Non-blocking Collective Operations for MPI

- Towards Coordinated Optimization of Computation and Communication in Parallel Applications -

Torsten Hoefler

Open Systems Lab
Indiana University
Bloomington, IN, USA

Lawrence Livermore National Lab
Livermore, CA, USA
25th August 2008

Outline

Outline

Fundamental Assumptions (I)

We need more powerful machines!

- Solving real-world scientific problems needs huge processing power (more than available)

Capabilities of single PEs have fundamental limits

- The scaling/frequency race is currently stagnating
- Moore's law is still valid (number of transistors/chip)
- Instruction level parallelism is limited (pipelining, VLIW, multi-scalar)

Explicit parallelism seems to be the only solution

- Single chips and transistors get cheaper
- Implicit transistor use (ILP, branch prediction) have their limits

Fundamental Assumptions (II)

Parallelism requires communication

- Local or even global data-dependencies exist
- Off-chip communication becomes necessary
- Bridges a physical distance (many PEs)

Communication latency is limited

- It's widely accepted that the speed of light limits data-transmission
- Example: minimal 0-byte latency for $1m \approx 3.3ns \approx 13$ cycles on a $4GHz$ PE

Bandwidth can hide latency only partially

- Bandwidth is limited (physical constraints)
- The problem of "scaling out" (especially iterative solvers)

Assumptions about Parallel Program Optimization

Collective Operations

- Collective Operations (COs) are an optimization tool
- CO performance influences application performance
- optimized implementation and analysis of CO is non-trivial

Hardware Parallelism

- More PEs handle more tasks in parallel
- Transistors/PEs take over communication processing
- Communication and computation could run simultaneously

Overlap of Communication and Computation

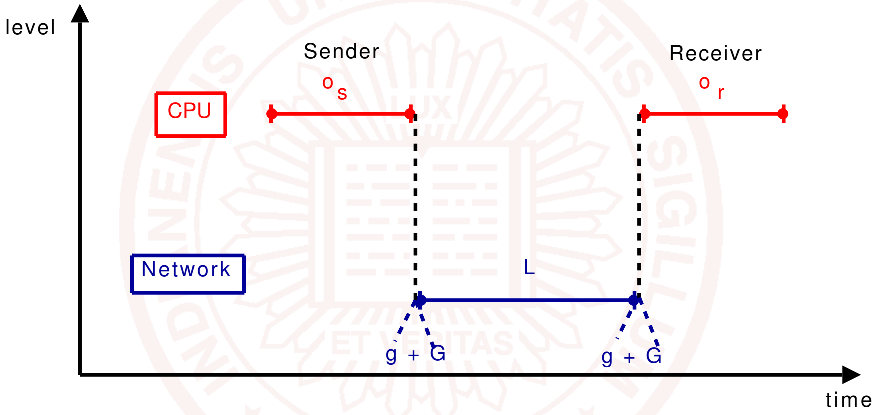
- Overlap can hide latency
- Improves application performance

We need more (functional) parallelism in our algorithms and codes!

This is hard work!

So, how much can we gain?

The LogGP Model



$\Rightarrow$ sending message of size s: $L + 2 \cdot o + (s - 1) \cdot G$

Resulting Interconnect Trends

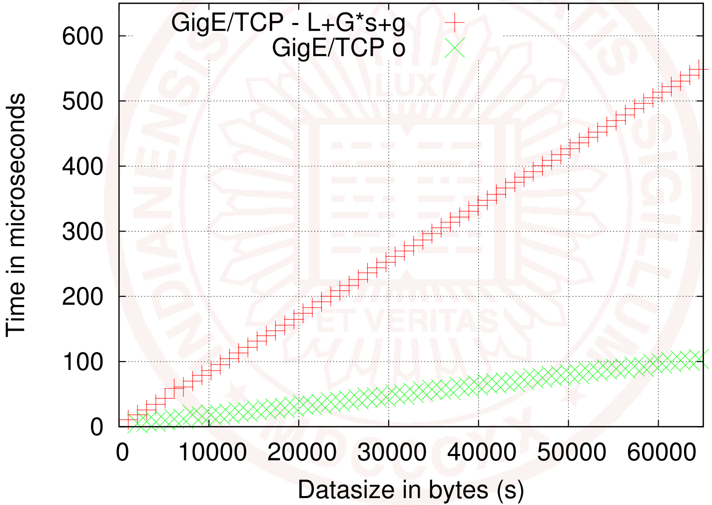
Ongoing Technology Change

- modern interconnects offload communication to co-processors (Quadrics, InfiniBand, Myrinet)
- Ethernet is optimized for lower overhead (e.g., Gamma)
- many Ethernet adapters support protocol offload

