Simulating Laser-Plasma Interaction in Experiments at the National Ignition Facility on a Cray XE6

Steven Langer*, Abhinav Bhatele*, Todd Gamblin*, Bert Still*, Denise Hinkel*,
Mike Kumbera*, Bruce Langdon*, Ed Williams*
* Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore CA 94550
{langer1, bhatele, tgamblin, still1, hinkel1, kumbera1, langdon1, williams16}@llnl.gov

***Abstract*—The National Ignition Facility (NIF) [1] is a high energy density experimental facility run for the National Nuclear Security Administration (NNSA) by Lawrence Livermore National Laboratory. NIF houses the world's most powerful laser. The National Ignition Campaign (NIC) has a goal of using the NIF laser to ignite a fusion target by the end of FY12. Achieving fusion ignition in the laboratory will be a major step towards fusion energy.**

NIC is currently considering several possible ignition target designs. The NIF laser can fire a limited number of shots, so simulations play a major roll in selecting the designs to be used in experiments.

The NIF lasers beams reach intensities of $10^{15}\mathrm{W/cm^2}$ in spots. That is high enough that interactions between the laser beams and fluctuations in the density of the ions and electrons may scatter laser light away from the target.

pF3D [2], [3], [4] is a laser-plasma interaction code used to assess proposed experimental designs for expected levels of scattering and to help understand measurements of scattered light in NIF experiments.

NIF experiments have shown that laser-plasma interactions transfer significant amounts of energy between beams and increase the amount of backscattered light relative to what would occur without energy transfer. pF3D has run several simulations with two interacting beams and is starting to run simulations with three interacting beams. These simulations require over 200 billion zones and run for several weeks. The pF3D simulations presented in this paper were run on Cielo, a Cray XE6 at Los Alamos National Laboratory. These simulations help us understand key experiments currently being carried out with the NIF laser.

This paper reports on several modifications we have made to pF3D in the past year. These changes help pF3D run better on Cielo and they are also a step in preparing for future exascale computers.

I. Introduction

The National Ignition Facility (hereafter NIF; [1]) is a large NNSA experimental facility that houses the world's most powerful laser. One of the key goals of the NIF is to compress a target filled with deuterium and tritium to a temperature and density high enough that fusion ignition occurs. Figure 1 shows the laser beams entering a hohlraum through holes in both ends and propagating through the plasma (ionized gas) until they reach the hohlraum wall. The laser beams deposit their energy and the wall becomes hot enough that it radiates x-rays. The x-rays fall on the surface of the capsule at the center of the hohlraum and cause it to implode.

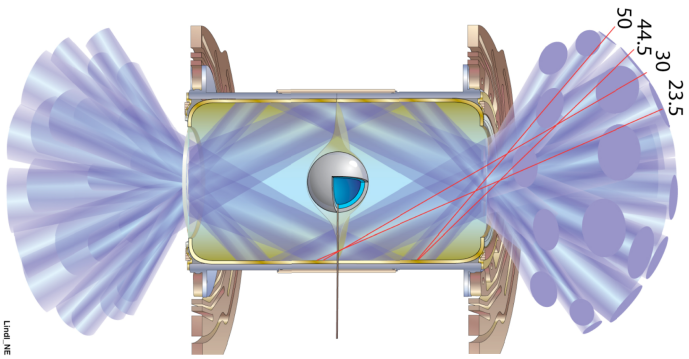


Fig. 1: The figure shows the laser beams entering the hohlraum through holes in both ends and propagating through the plasma until they reach the hohlraum wall. The hohlraum is a can-like object used to trap the x-rays emitted by the hot interior walls of the hohlraum.

There are 192 beams in 48 beam clusters. and they heat the plasma to a few tens of millions of degrees. The laser intensity in a NIF hohlraum is high enough that laser-driven plasma instabilities may grow to significant levels. In particular, it is possible for some of the laser light to be backscattered and leave the hohlraum without heating the plasma.

pF3D is a multi-physics code used to simulate interactions between laser beams and the plasma in NIF experiments. pF3D is used to evaluate proposed hohlraum designs and help pick the ones with acceptably low levels of backscatter.

Exascale systems are expected to differ from current systems in several ways. The floating point performance will grow faster than other system characteristics. The ratio of the memory size, the memory bandwidth, the interconnect bandwidth, and the file system bandwidth to the floating point performance will all decrease. The mean time between application interrupt for full system simulations will decrease relative to current systems. In all cases, the decrease is expected to be big enough that significant modifications to applications will be required to avoid a drop in application throughput relative to peak performance.

Most of these trends are already apparent in systems coming online today, but they are small enough that they have not yet forced major rewrites of most existing multi-physics codes. We

reported on our initial experience in tuning pF3D to run well on Cielo, a Cray XE6 at Los Alamos National Laboratory, at this conference in 2011 [4]. This article reports on additional modifications that allow pF3D to work better on Cielo and discusses how this work relates to our preparations for future exascale systems.

Resolving the interactions between beams requires the use of zones that are less than a laser wavelength (0.35 μm) across. The plasma is several mm across, so simulations require many billions of zones. We have completed three simulations of two interacting beams using 32K Cielo processors. These simulations used up to 225 billion zones. We are preparing to simulate three interacting beams using 64k or 96k Cielo processors. Three beam simulations will require 400-500 billion zones.

The remainder of our paper is organized as follows. In Section II we give a general description of pF3D and its communication patterns.

Section III presents results on message passing performance. Message passing can consume a significant fraction of the run time and controls the scaling of pF3D. Our results show that the mapping between physical domains and position on the Cielo interconnect has a significant impact on performance.

Section IV discusses our work on adding OpenMP parallelism to the MPI parallelism already present in pF3D. This hybrid form of parallelism should permit higher message passing rates than would be possible using pure MPI on systems with low memory per core.

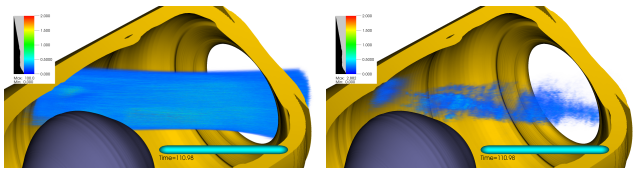
Section V discusses a fault tolerance approach named SCR that has been incorporated into pF3D [5], [6]. A system like Cielo has a very large number of components so it is no surprise that the mean time between interruption (MTBI) of pF3D is short enough that we must have a resiliency strategy. pF3D periodically writes the state of the simulation to storage to permit recovery in the event of a system error. SCR uses RAM disks or other local storage to permit rapid writing and reading of checkpoints. Checkpoints are occasionally written to a parallel file system. This approach reduces time spent on defensive I/O and thereby allows pF3D to produce more results per day.

Section VI shows some of the results from our simulations and compares them to data from NIF experiments. The simulations we have run on Cielo are helping us gain a more detailed understanding of how multiple beams interact to produce backscattered light.

In section VII we state our conclusions and discuss future work.

II. pF3D Overview

pF3D [2], [3] is a multi-physics code that carries out time-dependent simulations of the interaction between a laser beam and a plasma in experiments being carried out at the National Ignition Facility (NIF - http://lasers.llnl.gov). pF3D solves wave equations for the laser light and two kinds of backscattered light. The light waves are coupled together through interactions with electron plasma waves and ion acoustic waves. Figure 2 shows the laser beam and the backscattered light from a pF3D simulation.



