Trillion Particles, 120,000 cores, and 350 TBs: Lessons Learned from a Hero I/O Run on Hopper

Surendra Byna*, Andrew Uselton*, Prabhat*, David Knaak†, and Yun (Helen) He*

*Lawrence Berkeley National Laboratory, USA. Email: {sbyna, acuselton, prabhat, yhe}@lbl.gov
†Cray Inc., USA. Email: knaak@cray.com

***Abstract*—Modern petascale applications can present a variety of configuration, runtime, and data management challenges when run at scale. In this paper, we describe our experiences in running VPIC, a large-scale plasma physics simulation, on the NERSC production Cray XE6 system Hopper. The simulation ran on 120,000 cores using ∼80% of computing resources, 90% of the available memory on each node and 50% of the Lustre scratch file system. Over two trillion particles were simulated for 23,000 timesteps, and 10 one-trillion particle dumps, each ranging between 30 and 42 TB, were written to HDF5 files at a sustained rate of ∼27 GB/s. To the best of our knowledge, this job represents the largest I/O undertaken by a NERSC application and the largest collective writes to single HDF5 files. We outline several obstacles that we overcame in the process of completing this run, and list lessons learned that are of potential interest to HPC practitioners.**

I. Introduction

Running applications on petascale machines at full scale can present a variety of challenges. When applications use a significant portion of a supercomputing system, they push the system's computation, memory, network, and parallel I/O resources to their limits. Such pressure on various subsystems can cause configuration, runtime, and data management challenges. Successful execution of these applications requires careful planning and tuning.

In this paper we describe our experiences in running VPIC, a large-scale plasma physics application on the Hopper system [1], a Cray XE6 with over 153,000 cores installed at the National Energy Research Scientific Computing (NERSC) Center. The application simulates magnetic reconnection with two trillion particles. It uses 120,000 cores on 5,000 nodes of Hopper. The simulation uses ∼29 GB of the available 32 GB memory on each node with a total of ∼142 TB memory footprint. The VPIC simulation dumps particle data ranging between 30 TB and 42 TB per time step at an interval of ∼2,000 time steps. The total amount of data produced in this simulation was ∼350 TB. The simulation also produced another ∼140 TB of checkpoint data. In total, the data produced here occupied 50% of the `/scratch2` Lustre file system on Hopper.

We discuss the following list of lessons learned in running the simulation.

1) **Collective writes to a single shared HDF5 file can work as well as file-per-process writes.** We demonstrate that collective writes from 20,000 MPI processes to a single, shared ∼40 TB HDF5 file using collective buffering can achieve a sustained performance of 27 GB/s on a Lustre file system. The peak performance of the system is ∼35 GB/s, which is achieved by our code for a substantial fraction of the runtime. This performed as well as the strategy where each process wrote a separate file (i.e. a total of 20,000 files).
2) **Tuning multiple layers of a parallel I/O subsystem is a challenging task.** The Cray XE6 parallel I/O stack consists of various levels of software libraries (HDF5, MPI-IO) and Lustre file system components. The libraries and file system offer various tuning parameters; which need to be set appropriately to obtain best system performance. We designed a benchmark kernel to explore this space manually and devised optimal configurations for our large scale VPIC run. Furthermore, we modified the HDF5 library to implement an important optimization (see Section III-B). In this paper, we show performance improvement obtained with HDF5 and Lustre file system optimizations.
3) **Advance verification of filesystem hardware is important for obtaining peak performance.** Our initial execution of VPIC achieved only 65% of Lustre peak performance. With the use of Lustre Monitoring Toolkit (LMT) [8], [18], we pinpointed the problem to a small set of slow OSTs, which were exhibiting degraded performance. We temporarily excluded these OSTs from our tests, and were able to demonstrate ∼80% of the peak I/O rates. Advance verification for slow OSTs is critical for avoiding performance pitfalls.
4) **Advance verification of available resources for memory-intensive applications is important.** Unreleased memory from previous applications could cause out-of-memory errors. Since the simulation requires 90% of the memory on each node, we routinely verified that each node had sufficient memory before undertaking the full-scale run.
5) **Scalable tools are required for diagnosing software and hardware problems.** There are many points of failure in a HPC system, and it can be difficult to isolate faults either prior to or during the execution of a large scale application. Scalable tools for fault detection are essential for quickly identifying and addressing problems.

In the following section, we provide the details of the VPIC application, the system configuration, and the configuration of the trillion particle simulation. In Section III, we discuss each of the above lessons in detail. We then conclude the discussion in Section IV.

II. VPIC Simulation

A. Application

Collisionless magnetic reconnection is an important mechanism that releases explosive amounts of energy as field lines break and reconnect in plasmas, such as when the Earth's magnetosphere reacts to solar eruptions. The massive energy released by the solar eruptions can damage satellite communication equipment as well as produce beautiful aurora borealis at the poles of the Earth. Such a reconnection also plays an important role in a variety of astrophysical applications involving both hydrogen and electron-positron plasmas.

Simulation of magnetic reconnection with VPIC (vector particle-in-cell) [3] is inherently a multi-scale problem. It is initiated in the small scale around individual electrons but eventually leads to a large-scale reconfiguration of the magnetic field. Recent simulations have revealed that electron kinetic physics is not only important in triggering reconnection [6], [5], [7], [11], [14], [13], [19], but also in its subsequent evolution. Based on this conclusion, plasma physics scientists find that they need to model the detailed electron motion, and that modeling poses severe computational challenges for 3D simulations of reconnection. A full-resolution magnetosphere simulation is an exascale-class computing problem.

