



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Coarse and Fine Grain Parallelism Performance Exploration in Ares

M. R. Collette, I. Karlin

January 16, 2013

NECDC 2012
Livermore, CA, United States
October 22, 2012 through October 26, 2012

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Coarse and Fine Grain Parallelism Performance Exploration in Ares (U)

Mike Collette¹, Ian Karlin¹

¹*Lawrence Livermore National Laboratory, Livermore, CA*

ASC's newest machine, BlueGene/Q Sequoia presents significant scalability challenges, because it has over an order of magnitude more processors than any previous ASC machine. In addition, hardware threads add the ability to run four MPI tasks or OpenMP threads per core resulting in a 44x increase in parallelism over the previous largest machine Cielo. However, BG/Q contains only 16 GB of memory per node leaving just 1 GB per core or 256 MB per hardware thread, leading to memory constraint challenges in addition to the parallelism challenges. In this paper, we look at how a subset of Ares scales on BG/Q and compare its performance to other current and past ASC platforms, such as, Cielo, BG/P, Zin and Purple. A radiation diffusion problem and an advection problem are used for this comparison. Weak scaling studies of these two problems show that BG/Q scales unprecedentedly well when compared to historic ASC machines. We then show that current implementation using MPI and domain level threads leaves potential performance on the table, for BG/Q nodes, regardless of ability to fit within memory. Finally, we demonstrate that by using loop level threading we achieve better performance than either pure MPI or using MPI and domain level threading.

(U)

Introduction

Emerging computer technologies present numerous challenges for high performance software application developers [1]. Power constraints are causing stagnant or declining clock speeds, while machines are gaining capability by increasing the number of cores per node [2]. Parallelism within a core is also growing as many cores now contain single instruction multiple data (SIMD) units and multiple hardware execution threads capable of sharing the execution pipeline [3, 4]. Also, power requirements are causing the amount of memory per core to decline, making large scale data replication impractical [5]. When these nodes are combined into large systems they present an unprecedented challenge to programmers who need both good weak scaling between nodes and strong scaling on a node to efficiently use system resources.

The BlueGene/Q platform [6] of which Sequoia at LLNL is the largest installation has many of these design points included in it. With 1.6 GHz cores its clock speed is significantly slower than the previous premier system, ASC Cielo. BG/Q has the same number of cores per node, but has over eleven times the

UNCLASSIFIED

number of nodes. It also has four hardware threads per core leading to 64 way parallelism on a node. However, it only has 16 GB of memory per node, 1 GB per core or 256 MB per hardware thread. With over 1.5 million physical cores and 6 million hardware threads Sequoia is the first ASC system to contain a large amount of on-node parallelism, limited memory per core and a large number of nodes simultaneously.

In this paper, we examine how Sequoia compares to current and past large machines in terms of overall performance, node level performance and weak scalability using two problems run on the Ares subset `bigadv3d` and `zrad3d`. These are an advection stress test and a radiation diffusion problem. Our scalability analysis looks at MPI everywhere and MPI tasks plus domain level threads. It shows that MPI everywhere produces superior performance on BG/Q compared to combining MPI with domain level threads, which is different than on other architectures. We also, discover that despite being the largest system we have ever run on, its weak scalability is better than any system we have run on in the past.

After our weak scaling analysis we look at how best to strong scale our problem within a node on BG/Q. We determine that the optimal point of running pure MPI, when strong scaling the MPI within a node, does not use all the hardware threads. Therefore, we investigate using hardware threads to accelerate the MPI and show this leads to an additional 5% runtime reduction. Finally, we present scalability data of our hybrid code and compare its performance to both pure MPI and Domain level threads.

