Contents

1 HPC

- A supercomputer is a computer at the frontline of contemporary processing capacity – particulary speed of calculation.
- We speak of parallel programming whenever a number of ‚compute elements' (e.g. cores) solve a problem in a cooperative way.
- The LINPACK benchmark solves a desnse system of linear equations of unspecific size.
- The top 10 systems in the top500 list are dominated by the companies IBM and CRAY today.
- Shared-memory parallelization with OpenMP.
- Distributed-memory parallel programming with MPI.
- A shared-memory parallel computer is a system in which a number of CPUs work on a common, shared physical address space.
- UMA system use ‚flat memory model': Latencies and bandwidth are the same for all processors and all memory locations.
    - Also called Symmetric Multiprocessing (SMP).
- ccNUMA systems share logically memory that is physically distributed (similar like distributed-memory systems).
    - Network logic makes the aggregated memory appear as one single address space.
- Shared-memory programming enables immediate access to all data from all processors without explicit communication.
    - OpenMP is domimant shared-memory programming standard today.
- A distributed-memory parallel computer establishes a ‚system view' where no process can access another process's memory directly.
- Distributed-memory programming enables explicit message passing as communication between processors.
    - MPI is domimant distributed-memory programming standard today.
- A hierchial hybrid parallel computer is neither a purely shared-memory nor a purely distributed-memory system but a mixture of both.
- Large-scale ‚hybrid' parallel computers have shared-memory building blocks interconnected with a fast network today.
- Hybrid systems programming uses MPI as explicit internode communication and OpenMP for parallelization within the node.
- Increasing number of ‚new' emerging system architectures.
    - Often in state of flux/vendor-specific, quickly outdated.
- Parallel applications.
    - Parallel software programming according to numerical models and known physical laws.
    - Intensive re-use of proven mathematical/physical libraries and various compilers.
- Results today only possible due to extraordinary performance of Accelerators – Experiments – Grid computing.
- HPC systems typically provide a software environment that support the processing of parallel applications.
- Scheduling is the method by which user processes are given access to processor time (shared).
- HPC faced a significant change in practice with respect to performance increase after years.

  - Getting more speed for free by waiting for more CPU generations does not work any more.
  - Multicore processors emerge that require to use those multiple resource efficiently in parallel.
- Reducing clock frequency enables more than one CPU core on the same die (with the same power), better than increasing clock frequency of a single core and thus increasing heat and requiring more cooling.
  - Multicores a solution for this ‚power-performance limitation‘.
- Today multicore has been adapted to all major processor manufacturers (e.g. Intel, AMD, ..).
- Multithreading is built into many current processor designs (retain register/control per thread).
  - Threading capabilities use the architectural state of the CPU core that is present multiple times.
  - Known examples of multithreading are ‚hyperthreading‘ or ‚simultanous multithreading‘.
- The DRAM gap is the large discrepancy between main memory and cache bandwidths.

2 Parallelization Fundamentals

- Moore's Laws says that the number of transistors on integrated circuits doubles approximately every two years (exponentional growth, figure logarithmic scale).
- A single core is too slow to perform the required task(s) in a certain constrained amount of time.
- The available memory on a single system is not sufficient to tackle a problem in a required granularity or precision.
- In a Single Program Multiple Data (SPMD) paradigm each processor executes the same ‚code' but with different data.
- In the Multiple Program Multiple Data (MPMD) paradigm each processor executes ‚different' code with different data.
- Data Parallelism: Work distribution; Assign N parts of the grid to N processors.
  - In parallel computing a Grid distribution can be related to solving variables in linear equations (or find the best estimates of values).
- Scalability is the ability of a system, network, or process to handle a growing amount of work in a capable manner or its ability to be enlarged to accomodate that growth.
- Load imbalance (not all workers might execute their tasks in the same amount of time) hampers performance, because some resources are underutilized.
- Parellization with Serial Elements
  - Amount of work/overall problem size: s (serial part) + p (parallel part) = 1
- Scalability metrics quantify how well a task can be parallized.
- Two major quantities in HPC are named as ‚Strong Scaling' and ‚Weak Scaling'.
- Single worker serial runtime for a fixed problem size:
  - $T_f^S = s + p$
- N parallel workers runtime for a fixed problem size:
  - $T_f^P = s + \frac{p}{N}$
  - Results in Strong Scaling.
- Strong Scaling: How the time to solution varies with the number of processors for a fixed total problem size.
- Serial runtime for a scaled (variably-sized) problem (some power of N, α positive):
  - $T_v^S = S + P * N^{\alpha}$
- Parallel runtime for a scaled (variably-sized) problem:
  - $T_v^P = S + P * N^{\alpha - 1}$
  - Results in Weak Scaling.
- Weak Scaling: How the time to solution varies with the number of processors for a fixed problem size/processor.
- Serial performance for fixed problem with $T_f^S = s + p$:
  - $P_f^S = \frac{s+p}{T_f^S} = 1$
- Parallel performance for fixed problem with $T_f^P = s + \frac{p}{N}$
  - $P_f^P = \frac{s+p}{T_f^P(N)} = \frac{1}{s + \frac{1-s}{N}}$
- Application Speedup (Amdahl's law)
  - Scalability is dependend from the serial application parts.

- $S_f = \frac{P_f^P}{P_f^S} = \frac{1}{S+\frac{1-s}{N}}$, wherea $\frac{1-s}{N}$ reaches 0 as N reaches inf.
  - 1-s is the ‚parallizable part' of the problem.
  - When unlimited workers in place we have N -> inf.
  - Amdahl's law limits application speedup thus to $\frac{1}{s}$.
    - It says that scaling of massively parallel applications is hindered by the domination of it's serial parts.

3 HPC - MPI

3.1 Parallel Programming - MPI

- A distributed-memory parallel computer establishes a ‚system view' where no process can access another process's memory directly.
- Distributed-memory programming enables explicit message passing as communication between processors.
- MPI is domimant distributed-memory programming standard today.
- ‚Computing nodes' are independant computing processors (that may also have N cores each) and that are all part of one big parallel computer.
- Each processor has its own own data in its memory that can not ben seen/accessed by other processors.
- Broadcast (one-to-many) distributes the <u>same data</u> to many or even all other processors.
- Scatter (one-to-many) distributes <u>different data</u> to many or even all other processors.
- Gather (many-to-one) <u>collects data</u> from many or even all other processors or one specific.
- Recude (many-to-one) <u>combines collection with computation</u> based on data from many or even all other processors.
    - Usage of reduce includes finding a global min, global max, sum, or product of the different data located at different processors.
- MPI is not designed to handle network communcation.
    - Establishing/closing connections again and again not good here -> slow performance.
    - No security beyound firewall, no message encryption directly available, etc.
- MPI is an open standard that significantly supports the portability of parallel applications.
    - Portability can be limited to MPI versions and library versions.

SPMD: Single Processor, Multiple Data.

General:

- int main(int argc, char** argv)
    - The main() function is automatically started when launching a C program.
- #include <mpi.h> required to access the MPI library.
- Using communicators wisely in collective functions can reduce the number of affected processors.
- Point-to-point communication takes place among exactly one sender and exactly one receiver.
    - Both ends are identifieid uniquely by their ranks.
- MPI_Send() performs a blocking send.
    - Block until message is received by the destination point.
- MPI_Recv() performs a blocking receive for a message (until arrival).

3.2 Practical Lecture

Basic commands

- Maui commands: showq, checkjob [job Id], checknode.
- Torque commands: qsub, qstat, qdel.