We recently ran the highly optimized VPIC code [3], which simulates collisionless magnetic reconnection with open boundaries in 3D. The simulation tracks two trillion particles on the Hopper Cray XE6 supercomputer at NERSC. Particle properties of interest in this simulation include spatial location (x,y,z), energy, and projection of velocity components on the directions parallel and perpendicular to the magnetic field $U_{\parallel}$, $U_{\perp,1}$, and $U_{\perp,2}$.

B. System Configuration

Hopper is a NERSC Cray XE6 system consisting of 6,384 compute nodes, each containing two 12 core AMD 2.1 GHz MagnyCours processors. Each compute core has a peak performance of 8.4 Gflops/se resulting in a system with a peak performance of 1.28 PFlops. All but 384 compute nodes have 32 GB memory while the remaining larger nodes have 64 GB memory creating a system with over 217 TB of memory. It employs the Gemini interconnect with a 3D torus topology.

Hopper has two identical local Lustre parallel file systems: `/scratch` and `/scratch2`, each has a peak performance of 35 GB/sec and a capacity of 1.1 PB. The Lustre file system is made up of an underlying set of IO servers and disks called Object Storage Servers (OSSs) and Object Storage Targets (OSTs) respectively. Each `/scratch` file system has 156 OSTs which is the lowest software abstraction layer with which users need to interact. When a file is created in `/scratch` it is "striped" or split across multiple OSTs. The default stripe count on Hopper is 2 and the default stripe size is 1 MB.

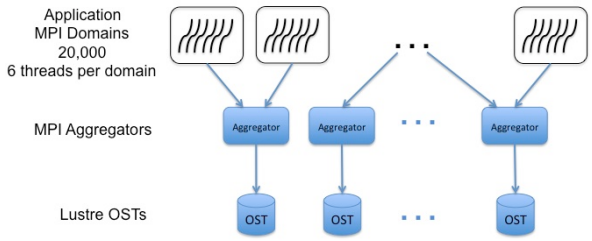


Fig. 1. Configuration of our VPIC simulation using MPI and OpenMP threads for computation. Parallel I/O uses MPI-IO in collective buffering mode and Lustre parallel file system

VPIC uses Cray's MPICH2 library (xt-mpt 5.1.2) and HDF5 version 1.8.8 with modifications as described in Section III-B. The particle data is written with H5Part version 1.6.5, along with Cray's MPI-IO implementation.

C. Parallel I/O and Data

The VPIC simulation writes a significant amount of data at a user-prescribed interval. In our simulation of 2 trillion particles (including 1 trillion ions and 1 trillion electrons), we store particle data sets, field and hydro data sets, and checkpoint data sets. The particle data set for a given time step comprises $\sim$1 trillion electrons. The data size of each electron particle is 32 bytes, representing the particle properties mentioned in Section II-A. The number of particles increases as the simulation progresses. In our simulation, the data sets written in each time step varied between 30 TB to 42 TB of data. The simulation wrote the particle data set at 10 time steps and the total particle data size was $\sim$335 TB. The data set for the ions is not stored to disk to reduce storage requirement. Furthermore, most of the physics questions can be answered with only information about electrons. The field data set includes information such as electric and magnetic field strength, and the particle data set includes information about its position, momentum and energy. The field data set is relatively small, on the order of tens of gigabytes. We dumped field data at 33 time steps and size of this data set was $\sim$15 TB. The simulation also wrote a checkpoint data set multiple times. We configured the simulation to maintain two such data sets. The size of each checkpoint data set was between $\sim$60 TB and $\sim$72 TB. Overall, the total amount of data after finishing the simulation was $\sim$490 TB. After discarding the checkpoint data sets, the total amount of data we had to analyze was $\sim$350 TB.

Figure 1 shows an overview of the VPIC simulation set up on Hopper. VPIC uses 20,000 MPI processes, where each MPI

process spawns 6 OpenMP threads to perform computations.[1] Each OpenMP thread runs on a CPU core and the total of CPU cores used in this simulation is 120,000. The figure also shows MPI-IO aggregators to collect data from multiple MPI domains before writing data to the Lustre file system. We will explain our work in using MPI-IO, HDF5, and H5Part with collective buffering enabled in Section III-A.

III. Lessons Learned

A. Lesson 1. Collective writes to a single shared HDF5 file can work as well as file-per-process writes

As with many HPC applications, the I/O pattern in VPIC is "banded", i.e. there is a period of computation and communication across all nodes and then a barrier at which all MPI ranks must rendezvous. Figure 2 illustrates a sample banded pattern of interleaved computation and I/O phases. After the barrier the I/O takes place and the compute and communication resources are idle (at least so far as the application is concerned). Once the I/O is complete, all the nodes rendezvous again, and then the next phase of computation and communication proceeds. This pattern repeats for multiple cycles.

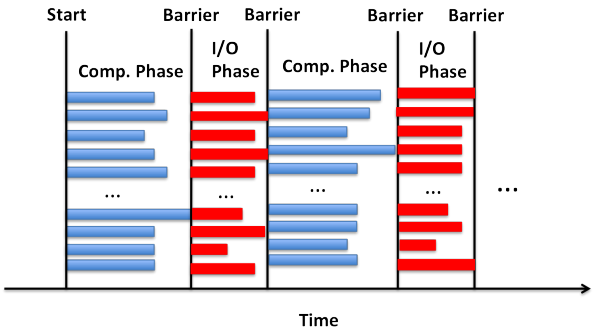


Fig. 2. An example banded pattern showing computation and I/O phases. The slowest process reaching rendezvous dictates the finishing time of a phase. During the I/O phase, computation resources stay idle wasting energy.

In order to get the most science done we'd like the I/O bands to be as narrow as possible. We can accomplish this by limiting the amount of data written and/or read and by achieving the best possible I/O performance given the I/O pattern. As mentioned in Section II-C, we only saved the electrons, which halved the amount of I/O necessary. It is commonly assumed that a *file per process* (*fpp*) I/O model will achieve the best performance. This assumption is based on the fact that there is no need for file system range locking when each rank has exclusive access to its target file.

