, multimedia, interactive, distributed, and parallel.

Problem Description

Computer manufacturers are constantly striving to increase the performance of their products. By enlarging cache sizes, improving memory bandwidth and latency, clocking processors at higher speeds, and incorporating new architectural features, computer performance grows at an exponential rate. Nevertheless, we are currently witnessing a slowing down of this process, and truly innovative architectural features are becoming increasingly rare. For example, the rate of increase in processor speeds has declined over the last few years [35, 54]. Even Moore's Law [40], which predicts the doubling of processors' transistor count every 18 months, is expected to lose its validity sometime in this decade or the next, because of increasing power requirements [2, 26, 29, 31].

To try and sustain the same rate of growth in computer performance that has been witnessed in the past, manufacturers have been turning to parallelism [28, 33, 34, 61, 62]. One example is *symmetric multiprocessing* (SMP), which in the simplest instantiation consists of two or more main processors (CPUs) running in one box [38]. Recently, one manufacturer began offering "deskside workstations", supporting up to 96 processors in one box, with no special cooling requirements [48]. Other forms of in-box parallelism include *superthreading* [66], *Simultaneous Multithreading* (SMT) and *hyperthreading* [13, 67], and *chip multiprocessors* (CMP) [27].

The most actively developed form of parallelism is multicore chips, which allow a single chip die to contain multiple processors, and will soon be offered

by all major CPU manufacturers [23, 36, 56, 61]. Processor roadmaps plan for 2-, 4-, and 8-core processors in the near future, with an additional layer of parallelism by including several such chips in one node [32, 33]. These computers are increasingly being used for a variety of applications, running the gamut from office applications to heavily loaded server applications.

The advances toward commodity parallel hardware are meaningless without the software to use it [62]. Redundant computing capability does not improve the performance of serial programs. What it does allow, however, is the use of concurrent serial and interactive programs, distributed programs, parallel programs, and combinations thereof. Many software authors already incorporate support for such architectures in their performance-hungry applications, e.g. graphic design and animation, web servers, and database engines.

With the newfound versatility in classes of programs that run on low- and mid-end computers comes the connoted requirement for OS pliability, and in particular regarding process scheduling policies [6, 37, 47, 60]. For example, interactive programs such as video players require near-real-time responsiveness from the OS to avoid image skipping [14, 24, 45]; tightly coupled parallel programs require all the synchronizing processes to be coscheduled concurrently [17, 22]; distributed programs have minimal interaction with other processes and require less coordination [5, 65]; and serial (sequential) programs pose no scheduling demands other than the normal local considerations, such as fairness and resource overlapping.

Unfortunately, the significant advances in hardware were not matched by similar performance advancements in OSs [49]. Most common desktop operating systems, such as Windows, GNU/Linux, and MacOS, offer only rudimentary support for parallel hardware, and little or no support for parallel and distributed software. Most desktop OSs use a simple scheduling scheme that has changed very little in 30 years [15, 57]. None of these operating systems offers mechanisms to explicitly or implicitly identify the special requirements of all types of programs and schedule the machine's resources accordingly.[1] The lack of support for collaborative programs leads to significant degradation in their performance and to underutilization of the computer's resources [17, 21]. This predicament in turn hinders the widespread adoption of collaborative programs in the desktop environment, rendering the advances in hardware parallelization largely moot, and creating a cycle of stopped development. Furthermore, scheduling policies which are tailored specifically for parallel jobs are exposed to the woes of Amdhal's Law [3] (which determines that the scalability of a parallel program is limited by its serial parts) rendering the machine again underutilized.

To address these problems, the goal of this research is to develop performance-oriented, practical scheduling policies for commodity systems that support any class of applications and workloads.

[1] There is a significant body of work concentrating on implicit or explicit identification of multimedia tasks, and some work on identification of parallel/distributed tasks [4, 5, 8, 9, 12, 15, 18, 19, 24, 46, 51, 69, 72]. However, these studies make a clear distinction between commodity and high-end systems, which may no longer be always valid.

