Massively Parallel Message Passing:

Time to Start from Scratch?

Anthony Skjellum, PhD

University of Alabama at Birmingham
Department of Computer and Information Sciences
College of Arts and Sciences
Birmingham, AL, USA
tony@cis.uab.edu

Abstract

Yes: New requirements have invalidated the use of MPI and other similar library-type message passing systems at *full scale* on systems of 100's of millions of cores. Uses on substantial subsets of these systems should remain viable, subject to limitations.

No: If the right set of implementation and application decisions are made, and runtime strategies put in place, existing MPI notations can used within exascale systems at least to a degree.

Maybe: Implications of instantiating "MPI worlds" as islands inside exascale systems cannot damage overall scalability too much, or they will inhibit exascalability, and thus fail.

Categories and Subject Descriptors F.1.2. [**Theory of Computation**]: Modes of Computation - Parallelism and concurrency.

General Terms Performance, Design, Reliability, Experimentation, Standardization, Languages.

Keywords message passing, exascale; MPI; CSP; MPI/RT; explicit parallel programming; disruptive innovation; scalability

1. Introduction

The key parallel programming model of the 1980's—communicating sequential processes [1], as implemented by independent processes connected with runtime libraries providing message passing (send-receive, global operations, or both), has been successfully implemented (*e.g.,* vendor message passing libraries for disparate systems, P4 [2], PARMACS [3], PVM [4], Intel NX [5], Zipcode [6], MPL [7], etc), standardized (*cf,* MPI Forum [8], MPI-1 [9]), extended (*cf,* MPI-2 [10], MPI/RT [11]), and put into broad use, supporting enhanced performance portability. Extremely high performance systems have been designed over the past fifteen years, including systems designed to accelerate MPI-1 primitives (*e.g.,* Portals [12], ASCI Red Storm [13], Cray XT5 [14]). However, we have the programming notation and abstractions of the early 1980's as the underpinnings of these systems, plus some early 1990's put/get semantics, and these primitive abstractions do not do enough to abstract *how* operations are performed compared to *what* is to be performed. Extensions to and attempts to standardize with distributed shared memory (*cf,* MPI-2 one-sided primitives [10]) have been included, but are currently somewhat incompatible with PGAS languages and systems [15], and furthermore MPI isn't easy to mutually convert with BSP-style notations or semantics either [16]. Connection with publish-subscribe models [17] used in military / space-based fault-tolerant message passing systems are absent (see also [18]).

The hypothesis of this paper is that new requirements arising in exascale computation have at least partly invalidated the use of the parallel model expressed and implied by MPI and other similar library-type message passing systems at *full scale* on systems of many millions of cores (or other computational entities). Uses on representative subscales of these systems should remain viable, subject to limitations of implementations, usages by applications, and composition (cross-platform integration). Backward compatibility can perhaps be better obtained once exascale architecture that achieves performance and scalability is first established.

Nonetheless, the author posits that many efforts will try to drive the legacy system forward using MPI globally (via a feasible path of adaptation), as this follows the common evolution of large-scale systems: adaptability of the system and its runtime software to keep valuable applications working is the common history of many if not most applications of any size, enduring value, and extended lifetime ("legacy").

Combination of "old" and "new" appears to be the best way to proceed (balancing technical, practical, adoption risks, and economic concerns), allowing for a complete rethink of massively scalable runtime systems and data motion, and then adding the layering of suitable portions of MPI or MPI-like semantics (and syntax) onto enclaves of those systems, implemented in ways that do not imply global impact on scalability as a result of local implementation decisions, or local utilization of such notations.

Exascale requirements include the following: management of systems with increasingly higher risk of frequent faults, coping with the growing complexity of systems: memory hierarchy, network hierarchy, heterogeneous memory types, massively parallel components with heterogeneity, apparent penalties associated with skew of clocks, operations, and activities across an ensemble, and a sliding scale of memory size to computational entity (core, thread, warp) granularity. Because of these requirements, certain data structures that could be enumerated previously will not be able to be enumerated/stored in future systems at full scale; global state, in short, becomes highly problematic.

