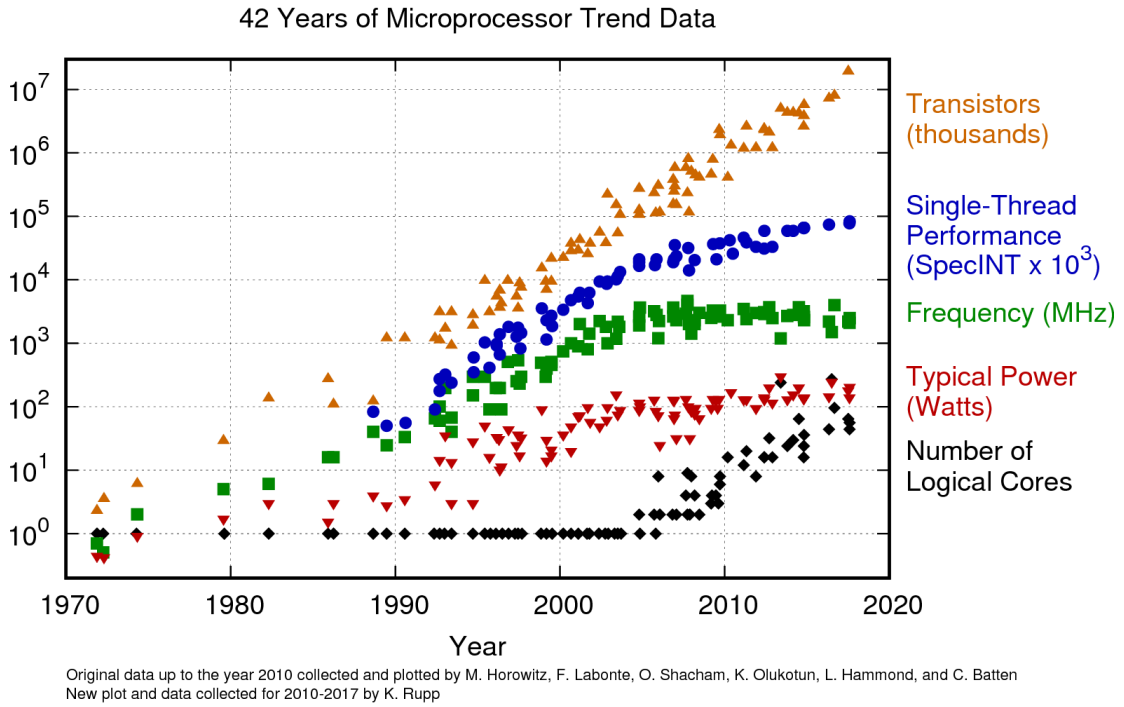
Using Performance Engineering to Navigate the Revolution in Computing

William Gropp
wgropp.cs.Illinois.edu

End of Easy Performance Gains

- Dennard (Frequency) Scaling ended around 2005-2006
  - Previously neglected effect, leakage current, became important
  - Clock speeds stagnated at 2.5-3.5 GHz
  - Almost 20 years later, clock speeds are nearly the same
- But the expectation of doubling of performance was strong
  - And Moore's "Law", though slowing down, is still providing higher component density
- Maintaining performance expectation required doing something different
  - Instruction parallelism (vectors)
  - Multiple cores per "processor"
  - Specialization, especially to vectors/streams/GPUs



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

The Rise of Specialization Wasn't Really a Surprise
CFD in 2030: NASA Study

Prediction in 2012 (published 2014)

Year	Feature size	Derived parallelism	Stream parallelsim	PIM Parallelism	Clock rate GHz	FMAs	GFLOPS (Scalar)	GFLOPS (Stream)	GFLOPS (PIM)	Processor per node	Processor (TFLOP)	nodes per system	Total (PFLOPS)
2012	22	16	512	0	2	2	128	1024	0	2	1	10000	23
2020	12	54	1721	0	2.8	4	1210	4819	0	2	6	20000	241
2023	8	122	3873	512	3.1	4	3026	12006	1587	4	17	20000	1330
2030	4	486	15489	1024	4	8	31104	61956	8192	16	101	20000	32401

Compare 2023 prediction to actual systems in 2022

- Not bad – if "processor" includes GPU, change to 8/node and 10k/system.
- PIM – stands in for HBM with operations. Maybe a miss (or not?)
  - Note that PIM still not a dominant part
- DP performance of an NVIDIA A100 is 9.7TF (19.5TF tensor core) – not too far from "Processor"
- Missed – Assumed tighter integration – "Stream" flops part of processor, not in a separate socket (GPU)
- *Enabling the environmentally clean air transportation of the future: a vision of computational fluid dynamics in 2030*, 10.1098/rsta.2013.0317

Achieving High Performance is Increasingly Difficult

- Systems are increasingly complex
  - It was bad enough with caches and vector instructions, now add HBM and GPUs – and not just 1 of each
  - Multi GPU common; more than one socket/node.
- Even effective use of a single CPU core (which means using appropriate vector and other instructions) is difficult
  - Compiler vectorization requires high levels of optimization and still misses optimization opportunities (45/151 in recent tests)
  - Best performance still requires specialized code, use of intrinsics, etc.
- Rapid innovation in architecture
  - Driven by the end of Dennard Scaling almost 20 years ago
  - Market demanding rapid change – 12 month cadence by NVIDIA

