The Quantum Research Kernels

Anne Matasuura
anne.y.matsuura@intel.com
Director, Quantum Applications & Architecture
Intel Labs

Tim Mattson
timothy.g.mattson@intel.com
Senior Principal Engineer
Intel labs

Notices and Disclaimers

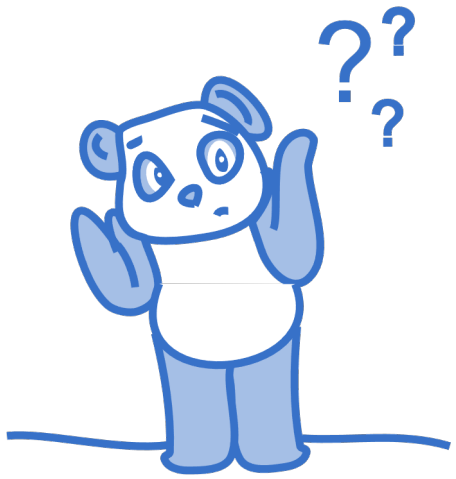
Forward-Looking Statements. Statements in this presentation that refer to business outlook, plans, and expectations are forward-looking statements that involve risks and uncertainties. Words such as "anticipate," "expect," "intend," "goal," "plans," "believe," "seek," "estimate," "continue," "committed," "on-track," "positioned," "ramp," "momentum," "roadmap," "path," "pipeline," "progress," "schedule," "forecast," "likely," "guide," "potential," "next gen," "future," "may," "will," "would," "should," "could," and variations of such words and similar expressions are intended to identify such forward-looking statements. Statements that refer to or are based on estimates, forecasts, projections, uncertain events or assumptions, including statements relating to Intel's strategy and its anticipated benefits; business plans; financial projections and expectations; total addressable market (TAM) and market opportunity; manufacturing expansion and investment plans; future manufacturing capacity; future products, technology, and services, and the expected availability and benefits of such products, technology, and services, including product and manufacturing plans, goals, timelines, ramps, progress, and future product and process leadership and performance; future economic conditions; future impacts of the COVID-19 pandemic; plans and goals related to Intel's foundry business; future legislation; future capital offsets; pending or future transactions; the proposed Mobileye IPO; supply expectations including regarding industry shortages; future external foundry usage; future use of EUV and other manufacturing technologies; expectations regarding customers, including designs, wins, orders, and partnerships; projections regarding competitors; ESG goals; and anticipated trends in our businesses or the markets relevant to them, including future demand, market share, industry growth, and technology trends, also identify forward-looking statements. Such statements involve many risks and uncertainties that could cause actual results to differ materially from those expressed or implied in these forward-looking statements. Important factors that could cause actual results to differ materially are set forth in Intel's earnings release dated January 26, 2022, which is included as an exhibit to Intel's Form 8-K furnished to the SEC on such date, and in Intel's SEC filings, including the company's most recent reports on Forms 10-K and 10-Q. Copies of Intel's SEC filings may be obtained by visiting our Investor Relations website at www.intc.com or the SEC's website at www.sec.gov. All information in this presentation reflects management's views as of April 20, unless an earlier date is indicated. Intel does not undertake, and expressly disclaims any duty, to update any statement made in this presentation, whether as a result of new information, new developments or otherwise, except to the extent that disclosure may be required by law.

Intel technologies may require enabled hardware, software or service activation. No product or component can be absolutely secure. Your costs and results may vary. Product and process performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex and www.Intel.com/ProcessInnovation. Future product and process performance and other metrics are projections and are inherently uncertain.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

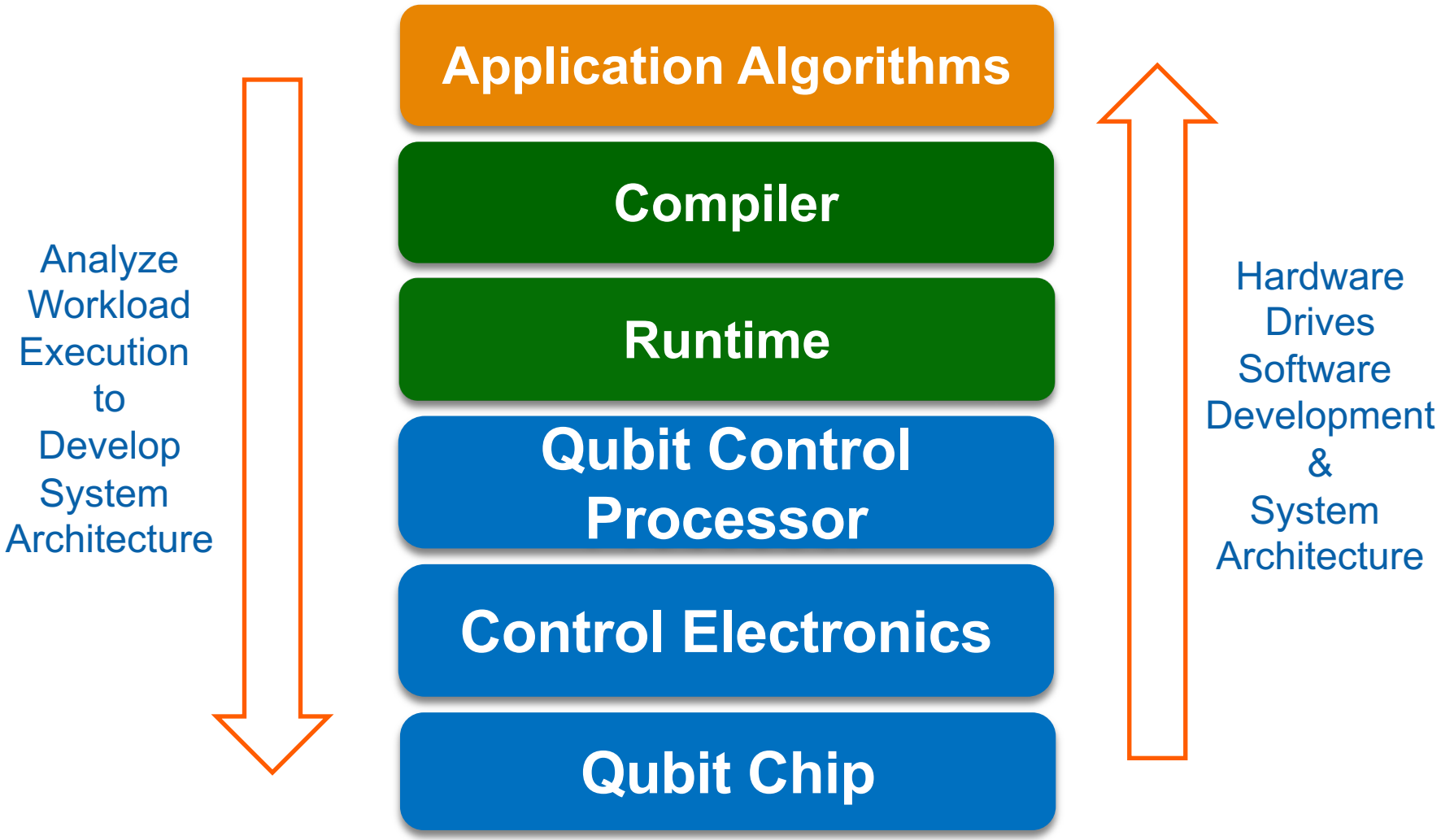
© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

How can classical compute help us learn to build a quantum machine?



BFI Institute CC By-SA 4.0

Hardware/Software Co-Design leads to a unified system



Hardware/Software co-design

- The best way to assure that computers serve the needs of target users is to design systems around key applications … we call this **Hardware/Software co-design**.

- It is very rarely done … not out of laziness, but because it is VERY hard to do well:
  - A complex system can take 3 to 5 years to design and build. We don't often know which applications will drive the target market in 3 to 5 years.
  - Real Applications are large complex software systems. Porting them to a new system and analyzing their performance is a huge job (many months of full-time work per application).
  - In early phases of a design process, key system components do not exist. Sometimes you only have "paper-and-pencil" designs to work with.

