

Lecture IN-2147 Parallel Programming

SoSe 2018

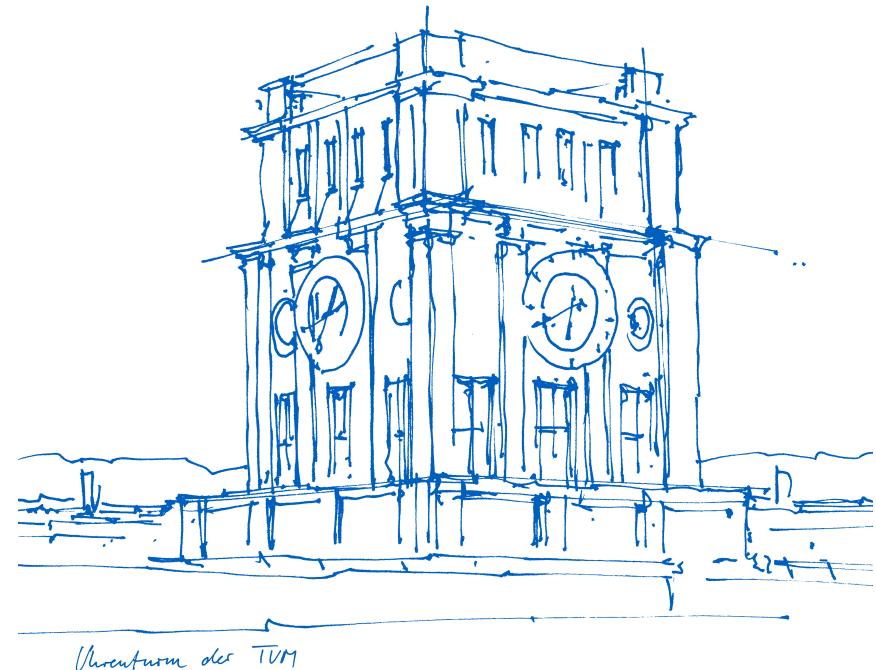
Martin Schulz

Exercises: Amir Raoofy

Technische Universität München

Fakultät für Informatik

Lecture 7:
HPC & SuperMUC
Introduction into MPI



Administrative Topics

Exam:

July 24th, 2018, 16:00 to 17:30 in MW 2001

please double check in TUM-Online

Please register in TUM-Online by June 6th

Repetition Exam:

October 5th, 2018, 11:00 to 12:30 in MW 1801

please double check in TUM-Online

Please register in TUM-Online by September 24th

Lecture Evaluation (June 6th to June 20th)

You will get an online link, please fill it out

We will provide an extra 15min time on June 18th

Summary from Last Time

Covered remaining topics for OpenMP

- Tasking to allow for more programmability
- OpenMP memory model and flush directive
- Device construct covered later

OpenMP Reduction

```
#pragma omp parallel for reduction(+: a)
for (int j=0; j<4; j++)
{
    a = omp_get_thread_num();
}
printf("Final Value of a=%d\n", a);
```

OMP_NUM_THREADS=4

Final value of a:

$$0+1+2+3 = 6$$

Clarification: OpenMP Reductions

OpenMP offers a special clause to specify reductions

reduction(*operator*: *list*)

This clause performs a reduction ...

- ... on the variables that appear in *list*,
- ... with the operator *operator* ...

Variables must be shared scalars

operator is one of the following:

+, *, -, &, ^, |, &&, ||

Reduction variable can only appear in statements with the following form:

- **x = x *operator* expr**
- **x binop= expr**
- **x++, ++x, x--, --x**

User defined reductions are an option in newer versions of OpenMP

Clarification: OpenMP Reductions

OpenMP offers a special clause to specify reductions

reduction(*operator*: *list*)

This clause performs a reduction ...

- ... on the variables that appear in *list*,
- ... with the operator *operator* ...
- ... across the values “at thread end”/“last iteration“ of **each thread (!)**

Variables must be shared scalars

operator is one of the following:

+, *, -, &, ^, |, &&, ||

Reduction variable can only appear in statements with the following form:

- **x = x *operator* expr**
- **x binop= expr**
- **x++, ++x, x--, --x**

User defined reductions are an option in newer versions of OpenMP

Example: Reduction

```
#pragma omp parallel for reduction(+: a)
for (int j=0; j<4; j++)
{
    a = omp_get_thread_num();
}
printf("Final Value of a=%d\n", a);
```

OMP_NUM_THREADS=4

Final value of a:

$$0+1+2+3 = 6$$

OMP_NUM_THREADS=2

Final value of a:

$$0+1 = 1$$

Example: Reduction (cont.)

```
#pragma omp parallel for reduction(+: a)
for (int j=0; j<100; j++)
{
    a = j;
}
printf("Final value of a=%d\n", a);
```

OMP_NUM_THREADS=4

Final value of a:

$$24+49+74+99 = 246$$

Thread 0:	0-24	last value is 24
Thread 1:	25-49	last value is 49
Thread 2:	50-74	last value is 74
Thread 3:	75-99	last value is 99

Example: Reduction (cont.)

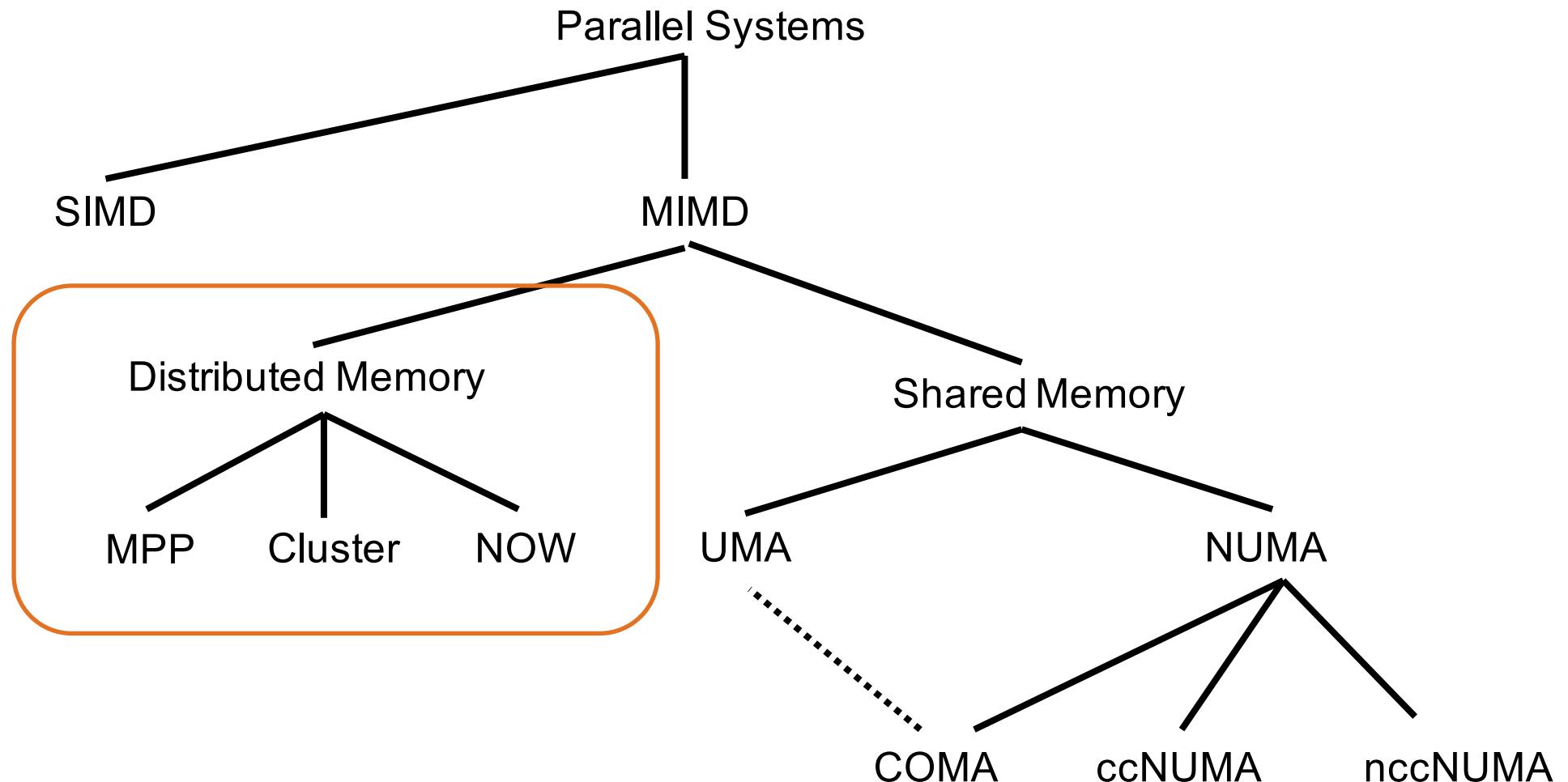
```
a=0
#pragma omp parallel for reduction(+: a)
for (int j=0; j<100; j++)
{
    a += j;
}
printf("Final value of a=%d\n", a);
```

Final value of a:

$$0+1+2+\dots+99 = (99*100)/2 = 4950$$

(for any number of threads)

Distributed Memory Machines



Summary from Last Time

Covered remaining topics for OpenMP

- Tasking to allow for more programmability
- OpenMP memory model and flush directive
- Device construct covered later

Distributed Memory Architectures

- Highly scalable to thousands (or more) nodes
- Network architecture and topology play an important role

Architectural choice has significant impact on programmability and usability

- Need a different programming model with explicit communication
- Need system software to tie nodes together
- Access to such a shared environment is more controlled

A “Close By” Example: LRZ’s SuperMUC



A photograph of the SuperMUC supercomputer at Leibniz Rechenzentrum (LRZ). The room is filled with rows of tall, black server racks. On top of these racks are yellow and orange components, likely network or storage hardware. A person stands in the center-left of the image, providing a sense of scale to the massive array of equipment. The floor is a light-colored tile, and the ceiling has a grid of fluorescent lights.

44	Leibniz Rechenzentrum Germany	SuperMUC - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM/Lenovo	147,456	2,897.0	3,185.1	3,423
45	Leibniz Rechenzentrum Germany	SuperMUC Phase 2 - NeXtScale nx360M5, Xeon E5-2697v3 14C 2.6GHz, Infiniband FDR14 Lenovo/IBM	86,016	2,813.6	3,578.3	1,481

SuperMUC is a Distributed Memory Architecture

18 partitions called islands with 512 nodes

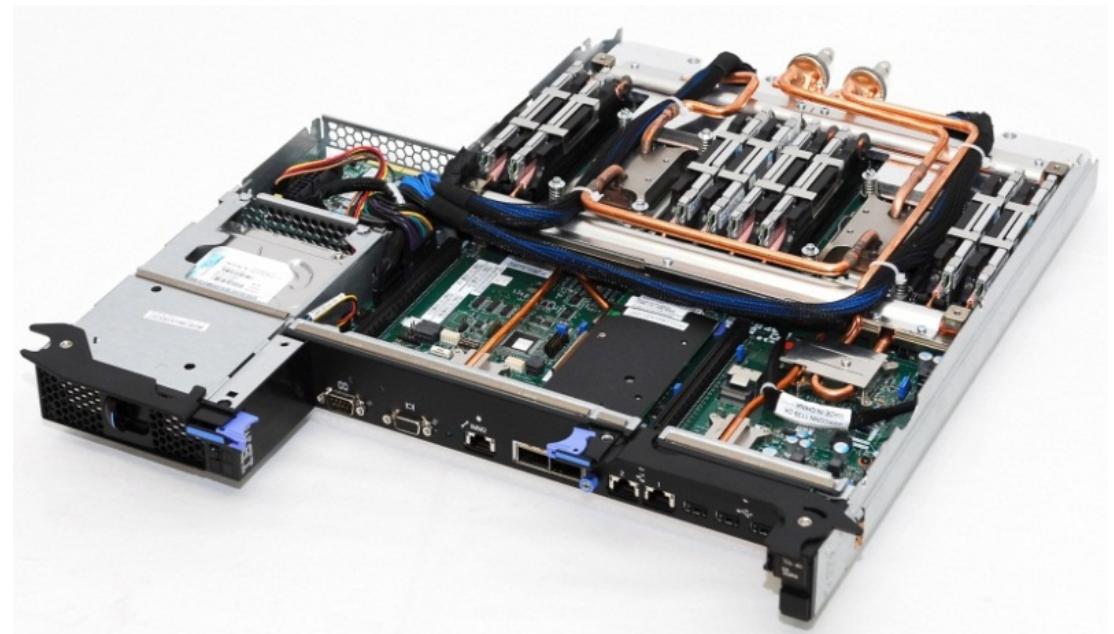
Node is a shared memory system
with 2 processors

- Sandy Bridge-EP
Intel Xeon E5-2680 8C
 - 2.7 GHz (Turbo 3.5 GHz)
- 32 GByte memory
- Inifiniband network interface

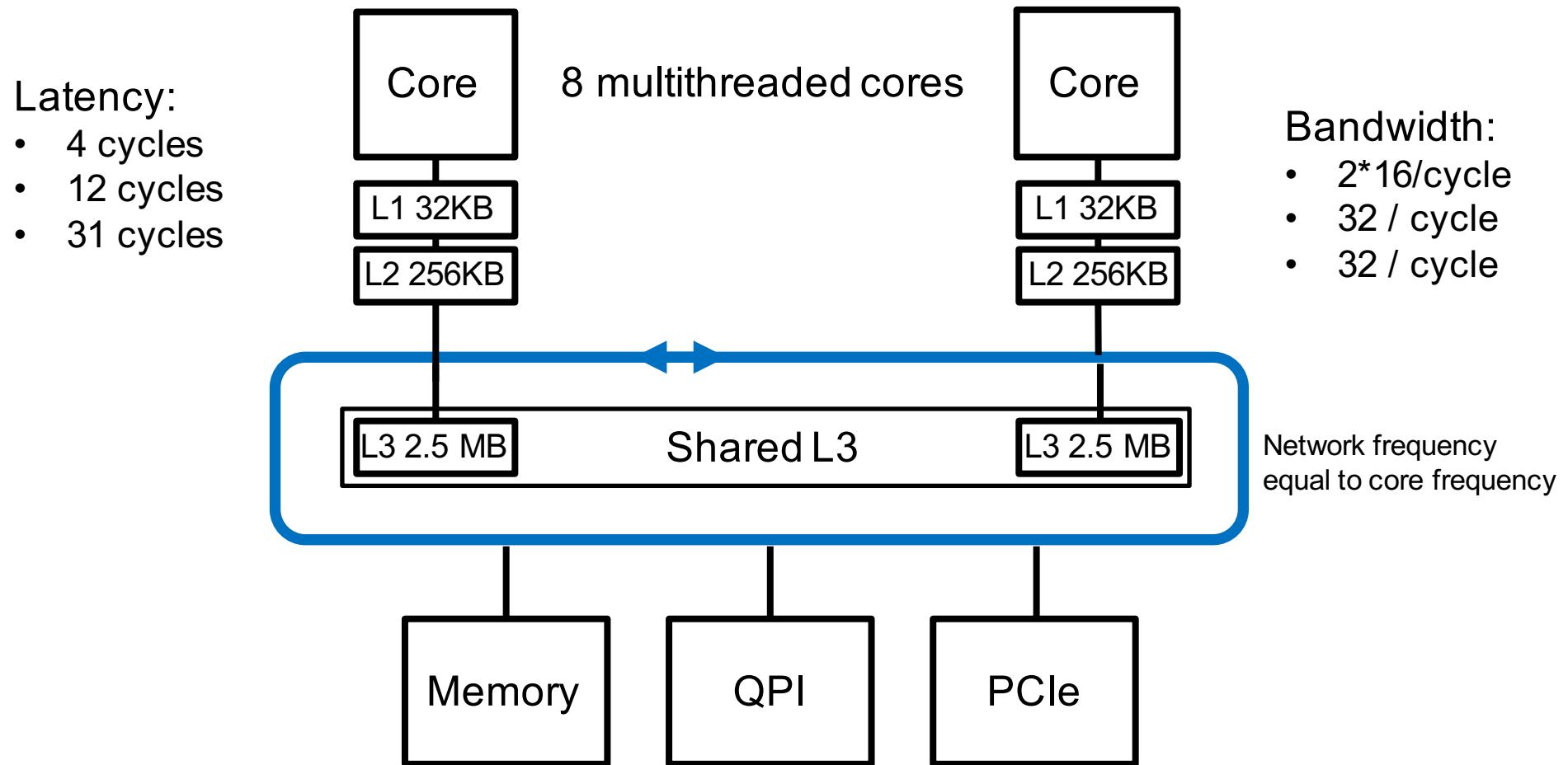
Processor/Socket has 8 cores

- 2-way hyperthreading
- 21.6 GFlops @ 2.7 GHz per core
- 172.8 GFlops per processor