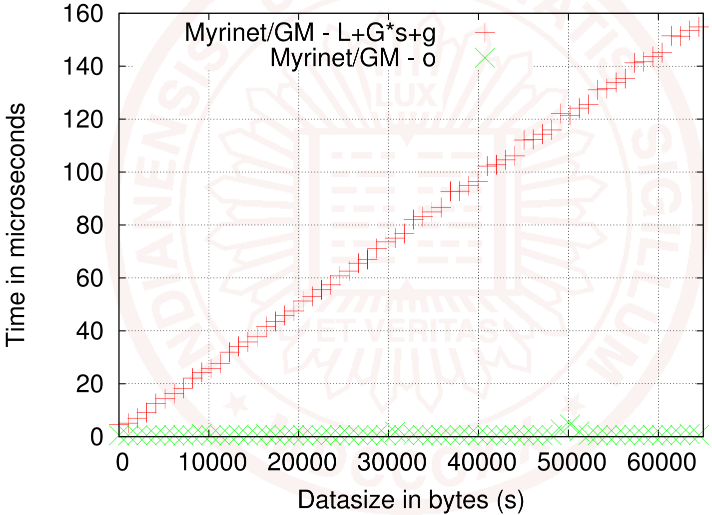
$$\Rightarrow L + g + m \cdot G >> o$$

$\Rightarrow$ we prove our expectations with benchmarks of the user CPU overhead

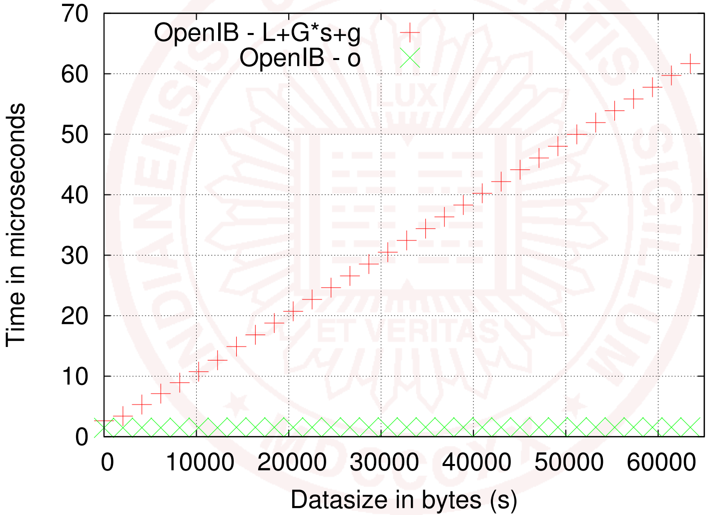
LogGP Model Examples - GigE/TCP



LogGP Model Examples - Myrinet/GM



LogGP Model Examples - InfiniBand/OpenIB



Outline

Isend/Irecv is there - Why Collectives?

- Gorlach, ’04: “Send-Receive Considered Harmful”
- $\Leftrightarrow$ Dijkstra, ’68: “Go To Statement Considered Harmful”

point to point

```
if (rank == 0) then
   call MPI_SEND(...)
else
   call MPI_RECV(...)
end if
```

vs. collective

```
call MPI_GATHER(...)
```

$\Rightarrow$ cmp. math libraries vs. loops

Sparse Collectives/Topological Collectives

"But my algorithm needs nearest neighbor communication!?"
⇒ this is a collective too, just sparse (cf. sparse BLAS)

- sparse communication with neighbors on process topologies
- graph topology can make it generic
- many optimization possibilities (process placing, overlap, message scheduling/forwarding)
- easy to implement
- not part of MPI but fully implemented in LibNBC and proposed to the MPI Forum

⇒ give MPI details about you communication pattern!

Performance Benefits of (Non-Blocking) Collectives

Blocking/Non-Blocking - Abstraction

- abstraction enables optimizations
- ease of use, avoids implementation errors
- performance portability

Non-Blocking - Overlap

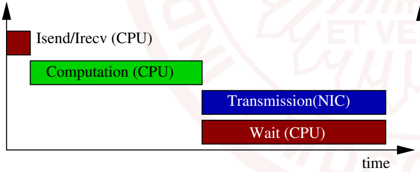
- leverage hardware parallelism (e.g. InfiniBand™)
- overlap similar to non-blocking point-to-point

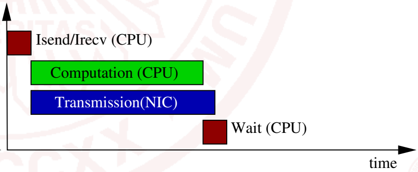
Non-Blocking - Pseudo Synchronization

- avoidance of explicit pseudo synchronization
- limit the influence of OS noise

Quantifying the Benefits - With LogGP

- collectives scale typically with $O(log_2 P)$ or $O(P)$ sends
- "wasted" CPU time: $log_2 P \cdot (L + (s-1) \cdot G)$
  - Gigabit Ethernet: L = 15-20 μs
  - InfiniBand: L = 2-7 μs
  - $1 \mu s \approx$ 6000 FLOP on a 3GHz Machine
- synchronization overhead not easy to assess

