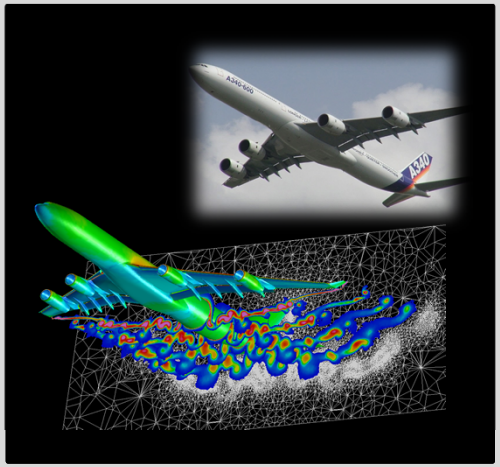


High Performance Computing

ADVANCED SCIENTIFIC COMPUTING

Dr. – Ing. Morris Riedel
Adjunct Associated Professor
School of Engineering and Natural Sciences, University of Iceland
Research Group Leader, Juelich Supercomputing Centre, Germany

LECTURE 4

Advanced MPI Techniques

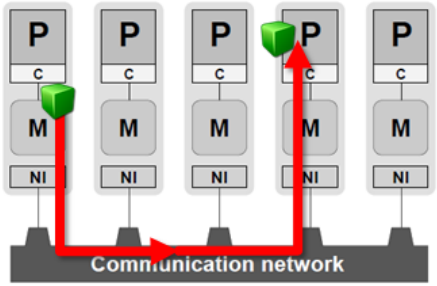
September 12th, 2017
Room TG-227

UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

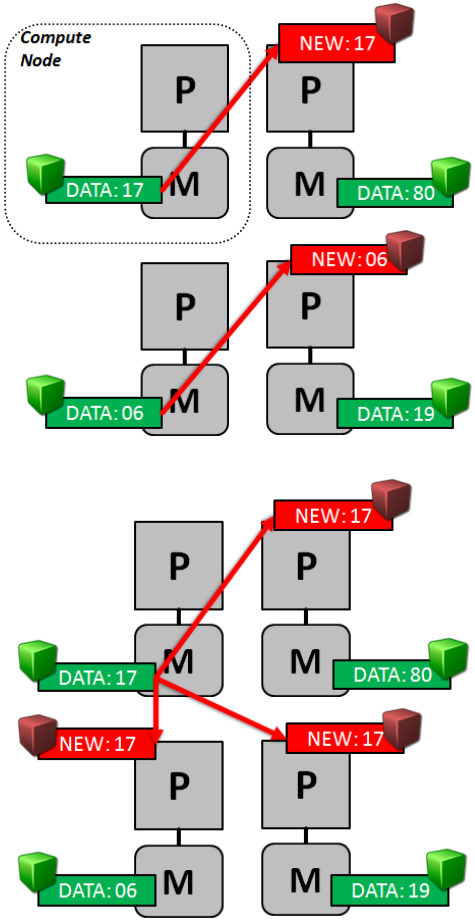
JÜLICH FORSCHUNGSZENTRUM

Review of Lecture 3 – Parallel Programming with MPI

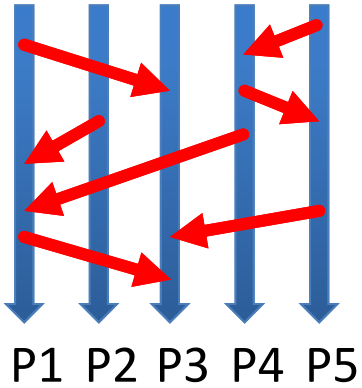
- MPI = Message Passing Interface

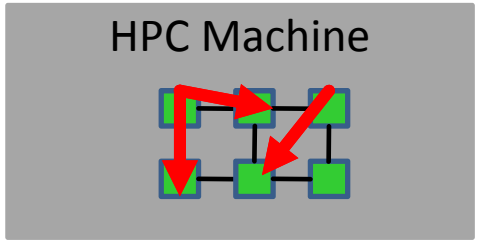


- HPC Communication Library



- (Complex) Parallel Programming





Modified from [1] Introduction to High Performance Computing for Scientists and Engineers

Outline of the Course

1. High Performance Computing
2. Parallelization Fundamentals
3. Parallel Programming with MPI
4. Advanced MPI Techniques
5. Parallel Algorithms & Data Structures
6. Parallel Programming with OpenMP
7. Hybrid Programming & Patterns
8. Debugging & Profiling Techniques
9. Performance Optimization & Tools
10. Scalable HPC Infrastructures & GPUs
11. Scientific Visualization & Steering
12. Terrestrial Systems & Climate
13. Systems Biology & Bioinformatics
14. Molecular Systems & Libraries
15. Computational Fluid Dynamics
16. Finite Elements Method
17. Machine Learning & Data Mining
18. Epilogue

+ additional practical lectures for our hands-on exercises in context

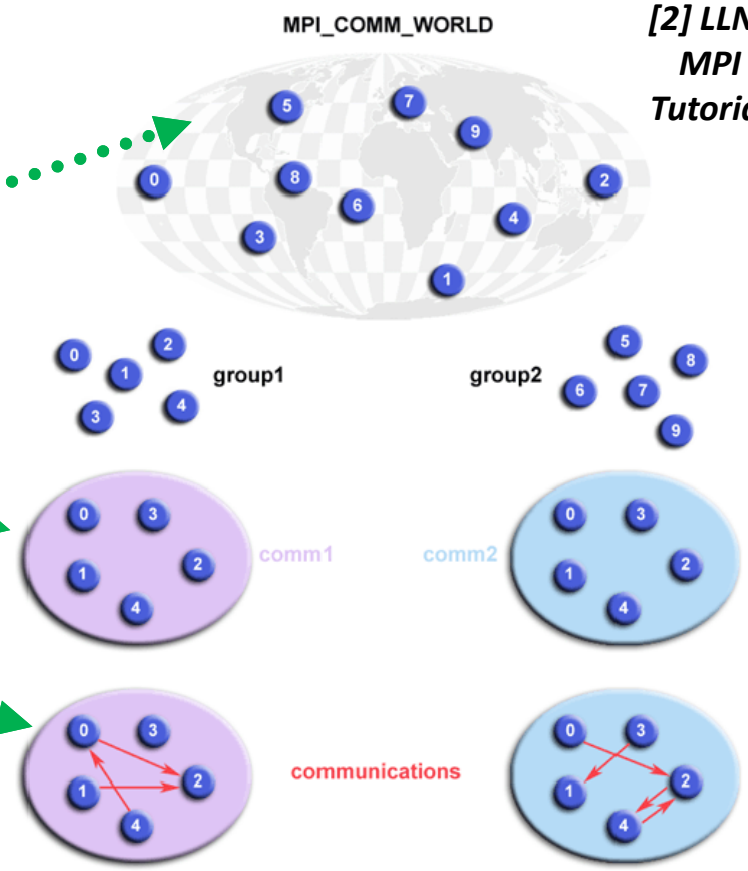
Outline

- MPI Communication Techniques
  - MPI Communicators & Sub-Groups
  - Cartesian Communicator
  - Hardware & Communication Issues
  - Network Interconnects & Task-Core Mappings
  - Application Examples in Context
- MPI Parallel I/O Techniques
  - I/O Terminologies & Challenges
  - Parallel Filesystems & Striping Technique
  - MPI I/O Techniques
  - Higher-Level I/O Libraries
  - Portable File Formats

- **Promises from previous lecture(s):**
- ***Lecture 1:* Lecture 3 & 4 will give in-depth details on the distributed-memory programming model with MPI**
- ***Lecture 3:* Lecture 4 will provide pieces of information about the often used MPI cartesian communicator**

MPI Communicators & MPI Rank – Revisited (cf. Lecture 3)

- Each MPI activity specifies the context in which a corresponding function is performed
  - MPI_COMM_WORLD (region/context of all processes)
  - Create (sub-)groups of the processes / virtual groups of processes
  - Peform communications only within these sub-groups easily with well-defined processes



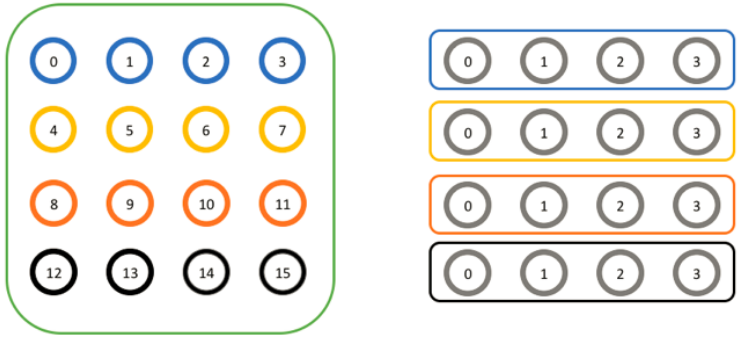
[2] LLNL MPI Tutorial

(numbers reflect unique identity of processor named 'MPI rank)

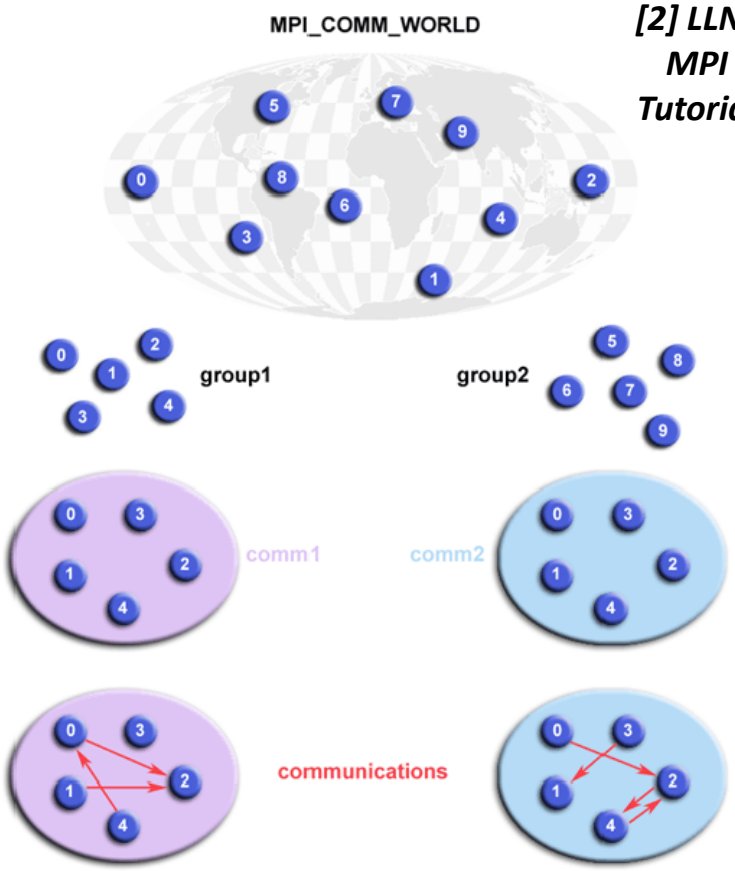
- **Using communicators wisely in collective functions can reduce the number of affected processors**
- **MPI rank is a unique number for each processor**
- **MPI ranks in different communicators represent different unique identifiers**

MPI Communicators – Create Sub-Group Communicators

- Create (sub-)groups of the processes / virtual groups of processes
  - E.g. split existing communicator `MPI_Comm_split()`
  - Creates a new smaller communicator out of a larger communicator by splitting its current ranks (e.g. rows)



[3] MPITutorial



[2] LLNL MPI Tutorial

(numbers reflect unique identity of processor named 'MPI rank)

➢ **Assignment #1 & #2 will make use of different communicators created using sub-groups**

