LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-SR-406177

The ASC Sequoia Programming Model

Mark Seager

August 11, 2008

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

The ASC Sequoia Programming Model

Mark Seager

August 5, 20008

1.0 The Livermore Model

In the late 1980's and early 1990's, Lawrence Livermore National Laboratory was deeply engrossed in determining the next generation programming model for the Integrated Design Codes (IDC) beyond vectorization for the Cray 1s series of computers. The vector model, developed in mid 1970's first for the CDC 7600 and later extended from stack based vector operation to memory to memory operations for the Cray 1s, lasted approximately 20 years (See Slide 5). The Cray vector era was deemed an extremely long lived era as it allowed vector codes to be developed over time (the Cray 1s were faster in scalar mode than the CDC 7600) with vector unit utilization increasing incrementally over time. The other attributes of the Cray vector era at LLNL were that we developed, supported and maintained the Operating System (LTSS and later NLTSS), communications protocols (LINCS), Compilers (Civic Fortran77 and Model), operating system tools (e.g., batch system, job control scripting, loaders, debuggers, editors, graphics utilities, you name it) and math and highly machine optimized libraries (e.g., SLATEC, and STACKLIB). Although LTSS was adopted by Cray for early system generations, they later developed COS and UNICOS operating systems and environment on their own.

In the late 1970s and early 1980s two trends appeared that made the Cray vector programming model (described above including both the hardware and system software aspects) seem potentially dated and slated for major revision. These trends were the appearance of low cost CMOS microprocessors and their attendant, departmental and mini-computers and later workstations and personal computers. With the wide spread adoption of Unix in the early 1980s, it appeared that LLNL (and the other DOE Labs) would be left out of the mainstream of computing without a rapid transition to these "Killer Micros" and modern OS and tools environments. The other interesting advance in the period is that systems were being developed with multiple "cores" in them and called Symmetric Multi-Processor or Shared Memory Processor (SMP) systems. The parallel revolution had begun.

The Laboratory started a small "parallel processing project" in 1983 to study the new technology and its application to scientific computing with four people: Tim Axelrod, Pete Eltgroth, Paul Dubois and Mark Seager. Two years later, Eugene Brooks joined the team. This team focused on Unix and "killer micro" SMPs. Indeed, Eugene Brooks was credited with coming up with the "Killer Micro" term. After several generations of SMP platforms (e.g., Sequent Balance 8000 with 8 33MHz MC32032s, Allian FX8 with 8 MC68020 and FPGA based Vector Units and finally the BB&N Butterfly with 128 cores), it became apparent to us that the killer micro revolution would indeed take over Crays and that we definitely needed a new programming and systems model. The model developed by Mark Seager and Dale Nielsen focused on both the system aspects (Slide 3) and the code development aspects (Slide 4). Although now succinctly captured in two attached slides, at the time there was tremendous ferment in the research community as to what parallel programming model would emerge, dominate and survive. In addition, we wanted a model that would provide portability between platforms of a single generation but also longevity over multiple - and hopefully - many generations. Only after we developed the "Livermore Model" and worked it out in considerable detail did it become obvious that what we came up with was the right approach.

In a nutshell, the applications programming model of the Livermore Model posited that SMP parallelism would ultimately not scale indefinitely and one would have to bite the bullet and implement MPI parallelism within the Integrated Design Code (IDC). We also had a major emphasis on doing everything in a completely standards based, portable methodology with POSIX/Unix as the target environment. We decided against specialized libraries like STACKLIB for performance, but kept as many general purpose,

portable math libraries as were needed by the codes. Third, we assumed that the SMPs in clusters would evolve in time to become more powerful, feature rich and, in particular, offer more cores. Thus, we focused on OpenMP, and POSIX PThreads for programming SMP parallelism. These code porting efforts were lead by Dale Nielsen, A-Division code group leader, and Randy Christensen, B-Division code group leader. Most of the porting effort revolved removing "Cray'isms" in the codes: artifacts of LTSS/NLTSS, Civic compiler extensions beyond Fortran77, IO libraries and dealing with new code control languages (we switched to Perl and later to Python). Adding MPI to the codes was initially problematic and error prone because the programmers used MPI directly and sprinkled the calls throughout the code. Later we learned to abstract out the MPI calls into generic macros that were more understandable to the domain specialists and also impose a discipline of requiring all MPI communications in setup blocks and post processing blocks. The former allowed the number of lines of communications in codes to be significantly reduced (3-5% of the overall code). The latter enabled node code development on workstations and laptops (for unclassified codes). Both enabled better code readability, maintainability and debugability.