MPI:

- All MPI (Message Passing Interface) programs must begin with a MPI_Init(&argc, &argv);
- The MPI_Comm_size() function determines the overall number of n processes in the parallel program: stores it in a variable size.
  - MPI_Comm_size(MPI_COMM_WORLD, &size);
- The MPI_Comm_rank() function determines the unique identifier for each processor: stores it in a variable rank wih values (0 ... n-1)
  - MPI_Comm_rank(MPI_COMM_WORLD, &rank);
- MPI_COMM_WORLD communicator constant denotes the ‚region of communication', here all processes.
- All MPI (Message Passing Interface) programs must end with a MPI_Finalize();
- Compiling a MPI program: mpicc program.c –o program.exe
- Check if master node by checking if rank == 0
  - Determine dest and source to the opposite processor, use rc = MPI_Send to send a message and rc = MPI_Recv to indicate that you're open for message receiving.
- rc = MPI_Get_count(&Stat, MPI_CHAR, &count) counts the number of received elements after pingponging messages.
- Example program (pingpong):

```
#include <mpi.h>
#include <stdio.h>
int main(argc,argv)
int argc; char *argv[]; {
int numtasks, rank, dest, source, rc, count, tag=1; char inmsg, outmsg='x';
MPI_Status Stat;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank == 0) {
  dest = 1; source = 1;
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  }
else if (rank == 1) {
  dest = 0; source = 0;
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  }
rc = MPI_Get_count(&Stat, MPI_CHAR, &count);
printf("Task %d: Received %d char(s) from task %d with tag %d \n", rank, count, Stat.MPI_SOURCE, Stat.MPI_
MPI_Finalize();
}
```

- MPI JobScript:

```
#!/bin/sh
#PBS -N TestJob
#PBS -lnodes=4
#PBS -M morris@hi.is
#PBS -m abe
#PBS -p 0
mpirun ./test.exe
```

4 HPC - OpenMP

4.1 Parallel Programming - OpenMP

- A shared-memory parallel computer is a system in which a number of CPUs work on a common, shared physical address space.
  - UMA (Unified Memory Access).
  - ccNUMA (Cache-coherent Nonuniform Memory Access).
    - Two memories connected with a coherent link to work as one.
- Shared-memory programming enables immediate access to all data from all processors without explicit communication.
- OpenMP is dominant shared-memory programming standard today.
- Threads are lightweight processes that work with data in memory.
- OpenMP Program:
  - fork()
    - initiated by master thread (exists always) creates team of threads.
  - Team of threads currently work on shared-memory data <u>actively</u> in parallel regions.
  - join() initiates the ‚shutdown‘ of the parallel region and terminates team of threads.
  - Team of threads maybe also put to sleep until next parallel region begins.
  - Number of threads can be different in each parallel region.
- OpenMP is an opan standard that significantly supports the portability of parallel shared-memory applications
  - But different vendors might implement it differently.
- OpenMP programs should always be written in a way that it does not assume a specific number of threads -> scalable programs.
- Requires #include <omp.h> to use the OpenMP library.
- Use something like Reduction: `#pragma omp parallel for reduction(+:sum)`
  - Reduce the sum by adding it to the global sum.

4.2 Practical Lecture

- Int nthreads, tid;
  #pragma omp parallel private(tid)
  - Shared variable nthreads, local variable tid.
    tid = omp_get_thread_num();
    - Get the current thread id.

    If(tid == 0)
    - If the master thread.

    nthreads = omp_get_num_threads();
      - Get the total number of threads in the parallel region.
- Compiling an OpenMP program: gcc program.c –fopenmp –o program.exe
  - OpenMP is a part of gcc, but required the –fopenmp parameter.
- #PBS –lnodes=1:ppn=4
  - Change the number of threads as a part of the job script.
- Export OMP_NUM_THREADS=4
  - Modify the global constant to be used on the machine.
- #pragma omp parallel private(tid)
  #pragma omp for private(n)
    - Already in parallel so statement omp for is enough when creating new parallel region.
- Example program (helloloop):

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

int main (argc, argv)
{

  int nthreads, tid;
  int n;

  #pragma omp parallel private (tid)
  {
     tid = omp_get_thread_num();
     printf("Hello World from thread = %d\n", tid);

    if (tid == 0)
    {
      nthreads = omp_get_num_threads();
      printf("Number of threads in parallel region = %d\n", nthreads);
    }

   #pragma omp for private(n)
   for (n=0; n<4; n++ )
   {
      printf("Thread No %d is working on iteration %d\n",tid, n);
   }

  }

  return 0;
}
```

- OpenMP jobscript:

```
#!/bin/sh
#PBS -N TestJob
#PBS -lnodes=1:ppn=4
#PBS -M morris@hi.is
#PBS -m abe

export OMP_NUM_THREADS=4

cd /home/morris/2014-HPC-A

./helloworldomp.exe
```

5 Algorithms and Data Structures

- MPI is designed to provide portable and efficient message passing functionality, but the performance of a given code is NOT directly portable across platforms.
- Scatter distributes different data (x, y) to many or even all other processors.
- Gather collects data from many or even all other processors to to one specific processor.

```
#pragma omp parallel for private(i) shared(x, y)
```

- The Sentinal is a special string that starts an OpenMP compiler directive.
  - Directive is optimized to enable a parallel loop (i.e. parallel for) starting a parallel region.
- PRIVATE defines local variables for each thread.
  - Each thread works independently and thus needs space to ‚store' local results – here i as index.
- SHARED defines global variables that exist only one time.
  - Each thread works independently but SHARED variables can be written and read from all threads.

Matrix-Vector Multiplication in MPI

```
/* Scatter matrix B */
MPI_Scatter(B, NCOLS, MPI_FLOAT, Bpart, NCOLS, MPI_FLOAT, 0, MPI_COMM_WORLD);
/* Scatter matrix C*/
MPI_Scatter(C, 1, MPI_FLOAT, Cpart, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);
```

- Each processor has a column of matrix B (name as Bpart).
- Each processor has an element of column vector C (named Cpart).

