

# Enabling Programmers to Design Efficient Parallel Algorithms for Many-Core Processors

David A. Bader  
Executive Director of High-Performance Computing  
College of Computing  
Georgia Institute of Technology  
Atlanta, GA 30332  
Email: [bader@cc.gatech.edu](mailto:bader@cc.gatech.edu)  
<http://www.cc.gatech.edu/~bader>

May 3, 2007

## Abstract

Since the inception of the desktop computer, performance of software has improved at an exponential rate, primarily driven by the steady technological improvements of microprocessors. Due to fundamental physical limitations and power constraints, we are now witnessing a radical change in commodity microprocessor architecture, to multi-core designs. Continued performance now requires the exploitation of concurrency at the algorithmic level. Automatic methods for detecting concurrency from sequential codes, for example with parallelizing compilers, have been successful in only a few regular cases. For several decades, a niche community mainly in scientific and technical computing has developed parallel computing methodologies; however, this community typically relies on specially-trained heroic programmers due to the complex programming methodologies that are often difficult to comprehend and use, even for the most sophisticated scientific programmer. We have designed and implemented **SWARM** (SoftWare and Algorithms for Running on Multicore), a portable parallel programming methodology for desktop users with commodity multi- and many-core processors by providing a productive, easy-to-use package.

## 1 Programming Manycore Algorithms

Since the inception of the desktop computer, performance of software has improved at an exponential rate, primarily driven by the steady technological improvements of microprocessors. Due to fundamental physical limitations and power constraints, we are now witnessing a radical change in commodity microprocessor architecture, to multi-core designs. Continued performance now requires the exploitation of concurrency at the algorithmic level. Automatic methods for detecting concurrency from sequential codes, for example with parallelizing compilers, have been successful in only a few regular cases.

Thus, we are facing a crisis for providing software engineers with productive and easy-to-use programming methodologies that can take full advantage of multi-core processors. This crisis is

primarily driven by three causes: 1) the microprocessor industry must shift to multi-core processor architectures, 2) computer science education has focused primarily on teaching sequential algorithms, and 3) automated methods, such as those in compilers, cannot deduce algorithmic concurrency from most sequential codes.

There is no more free lunch – continued performance improvements of client codes cannot rely solely on the ride we’ve taken with Moore’s Law for the past four decades. Instead, we must take stake out a new path for training programmers to use multicore (parallel) algorithms. We will need to educate programmers and developers on the design, analysis, and implementation, of parallel algorithms by providing a parallel programming methodology and example codes, and also provide libraries and tools that offer a rich foundation of efficient parallel primitives and kernels.

For the last several decades, the high-performance computing (HPC) community has designed parallel algorithms. Yet this community’s roots are primarily in solving the computational problems arising in physics, such as fluid flow, structural analyses, and molecular dynamics. And performance has relied heavily on specially-trained heroic programmers due to the complex programming methodologies such as explicit message passing that are often difficult to comprehend and use, even for the most sophisticated scientific programmer. Client workloads on desktop systems, on the other hand, consist of office and business applications, games and entertainment, digital content creation, web authoring, operating systems and utilities. While HPC is generally concerned with problems that use regular data structures such as matrices and tuned floating-point performance, client software often uses irregular data structures and a greater mix of integer and floating-point operations. SWARM is freely-available as open source and targets client applications that are often challenging to speed up when compared with the best sequential implementation.

## 2 SWARM Innovations

SWARM is a parallel programming methodology and library of multi-core parallel primitives and algorithms for a variety of fundamental tasks used by both regular and irregular applications. The library will build upon the successful prior work of the principal investigator for designing and implementing fast multi-threaded algorithms for symmetric multiprocessors.

A number of techniques are provided through library support, runtime system, and example codes, that demonstrate key methods for programming manycore processors. The *prefix-sum* algorithm is one of the most useful parallel primitives and is at the heart of several other primitives, such as array compaction, sorting, segmented prefix-sums, and broadcasting; it also provides a simple use of balanced binary trees. *Pointer-jumping* (or path-doubling) iteratively halves distances in a list or graph, thus reducing certain problems in logarithmic time; pointer jumping is at the heart of numerous sampling techniques in parallel algorithms. Determining the root for each tree node in a rooted-directed forest is a crucial step in handling equivalence classes—such as detecting whether or not two nodes belong to the same component; when the input is a linked list, this algorithm also solves the parallel prefix problem. An entire family of techniques of major importance in parallel algorithms is loosely termed *divide-and-conquer*—such techniques decompose the instance into smaller pieces, solve these pieces independently (typically through recursion), and then merge the resulting solutions into a solution to the original instance. Such techniques are used in sorting, in almost any tree-based problem, in a number of computational geometry problems (finding the closest pair, computing the convex hull, etc.), and are also at the heart of fast transform methods such as the FFT. A variation on this theme is a *partitioning strategy*, in which one seeks to decompose

the problem into independent subproblems—and thus avoid any significant work when recombining solutions; quicksort is a celebrated example, but numerous problems in computational geometry can be solved efficiently with this strategy (particularly problems involving the detection of a particular configuration in 3- or higher-dimensional space). Another general technique for designing parallel algorithms is called *pipelining*. In this approach, waves of concurrent (independent) work are employed to achieve optimality. Finally, *symmetry breaking* is often required to provide independent work between processors in self-similar problems; in a directed cycle, symmetry breaking can take the form of 3-coloring the cycle.