Split Communicator Example

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

int main (int argc, char** argv) {
  int world_rank, world_size;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  ...
  int color = world_rank / 4;
  ...
  MPI_Comm row_comm;
  MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &row_comm);
  ...
  int row_rank, row_size;
  MPI_Comm_size(row_comm, &row_size);
  MPI_Comm_rank(row_comm, &row_rank);
  ...
  printf("WORLD RANK/SIZE: %d/%d \t ROW RANK/SIZE: %d/%d\n",
         world_rank, world_size, row_rank, row_size);
  MPI_Comm_free(&row_comm);
  MPI_Finalize();
  return 0;
}
```

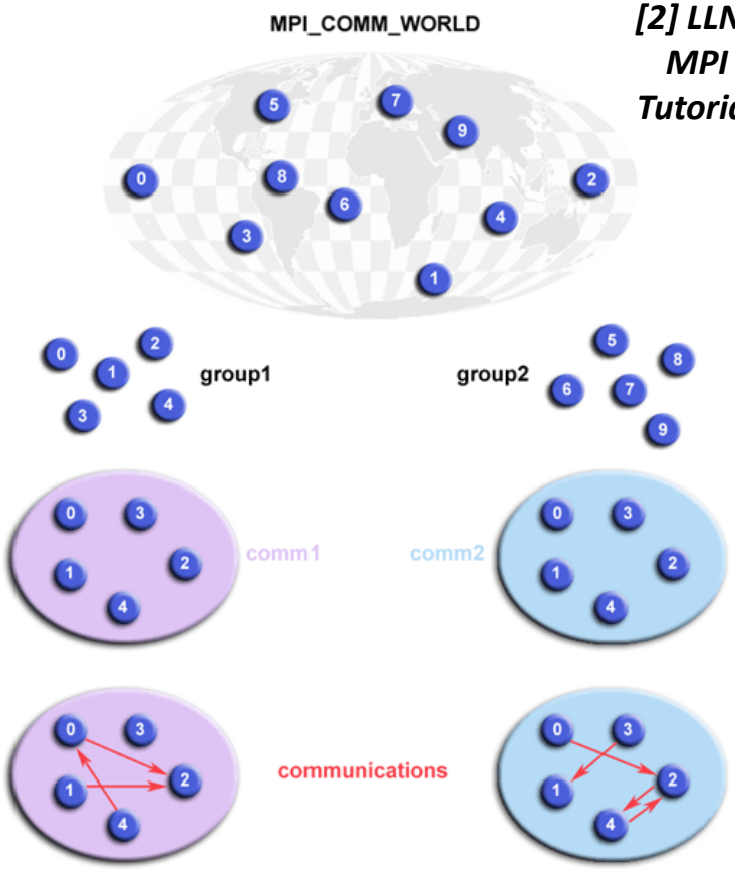
modified from [3] MPITutorial

- **MPI_COMM_WORLD with all processors (cf. Lecture 3)**
- **Splitting scheme according to illustration matching colors / rows**
- **Definition of a new communicator and a split of the existing MPI_COMM_WORLD communicator using the defined row scheme**
- **Different ranks and sizes for a row communicator**
- **Print different identities in both communicators**
- **Free communicator**

MPI Communicators – Create Cartesian Communicator

- Create (sub-)groups of the processes / virtual groups of processes
  - E.g. optimized for cartesian topology `MPI_Cart_create ()`
  - Creates a new communicator out of MPI_COMM_WORLD
  - Dims: array with length for each dimension
  - Periods: logical array specifying whether the grid is periodic
  - Reorder: Allow reordering of ranks in output communicator

[2] LLNL MPI Tutorial



(numbers reflect unique identity of processor named 'MPI rank)

➢ **Assignment #3 will make use of the cartesian communicator in a simple application example**

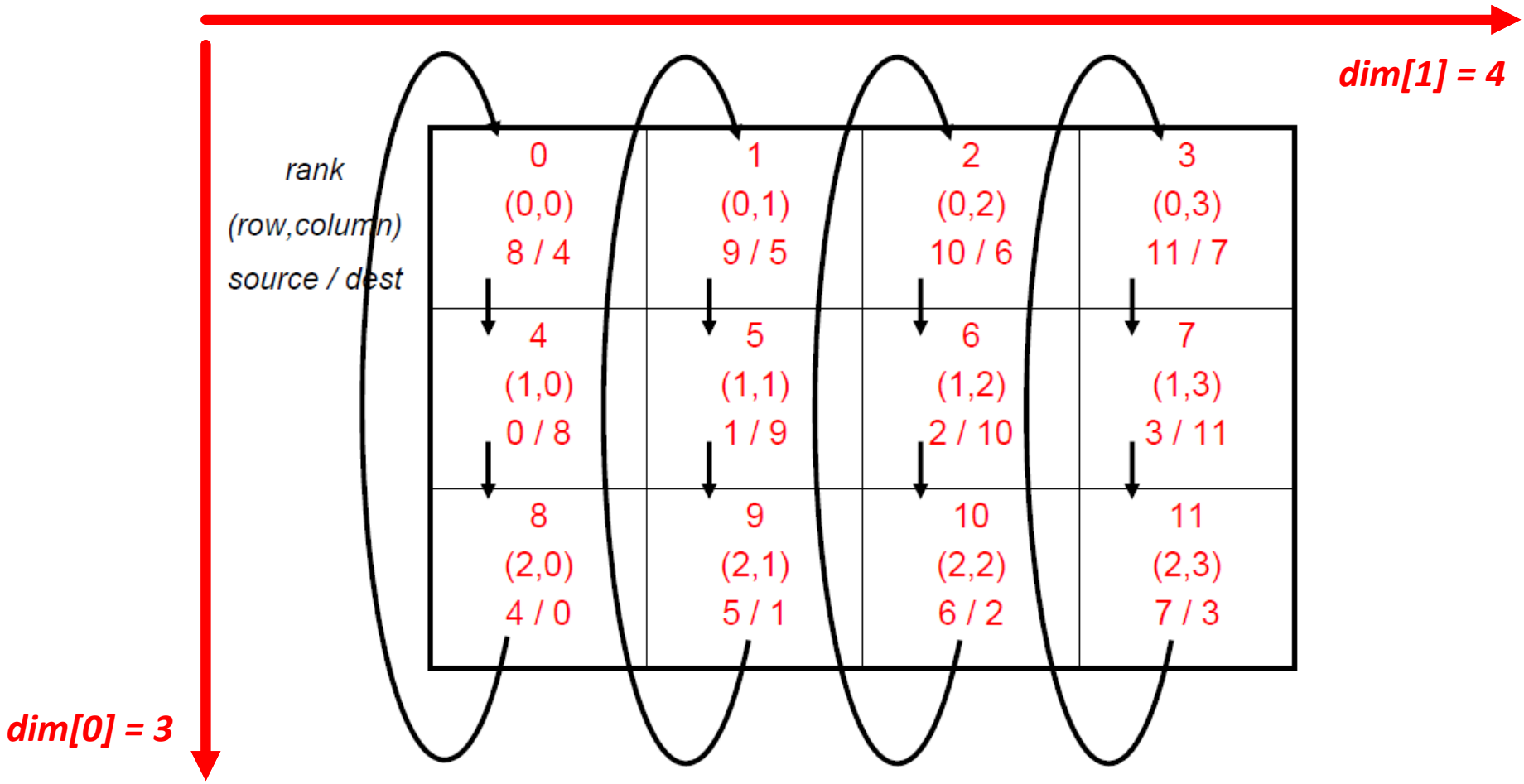
Cartesian Communicator Example (1)

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

int main (int argc, char** argv) {
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  dims[0]=3; dims[1] = 4;
  periods[0]=true; periods[1]=true;
  reorder = false;
  MPI_Cart_create(MPI_COMM_WORLD, 2, dims,
            periods, reorder, &comm_2d);
  MPI_Cart_coords(comm_2d, rank, 2, &coords);
  MPI_Cart_shift(comm_2d, 0, 1, &source, &dest);
  ...
  a = rank; b = 1;
  MPI_Sendrecv(a, 1, MPI_REAL, dest, 13, b, 1,
            MPI_REAL, source, 13, comm_2d, &status);
  ...
  MPI_Finalize();
  return 0;
}
```

modified from [4] German MPI Lecture

- **Preparing parameter dims as array with length for each dimension (here 3 x 4)**
- **Preparing parameter periods as logical array specifying whether the cartesian grid is period**
- **Preparing parameter reorder as not reordering of ranks in output communicator**
- **MPI_Cart_create() creates a new communicator (cartesian structure)**
- **MPI_Cart_coords() obtains process coordinates in cartesian topology**
- **MPI_Cart_shift() obtains 'ranks' for shifting data in cartesian topology**

Cartesian Communicator Example (2)



(‚cartesian structure‘)

modified from [4] German MPI Lecture

- **Assignment #3 will make use of the cartesian communicator in a simple application example**

Hardware & Communication Issues

- **Communication overhead can have significant impact on application performance**
- **Characteristics of interconnects of compute nodes/cpus affect parallel performance**

- Parallel Programming can cause communication issues
  - E.g. need for synchronisation in applications, e.g use of `MPI_Barriers()`
- Wide varieties of networks in HPC systems are available
  - Different network topologies of different types of networks used in HPC
- Gigabit Ethernet
  - Simple/cheep and good for high throughput computing (HTC)
  - Often too slow for parallel programs that require fast interconnects
- Infiniband
  - Fast, thus dominant distributed-memory computing interconnect today

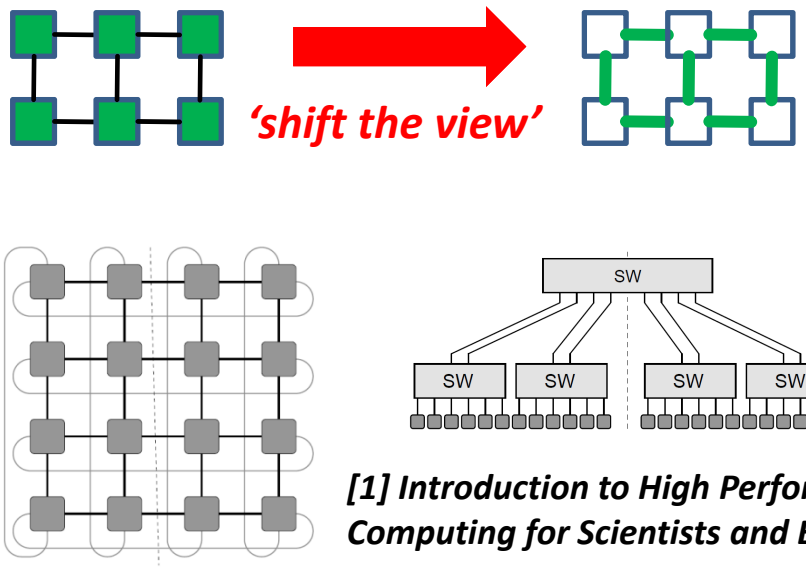
Communication Issues – Synchronisation with MPI Barrier

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

int main (int argc, char** argv) {
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if (rank == 0) {
    sleep(10);
  }
  ...
  if (rank == 1) {
    storeResultsToFile();
  }
  ...

  MPI_Barrier(MPI_COMM_WORLD);

