A Phase 1 Enhanced Intellectual Services- Direct PRAC Support- **Enabling Breakthrough Kinetic Simulations of the Magnetosphere via Petascale Computing**

Final Report

June 14, 2013

Team Lead:	[1]Homa Karimabadi
Other Personnel:	[1]Y. Omelchenko [2]W. Daughton [3]K. Germaschewski [2] A. Bereshnyak [4]F. Tsung, W. Mori [5]P. Alves [6]B. Loring
Affiliation:	[1]UCSD, [2]LANL, [3]UNH, [4]UCLA, [5]Instituto de Plasmas e Fusao Nuclear [6]Lawrence Berkeley National Lab.
Contact Information for Team Lead:	homakar@gmail.com

Our goal during this project was to evaluate the viability of three concepts / algorithms in particle simulations in order to gain performance advantages on petascale computers. The results were quite encouraging and we are proceeding to incorporate them into our production codes. Here we provide a summary of our findings.

1) ***Scalable Poisson solver***

There is a strong incentive to develop fully kinetic particle codes based on the Poisson solver. The reason is that explicit particle codes are constrained by the CFL condition for light waves $c\Delta t < \Delta x$. There are many problems where these waves are irrelevant, but a fully kinetic description is still needed. To get around this issue, many explicit calculations are done with artificial parameters, which may influence the physics. One solution is to employ semi-implicit differencing of Maxwell's equations for the light wave, but the rest of the algorithm remains explicit:

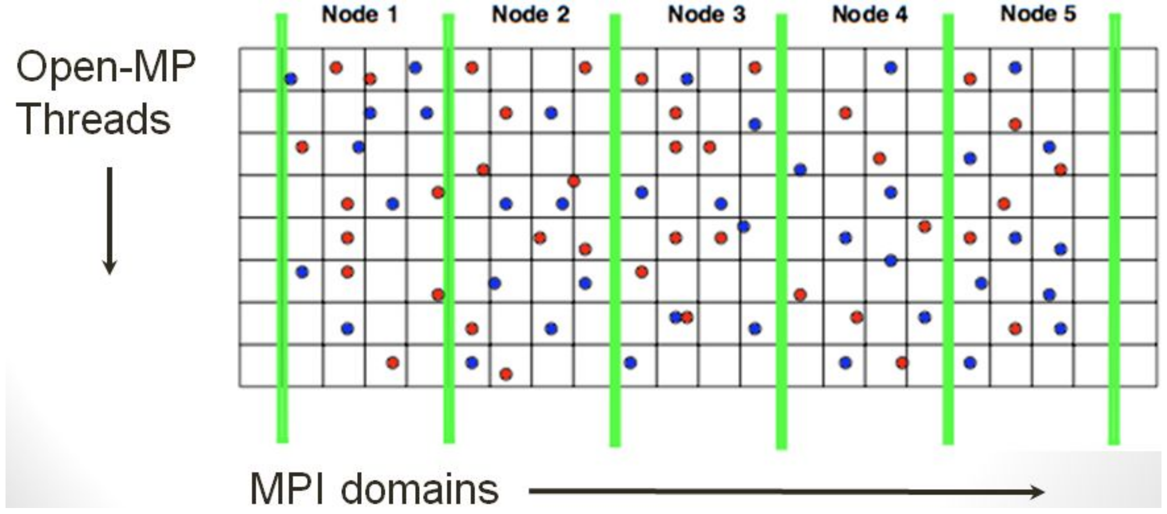
$$\nabla^2\phi = -4\pi\rho$$

$$\nabla^2\mathbf{A} - \frac{1}{c^2}\frac{\partial^2\mathbf{A}}{\partial t^2} = -\frac{4\pi}{c}\mathbf{J} + \frac{1}{c}\frac{\partial\nabla\phi}{\partial t}$$