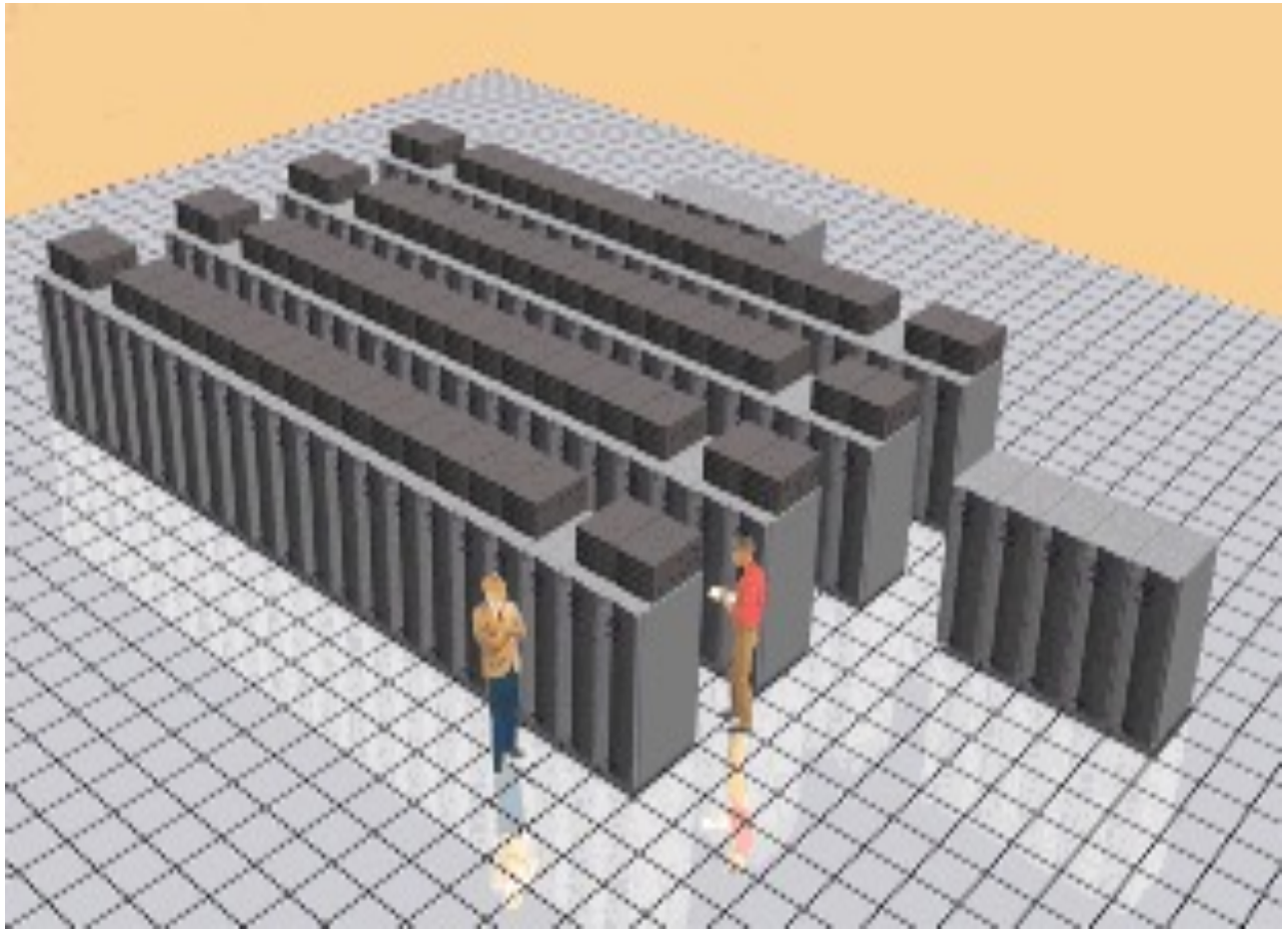
HW/SW co-design and the first TFLOP computer

My first HW/SW co-design project

In ~1995, Intel won a contract with the DOE* to build what was to be the world's first TeraFLOP computer (10^{12} floating point operations per sec).

We designed the system around key DOE workloads.

Intel's ASCI Red Supercomputer



The workloads were too complex to run on "paper-and-pencil" designs so we decomposed key workloads into tiny kernels that represented key bottlenecks that would limit the system.

For example, scalable molecular dynamics is limited by **vector reductions**.

Seismic depth migration limited by **transpose** (for the FFTs) and **synchronous nearest neighbor communication**

*DOE: Department of energy

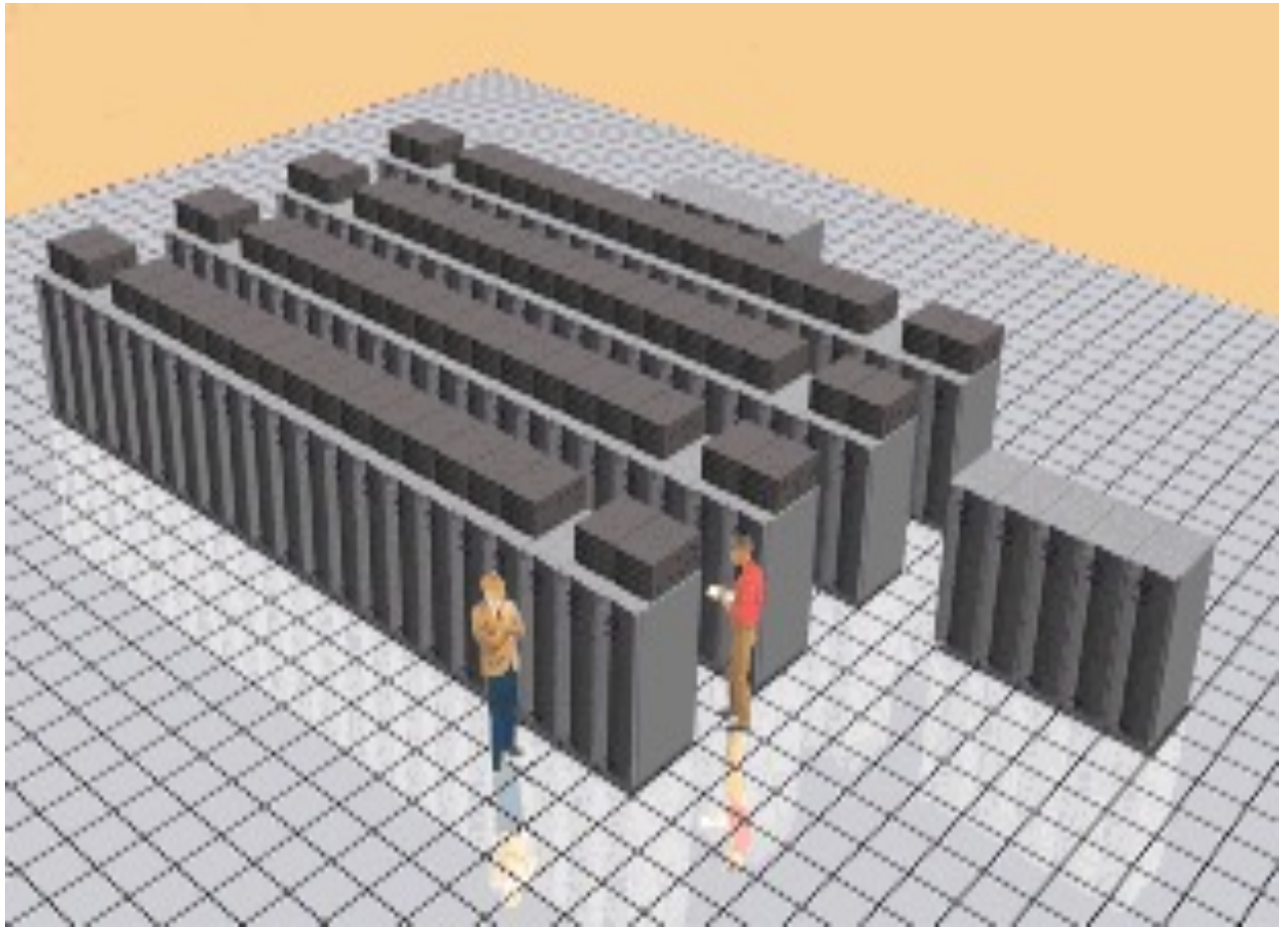
HW/SW co-design and the first TFLOP computer

The system used the new Intel® Pentium® Pro processor (P6)

December 21, 1995, we found buffering problems with the cache coherency protocol that limited bandwidth to memory.