```
for(j=0; j < NCOLS; j++)  /* Do the vector-scalar multiplication */
        Apart[j] = Cpart[0] * Bpart[j];
/* Reduce to matrix A */
MPI_Reduce(Apart, A, NCOLS, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);
```

- Each processor performs an independant vector-scalar multiplication (based on their Bpart and Cpart contents).
- Each processor has a part of the result vector A (named Apart) and is reduced on rank 0 as sum.

- Reduce combines collection with computation based on data from many or even all other processors.
- Usage of reduce includes finding a global minimum or maximum, sum, or product of the different data located at different processors.
- Fourier series -> Study of periodic phenomena.
  - Tool: Fast Fourier Transform in the West (FFTW).
    - Can use methods such as fftw_mpi_local_size_2d to find out which portion of a 2d array resides on each processor and this is used to know how much space is allocated, instead of allocating an entire 2d array on each process.

        - MPI_COMM_WORLD used to indicate which processes will participate in a transform.
- Some applications require data structures that are more sophisticated data types and formats.
- Derived MPI datatypes are constructed from existing other datatypes (e.g. basic data types).
    - Used to avoid repeated sends of varied basic types (i.e. slow, clumsy, and error prone).
    - Enable a suitable memory layout for complex data structures that consist of several different types.
- Hierachical 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.

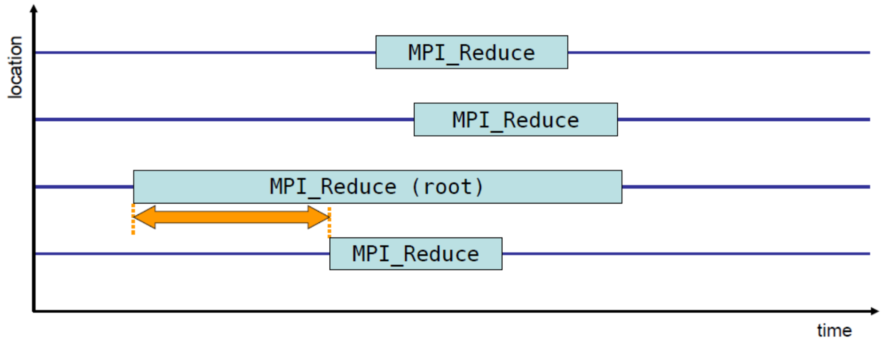
6 Debugging and profiling

- Debugging is a methodical process of finding and fixing flaws in software.
- A deadlock is a situation wherein two or more competing actions are each waiting for the other to finish, and thus ever is able to finish.
- A race condition can be a flaw in a process whereby the output and/or result of the process is unexpectedly and critically dependant on the sequence or timing of other events.
- Profiling: Understanding the program in terms of required execution time segments.
    - E.g. which of the different functions in the program takes the most time?
- Performance optimization is about tuning the program to enable a better performance and should be done when major flaws/bugs in the software are solved.
- Many parallel codes & libraries used in scientific computing don't implement the bug prevention approaches.
    - Software engineering principles.
    - Code readability.
    - Version control.
    - Well-defined code structures.
- Bug prevention by applying software engineering concepts and having good code readability.
- Bug prevention also means to check the HPC environments in which programs are executing.
- Printf debugging is not appropriate for the challenges of complex parallel program analysis.
- The ‚market of debugging tools' is dominated by strong commercial and expensive software.
- Wall-clock time is the actual time taken to complete a program and the sum of three different terms: CPU time, I/O time and the communication channel delay (E.g. message passing).
- The function MPI_Wtime() provides the elapsed wall-clock time of a parallel MPI program.
- The MPI profiling interface PMPI enables flexible writing of MPI functions wrapper routines.
    - Wrappers named as standard MPI_xyz routines internally call MPI standard routines via PMPI.
    - MPI offers an alias PMPI_xyz for each standard MPI routine, e.g. PMPI_Send() & MPI_Send().
- There is an overlap between tools used in parallel debugging, profiling & performance analysis.
    - Parallel performance analysis tools partly take advantage of profiling techniques & interface.
    - Tracing collects information about the program for post analysis – profiling aggregates statistics.
- Jotunn cluster.
    - 3 IBM eServer Blade Centers.
    - 14 blades / blade center.
    - 42 compute nodes cluster.
    - 550 GB Disk Space (FrontNode).

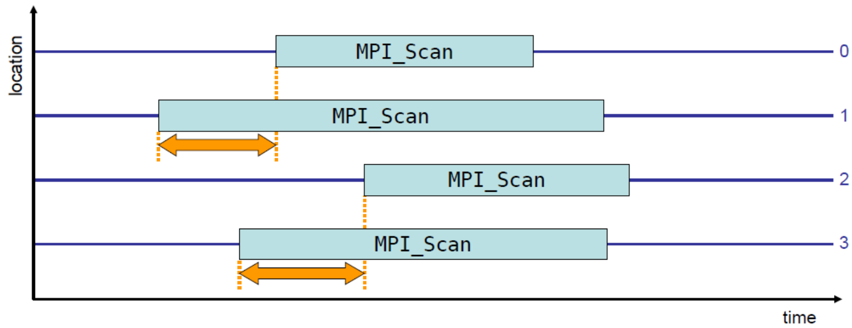
7 Performance Optimization

- Performance optimization requires scalable performance analysis tools and techniques.
- A scalable code is a code that keeps a good performance ratio / core by increasing cores.
- Getting a parallel code scalable is a ‚process cycle' that includes performance analysis & tuning.
- Large-scale applications parallel code needs not only good optimization techniques (also fault tolorance, etc.).
    - Fault tolorance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components.
- Metrics are required in order to have a clear understanding of what is measured in analysis steps.
- Generic metrics are CPU allocation time (execution and overhead), visits, and hardware counters.
- Metrics for parallel programs using MPI are based on time as part of the program execution time.
    - MPI metrics are Communication (collective/point-to-point), Syncronization, MPO I/O, and Init/Exit.
- Metrics for parallel programs using OpenMP are based on time as part of the program execution time.
    - OpenMP metrics are syncronization, fork (creating new threads), flush and Idle threads on CPUs.
