HIGH-PERFORMANCE ALGORITHMS AND APPLICATIONS FOR SMP CLUSTERS

David A. Bader[1]
The University of New Mexico
Electrical and Computer Engineering Department
Albuquerque, NM 87131
Phone: (505) 277-6724
E-Mail: dbader@eece.unm.edu

1 Introduction

The future of high-performance computing relies on the efficient and scalable use of clusters with symmetric multiprocessor (SMP) nodes and low-latency, high-bandwidth interconnection networks. Current examples of such platforms include Sun Ultra HPC machines, Compaq AlphaServers with Quadrics switches, SGI Origins, and the IBM SP system with SMP nodes. Moreover, the future of NASA mission-critical computing for computational aerosciences relies on the success of computational clusters (e.g., SMP Linux clusters at Goddard Space Flight Center, and large SGI Origin arrays at Ames Research Center). Hardware benchmark results reveal awesome performance rates for each component; however, few applications on SMP clusters ever reach a fraction of these peak speeds. While methodologies for symmetric multiprocessors (e.g., OpenMP [21] or POSIX threads [22]) and message-passing primitives for clusters (e.g., MPI [20]) are well developed, performance dictates the use of a hybrid solution. We present preliminary results of our complexity model and programming methodology that is based hierarchically upon realistic model components for message-passing and for symmetric multiprocessor parallel architectures. The current deployment of teraflops and the future development of petaflops systems will certainly require the exploitation of similar hybrid programming models.

Our goal is to validate and refine a core complexity model, efficient primitives and algorithmic libraries needed to support the effective use of SMP clusters for computational aeroscience (CAS) and earth and space science (ESS) codes. In addition, we will show how to design, implement, analyze, and refine algorithms that take advantage of the hybrid programming model and contribute to significant speedups for real computational science problems. These will include some basic combinatorial support tasks used in numerical aerodynamics simulations (e.g., sorting integers for particle-in-cell codes) and selected numerical computations (e.g., fast transforms).

These algorithmic kernels exhibit promising performance improvements. Our hybrid programming environment also must support mission-critical high-performance computing applications. Thus, we use real computational aeroscience and Earth and space science codes drive our algorithmic development. These applications include segmentation and classification of remotely-sensed imagery, and navigation, calibration, and registration of terascale data sets, for remote sensing (AVHRR and MODIS) and three-dimensional numerical relativity with the CACTUS computational astrodynamics toolkit.

[1]Supported in part by Department of Energy Sandia National Laboratories contract number AX-3006 and National Science Foundation CISE Postdoctoral Research Associate in Experimental Computer Science No. 96-25668.

2 SIMPLE: A Programming Methodology for SMP Clusters

The RAM model has been successfully used to design efficient sequential algorithms and to predict their performance. Parallel algorithmic models are inherently more difficult because of the added dimensions, for example, multiple processors and interconnection networks. The Parallel RAM (PRAM) model [15] is often used to design parallel algorithms, but because of idealistic assumptions, fails to provide an accurate performance measure for real platforms. On current parallel machines we assume processing nodes are completely interconnected and communication is subject to the restrictions imposed by the latency and bandwidth properties of the network. Thus, network topology models, such as mesh, tree, and hypercube, which were popular in the past, no longer are efficient on general machines. In addition, these models seldom provide portable algorithms. Recently, more realistic parallel models such as LogP [10] and BSP [24] predict algorithmic performance using a small number of parameters representing the machine configuration and communication properties. However, writing efficient communication operations using these models is tedious and seldom yields algorithms that are efficient on other machine configurations. Our previous work [4, 5, 6, 8, 13, 14] has successfully augmented such models with a rich set of collective communication primitives that can be implemented efficiently on a variety of machines. The user may easily describe an algorithm in terms of these primitives, simplifying the design process and alleviating the need to re-write the algorithm when using other platforms.