SIGPLAN'05 June 12–15, 2005, Location, State, Country.

2. Reviewing the Current Programming Model

The following aspects underpin explicit parallel programming based on message passing today:

- The systems are "fault free": faults are hidden from the CSP system, so applications have no fault models [9].
- The systems behave deterministically within bounds, including bounds on communication delays [1,9].
- The systems are compositions of sequential programs implemented as traditional Unix processes or POSIX threads (or similar) plus *library- based* primitives [1,9].
- Each process has sufficient store to hold the complete local state and partial global state of the message passing system, including enumerated lists representing connections, ranks, communicating partners, buffers, and in-progress data motion. Storage complexity is linear in the number of addressable entities, with superlinear storage complexity plausible in some situations.
- Scheduling, resources, and programs often will not exhibit single-peaked execution times; there is often considerable jitter, and cascade failures in a performance sense are possible because of a lack of predictability of execution time at massive scale.
- Progress of transfers may or may not be well separated from the execution of user threads, depending on implementation, and even within an implementation depending on fabrics traversed; this closely relates to nonblocking and asynchronous operations and their completion semantics.
- Middleware does not necessarily strive minimize time to completion of programs, but rather zero-message latency and asymptotic bandwidth of ping-pong benchmarks, or they may attempt to make trades between overhead and latency.
- It is generally accepted that hundreds of CPU instructions must be executed as part of transfer "sends" and "receives".
- Overlapping of communication, computation, and I/O is not strongly supported or emphasized in real application uses of the systems.
- Because of the goal for zero-copy semantics for large transfers, and even zero-sided communication (*i.e.,* MPI/RT [11]), as well as to avoid internal buffering assumptions, data motion is not transactional, usually not reversible, and programs are in dire straits if a fault occurs; once written, user buffers cannot be unwritten.
- APIs and semantics of the system have costs, even when those subsystems are not engaged, or program semantics are simpler than the general case (*e.g.,* use of MPI tags in communication of SPMD programs with static resource utilization and no wildcard receives).
- Planned transfer for temporal locality (static or persistent communication) is mostly deemphasized, even though it is a source of considerable opportunity for runtime optimizations at scale. MPI-3 appears prepared to address these concerns at least partially [19,26].
- Library implementations are mandated, if not by users, then by tool concerns, and also by the standards [9,10].
- Subsets (profiles) have no benefits for performance or scaling, whether because the standards fail to recognize subsets, or because subsets across classes of functionality and subclasses of operations do not lead to the same quantitative benefits on different platforms.
- Compilers don't do anything to help at scale.
- It is commonly recognized that few if any parties have sufficient funding to build complex new runtime systems and middleware and/or domain-specific languages [21] to offer a competent alternative.

3. Facts of Exascale Life

MPI library implementations and specifications (middleware) will likely impede exascale systems performance, though providing continued nominal portability. Whether because of faults, jitter, memory to compute-entity granularity mismatch, or for several of these factors, MPI libraries will imply heavier and heavier relative overheads compared to that which resulted in existing and earlier systems architectures. Such middleware will therefore need to be "disrupted" [20] by newer explicit parallel programming models that at first are perhaps lower in performance, portability, and/or productivity, but which will have the ability over time to overtake legacy MPI systems at exascale, thereby replacing them as a dominant way of programming parallel systems.

Legacy systems will continue to use MPI, whether as a library, or as a notation implemented by a domain-specific language (DSL) [21] that provides enough MPI-type capability to move MPI applications to exascale enclaves (fractional sizes). Between enclaves, significant changes to applications will be needed top-to-bottom. And inter-enclave communication and interactions cannot be fundamentally impaired by the intra-enclave communications. Several level hierarchical systems may also appear.