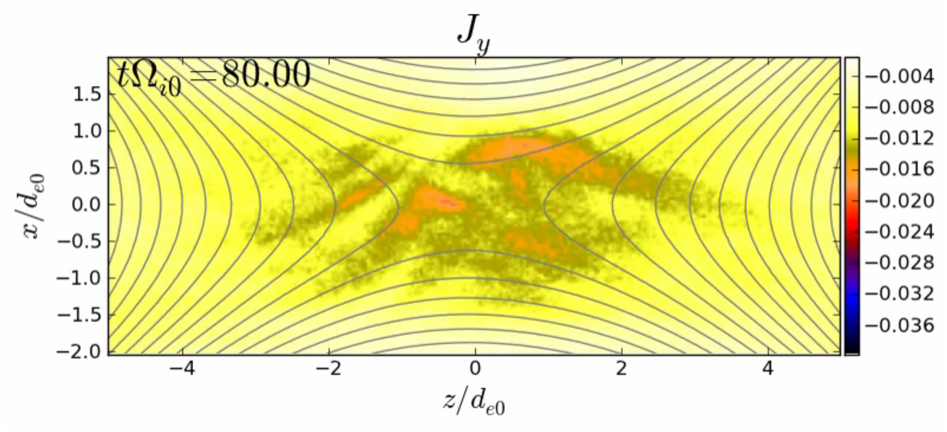
difference implicitly here

Solution requires 5 matrix inversions in 2D

This algorithm is based on the potential-formulation and requires a scalable Poisson solver. Due to challenges in developing scalable Poisson solvers, however, this approach has not been extended for petascale simulations. Fast Fourier transforms (FFTs) allow fast direct solutions but are difficult to scale to large numbers of MPI domains. Parallel FFT relies upon transpose operations, which involve a large number of communications. Our solution was to limit domain decomposition to one direction (2D) or two dimensions (3D), and then employ threads to further parallelize.

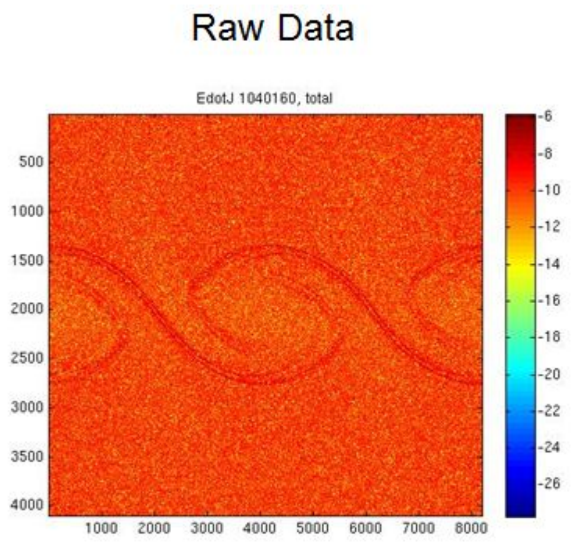


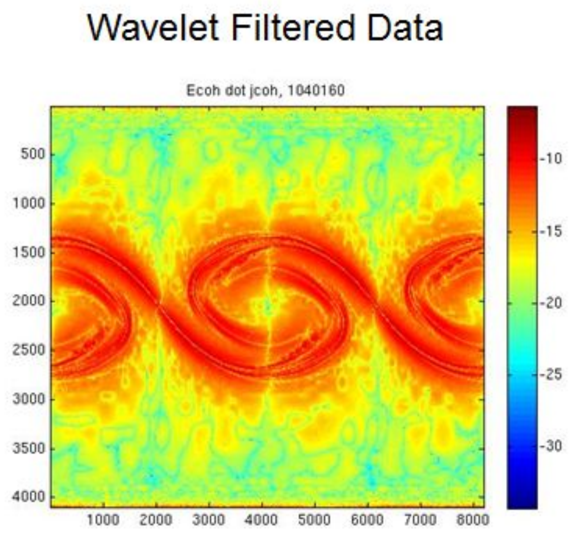
We tested the viability of this approach, starting with our pure MPI code NPIC using a semi-implicit algorithm and extended it to Open-MP threads. We used the FFTW FFT library. Our previous MPI version of the code scaled to ~ 500 cores. The new version scales linearly to ~10,000 cores. Using this new capability, we made a production run using 512 MPI domains with 16 threads per domain (8192 cores). We achieved 2 million particle pushes per second per core. Note that this version of our code does not have a fully optimized particle pusher. The semi-implicit algorithm enabled us to use a timestep 28 times larger than possible with explicit CFL. This leads to more than a factor of 7 speedup compared to the highly optimized VPIC code, and by optimizing our pusher we would be able to run a factor of 28 faster than VPIC. This capability enabled us to explore previously inaccessible regimes of magnetic reconnection. Explicit simulations are usually run for the ratio of electron plasma frequency to electron gyrofrequency ω_{pe}/ω_{ce} of 2. With the semi-implicit scheme we were able to run a simulation with ω_{pe}/ω_{ce} of 16, which is more representative of actual parameters in space plasmas. We uncovered new physics that was previously missed. The figure below shows electron scale turbulence in the reconnection region. This feature is absent at low ω_{pe}/ω_{ce}. A graduate student at Princeton (Jara-Almonte) who participated in this effort is currently preparing a publication, summarizing these new findings.



