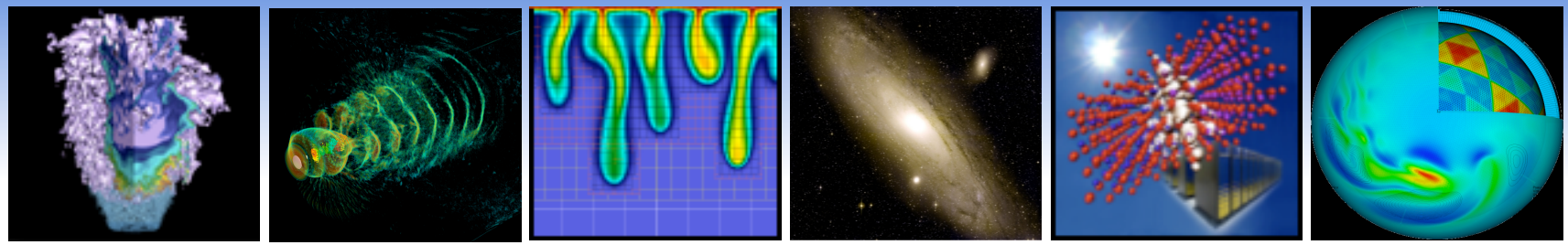


Parallel Application Scaling, Performance, and Efficiency

NUG 2010
Supplemental Material

(David Skinner and Katie Antypas)

U.S. DEPARTMENT OF ENERGY | Office of Science

NERSC

BERKELEY LAB

NERSC
NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

Overview

- Review Some Basic MPI
- Domain Decomposition
- Load Balancing
- Case Study: FLASH Scaling Performance
- Performance Monitoring with IPM
- Many-core chips and the future of parallel programming

Overview and History of MPI

- Library (not language) specification
- Goals
  - Portability
  - Efficiency
  - Functionality (small and large)
- Most basic communications are 2 sided
- Pros
  - Programmer has control at low level
  - Performance model understood
  - Can be very high performing
- Cons
  - Programmer has control at low level
  - Error prone
  - Questions about memory usage as cores/node increase

Generic Message Passing

send(void* sendbuffer, int num_elements, int destination_rank)

receive(void* recvbuffer, int num_elements, int source_rank)

```
Processor 0
x = 5
send(&x, 1, 1)
x = 7
```

```
Processor 1


receive(&x, 1, 0)
print x
```

What rules are needed so that processor 1 receives "5" and not "7"?

Ways to Send Data

	Blocking	Non-blocking
Non-Buffered	**Sending processor "blocks" or "waits" for receive**	**Sending processor completes call, but must be careful not to overwrite send buffer until receive operation has completed**
Buffered	Sending process completes call after sendbuf has been copied to another buffer	

Be careful with buffering …

send(void* sendbuffer, int num_elements, int destination_rank)

receive(void* recvbuffer, int num_elements, int source_rank)

```
Processor 0
do i=1, 1000
 produce_data(&x)
 send(&x, 1, 1)
end do
```

```
Processor 1
do i=1, 1000
 receive(&x, 1, 0)
 consume(&x)
end do
```

What could go wrong with buffered send?

Basic Point to Point

- Blocking – Non-buffered
  - MPI_Send()
  - MPI_Recv()

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

if (rank == 0) {
   MPI_Send(&work, 1, MPI_INT, dest, TAG, MPI_COMM_WORLD);
} else {
   MPI_Recv(&result, 1, MPI_INT, src, TAG, MPI_COMM_WORLD,
  &status);
}
```

NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

Non-Blocking Operations

- MPI_Isend()
- MPI_Irecv()
- “I” is for immediate
- Paired with MPI_Test()/MPI_Wait()

Non-Blocking Operations

```
MPI_Comm_rank(comm,&rank);

if (rank == 0) {
   MPI_Isend(sendbuf,count,MPI_REAL,1,tag,comm,&request);
   /* Do some computation */
   MPI_Wait(&request,&status);
} else {
   MPI_Irecv(recvbuf,count,MPI_REAL,0,tag,comm,&request);
   /* Do some computation */
   MPI_Wait(&request,&status);
}
```

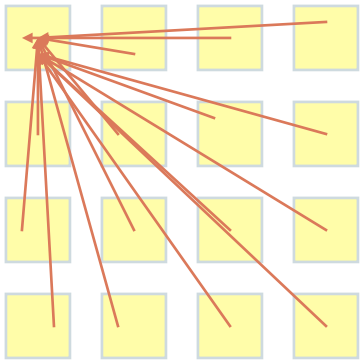
NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

Collective Operations

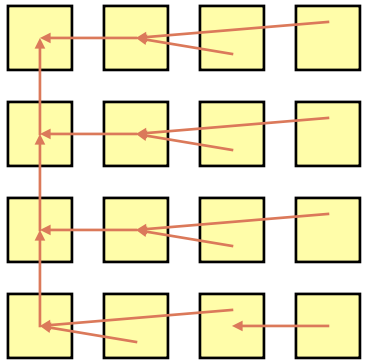
- May be layered on point to point
- May use tree communication patterns for efficiency
- Synchronization! (No non-blocking collectives)

Collective Operations

```
MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, comm);
```



O(P)



O(log P)

NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

Quiz: MPI_Send()

- **After I call MPI_Send()**
  - The recipient has received the message
  - I have sent the message
  - I can write to the message buffer without corrupting the message
- **I can write to the message buffer**

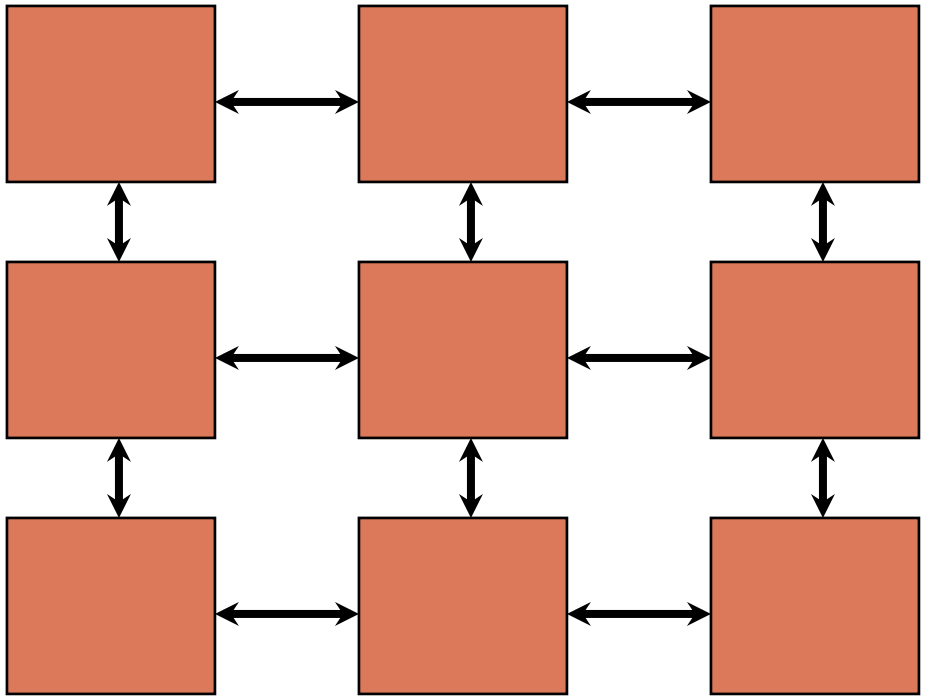
Quiz: MPI_Isend()

- After I call MPI_Isend()
  - The recipient has started to receive the message
  - I have started to send the message
  - I can write to the message buffer without corrupting the message
- None of the above (I must call MPI_Test() or MPI_Wait())

Minimizing Latency

- Collect small messages together (if you can)
  - One 1024-byte message instead of 1024 one-byte messages
- Minimize other overhead (e.g., copying)
- Overlap with computation (if you can)

Example: Domain Decomposition



Naïve Approach

```
while (!done) {
  exchange(D, neighbors, myrank);
  dored(D);
  exchange(D, neighbors, myrank);
  doblack(D);
}

void exchange(Array D, int *neighbors, int myrank) {
  for (i = 0; i < 4; i++)
    MPI_send(…);
  for (i = 0; i < 4; i++)
    MPI_recv(…);
}
```

NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

Naïve Approach

- **Deadlock! (*Maybe*)**
- **Can fix with careful coordination of receiving versus sending on alternate processes**
- **But this can still serialize**

MPI_Sendrecv()

```
while (!done) {
  exchange(D, neighbors, myrank);
  dored(D);
  exchange(D, neighbors, myrank);
  doblack(D);
}

void exchange(Array D, int *neighbors, int myrank) {
  for (i = 0; i < 4; i++) {
    MPI_Sendrecv(…);
  }
}
```

Immediate Operations

```
while (!done) {
  exchange(D, neighbors, myrank);
  dored(D);
  exchange(D, neighbors, myrank);
  doblack(D);
}