(a) (b)

Fig. 2: The figure on the left shows a NIF laser beam (blue) propagating through a hohlraum. pF3D solves the laser-plasma coupling equations in a box surrounding the beam. The hohlraum (gold) is shown to provide context but is not part of the simulation. The capsule (dark blue) that will be imploded by the x-rays generated at the hohlraum wall is shown at the lower left. In this simulation, the beam propagates from the hohlraum entrance on the right to the hohlraum wall at the left. The figure on the right shows the light that was scattered off of electron plasma waves. The scattered light is generated near the wall and propagates back through the entrance hole.

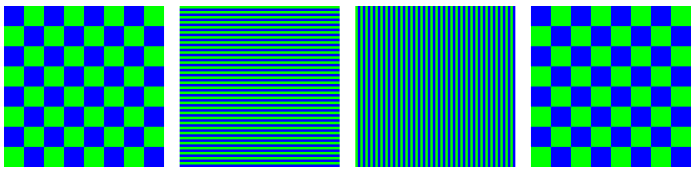


Fig. 3: The 2D FFT message passing scheme. MPI_Alltoall is used to transpose from a checkerboard decomposition, to decompositions by rows and columns, and back to a checkerboard. The FFTs are performed while the data is in rows or columns.

We present a brief overview of pF3D in this section. For more details, see [4]. pF3D solves its equations on a 3D Cartesian grid. The equations are solved in the paraxial approximation which assumes all light waves are traveling nearly in the z-direction (the laser propagation direction). The wave propagation and coupling are solved using Fourier transforms in xy-planes. The Fourier transforms require message passing across the full extent of an xy-plane. These messages are sent using MPI_Alltoall. Advection of the light and solving the hydrodynamic equations require passing planes of information to nearest neighbor domains using MPI_Send and MPI_Recv. The hydrodynamic equations are solved every 50 light time steps, so hydrodynamics consumes only a modest portion of the run time.

The pF3D grid is split into np-by-nq-by-nr domains in the x-, y-, and z-directions. A set of domains covering the full extent of the grid in x and y but extending only one domain in z is called an xy-slab. The messages passed to carry out a 2D FFT are confined to a single xy-slab.

Figure 3 shows a high-level overview of the message passing involved in performing a 2D FFT. A call to MPI_Alltoall

System	nxloc	nyloc	np	nq	xsize	ysize
Intrepid	320	96	32	32	7680	7680
Dawn	640	288	32	16	46080	92160
Cielo	640	192	16	16	61440	61440
Sequoia	160	96	64	32	1920	3840

TABLE I: This table shows the sizes of the domains for three recent pF3D runs and a possible run on Sequoia, a Blue Gene/Q system. The number of zones in a domain in a single xy-plane (nxloc*nyloc) and the number of processes into which the grid is decomposed (np and nq) in the x and y directions determine the size of the messages (xsize and ysize) sent during an MPI_Alltoall.

on x-communicators is used to transpose from the "checkerboard" decomposition to a state where each process has one or more rows of data. An ordinary single processor 1D FFT of length nx is performed on the rows. MPI_Alltoall is called to transpose back to the "checkerboard" decomposition. MPI_Alltoall on y-communicators is called to transpose to a state where each process has one or more columns of data. A 1D FFT of length ny is performed and MPI_Alltoall is called again to transpose back to the original "checkerboard" state. The message passing for the x-phase of the FFT is performed on nq*nr x-communicators simultaneously. The message passing for the y-phase of the FFT is performed on np*nr y-communicators simultaneously.

III. Message Passing Performance

pF3D can spend over 30% of its run time passing messages so optimizing message passing rates is important. Section II described the message passing in pF3D as it occurs in physical space. To understand how messages flow across the interconnect, we must know how the spatial domains are mapped into locations on the interconnect. The mapping can have a large impact on message passing performance on systems with mesh or torus based interconnects.

pF3D must run on very large numbers of processors, so the first requirement for any mapping from physical location to position on the interconnect is that it must be highly scalable. The majority of the message passing occurs during 2D FFTs, so it is important to choose a mapping that gives very good performance for FFT messages. The mapping must also give good performance for messages passed across faces to adjacent domains, but this is lower priority than the other requirements.

An interconnect delivers its full bandwidth only if the message size is greater than the product of the link bandwidth and its latency. pF3D requires high message passing rates, so we try to decompose the problem in such a way that we pass "large messages". The message size varies fairly rapidly as an xy-slab is decomposed into more domains.

Table I shows the number of zones in the x- and y-directions in each domain for recent large pF3D runs on Intrepid, Dawn, and Cielo. Intrepid and Dawn are IBM Blue Gene/P systems (hereafter BG/P) and Cielo is a Cray XE6. It also shows the number of domains into which an xy-plane is decomposed. The zones "owned" by each process are equally split amongst all processes in a row or column of the domain grid when performing an MPI_Alltoall. pF3D normally uses float complex variables that require 8 bytes per zone. The table shows the number of bytes per message if the Alltoall is implemented using point-to-point messages. The fourth row shows what the message sizes would be on Sequoia, an IBM Blue Gene/Q, for a pure MPI run using all 4 hardware threads per core. The number of zones per process was set to half that of the Intrepid run because Intrepid has 512 MB of memory per core and Sequoia has 256 MB per hardware thread.

The message sizes in row four are small enough that the effective bandwidth will be lower than it would be for large messages. A Sequoia run using 8 hardware threads per MPI process would have 2 GB of memory per process and would have the same large message sizes as the Cielo run.

The interconnect on BG/L and BG/P systems is a 3D torus, as is the interconnect on the Cray XE6. There is an important difference between these two systems. On a BG/L or BG/P a job runs on a private torus (or a mesh for small jobs). On an XE6, a job runs on a swath of nodes within the full torus. It is easier to reason about the impact of how processes are mapped from the 3D grid of domains in pF3D onto the private 3D torus on a Blue Gene system than it is to reason about the irregular set of nodes assigned to a job on an XE6. For this reason, we will start with a discussion of how we generate scalable, high-performance mappings on a Blue Gene. This includes a discussion of a tool we have written to easily generate a wide variety of mappings. We then discuss the message passing optimizations we employ on the XE6 and the impact of running on an irregular set of nodes. We report on the progress we have made in developing tools to generate good mappings for the XE6.

A. Blue Gene Systems

pF3D uses a regular 3D decomposition of the grid into domains and it runs on a private 3D torus on a BG/L or BG/P system. This makes it relatively simple to generate mappings between physical domains and locations on the torus.

A mapping will be (weakly) scalable if the hop count and level of contention for links remains the same as the number of xy-slabs is increased. A simple strategy for generating scalable mapping is to first divide the partition up into equal sized chunks with a number of cores equal to the number of processes in an xy-slab. The next step is to map xy-slabs that are adjacent in z to adjacent chunks on the torus. A job running with a fixed number of processes on a BG/L or BG/P will always get a partition of the same shape, so this mapping can be generated once and used for the entire run.

On Blue Gene systems, the default is to assign domains in "array order". In xyzt mode, the x-coordinate on the interconnect varies fastest and the "thread" coordinate varies slowest as the MPI rank increases. In zyxt mode, the z-coordinate on the interconnect varies fastest and the "thread" coordinate varies slowest as the MPI rank increases.

In the case of LLNL's BG/L system, large partitions are Nx32x32. If we run a simulation with np=nq=32 in xyzt mode,

an xy-slab is spread across an ever larger distance on the torus as the number of slabs increases. If we run the same simulations in zyxt mode, a 32x32 xy-slab maps to a 32x32 plane of the torus. Hop counts do not change as we add xy-slabs and there is no contention between FFT messages in different xy-slabs. Figure 4 (a) shows the message passing rate averaged over all communication phases as a function of the number of processes on a BG/L system using xyzt and zyxt mappings. The zyxt mapping is very scalable. The downside to this mapping is that it only uses one of the three sets of links during a given FFT messaging phase and is therefore slower than xyzt (which uses multiple sets of links) for small process counts.

