Extreme Scaling and Performance Across Diverse Architectures

Nicholas Frontiere  
University of Chicago/Argonne National Laboratory

Salman Habib  
Vitali Morozov  
Hal Finkel  
Adrian Pope  
Katrin Heitmann  
Kalyan Kumaran  
Venkatram Vishwanath  
Tom Peterka  
Joe Insley  
Argonne National Laboratory

David Daniel  
Patricia Fasel  
Los Alamos National Laboratory  

Zarija Lukic  
Lawrence Berkeley National Laboratory  

Justin Luitjens  
NVIDIA  

George Zagaris  
Kitware

HACC (Hardware/Hybrid Accelerated Cosmology Code) Framework
Motivating HPC: The Computational Ecosystem

- **Motivations for large HPC campaigns:**
  1. Quantitative predictions for complex, nonlinear systems
  2. Discover/Expose physical mechanisms
  3. System-scale simulations (‘impossible experiments’)
  4. Large-Scale inverse problems and optimization

- **Driven by a wide variety of data sources, computational cosmology must address ALL of the above**

- **Role of scalability/performance:**
  1. Very large simulations necessary, but not just a matter of running a few large simulations
  2. High throughput essential (short wall clock times)
  3. Optimal design of simulation campaigns (parameter scans)
  4. Large-scale data-intensive applications
Supercomputing: Hardware Evolution

- Power is the main constraint
  - 30X performance gain by 2020
  - ~10-20MW per large system
  - power/socket roughly const.
- Only way out: more cores
  - Several design choices
  - None good from scientist’s perspective
- Micro-architecture gains sacrificed
  - Accelerate specific tasks
  - Restrict memory access structure (SIMD/SIMT)
- Machine balance sacrifice
  - Memory/Flops; comm BW/Flops — all go in the wrong direction
  - (Low-level) code must be refactored
Supercomputing: Systems View

• HPC is not what it used to be!
  ‣ HPC systems were meant to be balanced under certain metrics — nominal scores of unity (1990’s desiderata)
  ‣ These metrics now range from ~0.1 to ~0.001 on the same system currently and will get worse (out of balance systems)
  ‣ RAM is expensive: memory bytes will not scale like compute flops, era of weak scaling (fixed relative problem size) has ended

• Challenges
  ‣ Strong scaling regime (fixed absolute problem size) is much harder than weak scaling (since metric really is ‘performance’ and not ‘scaling’)
  ‣ Machine models are complicated (multiple hierarchies of compute/memory/network)
  ‣ Codes must add more physics to use the available compute, adding more complexity
  ‣ Portability across architecture choices must be addressed (programming models, algorithmic choices, trade-offs, etc.)
Supercomputing Challenges: Sociological View

• Codes and Teams
  ‣ Most codes are written and maintained by small teams working near the limits of their capability (no free cycles)
  ‣ Community codes, by definition, are associated with large inertia (not easy to change standards, untangle lower-level pieces of code from higher-level organization, find the people required that have the expertise, etc.)
  ‣ Lack of consistent programming model for “scale-up”
  ‣ In some fields at least, something like a “crisis” is approaching (or so people say)

• What to do?
  ‣ We will get beyond this (the vector to MPP transition was worse)
  ‣ Transition needs to be staged (not enough manpower to entirely rewrite code base)
  ‣ Prediction: There will be no ready made solutions
  ‣ Realization — “You have got to do it for yourself”
Co-Design vs. Code Design

- **HPC Myths**
  - The magic compiler
  - The magic programming model/language
  - Special-purpose hardware
  - Co-Design (not now anyway, but maybe in the future —)

- **Dealing with Today’s Reality**
  - Code teams must understand all levels of the system architecture, but not be enslaved by it (software cycles are long)!
  - Must have a good idea of the ‘boundary conditions’ (what may be available, what is doable, etc.)
  - ‘Code Ports’ is ultimately a false notion
  - Start thinking out of the box — domain scientists and computer scientists and engineers must work together

Future heterogeneous manycore system, Borkar and Chien (2011)
HACC Application

Simulations with 6 orders of dynamic range, exploiting all supercomputing architectures