Goals and Objectives

The vision behind this proposal is a desktop/workstation operating system that automatically identifies the scheduling requirements of any type of process, and employs optimal scheduling heuristics to match those needs. To this end, the successful outcome of this research will be a set of kernel scheduling policies that satisfy the following objectives:

Performance: The main drive behind this proposal is the inadequate performance of certain kinds of processes with commodity operating systems, in particular those of parallel and distributed programs. This research should provide the operating system with mechanisms to identify scheduling performance problems, as well as the best-known policies to successfully deal with these problems.

Transparency: One of the important differences between commodity and specialized systems is the degree of operator involvement in configuration and tuning. Unlike a supercomputer OS, a commodity system should require as little user intervention and expertise as possible while still addressing changing needs. Several OS mechanisms were proposed in the past to optimize the scheduling of interactive processes or parallel processes with user-supplied hints [4, 9, 12, 24, 46, 51]. The philosophy behind this proposal is that the OS should monitor processes to obtain its own information on scheduling requirements, eliminating or reducing the need for the user to do so. It should then be flexible enough to allow for *any* type of application to run efficiently alongside different applications.

Expandability: Job scheduling is an active field of research, both for the commodity and high-end environments [16]. Scheduling policies are continuously proposed, developed, and investigated. To allow an OS to be current with recent developments, as well as adaptive to future ones, it should allow for a simple mechanism of "plugging-in" and evaluating new scheduling policies. Moreover, by designing expandable scheduling mechanisms for the OS from the start, this research can also provide a fertile ground for future studies by other researchers that can benefit from the available infrastructure.

Portability: Modifying operating systems is a difficult task, often resulting in nonportable code that only applies to a certain OS or even a specific OS version. The scheduling policies and mechanisms stemming from this work must be as general and separate from specific implementations as possible, possibly relying on external kernel modules and minimal interfaces.

Scalability: Since the trend is to increase the level of parallelism per node, an OS should scale gracefully with the number of processors, while avoiding the use of locks to the largest extent possible.

Scope of Research

The following is a list of issues to be addressed, in approximate proposed chronological order. Not all of the topics are expected to be studied at the same level of depth. It should be noted that the extensively studied and more theoretical topics of scheduling, such as offline scheduling, are not in the scope of this work.

Benchmarks, Metrics, and Evaluation

One of the main methodological challenges in the evaluation and development of online scheduling algorithms is the lack of standard metrics and benchmarks. More often than not, scheduling studies focus on very specific workloads that yield results that cannot be generalized to (or fare less favorably when compared to) many other workloads. Before developing novel and improved scheduling techniques, there is a need for a more general way of measuring any current problems and future improvements. To this end, a set of benchmarks and associated metrics needs to be developed that is meaningful in a large spectrum of scheduling scenarios and workloads. The premise behind this set will be that for different workloads, and even for different applications, some metrics have meaning, and others not. For example, slowdown is a useful metric for batch processes, but useless for continuous media. With this in mind, any benchmark set needs to be comprehensive enough to capture workloads and metrics that are meaningful to a large audience.

Evaluation of scheduling algorithms can be done both in a simulated environment, for ease of development and parameter space study, or in actual implementation, for realistic exposure tuning and performance issues. Simulations might possibly employ tools such as the Bossa scheduler simulation platform [6], the SimOS simulator [53, 71] or User Mode Linux [68]. Actual evaluations however will require different tools. To make meaningful measurements and statements on OS performance, extremely precise and low-latency performance measurement techniques need to be employed. Such techniques may include using hardware counters and specialized performance measurement tools [14, 52, 70]. Furthermore, having a comprehensive set of measurement tools, existing OS schedulers can be evaluated and compared to each other and to novel schedulers.

Scheduling Policies and Techniques

The main thrust of this work is the development of novel scheduling policies that can expose the full potential of emerging commodity platforms and work equally well with a plethora of different applications and workload characteristics. The following six principles, which are not widespread in contemporary schedulers, will be studied in the context of the hardware and software trends described above.