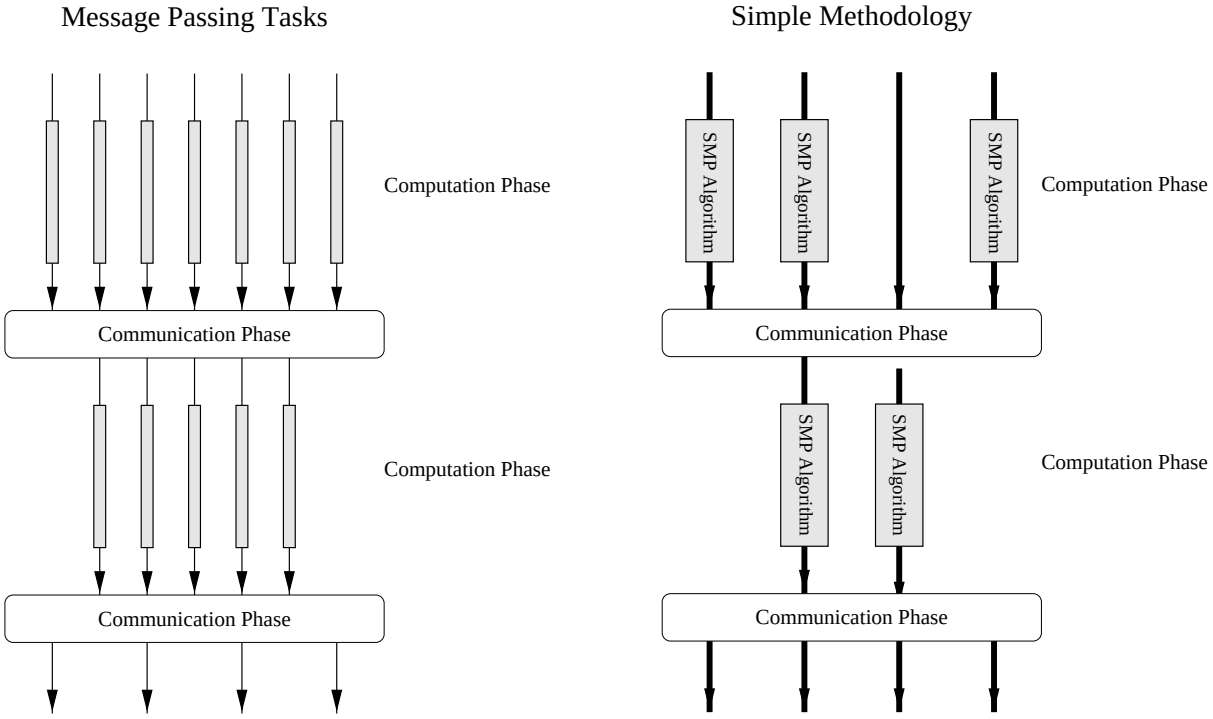
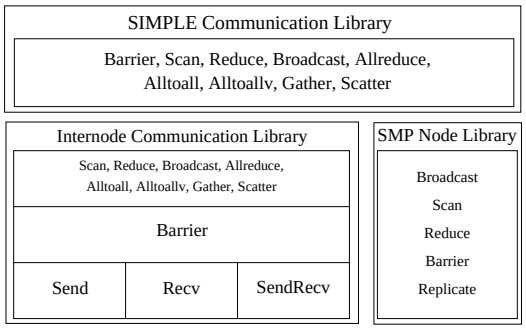
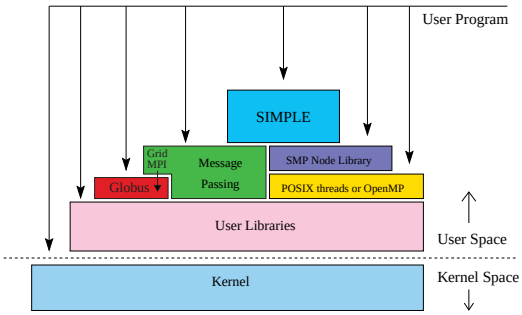


Figure 1: On the left, we show a message-passing algorithm where each task uses sequential code during computation phases. On the right, the SIMPLE approach may replace each computation step with an optimal SMP algorithm, rearrange the tasks across the SMP Cluster, and substitute hybrid communication routines for the original message-passing calls. Note that this general methodology is suitable for both regular and irregular algorithms.



Hierarchy of SMP, message-passing, and SIMPLE communication libraries



High-performance applications may access SIMPLE , SMP, message-passing, shared-memory, Globus grid, and standard user libraries. SIMPLE operates completely in user space.

