Parallel I/O, Analysis, and Visualization of a Trillion Particle Simulation

Surendra Byna, Jerry Chou, Oliver Rübel, Prabhat, Homa Karimabadi, William S. Daughton, Vadim Roytershteyn, E. Wes Bethel, Mark Howison, Ke-Jou Hsu, Kuan-Wu Lin, Arie Shoshani, Andrew Uselton, and Kesheng Wu

Lawrence Berkeley National Laboratory
One Cyclotron Road
Berkeley, CA 94720

Parallel I/O, Analysis, and Visualization of a Trillion Particle Simulation

Surendra Byna*, Jerry Chou†, Oliver Rübel*, Prabhat*, Homa Karimabadi‡, William S. Daughton§, Vadim Roytershteyn‡, E. Wes Bethel*, Mark Howison¶, Ke-Jou Hsu†, Kuan-Wu Lin†, Arie Shoshani*, Andrew Uselton*, and Kesheng Wu*

*Lawrence Berkeley National Laboratory, USA. Email: {sbyna, oruebel, prabhat, ewbethel, shoshani, auselton, kwu}@lbl.gov
†Tsinghua University, Taiwan. Email: jchou@cs.nthu.edu.tw, vidcina@gmail.com, asymplone@gmail.com
‡University of California - San Diego, USA. Email: {homakar, vroytersh}@gmail.com
§Los Alamos National Laboratory, USA. Email: daughton@lanl.gov
¶Brown University, USA. Email: mhowison@brown.edu

Abstract—**Petascale plasma physics simulations have recently entered the regime of simulating trillions of particles. These unprecedented simulations generate massive amounts of data, posing significant challenges in storage, analysis, and visualization. In this paper, we present parallel I/O, analysis, and visualization results from a VPIC trillion particle simulation running on 120,000 cores, which produces $\sim 30TB$ of data for a *single* timestep. We demonstrate the successful application of H5Part, a particle data extension of parallel HDF5, for writing the dataset at a significant fraction of system peak I/O rates. To enable efficient analysis, we develop hybrid parallel FastQuery to index and query data using multi-core CPUs on distributed memory hardware. We show good scalability results for the FastQuery implementation using up to 10,000 cores. Finally, we apply this indexing/query-driven approach to facilitate the first-ever analysis and visualization of the trillion-particle dataset.**

I. Introduction

Modern scientific discovery is increasingly driven by data [28]. Computational simulations routinely produce 100s of GBs to 10s of TBs of data per simulation. For instance, the Inter-governmental Panel on Climate Change multi-model CMIP-3 archive is about 35 TB in size. The next generation CMIP-5 archive, which will be used for the AR-5 report [2] is projected to contain over 10 PB of data. Large scale experimental facilities produce equally impressive amounts of data. The LHC experiment is capable of producing 1 TB of data in a second, many gigabytes of which are recorded for future analyses. The Large Synoptic Survey Telescope (LSST) will record many terabytes of data per night. The torrents of data is expected to overwhelm our capacity to make sense of them [14]. In the US, a serious national effort is underway to address challenges of managing and analyzing big data[1].

In this paper, we consider the challenges of analyzing the data from VPIC, a state-of-the-art plasma physics code that simulates 2 trillion particles (one trillion ions and one trillion electrons) on 120,000 cores. The simulation produces an unprecedented amount of data, making storage, analysis, and visualization extremely challenging. We highlight our scalable algorithmic and software strategy, and demonstrate how we can enable meaningful scientific analysis. Our technical contributions are as follows:

- We demonstrate the application of H5Part, a particle data extension of parallel HDF5, for enabling high performance parallel I/O in writing the one trillion electrons.
- We develop a hybrid parallel version of FastQuery using both MPI and pthreads to enable scalable indexing and querying for the trillion particle dataset.
- We use query-based visualization to quickly identify and render particles of interest.
- We apply all of these capabilities to target open scientific analysis problems, which were simply impossible to address before.

A. Plasma Physics Simulation

Collisionless magnetic reconnection is an important mechanism that releases energy explosively as field lines break and reconnect in plasmas spanning from the Earth's magnetosphere to solar eruptions. Such a reconnection also plays an important role in a variety of astrophysical applications involving both hydrogen and electron-positron plasmas. Furthermore, reconnection is the dominant mechanism that enables the plasma from the solar wind to enter the Earth's magnetosphere. Reconnection is inherently a multi-scale problem. It is initiated in the small scale around individual electrons but eventually leads to large-scale reconfiguration of the magnetic field. Recent simulations have revealed that electron kinetic physics is not only important in triggering reconnection, but also in its subsequent evolution. This finding suggests that we need to model the detailed electron motion, which poses severe computational challenges for 3D simulations of reconnection. A full-resolution magnetosphere simulation is an exascale computing problem.

The advent of petascale computers together with advances in particle simulations are now enabling us to conduct simulations a factor of 1000 times larger than the state-of-the-art just a few years ago. Our main code is the highly optimized

[1]http://www.whitehouse.gov/blog/2012/03/29/big-data-big-deal.

particle code VPIC [4]. This new capability is providing us with the first glimpse of details of collisionless reconnection in 3D. We have successfully conducted simulations with 1.2 trillion particles on the Kraken system at Oak Ridge National Lab (ORNL) using 100K cores, and consisting of $2,048 \times 2,048 \times 1,024$ computational cells in 2011 [10]. In this paper, we focus on the large data management and analysis challenges and demonstrate the effectiveness of our approach using a larger 2 trillion particle run conducted at the National Energy Research Scientific Computing center (NERSC).

B. Science Use Case

Computational Plasma physicists are generally interested in understanding the structure of high dimensional phase space distributions. For example, in order to understand the physical mechanisms responsible for producing magnetic reconnection in a collisionless plasma, it is important to characterize the symmetry properties of the particle distribution, such as agyrotropy. Agyrotropy is a quantitative measure of the deviation of the distribution from cylindrical symmetry about the magnetic field. Another question of significant practical importance in studies of magnetic reconnection is characterization of the energetic particles. Particle properties of interest include spatial location (x, y, z), energy, and projection of velocity components on the directions parallel and perpendicular to the magnetic field ($U_{\parallel}$,$U_{\perp,1}$,$U_{\perp,2}$).

In the scope of this paper, we will explore the following scientific questions:

- Analysis of highly energetic particles:
  - Are the highly energetic particles preferentially accelerated along the magnetic field?
  - What is the spatial distribution of highly energetic particles?
- What are the properties of particles near the reconnection hot-spot (the so-called X-line)?
  - What is the degree of agyrotropy in the spatial vicinity of the X-line? In other words, is the density plot of the $U_{\perp,1}$ vs. $U_{\perp,2}$ components highly asymmetrical?

While these questions can be addressed to some extent for smaller scale 2D and 3D simulations involving millions or billions of particles, it is challenging to address these questions when the number of particles reach beyond hundreds of billions or trillions. Hampered by the lack of scalable tools, physicists have largely ignored the particle data, used some form of sub-sampling, or relied on coarser gridded data for their analysis. To the best of our knowledge, this is the first study that offers flexible technical capabilities for analyzing trillion particle datasets.

