Performance and Portability Lessons from HACC

Hal Finkel, Nick Frontiere,
Katrin Heitmann, Vitali Morozov,
Adrian Pope
Argonne
National Laboratory

Zarija Lukic
Lawrence Berkeley
National Laboratory

David Daniel, Patricia Fasel
Los Alamos
National Laboratory

Salman Habib
HEP and MCS Divisions
Argonne National Laboratory

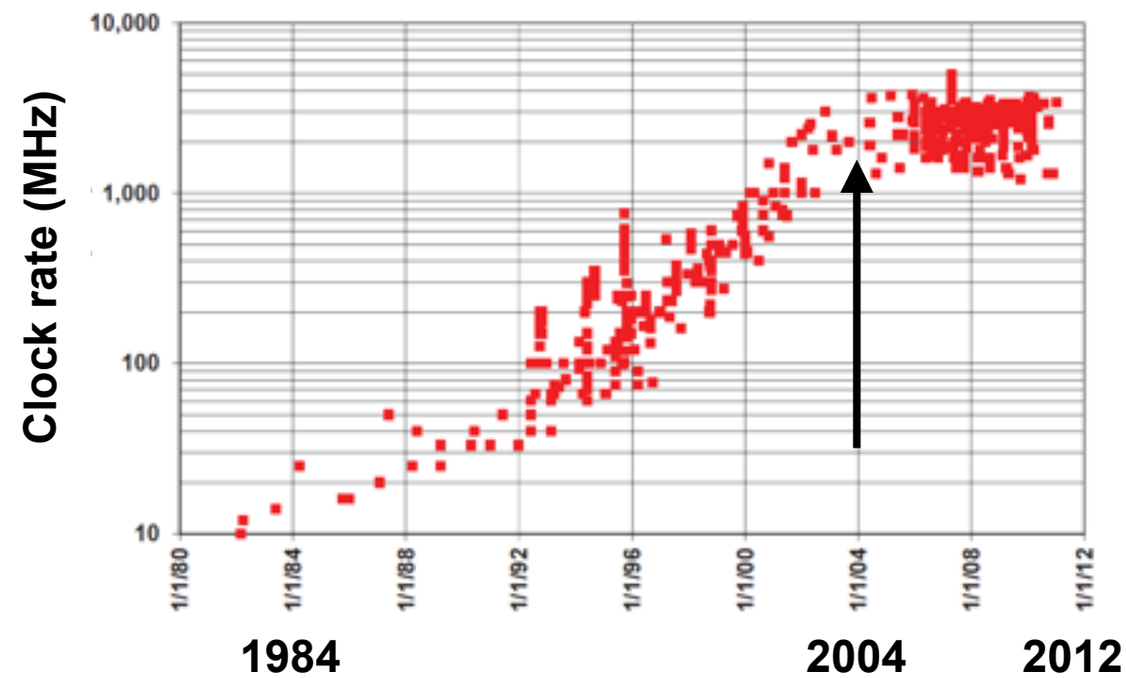
Computation Institute
Argonne National Laboratory
University of Chicago

Kavli Institute for Cosmological Physics
University of Chicago

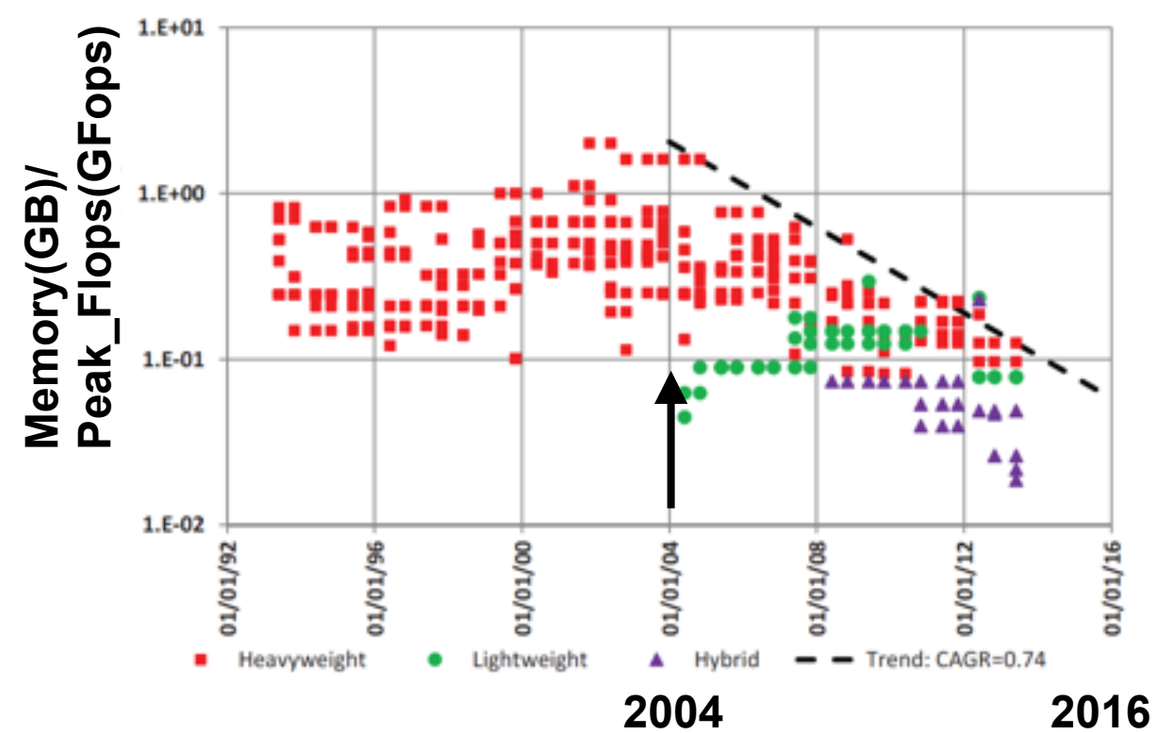
ATPESC
August 10, 2015

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



Kogge and Resnick (2013)



Motivating HPC

- **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 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”

Performance and Portability I

- **Performance (assuming you are solving a new problem, not doing 'ports')**
  - Are you sure you want brute speed? Is performance the true bottleneck? (There is always a price -- realize all HPC machines are poorly balanced)
  - Or do you just want to run a 'large' problem with acceptable time to solution? (This is the general case)
  - **Step I: Know** what you want, if performance is a priority it must be designed in right at the start, **you'll never get it afterwards** (optimizing gains are often minimal to non-existent)
  - **Step II:** If performance is needed, make sure you understand the global science problem(s) being addressed; you may have to start from scratch! **There's no replacement for domain knowledge**
  - Factor of two rule -- given human constraints (and Moore's law), **it is not usually worth it to go for the last factor of two**, but there are exceptions -- HACC is one
  - **Step III:** Obtaining performance is painful, so design for the future -- what can you rely on, what can disappear, what can change, what can break -- the more parameters you can control, the better -- HPC systems are not your laptop: **Learn from experience**
  - General Advice (mostly obvious): On-chip/node optimization comes first**,** minimize number of performance 'hot spots' to the extent possible, ditto with data motion (aim to be compute-bound, avoid look-ups), avoid forest/tree syndromes, think about sacrificing memory for speed wherever possible, vectorize everything, FMAs are your friends, talk to performance gurus, do not resort to assembly unless desperate, etc. etc.

Performance and Portability II

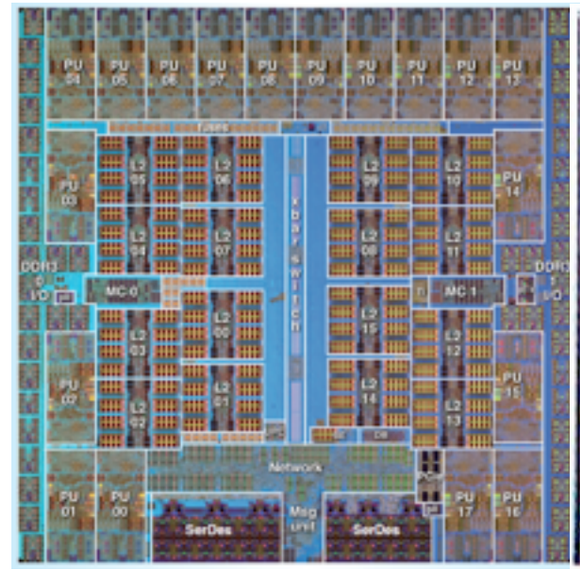
- **Portability (assuming you are developing new code)**
  - Three scales of code development: individual ('idiosyncratic'), small team ('hot shots'), big team to open source ('industrial')
  - Compute environment: small-scale ('individual PI', low diversity hardware), medium-scale ('single project', somewhat diverse hardware), large-scale ('multiple projects', very diverse hardware) -- **note scale here does not refer to problem size!**
  - **Step I:** Consider which categories your situation falls into, this will help set the portability constraints
  - Concrete advice is difficult; situations vary, look around you and see what other people are doing -- learn from them (adopt/reuse what works, dump what does not, **be ruthless**)
  - Simplicity is good (learn from Google!), avoid nonfunctional 'adornments'
  - Design for the future -- software life cycles should be long, **but often are not**
  - **Step II:** Most science projects start with a compact 'software core' that grows in multiple directions, pay attention to planning the structure of the core and the extension paths -- things will often not work as expected so make sure the structure is sufficiently flexible -- starting from scratch should be largely a reconfiguration of key software elements; identify these elements and design around them
  - Performance and portability are often in opposition, but they can be co-aligned -- as in HACC

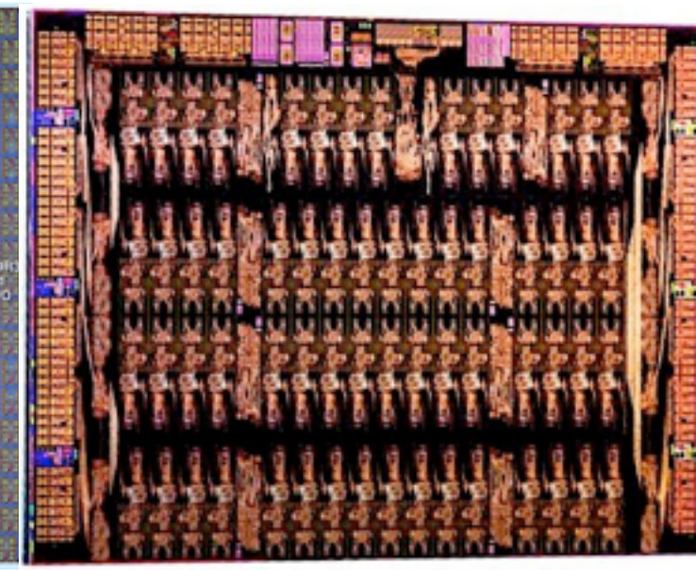
Co-Design vs. Code Design

- **HPC Myths**
  - The magic compiler
  - The magic programming model/ language (DSL)
  - Special-purpose hardware
  - Co-Design?
- **Dealing with (Current) HPC Reality**
  - Follow the architecture
  - Know the boundary conditions
  - There is no such thing as a 'code port'
  - Think out of the box
  - Get the best team
  - Work together

