Privacy & Legal Notice

Background

The MCR Linux cluster was built by Linux NetworX in Utah. See the LNXI Build and LLNL Integration information for more details and photos.

MCR Statement of Work
Contents

1 Background

1.1 Multiprogrammatic and Institutional Computing (M&IC)

Lawrence Livermore National Laboratory (LLNL) has identified high-performance computing as a critical competency necessary to meet the goals of LLNL's scientific and engineering programs. Leadership in scientific computing demands the availability of a stable, powerful, well-balanced computational infrastructure, and it requires research directed at advanced architectures, enabling numerical methods and computer science.

M&IC was created to encourage all programs to benefit from the huge investment being made by the Advanced Simulationand Computing Program (ASCI) at LLNL and to provide a mechanism to facilitate multiprogrammatic leveraging of resources and access to high-performance equipment by researchers.

The Livermore Computing (LC) Center, a part of the Computations Directorate Integrated Computing and Communications (ICC) Department can be viewed as composed of two facilities, one open and one secure. This acquisition is focused on the M&IC resources in the Open Computing Facility (OCF).

For the M&IC program, recent efforts and expenditures have focused on enhancing capacity and stabilizing the TeraCluster 2000 (TC2K) resource. Capacity is a measure of the ability to process a varied workload from many scientists simultaneously. Capability represents the ability to deliver a very large system to run scientific calculations at large scale.

In this procurement action, we intend to significantly increase the capability of the M&IC resource to address multiple teraFLOP/s problems as well as increasing the capacity to do many 100 gigaFLOP/s calculations.

1.2 Partnership with the Stockpile Stewardship Program (SSP)

The M&IC platforms form part of the unclassified computing environment at LLNL. This environment is called the Open Computing Facility (OCF). Some of the platforms in the OCF represent primarily Stockpile Stewardship Program investments (for instance, the unclassified ASCI systems). Others, such as the shared memory multiprocessors (SMP) clusters, represent fairly evenly divided investments between multiple programs (including the SSP) and the institution.

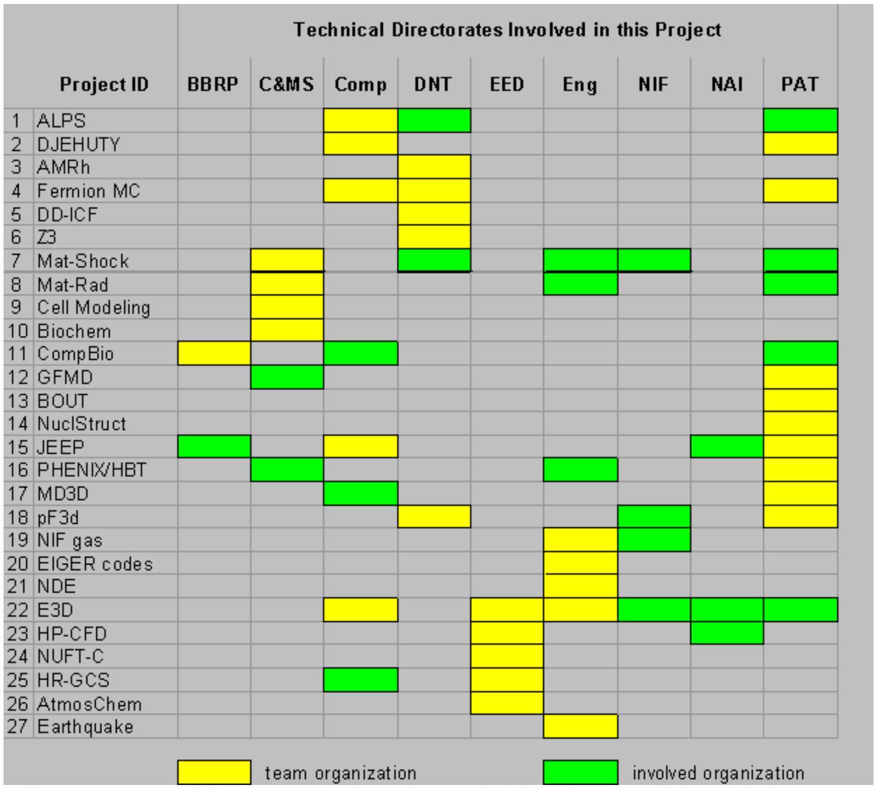
The support infrastructure (everything but the compute platforms) is covered both by M&IC and by the NNSA SSP, and the partners share the resources. For instance, the IBM HPSS storage environment represents an ~80/20 split, with the SSP making the heavier investment. On the other hand, the NFS home space environment was procured by M&IC. The visualization environment is more heavily funded by the SSP, but the System Area Network is supported by the M&IC. The LC balancesthese investments according to the utilization of the broad support infrastructure by the partners. The end result is a more powerful OCF than any program (or the Institution) could afford alone.

Back to Contents

1.3 M&IC Applications Overview

LLNL is in the forefront of the evolution toward effective and practical computational science in all its forms. To continue this role, we must continue to provide the computational tools that a wide user community needs to advance their scientific research.

LC recently conducted a detailed analysis of M&IC computing requirements, finding that our users support projects that address a wide range of important scientific issues and pressing national and international concerns. The total computing needs of these projects far exceed the current M&IC capability and capacity. The progress on many of these projects is being paced by the available computing resources, and access to additional computing cycles will result in faster progress. As shown in Table 1.3-1, these projects span the technical directorates and support the major programs of the Laboratory. The technical directorates noted in this table are Biology & Biotechnology Research Program (BBRP), Chemistry & Materials Science (C&MS), Computation (Comp), Defense & Nuclear Technologies (DNT), Energy & Environment Directorate (EED), Engineering (Eng),National Ignition Facility Programs (NIF), Nonproliferation, Arms Control & International Security (NAI), and Physics and Advanced Technologies (PAT).



	Technical Directorates Involved in this Project								
Project ID	BBRP	C&MS	Comp	DNT	EED	Eng	NIF	NAI	PAT
1 ALPS			🟨	🟩					🟩
2 DJEHUTY			🟨						🟨
3 AMRh				🟨					
4 Fermion MC			🟨	🟨					🟨
5 DD-ICF				🟨					
6 Z3				🟨					
7 Mat-Shock		🟨		🟩		🟩	🟩		🟩
8 Mat-Rad		🟨				🟩			🟩
9 Cell Modeling		🟨							
10 Biochem		🟨							
11 CompBio	🟨		🟩						🟩
12 GFMD		🟩							🟨
13 BOUT									🟨
14 NuclStruct									🟨
15 JEEP	🟩		🟨					🟩	🟨
16 PHENIX/HBT		🟩				🟩			🟨
17 MD3D			🟩						🟨
18 pF3d				🟨			🟩		🟨
19 NIF gas						🟨	🟩		
20 EIGER codes						🟨			
21 NDE						🟨			
22 E3D			🟨		🟨	🟨	🟩	🟩	🟩
23 HP-CFD					🟨			🟩	
24 NUFT-C					🟨				
25 HR-GCS			🟩		🟨				
26 AtmosChem					🟨				
27 Earthquake						🟨			

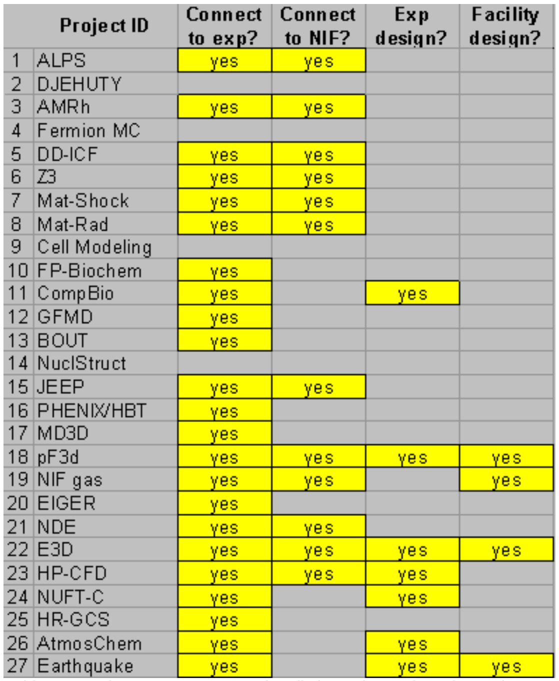
🟨 team organization 🟩 involved organization

Table 1.3-1. A recent analysis of M&IC computing requirements included projects from all nine technical directorates at the Laboratory.