void exchange(Array D, int *neighbors, int myrank) {
  for (i = 0; i < 4; i++) {
    MPI_Isend(…);
    MPI_Irecv(…);
  }
  MPI_Waitall(…);
}
```

Basic Functions

MPI_Init	***Initializes MPI***
MPI_Comm_size	***Returns # tasks in communicator***
MPI_Comm_rank	***Returns ID of current proc***
MPI_Send	***sends data***
MPI_Recv	***receives data***
MPI_Reduce	***reduce data to single processor***
MPI_Allreduce	***reduce all procs data to all procs***
MPI_Bcast	***broadcasts to all procs***
MPI_Finalize	***Closes MPI***

NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

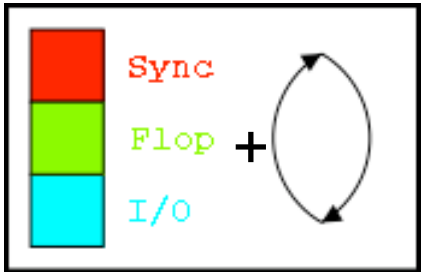
Load Balancing

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

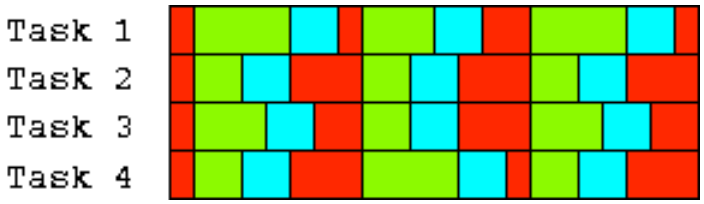
Load Balance : cartoon

The Universal Parallel Science App

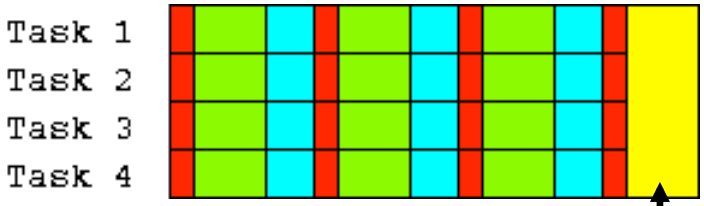


~All apps come down to the same basic pattern. Ok, Maybe there is no I/IO.

Unbalanced:

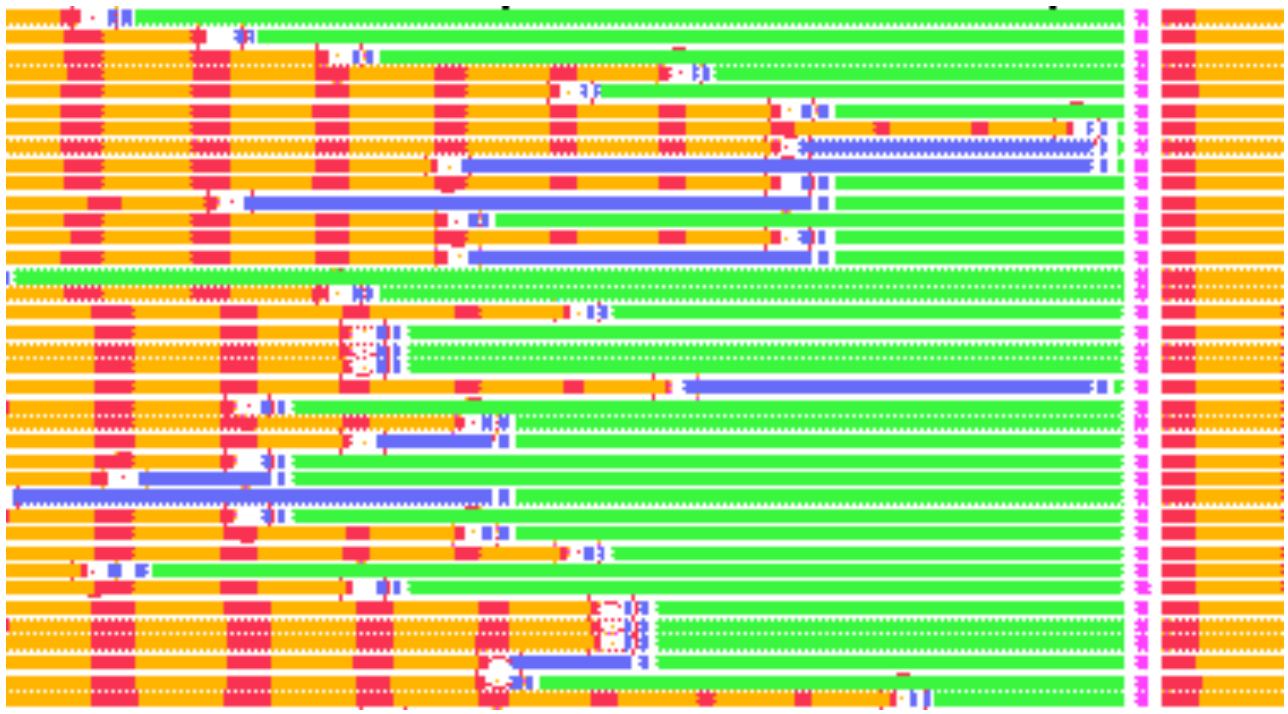


Balanced:



Time saved by load balance

Load Balance: real code



Flops
Exchange
Sync

MPI Rank →

Time →

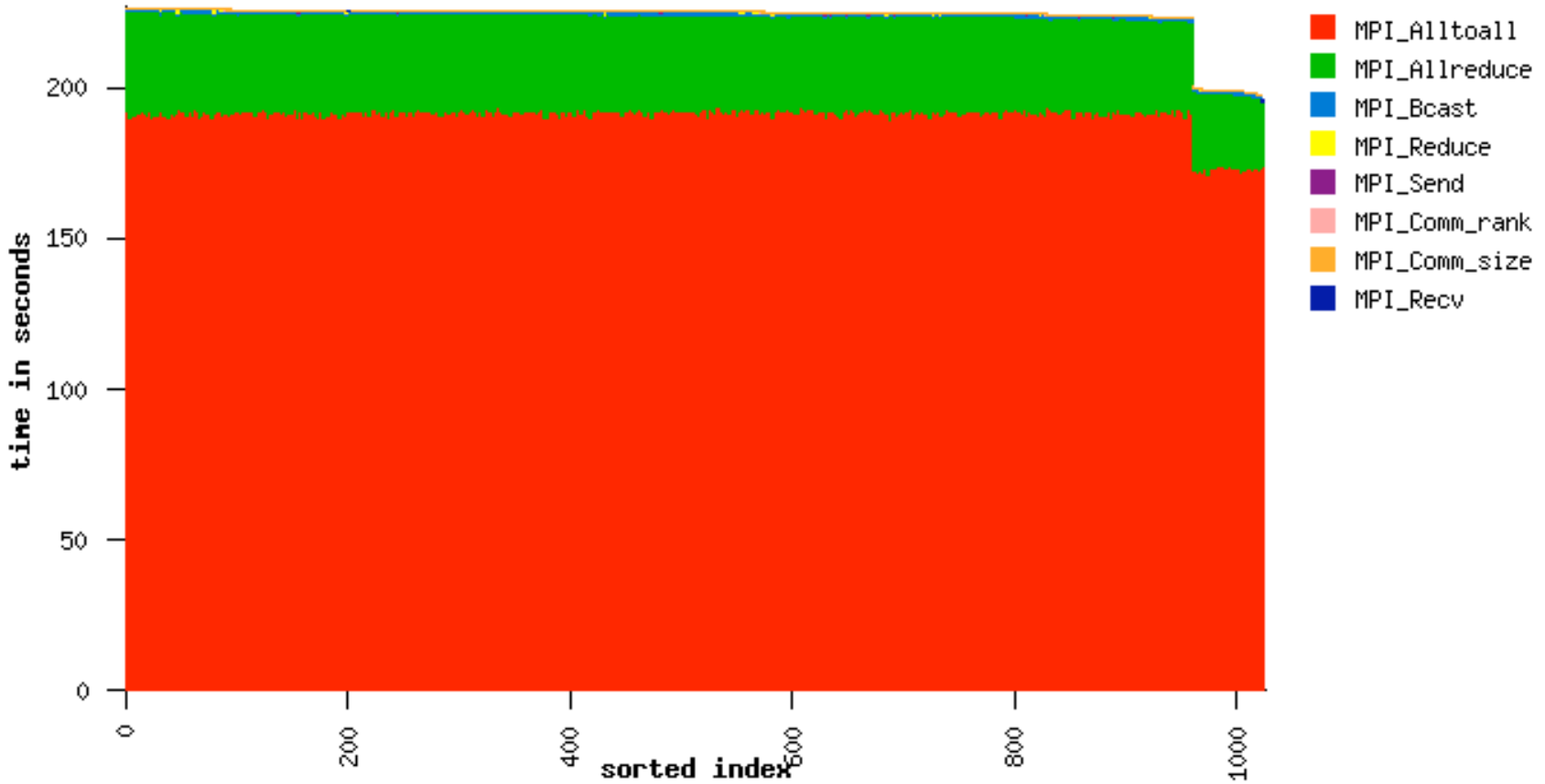
NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Load Balance : performance data

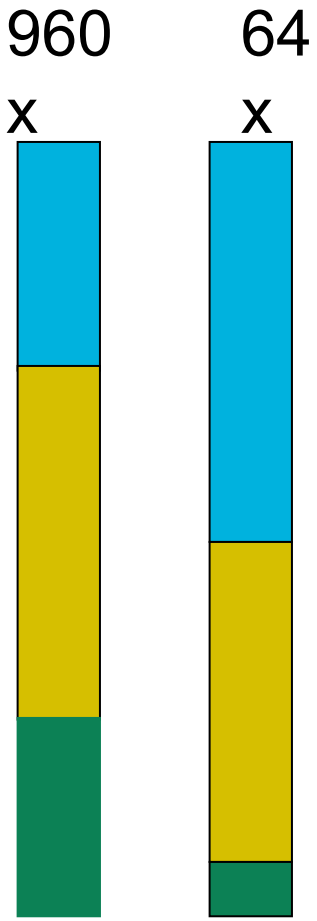
Communication Time: 64 tasks show 200s, 960 tasks show 230s



MPI ranks sorted by total communication time

Load Balance: ~code