1) Writing file with file per process model: In the original implementation of the VPIC code, each MPI domain writes a file, in binary format, containing its particle data [12]. Each of the files has a header with the cell offsets and the number of particles in the file. The *fpp* approach is able to achieve a good fraction of system I/O bandwidth, but has a number of limitations. The first is that the number of files at large scale becomes too large. For example, in our largest scale test, the simulation generates $20,000$ files per timestep corresponding to $20,000$ MPI domains. Performing even a simple ls command on the directory containing these files has significant latency. Second, the *fpp* model dictates the concurrency of subsequent stages in the analysis pipeline. Often a post-processing step is necessary to refactor *fpp* data into a format that is readable by analysis tools. Third, many data management and visualization tools only support standard scientific data formats, such as HDF5 [16] and NetCDF [17]. Fourth, the *fpp* I/O model might be suboptimal, as we'll discuss shortly.

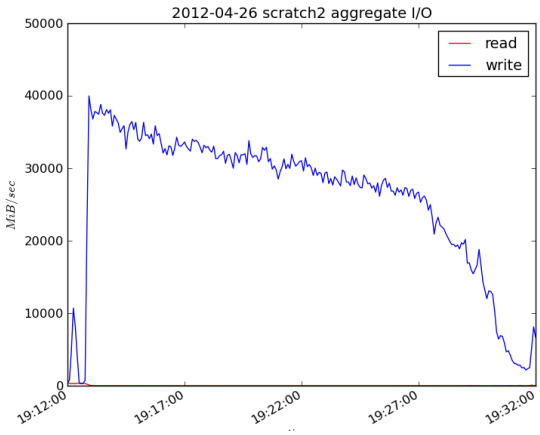
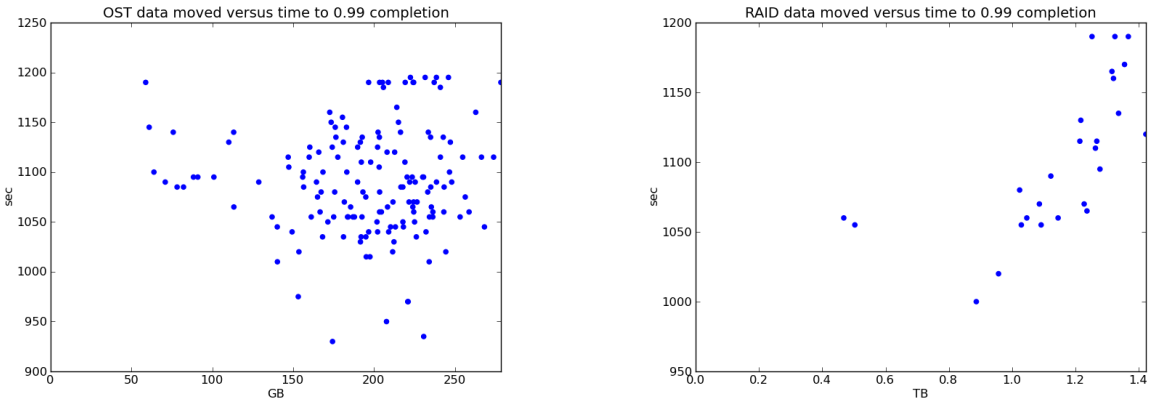


Fig. 3. A file-per-process I/O model initially performs well, but its performance degrades over time. This comes from an imbalance in the load on the OSTs.

Performance of an *fpp* application can be undermined by a load imbalance if the individual files are unevenly assigned (in Lustre) to the Object Storage Targets (OSTs). If the OSTs perform at their maximum rate (the same for all) and some OSTs are assigned more work than others, then those OSTs will take longer. The files assigned to those OSTs, and the MPI ranks for those files, will complete their I/O later than the rest, so those ranks will arrive later to the rendezvous. The result is a larger I/O band than might have been achieved with uniform load balancing. The time series of server I/O observations in Figure 3 shows that the file system can deliver in the range of 35 GB/s, and the load imbalance shows up as a long downward sloping tail in the I/O rate as the slowest OSTs straggle in. The resulting effective I/O rate ends up at 27,007 MB/s for writing 29.7 TB of particle data.

Figure 4(a) shows the amount of data assigned to each of the 156 `/scratch2` OSTs on Hopper (*x-axis* in GB) versus the time-to-completion of the I/O on that OST (*y-axis* in seconds). Figure 4(b) gathers the load for the six OSTs on each RAID controller (*x-axis* in TB), it is clear that the uneven load leads directly to the uneven completion time. That imbalance is what gives Figure 3 its characteristic shape. Some form of manual load balancing could help with the performance penalty, but

[1] In subsequent discussion we refer to an "MPI process" as an "MPI domain" in order to highlight the fact that each MPI process has multiple OpenMP threads.



(a) OSTs (b) RAIDs

Fig. 4. The 20,000 VPIC files are of comparable size, but they are distributed unevenly over the OSTs (left) and RAID controllers (right), with six OSTs per controller. There is a wide variety of time-to-completion among the OSTs. For the controllers, it is clear that the load largely determines the time to completion.

the *fpp* I/O model may not be the best choice for all the foregoing reasons.

