;login:

inside:

DEVELOPMENT

Lumb: Production HPC Reinvented

production HPC reinvented

Overview

Over the years, production High Performance Computing (HPC) was synonymous with scientific computing on "Big Iron" supercomputers. No longer dominated by just physical scientists and their Grand Challenge Equations, production HPC now embraces a variety of compute architectures. Though framed in the broader context of non-traditional HPC, attention here focuses on parallel computing via the Message Passing Interface (MPI). Problems cast as MPI applications are seen to have a parallel-computing bias that reaches back into the numerical methods that have been used and even to the originating science. Whereas MPI itself shows significant promise in addressing current computing challenges, in practice some serious shortcomings must be addressed in order for production HPC to be realized.

by Ian Lumb

Ian Lumb is a product solutions architect at Platform Computing, Inc. His interests include computing architectures, parallel computing, parametric processing, and grid computing. In a former life, Ian was a physical scientist working on problems in global geophysics at York University (Toronto, Canada).

ilumb@platform.com

Workload-management system software closes the gap between MPI applications and their compute architectures, resulting in a solution for production HPC. A specific example of production HPC for the Linux operating environment shows that such solutions exist today. Moreover, the workload-management methodologies that apply at the cluster level have a natural affinity for extension to the Grid. Overall, organizations are able to better empower the pursuit of science and engineering during MPI application development, deployment, and use.

Five Steps to Scientific Insight

To motivate the applications and architectures discussion, consider a scientific-inquiry example from the physical sciences.[1] Once the problem under investigation has been determined, the first task is to determine the relevant physics, chemistry, etc. (Figure 1, Step 1). This consideration results in a mathematical description of the problem that needs to be solved (Figure 1, Step 2). On the positive side, the mathematical description typically exists, i.e., there is rarely a need to invent the mathematical description. In many cases, the required mathematical description can be formulated by combining existing descriptions. Although mathematics is the oldest and most deeply explored discipline, mathematical methods are often insufficient, except in idealized situations subject to simplifying assumptions, to solve the resulting equations. In mathematical terms, it is often difficult to near impossible to derive *analytic* solutions to many scientific equations. To make matters worse, in some cases it is difficult to prove that such solutions even exist. Such existence theorems serve as a cornerstone for the practice of mathematics. Thus science exposes serious mathematical challenges – in stark contrast to our childhood experiences with mathematics.

Given this challenging mathematical context, numerical methods are used to permit progress on otherwise unsolvable scientific problems (Figure 1, Step 3). Typically, this involves a discrete representation of the equation(s) in space and/or time, and performing calculations that trace out an evolution in space and/or time.[2] It's important to note that the underlying structure of the resulting set of equations influences the types of numerical methods that can be applied.[3]

Thus, numerical experiments are acts of modeling or simulation subject to a set of pre-specified constraints (Figure 1, Step 4). Problems in which time variations are key need to be seeded with *initial conditions*, whereas those with variations in space are

1. The analogous steps for informatics-heavy sciences such as the life sciences will be left for future consideration.

2. Symbolic algebraic manipulation (SAM) provides a notable analytic exception to this discrete approach. Maple ([1]) serves as a representative example of SAM technology. It is not uncommon to use SAM in conjunction with the discrete approach described here.

3. There is an extensive literature base on this topic. Implementations of equations involving a collection of methods are often organized into libraries.
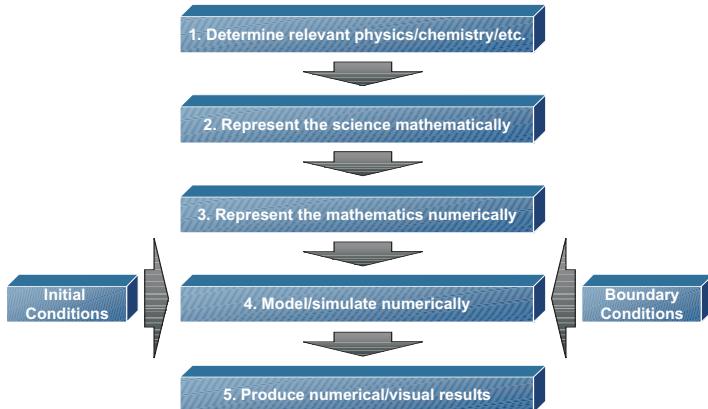
Figure 1. Five steps to scientific insight

subject to *boundary conditions*; it is not uncommon for problems to specify both kinds of constraints. The numerical model or simulation subject to various constraints can be regarded as a scientific application. Thus the solution of a scientific problem results in numerical output that may or may not be represented graphically (Figure 1, Step 5). One of four primary types (see "Applications and Architectures", below), this application is further regarded as scientific workload that needs to be managed as the calculations are carried out. Because the practices of science and engineering are actually undertaken as a process of discovery, Figure 1 should be regarded as a simplifying overview that does not depict the recursive nature of investigation.

It may appear that numerical methods are the cure-all for any scientific problem that cannot be solved by mathematical methods alone. Unfortunately, that is not the case. On their own, many of the equations of classical physics and chemistry push even the most powerful compute architectures to the limits of their capability. Irrespective of numerical methods and/or compute capability, these Grand Challenge Equations afford solutions based on simplifying assumptions, plus restrictions in space and/or time. Because these particularly thorny equations are critical in science and engineering, there is an ongoing demand to strive for progress. Examples of Grand Challenge problems are provided elsewhere ([2]).

Applications and Architectures

Science dictates mathematics and mathematics dictates numerics[4] (Figure 1). Thus a numerics bias exists in all applications of scientific origin. This predisposition motivates four types of applications (Figure 2) revealed by exploring process granularity. Granularity refers to the size of a computation that can be performed between communication or synchronization points ([3]). Thus, any point on the vertical axis of Figure 2 identifies a specific ratio of computation (increasing from bottom to top) to communication (increasing from top to bottom).[5] Task parallelism, increasing from left-to-right on the horizontal axis, refers to the degree of parallelism present in the application; "fine" through "coarse" are used as qualitative metrics, as shown.

4. "Numerics" is used here as a shorthand for numerical methods.

5. If the interest is computing, then communication is viewed as the "overhead' required to achieve the computation. Similarly, computation might be regarded as the overhead required to facilitate certain communications.

6. The mathematical convention of numbering quadrants counter-clockwise from the upper-right-hand corner is used here.

7. Although high-density, rack-mounted single/dual processor servers have been used in compute farms, there is an intensifying trend toward the use of higher density blade servers in these configurations.

Most scientific problems are implemented initially as *serial* applications (Figure 2, Quadrant II).[6] These problems require that each step of the scientific calculation be performed in sequence. Serial applications can be executed on compute architectures ranging from isolated desktops, servers, or supercomputers to compute farms. Compute farms are loosely coupled compute architectures in which system software is used to virtualize compute servers[7] into a single system environment (SSE).