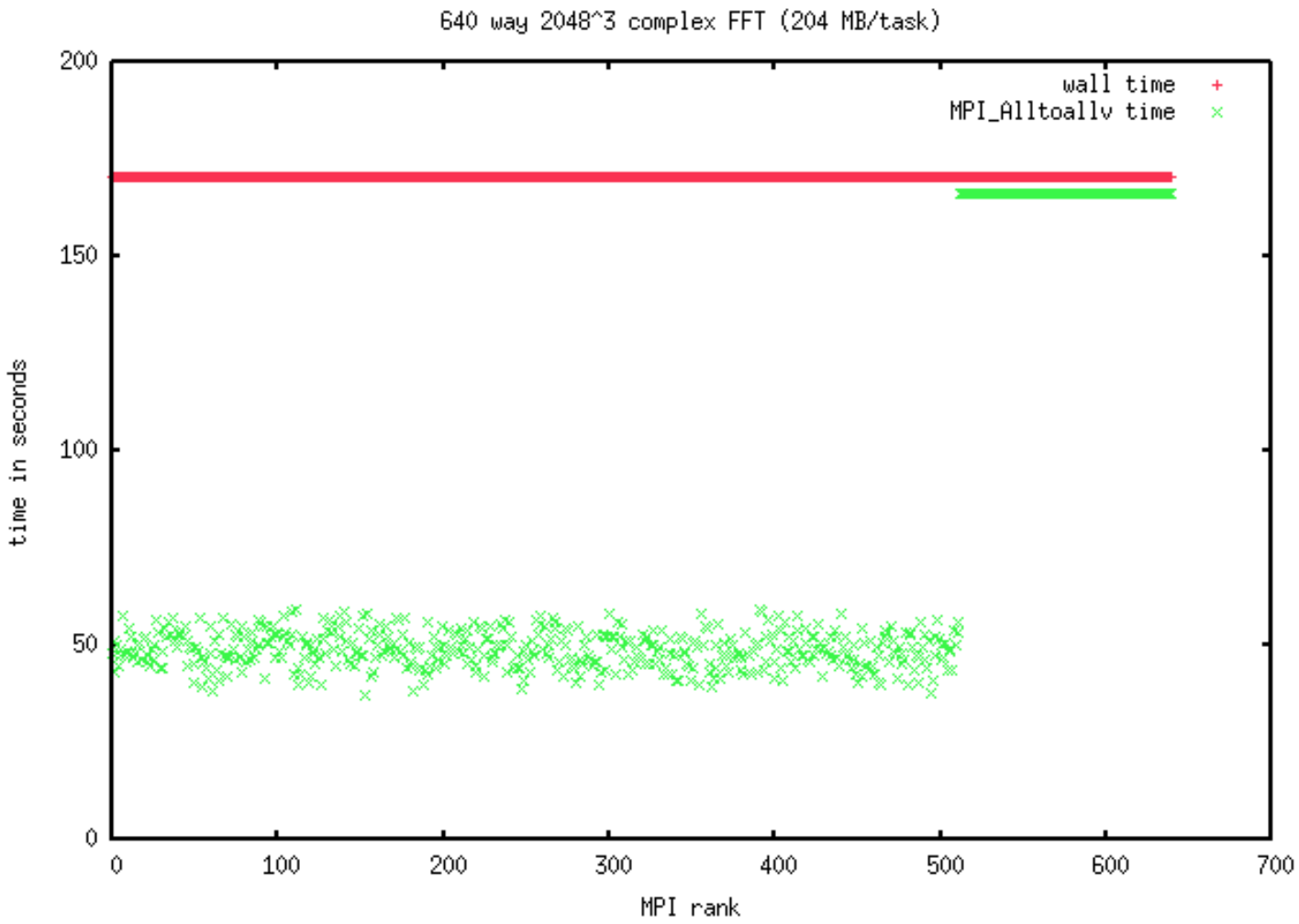
```
while(1) {
  do_flops(N_i);
  MPI_Alltoall();
  MPI_Allreduce();
}
```



Load Balance : analysis

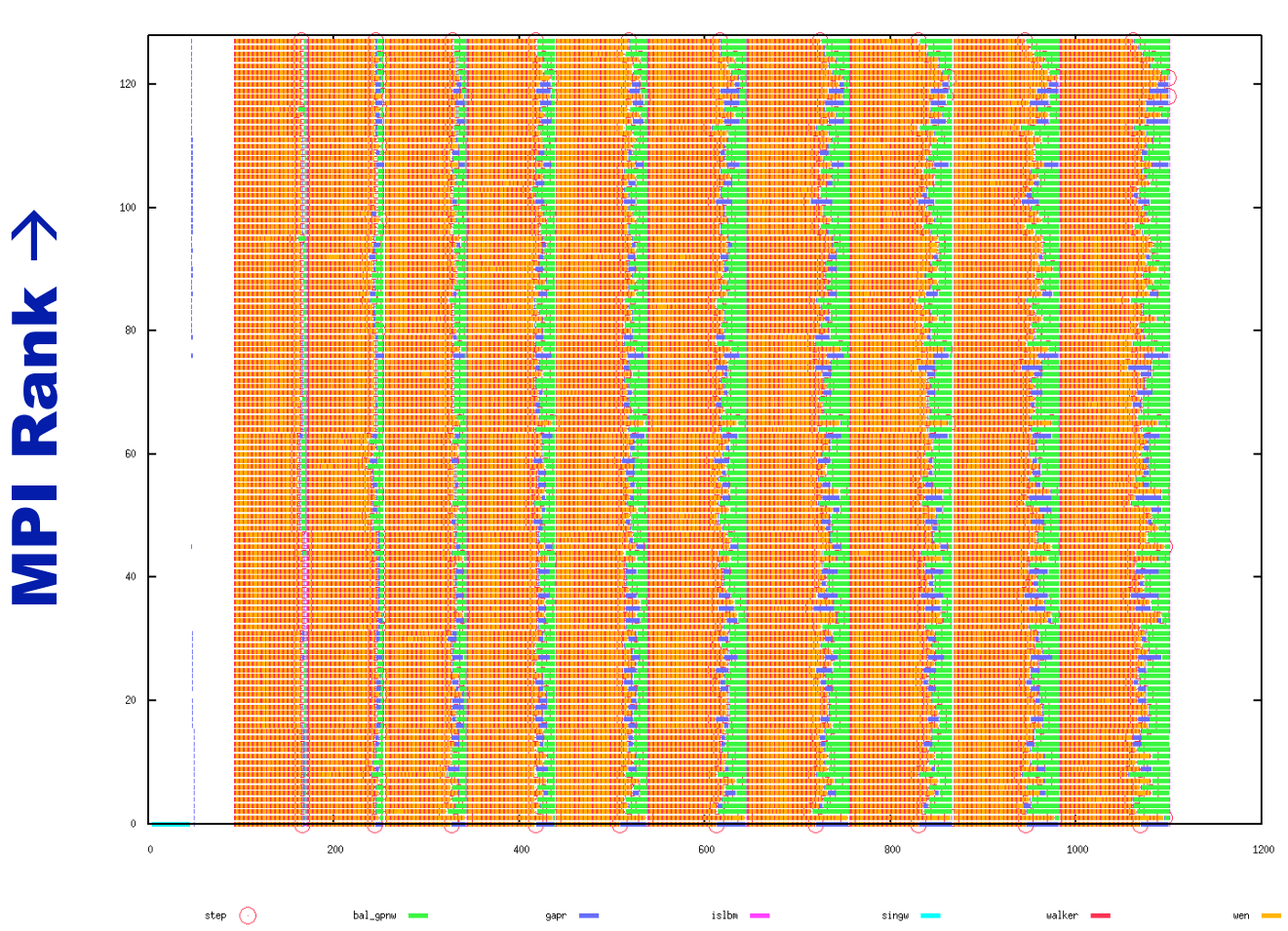
- The 64 slow tasks (with more compute work) cause 30 seconds more “communication” in 960 tasks
- This leads to 28800 CPU*seconds (8 CPU*hours) of unproductive computing
- All imbalance requires is one slow task and a synchronizing collective!
- Parallel computers allow you to scale both your computation and your load imbalance.

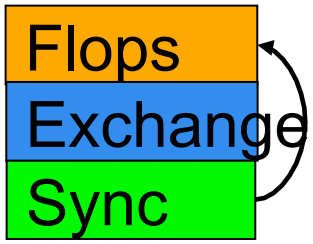
Load Balance : FFT



When is imbalance good?

Dynamical Load Balance: Motivation





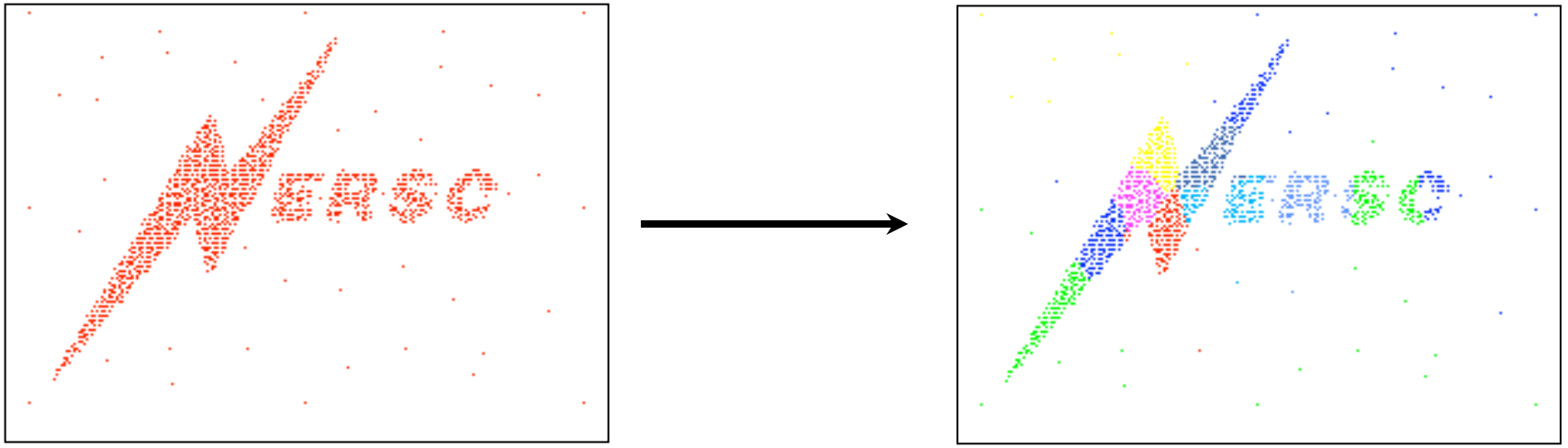
NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Load Balance: Summary

- Imbalance most often a byproduct of data decomposition
- Must be addressed **before** further MPI tuning can happen
- Good software exists for graph partitioning / remeshing



- Dynamical load balance may be required for adaptive codes

NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

Scaling Study

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

Scaling: definitions

- Scaling studies involve changing the degree of parallelism.

- Strong scaling
  - Fixed problem size, more computer
- Weak scaling
  - Problem size grows with concurrency

Parallel Performance Measurements

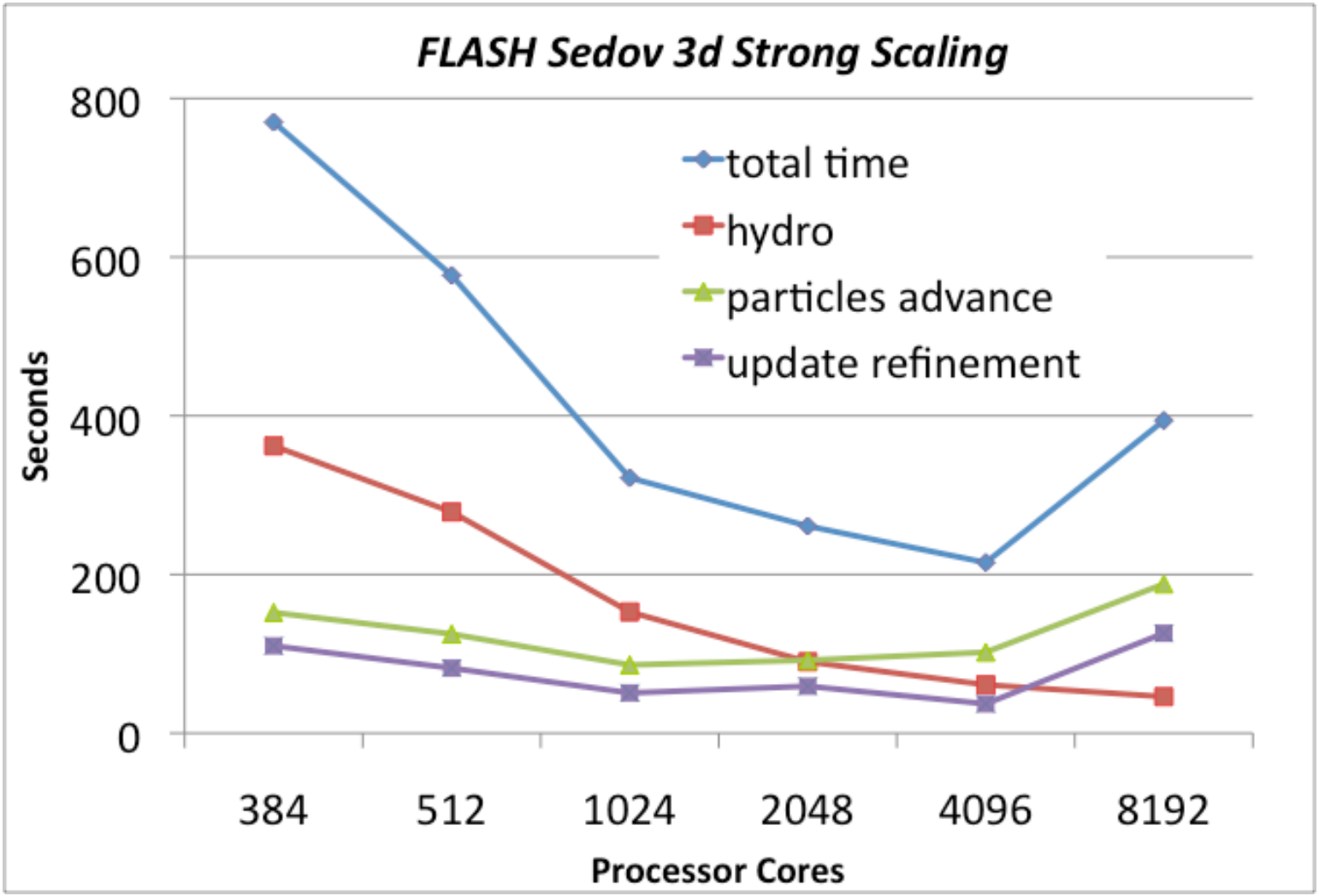