1. **Cooperative scheduling:** Scheduling multiple processes together on different processors can have significant performance repercussions. Instead of an expected speedup resulting from parallelism, processes can actually suffer from slowdown when mis-scheduled [10, 59]. One example is when competing, memory-intensive processes create severe contention on shared caches and memory buses [43]. Another example is embodied in fine-grained synchronous processes that suffer significant slowdowns if not coscheduled [17]. The goal of cooperative scheduling is to maximize the collaboration between those processes that benefit from it, while minimizing the contention between mutually competitive processes. The basic idea is that the OS continually tries to coschedule different processes together, and observes the effect on their progress, then schedules processes in the configurations that are most beneficial to them. The progress of a process can be measured by various metrics, that need to be studied and selected from, and that might employ fine-grained hardware counters. Examples of such possible metrics include CPU time and cache misses [55], or the joint "symbiotic throughput" [58] and "weighted speedup" [60], wherein the scheduler also has an initial phase of testing out different combinations.

2. **Classification of processes:** The OS can use an abstraction level to assist in scheduling decisions by classifying processes based on their scheduling and synchronization requirements. This approach proved itself to be highly effective on various scenarios in the parallel job scheduling arena [21, 22]. The same principles can be generalized to include the synchronization needs in the commodity system, of parallel and serial processes alike. For example, continuous media processes can be classified as such, and the scheduling algorithm adjusted so that they receive control of the CPU every time they need to express the next frame of output in a timely manner, and no more. Another way of looking at multimedia processes is as fine-grained synchronous parallel processes running contemporaneously. With this perspective, multimedia processes comprise producers and consumers (e.g. one processes to read raw video data, another to convert the data to the display parameters, and a third to put the data on screen, all communicating at 30 frames per second). As such, these processes exhibit the same characteristics of classic parallel programs, and impose similar synchronization requirements [72]. The classification is again obtained by measuring various process metrics, including the interaction of each process with other processes and I/O devices. This classification could, and probably should, be revisited over time, to account for changing behavior of applications.

3. **Adaptabilty:** Adaptability addresses changing workload and application characteristics. [22, 55], and can be expressed in adaptive timeslice frequency and length, as well as changing scheduling policy. For example, most contemporary OSs use fixed-length timeslices, which hinders time-sensitive, multimedia, synchronous, and parallel applications [24, 41, 45, 50]. Contemporary hardware allows for timeslicing at much

finer granularities than when multiprogramming was first developed [14]. A better approach therefore might be to allow for dynamic timeslices, based on the application requirements. For example, if a media player requires exactly $0.2\,ms$ to display a frame every $1/30\,s$, it could receive a dedicated processor at that frequency, for exactly $0.2\,ms$, thus ensuring smooth playback while avoiding the time wasted on overly long timeslices. Another example might include a parallel program that needs to be gang-scheduled across several processors. The scheduler can allocate relatively long timeslices dedicated for the program, and avoid disruptive timer interrupts for the duration of the timeslice execution.

4. **Scheduling memory:** Most applications tend to retain their characteristic behavior over time and across runs. For example, media players, technical and graphical processing applications, and download managers tend to have similar requirements from the OS across runs, even if their data changes. Therefore, an OS that maintains a profile of applications' scheduling characteristics in nonperishable memory could potentially make correct scheduling decisions promptly. Naturally, characteristics could change over time or across runs, and the OS needs to recognize these changes and adjust the scheduling accordingly. But for the large majority of cases, scheduling memory might significantly decrease the time the OS spends learning the applications and finding an appropriate scheduling policy.

5. **Tuning-free operation:** The choice of scheduling policy along with the parameters used by the policy can conceivably be adaptive and adjust to different types of architectures and workloads. The philosophy remains that users should not be required to tune any parameter, and yet the OS will be able to make different scheduling decisions based on many factors. Examples of these factors include different types of parallelism (e.g. hyperthreading vs. multicore, or single-processor), architectural constraints such as power consumption [44] or memory limits (e.g., synchronous jobs must avoid swapping at any reasonable cost [7]), and application requirements such as prioritized network or screen access [59].