  ...
  MPI_Finalize();
  return 0;
}
```

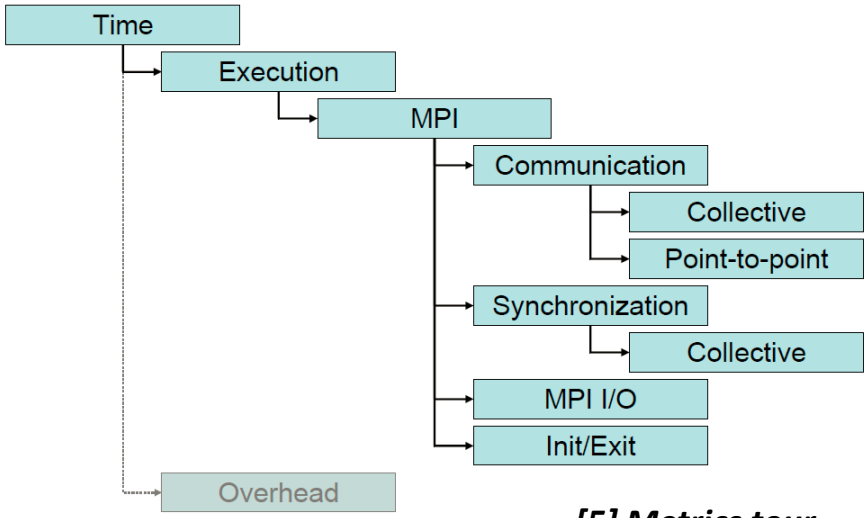
- **One reason to require a synchronisation across processors is that one rank is performing some extraordinary long work, not others**
- **Sleep() is a function that puts a processor to sleep and thus doing basically nothing. Still a parallel computing resource is not usable for other users since it is typically exclusively allocated to one user.**
- **Another reason to require a synchronisation across processors is that one rank performs I/O operations of some kind**
- **MPI_Barrier() blocks the caller until all processes in the communicator have called it for synchronisation**

Optimization & Dependencies on Hardware & I/O

- Optimizations in terms of software & hardware are important
  - Optimization can be interpreted as using ‘dedicated‘ hardware features
  - E.g. network interconnections enable different used ‘network topologies‘
  - E.g. parallel codes are tuned applying parallel I/O with parallel filesystems



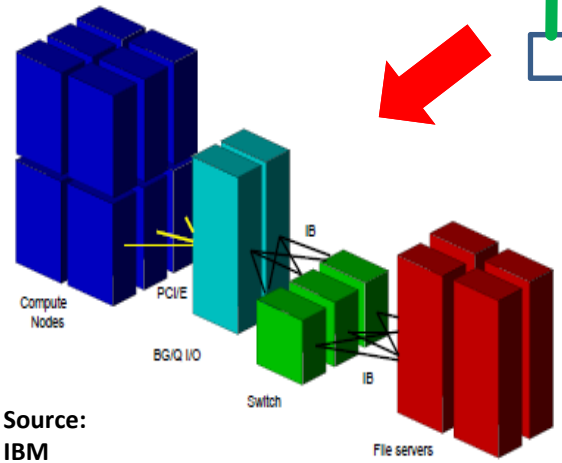
[1] Introduction to High Performance Computing for Scientists and Engineers



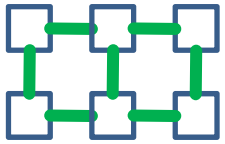
[5] Metrics tour

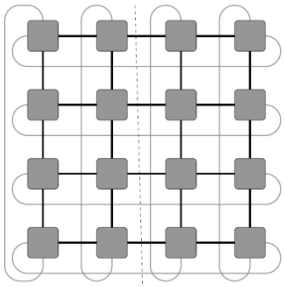
Complex Network Topologies & Challenges

- Large-scale HPC Systems have special network setups
  - Dedicated I/O nodes, fast interconnects, e.g. Infiniband (IB)
  - Different network topologies, e.g. tree, 5D Torus network, mesh, etc. (raise challenges in task mappings and communication patterns)



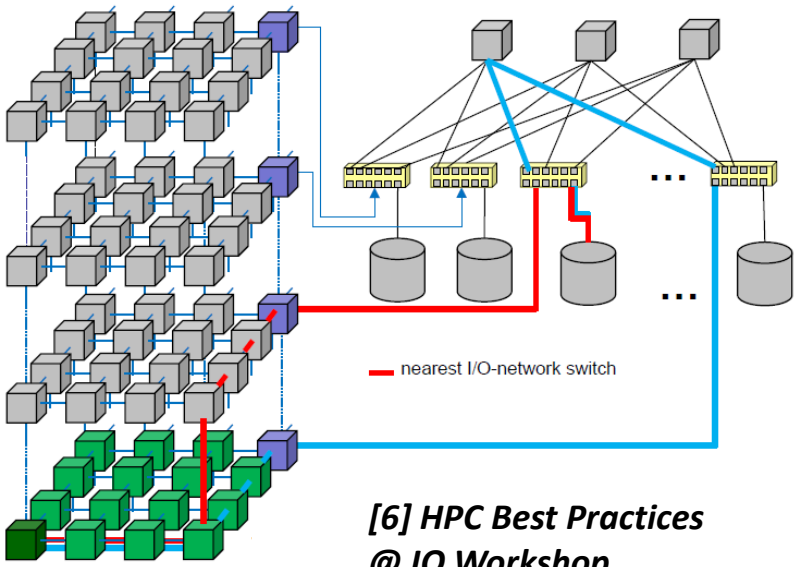
Source: IBM





[1] Introduction to High Performance Computing for Scientists and Engineers

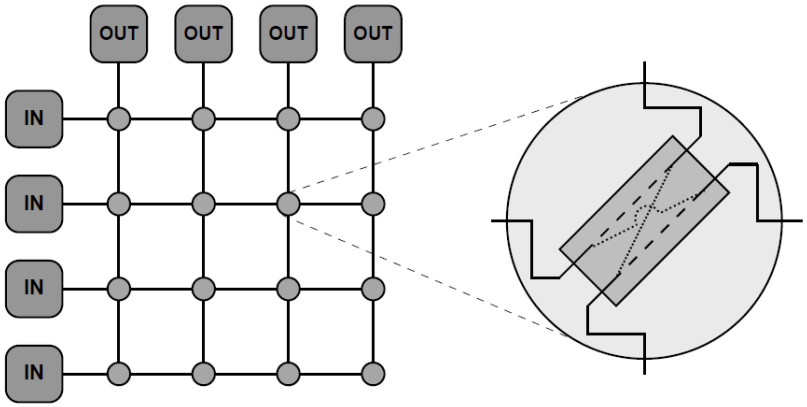
Contention and cross traffic caused by multiple I/O streams



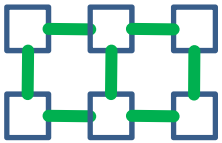
[6] HPC Best Practices @ IO Workshop

Network Building Block 'Switch' inside a HPC system

- Single fully non-blocking switch
  - All pairs of ports can use their full bandwidth concurrently
  - E.g. 2D cross-bar switch and each circle represents possible connections between two involved IN/OUT devices
  - Aka '2x2 switching element'
  - Aka 'four-port non-blocking switch'



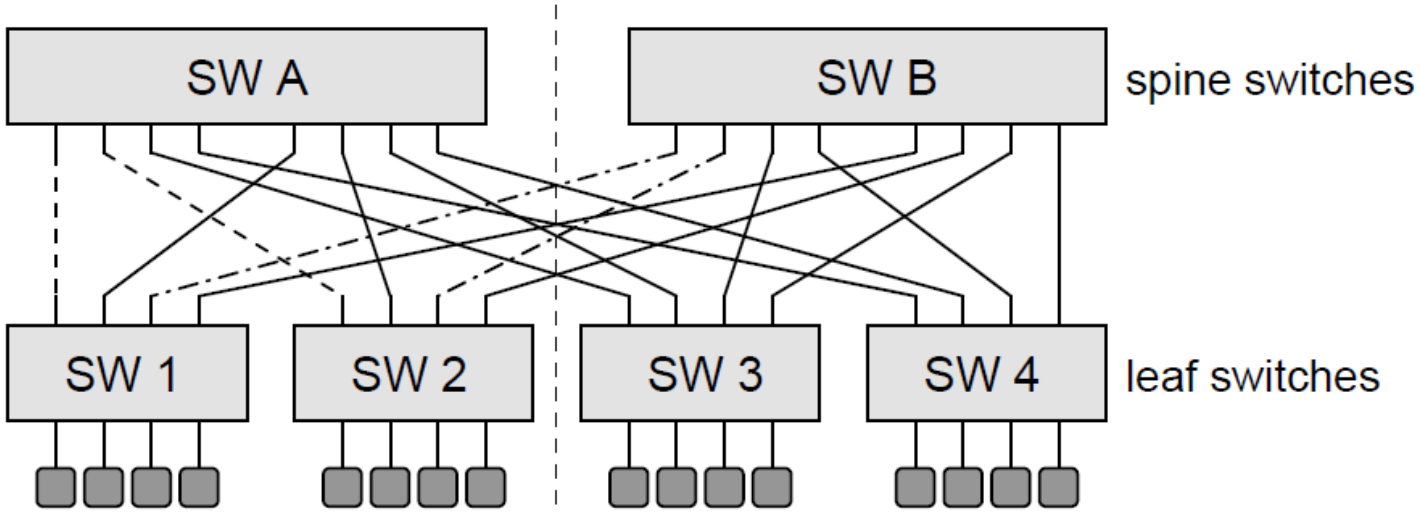
[1] Introduction to High Performance Computing for Scientists and Engineers



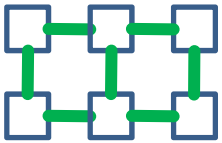
- Alternative is a [partly/completely] single switch with bus design with limited bandwidth

- **Think about workers processing data and interacting with each other → switch matters!**
- **Advanced programming techniques need to take the hardware interconnect into account**

Combining Network Building Blocks as FatTree (1)



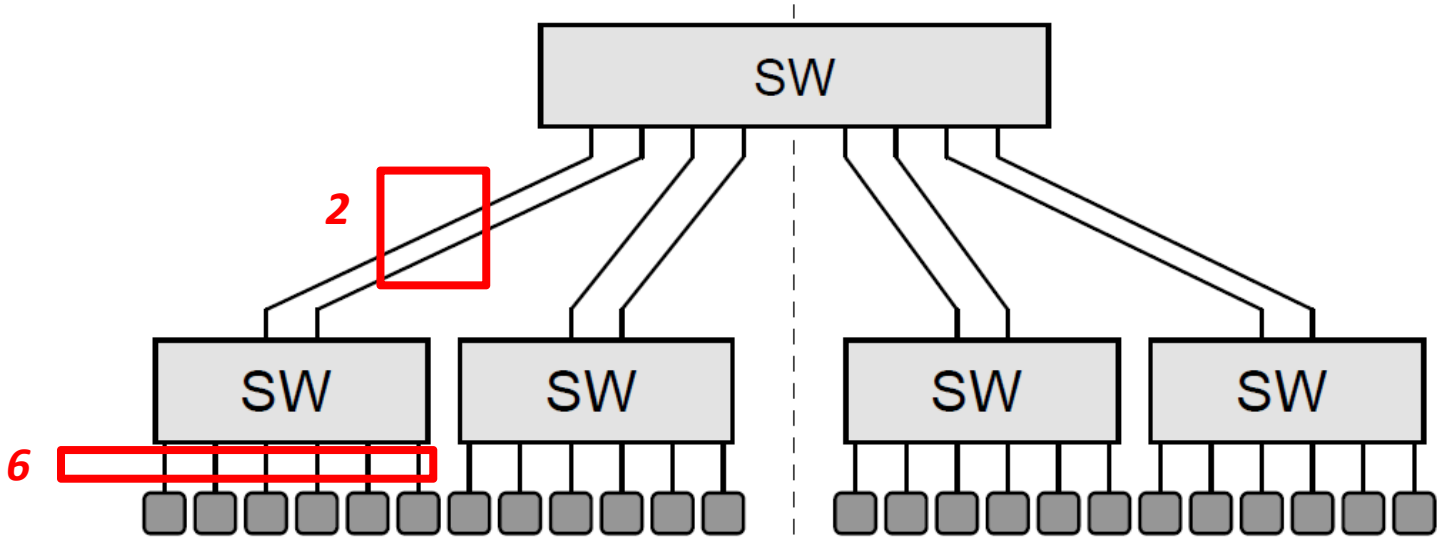
[1] Introduction to High Performance Computing for Scientists and Engineers



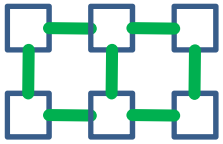
- Fully non-blocking full-bandwidth fat-tree network
  - Having two switch layers (leaf and spine)
  - Keeps the 'non-blocking' feature across the whole system via two layers

- **Here a group of workers processing data 'enjoy' full non-blocking communication**
- **Location of the workers here is not very crucial to the application performance**

Combining Network Building Blocks as FatTree (2)



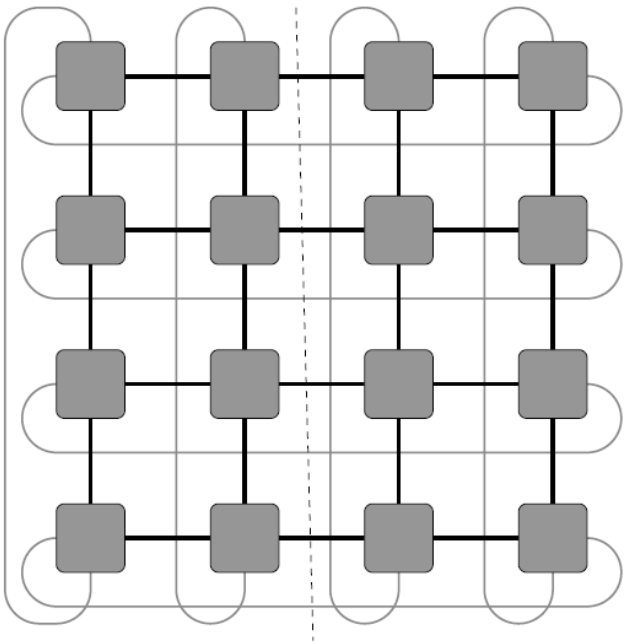
Modified from [1] Introduction to High Performance Computing for Scientists and Engineers