This approach provided programming model stability in an era of turmoil in the industry. It did so by leveraging the desktop environment (but now coupled with a high speed, low latency interconnect) and by tracking and responding to the desktop as that environment rapidly evolved. It also allowed the development of node code on the desktop with parallel algorithm development and application scalability on the centralized parallel machines. As a direct consequence of this strategy, the migration from Unix to Linux on the desktop and centralized parallel machines was relatively smooth one.

In 1991 we executed a procurement to bring in the first classified cluster of SMPs, the Meiko CS-2 with 256 SPARC cores, in order to incentivize applications developers to port to Unix, Microprocessor and implement MPI parallelization into the IDC codes. This effort took 2-3 years, but all the mainline LLNL codes were transformed from the previous Cray vector era, NLTSS, vector serial to standards based MPI for microprocessors. This transition was traumatic for the Stockpile Stewardship Program at LLNL and reduced the rate of implementation of additional physics models in the codes during this period.

By the mid-life of the Meiko, the Accelerated Strategic Computing Initiative (ASCI) leveraged many of the "lessons learned" out of the Cray to Cluster of SMP migration as a basis for the platform and applications strategies. In particular, targeted clusters of SMPs for the Blue (3.0 teraFLOP/s Blue-Mountain at LANL and 3.6 teraFLOP/s Blue-Pacific at LLNL) in a joint procurement, the White (12.3 teraFLOP/s), Q (20 teraFLOP/s at LANL) and Purple (100 teraFLOP/s) procurements and associated programmatic code development efforts. However, early experience with OpenMP and POSIX threads indicated that the balance between computational power and message passing power of these platforms favored the MPI everywhere approach. There were severe drawbacks associated with OpenMP. First, although OpenMP works well for well structured meshes and algorithms highly dependent on linear algebra, it was not well suited for algorithms that were expressed as a large number of small loops (many with critical sections) and a non-trivial amount of work in the serial sections between loops (e.g., setup and post processing of major computations). Thus in many IDC packages, OpenMP overheads limited scalability, with a few algorithms having good OpenMP scalability. PThreads and other threads based SMP programming methodologies were also exploited, but had more problems with debugging due to hard-to-find race conditions. In addition, both OpenMP and PThreads, when coupled with MPI presented several additional challenges. First, bugs in the infrastructure supporting threaded MPI, OpenMP and PThreads were problematic because this model was not widely stressed by a large number of customers. ASCI was clearly way out in front on applications scalability. Second, the network interfaces of the day were not very good at delivering a high percentage of peak bandwidth for small and moderately sized messages (characteristic of IDC) from a single MPI task, but did deliver a higher aggregate messaging rate for multiple tasks. This favored the MPI everywhere approach for scalability.

However, the MPI everywhere approach also had its problems. First, MPI scalability was limited in the IDC by the ability to scale MPI_Barrier and MPI_Allreduce (on 64b floating point min, max and sum) among other factors such as MPI task load balancing. On ASC White the IDC codes spent 50% of the runtime (not 50% of the communications time) in executing MPI_Barrier and MPI_Allreduce when scaling beyond 4,096 MPI tasks on ASC White (with 8,192 being the maximum on that system). Also, operating system inflicted MPI task runtime variations from task to task on a node (often referred to as OS noise or jitter) on White and Purple introduced scaling problems and runtime variations of 2-4x IDC time step to time step. These problems are so severe that one core (out of 16 on White and 8 on Purple) of the compute nodes is typically left idle by the applications for occasional OS use.

All of the above lessons learned and practical realties inform our programming model decisions for Sequoia. In addition, they also give us a good indication where to focus our resources on risk reduction.