- Tracing collects information about the program for post analysis – profiling aggregates statistics.
- Tracing technique:
    - Automatic/manual code instrumenter is used to enable runtime measurements and event tracing (use of MPI profiling interface).
    - Tracing requires a specific measurement library for runtime summary & event tracing (basic MPI techniques are limited).
    - Trace architecture enbales serial and parallel event trace analysis.
    - Use of analysis report examiner tools for interactive exploration of measured execution performance properties & metrics.
- Using the tracing technique has an impact on the runtime of scalability of codes (e.g. I/O & # files).
    - Replay and analysis of original parallel code requires parallel tools & techniques to be scalable too.
- The open trace format is a standardized data structure and API specification for tracing data.
- There is an overlap between tools used in parallel debugging, profiling & performance analysis.
    - Parallel performance analysis tools partly take advantage of profiling techniques & interfaces.
- A powerful analysis report examiner enables to determine (a) which performance problem is faced, (b) where in the program, and (c) which processes of the HPC machine are affected.

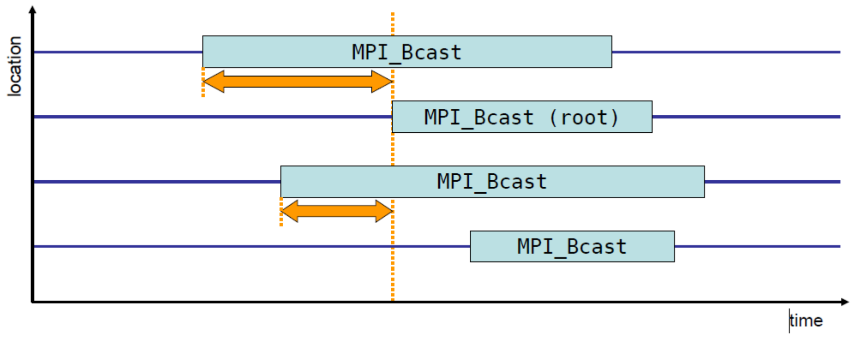
- Good usage of MPI collective operations can significantly reduce the overall runtime (i.e. walltime).
    - E.g. using one MPI_Allgather instead of a for loop with MPI_Bcast within (multiple MPI_Bcast()).
- Bad usage of MPI collective operations are one cause for many ‚wrong usage patterns & problems‘.
- MPI_Scan() computes the scan (partial reduction) of data on a collection of processes – prefix reduction on process i includes the data from process i (here: 4 ranks (see following)).
- Early reduce problem: Waiting time if the destination process (root) of a collective N-to-1 operation enters the operation earlier than its sending counterparts.



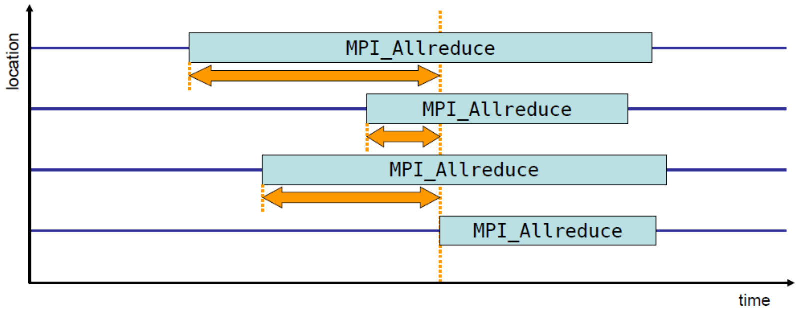
- Applies to: MPI_Reduce(), MPI_Gather(), MPI_Gatherv().
- Early scan problem: Waiting time if process n enters a prefix reduction operation earlier than its sending counterparts.



- Applies to: MPI_Scan()
- Late broadcast problem: Waiting time if the destination processes of a collective 1-to-N operation enter the operation earlier than the source process (root).

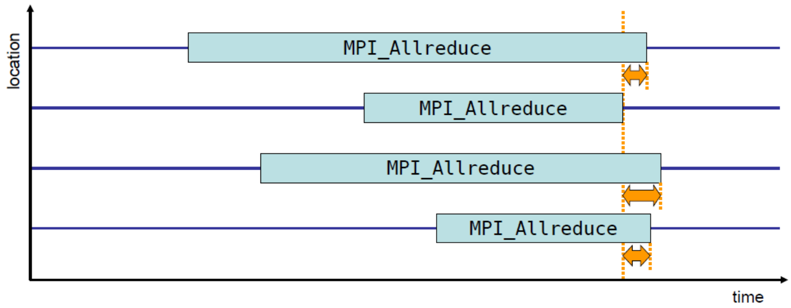


- Applies to: MPI_Bcast(), MPI_Scatter(), MPI_Scatterv().
- Wait at NxN problem: Time spent waiting in front of a syncronizing collective operation call until the last process reaches the operation.



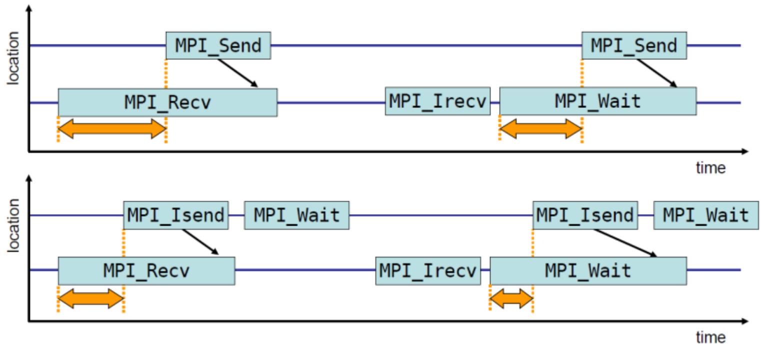
- Applies to: MPI_Allreduce(), MPI_Alltoall(), MPI_Alltoallv(), MPI_Allgather, MPI_allgatherv(), MPI_Reduce_scatter().

- NXN Completion Problem: Time spent in syncronizing collective operations after the first process has left the operation.



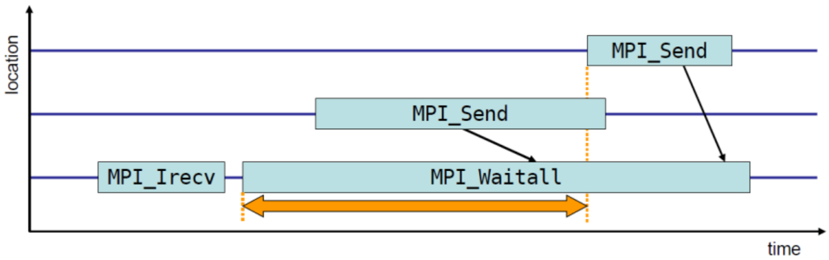
- Applies to: MPI_Allreduce(), MPI_Alltoall(), MPI_Alltoallv(), MPI_Allgather, MPI_allgatherv(), MPI_Reduce_scatter().

- Late sender problem: Waiting time caused by blocking receive operation posted earlier than the corresponding send operation.



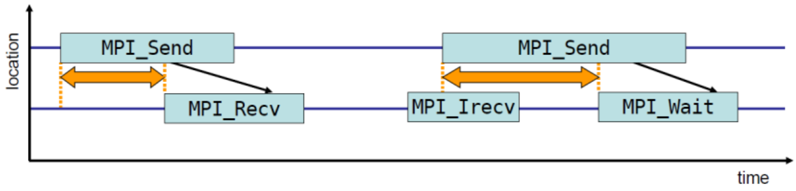
- Applies to blocking as well as non-blocking communication.
  - Blocking vs. Non-blocking: MPI_Send() blocks until data is received, MPI_Isend() continues.

- Late sender problem (2): While waiting for several messages, the maximum waiting time is accounted.

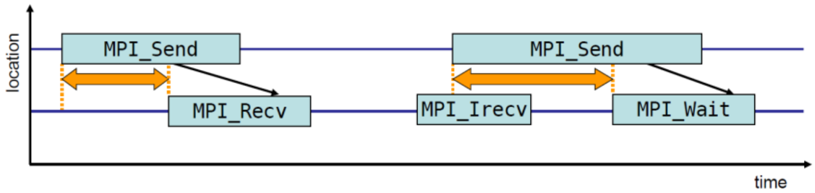


- Applies to: MPI_Waitall(), MPI_Waitsome().

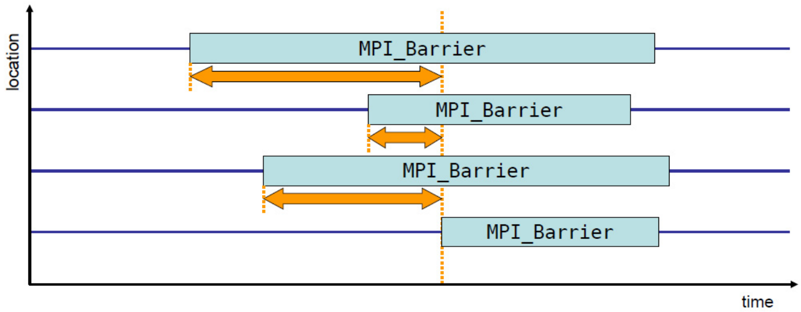
- Late sender problem (3): Refers to Late Sender situations which are caused by messages received in wrong order.



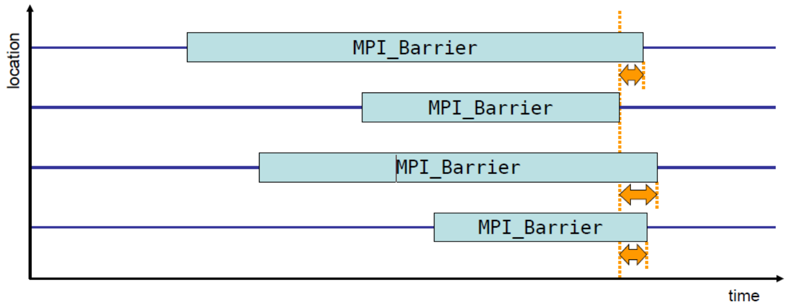
  - Two flavours: (a) Messages sent from some source location; (b) Messages sent from different source locations.
- Late receiver problem: Waiting time caused by a blocking send operation posted earlier than the corresponding receive operation.



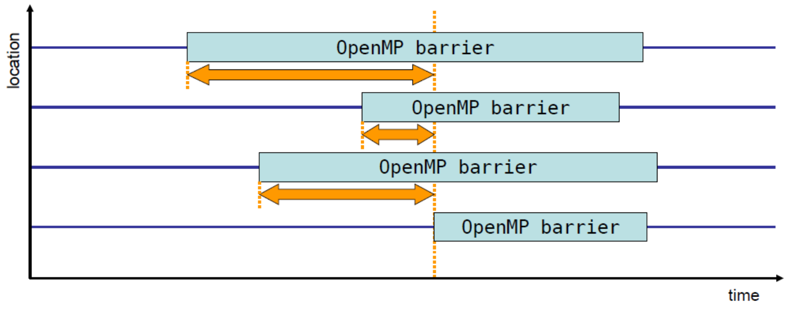
  - Calculated by receiver but waiting time attributed to sender.
  - Applies not to non-blocking sends.
- Wait at Barrier problem: Time spent waiting in front of a barrier call until the last process reaches the barrier operation.



  - Applies to: MPI_Barrier().
- Barrier completion problem: Time spent in barrier after the first process has left the operation.

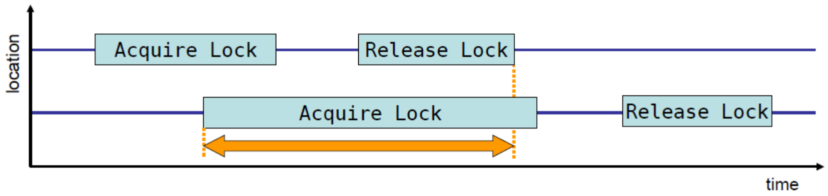


  - Applies to: MPI_Barrier()
- Wait at Barrier problem: Time spent waiting in front of a barrier call until the last process reaches the barrier operation.



- Applies to: Implicit/explicit barriers.

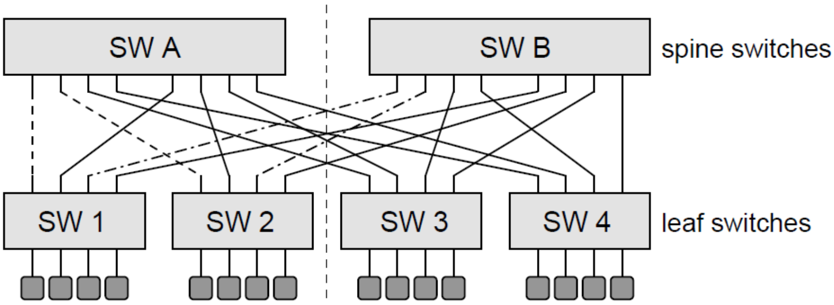
- Lock Completion (API & Critical Regions) problem: Time spent waiting for a lock that has been previously acquired by another thread.

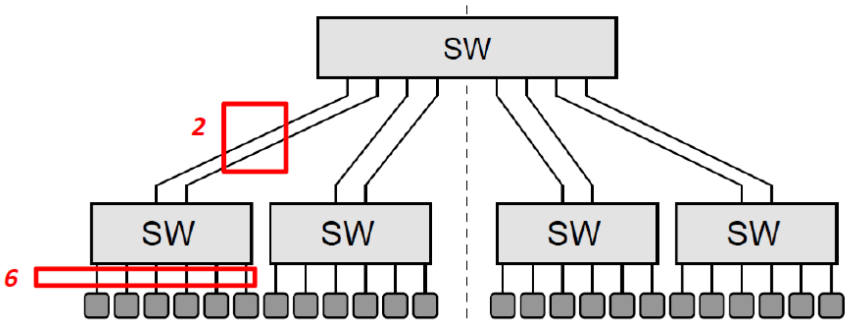


- Applies to: critical sections, OpenMP lock Application Programming Interface (API).

- Optimization in terms of software & hardware are important.

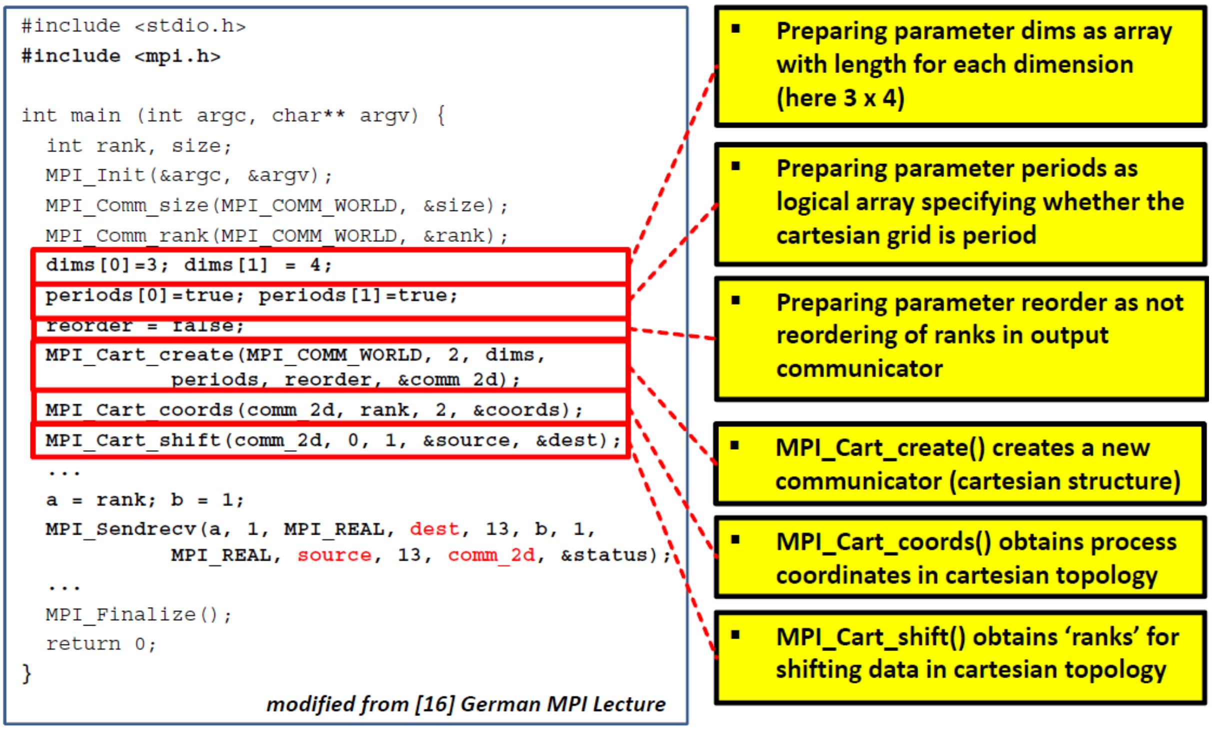
8 Further Parallel Techniques