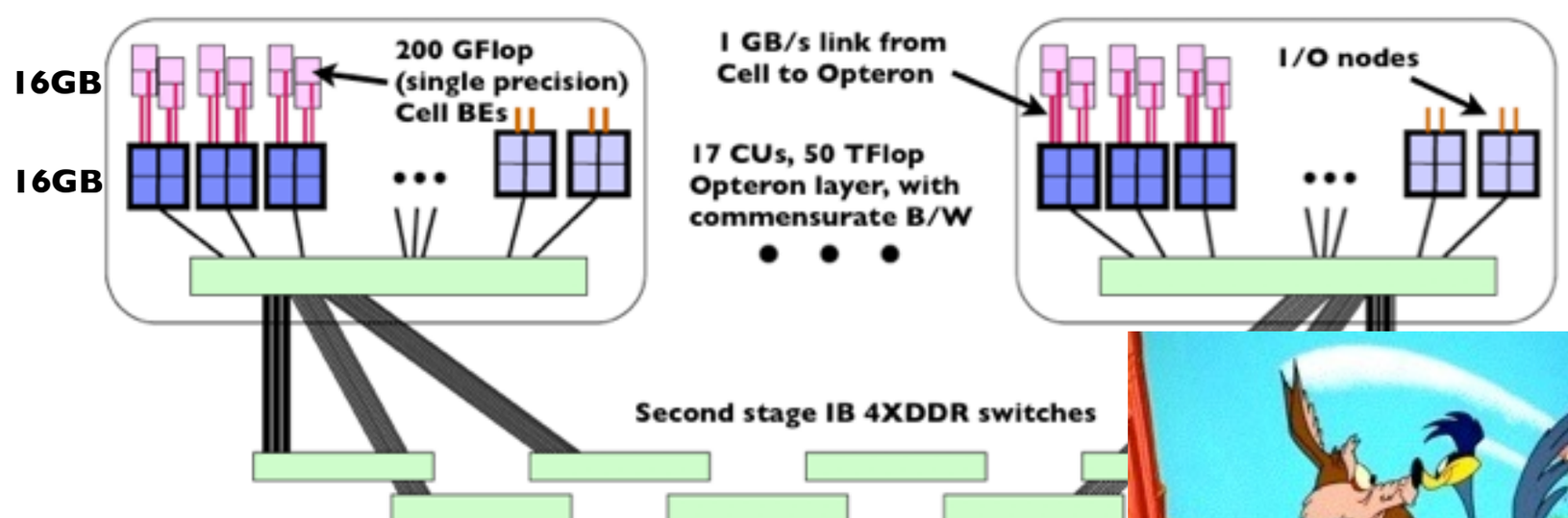
BQC:
- 16 cores
- 205 GFlops, 16 GB
- 32 MB L2, crossbar at 400 GB/s (memory connection is 40 GB/s)
- 5-D torus at 40 GB/s

Xeon Phi (Knights Corner):
- 60 cores
- 1 TFlops, 8 GB
- 32 MB L2, ring at 300 GB/s (connects to cores and memory)
- 8 GB/s to host CPU





Average performance speed-up on ~10 applications codes on Titan is ~2 (ranging from 1-8), but of Titan's 27 PFlops, only 2.5 PFlops are in the CPU! What is wrong with this picture? (BTW, it's not Titan's fault!)



Roadrunner: The Original Driver for HACC

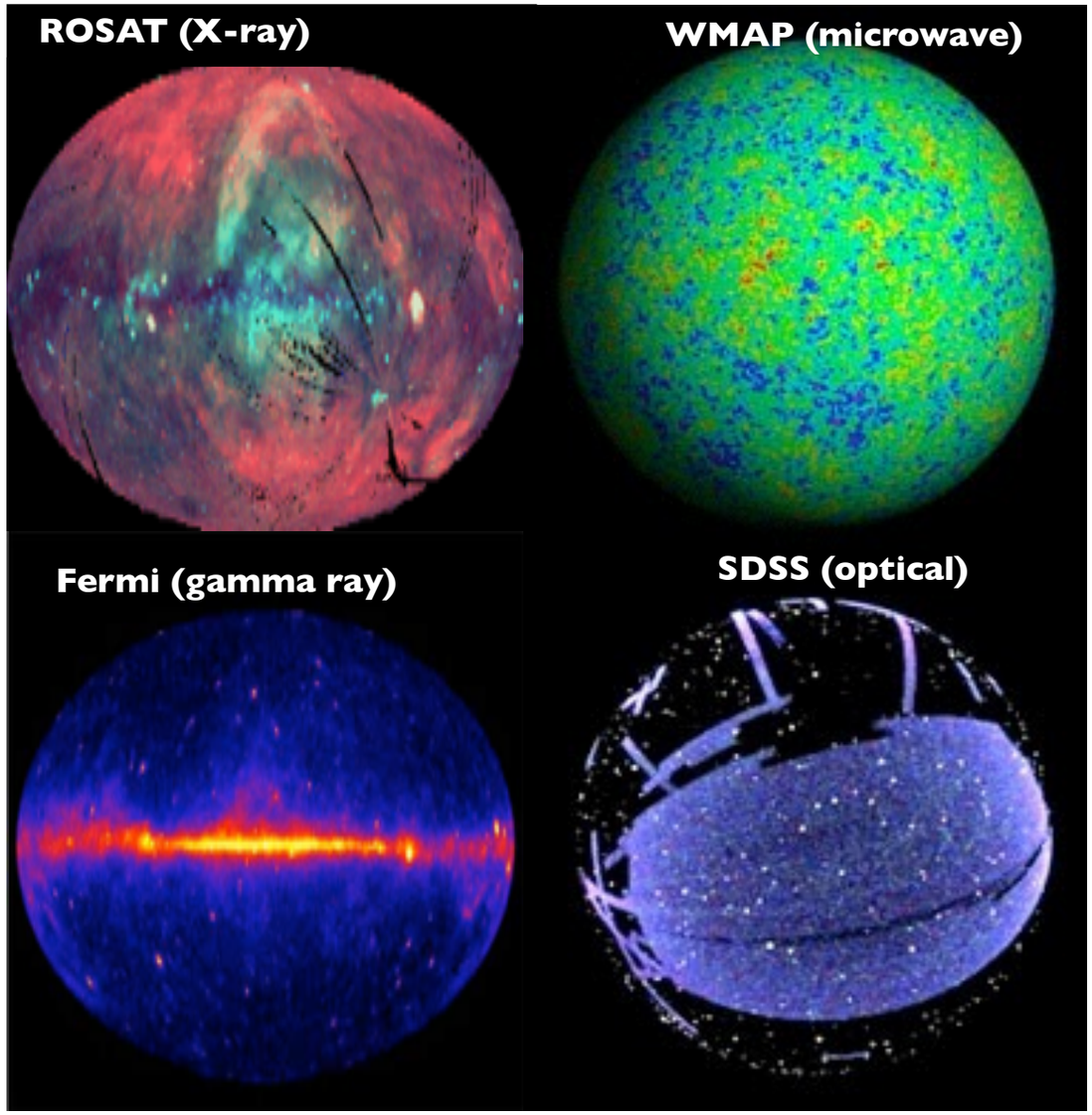
Why HACC?: ‘Precision’ Cosmology I

- Instrumentation Advances
- Cosmic Acceleration
- Nature of Dark Matter
- Primordial Fluctuations
- Neutrinos
- Cosmic Structure Formation

