ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

MPI-3.0: A Response to New Challenges in hardware and Software

Torsten Hoefler

Invited plenary talk at Multicore Challenge 2012
Stuttgart, Germany, Sept. 2012

What is the Message Passing Interface?

- An open standard library interface for message passing, ratified by the MPI Forum
  - Versions: 1.0 ('94), 1.1 ('95), 1.2 ('97), 2.0 ('97), 1.3 ('08), 2.1 ('08), 2.2 ('09), 3.0 (probably '12)
- Common misconceptions:
  - MPI parallelizes your application
  - MPI is for distributed memory only
  - MPI (a library interface) is not scalable
  - MPI is fundamentally slower then PGAS etc.
  - MPI is a programming model
- Really, if you don't know what MPI is, you won't enjoy this talk ☺

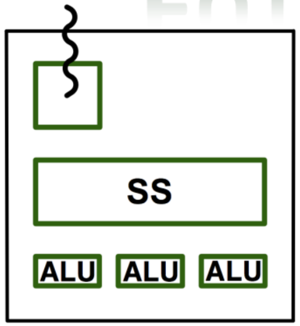
How did the MPI-3.0 Process Work

- Organization and Mantras of the MPI Forum:
  - Chapter chairs (convener) and (sub)committees
  - Avoid the “Designed by a Committee” phenomenon
    → standardize common practice
  - 99.5% backwards compatible
  - Final vote this week in Vienna!
- Adding new things:
  - Review and discuss early proposals in chapter
  - Bring proposals to the forum (discussion)
  - Plenary formal reading (usually word by word)
  - Two votes on each ticket (distinct meetings)
  - Final vote on each chapter (finalizing MPI-3.0)

THE MOST COMPLEX PART

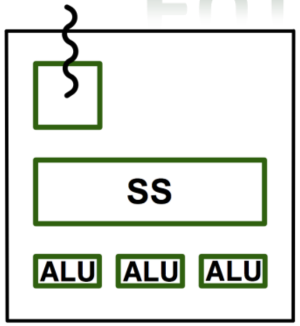
- MPI has been there since ~20 years
  - Likely to remain another 20 years
- MPI-1's design was future proof
  - Worked well for 15 years
- How will hardware look in 10 years from now?

Future Hardware Speculations

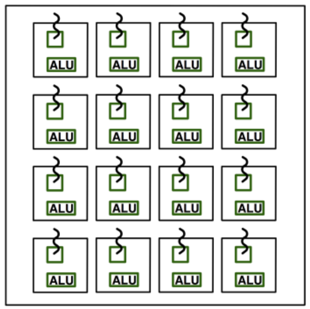


Only "Big Cores" (speed saturated, facing process problems)

FUTURE HARDWARE SPECULATIONS

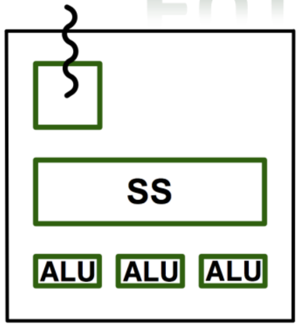


Only “Big Cores” (speed saturated, facing process problems)

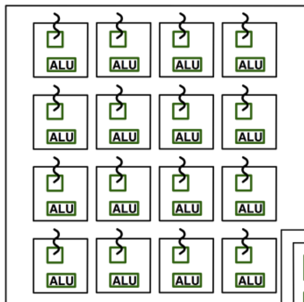


Only “Small Cores” (BlueGene Family, weak scaling is constrained by memory, Amdahl’s law)

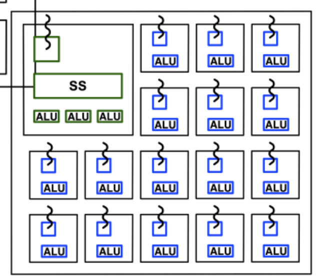
Future Hardware Speculations



Only “Big Cores” (speed saturated, facing process problems)

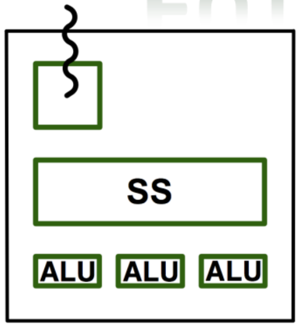


Only “Small Cores” (BlueGene Family, weak scaling is constrained by memory, Amdahl’s law)

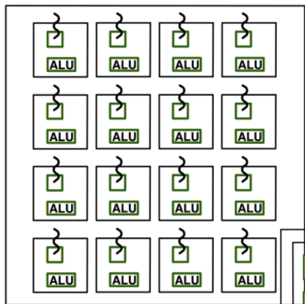


“Big & Small Cores” SoC (NVIDIA Echelon, DEEP, combine high speed and throughput)

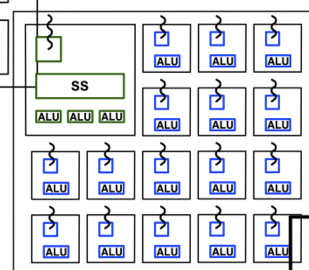
Future Hardware Speculations



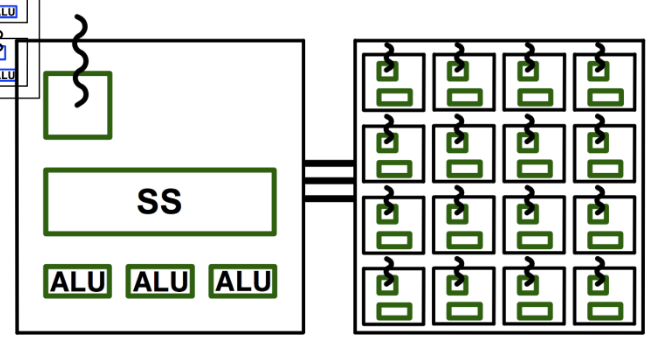
Only "Big Cores" (speed saturated, facing process problems)



Only "Small Cores" (BlueGene Family, weak scaling is constrained by memory, Amdahl's law)

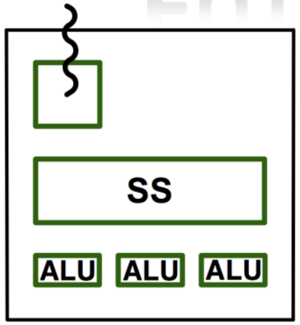


"Big & Small Cores" SoC (NVIDIA Echelon, DEEP, combine high speed and throughput)

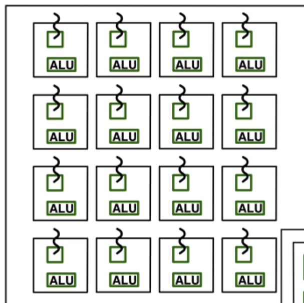


Accelerated Commodity
(GPUs, MIC, easy and cheap to build)
→ will probably be the mass market in the near future!

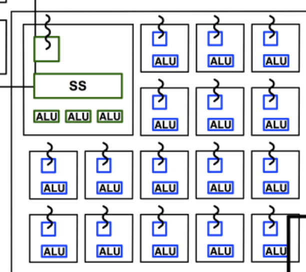
Future Hardware Speculations



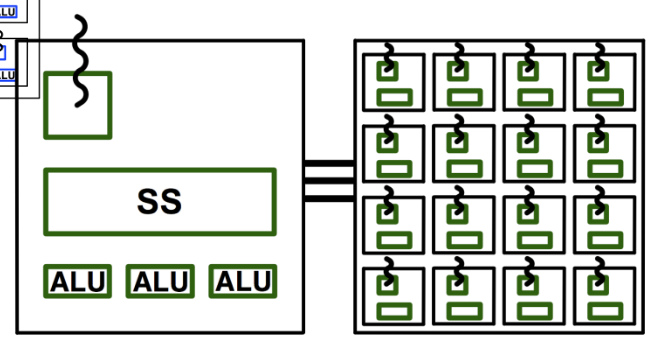
Only “Big Cores” (speed saturated, facing process problems)



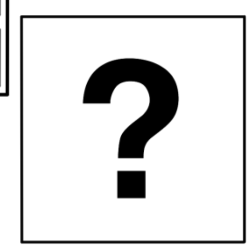
Only “Small Cores” (BlueGene Family, weak scaling is constrained by memory, Amdahl’s law)



“Big & Small Cores” SoC (NVIDIA Echelon, DEEP, combine high speed and throughput)



Accelerated Commodity
(GPUs, MIC, easy and cheap to build)
→ will probably be the mass market in the near future!



Something Completely Different? PIM?