2.0 Sequoia Code Development Strategy

The prospect of scaling codes, with improved scientific models, databases, input data sets and grids, to O(1M) way parallelism is a daunting task, even for an organization with successful scaling experience up to 212,992-way parallelism with BlueGene/L. The fundamental issue here is how to deal with the geometric increase in cores/threads on a processor for the foreseeable future. To add insult to injury, the memory density Moore's Law now has a doubling rate half that of the core increase rate: 2x every 3 years. Thus simply scaling up the current practice of one MPI task per core, no longer appears to be sustainable.

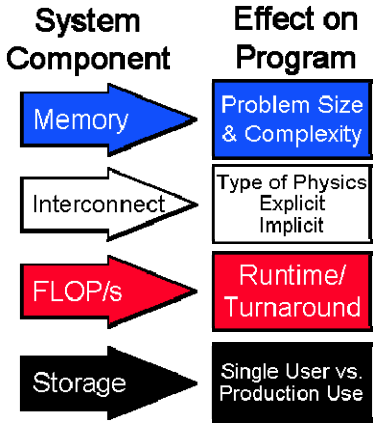

Figure 1: ASC major system component prioritization from 1997 shows that memory and interconnect are the two most important system attributes.

These difficulties are summarized by the fact that obtaining reasonable code scaling to O(1M) MPI tasks will require that the serial work and task load imbalance combined in all physics packages in an IDC be reduced to less than 1 in O(1M). Given that code development tools will not have the resolution to differentiate the 1 in O(1M) differences in subroutine execution times as well as the difficulty of balancing the workload to that level, together lead one to seriously consider that scaling to this number of MPI tasks may well be an insurmountable obstacle. These considerations, among others, lead one to consider using multiple cores/tasks per MPI task.

Let us consider the highest priority system resource for Stockpile Stewardship: memory. The ASC codes require at least 1GB per MPI task (not per core, not per node and not a bytes per FLOP/s ratio) and would significantly benefit from 2GB per MPI task. This is **a *critical*** platform attribute. An application mapping of one MPI task per core would lead to a platform with aggregate memory requirement on the order of 2-4PB, which is not affordable. It is also not practical (due to MTBF and power considerations) in the projected Sequoia build timeframe of 2010-2011. This also leads one to consider using multiple cores/threads per MPI task.

If one considers the second ***critical*** system attribute for ASC codes, the ASC Program requires >2 million messages per second per MPI task. Again, in mapping one MPI task per core onto a multicore processor per socket and one or more sockets per node with each node having one or multiple interconnect interfaces, the resulting interconnect requirements make the overall system either too expensive, or too specialized, or too high risk (or a combination of all three) to be general purpose. This again leads one to focus on using multiple cores/threads per MPI task.

By considering the use of a reasonable number of cores/threads per MPI task (i.e., SMP parallelism within the MPI node code), one has theoretically factored an impossible problem (scaling to O(1M) MPI tasks) into one that is doable (scaling to O(50-200K) MPI tasks) and another that is merely difficult (adding effective SMP parallelism to the MPI node code). This is described in Slide 15. As a first step, the ASC Program has already initiated its efforts within the Tri-Laboratory community on scaling the IDC and Weapon Science (WC) codes to higher MPI parallelism with an addition to the BlueGene/L platform of 32 higher memory racks in order to determine the limits of MPI scalability for various algorithms and packages.

Thus the extension to the existing ASC programming model that we are looking at is to reinvigorate the SMP parallelism within IDC codes and to exploit this additional form of SMP parallelism within the Weapon Science (WS) codes. This is a minimally invasive approach that allows for incremental introduction of SMP parallelism, package by package and/or incremental improvement of previous parallel implementations over time. This is a similar methodology employed on the previous generation IDC codes that were incrementally improved to employ or better exploit vectorization within the Cray Vector era.

However, ASC IDC and WS codes must remain ubiquitously portable, which means any innovation on compiler, runtime and hardware technology for solving the concurrency problem must have open runtime and operating interfaces and be comprised of incremental changes in the existing C, C++ and Fortran standard language specifications. It must also be adopted by multiple vendors in the marketplace.