2) Writing file with HDF5 and H5Part: In our updated implementation of writing particle datasets, we take the approach of writing a single global file with a standard data format known as HDF5 [16]. More specifically, we use a particle data extension of parallel HDF5 called H5Part. Parallel HDF5 has demonstrated competitive I/O rates on modern computational platforms [10]. As far as we know, we are the first group to successfully write tens of terabytes in a single shared HDF5 file. The H5Part [9] extension to HDF5 improves the ease of use in managing large particle counts. H5Part is a veneer API for HDF5: H5Part files are also valid HDF5 files and are compatible with other HDF5-based interfaces and tools. By constraining the usage scenario to particle-based simulations, H5Part is able to encapsulate much of the complexity of implementing effective parallel I/O in HDF5. That is, it trades off HDF5's flexibility and complexity in supporting arbitrary data models for ease-of-use with a specific, particle-based data model.

Using a small set of H5Part API calls, we were able to quickly integrate parallel HDF5 I/O into the VPIC codebase. Our simple H5Part interface for writing VPIC particle data is outlined in the following lines of code:

```
h5pf = H5PartOpenFileParallel (fname, H5PART_WRITE |
                    H5PART_FS_LUSTRE, MPI_COMM_WORLD);
H5PartSetStep (h5pf, step);
H5PartSetNumParticlesStrided (h5pf, np_local, 8);

H5PartWriteDataFloat32 (h5pf, "dX", Pf);
H5PartWriteDataFloat32 (h5pf, "dY", Pf+1);
H5PartWriteDataFloat32 (h5pf, "dZ", Pf+2);
H5PartWriteDataInt32   (h5pf, "i",  Pi+3);
H5PartWriteDataFloat32 (h5pf, "Ux", Pf+4);
H5PartWriteDataFloat32 (h5pf, "Uy", Pf+5);
H5PartWriteDataFloat32 (h5pf, "Uz", Pf+6);
H5PartWriteDataFloat32 (h5pf, "q", Pf+7);

H5PartCloseFile (h5pf);
```

The H5Part interface opens the particle file and sets up the attributes, such as the timestep information and the number of particles. The *H5PartWrite*$\cdots()$ calls wrap the internal HDF5 data writing calls.

The H5Part interface opens the file with MPI-IO collective buffering and Lustre optimizations enabled. Collective buffering breaks the parallel I/O operations into two stages. The first stage uses a subset of MPI tasks to aggregate the data into buffers, and the aggregator tasks then write data to the I/O servers. With this strategy, fewer nodes communicate with the I/O nodes, which reduces contention. The Lustre-aware implementation of Cray MPI-IO sets the number of aggregators equal to the striping factor such that the stripe-sized chunks do not require padding to achieve stripe alignment [4]. Because of the way Lustre is designed, stripe alignment is a key factor in achieving optimal performance.

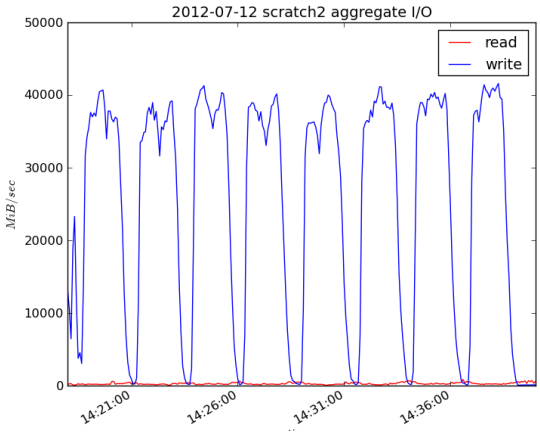
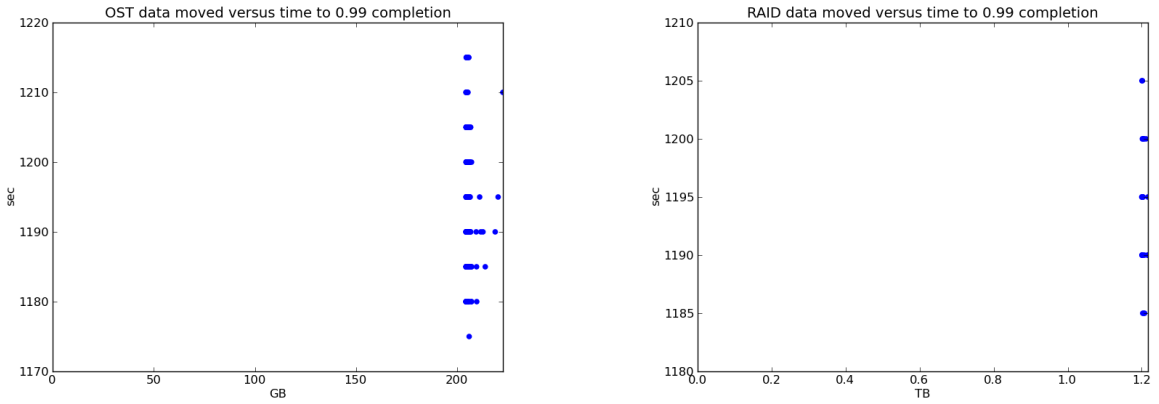


Fig. 5. A single-file I/O model must contend with range locking but will automatically load balance across its OSTs. This particle dump achieved the same (within 1 percent) I/O rate as the file-per-process example in Figure 3: 27 GB/s. The load balancing is better, but the H5Part wrapper imposes an extra barrier between each of the eight variable dumps.



(a) OSTs (b) RAIDs

Fig. 6. The load across the OSTs and the RAID controllers is very evenly balanced. The time to completion for the dump varied across the OSTs by about 40 seconds compared to about 200 seconds for the file-per-process example in Figure 3.

One advantage of using a single file I/O model is that the load automatically gets uniformly distributed to all the I/O resources. On the other hand, H5Part also introduces a global rendezvous (MPI-IO collective operation) between each of the eight variable dumps (Figure 5). This creates little "gaps" that make the I/O band larger than it otherwise might be, and makes even more important the need to get well coordinated I/O.