NUMA architecture



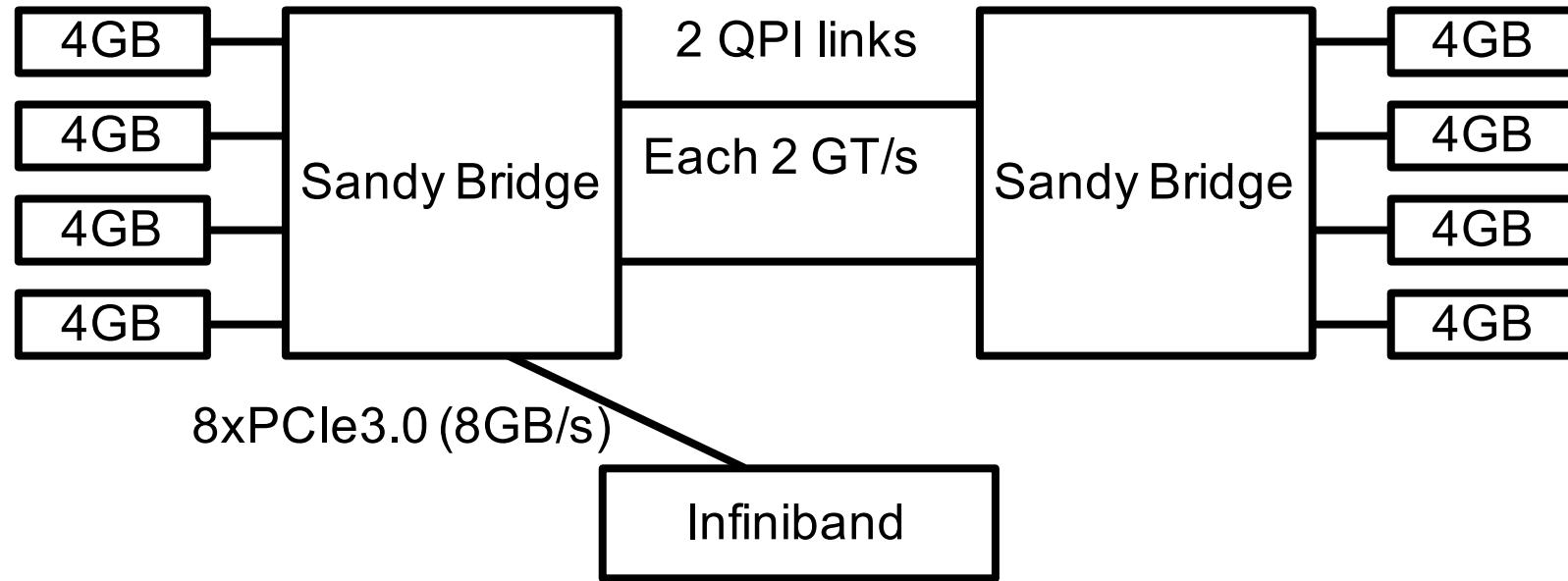
Sandy Bridge Processor



L3 cache

- Partitioned with cache coherence
- Physical addresses distributed by a hash function

NUMA Node



2 processors with 32 GB of memory

Aggregate memory bandwidth per node 102.4 GB/s

Latency

- local ~50ns (~135 cycles @2.7 GHz)
- remote ~90ns (~240 cycles)

9288 Compute Nodes

Picture:
Cold Corridor

Infiniband (red)
and
Ethernet (green)
cabling



Interconnection Network

Infiniband FDR-10

- FDR means Fourteen Data Rate
- FDR-10 has an effective data rate of 38.79 Gbit/s
- Latency: 100 nsec per switch, 1usec MPI
- Vendor: Mellanox

Intra-Island Topology: non-blocking tree

- 256 communication pairs can talk in parallel.

Inter-Island Topology: Pruned Tree 4:1

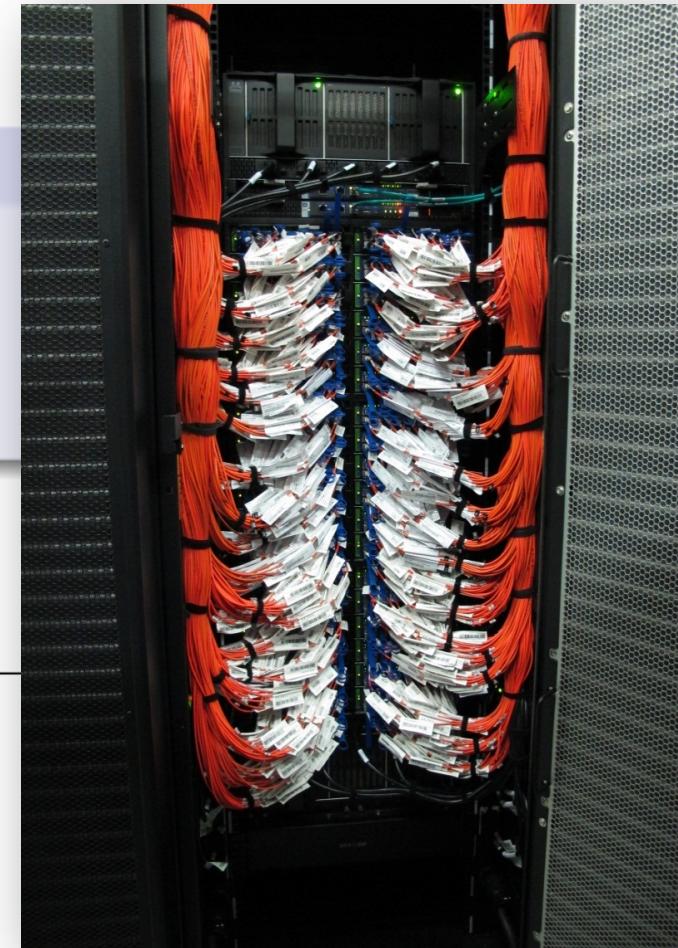
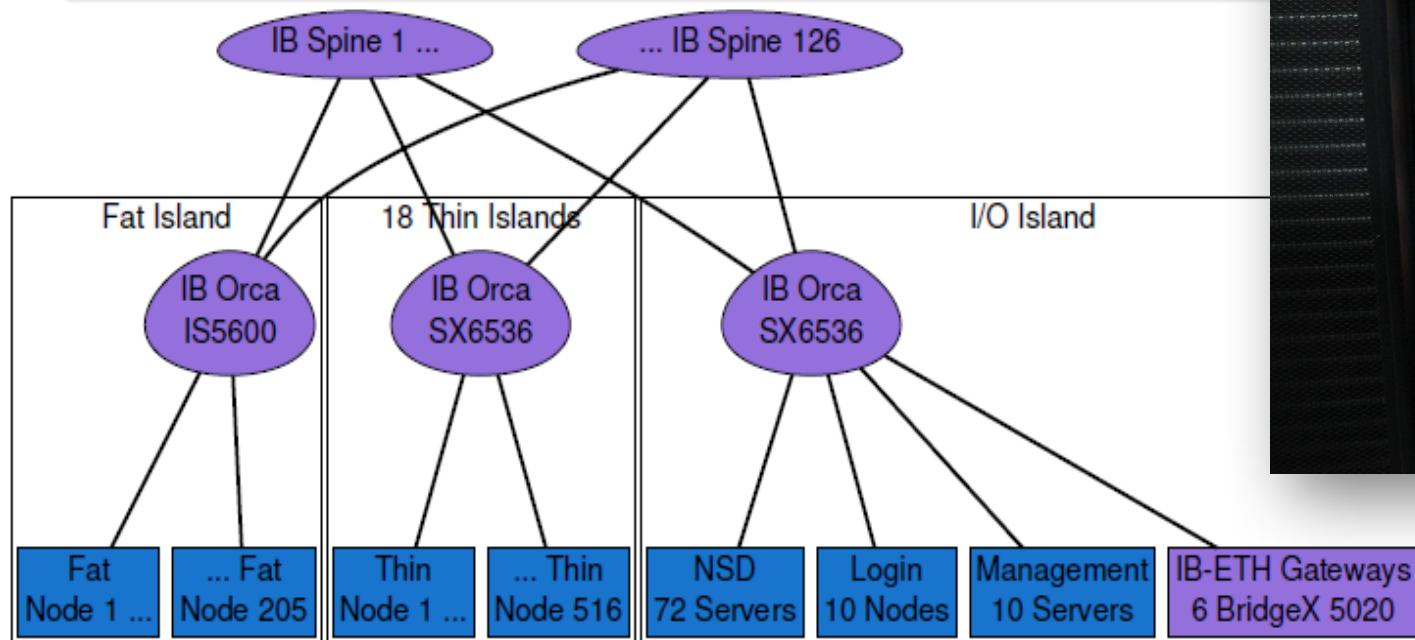
- 128 links per island to next level

Infiniband Interconnect

Pruned Fat Tree

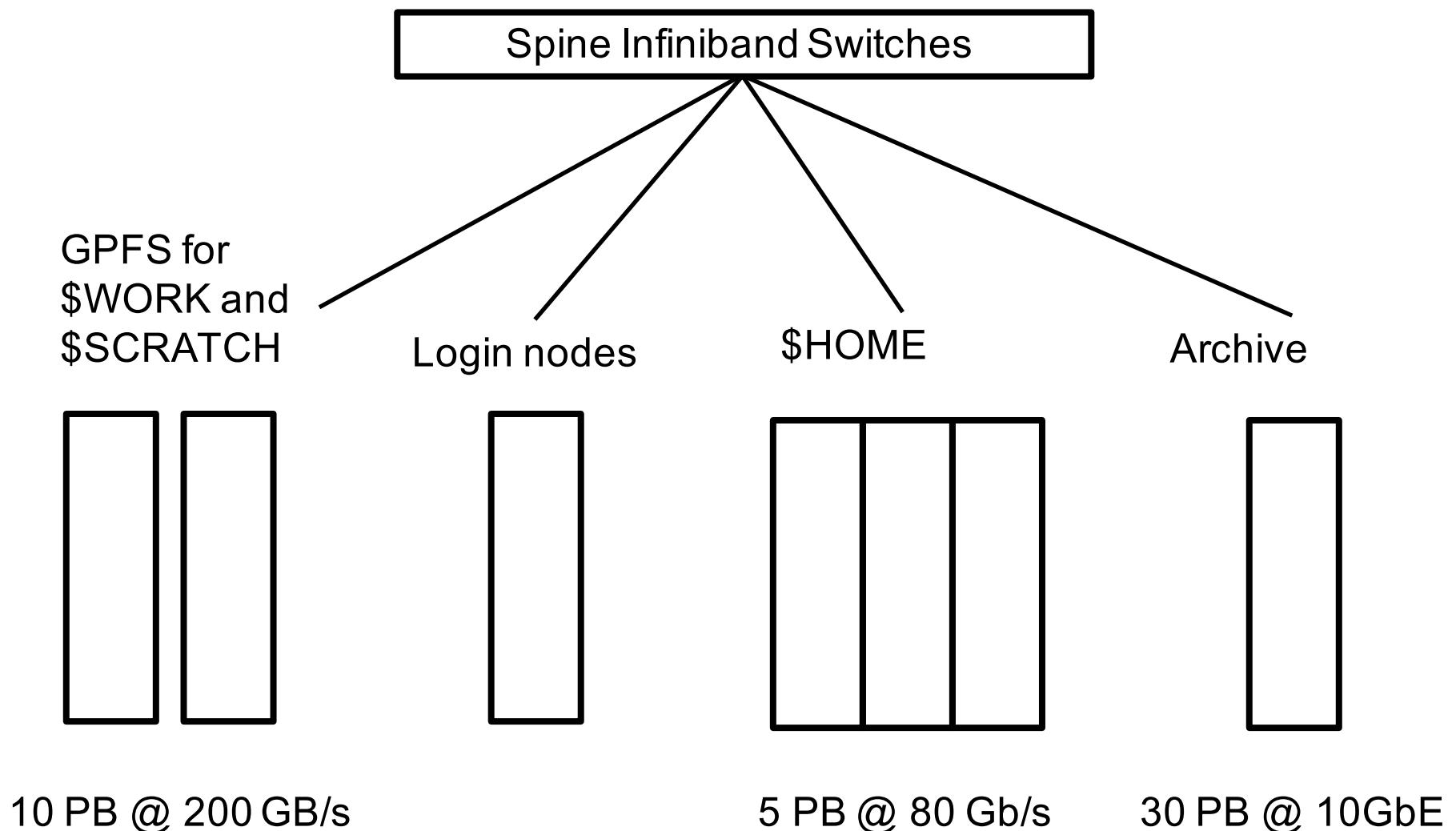
Simplified Illustration

- 126 Spine Switches (Mellanox SX6036, FDR10)
- 1 Island with 205 Fat Nodes (QDR)
- 18 Islands with 516 Thin Nodes each (FDR10)



In total 11900
Infiniband
Cables

I/O System



Parallel File System GPFS



10 Pbyte, 200 GigaByte/s I/O Bandwidth

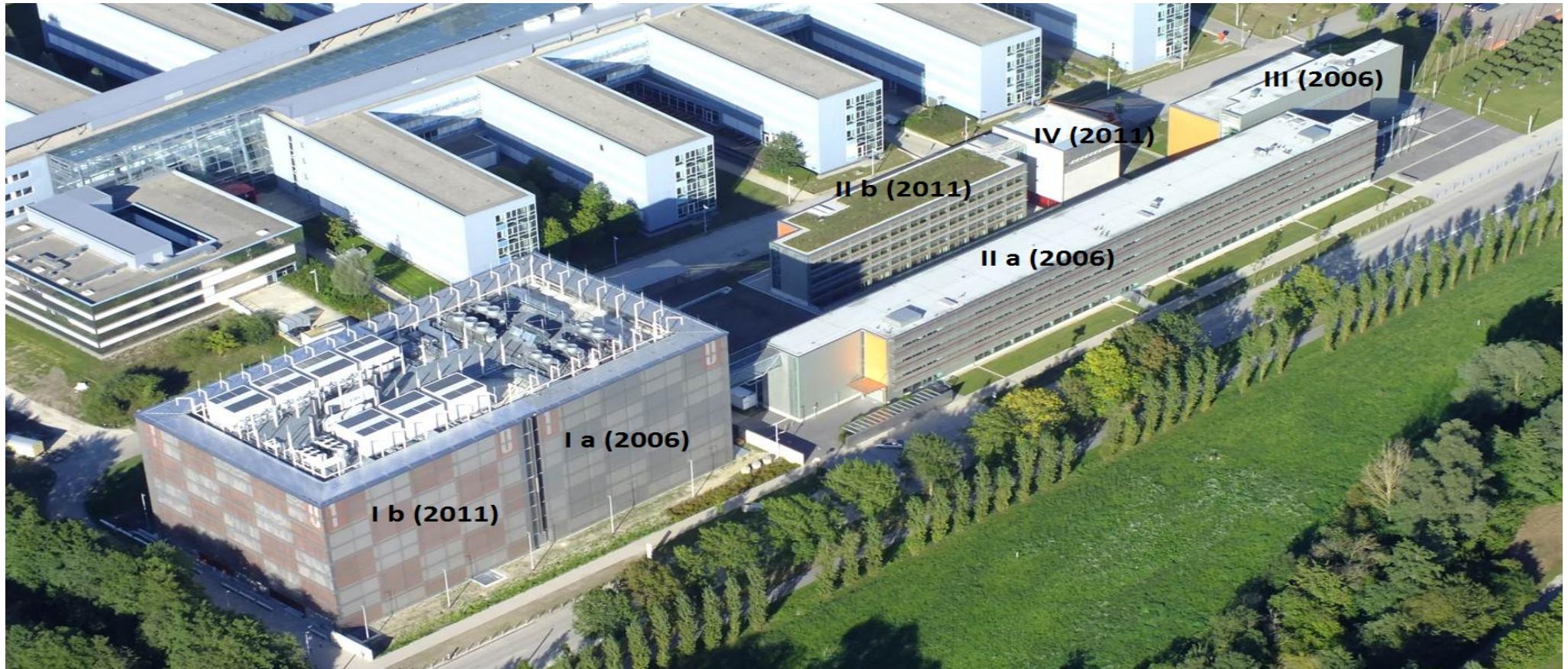


9 DDN SFA 12k Controller
5040 3 TByte SATA Disks

SuperMUC Costs

	2010-2014 Phase 1	2014-2016 Phase 2
High End System		
Investment Costs (Hardware and Software)	53 Mio €	~ 19 Mio €
Operating Costs (Electricity costs and maintenance for hardware und software, some additional personnel)	32 Mio €	~ 29 Mio €
SUM	85 Mio €	~48 Mio €
Extension Buildings (construction and Infrastructure)	49 Mio €	