In particular this paper makes the following novel contributions:

- We show that the MPI library and network of the BG/Q architecture are extremely scalable relative to historical machines with MPI overhead increasing from only 8% at a small scale to 12% when scaled out across over 1.5 million tasks.
- We demonstrate in a memory constrained world that low level threading is necessary to get the best performance out of our code. Even with our nicely scaling MPI we are not able to use all resources effectively without loop level threading.
- We show how our combined code outperforms our baseline achieving up to 5% better performance in hybrid mode with loop level threading than either pure MPI or MPI plus domain level threading on 16 and 32 nodes of BG/Q.

The rest of this paper is organized as follows: Section 2 describes the current parallelism available in Ares and looks at the performance and scalability of the current Ares code using both pure MPI and MPI plus domain level threads. We also include a comparison of scaling performance on BG/Q to other past ASC machines and show the limitations of strong scaling the MPI on BG/Q. Next Section 3 presents our work threading the `zrad` and `bigadv` problem at the loop level and analyses how to best thread. Section 4 analyses the combined performance of our loop level threaded code to find the optimal performance for it. Finally, Section 5 presents conclusions and future work.

Performance of Current Ares Parallelism

UNCLASSIFIED

Currently Ares contains domain level parallelism implemented in two ways, OpenMP and MPI. When launching a job the user can launch a pure MPI job or have each MPI rank assigned multiple domains, each of which is operated on by an OpenMP thread. In this section, we examine the performance of the current parallelism in Ares. First we present two test problems and the machines we have run them on. We then look at how well the code weak scales using just MPI on some of the largest systems past and present within the DOE complex. Next, we examine the performance tradeoffs of swapping MPI processes for Domains on each of these machines and determine the optimal tradeoff at various scales. Finally, we look at the limitations of this approach in terms of using all available system resources on our current flagship machine Sequoia and the implications for future machines.

Test Environment and Methodology

| Machine | Processors | CPU Speed | Lightweight Kernel | Memory/cores per Node | Peak |
|--------------|------------|-----------|--------------------|-----------------------|----------|
| Purple | 12,288 | 1.9 GHz | No | 32 GB / 8 | 0.09 PF |
| TLCC2 Zin | 46,656 | 2.6 GHz | No | 32 GB / 16 | 0.97 PF |
| Cielo XE6 | 143,104 | 2.4 GHz | Yes | 32 GB / 16 | 1.37 PF |
| Dawn BG/P | 147,456 | 0.85 GHz | Yes | 4 GB / 4 | 0.50 PF |
| Sequoia BG/Q | 1,572,864 | 1.6 GHz | Yes | 16 GB / 16 | 20.14 PF |

Table I: Hardware characteristics of selected past and present ASC machines.

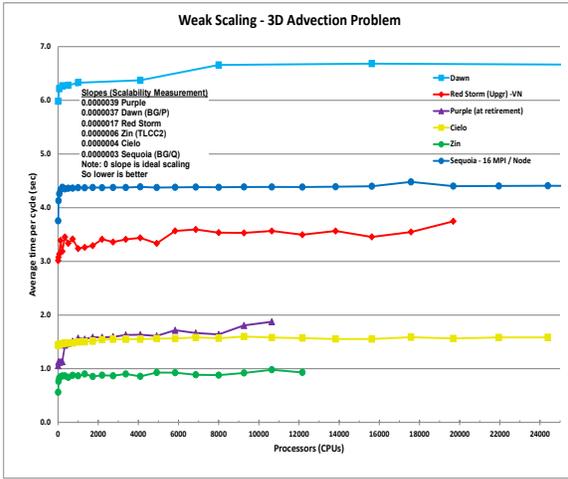
Tests were run on two test problems, bigadv3d and zrad3d on current and historical systems. Some systems of note are listed in Table I along with key hardware features of these machines. In all cases the problems were run with 25^3 elements per domain.

The two test problems were selected because they show two different communication patterns that result in different scaling performance characteristics and a wealth of historical data is available for each. The bigadv3d problem is an advection stress test with only nearest neighbor stencil communication. Therefore, it performs 26 communications to its neighbors per timestep. The zrad3d problem solves a linear system and requires a varying number of iterations as it scales. It requires intense all-reduce collective messages that stress the interconnect in different ways.