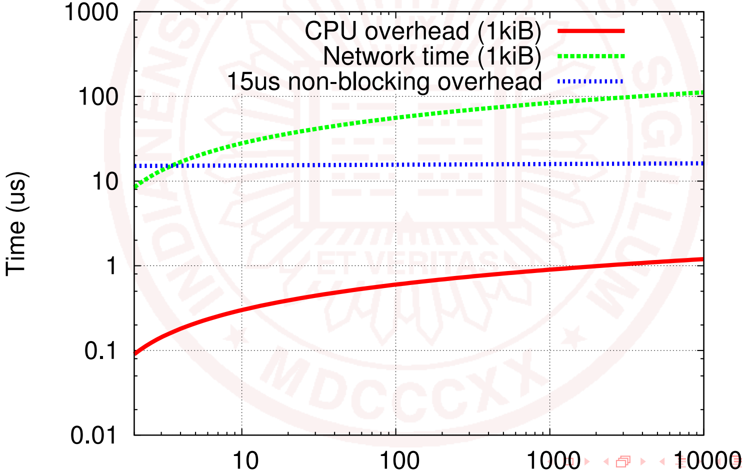




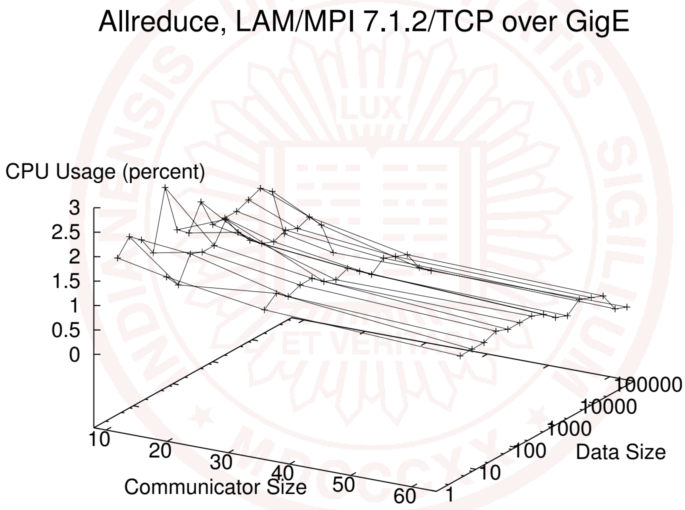
Overlap - Overhead Modelling

LogGP Model for Allreduce:

$$t_{allred} = 2 \cdot (2o + L + m \cdot G) \cdot \lceil log_2 P \rceil + m \cdot \gamma \cdot \lceil log_2 P \rceil$$



Overlap - Overhead Benchmarks



Synchronization - Process Skew

- caused by OS interference or unbalanced application
- worse if system is oversubscribed
- interference multiplies on big systems
- can cause dramatic performance decrease
- all nodes wait for the last

Example

Petrini et. al. (2003) *"The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q"*

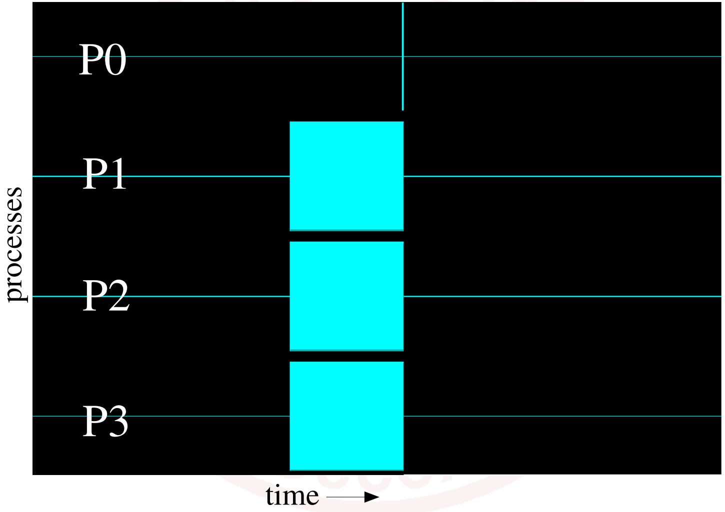
Synchronization - Process Skew

- caused by OS interference or unbalanced application
- worse if system is oversubscribed
- interference multiplies on big systems
- can cause dramatic performance decrease
- all nodes wait for the last