Back to Contents

Computation is now a mainstream method in theoretical science at LLNL—essential when highly simplified but analyticaly intractable models are explored or complex multiphysics phenomena must be understood quantitatively. As we understand more and more truly basic science, the Laboratory is looking to computation to make the vital numerical connections among disparate models that constitute the foundation of both pure and applied science.

An enormous role is also being played in experimental science by M&IC projects. Many of these projects have achievedsuch sophistication that direct comparisons between full-scale experiment and simulations based on ab initio models are now possible. But it is often the case that these direct comparisons are limited by the available computing resources. As seen in Table 1.3-2, the vast majority of M&IC projects are closely tied to experiment, and nearly half provide some supportto NIF, a major program at the Laboratory. Many projects have an even stronger experimental tie, being used to design experiments or facilities.



Project ID	Connect to exp?	Connect to NIF?	Exp design?	Facility design?
1 ALPS	yes	yes		
2 DJEHUTY				
3 AMRh	yes	yes		
4 Fermion MC				
5 DD-ICF	yes	yes		
6 Z3	yes	yes		
7 Mat-Shock	yes	yes		
8 Mat-Rad	yes	yes		
9 Cell Modeling				
10 FP-Biochem	yes			
11 CompBio	yes		yes	
12 GFMD	yes			
13 BOUT	yes			
14 NuclStruct				
15 JEEP	yes	yes		
16 PHENIX/HBT	yes			
17 MD3D	yes			
18 pF3d	yes	yes	yes	yes
19 NIF gas	yes	yes		yes
20 EIGER	yes			
21 NDE	yes	yes		
22 E3D	yes	yes	yes	yes
23 HP-CFD	yes	yes	yes	
24 NUFT-C	yes		yes	
25 HR-GCS	yes			
26 AtmosChem	yes		yes	
27 Earthquake	yes		yes	yes

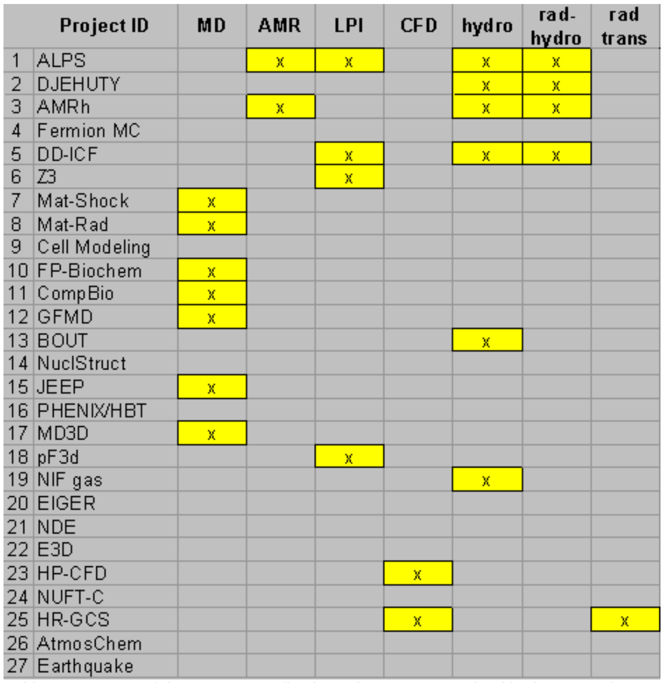
Table 1.3-2. These M&IC computationally intensive projects have close connection with experiment.

Back to Contents

M&IC applications codes span the physical, environmental, and biological sciences. Typically these codes are complex,time-dependent simulations of multiple physical processes, where the processes are often tightly coupled and will require physics models linking microscale phenomena to macroscopic response. Generally, these simulations are multidimensional, with the trend toward full three-dimensional treatment of physical space. Table 1.3-3 illustrates that many of these applications fall into the following broad classifications:

- molecular dynamics simulations (MD)
- adaptive-mesh-refinement codes (AMR)

- laser-plasma-interaction simulations (LPI)
- computational-fluid-dynamics simulations (CFD)
- hydrodynamic simulations (hydro)
- coupled radiation-transport and hydrodynamic simulations (rad-hydro)
- radiation-transport simulations (rad trans)



	Project ID	MD	AMR	LPI	CFD	hydro	rad-hydro	rad trans
1	ALPS		x	x		x	x	
2	DJEHUTY					x	x	
3	AMRh		x			x	x	
4	Fermion MC							
5	DD-ICF			x		x	x	
6	Z3			x				
7	Mat-Shock	x						
8	Mat-Rad	x						
9	Cell Modeling							
10	FP-Biochem	x						
11	CompBio	x						
12	GFMD	x						
13	BOUT					x		
14	NuclStruct							
15	JEEP	x						
16	PHENIX/HBT							
17	MD3D	x						
18	pF3d			x				
19	NIF gas					x		
20	EIGER							
21	NDE							
22	E3D							
23	HP-CFD				x			
24	NUFT-C							
25	HR-GCS				x			x
26	AtmosChem							
27	Earthquake							

Table 1.3-3. Many of these M&IC applications share common algorithmic approaches, while differing markedly in the details of the implementations.

Back to Contents

Demanding national security issues are driving intense development of M&IC applications codes. Advanced numerical algorithms require innovative software techniques to achieve the necessary delivered performance on advanced computing platforms. Current application characteristics and expected trends are described below, although M&IC applications codes and numerical algorithms will continue to evolve rapidly.

The following are some of the major M&IC code characteristics essential to an understanding of the vision of an ideal computing environment.

The codes often model multiple types of physical/chemical/biological/environmental processes, generally in a single (usually monolithic) application, in a time-evolving manner with direct coupling between all simulated processes. They do so using a variety of computational methods, often through a separation or "split" of the various processes and coupling terms. This process often involves solving first one type of model, then the next, then another, and then repeating this sequence for every time step. Some algorithms are categorized as explicit in time while others are fully implicit or semi-implicit and typically involve iterative solvers of some form. Some special wave front "sweeps" are employed for specific direct-solve algorithms. Numerous research efforts are actively exploring novel and alternative methods and algorithms for possible application to problems of interest.

The calculations are of various sizes, with some treating millions of particles or spatial zones (cells), with an expected requirement for many applications to get to the point of using upwards of a billion or more particles/cells. The equations are typically solved by spatial discretization. Discretization over energy and/or angle, in addition, can increase the data space size by 10 to 1000 times. In the final analysis, thousands of variables will be associated with each zone. Monte Carlo algorithms will treat millions to billions of particles distributed throughout the problem domain. The parallelization strategy for many codes is based upon decomposition into spatial domains. For some applications, codes will use decomposition over angular or energy domains, as well.

Currently, almost all codes use the standard message passing interface (MPI) for parallel communication, even between processes running on the same SMP. In addition, some applications utilize OpenMP for SMP parallelism. The efficiency of OpenMP SMP parallelism depends highly on the underlying compiler implementation (i.e., the algorithms are highly sensitive to OpenMP overheads). Also, it is possible in the future that different physics models within the same application might use different communication models. For example, an MPI-only main program may call a module that uses the same number of MPI processes, but also uses threads (either explicitly or through OpenMP) In the ideal system, these models should interoperate as seamlessly as possible. Mixing such models mandates thread-safe MPI libraries. Alternative strategies may involve calling MPI from multiple threads with the expectation of increased parallelism in the communications; such use implies multithreaded MPI implementations as well.

Current codes are based on a single program multiple data (SPMD) approach to parallel computing. However, director/worker constructs are often used. Typically, data are decomposed and distributed across the system and the same execution image is started on all MPI processes and/or threads. Exchanges of remote data occur for the most part at regular points in the execution, and all processes/threads participate (or just pretend to) in each such exchange. Data are actually exchanged with individual MPI send-receive requests, but the exchange as a whole can be thought of as a "some-to-some" operation with the actual data transfer needs determined from the decomposition. Weak synchronization naturally occurs in this case because of these exchanges, while stronger synchronization occurs because of global operations, such as reductions and broadcasts (e.g., MPI_ALLREDUCE), which are critical parts of iterative methods. It is quite possible that future applications will use functional parallelism, but mostly in conjunction with the SPMD model. Parallel input-output (I/O) and visualization are areas that may use such an approach with functional parallelism at a high level to separate them from the physics simulation, yet maintain the SPMD parallelism within each subset. There is some interest in having visualization tools dynamically attach to running codes and then detach for interactive interrogaton of simulation progress. Such mixed approaches are also under consideration for some physics models.

