

**Parallel Processing**

High Performance Business Apps Free  
Technical White Paper  
[www.roguewave.com](http://www.roguewave.com)

**Entry-Level 1U Server**

HPC Intel Xeon Server for Less. Request a  
Custom Quote Now.  
[www.appro.com](http://www.appro.com)

**Process Payments Online**

Businesses: Process payments online. Free  
setup. See our demo!  
[PaySimple.com/Processing](http://PaySimple.com/Processing)



Ads by Google



## Multicore and Parallelism: Catching Up

It's a new day when parallelism is essential for speeding up everything

By Jonathan Erickson, [Dr. Dobb's Journal](#)

Mar 30, 2009

URL: <http://www.ddj.com/hpc-high-performance-computing/216401662>



*David Bader is Executive Director of High-Performance Computing in the College of Computing at Georgia Institute of Technology and author of [Petascale Computing: Algorithms and Applications](#).*

**Q:** David, you've been actively involved with multicore and parallelism since the early 1990s and the rest of the computing world seems just now catching up. Any idea of why it's taken so long?

**A:** Overnight the computing world has changed, and we are starting a new day when parallelism is essential not just for solving grand challenge applications in science and engineering, but for speeding up everything: our 3G cellphone apps, our desktop spreadsheets and office productivity tools to our web browsers, media players, and Web services. Over the past few decades the parallel computing community, which has been a niche from academia, industry, and national labs, has performed a terrific job of harnessing massive parallelism to solve fundamental science questions in areas such as climate prediction, renewable energy, and molecular dynamics, as evidenced by the first petascale computing systems with phenomenal capabilities. The [Los Alamos Roadrunner](#) system harnesses thousands of IBM [Cell Broadband Engine](#) heterogeneous multicore processors, the same line of PowerPC processor in the Sony PlayStation 3, to solve some of our nation's most critical energy and material science problems. Oak Ridge's [Jaguar Cray XT5](#) supercomputer also broke through the petascale record and is the world's most powerful supercomputer today for science. The parallel processing community magnificent successes over the past several decades have gone mostly unnoticed by the masses, because we've been able to inject our new technologies into mainstream computing unbeknownst to the user. Processors have employed astonishing amounts of parallelism using superscalar microprocessors and pushed the limits of instruction-level parallelism, and systems have employed accelerators ubiquitously from floating-point math units to graphics processors (GPUs). The end user could continue to write her sequential algorithms, and the responsibility for exploiting the implicit concurrency was assumed by the architecture, compiler, and runtime systems. We could maintain this illusion and exploit concurrency at a fine-grain within the system in order to achieve the speedup scaling with Moore's Law that everyone has come to expect.

Let me give you analogy of the situation today. Over the past century, our transportation systems have improved from riding horses to driving cars. And our car engines have gotten faster and smarter over the last few decades. We still have pretty much the same driver's seat from the Model T to our fastest race cars -- complete with a steering wheel, accelerator and brakes. Imagine that our car engines reached their physical limits, so the industry has decided to provide the only solution available and sell us cars that have two engines, four engines, and even some with a thousand engines. And because of the added weight to our vehicles, these engines actually slow down our cars, even though they provide tremendous capability. We would have to rethink what it means to drive a car, and perhaps our entire transportation systems. The same is true in computing. Our ability to exploit the concurrency in application code has reached its limit, and we are now telling the application developers that it is their responsibility to rethink their applications and explicitly reveal the parallelism.

My experience with parallel computing goes back to the early 1980s, and I've found that the most challenging task often is designing efficient parallel algorithms. I am optimistic that many applications can be parallelized, yet there is an enormous gap in our capabilities for designing new parallel algorithms and for educating our programmers. We are taking a leadership position at Georgia Tech and completely revamping the curriculum. We've launched a new academic department and programs in [Computational Science and Engineering](#) that are focused at the interface between computing and the sciences and engineering and addressing the software needs for productively using hybrid, multicore and manycore processors. We are producing computational thinkers who are experts in multicore computing and parallelism and can readily transform computing.

**Q:** One of your early projects is SWARM. Can you tell us what it is and what problems it addresses? Is there anything like it out there?

