Build It and They Will Come: How Hardware Influences Large-Scale Simulation

Science at Extreme Scales: Where Big Data Meets Large-Scale Computing Tutorials

14 Sep 2018

Jeffrey A. F. Hittinger
Director
Thanks to

- David Keyes, KAUST
- Rob Neely, LLNL
- And others...
Advancements in HPC have happened over three major epochs, and we’re settling on a fourth (heterogeneity)

Each of these eras defines a new common programming model and transitions are taking longer; we are entering a fourth era
Pre-WWII – the theoretical foundations of modern computing are built in the U.S. and England

- **1830’s – The Difference Engine**
  - Charles Babbage / Ada Lovelace
  - Not built during his lifetime

- **1930’s – The Turing Machine**
  - Alan Turing (England / Princeton)
  - Theory of programmable algorithms

- **1940’s – ENIAC**
  - Mauchly/Eckert – Moore School of Eng. (Philadelphia)
  - 1st general programmable electronic computer

- **1940’s – EDVAC design.**
  - John Von Neumann (Los Alamos)
  - Basis for “stored program architecture” still used today

Ballistics projectile calculations were the first “killer app” for computers, and were historically done by a pipeline of human “computers”
The “mainframe” era – general purpose computers designed for scientific computing

- **Univac 1**
  - First machine installed at LLNL in 1953

- **IBM 701**
  - Installed at Los Alamos also in 1953
  - Williams tubes (fast memory, but unreliable)

- **IBM 704**
  - Core memory, floating point arithmetic, CRT
  - Commercially successful

- **IBM 709**
  - Seamless porting

- **IBM 7090**
  - Transistor-based
  - Large speed increases

- **Univac LARC**
  - Co-designed with, and for, LLNL
  - One of the first transistor-based machines
The “mainframe” era – continued

- **IBM 7030 (Stretch)**
  - Competitor to LARC
  - Considered a failure at the time (only achieved 50% of performance goals)
  - Introduced many concepts that went into the IBM System/360

- **CDC 1604**
  - First designed by Seymour Cray

- **CDC 3600**
  - 48-bit words (higher precision)

- **CDC 6600**
  - Considered the first real “supercomputer” (1st at CERN)
  - Full separation of input/output from computing (precursor to RISC designs)

- **CDC 7600**
  - Hierarchical memory design
  - Instruction pipeline (allowed for modest parallelism)
  - Fastest computer in world from 1969-1975
Programming in the mainframe era

- Programming was often an intimate exercise with the particular computer you were targeting
- Entirely serial programming
- Lack of sufficient memory was often the limiting factor in scientific software design
- This era saw huge innovations in hardware and software, despite computer science not yet being an established discipline
  - Development of compilers (e.g. Fortran)
  - Operating systems (e.g. CTSS)
  - Batch environments and time-sharing
  - Use of terminals for viewing output
  - LLNL’s “Octopus” network
- U.S. stood largely alone in dominating the HPC market
The vector era

- **CDC Star 100**
  - First CDC design not led by Seymour Cray
  - One of the first commercial vector machines
  - Hard to achieve anything near peak performance
  - Large disparity between serial and vector code

- **Illiac IV (Burroughs & Univ of Illinois)**
  - First massively parallel machine portends future trends
  - Moved to NASA Ames in 1975 (defense at a public university? Bah)

- **Cray 1**
  - First machine developed at Seymour Cray’s new company
  - Applications saw immediate gains in performance (vs CDC 7600), and could incrementally improve with additional vectorization

- **Cray X-MP**
  - Designed by Steve Chen at Cray
  - First multi-processor supercomputer
The vector era – soon dominated by Cray in the U.S.

Cray 2
- Seymour’s 2nd design
- Flourinert-cooled
- Most performance gains were due to very fast memory

Cray Y-MP
- Successor to X-MP
- First delivered with Unix-based UNICOS OS

Cray C90

Cray J90
- “Mini” supercomputer – helped bridge gap into MPP era

Cray T90
- End of an era!

Other players in vector computer design:
US
- Convex
- Thinking Machines
- ETA (spinoff of CDC)
- IBM
Non-US
- NEC
- Fujitsu
- Hitachi
Programming vector computers in that era

- Machines were often delivered with no operating system—just hardware
  - Labs created their own software environments
  - Took years for the Cray compiler to catch up to the NNSA lab compilers

- Vector computers are designed to work with arrays and loops
  - Lots of those in scientific codes!

