

An Interactive Approach to Parallel Combinatorial Algorithms with Star-P

John Gilbert and Viral Shah
University of California,
Santa Barbara
gilbert@cs.ucsb.edu and
viral@cs.ucsb.edu

Todd Letsche and Steven Reinhardt
SGI
letsche@sgi.com and
spr@sgi.com

Alan Edelman
Massachusetts Institute of Technology
and Interactive Supercomputing
edelman@math.mit.edu and
edelman@interactivesupercomputing.com

High Productivity Computing Design Philosophy

This paper explores the use of the Star-P interactive high performance computing system to study and illustrate our core beliefs about high productivity computing:

1. High productivity computing can be possible if a sufficiently rich set of computational primitives and tools are available and used on high performance systems.
2. This set must interoperate with other user defined calls.
3. They must be available interactively so that programmers can explore their use to best advantage.

Of course one further requirement is necessary to achieve high performance with high productivity:

4. The primitives and tools must be implemented as high performance kernels so many users get the benefit of the available speedups.

This is quite different from traditional high performance computing which has so very often been a “do it yourself” methodology. In this methodology users hack for performance but the benefits are rarely available for alternative uses.

Combinatorial and Graph Algorithms

In the context of combinatorial and graph algorithms, we wish to offer the programmer such a rich set of primitives and tools. The Star-P system, now a commercial product of Interactive Supercomputing, extends MATLAB style interactive programming to a high performance machine through a backend server [1,2,3,4,5,6]. Our conclusion is that this system can offer highly tuned kernels for combinatorial and graph algorithms with the above core beliefs in mind. This is far more preferable and practical than asking users to learn to obtain speedups from scratch, reinventing the wheel each time.

The SSCA Kernel

To drill down the principles above in this context we have implemented the High Productivity Computing System’s SSCA#2 graph kernels. We have found that

- Graphs are well expressed and stored as a sparse matrix data structure.
- This data structure is more than a nice artifice for expressing when and with what strength “i” is connected to “j”.
- We also obtain the expressiveness of data parallel operations such as matrix times vector, and collective operations on matrices that have such nice combinatorial uses.
- We further obtain the expressiveness of linear algebra operations such as sparse eigenvalue routines that are important components for visualizing and partitioning combinatorial structures.

We chose SSCA#2 in part because of the underlying philosophy of the SSCA codes as a whole and our interest in graph algorithms in particular. For readers that may not be familiar, The High Productivity Computing Systems Program is working on identifying how the promises of high performance computing can allow workflows to be more productive. (See <http://www.highproductivity.org/>).

As part of this program, benchmarks are being created intended to abstract the true nature of many high performance programs. The Scalable Synthetic Compact Applications [7] are a particularly nice set in that they attempt to capture pieces of real applications while remaining of a manageable size.

As part of the application, a graph is generated before the timing begins. As a sample of the genre of the graph, as well as our interactive tool philosophy, we used the eigenvalues of the graph to produce the picture in Figure 1:

8192-vertex graph from Kernel 1 plotted with Fiedler coordinates

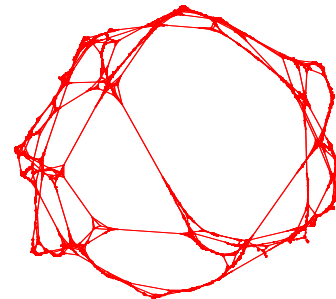


Figure 1:

SSCA#2 consists of a data generator (which generates labelled directed edges) and four kernels. Kernel 1 builds the multigraph data structure; Kernel 2 searches the graph by edge label; Kernel 3 locates all edges and vertices within a specified distance of a set of start vertices; and Kernel 4 partitions the entire graph into tightly connected clusters

The concise version, cSSCA2 is almost but not quite a full implementation of SSCA#2. The written specification includes both integer and string-valued edge labels; cSSCA2 implements only integer-valued labels. We should also note that the clustering algorithm used in Kernel 4 of cSSCA2 differs significantly from that in the executable spec. The executable spec uses a sequential seed-growing method based on Koester [8]; cSSCA2 uses a spectral method based on minimizing cluster isoperimetric numbers.