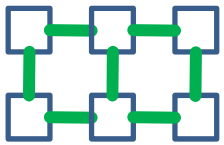
- Fat-tree network with bottleneck (when # CPUs high)
  - Bottleneck is '1:3 oversubscription' of communication link to spine
  - Only four nonblocking pairs of connections are possible
  - Common in very large systems → safe costs (cable & switch hardware)

- **The location of the workers processing data is crucial for application performance here**

Mesh Networks



[1] Introduction to High Performance Computing for Scientists and Engineers

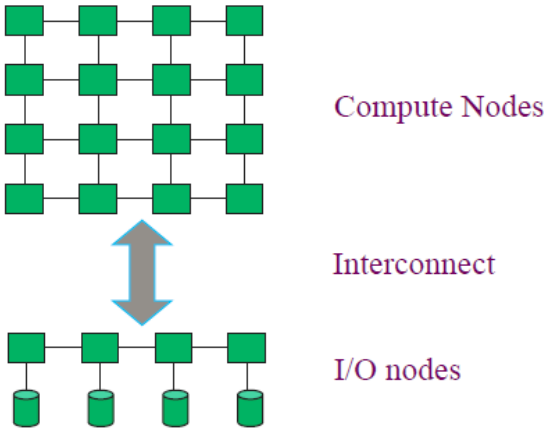


- Often in the form of multi-dimensional hypercubes
- Computing entity is located at each Cartesian grid intersection
- Idea: connections are wrapped around the boundaries of the hypercube to form a certain torus topology
- No direct connections between entities that are not next neighbours (but ok!)
- Example: A 2D torus network
- Bisection bandwidth scales with $\sqrt{N}$

- **Fat-Tree switches have limited scalability in very large systems (price vs. performance)**
- **Bisection bandwidth with scaling in large systems often via mesh networks (e.g. 2D torus)**

Example of Large-scale HPC Machine and I/O Setup

[8] JUQUEEN HPC System



[7] R. Thakur, PRACE Training, Parallel I/O and MPI I/O

- Example: JUQUEEN
  - IBM BlueGene/Q
- Compute Nodes
  - 28 racks (7 rows à 4 racks) 28,672 nodes (458,752 cores)
  - Rack: 2 midplanes à 16 nodeboards (16,384 cores)
  - Nodeboard: 32 compute nodes
  - Node: 16 cores
- I/O Nodes
  - 248 (27x8 + 1x32) connected to 2 CISCO Switches

- **The I/O node cabling connects the computing nodes via dedicated I/O nodes to storages**

Communication Optimization by Task-Core Mappings (1)

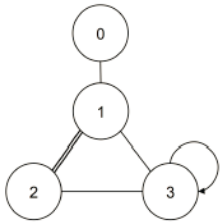
- Approach:
  - Place often-communicating processes on neighboring nodes
  - Requires known communication behavior
  - Measurements via MPI profiling interface

Execution units
i.e. processes

processing elements
i.e. CPUs

- **Optimal placement is an NP-hard-problem**
- **n! possibilities to map n execution units to the same number of n processing elements**
- **Topology aware task mapping for I/O patterns**

- Identification of applicable 'task-core mapping' approach
  - E.g. graph model describes task communication & hardware characteritics
  - Obtain communication characteristics via sourcecode or profiling
  - Obtain hardware characteristics via vendor information (e.g. IBM redbooks)



$G_t = (T, E, f)$

$T = \{t_0, t_1, t_2, t_3\}$

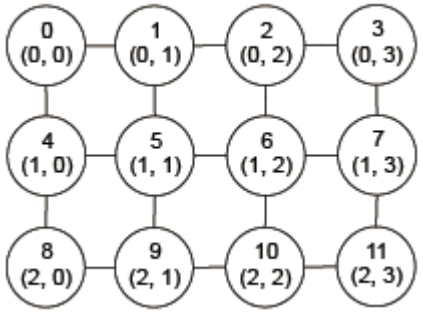
$E = \{\{t_0, t_1\}, \{t_1, t_2\}, \{t_1, t_3\}, \{t_2, t_3\}, \{t_3\}\}$

f:

$e \in E_1$	$f_1(e)$
$\{t_0, t_1\}$	1
$\{t_1, t_2\}$	2
$\{t_1, t_3\}$	1
$\{t_2, t_3\}$	1
$\{t_3\}$	1

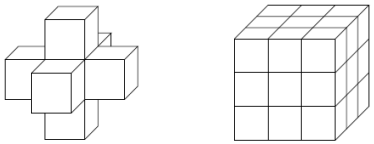
Communication Optimization by Task-Core Mappings (2)

- Application of calculated mappings
  - For regular graphs (tori): Mapping of regular shapes
  - E.g. experiments run on Bluegene/Q JUQUEEN



- Scientific application (cf. Lecture 2)
  - Heatmap as three-dimensional simulation for heat expansion

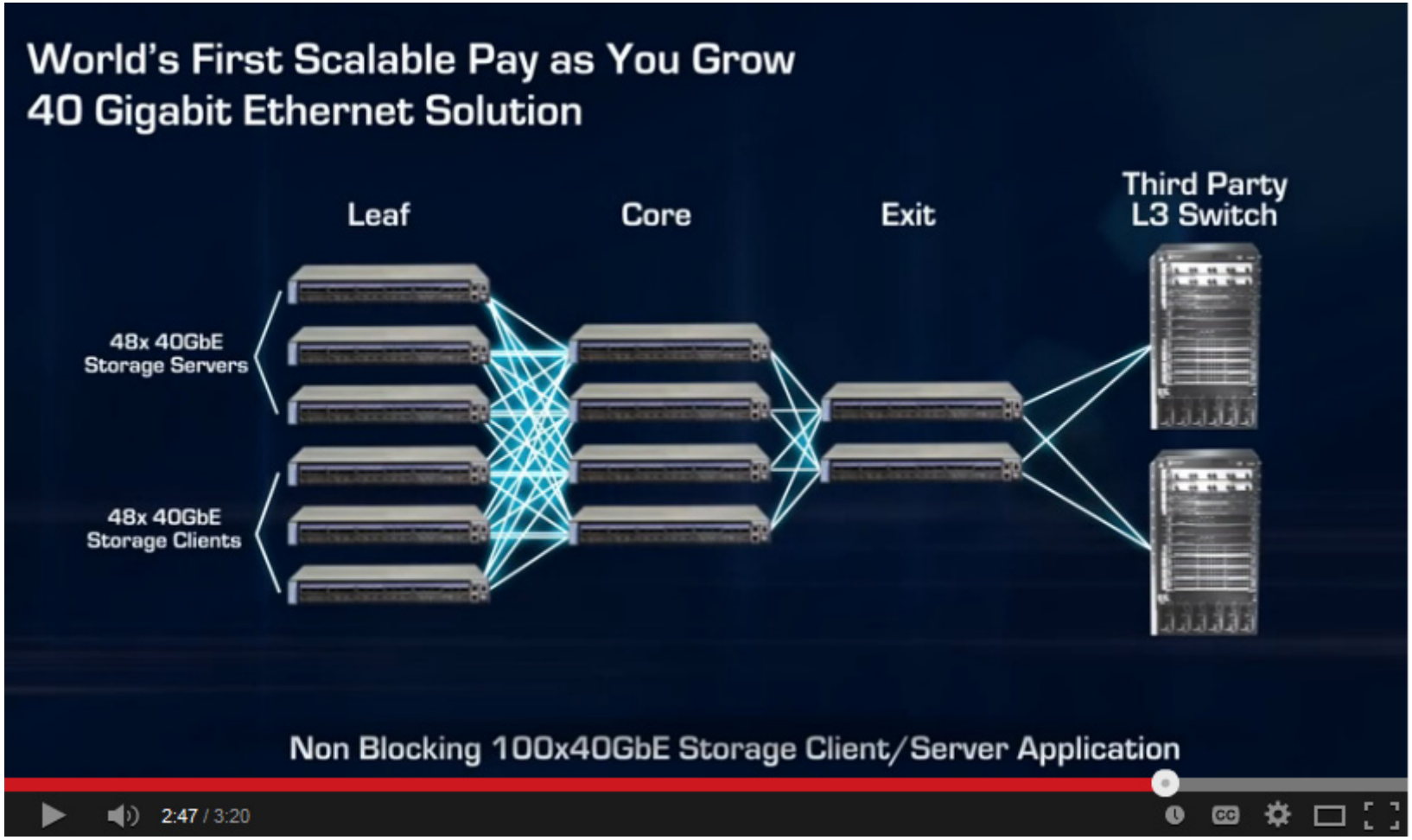
  - Values of boundary cells are exchanges with neighboring placed ranks
  - Heatmap is divided into equally sized cubes
  - Heat expansion per cube is calculated by a single rank
  - Two different expansion algorithms
  - Using e.g. 'heuristics' for task/core placements




