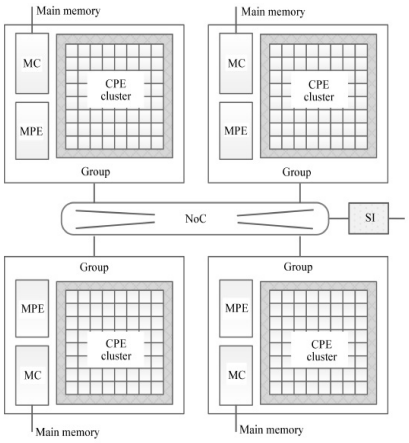
Challenges in Programming Extreme Scale Systems

William Gropp

wgropp.cs.illinois.edu

Towards Exascale Architectures



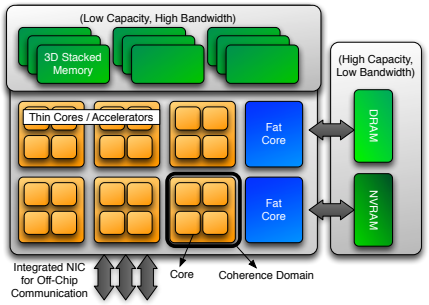
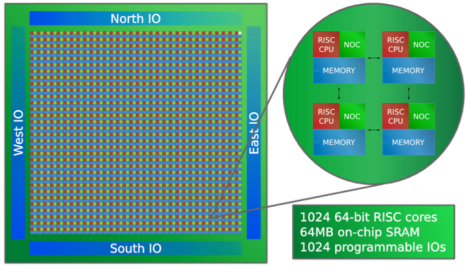
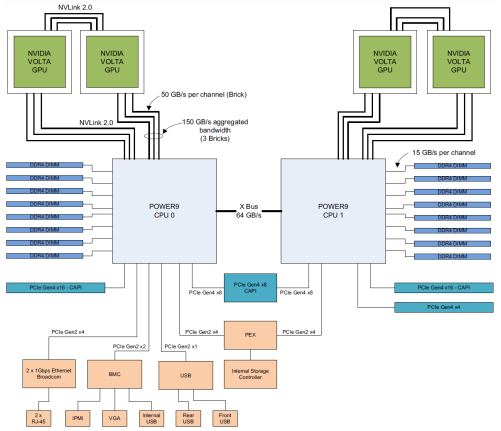


Figure 2.1: Abstract Machine Model of an exascale Node Architecture





Sunway TaihuLight

- Heterogeneous processors (MPE, CPE)
- No data cache
- Tianhe2a has some data cache

From "Abstract Machine Models and Proxy Architectures for Exascale Computing Rev 1.1," J Ang et al

Adapteva Epiphany-V

- 1024 RISC processors
- 32x32 mesh
- Very high power efficiency (70GF/W)

DOE Sierra

- Power 9 with 4 NVIDA Volta GPU
- 4320 nodes

DOE Summit similar, but

- 6 NVIDIA GPUs/node
- 4608 nodes

Separate the Programming Model from the Execution Model

- What is an execution model?
  - It's how you think about how you can use a parallel computer to solve a problem
- Why talk about this?
  - The execution model can influence what solutions you consider (see the *Whorfian hypothesis* in linguistics)
  - After decades where many computer scientists only worked with one execution model, we are now seeing new models and their impact on programming and algorithms

Examples of Execution Models

- Von Neumann machine:
  - Program counter
  - Arithmetic Logic Unit
  - Addressable Memory
- Classic Vector machine:
  - Add “vectors” – apply the same operation to a group of data with a single instruction
    - Arbitrary length (CDC Star 100), 64 words (Cray), 2 words (SSE)
- GPUs with collections of threads (Warps)

Common Execution Models With Performance Metrics

- Simplest
  - Each processor – 1 floating point operation per cycle. No memory cost (data is available when needed)
- Consequences for algorithm design
  - Focus on floating point operations for algorithms
    - Ignores memory effects
    - Ignores trade-offs of more floating point operations for better memory use to achieve faster time-to-solution
    - Emphasizes scalar operations (easier to think about)
- Adding Parallelism
  - May include Amdahl limits – impact of serial fraction $T = (1-f)T_0/p + fT_0$
  - Between processors (nodes) – moving n words of data takes time $T = s + r n$

Improving the Model – Each "Processor"

- Add memory cost
  - Sustained performance to memory
    - *Achieving High Sustained Performance in an Unstructured Mesh CFD Application*, W. K. Anderson, William D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith, Proceedings of the ACM/IEEE SC99 Conference on High Performance Networking and Computing. https://ieeexplore.ieee.org/document/1592711
    - T = max(c / flops, (r+w) / membw)
      - Paper also considers instruction rate, important for codes with significant pointer and index manipulations (example later in talk)
    - "Roofline" extends this to use "arithmetic intensity" as measure of an algorithm
  - Execution Cache Memory
    - *Execution-Cache-Memory Performance Model: Introduction and Validation,* Johannes Hofmann, Jan Eitzinger, and Dietmar Fey. https://arxiv.org/pdf/1509.03118.pdf

Improving the Model – Between Processors

- Separate latency from overhead
  - *LogP: a practical model of parallel computation*, David E. Culler, Richard M. Karp, David Patterson, Abhijit Sahay,Eunice E. Santos, Klaus Erik Schauser, Ramesh Subramonian, and Thorsten von Eicken
  - Many variations (e.g., LogGp); also variations that include network capacity and bisection bandwidth

Programming Models and Systems

- In past, often a tight connection between the execution model and the programming approach
  - Fortran: FORmula TRANslation to von Neumann machine
  - C: e.g., “register”, ++ operator match PDP-11 capabilities, needs
- Over time, execution models and reality changed but programming models rarely reflected those changes
  - Rely on compiler to “hide” those changes from the user – e.g., auto-vectorization for SSE(n)
- Consequence: Mismatch between users’ expectation and system abilities.
  - Can’t fully exploit system because user’s mental model of execution does not match real hardware
  - Decades of compiler research have shown this problem is extremely hard – can’t expect system to do everything for you.

New Applications Will Be As Varied and Demanding

- Wide range of applications today
  - More than CFD, Structural Mechanics, Molecular dynamics, QCD
  - Include image processing, event-driven simulations, graph analytics
- Rising importance of machine learning and ***Imitation Intelligence***
  - The appearance of intelligence without anything behind it
  - Still incredibly powerful and useful, but …
  - Not ***Artificial intelligence*** (though this is the correct name for the *field*)
    - Intelligence achieved through artificial means
  - Training required for each "behavior" (one reason this is II, not AI)
  - Current methods require large amounts of data and compute to train; application of the trained system is not (relatively speaking) computationally intensive
- Workflows involving all of the above
  - One example:
    - Use Einstein Toolkit to compute gravitational waves from cataclysmic events
      - This is classic time-dependent PDE solution
    - Use waveforms to train a machine learning system
    - Use that system to provide (near) real time detection of gravitational waves from aLIGO
    - https://journals.aps.org/prd/abstract/10.1103/PhysRevD.97.044039

