On the Need for a Consortium of Capability Centers

William Gropp & Marc Snir

University of Illinois at Urbana-Champaign

7/6/2009

1. Introduction

1.1. Background

Users and operators of the largest, most powerful computers face many challenges. Often, each of these systems is treated as an entirely unique resource, requiring unique solutions, particularly in software. While there is some truth to this, there are also many common features. Most programs are (currently) written with C, C++, and/or Fortran, combined with MPI and sometimes OpenMP. Many applications require some sort of workflow support. Operators require scalable tools for monitoring and diagnosing the system. In this paper, we propose a consortium of the high-end or capability centers that will work together to create greater commonality in applications software and share experiences that can improve the user experience and lower the cost of operation.

Our proposal is motivated by the following observations:

A. Most, if not all of the users of the top NSF Supercomputing centers are using resources at more than one center; in particular, smaller systems and regional centers will be used for developing codes for the larger capability systems – in particular, Blue Waters. Users will be greatly advantaged if the various platforms they use have compatible application development environments and execution environments: same interfaces, languages, libraries and tools, and similar procedures.

B. No vendor provides a complete solution to the need of the HPC community; any large platform will deploy a variety of libraries and tools that were developed by national labs or academic researchers. The development, porting, tuning, and maintenance of such software packages require collaborations with a variety of partners.

C. The desire for compatibility across platforms often lead application teams to seek the "common denominator" – to use only basic languages, libraries and tools that are guaranteed to be available and well supported on all platforms. New approaches with the potential to increase the productivity of the application programmers are not adopted because of this well known vicious circle: application programmers are reluctant to use software that is not well-supported on most platforms; and platform providers are reluctant to support software that is not used by a large number of applications.

The problem will worsen as we get into the petascale domain and beyond. It is widely recognized that new software solutions -- new programming models, new tools and new system structures will be needed in this domain. Research will be needed to develop the required technologies. But past experience has shown that the main obstacle to the introduction of new technology is often not at the research stage, but at the transition from research to advanced development and to deployment. This stage requires somewhat expensive bets on technologies with a high failure probability. The high visibility of top performing platforms facilitates expensive bets by national labs on new HW prototypes. The same cannot be said of new SW prototypes. Furthermore, supercomputing centers in academia are more financially constrained in their ability to invest in new technologies, even though their needs may often diverge from the needs of national labs. A collaboration of these centers and a pooling of resources can alleviate this problem.

1.2. Goals

We propose to create mechanisms that

- Facilitate the sharing of expertise and information about user needs, system operation and HPC software among the top supercomputing centers.
- Facilitate the sharing of expertise and information about the use of large HPC systems among the users of top supercomputing platforms.
- Facilitate collaborations between these centers.
- Encourage the deployment of common software on all major HPC platforms used by scientists; in particular, encourage the deployment of new languages, libraries and tools.

This initiative is synergistic with other extant initiatives:

- XD [46]: Our proposed initiative is (a) focused at the very high-end of the performance pyramid; and (b) is not aimed, like XD, at developing a specific s/w infrastructure, but at sharing information and collaborating in the deployment of any s/w that has can be common to many capability platforms.
- Exascale Software Project [5]: Our proposal is aimed at creating strong interactions between the current, petascale centers. Such interactions are essential in providing a transition from new research products to actual deployment and utilization on available systems. Our initiative will provide a receptive environment for the technologies emerging from excascale s/w research.

- PRACE [7]: The PaRtnership for Advanced Computing in Europe provides a common meeting point for the top HPC centers in the EU. Our initiative can have a similar role in the US and can establish a strong collaboration with PRACE.

The rest of this document describes activities that such a consortium may undertake, ways in which it may be organized, issues such as funding, and closes with a brief inventory of the software that is commonly used by applications and for which providing common interfaces and capabilities would both simplify the environment for the applications developer and user as well as reduce the cost of operation. The role of this document is not to present a detailed proposal but to begin discussion of what a consortium of capability centers could achieve.