- MPI is designed to provide portable and efficient message passing functionality, but the performance of a given code is NOT directly portable across platforms.
- Using communicators wisely in collective functions can reduce the number of affected processors.
- Communication overhead can have significant impact on application performance.
- Characteristics of interconnects of compute nodes/cpus affect parallel performance.
- Measuring point-to-point communication: T = $T_{L\ (latency)} + \frac{N\ (message\ of\ size\ N\ [bytes])}{B\ (Bandwidth\ [\frac{Mbytes}{sec}])}$
- $B_{EFF} = \frac{N}{T_L + \frac{N}{B}}$
    - The bandwidth B depends on the message size N.
- Direct measurement of latency would be possible with setting N = 0 bytes.
- Small message sizes: latency dominates the transfer time.
- Large message sizes: latency plays no significant role and bandwidth saturates.
- Think about workers processing data and interacting with each other -> switch matters!
- Advanced programming techniques need to take the hardware interconnection into account.
- Combining Network Building Blocks as FatTree:
    - 

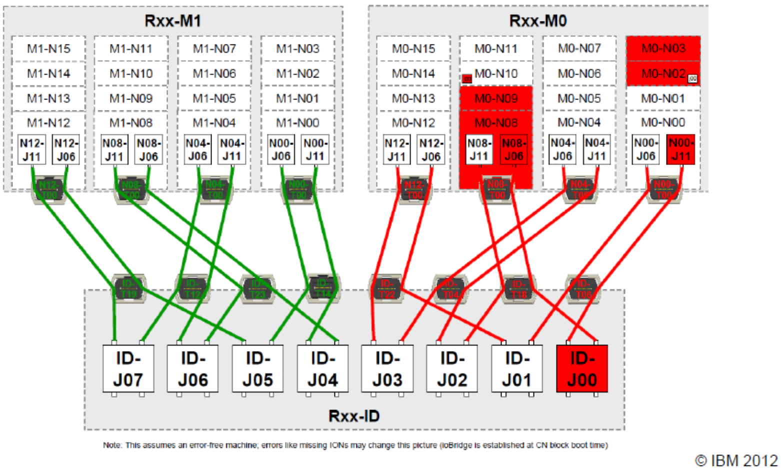
        - Here a group of workers processing data ‚enjoy' full non-blocking communication.
            - Location of the workers here is not very crucial to the application performance.
    - 

        - Here with a 1:3 bottleneck (when # CPUs high).
            - The location of the workers processing data is crucial for application performance here.
                - Common in very large systems -> safe costs (cable & switch hardware).
- Fat-Tree have limited scalability in very large systems (price vs. Performance).

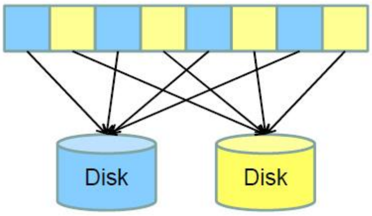
  - Bisection bandwidth with scaling in large systems often via mesh networks (e.g. 2d torus).
- Example using communicators & network topology:



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



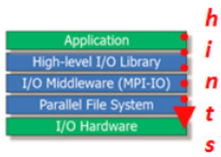
- Communication Optimization by Process Placement:
  - Optimal placement is an NP-hard-problem.
  - n! Possibilities to map n execution units to the same number of n processing elements.
  - Topology (is. *grannfræði*) aware task mapping for I/O patterns.
- Optimized task core mappings enable performance gains between 1-3% (heatmap example).
- Input/Output (I/O) stands for data transfer/migration from memory to disk (or vice versa).
- An I/O pattern reflects the way of how an application makes us of I/O (files, processes, etc.)
- 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 configurd size (e.g. typically less than 1MB each).
- Concurrent file 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.
- Striping refers to a technique where one file is split into fixed-sized blocks that are written to separate disks in order to faciliate parallel access.
  - 

- Parallel file systems use buffering to reduce the need for disk accesses (increased performance).
- Widely used parallel file systems are GPFS (commercial) and Lustre (open source).
  - General Parallel File System.
  - Lustre bought by Sun.
- 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.
- Parallel I/O is support by multiple software layers with distinct roles that are high-level I/O libraries, I/O middleware, and parallel file systems.
- Using ‘hints' (pass along ‘hints' about the parallel filesystem to MPI-IO) 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.
- 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:
    - Hierarchical Data Fromat (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 (is. *sundurleit*) HPC environments.
- HDF is a technology suite that enables the work with extremely large and complex data collections.
    - ‚HDF5 (HDF version 5) file is a container' to organize data objects.
- NETCDF is a portable and self-describing file format used for array-oriented data (e.g. vectors).
- Serialization on the File System (FS) Block level for locking in ‚parallel tasks' (bottlenecks in POSIX I/O).
- Scalable I/O libraries such as SIONlib enables ‚logical partitioning of shared files': e.g. dedicated data chunks per ‚parallel tasks'.

9 – Hybrid Programming & Patterns

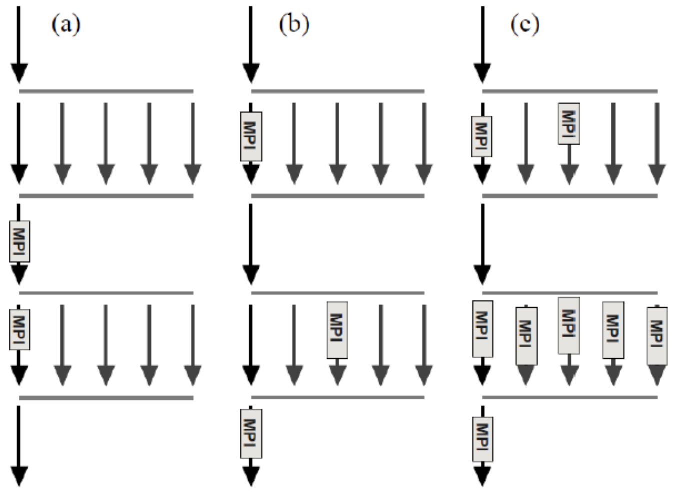
- A hierchial hybrid parallel computer is neither a purely shared-memory nor a purely distributed-memory type system but a mixture of both.
- Large-scale ‚hybrid' parallel computers have shared-memory building blocks interconnected with a fast network today.
- Avoiding the memory requirements of individual MPI processes that include memory space for data, text, heap and stack (needed for processing).
- Safe buffer space allocated for MPI communication for each individual MPI processes that consume valuable memory space (e.g. also for I/O buffers).
- Hybrid systems programming uses MPI as explicit internode communcation and OpenMP for parallelization within the node – but achieving a speed-up & scalability is not always the goal.
- Using hybrid systems programming reduces the memory requirement overhead from multiple processes – bears the potential to get access to more memory/process in applications.
- Programming hybric example:

'simplified demo code'