The Leibniz Supercomputing Centre (LRZ)



LRZ Data Center Facts



Some more Facts

- **3160.5 m² (34 019 ft²) IT Equipment Floor Space (6 rooms on 3 floors)**
- **6393.5 m² (68 819 ft²) Infrastructure Floor Space**
- **2 x 10 MW 20kV Power Supply**
- **Powered Entirely by Renewable Energy**
- **> 6M € Annual Power Bill**

Towers

Routing

Servers, Network

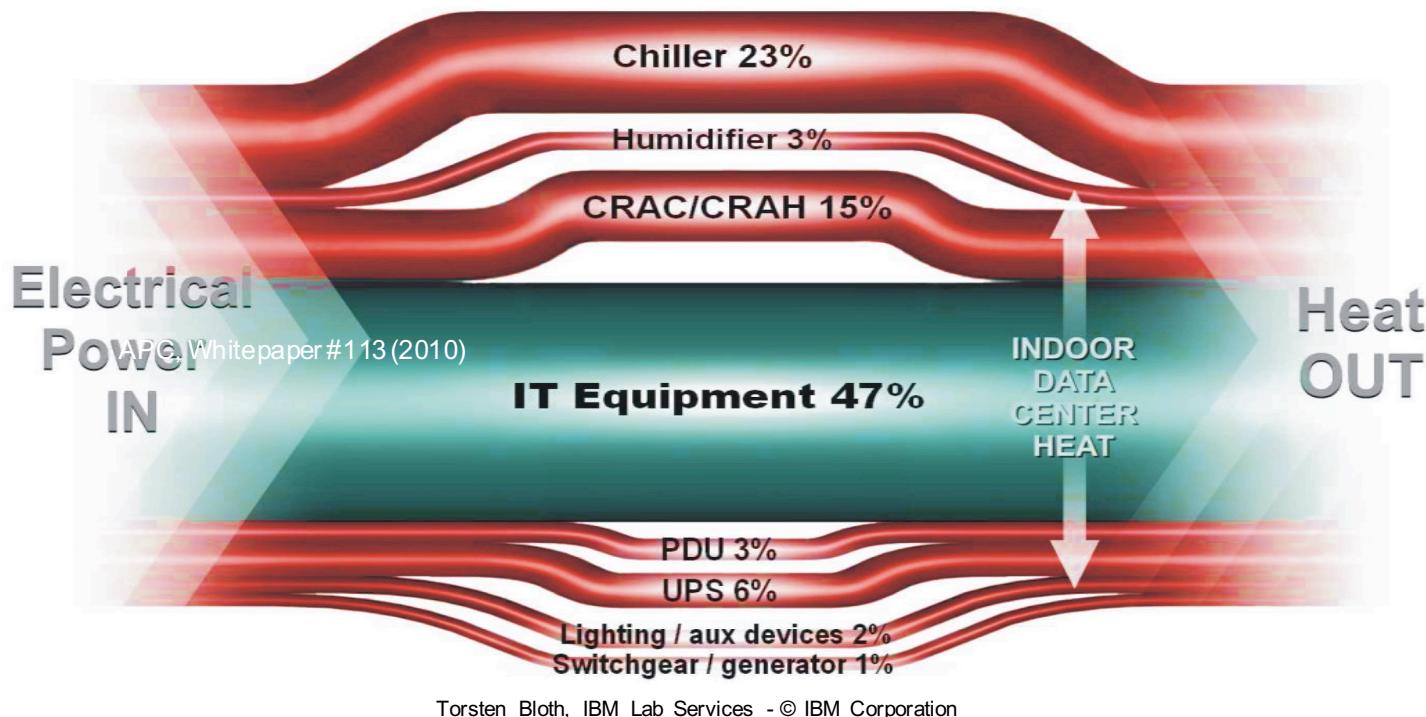
Archive/Backup

Storage Processing

Transformers,

Energy Consumption in Data Centers

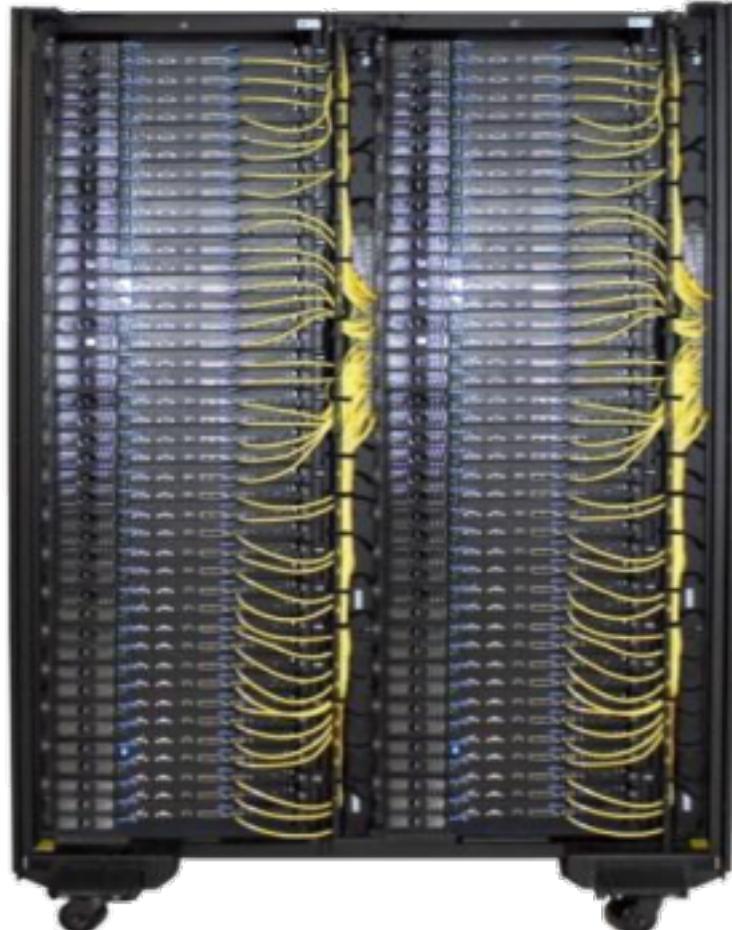
Data Centers are “Heaters with integrated logic”



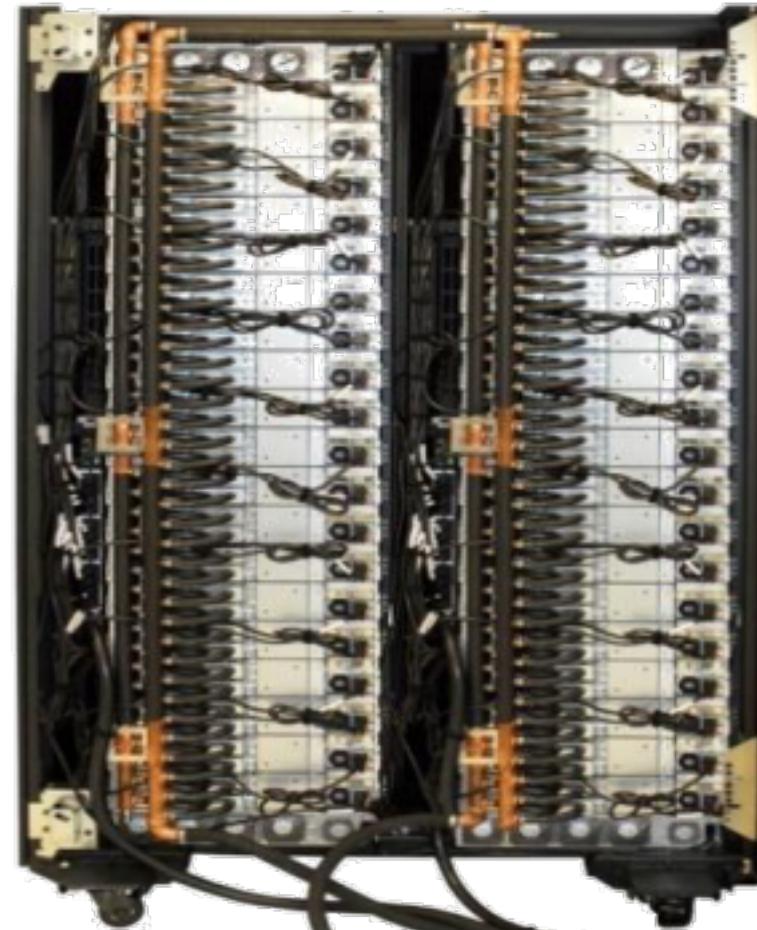
Cooling Setup for SuperMUC at LRZ



IBM System x iDataPlex Direct Water Cooled Rack



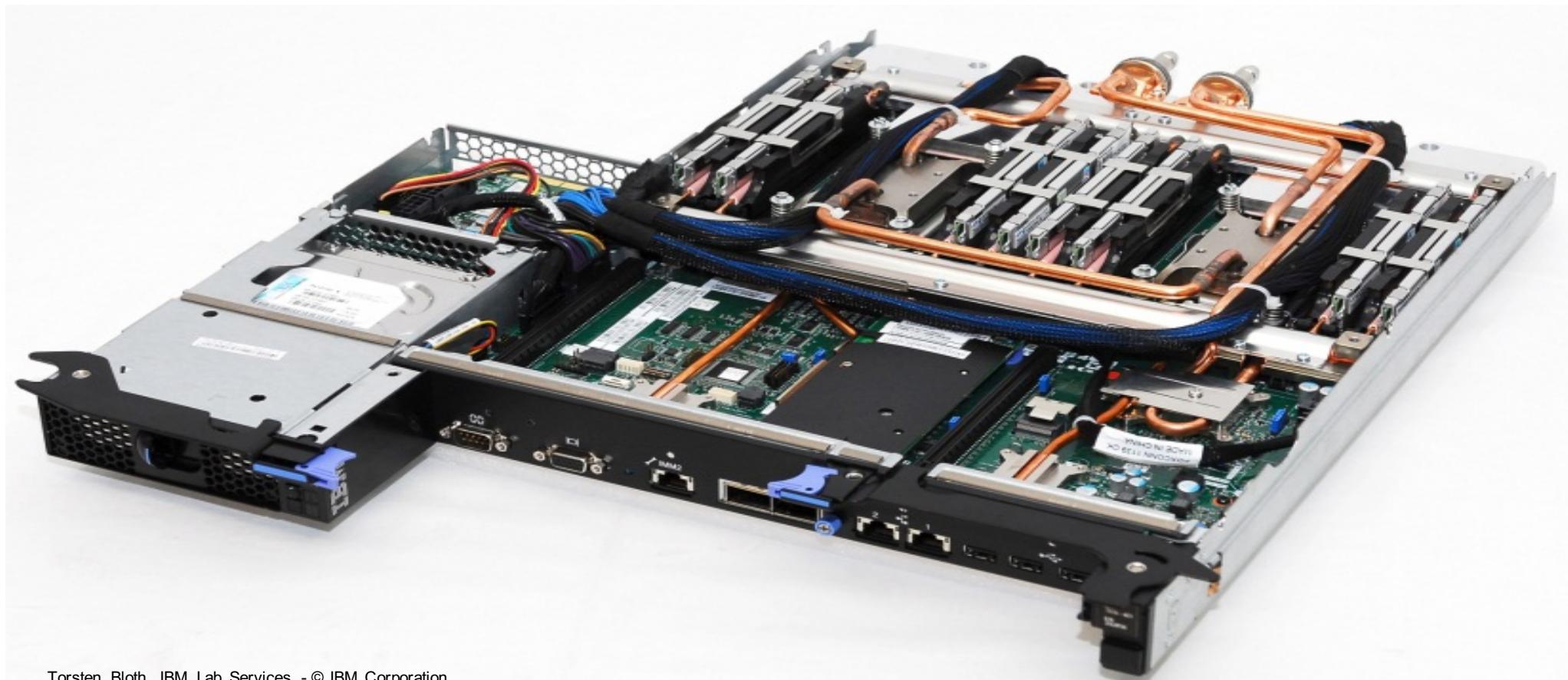
iDataplex DWC Rack
w/ water cooled nodes
(front view)



iDataplex DWC Rack
w/ water cooled nodes
(rear view of water manifolds)



IBM iDataplex dx360 M4



Cooling Infrastructure



Photos: StBAM2 (staatl. Hochbauamt München 2)

Why Is the Infrastructure Important?

- Determines data center overheads
 - Matching average operating power consumption with cooling infrastructure will reduce overheads
- Infrastructure limits the possible cooling technologies for the HPC systems
 - Being set up only for air cooling will not allow an easy switch to water cooled systems
- Trade-offs to reduced overhead can be a source for additional costs later on
 - Switching of power conditioning to reduce overheads can allow brown-outs to shutdown or damage system parts
- Mistakes made here can be costly in the long run
 - HPC system replaced every 3-5 years
 - Infrastructure replaced every 10-20 years

All The Way to Earthquake Safety RIKEN's K Computer





Wide Range of HPC Application Spaces



Predictive Simulation is Becoming a Key Aspect

Climate modeling

Weather forecasting

Nuclear Physics

Oil and Gas

- Reservoir modeling

Bioscience, Medicine

- Genomic research

Material Science

Automobile/Aeronautics Industry

- CFD
- Virtual Crashtests

City planning

Graph analysis

- Security application

Finance



#hpcmatters

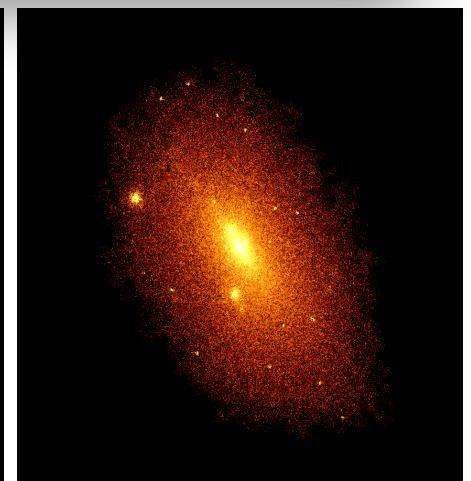
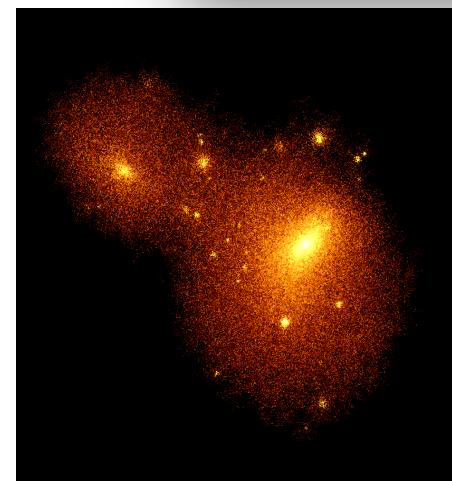
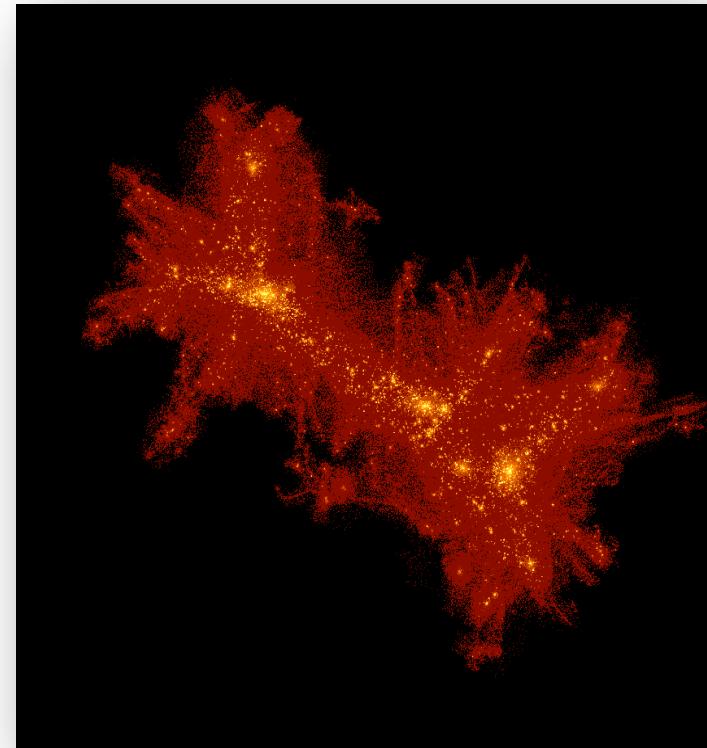
Astrophysics: Simulation of Galaxies

Example / Astrophysikalisches Institut Potsdam

Goal: Study the formation and movement of the non-linear dynamics of galaxies and galaxy clusters

Methods: Highly efficient parallel adaptive refinement Tree (ART) code, which allows to study the influence of small changes in the early universe until today.