CMB SZ Sky Map  Strong Lensing  Synthetic Catalog

The Outer Rim Simulation

Large Scale Structure

Scientific Inference: Cosmological Parameters

Merger Trees
Large Scale Structure: Vlasov-Poisson Equation

\[
\frac{\partial f_i}{\partial t} + \dot{x} \frac{\partial f_i}{\partial x} - \nabla \phi \frac{\partial f_i}{\partial p} = 0, \quad p = a^2 \dot{x},
\]

\[
\nabla^2 \phi = 4\pi Ga^2 (\rho(x, t) - \langle \rho_{dm}(t) \rangle) = 4\pi Ga^2 \Omega_{dm} \delta_{dm} \rho_{cr},
\]

\[
\delta_{dm}(x, t) = (\rho_{dm} - \langle \rho_{dm} \rangle)/\langle \rho_{dm} \rangle,
\]

\[
\rho_{dm}(x, t) = a^{-3} \sum_i m_i \int d^3p f_i(x, \dot{x}, t).
\]

- Properties of the Cosmological Vlasov-Poisson Equation:
  - 6-D PDE with long-range interactions, no shielding, all scales matter; models gravity-only, collisionless evolution
  - Jeans instability drives structure formation at all scales from smooth Gaussian random field initial conditions
  - Extreme dynamic range in space and mass (in many applications, million to one in both space and density, ‘everywhere’)
Large Scale Structure Simulation Requirements

- **Force and Mass Resolution:**
  - Galaxy halos ~100kpc, hence force resolution has to be ~kpc; with Gpc box-sizes, a dynamic range of a million to one
  - Ratio of largest object mass to lightest is ~10000:1

- **Physics:**
  - Gravity dominates at scales greater than ~Mpc
  - Small scales: galaxy modeling, semi-analytic methods to incorporate gas physics/feedback/star formation

- **Computing ‘Boundary Conditions’:**
  - Total memory in the PB+ class
  - Performance in the 10 PFlops+ class
  - Wall-clock of ~days/week, in situ analysis

---

Can the Universe be run as a short computational ‘experiment’?
Architectural Challenges: The HACC Story

Roadrunner: Prototype for modern accelerated architectures, first to break the PFlops barrier

Architectural ‘Features’

- Complex heterogeneous nodes
- Simpler cores, lower memory/core, no real cache
- Skewed compute/communication balance
- Programming models?
- I/O? File systems?
- Effect on code longevity

HACC team meets Roadrunner
Combating Architectural Diversity with HACC

- **Architecture-independent performance/scalability:** 'Universal' top layer + 'plug in' node-level components; minimize data structure complexity and data motion

- **Programming model:** 'C++/MPI + X' where X = OpenMP, Cell SDK, OpenCL, CUDA, --

- **Algorithm Co-Design:** Multiple algorithm options, stresses accuracy, low memory overhead, no external libraries in simulation path

- **Analysis tools:** Major analysis framework, tools deployed in stand-alone and in situ modes

![Power spectra ratios across different implementations (GPU version as reference)]
HACC Structure: Universal vs. Local Layers

HACC Top Layer:
3-D domain decomposition with particle replication at boundaries (‘overloading’) for Spectral PM algorithm (long-range force)

Host-side: Scaling controlled by FFT

HACC ‘Nodal’ Layer:
Short-range solvers employing combination of flexible chaining mesh and RCB tree-based force evaluations

Performance controlled by short-range solver

RCB tree levels

~50 Mpc

~1 Mpc

Newtonian Force
Noisy CIC PM Force
6th-Order sinc-Gaussian spectrally filtered PM Force

Two-particle Force
HACC: Algorithmic Features and Options

- **Fully Spectral Particle-Mesh Solver**: 6th-order Green function, 4th-order Super-Lanczos derivatives, high-order spectral filtering, high-accuracy polynomial for short-range forces
- **Custom Parallel FFT**: Pencil-decomposed, high-performance FFT (up to $15K^3$)
- **Particle Overloading**: Particle replication at ‘node’ boundaries to reduce/delay communication (intermittent refreshes), important for accelerated systems
- **Flexible Chaining Mesh**: Used to optimize tree and P3M methods
- **Optimal Splitting of Gravitational Forces**: Spectral Particle-Mesh melded with direct and RCB (‘fat leaf’) tree force solvers (PPTPM), short hand-over scale (dynamic range splitting $\sim 10,000 \times 100$); pseudo-particle method for multipole expansions
- **Mixed Precision**: Optimize memory and performance (GPU-friendly!)
- **Optimized Force Kernels**: High performance without assembly
- **Adaptive Symplectic Time-Stepping**: Symplectic sub-cycling of short-range force timesteps; adaptivity from automatic density estimate via RCB tree
- **Custom Parallel I/O**: Topology aware parallel I/O with lossless compression (factor of 2); 1.5 trillion particle checkpoint in 4 minutes at $\sim 160$GB/sec on Mira
HACC on the IBM Blue Gene/Q