- Speed up = $T_{serial} / T_{parallel}(n)$
  - T_{serial} = 100 secs
  - $T_{parallel}(2)$ = 80 secs
  - 25% speed up

- Efficiency = $T_{serial}/(n*T_{parallel}(n))$
  - 100/(2*80) =
  - 62% efficiency
- Perfect Scaling?

Be aware there are multiple definitions for these terms

NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

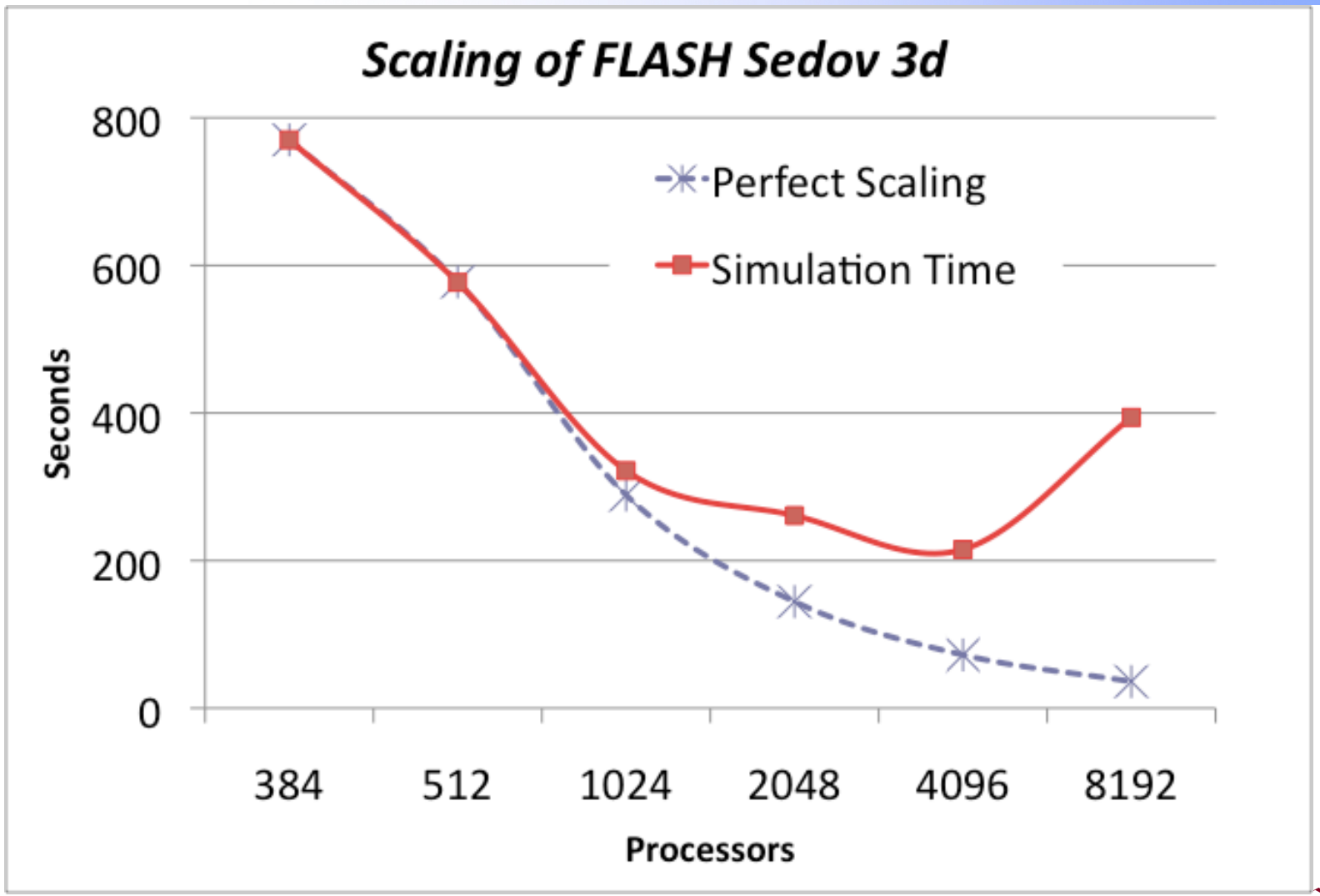
FLASH Sedov 3d problem with Particles



U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

FLASH Sedov 3d problem with Particles



NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

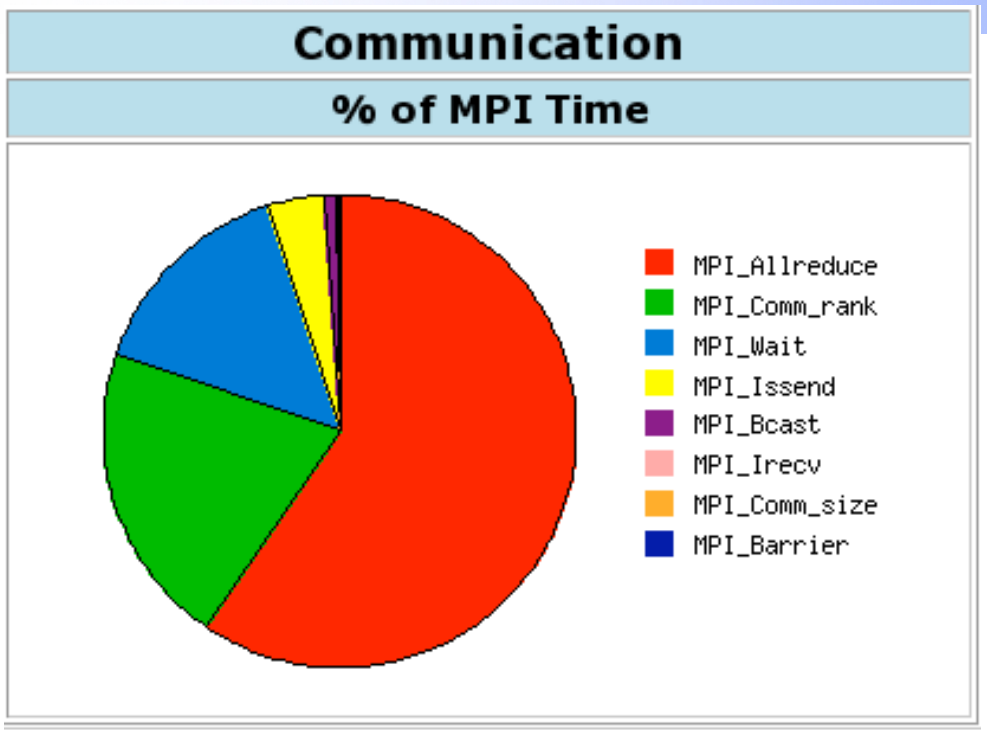
Scaling: Analysis

- **What is happening in the 8192 case?**
  - Compute per core decreasing
  - Synchronization rate increasing
  - Surface to volume ratio increasing
- **What else could happen?**
  - Algorithmic scaling may change
  - Maybe we hit an architectural boundary in the machine (switch level, mid-plane, queue, etc.)
  - Maybe depleted some buffer space resource
  - Many more things…performance debugging at scale is detective work in the application + architecture space

Parallel programs are easier to mess up than serial ones. Here are a couple common pitfalls.

NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

What's wrong here?

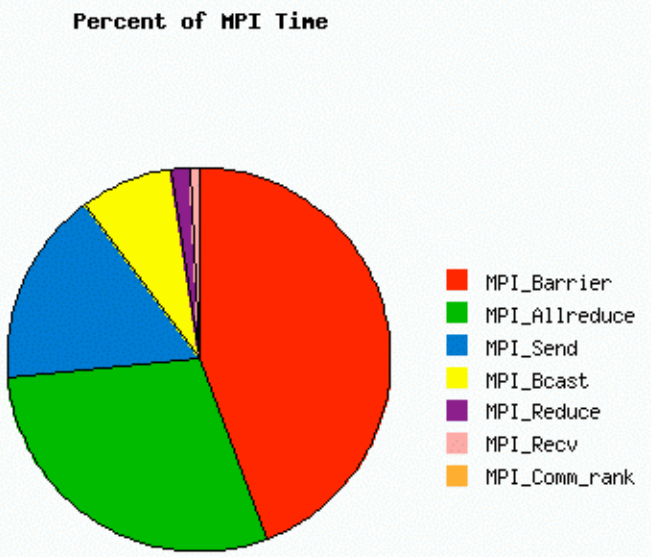