Figure 2: Library Design

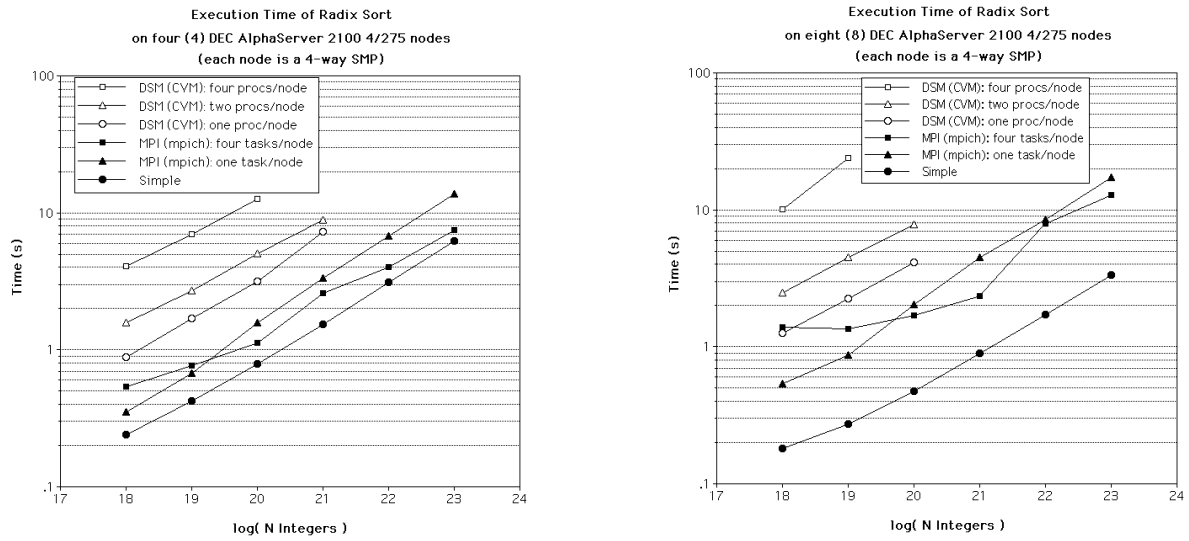
None of these models, however, address the hybrid methodology of using both shared-memory nodes and message-passing communication between these nodes. We are developing an integrated complexity cost scheme for SMP clusters that accurately models the hierarchical memory system and networks, while retaining the benefits of the RAM model. Namely, our complexity model will be simple to apply, provide a "rule of thumb" for algorithmic complexity, and aid algorithmic designers in discovering efficient and portable algorithms on clusters.

We call this approach SIMPLE , referring to the joining of the SMP and MPI-like message-passing paradigms and the *simple* programming approach (see Figure 1). SIMPLE is described in more detail in [7].

Existing programming methodologies for SMP clusters fall into two categories [12]. The first, distributed shared-memory (DSM) systems (for example, TreadMarks [2] from Rice University, Multigrain Shared-Memory (MGS) [25] from MIT and Coherent Virtual Machine (CVM) [18] from University of Maryland), provides a software layer which simulates coherent shared-memory between nodes by internally using messaging to move around specific data or referenced memory pages. The second, based on message-passing primitives (for example, MPI), enforces a shared-nothing paradigm between tasks, and all communication and coordination between tasks are performed through the exchange of explicit messages, even between tasks on a node with physically shared-memory. For example, the models used in [19] and [23] assume that each processor in the cluster will be assigned a message-passing (MPI-level) process, with lower latency communication between processes on the same SMP node than with inter-node messages. Our work differs from both of these approaches, in that we advocate a hybrid methodology that maps directly to underlying architectural aspects. We combine shared-memory programming on shared-memory nodes with message-passing communication between these nodes.

A preliminary proof-of-concept for SIMPLE has been implemented [7] that confirms the applicability of this computational model for SMP clusters and provides encouraging empirical results which support our thesis.

The SIMPLE communication library for SMP clusters uses internode communication (MPI) and SMP-node libraries to implement hybrid primitives. These hierarchical layers of our communication libraries are pictured in Figure 2. These libraries may be implemented at a high level, completely in user space as a layer on top of standard parallel and distributed computing libraries. Because no kernel modification is required, these libraries easily port to new platforms. The application may use the SIMPLE abstractions or its underlying layers; for example, MPI, POSIX threads, OpenMP, and Globus [11].



Four Nodes Eight Nodes

Figure 3: Comparison of DSM, MPI, and SIMPLE Radix Sort on a cluster of DEC AlphaServer 2100 4/275 nodes. Note that we tested the DSM/CVM radix sort implementation using one to four processes per node, and the MPI/MPICH implementation using both one and four MPI tasks per node. The SIMPLE implementation uses four threads per SMP node, and four and eight nodes on the left and right, respectively.