The Easy Part – Internode communication

- Often focus on the “scale” in Exascale as the hard part
  - How to deal with a million or a billion processes?
  - But really not too hard
    - Many applications have large regions of regular parallelism
  - Or nearly impossible
    - If there isn’t enough independent parallelism
  - Challenge is in handling definition and operation on distributed data structures
  - Many solutions for the internode programming piece
  - The dominant one in technical computing is the Message Passing Interface (MPI)

Modern MPI

- MPI is much more than message passing
  - I prefer to call MPI a programming *system* rather than a programming *model*
    - Because it implements several programming *models*
- Major features of MPI include
  - Rich message passing, with nonblocking, thread safe, and persistent versions
  - Rich collective communication methods
  - Full-featured one-sided operations
    - Many new capabilities over MPI-2
    - Include remote atomic update
  - Portable access to shared memory on nodes
    - Process-based alternative to sharing via threads
    - (Relatively) precise semantics
  - Effective parallel I/O that is not restricted by POSIX semantics
    - But see implementation issues …
  - Perhaps most important
    - Designed to support “programming in the large” – creation of libraries and tools

MPI (The Standard) Can Scale Beyond Exascale

- MPI implementations already supporting more than 1M processes
  - Several systems (including Blue Waters) with over 0.5M independent cores
- Many Exascale designs have a similar number of nodes as today's systems
  - MPI as the internode programming system seems likely
- There are challenges
  - Connection management
  - Buffer management
  - Memory footprint
  - Fast collective operations
  - One sided notification still limited (and under discussion)
  - Fault Tolerance remains an open issue
    - But the importance of this is unclear
  - …
  - And no implementation is as good as it needs to be, but
  - There are no intractable problems here – MPI implementations can be engineered to support Exascale systems, even in the MPI-everywhere approach
- MPI continues to evolve –MPI 4.0 Draft released at SC in Dallas earlier this month

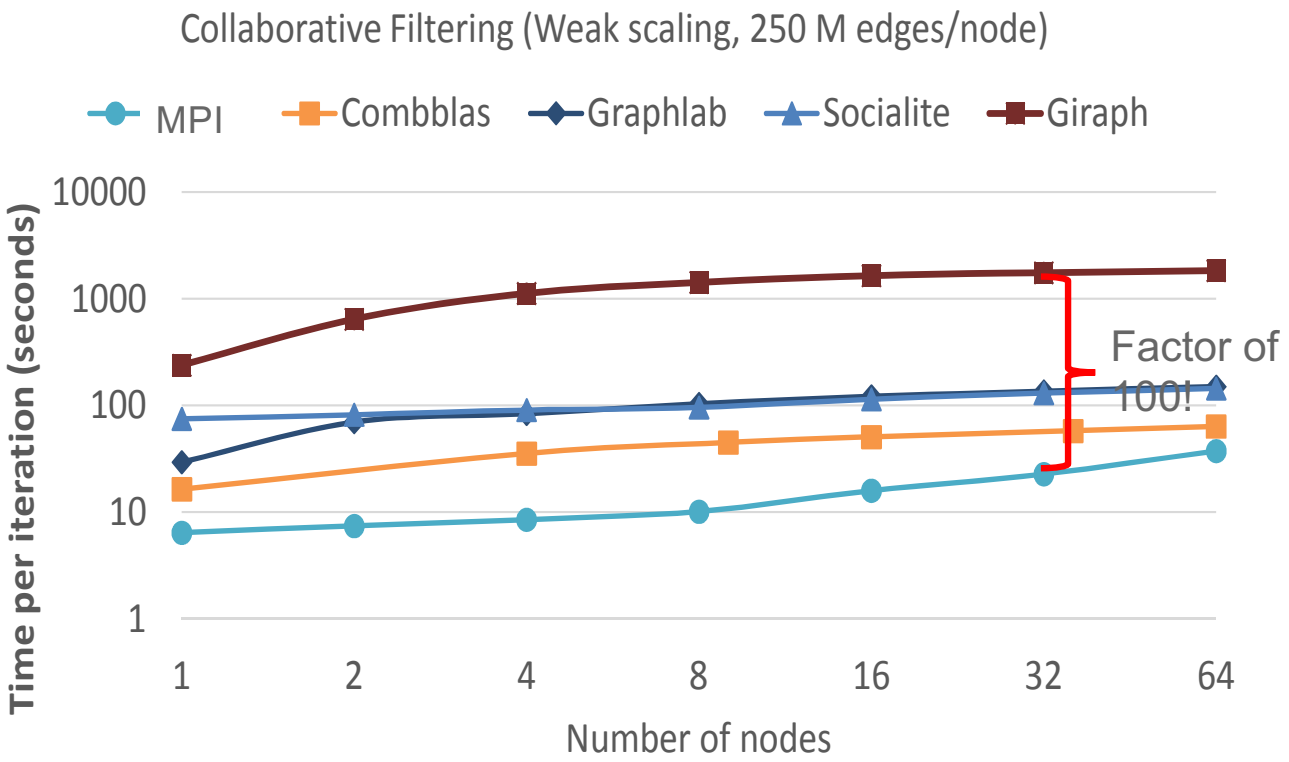
Applications Still Mostly MPI-Everywhere

- "the larger jobs (> 4096 nodes) mostly use message passing with no threading." – Blue Waters Workload study, https://arxiv.org/ftp/arxiv/papers/1703/1703.00924.pdf
- Benefit of programmer-managed locality
  - Memory performance nearly stagnant (will HBM save us?)
  - Parallelism for performance implies locality must be managed effectively
- Benefit of a single programming system
  - Often stated as desirable but with little evidence
  - Common to mix Fortran, C, Python, etc.
  - But…Interface between systems must work well, and often don't
    - E.g., for MPI+OpenMP, who manages the cores and how is that negotiated?

MPI is not a BSP system

- BSP = Bulk Synchronous Programming
  - Programmers **like** the BSP model, adopting it even when not necessary (see “A Formal Approach to Detect Functionally Irrelevant Barriers in MPI Programs”)
  - Unlike most programming models, *designed* with a **performance model** to encourage *quantitative* design in programs
- MPI makes it easy to emulate a BSP system
  - Rich set of collectives, barriers, blocking operations
- MPI (even MPI-1) sufficient for dynamic adaptive programming
  - The main issues are performance and “progress”
  - Improving implementations and better HW support for integrated CPU/NIC coordination the answer