We would like to find a mapping that is as scalable as zyxt, but which uses all three sets of links. We can do this by mapping an xy-slab into a 3D block of the interconnect. The blocks are placed on the torus so that domains which are next to each other in physical space are also adjacent on the interconnect. Messages passed in the z-direction are sent from the source block to the same position in the (adjacent) destination block. The number of links a z-message must cross does not change as the number of xy-slabs is increased.

Messages for a 2D FFT are confined to the block for that xy-slab. The message passing can use links in three directions in contrast to the zyxt method which only uses links in one direction. The messages are large enough that the extra bandwidth obtained by using all three pairs of links outweighs an increase in the number of hops for send/receive messages.

The scalable mappings discussed so far map an xy-slab in physical space to a plane or rectangular block of the torus. These mappings are simple enough that they can be generated by custom scripts written for a particular case.

We create more complex network mappings using Rubik, a tool designed at LLNL to create cartesian network mappings for applications with structured domain decompositions. Rubik allows the application developer to quickly specify groups within arbitrary-dimensional cartesian spaces and map them to potentially differently shaped groups in another cartesian space. Rubik also allows us to perform permutations of ranks within groups, e.g. the tilt operation discussed in this paper. Rubik is designed to perform mappings onto 3- and higher-dimensional torus networks used in large supercomputers like the IBM BG/P, BG/Q, and Fujitsu K machines. Cray XT/XE series systems have a toroidal interconnect, but applications do not get their own private torus as on Blue Gene systems or K. Future work will modify Rubik to work on the Cray XE/XK. We use Rubik to map the planes in pF3D's simulated physical domain to blocks and tilted planes on the hardware torus. Rubik is available on request from the authors of this paper.

Rubik can easily reproduce both of the scalable mappings described above. Rubik can also take one of these mappings and apply a "tilt" to it in one or more directions. Figure 5 shows how a simple partitioning of the BG/P torus can be turned into a tilted mapping. The 8x8x8 BG/P partition has been divided into 8 slabs on the left. Stopping at this point would lead to messages that only use one pair of links during a given message phase. Each slab is mapped to an xy-slab in physical space. The slabs have been tilted in two directions on the right. The mapping connects an xy-slab in physical space to a tilted plane on the interconnect. Messages passed within an xy-slab can use all three link directions with this mapping, producing a higher message passing rate.

Mapping	CPU time (s)	Msg time (s)	Msg rate
TXYZ	457.496	55.218	54.735
XYZT	450.012	23.495	128.639
Tiling	449.505	22.737	132.929
TiltZ	449.331	17.756	170.213
TiltZY	449.481	15.435	195.814

TABLE II: This table shows message passing rates on a BG/P system for five different mappings generated by Rubik.

Table II shows message passing rates on Intrepid for 5 different mappings generated by Rubik. The best mapping for this problem tilts the slabs shown in figure 5 in both the y and z directions. There is nearly a factor of 4 difference in performance between the slowest and fastest mappings.

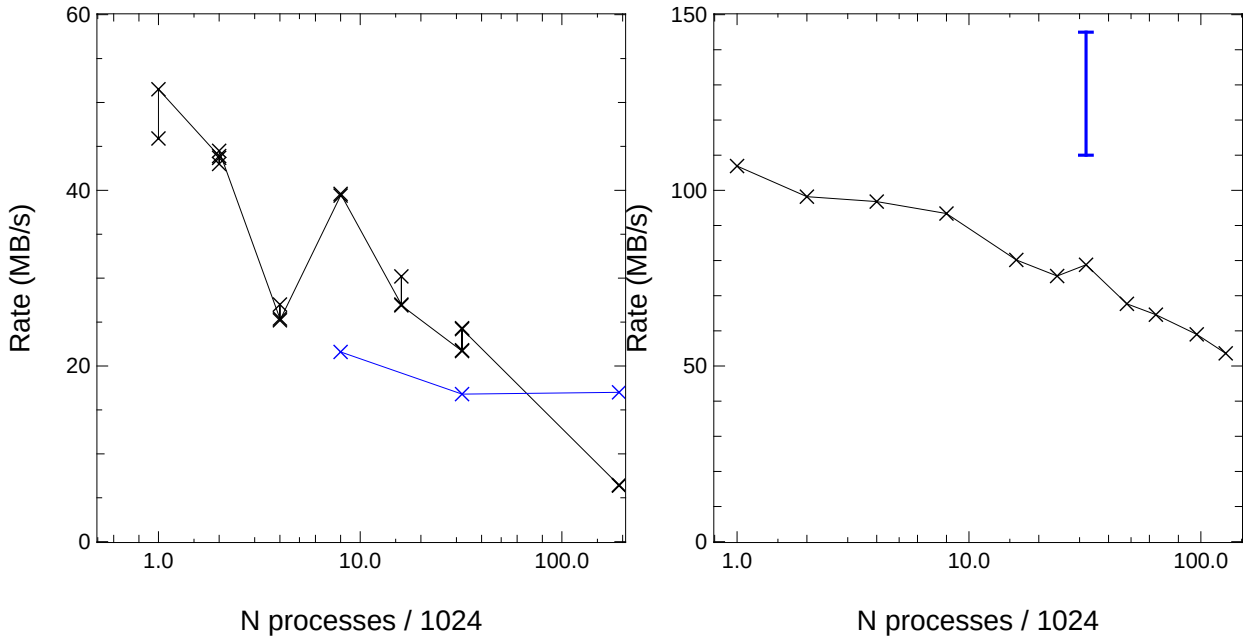
The tilted mappings perform very well in these test problems run on an 8x8x8 node partition. We are currently starting a three beam simulation on Dawn. This simulation has roughly 300 billion zones and runs on a 48x32x16 node partition. The default xyzt mapping produces a rate of 8 MB/s, which leads to the message passing time being about equal to the compute time.

Three tilted mappings deliver rates high enough that pF3D is compute bound, but they are significantly lower than the rates in table II. The first custom scheme maps a 32x16 xy-slab to a 32x16 plane of the interconnect and then tilts twice to deliver a rate of 28 MB/s. This mapping takes advantage of the toroidal links, but the maximum hop count is 48=(32+16+48)/2, much higher than the 12=(8+8+8)/2 of the test problems. The second scheme maps a 32x16 xy-slab to an 8x8x8 chunk of the partition and then tilts each block twice within its own volume. This scheme has a maximum hop count of 24=8+8+8 (the block is small enough that the toroidal wraparound links are not used). The third scheme is a variant of the 8x8x8 chunking scheme. The second and third schemes both deliver 31 MB/s.

These results suggest that the number of hops becomes more important for large partitions, but using all three sets of links is still very important. Additional mapping schemes will be tried in the future to see if it is possible to get closer to the 200 MB/s rates obtained for small partitions.