Figures 6(a) and 6(b) show a very uniform load across the 156 OSTs and the RAIDs, respectively. There is still some variability in time-to-completion. Because of the MPI-IO collective operation after each variable is dumped, we pay the price for this variability eight times. Nevertheless, the achieved data rate of 27,035 MB/s for 31.3 TB compares favorably with the results from the *fpp* version.

B. Lesson 2. Tuning multiple layers of parallel I/O subsystem is a challenging task

Performance of parallel I/O depends on multiple software layers of the parallel I/O stack. Figure 7 shows a contemporary parallel I/O software stack with HDF5 as high-level I/O library, MPI-IO as middleware layer, and Lustre as the parallel file system. Each layer offers tunable parameters for improving performance, and hopes to provide reasonable default settings for the parameters.

Tuning HDF5, MPI-IO, and Lustre parameters was an important part of achieving our I/O performance goals. We modified the implementation of HDF5 version 1.8.8 to disable the file size verification process. For Lustre, we varied the stripe count and the stripe size. The stripe count is the number of OSTs used in writing contiguous chunks of data and the stripe size is the size of the contiguous chunk. As mentioned earlier, the Lustre-aware implementation of Cray MPI-IO sets the MPI-IO collective buffering parameters. In this implementation, the number of MPI-IO aggregators is equal to the stripe count, and the size of the collective buffer is equal to the stripe size.

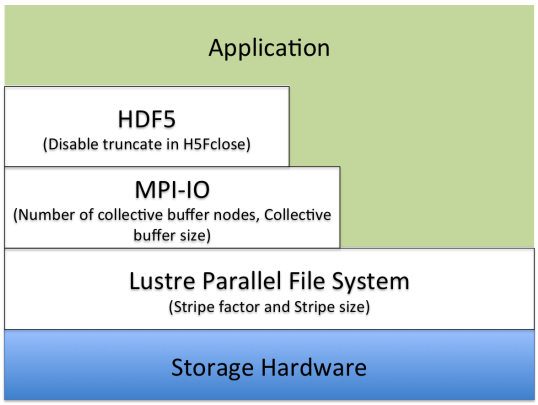


Fig. 7. Parallel I/O Stack and the parameters we tuned and modifications we made in this effort

Tuning I/O for the VPIC simulation typically requires running the simulation multiple times while varying the values of tunable parameters. However, VPIC is a computationally intensive application, and it is impractical to run the entire code repeatedly, at scale for tuning. Instead, we developed a simplified parallel I/O kernel, called VPIC-IOBench. The kernel uses the same H5Part calls shown in Section III-A for writing VPIC particle data. VPIC-IOBench disables the simulation component of the VPIC code and uses random data. The kernel contains the full data volume generated by the code with a slightly simplified pattern. In VPIC-IOBench, each MPI process writes an equal number of particles to a shared file, whereas in a VPIC simulation, each MPI domain writes a slightly varying (up to 15%) number of particles. The amount of data the kernel writes is proportional to the number of MPI processes.

1) Tuning HDF5: When a HDF5 file is closed, the *H5Fclose* function ensures that the size of the file matches its allocated size. HDF5 tracks the size of the file for two reasons: to detect external file modification/corruption (i.e. from

something other than the HDF5 library modifying/corrupting the file), and to allocate space within the file for changes to the file's structure. HDF5 currently verifies the file's size by truncating the file (using a POSIX or MPI operation) to the size of the allocated space within the file. This truncate-based verification step initiates several metadata server operations that degrade performance significantly. Disabling the file's size verification in some cases will likely make the file unreadable. For example, if the truncate operation during *H5Fclose* would actually extend the file (instead of truncate it), but doesn't occur (because of disabling truncate), data will be double allocated in the file, causing corruption later. If the truncate was actually truncating the file (instead of extending it), it is less of a problem - space in the file will be leaked, but the file will still be readable. We made a decision to disable the file's size verification for performance. In all our datasets, disabling truncate did not affect the ability to read it. We modified the HDF5 library source code (version 1.8.8) to disable the file size verification process, which improved performance of writing files by a factor 3-5X. Figure 8 shows the impact of disabling the file size verification. The plot shows improvement of the I/O bandwidth of VPIC-IOBench over a range of MPI processes counts, where each MPI process writes 32 MB data. A better alternative to simply disabling the truncate call is to track the valid section of the file by adding more metadata to the file format. This optimization will soon be a part of upcoming HDF5 release allowing all parallel HDF5 applications to more productively use HDF5 for I/O.

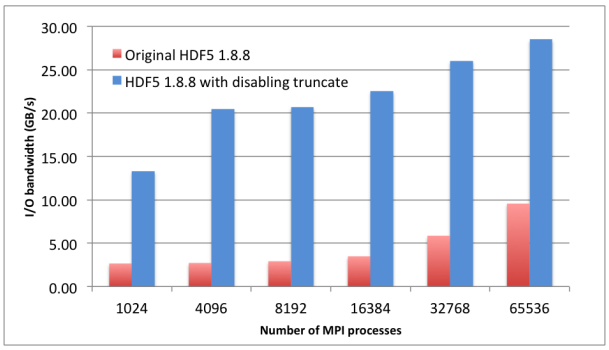


Fig. 8. Performance improvement with patching HDF5 truncate

2) Tuning Lustre file system and MPI-IO parameters: We conducted a series of tests with VPIC-IOBench using 8 K tasks, and varied the stripe count from 64 OSTs to the maximum of 156 and the stripe size from 1 MB to 1 GB. The Cray Lustre-aware, MPI-IO implementation varies the MPI-IO collective buffer aggregators and their buffer size to match the corresponding stripe count and stripe size. Our prior experiments indicated that the attainable data rate did not increase with stripe counts beyond 144. The last few OSTs didn't add any performance. We settled on using 144 OSTs and stripe size 64 MB.