MPI is not only for Scientific Computing



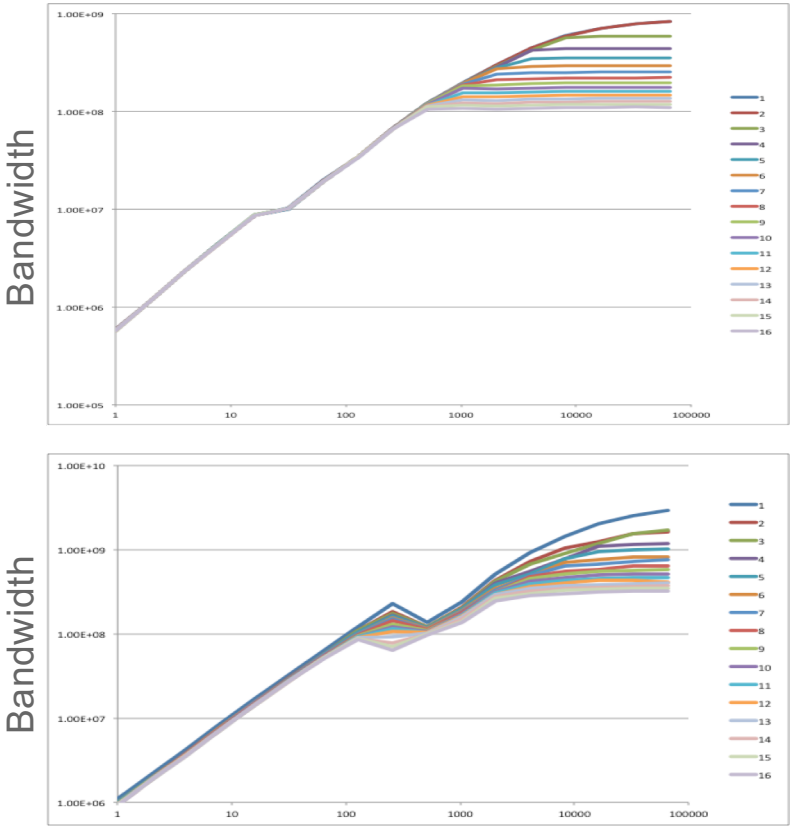
Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets

Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M. Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey

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?

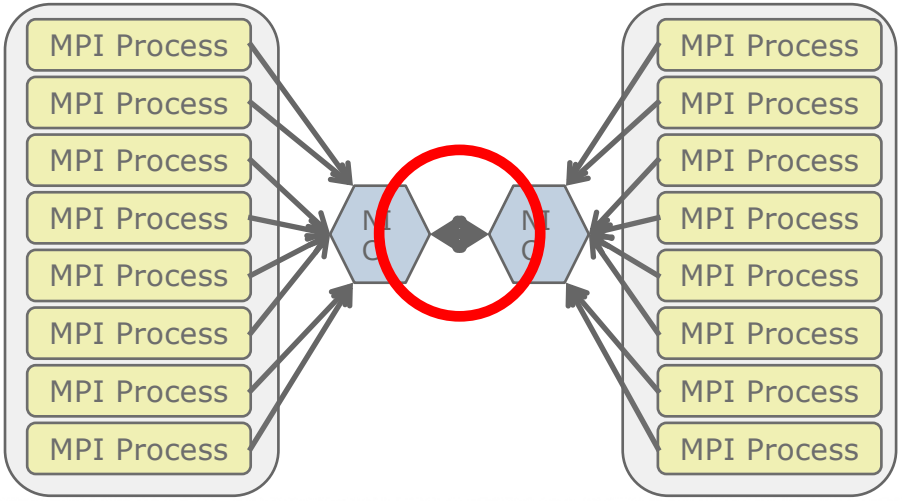
Rates Per MPI Process



- 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?

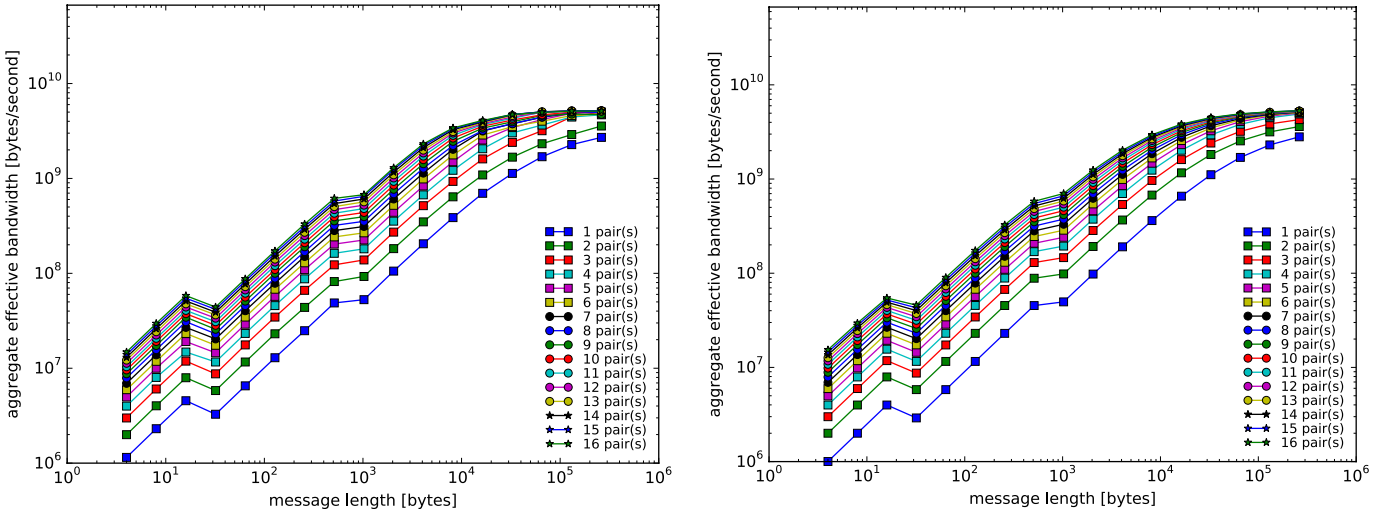
- The T = s + r n model predicts the *same* performance independent of the number of communicating processes
  - What is going on?
  - How should we model the time for communication?



A Slightly Better Model

- For k processes sending messages, the sustained rate is
  - $\min(R_{NIC\text{-}NIC}, k\ R_{CORE\text{-}NIC})$
- Thus
  - $T = s + k\ n/\min(R_{NIC\text{-}NIC}, k\ R_{CORE\text{-}NIC})$
- 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
- But this new term is by far the dominant one

Comparison on Cray XE6



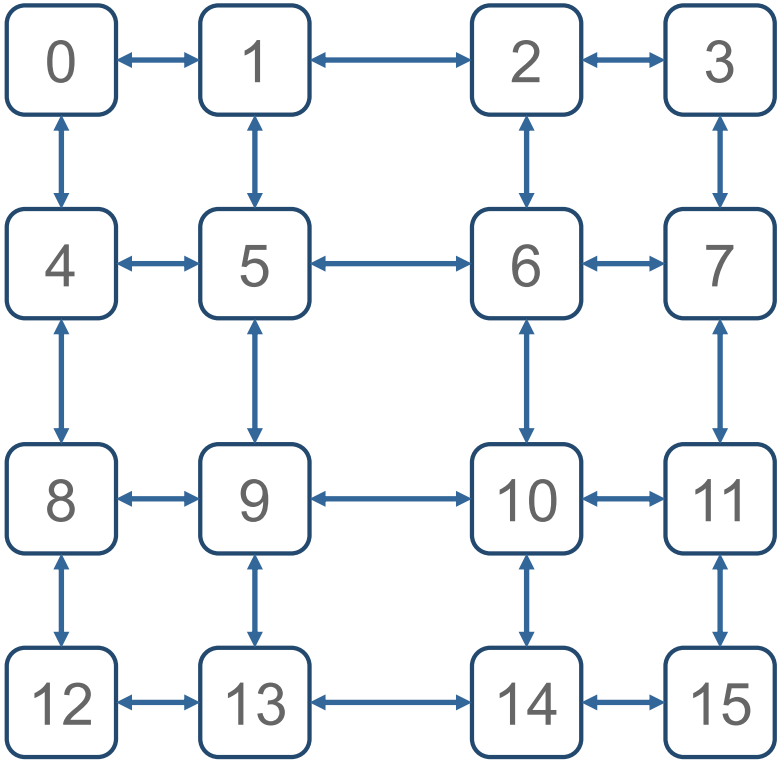
Measured Data Max-Rate Model

Modeling MPI Communication Performance on SMP Nodes: Is it Time to Retire the Ping Pong Test, W Gropp, L Olson, P Samfass, Proceedings of EuroMPI 16, https://doi.org/10.1145/2966884.2966919