**A:** Continued performance on multicore and manycore processors now requires the exploitation of concurrency at the algorithmic level. [SWARM](#) is a freely available as open-source software package from my group at Georgia Tech for "SoftWare and Algorithms for Running on Multicore and Manycore processors". SWARM brings together 15 years of research on shared-memory algorithms, and addresses the key issues in algorithm design for multicore processors. And we are grateful for the generous support from the National Science Foundation, IBM, Microsoft, Sun Microsystems, and Intel, that has enabled our research and development in parallel algorithms. SWARM, with continued development since 1994, is a portable open-source parallel library of basic primitives that fully exploit multicore and manycore processors. Using this framework, we have implemented efficient parallel algorithms for important, new primitive operations such as prefix-

sums, pointer-jumping, symmetry breaking, and list ranking; for combinatorial problems such as sorting and selection; for parallel graph theoretic algorithms such as spanning tree, minimum spanning tree, graph decomposition, and tree contraction; and for computational genomics applications such as maximum parsimony.

Several other multicore programming productivity efforts have caught my eye. For instance, [Cilk++](#) from Cilk Arts answers the multicore challenge and provides a smooth transition path from today's legacy applications to tomorrow's multicore implementations. We have been early adopters of Cilk++ and are impressed with its productivity gains. Other language efforts are also quite amazing. Sun Microsystems is developing a new high performance computing language with high programmability called [Fortress](#) with new features for implicit parallelism, transactions, and a flexible, space-aware, mathematical syntax. Intel is developing an innovative product called [Concurrent Collections](#) that provides mechanisms for constructing high-level parallel programs while allowing the programmer to ignore lower-level threading constructs. And IBM has an outstanding compilers team focused on improving [OpenMP](#) for heterogeneous multicore processors and developing new high productive languages such as X10.

**Q:** You once said that "In order to develop high-performance practical parallel algorithms, you must be a good sequential algorithmist." Could you expand on that?

**A:** Yes, this statement comes from two observations I've made during my professional career in designing high-performance parallel algorithms. First, one must have expertise in designing sequential algorithms before mastering parallel algorithms. It is important to start with an efficient sequential algorithm and identify its bottlenecks, before taking the challenging step of designing an efficient parallel algorithm. Second, in a good parallel implementation, the performance is often dominated by the work performed by each thread or core. For example, in multicore computing, one must still know how to achieve performance from the algorithm subtask running on each core for a speedup of the entire application. And it goes both ways. Through the study of parallel algorithms, we have occasionally discovered an asymptotic improvement to a sequential algorithm. And an algorithmic improvement can offer dramatically higher speedups than relying on technological factors alone.

**Q:** What is your current research interest?

**A:** At Georgia Tech my group is pursuing an ongoing effort to design and implement novel parallel graph algorithms to efficiently solve advanced small-world network analysis queries, enabling analysis of networks that were previously considered too large to be feasible. For tractable analysis of large-scale networks, we are developing [SNAP](#) ("Small-world Network Analysis and Partitioning"), an open-source graph analysis framework that builds on our SWARM library and supplements existing static graph algorithms with relevant ideas from dynamic graph algorithms, social network analysis, and parallel and multicore processing.

Large-scale network analysis broadly applies to real-world application domains such as social networks (friendship circles, organizational networks), the Internet (network topologies, the web-graph, peer-to-peer networks), transportation networks, electrical circuits, genealogical research and computational biology (protein-interaction networks, population dynamics, food webs). There are several analytical tools for visualizing social networks, determining empirical quantitative indices, and clustering. Real-world networks from diverse sources have been observed to share features such as a low average distance between the vertices (the small-world property), heavy-tailed degree distributions modeled by power laws, and high local densities. Modeling these networks based on experiments and measurements, and the study of interesting phenomena and observations, continue to be active areas of research.

SNAP is a modular infrastructure that provides an optimized collection of algorithmic building blocks (efficient implementations of key graph-theoretic analytic approaches) to end-users. SNAP provides a simple and intuitive interface for the network analyst, effectively hiding the parallel programming complexity involved in low-level algorithm design from the user while providing a productive high-performance environment for complex network queries.

In order to achieve scalable parallel performance, we exploit typical network characteristics of small-world networks, such as the low graph diameter, sparse connectivity, and skewed degree distribution. The SNAP parallel algorithms, coupled with aggressive algorithm engineering for small-world networks, give significant running time improvements over current state-of-the-art social network analysis packages. We are extending the SNAP framework with efficient data structures and algorithms aimed specifically at manipulation and analysis of large dynamic networks, with applications in social networks, healthcare, and critical infrastructures such as the Internet and the power grid.



Copyright © 2006 [CMP Media LLC](#)