C. Research Challenges

Motivated by these scientific questions and the desire to support this new regime of petascale plasma physics simulations, we tackle the following computer science research problems:

- What is a scalable I/O strategy for storing massive particle data output?
- What is a scalable strategy for conducting analysis on these datasets?
- What is the visualization strategy for examining these datasets?

In this paper, we demonstrate the use of H5Part and HDF5 to address the scalable I/O challenges in Section II-A. Section II-B describes our effort in developing a hybrid parallel version of FastQuery, and applying it for indexing and querying the trillion particle dataset. Section II-C outlines our approach for query-based visualization in VisIt to select scientifically relevant particles to render on the screen.

II. Approach

We now highlight our technical strategy for addressing the research challenges outlined in the previous section.

A. Parallel I/O with H5Part/HDF5

The VPIC simulation writes a significant amount of data at a user-prescribed interval. In this study, we use the data files from a simulation of 2 trillion particles (including 1 trillion ions and 1 trillion electrons). The simulation produces field data and particle data. The field data include information such as electric and magnetic field strength, and the particle data include information about its position, momentum and energy. The data 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 is relatively small, on the order of tens of GB. The particle data is much larger, on the order of tens of TB. The challenge we need to address is to develop a convenient and efficient storage strategy for handling such large data sets.

In the original implementation of the VPIC code each MPI domain writes a file in binary format containing for its particle data[20]. Each of the files has a header with the cell offsets and the number of particles in the file. This *file-per-process (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 time step. 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 re-factor fpp data into a format that is readable by analysis tools.

In this work, we take the approach of writing a single global file with a standard data format known as HDF5 [32]. 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 [18]. As far as we know, we are the first to attempt to write tens of terabytes in a single HDF5 file. The H5Part [16] 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 time step 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 [9]. Because of the way Lustre is designed, stripe alignment is a key factor in achieving optimal performance.

B. Indexing/Querying with Hybrid Parallel FastQuery

In this work, we use FastQuery [8], [6], [7] to accelerate the data analysis process of the trillion particle dataset. Here, we briefly recap the salient features of FastQuery, and elaborate on the new hybrid parallel implementation.

1) FastQuery: FastQuery is a parallel indexing and querying system for array-based scientific data formats. It uses the FastBit bitmap indexing technology [36] to accelerate data selection based on arbitrary range conditions defined on the available data values, e.g., "energy $> 10^5$ and temperature $> 10^6$." FastQuery has a unified plug-in interface to enable the query functionality on varied array-based data formats. Currently, our implementation of the interface supports a wide range of scientific data formats including HDF5 [32], NetCDF [33], pNetCDF [23] and ADIOS-BP [25].

The two main functions of FastQuery are indexing and querying. Indexing builds the indexes of the data and stores them into a single file, so it can be conveniently accessed later for evaluating different queries. The indexing operation contains three main steps: (1) read data values from the file, (2) construct indexes data structure in memory, (3) write bitmaps to the file.

Querying uses the indexes to evaluate user-specified query conditions on data and retrieve data records satisfying the conditions. If the necessary indexes have not been built, FastQuery would scan through the data values to evaluate the query. Typically, the indexes are available and the querying process includes two main steps (1) load bitmap from file, and (2) evaluate indexes for query.

2) Hybrid Parallel Implementation: In order to process massive datasets, FastQuery uses parallelism across distributed memory nodes, as well as multiple cores available on each node. The basic strategy of the parallel FastQuery implementation is to partition a large-scale dataset into multiple fixed-size sub-arrays and to assign the sub-arrays among processes for indexing and querying. When constructing the indexes, the processes build bitmaps on sub-arrays one after another, and store them into the same file by using collective I/O calls in high-level I/O libraries, such as HDF5. When evaluating a query, the processes apply the query on each sub-array and return the aggregated results.

In our previous work [8], we have shown that this parallel strategy can be implemented using MPI alone. While this is an effective approach, there are limitations to this flat-MPI based approach. (1) Each MPI task needs to maintain information about the others so it knows how to communicate with them. As more MPI domains are used, more memory space has to be devoted to MPI, which reduces the memory available to the user operations [3]. (2) For operations on the same node, efficient MPI implementations will make use of the shared memory. However, explicitly using the shared memory in user code is typically more efficient than going through the MPI interface [17]. (3) It is faster to synchronize and load-balance among the small number of threads on a computer node than across a large number of MPI tasks.

In this work, we seek to improve FastQuery by implementing a hybrid parallelism approach with MPI and Pthreads [12], [17]. This extended implementation supports the HDF5 data format. Our strategy is to let each MPI task create a fixed number of threads. The MPI tasks are only responsible for holding shared resources among threads, such as the MPI token for inter-process communication and the memory buffer for collective I/O, while the threads do the actual processing tasks of creating indexes and evaluating queries.

The hybrid parallel FastQuery divides a dataset into multiple fixed size sub-arrays, and builds the indexes of those sub-arrays iteratively. In each iteration, one sub-array is assigned to each thread, the indexes for the sub-arrays are collected together and stored to the same HDF5 dataset in the index file. In each iteration, a HDF5 collective IO call (i.e. H5Dcreate and H5Dset_extent) is required to create or extend the dataset for storing indexes, the building process is synchronized among MPI tasks before the indexes can be written to a file. Since only one thread spawned by a MPI domain can participate in a collective call, we select one thread to be the master thread, and use it for making the collective IO calls. In other words, in each iteration, a thread reads data and constructs indexes independently for a separate sub-array. Next, the indexes from all threads are collected to a shared memory buffer. At that point the master thread can make the HDF5 collective call along with the other MPI tasks, which will create the bitmap dataset and write the indexes to the file.

During query processing, the hybrid parallel FastQuery is able to load indexes from the index file and evaluate bitmaps in-memory without involving any HDF5 collective calls. We implement a hierarchical load balancing strategy. At the MPI level, we use a static assignment to divide sub-arrays to MPI domains evenly, so that no synchronization or communication overhead is introduced among domains. Once groups of sub-arrays are assigned to an MPI task, it becomes a shared working pool among threads. Among the threads, we developed a dynamic load balancing strategy that enable the threads to request sub-arrays one at a time from the common pool. Since a query may produce different number of hits on a sub-array, we expect the dynamic approach to provide better load-balancing and improve overall performance.

C. Query Driven Parallel Visualization with VisIt

We use VisIt for rendering selected output from the particle simulation. Even though VisIt is demonstrated to operate at scale [5], a brute force rendering of one trillion particles is infeasible. Typical computer displays contain O(1M) pixels, which roughly implies an overdraw factor of O(1M) if all particles were transparently rendered. Reducing the number of particles before rendering can be achieved in a number of ways: a statistical down-sampling technique that preserves the statistical characteristics of the data could be used, a 'super-particle' approach (wherein a single representative particle can be rendered instead of a collection) can be used, or a scientifically motivated query-driven criteria can be used to select particles of interest. All three options are feasible, in the current paper, we choose the query-driven option to first down-select scientifically interesting particles, and then visualized them with VisIt [31]. A novel feature that we use in this paper is Cross-Mesh Field Evaluation (CMFE) which enables us to correlate particle data with the underlying magnetic field and evaluate properties such as field direction and field strength at particle locations. We use this powerful feature to address the scientific use cases listed in the previous section.