It put the whole project at risk

Intel's ASCI Red Supercomputer

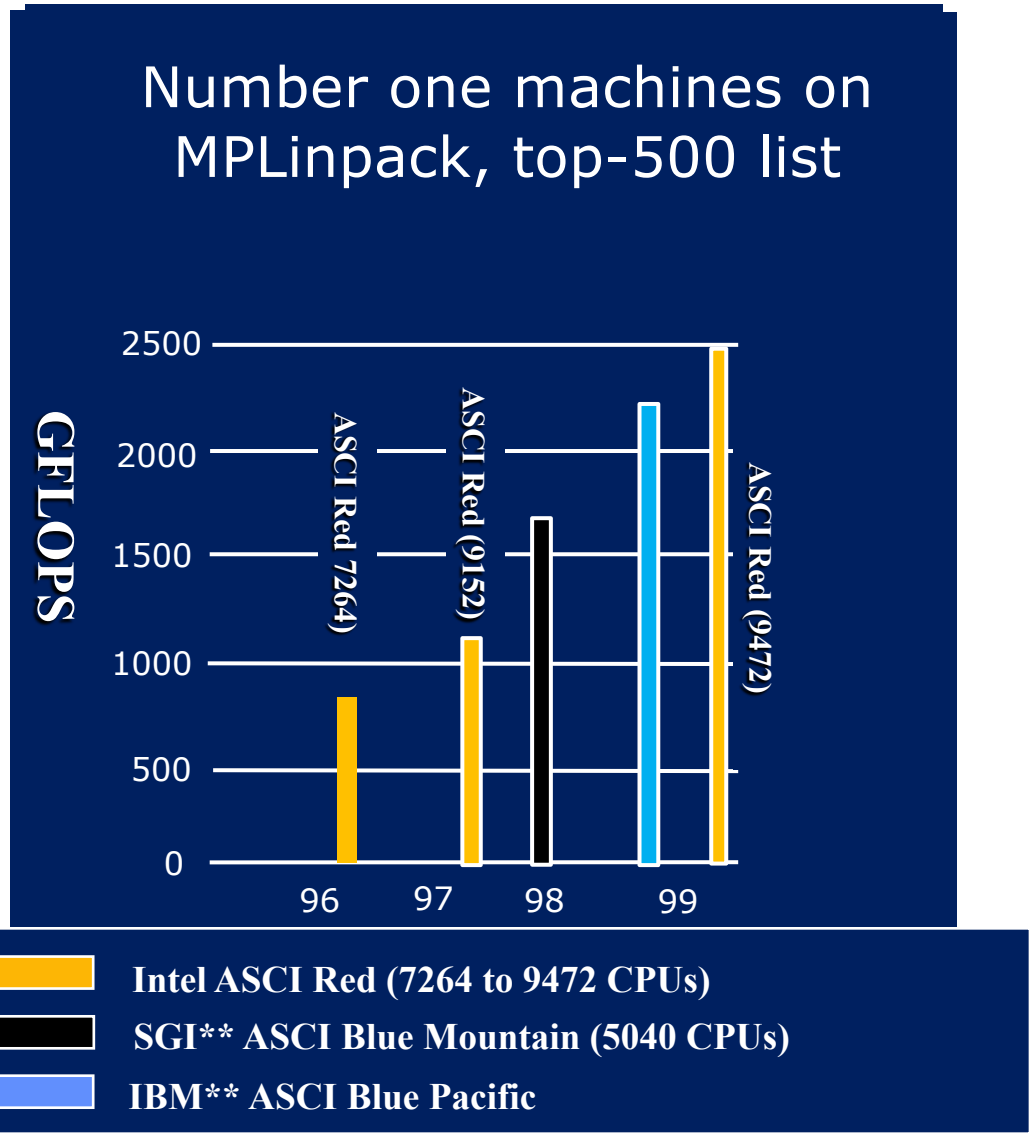


Using our low level "research kernels", however, we were able to show that the effects would be observable but they wouldn't prevent the machine from meeting contractual deliverables.

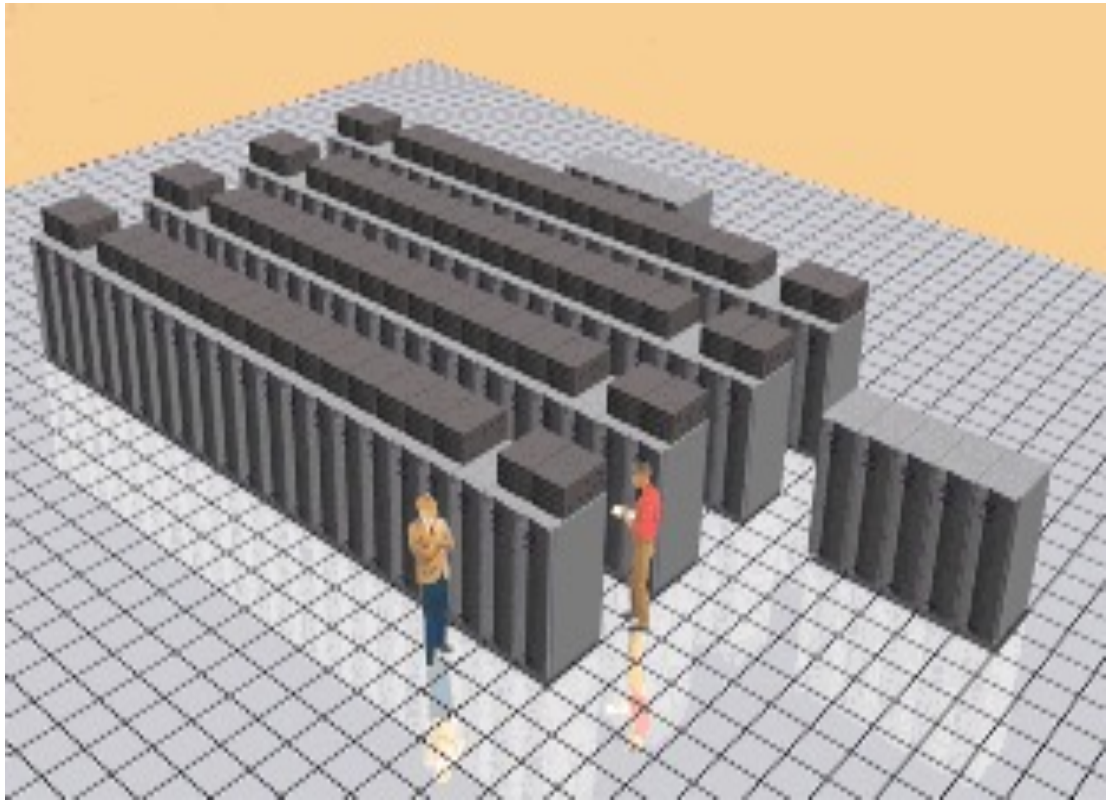
Research Kernels as HW probes saved the day!!!!

*DOE: Department of energy

… And it worked!!! Our machine was a record setter



Intel's ASCI* Red Supercomputer (1996)



~9000 CPUs

One megawatt of electricity.

1600 square feet of floor space.