One other point needs to be made. We are not advocating the development of a single standard software stack. While such a single stack provides the simplest environment for users, it can force the same sort of "greatest common denominator" restriction that we mention above. Rather, we believe that many users (and operators) could benefit from more commonality in software (for example, a few, well defined profiles) that could be universally available. Users would be free to use center-specific software to access special capabilities. However, by providing a (small) collection of common software capabilities, applications will have more time to explore the use of any unique capabilities. Thus we feel that this consortium can encourage innovation by removing the overhead of dealing with unnecessary variation in tools and software.

2. Potential Activities

We see four levels of interaction

1. Information sharing: the exchange of information across centers
2. Information aggregation: the aggregation of shared information in ways that makes it more accessible and more useful.
3. Collaboration: Joint activities to create new software and new information artifacts.
4. Standardization: Joint activities to agree on common profiles and common interfaces.

2.1. Information Sharing

Information sharing can occur through

- Periodic phone conferences
- Periodic workshops. Possibly focused on topics of common interest, such as parallel file systems
- Shared social networking tools (wiki, discussion groups, mailing lists, etc.)

Different mechanisms may be used for different groups – with an emphasis on regular interactions for the centers and on social networking tools for the users.

2.2. *Information Aggregation*

Shared information can be made more useful by collecting it in a common format and aggregating it. Possible examples include

- An inventory of used open source software
- An integrated directory of people: a list of contacts at the various centers for various subjects.
- An integrated directory of documentation and educational materials
- Aggregated statistics on system utilization, types of applications run, etc.: centers will agree to a core of consistent metrics
- Aggregated customer surveys: centers will agree to a core of common questions in their surveys, so as to enable aggregation of the results
- Aggregated bug reports: for vendor/platform related bugs, this will probably need to be done on a per platform basis and possibly kept confidential; for open source software, the information should be public. Centers will need to agree to consistent ontologies.

2.3. *Collaborations*

Collaborations can reduce duplicate work, and increase efficiencies in the various centers. Such collaborations may include

- Shared development of tools (e.g., application performance tools or system monitoring tools). The development is likely to occur in one place, but early interaction with other potential users will increase the odds that tools are portable and satisfy the needs of a broader community
- Shared testing: the development of good regression test suites is expensive; the sharing of general test suites, as well as tests focused on specific issues (such as OS jitter) can greatly benefit the centers
- Collaboration in the deployment of new software
- Collaborations in the evaluation of various tools and environments
- Collaborations in the development of education material and user guides

2.4. *Standardization*

Different centers have platforms from different vendors, with different software environments and different users; extensive homogenization of these environments is neither possible nor desirable. On the other hand, the differences between platforms are often spurious. Discussions between the centers could lead to agreements on a minimum common s/w stack for petascale/exascale platforms, either through support for the same tools and libraries, or through the provision of compatible profiles.

3. Issues

In order to establish the proposed consortium we need to resolve some of the issues listed below.

3.1. Funding

The consortium will need core funding for meetings and for support activities. It will be important that activities at the centers be funded from a budget managed by the consortium, to ensure that commitments are met.

3.2. Organization

The consortium needs a management model that ensures that decisions can be reached in a timely manner, while providing buy-in from the involved centers. This would probably involve a small executive committee coupled with a board representing all participating groups.

3.3. Balance between Commonality and Diversity

The consortium will need to balance the desire for a greater commonality in the software deployed at various center with the need to preserve the autonomy of these centers and the diversity of their platforms. In particular:

1. Do we specify a particular version or range of versions (at least as a default)? What do we do if some version has a security hole and needs a quick fix (what is our contract with our users about stability of the choices)?
2. Do we specify a base version and allow extensions? If so, how do we make the base strong enough so that many/most users can and will choose to stay within that base level? Should there be more than one? E.g., there could be a standards-compliant level (POSIX) and an enhanced level (GNU+POSIX). Since the software stack will continue to evolve quite rapidly, easy composability – the ability to add components developed by other groups -- might be more important than a standard core. To achieve this, it may be more important to specify standard interfaces for extensions – rather than detailed core functionality.
3. How do we track changes and evolution of software/standards? Should we have a shared repository (containing version information, header files, if appropriate, and source files, if appropriate).
4. How do we test compliance (more gently, how can sites quickly assess whether their environment conforms to the spec)?
5. How do we make sure that users adopt? What is the process for user buy-in? How do we assess success in getting users to work within the base set(s)? Good social networking tools that facilitate the sharing of experience by users may be an important component of the solution to this problem.

4. Inventory

We list below some of the software components that are relevant to the proposed consortium, issues related to those components, and joint activities that could improve the quality of HPC software.

4.1. Parallel Programming Models

The predominant programming model used in HPC is message passing with *MPI* [26]. As nodes contain more cores, a hybrid programming model of MPI + *OpenMP* [21] becomes more important. Partitioned Global Address Space (PGAS) languages (*UPC* [25]and *CAF* [47]) provide promising ways of leveraging fast interconnects in a scalable manner [20].

All these programming models are still evolving. MPI3 [6] may include new capabilities, such as better one-sided communication and non-blocking collectives that are important for scalability and jitter avoidance. Various proposals are being discussed for UPC and CAF extensions with collectives, teams, and richer partitions. These extensions are important for the continued usability of MPI and to provide sufficient expressiveness to PGAS languages. Therefore, it is important to ensure that capability platforms support the same version of these languages and libraries, and that this version incorporates as soon as possible those advanced features.

The use of a hybrid programming model (OpenMP+MPI) is hampered by the lack of an interoperability model for MPI and OpenMP: The current, implicitly supported model is to execute MPI calls in serial parts of the code; this model does not provide good performance for large shared memory nodes [16]. Similarly, the use of PGAS languages is hampered by the lack of interoperability between those languages and MPI. Large MPI codes will not be ported to PGAS languages at once – it is important to support the gradual conversion of code and, hence, to support mixed code.

Vendors have developed comprehensive test suites for languages such as Fortran and C. The same is far from true for emerging languages such as UPC. Worse, existing test suites are not always correct. Existing test suites focus on correctness; performance is handled using benchmarks – usually simplified application codes deemed to be representative of the application domain (e.g., *spec OMP*). The obvious danger of benchmarks is that systems can be narrowly optimized around them. Furthermore, performance on HPC systems is often brittle, with small code variations leading to large, unforeseen performance variations. Performance stability Is essential to tuning. There is a need for test codes to test the stability of performance: Is the same performance achieved by essentially equivalent versions of an algorithm? Is performance improved when blocking MPI communication is replaced by non-blocking communication? A strong common set of correctness and performance tests will improve the quality of HPC software.

4.2. Compilers and Linkers

These are primarily provided by vendors (hardware or software) or GNU. A key issue is which languages (and which versions) are available – E.g., Fortran 2003 [34, 36], C99 [35, 49], etc. This could be a significant problem, as some vendors are slow to conform to current standards. Many languages have extensions that are broadly supported and used in many applications. GCC implements many such extensions. Can we standardize on these extensions or on GCC?

Users are often stumped by prosaic Issues such as differences in common command line arguments, particularly for include and library paths. Common profiles could remove in large part such obstacles.

Different linking conventions for shared and dynamic libraries can also be a problem – AIX has a very different approach to shared and dynamic libraries than most other Unix implementations (Mac OS/X also has a different -- though less so).

4.3. Build tools (make, configure, etc.)

Make in various forms is provided by the vendor and by GNU. The GNU version has extensive, broadly used extensions. A common interface that includes none of these may be too limiting.

"Meta-make" tools such as the GNU configure and build system [53, 28] (consisting of *autoconf* [42], *automake* [22] and *libtool* [43]), *or CMake* [1] are essential for building more complex environments. The GNU configure and build system doesn't handle cross compilation environments well and most autoconf scripts are not correct with respect to cross-compilation – this may be a significant issue for some HPC platforms.