III. System Configuration

In our work, the test data is produced by running VPIC on the NERSC Cray XE6 system "Hopper." Hopper has $6,384$ twenty-four core nodes with $32GB$ of memory. It employs the Gemini interconnect with a 3D torus topology. The file system storing our data is a Lustre parallel file system with 156 Object Storage Targets (OSTs) and a peak bandwidth of about $35GB/s$.

VPIC uses Cray's MPI library (xt-mpt 5.1.2) and HDF5 version 1.8.8 with local modifications. The particle data is written with H5Part version 1.6.5, along with Cray's MPI-IO implementation. The local modification to the HDF5 library source code disables the *truncate* call, which was causing significant overhead in closing files. We use VisIt 2.4 for our visualization needs, and a development version of FastQuery for all of the results reported in the next section.

IV. Results

A. Parallel I/O in VPIC

The VPIC simulation uses $20,000$ MPI domains - four per node, where each MPI domain spawns 6 OpenMP threads for a total of $120,000$ cores in the simulation. Each MPI domain processes ~ 51 million ($\pm 15\%$) particles. VPIC produces field and particle data, which are periodically dumped to the file system. The field portion of the I/O is about $80GB$ in size and is not considered for the performance study. Each particle has eight four-byte fields, and the dump for all trillion electron particles amounts to $\sim 30TB$.

VPICBench is a parallel I/O kernel that uses the same H5Part calls shown in Section II-A for writing VPIC particle data. This simplified kernel contains the full data volume generated by the code with a slightly simplified pattern. VPICBench disables the simulation component of the VPIC code, which enables testing without exhausting our project's compute allocation. The simplified pattern uses an equal number of particles on all participating cores, whereas the number of particles in a real VPIC run varies across cores by a small amount. The I/O rate for VPICBench (and for VPIC) is the total amount of data written divided by the total time in opening, writing all the variables, and closing the file.

A parallel file system (Lustre in this case) can have a significant impact on performance based on properties of the file established at the time it is opened. The number of I/O resources available (the number of OSTs) can be set as the file's *stripe count*, and the amount of data sent to one OST as a contiguous region of the file can be set as the *stripe size*. We conducted a series of tests with VPICBench ran using $8k$ tasks, and varied the stripe count from 64 OSTs to the maximum of 156. The best performance was at 144 OSTs. Similarly, a series of tests using stripe sizes from $1MB$ to $1GB$ established that choosing $64MB$ gave the best performance. All the tests reported here use stripe count 144 and stripe size $64MB$.

1) Weak scaling study: Figure 1 shows the results of a scaling study for $1K$ to $128K$ MPI tasks. This is a *weak* scaling study in that the number of particles per task is constant at eight million. 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 reduces the maximum I/O rate could be achieved. At the scale of $128K$ cores, VPICBench occupies 85% of Hopper, which reduces the interference from other jobs sharing the I/O system. The $128K$ task instance writes about $32TB$ of data, and Figure 1 shows that at that scale the delivered I/O performance is about $27GB/s$, which compares favorably with the rated maximum on Hopper of about $35GB/s$.

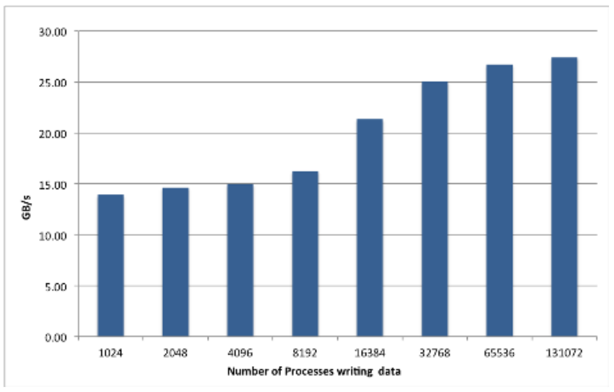

Fig. 1. VPICBench weak scaling study: I/O performance with increasing number of processes writing data to a HDF5 file using H5Part.

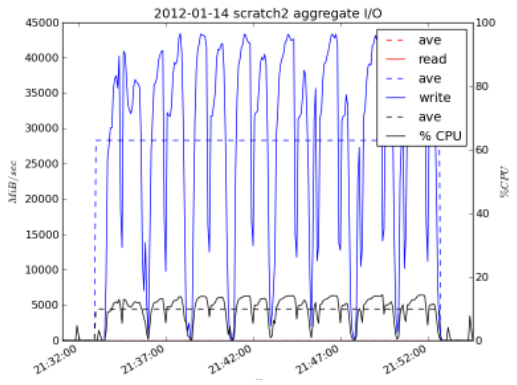

Fig. 2. The eight VPIC variables are written out in sequence, for $32TB$ in the $128K$ task test. Transient I/O rates at the servers can exceed the rated maximum bandwidth for Hopper. The dotted lines for "ave" indicate the actual begin and end of the I/O.

Figure 2 shows the transient I/O rates for the largest VPICBench test in the scaling study. In this case, writing the $32TB$ of data takes around 20 minutes. In the graph, time is along the x-axis and the aggregate observed data rate at the severs is on the y-axis. The data is gathered on Hopper via the Lustre Monitoring Tool (LMT)[13], [34] by recording the server I/O counters (*bytes_read* and *bytes_written*) every five seconds. The difference between successive *bytes_written* gives a data rate for each OST and the sum of those values (across all OSTs) for a five second interval gives the aggregate rate observed by the servers. Note that transient values in the graph can be well above the rated maximum bandwidth for Hopper of $35GB/s$. This is not surprising, since any sustained test of I/O performance is going to amortize very fast transient behavior with other, slower behavior, e.g. while files are being opened or closed. Section IV-A-2 will return to the LMT data while reviewing the results of the trillion particle VPIC I/O.

Historically[1], the performance of MPI-I/O collective, single file I/O was considered inferior to a POSIX, file-per-process I/O model due to concerns with lock contention. Figure 2 shows that the current MPI-I/O and HDF5 libraries can perform quite well, with no obvious penalty for lock

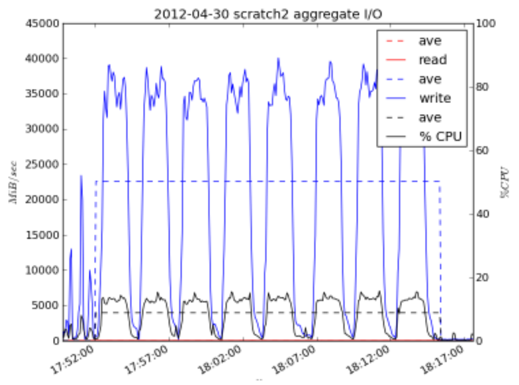

Fig. 3. The $120K$ core VPIC run showed comparable performance except for a couple of slow servers. The slower servers lead to a small amount of I/O continuing after the bulk had completed, and leads to the slightly wider gaps between individual variable dumps.

management. The only odd feature in Figure 2 is that the aggregate rate goes to zero briefly after a variable is written. This is due to an implicit MPI collective operation in the MPI I/O layer, an *MPI_Allgather()*, at the beginning of each variable's I/O in the collective buffering algorithm.