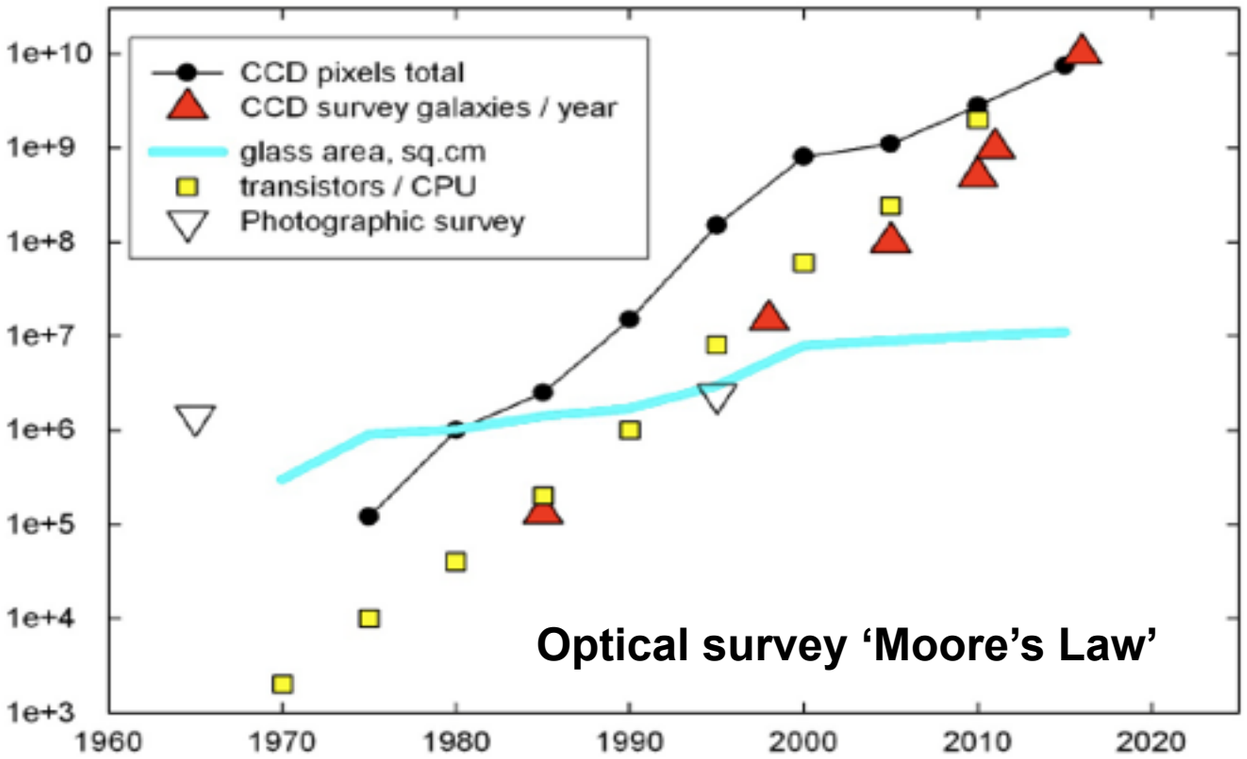
Boyle, Smith

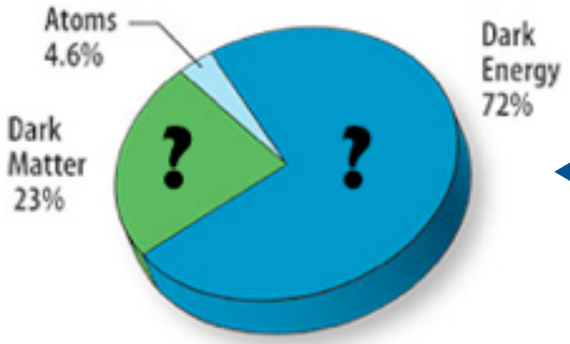
Perlmutter, Riess, Schmidt

Mather, Smoot



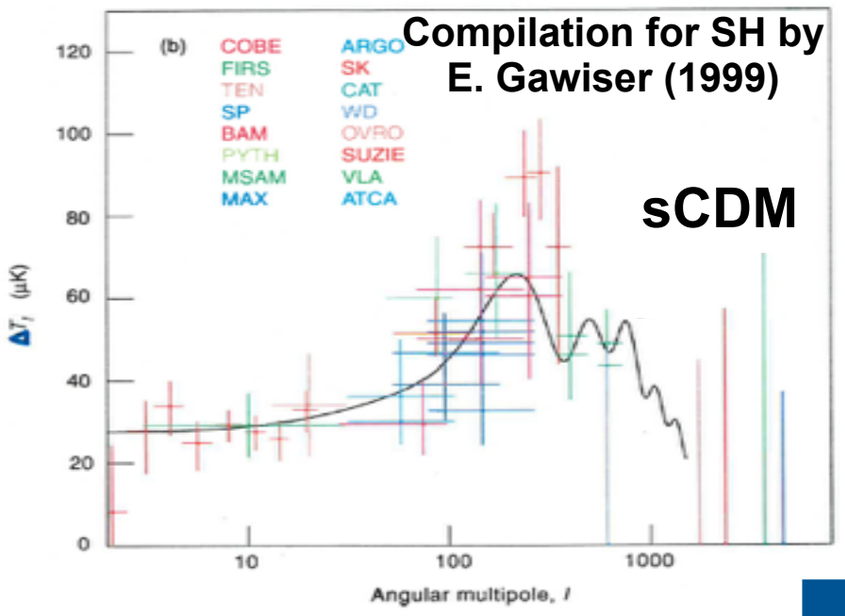
The Source of Knowledge: Sky Surveys

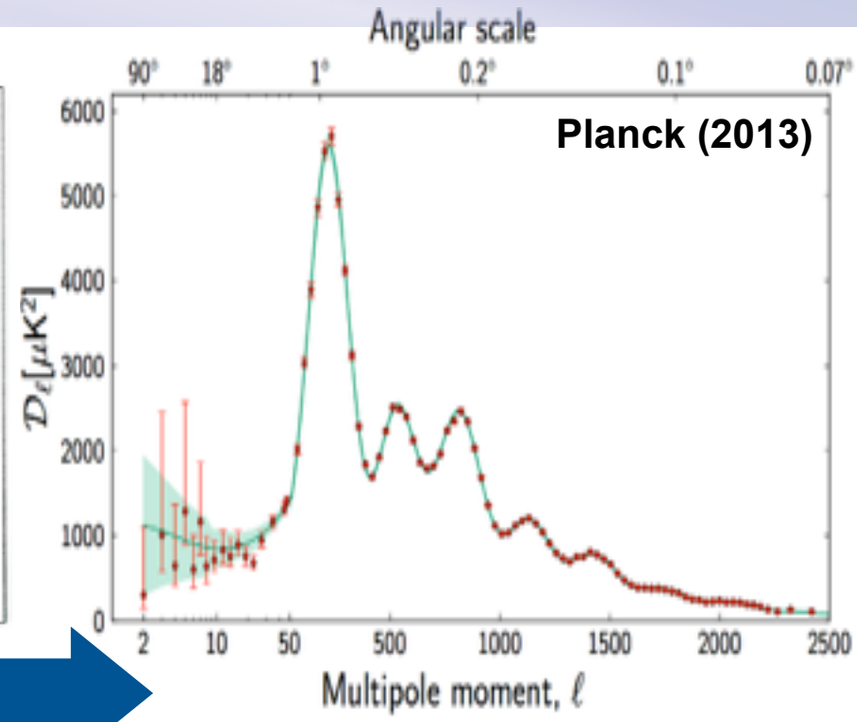




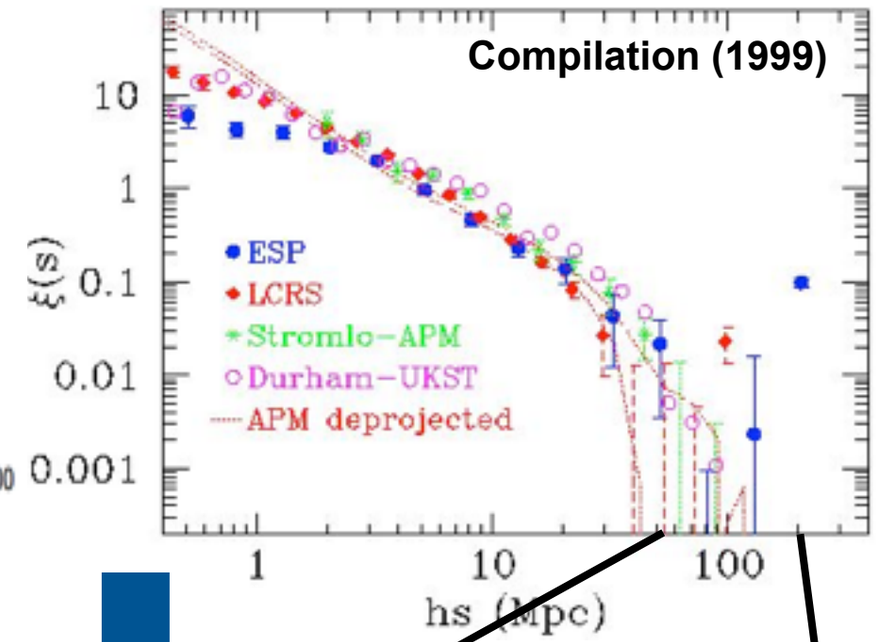
The Cosmic Puzzle: Who ordered the rest of it?