HACC BG/Q Experience

- **System:** BQC chip — 16 cores, 205GFlops, 16GB RAM, 32MB L2, 400GB/s crossbar; 5-D torus network at 40GB/s

- **Programming Models:** Two-tiered programming model (MPI+OpenMP) very successful, use of vector intrinsics (QPX) essential

- **I/O:** Custom I/O implementation (one file per I/O node, disjoint data region/process) gives ~2/3 of peak performance under production conditions

- **Job Mix:** Range of job sizes running on Mira, from 2 to 32 racks
HACC on the BG/Q

HACC BG/Q Version

- **Algorithms:** FFT-based SPM; PP+RCB Tree
- **Data Locality:** Rank level via ‘overloading’, at tree-level use the RCB grouping to organize particle memory buffers
- **Build/Walk Minimization:** Reduce tree depth using rank-local trees, shortest hand-over scale, bigger p-p component
- **Force Kernel:** Use polynomial representation (no look-ups); vectorize kernel evaluation; hide instruction latency

13.94 PFlops, 69.2% peak, 90% parallel efficiency on 1,572,864 cores/mpi ranks, 6.3M-way concurrency

HACC: Hybrid/Hardware Accelerated Cosmology Code Framework

3.6 trillion particle benchmark*

HACC weak scaling on the IBM BG/Q (MPI/OpenMP)

*largest ever run

Habib et al. 2012
Accelerated Systems: HACC on Titan (Cray XK7)

Imbalances and Bottlenecks

- Memory is primarily host-side (32 GB vs. 6 GB) (against Roadrunner’s 16 GB vs. 16 GB), important thing to think about (in case of HACC, the ‘grid/particle’ balance)

- PCIe is a key bottleneck; overall interconnect B/W does not match Flops (not even close)

- There’s no point in ‘sharing’ work between the CPU and the GPU, performance gains will be minimal — GPU must dominate

- The only reason to write a code for such a system is if you can truly exploit its power (2 X CPU is a waste of effort!)

Strategies for Success

- It’s (still) all about understanding and controlling data motion

- Rethink your code and even approach to the problem

- Isolate hotspots, and design for portability around them (modular programming)

- Pragmas will never be the full answer (with maybe an exception or two)
HACC on Titan: GPU Implementation (Schematic)

P3M Implementation (OpenCL):
- Spatial data pushed to GPU in large blocks, data is sub-partitioned into chaining mesh cubes
- Compute forces between particles in a cube and neighboring cubes
- Natural parallelism and simplicity leads to high performance
- Typical push size ~2GB; large push size ensures computation time exceeds memory transfer latency by a large factor
- More MPI tasks/node preferred over threaded single MPI tasks (better host code performance)

New Implementations (OpenCL and CUDA):
- P3M with data pushed only once per long time-step, completely eliminating memory transfer latencies (orders of magnitude less); uses ‘soft boundary’ chaining mesh, rather than rebuilding every sub-cycle
- TreePM analog of BG/Q code written in CUDA, also produces high performance
HACC on Titan: GPU Implementation Performance

- P3M kernel runs at 1.6TFlops/node at 40.3% of peak (73% of algorithmic peak)
- TreePM kernel was run on 77% of Titan at 20.54 PFlops at almost identical performance on the card
- Because of less overhead, P3M code is (currently) faster by factor of two in time to solution
Summary

Basic Ideas:

• Thoughtful design of flexible code infrastructure; minimize number of computational ‘hot spots’, explore multiple algorithmic ideas — exploit domain science expertise

• Because machines are so out of balance, focusing only on the lowest-level compute-intensive kernels can be a mistake (‘code ports’)

• One possible solution is an overarching universal layer with architecture-dependent, plug-in modules (with implications for productivity)

• Understand data motion issues in depth — minimize data motion, always look to hide communication latency with computation

• Be able to change on fast timescales (HACC needs no external libraries in the main simulation code — helps to get on new machines early)

• As science outputs become more complex, data analysis becomes a very significant fraction of available computational time — optimize performance with this in mind
EXTRA SLIDES

NOT SURE IF...
THE SLIDESHOW IS OVER
OR IF THIS IS THE END
SLIDE
The problem: What are $f_{long}(r1 - r2)$ and $f_{short}(r1 - r2)$?

The answer: $f_{long}(r1 - r2)$, the “grid softened force”, can be determined empirically. The force computed by the particle-mesh technique is sampled for many particle separations, and the resulting samples are fit by a polynomial. $f_{short}(r1 - r2)$ is then trivially determined by subtraction.

The question: How to best compute $f_{short}(r1 - r2)$.