Various factors – time-to-results, overall efficiency, etc. – combine to demand performance improvements beyond what can be achieved by "legacy" serial applications alone. For those applications whose focus is on data processing, it is natural to seek and exploit any parallelism in the data itself. Such *data parallel* applications (Figure 2, Quadrant I) are termed *embarrassingly parallel* since the resulting applications
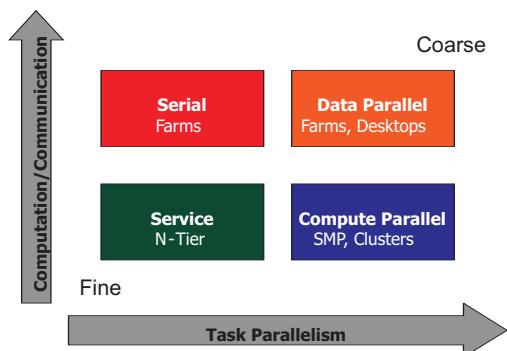

Figure 2. Applications and architectures

are able to exploit the inherent coarse granularity. The parallelism in data is leveraged by (Figure 3):

- Subdividing input data into multiple segments;
- Processing each data segment independently via the same executable; and
- Reassembling the individual results to produce the output data.

This data-driven approach accounts for one of four classes of parametric processing (Figure 4). Although data-parallel applications are a solid match for isolated systems and compute farms, there is an increasing trend to harvest compute cycles from otherwise idle desktops. Regarded as opportunistic compute resources, desktops "pull" processing tasks from coordinating servers that they will execute as a background process – especially in place of a traditional screensaver. Desktops as compute elements gained initial popularity through the peer-to-peer (P2P) movement and more recently in the context of grid computing. Ray-tracing applications provide a classic demonstration of this processing architecture. Despite the added challenge of managing data, successful implementations of embarrassingly parallel applications exist in many industries – genome sequencing in the life sciences, Monte Carlo simulations in high energy physics, reservoir modeling in petroleum exploration, risk analysis in financial services, and so on. This approach is so appealing and powerful that it's often perceived to be of general utility. Unfortunately, this simply isn't the case – and this is especially true for the Grand Challenge Equations identified previously.

Input   Output
Engine

Figure 3. Data-driven parametric processing

If it exists at all, parallelism in the Grand Challenge Equations can be exploited at the source-code level – e.g., by taking advantage of loop constructs in which each calculation is independent of others in the same loop. Parallelism in data is absent or of minor consequence. This code-level parallelism lends itself to *compute parallel* (Figure 2, Quadrant IV) applications. Compute parallel applications are further segmented on the basis of memory access – i.e., shared versus distributed memory. With minimal language extensions, and explicit code-level directives, OpenMP ([5]) and, more recently, Unified Parallel C (UPC, [6]) offer up parallel computing with shared-memory programming semantics. Symmetric multiprocessing (SMP) systems allow for shared-memory programming semantics via threads[8] through uniform (UMA) and nonuniform (NUMA) memory access architectures.

Parallel Virtual Machine (PVM, [7]) has given way to the Message Passing Interface (MPI, [8]) as the standard for parallel computing with distributed-memory programming semantics. Likened to the "assembly language" for parallel programming ([9]), MPI requires a significant investment at the source-code level.[9] In contrast to the use of threads in the shared-memory context, distributed processes are employed in the MPI case to achieve parallelism. MPI applications are typically implemented for tightly coupled compute clusters (see HPC Application Development Environment, below, for additional details). Although other factors (e.g., architecture access) need to be considered, numerics do influence the choice of parallel computing via shared versus distributed memory.[10] Both shared and distributed-memory parallel computing methodologies have been applied to scientific and engineering problems in a variety of industries – e.g., computational and combinatorial chemistry in the life sciences, computational fluid dynamics, crash simulations and structural analyses in industrial manufacturing, Grand Challenge problems in government organizations and educa-
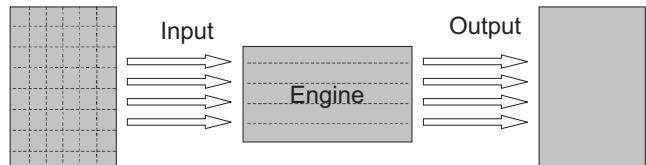
8. In addition to the fork-and-exec creation of child processes, a parent process may also involve threads. Distinguishable by the operating system, threads can share or have their own memory allocations with respect to their parent process.

9. Recent advances allow serial applications to be automatically enabled for MPI ([10]). By identifying code regions suitable for parallelization (e.g., repetitive calculations) via a templating mechanism, code-level modifications are applied. This approach is being applied extensively in financial services, where numerical models change frequently.

10. Hybrid OpenMP-MPI applications allow scientists and engineers to simultaneously use threads and distributed processes when using a combination of SMP and clustered architectures.

tional institutions, and scenario modeling in financial services. MPI compute parallel applications, traditional HPC, will be the focus of our attention here.

Because MPI provides such a rich framework for computing in general, there are examples of MPI applications that communicate extensively while carrying out minimal processing (Figure 2, Quadrant III) – e.g., remote-to-direct-memory applications (RDMA), or certain classes of search algorithms. In addition, *service* applications whose focus is networking itself or Web services ([11]) themselves would also fall into this area. As before, MPI applications would require tightly coupled architectures; whereas networking applications can be applied in a variety of contexts, loosely coupled architectures can be used in the instantiation of Web services.

HPC Application Development Environment

Together with the "commoditization" of low-processor-count, high-density servers and the emergence of low-latency, high-bandwidth interconnect technologies, MPI has played a key role in the widespread adoption of tightly coupled compute clusters for distributed memory-parallel computing ([12]):

MPI is available everywhere and widely used in environments ranging from small workstation networks to the very largest computers in the world, with thousands of processors. Every parallel computer vendor offers an MPI implementation, and multiple implementations are freely available as well, running on a wide variety of architectures. Applications large and small have been ported to MPI or written as MPI programs from the beginning, and MPI is taught in parallel programming courses worldwide.

Applied through source-code-level modifications, MPI-specific directives are referenced against an MPI library at application link time. MPI libraries are architecture specific and may come from a variety of sources – e.g., a system vendor, an interconnect vendor, or via an open source contribution. In each case, the relevant library implements the MPI specification11 to some degree of compliance. This MPI library, in combination with the tools and utilities that support developers, collectively forms the application development environment for a particular platform (Figure 4).

The situation described above might lead one to conclude that all of the requisites are present to smoothly enable MPI adoption. In practice, however, MPI has the following challenges:

- Resynchronization and reconnection were not even factored in at the specification level ([9]). There are no MPI implementations that account for this shortcoming. This is in striking contrast to PVM, whose implementation allows for this.
- Fault tolerance was not factored in, even at the specification level ([9]); again, there are no MPI implementations that account for this shortcoming. This can mean, for example, that an application can lose some of its processes, run to completion, and yield results of dubious validity.
- Hosts and numbers of processors need to be specified as static quantities, irrespective of actual usage conditions.
- Single point of control is absent. Although some recent MPI implementations offer a process daemon to launch MPI applications, there is little in the way of real application control.
- Multiple versions of MPI may exist on the same architecture. Applications need to carefully identify the relevant MPI libraries. The situation is more complex for MPI applications that span more than one execution architecture.