Hardware Implications For Programs

- Heterogeneity in many ways
  - Processor – complex compute modes with scalar and vector, prefetch, etc.
  - Many (but not all) include separate accelerators (GPUs and others)
  - Memory – Cache was bad enough; now HBM, other
  - I/O – Burst buffers (often violating POSIX semantics), on node, central, remote (cloud)
- For algorithm developer and programmer, the issue is *Performance Heterogeneity*
  - Whether the implementation uses more than one chip(let) isn't the issue – can you see performance impact of the different elements?
  - Even vectorization counts as performance heterogeneity in this view
    - Compilers still not great at vectorizing code, and often algorithmic changes needed to take full advantage of vectorization (which specializes code, makes it hard to reason about performance)
- Impacts algorithm choice and program realization

Programming issues

- Caveat: It is possible to write and run programs on heterogenous nodes (nodes of different types). But it is often difficult and few applications even consider it
- Some of the issues include
  - Load balancing (performance is just one consideration)
    - Common to assume uniform node (or even core) performance when designing algorithms and code
    - Process assignment (MPI_Cart_create or parmetis for mesh decomposition)
  - Runtime (startup, networking, data access)
  - Scheduling
    - Must allow access to different resource types; possibly dynamic, guidance to users in choices (is that a place for AI?)
    - Typically provides a choice of homogenous job resources but from a heterogeneous collection of resources
- Success metrics (time to solution, performance, total cost including energy, productivity)

Where are We Really

- Even simple heterogeneity hard
  - Limits to CPU vectorization (compilers miss opportunites) or GPU programming with OpenACC/OpenMP
- Multiple tools to manage tasks
  - OpenMP, PaRSEC, StarPU, …
  - Mapreduce an example of a specialized system
- Few examples of HPC applications or benchmarks that are heterogeneous in my sense
- I'm going to focus on one easy case – multicore/socket nodes

Gcc	Gcc14	CrayCC	Nvc
2.03	1.95	1.00	1.25
1.23	1.25	1.23	1.00
1.11	1.00	1.27	1.28
1.24	2.10	1.15	1.00
2.55	4.79	1.00	1.66
1.58	1.71	1.00	1.45
1.21	1.20	1.05	1.00
1.12	1.11	1.00	1.02
2.10	1.00	1.50	2.07
1.07	1.07	1.00	1.01
1.00	1.01	1.24	1.26
1.00	1.01	1.00	1.04
1.03	1.22	1.03	1.00
1.04	1.00	1.03	1.03
1.00	1.04	1.01	1.01
1.91	1.02	1.00	1.02
1.08	1.00	1.18	1.12
1.02	1.09	1.00	1.02
1.00	1.13	1.11	1.10
1.04	1.03	1.00	1.16
1.31	1.00	1.27	1.26
1.14	1.09	1.02	1.00
1.00	1.04	1.01	1.01
1.45	1.40	1.00	1.11
1.01	1.00	1.01	1.00
1.00	1.04	1.02	1.01
1.85	1.04	1.00	1.18
1.00	1.03	1.02	1.06
1.11	1.03	1.00	1.05
1.52	1.06	1.00	1.06
3.23	1.04	1.02	1.00
1.71	1.04	1.02	1.00
1.41	1.17	1.00	1.00
1.18	1.03	1.00	1.00
4.53	1.04	1.01	1.00
1.49	1.00	1.46	1.46
1.32	1.44	1.00	1.19
1.44	1.60	1.19	1.00
1.57	1.55	1.00	1.31
1.84	1.00	1.16	3.16
1.00	1.01	4.48	4.45
1.00	1.01	1.00	2.02
7.88	1.28	8.58	1.00
1.06	1.07	1.00	1.83