High resolution is for precise computations necessary. Only the largest supercomputers can be used for such simulations. Most important is the amount of main memory.

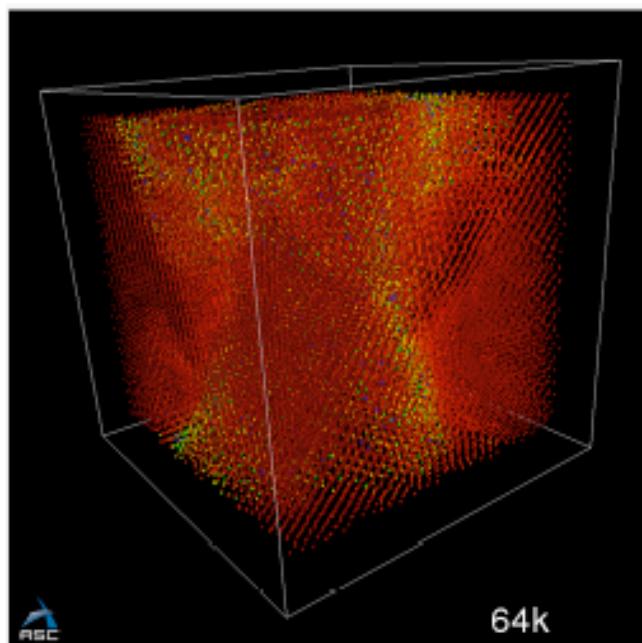


Detail of a cosmological simulation of galaxy formation, credit: S. Gottlöber, A. Klypin, A. Kravtsov

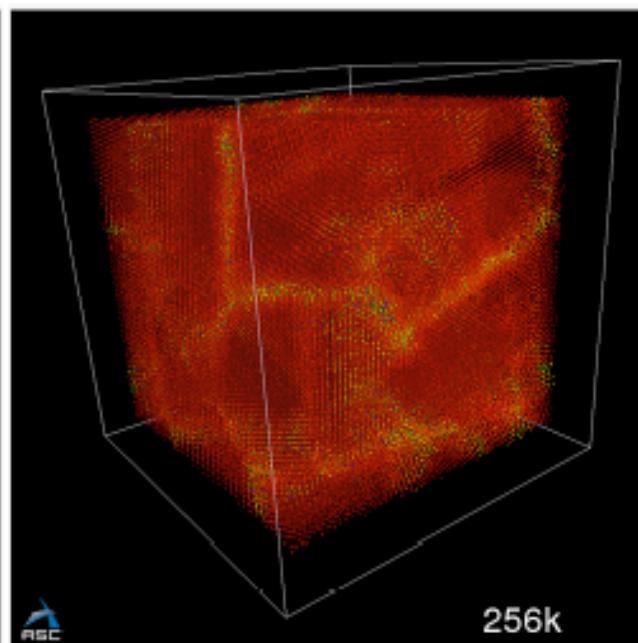
Material Solidification Process

Molecular dynamics at LLNL with ddcMD: 2 Million atom run (2005)

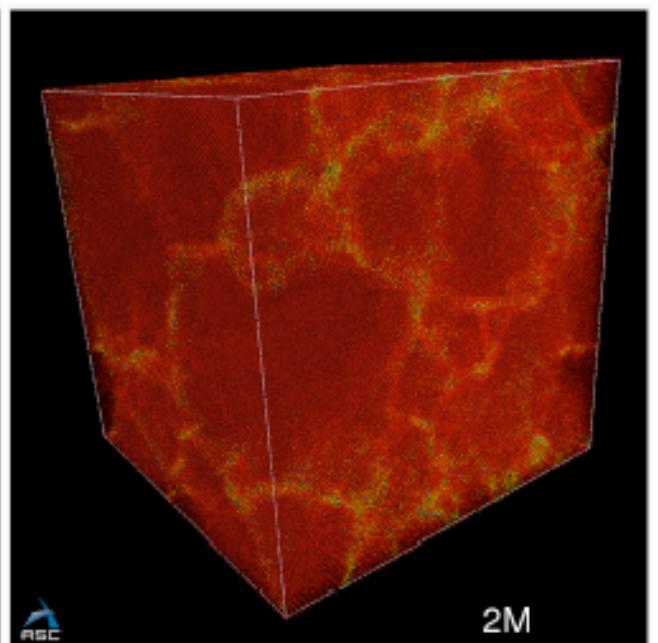
- Used all of Blue Gene/L (128K cores)
- Close to perfect scaling
- New scientific observations



64,000 atoms

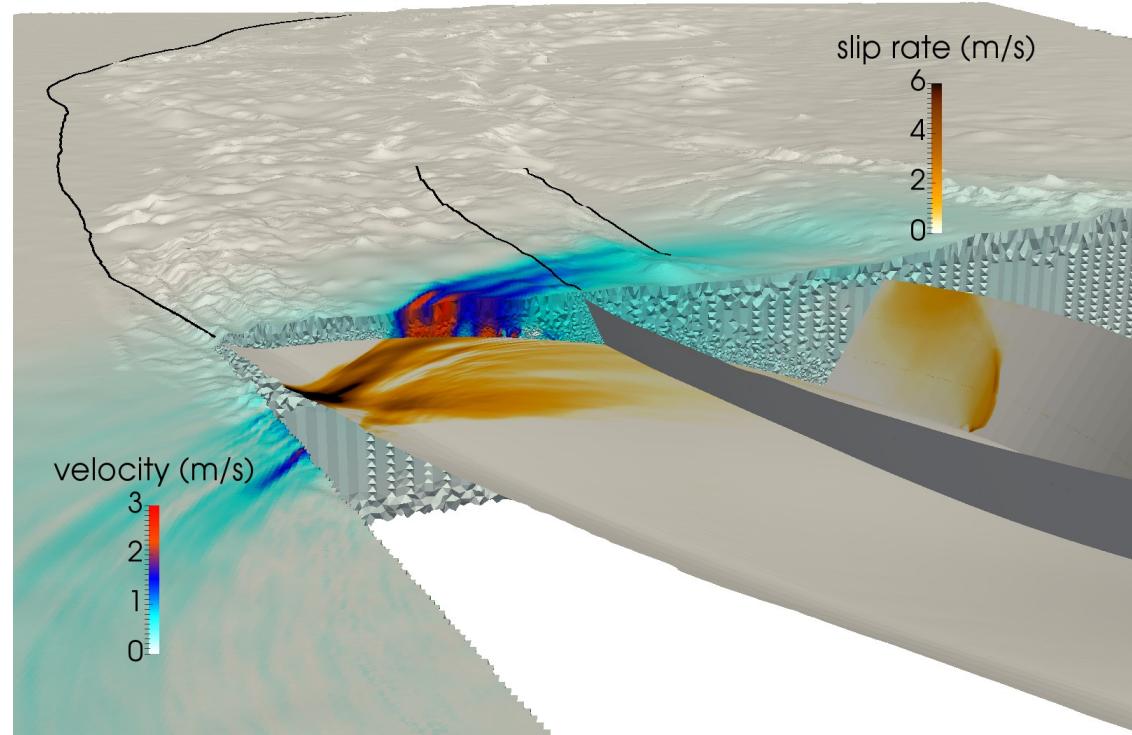
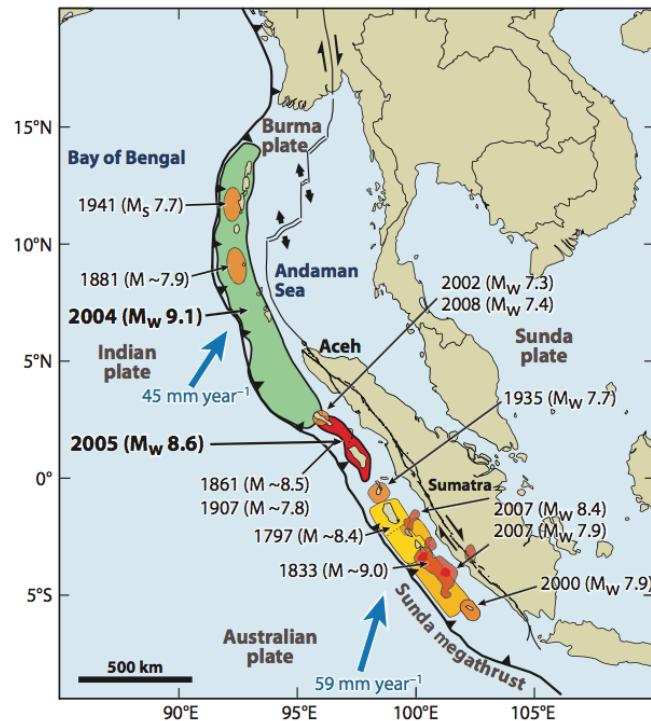


256,000 atoms



2,048,000 atoms

Example – Simulation of the Sumatra Earthquake



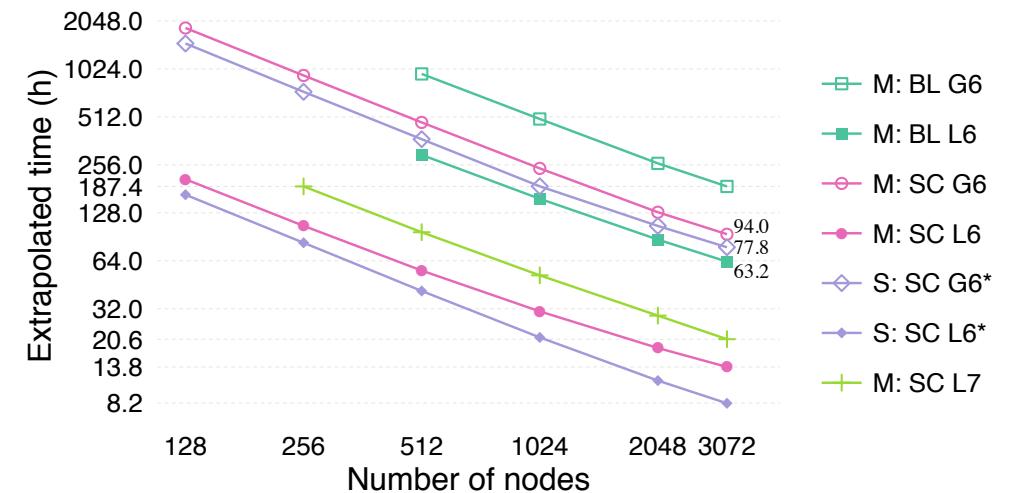
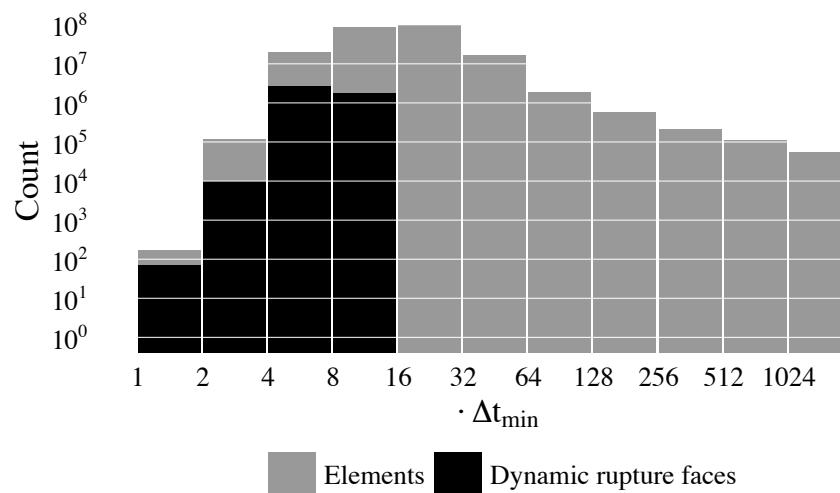
Uphoff et al.: Extreme Scale Multi-Physics Simulations of the Tsunamigenic 2004 Sumatra Megathrust Earthquake, SC17

- simulates rupture process and seismic wave propagation
- high-order discontinuous Galerkin method on an adaptive unstructured tetrahedral mesh