Back to Contents

Many applications use unstructured spatial meshes. Even codes with regular structured meshes may have unstructured data if they use cell-by-cell continuous AMR. In an unstructured mesh, the neighbor of zone (i) is not zone (i+1), and one must use indirection or data pointers to define connectivity. Indirection has been implemented in several codes through libraries of gather-scatter functions that handle both on-processor as well as remote communication to access that neighbor information. The communication support is currently built on top of MPI and/or shared memory to get that neighbor information. These scatter-gather libraries are two-phased for good efficiency. In phase one, the gather-scatter pattern is presented and all local memory and remote memory and communications structures are initialized. Then in phase two, the actual requests for data are made, usually many, many times. Thus, the patterns are extensively reused over and over again. Also, several patterns will coexist simultaneously during a time step for various data. Techniques like AMR and reconnecting meshes can lead to pattern changes at fixed points in time, possibly every cycle or maybe only after several cycles.

Memory for arrays and/or data structures is typically allocated dynamically, avoiding the need to recompile with changed parameters for each simulation size. This allocation requires compilers, debuggers, and other tools that recognize and support such features as dynamic arrays and data structures, as well as memory allocation intrinsics and pointers in the various languages.

Many of the physics modules will have low compute-communications ratios. It is not always possible to hide latency through non-blocking asynchronous communication, as the data are usually needed to proceed with the calculation. Thus, a low-latency communications system is crucial.

Many of the physics models are memory intensive, and will perform only about 1 FLOP per load from memory. Thus, performance of the memory sub-system is crucial, as are compilers that optimize in terms of cache blocking, loop unrolling-rolling, loop nest analysis, etc. Many codes have loops over all points in an entire spatial decomposition domain. This coding style is preferred by many for ease of implementation and readability of the physics and algorithms. Although recognized as problematic, effective automatic optimization is preferred, where possible.

The multiple physics models embedded in a large application may have dramatically varying communication characteristics, i.e., one model may be bandwidth-sensitive, while another may be latency-sensitive. Even the communications characteristics of a single physics model may vary greatly during the course of a calculation as the spatial mesh evolves or different physical regimes are reached and the modeling requirements change. In the ideal system, the communications system should handle this disparity without requiring user tuning or intervention.

Although static domain decomposition is used for load balancing as much as possible, there are also definite needs for dynamic load balancing, in which the work is moved from one processor to another. One obvious example is for codes using AMR methods, where additional cells may be added or removed during the execution wherever necessary in the mesh. It is also expected that different physical processes will be regionally constrained and, as such, will lead to load imbalances that can change with time as different processes become "active" or more difficult to model. Any such dynamic load balancing is expected to be accomplished through associated data migration explicitly done by the application itself. This re-balancing might occur inside a time step, every few time steps, or infrequently, depending on the nature of the problem being run. In the future, code execution may also spawn and/or delete processes to account for the increase and/or decrease in the total amount of work the code is doing at that time.

Back to Contents

1.4 M&IC Scientific Software Development Environment

The following are some of the major characteristics of the software development environment in an ideal scenario.

A high degree of code portability and longevity is a major objective. Many M&IC codes must execute at multiple sites. Development, testing and validation of 3D, full-physics, full system applications requires four to six years. The productive lifespan of these codes is at least ten years. Thus, these applications must span not only today's platforms but any possible future system. Codes will be developed in standards-conforming languages, so non-standard compiler features are of little interest unless they can be made transparent. The use of Cray Pointers in Fortran is an exception to our reliance on standard features. We also will not take advantage of any idiosyncratic features of optimization, unless they can be hidden from the codes (e.g., in a standard library). Non-standard "hand tuning" of codes for specific platforms is antithetical to this concept.

A high-performance, low-latency MPI environment that is robust and scalable is crucial to us. Today, applications are utilizing all the features of MPI 1.1 functionality. Many features of MPI-2 functionality are also in current use. Hence, a full, robust and efficient implementation of MPI-2 is of tremendous interest. A POSIX compliant-thread environment is also crucial and a Fortran95-threads interface is also important. All libraries need to be thread-safe. MPI should be multithreaded as well as thread-safe. We should not have to tune the MPI-runtime environment for different codes and different problem sizes. In our estimation, bandwidth of 0.2 bytes/second/peak OP/second/SMP and an end-to-end MPI ping-pong latency of less than 10 microseconds or better will provide the desired performance. Because we are talking about systems with tens of thousands of processors, it is vitally important that the MPI implementation scale to the full size of the system. This scaling is both in terms of efficiency (particularly of the MPI_ALLREDUCE functionality) as well as the efficient use of buffer memory. M&IC applications are carefully programmed so that MPI RECEIVE operations are posted before the corresponding SEND operation. This allows for minimal (and hence scalable) MPI buffer space allocations.

M&IC applications require the ability for each MPI task to access up to 2.0 GiB of physical memory. The large memory sizes of MPI tasks requires that nodes be configured with 2–4 GiB of real memory per processor.

We will expect the compilers to do the vast majority of code optimization through simple easy-to-use compiler switches (e.g., -On). Also, we will expect the compilers to have options to range check arrays under debug mode, as well as a way to trap underflow, overflow, divide by zero, etc. Parallelization through the OpenMP specifications is of particular interest and is expected for Fortran95, C, and C++. OpenMP parallelization must function correctly in programs that also use MPI. OpenMP Version 20 support for Fortran95, Version 1.0 for C/C++ is highly favored, while automatic parallelization is of some interest, if it is efficient and does not drive compile times to unreasonable lengths. Any information the compiler can provide about the optimizations it performed is useful. Compiler parallelism has to work in conjunction with MPI. All compilers must be fully ANSI-compliant.

Back to Contents

The availability of standard, platform-independent tools is necessary for a portable and powerful development environment. Examples of these tools are GNU software (especially GNU make, but others as well), TotalView debugger (the current debugger on all M&IC platforms), dependency builders (Fortran USE & INCLUDE as well as #include), preprocessors (CPP, M4), source analyzers (lint, flint, etc.), hardware counter libraries and communications profilers (VAMPIR, etc.). Tools that work with a source code should fully support the most current language standards. A standard API to give debuggers and performance analyzers access to the state of a running code will allow us to develop our own tools or to use a variety of tools developed by others. The Distributed Process Class Library (DPCL) is an emerging public domain API that meets this need. These performance and debugging tools must not require privileged access modes, such as root user nor compromise the security of the runtime environment.

We must have parallel debuggers that allow us to debug parallel applications within an SMP or node and that permit parallel application debugging utilizing multiple nodes or SMPs. This includes MPI-only as well as mixed MPI + threads and/or OpenMP codes. Ideally, the debugger will allow effective debugging of jobs using every CPU on the system. Practical use of large fractions of the machine by an application under the control of the debugger requires that the debugger be highly integrated into the system initiated parallel checkpoint/restart and Gang scheduling mechanisms. Some specific features of interest include the following:

- breakpoints,
- fast conditional breakpoints,
- fast conditional watchpoints on memory locations,
- a save-restore state for backing up via checkpoint/restart mechanism,
- complex selections for data display (possibly even programming support with loops, conditionals, local variables, etc),
- support for array statistics (min, max, etc.),
- attaching/detaching to running jobs,
- an initialization file that knows where the sources are and what options we want, etc., and
- a command-line interface in addition to a GUI (e.g., for working over slow phone lines from home).

The capability to visually examine slices and subsets of multidimensional arrays is a feature that has proven useful. The debugger should allow complex selections for data display to be expressible with Fortran95 and C language constructs and features. It should support applications written in a mixture of the baseline languages (Fortran95, C, and C++), support Cray-style pointers in Fortran77, and be able to dive on templated functions and handle complex template evaluation capabilities in C++. It should be able to debug compiler-optimized code since problems sometimes go away at debug levels, although less symbolic and contextual information will be available to the debugger at higher levels of optimization. Our build environment involves accessing source code from-NFS mounted file systems with likely compiling and linking of the executable in alternate directories. This process may have implications, depending on how the compiler tells the debugger to find the source code. The debugger currently used in the Tri-Laboratory ASCI PSE CDE is the TotalView debugger from Etnus.