4.4. Debuggers

These are primarily provided by system vendors or GNU. There are few products with truly scalable user interfaces. (*Totalview* [54], which is produced by one of the very few HPC ISVs, is the exception.) There is no much uniformity among the various solutions.

A debugger such as Totalview has to use a different infrastructure to control executing threads on different systems. A standard communication and control API would facilitate the task of the developers of scalable debuggers. On the other hand, it is too early to standardize more than very basic user interfaces.

4.5. Performance tools

- Low-level sensors such as *PAPI* [15] that provide access to performance counters, *gprof* [30] that supports statistical sampling of a running thread, or *mpiP* [55, 37] and *FPMPI* [31] that provide information on MPI calls.

- Parallel performance tools, such as *Tau, VAMPIR* [45] (now *Intel Trace Analyzer [4]*), *Jumpshot* [57]*, Scalasca* [29]*,* etc. These integrate the information provided by a variety of low-level sensors to provide a global view of execution performance and detect performance anomalies.

In addition, tools are available to analyze performance bottlenecks in multi-threaded shared memory code (e.g., OpenMP code) and to present performance data in terms that are meaningful for higher-level programming models, such as UPC or CAF.

The huge volume of performance data that can be generated on a large system creates a need for intelligent compression, both for reducing the data volume and for presenting to the user an intelligible picture of performance. Therefore, we can expect tools to evolve rapidly from visualization of raw performance data to sophisticated "performance data analytics". This evolution can be facilitated by more standardization of the low level APIs for the control of performance sensors. It is desirable that hardware-oriented sensors be available as kernel services, rather than source-level patches, and that a common core of statistics be available on all systems (with the same semantics).

4.6. Visualization and Data Analytics

There is less consensus about tools for visualization and data analytics. For example, many applications make use of ad hoc file formats for input and output files, combined with their own analysis tools and even visualization packages. Two file formats that have significant communities are NetCDF and its parallel version pnetCDF [38] and HDF5 [2]. Data analysis tools have been created for both of these formats. In the area of visualization, VisIt [11] is one of the few parallel visualization tools that has scaled to over ten thousand cores. In general, scaling remains a challenge for many data analysis tools, with users often resorting to single processor or single node tools. A consortium can help users find and use existing tools as well as provide a forum for identifying the features required in any common toolset.

4.7. Parallel File Systems

Two important efforts in this area are *Lustre* [51, 52] and *GPFS* [50, 3]. Both systems provide full POSIX semantics, which can impact both performance and stability of the file system [40]. On the other hand, *PVFS* [32] relaxes POSIX semantics in order to improve performance.

It is not clear that strict POSIX semantics is required: processes within one parallel computation do not normally communicate through the file system, so that full data coherence is overkill; and a model where each process in a large computation frequently creates new files is not scalable. There are efforts to define more scalable POSIX APIs for file system metadata (e.g., to more efficiently handle directories with tens of thousands of files) [8]. A possible alternative is to provide good support to large collections of storage objects.

Progress in this area seems to be hampered by the lack of a clear understanding of the issues by users, the lack of a consensus on alternative storage models and the strong need for portability of data formats across systems. A consortium could break this logjam by leading to a broadly accepted consensus, on better structures for parallel I/O.

4.8. Integrated Development Environments (IDEs)

The productivity of commercial software developers has been enhanced by the use of IDEs, such as *Eclipse* [24] or *Microsoft Visual Studio* [44]. Eclipse is open source and has an increasing number of extensions that are relevant to HPC, such as the Eclipse *Parallel Tools Platforms* (PTP) [9, 56].