- Vectorizing was difficult, and many codes did not lend themselves to this
  - Loop dependencies (e.g. \( \text{fib}[i] = \text{fib}[i-1] + \text{fib}[i-2] \))
  - Conditional statements
  - Non-constant stride through memory
  - I/O or system calls
  - Function calls

Machine details matter for effective use...
Compilers became increasingly sophisticated at being able to perform some code transformations — Yet “automatic parallelization” never fully achieved its academic promises

Vectorized portions of code could realize anywhere from 2 – 50x speedups over a serial counterpart

Vectorization (via SIMD units) eventually made its way into commodity CPUs with the Pentium III (SSE) — Also, AVX, ARM Neon, Power QPX — (Not quite the same, but close enough...)

Stable programming model, but — Lots of effort for potentially limited payoff — Vectorized coding could get ugly!

Vectorization lives on in the form of SIMD units (e.g., Intel AVX)
Soon...mainframes (vector or not) faced the “attack of the killer micros”

- **Late 1980s; heavy influence by the PC/workstation market**
  - Reduced instruction set computing (RISC) processors dominated price-performance
  - Mainframes became “dinosaurs running COBOL”

- **Revolution driven largely by cost and scaling**
  - Cost: computing went mainstream
  - Scaling: commodity processors could be interconnected
  - Programming: rewrite from vector code to commodity processors

With industry focusing its investment on commodity processors, the writing was on the wall for the labs: leverage or fall behind the curve...
Commodity hardware components thus defines the distributed memory era

- Workstations (e.g. Silicon Graphics) were doing the work of mainframes for 50x cheaper!
  - SGI acquires Cray Research

- The relative ease of building distributed memory systems launched a wave of innovation and new supercomputer companies in the 90’s
  - IBM, Cray [Research, Computer, Inc], SGI, Intel, nCube, MasPar, Alliant, Multiflow, Kendall Square, BBN, DEC, Sequent, Convex, Encore, HP, Meiko, Supertek, Floating Point Systems, Tera, MIT J-machine, ...

- Fast commodity Ethernet and Linux eventually lead to the “Beowulf” explosion
  - Anyone can build a supercomputer now!
  - Great boon to universities and small companies
  - By the 2000’s, the list of dead or dying U.S. Supercomputing companies exceeded those still alive
The distributed computing or MPP era

- **ASCI program launches in ~1995**
  - Replace underground nuclear testing with science-based stockpile stewardship (simulation and experimental facilities)

- **Meanwhile, the Japanese “stay the course” with vectors**
  - Gov’t funding pumped up their market
  - Some concern that US had forever lost it’s edge when the Japanese Earth Simulator was deployed in 2002.
  - Many here deemed it a Pyrrhic victory
    - It had a HUGE footprint and power use

- **BlueGene/L (@ LLNL) retakes the crown in 2004 for 3.5 straight years, putting a nail in the coffin of the Japanese dominance**

- **DOE Leadership Computing Facilities (ANL/ORNL) launched in ~2004 to help reestablish US lead in open scientific computing**
  - Currently housing Summit (#1 @ ORNL), Mira (#17 @ ANL)
  - Argonne scheduled to receive first exascale computer in 2021-22 (A21)
Programming in the Distributed Computing Era

- **Going on 20+ years of stability in programming model**
  - PVM, and then ultimately MPI, provide performance portability
  - Unix becomes the de-facto standard for HPC
  - Object-oriented design helps balance complexity and performance
  - Programming Languages and compilers continue to improve
  - Emergence of grid and distance computing

- **Mission-delivery and scientific advancement has advanced rapidly, partly due to this stability**
  - Seldom must one spend majority of time porting or rewriting

- **Some feel disenfranchised with the transition from vector**
  - Reaching anything near peak performance on RISC architectures is even more difficult than on vector machines
  - This is expected, and in fact defines the boundaries between “eras”
Each era represents a radical shift in hardware technology to obtain greater performance

- **Mainframe->Vector: FLOPs at any cost**
  - Vector hardware could simultaneously perform the same operation across all elements of an array
  - Radical change in programming model

- **Vector->Distributed: Large scale, low cost**
  - Calculation partitioned among many independent, relatively simple, compute nodes
  - The development of high performance interconnects became crucial
  - Inter-node communication via message passing (MPI)

- **Distributed->Heterogeneous: Power Efficiency**
  - Still distributed, but *much* more on-node parallelism
  - “MPI+X”, where ‘X’ is threading (and vectorization)
Data motion and memory capacity are becoming the limiting factors in high performance computing.

EXASCALE:

100x MORE FLOPS

with only

5-8x MORE BANDWIDTH

and

0.1x MEMORY/CORE

MEMORY IS POWER-HUNGRY

NUMBER AND BANDWIDTH OF PINS IS LIMITED
Power has become the dominant constraint

Based on current technology, scaling today’s systems to an exaflop level would consume more than a gigawatt of power, roughly the output of Hoover Dam

Processor trends tell the parallelism story....