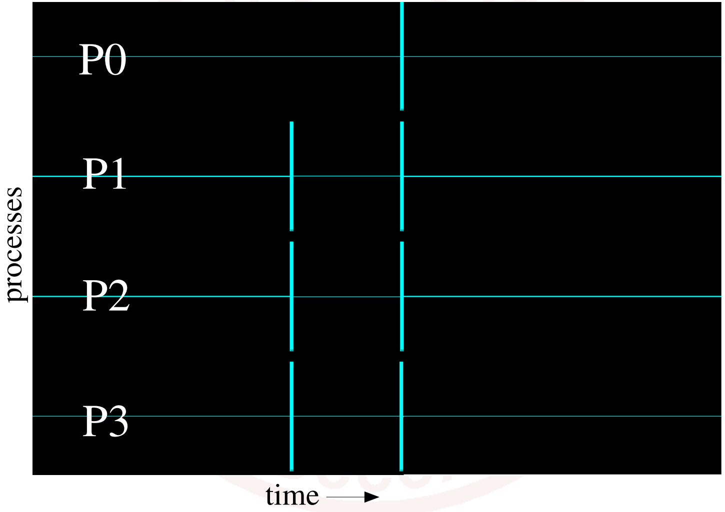
Example

Petrini et. al. (2003) *"The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q"*

MPI_Bcast with P0 delayed - Jumpshot



MPI_Ibcast with P0 delayed + overlap - Jumpshot



Outline

LibNBC - Interface

- extension to MPI
- uses NBC_Requests and NBC_Test/NBC_Wait
- IB/OFED optimized Transport Interface
- fully threaded (blocking OFED or MPI_THREAD_MULTIPLE)

Interface

```
NBC_Ibcast(buf, count, MPI_INT, 0, comm, &req);
/* compute simultaneously to communication */
NBC_Wait(&req);
```

Proposal

Hoefler et. al.: *"Non-Blocking Collective Operations for MPI-2"*

LibNBC - Interface

- extension to MPI
- uses NBC_Requests and NBC_Test/NBC_Wait
- IB/OFED optimized Transport Interface
- fully threaded (blocking OFED or MPI_THREAD_MULTIPLE)

Interface

```
NBC_Ibcast(buf, count, MPI_INT, 0, comm, &req);
/* compute simultaneously to communication */
NBC_Wait(&req);
```

Proposal

Hoefler et. al.: *"Non-Blocking Collective Operations for MPI-2"*

Non-Blocking Collectives - Implementation

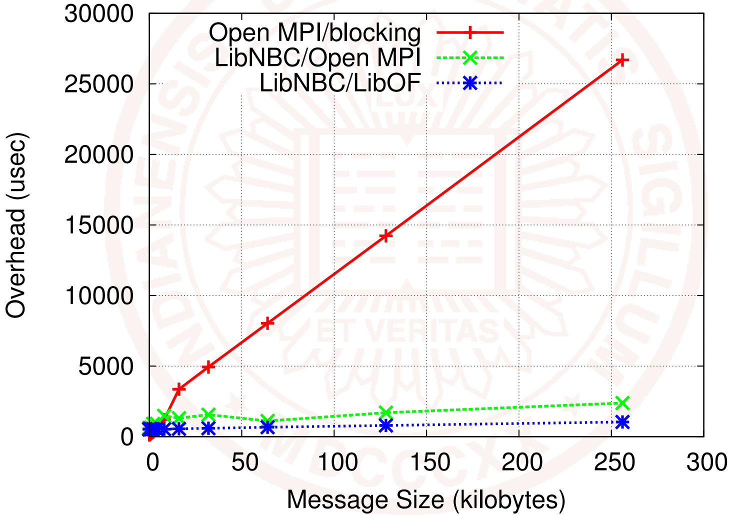
- implementation available with LibNBC
- written in ANSI-C and uses only MPI-1
- central element: collective schedule
- every coll. algorithm can be represented as a schedule
- trivial addition of new algorithms

Example: dissemination barrier, 4 nodes, node 0:

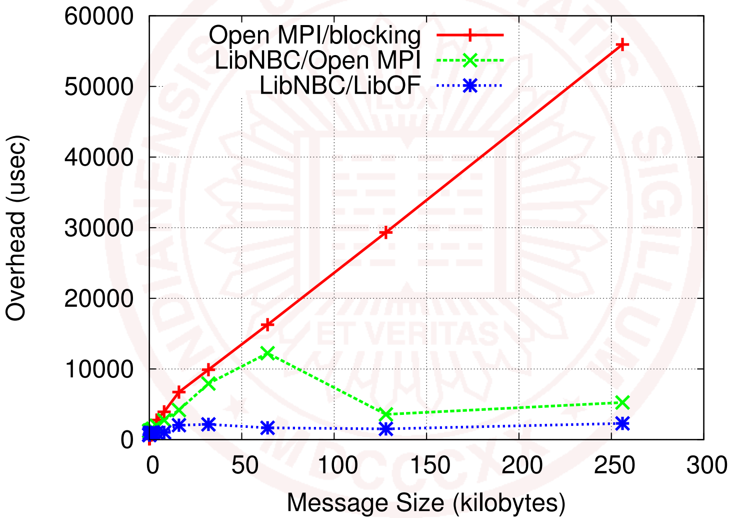
send to 1	recv from 3	end	send to 2	recv from 2	end

LibNBC download: `http://www.unixer.de/NBC`

Benchmarks - Gather with InfiniBand on 64 nodes



Benchmarks - Alltoall with InfiniBand on 64 nodes



Progression Issues

Threaded Progression

- works with MPI_THREAD_MULTIPLE and InfiniBand™
- thread “blocks” on MPI_Wait or IB file descriptor
- different OS scheduling issues (see Cluster 2008 article)

Manual Progression

- call NBC_Test to progress communication
- is necessary to advance in schedule (rounds)
- necessary frequency depends on the collective

$\Rightarrow$ progression issues are not trivial!