```
for (z = 1; z <= size; z++) {
      for (y = 1; y <= size; y++) {
        for (x = 1; x <= size; x++) {
          new_map[x][y][z] = ( old_map[x][y][z-1]
  + old_map[x][y-1][z] + old_map[x-1][y][z]
  + old_map[x][y][z] + old_map[x+1][y][z]
  + old_map[x][y+1][z] + old_map[x][y][z+1] ) / 7;
        }
      }
    }
```

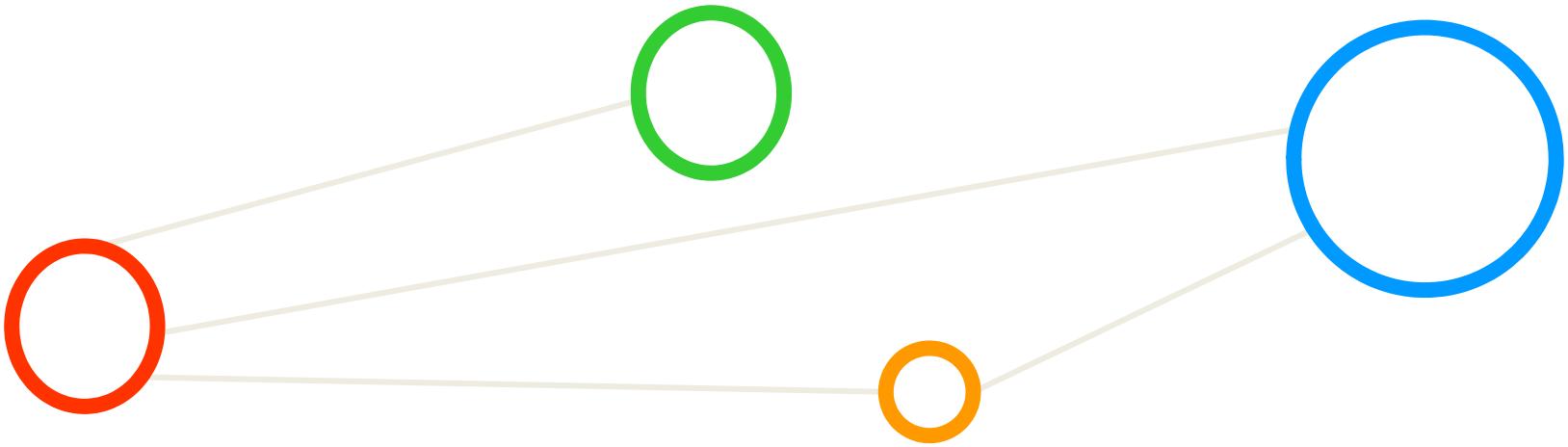
- **Optimized task core mappings enable performance gains between 1-3% (heatmap example)**

[Video] Mellanox



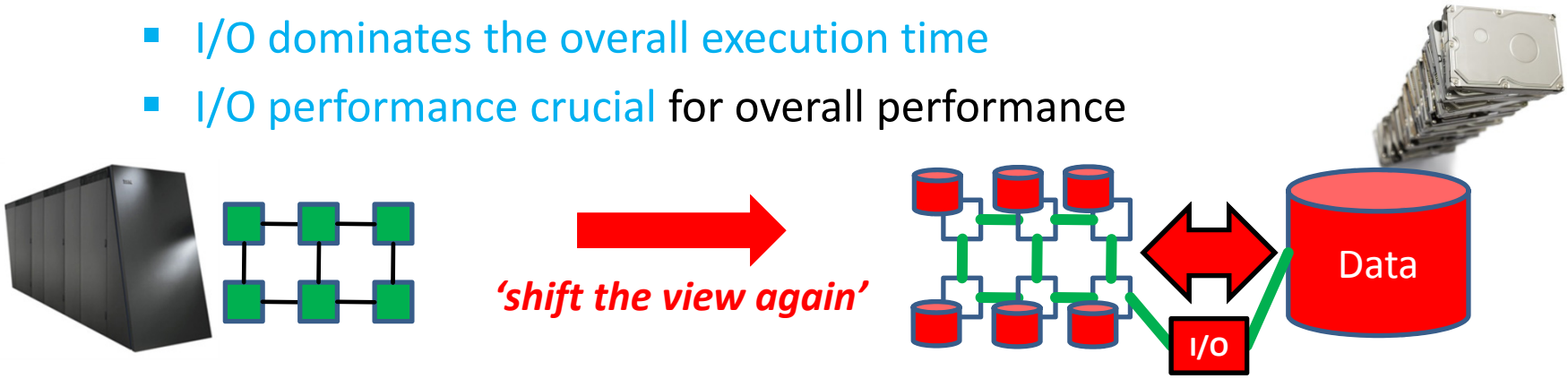
[9] Mellanox YouTube Video

Parallel I/O Techniques



Parallel I/O Techniques – Motivation

- (Parallel) applications that emphasize on the importance of data
  - Not all data-intensive or data-driven applications are 'big data' (volume)
  - HPC simulations of the real world that generates very large volumes of data
- Synthesize new information from data that is maintained in distributed (partly unique) repositories and archives
  - Distributed across different organizations and computers/storages
- Data analysis applications that are 'I/O bound'
  - I/O dominates the overall execution time
  - I/O performance crucial for overall performance



What means I/O?

- **Input/Output (I/O) stands for data transfer/migration from memory to disk (or vice versa)**

Modified from [1] Introduction to High Performance Computing for Scientists and Engineers

- Important (time-sensitive) factors within HPC environments
  - Characteristics of the computational system (e.g. dedicated I/O nodes)
  - Characteristics of the underlying filesystem (e.g. parallel file systems, etc.)

➢ **The complementary Cloud Computing & Big Data Course offers distributed file system concepts**

I/O Challenges

- **An I/O pattern reflects the way of how an application makes use of I/O (files, processes, etc.)**

- I/O performance bottlenecks in many 'locations in applications'
  - Understanding depends on network & I/O patterns

- During an HPC application run
  - Consider the number of processes performing I/O
  - The number of files read or written by processes
  - Take into account how the files are accessed:
    (a) serial access via one process
    (b) shared access via multiple processes

- Before/After HPC application run
  - How can necessary files be made available/archived (e.g. tertiary storage)
  - E.g. retrieving a high number of small files from tapes takes very long time

Parallel Filesystems Concept

- **A parallel file system is optimized to support concurrent file access**
- **One file that is written to a parallel filesystem is broken up into 'blocks' of a configured size (e.g. typically less than 1MB each)**

- File Blocks
  - Distributed across multiple filesystem nodes
  - A single file is thus fully distributed across a 'disk array'
- Advantages
  - High reading and writing speeds for a single file
  - Reason: 'Combined bandwidth' of the many physical drives is high
- Disadvantages:
  - Filesystem is vulnerable to disk failures (e.g. one disk fails → lose file data)
  - Prevent data loss with 'RAID controllers' as part of the filesystem nodes
  - Redundant Array of Inexpensive Disks (RAID) levels trade-off vs. data loss

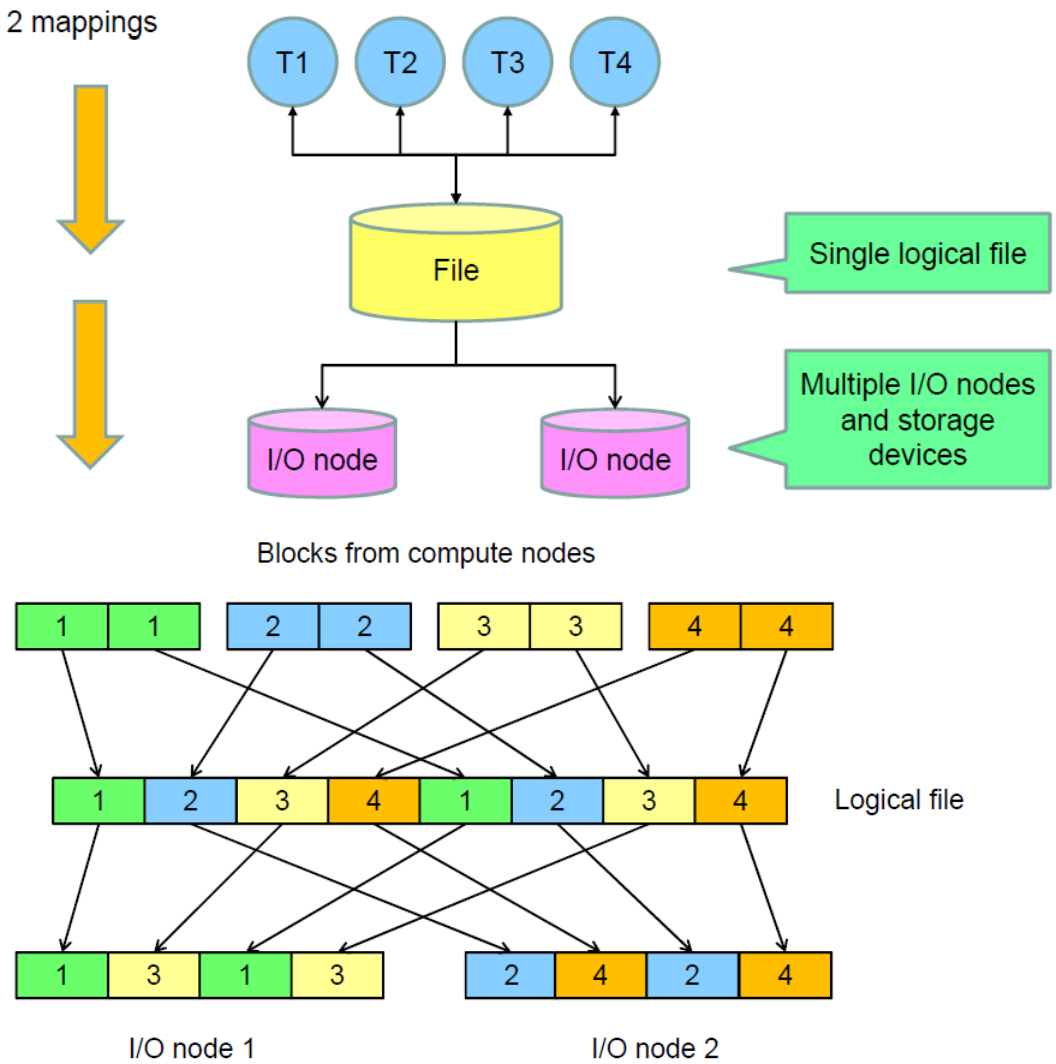
➢ **The complementary Cloud Computing & Big Data Course offers data storage details (e.g. RAIDs)**

Examples of Parallel File Systems

- **Widely used parallel file systems are GPFS (commercial) and Lustre (open source)**
- **In 2015 IBM rebranded GPFS as IBM Spectrum Scale due to 'Big Data' customers**

- General Parallel File System (GPFS) / IBM Spectrum Scale
  - Developed by IBM
  - Available for AIX and Linux
- Lustre
  - Developed by Cluster File Systems, Inc. (bought by Sun)
  - Movement towards 'OpenLustre'
  - Name is amalgam of 'Linux and clusters'
- Parallel Virtual File System (PVFS)
  - Platform for I/O research and production file system for cluster of workstations ('Beowulfs')
  - Developed by Clemson University and Argonne National Laboratory

Concurrent File Access & Two Level Mapping



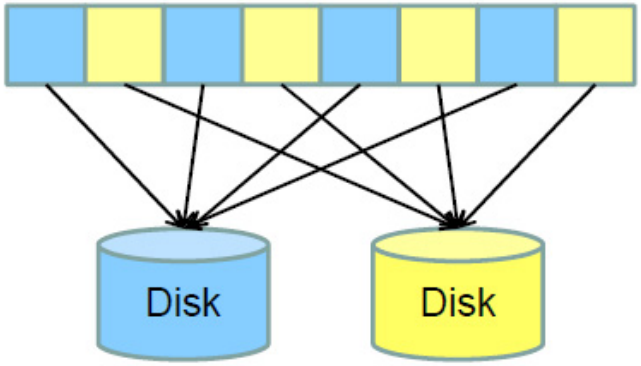
- **Concurrent file access means that multiple processes can access the same file at the same time**
- **Parallel file systems handle concurrent file access via 'single logical files' over multiple I/O nodes**
- **A two Level Mapping means to distribute blocks from compute nodes via logical files (1st level) using underlying multiple I/O nodes (2nd level)**

General Striping Technique

- Striping technique transforms view from a file to flexible 'blocks'

- **Striping refers to a technique where one file is split into fixed-sized blocks that are written to separate disks in order to facilitate parallel access**

- Striping is a general technique that appears in different contexts
- Two major important factors (to be configured)
  - (e.g. used in MPI I/O 'hints' also → later in this lecture)
  - 'Striping factor': number of disks
  - 'Striping unit': block size
  - Bit-level vs. block-level striping

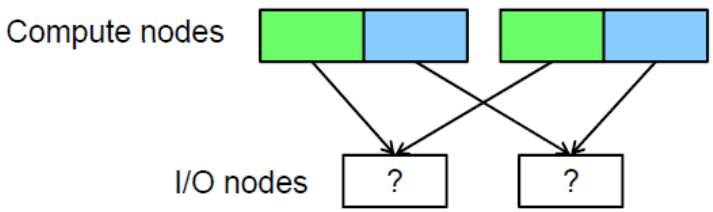


Parallel File Access

- Comparison with ‘sequential file system’ increases understanding
  - File system translates ‘file name’ into a File Control Block (FCB)
- Parallel File Systems
  - Every ‘I/O node’ manages a subset of the blocks