For applications that truly require (and deserve) exascale systems, the reality that runtime and message passing system changes are mandated is analogous to the fact that for maximum scalability, convergence must be sacrificed (*e.g.,* Gauss-Seidel vs. Gauss-Jacobi iteration, or ILU vs. Jacobi preconditioning, or between nonlinear and linear levels of a nonlinear system solver). Many researchers have pointed out such tradeoffs and they are commonly incorporated into sequential and parallel solvers. Analogous tradeoffs between the so-called fastest sequential algorithm and the so-called fastest parallel algorithm have been reported in some situations, though not in others. While this has sometimes led to better overall algorithms, for many classes of systems, less precise and tuned systems with more intermediate internal error are needed to achieve maximum scalability. So it should come as no surprise that runtime systems and expressions of parallel computations would also have to be weakened and be less "human tractable" and "safe" at exascale, given that the underlying algorithms must themselves compromise to achieve scalability, and allow for more error and uncertainty. (Shannon's Information Theory evidently plays a key role in (exa)scalability [25].)

4. Analogy to N-Body Simulations

The scalability of future message passing systems almost definitely requires that they do even less to damage the overall scalability of systems than have previous middleware layered on, underlying APIs, protocols, and networks. Since layered systems cannot generally add to scalability, but only reduce or prove neu-

tral, layered systems such as are commonly implemented today represent a serious challenge to systems designs of the future.

N-body computations [21] illustrate the need for systems that might have locally quadratic (all-to-all) state between "near" entities, yet globally have only linear time complexity, and lumped (vague) state information. This analog is probably of great value in composing massively parallel exascale systems... they will also have to be always on, slow to start up and slow down, constantly failing in parts, and always computing, and always adapting. Furthermore a portion of the system has only a vague concept of portions of the system far away, unless there is a huge amount of repetitive traffic, coupling, or interaction, in which case those arcs of connectivity a) represent closeness, b) represent a justification for on-going accretion of additional shared state information.

5. Legacy Drives Research and Standardization

Conserving existing C/C++/Fortran plus MPI applications restricts and drives almost all the efforts of the current MPI Forum [8], with regard to explicit parallel programming. There may be no "normal science" [24] path to its exascale successor, but that as evidence grows for the existing systems having limitations, a fresh look at how to compose the applications, runtime systems, data motion, algorithms, applications, and scheduling must be done to achieve exascalability. Classes of the most demanding systems will not initially be easily susceptible to the parts of the performance-portability-productivity space where MPI has up to now proven quite useful. Emergence of disruptive parallelism such as GPUs [22], which are "hostile" to the MPI model, may help to drive the transition from conserving CSP at all scales.

6. Conclusions

Full exascale systems can't be programmed with an expectation of feasibility/scalability with MPI-type middleware. Far greater control of resources (static and dynamic) is needed. Starting from requirements and reintroducing MPI enclaves as object-oriented/aspect-oriented instantiations within limited subsystems appears plausible. Long-range, repetitive, and highest demand transfers, including transfers between subsystems that do not mutually synchronize, will have to be non-blocking, reversible, and/or (split-phase) transactional. Failure and retry will have to be part of these operations, and at least some failures will have to be exposed to applications. Computational sessions will have to build up, rather than build down. Exascalability means accepting higher latency, and so requires more latency hiding. Performance decisions made near the source of messages that require global state will not be possible for maximum scalability (*e.g.,* source routing). Scalability and availability decisions (made near destinations) will require the delivery of messages to have hierarchical demultiplexing, and potentially oversubscribed resources and reactive program semantics in order to achieve scalability. Messages themselves may need internal concurrent semantics so that they can be decomposed and recomposed in flight for variable granularity and flexible remote processing. Greater asynchrony will be needed. Single research groups will not be able to implement the runtime systems and libraries; systems will have to interoperate. Greater emphasis on predictability and low jitter in scheduling will be needed at full scale to avoid loss of significant performance. Global state must be minimal. Fault models may well have to be managed as aspects to improve portability.

References

[1] Hoare, C.A.R., ed., *Developments in Concurrency and Computation, Addison-Wesley, 1990.*

[2] Butler, R. M and Lusk, E. L, *Monitors, Messages, and Clusters: the p4 Parallel Programming System,* URL: http: //www.mcs .anl.gov/~lusk/oldpapers/p4/paper.html., Accessed: 3/31/2010.