Communication Event Statistics (100.00% detail)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Allreduce	8	3278848	124132.547	0.000	114.920	59.35	16.88
MPI_Comm_rank	0	35173439489	43439.102	0.000	41.961	20.77	5.91
MPI_Wait	98304	13221888	15710.953	0.000	3.586	7.51	2.14
MPI_Wait	196608	13221888	5331.236	0.000	5.716	2.55	0.72
MPI_Wait	589824	206848	5166.272	0.000	7.265	2.47	0.70

MPI_Barrier

Function	Total calls	Total time (sec)		Total buffer size (MB)	Avg. Buffer Size/call (Bytes)
MPI_Barrier	6.02e+05	3.48e+05	44.23%	0	0
MPI_Allreduce	3.18e+07	2.31e+05	29.33%	3.61e+05	11,936
MPI_Send	1.29e+08	1.29e+05	16.36%	5.24e+04	426
MPI_Bcast	5.73e+07	6.08e+04	7.73%	5.39e+04	987
MPI_Reduce	1.08e+08	1.24e+04	1.58%	1.66e+05	1,620
MPI_Recv	1.29e+08	6.11e+03	0.78%	5.24e+04	426
MPI_Comm_rank	1.14e+03	5.92e-01	7.52e-05%	0	0
MPI_Comm_size	6.66e+02	0	0%	0	0



Is MPI_Barrier time bad? Probably. Is it avoidable?

~three cases:

1) The stray / unknown / debug barrier
2) The barrier which is masking compute imbalance
3) Barriers used for I/O ordering

How to use IPM : basics

Many of the graphs in this talk were generated with a tool called IPM – Integrated Performance Monitoring: free and easy to install

http://ipm-hpc.sourceforge.net/

On galera

1) >> mpicc test.c -lipm

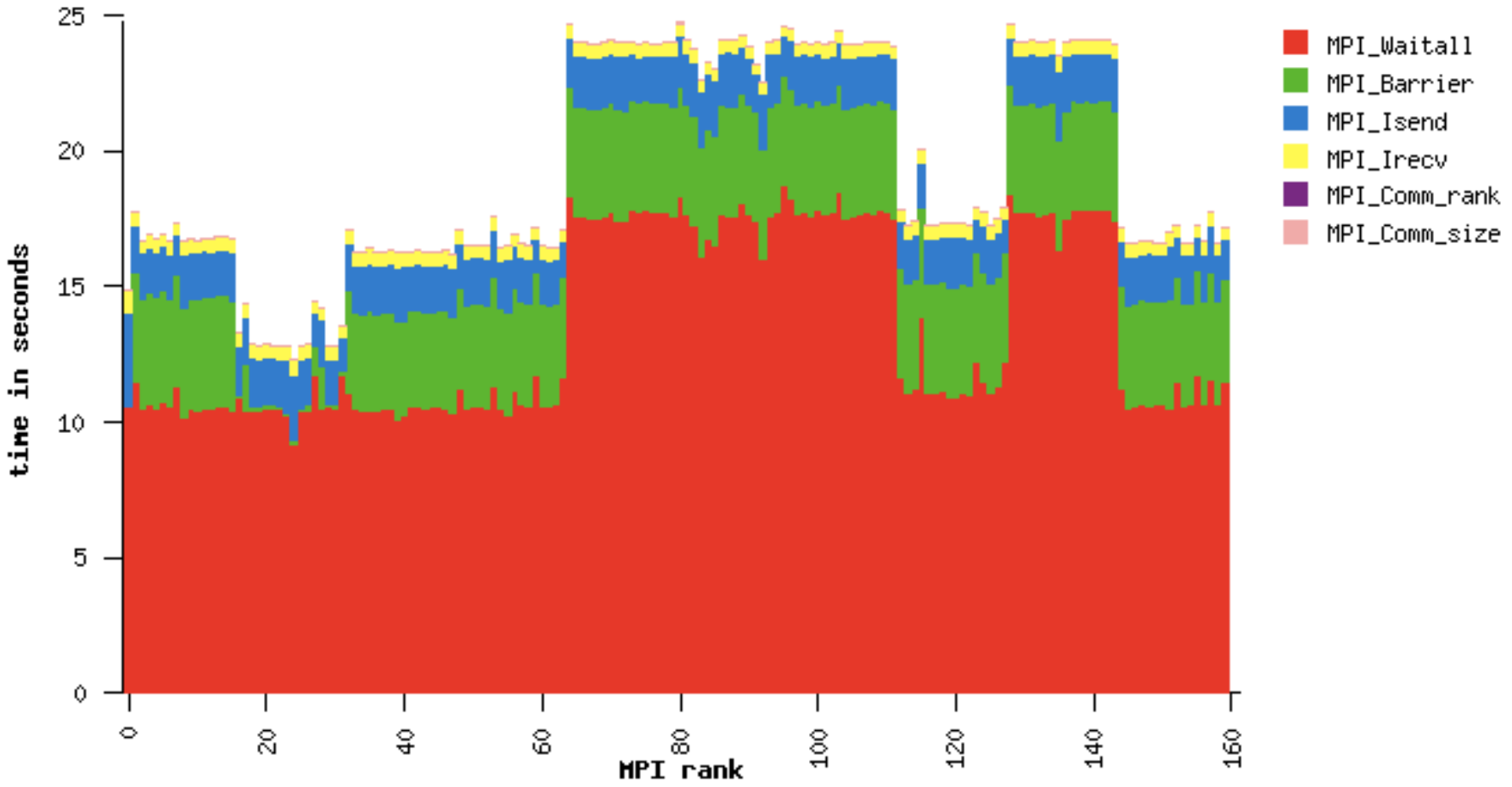
2) Run job as usual

3) Appended to your output file

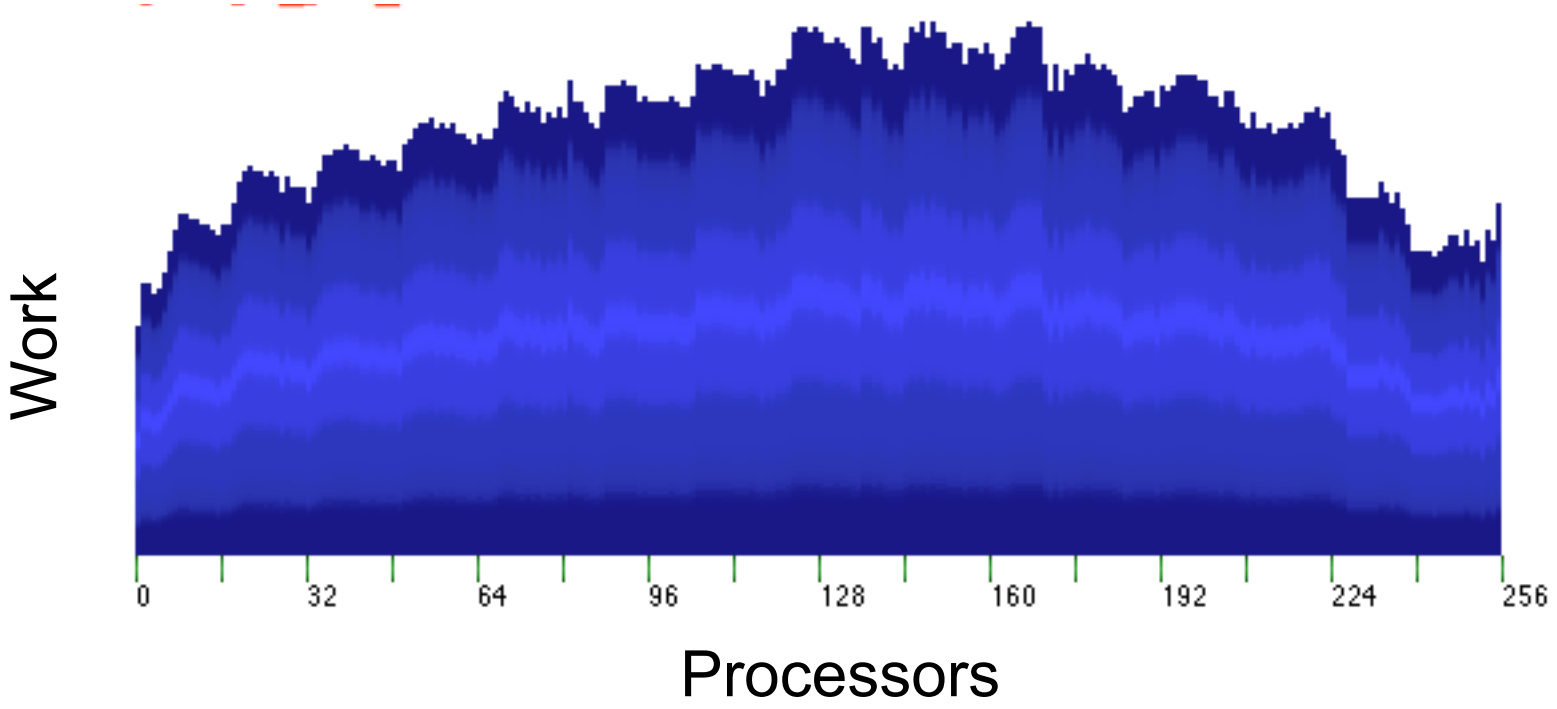
How to use IPM : basics