Because most M&IC codes are memory-access intensive, optimizing the spatial and temporal locality of memory accesses is crucial for all levels of the memory hierarchy. To tune memory distribution in a NUMA machine, it is necessary to be able to specify where memory is allocated. To optimally use memory and to reuse data in cache, it is also necessary to cause threads to execute on CPUs that quickly access particular NUMA regions and particular caches. Expressing such affinities should be an unprivileged operation. Threads generated by a parallelizing compiler (OpenMP or otherwise) should be aware of memory-thread affinity issues as well.

Back to Contents

Other ramifications of the large memory footprint of M&IC codes is that they require large delivered memory bandwidth as seen by the application's actual memory reference patterns. This is a requirement that stresses the memory subsystem when the applications display regular memory reference patterns and have a high degree of cache utilization and high degree of cache line utilization for application memory reference payload delivery. In addition, because many of these memory-access intensive codes have random memory access patterns (due to indirect addressing or complex C++ structure and method dereferencing brought about from implementing discretization of spatial variables on block structured or unstructured grids) and therefore access thousands to millions of standard UNIX 4 KB VM pages every time step, "large page support" in the operating system for efficient utilization of the microprocessor virtual to real memory translation functionality and caches is required for efficient use of the hardware. This is because hardware TLBs have a limited number of entries (although caching additional entries in L1 cache helps but does not solve the problem), and having, say, 256 MiB page size will significantly reduce the number of TLB entries required for large memory-access M&IC code VM to real memory translations. Because TLB misses (that are not cached in L1) are very expensive, this feature can significantly enhance M&IC application performance.

Many of our codes could benefit from a high-performance, standards-conforming, parallel I/O library such as MPI-I/O. Many M&IC applications development teams now consider the ability to do MPI-2 dynamic tasking an essential item for future M&IC code development efforts. In addition, low latency GET/PUT operations for transmission of single cache lines is viewed as essential for domain overloading on a single SMP or node. However, many implementations of the MPI-2 MPI_GET/MPI_PUT mechanisms do not have lower latency than MPI_SEND/MPI_REC but do allow for multiple outstanding MPI_GET/MPI_PUT operations to be active at a time. This approach, although appealing to MPI-2 library developers, puts the onus of latency hiding on the applications developer, who will rather think about physics issues. Future M&IC applications require a very low latency (as close to the SMP memory copy hardware latency as possible) for GET/PUT operations.

It is advantageous to have support for translating big-endian, little-endian, and Cray Research PVP data representations to the system's native data forms. Especially useful will be automatic I/O filters on a file-by-file basis that will do this at read-write time.

Effectively tuning an application's performance requires detailed information on its timing and computation activities. On an SMP or node, a timer that is consistent between threads or tasks running on different CPUs in that same SMP or node is useful. Frequent use of the timer implies high-resolution (10 microseconds or better) and low overhead. In addition, other hardware performance monitoring information such as the number of cache misses, TLB misses, floating-point operations, etc., can be very helpful. All modern microprocessors contain counters that gather this kind of information. The data in these counters can be made available separately for each thread or process through tools or programming libraries accessible to the user. For portability, our tools are targeting the PAPI library for hardware counters. To limit instrumentation overhead, a version of their tools that support multiplexing of hardware counters and sampling of instructions in the pipeline is easier to use. Note that this facility requires that the operating system context switch these counters at process or heavyweight (OS scheduled) thread level and that the POSIX or OpenMP runtime libraries context switch the counters on lightweight (library scheduled) thread level. Furthermore, these counters can be available to users that do not have privileged access, such as the root user. Per-thread OS statistics must be available to all users via a command-line utility as well as a system call. One example of such a feature is the kstat facility: a general-purpose mechanism for providing kernel statistics to users. Both hardware and counter statistics must provide virtualized information, so that users can make the correct attribution of performance data to application behaviors.

We need to have early access to new versions of system and development software, as well as other supplied software.Software releases of the various products should be synchronized with operating system releases to ensure compatibility and interoperability.

Back to Contents

1.5 M&IC Applications Execution Environment

The following are some major characteristics of the M&IC ultra-scale applications execution environment.

It is crucial to be able to run a single parallel job on the full system using all resources available for a week ormore at a time. This is called a "full-system run." Any form of interruption should be minimized. The capability for the system and application to "fail gracefully" and then recover quickly and easilyis an extremely important issue for such calculations. We also expect to be running a large number of jobs on thousands of processors each for hundreds of hours. These will require significantsystem resources, but not the entire system. The capability of the system to "fail gracefully," so that a failure in one section of the system will only affect jobs running on that specific section, is important. From the applications perspective, the probability of failure should be proportional to the fraction of the system utilized. A failed section should be repairable without bringingdown the entire system.

A single simulation may run over a period of months as separate restarted jobs in increments of days running on varying numbers of processors with different physics models activated. Output files produced by a code on one set of processors need to be efficiently accessible by another set of processors, orpossibly even by a different number of processors, to restart the simulation. Thus an efficient cluster wide file system is essential. Ideally, file input and output between runs should be insensitive to the number of processors before and after a restart. It should be possible to restart a job across a larger or smaller number of processors than originally used, with only a slight difference in performance visible.

M&IC applications write many restart and visualization dumps during the course of a run. A single restart dump will be about the same size as the job's memory image, while visualization dumps will be perhaps 1–10% of that size. Restart dumps will typically be scheduled based on wall clock periods, while visualization dumps are scheduled entirely on the basis of internal physics simulation time. We usually create visualization dumps more frequently than restart dumps. System reliabilitywill have a direct effect on the frequency of restart dumps; the less reliable the system is, the more frequently restart dumps will be made and the more sensitive we will be to I /O performance. We have observed on previous generation M&IC platforms that restart dumps comprise over 75% of the data written to disk. Most of this I/O is wasted in the sense that restart dumps are overwritten as the simulation progresses. However, this I/O must be done so that the simulation is not lost to a platform failure. This leads us to the notion that cluster wide file system (CWFS) I/O can be segregated into two portions: productive I/O and defensive I/O. Productive I/O is the writing of data that the user needs to do science (visualization dumps, traces of key physics variables over time, etc.). Defensive I/O is done to manage a large simulation run over a period of time much larger than the platform Mean Time Between Failure (MTBF). Thus, onewill like to minimize the amount of resources devoted to defensive I/O and computation lost due to platform failure. This can be accomplished by procuring resources with a high MTBF.

Operationally, applications teams push the large restart and visualization dumps (already described) off to HPSS tertiary storage within the wall clock time between making these dumps. The disk space mentioned elsewhere in this document is insufficient to handle M&IC applications long-term storage needs.HPSS is the archive storage system of M&IC and compatibility with it is needed. Thus, a highly usable mechanism is required for the parallel high-speed transport of ones to tens of TB of data from the CWFS to HPSS.

We need a resource manager–job scheduler that deals with all aspects of the system's resources, not with just the processors and the time allocations. Factors that should be considered include processors, processes, memory, interconnects, disks, and visualization engines. It will be essential for thisresource manager-scheduler to handle both batch and interactive execution of both serial and parallel programs (MPI and threaded) from a single processor to the full cluster. The manager-scheduler will provide a way to implement policies on selecting and executing various problems (problem size, problem runtime, timeslots, preemption, users' allocated share of machine, etc.). Also, a method will be provided for users to connect to executing batch jobs to query or change problem status or parameters. The tool(s) will schedule jobs to provide for process-to-processor affinity. We are currently using LLNL's Distributed Production Control System (DPCS) on the ASCI Blue-Pacific and White systems as well as existing Linux clusters and other M&IC resources.

Our codes and users will benefit from a robust, globally visible, high-performance, parallel file system. It is essential that all file systems have large file (64 b file pointer) offsets. A 32 b file pointer is clearly insufficient.

Back to Contents

1.6 M&IC Operational Environment

LC operates our production systems 24 hours per day, 7 days per week, including holidays. The prime shift is from 8 am. to 5 p.m. Pacific Time. LLNL local users access these systems via the 1 Gigabit Ethernet local-area network (LAN). MCR will operate in this environment.

The prime shift period will be devoted primarily to interactive applications development, interactive visualization,relatively short time-limit, large CPU count (e.g., over half the system CPUs), high priority production runs and extremely long-running, smaller CPU count (e.g., 64-512), lower priority productionruns. Night shifts, as well as the weekend and holiday periods, will be devoted to extremely long-running jobs. Checkpointing and restarting jobs will take place as necessary to schedule this heterogeneous mix of jobs under dynamic load and job priorities on MCR. Because the workload is so varied and the demands for CPU time oversubscribe the machine by several factors, resource scheduling is an essential production requirement. In addition to application-initiated checkpoint/restart, M&IC applications have the ability to do application based restart dumps These interim dumps, as well as visualization output, will be stored on HPSS-based archival systems or sent to the VIEWS visualization corridors via the system-area network (SAN) and external "Jumbo Frame" 1 Gigabit Ethernet interfaces. Depending upon system protocol support, IP version 4 and lightweight memory-to-memory protocol (e.g., iWARP) traffic will be carriedin this environment.