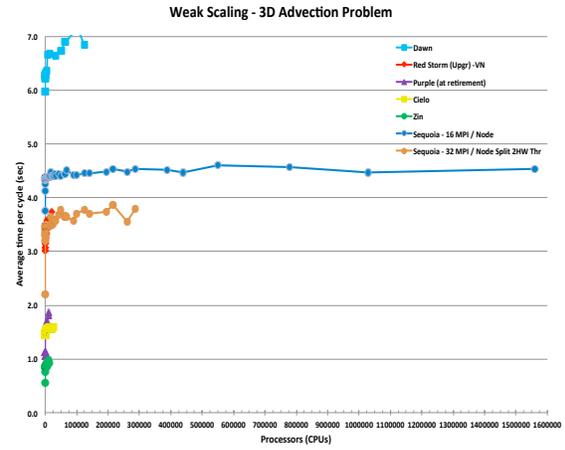
Weak Scaling

Weak scaling studies are useful at showing how communication costs vary at different processor counts. In this section we compare the weak scaling of our two test problems on various historical machines. All the tests in this section were run using pure MPI only.

Figure 1 shows how the advection test problem scales across multiple historical machines. As seen in the figure other than a small performance decrease when internode communication is introduced at small scale all machines scale similarly. While Sequoia is significantly better on our measure of scalability it is barely ten times better than the worst machine and the differences do not impact the performance of any of the other systems noticeably at scale.

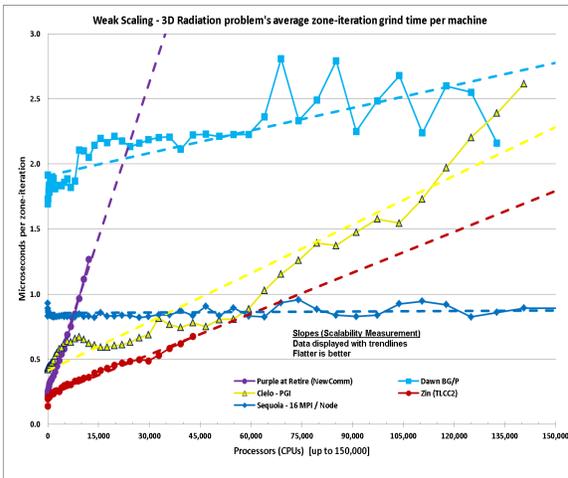


(a) Small Scale

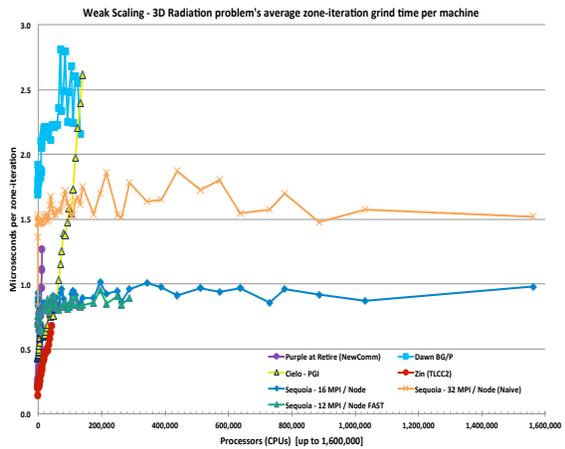


(b) Large Scale

Figure 1: Historical performance of the bigadv test problem.



(a) Small Scale



(b) Large Scale

Figure 2: Historical performance of the zrad3d test problem.

Figure 2 shows how the zrad3d test problem scales across multiple historical machines. As seen in the figure the zrad problem historically has not scaled well with a factor of three or more performance degradation occurring when scaling from a single node to the whole machine on Zin, Cielo and Purple. It should be noted though that the newer machines, Zin and Cielo improve on Purple by over a factor of six in terms of their scalability allowing them to scale to significantly larger processor counts than Purple. Sequoia, however, is in a class of its own in terms of scalability of the zrad problem. It scales nearly perfectly out to a million MPI tasks with communication time only increasing from 8% to 12% of overall runtime.