Figure 9 shows the results of a scaling study for 1 K to 128 K MPI tasks. This is a *weak* scaling study in that the

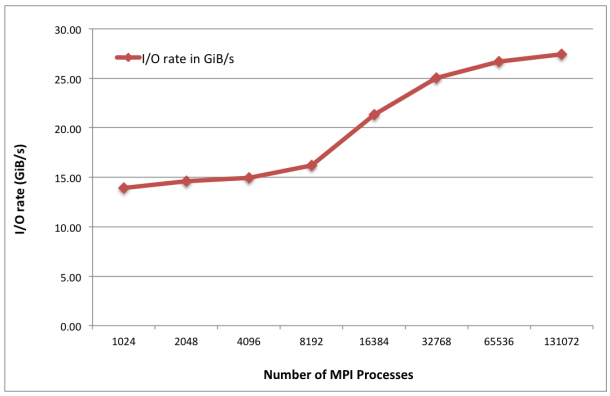


Fig. 9. VPIC-IOBench weak scaling study: I/O performance with increasing number of processes writing data to a HDF5 file using H5Part.

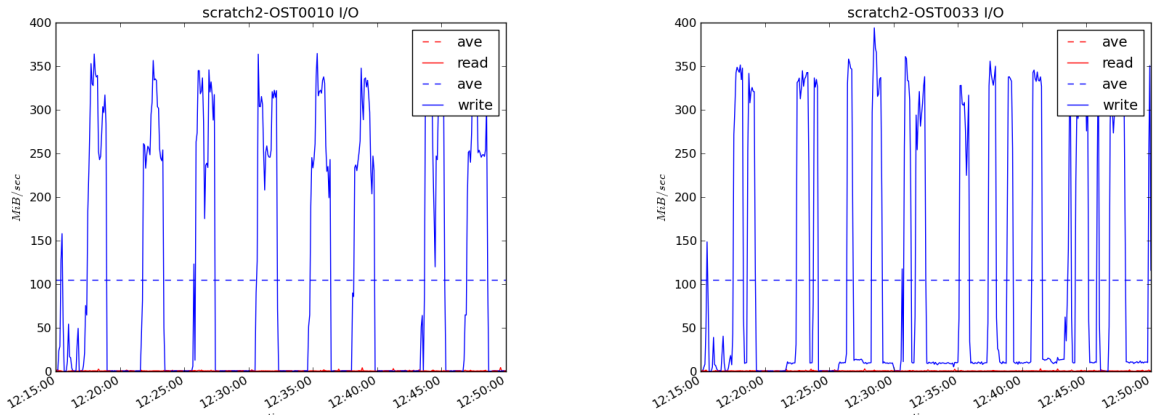
number of particles per task is constant at eight million. These experiments use the modified HDF5 library that has the patch for disabling file size verification when a HDF5 file closes. As the number of MPI tasks increases, the I/O rate becomes greater. With fewer MPI tasks running on a highly shared system, such as Hopper, interference from I/O activity of other jobs can reduce the attained I/O rate. At the scale of 128 K cores, VPIC-IOBench occupies 85% of Hopper, which reduces the opportunity for interference from other jobs sharing the I/O system. The 128 K task instance writes about 32 TB of data, and Figure 9 shows at that scale the delivered I/O performance is about 27 GB/s, which compares favorably with the rated maximum on Hopper of about 35 GB/s. It is also comparable with the best rates achieved with an *fpp* model.

The trillion particle VPIC simulation used the tuning parameters from this study and achieved 27 GB/s as shown in Figure 5. Our study motivates that applications performing large parallel I/O can benefit from a careful tuning process. It will be even more beneficial to have automatic and scalable I/O tuning tools.

C. Lesson 3. Advance verification of filesystem hardware is important for obtaining peak performance

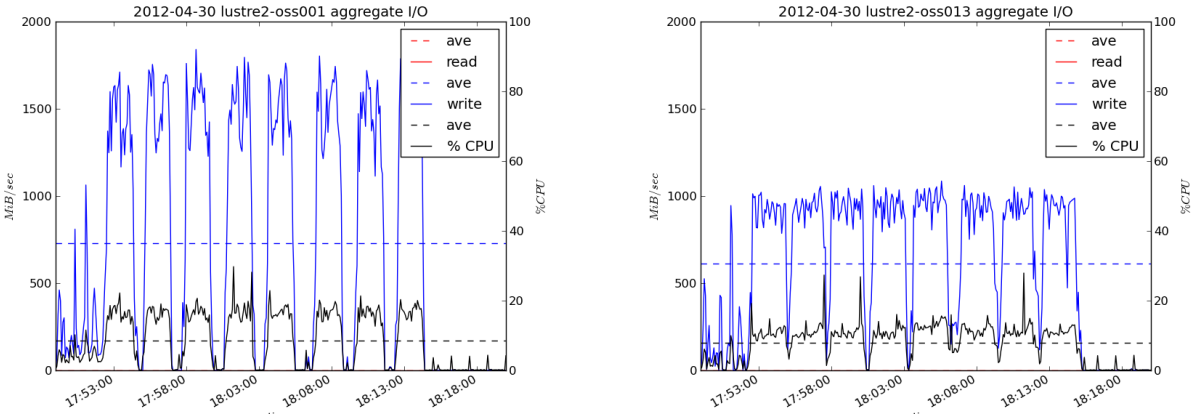
During an early VPIC run, we found that the particle dumps (of eight variables) took much longer than we had expected. In Figure 10(a) one OST shows good behavior as it quickly completes the I/O for each of the eight variables. Nevertheless, it must wait for a global rendezvous before starting the next, and this leads to large gaps in its I/O pattern. For this run 144 OSTs were used, and to achieve an aggregate rate of 35 GB/s each OST needs to be performing at around 250 MB/s. Clearly, OST0010 can reach that I/O rate, but its average value ends up at half that due to the delays.