11. Most MPI libraries fully implement version 1.x of the MPI specification, while many libraries are today supporting some subset of the version 2.x specification.

The upshot is clear: MPI places the responsibility for these shortcomings on the application developer and user. Because MPI applications are a challenge to control and audit, a better production HPC solution is required.

Production HPC Reinvented

There is a gap between the potential for distributed-memory parallel computing via MPI and what is actually achievable in practice. The use of system software allows this gap to be closed and the promise of MPI to be fully realized. In the process, the notion of production HPC is redefined. To start, consider a modified version of Figure 4 in which the newly added workload-management system software is shown in black on white (Figure 5).

Figure 5 introduces the following three components to the MPI application development environment:

- **Core workload management services.** This system software component allows a heterogeneous collection of compute servers, each running its own instance of an operating system, to be virtualized into a compute cluster.[12] Sensory agents are used to maintain static and dynamic information in real time across the entire compute cluster. Primitives for process creation and process control across a network are also provided.
- **Parallel application management.** Challenges specific to the management of MPI parallel applications include the need to:
  - Maintain the communication connection map;
  - Monitor and forward control signals;
  - Receive requests to add, delete, start, and connect tasks;
  - Monitor resource usage while the user application is running;
  - Enforce task-level resource limits;
  - Collect resource usage information and exit status upon termination; and
  - Handle standard I/O.
- **Parallel scheduling services.** Workload management solutions typically employ a policy center to manage all resources – e.g., jobs, hosts, interconnects, users, and queues. Through the use of a scheduler, and subject to predefined policies, resource demands are mapped against the supply of resources in order to facilitate specific activities. Scheduling policies of particular relevance in parallel computing include advance reservation, backfill, preemption, and processor and/or memory reservation.

The combined effects of these three layers of a workload-management infrastructure allow the shortcomings of MPI to be directly addressed:

- Absence of resynchronization and reconnection: Although the workload-management infrastructure (Figure 5) cannot enable resynchronization or reconnection, by introducing control across all of the processes involved in an MPI application, there is greatly improved visibility into synchronization and/or connection issues.
- Absence of fault tolerance: At the very least, the introduction of a workload-management infrastructure provides visibility into exceptional situations by trapping and propagating signals that may be issued while workload is executing. These signals can be acted upon to automatically re-queue workload that has thrown an undesirable exception. Even better, when integrated with the checkpoint/restart infrastructure of a workload manager, interrupted workload can continue to execute from the last successful checkpoint, often without user intervention.
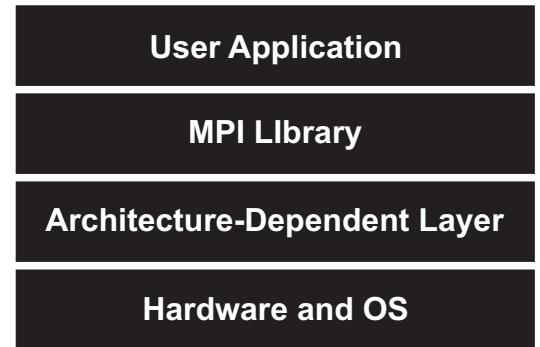

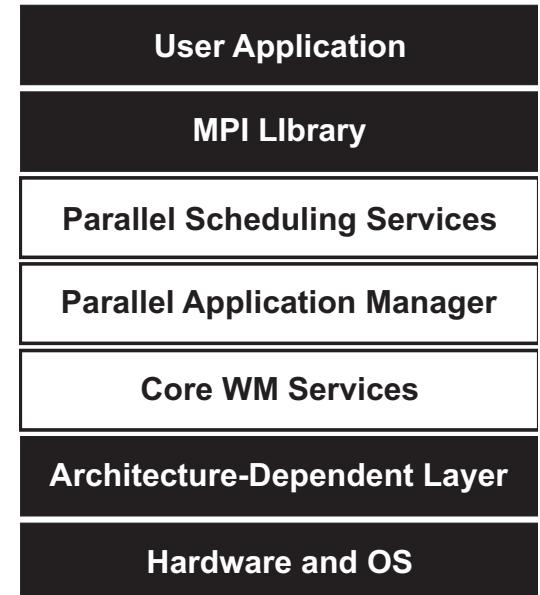Figure 4. MPI application development environment

Figure 5. Enhanced application development environment via workload-management system software

12. This layered-services approach has been contrasted with Beowulf clustering (via a distributed process space) elsewhere ([13]).

13. Experience dictates that every parallel application show acceptable performance characteristics over a range of processors. A typical criterion is that the performance remain close to linear as the number of processors increases. This is referred to as "linear speedup." Effective workload-management systems allow this processor count to be specified as a range at workload submission time. This serves to automate the load balancing situation and to enhance overall effectiveness of the scheduling services.

14. The need to bind processes to processors serves as one example of an execution environment requirement. In such cases, the workload-management infrastructure works in tandem with the operating system, interconnect manager, etc., to address the requirement.

15. Historically, Alpha-based processors were used because of their excellent floating-point-performance characteristics. With the advent of first-generation Itanium processor family CPUs, it is expected that 64-bit Intel Architecture (IA-64) will eventually dominate in this space. In the interim, fourth-generation IA-32 Pentium processor family CPUs hold the price/performance niche.
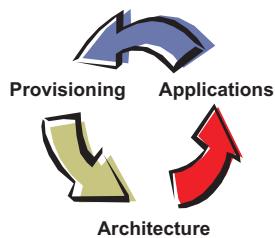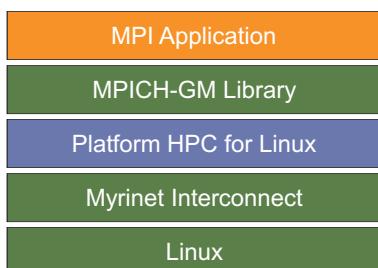

Figure 6. Production HPC reinvented

Figure 7. Production HPC for Linux

- Absence of load balancing: The need to explicitly identify hosts and numbers of processors can be regarded as an absence of load balancing – i.e., the specification of resources that ignores current resource-usage conditions. Because workload-management infrastructures maintain dynamic load state across the entire compute infrastructure in real time, this shortcoming is completely eliminated.
- Absence of single point of control: A parallel application manager provides a single point of control. This ensures that all the distributed processes that collectively make up the MPI application are managed and accounted for. Again a key shortcoming becomes a core competence via workload-management system software.
- Multiple versions of MPI: Through configuration information, MPI applications are able to specify the relevant MPI library. A very flexible architecture also allows the workload-management system software to work in concert with the existing parallel application development and runtime environment.

Together with the existing parallel application development and runtime environment, workload-management system software allows the inherent shortcomings of MPI to be effectively eliminated. The cumulative effect is to practically reinvent HPC via MPI. This threefold reinvention is captured in Figure 6.

