Chapter 1
Driving Issues in Scalable Libraries: Poly-Algorithms, Data Distribution Independence, Redistribution, Local Storage Schemes*

Anthony Skjellum† Purushotham V. Bangalore‡

Abstract

In this paper we describe our perspective of the issues and strategies involved in state-of-the-art scalable parallel library research and development. We divide the discussion into four key areas: data distribution independence, issues in redistribution, local storage schemes, and the role of poly-algorithms.

1 Introduction

Parallel algorithms, encapsulated in scalable libraries, are inherently constructed bottom-up, that is, from the machine up to the user interface, with the hope that the resulting syntax and semantics will be of use in a non-trivial set of applications. Conversely, parallel algorithms, the driving force for parallel applications, work from performance requirements and application goals, down to libraries, machine instructions, and so on. The hope of the application designer is that the performance requirement posed at the outset matches his/her expectations when finally implemented. Normally, to account for shortfalls, the application programmer hedges by demanding portability (in other words, a second chance to move to a better or newer piece of hardware with more native performance for the application). Conversely, to provide for usability, the library designer must provide flexibility and portability.

We describe our perspective of the issues and strategies involved in state-of-the-art scalable parallel library research and development by dividing the discussion into four key areas: data distribution independence, issues in redistribution, local storage schemes, and the role of poly-algorithms. In this paper, we do not consider the related, important issues of message-passing system semantics and syntax as they relate to library isolation and correctness, but defer that to [4, 7].

---

*We acknowledge financial support by the NSF Engineering Research Center for Computational Field Simulation (NSF ERC), Mississippi State University.

†Assistant Professor, NSF Engineering Research Center for Computational Field Simulation & Department of Computer Science, Mississippi State University

‡Graduate Research Assistant, NSF Engineering Research Center for Computational Field Simulation & Department of Computer Science, Mississippi State University

2 Data Distribution Independent Design

At the conceptual interface between top-down and bottom-up design efforts[1] of applications and libraries, lies the *potential* for scalable parallel libraries to provide extremely useful, higher-level portability and performance-achieving characteristics similar to those achieved in sequential platforms (where the Fortran-77 memory model was reasonably sufficient). The canonical parallel application, a sequence of solution steps, requires data distribution and re-distribution as a natural part of calculation phases. This puts an especially greater burden on both libraries and application programming: the optimal data locality of each phase is typically different, and explicit redistribution is expensive (and this expense cannot be ignored). Thus, we assert that an important class of libraries will have to accept flexible specifications about the kind of data they expect to input and output as part of a chain of calculations. Therefore, part of our paper describes flexible data-layout management, which we call data-distribution independence, following Van de Velde [5, 6, 8, 9].

The technology of efficient mappings are described (this is a continued discussion of the concept of scalable permutations). The existing theory of different "strengths" of distributions extends the interest in such algorithms to cases where both complete and incomplete mappings make sense for an application [3, 5].

3 Redistribution of Data

The commonly held strategy in library design is that data distribution is a nominal issue, and that the block-scatter distribution is a "pretty good choice" for attaining load-balance and blocking (for local data re-use). Our purpose (long term) is to illustrate why this strategy is, in general, sub-optimal from the performance perspective for real applications that do more than one phase of a computation. Data is created, used by a library, then used by other libraries or an application. In many cases, overall cost is reduced by accepting mal-distribution. Furthermore, many data-parallel-like algorithms incorporate sufficient data motion to permit incremental redistribution during the course of a calculation. Yet, there is the potential for cost savings by avoiding explicit redistribution at the entry- and exit- interfaces of a library, because of these optimizations.

In [6] the performance results of a non-blocking, data-distribution-independent, level-2 BLAS LU factorization algorithm has been presented for different data distributions and logical grid shapes. Peak-performance for this algorithm was observed for a particular logical grid shape, 18×28, with a scatter-scatter distribution on an order 10000×10000 dense matrix on the Intel Delta. For distributions other than scatter-scatter the performance degrades, but not by more than a factor of two in the worst case. Hence the data-distribution-independent algorithm can generate high performance for the distribution that is "optimal" for it and can still function when an application uses a different distribution.

The argument is that data-distribution-independent algorithms are sometimes, depending on the application requirements, more efficient than fixed-data-distribution counterparts, because on-entry redistribution of data may be costly. It may not be economical explicitly to redistribute data for all problems and problem sizes. Redistribution costs can be ignored only when the cost of solving the problem in the "optimal data layout" is an order-of-magnitude greater than the cost of explicit data redistribution. Also explicit data redistribution limits the maximum size of the problem that can be solved on a particular

[1] Object-oriented design has a comparable, iterative nature to repetitive top-down and bottom-up studies of software structuring.

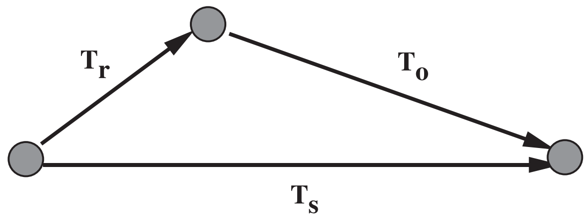

FIG. 1. *The argument is that* $T_s < T_r + T_o$ *for certain class of algorithms*

machine, as there should a enough memory to store a second copy of the local matrix used in the data redistribution[2].