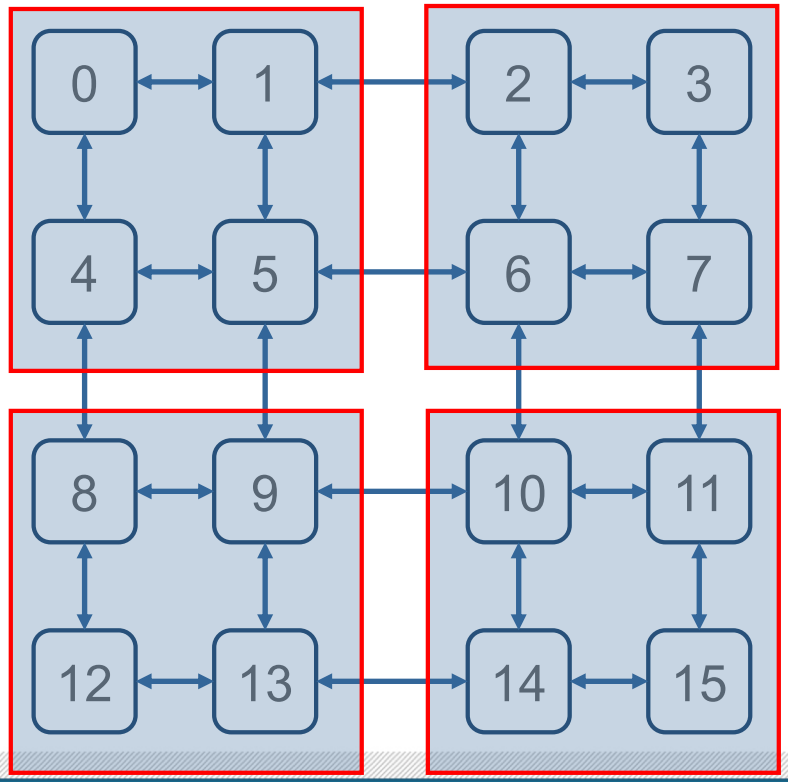
MPI Virtual Process Topologies

- Lets user describe some common communication patterns
- Promises
  - Better performance (with “reorder” flag true)
  - Convenience in describing communication (at least with Cartesian process topologies)
- Reality
  - “Reorder” for performance rarely implemented
    - Few examples include NEC SX series and IBM BlueGene/L
  - Challenge to implement in general
    - Perfect mapping complex to achieve except in special cases
      - And perfect is only WRT the abstraction, not the real system
- Rarely used in benchmarks/applications, so does not perform well, so is rarely used in benchmarks/applications

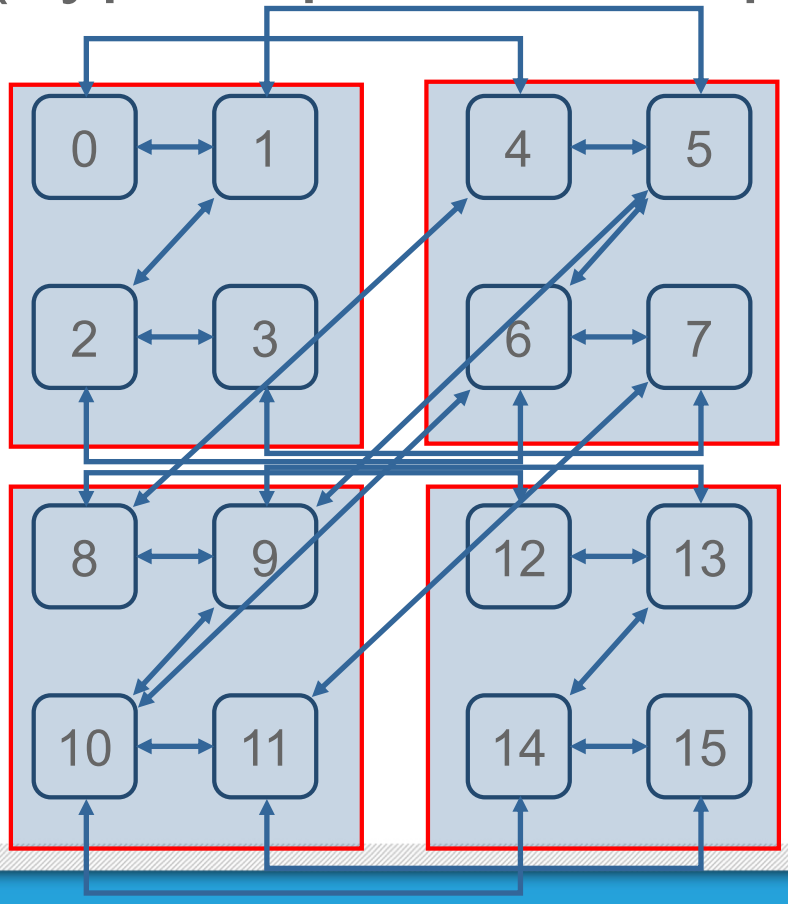
Example Cartesian Process Mesh



Example Cartesian Process Mesh – Four Nodes (Desired)



Example Cartesian Process Mesh – Four Nodes (Typical process mapping)



Can We Do Better?

- 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 **intra**node for **inter**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

Algorithm

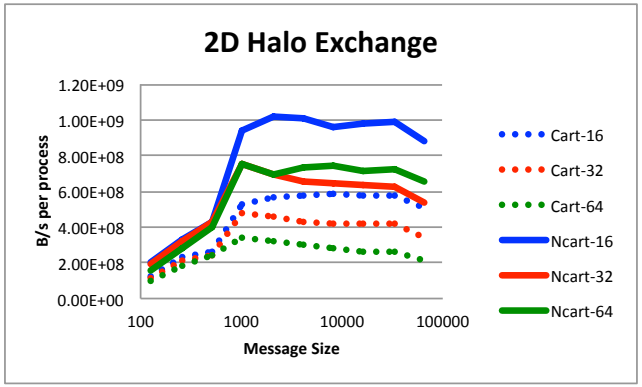
- Find the nodes
  - MPI Provides a way to split a communicator based on a characteristic; MPI_COMM_TYPE_SHARED works on all systems
- Create communicators of (a) all processes on the same node (nodecomm) and (b) the 0^{th} process from each node (leadercomm)
  - All processes now know number of processes on each node and the number of nodes
- Form a 2 (or 3) level decomposition of the process mesh
  - Factor dimensions and find consistent pair in each dimension
- From rank in nodecom and leadercomm, compute coordinates in node and among nodes. Gives new coordinate in mesh and hence new rank
- Use MPI_Comm_split on this rank to form new Cartesian communicator