```
##IPMv0.982#########################################################################
#
# command : unknown (completed)
# host    : n9-1-6/x86_64_Linux            mpi_tasks : 4 on 1 nodes
# start   : 07/19/10/14:34:24              wallclock : 0.013101 sec
# stop    : 07/19/10/14:34:24              %comm     : 11.34
# gbytes  : 0.00000e+00 total              gflop/sec : 0.00000e+00 total
#
###########################################################################
# region  : *       [ntasks] =      4
#
#                           [total]         <avg>           min           max
# entries                         4             1             1             1
# wallclock               0.0453982     0.0113496    0.00918508     0.0131011
# user                     0.161974     0.0404935      0.035994      0.045993
# system                   0.123979     0.0309947      0.023996      0.034994
# mpi                     0.00594119     0.0014853   0.000761837    0.00176443
# %comm                                  11.3372       5.82577       19.2098
# gflop/sec                       0             0             0             0
# gbytes                          0             0             0             0
#
#
#                            [time]       [calls]        <%mpi>      <%wall>
# MPI_Allreduce           0.00343561           372         57.83         7.57
# MPI_Recv                0.00194091           558         32.67         4.28
# MPI_Send                0.000562498          558          9.47         1.24
# MPI_Comm_size           1.21177e-06            4          0.02         0.00
# MPI_Comm_rank           9.65199e-07            4          0.02         0.00
###########################################################################
```

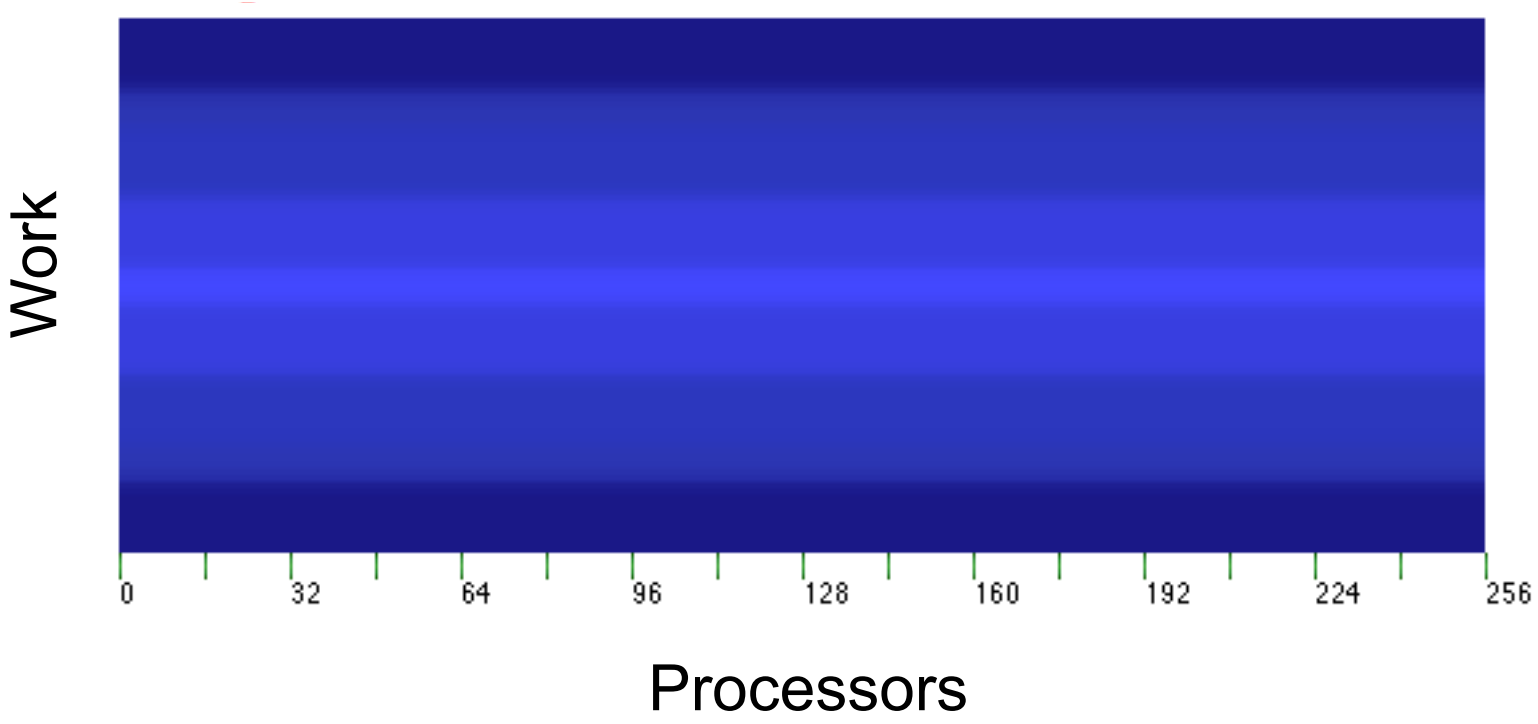
Profiling Codes Using IPM



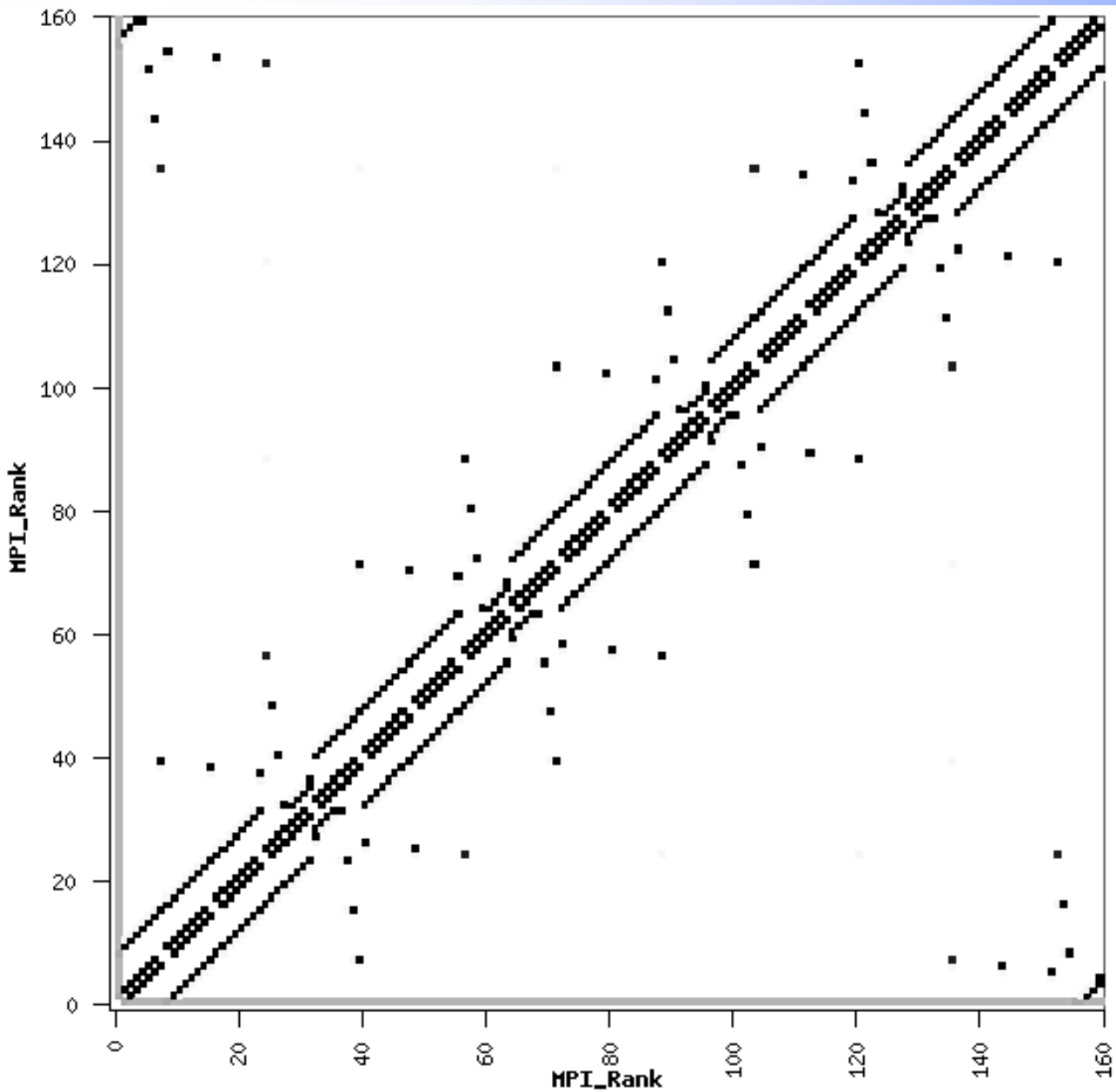
An Unbalanced Code



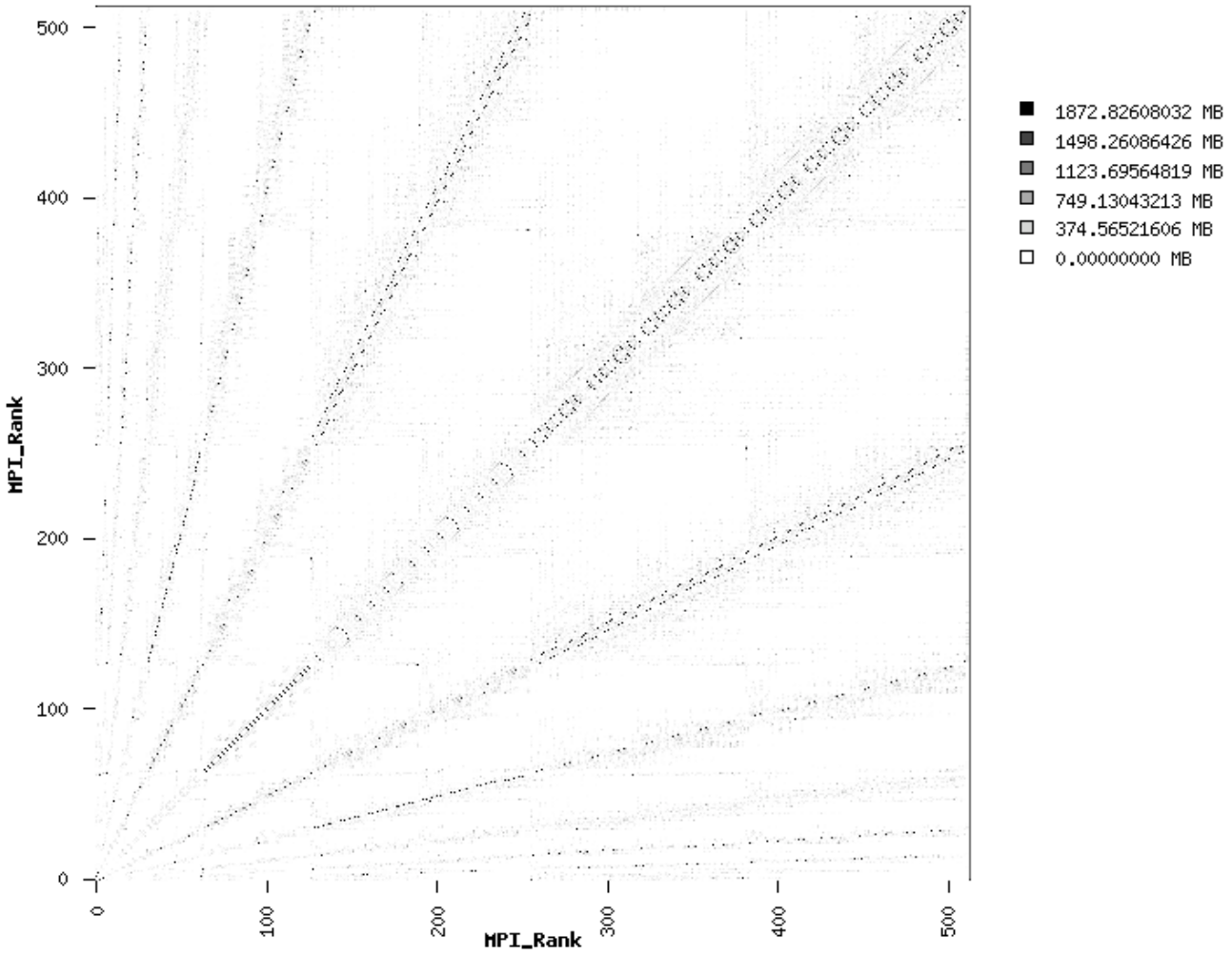
A Balanced Code



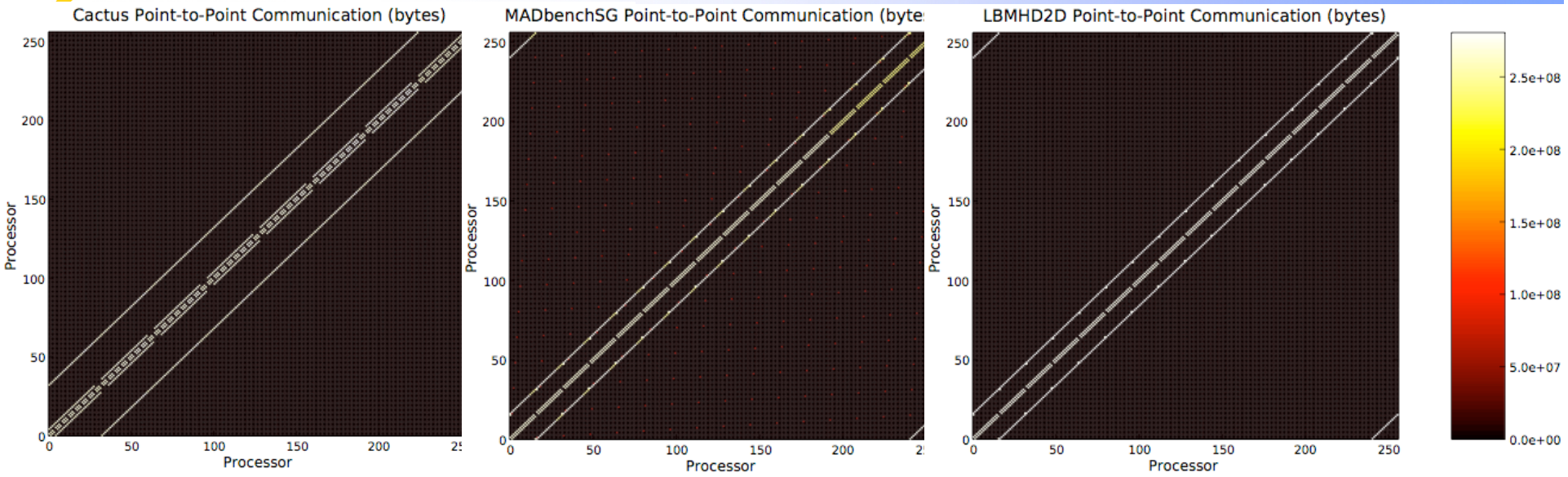
Communication Patterns

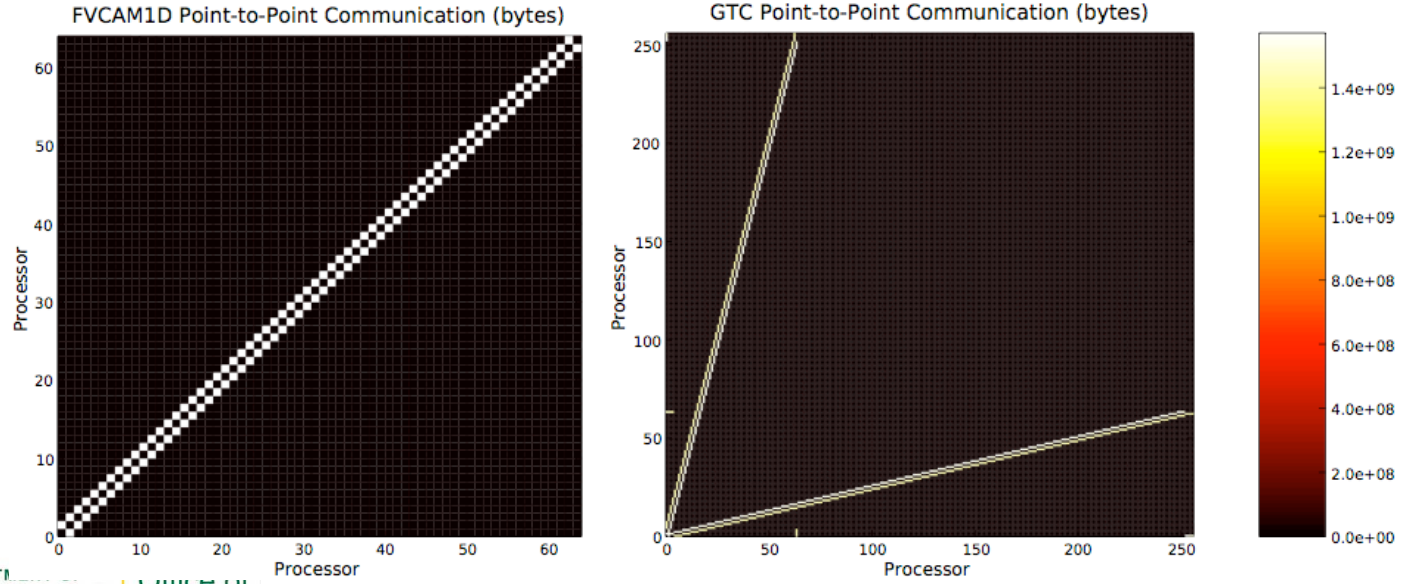


Communication Patterns



Low Degree Regular Mesh Communication Patterns

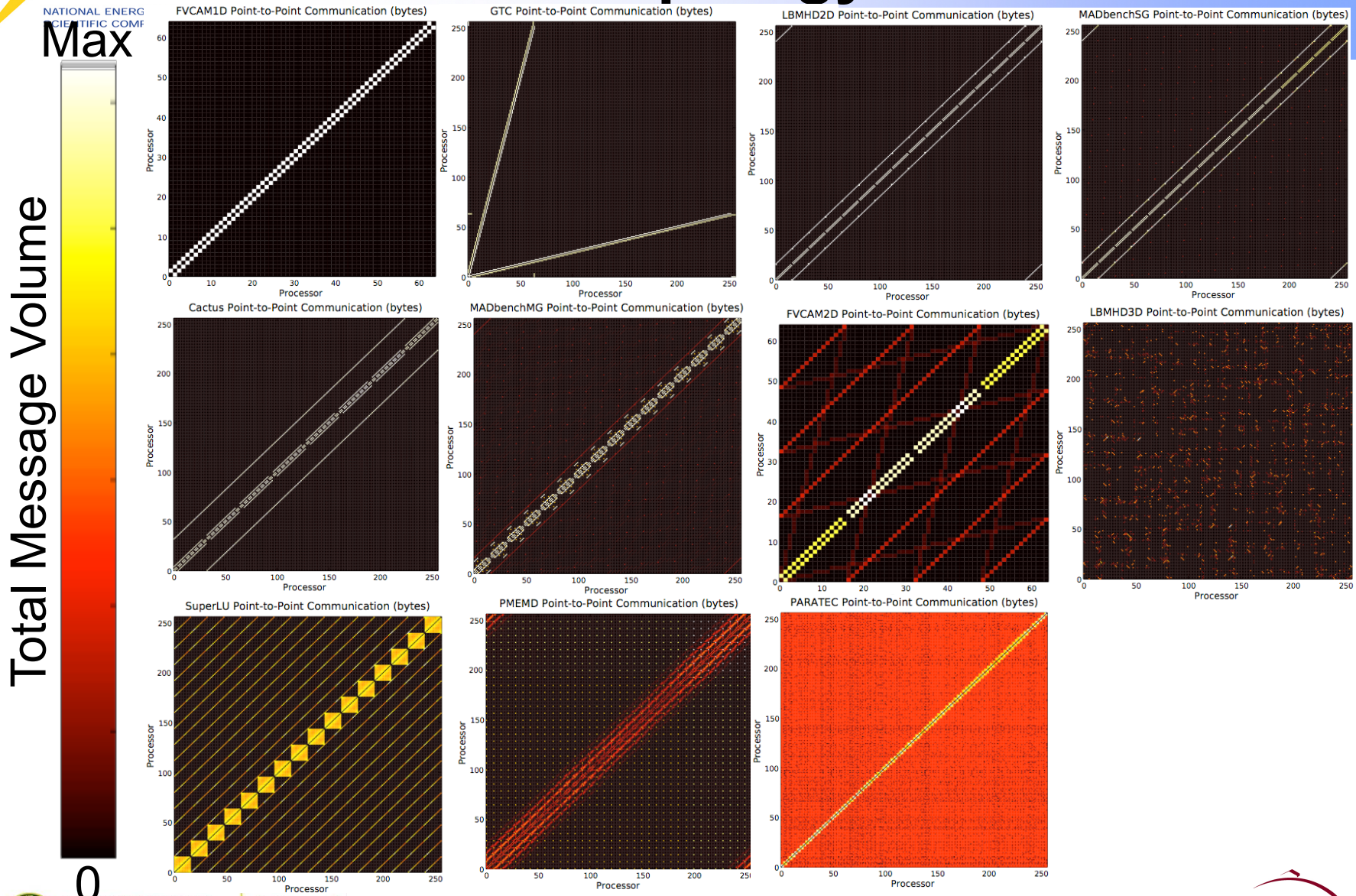




U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

P2P Topology Overview



NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

NERSC
NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

Parallel Programming in the future with many-core architectures.

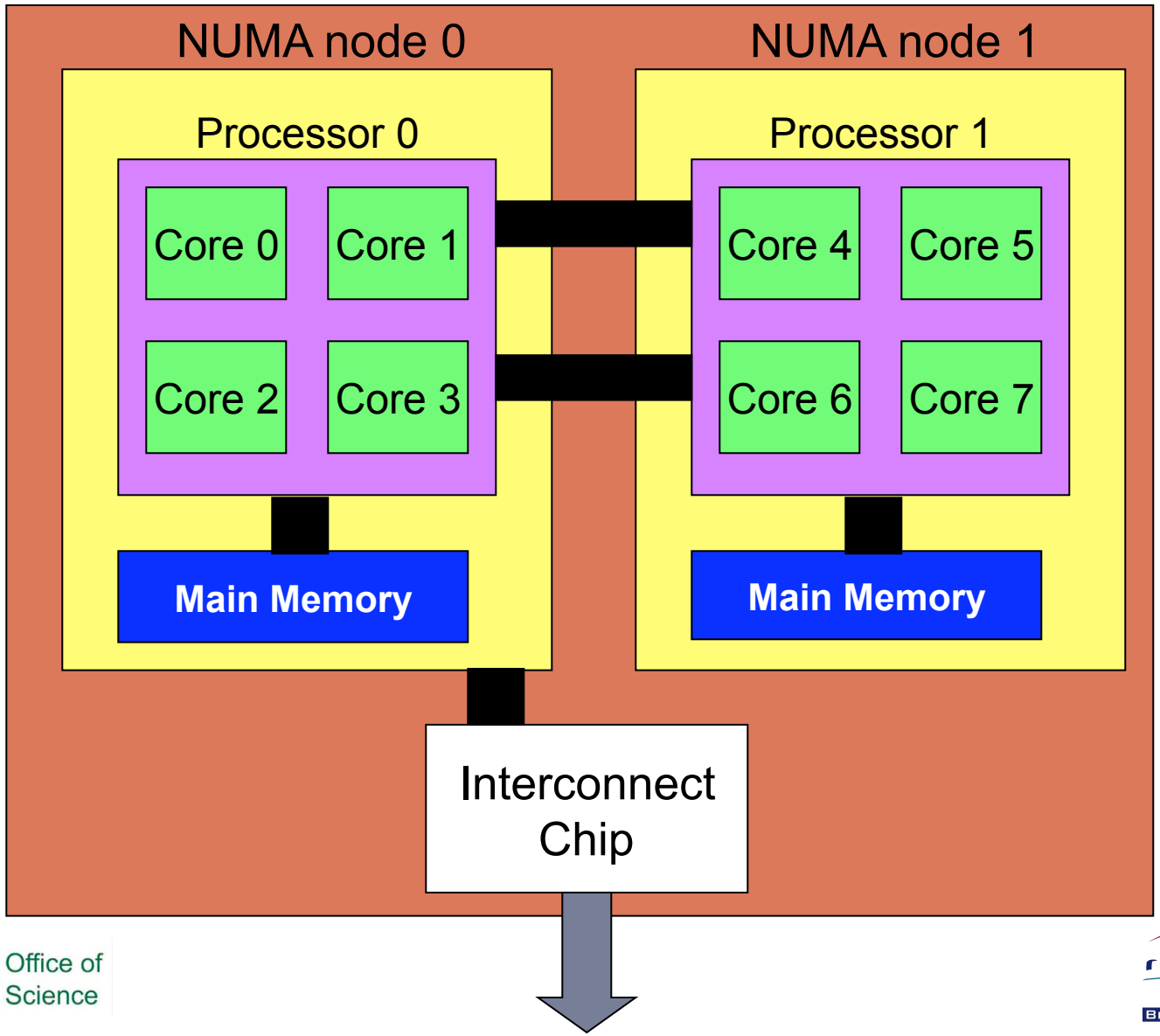
Can MPI everywhere survive?

Slides for John Shalf's Future Architecture's Talk

U.S. DEPARTMENT OF ENERGY | Office of Science

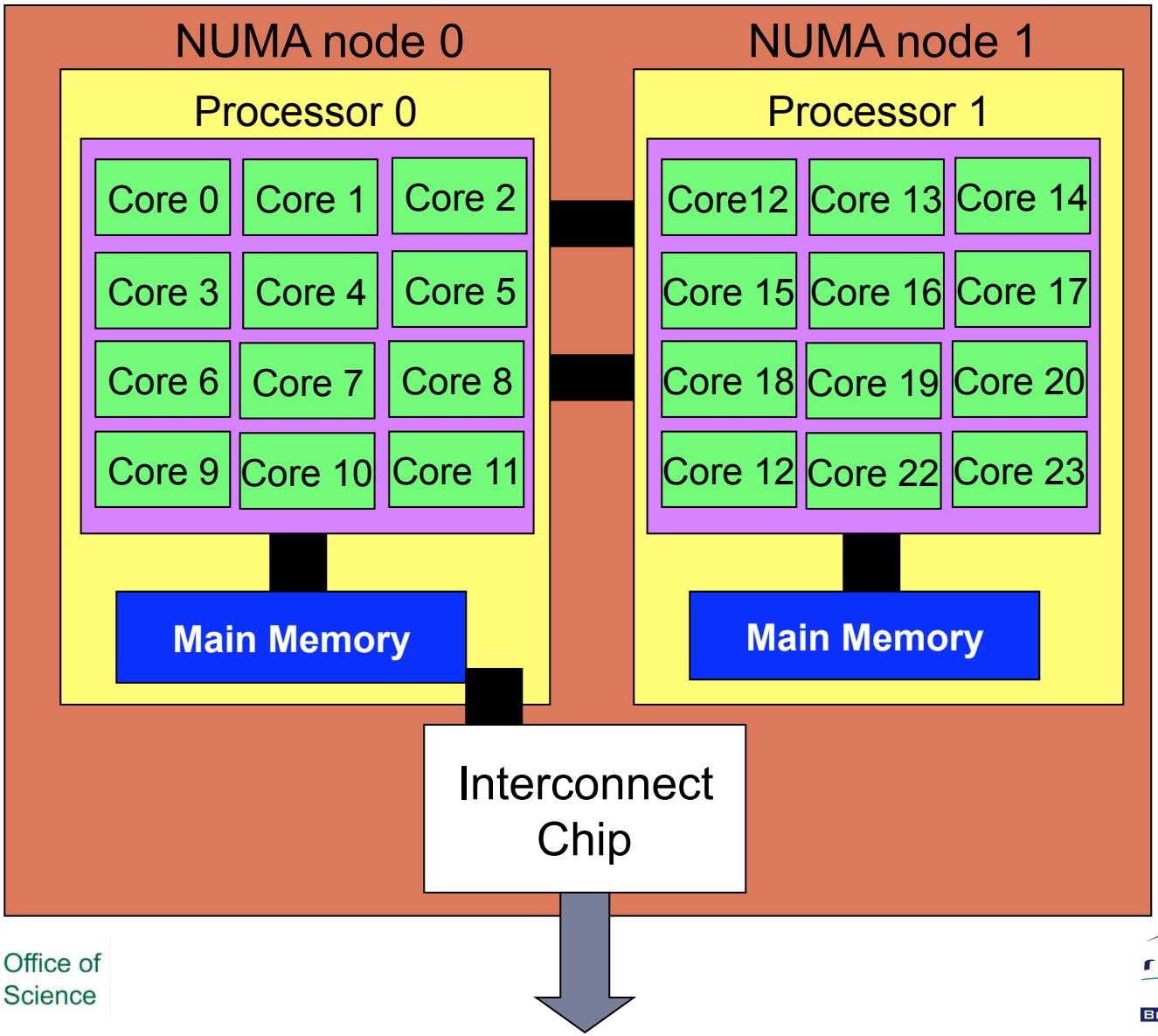
BERKELEY LAB