Why HACC: Precision Cosmology II

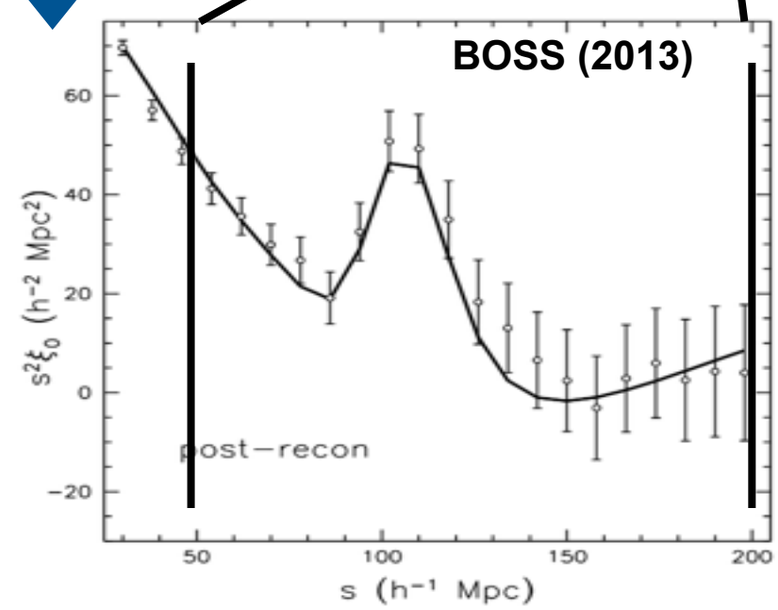


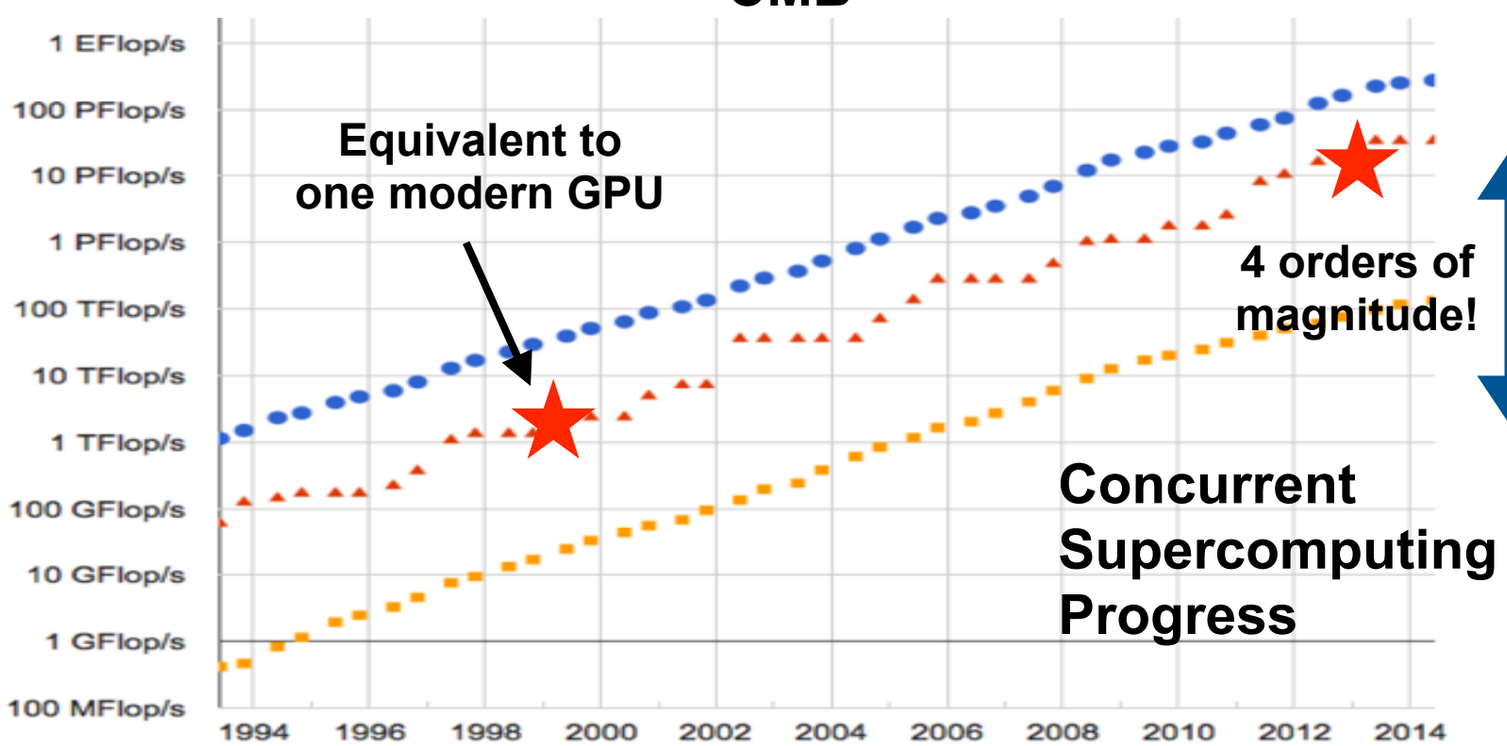


CMB



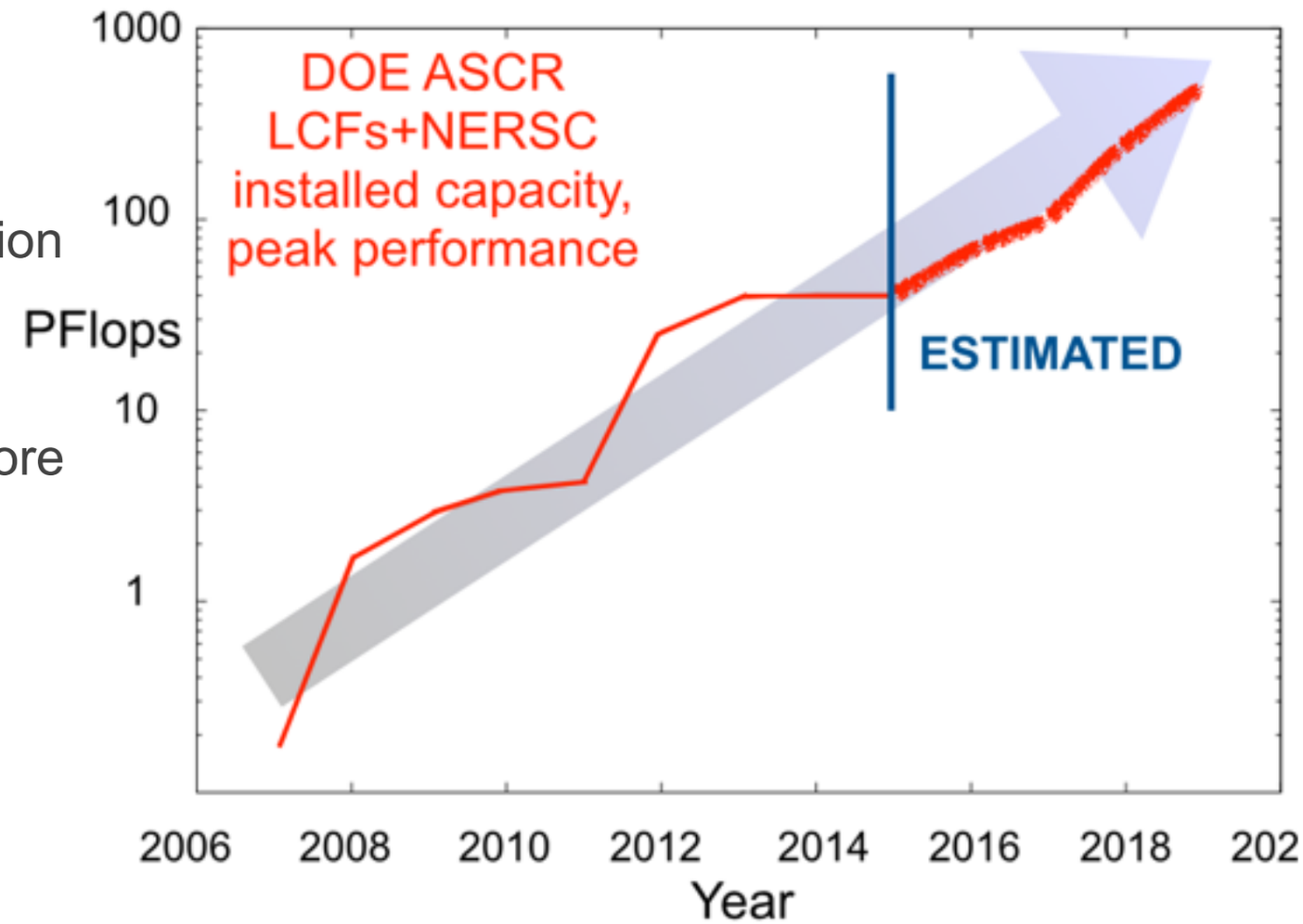
LSS

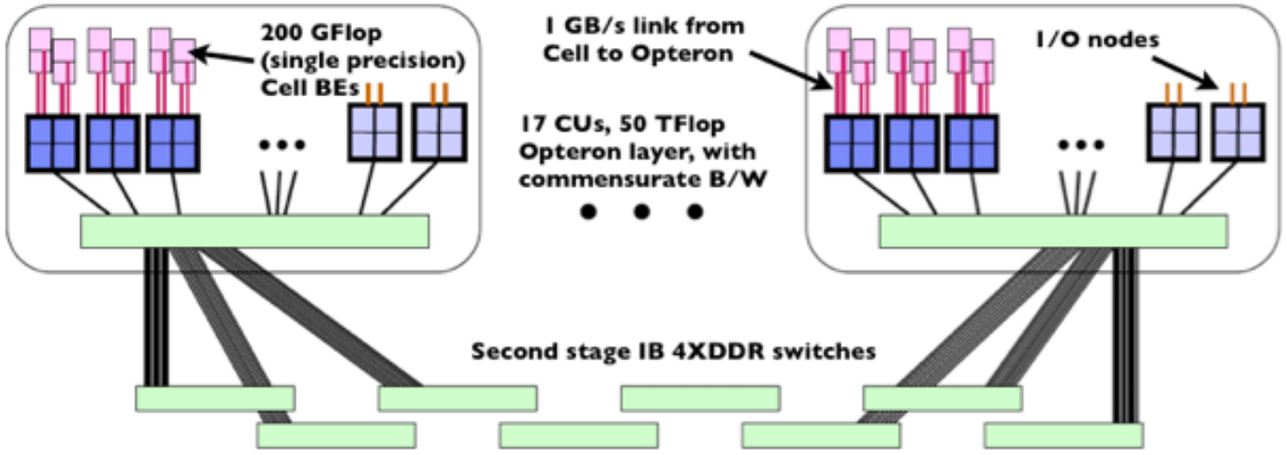




Why Another "Code"?

- **HPC systems:** "faster = more"
  - More nodes
    - Separate memory spaces
    - Relatively slow network communication
  - More complicated nodes
    - Architectures
      - Accelerators, multi-core, many-core
    - Memory hierarchies
      - CPU main memory
      - Accelerator main memory
      - High-bandwidth memory
      - Non-volatile memory
- **Portable performance**
  - Massively parallel/concurrent
  - Adapt to new architectures
    - Organize and deliver data to the right place in the memory hierarchy at the right time
    - Optimize floating point execution
  - Not possible with off-the-shelf codes





Roadrunner Architecture (2008)

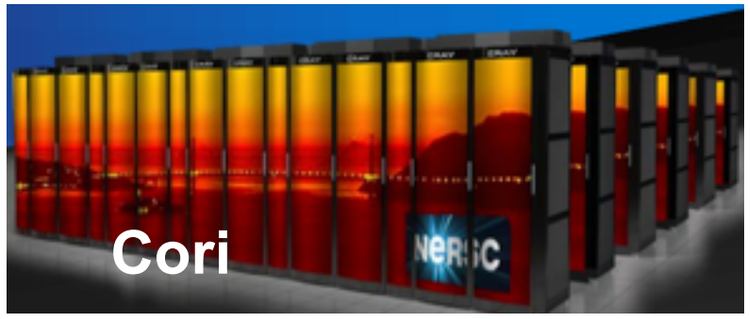
HACC Ideas and Features

What is HACC?

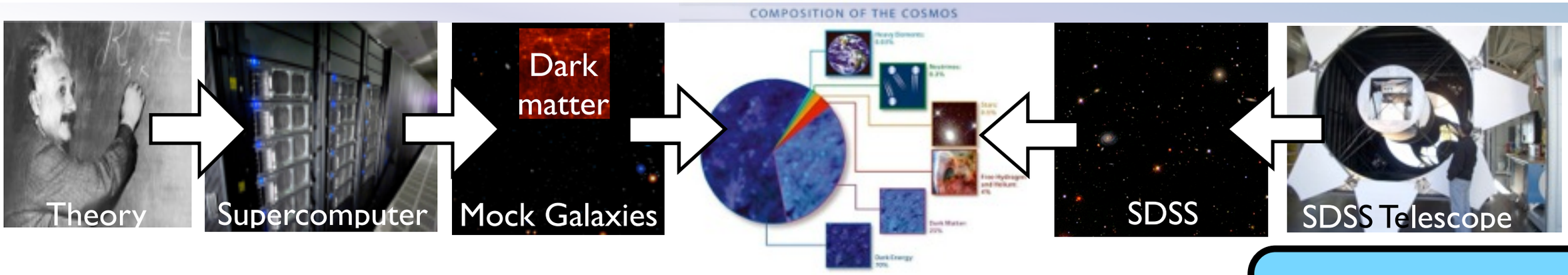
HACC (Hardware/Hybrid Accelerated Cosmology Code) Framework

- **HACC does very large high-resolution cosmological simulations**
  - Design Imperative: Must run at high performance on all supercomputer architectures at full scale
  - First production science code to break 10PFlops (sustained)
  - Combines a number of algorithms using a ‘mix and match’ approach based on a 2-level structure
  - Perfect weak scaling
  - Strong scales to better than 100 MB/core
  - World’s largest high-resolution cosmology simulations on Mira and Titan
  - CORAL benchmark code: Chosen for early science projects on Cori, Summit, and Theta

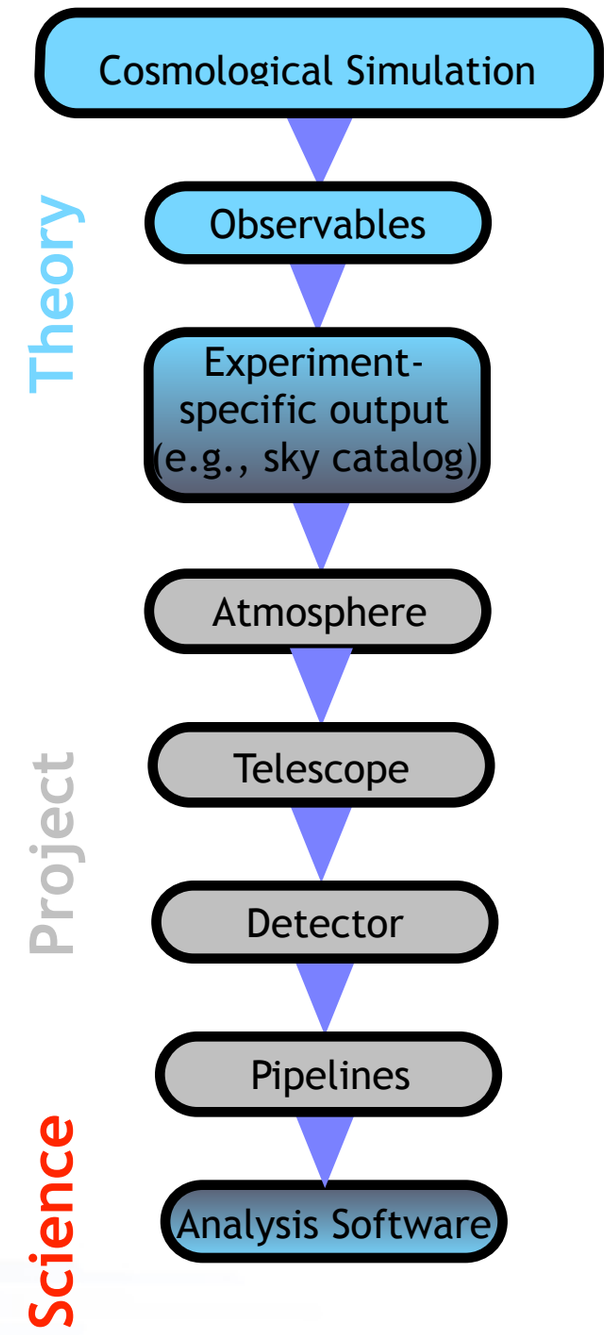
Next up for HACC:



Role of Computation in Cosmology



- **Three Roles of Cosmological Simulations**
  - Basic theory of cosmological probes
  - Production of high-fidelity 'mock skys' for end-to-end tests of the observation/analysis chain
  - Essential component of analysis toolkits
- **Extreme Simulation and Analysis Challenges**
  - Large dynamic range simulations; control of subgrid modeling and feedback mechanisms
  - Design and implementation of complex analyses on large datasets; new fast (approximate) algorithms
  - Solution of large statistical inverse problems of scientific inference (many parameters, ~10-100) at the ~1% level

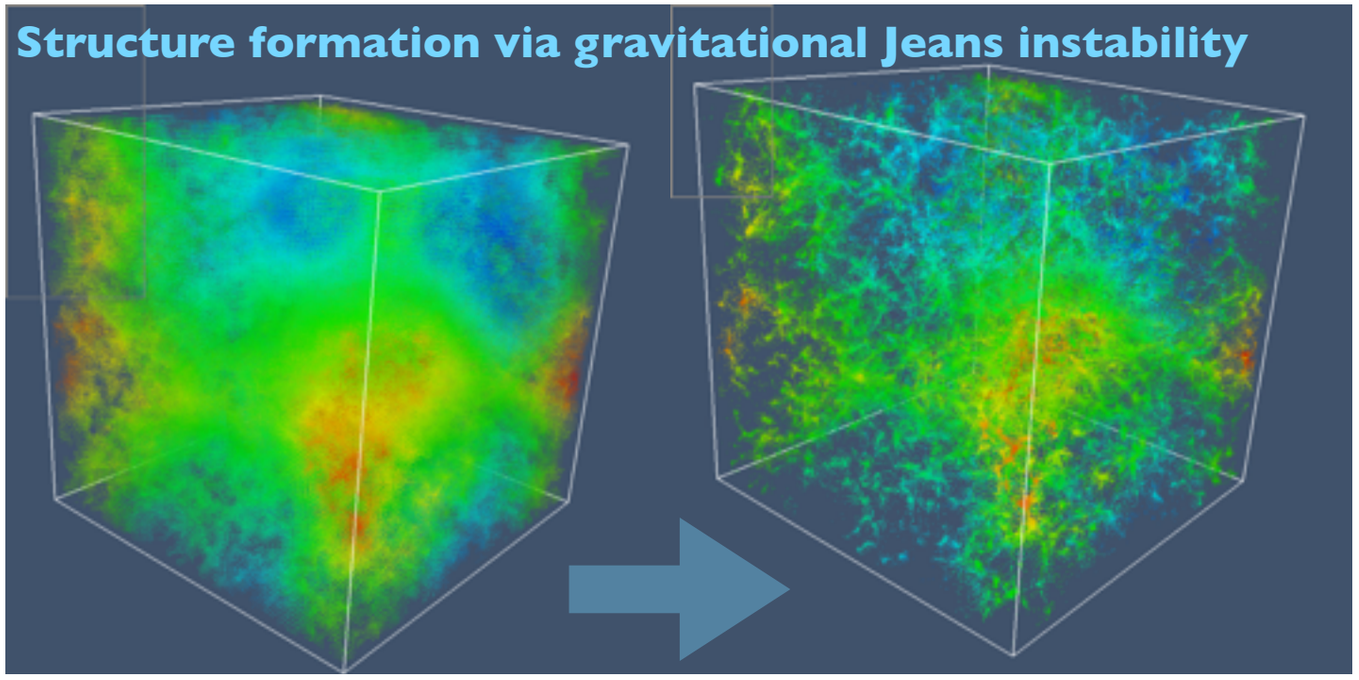


Simulating the Universe

- **Key Role of Gravity**
  - Gravity dominates at large scales: the Vlasov-Poisson equation (VPE)
  - VPE is 6-D; cannot be solved as a PDE
- **N-Body Methods**
  - No shielding in gravity (essentially long range interactions)
  - Technique is naturally Lagrangian
  - Are errors controllable?
- **More Physics**
  - Smaller scale 'gastrophysics' effects added via hydro solvers, subgrid modeling, or post-processing (major topic)
- **Phenomenology**
  - Calibrate simulations against observations