2) Writing one trillion particles: The I/O performance of VPICBench in the weak scaling study was encouraging, and the VPIC case study adopted the same H5Part interface and the same file system tuning parameters. The simulation uses 120,000 cores of hopper. The write phase for a dump produced 30TB, and Figure 3 shows the observed I/O rates.

In Figure 3, the initial spikes are due to the simulation's magnetic field data dump - the small file-per-process phase that is not part of this study. After that, each peak corresponds to writing one of the eight variables of the particle data. In addition to providing a time series plot of the I/O, the LMT data also gives some confidence that no other I/O intensive activity was taking place at the same time on Hopper. All of the I/O in the graph is accounted for by the expected data dump volume. Figure 3 shows transient I/O rates above $35GB/s$ as was the case in Figure 2. The I/O peak for each variable is followed by a short interval of slower I/O activity, which reduces the amortized I/O rate to about $23GB/s$. Two servers shared a failed RAID controller and had their traffic diverted to it's fail-over partner. The twelve affected OSTs ran 30% slower but otherwise performed correctly. That delay shows up as the small continuing I/O following each variable's main peak. Without the faulty OSTs, VPIC's data dump with H5Part and HDF5 will achieve the similar performance seen in the VPICBench.

The question arose as to whether the *fpp* strategy would have performed better. For comparison, a $120K$ core instance of VPIC ran using an *fpp* model writing directly via the POSIX interface. Figure 4 shows the result and also shows two important features. First, there is no pause for an *MPI_Allgather()* between the variables, which gives it a small advantage. Second, the aggregate I/O rate across all the OSTs starts out quite high but then trails off alarmingly.

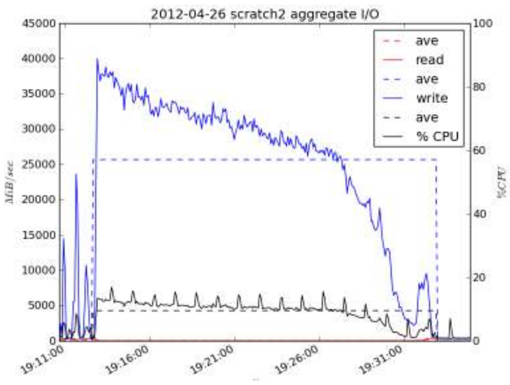

Fig. 4. A 120K core VPIC test run using a file-per-process (*fpp*) model does not show the pause between variables but does exhibit a common feature of fpp runs at scale. The files can end up non-uniformly distributed over the OSTs. The OST with the greatest load takes the longest, slowing down throughput.

This is partly due to the same slow OSTs already mentioned, but is also a common feature of *fpp* I/O at scale[35]. The distribution of the $20,000$ individual files among the OSTs is not uniform. Some OSTs will be assigned significantly more files than others. The OST with the heaviest burden will take the longest, while lightly loaded OSTs complete their work early. That is why the aggregate rate tends to drop towards the end of the job. Nevertheless, the amortized, aggregate data rate was a respectable $26GB/s$. Despite this, the *fpp* I/O model does have disadvantages compared to a single-file I/O model like H5Part/HDF5. For example, with a single file, the file system can apportion data uniformly across OSTs, but with the *fpp* approach, it is very unlikely the data can be allocated evenly across the OSTs. Furthermore, the *fpp* is only effective for writing, but not for later data analyses as discussed before.

Considering the ease of use and metadata management provided by the HDF5 and H5Part libraries, their use is well justified in VPIC. Finally, the barrier between the I/O of each variable in both VPIC and VPICBench is not strictly necessary, and its removal may allow a little more concurrency in the I/O, thereby further improving the performance.

B. Parallel Indexing/Querying

To demonstrate scalability of our indexing and querying approach, we measured FastQuery performance on two sets of VPIC particle data, one with 100 billion electrons and another with one trillion electrons. The data is stored in HDF5 files, with one file per time step. With 100 billion particles, each HDF5 file is $\sim 3.2TB$. As mentioned earlier, with 1 trillion particles, each HDF5 file is $\sim 30TB$. We use the smaller data set to study the performance of building indexes and use the larger data set to study both indexing and querying functions. The visualization tools described in the next section use the indexes generated in this process for accelerating analysis of the particle data.

In our strong scaling study, we vary the number of cores from 500 to 10,000. Given the fixed number of cores, we

TABLE I
THE TOTAL INDEXING TIME (IN SECONDS) FOR 100-BILLION PARTICLE DATASET.

#cores	500	1,250	2,500	5,000	10,000
MPI-alone	1704s	935s	572s	423s	280s
hybrid	1660s	850s	587s	347s	256s

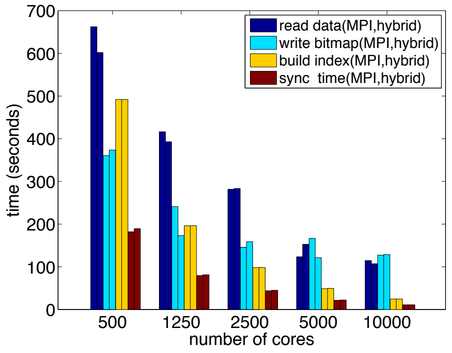

Fig. 5. Time for indexing 100-billion particle dataset with different number of cores.

arrange them either in an MPI-only configuration or in a hybrid parallel configuration. The hybrid configuration launches 3 threads for each MPI process.

Based on earlier study of sub-array size to use for FastQuery, we have chosen the sub-array size to be ∼10 million [8]. We carefully choose this number so that the total number of particles can be evenly distributed among the sub-arrays and the sub-arrays can then be evenly divided among the cores.

1) Strong Scaling: In this strong scaling study, the number of cores increases but the data set of 100-billion particles is the same. We measure the time to index 4 variables (the x, y, and z coordinates, and the energy field), which is half of the variables in the data file. The indexes use a 3-digit precision binning option from the FastBit indexes. This option allows us to answer most of the user queries without going to the raw data while at the same time keep the index size relatively small. The size of the resulting index file is $\sim 1.3TB$, which is about 80% of the original data size ($\sim 1.6TB$) for the corresponding variables.

Table I summarizes the total time spent in building indexes using 500, 1250, 2500, 5000 and 10,000 cores. As shown from

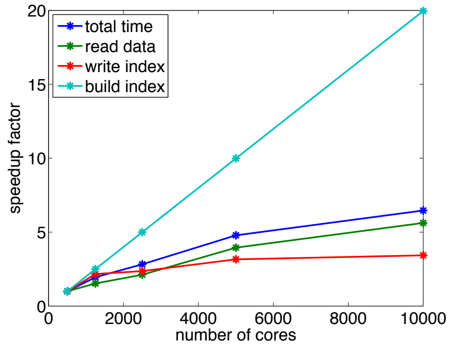

Fig. 6. The speedup factor of each indexing steps for the hybrid configuration.

