Optimizing Vector Particle-In-Cell (VPIC) for Memory Constrained Systems Using Half-Precision

Nigel Tan and Michela Taufer (advisor)
University of Tennessee Knoxville
Knoxville, Tennessee

Robert F. Bird (advisor)
Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

Vector Particle-In-Cell (VPIC) is one of the fastest plasma simulation codes in the world, with particle numbers ranging from one trillion on the first petascale system, Roadrunner, to more recent 10 trillion particles on the Trinity supercomputer. Current memory systems limit VPIC simulations greatly as the maximum number of particles that can be simulated directly depends on the available memory. In this work we present a suite of VPIC optimizations (i.e., particle weight storage and half-precision position storage optimizations) that enable significant increases to the number of particles. We assess the optimizations' impact on a GPU-accelerated Power9 system. We show how our optimizations enable a 40% increase in the number of particles simulated in VPIC.

CCS CONCEPTS

• **Applied computing** → **Physics**.

KEYWORDS

particle-in-cell, VPIC, mixed-precision, GPU

ACM Reference Format:
Nigel Tan and Michela Taufer (advisor) and Robert F. Bird (advisor). 2020. Optimizing Vector Particle-In-Cell (VPIC) for Memory Constrained Systems Using Half-Precision. In *Proceedings of Supercomputing '20 (SC '20).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

VPIC is a high performance particle-in-cell code that models plasma phenomena such as magnetic reconnection, fusion, solar weather, and particle acceleration. VPIC is one of the fastest PIC codes in the world having performed some of the largest plasma simulations in history. VPIC simulations require large numbers of particles (i.e., on the order of trillions of particles [1]) to model the above-mentioned real world phenomenon. As we move the VPIC code from CPUs to accelerators (i.e., GPUs) and increase the number of simulated particles, the VPIC simulations become limited by memory rather than compute capabilities: modern CPUs can access up to 4 TB of memory while GPUs are limited to 32 GB, a factor of 128 difference. Moreover, data movement between CPUs and GPUs is also costly with PCIe 4.0 limited to 32GB/s in one direction. Moving VPIC to accelerators while scaling up the number of particles requires rethinking the way the code represents particles and uses memory.

In this work we tackle this scalability problem by developing a suite of optimizations for VPICs particle storage format (i.e., particle weight optimizations and half-precision position optimizations). Our new particle representation reduces particle memory usage by up to 31.25% and enables an increase in number of particles by up to 40%. We demonstrate how our optimizations enable significantly larger simulations and that the optimized simulations produce accurate scientific results.

2 VPIC CODE AND MEMORY USAGE

VPIC operates by (a) defining a simulation space that is divided into a grid of cells and (b) modeling particle movement across the cells. VPIC is an iterative code and its workflow can be broken down into four key steps. First, particles have their position advanced based on the forces calculated from the fields. Second, current generated by the particles' movements is accumulated for each cell. Third, fields are advanced based on the accumulated current. Lastly, the fields are interpolated to the grid points and the next iteration starts. The first step takes most of the simulation time and touches all the simulated particles. Each simulated particle is a macroparticle of 32 bytes (i.e., 3 floats for position, 3 floats for momentum, 1 integer for the cell index, and 1 float for weight). Particles can take over 90% of a simulation's peak memory usage.

3 OPTIMIZATIONS OVERVIEW

Our suite of optimizations reduces memory usage associated to particles' weight and position. In this work we do not tackle momentum and cell index as they would require a deeper change to VPICs algorithms.

3.1 Particle Weight Storage

Each simulated particle in VPIC is a macroparticle modeling many real particles; the macroparticle's weight represents how many real particles are represented by the simulated particle. Particle weight generally does not change during a simulation. We use this property to optimize memory usage by adjusting how the weight is stored.

If the particle weight varies but has a limited range of values, we can replace the single-precision weight with a 16-bit integer denoting a multiple of a known base weight. Alternatively, the 16-bit integer can be an index for a lookup table of particle weights. Both methods allow 65,536 different weights while reducing particle memory by 6.25%. We call this optimization short weight (SW).

If the particle weight remains constant for all particles of the same species throughout a simulation, particle weight can be removed entirely and replaced with a per species constant weight.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SC '20, November 09–19, 2020, Atlanta, GA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

This reduces memory usage by 12.5% over default VPIC. We call this optimization constant weight (CW).

3.2 Particle Position Storage

Particle position can be stored in two ways based on the coordinate system. Global coordinates stores the absolute position in 3-D space. The cell in which a particle resides is calculated based on its global position. Alternatively, position can be stored in local coordinates with each particle storing its cell index and position within the cell. This representation requires extra space for the cell index (integer) but allows VPIC to more accurately represent each particle's position. This is because the floating-point formats distribution of values follows an almost logarithmic distribution. Values further away from zero are less precise and approximately half of all floating-point values exist in $[-1, 1]$. Local coordinates have a more even and dense distribution of values to represent position. Additionally, the cell grid's resolution affects accuracy. As the number of cells increases, the total number of floating-point values within the grid increases, improving overall accuracy. Default VPIC uses local coordinates and single-precision for particle position.