Outline

Independent Computation Exists in Algorithm

1) Linear Solvers - Domain Decomposition

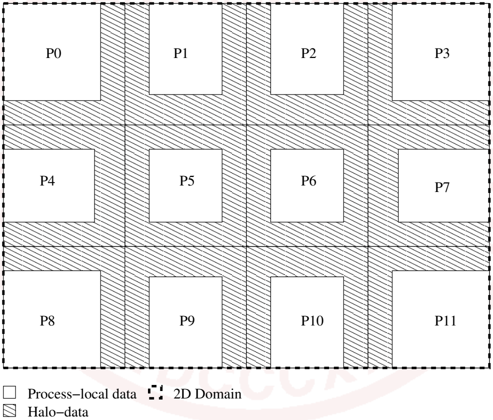
- iterative linear solvers are used in many scientific kernels
- often used operation is vector-matrix-multiply
- matrix is domain-decomposed (e.g., 3D)
- only outer (border) elements need to be communicated
- can be overlapped

2) Medical Image Reconstruction - Loop Iteration Pipelining

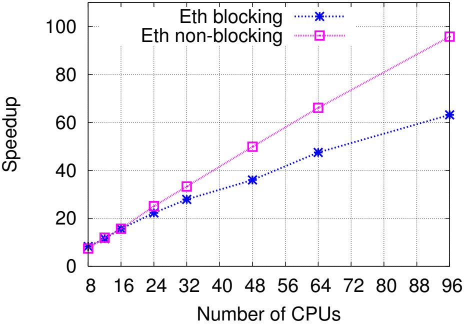
- iterations have independent parts
- communication of iteration i can be overlapped with parts of $i + 1$

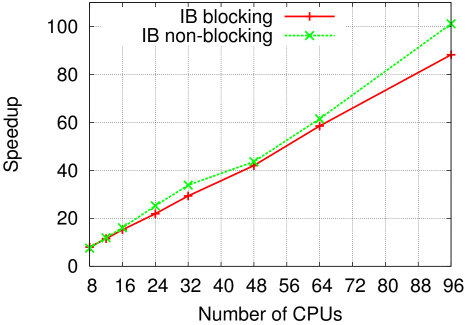
1) Linear Solver - Domain Decomposition

- nearest neighbor communication
- can be implemented with sparse/topological collectives



1) Linear Solver - Parallel Speedup (Best Case)

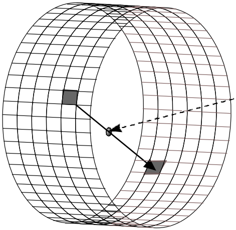




- Cluster: 128 2 GHz Opteron 246 nodes
- Interconnect: Gigabit Ethernet, InfiniBand™
- System size 800x800x800 (1 node $\approx$ 5300s)

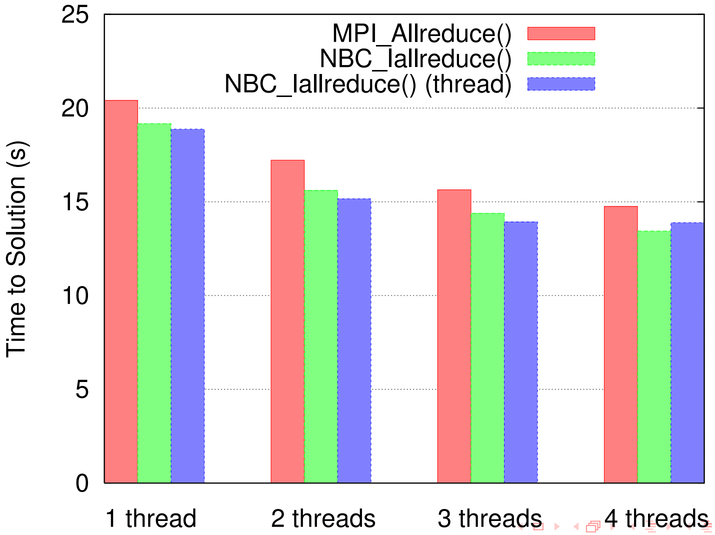
2) Medical Image Reconstruction

- OpenMP + MPI (collectives only) parallelized
- compute A_{i+1} while communciating c_l



for each(iteration k){
 for each(subiteration l){
  for (event $i \in S_l$)
   compute A_i
   compute $c_l + = (A_i)^t \frac{1}{A_i f_l^k}$
   allreduce c_l}
  $f_{l+1}^k = f_l^k c_l$}
 $f_0^{k+1} = f_{l+1}^k$}