A single point of system administration will allow the configuration of the entire system from a common server. The single server will control all aspects of system administration in aggregate. Examples of system administration functions include modifying configuration files, editing mail lists, software, upgrades and patch (bug fix) installs, kernel parameter changes, file system-disk manipulation, reboots, user account activities (adding, removing, modifying), performance analysis, hardware changes,and network changes. A hardware and software configuration management system that profiles the system hardware and software configuration as a function of time and keeps trackof who makes changes is essential.

The ability to dynamically monitor system functioning in real time and allow system administrators to quickly diagnose hardware, software (e.g., job scheduler) and workload problems and take corrective action is also essential. These monitoring tools must be fast and scalable and display data in a hierarchal schema. The overhead of system monitoring and control will necessarily need to be low in order to not destroy large job scalability (performance).

LLNL's DPCS will manage the queue of pending batch jobs, deciding when to initiate pending jobs so as to achieve LLNLmanagement objectives. DPCS is a mature system that has been managing LLNL's supercomputer workloads since 1992. DPCS requires an underlying resource manager to allocate nodes inthe cluster for jobs being initiated, initiate the required tasks, and establish the switch interconnects between these tasks. We intend to utilize the Quadrics Resource Management System(RMS) in this role initially. Our intent is to replace RMS with a more scalable, Open Source resource manager presently under development at LLNL: Simple Linux Utility for Resource Management (SLURM). SLURM will support initiating and managing MPI jobs utilizing QsNet and should be ready for deployment in late 2002.

The operating environment will conform to DOE security requirements for unclassified systems. Software modificationswill be made in a timely manner to meet changes to those security requirements.

Back to Contents

1.7 Utilization of Existing Facilities

An existing facility, the main computer floor of B439, will be used for siting the MCR system. This facility has approximately 8,000–9,000 ft^2 and 1.9 MW of power for the computing system and peripherals and associated cooling available for this purpose.

Back to Contents

2 MCR Strategy and Architecture

This section describes the overall MCR hardware and software strategy and architecture and the Linux development andsupport strategy and outlines a plan for MCR build.

2.1 LC Hardware and Software Strategy and MCR Architecture

The University's scalable systems strategy (known as the Livermore Model, Figure 2.1-1) is to have a unified softwareenvironment available on all cluster systems we put into production. The main purpose of this strategy is to enable highly complex scientific simulation applications to be portable across multiple platforms at any given point in time and to provide a stable target environment over multiple generations of platforms. This strategy has been successful in providing a stable target applications environment since about 1992, when the Meiko CS-2 MPP was introduced at LLNL.

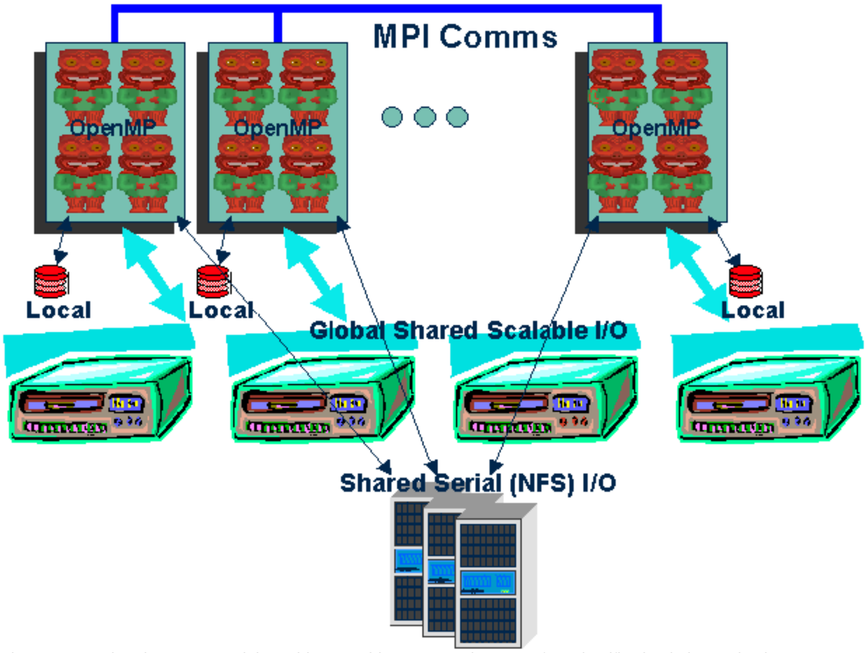


Figure 2.1-1. The Livermore Model provides a stable target environment for scientific simulation codes by abstracting the parallelism and I/O model. Parallelism is MPI tasks exchanging data over a high-speed, low-latency communication mechanism and a small number of OpenMP threads per MPI task. This model includes an OS on every node and three types of I/O. It also includes C, C++, and Fortran compilers, TotalView debugger and node hardware, and MPI/OpenMP performance analysis tools.

Back to Contents

The basic idea of the Livermore Model is to abstract the parallelism and I/O model of the computing environment. At ahigh enough level of standards-based abstraction, the computing environment evolves fairly slowly, and most machines of a given era are roughly equivalent. The parallelism abstraction is based on SMPs interconnected with a high-speed, low-latency interconnect. Each SMP has a hierarchical shared memory: processor registers; on-chip and off-chip caches; (possibly NUMA) memory. Applications must thus utilize MPI to communicate between the distributed-memory elements. In addition, each MPI task can utilize compiler generated OpenMP threads foron SMP parallelism. Each SMP or node is assumed to have a local, full functioning POSIX-compliant operating system. In addition to the local disk for OS swapping, applications use highly scalable I/O by writing to the local disk. The drawback of this flavor of I/O is that it is local: data must be migrated to the local disk before application execution and retrieved after execution. Thus, local I/O is predominantly used for intermediate, temporary results. A second flavor of I/O that is heavily utilized is global serial (NFS) I/O. This has the advantage of being global but the disadvantage that the performance only scales to the limit of a single NFS server (currently 20-100 MB/s). This type of I/O can be utilized for home directory, application source, and binaries, but not parallel I/O. The third flavor of I/O is global parallel I/O. The advantage of this is that the I/O rate delivered to an application tends to scale up as the number of writers is increased (up to a point, and then performance either stays constant or begins to decrease). This type of I/O is utilized for parallel reading of the input (or restart dump) and for parallel writingof science data and restart dumps. The disadvantage of global parallel I/O is that good parallel file systems are hard to come by and Open Source parallel file systems are even scarcer.

In addition to the programming model abstraction, the Livermore Model assumes parallel cluster management tools, resource management and control with near-real-time accounting, and job scheduling for allocation management, C, C++, and Fortran compilers, scalable MPI/OpenMP GUI debugger, and performance analysis tools.

Ideally, the I/O subsystem will provide for a scalable, cluster-wide file system that provides fault-tolerant services to MCR with no single point of failure. The I/O subsystem will have a disk capacity of at least the baseline requirement and will support RAID level 5. Due to the amount of storage required for MCR, the architecture should provide an I/O subsystem with large MTBF characteristics. The system will have sufficient bandwidth to read and write in parallel large volumes of data, in two distinctly different usage patterns. Very large single files accessed by a large number of MPI tasks, one per CPU, in a non-overlapping fashion is one usage pattern. The second is one proportionallysmaller file per MPI task, one per CPU, with all files in a single directory. This large-number-of-files situation requires a fast file-creation rate when a large number of MPI tasks open files from the same directory approximately contemporaneously. The I/O subsystem will also support a scalable parallel file system accessible from every node in the system. The ideal I/O subsystem will also be capable of providing services to requests beyond the MCR cluster, indicated in Figure 2.1-2 by the yellow switch bar that extends outside the cluster-wide box. As the parallel file system is a critical system resource, it will be highly reliable. For example, the Lustre Object File System is expected to provide the desired file-system characteristics and will leverage ASCI PathForward investments in Open Source software.