Next we describe the superior performance of SIMPLE 's hybrid approach versus DSM or message-passing implementations using an integer sorting example, an irregular problem required by a variety of important applications such as particle-in-cell codes. Figure 3 provides a summary of the performance of the SIMPLE methodology with DSM/CVM or MPI/MPICH on an SMP cluster of eight Digital AlphaServer 2100 4/275 nodes. In this experiment, we compare the performance of a SIMPLE radix sort code using both four and eight 4-way SMP nodes with that of both DSM/CVM and MPI/MPICH code for various cases, such as using one or multiple threads of execution per node. In all situations on the cluster of SMPs testbed, the SIMPLE algorithm substantially outperforms that of both the distributed shared memory and the message passing implementations.

3 Kronos: A Terascale Processing System for AVHRR Imagery

At regional scales, satellite-based sensors are the primary source of information to study the earth's environment, as they provide the needed dynamic temporal view of the earth's surface. Raw satellite orbit data have to be processed and mapped into a standard projection to produce multitemporal data sets which can then be used for regional or global earth science studies such as land cover dynamics, global carbon cycle, planetary-scale climate dynamics, regional agricultural crop monitoring, desertification studies and drought monitoring, terrestrial environmental monitoring, and global water and energy balance studies. For a given sensor, different applications may require different processing chains with the same few core steps. Application dependent processing steps include atmospheric correction, spatial and temporal subsetting, and output image projection. However, the data sets that are currently available to the scientific community are generated using a predetermined processing chain in a fixed projection. Generating products that are different than the standard ones can be difficult and will result in at least a re-sampling step and hence some loss of accuracy.

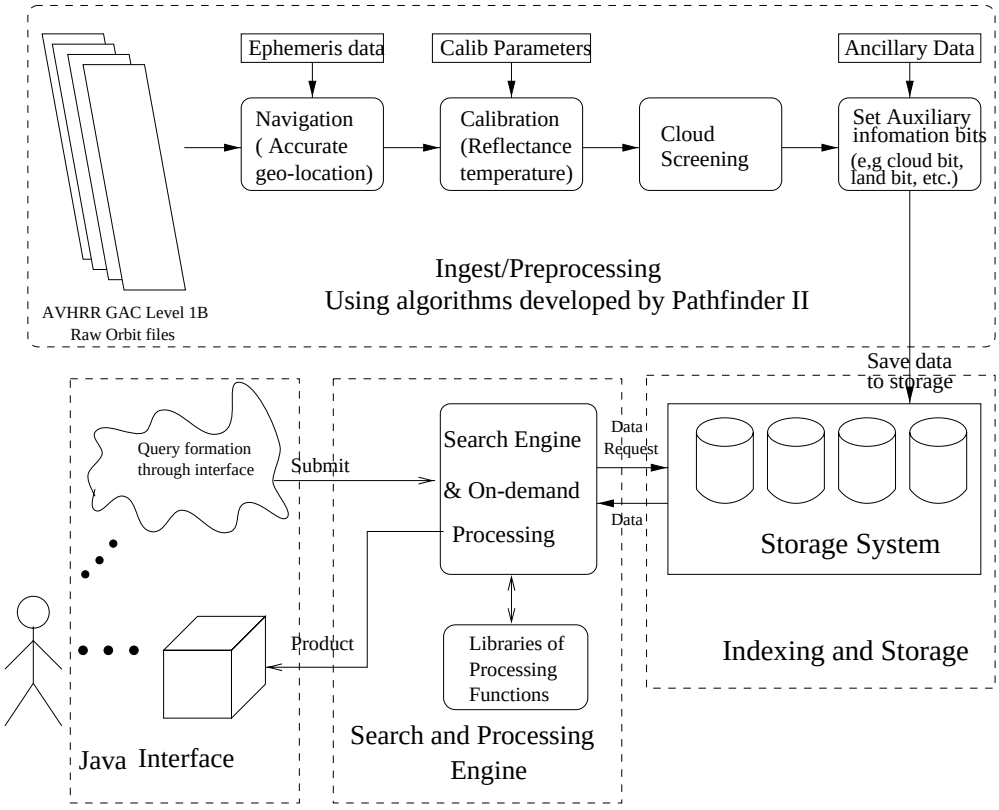


Figure 4: Kronos processing system architecture

The Earth science community has recognized the need for consistently processed and calibrated global, long-term data sets from instruments such as the Advanced Very High Resolution Radiometer (AVHRR). For instance, scientists studying land use derive the normalized difference vegetation index (NDVI), a measure of the amount of "greenness" of land cover, from the AVHRR record. The AVHRR instruments flown aboard polar-orbiting NOAA platforms have collected multi-band radiometric global-cover observations of the Earth spanning more than 18 years (1981 - present), with almost 600MB of raw information stored per day, or approximately 4 *Terabytes* for the entire record.

