Terabyte-scale Particle Data Analysis: An ArrayUDF Case Study

Bin Dong[1], Patrick Kilian[2], Xiaocan Li[2], Fan Guo[2], Suren Byna[1], Kesheng Wu[1]
dbin@lbl.gov,{pkilian,xiaocanli,guofan}@lanl.gov,{sbyna,kwu}@lbl.gov
[1]Lawrence Berkeley National Laboratory, Berkeley, CA, USA, [2]Los Alamos National Laboratory, Los Alamos, NM, USA

ABSTRACT

A prime question for plasma physicists is how a fraction of charged particles is accelerated to very high energy.To answer this question, physicists simulate trillions of particles with detailed dynamics and analyze their trajectories. This process requires a range of data analysis tasks with high diversity. In this paper, we present a use case of formulating various analysis tasks on terabyte-scale particle data with a novel data analysis framework called ArrayUDF. The flexibility of ArrayUDF allows it to compose a wide range of particle data operations. We also present optimization strategies to avoid frequent global reduction and to take full advantage of the data locality. Tests show that our optimization methods could accelerate these particle data analysis operations by up to 1,600 times.

CCS CONCEPTS

• **Information systems** → Parallel and distributed DBMSs.

1 INTRODUCTION

Modern science discoveries are data driven. VPIC, a first-principle 3D electromagnetic kinetic particle-in-cell plasma code [1, 3, 8], is a representative scientific application that produces a large amount of data and the data needs to be extensively analyzed to gain insights. The challenges in analyzing the VPIC data are threefold: complexity, diversity, and scalability. The VPIC datasets have complex data structures from one-dimensional arrays to three-dimensional arrays. Different domain scientists, or even a single domain scientist, may apply different analysis operations on the data for different purposes. A VPIC simulation may run hundred thousands of time steps and the data size for a single time step could accumulate to tens to hundreds of terabytes [3].

Domain scientists now usually spend a lot of time and effort in developing code for analyzing VPIC data from scratch. This is especially true for particle acceleration studies, where domain scientists have to add extra analysis for particle trajectories and statistics [7, 8]. One major obstacle to use popular database management systems (DBMS) to perform these VPIC data analysis tasks is the significant overhead of loading the data into a DBMS. In addition, database systems targeting scientific data, such as SciDB [2], are not designed for high performance computing (HPC) systems and therefore suffer from scalability issues [5]. Modern data analysis systems, e.g., MapReduce, Spark [4], obtain high scalability, but do not intrinsically support array data model. We have developed ArrayUDF [5], which is a novel data analysis system targeting arrays. In terms of scalability and flexibility in supporting various analysis operations, the superiority of ArrayUDF over SciDB and Spark has been demonstrated [5]. However, ArrayUDF has not been battle-tested using massive and complex science datasets, such as those of VPIC. We target the following questions in this paper: *How does ArrayUDF perform to analyze terabyte-scale data? What performance optimizations are needed in ArrayUDF at this scale?*

This paper presents a case study for using ArrayUDF [5] to compose VPIC data analysis operations. We directly define and execute these analysis operations on the multi-dimensional arrays that are stored in HDF5 [9] files produced by VPIC simulation. We also explored different optimization techniques available in ArrayUDF to improve the performance of these analysis operations. In summary, contributions in this paper include:

- Describe a number of representative particle data analysis operations using the ArrayUDF. These operations involve complex computations requiring multiple arrays representing VPIC field data, particle data, and mesh metadata.
- An implementation of the analysis operations in ArrayUDF through its UDF interface (i.e., *Apply*) and *Stencil* data abstraction.
- We describe how to use advanced features of ArrayUDF to accelerate the performance of user analysis operations. This optimizations include a two-level *Reduce* method and a strategy to fully utilize particle locality.

We demonstrate performance and productivity benefits of using ArrayUDF to perform VPIC data analysis on Cori, a Cray XC40 system at the National Energy Research Scientific Computing Center (NERSC). We observed that the advanced features of ArrayUDF can accelerate the analysis operations up to 1600X in the tests.