Limits to Reality

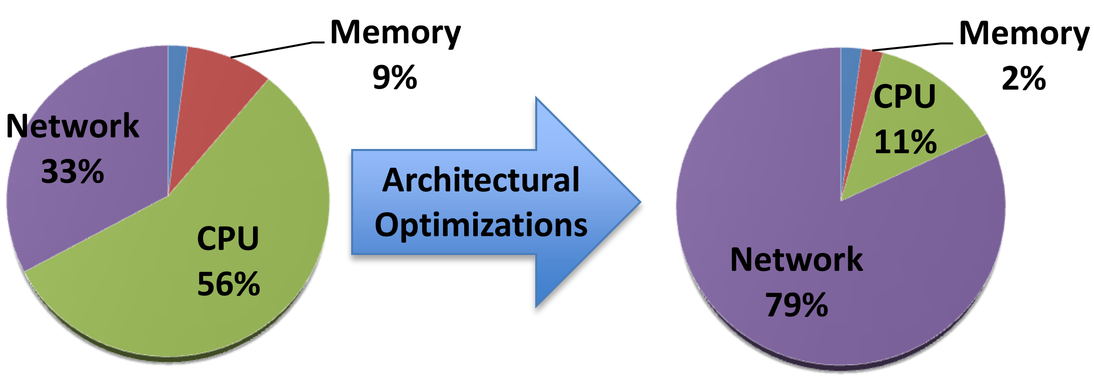
- Optimize performance constrained by
  - Purchasing cost (max. ~$200M)
  - Power (max. ~20 MW)
  - Programmer productivity (hard to measure)
- We may not be able to continue "as usual"
  - New hardware challenges!
  - Will discuss most significant challenges
  - Then we will discuss strategies to address them

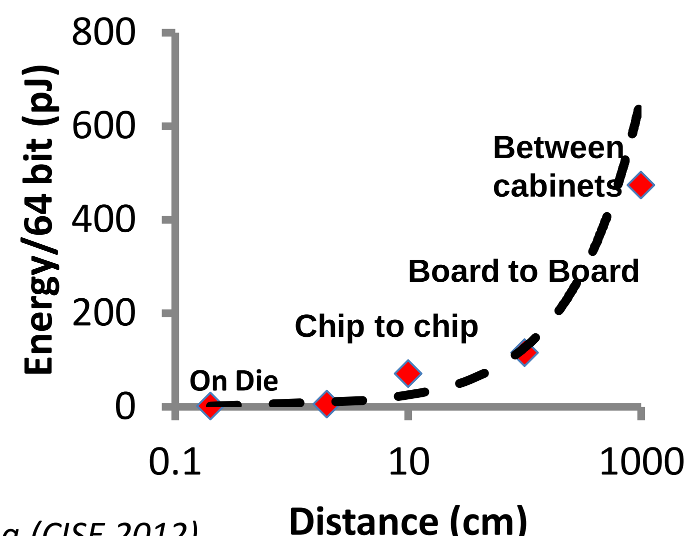
PRESENTATION OUTLINE

- Motivate five hardware challenges:
  - (1) Data Movement and Energy, (2) Failing Systems, (3) Complex Parallelism, (4) Hybrid Systems, (5) System Noise
- Show seven cross-cutting research topics:
  - (1) System Noise, (2) Parallelism and Networks, (3) Flops vs. Data Movement, (4) Self-Adaptation and Tuning, (5) User-Level Networking, (6) Hybrid Programming, (7) Fault Resiliency
  - And how they can be addressed with MPI-3.0
- My main goal: **inspire** young researchers!

Hardware Challenge #1: Data Movement

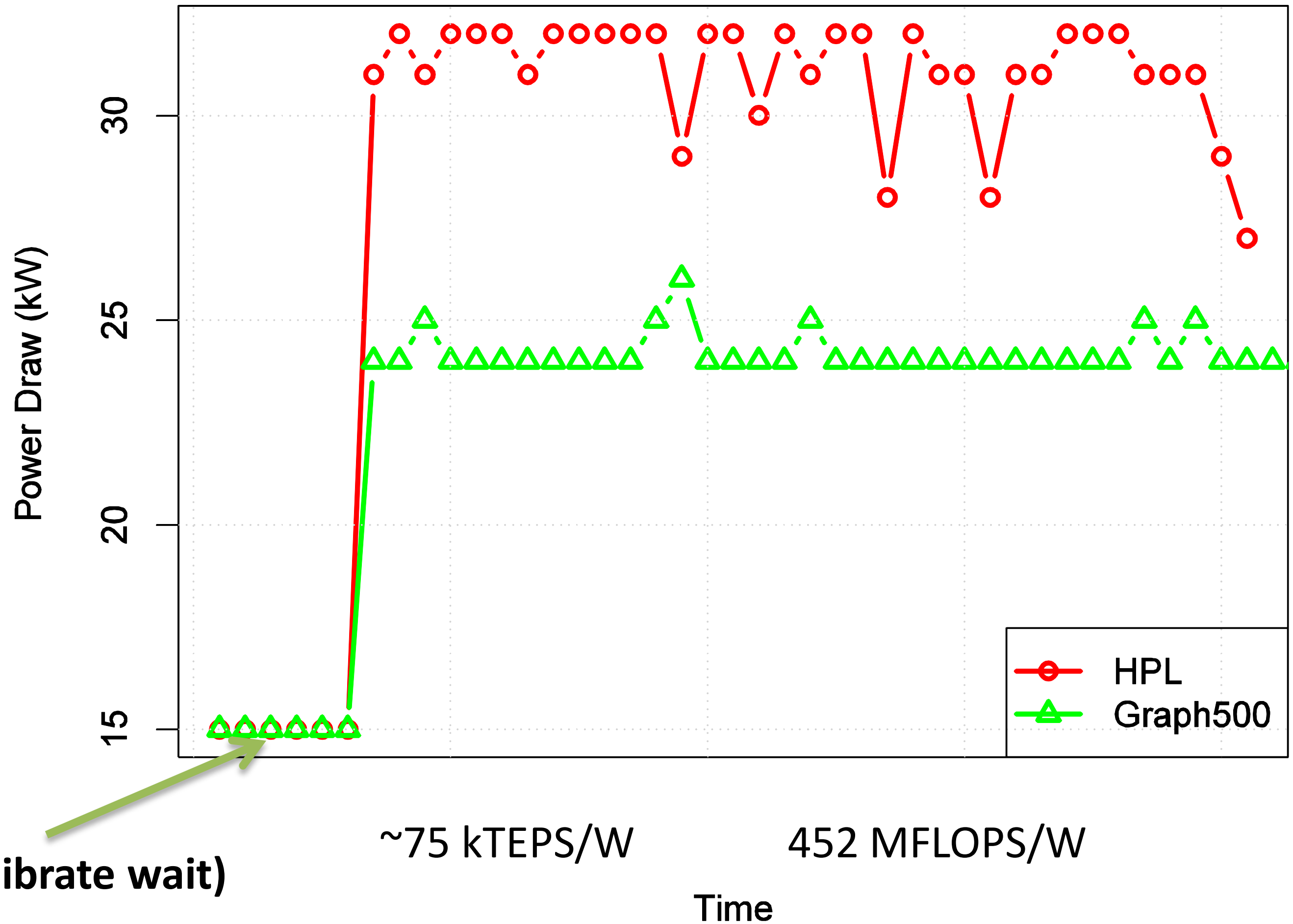
- Data movement will be most expensive
  - $E = P_{leak} \times T + E_{op} \times N + E_{byte} \times M$
    - Idle energy: 46% on today's commodity systems
    - Most networks draw constant power ☹
  - On-chip optics may change the game
    - But have high constant energy





Hoefler: Software and Hardware Techniques for Power-Efficient HPC Networking (CISE 2012)