It would be very desirable to increase the usage of IDEs in HPC. This requires an IDE that is common across platforms and that integrates the core services needed for the development and execution of HPC code:

- Support for main languages and libraries (C, C++, Fortran 2003, OpenMP, UPC, CAF, MPI, SHMEM, Python, etc.)
- Support for hybrid code (e.g., OpenMP + MPI, Python + Fortran + C++, etc.)
- Support for parallel debugging and performance tuning
- Job control (mpiexec, batch job submission, job status)
- Support for remote development and execution.

Common profiles that remove spurious differences and ease localization will be very important.

4.9. Execution Environment

It would be helpful standardizing the line command arguments and the environment variables that control the execution of an OpenMP or MPI job – to the extent the differences are not due to differences in hardware. (See the DOE SciDAC project on system software that created a component-based framework for job management into which 3rd party components could be included.) Standard mechanisms for the creation of the execution software environments (e.g., *Module [27], Softenv*) [10] should be provided.

4.10. Scientific Libraries

This includes Standard, stable libraries (e.g., *BLAS* [41, 23]*, LAPACK* [13, 12]*, ScaLAPACK* [17, 14]); libraries from research groups that have a significant user community but are still evolving (e.g., *PETSc, SPRNG, WSMP, FFTW*). It would be useful to have a set of core libraries that are available on all platforms, possibly augmented with set of libraries that are specific to various application domains. Libraries should be located at a standard location..

4.11. Batch schedulers and resource managers

Different batch schedulers are in use, some proprietary (e.g., *LSF* [48], *LoadLeveler* [39, 33], *Moab Workload Manager* [18]) and some open source (e.g., *Torque* [19]). A split seems to occur between policy-based *metascheduler*s, such as *Moab*, that can manage workloads and workflows across multiple platforms and multiple OS'es, and *resource managers* that are platform specific and handle resource reservations on one platform and the initialization and control of parallel jobs on the platform. This evolution will be facilitated by standardizing the resource management APIs provided by different platforms.

4.12. Monitoring and error handling

It is widely accepted that resilience will be a major challenge for ultrascale platforms. Yet, there is very little information on the failure types and failure rates of various platforms. Progress in this area would be greatly facilitated by the publication of information on system failures. This requires a standard ontology for logged events and a standard (XML?) syntax to encode them. Current capability platforms seem to have limited abilities to store wide volumes of system performance and system health data. The

4.13. Software Porting and Maintenance

Capability platforms are increasingly dependent on open source software. In some cases, companies are created to continue maintaining and evolving a successful software product – but this is far from universal, as the business case for software targeting capability platforms is not always strong. Supercomputing centers are creating their own ad-hoc support infrastructure for such software, involving the original software developer as needed.

Collaboration between the centers could enhance these ad-hoc support systems, help harden research codes, increase portability, improve testing coverage and error reporting and reduce redundant work. One can envisage a common ticket system or a common repository of known problems that would avoid repeated work in diagnosing software bugs and will facilitate sharing bug fixes; various centers could help with porting and scaling on different platforms.

5. Summary

Capability supercomputing centers can provide better support to their users and can accelerate the deployment of the new software solutions needed for extreme scale computing, by joining forces and collaborating. We have outlined in this short paper the need for such collaboration, its possible mechanisms, key issues, and the inventory of software products such collaborations can impact. The time has come for action.

References

[1] *CMake*, http://www.cmake.org.

[2] *HDF5*, http://www.hdfgroup.org/HDF5/.

[3] *IBM General Parallel File System* http://www-03.ibm.com/systems/clusters/software/gpfs/index.html.

[4] *Intel® Trace Analyzer and Collector 7.2*, http://software.intel.com/en-us/intel-trace-analyzer/.

[5] *International Exascale Software Project*, http://www.exascale.org/iesp/Main_Page.

[6] *MPI Forum -- MPI 3.0 Standardization Effort*, http://meetings.mpi-forum.org/MPI_3.0_main_page.php.

[7] *Partnership for Advanced Computing in Europe*, http://www.prace-project.eu/.

[8] *POSIX Extensions*, http://www.pdl.cmu.edu/posix/.