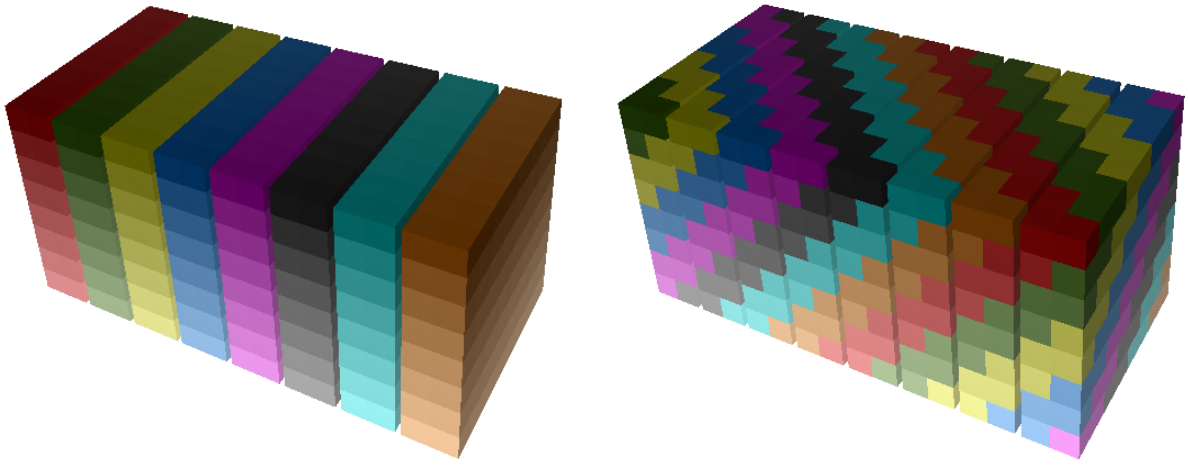
B. Cielo

Cray XE6 systems have an interconnect that is a 3D torus. The shape of the set of nodes assigned to a job is irregular and will change from one submission to the next. There will be holes in the allocation due to I/O nodes, out of service nodes, and nodes already allocated to other jobs. Message contention between applications can occur because they all run on the same torus. Regular mappings from physical domain to location on the torus are not possible due to the irregular



(a) (b)

Fig. 4: Aggregate messing passing rate. (a) The xyzt mapping (black) works well on BG/L systems for small process counts but is very slow for 192k processes. The zyxt message passing rate (blue) is essentially constant from 32k to 192k processes. It is slightly faster for 8k processes than for the other points because the torus is 16x32 instead of 32x32 in the yz plane. (b) The message passing rate on Cielo drops slowly as the number of processes increases. The error bar shows the rate when Cielo uses a different node allocation strategy.



(a) (b)

Fig. 5: This figure shows how a simple partitioning of the BG/P torus can be turned into a tilted mapping. (a) The 8x8x8 partition has been divided into 8 slabs. Each slab corresponds to an xy-slab in physical space. (b) The slabs have been tilted in two directions. The mapping connects an xy-slab in physical space to a tilted plane on the interconnect. Messages passed within an xy-slab can use all three link directions with this mapping, producing a higher message passing rate.

shape of the node set. It is possible to create "mostly regular" mappings, but they must be generated at run time.

Our large Cielo runs have, so far, all used the default mapping from processes to the set of nodes on the torus. On Cray XE6 systems, the default is to assign domains in "node order". In this case, node order means in the numeric order of the nodes assigned to the process. The Gemini chips on the XE6 are numbered with the z-coordinate varying fastest, then the y-coordinate, then the x-coordinate varying slowest. Each Gemini chip is a 1x2x1 chunk of the torus. The 32k simulations we have run on Cielo use a 16x16x128 decomposition. If we ignore the nodes dedicate to I/O and to other jobs, a 16x16 slab of domains tended to map to a 1x2x8 chunk of the Cielo torus before September 2011. The default node allocation strategy of Cielo was changed in September 2011. In our latest Cielo simulation, an xy-slab tends to map to a 2x2x4 chunk of the torus.

The change in default node assignment has improved the message passing rate. The pF3D message passing rates for 32k process runs varied from job-to-job between 60 MB/s/process and 80 MB/s/process with the original node assignment. The rates vary between 110 MB/s/process and 145 MB/s/process with the new strategy. In both cases, the variation from time step to time step within a single run is at roughly the 5% level. This suggests that the main variability is due to how the communicators are laid out on the torus, not interference due to messages from other processes.

One message passing optimization is easily performed on Cielo. By decomposing into 16 domains in the x-direction, all messages for x-FFTs are passed using shared memory within a node. The on node rate is several times faster than the off node rate, so this choice of decomposition allows pF3D to pass x-messages quickly on Cielo.

The Cielo runs in the scaling study of figure 4 all used the original default mapping between MPI rank and location within the set of nodes. The message passing performance drops slowly from 105 MB/s to 55 MB/s as the number of processes increases from 1k to 128k. The roughly 25% variation from job-to-job at 32k processes is nearly half the size of the variation between 1k and 128k processes in the scaling study. This suggests that much of the variation in both cases is due to the placement of processes on the Cielo torus.

All Y messages go off node and are significantly slower than the X messages. The message passing rate for Y messages summed over all 16 processes is probably about 1 GB/s per node for the older assignment scheme. That is significantly less than the roughly 5 GB/s rate that MPI benchmarks achieve. Our run times would be significantly shorter if we could pass messages at 5 GB/s.

We are currently working on a dynamic mapping strategy for Cielo. We will allocate a few extra nodes and select a subset of the nodes such that all y-communicators have the same shape. This should avoid Y-communicators that are longer than the nominal 2x2x4 and thus avoid slow communicators. We do not yet know if the improvement in communication speed will be enough to justify idling a (presumably small) percentage of the nodes. In the longer term, we hope to investigate clustering strategies which achieve better performance than the default mapping without idling a significant number of nodes.

IV. OpenMP and SIMD vectorization

One approach to improving the performance per Watt of a chip is to reduce the clock speed, simplify the cores, reduce the cache per core, and put more cores on each chip. The Intel MIC and IBM Blue Gene/Q processors are current examples of this approach. The simplified instruction issue units on these cores mean that multiple hardware threads must be used on each core to attain peak performance. It may be advantageous to use multiple hardware threads even on high performance processors (see figure 9). With slowly growing memory per chip and a rapidly growing number of hardware threads per chip, the memory per hardware thread will decrease fairly rapidly in the next few years. An application that employs only MPI parallelism will have much less memory per process than on current systems.

The trend of more cores per GB of memory will probably force pF3D to move away from being a pure MPI code. The pF3D executable is fairly small and pF3D doesn't need any large data tables. A pF3D run will fit in a small amount of memory per process. If the overhead of thread coordination is low, pF3D could still achieve good computational efficiency. The main problem with running small processes may be difficulties in achieving high enough message passing rates.

As the number of zones per process decreases, the size of the messages passed drops even faster. Figure 6 shows three possible decompositions of the same physical space. The message size in (b) is 1/8 of the message size for (a) because each process has 1/4 as many zones and it sends messages to twice as many processes. The resulting messages may be short enough that the effective bandwidth will drop. In (c), there are 4 threads per process. Each thread has the same amount of memory as a process in (b). Messages are passed by processes, so their size is the same as in (a).

We are in the process of adding OpenMP parallelism to the MPI parallelism already present in pF3D. Messages will be passed by the process, not OpenMP threads. The number of zones in a domain will remain large, so messages will remain large. The challenge then becomes obtaining OpenMP overheads low enough that several threads can cooperate efficiently on an amount of work previously handled by one core. Once we have an efficient OpenMP+MPI code, we intend to add OpenAcc directives and explore the use of accelerators like the Nvidia Tesla.

Several of the pF3D functions that consume the most time during a run have been converted into kernels. OpenMP directives have been added to the kernels and are being used to investigate how to obtain good performance when using multiple threads.