The Star-P Implementation

We illustrate the expressive power of Star-P by comparing the number of lines of executable code (per kernel) between the Star-P implementation, the executable spec, and Bader and Madduri's C/Pthreads/SIMPLE implementation [9]. The direct comparison with the executable spec should be taken with a grain of salt: while it is also written in serial Matlab, its style is intended to facilitate message-passing parallel implementation, and of it includes more bookkeeping code (e.g. for string-valued edge labels) than cSSCA2. Nonetheless, we believe that the following statistics are striking.

Table: Lines of executable code (excluding I/O and graphics)

	cSSCA2	executable spec	C/Pthreads/SIMPLE
Kernel 1:	29	68	256
Kernel 2:	12	44	121
Kernel 3:	25	91	297
Kernel 4:	44	295	241

The productivity improvement

The improvement in productivity (as measured by lines of code) in cSSCA2 is primarily due to the use of data-parallel primitives for expressing sparse graph manipulation in a natural way. For example, Kernel 1 (which converts an unordered list of "edge triples" to a graph represented as a cell array of distributed sparse adjacency matrices) is essentially the "sparse" constructor in Star-P. Kernel 2 (which searches for maximum and specific edge labels) is essentially Star-P's "max" and "find" operations on distributed arrays. Kernel 3 (which follows graph paths from specific vertices) is implemented by Star-P's distributed sparse matrix-vector multiplication, which is extremely concise and reasonably efficient. An implementation (in progress) using distributed sparse vectors as well as dsparse matrices will increase its efficiency. Kernel 4 (which finds clusters) uses Star-P's built-in eigenvalue/eigenvector primitives, which calls the

PARPACK library, and performs an isoperimetric computation in an efficient data-parallel style.

We timed Kernel 1 on a four processor linux cluster and found that we were approaching the asymptotic regime of speedup. The actual timings were 6, 8, 15, and 29 seconds for problems of size 2^{15} , 2^{16} , 2^{17} , and 2^{18} respectively. We will be running much larger problems in the near future on bigger machines, but these timings already confirm the benefits of the approach. More important than the timings are the benefits to productivity.

In conclusion, by expressing parallel algorithms for combinatorial problems using sparse matrix primitives, we illustrate the productivity gains as measured by lines of code. The raising of the level of abstraction allows library writers or commercial vendors to accelerate the primitives for the benefit of not just one implementer but for a large class of users. This is the scalability that we should strive for.

References.

- [1] R. Choy and A. Edelman, "Parallel MATLAB: Doing it right," *Proceedings of the IEEE*, Vol.93, No.2, Feb 2005, pages 331-341.
- [2] R. Choy, *MATLAB*p 2.0, Interactive Supercomputing Made Practical*, M. Science Thesis, Massachusetts Institute of Technology, Cambridge, 2002.
- [3] P. Husbands and C. Isbell, "The Parallel Problems Server: A Client-Server Model for Large Scale Scientific Computation." *Proceedings of the Third International Conference on Vector and Parallel Processing*. Portugal, 1998.
- [4] P. Husbands, *Interactive Supercomputing*, PhD Thesis, Massachusetts Institute of Technology, Cambridge, 1999.
- [5] A Edelman, P. Husbands, and C. Isbell, *What is MIT MATLAB?* <http://citeseer.ist.psu.edu/333211.html>, 1998.
- [6] R. Choy and A. Edelman, "Solving Multiple Classes of Problems in Parallel with MATLAB*P," *Proceedings of the 2004 SINGAPORE MIT Alliance*, available on dspace: <https://dspace.mit.edu/handle/1721.1/3874>
- [7] <http://www.highproductivity.org/SSCABmks.htm>
- [8] D. Koester, "Parallel Block-Diagonal-Bordered Sparse Linear Solvers for Power Systems Applications," PhD Thesis, Syracuse University, Syracuse, New York. <http://www.npac.syr.edu/techreports/hypertext/scs-745/>
- [9] D. Bader and K. Madduri, "SSCA#2 graph theory: C/pthreads implementation using shared memory" <http://www.highproductivity.org/SSCABmks.htm>