Cray XE-6 Power Consumption



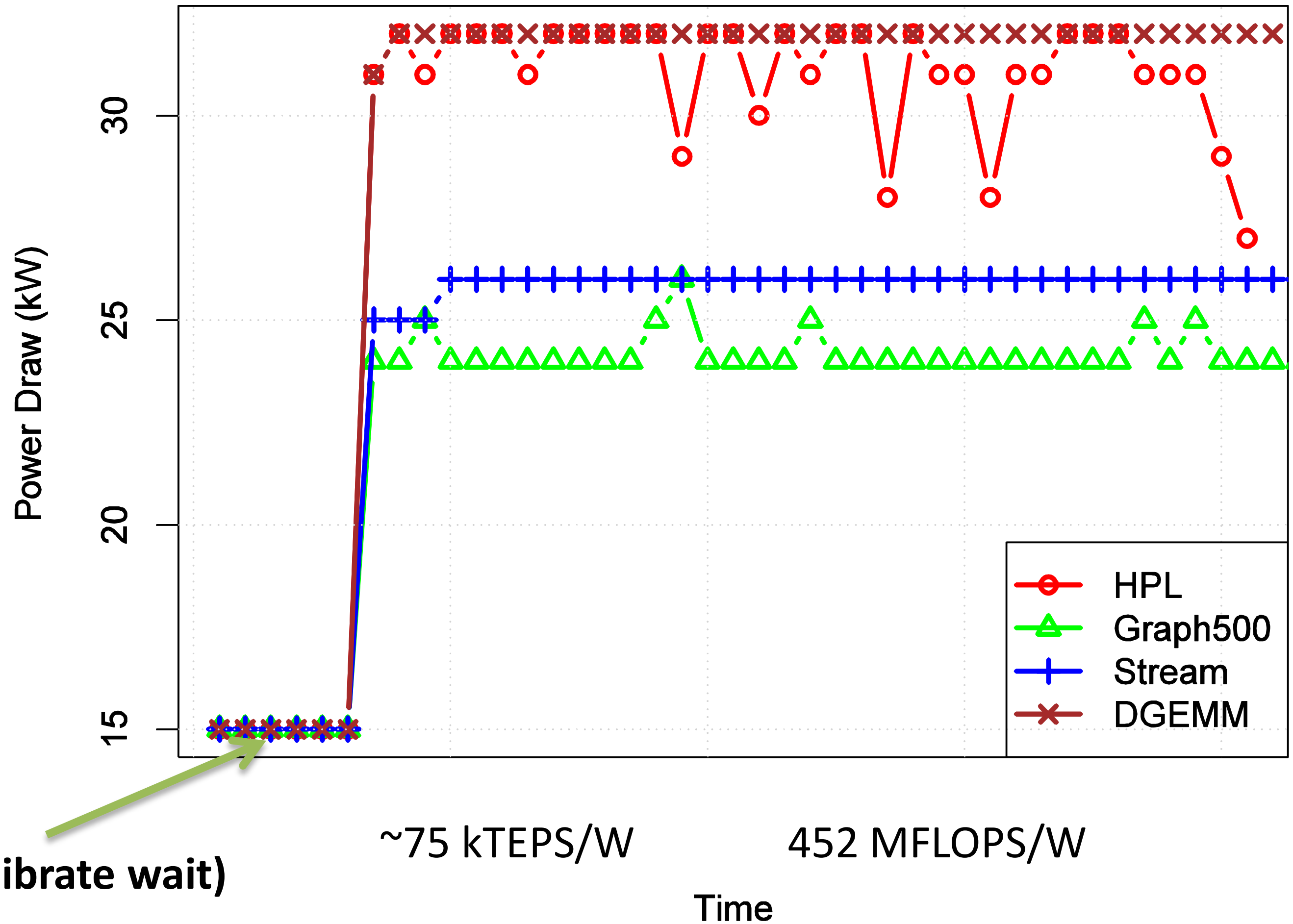
Scale=32

~75 kTEPS/W

452 MFLOPS/W

Idle (calibrate wait)

Cray XE-6 Power Consumption



Scale=32

~75 kTEPS/W

452 MFLOPS/W

Idle (calibrate wait)

Hardware Challenge #2: Failures

- Has been discussed as “blocker” for Petascale
  - Application-based checkpointing goes a long way!
  - May be a problem for Exascale?
  - Can be addressed in hardware (cf. ECC, IBM System z)
- Programming support would be great
  - Very hard problem!
  - → Distributed Consensus

Impossibility of **distributed consensus** with one faulty process
MJ Fischer, NA Lynch, MS Paterson - Journal of the ACM (JACM), 1985 - dl.acm.org
Abstract The **consensus** problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of ...
Cited by 3180 Related articles All 164 versions

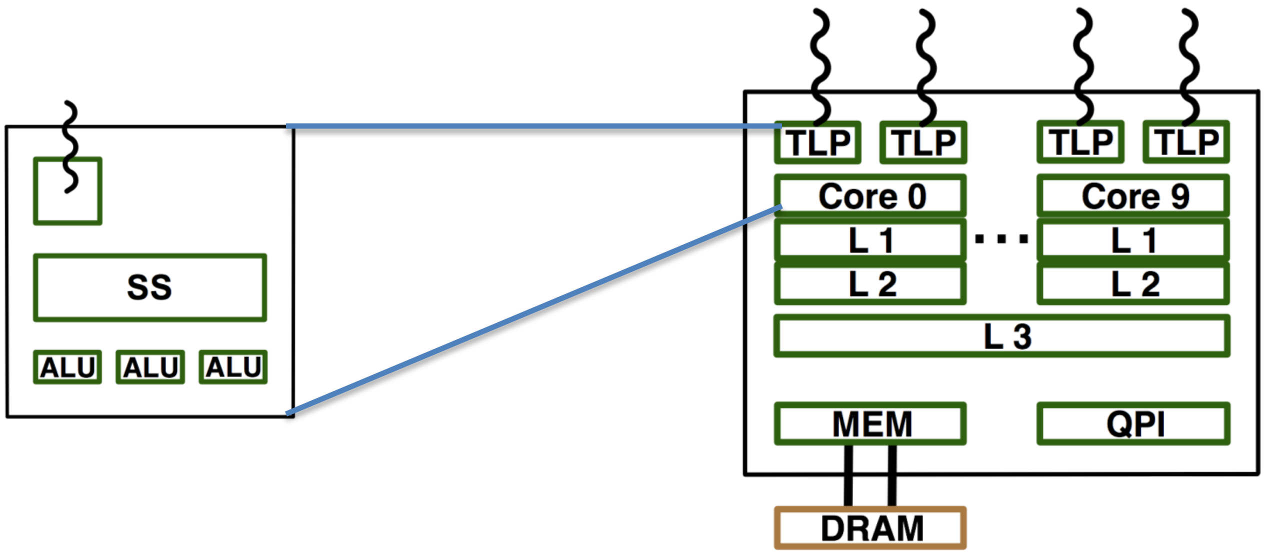
Distributed Consensus and Failure Detectors

- When one process fails, others cannot agree!
  - Unless they (collectively) declare the process dead
- Needs a failure detector!
  - Not trivial, several tradeoffs:
    - E.g., sporadic (with application messages) vs. periodic (using extra messages)
- May also rely on HW watchdogs
  - Or extra monitoring chips

Kharbas, Kim, Hoefler, Mueller: Assessing HPC Failure Detectors for MPI Jobs, PDP'12

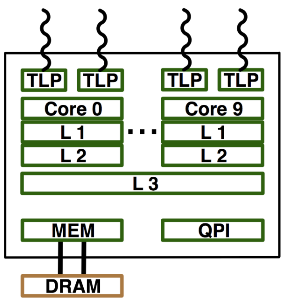
Hardware Challenge #3: Parallelism

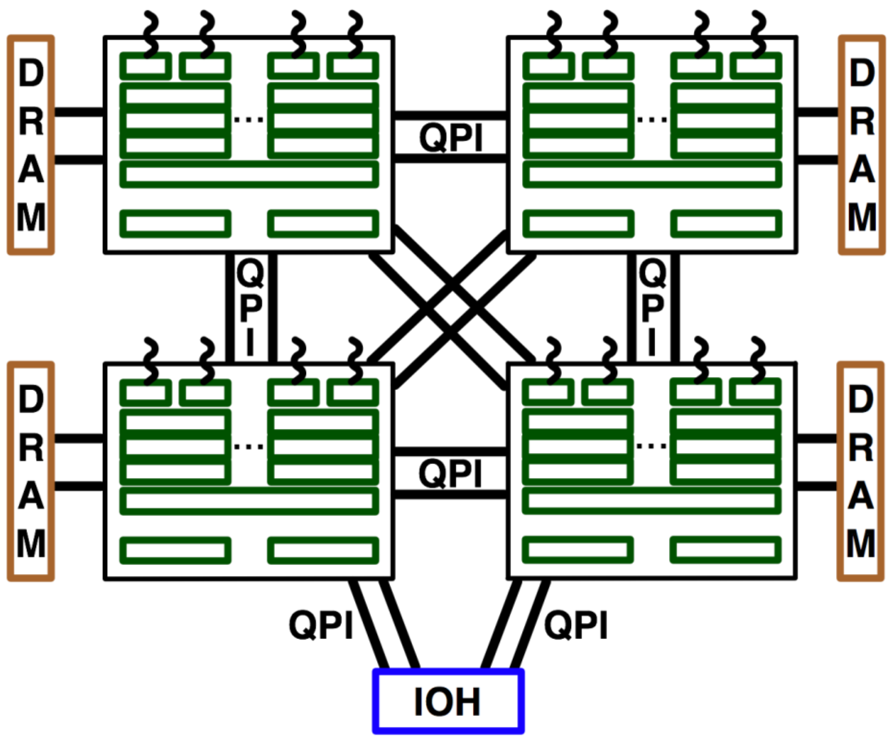
- Everything will be parallel:
  - Execution units, Pipelines, Vectors, CPU threads, Cores, Sockets, Nodes, Cabinets …
  - Intel Westmere MX CPU (10 cores):



Hardware Challenge #3: Parallelism

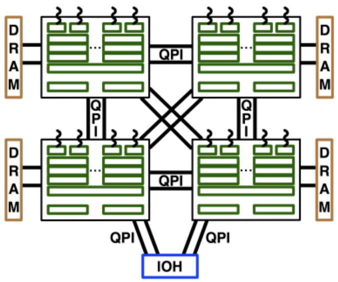
- Everything will be parallel:
  - Intel Westmere MX node (4 sockets):