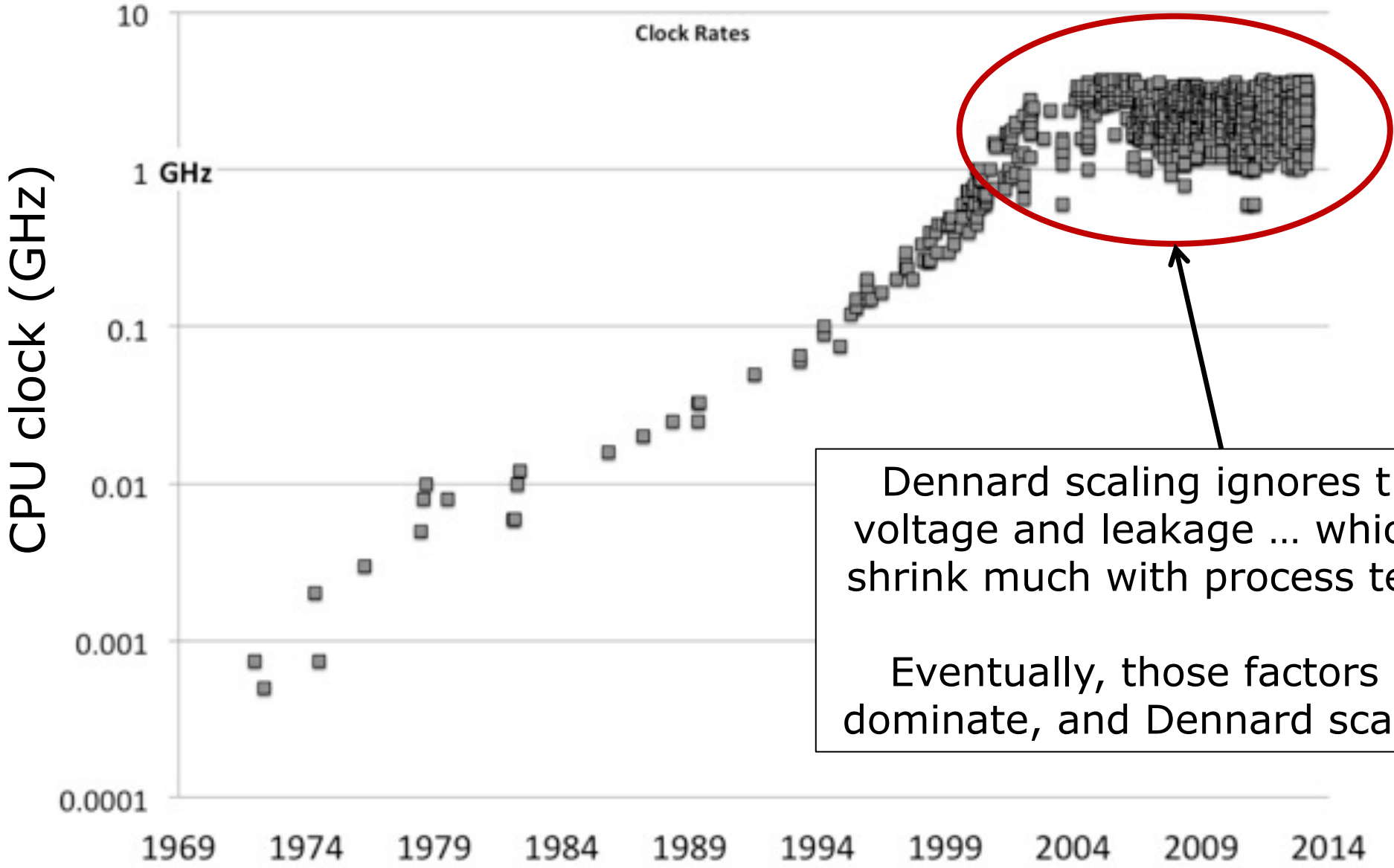
*Data from the Linpack Report, CS-89-85, April 11, 1999

**Other brands and names are the property of their respective owners.

*ASCI: Accelerated Strategic Computing Initiative

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/procs/perf/limits.htm <http://www.intel.com/procs/perf/limits.htm> or call (U.S.) 1-800-628-8686 or 1-916-356-3104.

Leap Forward 10 years … Birth of the multicore era



Dennard scaling ignores threshold voltage and leakage … which do NOT shrink much with process technology.

Eventually, those factors came to dominate, and Dennard scaling ended

Source: James Reinders (from the book "structured parallel programming")

HW/SW co-design: Scalable computing in the multicore era

- **Hardware designers needed input from the application community.**
  - We didn't know which multi-core applications would drive the market.
  - We didn't have full applications … outside of HPC there weren't scalable versions of many apps people cared about.
  - Applications took too long to port and analyze.
  - We needed to cover clusters of multicore nodes too … distributed and multiprocessor computing in one platform.

Hypothesis: We may not know what the key future applications will be, but we do know the features of a system that will limit their performance.

The Parallel Research Kernels (PRK)

PRK: Low level constructs that capture the essence of what parallel programmers require from parallel computers.

- **The PRK**
  - A set of low level “research kernels” to guide the design of future systems.
  - Output from full apps analysis overwhelming, hard to digest. The PRK can even be understood by HW engineers.
  - They are simple, small and easy to run … support execution on simulators or even “ paper and pencil” machines.

- **Selection Methodology**
  - Anecdotal analysis by a panel of experienced parallel application programmers at Intel.
  - Great Care was taken to make sure their experience covered the full range of typical HPC applications.

PRK Goals

- A minimal set that address the issues exposed by the parallel patterns encountered in real applications
- Provide generic reference implementations: well-written, but not tuned to a particular architecture
- Make sure each kernel does some real work*.
  - Corollary: performance metric = work/time
- Keep implementations compact (O(2-3 pages)): easy porting to new runtimes/languages
- Small, “easy” to port, self-testing, data generators, scalable

https://github.com/ParRes/Kernels

**spec only describes work, not how to do it. E.g. no back-to-back barriers without intervening work, or no empty regions protected by a lock*

Parallel Research Kernels

Name	Definition	Exposed System Features
Transpose	Transpose a dense matrix	Bisection Bandwidth
reduction	Sum across vectors	Message passing and local memory bandwidths
Sparse	Sparse-matrix vector product	Scatter/Gather operations
Random	Random updates to a table	Memory bandwidth for random updates
Synch_global	Global Synchronization	Collective synchronization (barrier)
Synch_p2p	Wavefront stencil over a “pencil”	Message passing latencies and remote atomics
Stencil	Finite Difference method stencil	Nearest neighbor, asynchronous communication
refcount	Update shared counters	Mutual exclusion locks
Nstream*	Daxpy over large vectors (stream triad)	Peak memory bandwidth
DGEMM	Dense matrix product	Peak floating-point operations rate
Branch*	Inner loop wirh lots of branches	Instruction cache misses and branch prediction
PIC#	Particle in Cell	Unstructured asynchronous multitasking
AMR#	Adaptive mesh refinement	Hierarchical synchronous multitasking

**embarrassingly parallel*

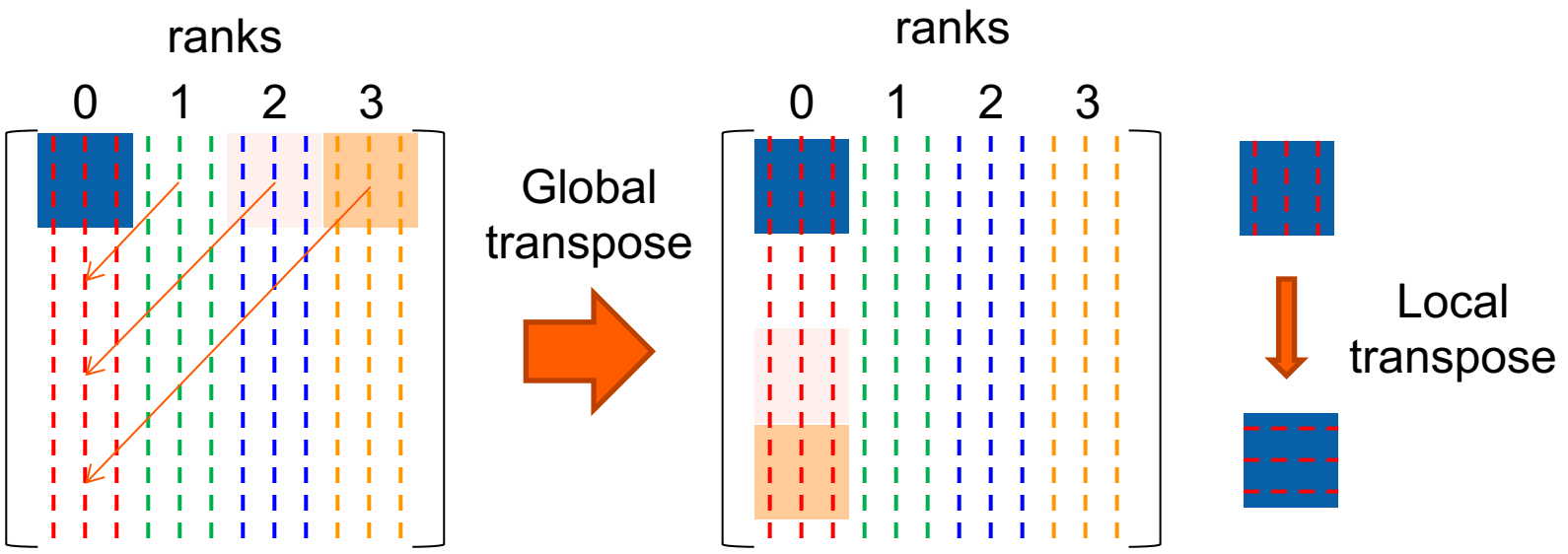
#*Newer kernels (after 2015), not heavily used*