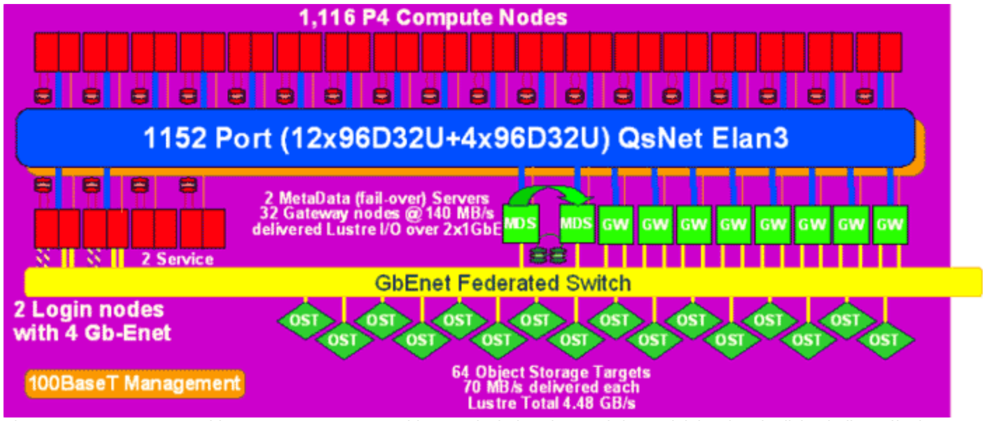


Figure 2.1-2. MCR SAN architecture. MCR system architecture includes clustered I/O model, local node disks,dedicated login nodes, dedicated visualization nodes and compute nodes and network attached RAID disk resources. Scalable user applications (either batch or interactive) will only run in the "parallel batch/interactive nodes" compute partition.The login nodes host interactive login sessions and code development activities, but not for running MPI jobs. The visualization nodes host parallel batch and interactive visualization jobs.

It is our intention to use the MCR system as the vehicle for providing the first SAN in to the LLNL unclassified computing infrastructure (see Figure 2.1-2); therefore, it is essential that the I/O subsystem for connections for MCR be based on 1 Gb/s Ethernet. Our strategy is to accrete other systems (e.g., IBM SP-based High Performance Storage System) to this SAN environment after MCR is integrated. With Lustre as the cluster wide file system, the aim is to have MCR provide scalable cluster wide parallel file system service for all OCF resources over time. Thus, the

Lustre Lite file system integrated with MCR will provide scalable global parallel performance to the other resourcesmentioned by having these other resources connect to the 1 Gb/s Ethernet SAN and by adding more OST resources to Lustre Lite.

Back to Contents

2.1.1 LC Linux Strategy for HPTC Scalable Clusters

Our strategy is to extend the Livermore Model from proprietary systems in use at LLNL (e.g., IBM SP with AIX and PSSP, Compaq Sierra with TruUNIX) to commodity (i.e., IA-32) nodes with Linux. The first implementation of this strategy was the Parallel Capacity Resource (PCR) procurement last year. Thatprocurement produced two clusters (one 128 dual P4 node and one 88 dual P4 node) that are now migrating into production usage in the Secure Computing Facility (SCF) for use by the ASCIongoing computing element.

Over the past 5 years, the Open Source community development model, popularized by the GNU project and the Linux OS development effort, has shown remarkable capability to deliver freely available software that satisfies a broad range of computing requirements in an astounding range of computingenvironments: from desktops to high-availability configurations and embedded systems to teraFLOP/s clusters. These efforts have had a significant impact on the high-performance technical computing (HPTC) landscape as witnessed by the fact that all major computer system manufacturers now offer Linux solutions. In the HPTC environment, the Open Source movement has created an environment in which multiple organizations can contribute software development and enhancements to cluster solutions. These development efforts have reached a critical mass and are now producing multiple cluster offerings that are competitive with other vendor proprietary solutions. In addition, the price performance of these solutions, when based on commodity hardware components, is extremely attractive for HPTC sites.

It is for these reasons that the University launched exploratory Linux efforts over three years ago. During this time, the power of the Open Source development model became even more persuasive. The benefits of Open Source for HPTC sites include:

- Access to software source for quick bug-fix ability.
- Hedge against vendor "change in support" status.
- HPTC sites tend to have unique and demanding requirements that vendors can't make money supporting.
- Joint development model has been proven effective multiple times.
- HPTC sites are developers and early adopters of critical, widely applicable technologies.
- Multiple Open Source HPTC sites find more bugs.
- The perception that Open Source has become the next wave of software development by increasing return on investment for companies that figure out how to leverage "free software."
- Fosters community development model that leads to accelerated innovation through competition.

LC is actively pursuing an Open Source development model to leverage these astounding benefits in a "Generally Available" or Production computing resource. To this end, our strategy is to focus on high-performance, scalable cluster computing environments with as much Open Source software as possible. Thedeployment strategy for Open Source technology is to start with small clusters and increase the number and size of clusters based on Open Source technology in Production over time.LC has demonstrated that this is a viable approach through the successful development of the PCR, a Linux-based set of clusters first deployed at the end of FY2001. With PCR, we used this strategy to fill the "capacity computing for capability (MPI parallelized) jobs" niche. The first step of this strategy allowed LC the flexibility, while still meeting SSP programmatic objectives, to start with small-sized clusters and then expand the capacity environment by adding clusters and by increasing the size of the clusters deployed over time as the Open Source software technology matures and LC gets more proficient at deploying Linux clusters into Production. This procurement represents the next step in this direction with the development of a capacity resource based on an Open Source development model.

The successful migration of LC computing to the commodity hardware and Open Source software technology can only be completed with LC fully contributing to and fully engaging in the community development and support model. However, this is only one aspect of the overall strategy. Past large-systemdeployments at LC have all been accomplished via long-term relationships with computing system manufacturers in the form of partnerships. This is a key methodology that LC hasutilized to build the University's existing unclassified M&IC environment.

The current MCR effort builds on that vendor partnership model and represents a solicitation for partnering between LC and a vendor partner in Open Source development efforts described below or those of interest to the partner. From market survey discussions in preparation for this RFP, it became clear that the cluster strategy espoused by LC and concretized by this procurement represents a "productizable" model that multiple potential vendor partners find highly attractive. Thus,these Open Source cluster efforts with a vendor partner should accomplish the following long-range goals:

- Enhance the state-of-the-art in high-end clusters.
- Provide competitive products for the vendor partner.
- Provide cost effective production clusters to accomplish LLNL M&IC and NNSA SSP program goals.

There remain challenges in the HPTC Linux cluster environment. The most prominent challenge is including the lack ofan Open Source, scalable, high-performance parallel file system. The second most prominent challenge is the lack of effective cluster scheduling (including checkpoint restart and Gang scheduling) technology. We intend to address these issues with future Open Source development in this partnership.

Back to Contents

2.1.2 MCR Hardware Architecture

The above M&IC requirements for an extremely cost-effective, large-scale scientific computing platform lead LC to large cluster architecture based on IA-32, Quadrics QsNet and BlueArc OST. IA-32 based nodes were selected after running a series of M&IC applications benchmarks that indicate that Pentium 4/Xeon Foster and Prestonia processors deliver the best performance and cost performance of any option available. As indicated in Section 1, M&IC applications are floating point arithmetic and memory bandwidth intensive. However, given that IA-32 bus bandwidth at 3.2 GB/s (1.7–1.9 GB/s delivered) in Foster/i860/RDRAM or Prestonia/(e7500|ServerWorks GrandChampion LE)/PC200 DDRSDRAM based motherboards is exhausted for most M&IC applications with a one or two active processes or HyperThreads (a few can utilize more processors), dual nodes have been selected by M&IC users for the MCR. The SuperMicro P4DPG motherboard with Intel e7500 chipset and 2.4 GHz Prestonia processors was selected as the compute node.

M&IC applications are quite demanding on delivered interconnect latency and bandwidth. Thus, Quadrics QsNet Elan3 wasselected because it delivers high bandwidth (>300 MB/s) and low latency (<5.0 µs) at commodity interconnect pricing. In addition, M&IC plans to run the MCR as a combined capabilityand capacity machine and therefore a reduced minimum bisection bandwidth can be tolerated. Thus, we have chosen a QsNet configuration with 128-port (128-way) switch elements constructed as a two-stage, fat-tree, federated switch configuration with twelve first-level and four second-level switches: a total of sixteen 128-way switches (see Figure 2.1-3). The first-level switches are configured with 96 ports for nodes and 32 ports for connecting to other QsNet switches (96D32U). The second-level switches are configured with 96 ports for first level switches and 32 unused ports (96D32U). This configuration costs less than a 64D64U level one switch configuration, at the expense of reduced bisection bandwidth.