An OpenMP loop has a barrier at the end and it takes time to start up the threads and assign loop iterations to them at the top of the loop. The computation time needs to be long compared to these overhead times to achieve good OpenMP

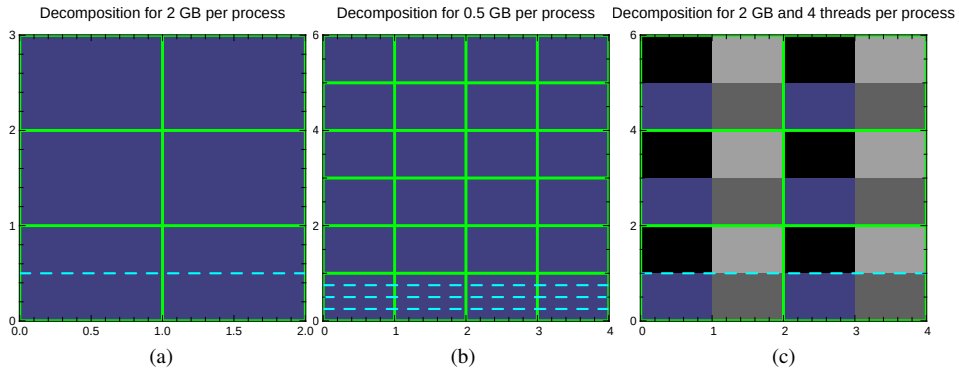


Fig. 6: This figure shows two decompositions for a single xy-slab. (a) The slab is decomposed into 2 domains in the x-direction and 3 domains in the y-direction. (b) The slab is decomposed into 4 domains in the x-direction and 6 domains in the y-direction. (c) The slab is decomposed into 2 domains in the x-direction and 3 domains in the y-direction. There are 4 threads per domain (shown by 4 colors). Green lines show domain boundaries. The domains in the bottom "row" are divided by dashed cyan lines into the portions that will be sent to other processes during an alltoall message.

performance. This effect means that the number of zones per process may need to grow as the number of threads per process grows to maintain good performance. The limited memory on future systems may make it difficult to increase the zones per process. Another solution is to modify chips to have faster thread synchronization primitives so that more threads can be used before running into an overhead problem.

The number of zones per process for a run of the kernels is chosen to be similar to the number of zones per process in Table I.

There are several different ways of evaluating the efficiency of an OpenMP code. In a strong scaling study, the total amount of work (zones) is held fixed and the number of threads is increased. At some point, thread overhead will become important. In a weak scaling study, the work per thread is held fixed. Thread overhead is much less important than for strong scaling. In both cases, contention for memory bandwidth or other shared resources increases as the number of threads increases. If there isn't enough bandwidth to go around, pure OpenMP scaling will suffer.

The third sort of study (hereafter hybrid scaling) is performed using a code with mixed MPI and OpenMP parallelism. The product of the number of processes and the number of threads per process is held fixed, typically at the number of cores (or hardware threads) on the node. The work per core and the load on shared resources like memory is independent of the number of processes.

pF3D simulations will use all the cores on a chip. Our goal when running a scaling study is to determine what combination of threads and processes will deliver the highest throughput. A hybrid scaling study is the best way to answer this question.

Figure 7 shows the rate at which a core can update zones for the advancefi kernel. In an ideal computer system with no contention for shared resources like memory bandwidth and no overhead for thread coordination, the rate would be independent of the number of threads. The strong scaling study

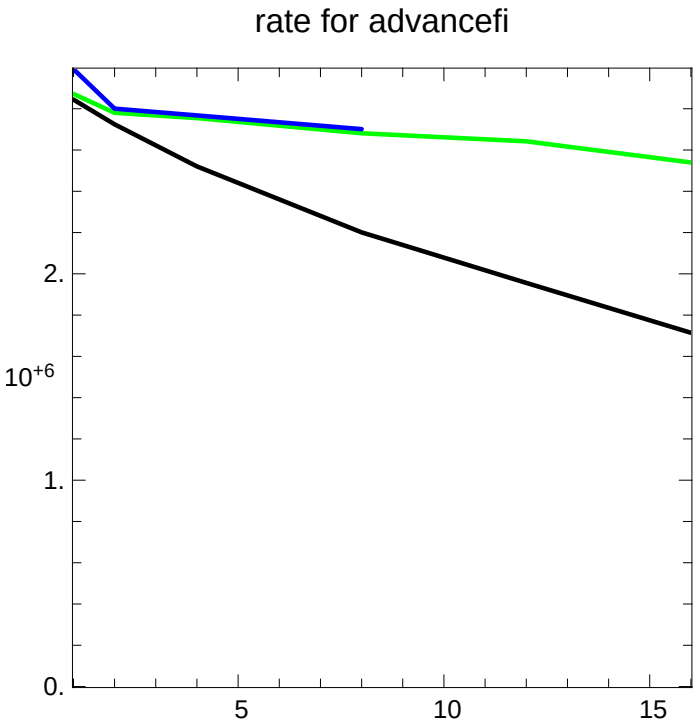


Fig. 7: The figure shows the zones updated per second per core for the advancefi kernel on a Sandy Bridge processor as a function of the number of threads. Weak scaling (green) and hybrid scaling (blue) have times nearly independent of the number of threads. The performance drops significantly in the strong scaling study (black).

shows a drop of roughly 30% between 1 and 16 threads. Weak scaling and hybrid scaling show only a modest drop. This sort of difference between weak and strong scaling indicates that the thread coordination time becomes comparable to the compute time in the strong scaling study.

Several of the other pF3D kernels (absorbdt, couple4, and couple5) have good strong scaling, indicating that there is enough work to amortize thread coordination costs when the total number of zones is similar to the zones per domain of typical pF3D simulations.

Another trend in recent processors is an increase in the width of the SIMD (single-instruction, multiple-data)registers in the floating point unit. pF3D usually operates on 3D arrays of float complex numbers. Older x86 processors have 128 bit wide SIMD registers with a peak float vector performance 4X the scalar peak. The Intel Sandy Bridge has 256 bit wide registers with a peak float performance of 8X times the scalar rate. The Intel MIC chip has 512 bit wide registers with a peak float performance of 16X times scalar. The AMD Interlagos chip has a 256 bit wide floating point unit shared by two integer cores. Its peak float performance is 4X times its scalar peak, the same as earlier x86 systems. It is worth putting effort into SIMD vectorization when there is a potential 16X gain in peak performance.

A fairly high fraction of the key loops in pF3D vectorize if the C99 restrict modifier is applied to the appropriate array pointers. In practice, speedups are much less than the ratio of peak performance. The limited speedups may be due to main memory bandwidth constraints or even to cache bandwidth (some cores can only fetch one SIMD register from the L1 cache per clock tick).

Figure 8 compares the rate at which zones are updated per core for a SIMD vectorized and an unvectorized version of the couple4 kernel in a hybrid scaling study. There is a significant performance advantage (of order 2X) to SIMD vectorization, but the advantage is much less than the theoretical 8X increase in peak performance. The reason for the limited speedup might be contention for memory bandwidth.

Several recent processors have multiple hardware threads per core. Each thread has its own set of registers, so instructions can be issued from any of the hardware threads without a context switch. The figure shows the performance with hyperthreading (Intel's name for Xeon hardware threads) enabled and the same number of zones per *core* as the other runs. Hyperthreading produces a noticeable increase in throughput.

The SIMD performance for a single process using all cores/threads is significantly lower (1) than with two or more processes (1.5-2X). The single process allocates all of its memory on a single socket. Two or more processes use memory on both sockets. The single process job runs slower because only half the memory bandwidth of the socket is used.

There is more thread-to-thread variability in performance for the SIMD vectorized kernel than for the scalar kernel. The reason for that behavior is not yet understood.