We next describe a software system *Kronos* for the generation of custom-tailored data products from the Advanced Very High Resolution Radiometer (AVHRR) sensor. Kronos [26, 17, 16] allows the generation of a rich set of products that can be easily specified through a Java interface by scientists wishing to carry out earth system modeling or analysis based on AVHRR Global Area Coverage (GAC) data. Kronos is based on a flexible methodology and consists of four major components: ingest and preprocessing, indexing and storage, search and processing engine, and a Java interface (see Figure 4).

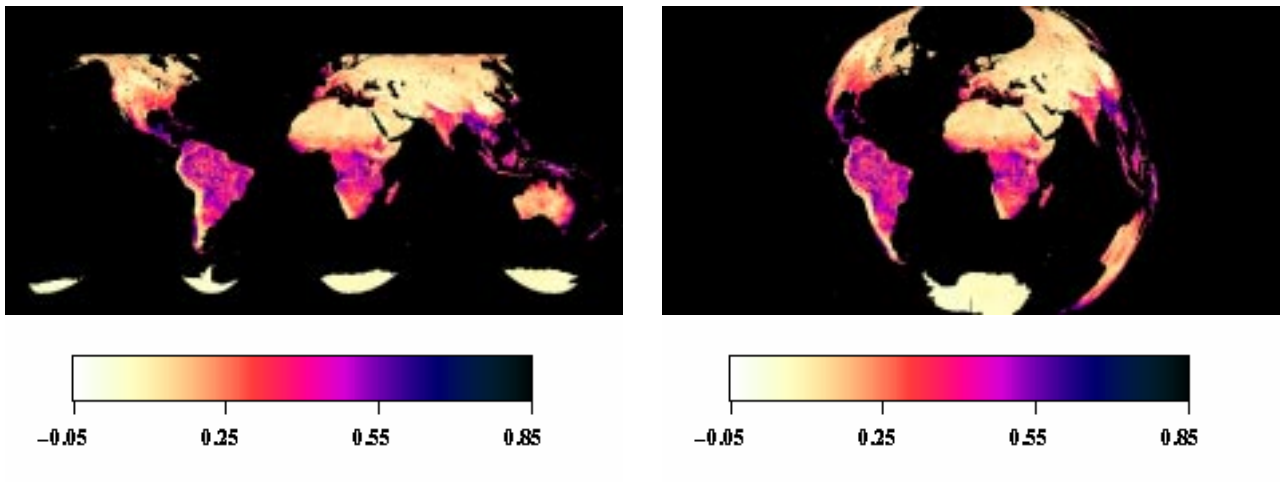


Figure 5: A global NDVI image from the Kronos system for processing and retrieval of AVHRR terascale data sets. 10-day composite from days 1 to 10 of 1989 in Goode Homolosine (left) and Lambert Azimuthal Equal Area (right) projections.

During the ingest stage, the system extracts observations from collections of raw satellite data (for example, AVHRR GAC Level 1B orbit files or Landsat Thematic Mapper scenes), and for each observation 1) geo-locates it to determine the most accurate spatial geometry, 2) calibrates the remotely sensed, raw sensor digital counts with community accepted methods, and 3) determines secondary (or derived) observation parameters. Next, the storage module accepts each observed object and efficiently archives it in a terascale high-performance storage system using spatial indexing techniques. Finally, Earth science users interact with the system via the third stage to acquire packaged products in standard formats that resolve each user's query. We illustrate the power of our methodology by including a few special data products generated by Kronos. Example imagery for global NDVI products in two commonly requested projections are shown in Figure 5.

4 CACTUS: A Grand Challenge Application on a SuperCluster

The University of New Mexico / National Computational Science Alliance Roadrunner Linux SMP SuperCluster is an Alta Technology Corporation 64-node AltaCluster containing 128 Intel 450 MHz Pentium II processors. The SuperCluster runs the 2.2.12 Linux operating system in SMP mode with communication between nodes provided via a high-speed Myrinet network (full-duplex 1.2 Gbps) or with Fast Ethernet (100 Mbps). Each node contains components similar to those in a commodity PC, for instance, a 100 MHz system bus, 512KB cache, 512 MB ECC SDRAM, and a 6.4 GB hard drive. Using several grand challenge applications, we have demonstrated supercomputing-class performance on a Linux SuperCluster [9].