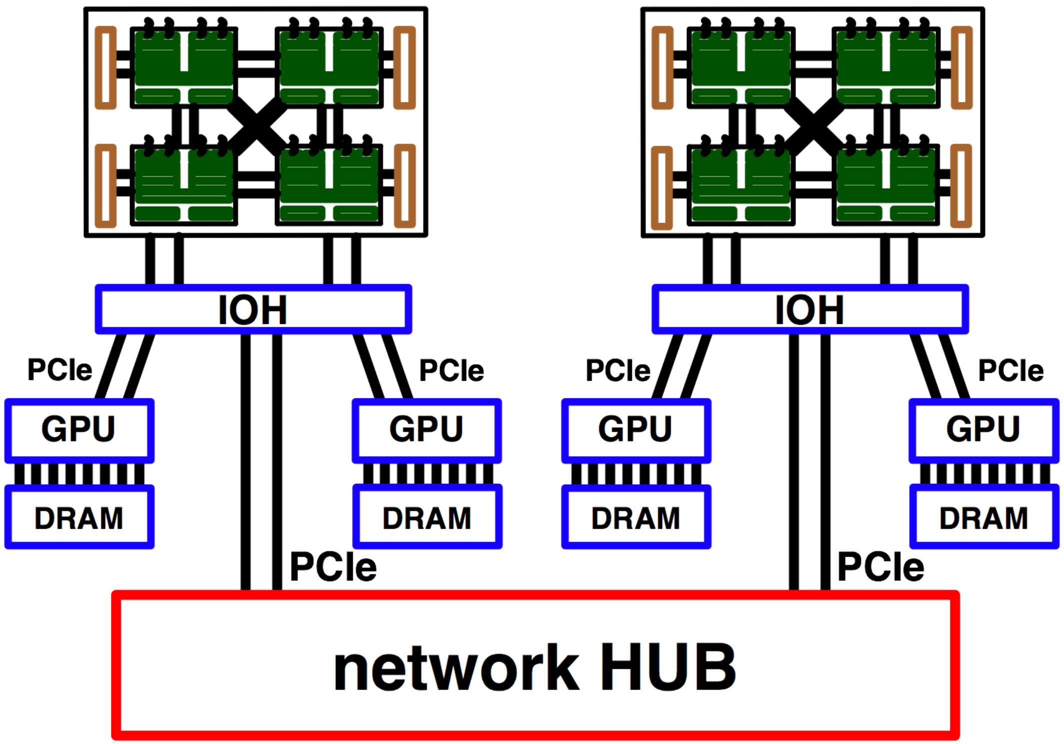




Hardware Challenge #3: Parallelism

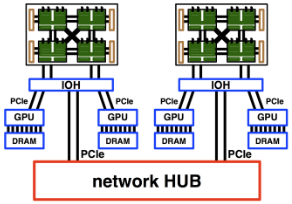
- Everything will be parallel:
  - Accelerated Intel Westmere MX board (2 nodes):

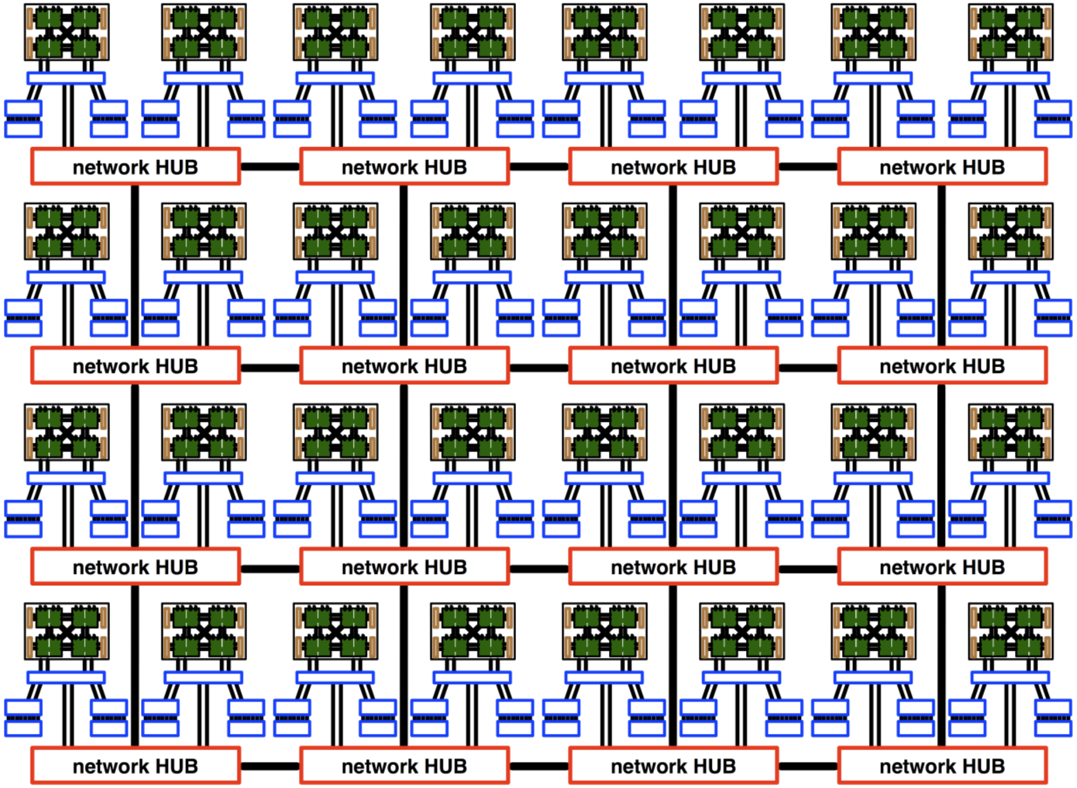




Hardware Challenge #3: Parallelism

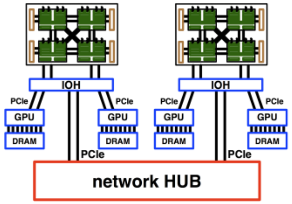
- Everything will be parallel:
  - Accelerated Intel Westmere MX network:

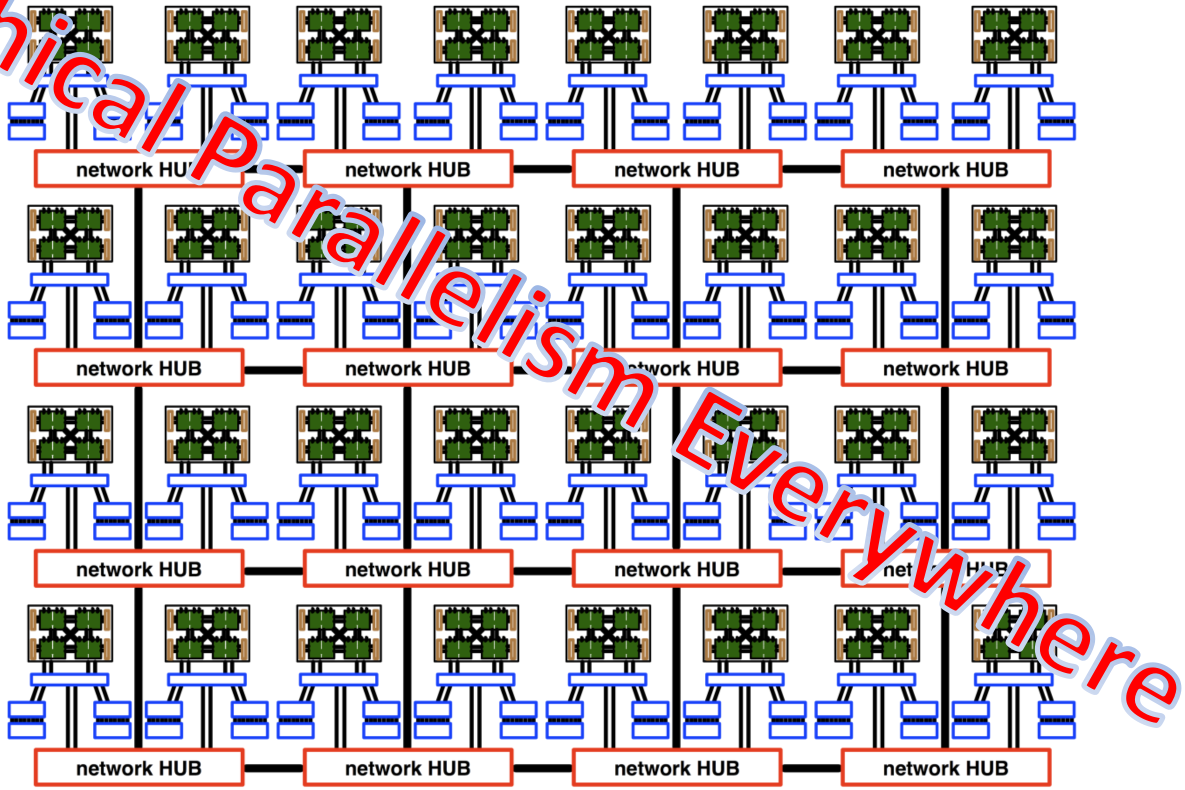




Hardware Challenge #3: Parallelism

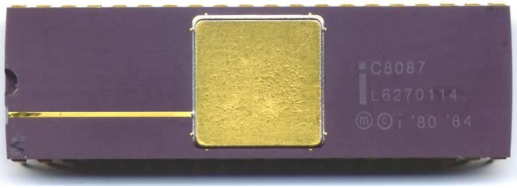
- Everything will be parallel:
  - Accelerated Intel Westmere MX network:



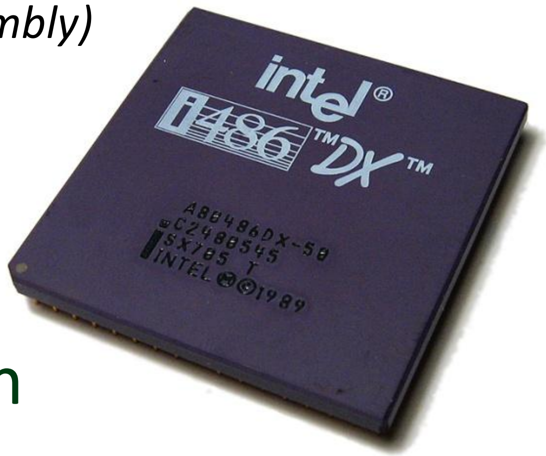


Hardware Challenge #4: Hybrid

- Systems will be hybrid
  - GPU, MIC, XYZ … we had this before: x87



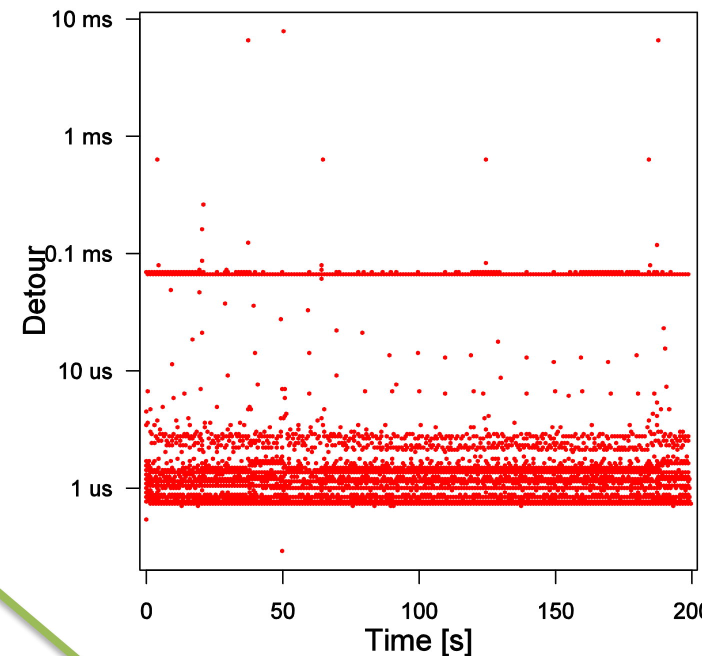
Intel's 8087, 1980, ~$150
5 MHz, 50 kF, 2.4 Watts
Special interface (F assembly)*



  - Nine years later: integrated FPU
    - Same instruction set/stream etc.
    - Transparent to programmer
  - MT units will be integrated … but can they be handled by a compiler/HW?
    - Unclear! Facing hard compiler problems!

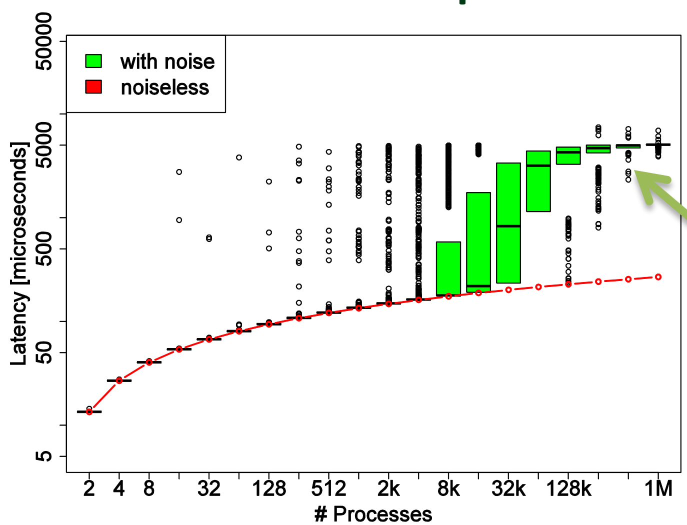
Hardware Challenge #5: Noise

- "System noise" is due to lost CPU cycles
  - Less than 0.02% overhead
  - Some noise cannot be avoided!
- Process synchronization may propagate noise to other procs.



Noise Signature

Allreduce on a Large-Scale System with noise!



Hoefler et al.: Characterizing the Influence of System Noise on Large-Scale Applications by Simulation, SC10

Software to the Rescue!

- It is possible to construct a large-scale machine!
  - But how to use/program it?
- From an MPI perspective:
  - Some challenges require new implementation techniques
  - Some challenges require new or extended interfaces (MPI-3.0)

- → hardware issues quickly turn into bigger software problems

Software *Design* Must Change

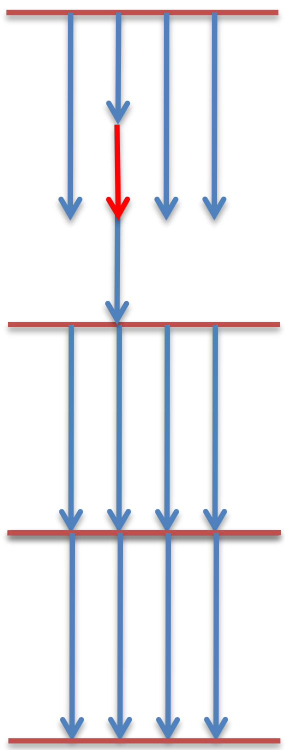
- Finally, since a long time …
  - MPI is trying to help but cannot always succeed
  - Many changes go up to an algorithmic level
- The following will address two target audiences:
  - Designers of scientific applications
    - How to cope with new challenges
  - Researchers in parallel processing
    - MPI's directions, interesting new research directions

Good Programming Abstractions

- A (parallel) programming model defines the user's view of the hardware
  - Has to be abstract (portable) but also needs to represent the machine (performance) model well
  - **and** easy to use ☺
- A good programming model:
  - Hides everything that it can hide (superscalar, pipeline, …)
  - Virtualizes everything else (vectorization, parallelism …)
  - We'll discuss things that cannot be hidden and how they can be handled in MPI
    - Attention: MPI is **not** a programming model!

Topic 1: System Noise

- Problem: noise propagation at large-scale (#5)
- Remedy: synchronization avoiding algorithms
  - Reduce synchronization
    - Not always possible
  - Relax synchronization
    - Nonblocking operations
  - Global synchronization
    - Nonblocking collective operations
    - Introduce synchronization windows that absorb noise



Nonblocking Collective Operations

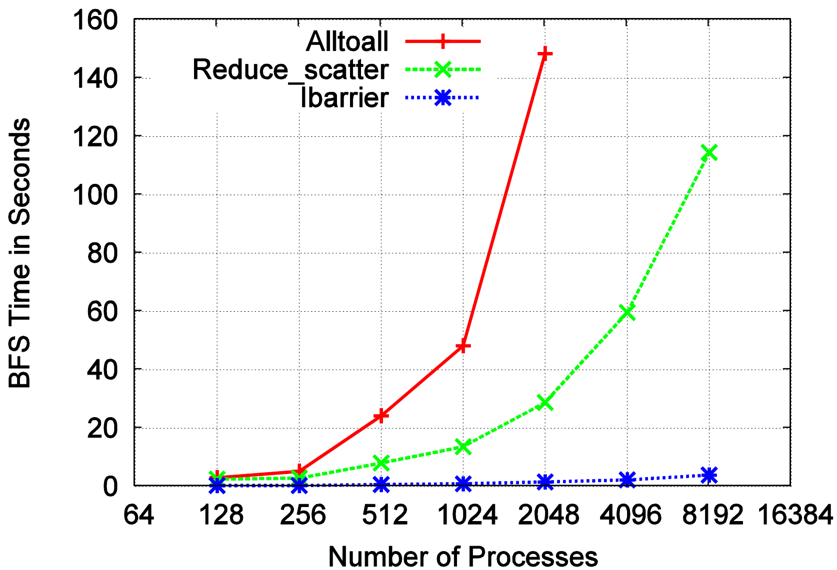
- E.g., MPI_Ibcast(…, &req); MPI_Wait(&req);
- Simple to understand, some things to note:
  - Requests are normal MPI_Requests, can be mixed
  - Progress is not guaranteed!
  - The init call will return independently of remote procs
  - All buffers (including arrays for vector colls) shall not be modified (or accessed) until the op completes
  - No matching with blocking collectives
  - Collectives must be called in order (as for threading)

Hoefler et al.: Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI, SC07

NBC Opportunities: DSDE

- NBC enable completely new algorithms!
  - → e.g., **D**ynamic **S**parse **D**ata **E**xchange
  - Process i has $k_{i,j}$ (0<i,j<P-1) items to send to process j, but no more than O(PlogP) $k_{i,j}$ are > 0 (sparse exchange)
- Protocols:
  - Alltoall
  - Reduce_scatter
  - Nonblocking Barrier

distributed level-wise BFS



Hoefler et al.: Scalable Communication Protocols for Dynamic Sparse Data Exchange, PPoPP'10

Topic 2: Parallelism and Networks

- Complex networks will be everywhere (#3)
  - Can be captured as a graph: $\mathcal{H} = (V_{\mathcal{H}}, C_{\mathcal{H}}, c_{\mathcal{H}}, \mathcal{R}_{\mathcal{H}})$
    - $V_{\mathcal{H}}$ set of physical nodes
    - $C_{\mathcal{H}}(u)$ number of PEs in node
    - $c_{\mathcal{H}}(u,v)$ link capacity (bandwidth) of link
    - $\mathcal{R}_{\mathcal{H}}$ set of routes (may be multiple routes from u to v)
  - Application topologies are simpler: $\mathcal{G} = (V_{\mathcal{G}}, \omega_{\mathcal{G}})$
    - $V_{\mathcal{G}}$ is the set of processes
    - $\omega_{\mathcal{G}}$ represents the communication volume
- How would you define an abstract interface?