The process starts with MPI applications that are developed in-house or acquired from commercial providers.

On job submission, these applications are accompanied by a description of their runtime resource requirements – e.g., a processor-count range,[13] data-management directives (e.g., a file transfer), plus other environmental requirements.[14] The scheduling agent takes into account the pre-specified resource requirements, the unique characteristics of the compute architectures that it has available, and the policies that reflect organizational objectives. Because the scheduler creates a runtime environment for the workload, its task is one of dynamic provisioning. On dispatch, the scheduler has optimally matched the workload to an appropriate runtime architecture subject to established policies. The application executes in a fully managed environment. A comprehensive audit trail ensures that all activities are accounted for. In this way, there is a closed loop for production HPC, one that enhances the developer and user experience rather than encumbering it. A specific solution example is provided in the following section.

Production HPC for Linux

It has been suggested that production HPC can be reinvented through the use of an integrated development and runtime environment in which workload-management system software plays a key role. A complete example is considered here to further illustrate this reinvention. Consider the integrated production HPC solution stack shown in Figure 7. Although this example is based on the Linux operating environment, similar stacks for other operating environments can be crafted.

At the base of the production HPC solution stack for Linux are low-processor-count servers,[15] each running its own instance of the GNU/Linux operating system. Ethernet-based network interface cards (NICs) allow for standard TCP/IP-based services between these systems. Because TCP/IP over Ethernet necessitates assumptions regarding shared access and the occurrence of collisions, the result is a communications protocol that is latency heavy and therefore inefficient for message-passing in support of parallel computing. Hence, each system implements Myricon's GM message-passing

protocol across low-latency, high-bandwidth, multi-port Myricom Myrinet switches ([14]) used solely to support parallel computing via MPI and the GM driver. By identifying each available Myrinet port as a resource to the core workload-management services provided by Platform HPC for Linux ([15]), the provisioning component of this infrastructure is aware of the static and dynamic attributes of the architecture it can apply parallel scheduling policies against. Platform HPC for Linux also provides the control and audit primitives that allow parallel applications to be completely managed. Users' MPI applications need to be compiled and linked against the application development environment provided by Myricom. This ensures that the appropriate GM-protocol modifications to the widely adopted open source MPICH implementation of MPI are used. Through configuration information, the workload-management system software based on Platform HPC for Linux is made aware of the enhanced MPI runtime environment.

Portability was identified as a design goal for MPI. This objective has been carried through in MPI implementations such as MPICH. Despite this fact, heterogeneous parallel applications based on MPI must not only use the same implementation of MPI (e.g., MPICH) but also the same protocol implementation (e.g., GM). In other words, MPI is a multi-protocol API (Figure 8) in which each protocol implements its own message formats, exchange sequences, and so on.

Because they can be regarded as a cluster of clusters, it follows that computational grids might provide a suitable runtime environment for MPI applications. The fact that grids are concerned with resource aggregation across geographic (and other) domains further enhances the appeal. Fortunately, Platform HPC for Linux is consistent with Platform MultiCluster – system software that allows independent clusters each based on Platform LSF to be virtualized into an enterprise grid. This combination introduces the possibility for exciting new modes of infrastructural provisioning through various grid-centric scheduling policies – e.g., Grid Advance Reservation, Grid Fairshare, and Grid Resource Leasing.

Of these grid-centric policies, Grid Resource Leasing (GRL) is particularly novel and powerful. GRL allows sites to make available fractions of their resources for use by other sites participating in their enterprise grid. These collective resources can be occupied by MPI applications through requirement specifications. In this fashion, users can *co-schedule* MPI applications to span more than one geographic location. Even more impressive is the fact that this co-scheduling is automatically enabled at runtime, that is, existing MPI applications do not need to be re-linked with special libraries. By jointly leveraging the parallel application management capability already present in Platform HPC for Linux, in concert with this grid-level scheduling policy of Platform MultiCluster, MPI application users take advantage of their entire enterprise grid in a transparent and effective fashion. Platform MultiCluster and its grid-level scheduling policies are considered in detail elsewhere ([17]).

Figure 8. The multi-protocol nature of MPI (after [16])

Summary

Production High Performance Computing (HPC) incorporates a variety of applications and compute architectures. Widespread use of the Message Passing Interface (MPI) is better enabled through the use of workload-management system software. This software allows MPI applications and the compute architectures on which they execute to be provisioned on demand. This critical link significantly reduces complex-
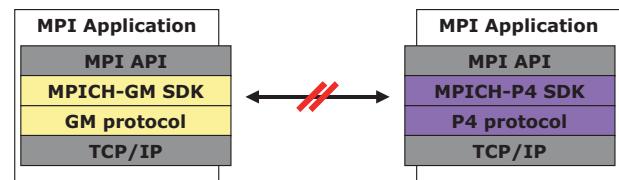
ity for the scientist or engineer, thus reducing time to results and ensuring overall organizational efficiency. A tightly coupled cluster based on the Linux operating environment was shown to be a particularly attractive and viable compute architecture. The incorporation of this environment into a compute grid was also shown to be a natural progression. Overall, organizations are able to better empower the pursuit of science and engineering during application development, deployment, and use.

ACKNOWLEDGMENTS

REFERENCES

[1] Maple, *http://www.maplesoft.com*.

[2] The Grand Challenge Equations, *http://www.sdsc.edu/Publications/GCequations*.

[3] B. Wilkinson & M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers* (Upper Saddle River, NJ: Prentice-Hall, 1999).

[4] I. Lumb, B. Bryce, B. McMillan, and K. Priddy, "From the Desktop to the Grid: Parametric Processing in High-Performance Computing," Sun User Performance Group, Amsterdam, Netherlands, October 2001.

[5] OpenMP, *http://www.openmp.org*.

[6] Unified Parallel C, *http://upc.gwu.edu*.

[7] Parallel Virtual Machine, *http://www.csm.ornl.gov/pvm/pvm_home.html*.

[8] Message Passing Interface, *http://www.mpi-forum.org*.

[9] K. Dowd and C.R. Severance, *High Performance Computing*, 2d ed. (Sebastopol, CA: O'Reilly & Associates, 1998).

[10] Powerllel, *http://www.powerllel.com/powerllel/content/index.shtml*.

[11] Web services, *http://www.w3.org/2002/ws/*.

[12] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, 2d ed. (Cambridge, MA: MIT Press, 1999).

[13] I. Lumb, "Linux Clustering for High-Performance Computing," *;login:*, vol. 26, no. 5, August 2001, pp. 79–84.

[14] Myricom Myrinet, *http://www.myricom.com/myrinet*.

[15] Platform Computing Corporation, "Using LSF with MPICH-GM," technical documentation, April 2002.

[16] MPICH-G2, *http://www.hpclab.niu.edu/mpi*.