TABLE II
THE TOTAL TIME (SECONDS) OF QUERYING ON 1-TRILLION PARTICLES.

#cores	scan	MPI-alone	hybrid
250	975	10.1	10.8
500	532	8.6	5.5
1250	266	4.1	2.7

this table, the total time reduces from 30 mins to less than 5 mins as the number cores increases from 500 to 10,000. Furthermore, the hybrid configuration shows consistent improvement over the MPI-alone configuration.

Figure 5 shows the breakdown of total time in building indexes. Since the size of data, 1.6TB, is more than the size of indexes, 1.3TB, it requires more time for reading data than writing bitmaps. However, as shown in Figure 6, the scalability of writing bitmaps is worse than reading data and in-memory computation. Thus, as the number of cores increases, the write time quickly becomes the most significant part of the total indexing time.

While building an index, FastQuery iterates through groups of sub-arrays. The "sync time" shown in Figure 5 measures the delay between two consecutive iterations. This delay is caused by the synchronization implicitly in the HDF5 collective operations, but are not completely captured by the timer around the write operation. In general, as more cores are used, there are fewer iterations and therefore less delays to be accounted for by this "sync time."

From Table I, we see that the MPI-only configuration takes more time than the hybrid configuration. From Figure 5, we see that the in-memory computation time for the two configurations are very close. Therefore, the main difference must come from the I/O time. The key difference between the two configurations in FastQuery is that the hybrid parallel configuration consolidates the write operations into a smaller number of cores than the MPI-only case. In general, reducing the number of concurrent I/O calls can reduce I/O contention and improve I/O throughput [12], [17]. Figure 6 provides another view of the relative efficiency of the three indexing steps by showing the speedup factors. This figure shows that the in-memory computation time is perfectly scalable, but the speedup factor of I/O time gradually decades toward some I/O rate limit. With 10,000 cores, the maximum I/O rate we achieved is around 14GB/s for read and 12GB/s for write.

2) Indexing/Querying Trillion Particles: On the larger data set with 1 trillion electrons, we indexed the variable "Energy" using 10,000 cores. The total time of building the index using MPI-alone configuration is 629 seconds, while using hybrid configuration is 511 seconds. The hybrid configuration used about 18.8% less time than the MPI-only configuration. In the hybrid configuration, FastQuery took 215 seconds to read 3.8TB data from the file, built indexes in 67 seconds and then wrote the 2.6TB indexes to file in 172 seconds. The I/O rate was 17.7GB/s for read and 15.1GB/s for write.

For measuring the time spent in querying functions, we use a sample query of the form "Energy $>$ 1.2." Table II and Figure 7 show the time needed to answer this query on 1 trillion particles. From the total time in Figure II, we see that the time needed to answer the same query without index (marked as scan) is 60 – 100 times longer than using FastBit indexes. Without index, it took more than 4 minutes to scan through the 3.8T data by using 1250 cores. In contrast, with index, the query can be resolved in 10 seconds by using just 250 cores. With 1,250 cores, MPI-alone implementation took 4.7 seconds, but hybrid FastQuery only took 2.7 seconds.

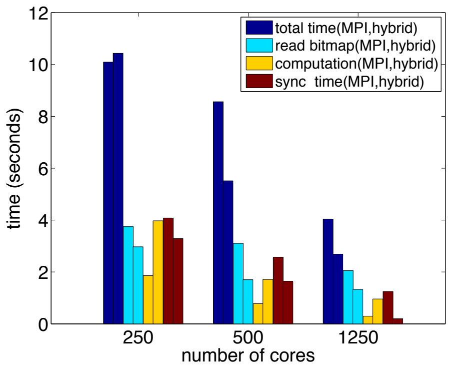

Fig. 7. Time for querying 1-trillion particles with different number of cores.

Between the two configurations of FastQuery, the hybrid option is typically better and in some cases, a lot better. The reduction in execution time seems to be mostly due to the reduce in time needed to perform the read operation according to Figure 7. This agrees with our earlier observations based on indexing time as well as those published in literature. We use the "sync time" to indicate the average time of waiting the last MPI task to finish in Figure 7. Because our hybrid implementation dynamically assigns sub-arrays among threads, the load could be more evenly distributed. Thus we also observed less synchronization time for hybrid implementation.

Overall, hybrid parallel FastQuery performs both indexing and querying more efficiently than the put MPI implementation of FastQuery.

C. Scientific Use cases

We developed a plugin within VisIt that uses the hybrid parallel FastQuery software for parallel evaluation of queries on H5Part files. The plugin is capable of operating in parallel on distributed memory nodes. Armed with this powerful capability, we now revisit the scientific questions postulated in Section I-B.

1) Analysis of highly energetic particles: Identification of mechanisms leading to particle energization remains an important unsolved problem in plasma physics. There are indications that the energization mechanism may be different in 2D and 3D models. A critical analysis capability for identification of acceleration mechanism is the ability to i) determine preferential acceleration direction with respect to local magnetic field and ii) determine where energetic particles are located and how their concentration correlates with magnetic field structures. Specifically, an important physics result obtained in 3D magnetic reconnection simulations of the type considered here is the formation, evolution, and interaction of so-called

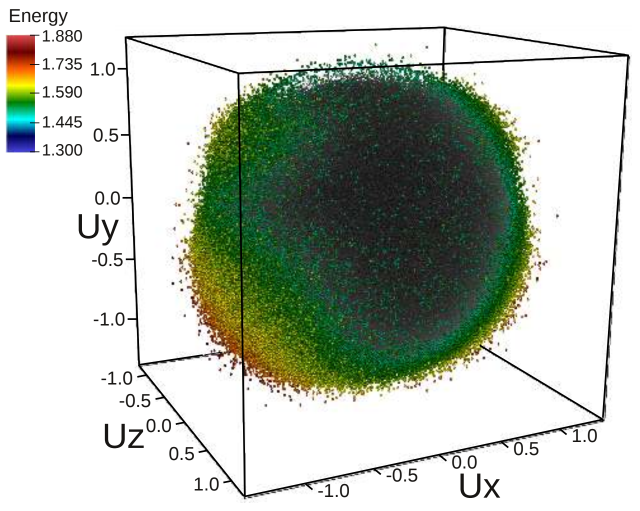

Fig. 8. Visualization of the 1 trillion electron dataset at timestep 1905 showing all particles with Energy > 1.3 (gray). In addition all particles with Energy > 1.5 are shown in color, with color indicating Energy. A total of $164,856,597$ particles with Energy > 1.3 and $423,998$ particles with Energy > 1.5. The particles appears to be accelerated preferentially along the direction of the mean magnetic field (oriented at $45°$ in the $x-y$ plane), corresponding to formation of four jets. The distribution of energetic particles is asymmetric, with the most energetic particles acquiring negative U_y.