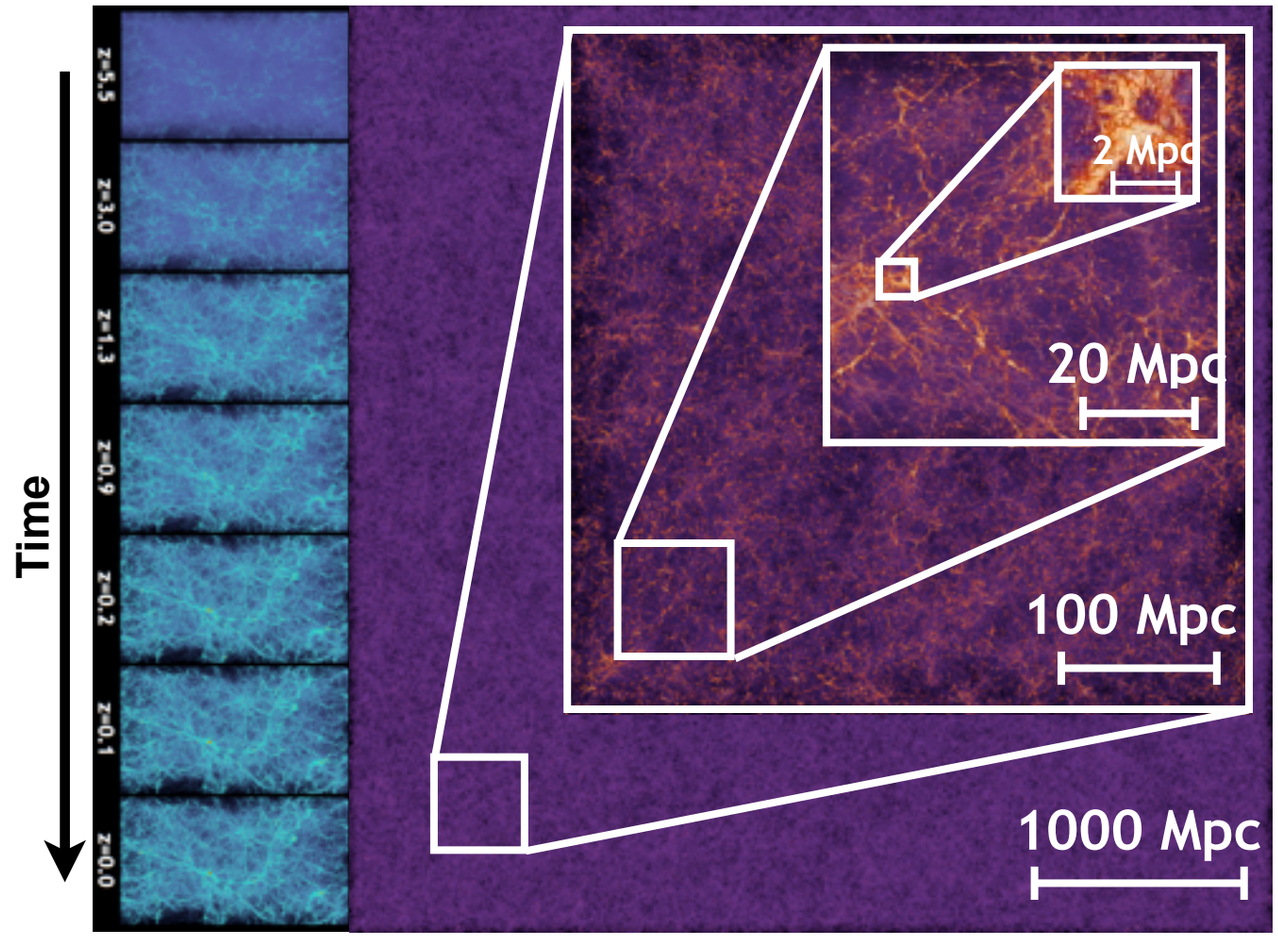
$$
\begin{aligned}
\frac{\partial f_i}{\partial t} + \dot{\mathbf{x}}\frac{\partial f_i}{\partial \mathbf{x}} - \nabla\phi\frac{\partial f_i}{\partial \mathbf{p}} &= 0, \qquad \mathbf{p} = a^2\dot{\mathbf{x}}, \\
\nabla^2\phi &= 4\pi G a^2(\rho(\mathbf{x},t) - \langle\rho_{\rm dm}(t)\rangle) = 4\pi G a^2 \Omega_{\rm dm}\delta_{\rm dm}\rho_{\rm cr}, \\
\delta_{\rm dm}(\mathbf{x},t) &= (\rho_{\rm dm} - \langle\rho_{\rm dm}\rangle)/\langle\rho_{\rm dm}\rangle), \\
\rho_{\rm dm}(\mathbf{x},t) &= a^{-3}\sum_i m_i \int d^3\mathbf{p} f_i(\mathbf{x},\dot{\mathbf{x}},t).
\end{aligned}
$$

Cosmological Vlasov-Poisson Equation: A 'wrong-sign' electrostatic plasma with time-dependent particle 'charge', Newtonian limit of the Vlasov-Einstein equations



HACC's Domain: The "Bleeding Edge"

- **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 — ~trillion particle simulations
  - Ratio of largest object mass to lightest is ~100,000:1
- **Physics:**
  - Gravity dominates at scales greater than ~0.1 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

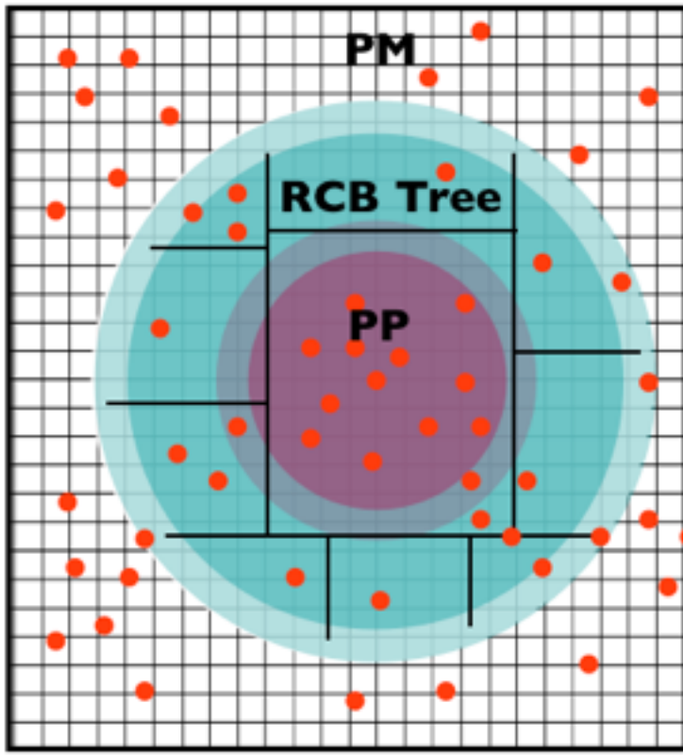


Gravitational Jeans Instability: 'Outer Rim' run with 1.1 trillion particles

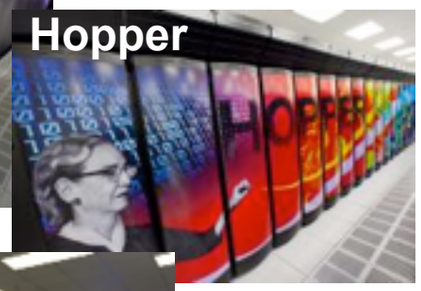
Key motivation for HACC (Hardware/ Hybrid Accelerated Cosmology Code): Can the Universe be run as a short computational 'experiment'?

Opening the HACC 'Black Box': Design Principles

- **Optimize Next-Generation Code 'Ecology':** Numerical methods, algorithms, mixed precision, data locality, scalability, I/O, in situ analysis -- life-cycle significantly longer than architecture timescales
- **Framework design:** Support a 'universal' top layer + 'plug-in' optimized node-level components; minimize data structure complexity and data motion -- support multiple programming models
- **Performance:** Optimization stresses scalability, low memory overhead, and platform flexibility; assume 'on your own' for software support, but hook into tools as available (e.g., ESSL FFT)
- **Optimal Splitting of Gravitational Forces:** Spectral Particle-Mesh melded with direct and RCB tree force solvers, short hand-over scale (dynamic range splitting ~ 10,000 X 100)
- **Compute to Communication balance:** Particle Overloading
- **Time-Stepping:** Symplectic, sub-cycled, locally adaptive
- **Force Kernel:** Highly optimized force kernel takes up large fraction of compute time, no look-ups due to short hand-over scale
- **Production Readiness:** runs on all supercomputer architectures; exascale ready!