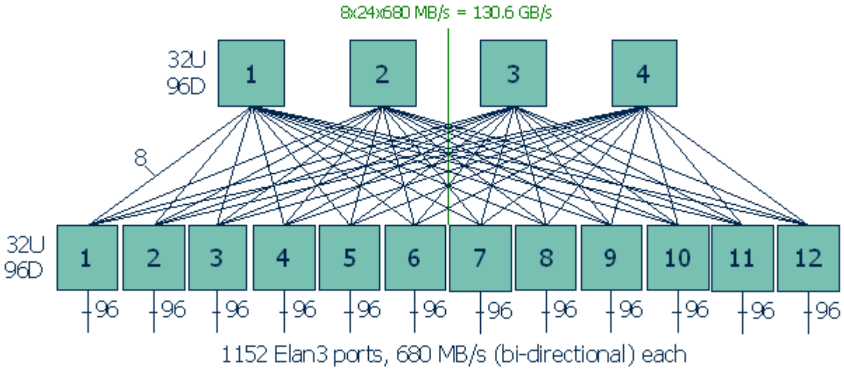

8x24x680 MB/s = 130.6 GB/s
32U
96D
1
2
3
4
8
32U
96D
1
2
3
4
5
6
7
8
9
10
11
12
96
96
96
96
96
96
96
96
96
96
96
96
1152 Elan3 ports, 680 MB/s (bi-directional) each

Figure 2.1-3. Quadrics QsNet Elan3 for MCR is based on 16 128-port Elan3 switches configured as a two-stage, fat-tree federated switch. Each of the Elan3 twelve first-level switches is configured with 96 ports for nodes and 32 ports for the second-level switches (96D32U). The four second-level switches are configured with 96 ports for connecting the first-level switches and 32 unused ports (96D32U).

This switch configuration leads to a natural scalable unit size of 96 nodes. Thus, the University envisions buildingthe MCR in ten equal size scalable units of 96 nodes each. From Figure 2.1-2 it is clear that a vast majority of the MCR nodes are compute nodes. Hence, if we define a single scalable unit, to be built first, called the First Scalable Unit (FSU) with the gateway nodes, login nodes, management nodes (not on QsNet switch), and Meta Data Server (MDS) (fail-over pair) nodes and enough compute nodes to fill out the remainder of the first scalable unit, then the nine remainder scalable units are identical. These nine scalable units are called Compute Node Scalable Unites (CNSU)

The Lustre Lite file system has several ramifications on the MCR hardware architecture. First, Lustre Lite has file system clients that provide global file system access on every compute node. This implies that the high-speed, low-latency communication mechanism for the file system is QsNet. Second, Lustre Lite requires a MDS fail-over pair. These MDS nodes must be configured to support Kimber Lite Linux High Availability fail-over schemes. This means a dedicated TTY and 100BaseT Ethernet for heartbeat and shared disk for metadata. The MDS needs about 2.5% of object storage target capacity (or about 2.5 TB of usable RAID5 disk) for the meta data accessible at 256 MB/s delivered of 512B block raw read/write (or 500 512B I/Os per second) performance from either node. Two Gb/s FibreChannel-attached RAID5 disks are ideal for this. Third, as Lustre Lite migrates to full Lustre, the Lustre file system will be extended beyond the MCR cluster boundary. An extensible, interoperable, commodity SAN technology is required for extending Lustre into this heterogeneous environment. We plan to integrate the Federated Gb Ethernet switch infrastructure in FY03. Until then, the Lustre Lite Object Storage Targets (OST) will be direct attach or small port count Gb Ethernet switch connected for Gateway node fail-over tolerance. Fourth, Lustre Lite requires OST to manage creating, locking, writing, reading, and deleting ofobjects (aggregations of disk blocks). The chosen SAN-based OST (NAS) for MCR is the BlueArc Silicon Server Si7500 with OST enhancements on 1 GB/s Etherenet.

Back to Contents

2.2 LC Software Environment for Linux Clusters

To execute the LC Linux software strategy, LC provides a complete software environment for Linux clusters called CHAOS (Clustered High Availability Operating System) that meets the programmatic and operational requirements described in the sections above. In addition, LC currently is actively involved in the Open Source development of Linux clustering tools, the SLURM resource manager, the DPCS metabatch scheduler and resource accounting system, and the Lustre CWFS. These components and the rest of the CHAOS environment will be installed on the MCR cluster after it is delivered to LLNL.

2.2.1 Clustered High Availability Operating System (CHAOS)

Livermore Computing produces and supports CHAOS, a cluster operating environment based on RedHat Linux. At the core of CHAOS is a RedHat "boxed set" distribution. Some components of that distribution are modified to meet the demands of high-performance computing and the LC center. A number of additional cluster-aware components are added on.

A CHAOS distribution contains a set of RPM (RedHat Package Manager) files, RPM lists for each type of node (compute, management, Gateway, and login), and a methodology for installing and administering clusters. It is produced internally and therefore supports a short list of hardware and software. This focus on the LC environment permits the Laboratory to support CHAOS with a small staff and to be agile in planning its content and direction.

In addition to the products of Open Source development described below and the RedHat boxed set distribution, CHAOS includes the following software components:

- **kernel**—The CHAOS kernel is based on a RedHat kernel with additions in the areas of VM/device support for Quadrics Elan3, ECC and FLASH memory device support for i860 chipset, Mission Critical Linux (MCL) crash dump support, miscellaneous bug fixes, and optimized configurations for our hardware.
- **Quadrics RMS, libelan, MPI, etc**.-The Quadrics software environment used to run parallel programs.
- **Crash**—The MCL crash dump analysis suite is used to examine post mortem contents of a kernel crash dump.
- **lm_sensors**—Hardware monitoring, linked to the SNMP host monitoring system, monitors motherboard chipset sensors such as temperature, fan speed, and power supply voltages.
- **fping**—Fping is a rudimentary node status tool that can ping nodes in parallel. In combination with genders tools, fping can quickly find nodes in the cluster that are turned off or otherwise unreachable on management network.
- **OpenSSH**—OpenSSH provides encrypted remote login/shell service that is PAM-aware.
- **PAM Tools**—PAM modules for One Time Passwords (OTP), Kerberos V, and RMS (to authorize a user's access to a compute node only when RMS is running that user's job) are used to leverage LC's DCE and OTP infrastructure for PAM-aware applications.
- **Firmware**—Firmware images for motherboards, including FLASH/CMOS support software, is included in CHAOS. Firmware and supportsoftware for power control/serial console hardware is also included.
- **MPI Test Suite**—The Pallas MPI Benchmark (PMB), Effective Bandwidth test (BEFF), and Quadrics MPI ping-pong test (mping) are packaged with CHAOS with a script to maintain a continuous MPI workload under RMS for testing purposes.
- **NTTCP**—The NTTCP TCP bandwidth test is packaged along with genders-aware scripts that can simulate load on the management Ethernet for testing purposes.
- **super**—Super extends administrative privileges to non-root users.
- **Intel Compilers**—The Intel IA32 FORTRAN, C, and C++ compilers.
- **PGI Compilers**—The Portland Group Fortran and C Compilers.
- **TotalView**—The TotalView parallel Debugger from Etnus.
- **Other Libraries/Tools**—Other libraries and tools such as Atlas, COG, NDF, netCDF, Hyper, ScaLAPAC, OpenGL, Yorick, Silo, VTK, and Findentry are maintained on LC Linux systems.

Back to Contents

2.2.1.1 CHAOS Status

CHAOS version 1.0 currently runs on the PCR systems procured in FY2001. These are 26-, 88-, and 128- way clusters based on dual 1.7 Ghz Pentium 4/Xeon nodes and Quadrics Elan3 interconnect. Version 1.0 is based on RedHat 7.1 and the RedHat 2.4.9 errata kernel series.

CHAOS 1.0, the first "official" CHAOS released after formal integration testing, was installed on the PCR clusters in July 2002 time frame. It is based on RedHat 7.3 and the RedHat 2.4.18 kernel.

CHAOS was extended to operate on the MCR cluster as needed to account for the different motherboards and LinuxBIOS.