[17] Platform Computing Corporation, *Platform MultiCluster Guide*, technical documentation, June 2002.

production HPC reinvented

by Ian Lumb

Ian Lumb is a product solutions architect at Platform Computing, Inc. His interests include computing architectures, parallel computing, parametric processing, and grid computing. In a former life, Ian was a physical scientist working on problems in global geophysics at York University (Toronto, Canada).

ilumb@platform.com

Overview

Over the years, production High Performance Computing (HPC) was synonymous with scientific computing on "Big Iron" supercomputers. No longer dominated by just physical scientists and their Grand Challenge Equations, production HPC now embraces a variety of compute architectures. Though framed in the broader context of non-traditional HPC, attention here focuses on parallel computing via the Message Passing Interface (MPI). Problems cast as MPI applications are seen to have a parallel-computing bias that reaches back into the numerical methods that have been used and even to the originating science. Whereas MPI itself shows significant promise in addressing current computing challenges, in practice some serious shortcomings must be addressed in order for production HPC to be realized.

Workload-management system software closes the gap between MPI applications and their compute architectures, resulting in a solution for production HPC. A specific example of production HPC for the Linux operating environment shows that such solutions exist today. Moreover, the workload-management methodologies that apply at the cluster level have a natural affinity for extension to the Grid. Overall, organizations are able to better empower the pursuit of science and engineering during MPI application development, deployment, and use.

Five Steps to Scientific Insight

To motivate the applications and architectures discussion, consider a scientific-inquiry example from the physical sciences.[1] Once the problem under investigation has been determined, the first task is to determine the relevant physics, chemistry, etc. (Figure 1, Step 1). This consideration results in a mathematical description of the problem that needs to be solved (Figure 1, Step 2). On the positive side, the mathematical description typically exists, i.e., there is rarely a need to invent the mathematical description. In many cases, the required mathematical description can be formulated by combining existing descriptions. Although mathematics is the oldest and most deeply explored discipline, mathematical methods are often insufficient, except in idealized situations subject to simplifying assumptions, to solve the resulting equations. In mathematical terms, it is often difficult to near impossible to derive *analytic* solutions to many scientific equations. To make matters worse, in some cases it is difficult to prove that such solutions even exist. Such existence theorems serve as a cornerstone for the practice of mathematics. Thus science exposes serious mathematical challenges – in stark contrast to our childhood experiences with mathematics.

Given this challenging mathematical context, numerical methods are used to permit progress on otherwise unsolvable scientific problems (Figure 1, Step 3). Typically, this involves a discrete representation of the equation(s) in space and/or time, and performing calculations that trace out an evolution in space and/or time.[2] It's important to note that the underlying structure of the resulting set of equations influences the types of numerical methods that can be applied.[3]

Thus, numerical experiments are acts of modeling or simulation subject to a set of pre-specified constraints (Figure 1, Step 4). Problems in which time variations are key need to be seeded with *initial conditions*, whereas those with variations in space are

1. The analogous steps for informatics-heavy sciences such as the life sciences will be left for future consideration.

2. Symbolic algebraic manipulation (SAM) provides a notable analytic exception to this discrete approach. Maple ([1]) serves as a representative example of SAM technology. It is not uncommon to use SAM in conjunction with the discrete approach described here.

3. There is an extensive literature base on this topic. Implementations of equations involving a collection of methods are often organized into libraries.
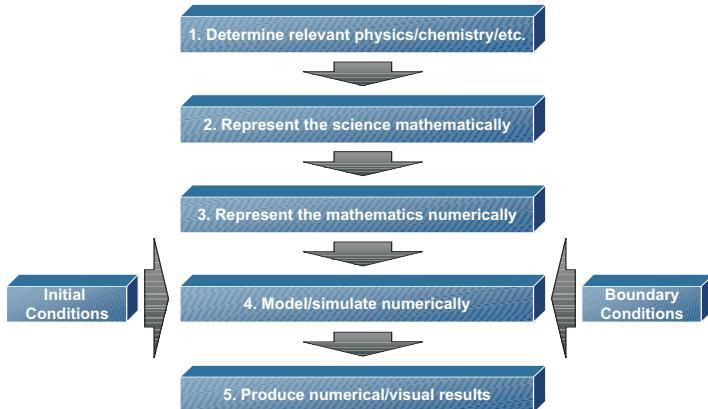
Figure 1. Five steps to scientific insight

subject to *boundary conditions*; it is not uncommon for problems to specify both kinds of constraints. The numerical model or simulation subject to various constraints can be regarded as a scientific application. Thus the solution of a scientific problem results in numerical output that may or may not be represented graphically (Figure 1, Step 5). One of four primary types (see "Applications and Architectures", below), this application is further regarded as scientific workload that needs to be managed as the calculations are carried out. Because the practices of science and engineering are actually undertaken as a process of discovery, Figure 1 should be regarded as a simplifying overview that does not depict the recursive nature of investigation.

It may appear that numerical methods are the cure-all for any scientific problem that cannot be solved by mathematical methods alone. Unfortunately, that is not the case. On their own, many of the equations of classical physics and chemistry push even the most powerful compute architectures to the limits of their capability. Irrespective of numerical methods and/or compute capability, these Grand Challenge Equations afford solutions based on simplifying assumptions, plus restrictions in space and/or time. Because these particularly thorny equations are critical in science and engineering, there is an ongoing demand to strive for progress. Examples of Grand Challenge problems are provided elsewhere ([2]).

Applications and Architectures

Science dictates mathematics and mathematics dictates numerics[4] (Figure 1). Thus a numerics bias exists in all applications of scientific origin. This predisposition motivates four types of applications (Figure 2) revealed by exploring process granularity. Granularity refers to the size of a computation that can be performed between communication or synchronization points ([3]). Thus, any point on the vertical axis of Figure 2 identifies a specific ratio of computation (increasing from bottom to top) to communication (increasing from top to bottom).[5] Task parallelism, increasing from left-to-right on the horizontal axis, refers to the degree of parallelism present in the application; "fine" through "coarse" are used as qualitative metrics, as shown.

Most scientific problems are implemented initially as *serial* applications (Figure 2, Quadrant II).[6] These problems require that each step of the scientific calculation be performed in sequence. Serial applications can be executed on compute architectures ranging from isolated desktops, servers, or supercomputers to compute farms. Compute farms are loosely coupled compute architectures in which system software is used to virtualize compute servers[7] into a single system environment (SSE).

Various factors – time-to-results, overall efficiency, etc. – combine to demand performance improvements beyond what can be achieved by "legacy" serial applications alone. For those applications whose focus is on data processing, it is natural to seek and exploit any parallelism in the data itself. Such *data parallel* applications (Figure 2, Quadrant I) are termed *embarrassingly parallel* since the resulting applications

4. "Numerics" is used here as a shorthand for numerical methods.

5. If the interest is computing, then communication is viewed as the "overhead' required to achieve the computation. Similarly, computation might be regarded as the overhead required to facilitate certain communications.