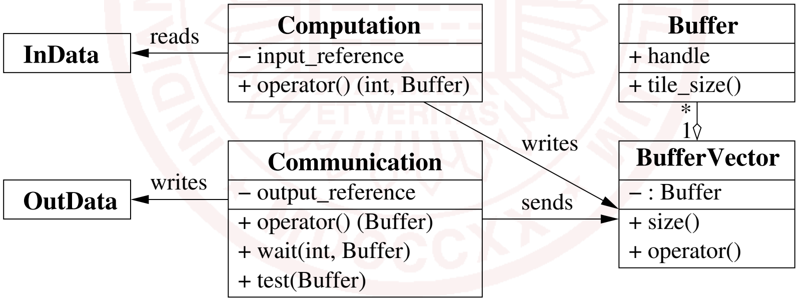
2) Medical Image Reconstruction (32 Nodes)



Data-parallel Computations

Automated Pipelining with C++ Templates

- loop tiling
- automated overlap with window of outstanding communications
- optimizing tiling factor and window size



Data-parallel Examples

1) Parallel Data Transformation (e.g., Compression)

- scatter from source, transformation, gather to destination
- scatter/gather step pipelined
- example uses bzip2 algorithm

2) 3d Fast Fourier Transformation

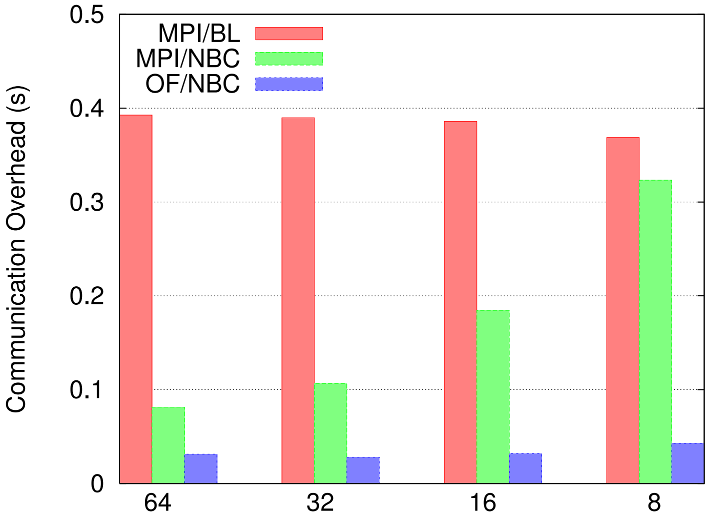
- 1d-distribution identical to “normal” parallel 3d-FFT
- start communication as early as possible
- start MPI_Ialltoall as soon as first xz-plane is ready
- calculate next xz-plane
- start next communication accordingly ...
- collect multiple xz-planes (tile factor)

1) Parallel Compression

```
my_size = 0;
for (i=0; i < N/P; i++) {
  my_size += compress(i, outptr);
  outptr += my_size;
}
gather(sizes, my_size);
gatherv(outbuf, sizes);
```

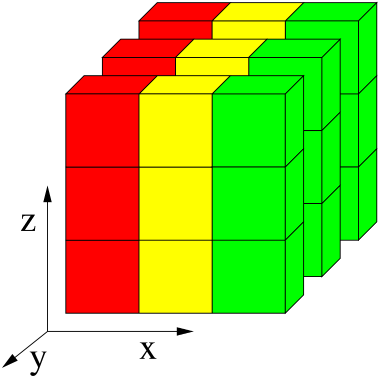
```
for (i=0; i < N/P; i++) {
  my_size = compress(i, outptr);
  gather(sizes, my_size);
  igatherv(outbuf, sizes, hndl[i]);
  if(i>0) waitall(hndl[i-1], 1);
}
waitall(hndl[N/P], 1);
```

1) Parallel Compression Communication Overhead



Transformation in z Direction

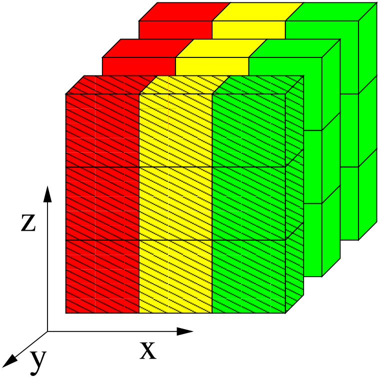
Data already transformed in y direction



1 block = 1 double value (3x3x3 grid)