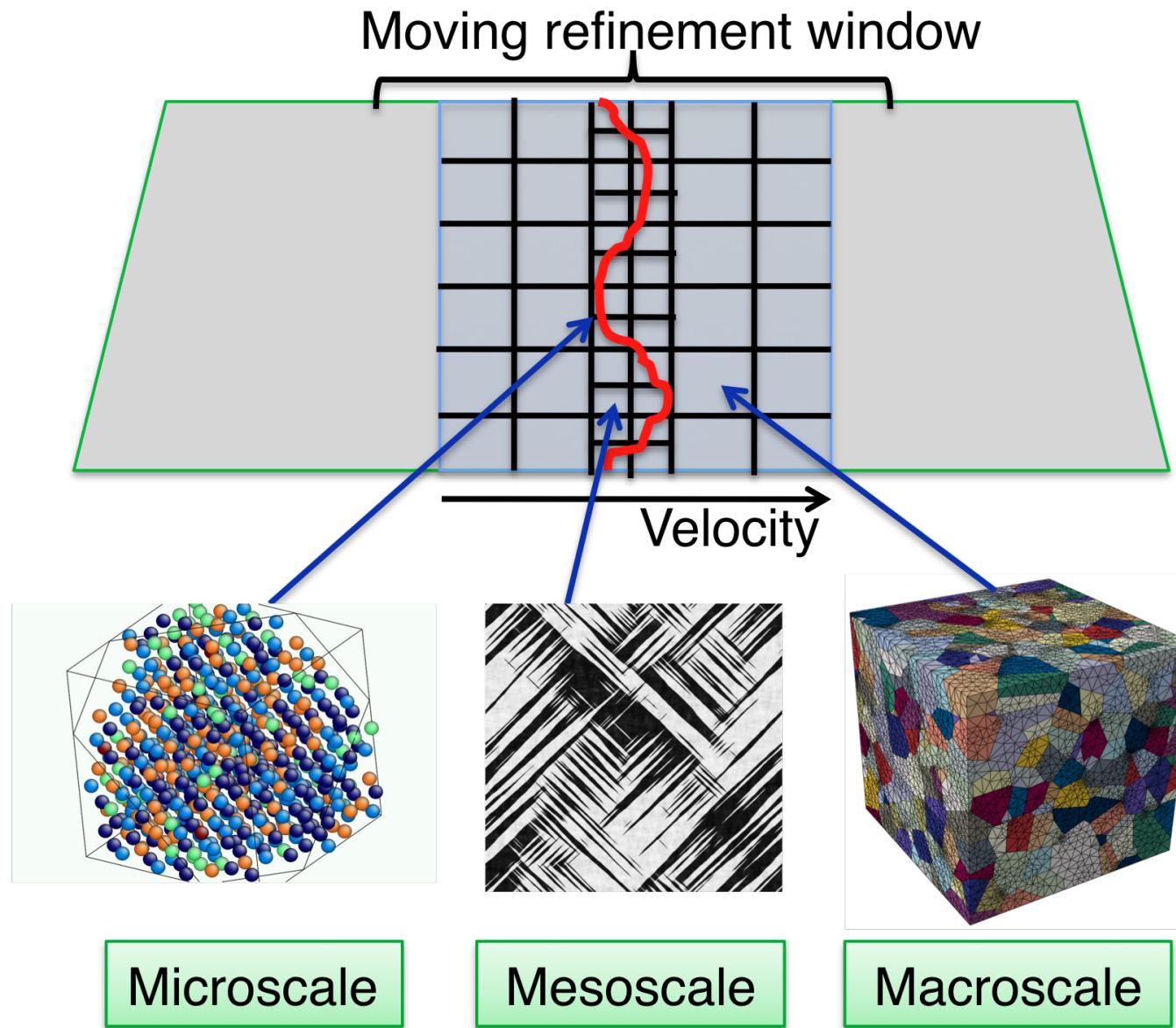
Example – Simulation of the Sumatra Earthquake

HPC-Related Data:

- 221 Mio grid cells, more than 10^{11} unknowns
- more than 3 Mio time steps on the smallest grid cells
(11 clusters with $\times 2$ -increasing timestep size)
- production run: 13.9h on SuperMUC phase 2(86,016 cores)
- 13 TB checkpoint data, 2.8 TB for post-processing
(asynchronous IO; costs entirely overlapped by computation)

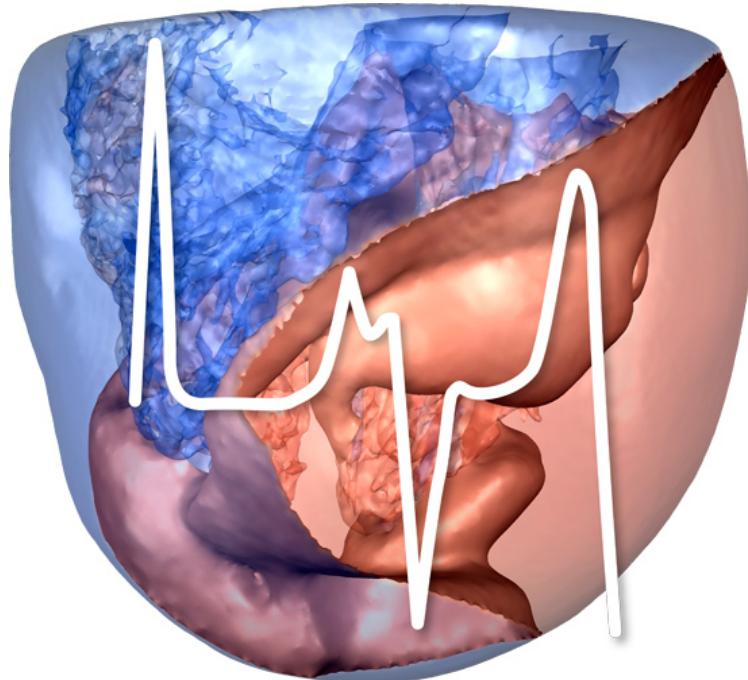


Scale Bridging Example: Material Science



Cardiac Simulation (BG/Q at LLNL)

<https://education.llnl.gov/programs/science-on-saturday/lecture/541> and <https://str.llnl.gov/Sep12/streitz.html>



Electrophysiology of the human heart

- Close to real time heart simulation
- Near cellular resolution

Parallel programming aspects

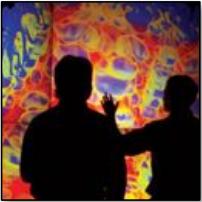
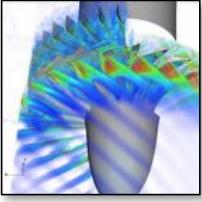
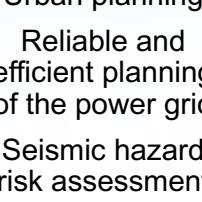
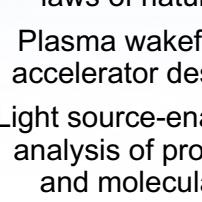
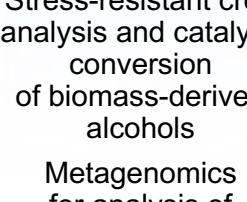
- Strong scaling problem
- Simulation on 1.600.000 cores
- "Bare metal" programming
- Achieved close to 12 Pflop/s



US Exascale / ECP Application Targets



Capable exascale system applications will deliver broad coverage of 6 strategic pillars

National security	Energy security	Economic security	Scientific discovery	Earth system	Health care
Stockpile stewardship  	Turbine wind plant efficiency Design and commercialization of SMRs Nuclear fission and fusion reactor materials design Subsurface use for carbon capture, petro extraction, waste disposal High-efficiency, low-emission combustion engine and gas turbine design Carbon capture and sequestration scaleup Biofuel catalyst design 	Additive manufacturing of qualifiable metal parts Urban planning Reliable and efficient planning of the power grid Seismic hazard risk assessment 	Cosmological probe of the standard model of particle physics Validate fundamental laws of nature Plasma wakefield accelerator design Light source-enabled analysis of protein and molecular structure and design Find, predict, and control materials and properties Predict and control stable ITER operational performance Demystify origin of chemical elements 	Accurate regional impact assessments in Earth system models Stress-resistant crop analysis and catalytic conversion of biomass-derived alcohols Metagenomics for analysis of biogeochemical cycles, climate change, environmental remediation 	Accelerate and translate cancer research 



Source: P. Messina, ECP

Seminar Announcement

David Montoya, Los Alamos National Laboratory



JUNE 1, 2018 | 14:30-15:30 | LRZ, SEMINARRAUM 2

„UNITED STATES EXASCALE
COMPUTING PROJECT (ECP)
OVERVIEW AND STATUS WITH A
FOCUS ON FACILITY INTEGRATION“

THIS TALK WILL BE PRESENTED IN ENGLISH

Joint special lecture hosted by LRZ and TUM Chair I10
(Computer Architecture and Parallel Systems)



HOW?

How to Program Distributed Memory Machines?

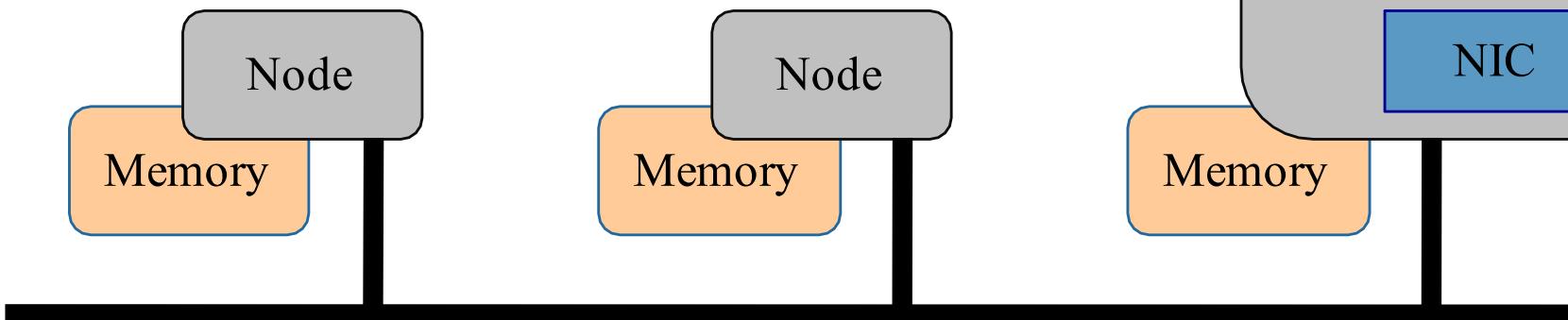
Independent nodes

- In most cases independent OS and software stacks
- Independent processes
- Connected by a network (in most cases high-speed)

Communication has to be explicit

- At least at some level of abstraction
 - Could be hidden in higher-levels of abstraction
 - Basic abstractions: send and receive

Data has to be distributed manually



The Message Passing Interface (MPI)

Designed in 1992, based on previous experiences with message passing libraries

- MPI 1.0 first ratified in 1994
- Most current version 3.1 (for the most backward compatible)
- All documents at: <http://www mpi-forum.org/>



Basic functionality

- Send and receive operations for point to point communication
- One sided communication operations
- Collective operations among groups of processes
- Process group management
- Supports C and Fortran (up to Fortran 2008)

Principles

- MPI is an API (Application Programming Interface) specification
 - No ABI (Application Binary Interface) guarantees
 - Changing MPI implementation means recompiling
- MPI is implemented as a library (most common: Open MPI, MPICH, MVAPICH)
- MPI does NOT define runtime and environment interactions
- MPI does NOT define a protocol (i.e., guarantee interoperability between MPIs)

MPI Principles and/vs. Practices

Independent processes with their own code

- Compiled against the same MPI implementation
- Different data type representations on each node
- Started independently using a non-specified mechanism

Common practice

- One source code compiler to one binary
 - Within code distinguishing of MPI processes or data elements
 - Common usage model: Single Program Multiple Data (SPMD)
- Compiled against same MPI implementation
- One datatype representation in entire system
- MPI implementation provide startup mechanism
 - Often integrated into resource manager and job scheduler
 - Platform specific

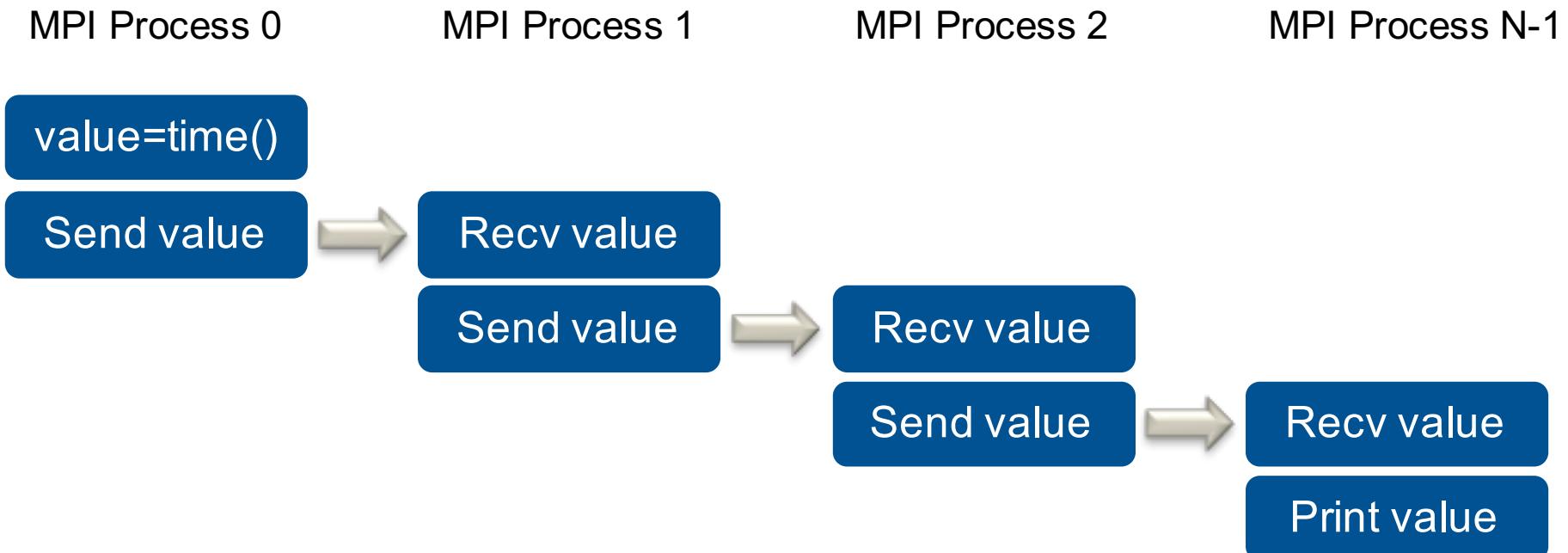
Note, though: MPI is **NOT** an SPMD model

- Often used as such
- MPI is more flexible
- Originally intended as runtime portability layer

A First Simple Example: Value Forwarding

Goal

- Start N processes
- Get time stamp on process 0
- Hand value through process 1, 2, 3, ... to process N-1
- Print value N



A First Simple Example: Value Forwarding



```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

A First Simple Example: Value Forwarding Standard Sequential Startup

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

A First Simple Example: Value Forwarding MPI as a Library Has a Header File

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value= MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

Note:

Prefix MPI_ is
reserved for the
MPI Standard

A First Simple Example: Value Forwarding Initializing and Finalizing the Use of MPI

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
MPI_Finalize ();
}
```

MPI_Init and MPI_Finalize

```
int MPI_Init( int *argc, char ***argv )  
    argc/argv pair, typically passed from main
```

Initializes the MPI library

- First MPI call that has to be called (with a few exceptions)
- Can only be called exactly once by each MPI process
- Needs to be called by all processes MPI processes launched at startup
 - Exact behavior depends on launcher
 - In some environments, only one process is launched that then spawns the rest during MPI_Init
- Strips MPI arguments from argc and argv

```
int MPI_Finalize( void )
```

Finishes the use of MPI and releases all resources

- Before finalize is called, all communication must be completed
- Has to be called by all MPI processes and is collective

Error Handling in MPI

(Almost) every MPI routine returns an error code

- In C: return value
- In Fortran: last argument "ierr (OUT)"
- Can also install error handlers

Error codes are implementation specific

- But can be mapped to error classes
- Error classes are standardized (Example: `MPI_ERR_ARG`)
- Success is indicated by returning `MPI_SUCCESS`

But (from MPI Standard 3.1, Section 2.8, page 21):

By default, an error detected during the execution of the MPI library causes the parallel computation to abort, except for file operations. However, MPI provides mechanisms for users to change this default and to handle recoverable errors.

The latter is not well supported/defined/used

- In most cases, MPI simply aborts
- Newer versions of the standard will improve this situation

A First Simple Example: Value Forwarding **MPI_COMM_WORLD**



```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

Communicators and MPI_COMM_WORLD

Central concept in MPI: Communicators

- Group of MPI processes
- Communication context

Any communication operations is done in the context of a communicator

- Argument to many functions
- Addressing is relative to a communicator
 - Each process has a rank in a communicator
 - Ranks go from 0 to N-1 are fixed
 - A process can have different ranks in different communicators
- Communication across communicators are not possible
 - Different context
 - But: one MPI process can be in multiple communicators

Default communicators:

- **MPI_COMM_WORLD**: all initial MPI processes
- **MPI_COMM_SELF**: contains only the own MPI process

Other communicators can be derived

MPI Process vs. (OS) Process vs. Rank

MPI Processes are basic units of concurrency in MPI

- Communication endpoint that can send and receive messages
- MPI processes are members of MPI Groups and MPI Communicators

MPI Processes are not the same as (OS) processes

- OS process is an abstraction / isolation concept in the OS
- No connection to the definition of the MPI library

However, in most cases MPI processes are implemented as OS processes

- Commonly familiar abstraction
- Enables isolation of global data

BUT: this is NOT guaranteed (!)

- There are implementations (e.g., CEA's MPC) where an MPI process is one thread
 - Running on one or more OS processes (one per node)
 - Global data is then shared among co-located MPI processes

Rank is an identifier of an MPI process relative to an MPI communicator

A First Simple Example: Value Forwarding Finding a Place in the World

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

MPI_Comm_get_size/rank

Each MPI process needs to understand its role/location in the MPI code

```
int MPI_Comm_size (MPI_Comm comm, int *size)
    IN   comm   Communicator
    OUT  size   Cardinality of the process group for comm
```

Returns the number of processes in the process group of the given communicator.