6. The mathematical convention of numbering quadrants counter-clockwise from the upper-right-hand corner is used here.

7. Although high-density, rack-mounted single/dual processor servers have been used in compute farms, there is an intensifying trend toward the use of higher density blade servers in these configurations.
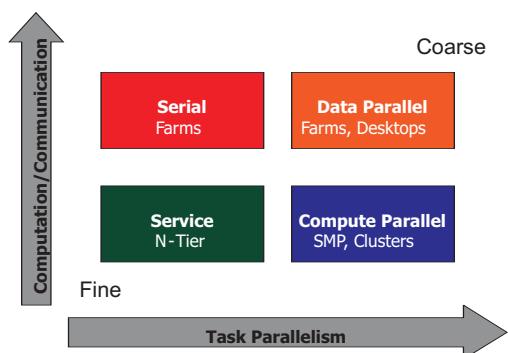

Figure 2. Applications and architectures

are able to exploit the inherent coarse granularity. The parallelism in data is leveraged by (Figure 3):

- Subdividing input data into multiple segments;
- Processing each data segment independently via the same executable; and
- Reassembling the individual results to produce the output data.

This data-driven approach accounts for one of four classes of parametric processing (Figure 4). Although data-parallel applications are a solid match for isolated systems and compute farms, there is an increasing trend to harvest compute cycles from otherwise idle desktops. Regarded as opportunistic compute resources, desktops "pull" processing tasks from coordinating servers that they will execute as a background process – especially in place of a traditional screensaver. Desktops as compute elements gained initial popularity through the peer-to-peer (P2P) movement and more recently in the context of grid computing. Ray-tracing applications provide a classic demonstration of this processing architecture. Despite the added challenge of managing data, successful implementations of embarrassingly parallel applications exist in many industries – genome sequencing in the life sciences, Monte Carlo simulations in high energy physics, reservoir modeling in petroleum exploration, risk analysis in financial services, and so on. This approach is so appealing and powerful that it's often perceived to be of general utility. Unfortunately, this simply isn't the case – and this is especially true for the Grand Challenge Equations identified previously.

Input   Output

Engine

Figure 3. Data-driven parametric processing

If it exists at all, parallelism in the Grand Challenge Equations can be exploited at the source-code level – e.g., by taking advantage of loop constructs in which each calculation is independent of others in the same loop. Parallelism in data is absent or of minor consequence. This code-level parallelism lends itself to *compute parallel* (Figure 2, Quadrant IV) applications. Compute parallel applications are further segmented on the basis of memory access – i.e., shared versus distributed memory. With minimal language extensions, and explicit code-level directives, OpenMP ([5]) and, more recently, Unified Parallel C (UPC, [6]) offer up parallel computing with shared-memory programming semantics. Symmetric multiprocessing (SMP) systems allow for shared-memory programming semantics via threads[8] through uniform (UMA) and nonuniform (NUMA) memory access architectures.

Parallel Virtual Machine (PVM, [7]) has given way to the Message Passing Interface (MPI, [8]) as the standard for parallel computing with distributed-memory programming semantics. Likened to the "assembly language" for parallel programming ([9]), MPI requires a significant investment at the source-code level.[9] In contrast to the use of threads in the shared-memory context, distributed processes are employed in the MPI case to achieve parallelism. MPI applications are typically implemented for tightly coupled compute clusters (see HPC Application Development Environment, below, for additional details). Although other factors (e.g., architecture access) need to be considered, numerics do influence the choice of parallel computing via shared versus distributed memory.[10] Both shared and distributed-memory parallel computing methodologies have been applied to scientific and engineering problems in a variety of industries – e.g., computational and combinatorial chemistry in the life sciences, computational fluid dynamics, crash simulations and structural analyses in industrial manufacturing, Grand Challenge problems in government organizations and educa-
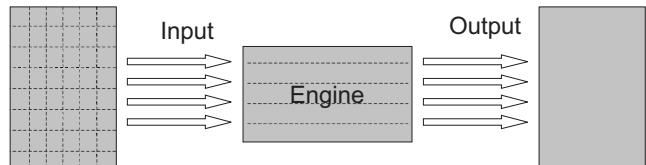
8. In addition to the fork-and-exec creation of child processes, a parent process may also involve threads. Distinguishable by the operating system, threads can share or have their own memory allocations with respect to their parent process.

9. Recent advances allow serial applications to be automatically enabled for MPI ([10]). By identifying code regions suitable for parallelization (e.g., repetitive calculations) via a templating mechanism, code-level modifications are applied. This approach is being applied extensively in financial services, where numerical models change frequently.

10. Hybrid OpenMP-MPI applications allow scientists and engineers to simultaneously use threads and distributed processes when using a combination of SMP and clustered architectures.

tional institutions, and scenario modeling in financial services. MPI compute parallel applications, traditional HPC, will be the focus of our attention here.

Because MPI provides such a rich framework for computing in general, there are examples of MPI applications that communicate extensively while carrying out minimal processing (Figure 2, Quadrant III) – e.g., remote-to-direct-memory applications (RDMA), or certain classes of search algorithms. In addition, *service* applications whose focus is networking itself or Web services ([11]) themselves would also fall into this area. As before, MPI applications would require tightly coupled architectures; whereas networking applications can be applied in a variety of contexts, loosely coupled architectures can be used in the instantiation of Web services.

HPC Application Development Environment

Together with the "commoditization" of low-processor-count, high-density servers and the emergence of low-latency, high-bandwidth interconnect technologies, MPI has played a key role in the widespread adoption of tightly coupled compute clusters for distributed memory-parallel computing ([12]):

> MPI is available everywhere and widely used in environments ranging from small workstation networks to the very largest computers in the world, with thousands of processors. Every parallel computer vendor offers an MPI implementation, and multiple implementations are freely available as well, running on a wide variety of architectures. Applications large and small have been ported to MPI or written as MPI programs from the beginning, and MPI is taught in parallel programming courses worldwide.
>
> Applied through source-code-level modifications, MPI-specific directives are referenced against an MPI library at application link time. MPI libraries are architecture specific and may come from a variety of sources – e.g., a system vendor, an interconnect vendor, or via an open source contribution. In each case, the relevant library implements the MPI specification11 to some degree of compliance. This MPI library, in combination with the tools and utilities that support developers, collectively forms the application development environment for a particular platform (Figure 4).

The situation described above might lead one to conclude that all of the requisites are present to smoothly enable MPI adoption. In practice, however, MPI has the following challenges:

- Resynchronization and reconnection were not even factored in at the specification level ([9]). There are no MPI implementations that account for this shortcoming. This is in striking contrast to PVM, whose implementation allows for this.
- Fault tolerance was not factored in, even at the specification level ([9]); again, there are no MPI implementations that account for this shortcoming. This can mean, for example, that an application can lose some of its processes, run to completion, and yield results of dubious validity.
- Hosts and numbers of processors need to be specified as static quantities, irrespective of actual usage conditions.
- Single point of control is absent. Although some recent MPI implementations offer a process daemon to launch MPI applications, there is little in the way of real application control.
- Multiple versions of MPI may exist on the same architecture. Applications need to carefully identify the relevant MPI libraries. The situation is more complex for MPI applications that span more than one execution architecture.