  - Consequence: Every file has (better: needs) an FCB on every I/O node
- File Access: Two ways to locate FCBs for a file
  - Every I/O node maintains directory structure
  - Central name server: Avoids replication of directory data
- File Creation
  - Filesystem chooses ‘the first’ I/O node (varies)
  - This particular I/O node (‘base node’) will store the first block of the file
  - Specific block is located when first I/O node and ‘striping pattern’ is known
- Question: What about ‘sequential consistency’ when writing?

Sequential Consistency

- Two processes on different compute nodes
  - Assumption: Both write to the 'same range of locations in a file'
- 'Sequential consistency' requires that all I/O nodes write their portions in the same order
  - Write request should appear to occur in well-defined sequence
  - But hard to enforce – I/O nodes may act independently
- Selected Possible Solutions
  - Locking entire files - Prevents parallel access (not an option)
  - Relaxed consistency semantics – application developer is responsible
  - Locking file partitions – prevents access to certain file partitions



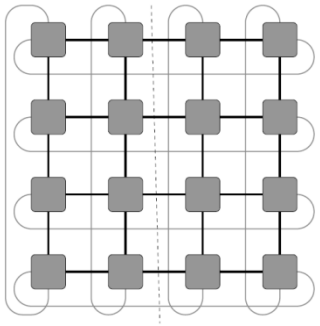
File Pointers

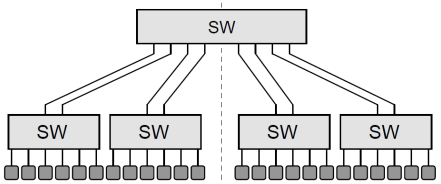
Thread 1:	Thread 2:
seek(location = 100);	seek(location = 200);
write(…, length = 50);	write(…, length = 150);

- Application needs to be aware of 'which processes use which parts of the file'
  - May require processes to skip file sections 'owned by others'
- Shared File Pointers
  - Common in shared-memory programs
  - Inefficient – serializes requests (update file pointer before completing request, 'eager update')
  - Inconsistencies if seek and write operations are separated
- Better use 'separate file pointers' or atomic seek & write
  - In UNIX pread() and pwrite() allow specification of 'explicit offset'

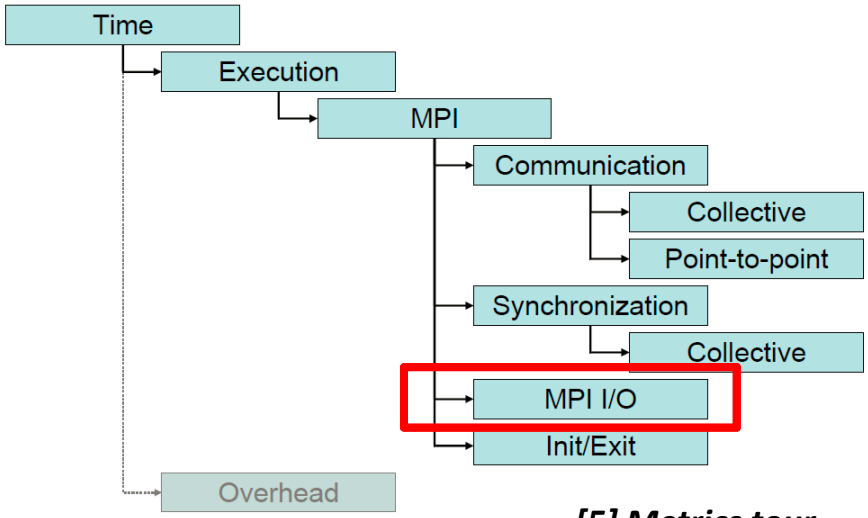
Optimization & Dependencies on Hardware & I/O Revisited

- Optimizations in terms of software & hardware are important
  - Optimization can be interpreted as using ‘dedicated‘ hardware features
  - E.g. network interconnections enable different used ‘network topologies‘
  - E.g. parallel codes are tuned applying parallel I/O with parallel filesystems





[1] Introduction to High Performance Computing for Scientists and Engineers

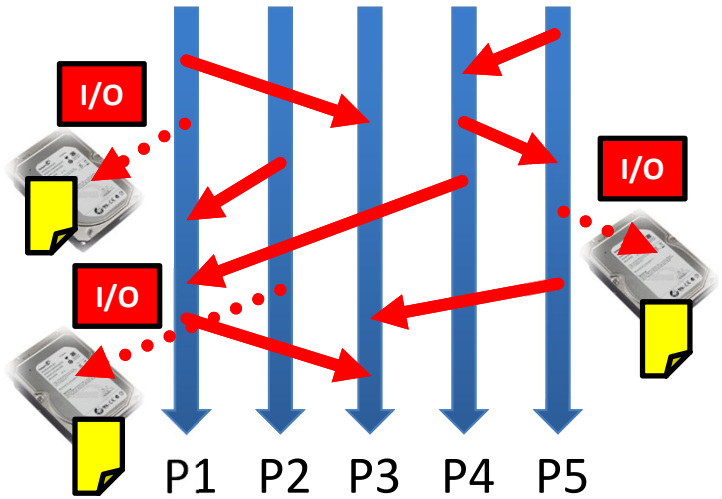


[5] Metrics tour

MPI I/O

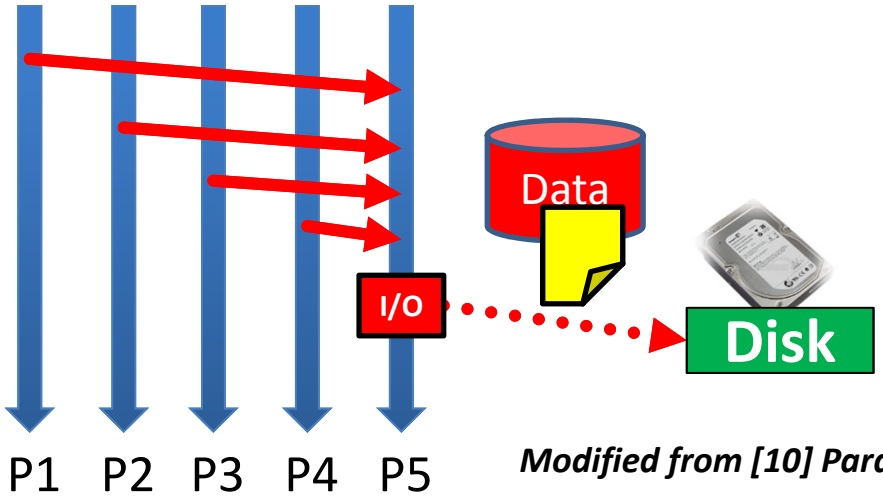
- **MPI I/O provides 'parallel I/O' support for parallel MPI applications**
- **Writing/Receiving files is similar to send/receive MPI messages, but to disk**

- Different operation modes
  - 'Blocking mode' to finish data operations, then continue computations
  - 'Non-blocking mode' (aka asynchronously) to perform computations while a file is being read or written in the background (typically more difficult to use)



- Supports the concept of 'collective operations'
  - Processes can access files each on its own or all together at the same time
- Provides advanced concepts like file views & data types/structures

Serial I/O: One process on behalf of many

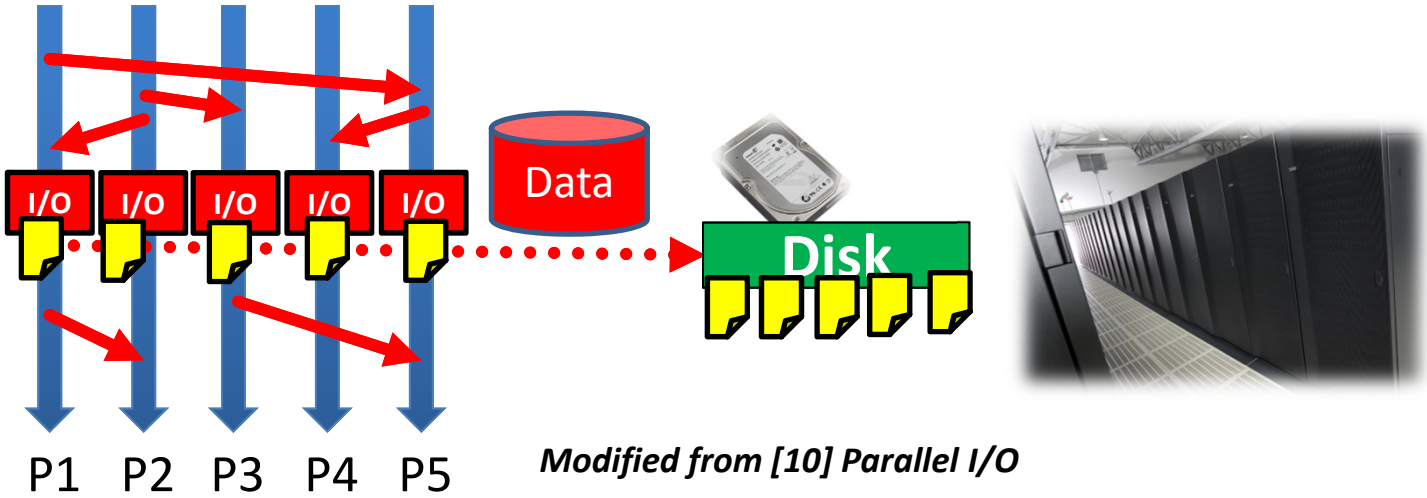


Modified from [10] Parallel I/O

- Only one process performs I/O on behalf of all other processors
  - Data aggregation or duplication
  - Limited by single I/O process
- No scalability for (big) data-intensive computing
  - Time increases linearly with amount of (big) data
  - Time increases with number of processes of the parallel application

- **Serial I/O: One process on behalf of many means that one process takes care of all I/O tasks**
- **Serial I/O increases communication and is slow as well as including load imbalance risks**

Parallel I/O: One file per process

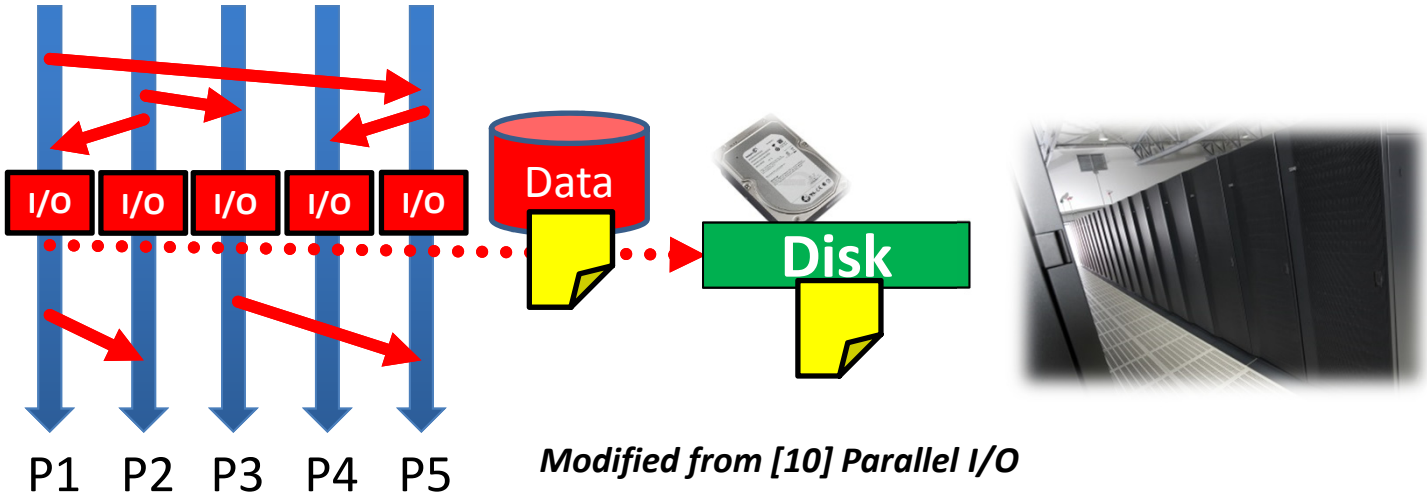


Modified from [10] Parallel I/O

- All processors perform I/O to individual files
  - Limited by file system capabilities
- No scalability for large number of processors
  - Number of files creates bottleneck with metadata operations
  - Number of simultaneous disk accesses creates 'contention' for file system resources (i.e. the disk cannot keep up with file I/O requests)

- **Parallel I/O: One file per process means that each process takes care of local I/O tasks alone**
- **Parallel I/O is good for scratch but not for output files in applications despite I/O balance**

Parallel I/O: Shared file



Modified from [10] Parallel I/O

- Each process performs I/O to a single file
  - The file access is 'shared' across all processors involved
  - E.g. MPI/IO functions represent 'collective operations'
- Scalability and Performance
  - 'Data layout' within the shared file is crucial to the performance
  - High number of processors can still create 'contention' for file systems

- **Parallel I/O: shared file means that processes can access their 'own portion' of a single file**
- **Parallel I/O with a shared file like MPI/IO is a scalable and even standardized solution**

Collective MPI-I/O: File Handles and Infos

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

int main (int argc, char** argv) {
  int rank, size;

  MPI_File fh;

  MPI_Info info;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  printf("Hello World, I am %d of %d\n", rank, size);

  MPI_Info_create(&info);

  MPI_Finalize();
  return 0;
}
```