2 ArrayUDF

ArrayUDF [5] is a novel data analysis framework that supports analyzing arrays natively. In specific, ArrayUDF provides generic user-defined functions (UDF) for users to customize analysis operations on arrays. The execution engine of ArrayUDF runs the UDF automatically on HPC systems with tens thousands of computing nodes. In Figure 1, we present an overview of ArrayUDF, showing an example of a UDF with a 2-D input array A and a 2-D output array B. The array A has two chunks, which are processed concurrently using two CPU cores. To avoid possible communication at chunk boundary cells, a ghost zone layer is added to the chunks when they are read from persistent storage (i.e., disk or SSD) to memory for processing.

Another key concept of ArrayUDF is the *Stencil* data abstraction (denoted with S) for UDF declaration. The Stencil represents a geometric neighborhood of an array. The *Stencil* has a *center cell* that is denoted with $S_{0,0,\ldots}$. Other neighborhood cells are represented

SSDBM '19, July 23–25, 2019, Santa Cruz, CA, USA

ACM ISBN 978-1-4503-6216-0/19/07…$15.00
https://doi.org/10.1145/3335783.3335805

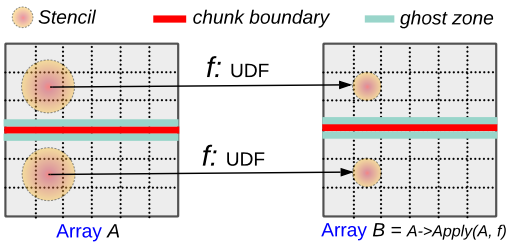


Figure 1: Demonstration of the generic idea of ArrayUDF by applying a user-defined function (UDF) f from array A to B.

```
#include "array_udf.h"
//Define a new avg function with the Stencil
float my_avg_udf(Stencil<float> S){
  return (S(0,0)+S(-1,0)+S(0,1)+S(1,0)+S(-1,0))/5;
}
//Apply my_avg on array A, output is ignored
void main(int argc, char *argv[]){
 std::vector<int>  cs{10,10};//chunk size
 std::vector<int>  gs{1,1};//ghost zone size
 Array<float>  A("file.h5:/data",cs,gs);
 A->Apply(my_avg_udf);
}
```

Figure 2: ArrayUDF C++ example code for user-defined average on a two dimensional array. The array is stored in a HDF5 dataset "/data" within a file "file.h5". In the following texts, we ignore the "main" function and focus on the UDF.

with $S_{i,j,\ldots}$, where i and j are relative offsets from the center cell. *Apply* function is the core of the ArrayUDF execution engine. In Figure 1, the execution engine creates *Stencil* instances for each cell of array A, executes UDF (f) on the *Stencil* instances, and stores the resulting *Stencil* in array B.

ArrayUDF is implemented in C++[1]. In Figure 2, we show an example of using it to define an averaging function. The "array_udf.h" header file provides declarations for *Array* and *Stencil*. The implementation uses MPI [6] for parallel programming and uses HDF5 [9] as the I/O library. The defined function is "Applied" onto an array in the main function. After compiling the code, it can run on multiple CPU cores of a single node or of multiple nodes.

3 VPIC DATA

One major task of plasma physics nowadays is to simulate and understand the plasma behavior, magnetic field evolution, and particle dynamics. A recent simulation using VPIC [3] is a 3D fully kinetic plasma simulation, which simulated a billion field cells and a trillion particles with hundred thousands of time steps.

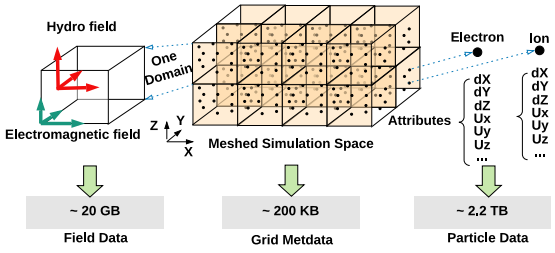


Figure 3: Overview of VPIC simulation and its data.