[9] *PTP - Parallel Tools Platform*.

[10] *SoftEnv man pages*, http://www.teragrid.org/userinfo/softenv/.

[11] *VisIt*, https://wci.llnl.gov/codes/visit/.

[12] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling and A. McKenney, *LAPACK Users' guide*, Society for Industrial Mathematics, 1999.

[13] E. Angerson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammarling, J. Demmel, C. Bischof and D. Sorensen, *LAPACK: A portable linear algebra library for high-performancecomputers*, 1990, pp. 2-11.

[14] L. S. Blackford, A. Cleary, J. Choi, E. d'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry and A. Petitet, *ScaLAPACK user's guide*, Society for Industrial Mathematics, 1997.

[15] S. Browne, J. Dongarra, N. Garner, G. Ho and P. Mucci, *A portable programming interface for performance evaluation on modern processors*, International Journal of High Performance Computing Applications, 14 (2000), pp. 189.

[16] F. Cappello and D. Etiemble, *MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks*, *Supercomputing, ACM/IEEE 2000 Conference*, 2000, pp. 12-12.

[17] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker and R. C. Whaley, *ScaLAPACK: A portable linear algebra library for distributed memory computers—design issues and performance*, Computer Physics Communications, 97 (1996), pp. 1-15.

[18] Cluster Resources, *MOAB workload manager*, http://www.clusterresources.com/products/moab-cluster-suite/workload-manager.php.

[19] Cluster Resources, *Torque Resource Manager*, http://www.clusterresources.com/products/torque-resource-manager.php.

[20] C. Coarfa, Y. Dotsenko, J. Mellor-Crummey, F. Cantonnet, T. El-Ghazawi, A. Mohanti, Y. Yao and D. Chavarra-Miranda, *An evaluation of global address space languages: Co-Array Fortran and Unified Parallel C*, 2005, pp. 36-47.

[21] L. Dagum and R. Menon, *OpenMP: An Industry-Standard API for Shared-Memory Programming*, IEEE Computational Science & Engineering (1998), pp. 46-55.

[22] David Mackenzie and T. Tromey, *The Automake Manual*, http://www.delorie.com/gnu/docs/automake/automake_toc.html.

[23] J. Dongarra, J. D. Cruz, I. Duff and S. Hammarling, *A set of Level 3 basic linear algebra subprograms,* , ACM Trans. Math. Software, 16 (1990), pp. 1-17.

[24] Eclipse Foundation, *Eclipse*, http://www.eclipse.org.

[25] T. El-Ghazawi, W. Carlson, T. Sterling and K. Yelick, *UPC: Distributed Shared Memory Programming*, Wiley, 2005.

[26] M. P. I. Forum, *{MPI}: {A} Message-Passing Interface Standard*, Int. Journal of Supercomputing Applications and High Performance Computing, 8 (1994), pp. 165--416.

[27] J. L. Furlani and P. W. Osel, *Abstract yourself with modules*, *USENIX Tenth System Administration Conference (LISA '96)* 1996.

[28] Gary V. Vaughn, Ben Ellison, Tom Tromey and I. L. Taylor, *GNU Autoconf, Automake, and Libtool* Sams, 2000.

[29] M. Geimer, F. Wolf, B. J. N. Wylie, E. Abraham, D. Becker and B. Mohr, *The Scalasca performance toolset architecture*, *Int'l Workshop on Scalable Tools for High-End Computing (STHEC)*, 2008.

[30] S. L. Graham, P. B. Kessler and M. K. McKusick, *Gprof: A call graph execution profiler*, ACM New York, NY, USA, 1982, pp. 120-126.

[31] W. Gropp and K. Buschelman, *FPMPI-2 fast profiling library for MPI*, www-unix. mcs. anl. gov/fpmpi (2006).

[32] I. F. Haddad, *PVFS: A parallel virtual file system for linux clusters*, Linux Journal, 2000 (2000).