```
#include <stdio.h>
#include <mpi.h>
int main (int argc, char** argv) {
  int rank, size, n, info;
  double *x, *y, *buff;
  MPI_Init_thread(&argc, &argv, MPI_THREAD_FUNNELED, &info);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  ...
  chunk = n / size;
  ...
  MPI_Scatter(buff, chunk, MPI_Double, x,
         chunk, MPI_DOUBLE, 0, MPI_COMM_WORLD);
  MPI_Scatter(&buff[n], chunk, MPI_DOUBLE, y,
         chunk, MPI_DOUBLE, 0, MPI_COMM_WORLD);
  ...
  #pragma omp parallel for private(i, chunk) shared(x,y)
  doSomething(&chunk, &done, X, &paramA, y, &paramB);
  ...
  MPI_Gather(x, chunk, MPI_DOUBLE, buff, chunk,
         MPI_DOUBLE, 0 MPI_COMM_WORLD);
  MPI_Finalize();
  return 0;
}
```

- **Change of MPI_Init() to MPI_Init_thread() to prepare the MPI environment that threads will be used in program**
- **MPI_Init_thread() has a parameter 'required' that specifies requested level of thread support (e.g MPI_THREAD_FUNNELED)**
- **MPI_Init_thread() returns a parameter with the actural 'provided' level of support from MPI library**
- **Use of OpenMP directives in MPI code but stick to level of thread safety**

- Thread safety:
  - User specifies ‚gurantees' to the MPI library in initialization.



- **(a) MPI_THREAD_FUNNELED: Only the master thread will make calls to the MPI library; thread that calls MPI_Init_thread is master thread**
- **(b) MPI_THREAD_SERIALIZED: Only one thread at a time will make calls to the MPI library; every thread is able to call an MPI routine**
- **(c) MPI_THREAD_MULTIPLE: Any thread will make calls to the MPI library at any time; MPI library is responsible for thread safety (slow)**

- Combining MPI with OpenMP:
  - Exploiting an additional level of finer granularity (Any MPI process spawns n worker threads) with ‚multi-threading' can be sometimes the only way to increase parallelism beyond MPI limits (e.g. application logic constraints).
- Hybrid Vector mode example:

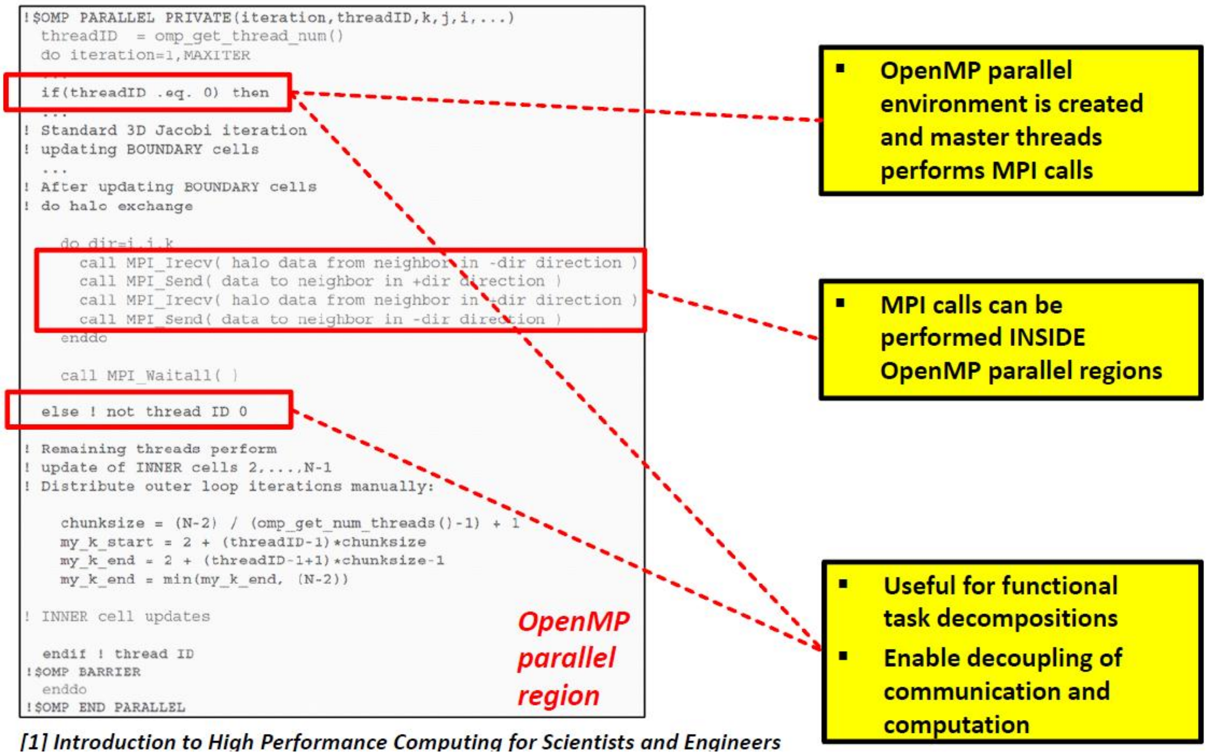
```
 do iteration=1,MAXITER
!$OMP PARALLEL DO PRIVATE(..)
   do k = 1,N
! Standard 3D Jacobi iteration here
! updating all cells
     ...
   enddo
!$OMP END PARALLEL DO