42 Years of Microprocessor Trend Data

- Moore’s Law – Alive and well!
- Barely hanging on (dynamic clock management, IPC tricks)
- Flat (or even slightly down)
- Mostly flat
- Exponential growth?

Transistors (thousands)
Single-Thread Performance (SpecINT x 10^3)
Frequency (MHz)
Typical Power (Watts)
Number of Logical Cores

Year


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp
Why? It’s Physics

- **Dennard scaling (aka MOSFET scaling)**
  - As transistors shrink (Moore’s Law) -> Decrease Voltage -> Constant Energy-Density
  - Clock rates increase to boost single-thread performance

- **End of Dennard c. ~2003-2005**
  - Can’t reduce voltage further w/o excessive leakage (heat)
  - Increasing power densities on chips
  - Energy per operation no longer dropping
  - Clock rates stagnate (or at least peak)

- **How to keep riding the Moore’s Law wave?**

  Parallelism
Node architectures are gaining many cores and deep memory hierarchies

Based on slide from J. Shalf
The drive for more capable high-end computing is driving disruption in HPC application development

The trickle down...

Assumption: Continued increased computational power is desired

External Constraints
- Power
  - Unreasonable operating costs
  - Today’s tech = $100M/year – exceeds capital investments

Hardware Designs
- Chip / processor efficiency
  - GPUs / accelerators
  - Simpler cores
  - Unreliability at near-threshold voltages – compounded by scale of systems

Software Supports
- On-node "inscaling" challenges
  - Complex hardware
  - Massive on-node concurrency requirements
  - New programming and memory models

Applications React
- Disruptive Changes
  - Code evolution
  - Algorithm rewrites
  - Platform uncertainty

Meanwhile, Data Analytics are driving industry investments: Social Networking, Machine/Deep Learning, Cloud Computing,…

Software innovation in data-centric HPC is thriving, but traditional HPC is facing strong headwinds if it can’t adapt
Exascale computing introduces several fundamental challenges

**Extreme Concurrency**
- Processing units \(\uparrow\)
- Bulk-synchronous will not scale
- Concurrency \(\uparrow\)
- Synchronization \(\downarrow\)
- Communication \(\downarrow\)
- Dynamic task parallelism

**Limited Memory**
- Memory gains less than processing
- Memory/core \(\downarrow\)
- Minimize memory usage
- Deeper, heterogeneous memory hierarchies

**Data Locality**
- Transfer gains less than processing
- Bandwidth/core \(\downarrow\)
- Energy and time penalties for data motion
- Greater need for data locality
- Reduce data transfers

**Resilience**
- Massive number of components: hard faults \(\uparrow\)
- Running closer to threshold voltage: soft faults \(\uparrow\)
- Bulk-synchronous checkpoint restart is dead
Alternative Emerging Programming Models for Exascale

- **PGAS (Partitioned Global Address Space)**
  - Gives the appearance of a global address space on top of scalable distributed memory
    - Communication done through array reads and writes
    - Programmer control over what’s local vs remote
  - UPC, Co-array Fortran (CAF), Titanium, Chapel

- **Task-based / Event-driven / Async Task Model (ATM)**
  - Define your problem as independent tasks
  - A run-time system manages scheduling of work
  - Good for load imbalanced problems and hiding communication latency
  - Legion, Charm++, HPX, OCR, ...

Both of these models can demand fine-grained messaging on the interconnect (i.e., lots of small messages)
An organizing principle for numerical algorithms is the **Mathematics Stack**

- **Problem Formulation**
- **Mathematical Modeling**
- **Model Discretization**
- **Numerical Solvers**
- **Data Analysis**
- **Robustness and Correctness**

**Questions to be answered?**
- Relevant processes & scales?
- Single forward simulation?
- Inverse problem?
- Optimization? UQ?

**Expressing the problem mathematically**
- Appropriate multiscale and multiphysics models?
- Coupling between models?
- UQ formulation?
- Optimization formulation?

