Modernizing Simulation Software for the Exascale Era

Nigel Tan, Advisor: Michela Taufer, University of Tennessee Knoxville

THE UNIVERSITY OF TENNESSEE KNOXVILLE
GCLab Dr. Michela Taufer UNIVERSITY OF TENNESSEE
Los Alamos NATIONAL LABORATORY
LA-UR-23-28993
NNSA National Nuclear Security Administration

Exascale Challenges

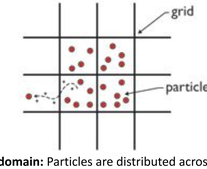
- HPC hardware is increasingly heterogeneous
  - Each vendor has hardware specific software stacks (Cuda, ROCm, OneAPI)
  - Ad-hoc re-engineering of codes for new hardware guarantees performance but is impractical due to increasing heterogeneity
- Hardware is focused on AI/ML applications rather than HPC
  - Features like half-precision increase compute but are difficult to use for HPC
  - Simulation scale is often limited by memory capacity and communication
- Growing need for data snapshots for resilience, reproducibility, and exploration
  - Storage requirements for large simulations are massive
  - IO capabilities cannot keep up with demand

We present an approach for modernizing software with the following goals:

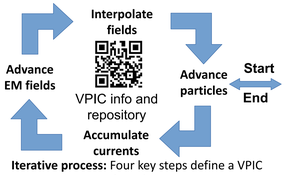
- **Attain higher performance and portability across different hardware**
- **Enable larger scale simulations with alternative number formats**
- **Enhance data snapshots with efficient data de-duplication**

Legacy Applications: VPIC

- Vector Particle-In-Cell (VPIC) is a high performance PIC code for plasma simulations:
  - Simulates magnetic reconnection, fusion, solar weather, and particle acceleration amongst other plasma phenomenon
  - Well optimized for CPUs but **NOT** for accelerators (e.g., GPUs)



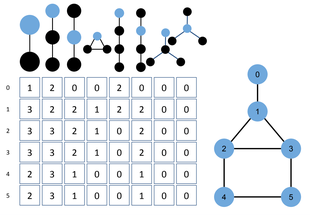
Spatial domain: Particles are distributed across an n-D space that is decomposed into a n-D grid



Iterative process: Four key steps define a VPIC iteration

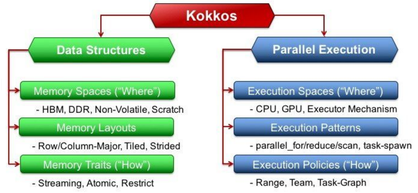
New Applications: ORANGES

- ORbit ANd Graphlet Enumeration at Scale (ORANGES) is a parallel graph application that calculates each vertices graphlet degree vector (GDV)
- Designed with performance portability in mind at the start
- Produces large amounts of data with sparse update patterns



Kokkos

- Kokkos is a portability ecosystem that enables the creation of production ready parallel applications that are hardware agnostic
- Both VPIC and ORANGES use Kokkos for parallel execution



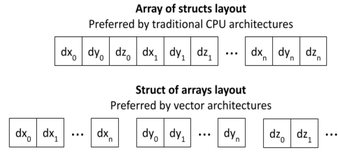
Data structures and parallel execution abstractions simplify portable applications
https://kokkos.github.io/kokkos-core-wiki/ProgrammingGuide/ProgrammingModel.html

Performance Portability

- Kokkos alone will not attain the same level of performance as a direct port
- Using VPIC we demonstrate how to optimize for different architectures while remaining portable using optimizations to data layout, sorting, and vectorization

Array of Structs vs Struct of Arrays

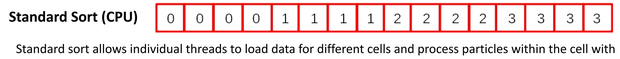
- Memory layout has a large impact on memory bound applications like VPIC
- Adjust data structures memory layout to favor target architecture



- Array of structs is preferred for CPUs since each thread can load particles independently.
- Vector architectures like GPUs prefer struct of arrays where individual variables are contiguous

Adjust Sorting Order Based Architecture

- Sorting is a common part of many simulations for improving cache performance
- Switching the sorting order can lead to significant performance improvements



Standard sort allows individual threads to load data for different cells and process particles within the cell with minimal overlap between threads. Threads can collect updates from particles locally before writing results.

Strided sort helps ensure that successive threads access successive locations in memory. This helps maximize memory bandwidth for vector architectures like GPUs. Writes to memory are also less likely to cause conflicts.