6. **Scalability:** Parallel applications are limited in their scalability and effective utilization by many factors, including their serial parts [3] and network performance [25]. By gracefully handling a combination of serial and parallel processes (that can dynamically change over time), the OS can avoid suffering from specific applications' performance problems, while increasing the system's utilization as a whole. If future systems grow to tens or hundreds of processors, scalability will only be attained if the gathering and use of information remains as localized as possible. Ideally, even with multiple processors, each processor will largely take care of its own scheduling, with little sharing of information, and no global locks. This principle also naturally leads to processor affinity, which in turn increases cache efficiency.

Cache and Data Locality

Memory access costs and latency hiding are increasingly becoming critical performance factors in computers, as processor clock speeds continue to grow at a faster rate than memory performance. The problem is exacerbated with parallelism. Running multiple programs on a single chip, on an SMP, or on any other configuration where processors share any level of memory hierarchy raises contention issues. For that data that is not shared, i.e. resides on separate memory modules, locality issues rise. If scheduled incorrectly, different processes will be competing for memory resources that would otherwise be dedicated to them. A paradoxical situation may occur wherein running a parallel program results in slowdown, rather than speedup [4, 11, 30, 39, 60]. Schedulers for multicore and SMT machines will need to be aware of all levels of parallel execution abstraction, from the thread, through the process, to the parallel program. As part of the collaborative scheduling principle, this study will explore ways for novel scheduling policies to incorporate memory performance considerations.

Heterogeneous and Energy-Aware Architectures

One research approach for future multicore architectures calls for heterogeneous chips, comprising processors of different abilities, performance, and energy requirement. For example, some manufacturers are designing multicore chips with multiple math coprocessors or Java hardware processors [32]. Other examples may include laptop chips, with low-power energy running for low-demand applications, and higher performance chips for bursts of computation [62]. It is likely that the OS will play an important role in allocating these resources efficiently, and will therefore have to factor the heterogeneity into its scheduling considerations. One way in which energy and performance considerations might be incorporated into the scheduling principles depicted above is via a cost function that reflects the priorities of the specific application and/or architecture.

Transparent Support for Cluster Computing

High-performance computing is a dynamic field with a growth rate that outpaces Moore's Law [64]. Almost paradoxically, the most pronounced trend in supercomputing architecture in the last few years is the move toward commodity-based, large-scale clusters. A large share of these clusters use commodity processors, running commodity operating systems. The OS requirements of supercomputers however can be far more demanding that those of commodity machines, in terms of communication, synchronization, and noise elimination [17, 20, 50, 63]. There have been several attempts in the past to enhance commodity operating systems with mechanisms for satisfying these needs [1, 5, 22, 42]. Some of these ideas can be integrated into the classification principle described above, so that the commodity OS automatically (and possibly with the addition of new system calls) recognizes synchronization needs of parallel and distributed programs, and attempts to schedule them accordingly. Such commodity OSs can be integrated into a high-performance cluster with little or no modification.

References

[1] Saurabh Agarwal, Gyu Sang Choi, Chita R. Das, Andy B. Yoo, and Shailabh Nagar. Coordinated co-scheduling in time-sharing clusters through a generic framework. In *IEEE International Conference on Cluster Computing*, Hong Kong, December 2003. Available from `www.cse.psu.edu/hpcl/papers/cluster2003.pdf`.

[2] Vikas Agarwal, M. S. Hrishikesh, Stephen W. Keckler, and Doug Burger. Clock rate versus IPC: the end of the road for conventional microarchitectures. In *27th International Symposium on Computer Architecture (ISCA)*, pages 248–259, 2000. Available from `citeseer.ist.psu.edu/agarwal00clock.html`.

[3] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 30, pages 483–485, April 1967.

[4] Christos D. Antonopoulos, Dimitrios S. Nikolopoulos, and Theodore S. Papatheodorou. Informing algorithms for efficient scheduling of synchronizing threads on multiprogrammed SMPs. In *30th International Conference on Parallel Processing (ICPP)*, pages 123–130, Valencia, Spain, September 2001. Available from `www.cs.wm.edu/~dsn/papers/icpp01.pdf`.

[5] Andrea Carol Arpaci-Dusseau. Implicit Coscheduling: Coordinated scheduling with implicit information in distributed systems. *ACM Transactions on Computer Systems*, 19(3):283–331, August 2001. Available from `portal.acm.org/ft_gateway.cfm?id=380764&type=pdf`.