Dense matrix Transpose

What: $A = B^T$, both A and B distributed identically, and by whole columns, using column-major storage format

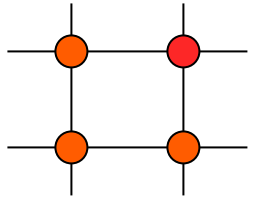
Why: Stride 1 on read, large stride on write (TLB, cache misses). All to all communication. Common operation in HPC (FFTs, numerical linear algebra)

How: Implemented in MPI, MPI+OpenMP, MPI+MPI3-shared memory, Charm++ (tiled local transpose to reduce misses)



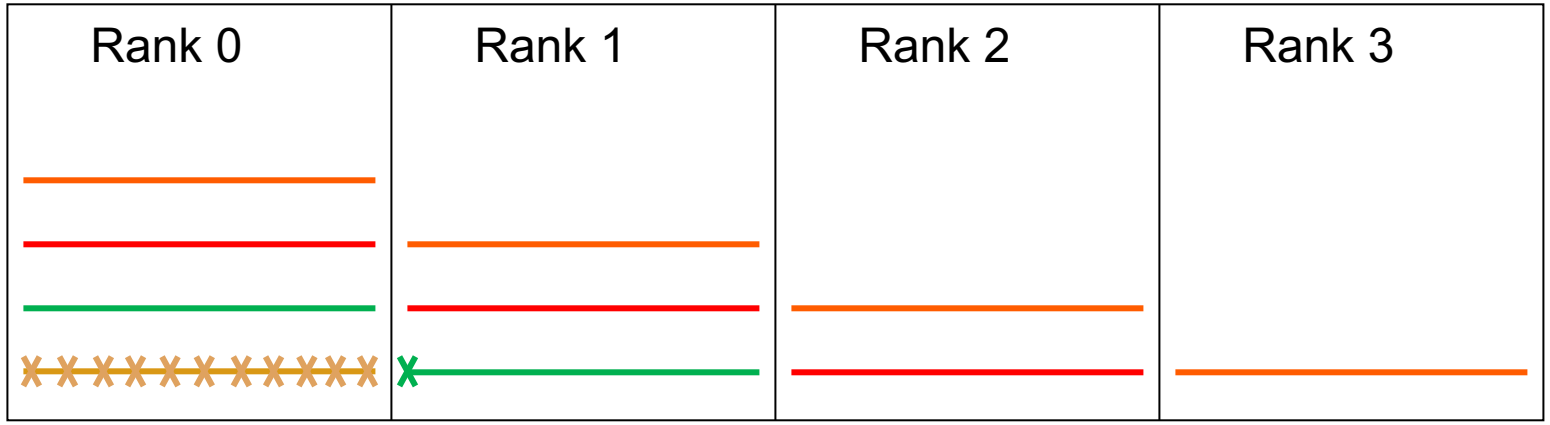
Point to point synchronization (Synch_p2p)

What: A(i,j) = A(i-1,j) + A(i,j-1) – A(i-1,j-1)
Pipelined solution of problem with non-trivial 2-way dependencies



Why: Threads/ranks forced to synchronize pairwise frequently during each iteration.
Stresses fine grain communication/synchronization.

How: Implemented in MPI, MPI+OpenMP, MPI+MPI3-shared memory, Charm++, Grappa

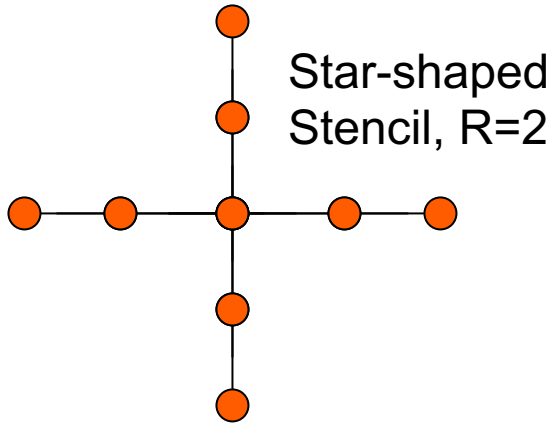


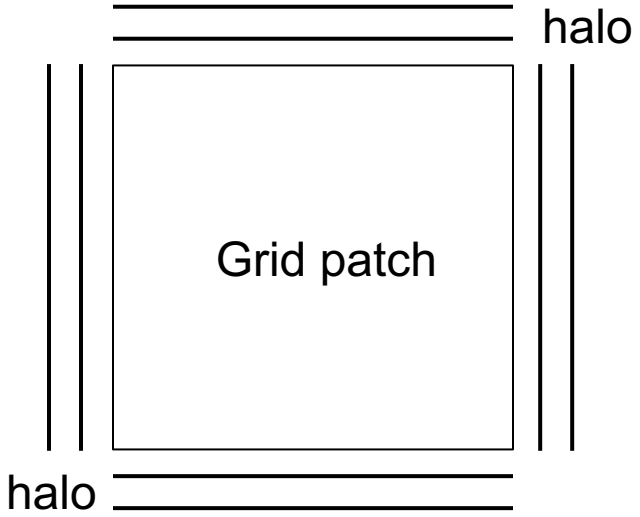
Stencil computation

What: For all points in 2D grid, compute a = S(b), S is star-shaped stencil operator (radius R), a and b are scalar grid variables (2D arrays)

Why: Data parallel; point-to-point bulk communication; common operation in HPC

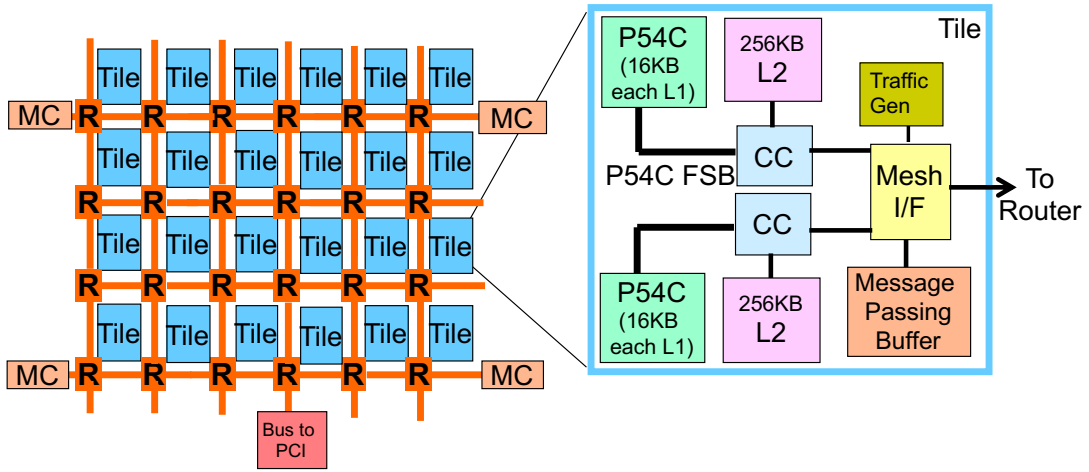
How: Implemented in MPI, OpenMP, MPI+MPI3-SHM, MPI+OpenMP, Charm++, Grappa; use 2D decomposition, collect halo values at start of each iteration