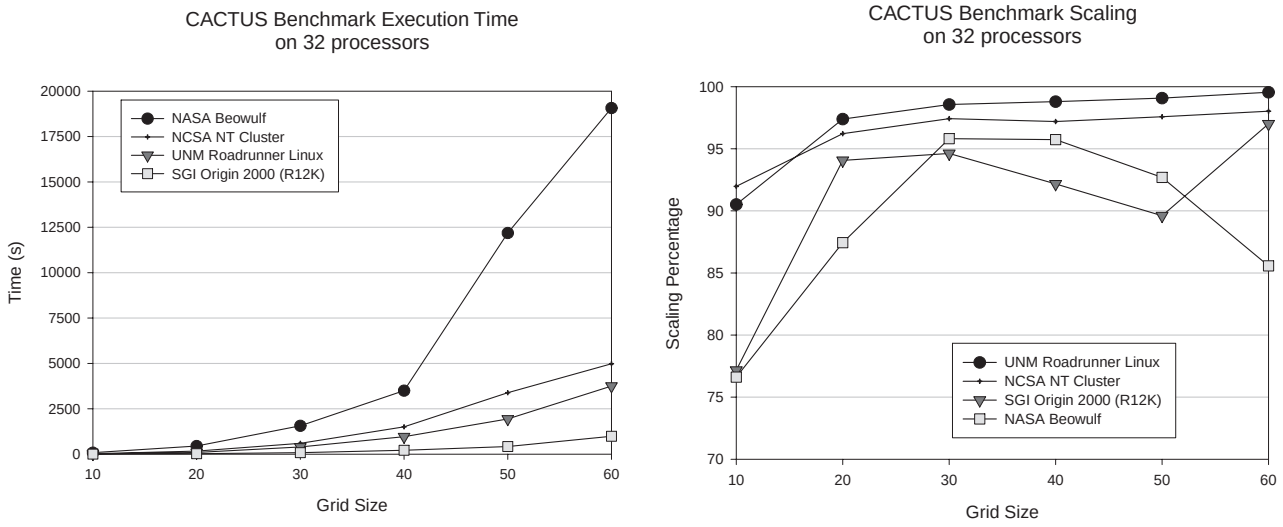


Figure 6: CACTUS performance

CACTUS [1] is a high-performance computing toolkit that implements a framework for solving three-dimensional computational astrophysics problems on parallel machines. In Figure 6 we show the scalability of the CACTUS grand challenge application for increasing problem sizes using an instance that solves the Bona-Mass hyperbolic formulation of the Einstein equations for numerical relativity calculations on the Roadrunner SuperCluster. As a key, 'NASA Beowulf' is a 64-node dual PPro 200 MHz, 64MB RAM, 'NCSA NT Cluster' runs Microsoft Windows NT with 32 dual PII 333 MHz with 512 MB RAM, and 'SGI Origin 2000' is the NCSA Origin 2000 with 32 R12K 300 MHz processors. Notice that Roadrunner SMP SuperCluster contains the best scalability for this problem, more than 99 % as the problem size increases.

We would like to thank Ed Seidel, Gabrielle Allen, and Oliver Wehrens (Max-Planck-Institut für Gravitationsphysik and University of Illinois) for the CACTUS code.

5 Combinatorial Algorithms

Graph theoretic techniques are used in variety of important computational problems in the areas of computational physics, mechanics, and fluid flow. Efficient sequential algorithms for many of these graph-based techniques are well-known, for instance, searching, decomposing, and finding network flows. However, when either very large graphs require the memory space of parallel computers or faster solutions aim more processing power at the problem, efficient implementations of some of these same techniques for high-performance computers at this point are not known. One graph problem of significant interest is detecting cycles in directed, planar graphs. This problem arises in the numerical stability of large-scale calculations that use finite-element meshes. Up to now, no efficient parallel implementations were known. In [3], we present a new, parallel algorithm for detecting cycles in partitioned, directed planar graphs that is both scalable in the graph and machine size, and performs well in practice.