**Expressing mathematical model discretely**
- Discretization in space / time
- Splitting between operators and spatial domains

**Solving the discrete system**
- Eigensolvers
- Linear and nonlinear solvers
- ODE integrators

**Understanding the results**
- Data and dimension reduction
- Automated analysis
- Integration of models, experiments, observations & simulations

**Trusting the results**
- Robustness to errors
- Faults
- Round-off error
- Discretization error
- Iteration error

Decisions in any one can significantly impact the others
Problem Formulation: A dramatic potential to change the questions we ask

- Forward Analysis
- Accurate & Efficient Forward Analysis
- Robust Analysis with Parameter Sensitivities
- Optimization of Design/System
- Quantify Uncertainties/Systems Margins
- Optimization under Uncertainty
- Systems of Systems

Mathematical Modeling: In forward simulation, we must consider new models

- Can we model additional physics?
- How else can we model the problem?
- Do some models expose more concurrency?
- Scale-bridging models
  - Hierarchical representations
  - Coarse-graining
- Particle vs. continuum

We must respect the physics!
Mathematical Modeling: Exascale will enable the solution of new optimization problems

- Concurrent-point methods
- Mixed-integer, simulation-based, and global optimization
- Multi-fidelity hierarchies
- Robust optimization and optimization under UQ
- Optimal design and coupling of experiments

Branch and Bound Tree for MIPDECO

- MIPDECOs generate huge search trees
- Each node is PDE-constrained optimization

[Branch and Bound Tree for MIPDECO] [Leyffer & Mahajan]
Mathematical Modeling: Uncertainty quantification plays a larger role at exascale

- Adaptive hierarchical methods
- Advanced multilevel methods
  - Model hierarchies
  - Stochastic hierarchies
- Architecture-aware UQ
- Adaptive and robust methods for fusing computation and experimental data

We must be clever in combating the curse of dimensionality
Discretization: High-order, partitioning, and adaptivity will play important roles

- **High-order discretizations**
  - High arithmetic intensity
  - Maximize on-node performance
  - Robustness? BCs?

- **Partitioned algorithms**
  - Models, equations, and operators
  - Spatial (FSI)
  - Temporal (multimethod)

- **Need better coupling strategies**
  - High-order
  - Splittings based on strength of coupling
  - Compatible interface treatments
  - Nonlinearly converged strategies

- **Adaptivity in mesh, model, discretization and order**

- **Scalable computational geometry and mesh generation**
Performance effects of order in CFD: Helmholtz solve in spectral element code for incompressible Navier-Stokes

For all element sizes, LIBXSMM offer the best performance.

- for order <= 16, the difference is small because the computation are memory bandwidth bound.
- for order <= 16, a boost is possible with the non-temporal stores (101.6 GiB/s).
- for order > 16, LIBXSMM ~ 2x is faster then Nek's `mxm_std` and up to 40% faster than Intel MKL.

c/o Hutchinson et al. (2016) ISC’16
Discretization / Solvers: Overcome sequential bottleneck of time integration

- **Parallel-in-time**
  - More concurrency, not faster clock speeds
  - Hierarchy of representations of varying fidelity
  - Iterative time advancement
  - Useful beyond some scale
  - Long-time integrations = more potential for large speed-ups

- **Research issues:**
  - Optimal convergence
  - Chaotic systems
  - Oscillatory systems
  - Hyperbolic systems

---

J. Schroder et al., XBRAID project

Heat equation, $257^2 \times 16,384$ space-time grid

Run-time (min) vs. Total Core Count

- Sequential time stepping
- Parallel only in time
- Space-time parallel

J. Schroder et al., XBRAID project
PIT has been demonstrated on real problems

Strand2D
- Vortex shedding, Re=100
- 7.5x speed-up @ 4k cores
- Cross-over @ 80 cores
- ~600 lines of code to couple
- ~3 weeks of effort

Cart3D
- Taylor-Green problem, Re=1600
- Promising rapid initial convergence
- Unexplained stalls in convergence

Helios
- Ultimate goal = 100x speed-up
- Long simulation times
- Periodic hovering = days to a week
- Non-periodic maneuvering = weeks

DoD Collaborators: Wissink, Sitaraman, Leffel, Atwood
Scalable Solvers: In solving the discrete system, numerous topics must be addressed

- Communication-avoiding
- Synchronization reduction
- Data compression
- Mixed-precision
- Randomization and sampling
- Adaptive load balancing
- Scheduling and memory management
- Autotuning algorithms
- Energy-efficient algorithms

Example: Timings on $100^3$ 7-point Laplacian stencil [E. Chow and A. Patel]

$$l_{ij} = u_{jj}^{-1} a_{ij} - u_{jj}^{-1} \sum_{k=1}^{j-1} l_{ik} u_{kj} \quad i > j$$

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \quad i \leq j$$
Many of the bits are error

11 bit exponent: 616 orders of magnitude

This is wasteful!
  — Use more work, power, or time than necessary
  — Move around lots of meaningless bits

Eliminate the bottlenecks: use only as many bits as needed
Can we make use of compression beyond I/O?