Particle position in VPIC is represented by three float values in the default code. We redesign particle position by switching to half-precision. In comparison to half-precision, alternatives are less efficient; bfloat16 is less precise and TensorFloat requires more memory. Switching to half-precision position reduces memory usage by 18.75% compared to default VPIC when we consider short weights and 31.25% with constant weights. For a fine resolution grid, half-precision can produce simulations of sufficient accuracy while enabling larger simulations.

4 RESULTS AND FUTURE WORK

We compare the impact of our optimizations versus default VPIC on a four node, GPU accelerated IBM Power9 system. Each node has 155 GB of memory, 32 cores with 128 threads; two nodes include two Nvidia Tesla V100 GPUs each. We test VPIC scalability in terms of number of particles using simulations modeling laser-plasma interactions. Figure 1 shows the scalability for the default VPIC code and when using our three optimizations (i.e., with short particle weight (SW), constant particle weight (CW), and half-precision particle position (HP)) for the four steps of a VPIC iteration. Missing columns indicate simulations crashing due to insufficient memory.

Measuring accuracy is difficult due to the lack of rigorous error quantification for the PIC method. We measure accuracy by comparing results of a simple 1D problem modeling two particles with a known analytical solution. The GPU half-precision results are compared with default VPICs single-precision results. Figure 2 presents the accuracy for the original VPIC and our optimized code when using constant weights and half-precision particle positions with 10,000 cells. Both VIPC versions achieve <10% relative error. The sinusoidal behavior is due to the particle periodically crossing the path of the analytical solution. The results demonstrate that the combination of constant weight and half-precision position reduces memory usage by 31.25% and enables up to 40% increase in number of particles using the same amount of memory. The weight optimizations introduces an integer to float conversion and a multiplication while the half-precision optimization introduces a half to

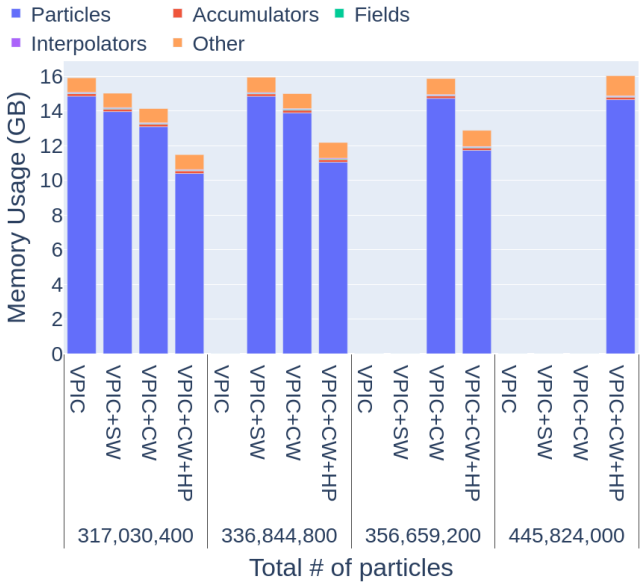

Figure 1: Memory usage of default VPIC and VPIC with our optimizations.

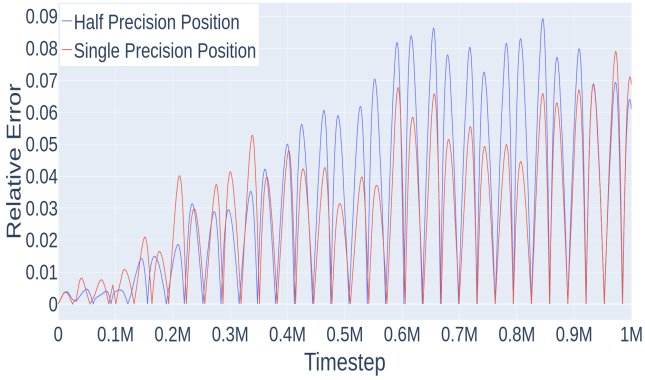

Figure 2: Accuracy comparison for default VPIC and our optimized code when using constant weights and half-precision particle positions.

float conversion. Unlike other work in mixed precision algorithms, our optimizations use reduced precision for storage rather than for accelerating computation. We also show that our optimizations not only greatly increase simulation scale on memory constrained hardware, but also achieve similar accuracy as default VPIC.

Future work includes studying the algorithmic changes necessary for using half-precision for particle momentum and their impact on scalability and accuracy. We will also study alternate formats for particle storage such as fixed-point on the same metrics.

LANL release number: LA-UR-20-26189

REFERENCES

[1] Kevin J Bowers, Brian J Albright, Benjamin Bergen, Lin Yin, Kevin J Barker, and Darren J Kerbyson. 2008. 0.374 Pflop/s trillion-particle kinetic modeling of laser plasma interaction on Roadrunner. In *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE, 1–11.