Hoefler and Snir: Generic Topology Mapping Strategies for Large-scale Parallel Architectures, ICS'11

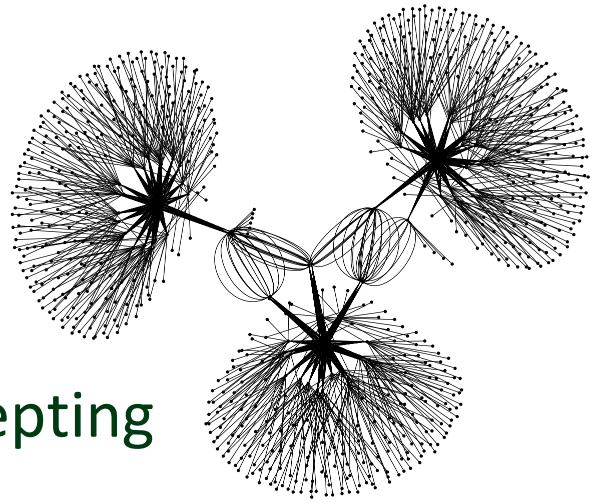
Topology Permutation Mapping

- Application topologies $\mathcal{G}$ are often only known during runtime
  - Often prohibits mapping before allocation
  - Topology-aware allocation → interesting research!
- MPI-2.2 defines interface for re-mapping
  - Scalable process topology graph
  - Permutes ranks in communicator
    - NP-hard problem ☹
  - Returns "better" permutation to the user
  - User needs to re-distribute data

Hoefler et al.: The Scalable Process Topology Interface of MPI 2.2, CCPE 2010

A Topology Mapping Library: LibTopoMap

- Implements the MPI-2.2 Topology Interface
  - Standard-compliant remapping of MPI applications
- Different Strategies:
  - Simple Greedy
  - Recursive Bisection
  - Hierarchical Multicore (partitioning)
  - Simulated Annealing / Threshold Accepting
  - SCOTCH Adapter
  - Graph Similarity (Reverse Cuthill McKee)
  - … and any combination of these

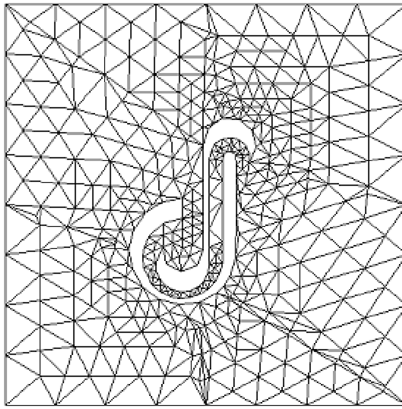


Network Graph of the
Deimos InfiniBand System

Hoefler and Snir: Generic Topology Mapping Strategies for Large-scale Parallel Architectures, ICS'11

Hiding Topology (A Programming Model)?

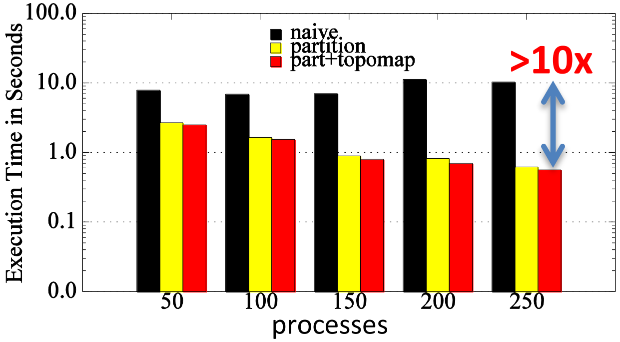
- Matrix Template Library - Linear Algebra
  - **Automatic** partitioning, load balancing, topology mapping, serial optimizations, neighborhood collectives



Parallel LU

```
for (std::size_t k= 0; k < num rows(LU)−1; k++) {
  if(abs(LU[k][k]) <= eps) throw matrix singular();
  irange r(k+1, imax); // Interval [k+1, n−1]
  LU[r][k] /= LU[k][k];
  LU[r][r] −= LU[r][k] * LU[k][r];
}
```

Single MatVec (ldoor)
Partitioning/Topology Mapping



Gottschling, Hoefler: “Productive Parallel Linear Algebra Programming […] “, CCGrid 2012

Topic 3: Flops vs. Data Movement

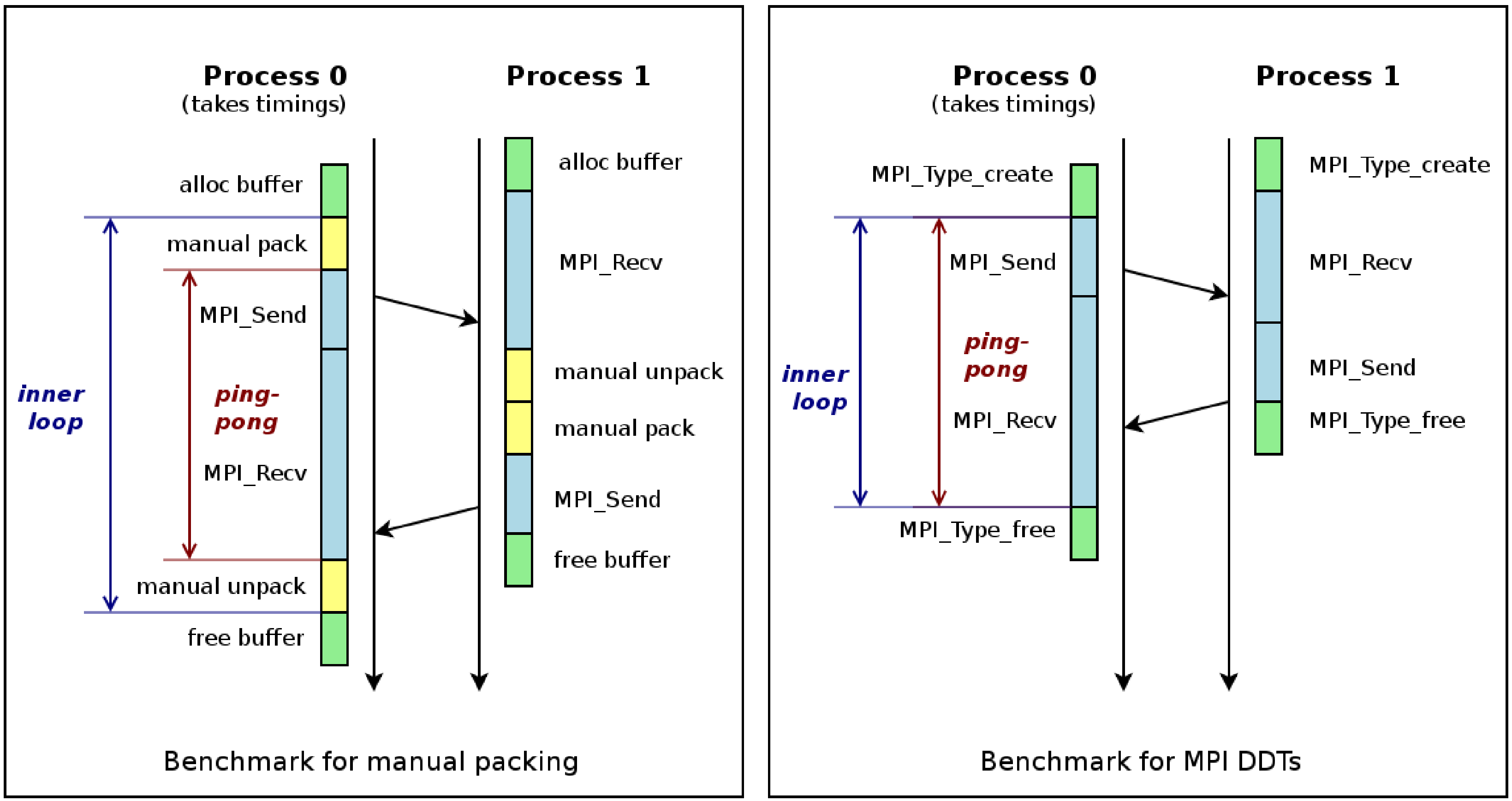
- Data movement will be most expensive (#1)
- Remedies:
  - Communication-reducing algorithms (Demmel et al.)
  - Mixed precision algorithms (Dongarra et al.)
  - Redundant computation (Curioni and others)
  - Topomapping for energy (libtopomap, cf. Topic 2)
  - Avoid extra copies (topic of today's discussion)

```
for(int i=0, j=0; i<N, i+=stride, j++)
  buf[j] = A[i]
MPI_Send(buf, N, MPI_DOUBLE, ...)
```

```
MPI_Recv(buf, N, MPI_DOUBLE, ...)
for(int i=0, j=0; i<N, i+=stride, j++)
  A[i] = buf[j]
```