[33] IBM, *LoadLeveler*, http://www-03.ibm.com/systems/clusters/software/loadleveler/index.html.

[34] ISO/IEC JTC 1, *Fortran 2003 -- ISO/IEC 1539 Standard*, http://www.nag.co.uk/SC22WG5/IS1539-1_2003.html.

[35] ISO/IEC JTC 1, *Programming Languages -- C, ISO/IEC 9899:TC2 Standard*, http://www.open-std.org/JTC1/SC22/WG14/www/C99RationaleV5.10.pdf.

[36] Jeanne C. Adams, Walter S. Brainerd, Richard A. Hendrickson, Richard E. Maine, Jeanne T. Martin and B. T. Smith, *The Fortran 2003 Handbook: The Complete Syntax, Features and Procedures* Springer-Verlag, 2008.

[37] Jeffrey Vetter and C. Chambreau, *mpiP: Lightweight, Scalable MPI Profiling*, http://mpip.sourceforge.net/.

[38] L. Jianwei, L. Wei-keng, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher and M. Zingale, *Parallel netCDF: A High-Performance Scientific I/O Interface*, *Supercomputing, 2003 ACM/IEEE Conference*, 2003, pp. 39-39.

[39] S. Kannan, M. Roberts, P. Mayes, D. Brelsford and J. F. Skovira, *Workload management with loadleveler*, IBM International Technical Support Organization (2001), pp. 6038-00.

[40] R. Latham, R. Ross and R. Thakur, *The Impact of File Systems on MPI-IO Scalability*, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 2004, pp. 87-96.

[41] C. Lawson, R. Hanson and D. Kincaid, *Basic linear algebra subprograms for Fortran usage*, ACM Trans. Math. Software, 5 (1878), pp. 308-323.

[42] D. Mackenzie, *Autoconf manual*, http://www.delorie.com/gnu/docs/autoconf/autoconf_toc.html.

[43] G. Matzigkeit, *The Libtool Manual*, http://www.delorie.com/gnu/docs/libtool/libtool_toc.html.

[44] Microsoft, *Visual Studio*, http://msdn.microsoft.com/en-us/vstudio/default.aspx.

[45] W. E. Nagel, A. Arnold, M. Weber and K. Solchenbach, *VAMPIR: Visualization and analysis of MPI resources*, Supercomputer (1996).

[46] NSF, *TeraGrid Phase III: eXtreme Digital Resources for Science and Engineering (XD). Program Solicitation 08-571*, http://www.nsf.gov/pubs/2008/nsf08571/nsf08571.pdf.

[47] R. W. Numrich and J. Reid, *Co-array Fortran for parallel programming*, SIGPLAN Fortran Forum, 17 (1998), pp. 1--31.

[48] Platform, *Load Sharing Facility*, http://www.platform.com/Products/platform-lsf.

[49] Samuel P. Harbison and G. Steele, *C; A Reference Manual*, Prentice-Hall, 2002.

[50] F. Schmuck and R. Haskin, *GPFS: A shared-disk file system for large computing clusters*, 2002.

[51] P. Schwan, *Lustre: Building a file system for 1000-node clusters*, 2003.

[52] Sun Microsystems, *Lustre*, www.lustre.org.

[53] I. L. Taylor, *The GNU configure and build system*, http://www.airs.com/ian/configure/.

[54] T. Technologies, *TotalView debugger*, http://www.totalviewtech.com/products/totalview.html.

[55] J. S. Vetter and M. O. McCracken, *Statistical scalability analysis of communication operations in distributed applications*, ACM New York, NY, USA, 2001, pp. 123-132.

[56] G. Watson, C. Rasmussen and B. Tibbitts, *Application development using eclipse and the parallel tools platform*, ACM New York, NY, USA, 2006.

[57] O. Zaki, E. Lusk, W. Gropp and D. Swider, *Toward scalable performance visualization with Jumpshot*, International Journal of High Performance Computing Applications, 13 (1999), pp. 277.