Basic Multi-core Compute Node Architecture



NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

Coming Soon ….

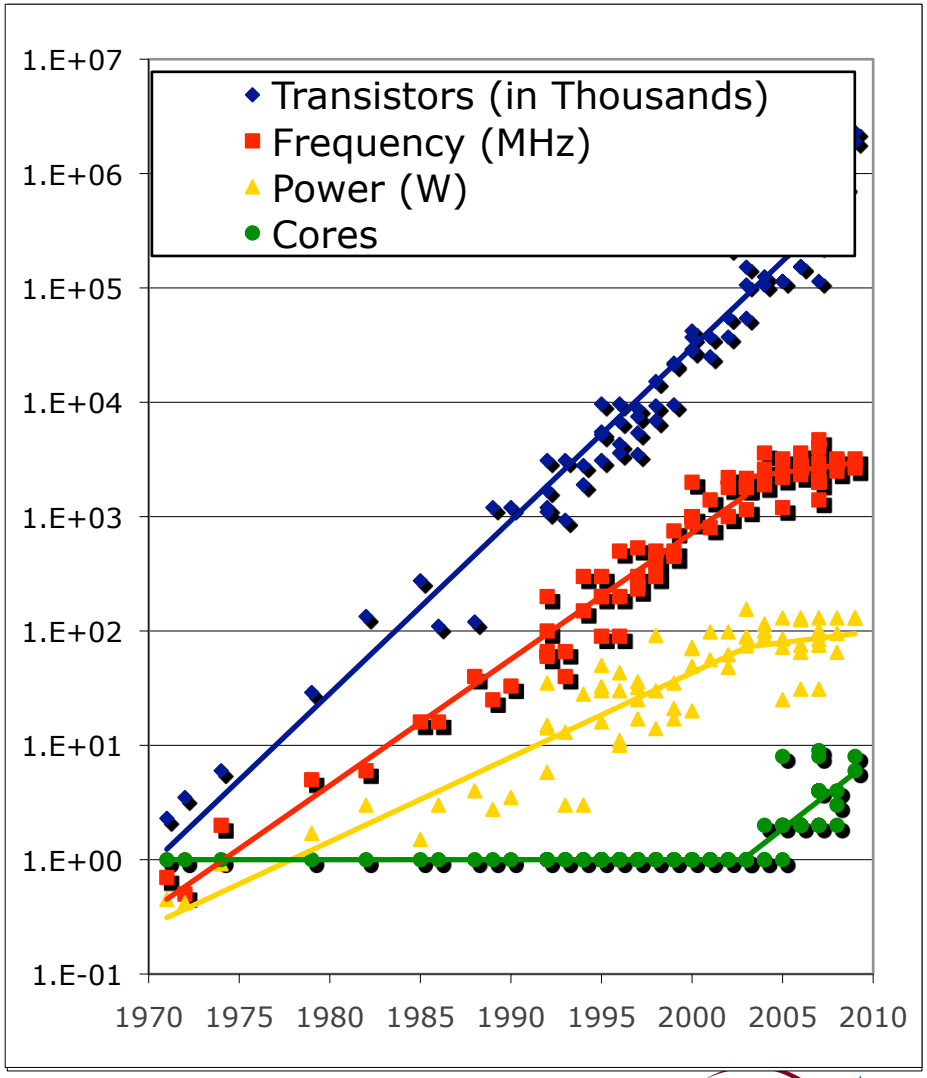


U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

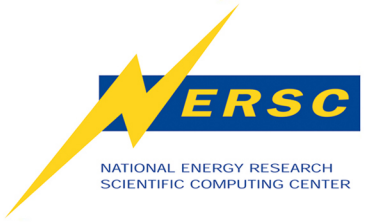
Trend #3: Multicore / Manycore

- Power density limit single processor clock speeds
- Cores per chip is growing
- Simple doubling of cores is not enough to reach exascale
  - Also a problem in data centers, laptops, etc.
- Two paths to exascale:
  - Accelerators (GPUs)
  - Low power embedded cores
  - (Not x86 clusters)



What's Wrong with MPI Everywhere

- We can run 1 MPI process per core (flat model for parallelism)
  - This works now and will work for a while
  - But this is wasteful of intra-chip latency and bandwidth (100x lower latency and 100x higher bandwidth on chip than off-chip)
  - Model has diverged from reality (the machine is *NOT* flat)
- How long will it continue working?
  - 4 - 8 cores? Probably. 128 - 1024 cores? Probably not.
  - Depends on performance expectations
- What is the problem?
  - Latency: some copying required by semantics
  - Memory utilization: partitioning data for separate address space requires some replication
    - How big is your per core subgrid? At 10x10x10, over 1/2 of the points are surface points, probably replicated
  - Memory bandwidth: extra state means extra bandwidth
  - Weak scaling: success model for the "cluster era;" will not be for the many core era -- not enough memory per core
  - Heterogeneity: MPI per CUDA thread-block?



However: MPI will likely persist

- Obviously MPI will not disappear in five years
- By 2014 there will be 20 years of legacy software in MPI
- Thus far, new systems are not sufficiently different to lead to new programming model
- MPI can evolve – (like Fortran, the Fortran from 50 years ago is very different from the Fortran used today)

Why use Hierarchical (hybrid) model for parallelism?

- **The machine is not flat**
  - We lose a lot of performance by lying to ourselves
- **Target: Get Strong scaling on-chip and weak-scaling off-chip**
  - 100x higher bandwidth between cores on chip
  - 100x lower latency between cores on chip
  - If you pretend that every core is a peer (each is just a generic MPI rank) you are leaving a lot of performance on the table
  - You cannot domain-decompose things forever (cannot weak-scale forever)
- **Potentially MPI between nodes and *X* within node**
  - Where X could be OpenMP, UPC, OpenCL, CUDA, etc…

What is X?

- X is it OpenMP?
  - Lots of synchronization
  - Poor expression of locality (will not scale)
- X might be UPC or PGAS language
  - Explicit definition of local vs. remote