In Figure 3, we show an overview of VPIC simulation datasets. Overall, VPIC breaks the simulation space into mesh cells. The particles are freely moved among these cells. VPIC generates three types of data: particle data, field data, and metadata. The particle data contains the properties of each particle, which include the relative location (dX,dY and dZ) of a particle within a cell. In this case study, we use a 2TB particle data with billions of particles. There are two types of fields: electromagnetic and hydro. Each of them is a multidimensional array that follows the structure of the simulation mesh. The electromagnetic field has six attributes that are required for the self-consistent evolution of the system: ex, ey, ez, bx, by and bz. VPIC in principle can simulate more than two species (electrons and ions), so the output could contain a large number of hydro fields. Each particle species has its own hydro fields, e.g., j_x, j_y, and j_z. The metadata describes the mesh structure, e.g., the number of cells and the size for each cell in all directions.

4 VPIC DATA ANALYSIS AND ArrayUDF

In this section, we describe some common analysis tasks which plasma physicists perform on the VPIC datasets.

4.1 Field Data Analysis

Analysis Formulation. A critical analysis operation on the field data is to calculate the total current and root-mean-square (RMS) of the absolute value of the current. The total current for a single mesh point is the sum of the current from the electrons $\vec{J_e}$ and the ions $\vec{J_i}$. Both $\vec{J_e}$ and $\vec{J_i}$ are vectors with three components in x, y, and z directions. One statistical property is the root-mean-square of the total current. In summary, for each mesh point, the total current and RMS of the total current can be formulated as,

$$\vec{J_t} = \vec{J_e} + \vec{J_i}, \tag{1}$$

$$J_{rms} = \frac{1}{N}\sqrt{\sum_{i,j,k} \vec{J_t}^2} \tag{2}$$

where N is the amount of points. The $\vec{J_t}^2$ is defined as $J_{tx}^2 + J_{ty}^2 + J_{tz}^2$, where J_{tx}, J_{ty} and J_{tz} are its components in x, y, z directions.

ArrayUDF Implementation. In Figure 4, we show the implementation of above equations in ArrayUDF. In the "total_current_udf" function, its input is *Stencil J* with two attributes, "e" and "i", where "e" is the electron current and "i" is the ion current. We use the virtual array of ArrayUDF to merge these two attributes into a single *Stencil*. The "rms_current_udf" function adds the root-square value of each point into "current_sq_sum", which is a global reduction function. As discussed in Section 2, both UDF functions can be applied to run over the whole field data arrays in parallel.

```
//Define a new avg on the Stencil
float total_current_udf(Stencil<float> J){
  return J.e + J.i;
}
Global current_sq_sum = 0
void rms_current_udf(Stencil<float> J){
  current_sq_sum += J^2
}
rms_current = sqrt(current_sq_sum) / N
```

Figure 4: Pseudocode of UDF functions in ArrayUDF for VPIC field data analysis.

Optimizations. One significant performance overhead of the UDF in Figure 4 is to update "current_sq_sum". When the UDF runs in parallel (e.g., 1000 CPU cores), it needs a global communication

[1] https://bitbucket.org/arrayudf/

to update "current_sq_sum" during each call of "rms_current_udf". The number of times for calling "rms_current_udf" is equal to the number of mesh points. Our implementation (in Figure 5) introduces a two-level reduction, where "current_sq_local_sum" is a variable for each CPU core and "rms_current_udf_v2" only adds its values to its local variable first. Once the "rms_current_udf_v2" is finished locally, a global reduction method "Reduce" adds all the local variables. Therefore, the global communication happens only once and the performance of the analysis code is improved.

```
Local current_sq_local_sum
void rms_current_udf_v2(Stencil<float> J⃗){
    current_sq_local_sum += J⃗²
}
Reduce(current_sq_local_sum, SUM, current_sq_global_sum)
rms_current = sqrt(current_sq_global_sum) / N
```

Figure 5: Pseudocode of optimized UDF functions in ArrayUDF for VPIC field data analysis.

```
float global_x_udf(Stencil<float> p){
  int i = p(0).i;
  return (i % (n_x[i]+2)+(pt(0).dX-1)/2.0)*d_x[i]+x0[i]
}
```

Figure 6: Pseudocode of UDFs to find global X for a particle. The input $Stencil$ has two attributes: the cell index i and local position dX. The UDFs for finding global Y and Z have the same shape and therefore we ignore them in the figure.