11. Most MPI libraries fully implement version 1.x of the MPI specification, while many libraries are today supporting some subset of the version 2.x specification.

The upshot is clear: MPI places the responsibility for these shortcomings on the application developer and user. Because MPI applications are a challenge to control and audit, a better production HPC solution is required.

Production HPC Reinvented

There is a gap between the potential for distributed-memory parallel computing via MPI and what is actually achievable in practice. The use of system software allows this gap to be closed and the promise of MPI to be fully realized. In the process, the notion of production HPC is redefined. To start, consider a modified version of Figure 4 in which the newly added workload-management system software is shown in black on white (Figure 5).

Figure 5 introduces the following three components to the MPI application development environment:

- **Core workload management services.** This system software component allows a heterogeneous collection of compute servers, each running its own instance of an operating system, to be virtualized into a compute cluster.[12] Sensory agents are used to maintain static and dynamic information in real time across the entire compute cluster. Primitives for process creation and process control across a network are also provided.
- **Parallel application management.** Challenges specific to the management of MPI parallel applications include the need to:
  - Maintain the communication connection map;
  - Monitor and forward control signals;
  - Receive requests to add, delete, start, and connect tasks;
  - Monitor resource usage while the user application is running;
  - Enforce task-level resource limits;
  - Collect resource usage information and exit status upon termination; and
  - Handle standard I/O.
- **Parallel scheduling services.** Workload management solutions typically employ a policy center to manage all resources – e.g., jobs, hosts, interconnects, users, and queues. Through the use of a scheduler, and subject to predefined policies, resource demands are mapped against the supply of resources in order to facilitate specific activities. Scheduling policies of particular relevance in parallel computing include advance reservation, backfill, preemption, and processor and/or memory reservation.

The combined effects of these three layers of a workload-management infrastructure allow the shortcomings of MPI to be directly addressed:

- Absence of resynchronization and reconnection: Although the workload-management infrastructure (Figure 5) cannot enable resynchronization or reconnection, by introducing control across all of the processes involved in an MPI application, there is greatly improved visibility into synchronization and/or connection issues.
- Absence of fault tolerance: At the very least, the introduction of a workload-management infrastructure provides visibility into exceptional situations by trapping and propagating signals that may be issued while workload is executing. These signals can be acted upon to automatically re-queue workload that has thrown an undesirable exception. Even better, when integrated with the checkpoint/restart infrastructure of a workload manager, interrupted workload can continue to execute from the last successful checkpoint, often without user intervention.
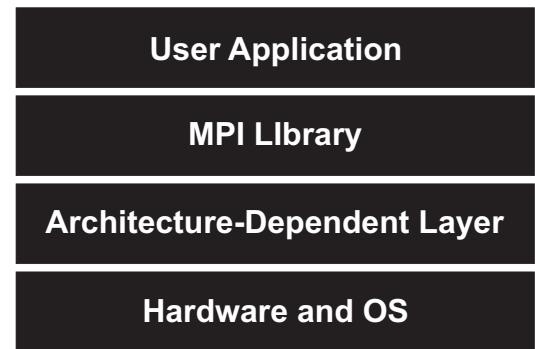
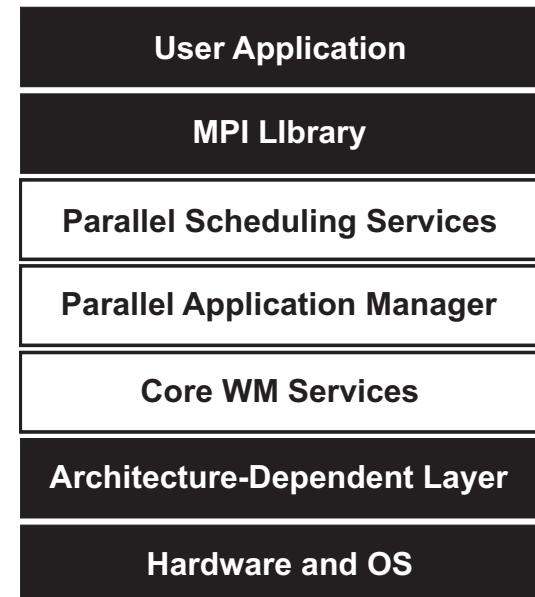Figure 4. MPI application development environment

Figure 5. Enhanced application development environment via workload-management system software

12. This layered-services approach has been contrasted with Beowulf clustering (via a distributed process space) elsewhere ([13]).

13. Experience dictates that every parallel application show acceptable performance characteristics over a range of processors. A typical criterion is that the performance remain close to linear as the number of processors increases. This is referred to as "linear speedup." Effective workload-management systems allow this processor count to be specified as a range at workload submission time. This serves to automate the load balancing situation and to enhance overall effectiveness of the scheduling services.

14. The need to bind processes to processors serves as one example of an execution environment requirement. In such cases, the workload-management infrastructure works in tandem with the operating system, interconnect manager, etc., to address the requirement.

15. Historically, Alpha-based processors were used because of their excellent floating-point-performance characteristics. With the advent of first-generation Itanium processor family CPUs, it is expected that 64-bit Intel Architecture (IA-64) will eventually dominate in this space. In the interim, fourth-generation IA-32 Pentium processor family CPUs hold the price/performance niche.
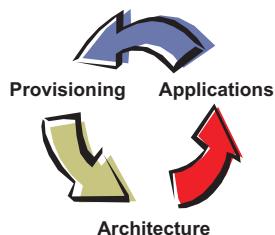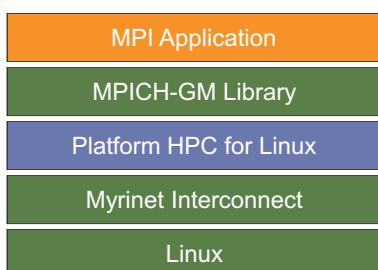
Figure 6. Production HPC reinvented

| MPI Application |
| MPICH-GM Library |
| Platform HPC for Linux |
| Myrinet Interconnect |
| Linux |

Figure 7. Production HPC for Linux

- Absence of load balancing: The need to explicitly identify hosts and numbers of processors can be regarded as an absence of load balancing – i.e., the specification of resources that ignores current resource-usage conditions. Because workload-management infrastructures maintain dynamic load state across the entire compute infrastructure in real time, this shortcoming is completely eliminated.
- Absence of single point of control: A parallel application manager provides a single point of control. This ensures that all the distributed processes that collectively make up the MPI application are managed and accounted for. Again a key shortcoming becomes a core competence via workload-management system software.
- Multiple versions of MPI: Through configuration information, MPI applications are able to specify the relevant MPI library. A very flexible architecture also allows the workload-management system software to work in concert with the existing parallel application development and runtime environment.

Together with the existing parallel application development and runtime environment, workload-management system software allows the inherent shortcomings of MPI to be effectively eliminated. The cumulative effect is to practically reinvent HPC via MPI. This threefold reinvention is captured in Figure 6.