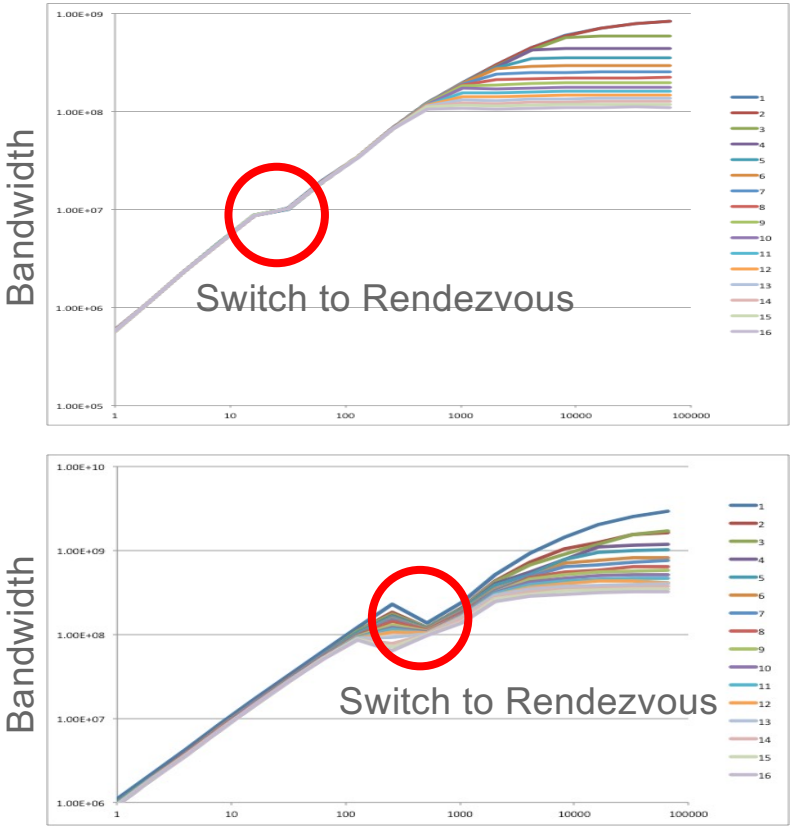
Algorithm Considerations

- Start with the choice of mathematical model/numerical method
  - E.g., higher-order approximations for finite difference/element/volume trade floating point operations, data motion, and data size
  - Higher level choices can provide better locality
    - E.g., nonlinear Schwarz, with “local” nonlinear solves
- Performance models needed to guide algorithm design/choice
  - Model does *not* need to be precise – just good enough to guide
  - This is fortunate, as highly accurate performance models are very difficult to create and validate
  - But they need to be accurate enough – and many models haven’t kept up with the evolution of architectures
- One Example: Node-aware algorithms
  - Performance model captures basic system hierarchy at node level
  - Avoid redundant data copies; optimize data motion for HW characteristics
  - Suggests a different approach for process topology mapping…

MPI On Multicore Nodes

- MPI Everywhere (single core/single thread MPI processes) still common
  - Easy to think about
  - We have good performance models (or do we?)
- In reality, there are issues
  - Memory per core declining
    - Need to avoid large regions for data copies, e.g., halo cells
    - MPI implementations could share internal table, data structures
      - May only be important for extreme scale systems
  - MPI Everywhere implicitly assume uniform communication cost model
    - Limits algorithms explored, communication optimizations used
- Even here, there is much to do for
  - Algorithm designers
  - Application implementers
  - MPI implementation developers
- One example: Can we use the single core performance model for MPI?
  - T = s + r n
  - Widely used and effective for designing parallel algorithms
  - Similar issues with logP, other models.

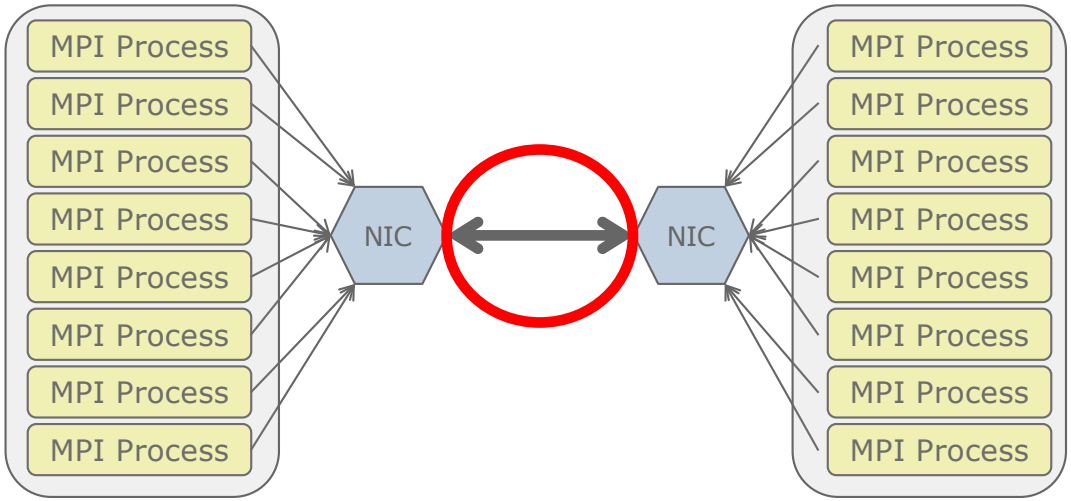
Rates Per MPI Process



- Ping-pong between 2 nodes using 1-16 cores on each node
- Top is BG/Q, bottom Cray XE6
- "Classic" model predicts a single curve – rates independent of the number of communicating processes

Why this Behavior?

- The T = s + r n model predicts the *same* performance independent of the number of communicating processes
  - What is going on?
  - How should we model the time for communication?