4.2 Particle Data Analysis

Analysis Formulation. An important analysis operation on particle data is to find the global position for each particle in the simulation space. During the VPIC simulation, particles are managed by a local domain, which is a group of cells on a single CPU core. Therefore, the location of the particles is recorded as dX, dY, and dZ, which are the relative location inside the cell containing the particle. The metadata of the VPIC records the locations (lower-left corner) of each domain as x_0, y_0, and z_0. The number of cells (n_x, n_y, and n_z) of the whole simulation space are also recorded in the metadata. The challenges of this data analysis is to transfer this local position of a particle to its global positions, denoted as x_g, y_g, and z_g. In all, the method to obtain the global location for a particle is presented as below:

$$x_g = \left(i \mod (n_x + 2) + \frac{dX - 1}{2.0}\right) \times dx + x_0, \tag{3}$$

$$y_g = \left(\frac{i}{n_x + 2} \mod (n_y + 2) + \frac{dY - 1}{2.0}\right) \times dy + y_0, \tag{4}$$

$$z_g = \left(\frac{i}{(n_x + 2) \times (n_y + 2)} + \frac{dZ - 1}{2.0}\right) \times dz + z_0, \tag{5}$$

In these equations, i is the domain index and d_x, d_y and d_z are the size of each cell from metadata.

ArrayUDF Implementation. The implementation of the UDF for ArrayUDF to find the global location of a particle is shown in Fig. 6. This UDF also uses the metadata arrays, n_x, dx and $x0$. These global metadata arrays are preloaded into memory before applying UDF. Its operator "[]" can return the value at an offset of "i".

```
float global_x_udf_v2(Stencil<float> p){
  static int i = -1;
  static float nx_v, dx_v, x0_v;
  if( i != p(0).i){
    nx_v = n_x[i]; dx_v = d_x[i]; x0_v = x0[i];
    i = p(0).i;
  }
  return (i%(nx_v+2)+(pt.dX-1)/2.0)*dx_v+x0_v
}
```

Figure 7: Pseudocode of optimized ArrayUDF UDF functions to find global X for a particle.

Optimizations. Recall that the particle data may be 2TB with billions of particles. Hence, applying "global_x_udf" to each particle can be tedious process. To reduce the time, we study the data locality of particle data. We found out that the particles belonging to the same domain are stored contiguously on the disk. Hence, when a single particle calls "global_x_udf", the following particles have a very high chance of belonging to the same cell. Based on this assumption, we present an optimized version of the UDF function in Fig 7. We declare the "i", "nx_v", "dx_v", and "x0_v" as static variables. Hence, if the next particle has the same cell, metadata can be reused without accessing metadata arrays. By avoiding slow memory access, we can reduce the overall data analysis time.

```
float inter_ex_udf(Stencil<float> p){
 i = ⌊(p.x_g + s_x)/d_x⌋; j = ⌊(p.y_g + s_y)/d_y⌋; k = ⌊(p.z_g + s_z)/d_z⌋;
 r_x = mod(p.x_g, d_x); r_y = mod(p.y_g, d_y); r_z = mod(p.z_g, d_z);
 ex00 = EX[i,j,k];        ex10 = EX[i,j + 1,k];
 ex01 = EX[i,j,k+1];      ex11 = EX[i,j + 1,k + 1];
 return ((1-r_y)*(1-r_z)*ex00+(1+r_y)*(1-r_z)*ex10+(1-r_y)\
        *(1+r_z)*ex01+(1+r_y)*(1+r_z)*ex11)/4.0;
}
```