Transformation in z Direction

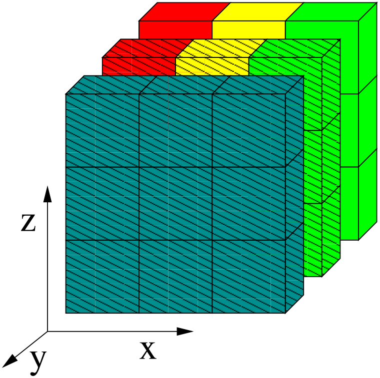
Transform first xz plane in z direction



pattern means that data was transformed in y and z direction

Transformation in z Direction

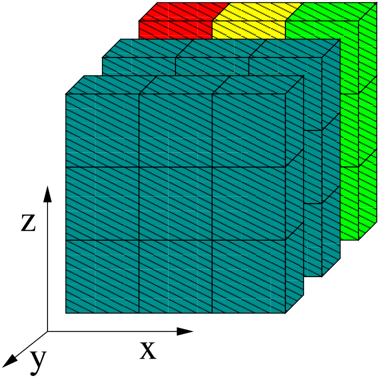
start MPI_Ialltoall of first xz plane and transform second plane



cyan color means that data is communicated in the background

Transformation in z Direction

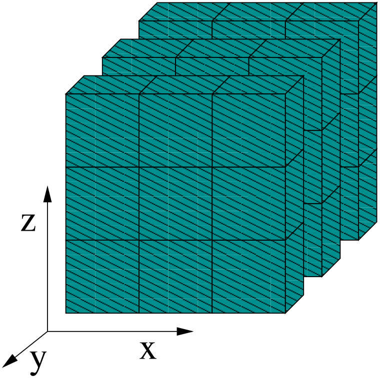
start MPI_Ialltoall of second xz plane and transform third plane



data of two planes is not accessible due to communication

Transformation in x Direction

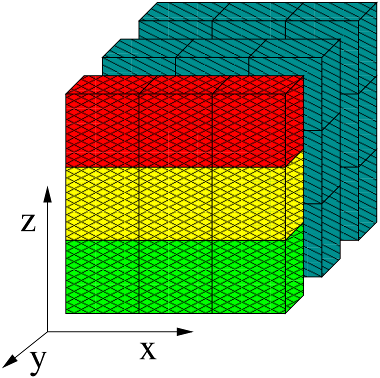
start communication of the third plane and ...



we need the first xz plane to go on ...

Transformation in x Direction

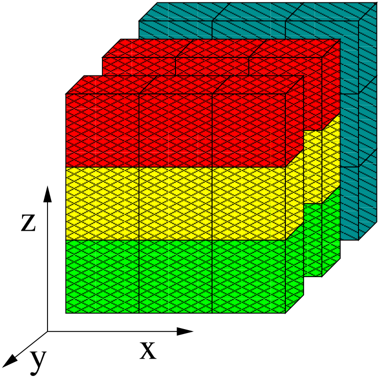
... so MPI_Wait for the first MPI_Ialltoall!



and transform first plane (new pattern means xyz transformed)