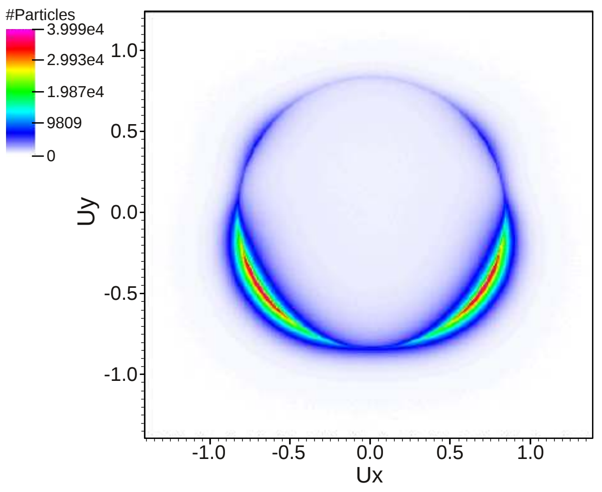

Fig. 9. Visualization of the 1 trillion electron dataset at timestep 1905 showing the density of all particles with Energy >1.3 (see also Figure 8).

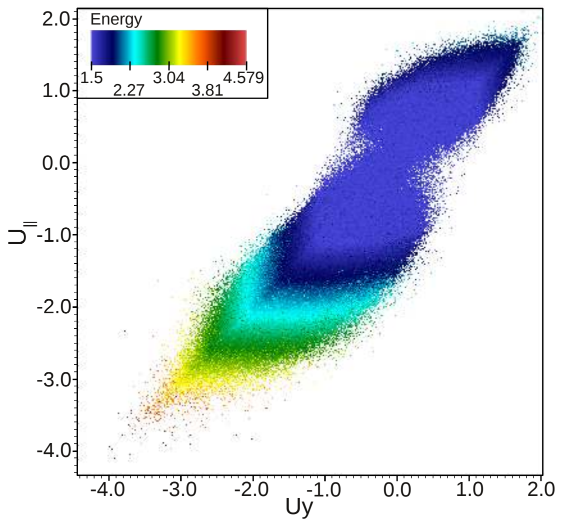

Fig. 10. Scatter plot showing all particles with Energy $>$ 1.5 (see also Figure 11) in Uy and $U_{||}$ space colored by Energy. We observe a strong positive correlation between Uy and $U_{||}$. The particles of highest Energy appear in regions of high negative $U_{||}$ (and Uy) values, indicating that the high energy particles are aligned (i.e., move parallel) to the magnetic field.

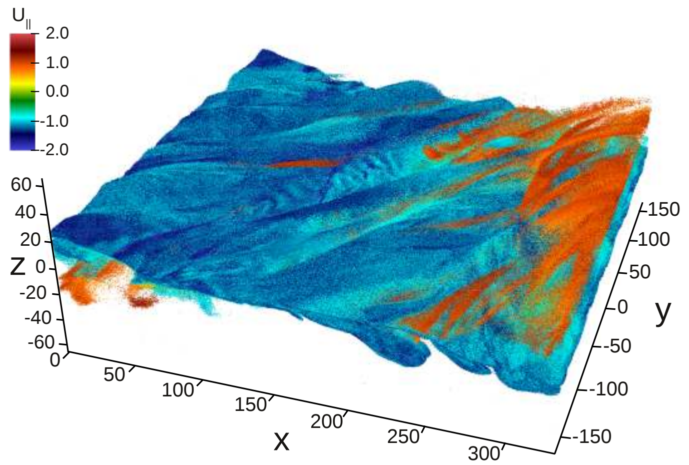

Fig. 11. Plot showing all particles with Energy $>$ 1.5. The query selects 57,740,614 out of the 114,875,956,837 particles, i.e., $\approx 0.05\%$ of all particles. Color indicates $U_{||}$. We observe different particle structures with strong positive (red) and negative (blue) $U_{||}$ values.

flux ropes — twisted bundles of the magnetic field lines. Some of the unresolved issues include the association of the energetic particles with flux ropes, the contribution of energetic particles to the overall current, and whether their energy predominantly corresponds to the motion parallel to the magnetic field. We applied the visualization tools developed in this paper in order to address the questions posed in Section I-B.

Are the highly energetic particles preferentially accelerated along the the magnetic field?

Figures 8 and 9 show the phase space of particles with energies > 1.3 from the 1 trillion particle dataset. Even though the dataset corresponds to an early time in the simulation, these two figures clearly show that magnetic reconnection has already started. Phase space formation of reconnection generated energetic jets at $45°$ in the $x-y$ plane, corresponding to the direction of average magnetic field, is apparent, especially in the 2D density plot in the $U_x - U_y$ plane (Fig. 9). These figures also show evidence of preferential acceleration of the plasma in the direction parallel to the average magnetic field as evidenced by the highly distorted distribution function in the $x-y$ plane in Figure 8. Another important finding evident from the phase space figures is that energetic particles carry significant current. These two findings, enabled for the first time through the new analysis capabilities discussed here, are quite encouraging and are leading us to formulate new questions regarding the particle behavior in 3D reconnection.

In order to understand properties of the energetic particle at

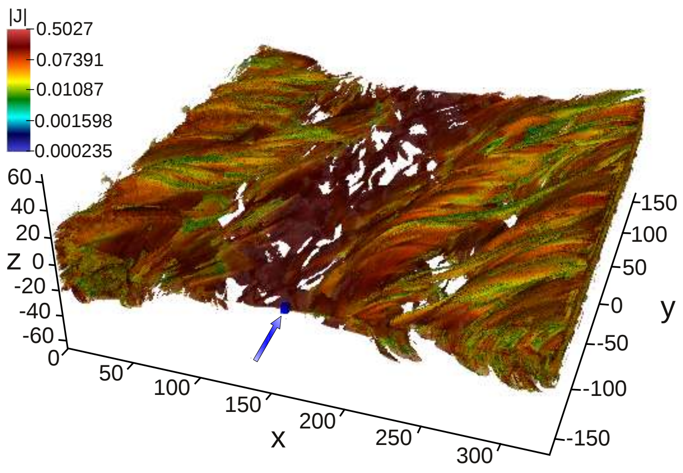

Fig. 12. Iso-surface plot of the positron particle density n_p with color indicating the magnitude of the total current density $|J|$. Note the logarithmic color scale. The blue box (indicated by the arrow) is located in the X-line region of the simulation and illustrates the query $(157.654 < x < 1652.441)\&\&(-165 < y < -160.025)\&\&(-2.5607 < z < 2.5607)$, which we use in Figure 13 to study agyrotropy.