Use of the Parallel Research Kernels

- We used them to design specific many core chips
- We used them to analyze different on-chip network designs



SCC 48 core research chip

- We used them to evaluate designs of future exascale systems.
- Researchers at DOE supercomputer centers found them valuable and asked us to open source them so they could be used by Intel's competitors

https://github.com/ParRes/Kernels

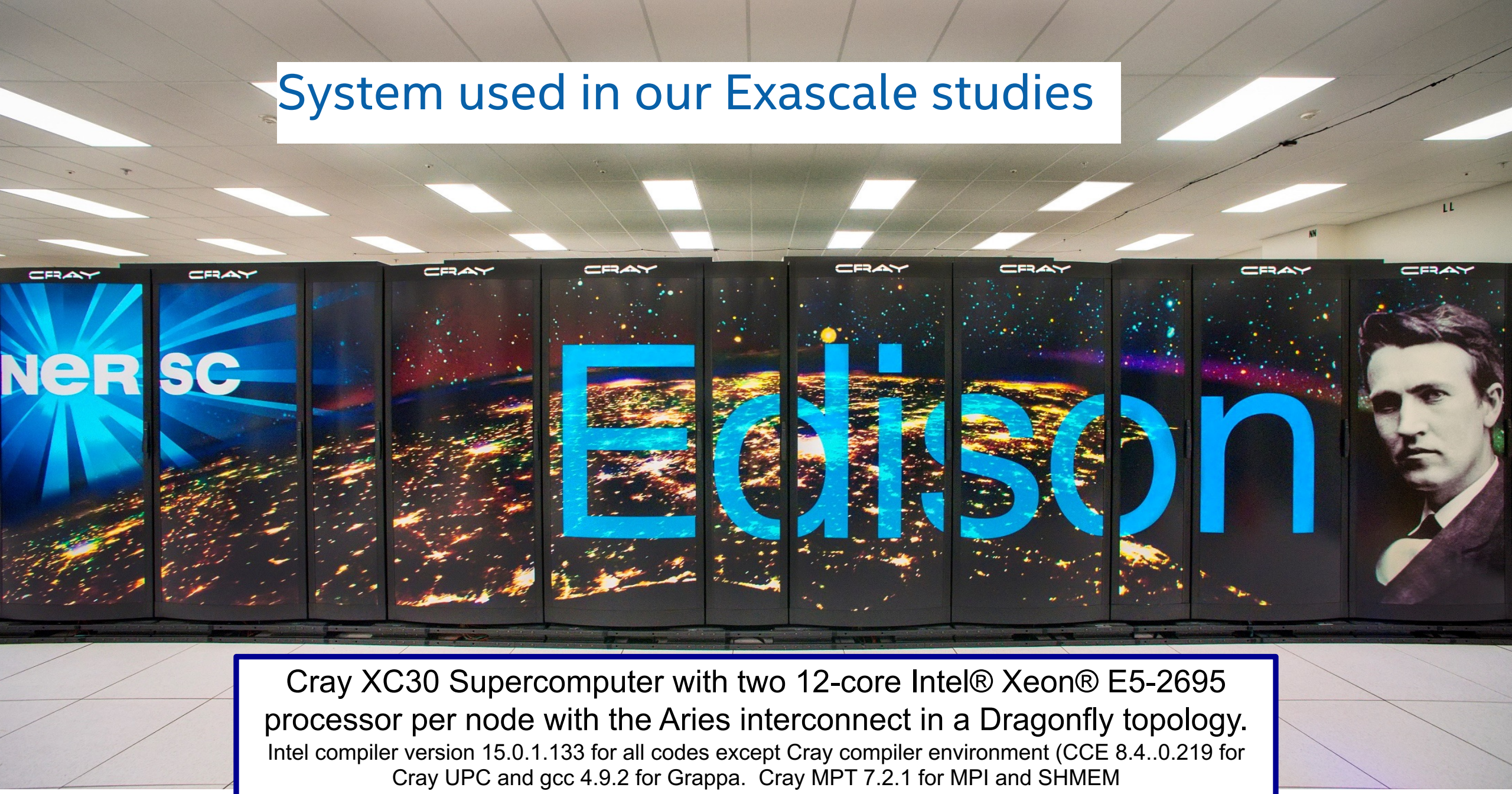
Since 2015, we've used them to study exascale programming models

- I am not aware of ANY collection of programs implemented in more highly scalable programming models than the PRK
  - MPI-1, MPI-3, SHMEM, OpenMP, CHARM++, Legion, OCR, Grappa, UPC, MPI-shared memory, AMPI, C++11, OpenMP tasks, Modern Fortran including co-arrays, Python, Julia, Rust, MPI+RMA, MPI+OpenMP, SHMEM, OpenCL, CUDA, Python/Numba, DASK, Kokos, Raja, Sycl, and many more.

- An ongoing project (we are always looking for collaborators) driven mostly by Jeff Hammond of Nvidia.

https://github.com/ParRes/Kernels

System used in our Exascale studies

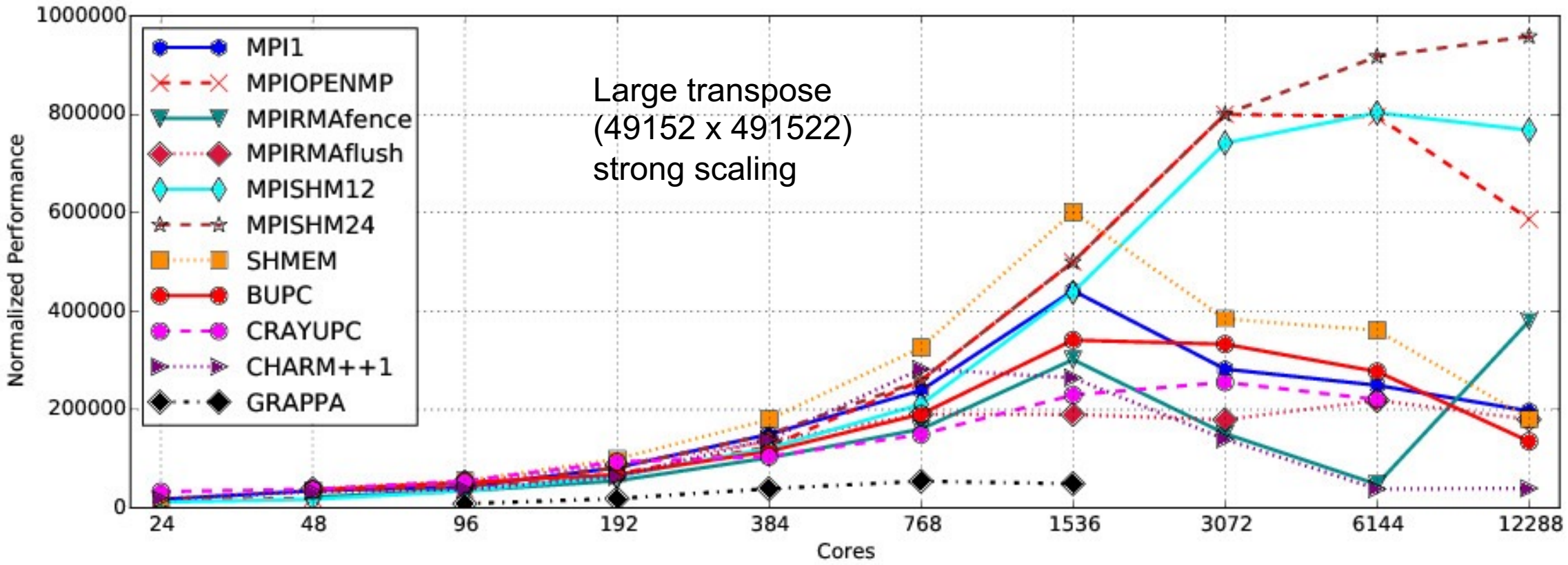


Cray XC30 Supercomputer with two 12-core Intel® Xeon® E5-2695 processor per node with the Aries interconnect in a Dragonfly topology.