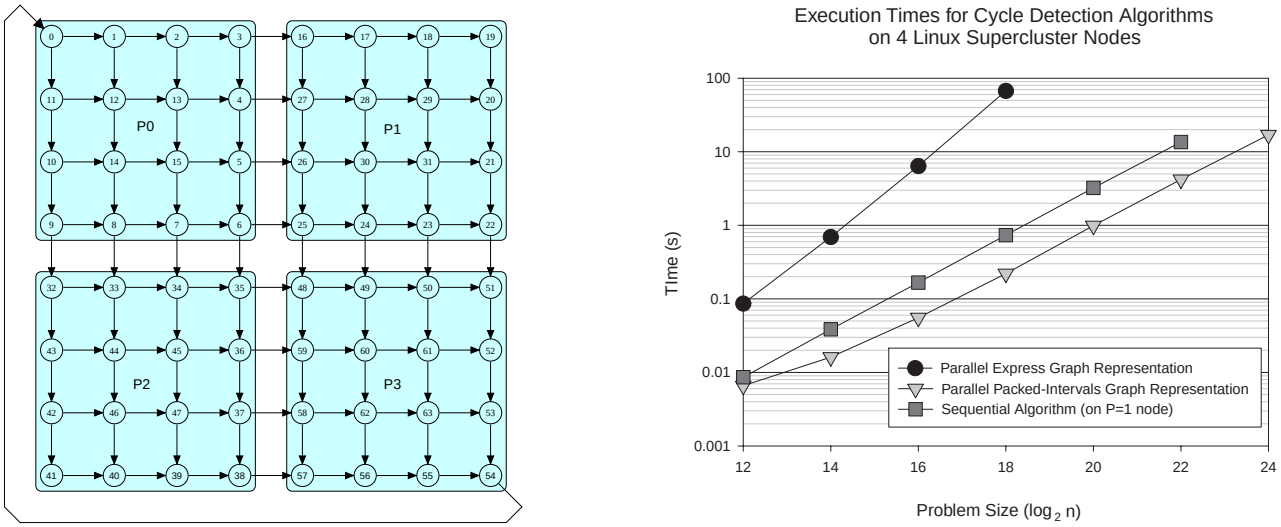


Figure 7: A practical, parallel algorithm for cycle detection of partitioned, planar digraphs.

Our novel approach contains three phases. During the first phase, each node's subgraph is locally examined for cycles. Arcs spanning node boundaries are discovered in the second phase. Finally, in the third phase, pairwise merging between subgraphs reduces the problem to a single processing node, all-the-while examining the graph for cycles and aggressively pruning vertices and incident arcs that do not contribute to cycles. The merging phase may be prohibitively expensive using a standard "Express" graph representation that reduces each subgraph to boundary vertices, subgraph-arcs directly connecting the boundaries, and arcs spanning node boundaries. We have discovered a novel approach that uses what we call the compact "Packed-Intervals" graph representation. Instead of recording each subgraph-arc, our algorithm records only the endpoint labels for each interval of reachable boundary vertices. Our parallel implementation using the Packed-Intervals graph representation now outperforms the sequential approach with nearly linear speedup in the number of processors and problem size (see Figure 7) and has a complexity cost dominated by the $O(\log p)$ algorithmic supersteps. Our parallel implementation proves the capability of solving extremely large inputs that, to our knowledge, have never been attempted before. For instance, on a 64-processor cluster, our implementation solves a difficult input graph (see Figure 7) with 16M ($M \equiv 2^{20}$) vertices in 12.2 seconds, with 64M vertices in 48.0 seconds, and with 256M vertices in 288 seconds. Thus, our new algorithm solves problem instances once thought impossible and scales almost linearly with problem size on a large cluster.

References

[1] G. Allen, T. Goodale, and E. Seidel. The Cactus Computational Collaboratory: Enabling Technologies for Relativistic Astrophysics, and a Toolkit for Solving PDE's by Communities in Science and Engineering. In *Proceedings of the Seventh IEEE Symposium on the Frontiers of Massively Parallel Computation*, pages 36–41, Annapolis, MD, February 1999.

[2] C. Amza, A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. TreadMarks: Shared Memory Computing on Networks of Workstations. *IEEE Computer*, 29(2):18–28, 1996.

[3] D.A. Bader. A Practical Parallel Algorithm for Cycle Detection in Partitioned Digraphs. Submitted, 1999.

[4] D.A. Bader, D.R. Helman, and J. JáJá. Practical Parallel Algorithms for Personalized Communication and Integer Sorting. *ACM Journal of Experimental Algorithmics*, 1(3):1–42, 1996. `http://www.jea.acm.org/1996/BaderPersonalized/`.