[6] Luciano Porto Barreto and Gilles Muller. Bossa: A language-based approach to the design of real-time schedulers. In *Tenth International Conference on Real-Time Systems (RTS)*, pages 19–31, Paris, France, March 2002. Available from `www.emn.fr/x-info/bossa/bossa_rts.ps`.

[7] Anat Batat and Dror G. Feitelson. Gang scheduling with memory considerations. In *14th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 109–114, May 2000. Available from `http://www.cs.huji.ac.il/~feit/papers/GangMem00IPDPS.ps.gz`.

[8] Francine Berman. High-performance schedulers. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 279–309. Morgan Kaufmann, 1999. Available from `citeseer.ist.psu.edu/54356.html`.

[9] John Bruno, Eran Gabber, Banu Özden, and Abraham Silberschatz. The Eclipse operating system: Providing quality of service via reservation domains. In *Usenix Technical Conference*, pages 235–246, New Orleans, LA, June 1998. Available from `citeseer.ist.psu.edu/bruno98eclipse.html`.

[10] James R. Bulpin and Ian A. Pratt. Multiprogramming performance of the Pentium 4 with hyper-threading. In *Second Annual Workshop on Duplicating, Deconstruction and Debunking (WDDD)*, pages 53–62, Munchen, Germany, June 2004. Available from `www.ece.wisc.edu/~wddd/2004/06_bulpin.pdf`.

[11] J. Bradley Chen and Brian N. Bershad. The impact of operating system structure on memory system performance. In *14th ACM Symposium on Operating Systems Principles (SOSP)*, pages 120–133, Asheville, NC, December 1993. ACM Press. Available from `citeseer.ist.psu.edu/chen93impact.html`.

[12] Kenneth J. Duda and David R. Cheriton. Borrowed-virtual-time (BVT) scheduling: Supporting latency-sensitive threads in a general-purpose scheduler. In *17th ACM Symposium on Operating Systems Principles (SOSP)*, pages 261–276, Charleston, SC, December 1999. Available from `citeseer.ist.psu.edu/duda99borrowedvirtualtime.html`.

[13] Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, and Dean M. Tullsen. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro*, 17(5):12–18, September/October 1997. Available from `citeseer.ist.psu.edu/eggers97simultaneous.html`.

[14] Yoav Etsion, Dan Tsafrir, and Dror G. Feitelson. Effects of clock resolution on the scheduling of interactive and soft real-time processes. In *SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, pages 172–183, San Diego, CA, June 2003. Available from `www.cs.huji.ac.il/~feit/papers/ClockRes03SIGMETRICS.ps.gz`.

[15] Yoav Etsion, Dan Tsafrir, and Dror G. Feitelson. Desktop scheduling: How can we know what the user wants? In *14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 110–115, County Cork, Ireland, June 2004. Available from `www.cs.huji.ac.il/~feit/papers/HuCpri04NOSSDAV.pdf`.

[16] Dror G. Feitelson. A survey of scheduling in multiprogrammed parallel systems. Research Report RC 19790 (87657), IBM T. J. Watson Research Center, October 1994. Revised version August 1997 available from `www.cs.huji.ac.il/~feit/papers/SchedSurvey97TR.ps.gz`.

[17] Dror G. Feitelson and Larry Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, 16(4):306–318, December 1992. Available from `www.cs.huji.ac.il/~feit/papers/GangPerf92JPDC.ps.gz`.

[18] Dror G. Feitelson and Larry Rudolph. Coscheduling based on run-time identification of activity working sets. *International Journal of Parallel Programming*, 23(2):136–160, April 1995. Available from `www.cs.huji.ac.il/~feit/papers/Cosched95IJPP.ps.gz`.

[19] Ian Foster. Automatic generation of self-scheduling programs. *IEEE Transactions on Parallel and Distributed Systems*, 2(1):68–78, January 1991. Available from `dx.doi.org/10.1109/71.80190`.