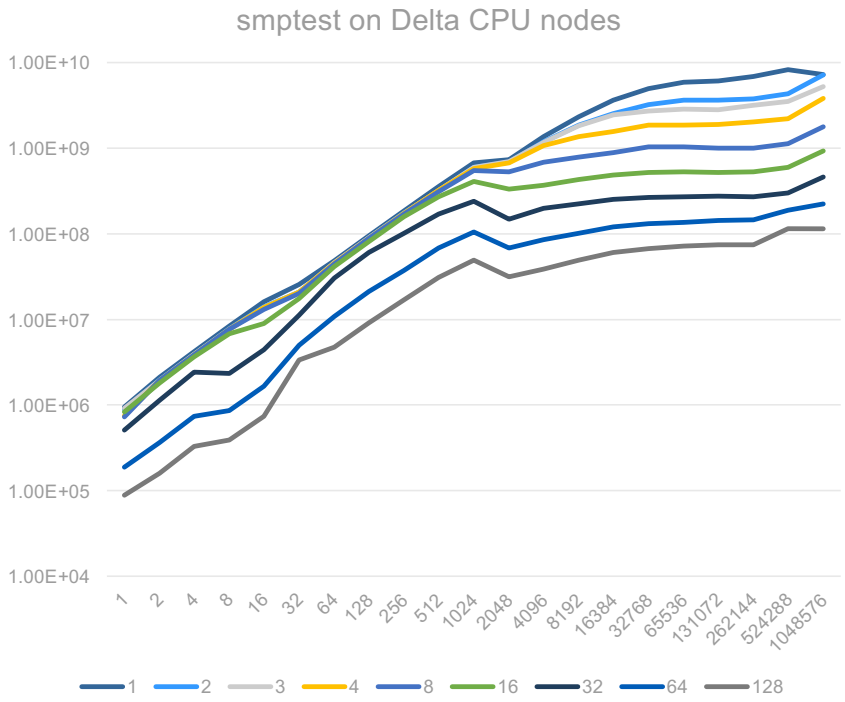


- Performance along data path is fast enough for more than one process but not for all processes
- This is a common design feature

A Better Model

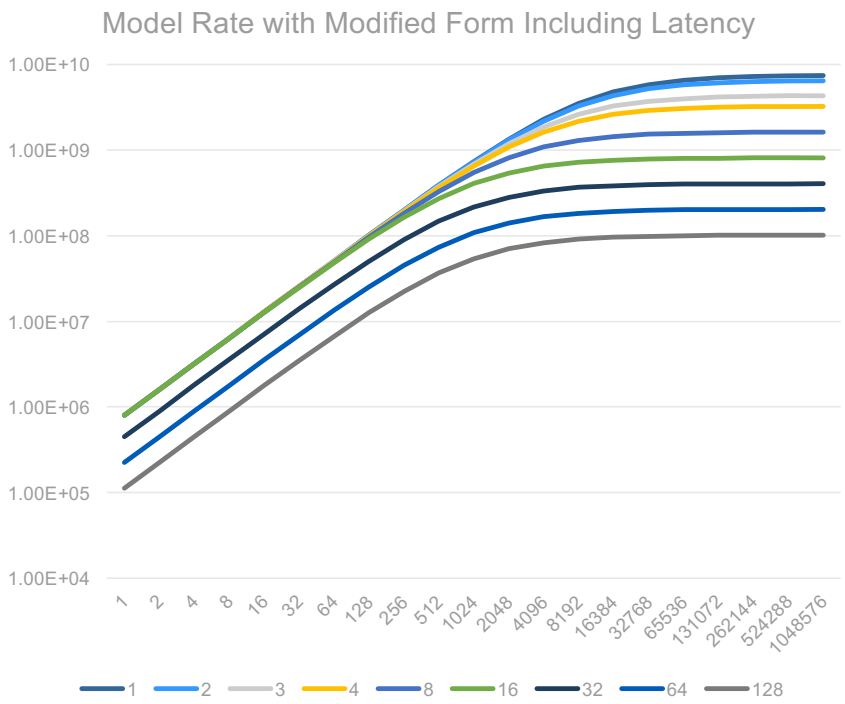
- For k processes sending messages, the sustained rate into network is
  - $\min(R_{NIC\text{-}NIC}, k\ R_{CORE\text{-}NIC})$
  - That is min of output rate ($R_{NIC\text{-}NIC}$) and input rate ($kR_{CORE\text{-}NIC}$)
- Thus
  - $T = s + k\ n/\min(R_{NIC\text{-}NIC}, k\ R_{CORE\text{-}NIC})$, or
  - $T = s + n(1/(R_{NIC\text{-}NIC}\min(R_{CORE\text{-}NIC}/kR_{NIC\text{-}NIC},1)))$
- Note if $R_{NIC\text{-}NIC}$ is very large (very fast network), this reduces to
  - $T = s + k\ n/(k\ R_{CORE\text{-}NIC}) = s + n/R_{CORE\text{-}NIC}$
- This model is approximate; additional terms needed to capture effect of shared data paths in node, contention for shared resources, etc.
- But this new term is by far the dominant one
- This is the ***max-rate*** model (for performance limited by the **max**imum available **rate** (bandwidth))
  - Logp model has a similar limitation and needs a similar modification

Updating the Maxrate Model for Large Core Count Nodes



- Original max rate
  - $T = s + n(1/(R_c \min(R_n/kR_c, 1)))$
  - Bandwidth limited to a "max rate" into network
- Updated time model:
  - Add limit to # of concurrent messages in latency term
  - $T = s \max(1, k/n_c) + n(1/(R_c \min(R_n/kR_c, 1)))$
- N_c = limit on number of concurrent messages (not exactly concurrent, but point at which requests start needing to wait longer than one latency s)
- R_n = rate off node ($R_{NIC\text{-}NIC}$)
- R_c = rate per process ($R_{CORE\text{-}NIC}$)
- k = number of concurrent communications
- For n_c > #cores, this reduces to the original max rate model
- R_n/R_c = # of processes that can communicate simultaneously at full rate