2.2.2 LLNL Cluster Tools

The following Open Source cluster tools are under active development and have been deployed on the PCR clusters:

- **pdsh**—The Parallel Distributed SHell utility executes processes across groups of nodes in parallel. It is also capable of running small MPI jobs on the Elan3 interconnect.
- **YACI**—Yet Another Cluster Installer is Livermore's system installation tool based on various cluster installers such as VA system imager and LUI that can fully install an 88-node cluster in about 10 minutes. It is image-based and uses an NFS pull from many network-booted nodes running in parallel.
- **Genders**—is a static system configuration database and rdist Distfile preprocessor. Each node has a list of "attributes" that in combination describe the configuration of the node. The genders system enables identical scripts to perform different functions depending on their context. An rdist Distfile preprocessor expands attribute macros into node lists.
- **ConMan**—The ConMan console manager manages serial consoles connected either to hardwired serial ports or remote terminal servers (telnet based). Performs logging of console output, and manages interactive sessions, permitting console sharing, console stealing, console broadcast, and interfaces for transmitting a serial break or resetting a node via PowerMan.
- **PowerMan**—The PowerMan power manager manages system power controllers and is capable of sequenced power on/off for groups of nodes and initiating reset (both plug off/on and hardware reset if available). Powerman currently supports the Linux Networx ICE box, WTI RPC's, and the API Networks modified Wake-on-lan. It can be extended to support new hardware.
- **Host Monitoring System**—LC's SNMP based host monitoring system stores current state in a MySQL database and long-term state in an RRD (roundrobin database). Collection software polls cluster nodes in parallel using SNMP bulk queries and a sliding window algorithm to reduce polling latency. Status is presented via the web using Apache and PHP.

Back to Contents

2.2.3 Simple Linux Utility for Resource Management

SLURM
is an Open Source, fault-tolerant and highly scalable cluster management and job scheduling system for clusters containing thousands of nodes. SLURM is presently under design and development at LLNL..

The primary functions of SLURM are:

- Monitoring the state of nodes in the cluster.
- Logically organizing the nodes into partitions with flexible parameters.
- Accepting job requests. While SLURM can support a simple queuing algorithm, DPCS will manage the order of job initiations through its sophisticated algorithms described in Section 2.2.4 of this document.
- Allocating both node and interconnect resources to jobs.
- Monitoring the state of running jobs, including resource utilization rates.

SLURM will utilize Kerberos V5 based authentication. The design also includes a scalable, general-purpose communications infrastructure. APIs will be available to support all functions for ease of integration with external schedulers. SLURM is written in the C language, with a GNU autoconf configuration engine. While initially written for Linux and Quadrics Elan3 interconnects, our design calls for ease of portability. We anticipate having a functional version of SLURM available in August 2002

Back to Contents

2.2.4 Distributed Production Control System (DPCS)

The DPCS is an Open Source product of the LC Center. The DPCS project began in 1991 when LC started to convert all of its production computer systems to UNIX platforms. DPCS was in production in October 1992 and has continued to develop since then.

The primary purpose of DPCS is to allocate computer resources, according to resource delivery goals, for LC's UNIX-based production computer systems. This is accomplished through a complex hierarchy of:

- Computer-share bank accounts.
- Time-usage monitoring tools.
- Run-control mechanisms.

DPCS lets organizations control who uses their computing resources and how rapidly those resources are used. It also manages an underlying batch system that actually runs production jobs guided by DPCS policies.

Resource Delivery Goals: Defined by LC management in coordination with program managers. Program managers oversee a group's access to production computer system resources. These goals are "programmed" into DPCS, which then attempts to meet those goals. In other words, DPCS is used to assure that the right people, projects, and organizations get appropriate access to a center's computer resources. The DPCS contains two major subsystems that work together to deliver resources as required. The Resource Allocation & Control System (RAC) provides mechanisms for allocating machine resources among diverse users and groups, while the Production Workload Scheduler (PWS) provides mechanisms for automatically scheduling batch (production) jobs on the machines.

The RAC system is used to declare production hosts, to create and manage recharge accounts, resource allocation partitions, resource allocation pools, and user resource allocations within the resource allocation pools. Caveat Emptor: a recharge account should not be confused with a user login account. DPCS uses the term bank as a synonym for "resource allocation pool."

The PWS is a set of daemons and utilities that work with the RAC system to schedule batch jobs on the DPCS production hosts. It employs policy as instructed through the mechanisms provided to DPCS managers and resource managers to prioritize and schedule production appropriately.

Back to Contents

2.2.5 Lustre Lite Cluster Wide File System

The University plans to utilize the Lustre Lite Cluster Wide File System on the MCR cluster. To that end, LC and Cluster File Systems, Inc., have been actively engaged in developing a "lite"version of Lustre to run on MCR in summer 2002.

Lustre Lite's impact on the MCR architecture is substantial. This solicitation includes two MDS nodes in fail over configuration and about 2.5 TB of disk for Lustre Lite meta data. In addition, the University will supply 64 OSTs with a combined I/O rate of 4.48 GB/s, 100 TB of RAID5 disk attached to the cluster via GbEthernet. The specified configuration includes 32 gateway nodes with two GbEthernet adapters and one QsNet adapter to gateway Lustre file system data (both meta data and objects) between the OST, MDS, and Lustre Clients (compute and login nodes).

Because Lustre Lite is under active development, one of the first activities envisioned for the MCR cluster during factory build is the scaling testing and performance tuning of Lustre Lite. To this end, the FSU should be built first (hence the name) and attached to the Government Furnished Equipment (GFE) OST infrastructure to facilitate this testing activity as early as possible.

2.2.6 The Livermore Computing Linux Cluster Support Model

LC Open Source developers (Cluster Tools, DPCS, SLURM, Lustre Lite) work closely with system administrators and users to resolve problems on production systems. For any given software package, there is a designated package owner who handles any support issues that arise in production. Depending on the nature of the package, owners may be the primary developer and fix bugs themselves, or they may be the liaison to an external support resource.

External support relationships are primarily developer-to-developer. In the case of RedHat, we have a full-time RedHat engineer on site who can work directly with Livermore systems and support people and act as the liaison to RedHat for everything in the RedHat boxed set. In the case of Quadrics, an ongoing Cooperative Research and Development Agreement (CRADA) and a support subcontract are leveraged to get bugs fixed in production.

2.2.7 Integration Testing

Each CHAOS release is subject to integration testing that includes regression tests for past problems, basic functionality tests, and real users' applications. Each of the software components is developed asynchronously, but come together in system (CHAOS) releases and separate package (DPCS, Lustre Lite, SLURM, Cluster Tools) releases. Due to this separation, system and package testing and installation on production clusters can be scheduled and executed independently. A 26-node development cluster called DEV, which can be rapidly reinstalled into any past CHAOS environment as well as new prototype environments, is available for unit testing of individual software components, integration testing of a complete CHAOS release, and debugging of defects that arise in production. This cluster, along with the project CVS repository, can accommodate external collaborators working with LC on the Open Source projects described above.

Back to Contents

2.3 MCR Build Strategy

The MCR build strategy is based on the scalable unit concept. Linux NetworX will build the FSU and install their HiPer Linux distribution including modifications for QsNet and Lustre Lite. The FSU, which contains all the login, management, gateway, Lustre Meta Data servers and disk, and a complement of compute nodes, will undergo initial functionality and performance testing and then be used as the vehicle to scale up Lustre Lite as MCR is built.

After the FSU is complete, Linux NetworX will assemble the remainder of the cluster consisting of eleven CNSU by adding CNSU to the Quadrics Federated QsNet switch in groups of three (up to nine CNSU) and allowing time for Lustre Lite scaling testing. After delivery to LLNL, Linux NetworX will add the final two CNSU to the configuration.

Once the MCR is built and Lustre Lite scaling to 960 nodes is complete, pre-ship testing commences. This level of testing consists of running a specific set of parallel applications across the machine. Once the pre-ship exit criteria defined in pre-ship test plan are met, the machine is disassembled and shipped to the LLNL site and reassembled. At LLNL, the post-ship test is re-run to verify that the hardware survived disassembly, shipment, and reassembly.

Once the hardware has been reassembled and passed the post-ship test and turned over to University personnel, the University will install the CHAOS environment with the aid of the Linux NetworX, Quadrics, BlueArc and Cluster File Systems, Inc. personnel. Acceptance testing will take place with the LLNL CHAOS 1.0 distribution installed.

Back to Contents

This page last modified on November 8, 2002
For information about this page, contact Mark Seager.

UCRL-CR-148022