[20] Eitan Frachtenberg, Kei Davis, Fabrizio Petrini, Jose Carlos Sancho, and Juan Fernandez. Designing parallel operating systems via parallel programming. In *Tenth Euro-Par*, volume 3149 of *Lecture Notes in Computer Science*, pages 689–696. Springer-Verlag, August 2004. Available from `www.cs.huji.ac.il/~etcs/pubs/`.

[21] Eitan Frachtenberg, Dror G. Feitelson, Juan Fernandez-Peinador, and Fabrizio Petrini. Parallel job scheduling under dynamic workloads. In Dror G. Feitelson and Larry Rudolph, editors, *Ninth Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 208–227. Springer-Verlag, 2003. Available from `www.cs.huji.ac.il/~etcs/pubs/`.

[22] Eitan Frachtenberg, Dror G. Feitelson, Fabrizio Petrini, and Juan Fernandez. Flexible CoScheduling: Mitigating load imbalance and improving utilization of heterogeneous resources. In *17th International Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, April 2003. Available from `www.cs.huji.ac.il/~etcs/pubs/`.

[23] W. Wayt Gibbs. A split at the core. *Scientific American*, 291(5):96–101, November 2004. Available from `www.sciam.com/article.cfm?articleID=00026625-6DF0-1179-ADF083414B7FFE9F`.

[24] Ashvin Goel, Luca Abeni, Charles Krasic, Jim Snow, and Jonathan Walpole. Supporting time-sensitive applications on a commodity OS. In *Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, pages 165–180, Boston, MA, December 2002. Available from `citeseer.ist.psu.edu/goel02supporting.html`.

[25] Anshul Gupta and Vipin Kumar. Performance properties of large scale parallel systems. *Journal of Parallel and Distributed Computing*, 19(3):234–244, 1993. Available from `citeseer.ist.psu.edu/gupta93performance.html`.

[26] Paul Hales. Intel's Grove warns of the end of Moore's Law. `www.theinquirer.net/?article=6677`, December 2002.

[27] Lance Hammond, Basem A. Nayfeh, and Kunle Olukotun. A single-chip multiprocessor. *IEEE Computer*, 30(9):79–85, 1997. Available from `citeseer.ist.psu.edu/hammond97singlechip.html`.

[28] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kauffman Publishers, Inc., 1 edition, 1990.

[29] International technology roadmap for semiconductors. `public.itrs.net/Files/2003ITRS/Home2003.htm`, 2003.

[30] Terry Jones, William Tuel, Larry Brenner, Jeff Fier, Patrick Caffrey, Shawn Dawson, Rob Neely, Robert Blackmore, Brian Maskell, Paul Tomlinson, and Mark Roberts. Improving the scalability of parallel jobs by adding parallel awareness to the operating system. In *15th IEEE/ACM Conference on Supercomputing*, Phoenix, AZ, November 2003. ACM Press and IEEE Computer Society Press. Available from `www.sc-conference.org/sc2003/paperpdfs/pap136.pdf`.

[31] Michael Kanellos. Intel scientists find wall for Moore's Law. `news.com.com/2100-7337-5112061.html`, December 2003.

[32] Michael Kanellos. Designer puts 96 cores on single chip. `news.com.com/2100-1006_3-5399128.html`, October 2004.

[33] Michael Kanellos. Intel expands core concepts for chips. `news.com.com/2100-1006_3-5494714.html`, December 2004.

[34] Michael Kanellos. Intel hastily redraws road maps. `news.com.com/2100-1006_3-5207837.html`, May 2004.

[35] Michael Kanellos. Intel kills plans for 4GHz Pentium. `news.com.com/2100-1006_3-5409816.html`, October 2004.

[36] Rakesh Kumar, Keith Farkas, Norman Jouppi, Partha Ranganathan, and Dean Tullsen. A multi-core approach to addressing the energy-complexity problem in microprocessors. In *Workshop on Complexity-Effective Design (WCED)*, June 2003. Available from `citeseer.ist.psu.edu/kumar03multicore.html`.

[37] Jullia L. Lawall, Gilles Muller, and Luciano Porto Barreto. Capturing os expertise in an event type system: the Bossa experience. In *Tenth ACM SIGOPS European Workshop on (EW)*.