Vectorization Abstractions

- Portable vectorization comes in three forms: automatic, guided, and manual
- Switching from auto to guided or manual vectorization can greatly improve performance while remaining portable

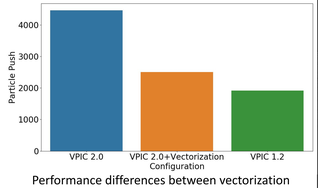
Auto
- Use Kokkos vector execution policy to auto vectorize
- Minimal effort
- Least effective

Guided
- Vectorize sections of code with OpenMP SIMD
- Moderate effort
- Effective

Manual
- Rewrite kernels with explicitly SIMD types
- High effort
- Very effective

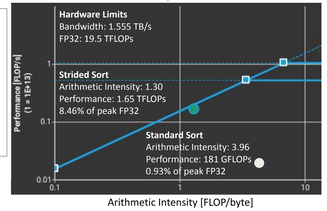
Switch to guided vectorization to improve performance without having to rewrite with explicit simd types

Particle Push Runtime on dual socket EPYC 7H12



Performance differences between vectorization implementations

Particle Push FP32 Roofline performance on A100 GPUs



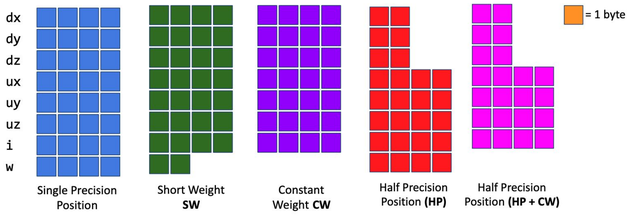
- Basic port using Kokkos (VPIC 2.0) uses auto vectorization and is slower than the other options
- Using guided vectorization regains the majority of performance without using explicit SIMD intrinsics
- VPIC 1.2 remains the fastest due to the tuned SIMD intrinsics
- Strided sort achieves 8.46% of peak FP32 on an A100 GPU and is memory bound
- Standard sort does not use available compute or bandwidth resources efficiently
  - Limited by memory latency rather than memory bandwidth or compute bound

R. Bird, N. Tan, S. Luedtke, S. Harrell, M. Taufer, and B. Albright. VPIC 2.0: Next Generation Particle-in-Cell Simulations. Journal of IEEE Transactions on Parallel and Distributed Systems (IEEE TPDS), 2734-2748, 2021.

Memory layout, sorting, and vectorization optimizations are necessary to achieve performance portability

Increase Scale with Mixed Precision

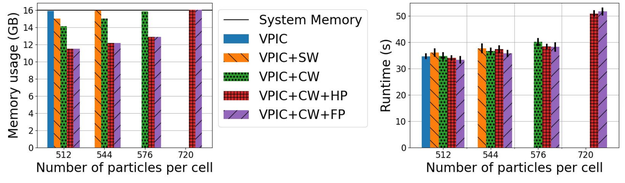
- Particle simulations require vast quantities of particles to model real world phenomenon with modern simulations reaching trillions of particles
- **Simulation scale is more limited by memory than compute power**
- Memory growth cannot keep up with compute growth
  - Modern CPUs support up to 6 TB of memory
  - GPUs are limited to at most 128 GB
  - Data movement between CPU and GPU is costly
- Optimize particle format with reduced precision to reduce memory usage



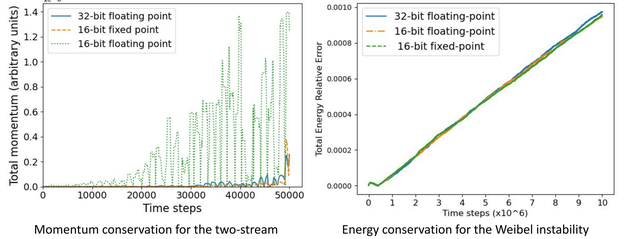
Particle memory usage comparison between VPIC, VPIC with our short weight (SW), constant weight (CW), half-precision (HP), and VPIC with both half-precision and constant weight optimizations (HP+CW).

Format	Space Reduction	Accuracy (Decimal Digits)
32-bit floating-point	1.00x	7.225
16-bit floating-point	0.81x	3.311
16-bit fixed-point	0.81x	4.515

Comparison of memory usage and accuracy for single-precision, half-precision, and 16-bit fixed point formats.