Testing the Hypothesis: The Systems

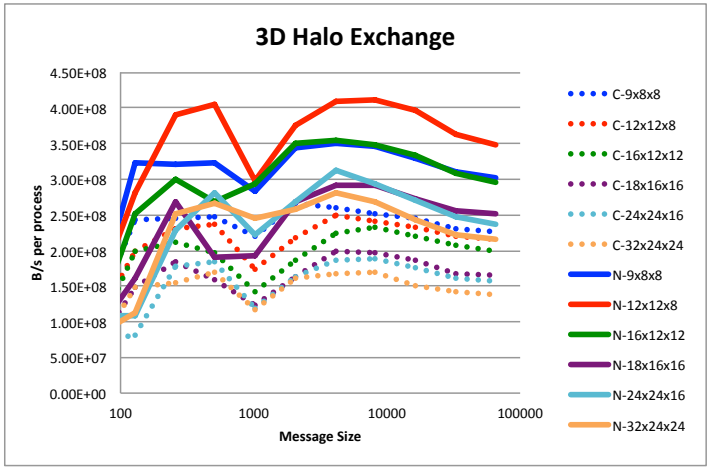
- Blue Waters at Illinois
  - Cray XE6/XK7
  - 3D mesh (Gemini); service nodes embedded in mesh
  - 22,636 XE6 nodes, each with 2 AMD Interlagos (and 4228 XK7 nodes)
- Theta at Argonne
  - Cray XC40
  - Dragonfly (Aires) interconnect
  - 4392 Intel KNL nodes
- Piz Daint at Swiss National Supercomputing Center
  - Cray XC50/XC40
  - Dragonfly (Aires) interconnect
  - 5320 XC50 and 1813 XC40 nodes

Comparing 2D Halo Exchanges

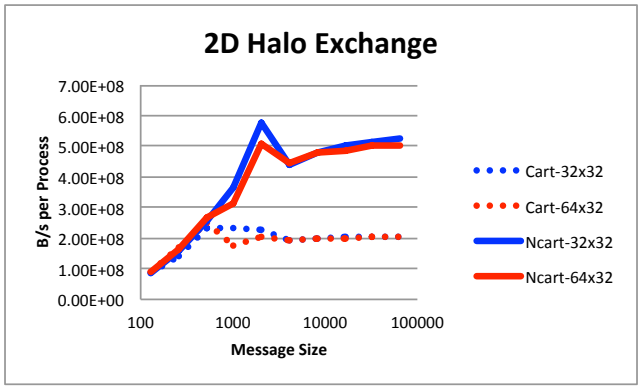
Blue Waters



Piz Daint

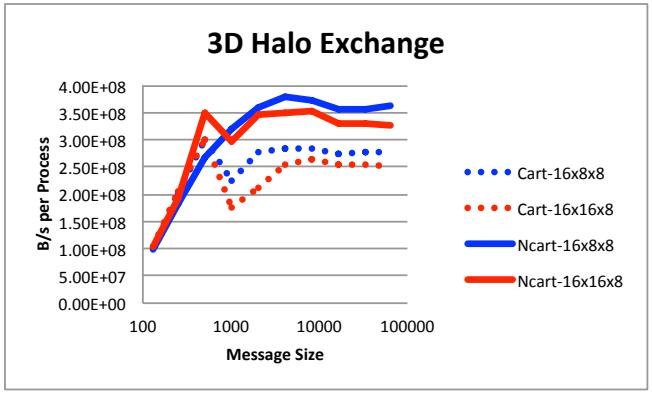


Theta

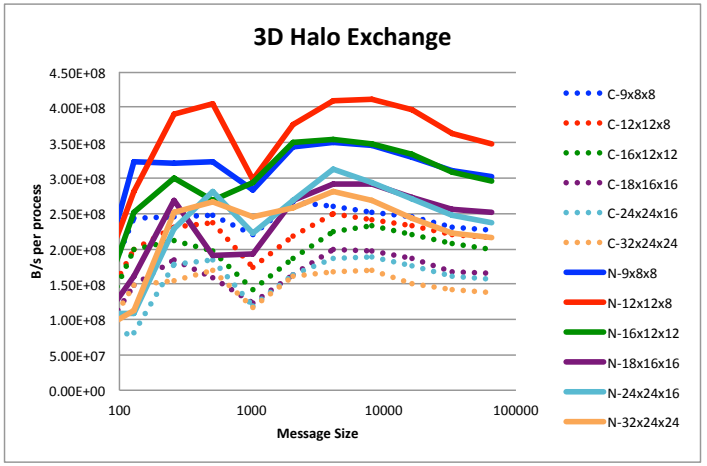


Comparing 3D Halo Exchanges

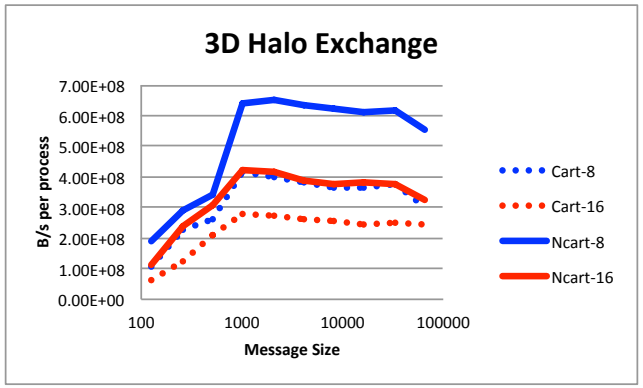
Blue Waters



Piz Daint



Theta



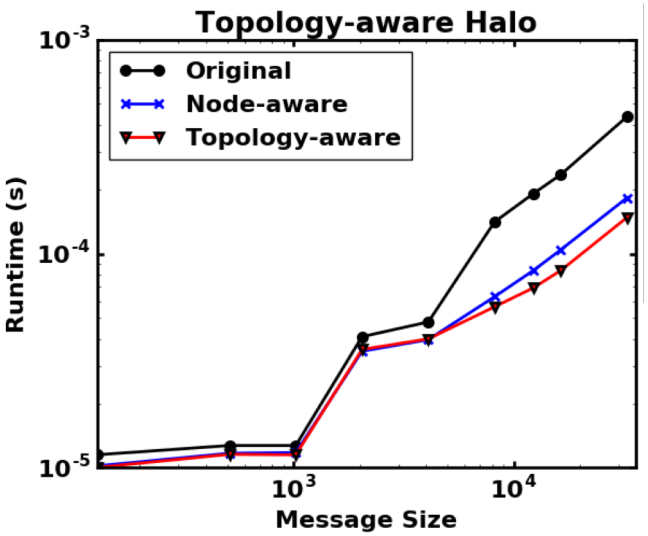
Comparing On- and Off-node Communication