```
int MPI_Comm_rank (MPI_Comm comm, int *rank)
    IN   comm   Communicator
    OUT  rank   Rank of the current process in comm
```

Returns the rank of the executing process relative to the communicator.

Note: every code should check the size of MPI_COMM_WORLD

- Parameter to mpirun is just a “guidance”

A First Simple Example: Value Forwarding Boilerplate



```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

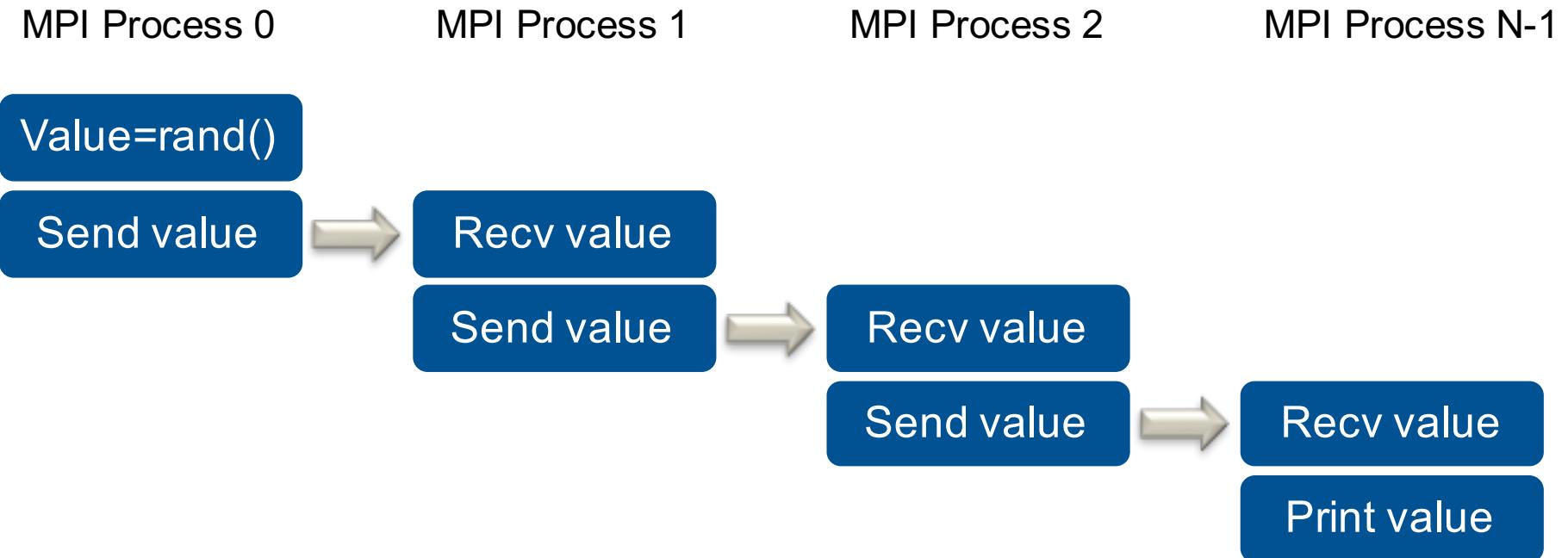
int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

A First Simple Example: Value Forwarding Implementing Communication

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

A First Simple Example: Value Forwarding Communication Pattern



```
if (rank>0)
    MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
if (rank<size-1)
    MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
if (rank==size-1)
    printf("Value from MPI Process 0: %f\n",value);
```

MPI_Send

Send one message to another MPI process on a given communicator

```
int MPI_Send (void *buf, int count, MPI_Datatype dtype,  
              int dest, int tag, MPI_Comm comm)
```

IN buf	Address of the send buffer
IN count	Number of data elements of type <code>dtype</code> to be sent
IN dtype	Data type
IN dest	Receiver (rank of target MPI process in <code>comm</code>)
IN tag	Message tag
IN comm	Communicator

Initiates message send and blocks until send buffer can be reused.

- Note: possibilities of deadlocks (!)

Most Common MPI Data Types (C Versions)

MPI_CHAR	char (treated as printable character)
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG	signed long long int
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_C_COMPLEX	float _Complex
MPI_C_DOUBLE_COMPLEX	double _Complex
MPI_BYTE	
MPI_PACKED	

Custom datatypes can be defined through special routines.

MPI_Recv

Receives of a message from another MPI process on a given communicator

```
int MPI_Recv (void *buf, int count, MPI_Datatype dtype,  
int source, int tag, MPI_Comm comm, MPI_Status *status)
```

OUT buf	Address of the receive buffer
IN count	Number of data elements of type <code>dtype</code> to be received
IN <code>dtype</code>	Data type
IN <code>source</code>	Sender (rank of source MPI process in <code>comm</code>)
IN tag	Message tag
IN <code>comm</code>	Communicator
OUT <code>status</code>	Status information

Blocking receive of a message

- Blocks execution until a matching message has been received
- Note: possibilities of deadlocks (!)

MPI_Recv Properties

A message to be received by this function must match

- Matching source and tag in the same communicator (context)

Sender and tag can be specified as wild cards

- **MPI_ANY_SOURCE** and **MPI_ANY_TAG**
- Can be used instead of source and tag arguments
- There is no wild card for the communicator/context

Messages with the same channel signature (source/dest, tag, comm) are ordered

- No overtaking of messages
- In future versions of the MPI standard this can be overwritten

Messages are supposed to match in their type signature

- This cannot be tested by MPI and hence user's responsibility
- Casting messages at send, recv or in transfer can be dangerous (!)

Message buffers must be at least as long as the message

- Partial receives are permitted

MPI_Status

Structure containing information on incoming message

In C, MPI_Status is a structure that includes the following fields:

- MPI_SOURCE: sender of the message
- MPI_TAG: message tag
- MPI_ERROR: error code

Important for wild card communication

- Identifies the actual channel the message was transmitted on

The actual length of the received message can be determined via

```
int MPI_Get_count(const MPI_Status *status,  
                  MPI_Datatype dtype, int *count)
```

IN buf	MPI_Status object returned from the receive call
IN dtype	Data type used in receive
OUT count	Number of data elements actually received

A First Simple Example: Value Forwarding Timing Routine



```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

MPI_Wtime and MPI_Wtick

`double MPI_Wtime(void)`

Return wall clock time since a reference time stamp in the past

- Unit: seconds
- Reference time stamp is fixed for one execution
- Time can be global, but doesn't have to be
 - In fact, in most cases it is not
 - Users have to synchronize, if needed
- Note: return value is not an error code

`double MPI_Wtick(void)`

Returns resolution of `MPI_Wtime`

- Number of seconds between two successive clock ticks
- Note: return value is not an error code

A First Simple Example: Value Forwarding



```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main (int argc, char** argv)
{
    double value;
    int size, rank;
    MPI_Status s;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    value=MPI_Wtime();
    printf("MPI Process %d of %d (value=%f)\n", rank, size, value);
    if (rank>0)
        MPI_Recv(&value, 1, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, &s);
    if (rank<size-1)
        MPI_Send(&value, 1, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD);
    if (rank==size-1)
        printf("Value from MPI Process 0: %f\n",value);
    MPI_Finalize ();
}
```

A First Simple Example: Value Forwarding Compiling and Running



```
.../ParProg> mpicc-openmpi-mp forw.c -o forw
```

- Most MPI implementation provide wrapper that set includes and libraries

```
.../ParProg> mpirun-openmpi-mp -np 4 --oversubscribe forw
```

- Most MPI implementations provide a run command
 - Often following suggestions in the standard
 - Common parameter –np 4 (or –n 4)
 - Oversubscription in this case needed for run on one laptop