Memory usage and runtime comparison of VPIC running on a V100 with our optimizations applied as the number of particles increases. Missing bars indicate out of memory errors.



Momentum conservation for the two-stream instability

Energy conservation for the Weibel instability

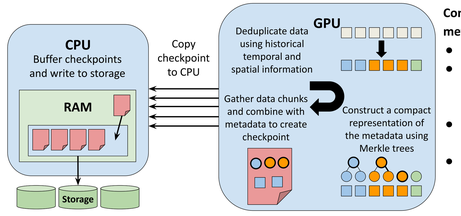
Accuracy can be maintained while using lower precision formats by adjusting the grid resolution.

N. Tan, R. Bird, G. Chen, S. V. Luedtke, B. Albright, and M. Taufer. Analysis of Vector Particle-In-Cell (VPIC) Memory Usage Optimizations on Cutting-Edge Computer Architectures. In Journal of Computational Science Volume 60, April 2022, 101566, 2022.
N. Tan, R. Bird, G. Chen, and M. Taufer. Optimize Memory Usage in Vector Particle-In-Cell (VPIC) to Break the 10 Trillion Particle Barrier in Plasma Simulations. In Proceedings of the International Conference on Computational Science (ICCS). 2021

Leveraging alternative number formats enables larger simulations while maintaining performance and accuracy

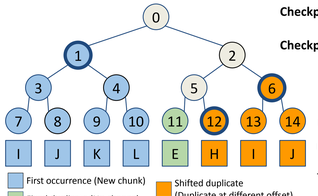
Reducing IO Overhead with De-Duplication

- Writing data snapshots is a common pattern in HPC for resilience, adjoint computations, and exploration of alternative paths that induces high IO overhead
- State of the art techniques like incremental checkpointing and compression do not take full advantage of spatial and temporal repeating patterns in the data
- We introduce a highly parallel de-duplication algorithm based on these principles
  - Identify repeating patterns with hashing that leverages spatial and temporal redundancy across the entire checkpoint record
  - Coalesce contiguous chunks to obtain a compact metadata representation
  - Assemble the compact metadata and unique chunks into a contiguous buffer
  - Leverage massive GPU parallelism in order to achieve high scalability

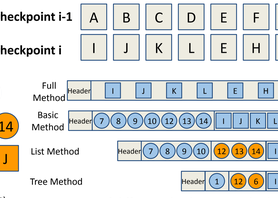


Compare four de-duplication methods
- **Full:** save everything
- **Basic:** save chunks that change between checkpoints
- **List:** spatial-temporal redundancy
- **Tree:** spatial-temporal redundancy and metadata compaction

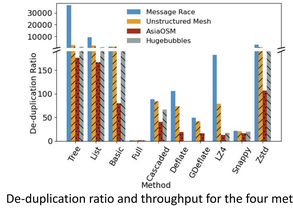
Overview of our Merkle tree inspired data de-duplication method using four design principles for checkpointing.

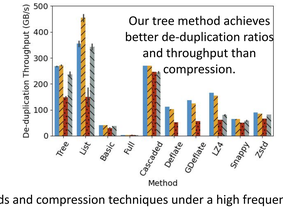


Example of our tree method for de-duplicating data and compacting metadata.

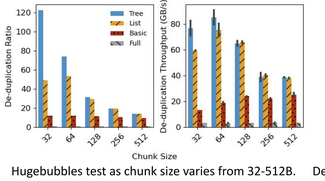


Comparison of different checkpoint methods. Note that the data chunks ( ) are much larger than the metadata nodes ( )

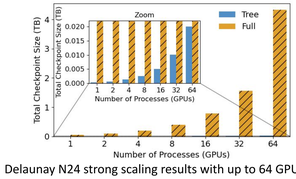




De-duplication ratio and throughput for the four methods and compression techniques under a high frequency checkpointing situation.



Hugebubbles test as chunk size varies from 32-512B. Smaller chunks improve de-duplication ratio and throughput for our tree method



Delaunay N24 strong scaling results with up to 64 GPUs. Our method achieves more than 100x size reduction

N. Tan, J. Luettgau, J. Marquez, K. Teranishi, N. Morales, S. Bhowmick, F. Cappello, M. Taufer, and B. Nicolae. Scalable Checkpointing of Applications with Sparsely Updated Data. In Proceedings of the 52nd International Conference on Parallel Processing (ICPP), 2023.

Identifying contiguous repeating patterns and removing both data and metadata redundancy across all checkpoints reduces IO overhead and storage