Updating the Maxrate model for large core count nodes



- Eyeball fit for NCSA Delta data
- Does not account for
  - Switch to rendezvous protocol
  - Short message protocol
- Does capture behavior for both long and short messages
- More accurate models use separate parameters for each message protocol (short, long, rendezvous)
- Only make the model as complicated (accurate) as needed

Modeling Communication from Manycore Nodes

- "Postal" model
  - $T = s + n(1/R)$
- Max rate
  - $T = s + kn/\min(R_n, kR_c) = s + n/\min(R_n/k, R_c) = s + n(1/R_c)/\min((R_n/R_c)/k, 1)$
  - $T = s + n(1/(R_c \min(1, (R_n/R_c)(1/k))))$
    - Limit on rate starts having an effect when $k \geq R_n/R_c$
- Many core max rate
  - Modifies the latency term to include a cost for processing more than n_c concurrent messages
  - $T = s \max(1, k/n_c) + n(1/(R_c \min(1, (R_n/R_c)(1/k))))$

Max number of messages that can be started at the same time

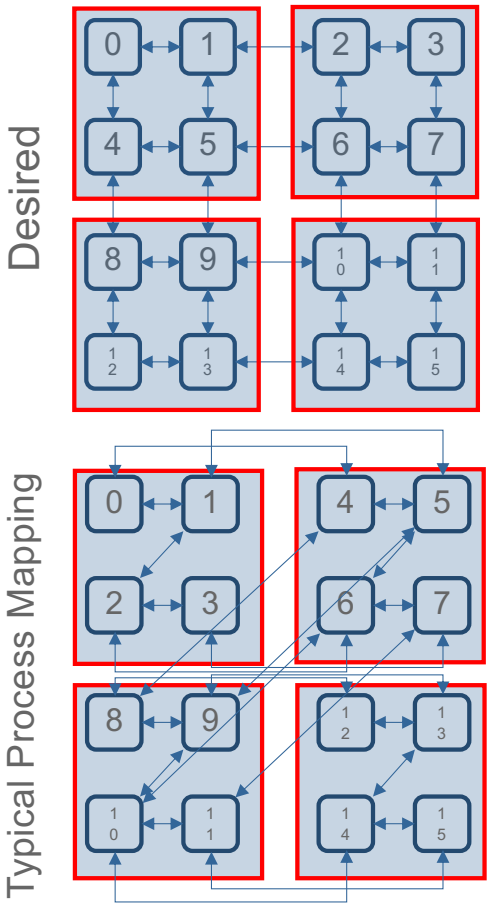
NIC is this times faster than single message BW

Implications for Algorithms

- This model
  - Must consider the impact of concurrent operations on node
    - Processes and threads are not performance independent
    - Another reason to reconsider BSP model – benefit in staggering communication
- General lessons
  - Shared resources typically have (at least) two parameters:
    - Total available (e.g., Bandwidth)
    - Total usable by any one resource (e.g., per thread bandwidth)
- Things to be wary of
  - Some systems have multiple data paths, but whether they are available depend on how processes are mapped to resources
    - MPI (version 4 and above) provides a partial solution to discovering this mapping

Performance Model to Algorithm

- Performance measurements of halo exchange show poor communication performance
  - Bandwidth per process low relative to “ping pong” measurements
  - Easy target – blame contention in the network
- But common default mapping of processes to nodes leads to more off-node communication
  - The max rate model predicts reduced performance once $R_{NIC\text{-}NIC}$ limit reached
- We can use this to create a better, and *simpler,* implementation of MPI_Cart_create