- **Address memory bandwidth limit while computing**
  - Store data in memory in compressed format
  - Decompress before computing
  - Recompress after computing

- **Ideally, handle compression/decompression in hardware**

- **How does this modify the simulation result?**
  - Compression errors can accumulate
  - Could effect accuracy and stability of algorithms
In lab codes, we have shown that 4x inline lossy compression reproduces results with little error

**LULESH**: Lagrangian shock hydrodynamics
- QoI: radial shock position
- 25 state variables compressed over 2,100 time steps
- At 4x compression, relative error < 0.06%

**pf3D**: Laser-plasma multi-physics
- QoI: backscattered laser energy
- At 4x compression, relative error < 0.1%

**Miranda**: High-order Eulerian hydrodynamics
- QoI: Rayleigh-Taylor mixing layer thickness
- 10,000 time steps
- At 4x compression, relative error < 0.2%

**MiFEM**: Cubic finite elements
- QoI: function approximation
- 6x compression with ZFP error < 0.7% relative to FEM error
We have derived theoretical bounds for error caused by inline compression

\[ \| D(C(\vec{x})) - \vec{x} \|_\infty \leq K_\beta \| \vec{x} \|_\infty \]

Decompression Operator

Compression Operator

Original Data

Error introduced through lossy compression and decompression is bounded in the max norm (pointwise)

\[ K_\beta := \mathcal{O}(\max\{2^{1-k}, 2^{1-\beta}\}) \]

Machine Precision

ZFP Fixed Precision (\(\beta\): bit-plane index)

Exponent Range: Sampled maximum and million from 1 million trials

\[ \text{Sampled maximum and million from 1 million trials} \]

\[ c_{\text{max}} - c_{\text{min}} = 14 \]
Assuming standard properties, the ZFP error bound can be used to bound inline compression error for iterative methods

- **Consider bounded advancement operators** \( ||Ak|| \leq M \)

  **Theorem:** \[ \| A(D(C(\vec{v}^t))) - A\vec{u}^t \|_\infty \leq M \sum_{j=0}^{t} K_{\beta_j} \| \vec{v}^j \|_\infty, \]

- **Example:** 1D Lax-Wendroff scheme with periodic boundary conditions \( (M \leq 2) \)

![Graphs showing the comparison between theoretical bounds, truncation error, and measured error with inline lossy ZFP.](image)
ZFP adaptive arrays improve accuracy in PDE solution over IEEE by 6 orders of magnitude using less storage
ARC prototype improves accuracy in Euler2D PDE solution over IEEE by 6 orders of magnitude using less storage

---

**Diagram Description:**

- **Y-Axis:** RMS error
- **X-Axis:** Time
- **Lines:**
  - IEEE (32-bit)
  - posit1 (32-bit)
  - zfp (32-bit)
  - arc (28-bit)

---

**Graph Analysis:**

- The graph illustrates the RMS error over time for different storage formats and compression methods.
- IEEE (32-bit) shows the highest error, followed by posit1 (32-bit) and zfp (32-bit).
- ARC (28-bit) demonstrates the lowest error, improving accuracy significantly compared to other methods.
Resilience and Correctness: Dynamic adaptation impairs determinism

- Reproducibility and verification techniques rely on determinism
- Can we justify cost of enforcing determinism?
- Should we interpret reproducibility and verification statistically?
- Analysis to understand the variability of deterministic algorithms

Sources of variability:

- Task-based scheduling
- Problem decomp
- Adaptive mesh
- Adaptive discretization
- Adaptive models
- Fault tolerance & recovery
Resilience and Correctness: Trusting the results in the presence of faults

- **Resilient programming models**
  - Skeptical
  - Relaxed bulk synchronous
  - Local failure, local recovery
  - Selective reliability

- **Algorithm-Based Fault Tolerance**
  - Protect from *silent data corruption*
  - Use properties of models and algorithms to detect (good) or be insensitive (better) to faults
  - Understanding how random faults alter solutions and/or convergence

![GMRES Convergence](image)

**GMRES Convergence**

**Deterministic Faulty SpMV's in Inner Solves**

- Data from M. Heroux, M. Hoemmen, K. Teranishi

What is the right approach for stochastic or chaotic models?
Exascale computing will allow us to compute in ways that are not feasible today

- It will result in significant scientific breakthroughs
- Transition poses numerous scientific and technological challenges
- Success will require close interdisciplinary collaboration
- Advances in applied mathematics will be essential
Many additional resources are available

**DOE Exascale Reports**

http://science.energy.gov/ascr/news-and-resources/program-documents

**DOE Grand Challenge Science Reports**

http://science.energy.gov/ascr/news-and-resources/workshops-and-conferences/grand-challenges