Revisiting the SMP parallelism approach now is potentially advantageous because the industry recognizes that new parallel programming models are required. The x86 ecosystem cannot continue to sell more cores without ultimately delivering faster response time to individual applications. Currently multi-core is exploited primarily by running multiple programs simultaneously (e.g., listening to MP3s while downloading from the internet and surfing websites). However, this form of parallelism does not change the response time of critical applications (e.g., web browser or word processor) other than remove contention for CPU resources by other programs. At some point soon, desktop applications themselves will need to be parallelized or sales of 8 or 16 core laptops will come to a grinding halt.

From the Sequoia market survey we understand in some detail that multiple researchers in industry are working on novel techniques to conquer SMP parallelism (e.g., Transactional Memory and Speculative Execution) for desktop applications in order to enable compelling applications for mainstream Windows and Linux desktop and laptop users. The ASC Program intends to ride this industry trend with a close collaboration with the “winning” Sequoia bidder through a separate R&D subcontract (and potentially the “runner up” via a second R&D subcontract).

In discussions with the potential bidders during the ASC Sequoia market survey, we have come to focus on a *specific target architecture* for the above ASC “Livermore Model” programming model extension that all potential bidders find acceptable, even advantageous to work on in partnership with ASC. Within these discussions we have identified the minimal extensions required for compilers, the runtime system, and the operating system. See Figure 2. This model allows the exploitation of different SMP paradigms within an application. This frees the algorithm developer and applications programmer to fit the parallel model to their problem rather than the other way around. In addition, this model allows diverse packages to be melded into a single application with minimal restrictions. First we assume at least three models of SMP parallelism: PThreads, OpenMP and something innovative the vendor Offeror bids.

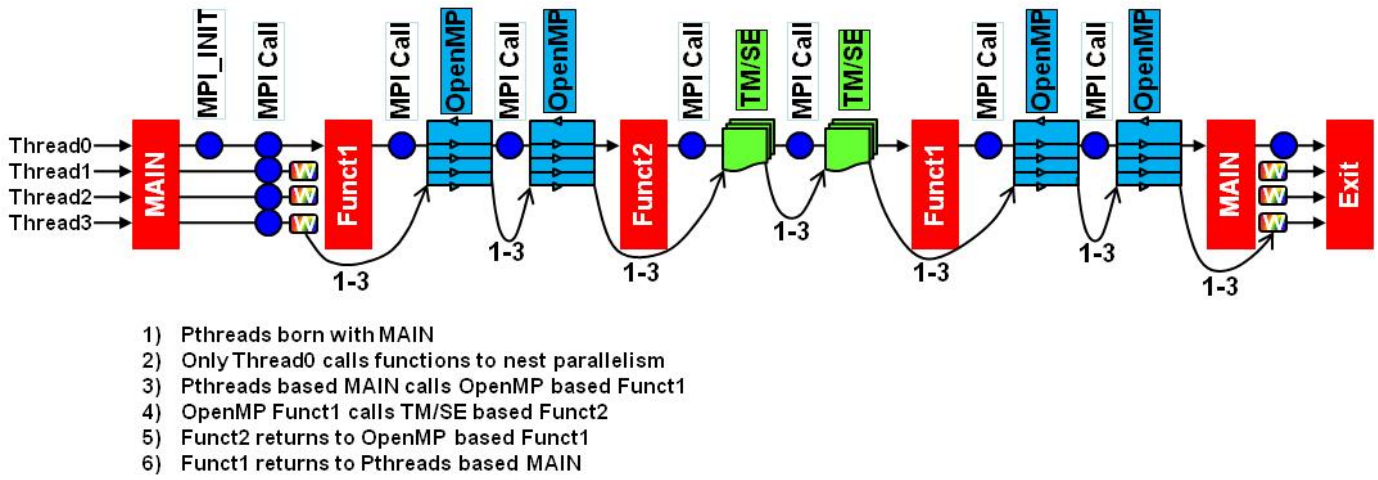

Figure 2: Nested node concurrency extension to the Livermore Model.