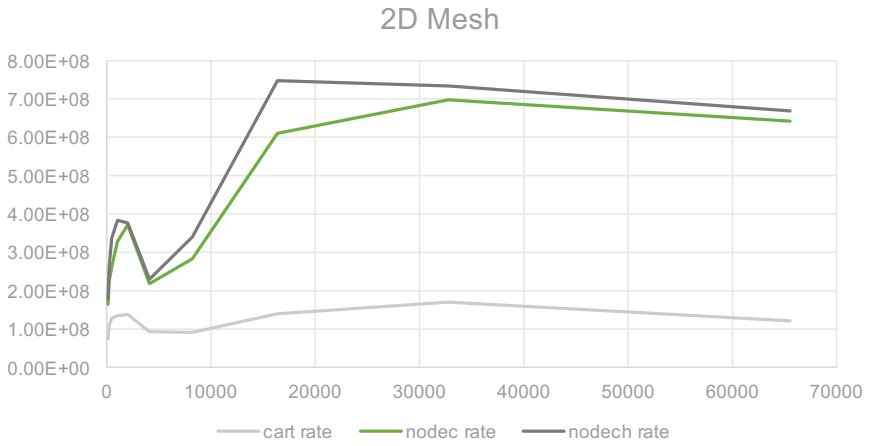


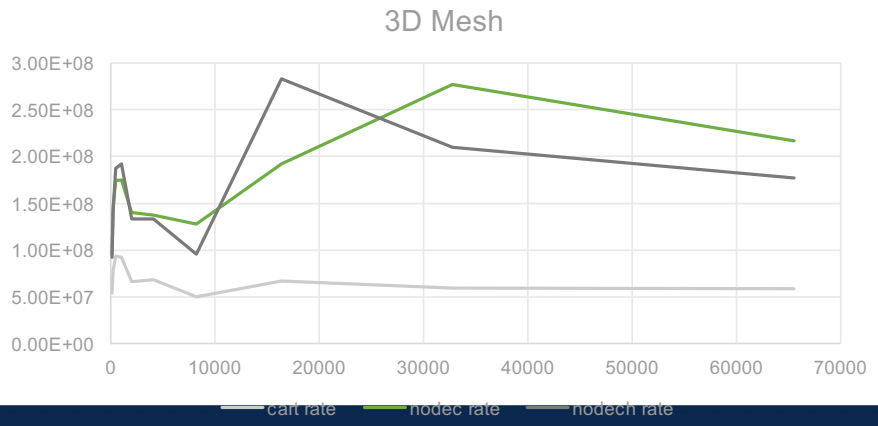
Building A Better MPI_Cart_create

- Hypothesis: A better process mapping **within** a node will provide significant benefits
  - **Ignore** the internode network topology
    - Vendors have argued that their network is fast enough that process mapping isn't necessary
    - They may be (almost) right – once data enters the network
- Idea for Cartesian Process Topologies
  - Identify nodes (see MPI_Comm_split_type)
  - Map processes *within* a node to minimize **inter**node communication
    - Trading **inter**node for **intra**node communication
    - *Using Node Information to Implement MPI Cartesian Topologies*, Gropp, William D., Proceedings of the 25th European MPI Users' Group Meeting, 18:1–18:9, 2018 https://dl.acm.org/citation.cfm?id=3236377
    - *Using Node and Socket Information to Implement MPI Cartesian Topologies*, Parallel Computing, 2019 https://doi.org/10.1016/j.parco.2019.01.001

Increasing Core Count Makes Proper Mapping More Important

- Cartesian mapping on Delta
  - CPU nodes have 2 AMD Milan x 64 cores each (GPU nodes have 1 AMD Milan and 4 A100 or A40 NVIDEA GPUs)
  - Slingshot-10 network
  - Performance in B/s (higher is better)
- Default mapping provides poor performance
  - Cart is MPI_Cart_create – also MPI_COMM_WORLD
  - Nodec uses node-awareness, inspired by max-rate model
  - Nodech extends to socket (3-level)





Tuning Must be in a Representative Environment

- For most processors and regular (e.g., vectorizable) computations
  - Memory bandwidth for a *chip* is much larger than needed by a single *core*
  - *Share of* memory bandwidth for a *core* (with all cores accessing memory) is much smaller than needed to avoid waiting on memory
- Performance tests on a single core can be very misleading
  - Example follows
  - Can use simple MPI tools to explore dependence on using one to all cores
  - Ask this question when you review papers ☺

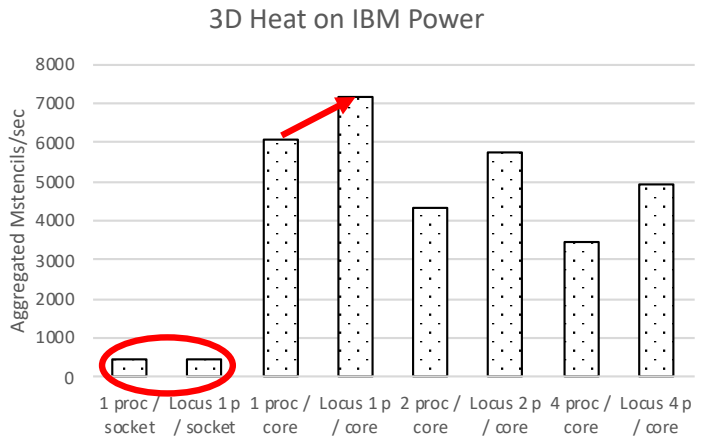
Stencil Sweeps

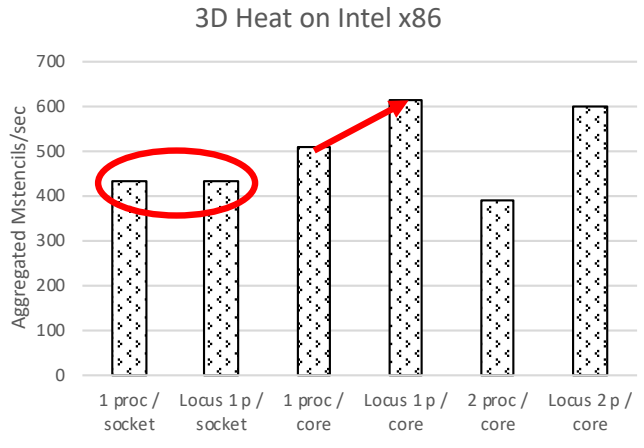
- Common operation for PDE solvers
  - Structured are often “matrix free”
  - Unstructured and structured mesh stencils have low ”computational intensity” – number of floating-point operations per bytes moved
- Conventional wisdom is that cache blocking and similar optimizations are ineffective
  - For example, “Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors” argues this, and provides experimental data to support it
  - https://epubs.siam.org/doi/10.1137/070693199 (accepted 2007, published 2009)
- But the analysis and experiments are usually based on one core per chip/socket
  - And the number of cores has grown substantially since 2007
  - What if every core is executing a stencil sweep?