Intel compiler version 15.0.1.133 for all codes except Cray compiler environment (CCE 8.4..0.219 for Cray UPC and gcc 4.9.2 for Grappa. Cray MPT 7.2.1 for MPI and SHMEM

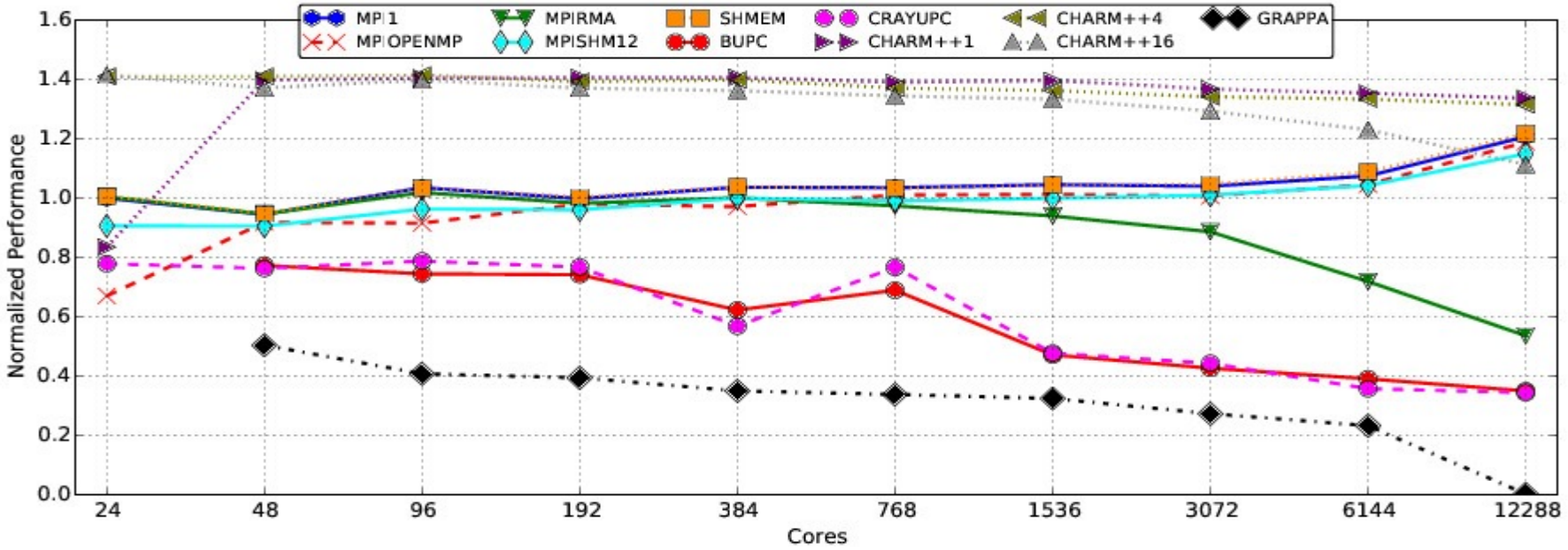
Third Party names are the property of their owners.

Transpose Parallel Research Kernel (PRK)



- MPI-OpenMP, MPI-SHMEM do very well since this coarse grained kernel is ideally suited to their SPMD pattern
- Grappa generates large numbers of small messages as the core-count increases .. .hence its performance is poor.

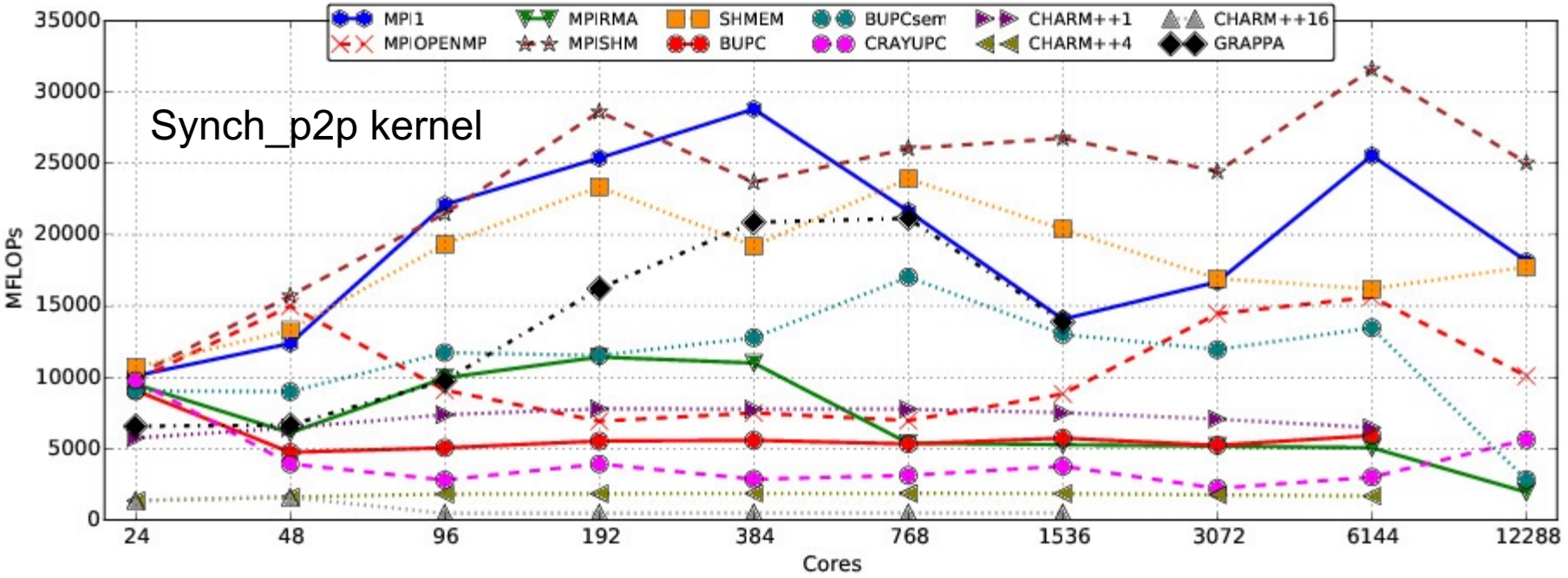
Stencil Parallel Research Kernel (PRK)



Stencil kernel: Normalized MFLOPS to MPI (2-sided) and number of nodes

- Runtimes that depend on barriers (MPIRMA, BUPC, CrayUPC) do not do well.
- Runtimes that use pair wise sync (Charm++, Grapa, SHMEM, MPI1) scale well.
- Charm++ handles this example particularly well (numbers 1, 4, and 16 are the degrees of over decomposition)

Synch_p2p Parallel Research Kernel (PRK)



- Performance dominated by (1) frequent pair-wise synchronization and (2) global sync after each iteration.
- MPIRMA and both versions of UPC used barriers after each iteration … this really hurt their pefformance.
- MPI1, SHMEM, and MPISHM have very similar perf.
- This workload is poorly suited to the dynamic execution profile charm++ was designed for.

Can we use the PRK strategy for Quantum Computing?

- Quantum Research kernels:
  - Hypothesis: We may not know the applications that will drive the market for quantum computers, but the quantum computing applications community knows the features of a future Quantum Computer that will limit them.

- We could follow the path used with the PRK
  - Gather a cross section of the applications community to work out the features of a quantum computer that will constrain the applications for these machines
    1. Define formally (i.e. not as code):
    2. Includes work that does “something”
    3. Generate input so we can scale to any systems
    4. Test output
    5. Define quality metrics

An Outline for Three candidate QRK

- Encode:
  - Create a sequence of classical values:

    foreach k from k=0 to k=N: A[k] = equal to $4k\pi/N$
  - Encode qubits with the values from the previous step.
  - Rotate each qubit by $\pi/6$
  - Read qubits and confirm that they have the correct rotated value
- Computational Area:
  - Product of a number of qubits and the number of operations that can be carried out on those qubits before the entangled state can no longer be maintained
- Parallel Streams:
  - How many independent streams of operations can execute in parallel

Quantum computing is not as mature as scientific computing (where the PRK emerged).