We expect Transactional Memory (TM) and Speculative Execution (SE) models to be proposed by multiple vendors. These models are described extensively in the literature. However the salient properties of TM is that it allows threads to execute a complex series of computations and interactions with memory atomically with the hardware keeping track of potential conflicts and restarting one or more of the conflicting transactions in this presumably unlikely event. SE provides the ability to the hardware, compiler and runtime system to identify complex instruction sequences that can be executed independently in parallel with greatly relaxed assumptions about conflicts because the hardware will catch such conflicts and restart or abandon a speculative execution segment in the unlikely event of actual conflict during runtime. *Note that this is vastly different from the current speculative execution found in many modern microprocessors that dynamically try to find more instruction level parallelism to exploit by out of order execution within a small window of candidate instructions within the instruction issue unit.*

We expect TM to be extremely beneficial in Monte Carlo simulations because it will allow complex updates to global state (such as energy in a transport simulation) without the need for locks. Memory locking is an exceedingly poor tool for Monte Carlo because locking a global array can dramatically hamper scalability. Locks themselves are notoriously bad on modern hierarchical memory subsystems as they flush the instruction and memory pipelines and thus cause extensive delays in single thread processing when the lock is obtained and released. Even worse disruption occurs when the lock is not obtained and the thread "spin-waits": this is the case even when there are no actual conflicts within the locked code region.

We expect several forms of SE to be exceedingly beneficial to codes that are not expressed well in either OpenMP or PThreads parallelism. For example, the Sequoia Unstructured Mesh Transport (UMT) benchmark is representative of the type of radiation transport relevant to IDC[1]. In this radiation transport solution technique sweeps are made through the 3D grid that are not aligned with any ordinal direction (X, Y, and Z). For UMT the sweep ambiguity is compounded with a fully unstructured grid. Fortunately, there is a first order recurrence relationship buried in this grid ambiguity that translates loosely into a producer consumer relationship of data behind the sweep being used to update grid zones in the update sweep. Thus, there is a huge amount of latent parallel work that is performed in these triply nested "for loops." However, there is no way, let alone a convenient way, to express this with Fortran or C/C++ and

[1] The Sequoia Benchmark suite is extensive and organized to test the spectrum of requirements coming from the IDC codes. It can be found at https://asc.llnl.gov/sequoia/benchmarks/

OpenMP or Pthreads. With careful analysis of these triply nested "for loops" in UMT, research compilers assuming hardware support for speculative execution from at least two potential bidders for Sequoia have found more than 128 way parallelism with over 90% efficiency in a worst case minimum parallelism test case.

In the model described in Figure 2, we represent one MPI task (of one or more) executing on a node. In this example, the MPI task starts execution with four software threads that are assumed to correspond to four hardware cores/threads in the node. These software threads live throughout the execution span of the MPI task and job. For this example, it is assumed that the MAIN entry point in the application is programmed with PThreads and must be carefully programmed so that only one thread (Thread0) calls MPI_INIT(). In addition, only one thread (Thread0) can call a function from a package programmed in another parallel programming paradigm (nesting level 1, OpenMP in this case). The other three threads (Thread1 through Thread3) call a special thread support library that allows them to be "repurposed" efficiently. The diagram shows Thread1 through Thread3 being repurposed to successive OpenMP loops in Funct1. Thread0 executing in Funct1 then calls another package with yet a third style of parallel programming (nesting level 2, TM/SE in this case). Threads1 through Thread3 are repurposed again to the TM/SE regions in Funct2. Finally Funct2 returns to Funct1 (nesting level 1) which performs more OpenMP loops and returns to MAIN (nesting level 0). This unwinds the parallel nesting.

This model restricts the way parallelism can be used in applications, but covers essentially all the real world cases we have encountered in our IDC and WS codes. Restrictions include the requirement that MPI tasks and HW/SW threads are created at job launch and terminated only upon job completion (no thread fork join model allowed). The number of MPI task times the number of threads per task must be less than or equal to the number of hardware cores/threads on a node. In addition, the runtimes for PThreads, OpenMP and TM/SE must cooperate to reuse the HW/SW threads effectively and most importantly: efficiently. We anticipate that the overhead associated with repurposing threads must be approximately the same as a subroutine call and jump. Lastly, we anticipate that efficient hardware support will be required for locks (e.g., SRAM lock boxes shared between the cores) and OpenMP constructs such as self scheduling do loops (e.g., fetch and add to registers or L1 memory) in order to make PThreads and OpenMP implementations more efficient and scalable.