! halo exchange
   ...
   do dir=i,j,k

     call MPI_Irecv( halo data from neighbor in -dir direction )
     call MPI_Isend( data to neighbor in +dir direction )

     call MPI_Irecv( halo data from neighbor in +dir direction )
     call MPI_Isend( data to neighbor in -dir direction )
   enddo
   call MPI_Waitall( )
 enddo
```

OpenMP parallel region

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

- **OpenMP worksharing constructs are put in context of compute-intensive loops**
- **MPI calls are performed OUTSIDE OpenMP parallel regions**
- **Halo regions need to be copied frequently since they are needed for computation while a halo is a copy of remote data**
- **Using instead OpenMP can reduce the size of halo regions that need to be stored (cf. Jacobi example)**

- Hybrid Task mode example:

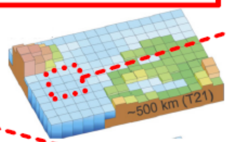


- Comparison of Vector mode and Task mode:
  - Vector Mode(recommended)
    - Vector mode implementation is straightforward and program and keeps clean code.
    - Programming hybrid like this means programming MPI/OpenMP parts independantly.
    - Applications benefit where the number of MPI processes are constraint by application logic.
  - Task Mode(only for experts and to get the most out of systems)
    - Task mode is the most flexible option for programming hybrid but also most difficult.
    - Programming hybrid like this means having MPI calls are part of the OpenMP parallel regions.
    - Convenient OpenMP worksharing parallelization directives not used to differentiate threads.
- Do hybrid programming only if pure MPI scalability is not satisfactorary (i.e. often infiniband on HPC).
  - Working hard on hybrid programming makes less sense, rather work on perfectly scaling MPI code.
  - Since multi-core systems are expected to grow, above statements needs to be reviewed every year.
- Cartesian communicators are useful methods to implement nearest neighbour communication patterns that are used in many applications in scientific computing and simulation sciences.

- Cartesian Communicator:

'simplified demo code'

```
#include <stdio.h>
#include <mpi.h>
#define SIZE 16
#define UP 0
#define DOWN 1
#define LEFT 2
#define RIGHT 3

int main (int argc, char** argv) {
  int numtasks, rank, source, dest, outbuf, i, tag=1;
  int inbuf[4]={MPI_PROC_NULL,MPI_PROC_NULL, MPI_PROC_NULL,
        MPI_PROC_NULL};
  int nbrs[4];
  int dims[2] = {4,4}, periods[2] = {0,0}, reorder=0;
  int coords[2];
  MPI_Comm cartcomm;

  MPI_Request reqs[8];
  MPI_Status stats[8];
   ...
  MPI_Init(&argc, &argv);
  ... // starting with MPI program...
}
```



modified from [8] MPI Tutorial

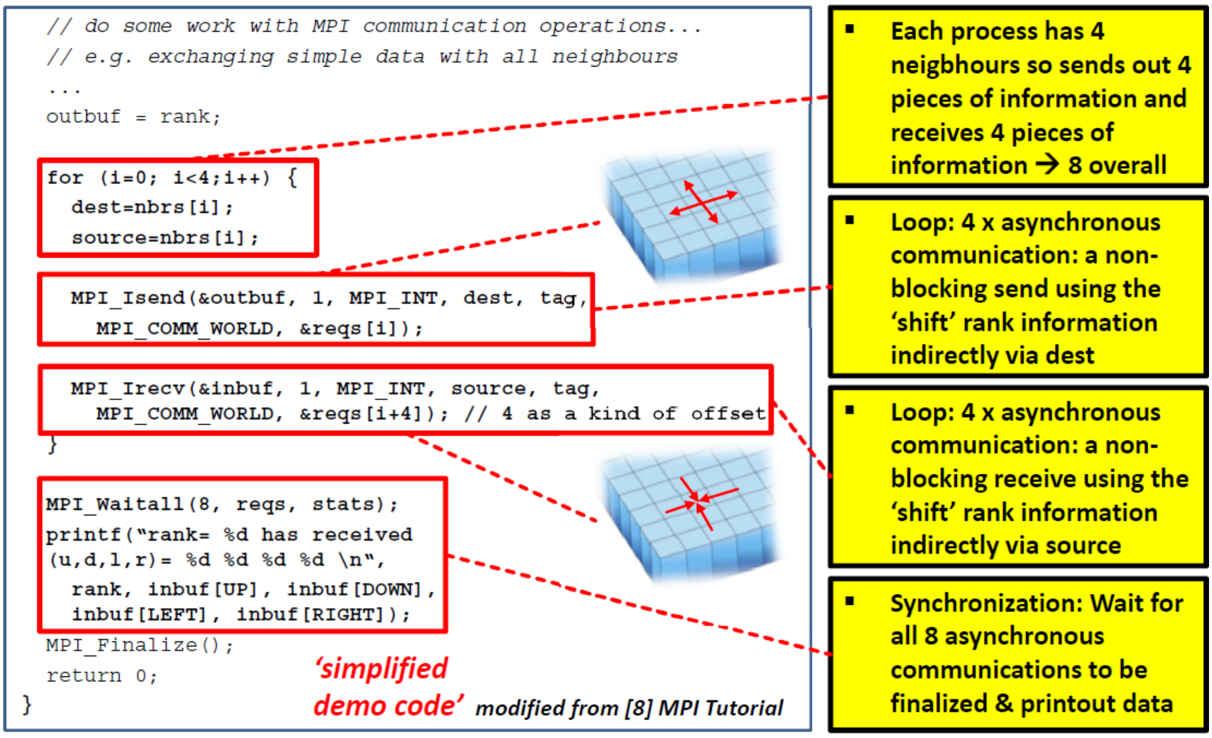
- 'constants for numbers': offer here better code readability, not a must
- Prepares variables to be used in asynchronous communication;
- MPI_PROC_NULL indicates a 'rank' for a so-called 'dummy process'
- Prepares variables related to our 2D problem,4x4 with 4 neighbours
- Prepares variables for creating a cartesian communicator later
- Prepares variables used for non-blocking MPI