Perhaps we need a more systematic approach.

If I were creating the PRK today from scratch, here is how I'd do it

Creating the PRK: An application driven approach

- The 7 ~~Dwarfs~~ Motifs of scientific computing (Phil Colella, 2004) … 7 distinct patterns of computation and communication that **ALL** scientific computing applications map onto

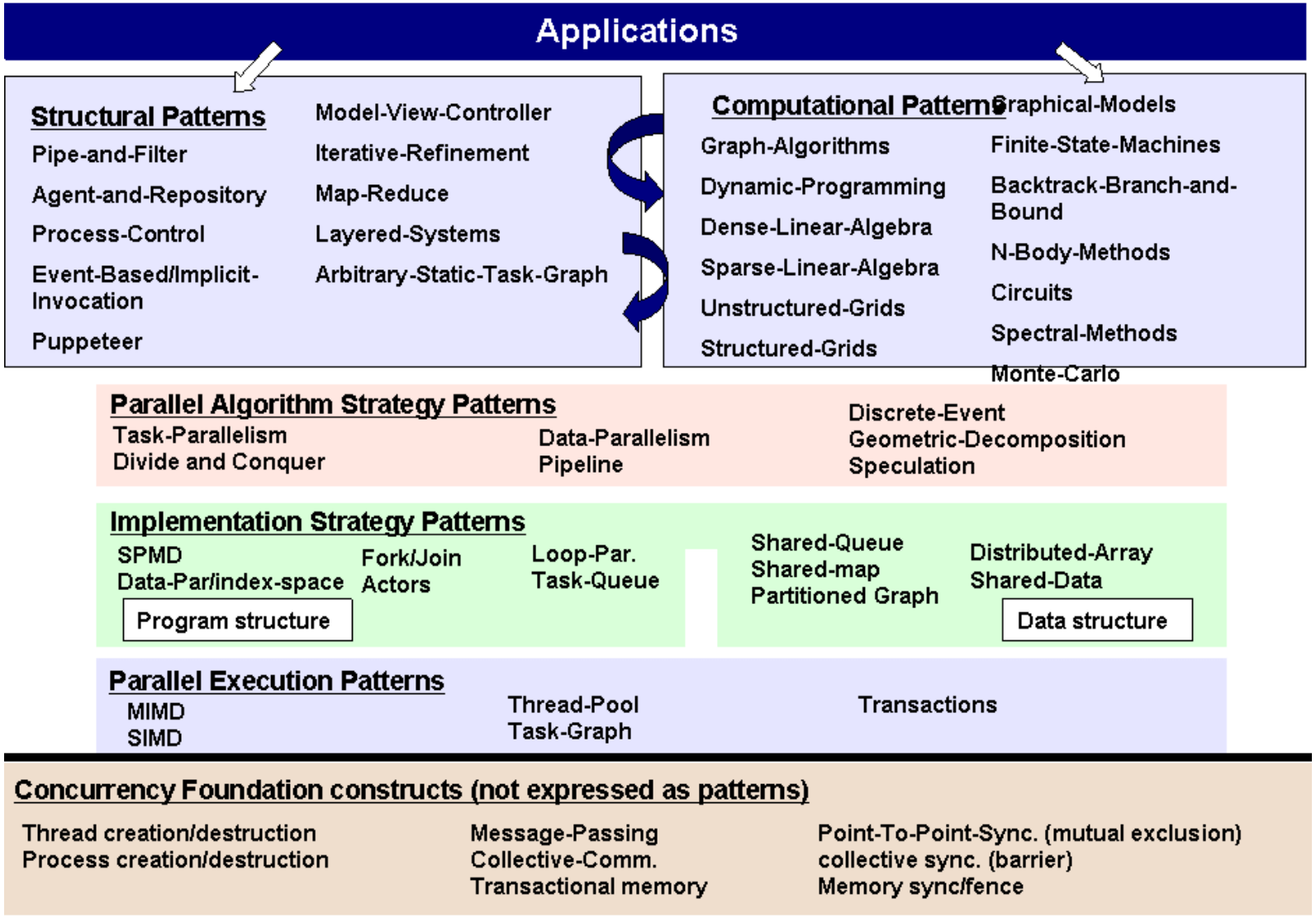
Motif	Patterns of Data, Communication, Synchronization
Dense Linear Algebra	Regular/predictable data distribution, synchronous
Sparse Linear Algebra	Irregular/unpredictable data distribution, loosely synchronous
Spectral Methods	All to All communication, Bulk synchronous pattern
N-body	Different methods typically based on Recursive splitting or all-reduce
Structured Grids	Regular/predictable data distribution, synchronous
Unstructured Grids	Irregular/unpredictable data distribution, loosely synchronous
Monte Carlo	Embarrassingly parallel (other than parallel random number generators)

Creating the PRK: Mapping PRK onto motifs

- The 7 Motifs of scientific computing (Phil Colella) … 7 distinct patterns of computation and communication that **ALL** scientific computing applications

Motif	Parallel Research Kernel
Dense Linear Algebra	DGEMM
Sparse Linear Algebra	Sparse(SpMV), Random
Spectral Methods	Transpose
N-body	Reduction (AllReduce)
Structured Grids	Stencil
Unstructured Grids	Sparse, Synch_P2P
Monte Carlo	Nstream, refcount, Synch_global

Eventually we went well beyond the motifs and crafted a Design Pattern Language to define the software architecture of parallel applications



Can we use this PRK strategy for Quantum Computing?

- What are the fundamental Motifs of quantum computing applications?
- Define a set of simple/small programs that cover the Motifs
- How do the Motifs map onto Design Patterns for Quantum Computing Algorithms?
- Which key design patterns address the software architecture of applications that will run on Quantum Computers?
  - These patterns can guide the design of ”programmer friendly” high level programming models for quantum computing.

A call to action

- We want the quantum applications community to come together and define a set of Quantum Research Kernels (QRK)
  - These would define for Quantum computing hardware developers what the applications community needs from future systems.
  - The QRK would be target kernels we could use to study high level programming models for quantum computing.

- Defining the QRK will only work if we get a ”complete” coverage of application classes … we need it to be the case that if hardware is designed that handles all the QRK well, then the resulting system will be an effective Quantum Computer for “general purpose” quantum applications.

- Challenges: We were able to do this for classical parallel computing since we had 25 years of parallel programming experience to draw on. Is it to early to define the QRK?

It is our hope that among the participants in this workshop, we can find a subset of applications people to work with us to get the first set of QRK defined and implemented (where everything would be Open Source with a non-viral license).

References:

- The Parallel Research Kernels: A tool for architecture and programming system investigation, Proceedings of the IEEE High Performance Extreme Computing Conference, 2014 Rob F. Van der Wijngaart and Timothy G. Mattson

- Comparing runtime systems with exascale ambitions using the Parallel Research Kernels, Proceedings of the International Supercomputing Conference, 2016. R. F. Van der Wijngaart, A. Kaya, J.R. Hammond, G. Jost, T. St. John,S. Sridharan, T.G. Mattson, J. Abercrombie, and J. Nelson

- Introducing the Quantum Research Kernels: Lessons from Classical Parallel Computing, A. Y. Matsuura, T. G. Mattson, https://arxiv.org/abs/2211.00844, 2023.

- Defining Software Requirements for Scientific Computing, P. Colella, presentation, 2004.

- The Landscape of Parallel Computing: The View from Berkeley, Asanovic, Rodic, Catanzaro, Gebis, Husbands, Keutzer, Patterson, Plishker, Shalf, Williams, and Yelik,2006, **http://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf**

- A design pattern language for engineering (parallel) software: merging the PLPP and OPL projects, PARAPLOP Proceedings, Keutzer, Massingill, Mattson, and Sanders, 2010