- The MPI c header file includes also the functions that are part of MPI-I/O
- A MPI_File represents a file handler that reprents the file and process group of a communicator
- A MPI_Info represents a list of key/value pairs used for providing information to MPI-I/O
- MPI_Info_create creates an MPI_Info object

Collective MPI-I/O: Opening/Closing a file

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

int main (int argc, char** argv) {
  int rank, size;
  MPI_File fh;
  MPI_Info info;
  char *file_name = "outputfile";

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  printf("Hello World, I am %d of %d\n", rank, size);

  MPI_Info_create(&info);

  int rc = MPI_File_open( MPI_COMM_WORLD, file_name,
                          MPI_MODE_CREATE | MPI_MODE_RDWR,
                          info, &fh);
  // do something with the file!!!

  rc = MPI_File_close(&fh);
  MPI_Finalize();
  return 0;
}
```

- **Specifying a file_name that should be opened (or even be created) – but attention: The format is highly implementation dependent**
- **MPI_File_open opens a specific file collectively across all specified processes being part of the used communicator and sets a file handle**
- **MPI_File_close closes a specific file identified via a certain file handle**

Collective MPI-I/O: Writing integers to a file example

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

int main (int argc, char** argv) {
  int rank, size;
  MPI_File fh;
  MPI_Info info;
  char *file_name = "outputfile";
  int buf[10];
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  printf("Hello World, I am %d of %d\n", rank, size);
  MPI_Info_create(&info);
  int rc = MPI_File_open( MPI_COMM_WORLD, file_name,
                          MPI_MODE_CREATE | MPI_MODE_RDWR,
                          info, &fh);
  buf[0] = rank;

  // MPI_File_write_ordered(fh, buf, 1, MPI_INT, &status);
  MPI_File_write(fh, buf, 1, MPI_INT, &status);

  rc = MPI_File_close(&fh);
  MPI_Finalize();
  return 0;
}
```

- **Specify a buffer (here integer array) of a certain size**
- **Specify values for the buffer, here the rank of each MPI process that might be used as identification for further values following in the next parts of the corresponding file**
- **MPI_File_write or related versions write the <u>binary output</u> to the file**

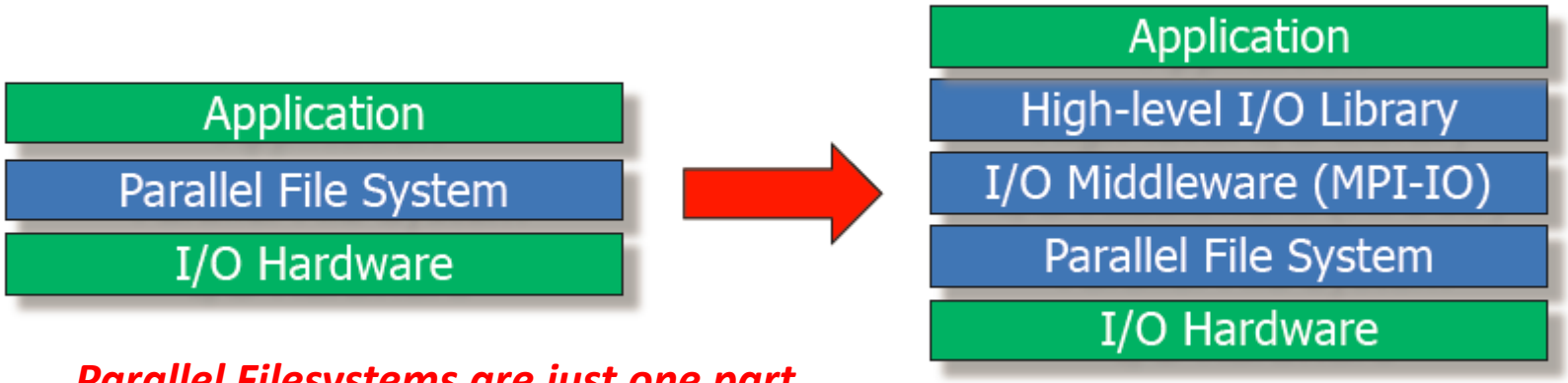
Collective MPI-I/O: Writing chars to a file example

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

int main (int argc, char** argv) {
  int rank, size;
  MPI_File fh;
  MPI_Info info;
  char *file_name = "outputfile";
  char buf[10];
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  printf("Hello World, I am %d of %d\n", rank, size);
  MPI_Info_create(&info);
  int rc = MPI_File_open( MPI_COMM_WORLD, file_name,
                          MPI_MODE_CREATE | MPI_MODE_RDWR,
                          info, &fh);
  buf[0] = 'A';

  // MPI_File_write_ordered(fh, buf, 1, MPI_INT, &status);
  MPI_File_write(fh, buf, 1, MPI_CHAR, &status);

  rc = MPI_File_close(&fh);
  MPI_Finalize();
  return 0;
}
```

- **Specify a buffer (here char array) of a certain size**
- **Specify values for the buffer, here just a simple A is used for each process (every process writes now the same content)**
- **MPI_File_write or related versions write the <u>text output</u> to the file**

MPI I/O & Parallel Filesystems

- Understanding and tuning parallel I/O is needed with 'big data'
  - Leverage aggregate communication and I/O bandwidth of client machines
- Support: Add additional software components/libraries layers
  - Coordination of file access
  - Mapping of application model to I/O model
  - Components and libraries get increasingly specialized / layer

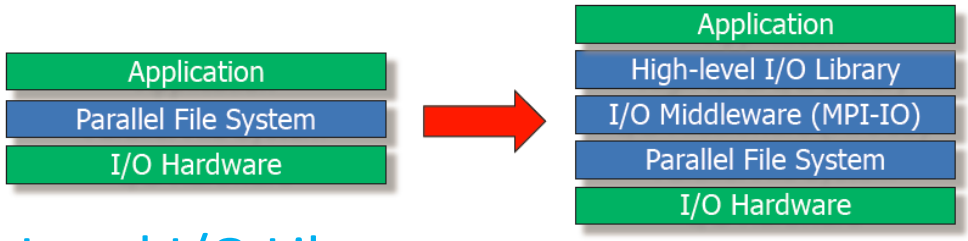


Parallel Filesystems are just one part out of three in the whole I/O process

[7] R. Thakur, PRACE Training, Parallel I/O and MPI I/O

I/O with Multiple Layers and Distinct Roles

- **Parallel I/O is supported by multiple software layers with distinct roles that are high-level I/O libraries, I/O middleware, and parallel file systems**

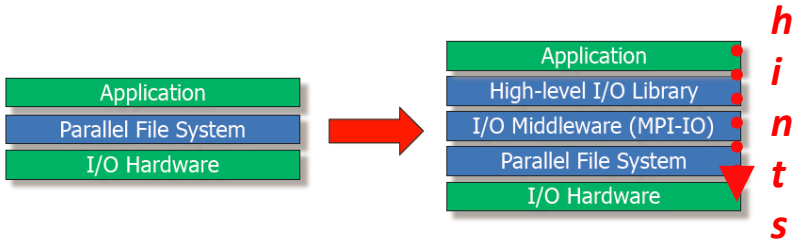


[7] R. Thakur, PRACE Training, Parallel I/O and MPI I/O

- High-Level I/O Library
  - Maps application abstractions to a structured portable file format
  - E.g. HDF-5, Parallel NetCDF
- I/O Middleware
  - E.g. MPI I/O
  - Deals with organizing access by many processes
- Parallel Filesystem
  - Maintains logical space and provides efficient access to data
  - E.g. GPFS, Lustre, PVFS

MPI I/O Hints

- Idea: pass along 'hints' about the parallel filessystem to MPI-IO



- Advantage:
  - Information about hardware (e.g. chunk sizes, etc.) increases performance

- **Using 'hints' MPI I/O can make better decisions about how to optimize the communication between MPI processes and the actual parallel file system to gain the best performance**

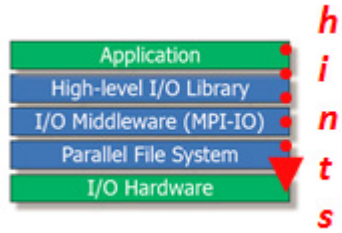
- Disadvantage (compared to idea of abstraction):
  - Application programmer needs to know parallel filesystem features
- MPI predefined hints
  - E.g. striping_unit, striping_factor, cb_buffer_size, cb_nodes
- Platform specific hints (tend to be more used)
  - E.g. start_iodevice, pfs_svr_buf, direct_read, direct_write

[11] 'Passing Hints'

Using MPI I/O Hints in MPI Programs

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

int main (int argc, char** argv) {
  int rank, size;
  MPI_File fh;
  MPI_Info info;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Info_create(&info);

  MPI_Info_set(info, "striping_factor", "4");

  MPI_Info_set(info, "striping_unit", "65536");

  MPI_File_open(MPI_COMM_WORLD, "testfile",
    MPI_MODE_CREATE | MPI_MODE_RDWR, info, &fh)

  MPI_Finalize();
  return 0;
}
```

- **A MPI_Info represents a list of key/value pairs used for providing information to MPI-I/O (cf. Lecture 8/8.1)**
- **No. of I/O devices to be used for file striping**
- **The stripping unit in bytes**

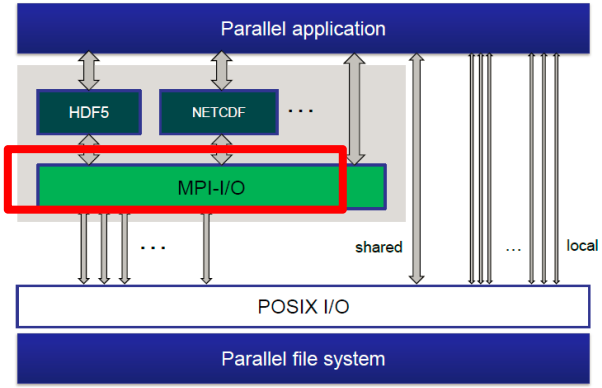


Higher level I/O libraries: HDF5 and Parallel NETCDF

- **Hierarchical Data Format (HDF) is designed to store & organize large amounts of numerical data**
- **Parallel Network Common Data Form (NETCDF) is designed to store & organize array-oriented data**
- **Portable data formats are needed to efficiently process data in heterogeneous HPC environments**

[12] HDF Group ***[13] Parallel NETCDF***

- Portable Operating System Interface for UNIX (POSIX) I/O
  - Family of standards to maintain OS compatibility, including I/O interfaces
  - E.g. read(), write(), open(), close(), …(very old interface, some say 'too old')
- 'Higher level I/O libraries' are abstraction from MPI-I/O
  - Integrated into a parallel application
  - Built on top of MPI I/O for portability
  - Offers machine-independent data access and data formats
  - Libraries often used in practice: HDF version 5 or Parallel NETCDF



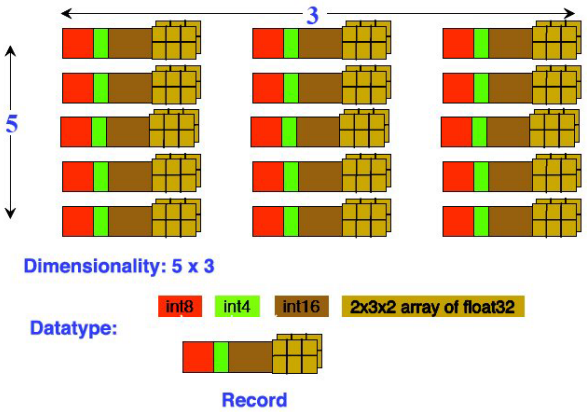
[14] SIONLib Web page

Hierarchical Data Format (HDF)

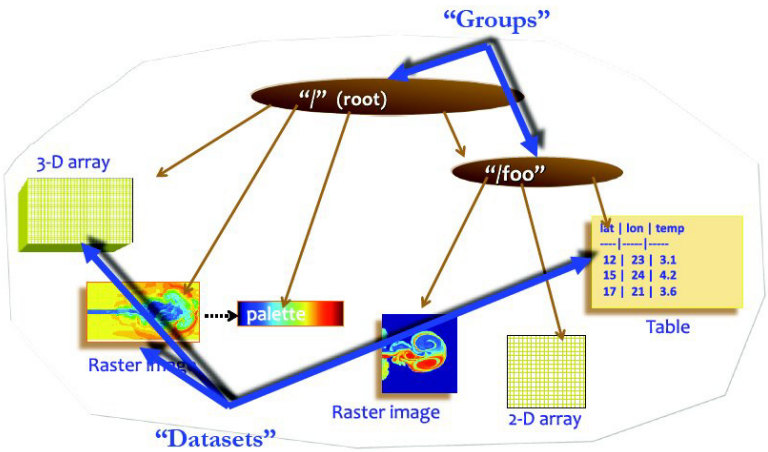
- **HDF is a technology suite that enables the work with extremely large and complex data collections**

[15] HDF@ I/O workshop

- Simple ‘compound type‘ example:
  - Array of data records with some descriptive information (5x3 dimension)
  - HDF5 data structure type with int(8); int(4); int(16); 2x3x2 array (float32)