Overall Sequoia's performance at a small scale is worse on a per core basis than all historical machines we compare it against other than Dawn. However, the scalability of Sequoia is significantly better resulting in the performance of the zrad problem to be better than any other machine on 75,000 or more cores. The results from these two problems show that while all ASC machines have been able to scale the bigadv problem well, Sequoia is the first machine that is able to efficiently weak scale the zrad problem.

Performance and Tradeoffs of Domain Level MPI and Threads

In the previous section we looked at the performance of weak scaling the MPI communication of our two test problems. In this section we examine the tradeoff of using OpenMP threads instead of MPI tasks for some domains.

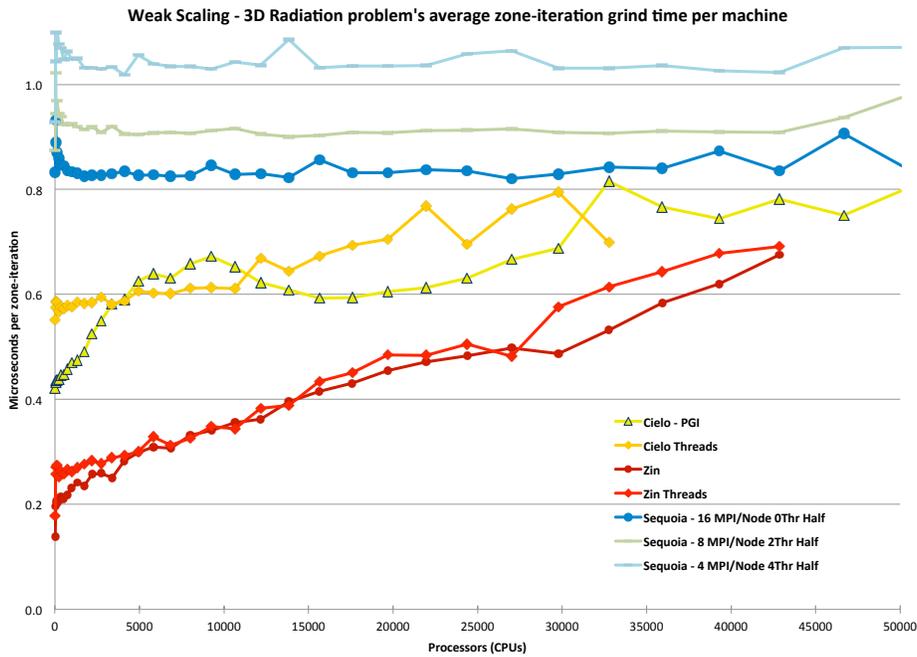


Figure 3: Domain level threading vs. pure MPI.

UNCLASSIFIED

Figure 3 shows the performance tradeoff of using OpenMP threads and MPI tasks to handle domains vs. pure MPI. It shows that using domain level threads results in better performance for some problem sizes on both Cielo and Zin, but never on Sequoia. For most machines there is a point where a hybrid MPI plus OpenMP approach begins to outperform a pure MPI approach. On each machine the point where switching to a hybrid approach is best is different and sometimes changes back at various scales, but until Sequoia there was always a processor count where using OpenMP threads for some domains was superior to a pure MPI approach. On Sequoia, however, the performance of using hardware threads instead of domains is never better than using a pure MPI approach. The lack of a crossover point is despite the fact that OpenMP overheads on BG/Q are lower than on the other systems. However, this is probably because the MPI overhead is reduced even more than the OpenMP overhead.