The answer: This depends on the architecture!
Force Splitting

- The gravitational force calculation is split into long-range part and a short-range part
- A grid grid is responsible for largest 4 orders of magnitude of dynamic range
- particle methods handle the critical 2 orders of magnitude at the shortest scales

Complexity:

- PM (grid) algorithm: $O(N_p) + O(N_g \log N_g)$, where $N_p$ is the total number of particles, and $N_g$ the total number of grid points
- tree algorithm: $O(N_{pl} \log N_{pl})$, where $N_{pl}$ is the number of particles in individual spatial domains ($N_{pl} \ll N_p$)
- the close-range force computations are $O(N_d^2)$ where $N_d$ is the number of particles in a tree leaf node within which all direct interactions are summed
Long-Range Algorithm:

- The long/medium range algorithm is based on a fast, spectrally filtered PM method.
- The density field is generated from the particles using a Cloud-In-Cell (CIC) scheme.

The density field is smoothed with the (isotropizing) spectral filter:

\[
\exp \left( -\frac{k^2 \sigma^2}{4} \right) \left[ \frac{2k}{\Delta} \sin \left( \frac{k\Delta}{2} \right) \right]^{n_s},
\]

where the nominal choices are \( \sigma = 0.8 \) and \( n_s = 3 \). The noise reduction from this filter allows matching the short and longer-range forces at a spacing of 3 grid cells.

- The Poisson solver uses a sixth-order, periodic, influence function (spectral representation of the inverse Laplacian).
- The gradient of the scalar potential is obtained using higher-order spectral differencing (fourth-order Super-Lanczos).
The “Poisson-solve” is the composition of all the kernels above in one single Fourier transform.

Each component of the potential field gradient then requires an independent FFT.

Distributed FFTs use a pencil decomposition.

To obtain the short-range force, the filtered grid force is subtracted from the Newtonian force.

**Mixed precision:**

- Single precision is adequate for the short/close-range particle force evaluations and particle time-stepping.
- Double precision is used for the spectral component.
Overloading

The spatial domain decomposition is in regular 3-D blocks, but unlike the guard zones of a typical PM method, full particle replication – termed ‘particle overloading’ – is employed across domain boundaries.
Overloading (cont.)

- Works because particles cluster and large-scale bulk motion is small
- Short-range force contribution is not used for particles near the edge of the overloading region
- The typical memory overhead cost for a large run is \( \sim 10\% \)
- The point of overloading is to allow sufficiently-exact medium/long-range force calculations with no communication of particle information and high-accuracy local force calculations

We use relatively sparse refreshes of the overloading zone! This is key to freeing the overall code performance from the weaknesses of the underlying communications infrastructure.
The time-stepping is based on a 2nd-order split-operator symplectic SKS scheme (stream-kick-stream). Because the characteristic time scale of the long-range force is much smaller than that of the short-range force, we sub-cycle the short-range force operator. The relatively slowly evolving longer range force is effectively frozen during the shorter-range sub-cycles.

\[ M_{\text{full}}(t) = M_{lr}(t/2)(M_{sr}(t/n_c))^{n_c}M_{lr}(t/2). \]  

The number of sub-cycles is \( n_c = 3 - 5 \), in most cases.
The short-range force is computed using recursive coordinate bisection (RCB) tree in conjunction with a highly-tuned short-range polynomial force kernel.

(graphic from Gafton and Rosswog: arXiv:1108.0028)
At each level, the node is split at its center of mass
During each node split, the particles are partitioned into disjoint adjacent memory buffers
This partitioning ensures a high degree of cache locality during the remainder of the build and during the force evaluation
To limit the depth of the tree, each leaf node holds more than one particle. This makes the build faster, but more importantly, trades time in a slow procedure (a “pointer-chasing” tree walk) for a fast procedure (the polynomial force kernel).
Due to the compactness of the short-range interaction, the kernel can be represented as

\[ f_{SR}(s) = (s + \epsilon)^{-3/2} - f_{\text{grid}}(s) \] (3)

where \( s = r \cdot r \), \( f_{\text{grid}}(s) = \text{poly}[5](s) \), and \( \epsilon \) is a short-distance cutoff.

- An interaction list is constructed during the tree walk for each leaf node
- When using fine-grained threading: using OpenMP, the particles in the leaf node are assigned to different threads: all threads share the interaction list (which automatically balances the computation)
- The interaction list is processed using a vectorized kernel routine (written using QPX/SSE compiler intrinsics)
- Filtering for self and out-of-range interactions uses the floating-point select instruction: no branching required
- We can use the reciprocal (sqrt) estimate instructions: no library calls