Figure 9 shows the benefit of SIMD vectorization and

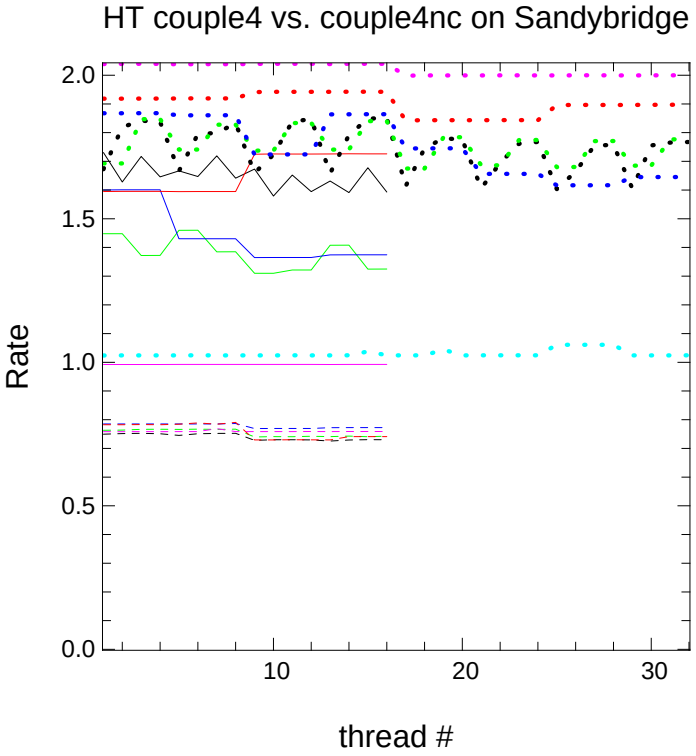


Fig. 8: The figure shows the rate at which zones are updated per core for the couple4 kernel on an Intel Sandy Bridge processor. The solid lines are for a SIMD vectorized version while the dashed lines are for a "scalar" version. Black is 16 processes with 1 thread each, green is 8 processes with 2 threads each, blue is 4 processes with 4 threads each, and red is 8 processes with 2 threads each, and magenta is 16 processes with one thread each. The dotted lines are for the SIMD vectorized kernel with hyperthreading enabled. Black is 32 processes with 1 thread each, green is 16 processes with 2 threads each, blue is 8 processes with 4 threads each, red is 4 processes with 8 threads each, magenta is 2 processes with 16 threads each, and cyan is 1 process with 32 threads. The zones per *core* are the same for all runs.

hyperthreading on an Intel Sandy Bridge processor for the couple4 kernel. These curves were obtained by averaging the rates in 8 over threads. SIMD vectorization provides a speedup in all cases. Hyperthreading provides little benefit for one process, but generally increases the throughput by 20% for 2 or more processes. The variability in the benefit may reflect the larger thread-to-thread variations for the hyperthreaded runs. Hyperthreading provides benefits in the 10-30% range for several other pF3D kernels and for two other LLNL codes.

Figure 10 compares the rate at which zones are updated per floating point unit by the couple4 kernel on Intel Sandy Bridge and AMD Interlagos processors. The rates for the Sandy Bridge are the same for all combinations of threads and processes and are very uniform from core to core during a given run. The Interlagos rates have significant variations from core to core. The variability in the Interlagos numbers

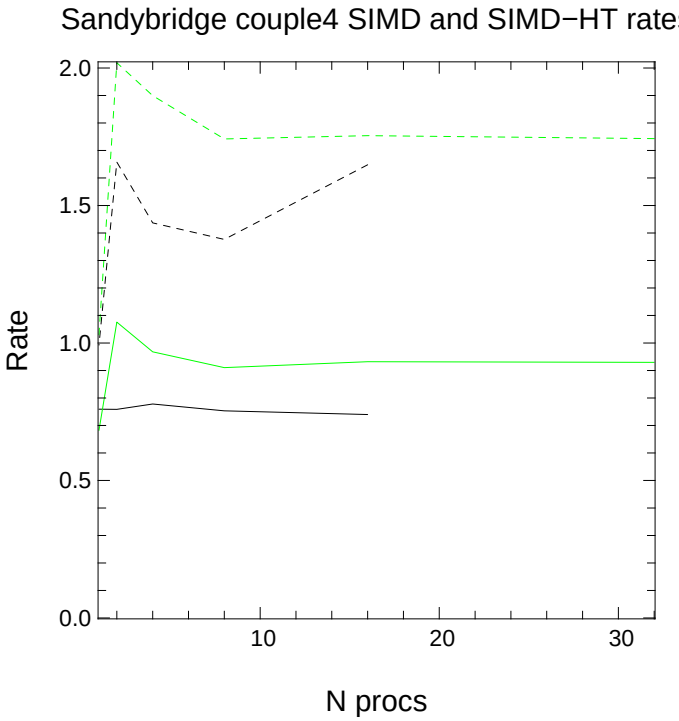


Fig. 9: The figure shows the rate (averaged over threads) at which zones are updated per core as a function of the number of processes for the couple4 kernel on an Intel Sandy Bridge processor. Solid lines are for the non-SIMD version and dashed lines for the SIMD version of the couple4 kernel.The black lines are for one thread per core and the green lines are for two threads per core. The zones per *core* are the same for all runs.

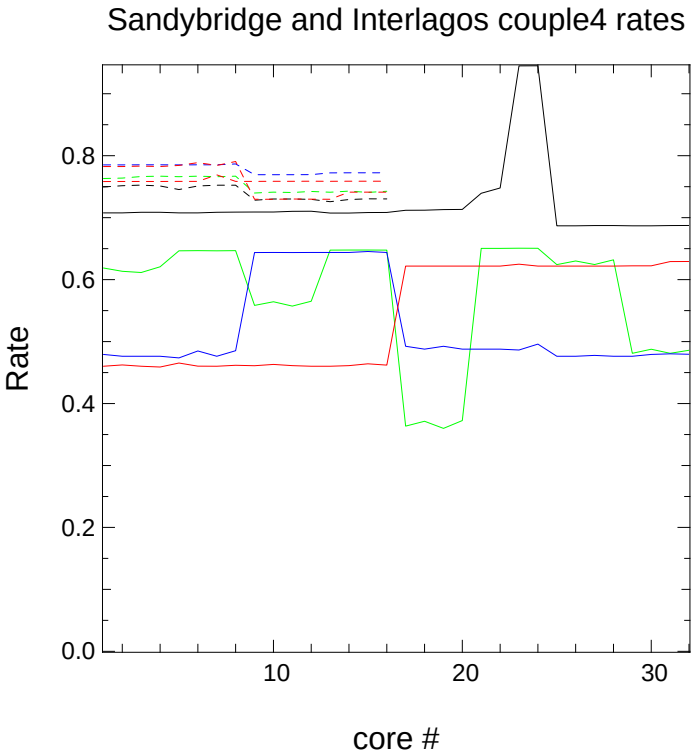


Fig. 10: The figure shows the rate at which zones are updated per core for the couple4 kernel on Intel Sandy Bridge (dashed lines) and AMD Interlagos (solid lines) processors. Black is 16 processes with 1 thread each, green is 8 processes with 2 threads each, blue is 4 processes with 4 threads each, and red is 8 processes with 2 threads each. The rates for the Interlagos chip are for a floating point unit and its two associated integer cores.

may be due to thread binding and memory affinity not being set properly (this is a test node).

Figure 11 compares the rate at which zones are updated per core by the couple4 kernel for Intel Sandy Bridge and AMD Magny Cours processors. The rates for the both processors are nearly the same for all combinations of threads and processes and are very uniform from core to core during a given run. The Sandy Bridge code was compiled with version 12.1 of the Intel C compiler, icc. icc does not bind threads to cores properly on Cielo, so we used gcc 4.6 for the Magny Cours. This kernel does not vectorize, so gcc produces competitive performance. The Sandy Bridge processor is faster than the Magny Cours for this kernel, but not by as much as these results generated with different compilers suggest.