2) *Higher Order Particles*

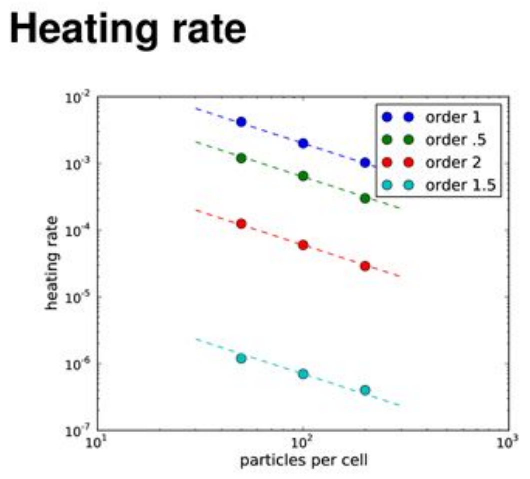
A major deficiency of particle codes is related to numerical noise associated with the discrete particle effects. This can lead to numerical heating, poor resolution of quantities of interest such as the spectrum of turbulence, and agyrotropy, among others. The figure below shows the plot of work done E.J from a particle simulation of shear turbulence, where E is the electric field and J is the current. The signature is hardly visible, and only after wavelet filtering one starts to see the signature out of the noise. The traditional way to deal with particle noise is to increase the number of particles per cell. An alternative is to use higher order particles, which, by virtue of spreading the particle shape over more cells, have a smoothing effect. There is a cost associated with the use of higher order particles. It has been generally assumed/thought that it would be more efficient to increase the number of particles/cell rather than incur the additional cost





associated with the higher order particles. Our goal was to test the viability of using higher order particles versus using larger numbers of particles. There is also a limit on how high the particle order can be before the physics becomes compromised.

There are two sources of non-physical numerical heating: a) Finite grid instability. Aliasing of unresolved grid modes gives rise to numerical instability if the Debye length is not resolved. b) Stochastic heating. Particle noise leads to errors in the electromagnetic fields that heat the plasma linearly proportional to 1/N. The figure below shows the heating rate and performance metrics for different order particles using our development full particle code PSC. It should be noted that the half-integer particle shape orders are a short-hand notation for alternating order

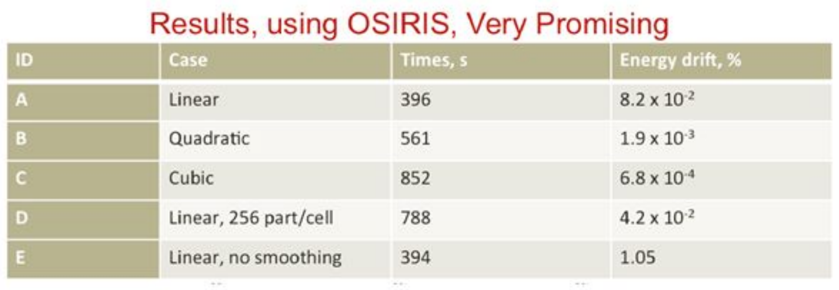


Performance
(16 core AMD Opteron / Nvidia K20X)

pusher	performance
order 2/1.5	23 M/sec
order 1	59 M/sec
order 1 (single)	78 M/sec
order 1 (SSE2)	94 M/sec
order 1 (CUDA)	824 M/sec