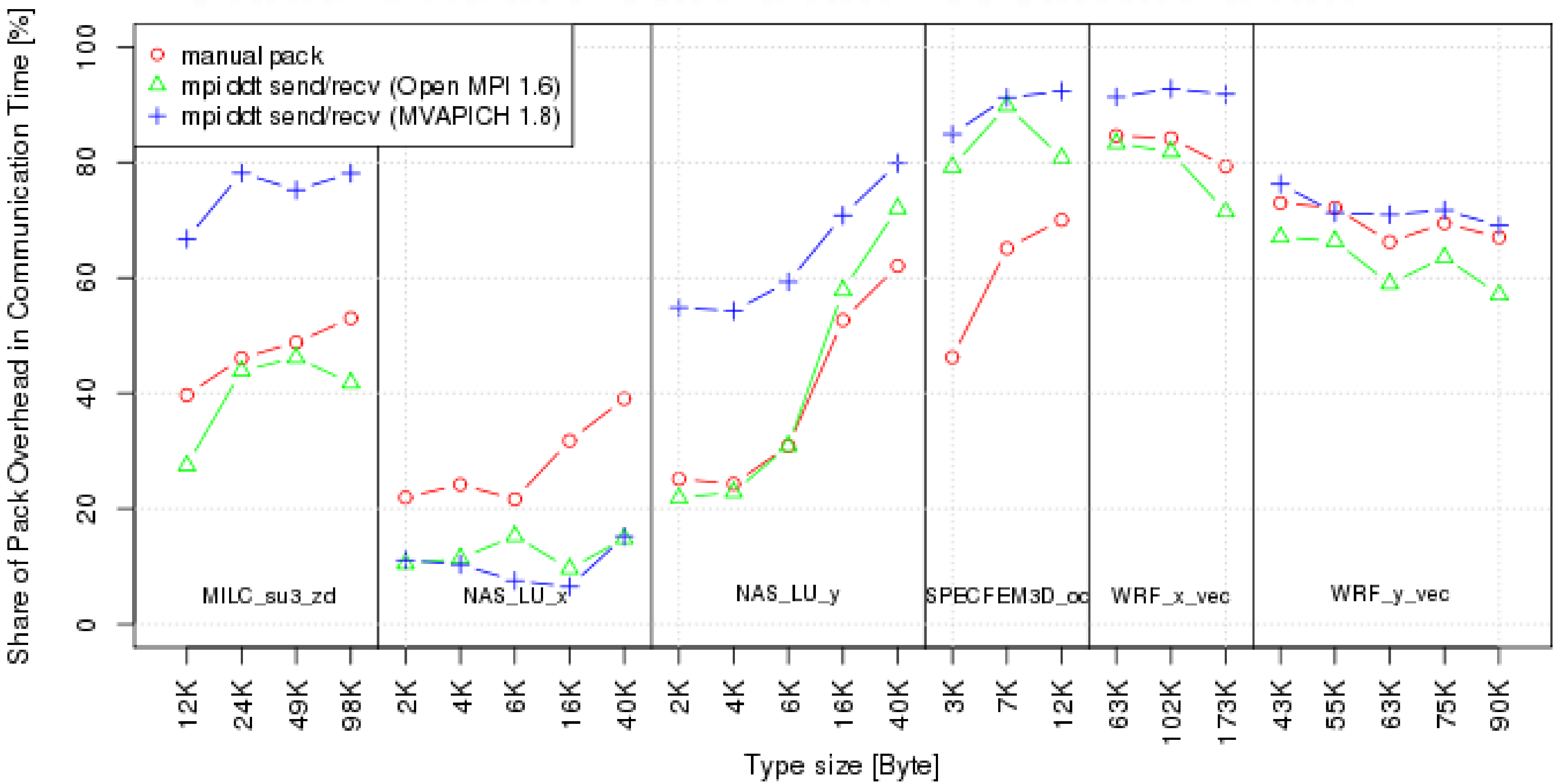
THE FORGOTTEN BYTES IN COMMUNICATIONS

- Think of a new ping-pong benchmark:



Schneider, Gerstenberger, Hoefler: Micro-Applications for Communication Data Access Patterns, EuroMPI 2012

TIME SPENT PACKING/UNPACKING



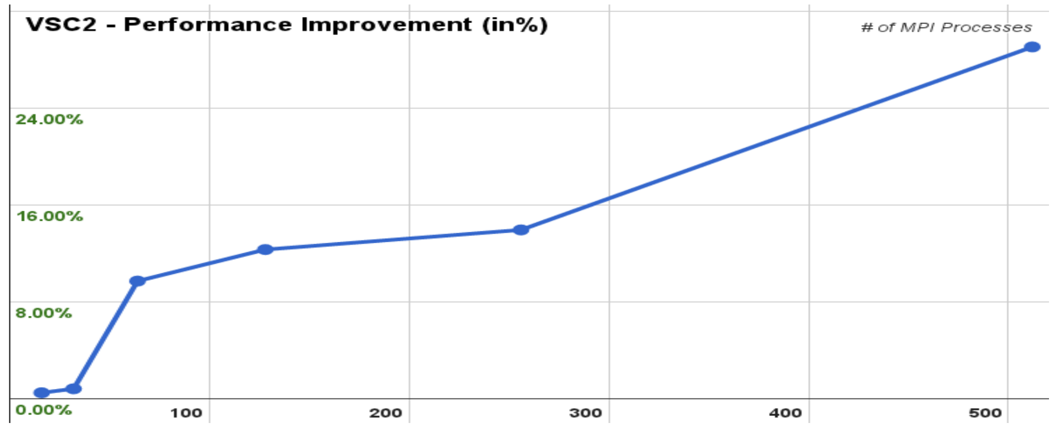
Schneider, Gerstenberger, Hoefler: Micro-Applications for Communication Data Access Patterns, EuroMPI 2012
Hoefler, Gottlieb: Parallel Zero-Copy Algorithms for FFT and Conjugate Gradient using MPI Datatypes, EuroMPI 2010

Topic 4: Self-Adaptation and Tuning

- Architectures are too complex for analytic tuning (#2, #3, #4) → empiric tuning
- Two options:
  - Tune MPI applications
    - E.g., move send/recv to maximize cache reuse
    - Requires static analysis of application code
  - Tune MPI libraries
    - E.g., change communication patterns to match architecture/topology
    - Requires high-level specification in application codes

MPI Static Analysis

- Compiled MPI project
  - With LLNL (Bronevetsky, Quinlan), IU (Lumsdaine)
  - In collaboration with S. Pellegrini and T. Fahringer
- Transform blocking MPI calls in nonblocking
  - Static for now, but exposes tuning parameters!
  - First results: up to 28% speedup!



Credits: S. Pellegrini

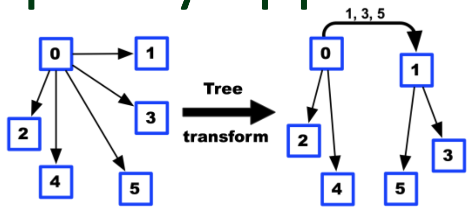
Pellegrini et al.: Exact Dependence Analysis for Increased Communication Overlap, EuroMPI'12

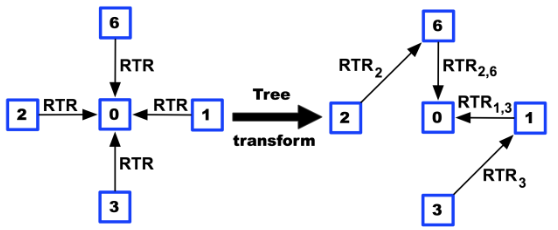
NEIGHBORHOOD COLLECTIVES

- MPI-3.0 allows to create arbitrary collectives
  - "User-defined collective communication"
  - Cf. MPI Datatypes
- Communication along a virtual topology
  - MPI_Neighbor_allgather() – same buffer to all
  - MPI_Neighbor_alltoall() – personalized send buffer
  - No user-defined reductions (yet!)
- Benefits:
  - Simplifies programming
  - Numerous optimization possibilities
  - Fits many applications (stencil, grid etc.)

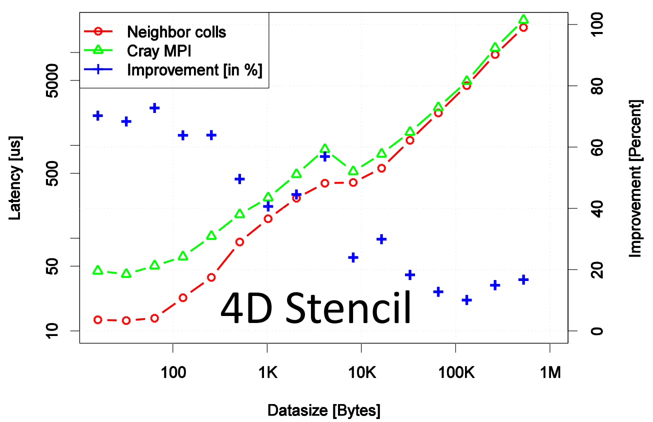
Optimizing Neighborhood Collectives

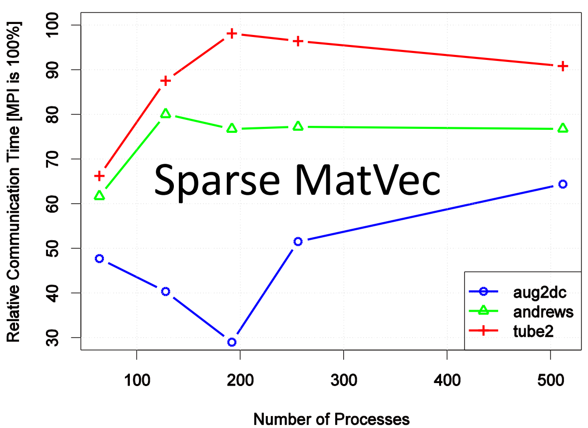
- Use principles known from traditional collectives
  - Specify application persistence in comm_create