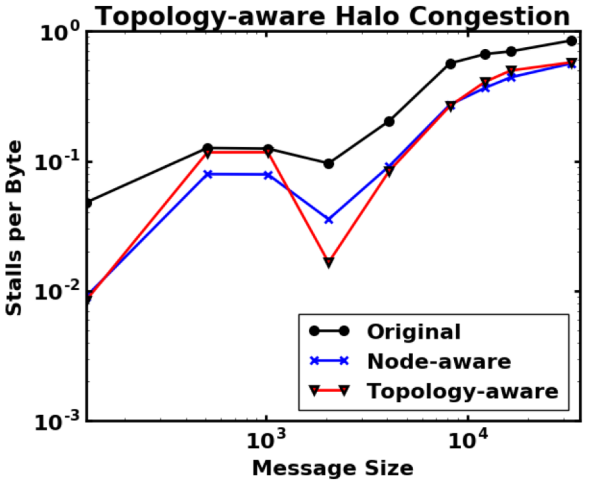
- Number of intranode (on-node) and internode (off-node) communication partners *per process*
- 16 processes per node
- Size is Cartesian virtual process topology
- “Nodecart” mapping is significantly better
  - In terms of reducing off-node communication in favor of on-node communication

			On-node			Off-node		
Type	Dim	Size	Min	Max	Avg	Min	Max	Avg
Cart	2	128x128	1	2	1.88	2	3	2.12
Nodecart	2	128x128	2	4	3	0	2	1
Cart	3	32x32x16	1	2	1.88	4	5	4.12
Nodecart	3	32x32x16	3	4	3.5	2	3	2.5

How Important is Network Topology?

- No answer yet, but…
- 432 nodes, 3D halo exchange on Blue Waters
  - Requested a cube of nodes, used non-standard routines to implement mapping for network topology
- Part of study into scalable Krylov methods (looking to avoid the blocking MPI_Allreduce)
- Nodecart version provides most of the benefit with no need for network topology information
- Some (nontrivial) further benefit possible by taking network topology into account
- But the largest contribution comes from node-awareness
- Thanks to Paul Eller for these results



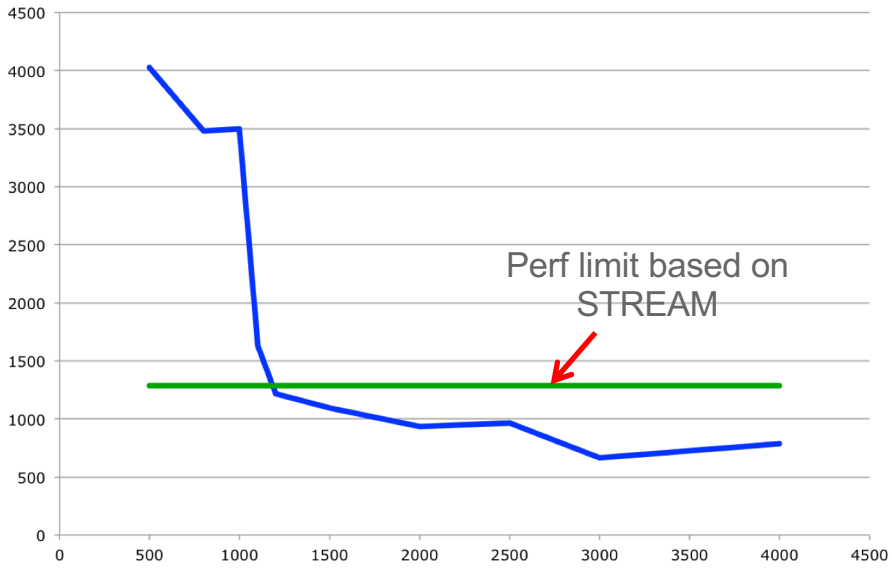


Dreams and Reality

- For codes that demand performance (and parallelism almost always implies that performance is important enough to justify the cost and complexity of parallelism), the dream is performance portability
- The reality is that most codes require specialized code to achieve high performance, even for non-parallel codes
- A typical refrain is “Let The Compiler Do It”
  - This is the right answer …
    - If only the compiler *could* do it
  - Lets look at one of the simplest operations for a single core, dense matrix transpose
    - Transpose involves only data motion; no floating point order to respect
    - Only a double loop (fewer options to consider)

A Simple Example: Dense Matrix Transpose

- do j=1,n
  do i=1,n
  b(i,j) = a(j,i)
  enddo
  enddo
- No temporal locality (data used once)
- Spatial locality only if (words/cacheline) * n fits in cache

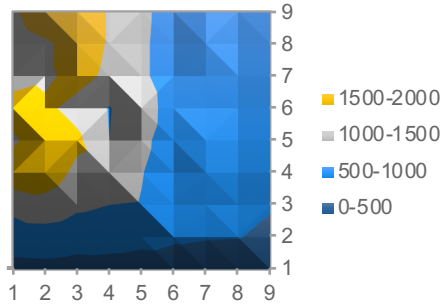


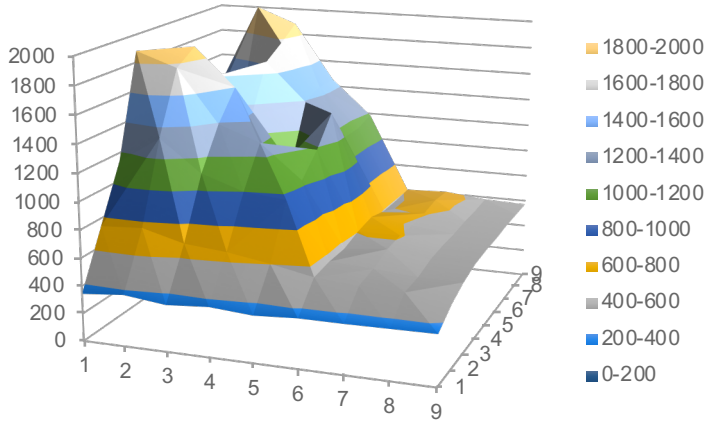
- Performance plummets when matrices no longer fit in cache

Blocking for Cache Helps

- do jj=1,n,stridej
  do ii=1,n,stridei
  do j=jj,min(n,jj+stridej-1)
  do i=ii,min(n,ii+stridei-1)
  b(i,j) = a(j,i)
- Good choices of stridei and stridej can improve performance by a significant factor (nearly 3X)
- How sensitive is the performance to the choices of stridei and stridej?