[5] D.A. Bader and J. JáJá. Parallel Algorithms for Image Histogramming and Connected Components with an Experimental Study. *Journal of Parallel and Distributed Computing*, 35(2):173–190, June 1996.

[6] D.A. Bader and J. JáJá. Practical Parallel Algorithms for Dynamic Data Redistribution, Median Finding, and Selection. In *Proceedings of the 10th International Parallel Processing Symposium*, pages 292–301, Honolulu, HI, April 1996.

[7] D.A. Bader and J. JáJá. SIMPLE: A Methodology for Programming High Performance Algorithms on Clusters of Symmetric Multiprocessors (SMPs). *Journal of Parallel and Distributed Computing*, 58(1):92–108, 1999.

[8] D.A. Bader, J. JáJá, D. Harwood, and L.S. Davis. Parallel Algorithms for Image Enhancement and Segmentation by Region Growing with an Experimental Study. *The Journal of Supercomputing*, 10(2):141–168, 1996.

[9] D.A. Bader, A.B. Maccabe, J.R. Mastaler, J.K. McIver III, and P.A. Kovatch. Design and Analysis of the Alliance / University of New Mexico Roadrunner Linux SMP SuperCluster. In *Proceedings of the IEEE International Workshop on Cluster Computing*, Melbourne, Austrailia, December 1999.

[10] D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993.

[11] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[12] W.W. Gropp and E.L. Lusk. A Taxonomy of Programming Models for Symmetric Multiprocessors and SMP Clusters. In *Proceedings of 1995 Programming Models for Massively Parallel Computers*, pages 2–7, Berlin, Germany, October 1995.

[13] D.R. Helman, D.A. Bader, and J. JáJá. A Randomized Parallel Sorting Algorithm With an Experimental Study. *Journal of Parallel and Distributed Computing*, 52(1):1–23, 1998.

[14] D.R. Helman, J. JáJá, and D.A. Bader. A New Deterministic Parallel Sorting Algorithm With an Experimental Evaluation. *ACM Journal of Experimental Algorithmics*, 3(4), 1997. `http://www.jea.acm.org/1998/HelmanSorting/`.

[15] J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley Publishing Company, New York, 1992.

[16] S.N.V. Kalluri, J. JáJá, D.A. Bader, Z. Zhang, J.R.G. Townshend, and H. Fallah-Adl. High Performance Computing Applications in Remote Sensing Studies for Land Cover Dynamics. *International Journal of Remote Sensing*, 1998. In press.

[17] S.N.V. Kalluri, Z. Zhang, J. JáJá, D.A. Bader, H. Song, N. El Saleous, E. Vermote, and J. Townshend. A Hierarchical Data Archiving and Processing System to Generate Custom Tailored Products from AVHRR Data. In *Proceedings of IEEE 1999 International Geoscience and Remote Sensing Symposium (IGARSS'99)*, Hamburg, Germany, June/July 1999.

[18] P. Keleher. *CVM: The Coherent Virtual Machine*. University of Maryland, 0.1 edition, November 1996.

[19] S.S. Lumetta, A.M. Mainwaring, and D.E. Culler. Multi-Protocol Active Messages on a Cluster of SMP's. In *Proceedings of Supercomputing '97*, San Jose, CA, November 1997.

[20] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. Technical report, University of Tennessee, Knoxville, TN, June 1995. Version 1.1.

[21] OpenMP Architecture Review Board. OpenMP: A Proposed Industry Standard API for Shared Memory Programming. `http://www.openmp.org/`, October 1997.

[22] Portable Applications Standards Committee of the IEEE. *Information technology – Portable Operating System Interface (POSIX) – Part 1: System Application Program Interface (API)*, 1996-07-12 edition, 1996. ISO/IEC 9945-1, ANSI/IEEE Std. 1003.1.

[23] W. Saphir, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.1 Results. Report NAS-96-010, Numerical Aerodynamic Simulation Facility, NASA Ames Research Center, Moffett Field, CA, August 1996.

[24] L.G. Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, 33(8):103–111, 1990.

[25] D. Yeung, J. Kubiatowicz, and A. Agarwal. MGS: A Multigrain Shared Memory System. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, Philadelphia, PA, May 1996.

[26] Z. Zhang, J. JáJá, D.A. Bader, S. Kalluri, H. Song, N. El Saleous, E. Vermote, and J. Townshend. Kronos: A Java-Based Software System for the Processing and Retrieval of Large Scale AVHRR Data Sets. *Photogrammetric Engineering & Remote Sensing*, 1999. In press.