[3] R. Calkin et al, *Portable programming with the PARMACS message-passing library, Par. Comp, 20(4), pp. 615-632, 1994.*

[4] Geist, et al, *PVM: Parallel Virtual Machine,* MIT Press, 1994.

[5] Pierce, P. *The NX Message Passing Interface, Par. Comp, 20(4), pp.* 463-480*, 1994.*

[6] Skjellum et al, *The Design and Evolution of Zipcode, Par. Comp, 20(4), pp.* 565--596*, 1994.*

[7] International Business Machines Corporation, MPL-to-MPI Conversion Reference, URL: http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.pe.do/ /pe_422/am10400430.html, Accessed: 3/31/2010.

[8] MPI Forum, URL: http://mpi-forum.org. Accessed: 3/31/2010.

[9] MPI Forum, URL: http://mpi-forum.org/docs/mpi-10.ps. Accessed: 3/31/2010.

[10] MPI Forum, MPI 2.2 Standard, URL: http://mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf. Accessed: 3/31/2010.

[11] Skjellum, A., Kanevsky, A, et al. *The Real-Time Message Passing Interface Standard (MPI/RT-1.1*) CCP&E 16(S1): 0-322, 2004.

[12] Brightwell, R, *The Portals 3.0 Message Passing System,* URL: www.cs.sandia.gov/pub/papers/bright/portals3v1-1.ps.gzwww.cs.sandia.gov/pub/papers/bright/portals3v1-1.ps.gz. Accessed: 3/31/2010.

[13] Sandia National Laboratories, Red Storm Home Page, http://www.sandia.gov/ASC/redstorm.html. Accessed: 3/31/2010.

[14] Cray, Inc: http://www.cray.com/Products/XT/Systems/XT5.aspx. Downloaded 3/31/2010.

[15] Chamberlain, B. L, and Callahan D., and Zima Hans P., *Parallel Programmability and the Chapel Language, DARPA HPCS,* www.highproductivity.org/HPPLM/final-chamberlain.pdf. Accessed: 3/31/2010.

[16] Bisseling, R., *Parallel Scientific Computing: A Structured Approach using BSP and MPI,* Oxford University Press, USA, 2004.

[17] Eisenhauer, G., Schwan, K, Bustamante, F, *Publish-Subscribe for High-Performance Computing,* IEEE Internet Computing, vol. 10, no. 1, pp. 40-47, Jan./Feb. 2006, doi:10.1109/MIC.2006.16.

[18] Batchu, R, Skjellum, A, et al, *MPI/FTTM: Architecture and Taxonomies for Fault-Tolerant, Message-Passing Middleware for Performance-Portable Parallel Computing,* pp.26, CCGrid'01, 2001.

[19] MPI Forum, *Preview of the MPI-3 Standard.* www.open-mpi.org/papers/sc-2009/MPI_Forum_SC09_BOF-2up.pdf. Accessed 3/31/2010.

[20] Christensen, C. M., *The Innovator's Dilemma*, HarperCollins, 2003.

[21] van Deursen, A., Klint, P., and Visser, J. 2000. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.* 35, 6 (Jun. 2000), 26-36. DOI= http://doi.acm.org/10.1145/352029.352035.

[22] Lars S. Nyland and Jan F. Prins and John H. Reif, *A Data-Parallel Implementation of the Adaptive Fast Multipole Algorithm,* Proc. of the 1993 DAGS/PC Symposium, June 1993, Dartmouth College.

[23] GPGPU.org: http://gpgpu.org/. Accessed 3/31/2010.

[24] Kuhn, Thomas, *Structure of Scientific Revolutions, 3/e, University of Chicago Press, 1996.*

[25] Shannon, C.E., *A Math. Theory of Comm., Bell System Technical Journal*, 27, pp. 379–423 & 623–656, July & October, 1948.

[26] T. Hoefler, P. Gottschling and A. Lumsdaine, *Leveraging Non-blocking Collective Communication in High-performance Applications, In Proc. of the 20th Annual Symposium on Par. in Alg. & Arch., SPAA'08, Munich, Germany, pp. 113-115, ACM, Jun. 2008.*