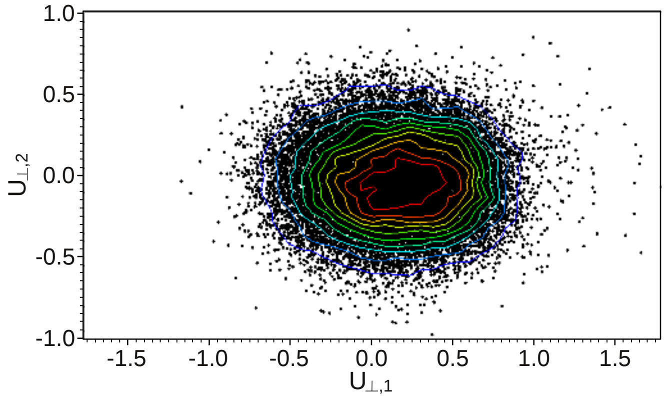

Fig. 13. Particle scatter plot (black) of $U_{\perp,1}$ vs. $U_{\perp,2}$ of all energetic particles (with $Energy > 1.3$) contained in the box in the x-line region indicated in Figure 12. Additional iso-contours indicate the associated particle density (blue=low density and red=high density). The complete query used to extract the particles is defined as: $(Energy > 1.3)\&\&(157.654 < x < 162.441)\&\&(-165 < y < -160.025)\&\&(-2.5607 < z < 2.5607)$. The query results in a total of 22,812 particles. The elliptical shape of the particle distribution is indicative of agyrotropy in the x-line region.

later timesteps, when the dynamics have evolved sufficiently far away from the initial conditions, we considered a 100-billion-particle simulation with equivalent physics[2]. We applied query based techniques to create a scatter plot of U_y vs $U_\parallel$ in Fig. 10. Two important results can be immediately deduced from the plot: i) the highest energy particles tend to have $U_\parallel \sim U_y$, which indicates that they are localized in the reconnection regions where the in-plane magnetic field vanishes and ii) the plot is asymmetric, with the highest energy particles having negative values of $U_\parallel$, indicating that they carry significant current in agreement with the analysis of the trillion particle simulation.

What is the spatial distribution of highly energetic particles?

As is illustrated by Fig. 11, energetic particles are predominantly located within the current sheet, suggesting they carry significant current. These results also suggest that the flux ropes can confine energetic particles (as illustrated by the red regions in Fig. 11), but more careful analysis is needed to resolve this issue, which is beyond the scope of this paper.

2) What are the properties of particles near the reconnection hot-spot?: Fig. 13 shows the particle distribution $F(U_{\perp,1}, U_{\perp,2})$ in the vicinity of an X-line. The particles are selected in a small box, as indicated in Fig. 12. The distribution clearly shows the agyrotropy of the distribution, i.e. the lack of cylindrical symmetry about the local magnetic field. Agyrotropy is an expected signature of the reconnection site in collisionless plasma. While it has been well-documented in simple 2D simulations, classification of agyrotropic distributions in 3D simulations have been much more challenging. While some information about agyrotropy can be recovered from coarser-level moment computations, a direct computation based on particle data provides richer information about the structure of particle phase space. With these new capabilities, we are now well poised to compute agyrotropy and other finer characterizations of distribution functions.

To summarize, the query-based visualization techniques presented in this paper have enabled us to explore and gain insights from massive particle datasets for the first time. We have verified localization behavior of energetic particles, gained insights into relationship between the structure of magnetic field and energetic particles, and discovered agyrotropic distribution of particles near the reconnection hot-spot in 3D. Several of these phenomena have been conjectured about in the past, but it is only by the development and application of these new analysis capabilities that we can unlock the scientific discoveries and insights present in these unprecedented simulations.

V. Related Work

A. Parallel I/O

High-level libraries such as Parallel netCDF (PnetCDF) [23] and ADIOS [22], [25], [24] provide support for writing and reading large files. PnetCDF is developed to perform parallel I/O operations on files larger than 4GB in size. The ADIOS library has demonstrated high I/O rates in writing large-scale simulation data. ADIOS provides a light-weight API for applications to modify their I/O interface and write data into a newly introduced BP format. ADIOS also provides various tools for converting data from BP to the standard file formats, such as netCDF5 HDF5. While the conversion cost is linear with respect to data sizes [24], for analyses and visualizations that touch datasets on the order of TB the cost can be very high. Both PnetCDF [11] and ADIOS support writing data into subfiles to reduce the number of nodes writing data to OSTs. To reduce the number of writers in file-per-process approach of writing data, Karimabadi et al. [20] used a technique called gating. This technique partially serializes I/O by controlling

[2]Due to a compute node failure, the trillion particle simulation did not produce data for timesteps later than 1905. We expect to have data for later timesteps of the trillion particle dataset available soon.

the number of processes that can write data concurrently. Filesystem-aware MPI-IO implementations, such as Cray's MPI library, optimize the number of aggregator nodes directly interacting with OSTs in writing or reading data [9].

In this work, we choose to use a particle data extension of HDF5, called H5Part, because its API conveniently match with the application of interest. It provides good read and write performance for the specific application while other formats maybe efficient for write only or read only.

B. Analysis

Most analysis systems assume the whole dataset could be stored in memory. As data sets grow in size, the analysis operations are forced to concentrate on the most relevant data records to reduce the memory requirement. Here we briefly mention a few examples that integrate querying functions with visualization and analysis [19], [29]. One of the earliest example is the VisDB system, which combines a guided query-formulation facility with relevance-based visualization [21]. Data items are ranked in terms of relevance to a query, and the top quartile of most relevant results are then input to a visualization and rendering pipeline. This approach examines all data records in order to determine relevance, even though it only displays the most relevant records. Another early system is the TimeFinder system [15], which supports interactive exploration of time-varying data sets. It provides a way to quickly construct queries, modify parameters, and visualize query results. However, it also needs to examine all data records in order to answer these queries.

To speed up the selection process, there has been a number of efforts on query-driven visualization and analysis which make use of database indexing techniques to accelerate data queries [31]. We have chosen to use a set of efficient bitmap indexing techniques in FastBit [37], [38] because they have been demonstrated to work well on scientific data [30], [27]. Rübel et al. demonstrated the use of FastBit to accelerate query driven visualization of laser plasma accelerator simulations containing on the order 100s of millions of particles per timestep [27], [26]. Evaluation of queries for single files were performed in serial in these efforts. In order to be able to evaluate queries efficiently also for trillions of particles, we integrated FastQuery with VisIt, enabling parallel evaluation of queries for massive data files.

VI. Conclusions

In this paper, we have addressed data management and analysis challenges posed by a highly scalable, plasma physics simulation that writes one trillion particles. On the parallel I/O front, we demonstrated state-of-the-art collective write performance using H5Part and HDF5 to a single, shared 30TB file. We demonstrate a write performance of 23GB/s, and peak rates utilizing the entire system I/O bandwidth. Without hardware and file system failures that we experienced, this I/O rate will be even higher.

We developed and applied a hybrid parallel version of FastQuery to index and query the trillion particle dataset. We show strong scaling for FastQuery up to 10,000 cores, and demonstrate indexing times of ≈ 9 minutes and querying times of ≈ 3 seconds to process the trillion particle dataset.

We apply query-driven visualization to render selected particles of interest in VisIt. We apply these techniques to address open scientific problems in plasma physics, and demonstrate that our approach holds much promise for data-driven scientific discovery for the future. The test runs of the new software have provided strong evidence for confirming the agyrotropy near X-line and preferential acceleration of energetic particles along the magnetic field direction. These insights are only possible with advanced data analysis techniques developed here.