V. Efficient Checkpointing

Large computers like Cielo have large numbers of components. Even with carefully designed RAS systems, applications running on a large fraction of the system must be prepared to deal with errors that interrupt execution. Jobs may be interrupted by hardware errors that cause a node to fail, they may be interrupted by ”soft” hardware errors (the hardware is still usable if the job is relaunched), and they may be interrupted by system software errors.

A resiliency strategy for dealing with hardware and system software faults is an important part of a massively parallel code today and it will become even more important on future systems. pF3D uses application checkpoints as its resiliency strategy. pF3D writes the state of the simulation to the parallel file system after a user selected number of simulation time steps. ”Productive I/O” is the I/O performed to permit the scientist to understand the simulation, extract results, and generate presentations. Writing a checkpoint set is an example of ”defensive I/O”. A checkpoint is no longer needed once a newer checkpoint has been written. Checkpoints are often deleted without ever having been read. A checkpoint written at the end of a batch time slot is not ”defensive” - it is required to resume the simulation at a later time.

If a fault occurs, the simulation is resumed from the most recent checkpoint. A restart involves the loss of the time spent computing since the last checkpoint and the time required to read in the checkpoint at the start of the new run. The checkpoint interval is chosen to balance the time spent in defensive I/O against the time lost recovering from a fault.

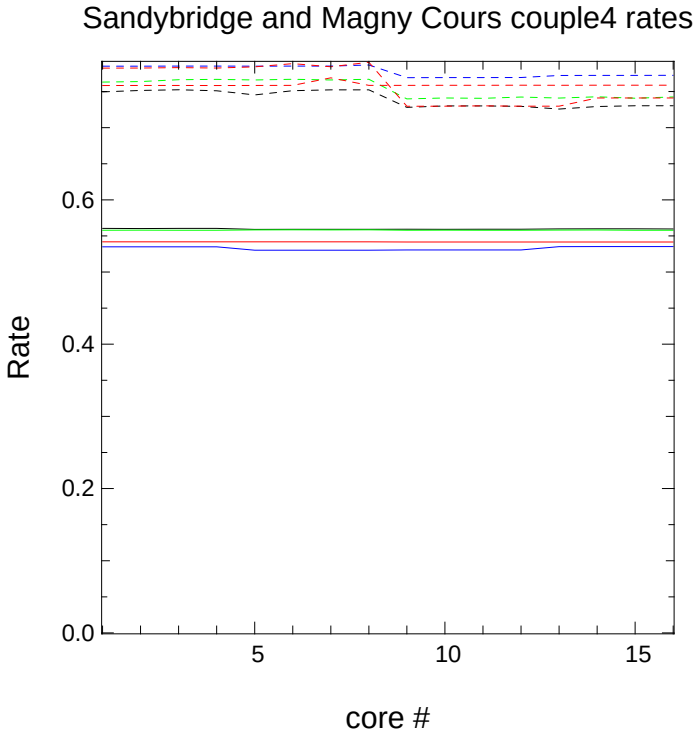


Fig. 11: The figure shows the rate at which zones are updated per core for the couple4 kernel on Intel Sandy Bridge (dashed lines) and AMD Magny Cours (solid lines) processors. Black is 16 processes with 1 thread each, green is 8 processes with 2 threads each, blue is 4 processes with 4 threads each, red is 8 processes with 2 threads each, and magenta is 16 processes with one thread.

System	N-proc	Size (TB)	Rate (GB/s)	FS type
Intrepid	128k	31.9	24.7	GPFS
Dawn	96k	58.0	27.6	Lustre
Cielo	32k	44.8	33.9	Panasas

TABLE III: This table shows the rate at which pF3D wrote checkpoint sets to disk in recent large runs on three different systems.

In practice, the checkpoint interval is chosen to be something like half the MTBI for the application.

pF3D uses a file-per-process strategy for restart dump files. Table III shows the rate at which checkpoint files were written in recent pF3D runs on three different systems. We have put significant effort into obtaining good I/O rates in pF3D. The I/O rates are the highest of any code currently running on Intrepid. The I/O rate on Cielo nearly matches the rate for the IOR benchmark code. The rate on Dawn is probably the highest of any code running on that system.

In spite of the very good performance we have achieved, defensive I/O performance is still an issue. It takes half an hour to write a checkpoint to disk for a 32k process run on Cielo. A simulation reads a checkpoint at the start of the run and writes three checkpoints to disk for a total of 2 hours of defensive I/O during a run that lasts roughly 23 hours. A 64k process run with twice as many zones would take twice as long for its checkpoint I/O - 4 hours out of 23.

Exascale systems are expected to need much longer to copy all of memory to magnetic disk than current systems. Writing checkpoints on future systems may take longer than the MTBI of the system. Writing checkpoints to disk is not a viable a resilience strategy on a system of that sort.

Defensive I/O produces files that only need to be retained for a short while - less than the duration of one batch job. That means they can be written to local storage that will be lost at the end of a batch job. Local storage often has a much higher aggregate bandwidth than a parallel file system, so it is an interesting target for defensive I/O. Most current large systems do not have local disks due to the operational difficulties of replacing them when they fail. A few systems have local flash disk and it is also possible to use memory as a RAM disk. The aggregate bandwidth of per-node flash disk and RAM disk is high enough that checkpoints can be written in a time much shorter than the MTBI. A checkpoint would only need to be copied to disk once per batch job.

SCR [5], [6] can write checkpoints to RAM disk in a few seconds. Checkpoints are broken up into several data chunks plus a parity chunk and written to the RAM disks of multiple nodes. When a node fails, SCR still has a complete checkpoint set on the remaining nodes. If a job allocates a few extra nodes, it can be restarted from RAM disk by issuing another aprun command within the same batch job. SCR copies a checkpoint to disk at the end of a batch slot (or more often, if desired).

Using SCR to write checkpoints can significantly improve the efficiency of pF3D. In the case of the 64k run mentioned above, the checkpoint I/O time would drop from 4 to 2 hours per 23 hour job, producing a throughput improvement of 10%.

The benefit of SCR is actually much larger than that. Local checkpointing is so fast that it can be done after every time step rather than every 6-7 hours. When a job crashes, the most recent checkpoint is less than an hour old and little compute time is lost. As noted above, SCR can also restart pF3D multiple times during a single batch time slot. Restarts during a job read the checkpoint from local storage and take a few seconds to complete. A pF3D run on 16k AMD Barcelona cores increased the results delivered per day by roughly 50% when it switched to using SCR.

SCR is currently undergoing testing on Cielo. Cielo is in the process of switching from a Panasas file system to a Lustre file system. We will evaluate the improvements in efficiency provided by SCR after that transition is complete.

VI. LPI Results

pF3D obtains its initial conditions (temperatures, densities, and materials) from a HYDRA simulation of the full capsule implosion. HYDRA includes all the physics necessary to model the implosion, but it does not have the ability to model backscattered laser light. The HYDRA simulations cover the full 20 ns duration of the experiment. pF3D simulations cover roughly 50 ps at interesting times in the pulse. The backscatter

levels from the pF3D simulation can be fed back to HYDRA as an energy loss from the laser.

The simulation we discuss here was run using the temperature and density at a time early in the peak of the laser pulse. The simulation shows bursts of SRS and SBS backscattered light. The experiments cannot resolve time scales this short, so we compare time-averaged backscatter levels to the data.