Limitations of Current Approach

While historically it has been possible to weak scale problems to all computational resources Sequoia presents us with the first machine where this is not a feasible approach. With only 1 GB of memory per core or 256 MB per hardware thread to effectively use all the compute resources on the machine we need to strong scale our problems. In this section we investigate how strong scaling the pure MPI version of Ares uses hardware threads on Sequoia. We choose node and task counts that keep the overall problem size constant. By keeping the overall problem size constant we remove potential cache effects and isolate the added costs of additional ghost zones. Also, for the zrad problem we keep the iteration count needed to solve the linear system constant and, therefore, do not introduce an artificial work imbalance.

| Nodes | Tasks | Zones per Domain | Runtime (s) | Communication Time (s) |
|-------|-------|------------------|-------------|------------------------|
| 16 | 512 | 20^3 | 9.19 | 0.660 |
| 16 | 1000 | 16^3 | 9.38 | 0.997 |
| 32 | 512 | 25^3 | 12.40 | 0.761 |
| 32 | 1000 | 20^3 | 9.24 | 0.697 |

Table II: Strong scaling of bigadv on Sequoia.

| Nodes | Tasks | Zones per Domain | Runtime (s) | Communication Time (s) |
|-------|-------|------------------|-------------|------------------------|
| 16 | 512 | 20^3 | 24.32 | 5.12 |
| 16 | 1000 | 16^3 | 27.00 | 7.06 |
| 32 | 512 | 25^3 | 34.35 | 5.42 |
| 32 | 1000 | 20^3 | 30.14 | 6.23 |

Table III: Strong scaling of zrad3d on Sequoia.

Tables II and III show the results of strong scaling the MPI on a fixed number of nodes. Both tables show that performance peaks at 32 MPI tasks per node. However, there are 64 hardware threads on a BlueGene Q node and only using 32 of them leaves resources idle. One reason performance degrades is the amount of time spent communicating data increases as we use more MPI tasks per node as shown in the last column of Tables II and III. Having more small domains increases the surface to volume ratio, which both increases the number of messages and the overall aggregate amount of data communicated.

UNCLASSIFIED

Another potential reason for the runtime increase is the coarseness of the threading approach currently used. The current approach of using only MPI and Domain level threads limits the ability to take advantage of fine grain parallelism within the code. Loops are either threaded at a high level or are run pure MPI. However, within a high level loop over a domain there can be many sub-loops with different computational characteristics that are better suited to different threading approaches. The next sections discuss in more detail the computational characteristic portion.

Loop Level Threads

There are multiple reasons for adding loop level threading to a code. Low level threading allows fine grain control of the amount of parallelism in each region. Not all loops benefit from being parallelized and loop level threading allows us to not thread those loops. Threading below the domain level also allows each domain to be larger in size improving its surface to volume ratio and decreasing the amount of data that needs to be communicated. However, using lower level threads increases overhead by creating more OpenMP regions. It also requires significantly more programming effort than domain level threads. In this section we describe the threading work we performed and its initial impact on performance.

Implementation

In Ares we tried adding loop level threads to all thread safe loops executed by the zrad3d and bigadv3d problems. We tested the performance gains from adding the threads by running on 4 nodes with 16 MPI tasks per node on BG/Q varying the number of threads per task from one to four. We then removed threading from any loops where threading did not increase performance. We did not thread any thread unsafe loops because the extra overhead introduced to make them thread-safe would likely be greater than the gains from threading when threads are only utilizing the hardware threads. Also, the loops we did thread show what we were hoping to see from this investigation; that loop level threading produces better performance than domain level threading on BG/Q.

Overall we tried threading eight loops in the zrad problem and 31 loops in the bigadv one. We did not thread one loop in zrad and five in bigadv due to safety concerns. Of the loops we tried threading we removed the threading from four in the zrad problem and three in the bigadv problem as they did not reduce runtime when threaded. We also did not try to thread numerous small loops that had insignificant runtimes.