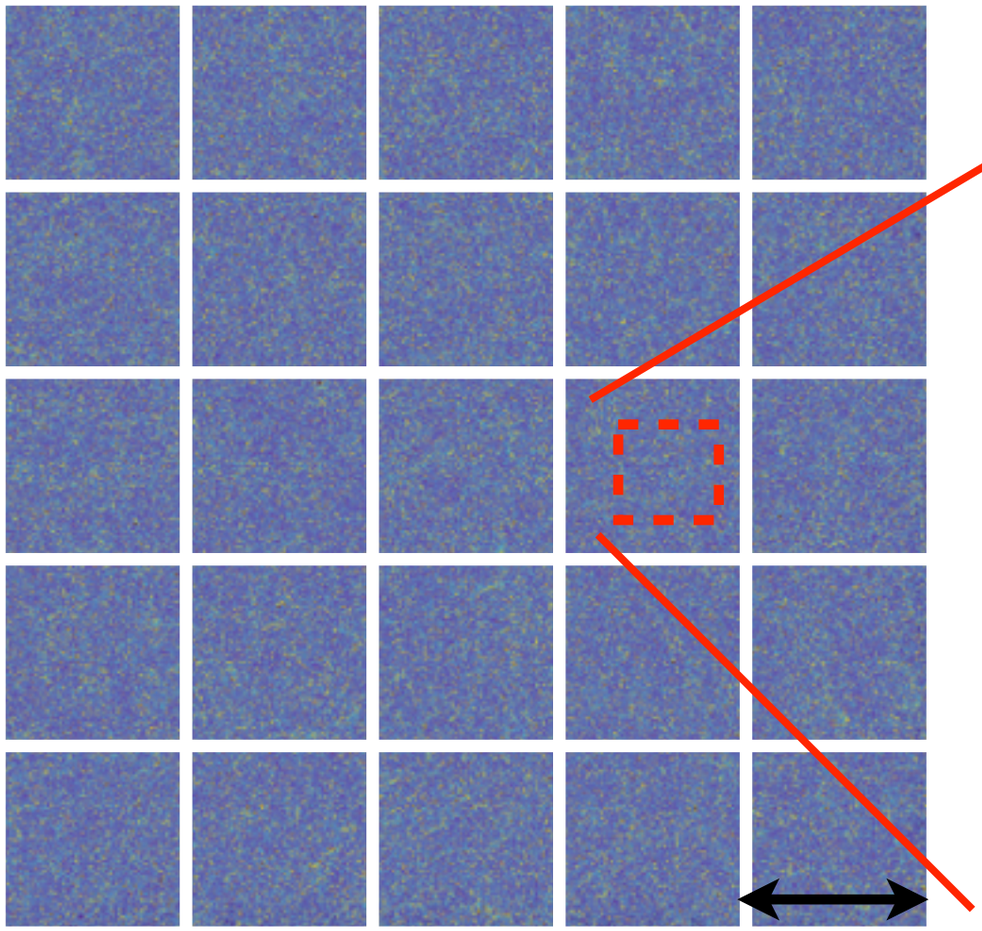
HACC force hierarchy (PPTreePM)

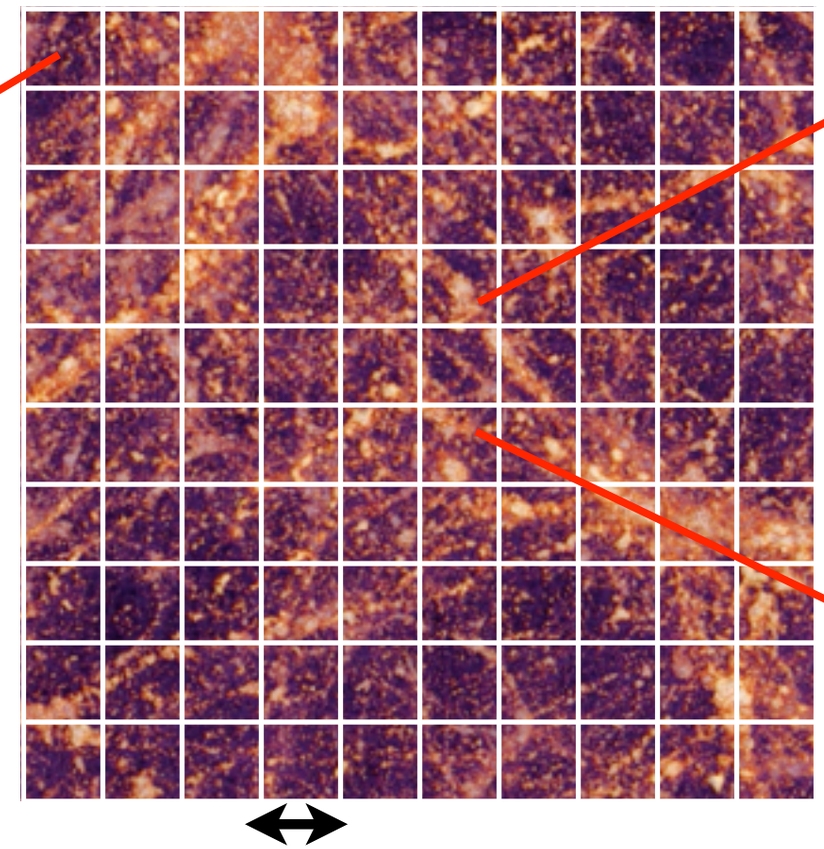


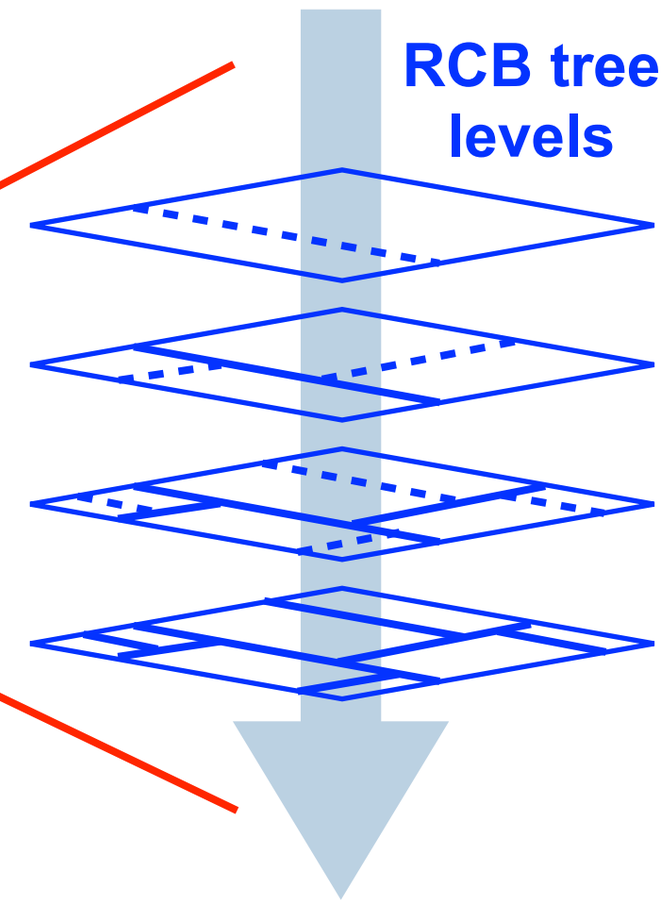
Dissecting the N-Body Problem

- **Physics Idea:**
  - Smooth long-range component — slowly varying “mass field”
  - Fluctuating short-range component — faster time-scale, discrete force contributions
- **Implementation:**
  - Smooth field produced from depositing N-body particles on a (large) grid — O(10,000) cube (4 orders of magnitude dynamic range); solve Poisson equation on these scales using a ‘quiet’ spectral method (matching and force-solving undertaken as a single task)
  - Small scales: match the small-scale force to the long-range force using spectral filtering at the smallest practical scales — O(100) dynamic range
  - Compute small-scale forces with the algorithm that makes the most sense for a given architecture (use mixed precision)
- **Scale Isolation:**
  - Small-scale forces are made (essentially) ultra-local using particle overloading
  - Sub-cycled time-stepping using a split-operator symplectic solver separates the force scales in time

‘HACC In Pictures’

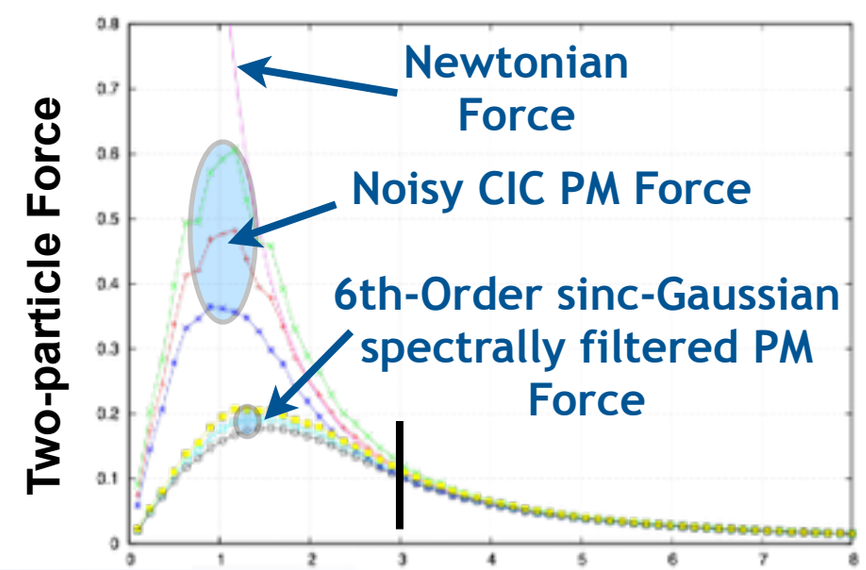






HACC Top Layer:
3-D domain decomposition with particle replication at boundaries (‘overloading’) for Spectral PM algorithm (long-range force)