Experimental data at this time show SRS levels ranging from 22% to 55% and SBS levels ranging from 6% to 22%. The simulation has time averaged levels of 16% SRS and 4.4% SBS. The levels of backscatter in the simulation are a bit lower than the lower end of the experimental range. The simulation was expected to have somewhat lower levels of backscatter than the experiments for a couple of reasons. Some of the region where SBS is generated was not included in this simulation. Experiments have shown that overlapping beams increase the level of backscatter. The reference beam in the simulation only has a neighbor on one, not both, sides. The agreement between simulation and experiment is fairly good, given these issues, and should improve as we continue to refine our simulations.

The SRS and SBS start in a few small regions near the back of the simulation volume, then grow in strength as they propagate back towards the laser entrance plane. The SRS spreads more rapidly in the transverse direction as it propagates than does the SBS. The SRS and the SBS both have multiple bursts. SRS bursts are strong enough to reflect a high percentage (over 40%) of the laser light at their peak. The unreflected light is at an intensity low enough that it does not trigger backscattering when it reaches the SRS and SBS generation region. This leads to the SRS and SBS turning on and off with a period roughly equal to the time it takes the laser to propagate through the region that has strong SRS growth.

Figure 12 shows the SRS backscattered light at a time shortly before the peak of the first burst. The SRS from the two beams is spatially separated at this time. Later on in the burst there is SRS in the region where the two beams overlap.

Figure 13 shows the SRS and SBS power as a function of time. The SRS has bursts with a regular period. The SBS starts later than the SRS. The first two SBS bursts coincide with an SRS burst. There are two SBS bursts at the time of the final SRS burst. These results are still being investigated, but it appears that the different temporal behavior of the two kinds of backscatter is due to a portion of the SBS being generated in a region of the beam that doesn't have much SRS.

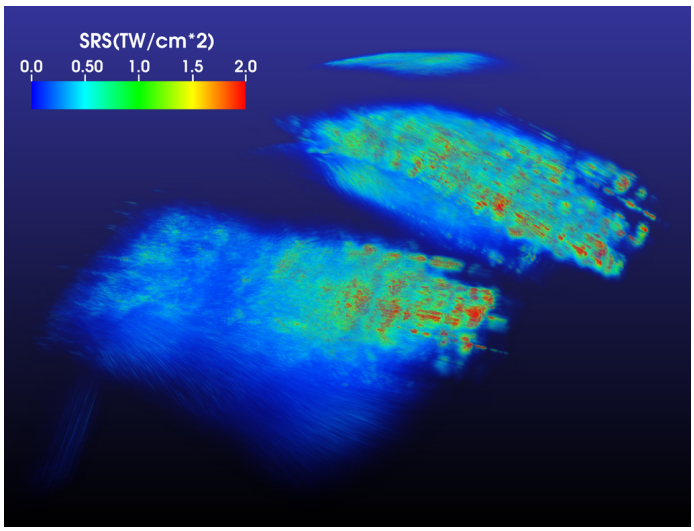


Fig. 12: The figure shows the SRS backscattered light at a time shortly before the peak of the first burst. The SRS from the two beams is spatially separated at this time. Later on there is SRS in the region where the two beams overlap.

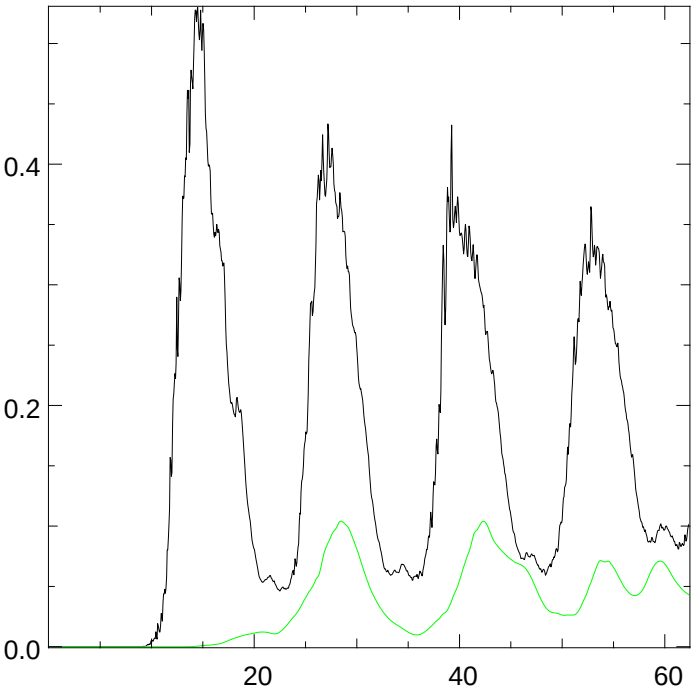


Fig. 13: The figure shows the SRS and SBS backscattered light as a function of time. The scattered light is shown as a fraction of the incident light.

VII. Conclusions and Future Work

We have successfully run three pF3D laser-plasma simulation of NIF experiments on Cielo. We obtained good I/O rates, good message passing rates, and good CPU performance. We have developed a tool that makes it easy to generate custom mappings between the location of a domain in physical space and its location on a Blue Gene interconnect. We are currently working on extending that capability to the Cray XE6. We are adding OpenMP threading to pF3D so that we can run efficiently with fewer zones per core than current simulations.

We are currently testing SCR on Cielo. Using SCR should significantly reduce the time pF3D spends on defensive I/O.

When the changes we have been working on go into full production on Cielo later this year, we should be able to generate significantly more results per day than with the current version of pF3D. The changes we have made will also put us in a good position to exploit even larger systems like Sequoia, a 20 Pflop/s IBM Blue Gene/Q.

ACKNOWLEDGMENT

This work was performed under the auspices of the Lawrence Livermore National Security, LLC, (LLNS) under Contract No. DE-AC52-07NA27344. This document was released as LLNL-PROC-547711.

This research used the Cielo capability computing resource at Los Alamos National Laboratory which is managed by the Los Alamos National Security, LLC, under Contract No. DE-AC52-06NA25396.

This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

REFERENCES

[1] E. I. Moses, "Overview of the national ignition facility," *Fusion Science and Technology*, vol. 54, no. 2, pp. 361–366, 2008.

[2] C. H. Still, R. L. Berger, A. B. Langdon, D. E. Hinkel, L. J. Suter, and E. A. Williams, "Filamentation and forward brillouin scatter of entire smoothed and aberrated laser beams," *Physics of Plasmas*, vol. 7, no. 5, p. 2023, 2000.

[3] R. L. Berger, B. F. Lasinski, A. B. Langdon, T. B. Kaiser, B. B. Afeyan, B. I. Cohen, C. H. Still, and E. A. Williams, "Influence of spatial and temporal laser beam smoothing on stimulated brillouin scattering in filamentary laser light," *Phys. Rev. Lett.*, vol. 75, no. 6, pp. 1078–1081, Aug 1995.

[4] S. Langer, B. Still, T. Bremer, D. Hinkel, B. Langdon, and E. A. Williams, "Cielo full-system simulations of multi-beam laser-plasma interaction in nif experiments," *CUG 2011 proceedings*, 2011.

[5] A. Moody and G. Bronovetsky, "Scalable i/o systems via node-local storage: Approaching 1 tb/sec file i/o," *SuperComputing 2008 Proceedings*, 2008.

[6] A. Moody, G. Bronovetsky, K. Mohror, and B. R. de Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing systems," *SuperComputing 2010 Proceedings*, 2010.

[7] D. H. Munro, "Using the yorick interpreted language," *Computers in Physics*, vol. 9, no. 6, p. 609, 1995.

[8] D. H. Munro, "The yorick home page," http://yorick.sf.net, 2011.