Simple, unblocked code compiled with O3 – 709MB/s





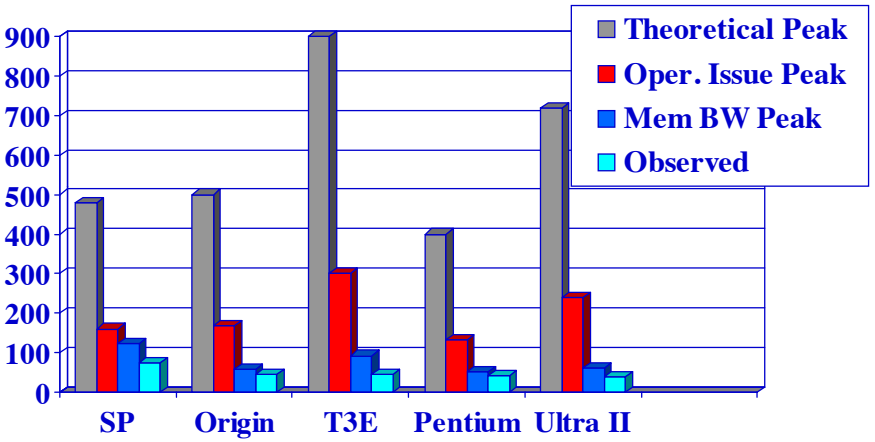
Sparse Matrix-Vector Product

- Common operation for optimal (in floating-point operations) solution of linear systems
- Sample code (common CSR format):

```
for row=1,n
     m   = i[row] - i[row-1];
     sum = 0;
     for k=1,m
          sum += *a++ * x[*j++];
     y[i] = sum;
```

- Data structures are a[nnz], j[nnz], i[n], x[n], y[n]
- Insert one slide with sparse matrix, including the Gordon Bell award

- Memory motion:
  - nnz (sizeof(double) + sizeof(int)) + n (2*sizeof(double) + sizeof(int))
  - Assume a perfect cache (never load same data twice)
- Computation
  - nnz multiply-add (MA)



Results From 1999 SC Paper

Real Codes Include Performance Workarounds

- Code excerpt from VecMDot_Seq in PETSc
- Code is unrolled to provide performance
  - Decision was made once (and verified as worth the effort *at the time*)
  - Remains part of the code forevermore
  - Unroll by 4 *probably* good for vectorization
    - But not necessarily best for performance
    - Does not address alignment

```
switch (j_rem=j&0x3) {
case 3:
  x2    = x[2];
  sum0 += x2*yy0[2]; sum1 += x2*yy1[2];
  sum2 += x2*yy2[2];
case 2:
  x1    = x[1];
  sum0 += x1*yy0[1]; sum1 += x1*yy1[1];
  sum2 += x1*yy2[1];
case 1:
  x0    = x[0];
  sum0 += x0*yy0[0]; sum1 += x0*yy1[0];
  sum2 += x0*yy2[0];
case 0:
  x   += j_rem;
  yy0 += j_rem;
  yy1 += j_rem;
  yy2 += j_rem;
  j   -= j_rem;
  break;
}
while (j>0) {
  x0 = x[0];
  x1 = x[1];
  x2 = x[2];
  x3 = x[3];
  x += 4;

  sum0 += x0*yy0[0] + x1*yy0[1] + x2*yy0[2] + x3*yy0[3]; yy0+=4;
  sum1 += x0*yy1[0] + x1*yy1[1] + x2*yy1[2] + x3*yy1[3]; yy1+=4;
  sum2 += x0*yy2[0] + x1*yy2[1] + x2*yy2[2] + x3*yy2[3]; yy2+=4;
  j    -= 4;
}
z[0] = sum0;
z[1] = sum1;
z[2] = sum2;
```

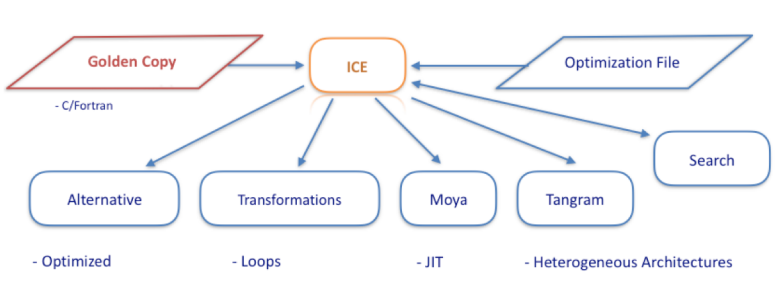
Practical Performance Optimization

- How to handle all the required optimizations together for many different scenarios?
- How to keep the code maintainable?
- How to find the best sequence of optimizations?

- Requirements
  - ”Golden Copy” code runs without ICE – do not require “buy in” to the system
  - Permit incremental adoption – apply ICE to subsets of the code, with subsets of tools
  - Coexist with other tools
  - Separate generation of optimized code from develop/run so that users do not need to install/run those tools. Allow tuning runs on “related” systems (e.g., x86 vectorization)
  - Support ways to find the best sequence of optimizations

Illinois Coding Environment (ICE)

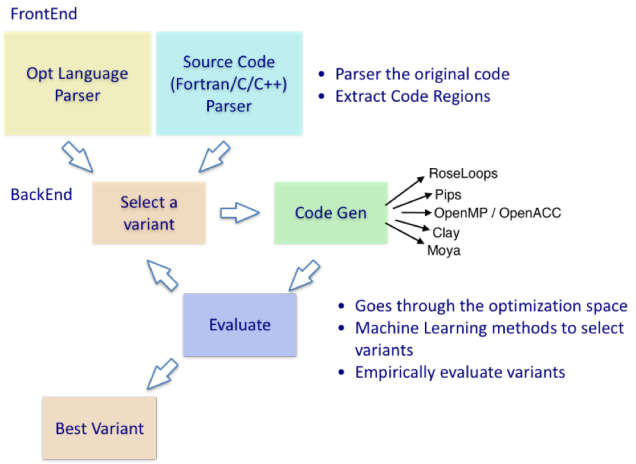
- One pragmatic approach
- Assumptions
  - Fast code requires some expert intervention
  - Can't all be done at compile time
  - Original code (in standard language) is maintained as reference
  - Can add information about computation to code
- Center for Exascale Simulation of Plasma-Coupled Combustion
  - http://xpacc.illinois.edu
  - ICE used to support "Golden Copy" code – version natural for computational scientist, without code optimizations
  - Used with primary simulation code, PlasCom2



- Approach
  - Annotations provide additional descriptive information
    - Block name, expected loop sizes, etc.
  - Source-to-source transformations used to create code for compiler
    - Exploit tool ecosystem – interface to existing tools
    - Original "Golden Copy" used for development, correctness checks
  - Database used to manage platform-specific versions; detect changes that invalidate transformed versions
    - Don't need to install/run transformation tools

ICE

- Source code is annotated to define code regions
- Optimization file notation orchestrates the use of the optimization tools on the code regions defined
- Interface provides operations on the Source code to invoke optimizations through:
  - Adding pragmas
  - Adding labels
  - Replacing code regions
- These operations are used by the interface to plug-in optimization tools
- Most tools are source-to-source
  - tools must understand output of previous tools