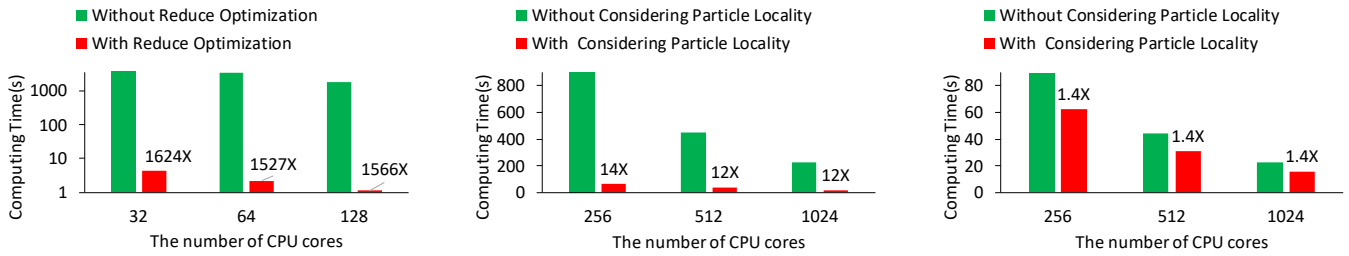
Figure 8: Pseudocode of UDF in ArrayUDF to interpolate EX to a particle. Other fields have the same shape.The s_x, s_y amd s_z are the shift of the particle from the original.

4.3 Fusing Fields and Particles Data

Analysis Formulation. The field data is recorded on the edge or face of the mesh. Particles are scattered over the whole space. One interesting analysis for domain scientists is to interpolate the field from the mesh to all particles. This interpolation is like a "join" operation among the particle data, field data and the metadata. Assume that the field data is represented by "EX", which is a 3D array. The d_x, d_y and d_z are the size of each cell. The i, j and k are the index of the domain in "EX". The interpolation from "EX" to a particle at dZ, dY, dZ within a domain is :

$$\begin{aligned} E_{px} =& (1 - dY)(1 - dZ)E_x(i \times d_x, (j - 0.5) \times d_y, (k - 0.5) \times d_z)/4 + \\ & (1 + dY)(1 - dZ)E_x(i \times d_x, (j + 0.5) \times d_y, (k - 0.5) \times d_z)/4 + \\ & (1 - dY)(1 + dZ)E_x(i \times d_x, (j - 0.5) \times d_y, (k + 0.5) \times d_z)/4 + \\ & (1 + dY)(1 + dZ)E_x(i \times d_x, (j + 0.5) \times d_y, (k + 0.5) \times d_z)/4 \end{aligned} \tag{6}$$

ArrayUDF Implementation. In Figure 8, we show the ArrayUDF implementation for the interpolation. The Stencil has three attributes x_g, y_g, and z_g for particle global locations. The metadata used here are the "shift" that record the offset from original in all dimensions. Adding the $shift_x/shift_y/shift_z$ to the $x_g/y_g/z_g$ of a particle turns them into positive.



(a) Field Data **(b) Particle Data** **(c) Field and Particle Data**

Figure 9: Optimizations introduced to ArrayUDF accelerate VPIC data analyses up to 1600X.

Optimizations. The idea for optimizing interpolation UDF is the same as the one for the particle data analysis in the previous section. We utilize the locality of particle by allowing "inter_ex_udf" to save current status for its following running. We declare the i, j, k, "ex00", "ex10", "ex01", and "ex11" as the static variables. When the following particle belongs to the same cell as the current one, the "ex00", "ex10", "ex01", and "ex11" can be reused. As a result, the time to run the interpolation can be reduced. Because of the page limits, we do not show the optimized pseudocode.

5 RESULTS

We run our performance tests on the Cori supercomputer at NERSC[2], which has 2, 388 Intel Xeon "Haswell" processor nodes and a Lustre file system. Detailed test configurations are presented while we report the experimental results.

Field Data Analysis. We evaluate the ArrayUDF based field data analysis (§ 4.1) using a 3D array of the size as [1024, 1024, 256]. The element type is float. We specifically focus on comparing the performance of ArrayUDF implementation with and without the two-level reduce optimization. Since the field data in this case is only 1GB, we used 32, 64 and 128 CPU cores to run the field analysis, respectively. Our test results are reported in the Figure 9a. Clearly, the two-level reduce optimizations can significantly improve the performance of the entire field data analysis operation on the data by 1600X. Meanwhile, from the results, we also notice that the ArrayUDF scales linearly from 32 to 128 CPU cores.