With one exception we stopped at just threading loops and seeing if performance improved. The exception is a loop that goes over 55 arrays that is the largest portion of the runtime of the conjugate gradient solver in the zrad problem. For this loop special versions have already been created for BlueGene and x86 architectures and its performance significantly slowed down when threaded. First we determined that when sixteen MPI tasks per node were used and the loop was not threaded that the BlueGene specific version was best. Therefore, we started with it and used it to improve the threaded version.

The loop itself performs two flops per array reference and is memory bound. The tuned version for the BG/L and BG/P architectures looped over thirteen of the arrays per iteration. On those machines there were fourteen stream prefetchers per core while on BG/Q there are sixteen. However, on BG/Q the stream

UNCLASSIFIED

prefetchers are shared among the four hardware threads per core. So when threading we needed to further fission the loops so that there were only seven arrays accessed per loop when using two threads and three arrays accessed per loop when using four threads. The result was performance equal to the unthreaded optimized code. Therefore, we concluded that for this memory bound kernel maximum bandwidth can be achieved without using the hardware threads.

Performance Analysis

| Threads | Loop 1 | Loop 1 & 2 | Loops 1 - 3 | Loops 1 - 4 |
|---------|--------|------------|-------------|-------------|
| 1 | 18.20 | 18.19 | 18.16 | 18.23 |
| 2 | 17.46 | 17.08 | 16.84 | 16.70 |
| 3 | 17.22 | 16.81 | 16.38 | 16.25 |
| 4 | 17.17 | 16.74 | 16.31 | 16.19 |

Table IV: Zrad problem runtime on 4 Nodes using 64 MPI tasks and a 25^3 domain per task.

Table IV shows the performance impact of threading the four profitable loops for the zrad problem on 4 nodes using 64 MPI tasks on Sequoia. It shows the overall performance of the calculation with successively more loops threaded and run on increasingly more threads. Overall, performance gains result in an 11% runtime reduction when all four loops are threaded and four hardware threads are used. The speedup was small due to the loop over 55 arrays in conjugate gradient solver being the largest cost in the routine. It is also noteworthy to point out that the first loop that was threaded resulted in about half the performance gain from threading this problem.

An analysis of the the bigadv problem reveals similar results. Despite being able to profitably thread 28 loops the loop with the largest performance gain from threading produced 48% of the runtime gains. Also, the five loops with the largest runtime decrease totaled about 90% of the overall runtime reduction. Overall there was a 22% runtime reduction from threading for the bigadv problem. However, the threaded code was about 2% slower when the threads were not used.

A more detailed analysis of how threading sped up individual loops revealed some noteworthy characterizations. Overall 21 loops performed best when using four hardware threads, while five were best when using three and two were best when using two threads. The runtime savings of the loops that were threaded was 41% of their runtime and an additional 2.5% reduction can be gotten by using the optimal thread count for each loop rather than four threads for all.

Finally, a look at the loops that scaled the best revealed the following characteristics. They often contained many branch statements or tightly coupled dependencies where the execution of one instruction depends on the result of a recently executed instruction. Following those loops in scaling well when using the hardware threads are compute intensive loops. That these two types of loops scale well on the hardware threads is not surprising as the threads are meant to hide latency, which is prevalent in many events in the first set of loops. Also, while the BlueGene Q processor can issue two instructions from a single thread in a cycle it can only retire one instruction from a thread. Therefore, in order to retire instructions at the same rate as it issues them two threads must be performing work. For compute heavy loops then there are extra

UNCLASSIFIED

resources made available by using hardware threads. Also, of note is that memory bound loops and loops with small runtimes scaled the worst. The small loops most likely suffer from thread launch and sync overhead and the memory bound ones can use all available bandwidth with one thread per core.