Building on a foundation of these techniques and kernels, we have assembled a higher-level library of manycore optimized parallel algorithms for list ranking, comparison-based sorting, radix-sort, tree operations such as maintaining indices and spatial data structures, and graph algorithms for breadth-first search, connected components, spanning tree, minimum spanning tree, and centrality. These problems are representative of important algorithms used in client workloads.

SWARM is portable on Microsoft Visual Studio with Intel multicore, Sun “Niagara” UltraSparc T1, and Linux desktop platforms, and provides an efficient library of basic primitives that fully exploit manycore processors. SWARM software and example codes are freely available as open source from SourceForge and are released under the Microsoft Permissive License (Ms-PL) that allows licensees to view, modify, and redistribute the source code for either commercial or non-commercial purposes and the Lesser GNU Public License (LGPL).

### 3 Brief Biography

David A. Bader is the Executive Director of High-Performance Computing and an Associate Professor in Computational Science and Engineering, a division within the College of Computing, Georgia Institute of Technology. He received his Ph.D. in 1996 from The University of Maryland, was awarded a National Science Foundation (NSF) Postdoctoral Research Associateship in Experimental Computer Science. He is an NSF CAREER Award recipient, an investigator on several NSF awards, a distinguished speaker in the IEEE Computer Society Distinguished Visitors Program, and was a founding member of the IBM PERCS team for the DARPA High Productivity Computing Systems program. Dr. Bader serves on the Steering Committees of the IPDPS and HiPC conferences, and was the General co-Chair for IPDPS (2004–2005), and Vice General Chair for HiPC (2002–2004). David has chaired several major conference program committees: Program Chair for HiPC 2005, Program Vice-Chair for IPDPS 2006 and Program Vice-Chair for ICPP 2006. He has served on numerous conference program committees related to parallel processing and computational science & engineering, is an associate editor for several high impact publications including the IEEE Transactions on Parallel and Distributed Systems (TPDS), Journal of Parallel and Distributed Computing (JPDC), the ACM Journal of Experimental Algorithmics (JEA), IEEE DSONline, and Parallel Computing, is a Senior Member of the IEEE Computer Society and a Member of the ACM. He has received several industry awards including the Sony-Toshiba-IBM Center of Competence for the Cell Broadband Engine processor, IBM Faculty Fellow, IBM Shared University Research (SUR), and Microsoft Research Faculty Award in parallelism and concurrency. Dr. Bader has been a pioneer in the field of parallel algorithms for problems in bioinformatics and computational genomics. He has co-chaired a series of meetings, the IEEE International Workshop on High-Performance Computational Biology (HiCOMB), written several book chapters, and co-edited special issues of the Journal of Parallel and Distributed Computing (JPDC) and IEEE TPDS

on high-performance computational biology. He has co-authored over 80 articles in peer-reviewed journals and conferences, and his main areas of research are in parallel algorithms, combinatorial optimization, and computational biology and genomics.

Representative publications are:

- D.A. Bader, V. Agarwal, and K. Madduri. On the Design and Analysis of Irregular Algorithms on the Cell Processor: A case study on list ranking. In *21th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Long Beach, CA, March 26-30, 2007.
- D. A. Bader and G. Cong. Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs. In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS)*, Santa Fe, NM, April 2004.
- D. A. Bader and G. Cong. A fast, parallel spanning tree algorithm for symmetric multiprocessors (SMPs). *Journal of Parallel and Distributed Computing*, 65(9):994–1006, 2005.
- D.A. Bader, G. Cong, and J. Feo. On the architectural requirements for efficient execution of graph algorithms. In *Proc. 34th Int'l Conf. on Parallel Processing (ICPP)*, Oslo, Norway, June 2005.
- D.A. Bader, A.K. Illendula, B. M.E. Moret, and N. Weisse-Bernstein. Using PRAM algorithms on a uniform-memory-access shared-memory architecture. In *Proc. 5th Int'l Workshop on Algorithm Engineering*, Århus, Denmark, 2001.
- D.A. Bader, V.N. Kanade, and K. Madduri. SWARM: A Parallel Programming Framework for Multi-Core Processors. In *First Workshop on Multithreaded Architectures and Applications (MTAAP)*, Long Beach, CA, March 30, 2007.
- D.A. Bader and K. Madduri. A Graph-Theoretic Analysis of the Human Protein-Interaction Network Using Multi-core Parallel Algorithms. In *Sixth IEEE International Workshop on High Performance Computational Biology (HiCOMB)*, Long Beach, CA, March 26, 2007.
- D.A. Bader and K. Madduri. Designing Multithreaded Algorithms for Breadth-First Search and st-connectivity. In *Proc. 35th Int'l Conf. on Parallel Processing (ICPP)*, Columbus, OH, August 2006.
- D.A. Bader and K. Madduri. Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks. In *Proc. 35th Int'l Conf. on Parallel Processing (ICPP)*, Columbus, OH, August 2006.
- D.A. Bader, S. Sreshta, and N. Weisse-Bernstein. Evaluating arithmetic expressions using tree contraction: A fast and scalable parallel implementation for symmetric multiprocessors (SMPs). In *Proc. 9th Int'l Conf. on High Performance Computing*, Bangalore, India, December 2002.
- G. Cong and D. A. Bader. An experimental study of parallel biconnected components algorithms on symmetric multiprocessors (SMPs). In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS)*, Denver, CO, April 2005.
- J.R. Crobak, J.W. Berry, K. Madduri, and D.A. Bader. Advanced Shortest Path Algorithms on a Massively-Multithreaded Architecture. In *First Workshop on Multithreaded Architectures and Applications (MTAAP)*, Long Beach, CA, March 30, 2007.