A close inspection of the performance of the individual OSTs showed that one was behaving erratically (Figure 10(b)). It looked like a disk was failing or the LUN was being rebuilt after a failure. The problem resolved itself by the time of the next run, but it contributed to a significant delay in that execution of the application. The particle dumps took about 35 minutes in that case.



(a) Well behaved OST (b) Erratic OST

Fig. 10. An OST (left) behaves as it should, but has to wait for all the other OSTs between each variable dump. In this case one OST was behaving erratically and delayed the whole dump step.



(a) Speedy (b) Slow

Fig. 11. The six OSTs on OSS-001 (left) perform well, but still have to wait for other OSTs between each variable dump. In this case two OSSs, including OSS-013 (right) have all their OSTs performing poorly compared to the rest. There was an identified problem with a RAID controller pair at this time. Once resolved, the I/O was able to proceed at its best speed.

A followup run later that month showed improved performance. The particle dumps took about 23 minutes instead of 35 minutes. There was still a significant gap between the individual variable dumps. In Figure 11(a) the six OSTs of an OSS cumulatively perform well, but again have to wait for a rendezvous when other OSTs are delayed. In this case all the OSTs on two OSSs performed at a consistently slower rate (Figure 11(b)). There was coincidentally a failed-over RAID controller pair, though the connection between that and the slower OSSs is not clear. Again, the issue resolved itself before the next run and subsequent runs were able to reduce dump time to around 20 minutes.

These observations, lead the team to test and review individual OST performance at the beginning of each run. In particular, the VPIC-IOBench I/O kernel acted as a system probe. Running that for a minute or two at the beginning of a VPIC run would be able to identify sluggish hardware, and points to a possible system level service that would do the same thing automatically.

D. Lesson 4. Advance verification of available resources for memory-intensive applications is important

The trillion particle simulation requires a total memory of ∼142 TB. Using 120,000 cores on 5,000 nodes, where each node has 24 cores, the memory requirement of the simulation on each node is ∼29 GB. On Hopper, each node has 32 GB memory and the memory requirement of ∼29 GB translates to 90% of memory footprint. Considering some lightweight OS related tasks running on the nodes, the 90% memory requirement puts a significant pressure on the node.

In our simulation runs on Hopper, we used a combination of tools to verify memory availability after experiencing an application failure due to an Out-of-Memory (OOM) error on one node. The Cray staff at NERSC used Cray's Node

Health Checker (NHC) [15] to check the available free memory on each node and "admindown" those nodes that have free memory below 29 GB. We also developed a Perl script that reads the free memory information in a compute node's "`/proc/buddyinfo`" file. This /proc file is the most accurate measure of available memory, for the application and for the kernel, on a Cray XT/XE/XK compute node. The script collects information on the available memory on all the nodes that are allocated for running the simulation. The script is placed in the PBS batch script of our simulation run to verify available memory before and after our simulation. We manually sorted the output based on the available memory to verify that none of the nodes have free memory less than 30 GB.

From this experience, we learned that when running applications at scale with greater than 75% memory requirement on each node, verification of available memory before and after running the application is helpful. This is also another opportunity for a system service automating what is currently a burdensome process.

E. Lesson 5. Scalable tools are required for diagnosing software and hardware problems before running applications using 100k cores

It can be time consuming and tedious to systematically verify system health prior to a hero run. Better tools and tools that scale to very large problem size are very much needed. When running applications at scale, a failure due to software problems or hardware problems or both can cause significant waste of system resources, including the waste of system time, staff time, and the energy consumed.

1) Scalable computation and memory resource checker: Cray's Node Health Checker (NHC) is useful to verify node status and available free memory, and we suggest automating its use before running large applications. In our experience, with help from Cray and NERSC staff, we used a combination of OS capabilities and local tools to verify the computation and memory resource availability. For instance, we used "`xtprocadmin`", a capability of the Cray XE6 OS to verify the current status of nodes. As mentioned above, we used the NHC and a local script to verify the availability of required memory. Automatic checks for resource availability can reduce idle system time. Currently on Hopper, the NHC script is used in production, with the threshold being adjusted as needed. Sometimes the script is used in "log-only" mode instead of "admindown" depending on the number of "bad" nodes detected, and the possibility of false-positive detection of "suspect" nodes which can cause the successive runs inside the same batch job to fail. Improving the robustness of NHC is desired.

2) Scalable I/O subsystem checker: The I/O subsystem consists of multiple software libraries as shown in Figure 7. While we used manual tuning with multiple layers, a scalable auto-tuning will help many applications. In many cases, applications rely on default I/O settings and suffer from poor I/O performance that becomes a performance bottleneck. Moreover, in applications with "banded pattern" of compute and I/O phases, poor I/O performance wastes a significant amount of CPU time and the corresponding system and energy wastage. We are currently working on a solution to auto-tune parallel I/O [2]. In this effort, we plan to identify common I/O patterns of applications and choose I/O tuning parameters from a pre-populated database.

3) Scalable Runtime I/O Monitor: As illustrated in Section III-C, even one slow hardware component in the file system can cause a dramatic slow down. To avoid such inefficiencies, we advocate scalable tools to monitor file system performance persistently. When an OST becomes sluggish for any reason, repairing it or removing it from the file system dynamically can help overall application performance. In our VPIC simulation, we were able to exclude OSTs that were over 85% full by using stripe factor of 1 and selecting stripe offset, but there is no systematic way that a certain application can exclude certain "bad" (sluggish, nearly full, etc.) OSTs. In production, setting these OSTs to "read-only" will impact other system scripts such as quota checking and purge scripts, so it is not recommended. Better control of OST targets within the application space is desired.