The process starts with MPI applications that are developed in-house or acquired from commercial providers.

On job submission, these applications are accompanied by a description of their runtime resource requirements – e.g., a processor-count range,[13] data-management directives (e.g., a file transfer), plus other environmental requirements.[14] The scheduling agent takes into account the pre-specified resource requirements, the unique characteristics of the compute architectures that it has available, and the policies that reflect organizational objectives. Because the scheduler creates a runtime environment for the workload, its task is one of dynamic provisioning. On dispatch, the scheduler has optimally matched the workload to an appropriate runtime architecture subject to established policies. The application executes in a fully managed environment. A comprehensive audit trail ensures that all activities are accounted for. In this way, there is a closed loop for production HPC, one that enhances the developer and user experience rather than encumbering it. A specific solution example is provided in the following section.

Production HPC for Linux

It has been suggested that production HPC can be reinvented through the use of an integrated development and runtime environment in which workload-management system software plays a key role. A complete example is considered here to further illustrate this reinvention. Consider the integrated production HPC solution stack shown in Figure 7. Although this example is based on the Linux operating environment, similar stacks for other operating environments can be crafted.

At the base of the production HPC solution stack for Linux are low-processor-count servers,[15] each running its own instance of the GNU/Linux operating system. Ethernet-based network interface cards (NICs) allow for standard TCP/IP-based services between these systems. Because TCP/IP over Ethernet necessitates assumptions regarding shared access and the occurrence of collisions, the result is a communications protocol that is latency heavy and therefore inefficient for message-passing in support of parallel computing. Hence, each system implements Myricon's GM message-passing

protocol across low-latency, high-bandwidth, multi-port Myricom Myrinet switches ([14]) used solely to support parallel computing via MPI and the GM driver. By identifying each available Myrinet port as a resource to the core workload-management services provided by Platform HPC for Linux ([15]), the provisioning component of this infrastructure is aware of the static and dynamic attributes of the architecture it can apply parallel scheduling policies against. Platform HPC for Linux also provides the control and audit primitives that allow parallel applications to be completely managed. Users' MPI applications need to be compiled and linked against the application development environment provided by Myricom. This ensures that the appropriate GM-protocol modifications to the widely adopted open source MPICH implementation of MPI are used. Through configuration information, the workload-management system software based on Platform HPC for Linux is made aware of the enhanced MPI runtime environment.

Portability was identified as a design goal for MPI. This objective has been carried through in MPI implementations such as MPICH. Despite this fact, heterogeneous parallel applications based on MPI must not only use the same implementation of MPI (e.g., MPICH) but also the same protocol implementation (e.g., GM). In other words, MPI is a multi-protocol API (Figure 8) in which each protocol implements its own message formats, exchange sequences, and so on.

Because they can be regarded as a cluster of clusters, it follows that computational grids might provide a suitable runtime environment for MPI applications. The fact that grids are concerned with resource aggregation across geographic (and other) domains further enhances the appeal. Fortunately, Platform HPC for Linux is consistent with Platform MultiCluster – system software that allows independent clusters each based on Platform LSF to be virtualized into an enterprise grid. This combination introduces the possibility for exciting new modes of infrastructural provisioning through various grid-centric scheduling policies – e.g., Grid Advance Reservation, Grid Fairshare, and Grid Resource Leasing.

Of these grid-centric policies, Grid Resource Leasing (GRL) is particularly novel and powerful. GRL allows sites to make available fractions of their resources for use by other sites participating in their enterprise grid. These collective resources can be occupied by MPI applications through requirement specifications. In this fashion, users can *co-schedule* MPI applications to span more than one geographic location. Even more impressive is the fact that this co-scheduling is automatically enabled at runtime, that is, existing MPI applications do not need to be re-linked with special libraries. By jointly leveraging the parallel application management capability already present in Platform HPC for Linux, in concert with this grid-level scheduling policy of Platform MultiCluster, MPI application users take advantage of their entire enterprise grid in a transparent and effective fashion. Platform MultiCluster and its grid-level scheduling policies are considered in detail elsewhere ([17]).

Figure 8. The multi-protocol nature of MPI (after [16])

Summary

Production High Performance Computing (HPC) incorporates a variety of applications and compute architectures. Widespread use of the Message Passing Interface (MPI) is better enabled through the use of workload-management system software. This software allows MPI applications and the compute architectures on which they execute to be provisioned on demand. This critical link significantly reduces complex-
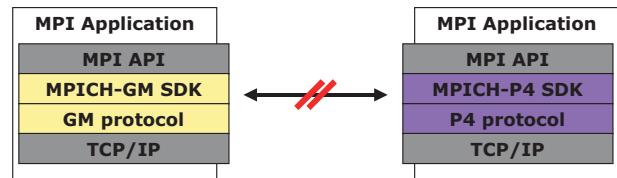
ity for the scientist or engineer, thus reducing time to results and ensuring overall organizational efficiency. A tightly coupled cluster based on the Linux operating environment was shown to be a particularly attractive and viable compute architecture. The incorporation of this environment into a compute grid was also shown to be a natural progression. Overall, organizations are able to better empower the pursuit of science and engineering during application development, deployment, and use.

ACKNOWLEDGMENTS

REFERENCES

[1] Maple, *http://www.maplesoft.com*.

[2] The Grand Challenge Equations, *http://www.sdsc.edu/Publications/GCequations*.

[3] B. Wilkinson & M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers* (Upper Saddle River, NJ: Prentice-Hall, 1999).

[4] I. Lumb, B. Bryce, B. McMillan, and K. Priddy, "From the Desktop to the Grid: Parametric Processing in High-Performance Computing," Sun User Performance Group, Amsterdam, Netherlands, October 2001.

[5] OpenMP, *http://www.openmp.org*.

[6] Unified Parallel C, *http://upc.gwu.edu*.

[7] Parallel Virtual Machine, *http://www.csm.ornl.gov/pvm/pvm_home.html*.

[8] Message Passing Interface, *http://www.mpi-forum.org*.

[9] K. Dowd and C.R. Severance, *High Performance Computing*, 2d ed. (Sebastopol, CA: O'Reilly & Associates, 1998).

[10] Powerllel, *http://www.powerllel.com/powerllel/content/index.shtml*.

[11] Web services, *http://www.w3.org/2002/ws/*.

[12] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, 2d ed. (Cambridge, MA: MIT Press, 1999).

[13] I. Lumb, "Linux Clustering for High-Performance Computing," *;login:*, vol. 26, no. 5, August 2001, pp. 79–84.

[14] Myricom Myrinet, *http://www.myricom.com/myrinet*.

[15] Platform Computing Corporation, "Using LSF with MPICH-GM," technical documentation, April 2002.

[16] MPICH-G2, *http://www.hpclab.niu.edu/mpi*.

[17] Platform Computing Corporation, *Platform MultiCluster Guide*, technical documentation, June 2002.