Matrix Multiplication Example

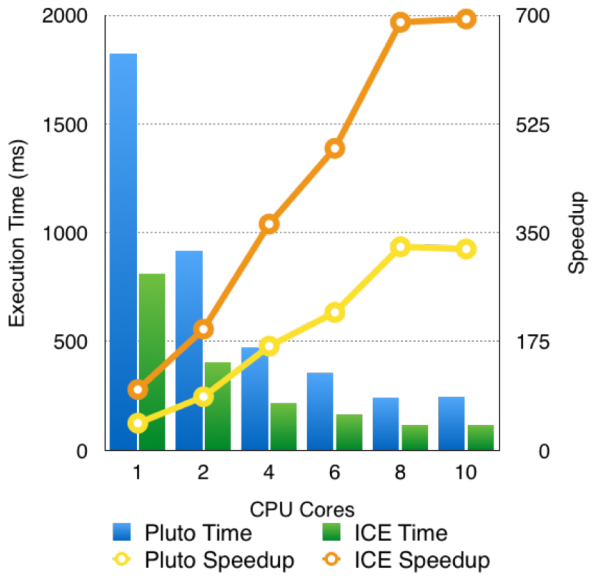
```
#pragma @ICE loop=matmul
      for (i=0; i<matSize; i++)
         for (j=0; j<matSize; j++) {
            for (k=0; k<matSize; k++) {
               matC[i][j] += matA[i][k] *
matB[k][j];
            }
         }
      }
```

+

```
—
# Built command before compilation
prebuildcmd:

# Compilation command before tests
buildcmd: make realclean;  make

#Command call for each test
runcmd: ./mmc

matmul:
   - Pips.tiling+:
 loop: 1
       factor: [2..512, 2..512, 2..512]
   - Pips.tiling+:
       loop: 4
       factor: [8, 16, 8]
   - OpenMP.OMPFor+:
loop: 1

…
```

```
#pragma omp parallel for schedule(static,1) private(i_t, k_t, j_t,i_t_t, k_t_t
,j_t_t, i, k,j)
 for (i_t = 0; i_t <= 127; i_t += 1)
   for (k_t = 0; k_t <= 127; k_t += 1)
   for (j_t = 0; j_t <= 3; j_t += 1)
   for (i_t_t = 4 * i_t; i_t_t <= ((4 * i_t) + 3); i_t_t += 1)
   for (k_t_t = 2 * k_t; k_t_t <= ((2 * k_t) + 1); k_t_t += 1)
   for (j_t_t = 32 * j_t; j_t_t <= ((32 * j_t) + 31); j_t_t += 1)
   for (i = 4 * i_t_t; i <= ((4 * i_t_t) + 3); i += 1)
   for (k = 8 * k_t_t; k <= ((8 * k_t_t) + 7); k += 1)
   for (j = 16 * j_t_t; j <= ((16 * j_t_t) + 15); j += 1)
   matC[i][j] += matA[i][k] * matB[k][j];
```

Matrix Multiplication Results



- Two levels of tiling + OpenMP
- Original version: 78,825 ms
- 98x speedup (1 core)
- 694x speedup (10 cores)
- Avg 2.2x speedup over Pluto

2048^2 ELEMENTS
ICC 17.0.1
INTEL E5-2660 V3
PLUTO PET BRANCH

Often Overlooked - IO Performance Often Terrible

- Applications just assume I/O is awful and can't be fixed
- Even simple patterns not handled well
- Example: read or write a submesh of an N-dim mesh at an arbitrary offset in file
- Needed to read input mesh in PlasComCM. Total I/O time less than 10% for long science runs (that is < 15 hours)
  - But long init phase makes debugging, development hard

	Original	Meshio	Speedup
PlasComCM	4500	1	4500
MILC	750	15.6	48

- Meshio library built to match application needs
- Replaces many lines in app with a single *collective* I/O call
- Meshio https://github.com/oshkosher/meshio
- Work of Ed Karrels

What Are Some of the Problems?

- POSIX I/O has a strong consistency model
  - Hard to cache effectively
  - Applications need to transfer block-aligned and sized data to achieve performance
  - Complexity adds to fragility of file system, the major cause of failures on large scale HPC systems
- Files as I/O objects add metadata "choke points"
  - Serialize operations, even with "independent" files
  - Do you know about O_NOATIME ?
- Burst buffers will *not* fix these problems – must change the semantics of the operations
- "Big Data" file systems have very different consistency models and metadata structures, designed for their application needs
  - Why doesn't HPC?
    - There have been some efforts, such as PVFS, but the **requirement** for POSIX has **held up** progress
- Real problem for HPC – user's "execution model" for I/O far from reality

Remember

- POSIX is not just “open, close, read, and write” (and seek …)
  - That’s (mostly) syntax
- POSIX includes strong semantics about concurrent accesses
  - Even if such accesses never occur
- POSIX also requires consistent metadata
  - Access and update times, size, …

No Science Application Code Needs POSIX I/O

- Many are single reader or single writer
  - Eventual consistency is fine
- Some are disjoint reader or writer
  - Eventual consistency is fine, but must handle non-block-aligned writes
- Some applications use the file system as a simple data base
  - Use a data base – we know how to make these fast and reliable
- Some applications use the file system to implement interprocess mutex
  - Use a mutex service – even MPI point-to-point
- A few use the file system as a bulletin board
  - May be better off using RDMA
  - Only need release or eventual consistency
- *Correct* Fortran codes do not require POSIX
  - Standard requires unique open, enabling correct and aggressive client and/or server-side caching
- MPI-IO would be better off without POSIX
  - Does not and never has required POSIX

The really hard part – Combining internode and Intranode programming systems

- Most common approach likely to be MPI + X
- What To Use as X in MPI + X?
  - Threads and Tasks
    - OpenMP, pthreads, TBB, OmpSs, StarPU, …
  - Streams (esp for accelerators)
    - OpenCL, OpenACC, CUDA, …
  - Alternative distributed memory system
    - UPC, CAF, Global Arrays, GASPI/GPI
  - MPI shared memory

What are the Issues?

- Isn’t the beauty of MPI + X that MPI and X can be learned (by users) and implemented (by developers) independently?
  - Yes (sort of) for users
  - No for developers
- MPI and X must either partition or share resources
  - User must not blindly oversubscribe
  - Developers must negotiate

More Effort needed on the “+”

- MPI+X won’t be enough for Exascale if the work for “+” is not done very well
  - Some of this may be language specification:
    - User-provided guidance on resource allocation, e.g., MPI_Info hints; thread-based endpoints, new APIs
  - Some is developer-level standardization
    - A simple example is the MPI ABI specification – users should ignore but benefit from developers supporting

Summary

- Challenges for Exascale programming are not just in scale
  - Need to achieve extreme power and cost efficiencies puts large demands on the effectiveness of single core (whatever that means) and single node performance
- MPI remains the most viable internode programming *system*
  - Supports a multiple parallel programming models, including one-sided and shared memory
  - Contains features for “programming in the large” (tools, libraries, frameworks) that make it particularly appropriate for the internode system
  - But some useful features still missing, especially WRT notification, and implementations don’t realize available performance
- Intranode programming for performance still an unsolved problem
  - Lots of possibilities, but adoption remains a problem
    - That points to unsolved problems, particularly in integration with large, multilingual codes
- Composition (e.g., MPI+X) is a practical approach
  - But requires close attention to “+”

Thanks!

- Philipp Samfass, Ed Karrels, Amanda Bienz, Paul Eller, Thiago Teixeira
- Luke Olson, David Padua
- Rajeev Thakur for runs on Theta
- Torsten Hoefler and Timo Schneider for runs on Piz Daint

- Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0002374
- ExxonMobile Upstream Research
- Blue Waters Sustained Petascale Project, supported by the National Science Foundation (award number OCI 07–25070) and the state of Illinois.
- Argonne Leadership Computing Facility