Particle Data Analysis. This section reports the test results of using ArrayUDF to execute the particle data analysis of VPIC (§ 4.2). We use 1.1TB particle data($\approx$ 3 billions particles). The metadata used in the test is 167KB containing the mesh information for the 4096 mesh cells. The tests are evaluated with 256, 512 and 1024 CPU cores, respectively. Our tests compare the performance particle data analysis with or without our optimization method, as reported in previous Section 4.2. Test results are shown in Figure 9b. Considering the locality of the particle data can reduce over 12X times of the time to execute the analysis code. Also, the ArrayUDF can scale well from 256 CPU cores to 1024 CPU cores.

Fusing Particle and Field Data Analysis. By using both the field data ($\approx$ 1GB) and the particle data ($\approx$ 1.1TB) in previous sections, we evaluate the performance of interpreting the field data to each particle. Our tests use 256, 512 and 1024 CPU cores. Test results are shown in the Figure 9c. We observe that considering the particle locality is 1.4X faster than the case without considering the locality. We also notice that the speedup of this test is smaller than the one for particle data analysis. The reason is that the memory access for metadata and field data plays a less significant role in the entire UDF execution.

[2]https://www.nersc.gov/systems/cori/

6 CONCLUSIONS

The VPIC data analysis operations are similar to most scientific data analyses in that they involve a large variety of complex operations on data stored in multidimensional arrays. ArrayUDF allows these operations to be expressed as user-defined functions, while providing a transparent parallelization and automatic data managements for these customized operations. In addition to demonstrating the flexibility of ArrayUDF, we have also explored optimizations, including a more effective way to take advantage of data locality and a new multi-level reduction algorithm, to accelerate the execution of analysis functions. Our optimization strategies can speedup the ArrayUDF implementation up to 1600X in our tests. In future, we plan to formalize these optimization methods and to provide a thorough comparison with other similar systems, such as Spark [10].

ACKNOWLEDGMENT

This effort was supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research under contract number DE-AC02-05CH11231 (program manager Dr. Laura Biven), the Los Alamos National Laboratory under Contract No.89233218CNA000001 with the U.S. Department of Energy. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility.

REFERENCES

[1] K. J. Bowers, B. J. Albright, B. Bergen, L. Yin, K. J. Barker, and D. J. Kerbyson. 0.374 pflop/s trillion-particle kinetic modeling of laser plasma interaction on roadrunner. In *SC*, 2008.
[2] P. G. Brown. Overview of SciDB: Large Scale Array Storage, Processing and Analysis. In *SIGMOD*, 2010.
[3] S. Byna, J. Chou, O. Rübel, Prabhat, H. Karimabadi, et al. Parallel I/O, Analysis, and Visualization of a Trillion Particle Simulation. In *SC*, 2012.
[4] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.
[5] B. Dong, K. Wu, S. Byna, J. Liu, W. Zhao, and F. Rusu. ArrayUDF: User-Defined Scientific Data Analysis on Arrays. In *HPDC*, 2017.
[6] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996.
[7] F. Guo, X. Li, H. Li, W. Daughton, B. Zhang, N. Lloyd-Ronning, Y.-H. Liu, H. Zhang, and W. Deng. Efficient Production of High-energy Nonthermal Particles During Magnetic Reconnection in a Magnetically Dominated Ion-electron Plasma. *Astrophysical Journal*, 818(1):L9, 2016.
[8] X. Li, F. Guo, H. Li, and J. Birn. The Roles of Fluid Compression and Shear in Electron Energization during Magnetic Reconnection. *Astrophysical Journal*, 855:80, Mar. 2018.
[9] H. Tang et al. Usage pattern-driven dynamic data layout reorganization. In *CCGrid 2016*, May 2016.
[10] M. Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI 2012*, 2012.