```
MPI Process 1 of 4 (value=1527450230.258663)
MPI Process 2 of 4 (value=1527450230.258661)
MPI Process 0 of 4 (value=1527450230.258765)
MPI Process 3 of 4 (value=1527450230.258802)
Value from MPI Process 0: 1527450230.258765
```

Preview MPI (for the next two lectures)

Point-to-Point operations

- Different variants of MPI_Send
- Non-blocking versions

Datatypes

One-sided communication

Collective operations

- Group operations across all processes of a communicator
- Examples: barriers, reductions, scatter/gather
- Neighborhood and non-blocking collectives

Group and communicator management

- Creates of groups and communicators
- Mapping of communicators to topologies

File I/O

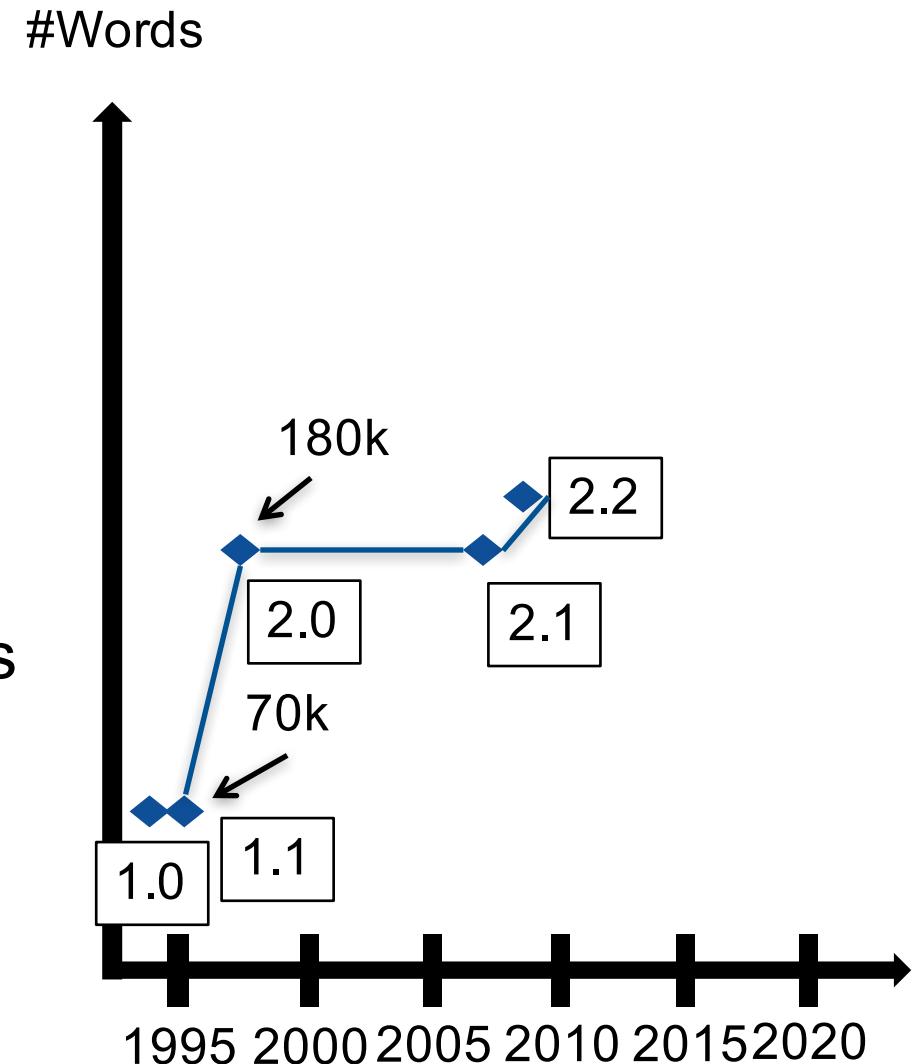
Persistent operations

History of the Message Passing Interface

- MPI 1.0 May 1994 – 228 pages
- MPI 1.1 Nov 1995 – 238 pages (128 functions)
- MPI 2.0 Nov 1997 – 608 pages

10 year break

- MPI 2.1 June 2008 – 608 pages
 - Merged document
- MPI 2.2 Sep 2009 – 647 pages
 - Small, conforming additions
- In parallel, start on MPI 3.0
 - Major new concepts
 - Completed in 2012



Notable Additions to MPI 3.0

Nonblocking collectives

Neighborhood collectives

MPI Tool Information Interface

One sided communication enhancements

Large data counts (messages more than 32bit count)

Topology aware communicator creation

Noncollective communicator creation

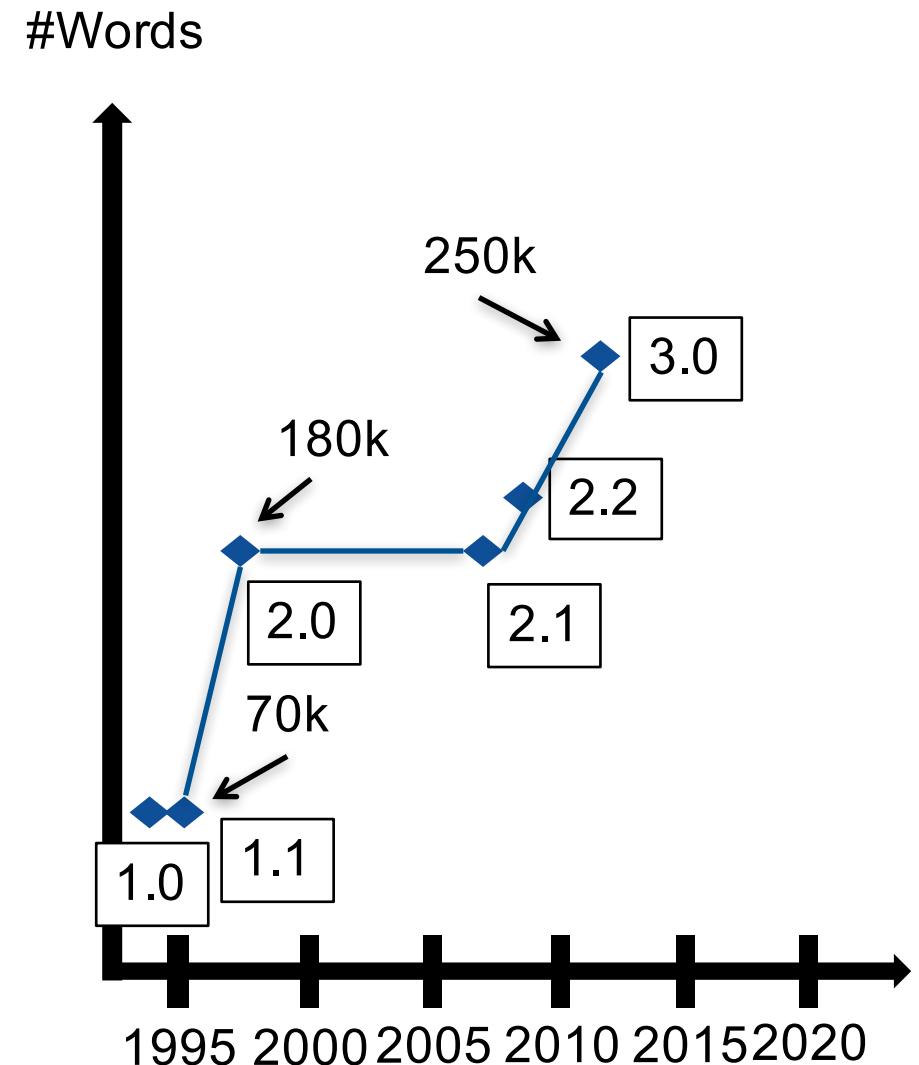
New language bindings

MPI Keeps Growing

- MPI 1.0 May 1994 – 228 pages
- MPI 1.1 Nov 1995 – 238 pages (128 fns)
- MPI 2.0 Nov 1997 – 608 pages

10 year break

- MPI 2.1 June 2008 – 608 pages
- MPI 2.2 Sep 2009 – 647 pages
- MPI 3.0 Sep 2012 – 852 pages (430 fns)



MPI 3.0 has 430 Functions



MPI_ABORT	MPI_ERRHANDLER_GET	MPI_GROUP_DIFFERENCE	MPI_TYPE_DELETE_ATTR
MPI_ACCUMULATE	MPI_ERRHANDLER_SET	MPI_GROUP_EXCL	MPI_TYPE_DUP
MPI_ADD_ERROR_CLASS	MPI_ERROR_CLASS	MPI_GROUP_F2C	MPI_TYPE_DUP_FN
MPI_ADD_ERROR_CODE	MPI_ERROR_STRING	MPI_GROUP_FREE	MPI_TYPE_EXTENT
MPI_ADD_ERROR_STRING	MPI_EXSCAN	MPI_GROUP_INCL	MPI_TYPE_F2C
MPI_ADDRESS	MPI_F_SYNC_REG	MPI_GROUP_INTERSECTION	MPI_TYPE_FREE
MPI_ALLGATHER	MPI_FETCH_AND_OP	MPI_GROUP_RANGE_EXCL	MPI_TYPE_FREE_KEYVAL
MPI_ALLGATHERV	MPI_FILE_C2F	MPI_GROUP_RANGE_INCL	MPI_TYPE_GET_ATTR
MPI_ALLOC_MEM	MPI_FILE_CALL_ERRHANDLER	MPI_GROUP_RANK	MPI_TYPE_GET_CONTENTS
MPI_ALLOC_MEM_CPTR	MPI_FILE_CLOSE	MPI_GROUP_SIZE	MPI_TYPE_GET_ENVELOPE
MPI_ALLREDUCE	MPI_FILE_CREATE_ERRHANDLER	MPI_GROUP_TRANSLATE_RANKS	MPI_TYPE_GET_EXTENT
MPI_ALLTOALL	MPI_FILE_DELETE	MPI_GROUP_UNION	MPI_TYPE_GET_EXTENT_X
MPI_ALLTOALLV	MPI_FILE_F2C	MPI_ALLGATHER	MPI_TYPE_GET_NAME
MPI_ALLTOALLW	MPI_FILE_GET_AMODE	MPI_ALLGATHERV	MPI_TYPE_GET_TRUE_EXTENT
MPI_ATTR_DELETE	MPI_FILE_GET_ATOMICTY	MPI_ALLREDUCE	MPI_TYPE_GET_TRUE_EXTENT_X
MPI_ATTR_GET	MPI_FILE_GET_BYTE_OFFSET	MPI_ALLTOALL	MPI_TYPE_HINDEXED
MPI_AVOID_PUT	MPI_FILE_GET_ERRHANDLER	MPI_ALLTOALLV	MPI_TYPE_HVECTOR
MPI_BARRIER	MPI_FILE_GET_GROUP	MPI_BARRIER	MPI_TYPE_INDEXED
MPI_CART_C2F	MPI_FILE_GET_INFO	MPI_BCAST	MPI_TYPE_I8
MPI_BSEND	MPI_FILE_GET_POSITION	MPI_BSEND	MPI_TYPE_MATCH_SIZE
MPI_BSEND_INIT	MPI_FILE_GET_POSITION_SHARED	MPI_EXSCAN	MPI_TYPE_NULL_COPY_FN
MPI_BUFFER_ATTACH	MPI_FILE_GET_SIZE	MPI_GATHER	MPI_TYPE_NULL_DELETE_FN
MPI_BUFFER_DETACH	MPI_FILE_GET_TYPE_EXTENT	MPI_GATHERV	MPI_TYPE_SET_ATTR
MPI_CANCEL	MPI_FILE_GET_VIEW	MPI_IMPROBE	MPI_TYPE_SET_NAME
MPI_CART_COORDS	MPI_FILE_IREAD	MPI_IMRECV	MPI_TYPE_SIZE
MPI_CART_CREATE	MPI_FILE_IREAD_AT	MPI_NEIGHBOR_ALLGATHER	MPI_TYPE_SIZE_X
MPI_CART_GET	MPI_FILE_IREAD_SHARED	MPI_NEIGHBOR_ALLGATHERV	MPI_TYPE_STRUCT
MPI_CART_MAP	MPI_FILE_IWRITE	MPI_NEIGHBOR_ALLTOALL	MPI_TYPE_UB
MPI_CART_RANK	MPI_FILE_IWRITE_AT	MPI_NEIGHBOR_ALLTOALLV	MPI_TYPE_VECTOR
MPI_CART_SHIFT	MPI_FILE_IWRITE_SHARED	MPI_NEIGHBOR_ALLTOALLV	MPI_UNPACK
MPI_CART_SUB	MPI_FILE_OPEN	MPI_INFO_C2F	MPI_UNPACK_EXTERNAL
MPI_CARTDIM_GET	MPI_FILE_PALLOCATE	MPI_INFO_C2F08	MPI_UNPUBLISH_NAME
MPI_CLOSE_PORT	MPI_FILE_READ	MPI_INFO_F082C	MPI_WAIT
MPI_COMM_ACCEPT	MPI_FILE_READ_ALL	MPI_INFO_F082F	MPI_WAITALL
MPI_COMM_C2F	MPI_FILE_READ_ALL_BEGIN	MPI_INFO_F2C	MPI_WAITANY
MPI_COMM_CALL_ERRHANDLER	MPI_FILE_READ_ALL_END	MPI_INFO_F2F08	MPI_WAITSOME
MPI_COMM_COMPARE	MPI_FILE_READ_AT	MPI_STATUS_SET_CANCELLED	MPI_WIN_ALLOC
MPI_COMM_CONNECT	MPI_FILE_READ_AT_ALL	MPI_STATUS_SET_ELEMENTS	MPI_WIN_ALLOCATE
MPI_COMM_CREATE	MPI_FILE_READ_AT_ALL_BEGIN	MPI_STATUS_SET_ELEMENTS_X	MPI_WIN_ALLOCATE_CPTR
MPI_COMM_CREATE_ERRHANDLER	MPI_FILE_READ_AT_ALL_END	MPI_T_CATEGORY_CHANGED	MPI_WIN_ALLOCATE_SHARED
MPI_COMM_CREATE_GROUP	MPI_FILE_READ_ORDERED	MPI_T_CATEGORY_GET_CATEGORIES	MPI_WIN_ATTACH
MPI_COMM_CREATE_KEYVAL	MPI_FILE_READ_ORDERED_BEGIN	MPI_T_CATEGORY_GET_CVARS	MPI_WIN_C2F
MPI_COMM_DELETE_ATTR	MPI_FILE_READ_ORDERED_END	MPI_T_CATEGORY_GET_INFO	MPI_WIN_CALL_ERRHANDLER
MPI_COMM_DISCONNECT	MPI_FILE_READ_SHARED	MPI_T_CATEGORY_GET_NUM	MPI_WIN_COMPLETE
MPI_COMM_DUP	MPI_FILE_SEEK	MPI_T_INTERCOMM_CREATE	MPI_WIN_CREATE
MPI_COMM_DUP_FN	MPI_FILE_SEER_SHAR_ED	MPI_T_INTERCOMM_MERGE	MPI_WIN_CREATE_DYNAMIC
MPI_COMM_DUP_WITH_INFO	MPI_FILE_SET_ATOMICTY	MPI_PROBE	MPI_WIN_CREATE_ERRHANDLER
MPI_COMM_F2C	MPI_FILE_SET_ERRHANDLER	MPI_RECV	MPI_WIN_CREATE_KEYVAL
MPI_COMM_FREE	MPI_FILE_SET_INFO	MPI_REDUCE	MPI_WIN_DELETE_ATTR
MPI_COMM_FREE_KEYVAL	MPI_FILE_SET_SIZE	MPI_REDUCE_SCATTER	MPI_WIN_DELEACH
MPI_COMM_GET_ALL	MPI_FILE_SET_NEW	MPI_REDUCE_SCATTER_BLOCK	MPI_WIN_DUP_FN
MPI_COMM_GET_ERRHANDLER	MPI_FILE_SYNC	MPI_RSND	MPI_WIN_F2C
MPI_COMM_GET_INFO	MPI_FILE_WRITE	MPI_IS_THREAD_MAIN	MPI_WIN_FENCE
MPI_COMM_GET_NAME	MPI_FILE_WRITE_ALL	MPI_SCAN	MPI_WIN_FLUSH
MPI_COMM_GET_PARENT	MPI_FILE_WRITE_ALL_BEGIN	MPI_SCATTER	MPI_WIN_FLUSH_ALL
MPI_COMM_GROUP	MPI_FILE_WRITE_ALL_END	MPI_SCATTERV	MPI_WIN_FLUSH_LOCAL
MPI_COMM_IDUP	MPI_FILE_WRITE_AT	MPI_SEND	MPI_WIN_FLUSH_LOCAL_ALL
MPI_COMM_JOIN	MPI_FILE_WRITE_AT_ALL	MPI_ISSEND	MPI_WIN_FREE
MPI_COMM_KEYVAL_CREATE	MPI_FILE_WRITE_AT_ALL_BEGIN	MPI_KEYVAL_CREATE	MPI_WIN_FREE_KEYVAL
MPI_COMM_NULL_COPY_FN	MPI_FILE_WRITE_AT_ALL_END	MPI_KEYVAL_FREE	MPI_WIN_GET_ATTR
MPI_COMM_NULL_DELETE_FN	MPI_FILE_WRITE_ORDERED	MPI_LOCK_ALL	MPI_WIN_GET_ERRHANDLER
MPI_COMM_RANK	MPI_FILE_WRITE_ORDERED_BEGIN	MPI_LOOKUP_NAME	MPI_WIN_GET_GROUP
MPI_COMM_REMOTE_GROUP	MPI_FILE_WRITE_ORDERED_END	MPI_MESSAGE_C2F	MPI_WIN_GET_INFO
MPI_COMM_REMOTE_SIZE	MPI_FILE_WRITE_SHARED	MPI_MESSAGE_F2C	MPI_WIN_GET_NAME
MPI_COMM_SET_ATTR	MPI_FINALIZE	MPI_probe	MPI_WIN_LOCK
MPI_COMM_SET_ERRHANDLER	MPI_FINALIZED	MPI_RECV	MPI_WIN_LOCK_ALL
MPI_COMM_SET_INFO	MPI_FREE_MEM	MPI_TEST	MPI_WIN_NULL_COPY_FN
MPI_COMM_SET_NAME	MPI_GATHER	MPI_TEST_CANCELLED	MPI_WIN_NULL_DELETE_FN
MPI_COMM_SIZE	MPI_GATHERV	MPI_TESTALL	MPI_WIN_POST
MPI_COMM_SPAWN	MPI_GET	MPI_TESTANY	MPI_WIN_SET_ATTR
MPI_COMM_SPAWN_MULTIPLE	MPI_GET_ACCUMULATE	MPI_TESTSOME	MPI_WIN_SET_ERRHANDLER
MPI_COMM_SPLIT	MPI_GET_ADDRESS	MPI_TOPO_TEST	MPI_WIN_SET_INFO
MPI_COMM_SPLIT_TYPE	MPI_GET_COUNT	MPI_TYPE_C2F	MPI_WIN_SET_NAME
MPI_COMM_TEST_INTER	MPI_GET_ELEMENTS	MPI_TYPE_COMMIT	MPI_WIN_SHARED_ALLOCATE
MPI_COMM_WORLD	MPI_GET_ELEMENTS_X	MPI_TYPE_CONTIGUOUS	MPI_WIN_SHARED_QUERY
MPI_COMPARE_AND_SWAP	MPI_GET_LIBRARY_VERSION	MPI_TYPE_CREATE_DARRAY	MPI_WIN_SHARED_QUERY_CPTR
MPI_CONVERSION_FN_NULL	MPI_GET_PROCESSOR_NAME	MPI_TYPE_CREATE_F90_COMPLEX	MPI_WIN_START
MPI_DIMS_CREATE	MPI_GET_VERSION	MPI_TYPE_CREATE_F90_INTEGER	MPI_WIN_SYNC
MPI_DIST_GRAPH_CREATE	MPI_GRAPH_CREATE	MPI_TYPE_CREATE_F90_REAL	MPI_WIN_TEST
MPI_DIST_GRAPH_CREATE_ADJACENT	MPI_GRAPH_GET	MPI_TYPE_CREATE_HINDEXED	MPI_WIN_UNLOCK
MPI_DIST_GRAPH_NEIGHBOR_COUNT	MPI_GRAPH_MAP	MPI_TYPE_CREATE_HINDEXED_BLOCK_K	MPI_WIN_UNLOCK_ALL
MPI_DIST_GRAPH_NEIGHBORS	MPI_GRAPH_NEIGHBORS	MPI_PACK_EXTERNAL	MPI_WIN_WAIT
MPI_DIST_GRAPH_NEIGHBORS_COUNT	MPI_GRAPH_NEIGHBORS_COUNT	MPI_PACK_SIZE	MPI_WTICK
MPI_DUP_FN	MPI_GRAPHDIMS_GET	MPI_PCONTROL	MPI_WTIME
MPI_ERRHANDLER_C2F	MPI_GREQUEST_COMPLETE	MPI_PROBE	
MPI_ERRHANDLER_CREATE	MPI_GREQUEST_START	MPI_PUBLISH_NAME	
MPI_ERRHANDLER_F2C	MPI_GROUP_C2F	MPI_PUT	
MPI_ERRHANDLER_FREE	MPI_GROUP_COMPARE		



MPI 3.0 has 430 Functions



MPI_ABORT	MPI_ERRHANDLER_GET	MPI_GROUP_DIFFERENCE	MPI_QUERY_THREAD	MPI_TYPE_DELETE_ATTR
MPI_ACCUMULATE	MPI_ERRHANDLER_SET	MPI_GROUP_EXCL	MPI_RACCUMULATE	MPI_TYPE_DUP
MPI_ADD_ERROR_CLASS	MPI_ERROR_CLASS	MPI_GROUP_F2C	MPI_RECV	MPI_TYPE_DUP_FN
MPI_ADD_ERROR_CODE	MPI_ERROR_STRING	MPI_GROUP_FREE	MPI_RECV_INIT	MPI_TYPE_EXTENT
MPI_ADD_ERROR_STRING	MPI_EXSCAN	MPI_GROUP_INCL	MPI_REDUCE	MPI_TYPE_F2C
MPI_ADDRESS	MPI_F_SYNC_REG	MPI_GROUP_INTERSECTION	MPI_REDUCE_LOCAL	MPI_TYPE_FREE
MPI_ALLGATHER	MPI_FETCH_AND_OP	MPI_GROUP_RANGE_EXCL	MPI_REDUCE_SCATTER	MPI_TYPE_FREE_KEYVAL
MPI_ALLGATHERV	MPI_FILE_C2F	MPI_GROUP_RANGE_INCL	MPI_REDUCE_SCATTER_BLOCK	MPI_TYPE_GET_ATTR
MPI_ALLOC_MEM	MPI_FILE_CALL_ERRHANDLER	MPI_GROUP_RANK	MPI_REGISTER_DATAREP	MPI_TYPE_GET_CONTENTS
MPI_ALLOC_MEM_CPTR	MPI_FILE_CLOSE	MPI_GROUP_SIZE	MPI_REQUEST_C2F	MPI_TYPE_GET_EXTENT
MPI_ALLREDUCE	MPI_FILE_CREATE_ERRHANDLER	MPI_GROUP_UNION	MPI_REQUEST_F2C	MPI_TYPE_GET_EXTENT_X
MPI_ALLTOALL	MPI_FILE_DELETE	MPI_ALLGATHER	MPI_REQUEST_FREE	MPI_TYPE_GET_NAME
MPI_ALLTOALLV	MPI_FILE_F2C	MPI_ALLGATHERV	MPI_REQUEST_GET_STATUS	MPI_TYPE_GET_TRUE_EXTENT
MPI_ALLTOALLW	MPI_FILE_GET_AMODE	MPI_ALLREDUCE	MPI_RGET	MPI_TYPE_GET_TRUE_EXTENT_X