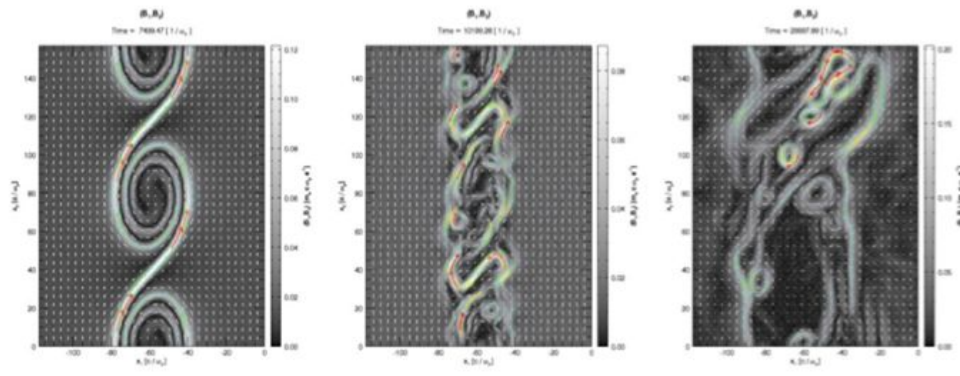
interpolation algorithms proposed by Sokolov (CPC, 2013), which use mixed 1st/2nd order interpolation to better conserve energy, which we confirm in these results.

We also performed a similar study using the OSIRIS code. We obtained consistent results from both studies: use of quadratic particles improves the energy conservation by a factor of ~40 while incurring less than a factor of 2 increase in cost.   This is very promising and indicates that use of higher order particles is an effective way to lower the particle noise in simulations. Figure below shows the time evolution of current density for the shear driven turbulence problem. At early times, the linear Kelvin-Helmholtz instability grows but soon the system develops into full turbulence.



Results, using OSIRIS, Very Promising

ID	Case	Times, s	Energy drift, %
A	Linear	396	8.2×10^{-2}
B	Quadratic	561	1.9×10^{-3}
C	Cubic	852	6.8×10^{-4}
D	Linear, 256 part/cell	788	4.2×10^{-2}
E	Linear, no smoothing	394	1.05



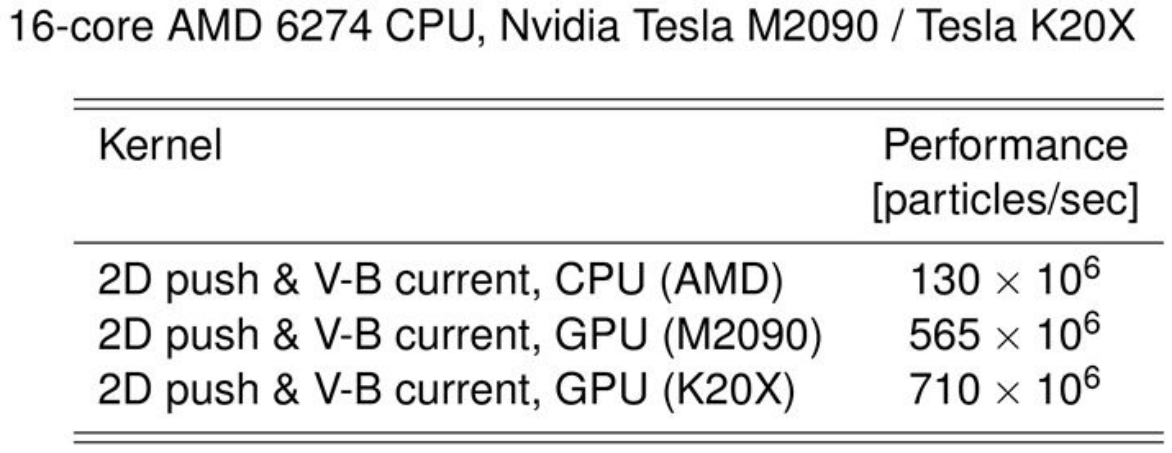
3) *GPUs*

Implementing a GPU-based particle pusher is comparatively straight-forward, since each particle moves independently from all others, so very fine grained parallelism is possible. During the push, electric and magnetic fields need to be read from the mesh and interpolated to the particle position. Performance can be enhanced significantly (5x) by using GPU shared memory to "cache" the gridded field values, so that the interpolation does not need to go to global memory. However, shared memory is not large enough to cache all of the fields, so particles need to be sorted by blocks of cells, e.g., 4x4 cells. One block is then processed by a thread block, loading all the fields into shared memory first and then processing all particles in that block of cells (typically a few thousand) by 128 or so threads. At the same time, other thread blocks advance particles in the other cell blocks. This kernel achieved performance of ~ 2 billion particles pushed / second / Tesla M2090.