Transformation in x Direction

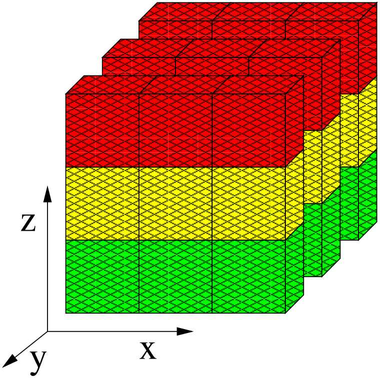
Wait and transform second xz plane



first plane's data could be accessed for next operation

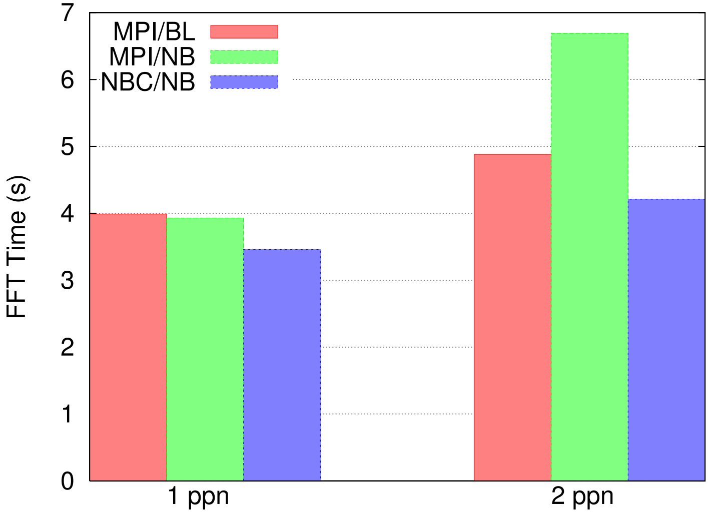
Transformation in x Direction

wait and transform last xz plane



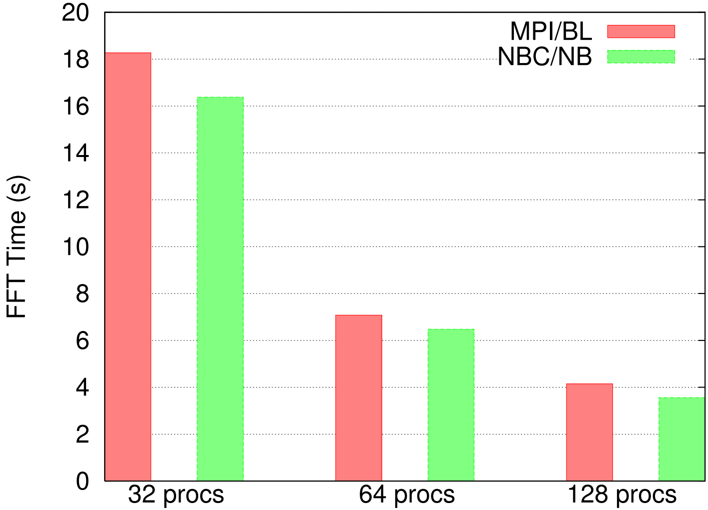
done! → 1 complete 1D-FFT overlaps a communication

2) 1024^3 3d-FFT over InfiniBand



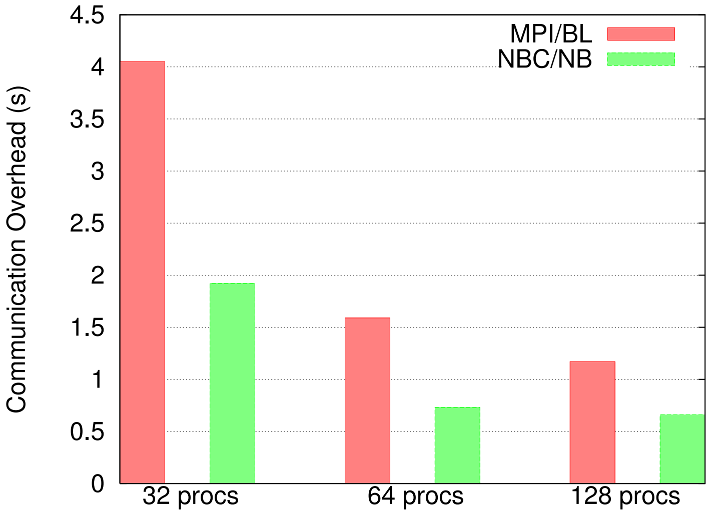
P=128, "Coyote"@LANL - 128/64 dual socket 2.6GHz Opteron nodes

2) 1024^3 3d-FFT on XT4



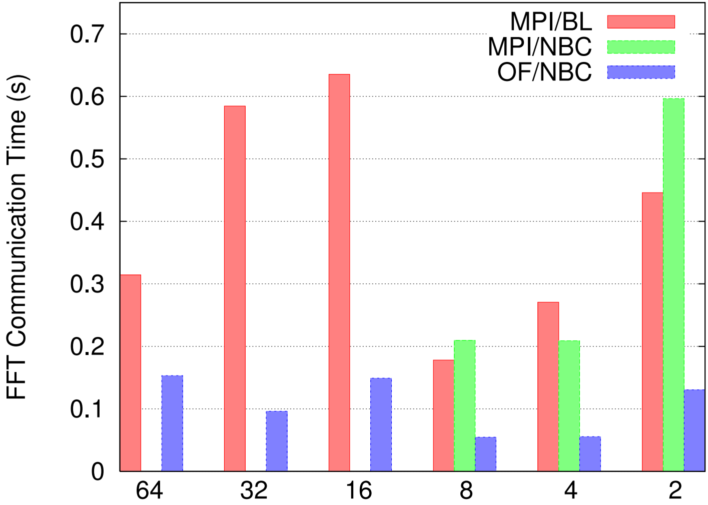
“Jaguar”@ORNL - Cray XT4, dual socket dual core 2.6GHz Opteron

2) 1024^3 3d-FFT on XT4 (Communication Overhead)



“Jaguar”@ORNL - Cray XT4, dual socket dual core 2.6GHz Opteron

2) 640^3 3d-FFT InfiniBand (Communication Overhead)



“Odin”@IU - dual socket dual core 2.0GHz Opteron InfiniBand

Outline

Ongoing Work

LibNBC

- optimized collectives and modeling
- more low-level transports (e.g., MX)
- analyze offloading/onloading collectives

MPI-Forum (MPI-3) Efforts

- proposed non-blocking collectives
- proposed sparse collective
- several proposals to enhance library support

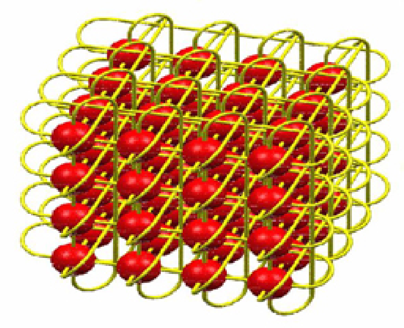
Applications

- work on more applications (apply C++ templates?)
- $\Rightarrow$ interested in collaborations (ask me!)

Discussion

THE END

Questions?



Thank you for your attention!