MPI_ATTR_DELETE	MPI_FILE_GET_ATOMICTY	MPI_ALLTOALL	MPI_RGET_ACCUMULATE	MPI_TYPE_HINDEXED
MPI_ATTR_GET	MPI_FILE_GET_BYTE_OFFSET	MPI_ALLTOALLV	MPI_RPUT	MPI_TYPE_HVECTOR
MPI_AVOID_PUT	MPI_FILE_GET_ERRHANDLER	MPI_BARRIER	MPI_RSEND	MPI_TYPE_INDEXED
MPI_BARRIER	MPI_FILE_GET_GROUP	MPI_BCAST	MPI_RSEND_INIT	MPI_TYPE_I_B
MPI_CART_C2F	MPI_FILE_GET_INFO	MPI_BSEND	MPI_SCATTER	MPI_TYPE_MATCH_SIZE
MPI_BSEND	MPI_FILE_GET_POSITION	MPI_EXSCAN	MPI_SCATTERV	MPI_TYPE_NULL_COPY_FN
MPI_BSEND_INIT	MPI_FILE_GET_POSITION_SHARED	MPI_GATHER	MPI_SEND	MPI_TYPE_NULL_DELETE_FN
MPI_BUFFER_ATTACH	MPI_FILE_GET_SIZE	MPI_GATHERV	MPI_SEND_INIT	MPI_TYPE_SET_ATTR
MPI_BUFFER_DETACH	MPI_FILE_GET_TYPE_EXTENT	MPI_IMPROBE	MPI_SENDRECV	MPI_TYPE_SET_NAME
MPI_CANCEL	MPI_FILE_GET_VIEW	MPI_IRECVR	MPI_SENDRECV_REPLACE	MPI_TYPE_SIZE
MPI_CART_COORDS	MPI_FILE_IREAD	MPI_NEIGHBOR_ALLGATHER	MPI_SIZEOF	MPI_TYPE_SIZE_X
MPI_CART_CREATE	MPI_FILE_IREAD_AT	MPI_NEIGHBOR_ALLGATHERV	MPI_SSSEND	MPI_TYPE_STRUCT
MPI_CART_GET	MPI_FILE_IREAD_SHARED	MPI_NEIGHBOR_ALLTOALL	MPI_SSSEND_INIT	MPI_TYPE_UB
MPI_CART_MAP	MPI_FILE_IWRITE	MPI_NEIGHBOR_ALLTOALLV	MPI_START	MPI_TYPE_VECTOR
MPI_CART_RANK	MPI_FILE_IWRITE_AT	MPI_NEIGHBOR_ALLTOALLW	MPI_STARTALL	MPI_UNPACK
MPI_CART_SHIFT	MPI_FILE_IWRITE_SHARED	MPI_OPEN	MPI_STATUS_C2F	MPI_UNPACK_EXTERNAL
MPI_CART_SUB	MPI_FILE_OPEN	MPI_PINFO_C2F8	MPI_UNPUBLISH_NAME	MPI_WAIT
MPI_CARTDIM_GET	MPI_FILE_PALLOCATE	MPI_STATUS_F082C	MPI_WAITALL	MPI_WAITALL
MPI CLOSE_PORT	MPI_FILE_READ	MPI_STATUS_F082F	MPI_WAITANY	MPI_WAITANY
MPI_COMM_ACCEPT	MPI_FILE_READ_ALL	MPI_STATUS_F2C	MPI_WAITSOME	MPI_WAITSOME
MPI_COMM_C2F	MPI_FILE_READ_ALL_BEGIN	MPI_STATUS_F2F08	MPI_WIN_ALLOC	MPI_WIN_ALLOCATE
MPI_COMM_CALL_ERRHANDLER	MPI_FILE_READ_ALL_END	MPI_STATUS_SET_CANCELLED	MPI_WIN_ALLOCATE_CPTR	MPI_WIN_ALLOCATE_SHARED
MPI_COMM_COMPARE	MPI_FILE_READ_AT	MPI_STATUS_SET_ELEMENTS	MPI_WIN_ALLOCATE_SHARED_CPTR	MPI_WIN_ATTACH
MPI_COMM_CONNECT	MPI_FILE_READ_AT_ALL	MPI_STATUS_SET_ELEMENTS_X	MPI_WIN_CREATE	MPI_WIN_C2F
MPI_COMM_CREATE	MPI_FILE_READ_AT_ALL_BEGIN	MPI_T_CATEGORY_CHANGED	MPI_WIN_CREATE_DYNAMIC	MPI_WIN_CALL_ERRHANDLER
MPI_COMM_CREATE_ERRHANDLER	MPI_FILE_READ_AT_ALL_END	MPI_T_CATEGORY_GET_CATEGORIES	MPI_WIN_CREATE_ERRHANDLER	MPI_WIN_COMPLETE
MPI_COMM_CREATE_GROUP	MPI_FILE_READ_ORDERED	MPI_T_CATEGORY_GET_CVARS	MPI_WIN_CREATE_KEYVAL	MPI_WIN_CREATE_KEYVAL
MPI_COMM_CREATE_KEYVAL	MPI_FILE_READ_ORDERED_BEGIN	MPI_T_CATEGORY_GET_INFO	MPI_WIN_DELETE_ATTR	MPI_WIN_DELETE_ATTR
MPI_COMM_DELETE_ATTR	MPI_FILE_READ_ORDERED_END	MPI_T_CATEGORY_GET_NUM	MPI_WIN_DELETAH	MPI_WIN_DELETAH
MPI_COMM_DISCONNECT	MPI_FILE_READ_SHARED	MPI_T_CATEGORY_GET_PVARS	MPI_WIN_F2C	MPI_WIN_F2C
MPI_COMM_DUP	MPI_FILE_SEEK	MPI_T_CVAR_GET_INFO	MPI_WIN_FENCE	MPI_WIN_FENCE
MPI_COMM_DUP_FN	MPI_FILE_SEEK_SHARED	MPI_T_CVAR_HANDLE_ALLOC	MPI_WIN_FLUSH	MPI_WIN_FLUSH
MPI_COMM_DUP_WITH_INFO	MPI_FILE_SET_ATOMICITY	MPI_T_CVAR_HANDLE_FREE	MPI_WIN_FLUSH_ALL	MPI_WIN_FLUSH_ALL
MPI_COMM_F2C	MPI_FILE_SET_ERRHANDLER	MPI_T_CVAR_READ	MPI_WIN_FLUSH_LOCAL	MPI_WIN_FLUSH_LOCAL
MPI_COMM_FREE	MPI_FILE_SET_INFO	MPI_T_CVAR_WRITE	MPI_WIN_FLUSH_LOCAL_ALL	MPI_WIN_FLUSH_LOCAL_ALL
MPI_COMM_FREE_KEYVAL	MPI_FILE_SET_NEW	MPI_T_ENUM_GET_INFO	MPI_WIN_FREED	MPI_WIN_FREED
MPI_COMM_GET_ATTR	MPI_FILE_SET_NEW	MPI_T_ENUM_GET_ITEM	MPI_WIN_FREE_KEYVAL	MPI_WIN_FREE_KEYVAL
MPI_COMM_GET_INFO	MPI_FILE_SYNC	MPI_T_FINALIZE	MPI_WIN_GET_ATTR	MPI_WIN_GET_ATTR
MPI_COMM_GET_NAME	MPI_FILE_WRITE	MPI_T_INIT_THREAD	MPI_WIN_GET_ERRHANDLER	MPI_WIN_GET_ERRHANDLER
MPI_COMM_GET_PARENT	MPI_FILE_WRITE_ALL	MPI_IS_THREAD_MAIN	MPI_WIN_GET_GROUP	MPI_WIN_GET_GROUP
MPI_COMM_IDUP	MPI_FILE_WRITE_ALL_BEGIN	MPI_ISCAN	MPI_WIN_GET_INFO	MPI_WIN_GET_INFO
MPI_COMM_GROUP	MPI_FILE_WRITE_ALL_END	MPI_ISCATTER	MPI_WIN_GET_NAME	MPI_WIN_GET_NAME
MPI_COMM_JOIN	MPI_FILE_WRITE_AT	MPI_ISCATTERV	MPI_WIN_LOCK	MPI_WIN_LOCK
MPI_COMM_KEYVAL_CREATE	MPI_FILE_WRITE_AT_ALL	MPI_ISEND	MPI_WIN_LOCK_ALL	MPI_WIN_LOCK_ALL
MPI_COMM_NULL_COPY_FN	MPI_FILE_WRITE_AT_ALL_BEGIN	MPI_ISSEND	MPI_WIN_NULL_COPY_FN	MPI_WIN_NULL_COPY_FN
MPI_COMM_NULL_DELETE_FN	MPI_FILE_WRITE_AT_ALL_END	MPI_KEYVAL_CREATE	MPI_WIN_NULL_DELETE_FN	MPI_WIN_NULL_DELETE_FN
MPI_COMM_RANK	MPI_FILE_WRITE_ORDERED	MPI_KEYVAL_FREE	MPI_WIN_POST	MPI_WIN_POST
MPI_COMM_REMOTE_GROUP	MPI_FILE_WRITE_ORDERED_BEGIN	MPI_LOCK_ALL	MPI_WIN_SET_ATTR	MPI_WIN_SET_ATTR
MPI_COMM_REMOTE_SIZE	MPI_FILE_WRITE_ORDERED_END	MPI_LOOKUP_NAME	MPI_WIN_SET_ERRHANDLER	MPI_WIN_SET_ERRHANDLER
MPI_COMM_SET_ATTR	MPI_FILE_WRITE_SHARED	MPI_MESSAGE_C2F	MPI_WIN_SET_GROUP	MPI_WIN_SET_GROUP
MPI_COMM_SET_ERRHANDLER	MPI_FINALIZE	MPI_MESSAGE_F2C	MPI_WIN_SET_INFO	MPI_WIN_SET_INFO
MPI_COMM_SET_INFO	MPI_FINALIZED	MPI_MPROBE	MPI_WIN_SET_NAME	MPI_WIN_SET_NAME
MPI_COMM_SET_NAME	MPI_FREE_MEM	MPI_MRECV	MPI_WIN_SHARED_ALLOCATE	MPI_WIN_SHARED_ALLOCATE
MPI_COMM_SIZE	MPI_GATHER	MPI_NEIGHBOR_ALLGATHER	MPI_WIN_SHARED_QUERY	MPI_WIN_SHARED_QUERY
MPI_COMM_SPAWN	MPI_GATHERV	MPI_NEIGHBOR_ALLGATHERV	MPI_WIN_SHARED_QUERY_CPTR	MPI_WIN_SHARED_QUERY_CPTR
MPI_COMM_SPAWN_MULTIPLE	MPI_GET	MPI_NEIGHBOR_ALLTOALL	MPI_WIN_START	MPI_WIN_START
MPI_COMM_SPLIT	MPI_GET_ACCUMULATE	MPI_NEIGHBOR_ALLTOALLV	MPI_WIN_SYNC	MPI_WIN_SYNC
MPI_COMM_SPLIT_TYPE	MPI_GET_ADDRESS	MPI_NEIGHBOR_ALLTOALLW	MPI_WIN_TEST	MPI_WIN_TEST
MPI_COMM_TEST_INTER	MPI_GET_COUNT	MPI_NULL_COPY_FN	MPI_WIN_UNLOCK	MPI_WIN_UNLOCK
MPI_COMM_WORLD	MPI_GET_ELEMENTS	MPI_NULL_DELETE_FN	MPI_WIN_BLOCK_ALL	MPI_WIN_BLOCK_ALL
MPI_COMPARE_AND_SWAP	MPI_GET_ELEMENTS_X	MPI_OP_C2F	MPI_WIN_WAIT	MPI_WIN_WAIT
MPI_CONVERSION_FN_NULL	MPI_GET_LIBRARY_VERSION	MPI_OP_COMMUTATIVE	MPI_WTICK	MPI_WTICK
MPI_DIMS_CREATE	MPI_GET_PROCESSOR_NAME	MPI_OP_CREATE	MPI_WTIME	MPI_WTIME
MPI_DIST_GRAPH_CREATE	MPI_GET_VERSION	MPI_OP_F2C		
MPI_DIST_GRAPH_CREATE_ADJACENT	MPI_GRAPH_CREATE	MPI_OP_FREE		
MPI_DIST_GRAPH_NEIGHBOR_COUN	MPI_GRAPH_GET	MPI_OPEN_PORT		
MPI_DIST_GRAPH_NEIGHBORS	MPI_GRAPH_MAP	MPI_PACK		
MPI_DIST_GRAPH_NEIGHBORS_COUNT	MPI_GRAPH_NEIGHBORS	MPI_PACK_EXTERNAL		
MPI_DUP	MPI_GRAPH_NEIGHBORS_COUNT	MPI_PACK_EXTERNAL_SIZE		
MPI_ERRHANDLER_C2F	MPI_GRAPHDIMS_GET	MPI_PACK_SIZE		
MPI_ERRHANDLER_CREATE	MPI_GREQUEST_COMPLETE	MPI_PCONTROL		
MPI_ERRHANDLER_F2C	MPI_GREQUEST_START	MPI_PROBE		
MPI_ERRHANDLER_FREE	MPI_GROUP_C2F	MPI_PUBLISH_NAME		
	MPI_GROUP_COMPARE	MPI_PUT		
		MPI_TYPE_CREATE_INDEXED_BLOCK		
		MPI_TYPE_CREATE_INDEXED_BLOCK		
		MPI_TYPE_CREATE_KEYVAL		
		MPI_TYPE_CREATE_RESIZED		
		MPI_TYPE_CREATE_STRUCT		
		MPI_TYPE_CREATE_SUBARRAY		



MPI Continuous to Evolve

MPI 3.0 ratified in September 2012

- Available at <http://www mpi-forum.org/>
- Several major additions compared to MPI 2.2



Available through HLRS
-> MPI Forum Website

MPI 3.1 ratified in June 2015

- Inclusion for errata (mainly RMA, Fortran, MPI_T)
- Minor updates and additions (address arithmetic and non-block. I/O)
- Adaption in most MPIS progressing fast



On the Road to MPI 4.0

Some of the topics currently under discussion

- Better support for persistent communication
- Improved tool interfaces
- Support for fault tolerance
- Improved support for Hybrid programming
 - How to better interact with OpenMP or CUDA
 - MPI+X model
- Streaming communication
- Improved runtime support for scaling

The MPI Forum Drives MPI

<https://www mpi-forum.org/>

Standardization body for MPI

- Discusses additions and new directions
- Oversees the correctness and quality of the standard
- Represents MPI to the community

Open membership

- Any organization is welcome to participate
- Consists of working groups and the actual MPI forum
- Physical meetings 4 times each year (3 in the US, one with EuroMPI conference)
 - Working groups meet between forum meetings (via phone)
 - Plenary/full forum work is done mostly at the physical meetings
- Voting rights depend on attendance
 - An organization has to be present two out of the last three meetings (incl. the current one) to be eligible to vote

Technical work driven by the working groups

- <https://www mpi-forum.org/mpi-40/>

Typical Way New Features Get Added to MPI

1. New items brought to a matching working group for discussion
2. Creation of preliminary proposal
3. Socializing of idea driven by the WG
Through community discussions, user feedback, publications, ...

Development of full proposal
In many cases accompanied with prototype development work
4. MPI forum reading/voting process
One reading
Two votes
Slow and consensus driven process
5. Once enough topics are completed:
Publication of a new standard



Summary

Distributed Memory Architectures

- Highly scalable to thousands (or more) nodes
- Example: SuperMUC
- Facilities are becoming more important as we scale

HPC Applications are diverse and complex

- Wide range of spectrum
- Different reasons for HPC: faster solution, larger problem, ensembles, ...

Programming distributed memory systems

- Individual processes communicating through the network
- Messaging libraries are necessary

MPI as the most widely used HPC standard for message passing

- Currently in version 3.1 with wide range of functionality
- Continues to evolve through the open standardization body MPI Forum
- So far covered: boilerplate and basic point to point communication
- Central concept: communicators as communication contexts