Acknowledgment

This work was 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 was supported in part by National Science Foundation under NSF grant OCI 0904734. This research used resources of the National Energy Research Scientific Computing Center. Simulations were also supported by an allocation of advanced computing resources provided the NSF at the National Institute for Computational Sciences and by NASA (Pleiades), and the National Center for Computational Sciences at Oak Ridge National Laboratory (Jaguar).

The authors would like to thank NERSC and Cray staff for troubleshooting I/O issues on hopper. We would also like to thank members of the HDF Group for their advice on HDF5 I/O optimizations, and Burlen Loring for his advice and support.

References

[1] K. Antypas and A. Uselton. MPI-I/O on Franklin XT4 System at NERSC. In *52nd Cray User Group Conference*, Edinburgh, UK, 2010.

[2] IPCC Fifth Assessment Report. http://en.wikipedia.org/wiki/IPCC_Fifth_Assessment_Report.

[3] P. Balaji, A. Chan, W. Gropp, R. Thakur, and E. L. Lusk. Non-data-communication overheads in MPI: Analysis on blue gene/P. In A. L. Lastovetsky, M. T. Kechadi, and J. Dongarra, editors, *PVM/MPI*, volume 5205 of *Lecture Notes in Computer Science*, pages 13–22. Springer, 2008.

[4] 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.

[5] H. Childs, D. Pugmire, S. Ahern, B. Whitlock, M. Howison, Prabhat, G. H. Weber, and E. W. Bethel. Extreme scaling of production visualization software on diverse architectures. *IEEE Computer Graphics and Applications*, 30:22–31, 2010.

[6] J. Chou, K. Wu, and Prabhat. FastQuery: A general indexing and querying system for scientific data. In *SSDBM*, pages 573–574, 2011. http://dx.doi.org/10.1007/978-3-642-22351-8_42.

[7] J. Chou, K. Wu, and Prabhat. FastQuery: A parallel indexing system for scientific data. In *IASDS*. IEEE, 2011.

[8] J. Chou, K. Wu, O. Rübel, M. Howison, J. Qiang, Prabhat, B. Austin, E. W. Bethel, R. D. Ryne, and A. Shoshani. Parallel index and query for large scale data analysis. In *SC11*, 2011.

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

[10] 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(7):539–542, July 2011.

[11] K. Gao, W. keng Liao, A. Nisar, A. Choudhary, R. Ross, and R. Latham. Using subfiling to improve programming flexibility and performance of parallel shared-file I/O. In *Proceedings of the 2009 International Conference on Parallel Processing*, ICPP '09, pages 470–477, Washington, DC, USA, 2009. IEEE Computer Society.

[12] D. S. Henty. Performance of hybrid message-passing and shared-memory parallelism for discrete element modeling. In *SC'00*, Washington, DC, USA, 2000. IEEE Computer Society.

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

[14] T. Hey, S. Tansley, and K. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft, Oct. 2009.

[15] H. Hochheiser and B. Shneiderman. Visual specification of queries for finding patterns in time-series data. In *Proceedings of Discovery Science*, pages 441–446, 2001.

[16] 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.

[17] M. Howison, E. W. Bethel, and H. Childs. MPI-hybrid parallelism for volume rendering on large, multi-core systems. In *Eurographics Symposium on Parallel Graphics and Visualization*, pages 1–10, 2010.

[18] 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.

[19] C. R. Johnson and J. Huang. Distribution-driven visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):734–746, Sept. 2009.

[20] H. Karimabadi, B. Loring, A. Majumdar, and M. Tatineni. I/O strategies for massively parallel kinetic simulations, 2010.

[21] D. Keim and H.-P. Kriegel. VisDB: Database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications*, 14(4):40–49, 1994.

[22] ADIOS. http://www.nccs.gov/user-support/center-projects/adios/.

[23] J. Li, W. keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. Parallel netCDF: A high-performance scientific I/O interface. In *SC'03*, page 39, New York, NY, USA, 2003. ACM.

[24] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich IO methods for portable high performance IO. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, IPDPS '09, pages 1–10, Washington, DC, USA, 2009. IEEE Computer Society.

[25] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In *CLADE'08*, pages 15–24, New York, NY, USA, 2008. ACM.

[26] O. Rübel, C. G. R. Geddes, E. Cormier-Michel, K. Wu, Prabhat, G. H. Weber, D. M. Ushizima, P. Messmer, H. Hagen, B. Hamann, and W. Bethel. Automatic beam path analysis of laser wakefield particle acceleration data. *IOP Computational Science & Discovery*, 2(015005 (38pp)), November 2009.

[27] O. Rübel, Prabhat, K. Wu, H. Childs, J. Meredith, C. G. R. Geddes, E. Cormier-Michel, S. Ahern, G. H. weber, P. Messmer, H. Hagen, B. Hamann, and E. W. Bethel. High Performance Multivariate Visual Data Exploration for Extemely Large Data. In *SuperComputing 2008 (SC08)*, Austin, Texas, USA, Nov. 2008.

[28] A. Shoshani and D. Rotem, editors. *Scientific Data Management: Challenges, Technology, and Deployment*. Chapman & Hall/CRC Press, 2010.

[29] G. Smith, M. Czerwinski, B. Meyers, D. Robbins, G. Robertson, and D. S. Tan. Facetmap: A scalable search and browse visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):797–804, Sept. 2006.

[30] K. Stockinger, E. W. Bethel, S. Campbell, E. Dart, , and K. Wu. Detecting Distributed Scans Using High-Performance Query-Driven Visualization. In *SC '06: Proceedings of the 2006 ACM/IEEE Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, Nov. 2006.

[31] K. Stockinger, J. Shalf, W. Bethel, and K. Wu. Query-driven visualization of large data sets. In *IEEE Visualization 2005, Minneapolis, MN, October 23-28, 2005*, page 22, 2005. http://doi.ieeecomputersociety.org/10.1109/VIS.2005.84.

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

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

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

[35] A. Uselton and B. Behlendorf. Visualizing I/O performance during the BGL deployment. In *8th LCI Conference on High-Performance Clustered Computing*, South Lake Tahoe, CA, 2007.

[36] K. Wu. FastBit: an efficient indexing technology for accelerating data-intensive science. *Journal of Physics: Conference Series*, 16:556–560, 2005. http://dx.doi.org/10.1088/1742-6596/16/1/077.

[37] K. Wu, S. Ahern, E. W. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. Geddes, J. Gu, H. Hagen, B. Hamann, W. Koegler, J. Lauret, J. Meredith, P. Messmer, E. Otoo, V. Perevoztchikov, A. Poskanzer, Prabhat, O. Rubel, A. Shoshani, A. Sim, K. Stockinger, G. Weber, and W.-M. Zhang. FastBit: Interactively searching massive data. In *SciDAC*, 2009.

[38] K. Wu, A. Shoshani, and K. Stockinger. Analyses of multi-level and multi-component compressed bitmap indexes. *ACM Transactions on Database Systems*, pages 1–52, 2010.