  - Some relevant optimization results:





Hoefler, Schneider: Optimization Principles for Collective Neighborhood Communications, SC12

Topic 5: User-Level Networking

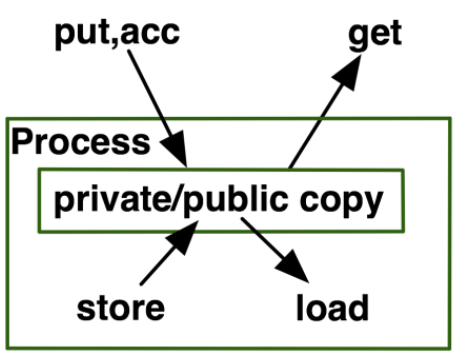
- Cannot afford kernel calls or additional copies (#1)
  - True since a while ("zero copy")
  - RDMA-capable networks (most of them are)
  - Programmed as a PGAS model
  - MPI-2 One-Sided had some issues
- → New MPI-3.0 One Sided Communications
  - Complex topic, see full MPI-3.0 tutorials at http://www.unixer.de/teaching/mpi_tutorials/

MPI-3.0 One Sided Overview

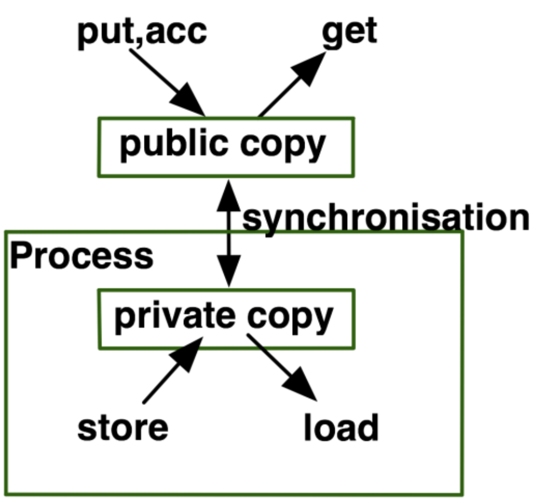
- Creation
  - Expose memory collectively - Win_create
  - Allocate exposed memory – Win_allocate
  - Dynamic memory exposure – Win_create_dynamic
- Communication
  - Data movement (put, get, rput, rget)
  - Accumulate (acc, racc, get_acc, rget_acc, fetch&op, cas)
- Synchronization
  - Active - Collective (fence); Group (PSCW)
  - Passive - P2P (lock/unlock); One epoch (lock _all)

MPI-3.0 One Sided Memory Models

- MPI offers two memory models:
  - Unified: public and private window are identical
  - Separate: public and private window are separate
- Type is attached as attribute to window
  - MPI_WIN_MODEL



MPI_UNIFIED



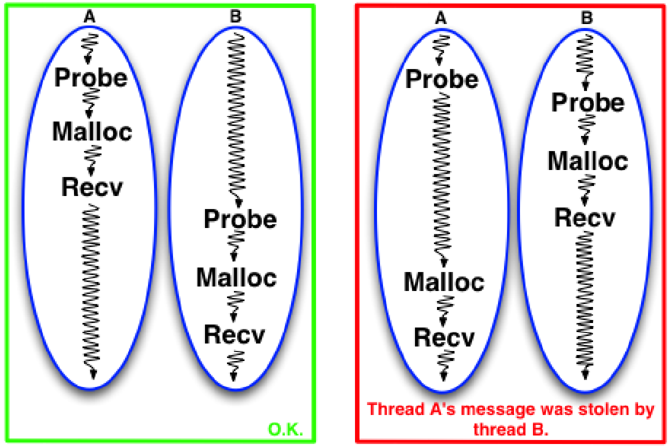
MPI_SEPARATE

Topic 6: Hybrid Programming

- Hybrid systems (multicore, accelerator) dominate (#4)!
- Multicore message-passing issues:
  - Threaded message passing (Mprobe)
  - On-node memory sharing
- Accelerator issues:
  - Separate address spaces (maybe?)
  - Memory copying (maybe?)

Thread-Safe Matched Probe

- MPI-2.2 point-to-point communication is not thread safe!




```
MPI_Probe(..., status)
size = get_count(status)*size_of(datatype)
buffer = malloc(size)
MPI_Recv(buffer, ...)
```

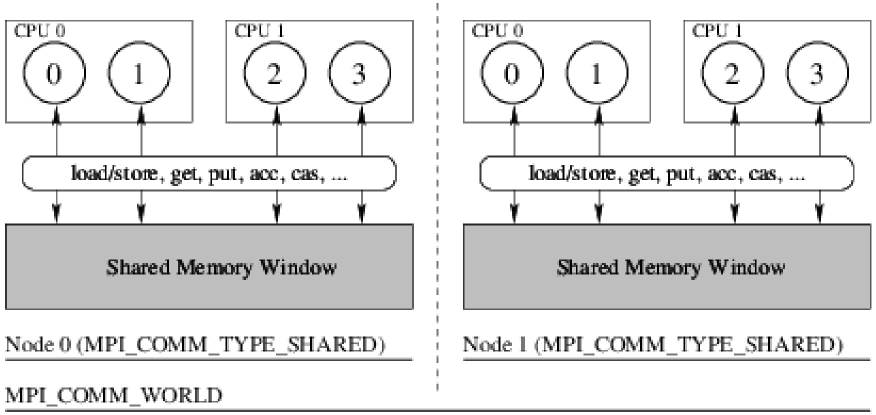
```
MPI_Mprobe(..., msg, status)
size = get_count(status)*size_of(datatype)
buffer = malloc(size)
MPI_Mrecv(buffer, ..., msg, ...)
```

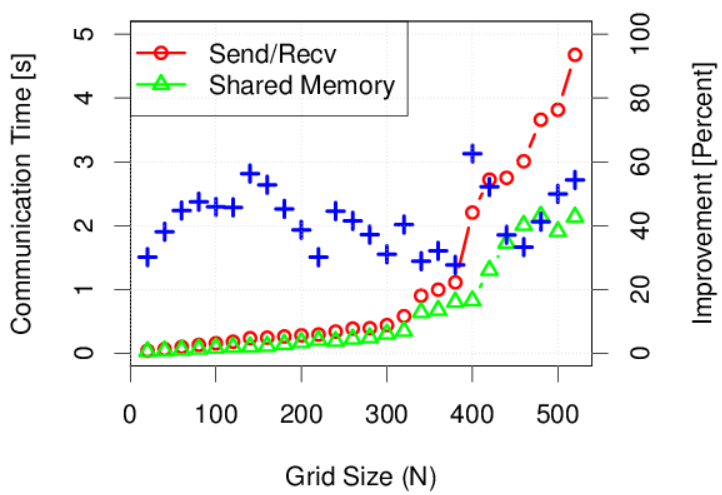
- Easy to fix: return a message handle!
  - Receive this message only through the handle
  - Easier to use and faster!

Hoefler et al.: Efficient MPI Support for Advanced Hybrid Programming Models, EuroMPI'10

Shared Memory Windows

- MPI-3.0 allows to create windows of shared memory (all processes have load/store access)
  - MPI_Comm_split_type() creates communicators
  - MPI_Win_alloc_shared() creates shared window
    - Allows direct load/store and all RMA accesses





Hoefler et al.: Leveraging MPI's One-Sided Communication Interface for Shared-Memory Programming, EuroMPI'12

Topic 7: Fault Resiliency

- MPI-2.2 makes fault resiliency a matter of quality of implementation
  - No guarantees, no standard but possible!
- So runtime may stay up in case of a crash-fault
  - Failure-detectors are possible
  - Communication functions can return appropriate errors (or invoke error handlers etc.)
- How can a code recover from a crash-fault?
  - Re-create or repair a communicator?

Gropp, Lusk: Fault Tolerance in MPI Programs, IJHPCA 2002

Noncollective Communicator Creation

- Cumbersome communicator repair in MPI-2.2
  - Or just live with holes and without collectives!
- MPI_Comm_create_group() allows to:
  - Allow to create communicators without involving all processes in the parent communicator
  - Very useful for some applications (dynamic sub-grouping) or fault tolerance (dead processes)

J. Dinan et al.: Noncollective Communicator Creation in MPI, EuroMPI'11

Summary and Conclusions

- The future will be exciting!
  - Frequency scaling comes to a halt → optimizations become more important!
  - Specialized hardware/accelerators can gain market share (even with "older" process technology)
- MPI is prepared for most likely scenario
  - Forms a stable baseline to go forward
    - Integrates with accelerators and multicore
  - Interesting research opportunities
    - For application and middleware developers
  - Some problems remain … MPI development continues!

ACKNOWLEDGMENTS

- The MPI Forum
  - Especially the collective and RMA WGs!
  - All co-authors (listed separately) and many others!

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

NSF

BLUE WATERS
SUSTAINED PETASCALE COMPUTING

DEPARTMENT OF ENERGY · UNITED STATES OF AMERICA

Sandia National Laboratories