Due to these modest restrictions, feature requirements for a Light Weight Kernel (LWK) and runtime are minimal. For example, no dynamic thread creation/destruction mechanisms are required; no thread scheduling or context switching mechanisms are required. This greatly reduces the complexity of the LWK and runtime implementations, testing and stabilization efforts.

3.0 Risk Mitigation Strategy

The industry prospect of dealing with multi-core can seem overwhelming due to the complexity of coming up with a parallel programming paradigm for the masses and pushing out the solution to the x86 ecosystem in order to make it ubiquitous. There is significant industry risk associated with this effort. Of course, extending the ASC IDC and WS codes to use muticore is also a substantial undertaking. *However, the two risks are not equivalent*. In fact, the ASC risk is substantially smaller. This is due to several factors. First the IDC and WS codes are already MPI parallel and have demonstrated, with considerable effort, significant scalability (8,192 for IDC and 212,992 for WS). Second, the primary IDC goal for Sequoia is to run IDC codes at 12x-24x Purple capability (8,192 MPI task level) in throughput mode: multiple 8,192 jobs running simultaneously with effective aggregate performance improvement of 12x-24x. This emanates from the Uncertainty Quantification mission of the machine. The primary WS goal for Sequoia is to run the WS codes at full scale and obtain 20x-50x improvement over BlueGene/L. This emanates from the mission to support Boost science, one of the key remaining unknowns impeding simulation with quantified uncertainties (predictive simulation). However, this effort is roughly

equivalent to the effort it took to scale them up from White (8,192 way parallelism) to BlueGene/L (at 212,992 way parallelism). We know how to do this effectively. This is difficult task but brings only medium risk and low to medium impact to the program: because if we achieve only a significant fraction of this goal, we can still make progress with the scientific discovery requisite for understanding Boost.

The most difficult risk comes from the needs of those IDC simulations that require more than 1GB of memory per core. We anticipate that Sequoia will have roughly 1GB per core. However, this risk shows a high probability of occurrence but presents only a medium to low impact to the program. This is the case because a great deal of work can be accomplished with the IDC codes running UQ with 1GB or less of memory per MPI task. For the balance of the runs, we will need to employ multiple cores/threads per MPI task or we can idle cores/threads to use their memory. The latter is a risk reduction strategy that lessens the impact of this risk at the expense of wasting ½ half of the peak speed of the nodes employed in a run. Given that much of the cost of the machine is sunk into memory and interconnect and far less into cores, this trade off, while painful, is a wise one. This buys the program the time to refine technology and improve efficiency even as computational work proceeds and mission deliverables ensue.

The risk mitigation strategies described position the program for success based on realistic, but conservative, estimations of what can be accomplished with Sequoia. We employed similar "undersell expectations" strategies for BlueGene/L and we were able to achieve not only the initial low expectations, but significantly exceed these with both the WS and IDC codes.