Stencil Sweeps

```
void heat3d(double A[2][N+2][N+2][N+2]) {
int i, j, t, k;
#pragma @LOCUS loop=heat3d
for(t = 0; t < T-1; t++) {
for(i = 1; i < N+1; i++) {
for(j = 1; j < N+1; j++) {
for (k = 1; k < N+1; k++) {
A[(t+1)%2][i][j][k] = 0.125 * (A[t%2][i+1][j][k] -
2.0 * A[t%2][i][j][k] + A[t%2][i-1][j][k]) + 0.125 * (A[t%2][i][j+1][k]
- 2.0 * A[t%2][i][j][k] + A[t%2][i][j-1][k]) + 0.125 * (A[t%2][i][j][k-
1] - 2.0 * A[t%2][i][j][k] + A[t%2][i][j][k+1]) + A[t%2][i][j][k]; } } } }
}
```

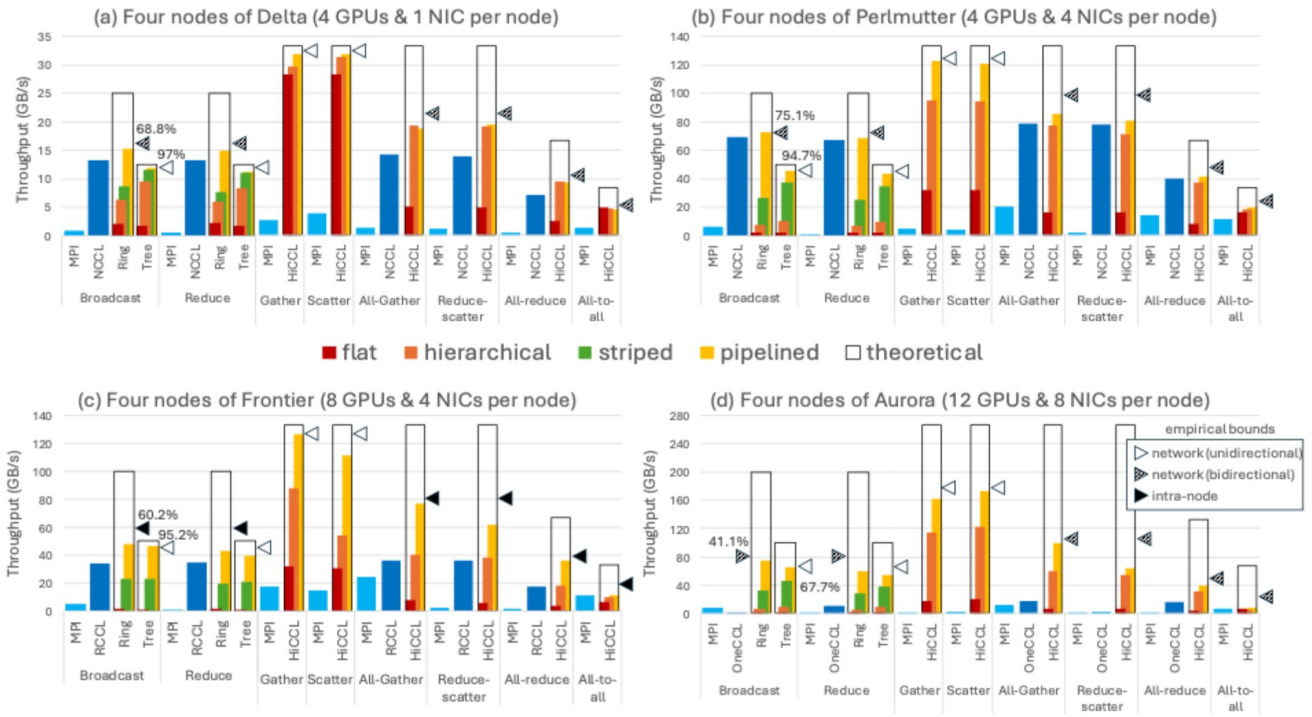
Memory Bandwidth > any individual core needs but < what all cores (could) need





Using Performance Models to Understand Limits

- Work of Mert Hidayetoglu and colleagues
- GPU aware Collectives, assembled from building blocks
- Hollow rectangles give performance model bounds
- Note allows *portable* construction of GPU aware collectives competitive with NVIDIA NCCL and faster than most MPI implementations



Some Final Thoughts

- Computing systems will continue to be heterogeneous; systems will either have a mixture of node types from different generations or be obsolescent for most of their lifetime
  - We need better algorithms, performance models, and especially software tools for heterogeneous systems
- Software is changing and we need to take advantage of new approaches
- AI. Both more and less than the hype. It will change how we develop algorithms and applications
  - Many questions remain about limits of AI, quality of code (training code written by humans often poor), cost relative to human

What does this have in common with HPL for a (macro) Heterogeneous Computing System?