[38] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 229–239, New York, NY, 1986. ACM Press. Available from `citeseer.ist.psu.edu/li89memory.html`.

[39] Jeffrey C. Mogul and Anita Borg. The effect of context switches on cache performance. In *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 75–84, Santa Clara, CA, April 1991. ACM Press. Available from `portal.acm.org/citation.cfm?id=106982`.

[40] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, April 1965. Available from `www.intel.com/research/silicon/moorespaper.pdf`.

[41] Ronald Mraz. Reducing the variance of point-to-point transfers for parallel real-time programs. *IEEE Parallel and Distributed Technology: Systems and Applications*, 2(4):20–31, Winter 1994. Available from `ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=345963`.

[42] Shailadh Nagar, Ajit Banerjee, Anand Sivasubramaniam, and Chita R. Das. Alternatives to coscheduling a network of workstations. *Journal of Parallel and Distributed Computing*, 59:302–327, 1999. Available from `dx.doi.org/10.1006/jpdc.1999.1576`.

[43] David Nagle, Richard Uhlig, Trever Mudge, and Stuart Sechrest. Optimal allocation of on-chip memory for multiple-API operating systems. In *21st International Symposium on Computer Architecture (ISCA)*, pages 358–369, Chicago, IL, April 1994. Available from `citeseer.ist.psu.edu/171399.html`.

[44] Rolf Neugebauer and Derek McAuley. Energy is just another resource: Energy accounting and energy pricing in the Nemesis OS. In *Eighth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 67–74, Schloss Elmau, Germany, May 2001. Available from `www.dcs.gla.ac.uk/~neugebar/pub/hotos01.pdf`.

[45] Jason Nieh, James G. Hanko, J. Duane Northcutt, and Gerard A. Wall. SVR4 UNIX scheduler unacceptable for multimedia applications. In *Fourth ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, November 1993. Available from `citeseer.ist.psu.edu/443381.html`.

[46] Jason Nieh and Monica S. Lam. The design, implementation and evaluation of SMART: A scheduler for multimedia applications. In *16th ACM Symposium on Operating Systems Principles (SOSP)*, pages 184–197, Saint-Malo, France, October 1997. Available from `citeseer.ist.psu.edu/article/nieh99design.html`.

[47] Jason Nieh and Monica S. Lam. Multimedia on multiprocessors: Where's the OS when you really need it? In *Eighth ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 1998. Available from `citeseer.ist.psu.edu/430008.html`.

[48] Orion deskside computer. `www.orionmulti.com/products/`.

[49] John K. Ousterhout. Why aren't operating systems getting faster as fast as hardware? In *Summer 1990 USENIX Conference*, pages 247–256, Anaheim, CA, June 1990. USENIX Association, IEEE Press. Available from `citeseer.ist.psu.edu/165263.html`.

[50] Fabrizio Petrini, Darren Kerbyson, and Scott Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *15th IEEE/ACM Conference on Supercomputing*, Phoenix, AZ, November 2003. Available from `www.c3.lanl.gov/~fabrizio/papers/sc03_noise.pdf`.

[51] Melissa A. Rau and Evgenia Smirni. Adaptive CPU scheduling policies for mixed multimedia and best-effort workloads. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 252–261, College Park, MD, October 1999. Available from `citeseer.ist.psu.edu/rau99adaptive.html`.

[52] John Regehr. Inferring scheduling behavior with Hourglass. In *USENIX Annual Technical Conference (FREENIX Track)*, Monterey, CA, June 2002. Available from `citeseer.ist.psu.edu/regehr02inferring.html`.

[53] Mendel Rosenblum, Stephen A. Herrod, Emmett Witchel, and Anoop Gupta. Complete computer system simulation: The SimOS approach. *IEEE parallel and distributed technology: systems and applications*, 3(4):34–43, Winter 1995. Available from `citeseer.ist.psu.edu/rosenblum95complete.html`.

[54] Stefan Rusu. Trends and challenges in high-performance microprocessor design. Presentation available from `www.eda.org/edps/edp04/submissions/presentationRusu.pdf`, April 2004.