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

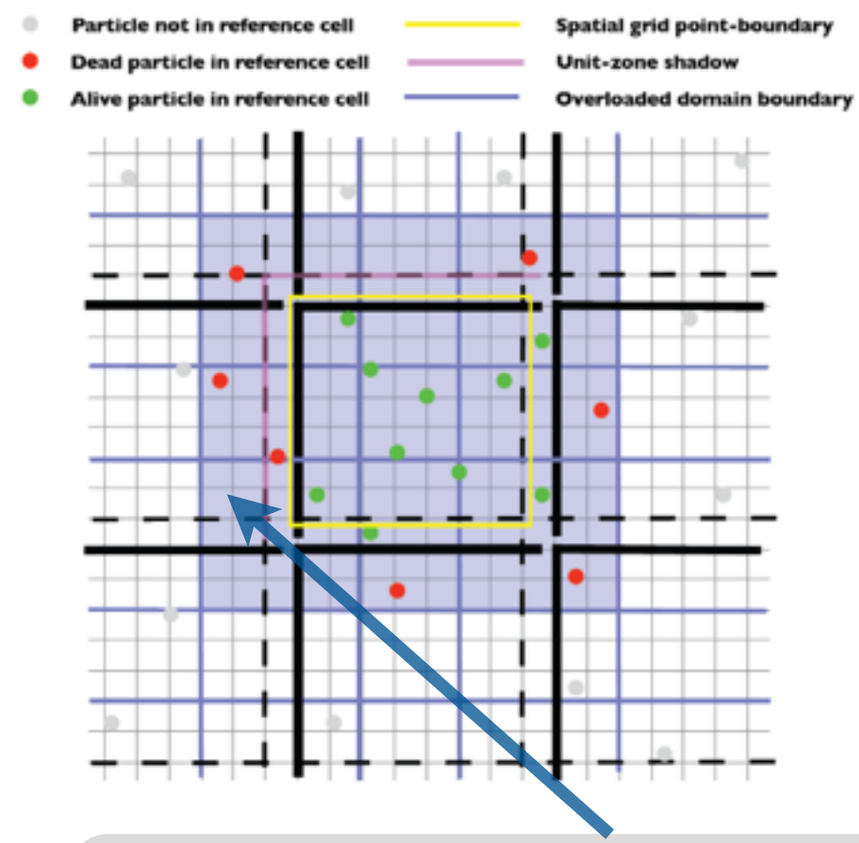


Host-side

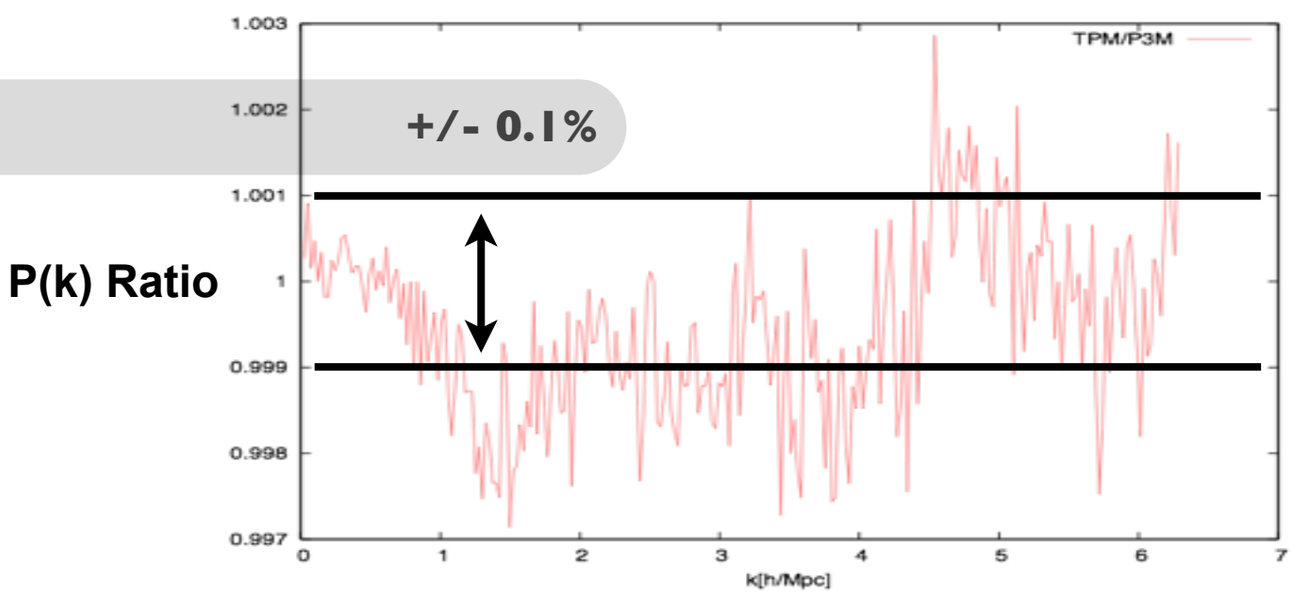
GPU: two options, P3M vs. TreePM

Particle Overloading and Short-Range Solvers

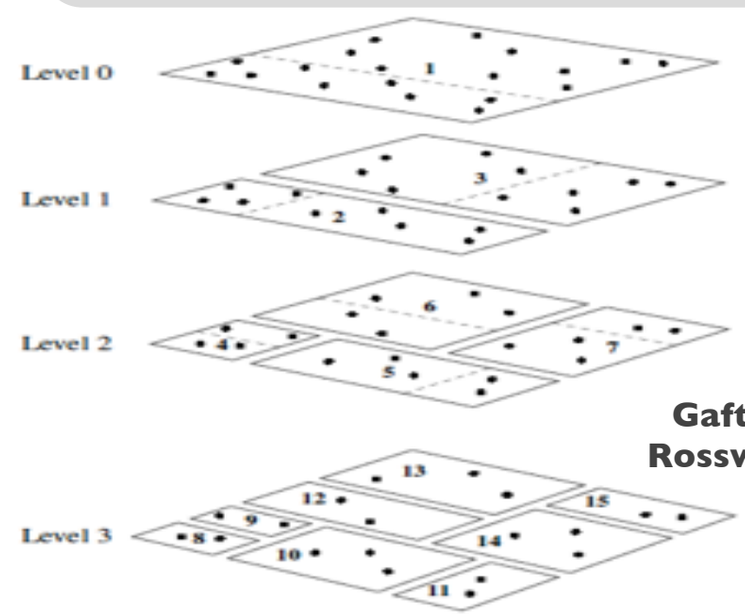
- **Particle Overloading:** Particle replication instead of conventional guard zones with 3-D domain decomposition -- minimizes inter-processor communication and allows for swappable short-range solvers **(IMPORTANT)**
- **Short-range Force:** Depending on node architecture switch between P3M and PPTreePM algorithms (pseudo-particle method goes beyond monopole order), by tuning number of particles in leaf nodes and error control criteria, optimize for computational efficiency
- **Error tests:** Can directly compare different short-range solver algorithms



Overload Zone (particle 'cache')



HACC Force Algorithm Test: PPTreePM vs. P3M

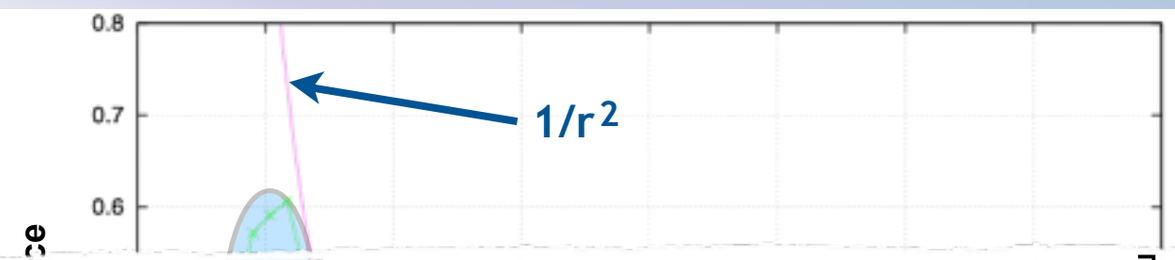


Gafton and Rosswog 2011

RCB Tree Hierarchy

Splitting the Force: The Long-Range Solver

- **Spectral Particle-Mesh Solver:** Custom (large) FFT-based method -- uses (i) 6-th



$$G_6(\mathbf{k}) = \frac{45}{128}\Delta^2 \left[\sum_i \cos\left(\frac{2\pi k_i \Delta}{L}\right) - \frac{5}{64}\sum_i \cos\left(\frac{4\pi k_i \Delta}{L}\right) + \frac{1}{1024}\sum_i \cos\left(\frac{8\pi k_i \Delta}{L}\right) - \frac{2835}{1024} \right]^{-1}$$

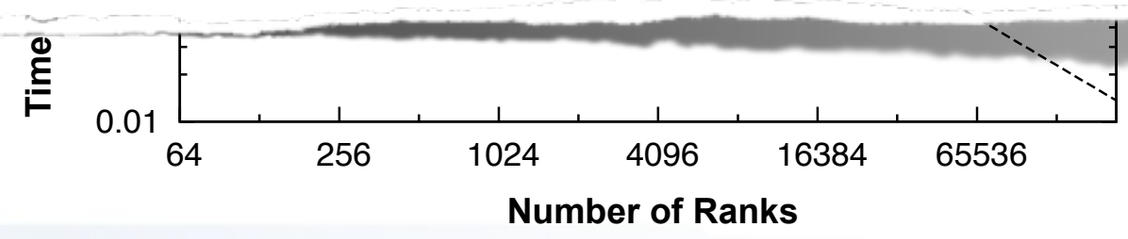
$$\left.\frac{\Delta f}{\Delta x}\right|_4 = \frac{4}{3}\sum_{j=-N+1}^{N} iC_j e^{(2\pi jx/L)} \frac{2\pi j\Delta}{L}\frac{\sin(2\pi j\Delta/L)}{2\pi j\Delta/L} - \frac{1}{6}\sum_{j=-N+1}^{N} iC_j e^{(2\pi jx/L)} \frac{2\pi j\Delta}{L}\frac{\sin(4\pi j\Delta/L)}{2\pi j\Delta/L}$$

where the C_j are the coefficients in the Fourier expansion of f

$$S(k) = \exp\left(-\frac{1}{4}k^2\sigma^2\right)\left[\left(\frac{2k}{\Delta}\right)\sin\left(\frac{k\Delta}{2}\right)\right]^{n_s}$$

$$f_{grid}(r) = \frac{1}{r^2}\tanh(br) - \frac{b}{r}\frac{1}{\cosh^2(br)} + cr\left(1+dr^2\right)\exp\left(-dr^2\right) + e\left(1+fr^2+gr^4+lr^6\right)\exp\left(-hr^2\right)$$