IV. Conclusions

We have successfully undertaken an unprecedented trillion particle simulation on 120,000 cores on Hopper. The fully-functional simulation produced 30 TB to 42 TB of data on a per timestep basis resulting in major I/O challenges. While we were utilizing a well-tuned production I/O stack consisting of HDF5, Cray MPI-IO and Lustre; we faced and addressed a number of issues at scale. We developed a custom protocol to detect OSTs with poor performance characteristics and compute nodes with low memory before the full scale runs. We implemented a patch for HDF5 that enabled the application to utilize all available system bandwidth with collective I/O. We believe that Hero runs at scale require support and contributions from teams of diverse personnel. In this case, researchers at LBL worked closely with NERSC system staff and Cray engineers to diagnose and address challenges as they came up. We believe that such a model of collaboration is critical for facilitating state-of-the-art applications, such as VPIC, in undertaking groundbreaking high resolution simulations and fully utilizing HPC platforms.

Acknowledgment

The authors gratefully acknowledge the support extended by NERSC and Cray staff in making this work possible. We would like to thank Tina Butler, Katie Antypas, Steve Luzmoor, Francesca Verdier, Woo-Sun Yang, Terence Brewer, Randell Palmer, Bill Anderson, Mark Pagel, Steven Oyanagi, and Harvey Wasserman. We thank Quincey Koziol from The HDF Group for HDF5 performance optimizations. Our VPIC collaborators Bill Daughton, Homa Karimabadi and Vadim Roytershteyn played a critical role in porting the code to Hopper, and configuring it to run at scale. We would like to thank John Wu, Mark Howison, Jerry Chou, Kevin Lin, Oliver

Rübel, Wes Bethel, Arie Shoshani, and John Shalf for their support and advice. This work is supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center.

REFERENCES

[1] NERSC Hopper system. http://www.nersc.gov/users/computational-systems/hopper/.

[2] B. Behzad, J. Huchette, H. Luu, R. Aydt, S. Byna, M. Chaarawi, Q. Koziol, Prabhat, and Y. Yao. Auto-tuning of parallel i/o parameters for hdf5 applications. https://sdm.lbl.gov/~sbyna/research/papers/SC12_poster.pdf. SC12 Research Poster Reception, 2012.

[3] K. J. Bowers, B. J. Albright, L. Yin, B. Bergen, and T. J. T. Kwan. Ultrahigh performance three-dimensional electromagnetic relativistic kinetic plasma simulation. *Physics of Plasmas*, 15(5):7, 2008.

[4] Getting Started with MPI I/O. http://docs.cray.com/books/S-2490-40/S-2490-40.pdf.

[5] W. Daughton, V. Roytershteyn, H. Karimabadi, L. Yin, B. J. Albright, B. Bergen, and K. J. Bowers. Role of electron physics in the development of turbulent magnetic reconnection in collisionless plasmas. *Nature Physics*, 7:539542, 2011.

[6] W. Daughton, J. D. Scudder, and H. Karimabadi. Fully kinetic simulations of undriven magnetic reconnection with open boundary conditions. *Physics of Plasmas*, 13, 2006.

[7] J. Egedal, W. Daughton, and A. Le. Large-scale electron acceleration by parallel electric fields during magnetic reconnection. *Nature Physics*, 8:321324, 2012.

[8] C. M. Herb Wartens, Jim Garlick. LMT - The Lustre Monitoring Tool. https://github.com/chaos/lmt/wiki/. Developed at Lawrence Livermore National Lab.

[9] M. Howison, A. Adelmann, E. W. Bethel, A. Gsell, B. Oswald, and Prabhat. H5hut: A High-Performance I/O Library for Particle-Based Simulations. In *Proceedings of 2010 Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS10)*, Heraklion, Crete, Greece, Sept. 2010. LBNL-4021E.

[10] M. Howison, Q. Koziol, D. Knaak, J. Mainzer, and J. Shalf. Tuning HDF5 for Lustre File Systems. In *Proceedings of 2010 Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS10)*, Heraklion, Crete, Greece, Sept. 2010. LBNL-4803E.

[11] H. Karimabadi, W. Daughton, and J. Scudder. Multi-scale structure of the electron diffusion region. *Geophys. Res. Lett.*, 34, 2007.

[12] H. Karimabadi, B. Loring, A. Majumdar, and M. Tatineni. I/O strategies for massively parallel kinetic simulations. http://nashi-submaster.ucsd.edu/movies/SC10/sc10-io-poster.pdf. SC10 Research Poster Reception, 2010.

[13] A. Klimas, M. Hesse, and S. Zenitani. Particle-in-cell simulation of collisionless reconnection with open outflow boundary conditions. *Physics of Plasmas*, pages 082102–082102–9, 2008.

[14] M. Shay, J. Drake, and M. Swisdak. Two-scale structure of the electron dissipation region during collisionless magnetic reconnection. *Phys. Rev. Lett.*, 99, 2007.

[15] J. Sollom. Cray's node health checker: An overview. In *Proceedings of the Cray User Gruop meeting (CUG) 2011*, Fairbanks, Alaska, May 2011.

[16] The HDF Group. HDF5 user guide. http://hdf.ncsa.uiuc.edu/HDF5/doc/H5.user.html, 2010.

[17] Unidata. The NetCDF users' guide. http://www.unidata.ucar.edu/software/netcdf/docs/netcdf/, 2010.

[18] A. Uselton. Deploying server-side file system monitoring at nersc. In *Cray User Group Conference*, Atlanta, GA, 2009.

[19] R. V., W. Daughton, H. Karimabadi, and F. S. Mozer. Influence of the lower-hybrid drift instability on magnetic reconnection in asymmetric configurations. *Phys. Rev. Lett.*, 108:185001, 2012.