  - Very lightweight communication
- X might be CUDA or OpenCL
  - OpenCL is very CUDA-like cross-platform extension to C language
  - CUDA is also being extended to also taret multicore
- For all X
  - Define better way to express fine-grained parallelism on-chip
  - must rigorously determine semantics for interoperation with MPI
  - Must interoperate with numerical methods that target strong scaling

MPI+X: Requirements for X

- **Must be able to write once and run everywhere**
  - Cannot develop architecture-specific code
  - Don't want to write code for each target! (just once please)
- **Needs to be ubiquitous**
  - Most people start a new code on a laptop and graduate to HPC systems
  - The complete development environment must be in both places (freely available)
- **Must emphasize ability to deliver strong-scaling on-chip to replace clock-frequency scaling**
  - Data parallelism might not be sufficient
  - We cannot rely on domain-decomposition for speed-up ad-infinitum (nothing to take up slack for CFL)
  - Functional partitioning (Happening at macro-scale with frameworks At micro-scale, requires bounded side-effects! its not magic)

Summary

- **Strong scaling on chip**
  - Memory is shrinking per chip and clocks stalled
  - Solutions: UPC on-chip, OpenCL, domain-specific code-generation
  - Not-solutions: CUDA, OpenMP (not locality aware)
- **Weak scaling between chips**
  - Memory size is staying same per node
  - Probably MPI, but could be UPC, PGAS or other distributed memory locality aware models
- **Frameworks for managing big programming teams**
  - Should focus on modularity and agreement on interfaces
  - Benefits from functional semantics
- **Languages for fine-grained parallelism + correctness**
  - Defining exec model for fine-grained explicit parallelism is the challenge of our decade

More Info

- **The Berkeley View/Parlab**
  - http://view.eecs.berkeley.edu
  - http://parlab.eecs.berkeley.edu/
- **NERSC System Architecture Group**
  - http://www.nersc.gov/projects/SDSA
- **LBNL Future Technologies Group**
  http://crd.lbl.gov/ftg

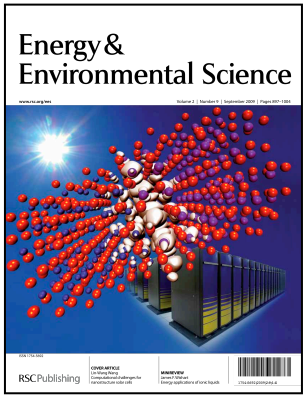
NERSC
NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER

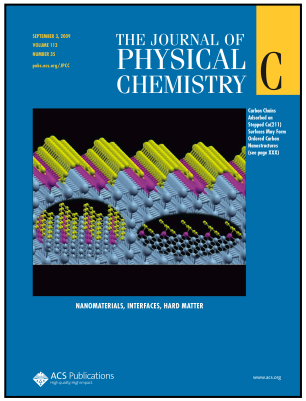
Extra

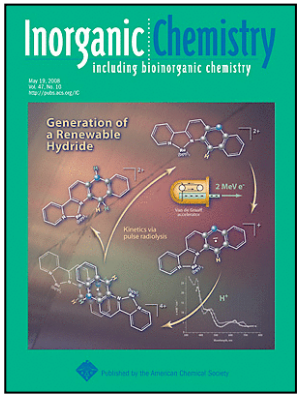
U.S. DEPARTMENT OF ENERGY | Office of Science

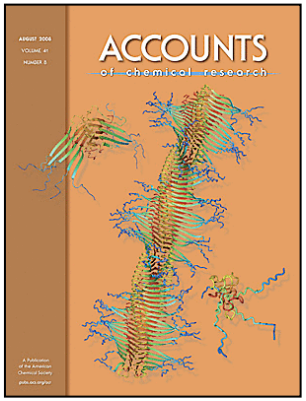
BERKELEY LAB

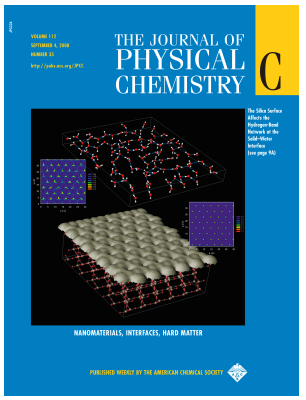
Cover Stories from NERSC Research

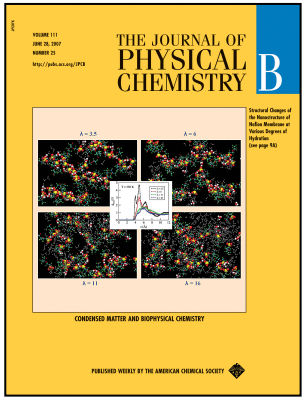


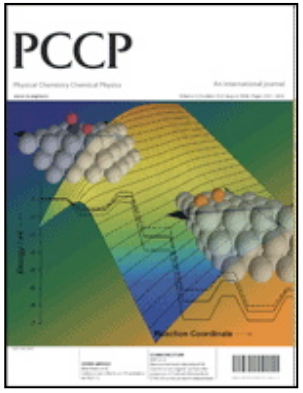












NERSC is enabling new high quality science across disciplines, with over *1,600 refereed publications* last year