Combined Performance

In this section we examine how loop level threading improve the performance of Ares. We also try and determine the best mix of MPI tasks and loop level threads on Sequoia.

| Nodes | Tasks | Threads per Task | Zones per Domain | Runtime (s) |
|-------|-------|------------------|------------------|-------------|
| 16 | 1000 | 1 | 16^3 | 9.39 |
| 16 | 512 | 1 | 20^3 | 9.19 |
| 16 | 512 | 2 | 20^3 | 8.75 |
| 32 | 1000 | 1 | 20^3 | 9.24 |
| 32 | 1000 | 2 | 20^3 | 8.83 |
| 32 | 512 | 4 | 25^3 | 9.91 |

Table V: Threading of bigadv on Sequoia.

| Nodes | Tasks | Threads per Task | Zones per Domain | Runtime (s) |
|-------|-------|------------------|------------------|-------------|
| 16 | 1000 | 1 | 16^3 | 27.4 |
| 16 | 512 | 1 | 20^3 | 24.1 |
| 16 | 512 | 2 | 20^3 | 23.4 |
| 32 | 1000 | 1 | 20^3 | 29.9 |
| 32 | 1000 | 2 | 20^3 | 29.0 |
| 32 | 512 | 4 | 25^3 | 29.3 |

Table VI: Threading of zrad on Sequoia.

Tables V and VI show representative performance data of tests run to determine the optimal thread vs. MPI task tradeoff. For both problems the results show that the best performance is found by using 32 MPI tasks on each node and 2 OpenMP threads per task. However, as we move to more nodes the performance of the code using four threads per MPI task starts to approach the performance of using two threads per task. In particular, the performance of the zrad problem is converging quicker than the bigadv. That the zrad problem is converging faster is expected because its communication cost grows faster than the bigadv problem as it scales. Therefore, for an at scale coupled physics simulation these results suggest the best performance might be found at a suboptimal task/thread tradeoff.

Conclusions and Future Work

In this paper, we presented a historical view of ASC machine performance on two test problems. We showed that all past machines have resulted in good weak scalability for the bigadv test problem that primarily performs nearest neighbor stencil communications. However, for the zrad3d problem that stresses the network more than bigadv only the brand new Sequoia computer was able to scale out to

UNCLASSIFIED

nearly all of its nodes without significant weak scaling performance degradation. However, we showed that while Sequoia exhibits better weak scaling characteristics than previous machines it presents significant strong scalability challenges. In particular, it requires fine-grain threading to take advantage of all hardware resources and extract the best possible performance from the hardware.

Since hardware threads and less memory per core are the architectural trends of the future we are going to build on this work in two ways. We plan to investigate how loop level threading impacts performance at large scales and the scalability of the code. In particular, we are interested in whether or not the runtime gains from loop level threading stay constant became larger as we scale up to larger machines. Also, we plan to use what we have learned from these test problems to begin exploring a robust way to put loop level threading into Ares. Included in this investigation will be how to best put in nested threading so we can take advantage of both domain level threading and loop level threading when using both results in the best performance.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.0 (LLNL-CONF-610872)

References

- [1] Vivek Sarkar, William Harrod, and Allan E. Snavely. Software challenges in extreme scale systems. In *Journal of Physics: Conference Series*, vol. 180, no. 1, p. 012045. IOP Publishing, 2009.
- [2] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. *High Performance Computing for Computational Science-VECPAR 2010 (2011)*: 1-25.
- [3] Angela C. Sodan, Jacob Machina, Arash Deshmeh, Kevin Macnaughton, and Bryan Esbaugh. Parallelism via multithreaded and multicore CPUs. *Computer* 43, no. 3 (2010): 24-32.
- [4] Roland Leia, Sebastian Hack, and Ingo Wald. Extending a C-like language for portable SIMD programming. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, pp. 65-74. ACM, 2012.
- [5] Kshitij Sudan, Karthick Rajamani, Wei Huang, and J. Carter. Tiered Memory: An Iso-Power Memory Architecture to Address the Memory Power Wall.” *Computers, IEEE Transactions on* , vol.61, no.12, pp.1697-1710, Dec. 2012
- [6] Megan Gilge. IBM System Blue Gene Solution: Blue Gene/Q Application Development. IBM Redbook Draft SG24-7948-00 (2012).