the global timestep, where 'S' is split into multiple 'SKS' local force steps

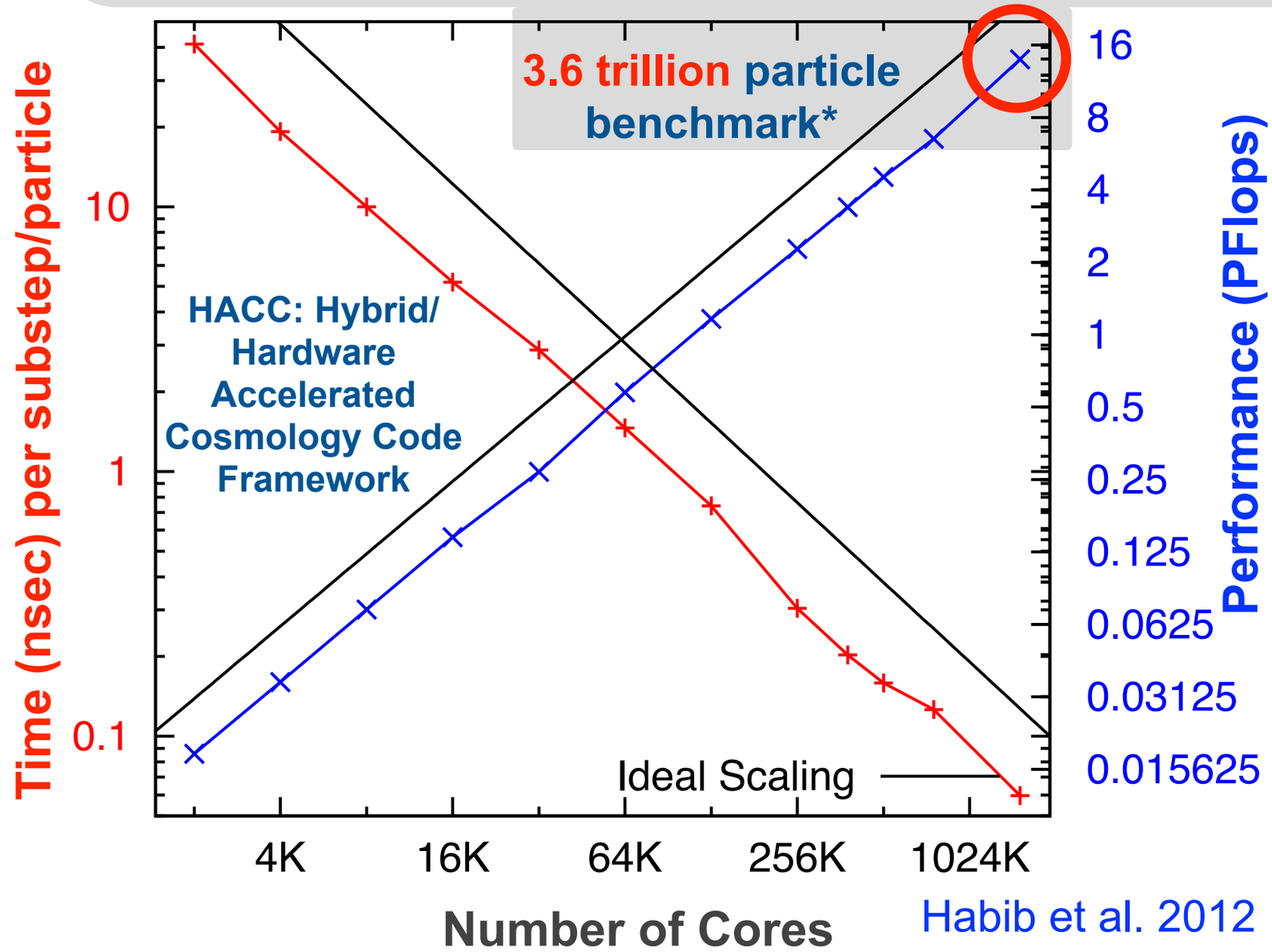


HACC on the BG/Q

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



Habib et al. 2012

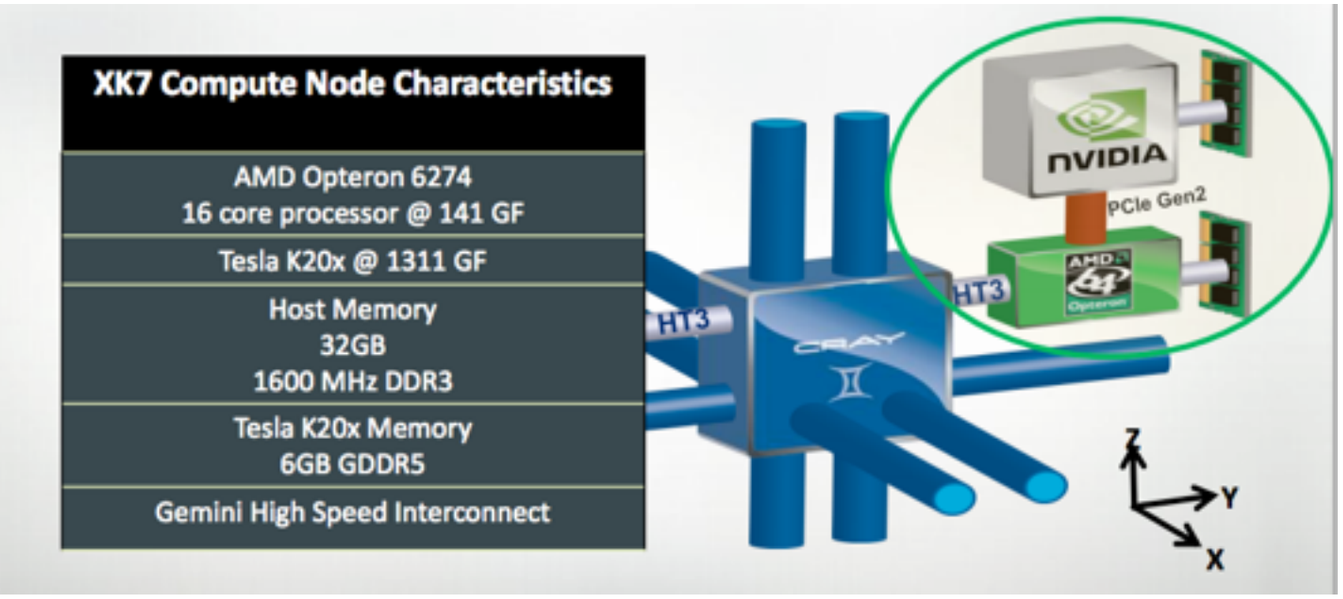
HACC weak scaling on the IBM BG/Q (MPI/OpenMP)

*largest ever run

Accelerated Systems: Specific Issues

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!)



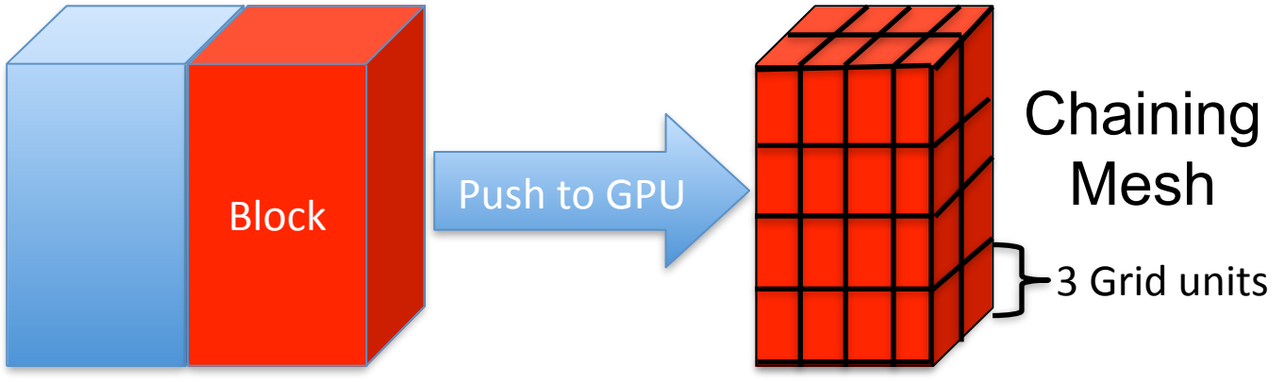
Strategies for Success

- 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)
- Like it or not, pragmas will never be the full answer

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)

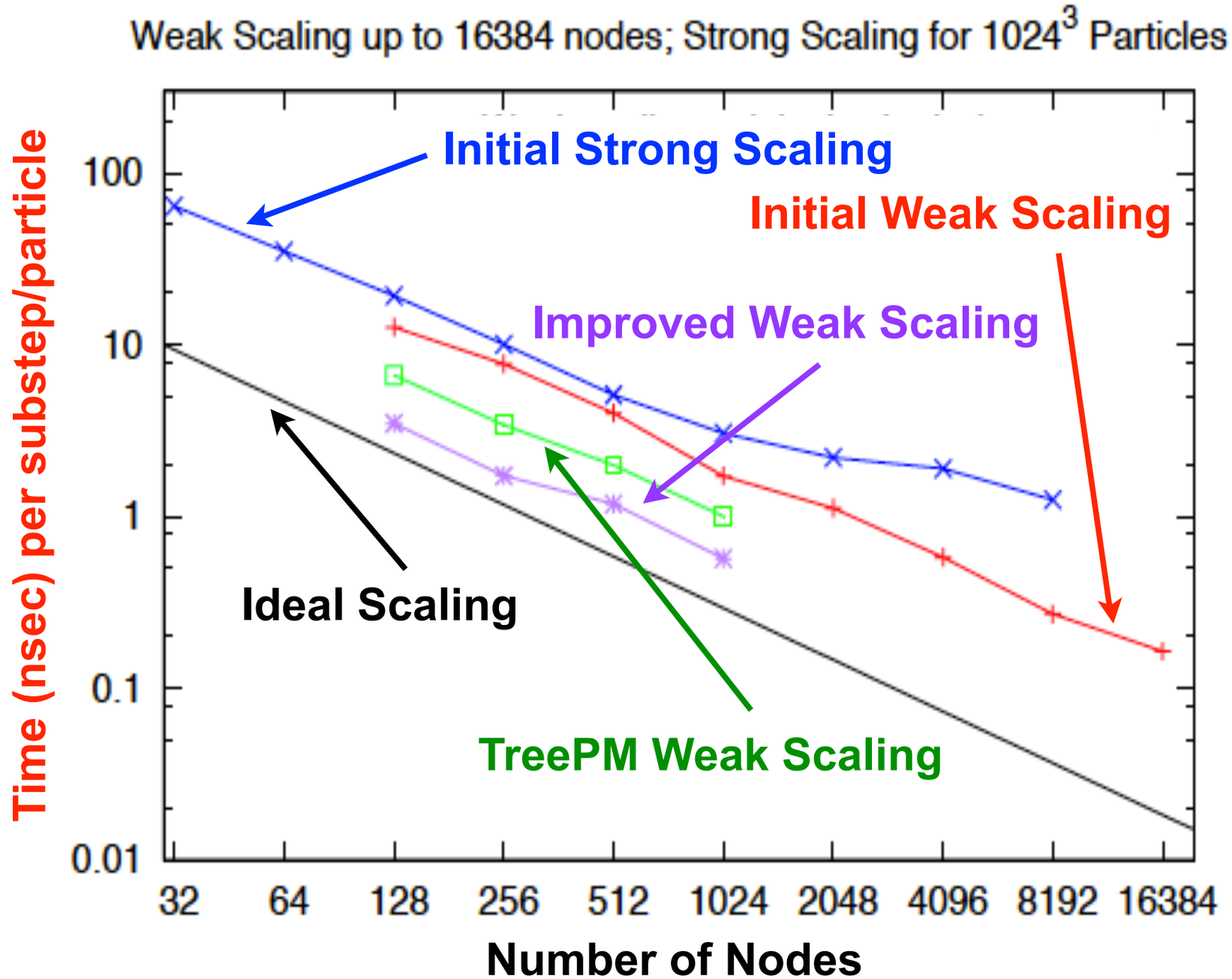


New Implementations (OpenCL and CUDA):

- 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

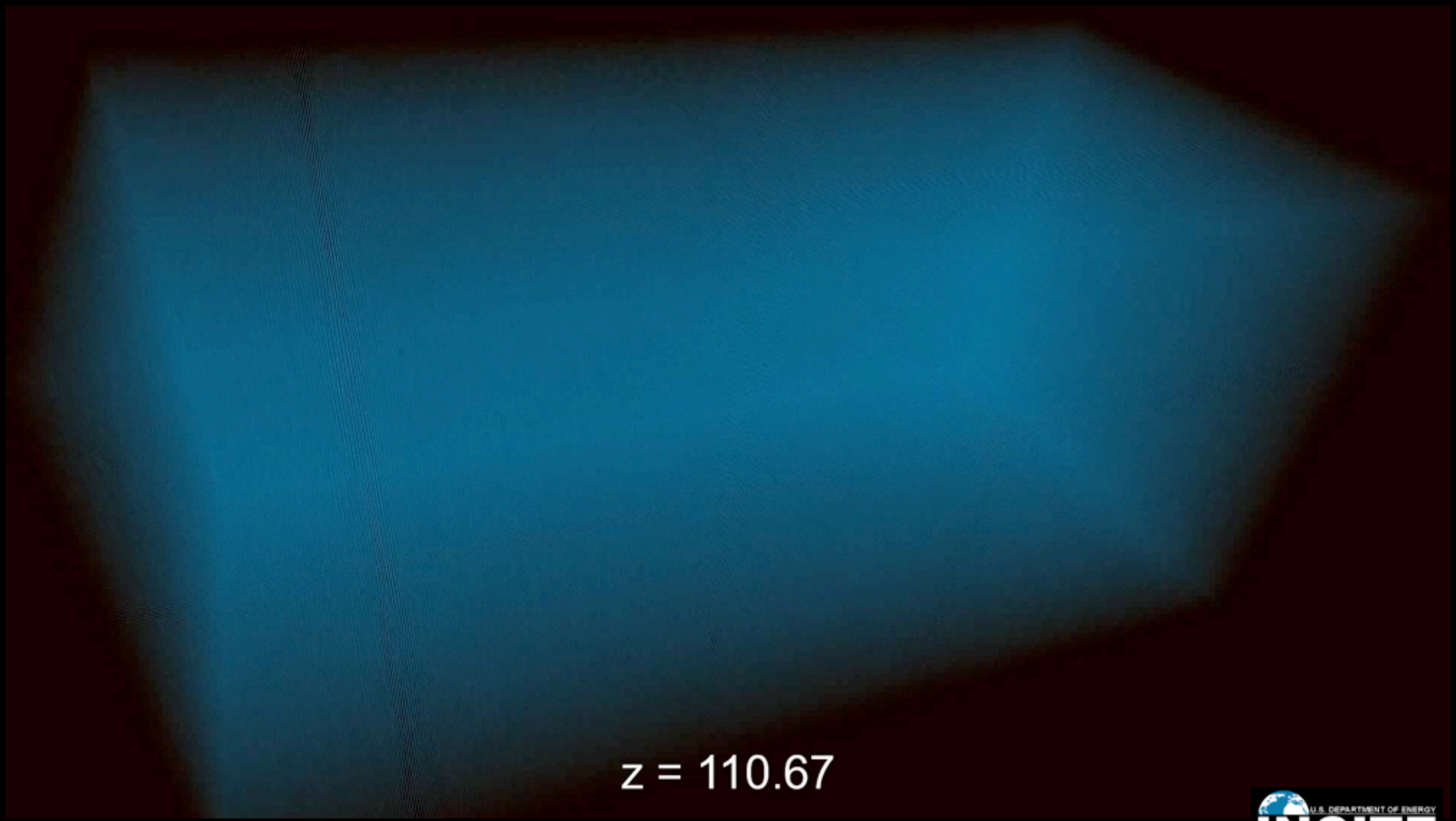
HACC on Titan: GPU Implementation Performance

- 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



99.2% Parallel Efficiency

The high resolution Q Continuum Simulation, finished July 13 on ~90% of Titan under INCITE, evolving more than half a trillion particles. Shown is the output from one node (~33 million particles), 1/16384 of the full simulation



Q Continuum: Extradimensional plane of existence

Visualization: Silvio Rizzi, Joe Insley et. al., Argonne

Summary

- 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
- Main HACC paper — http://arxiv.org/abs/1410.2805;
- Also — http://authors.elsevier.com/a/1RSrG_VU94HRMb (until Sep 20)