[55] Margo Seltzer and Christopher Small. Self-monitoring and self-adapting operating systems. In *Sixth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 124–129, Cape Cod, MA, May 1997. Available from `www.eecs.harvard.edu/vino/vino/papers/monitor.ps`.

[56] Stephen Shankland. Intel's dual-core Xeon due in 2006. `news.com.com/2100-1006_3-5416330.html`, October 2004.

[57] Christopher Small and Margo Seltzer. VINO: An integrated platform for operating systems and database research. Technical Report TR-30-94, Harvard University, Cambridge, MA, 1994. Available from `www.dogfish.org/chris/papers/tr-30-94.pdf`.

[58] Allan Snavely and Larry Carter. Symbiotic jobscheduling on the Tera MTA. In *Workshop on Multi-Threaded Execution, Architecture, and Compilers (MTEAC)*, Toulouse, France, January 2000. Available from `www-cse.ucsd.edu/users/tullsen/mteac2000/snavely.ps.gz`.

[59] Allan Snavely, Dean Tullsen, and Geoff Voelker. Symbiotic jobscheduling with priorities for a simultaneous multithreading processor. In *SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, pages 66–76, Marina Del Rey, CA, June 2002. Available from `citeseer.ist.psu.edu/528307.html`.

[60] Allan Snavely and Dean M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreading processor. In *Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 234–244, Cambridge, MA, November 2000. Available from `citeseer.ist.psu.edu/338334.html`.

[61] Herb Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), March 2005. Available from `www.gotw.ca/publications/concurrency-ddj.htm`.

[62] Architecting the Era of the Tera. www.intel.com/labs/teraera/, February 2004.

[63] Paul Terry, Amar Shan, and Pentti Huttunen. Improving application performance on HPC systems with process synchronization. *Linux Journal*, (127):68–73, November 2004.

[64] Top 500 supercomputers. `wwww.top500.org`.

[65] Dan Tsafrir and Dror G. Feitelson. Barrier synchronization on a loaded SMP using two-phase waiting algorithms. In *16th International Parallel and Distributed Processing Symposium (IPDPS)*, April 2002. Available from `www.cs.huji.ac.il/~feit/papers/Barrier02IPDPS.ps.gz`.

[66] Jean-Yuan Tsai and Pen-Chung Yew. The superthreaded architecture: Thread pipelining with run-time data dependence checking and control speculation. In *Fifth International Conference on Parallel Architecture and Compiler Techniques (PACT)*, pages 35–46, Boston, MA, 1996. Available from `citeseer.ist.psu.edu/tsai96superthreaded.html`.

[67] Nathan Tuck and Dean M. Tullsen. Initial observations of the simultaneous multithreading Pentium 4 processor. In *12th International Conference on Parallel Architecture and Compiler Techniques (PACT)*, pages 26–34, New Orleans, LA, September 2003. IEEE Computer Society. Available from `ieeexplore.ieee.org/iel5/8771/27774/01237999.pdf`.

[68] User Mode Linux. `user-mode-linux.sourceforge.net/`.

[69] Carl A. Waldspurger and William E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *First Symposium on Operating Systems Design and Implementation (OSDI)*, pages 1–11, Monterey, CA, November 1994. USENIX. Available from `citeseer.ist.psu.edu/waldspurger94lottery.html`.

[70] Clark Williams. Linux scheduler latency. `www.linuxdevices.com/files/article027/rh-rtpaper.pdf`, March 2002.

[71] Chulho Won, Ben Lee, and Chansu Yu. Linux/SimOS - a simulation environment for evaluating high-speed communication systems. In *31st International Conference on Parallel Processing (ICPP)*, pages 193–202, Vancouver, Canada, August 2002. Available from `citeseer.ist.psu.edu/won02linuxsimos.html`.

[72] Haoqiang Zheng and Jason Nieh. SWAP: A scheduler with automatic process dependency detection. In *First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, pages 183–196, San Francisco, CA, March 2004. Available from `www.ncl.cs.columbia.edu/publications/nsdi2004_fordist.pdf`.