Figure 1 illustrates our argument pictorially. If the time taken to solve the problem optimally assuming a particular data distribution is T_o, the time spent in redistributing the data is T_r, and the time taken to solve the same problem using a sub-optimal data-distribution-independent approach is T_s, then the notion is that $T_s < T_r + T_o$.

4 Local Storage Schemes

Local storage mechanisms traditionally used in the Fortran-77 model are insufficient in the parallel setting. The use of sequential-BLAS as high performance engines in parallel routines is hampered by the BLAS's assumptions of local data layout as dense, column-major matrices and vectors. For instance, complications arise when distributed sub-matrices are treated as first class objects. Generalized functions, capable of dealing with strided memory access, are needed, and must be optimized by vendors. Implicit here is a layering of parallel codes on sequential codes, that may not lead to optimal overlap of communication and computation.

5 Poly-Algorithms

Since a range of parallelism and architectures is desired by the application programmer, in order to get reasonable performance benefits, and because common architectures vary in their characteristics within reasonable bounds, one can contemplate linking together a number of similar algorithms into a unified user interface, and providing the most likely to be best algorithm for a given problem size and parallelism. Poly-algorithms, a concept evidently introduced by John Rice, refer to the use of two or more algorithms to solve the same problem with a high level decision-making process determining which of a set of algorithms performs best in a given situation (such as sorting of long lists vs. sorting of short lists). Though this is a difficult optimization problem, in general, an object-oriented language like C++ (and even C, with some effort) allows runtime experimentation with a number of options as the user or application developer learns to tune an application. Such poly-algorithms links data distribution independent and fixed distribution algorithms via common interfaces, and also describes common interfaces for related algorithms like semi-iterative linear system methods. They are the key to performance and portability of scalable libraries.

[2]Two copies of data assumes a redistribution algorithm where all the sends are posted before a receive.

6 Summary

In this paper we have briefly discussed the following four key areas involved in the design and development of scalable parallel libraries that can be summarized as follows:

- Data distribution independence does not imply redistribution of data at the interfaces
- Data redistribution must be performed only when absolutely essential
- Obtain high performance for general case only by generalizing BLAS
- Poly-algorithms
  - hide complexity of performance-oriented software
  - hide multiplicity of algorithms in the interface
  - support performance and portability

Yet, people are reluctant to accept these facts. Hence more experiments like [5] are needed, where data reuse is also exploited. We are currently working on a data distribution independent level-3 BLAS LU factorization algorithm to show the importance of the above mentioned issues [1]. We intend that the results obtained from these experiments should illustrate the need for more careful consideration of the design of scalable parallel libraries as novel, activities rather than message-passing re-fits to existing sequential software.

References

[1] P. V. Bangalore, *The Data-Distribution-Independent Approach to Scalable Parallel Libraries*, Master's thesis, Mississippi State University — Dept. of Computer Science, October 1994. Available as `ftp://cs.msstate.edu/pub/reports/bangalore_ms.ps.Z`.

[2] J. Choi, J. J. Dongarra, and D. W. Walker, *Parallel Matrix Transpose Algorithms on Distributed Memory Concurrent Computers*, in Proceedings of the Scalable Parallel Libraries Conference, A. Skjellum and D. S. Reese, eds., IEEE Computer Society Press, October 1993, pp. 245–252.

[3] R. D. Falgout, A. Skjellum, S. G. Smith, and C. H. Still, *The Multicomputer Toolbox Approach to Concurrent BLAS and LACS*, in Proceedings of the Scalable High Perfromance Computing Conference SHPCC-92, IEEE Computer Society, April 1992.

[4] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, 1994.

[5] A. Skjellum, *Concurrent Dynamic Simulation: Multicomputer Algorithms Research Applied to Ordinary Differential-Algebraic Process Systems in Chemical Engineering*, PhD thesis, Chemical Engineering, California Institute of Technology, May 1990.

[6] A. Skjellum and C. H. Baldwin, The Multicomputer Toolbox: *Scalable Parallel Libraries for Large-Scale Concurrent Applications*, Tech. Rep. UCRL-JC-109251, Lawrence Livermore National Laboratory, December 1991.

[7] A. Skjellum et al., *Scalable Libraries Using MPI: Abstraction, Performance, Portability*. In preparation, 1994.

[8] A. Skjellum, A. P. Leung, S. G. Smith, R. D. Falgout, C. H. Still, and C. H. Baldwin, *The Multicomputer Toolbox – First-Generation Scalable Libraries*, in Proceedings of HICSS–27, IEEE Computer Society Press, 1994, pp. 644–654. HICSS–27 Minitrack on Tools and Languages for Transportable Parallel Applications.

[9] E. F. Van de Velde and J. Lorenz, *Adaptive Data Distribution for Concurrent Continuation*, Tech. Rep. CRPC-89-4, California Institute of Technology, 1989. Caltech/Rice Center for Research in Parallel Computation.