Current deposition is a lot more challenging. Whereas the current contributions can be calculated for each particle independently, they all need to be accumulated on the mesh, which implies that multiple threads will add contributions to the same mesh points concurrently. The best approach we found is to use atomic accesses to shared memory to accumulate the currents before writing them back into global memory, which also requires particles to be sorted by cell blocks. In addition, the actual deposition algorithm is challenging as it contains branches depending on how each individual particle crosses cell boundaries. This kernel achieves ~ 650 million particles / second / GPU.

Finally, the previous two algorithms rely on the particles being sorted by cell blocks, so this order needs to be maintained. We developed a new sort algorithm, based on existing GPU radixsorts, that exploits the fact that particles are still "almost" sorted after a time step, since they can move at most into a neighboring block. This allows us to only use a single radix sort pass. We also integrated the sorting with other particle housekeeping tasks, in particular exchanging particles between neighboring MPI domains, which requires a copy to the CPU first. Our sort achieves ~ 770 million particles / second / GPU.

The overall performance for a typical time step, which additionally includes field push, etc., is ~ 710 million particles / second / GPU, which is more than twice as fast as the SSE4.1 SIMD pusher on 16 AMD Interlagos integer cores. The table below summarizes our benchmark results on TitanDev (M2090) and Bluewaters (K20X).



16-core AMD 6274 CPU, Nvidia Tesla M2090 / Tesla K20X

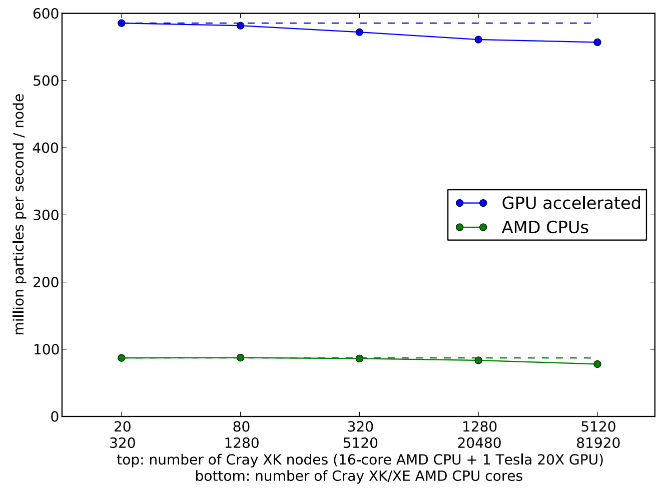
Kernel	Performance [particles/sec]
2D push & V-B current, CPU (AMD)	130×10^6
2D push & V-B current, GPU (M2090)	565×10^6
2D push & V-B current, GPU (K20X)	710×10^6