The challenges of multicore is a key risk associated with this procurement - and for this reason we have adopted the strategy above in conjunction with the sophisticated and extensive benchmark suite to evaluate the ability of our future vendor partner to identify and address this issue. We have three additional and complementary major risk mitigation strategies for dealing with adding SMP parallelism. In fact, the Sequoia procurement, deployment and programmatic usage model approach are each based on or represent components of our risk mitigation strategies. First, we have set a range of goals for the usage of Sequoia (12x-24x on IDC and 20x-50x on WS) and plan for a very long partnership (3 years prior to delivery and 5 years after acceptance) with the selected vendor to work together in partnership on implementing the above "nested node concurrency model" and using it effectively in WS and IDC. This extended period will allow the partnership several (3-5) software release cycles to get it working and then get it working well. So we expect to achieve the lower end of the performance range early in the life of Sequoia and the higher end of the performance ranges towards the end of life. The separate $12M R&D contract associated with the winning vendor *may well be heavily focused on this area*, but this is subject to discussion with the winning vendor to determine whether or not this scope of work is covered elsewhere in the procurement or other contracts. Second, we are partnering with 10 vendors including Intel (with their vast compiler, runtime and performance tool efforts), RedHat (who has significant gcc compiler and runtime development resources), Sun (with their compiler, runtime and debugger efforts) and UC Berkeley ParLab (http://parlab.eecs.berkeley.edu/) on the Hyperion project. Hyperion is now deploying a 100 teraFLOP/s, 1,152 node, 9,216 core, 18,432 Core+HT system. This system will be used to develop, test at scale and debug, key Linux cluster technology including compilers and runtime systems for multi-core among other technologies (e.g., Lustre and IO platforms). IBM, PGI and PathScale (now SiCortex) have all expressed interest in joining the Hyperion project. They all view the Labs in general and LLNL in particular as a good partner playing an independent 3rd party role that can help the market competitors find a workable solution. We have had partnership formation discussions along the lines of having the collaboration work on common interfaces and functionality for compiler extensions, runtime and OS extensions for multi-core and allow the vendors to innovate on the HW level and software implementation. By working in partnership with our Sequoia partner and Hyperion Project partners, we will be able to identify a multi-core solution that works with our benchmarks, IDC and WS codes to end up with a solution that should be ubiquitous. The third major risk reduction strategy is to focus initially on a small set (5-10) of first wave WS codes and scale these up to O(1M) parallelism and

then use lessons learned from that experience to bring the rest of the WS codes up to that level. This risk mitigation strategy was quite effectively employed on BlueGene/L. The first wave applications are chosen because: 1) they have the potential to scale well (with considerable effort in algorithm and code development); 2) will have high impact to the program if they are successful and; 3) the code groups have demonstrated ability to scale codes and are willing to put up with being part of the first wave in order to get a lot of early access to the system. It was three of these efforts that won successive Gordon Bell prizes. The first was MD achieving 100/360 of the machine peak (freezing of Tantalum under pressure to measure grain size) and the second was electronic structure achieving 200/360 efficiency on the system. A third GB prize was won employing MD for for Kelvin Helmholtz instability in which hydro scale phenomena were viewed from a simulation done on the atomic scale, in 2D and 3D. Here the efficiency was only ~100/590 on MD with a far higher ratio of communication to computation than the first GB effort. Each of these studies required sophisticated partnering work with the vendor and the development of machine level accommodations that made their way back into the system environment. Such is at the core of a successful partnership. Sequoia will be based on this model, but the effort will be formally funded and carefully managed from 2009 until 2016.

There are other highly detailed, tried and tested strategies employed in the procurement that stray from the multicore focus of this paper. This includes a negotiation with the vendor at contract time that defines "targets", which are firmed up in some systematic manner when a prototype system sits on the floor. This allows a vendor to "sign up" for aggressive deliverables before they are sure they can deliver. In the end, the deliverables (in aggregate) are met through the firmed-up requirements, even if some of the earlier targets are missed. This strategy both lowers cost and encourages vision.

With these mitigation strategies we have significantly reduced, arguably to low, both the probability of occurrence of these risks and the impact to the program should these eventuate. Reducing these risks does not mean that we can ignore these nor stop executing the mitigation strategies. Lowered risk depends on working these strategies on a daily basis over an extended period of time and investing significant resources in the effort, as was mentioned earlier. LLNL has, with both the Purple and BlueGene/L efforts (and White and Blue-Pacific before them), demonstrated the ability to keep its eyes on the ball for an extended period of time while not becoming unnecessarily distracted by periodic unrelated problems that would never derail the project. LLNL has also repeatedly shown the ability to identify root cause, plan and move on when adversity. We are certain that problems will occur during Sequoia development, build up, deployment and usage and we will be watching closely for early signs of trouble in order to effectively to derive the best possible outcome from unpleasant choices. These problems can emerge from expected risk areas, like threading and multicore, or they can (and usually do) come from unexpected directions and at high velocity. However, our experience in working a partnership with vendors has always allowed lemonade to be made from lemons, and it is both our hope and our expectation that Sequoia will offer challenges that will ultimately yield to early planning, careful thought, experience and partnership.