Answer:

Something we used to be able to build but can’t any more

More on Heterogeneity

- Why bother with heterogeneous applications?
  - Just use homogenous sections (or move to a bigger machine)
  - But:
    - Memory (total for problem)
    - Complex workflows (but this may already work as long as not tightly coupled)
    - Adaptive algorithms and codes that don't need homogeneity
- What are good benchmarks for heterogeneous nodes?
  - Not just for homogenous nodes with identical heterogeneous elements
  - And benchmarks drive (sadly?) innovation and investment
- What are the most effective programming models and systems for heterogeneous environments?
  - Note that we still struggle with vectorization and programming for accelerators.
  - I still don't have a version of HPL that looks like what we teach students for the algorithms and that is good enough to be used an both a benchmark and coding exemplar
- Algorithms and other issues
  - There are still communication tradeoffs – adding more nodes doesn't always improve performance
  - Greatest common denominator approach still limits choices

NCSA Directions

- NCSA has recently deployed two generations of supercomputers that provide a heterogeneous node environment
  - Delta (CPU and GPU)
  - Delta AI (and supplement)
  - Delta Supplement
  - DeltaAI direct liquid cooled
- **6 GPU node types**, including large memory
- All on same network, share file system including center-wide system
- Provides the majority of GPU cycles for NSF ACCESS; most popular resource in NAIRR Pilot 1st round
- Virtue of necessity

Name	Capabilities	Details
Delta	GPU-rich supercomputer for simulation, AI/ML, Data Science	100 4xA100, 5 8xA100,100 4xA40, 1 8xMI100, 124 dual Milan
DeltaAI incl Supplement	GPU-rich supercomputer for AI/ML	132 4xGH200
DeltaAI-IC	Funded by Campus	20 4xGH200
Delta Supplement	Supplement for large AI models	8 8xH200 (x86 CPU)

Software Implications for Programs

- Very fast cores and cheap, abundant memory has changed the design of programming systems and approaches
- Rise of python
  - Rich environment
  - Opportunities include code generation
    - There is a tension between generality and performance in conventional libraries – example, having a variable number of physical dimensions makes a library routine more flexible – but may suppress optimizations such as vectorization
    - A high level description makes it easier to apply code transformations, including ones driven by AI
- Rise of memory-safe languages (e.g., Rust)
  - More discipline in using memory; tradeoff in describing limits of access (sparse disjoint scatter gather becomes difficult)

Summary and Challenges

- Achieving performance is *hard*
  - Compilers, Libraries, and tools can *help*
  - But understanding what is achievable is essential
  - Relatively simple performance models can help answer “Is this as fast as it *should* be?”
- Performance models need to include *bottlenecks*
  - Not a new idea but not common in analysis of parallel programs
  - BSP-style programming exacerbates bottlenecks
- Rapid innovation in software and hardware requires a different approach to applications
  - Systems will be evergreen and heterogeneous or (soon) obsolescent
  - Operating environment will be more dynamic
  - Assumptions of uniform performance will limit opportunities

Some Performance Engineering/Analysis Papers

A very partial and somewhat personal list

- LogP: towards a realistic model of parallel computation
  - PPOPP 1993; Distinguish overhead from latency
- Hitting the memory wall: implications of the obvious
  - ACM SIGARCH News 1995; Memory performance is (often) the limit on performance
- Achieving high sustained performance in an unstructured mesh CFD application
  - SC99 Gordon Bell (Special category); Memory limits on performance, “Achievable performance”
- Experimental Analysis of Algorithms
  - Notices of the American Mathematical Society, March 2001 (also thesis)
- The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q
  - SC03; Detailed performance model gives insight into behavior
- Roofline: an insightful visual performance model for multicore architectures
  - CACM 2009; Performance from the view of architecture, not application
- Architectural constraints to attain 1 exaflop/s for three scientific application classes.
  - IPDPS 2011;
- A Multiplatform Study of I/O Behavior on Petascale Supercomputers
  - HPCD 2015; Measures how terrible (and it is terrible) I/O performance is in HPC. Uses Darshan
- Modeling MPI Communication Performance on SMP Nodes: Is it Time to Retire the Ping Pong Test
  - EuroMPI 2016; Impact of sharing resources among threads/processes; max-rate model
- Using node and socket information to implement MPI Cartesian topologies
  - ParComm 2019; Designing an algorithm based on max-rate performance model

Need Computing Time? Try Delta and DeltaAI!

- NCSA operates **Delta** and **DeltaAI**, the most GPU-performant systems in the NSF portfolio
  - Delta is a mostly A40 and A100 GPU nodes, with 8 new 8x H200 GPU nodes, along with dual CPU (AMD Milan) nodes
  - DeltaAI has quad Grace Hopper (NVIDIA H100 GPU) nodes
  - Shared filesystem including large SSD; post-POSIX IO coming
  - Batch, Open OnDemand access, and Science Gateway support
  - https://delta.ncsa.illinois.edu/ (for both Delta and DeltaAI)
  - While smaller than DOE or especially industry systems, useful for algorithm development
- Apply for time at https://access-ci.org/ or https://nairrpilot.org/
  - Most academic researchers in the US can get time, and it is free to you!
  - You don't need an NSF grant to apply!

Thanks!

- I have been blessed with many wonderful colleagues, students, and friends
- It has been an honor to work with so many of you and to meet new researchers interested in thinking quantitatively about performance
- My last time at ICERM: Synchronization-reducing and Communication-reducing Algorithms and Programming Models for Large-scale Simulations, Jan 8-13, 2012