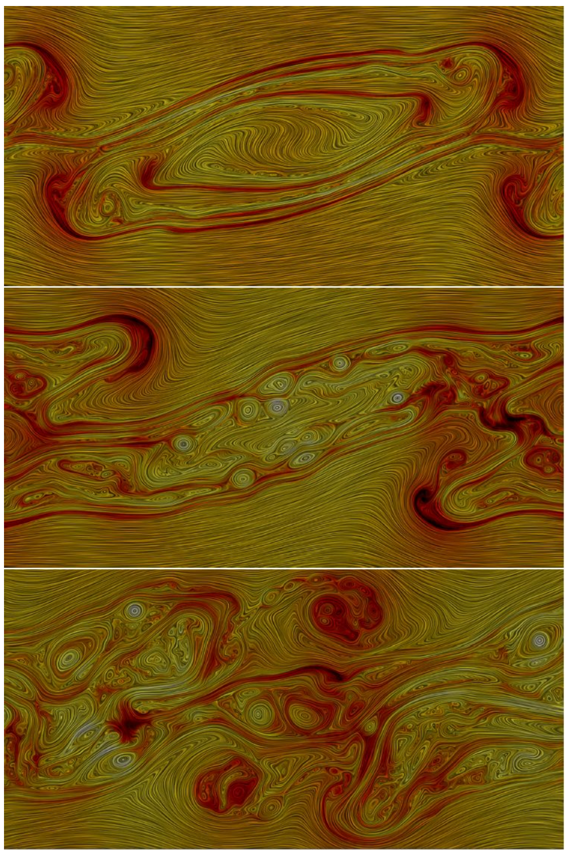
For best performance, one needs to use GPU and CPU simultaneously and we would expect a 1.2x speed up. Patch-based load balancing enables us to accomplish this. On each node, we have 1 MPI process that has ~ 30 patches that are processed on the GPU, and 15 MPI processes that have 1 patch each that are processed on the remaining CPU cores. This is currently implemented as an experimental feature in the PSC code, where currently target loads per MPI process are hard-coded. Initially, performance was disappointing due to the need to divide the domain into many more small patches. A 1000x1000 cell run that uses solely GPUs on, e.g., 100 nodes would just need to be divided into 100 patches of 100x100 cells each. In order to use the CPUs, too, we now need 4500 patches – 1500 would be processed by the CPUs (one per core, using 15/16 cores), and 30 patches would be processed on each GPU. Each patch would consequently be much smaller (approx. 15 x 15 cells), which does not provide enough parallelism to fully use all the GPUs multi-processors, and GPU performance dropped by more than 10-fold. We since changed the GPU code so that it does not process patches serially, but rather processes all patches (30 in this example) at the same time, which restores GPU performance to almost the same level as if it processed one large patch. We saw a performance increase of about 1.4x using the CPU in addition to the GPU at the time. However, by now we managed to significantly accelerate the GPU code over the older version, so that it is now about 5x faster than the CPU code (on a GPU vs 16-core CPU basis), so the potential gain is only about 20%, and we haven't recently repeated the experiment.

We were able to make a large run of the shear driven turbulence involving 15360x7680 cells on 900 GPUs and compared the results with our run with VPIC (shear driven turbulence – see below where we show the evolution of the magnetic field lines in time). The GPU version was showing about a factor of 2 larger numerical heating which led to excitation of artificial waves in time. This was puzzling because the algorithms in VPIC and PSC codes are very similar. We were finally able to trace the issue to one subtle difference in the algorithm between the two codes. This discrepancy was due to the standard (momentum-conserving) field interpolation scheme employed by PSC. This scheme is currently being replaced by an energy-conserving algorithm for particle fields derived using a finite element time domain treatment in a manner consistent with the charge-conserving algorithm for current deposition. Having resolved this issue, we are now able to push towards completing a paper on the GPU implementation of the full particle electromagnetic code to be submitted to JCP.

The figure below shows a comparative performance study of using CPUs vs GPUs, weakly scaling a simulation from 20 GPUs to 5120 GPUs, and also from 20 AMD 16-core CPUs (320 cores) to 5120 AMD 16-core CPUs (81920 cores), using runs on Bluewaters and Titan. This provides evidence that parallel scalability is quite good using both CPUs or GPUs, and it also shows an about 6x gain using the Kepler GPUs over the AMD 16-core CPUs – however, the CPU performance in those runs was not as high as previously achieved with

PSC (possibly because of a different compiler, gcc instead of Intel), but even comparing to VPIC performance of about 130 million particles / second, the GPU kernels of PSC still gain a factor of more than 4x.





Summary

This grant enabled us to assess the viability of three techniques and their potential benefits for particle simulations. The outcome is quite encouraging to the point that we are in the process of incorporating all three techniques for our production runs. There are at least three papers that we are preparing, which will summarize details of our findings. One paper will include assessment of particle noise and different remedies, using the plasma turbulence problem as a testbed. The second paper will summarize a description of our algorithm and benchmark study on the use of GPUs for particle simulations. The third paper deals with the new electron scale turbulence discovered through the use of our semi-implicit algorithm.