‘HDF5 file is a container’ to organize data objects



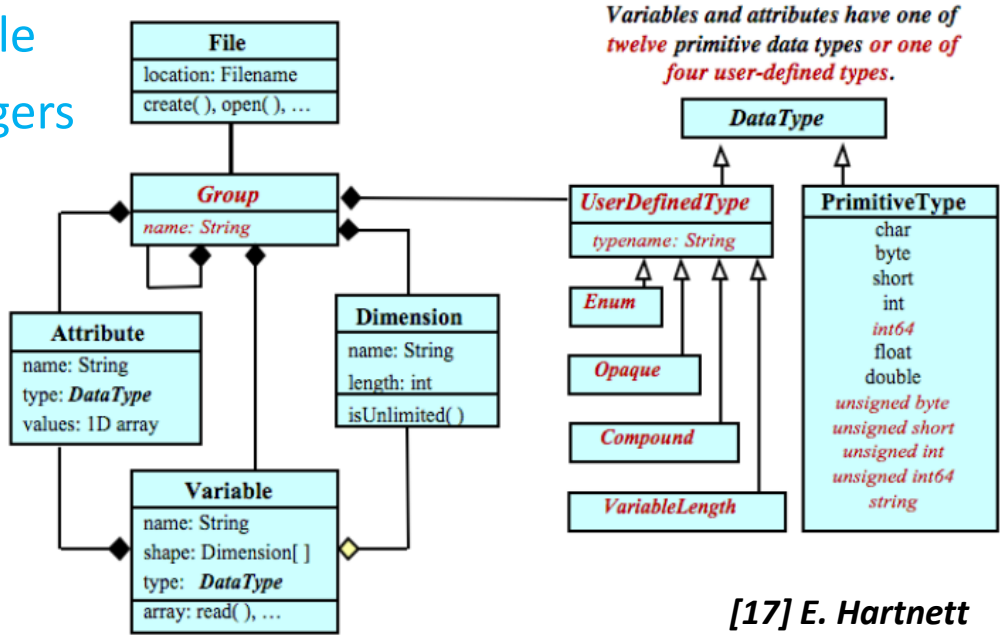
- ➢ **Lectures 12-17 will provide parallel application examples that take advantage of the HDF formats**

Parallel Network Common Data Form (NETCDF)

- **NETCDF is a portable and self-describing file format used for array-oriented data (e.g. vectors)**

[16] NetCDF @ I/O workshop

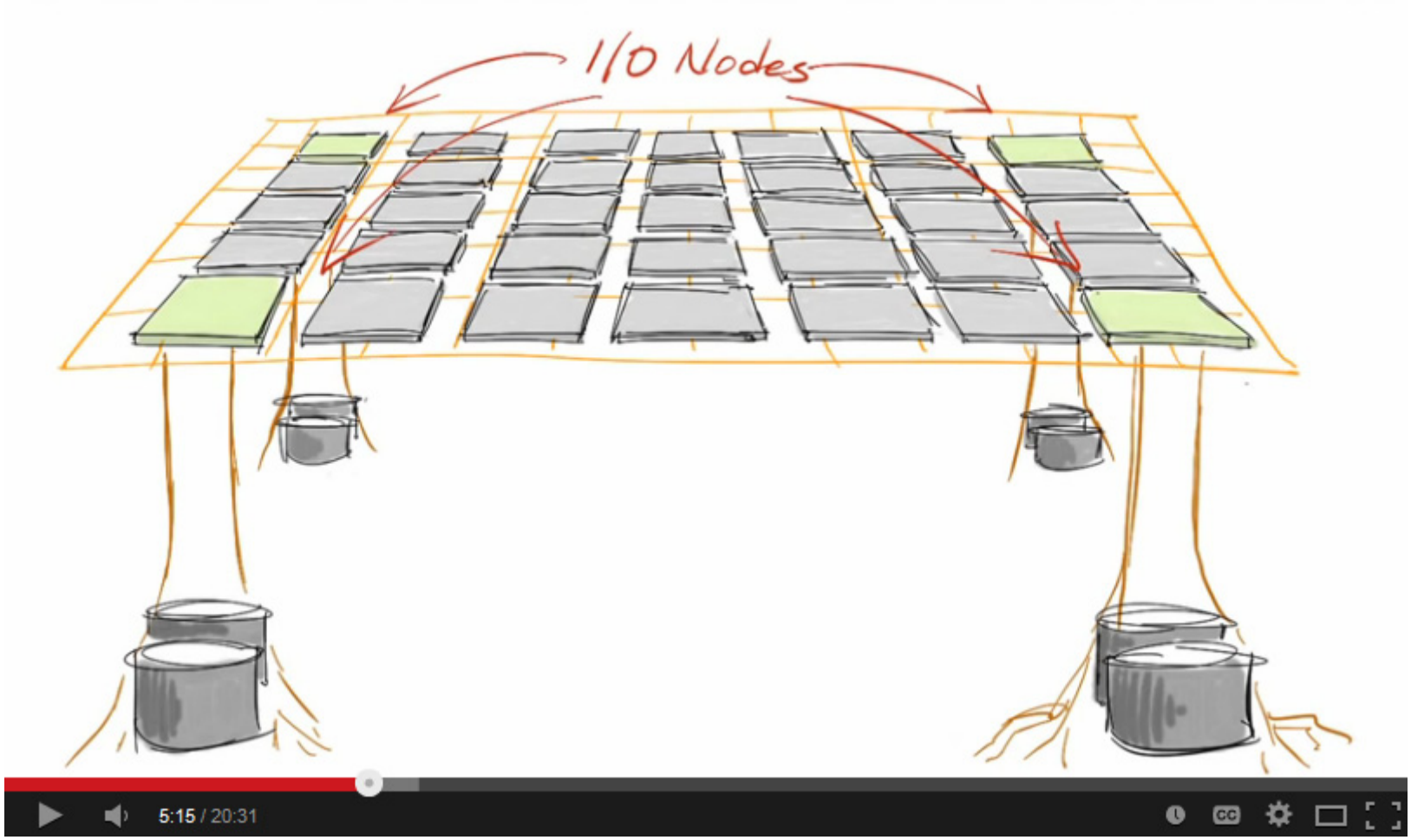
- Simple vector example:
  - Create a NetCDF data set with one one-dimensional variable
  - A local vector of 10000 integers should be allocated and initialized with the task number
  - Each task should write his vector to the NetCDF data set as a part of the global vector



[17] E. Hartnett

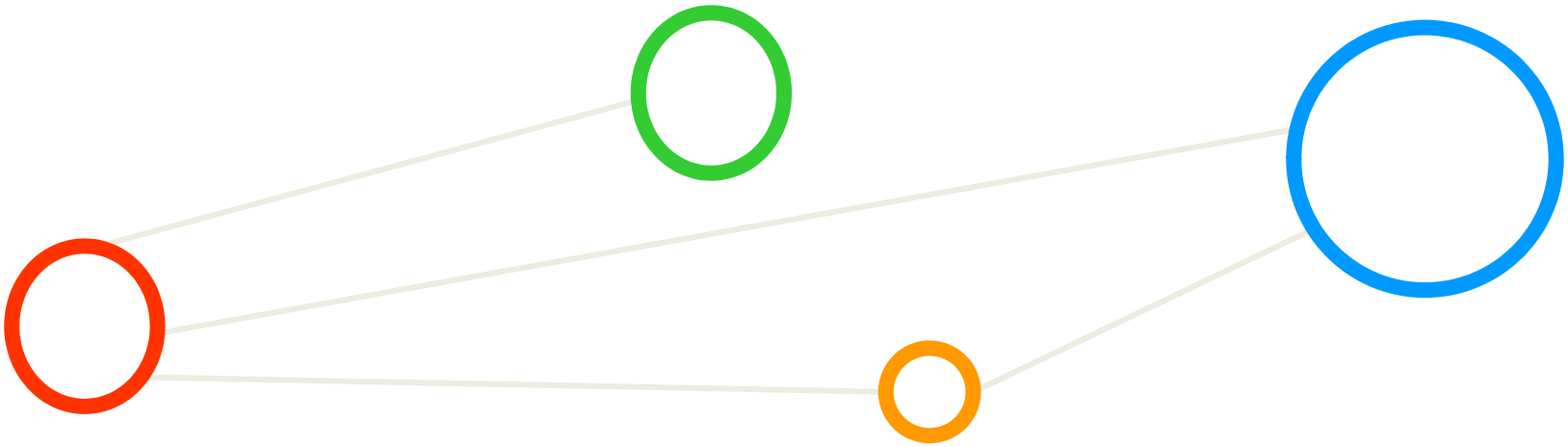
- ➢ **Lectures 12-17 will provide parallel application examples that take advantage of the NetCDF**

[Video] Parallel I/O with I/O Nodes



[18] YouTube Video, 'Simplifying HPC Architectures'

Lecture Bibliography



Lecture Bibliography (1)

- [1] Introduction to High Performance Computing for Scientists and Engineers, Georg Hager & Gerhard Wellein, Chapman & Hall/CRC Computational Science, ISBN 143981192X
- [2] LLNL MPI Tutorial,
  Online: https://computing.llnl.gov/tutorials/mpi/
- [3] Introduction to Groups and Communicators,
  Online: http://mpitutorial.com/tutorials/introduction-to-groups-and-communicators/
- [4] German Lecture 'Umfang von MPI 1.2 und MPI 2.0'
- [5] M. Geimer et al., 'SCALASCA performance properties: The metrics tour'
- [6] Wolfgang Frings, 'HPC I/O Best Practices at JSC', Online:
  http://www.fz-juelich.de/ias/jsc/DE/Leistungen/Dienstleistungen/Dokumentation/Praesentationen/folien-parallelio-2014_table.html?nn=469624
- [7] Rajeev Thakur, Parallel I/O and MPI-IO,
  Online: http://www.training.prace-ri.eu/uploads/tx_pracetmo/pio1.pdf
- [8] JUQUEEN,
  Online: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/JUQUEEN_node.html
- [9] YouTube Video, 'Mellanox 10 and 40 Gigabit Ethernet Switch Family',
  Online: http://www.youtube.com/watch?v=o9BLItx2vDg
- [10] Parallel I/O, YouTube Video,
  Online: http://www.youtube.com/watch?v=cXbEVsExU9c
- [11] Cornell Virtual Workshop, 'Passing Hints',
  Online: http://www.cac.cornell.edu/Ranger/MPIAdvTopics/passinghints.aspx
- [12] HDF Group,
  Online: http://www.hdfgroup.org/

Lecture Bibliography (2)

- [13] Parallel NETCDF,
  Online: http://trac.mcs.anl.gov/projects/parallel-netcdf
- [14] SIONLib Web page,
  Online: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/SIONlib/_node.html
- [15] Michael Stephan,'Portable Parallel IO - 'Handling large datasets in heterogeneous parallel environments',
  Online:
  http://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/parallelio-2014/parallel-io-hdf5.pdf?__blob=publicationFile
- [16] Wolfgang Frings, Florian Janetzko, Michael Stephan, 'Portable Parallel I/O - Parallel netCDF',
  Parallel I/O Workshop, Juelich Supercomputing Centre, 2014, Online:
  http://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/parallelio-2014/parallel-io-netcdf.pdf?__blob=publicationFile
- [17] E. Hartnett, 2010-09: NetCDF and HDF5 - HDF5 Workshop 2010
- [18] Big Ideas: Simplifying High Performance Computing Architectures,
  Online: https://www.youtube.com/watch?v=ISS_OGVamBk


services
measurement
within
fusion
new
funding
Policy-based
concepts
device
Analysis
forms
cross-disciplinary
resources
Centre
climate
Computer
expertise
year
Computational
E.g
Computing
disciplines
results
Approaches
construction
Fusion
Enable
like
Test
Cross-Disciplinary
EU
methods
High
big
nodes
PB
community
Environment
Modelling
different
Key
scientific
important
Science
centers
storage
basis
Infrastructure
files
etc
DKRZ
Use
simulations
ENES
technologies
model
formats
brain
increasing
via
EUDAT
physical
IPCC
Big
Project
Data
processing
Common
TB
SMAQ
international
engineering
References
Methods
computing
Using
European
MapReduce
create
databases
GB
Supercomputing
Work
images
codes
analysis
DLCL
stored
Resources
analyze
Large
Summary
Scientific
often
use
Level
JSC
computational
Research
Service
Matters
Concrete
external
levels
performance
research
e.g
science
HPC
Juelich
data-intensive
datasets
acquisition
system
Landscape
compute
System
Climate
modelling
AR5
Hardware
access
hundreds
Nutshell
Services
network
management
DLCLs
Understanding
structures
Technologies
Earth
Structure
cores
large-scale
distributed
current
Germany
directory
project
General
Simulation
Provide
NASA
Energy
systems
day
Health
Processing