```
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
  MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods,
            reorder, &cartcomm);
  MPI_Comm_rank(cartcomm, &rank);
  ...
  MPI_Cart_coords(cartcomm, rank, 2, coords);
  ...
  MPI_Cart_shift(cartcomm, 0, 1,
            &nbrs[UP], &nbrs[DOWN] );
  MPI_Cart_shift(cartcomm, 0, 1,
            &nbrs[LEFT], &nbrs[RIGHT] );

  printf("rank= %d coords= %d &d" having
    neighbours(u,d,l,r)=%d %d %d %d \n",
    rank, coords[0], coords[1],
    nbrs[UP], nbrs[DOWN], nbrs[LEFT], nbrs[RIGHT]);

  // do some work with MPI communication operations...
  ...
  MPI_Finalize();
  return 0;
}
```

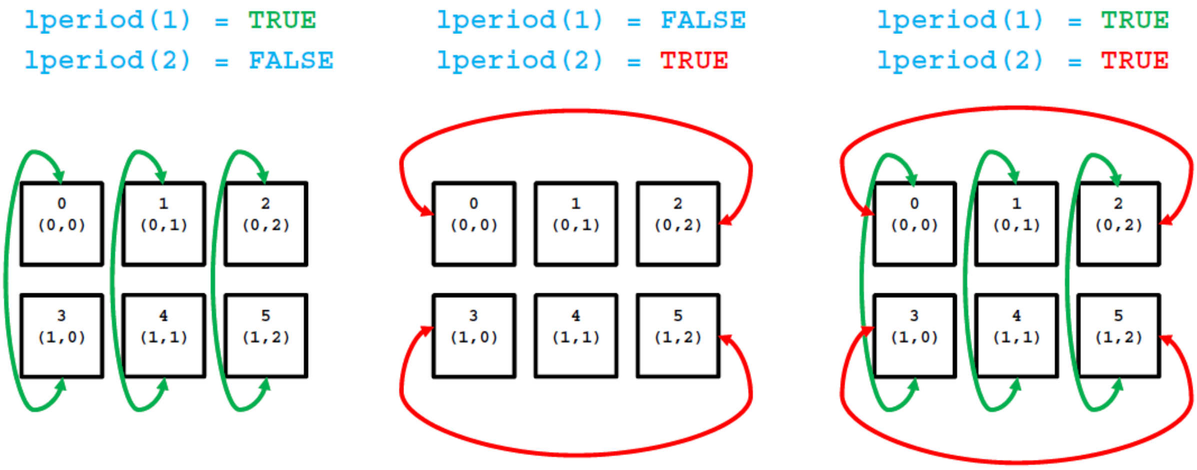
'simplified demo code'

modified from [8] MPI Tutorial

- Creates a cartesian coordinator based on our above initialized variables, here 2D → 4x4
- Obtains rank from each process, here from the cartesian communicator
- Obtains coordinate from each process from the cartesian communicator
- (just!) prepares a 'shift' to neighbours up and down as well as left and right according cartesian setup (obtain the rank of them)
- Prints out the neighbours with coresponding rank



- Cartesian Communicators – Periodicity:

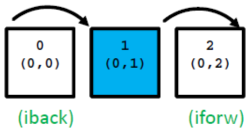


  - C uses the definition that false is ‚exact value of 0' and true ‚unqual 0' (e.g. 1)
  - The usefulness of the different levels of periodicity depends on the application logic of the corresponding scientific simulation.
  - Setting the periodic or non-periodic levels influences the shifts patterns.

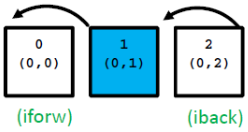
- Cartesian Communicators – Standard Shifts:
  `MPI_CART_SHIFT(comm, dir, disp, iback, iforw, ierr)`

```
int MPI_Cart_shift(
  MPI_Comm comm,
  int direction,
  int displ,
  int *source,
  int *dest
);
```

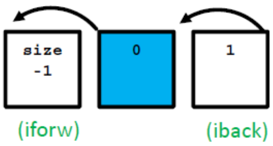
  - Positive Shift
    - `disp = +1`



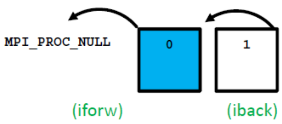
  - Negative Shift
    - `disp = -1`



    - Shifts prepares the communication with neighbours with send/receive operations along each different directions and obtain ranks to be used in send/receive operations.
    - Problematic Shifts:
      - Negative Shift (periodic)
        - `disp = -1`

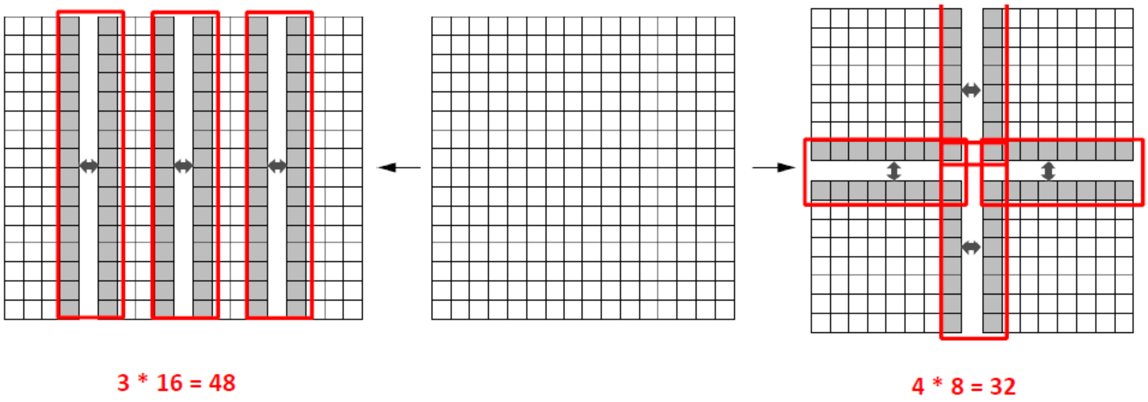


        - Size-1 indicates that the next shift is going to perform a ‚tournaround / period‘ given a periodic cartesian communicator setup.
      - Off-end Shift (non-periodic)
        - `disp = -1`



        - MPI_PROC_NULL ‚as dummy process‘ indicates here that the next shift is leaving the defined dimension of the cartesian communicator in a non-periodic setup.
- Stencil-based Iterative Methods.
  - Stencil-based iterative methods update array elements according to a fixed pattern called ‚stencil‘.
  - The key of stencil methods is its regular structure mostly implemented using arrays in codes.
  - Method is often used in computational science as part of scientific and engineering applications.
- The Jacobi iterative method is a stencil-based iterative method used in numerical linear algebra.
  - Algorithm for determining the solutions of diagonally dominant system of linear algebra.
  - The isotropic lattice term is derived from ‚isotropy‘ that stands for uniformity on all orientations.
- Halo regions are needed for local computations while a halo / ghost layer is a copy of remote data.

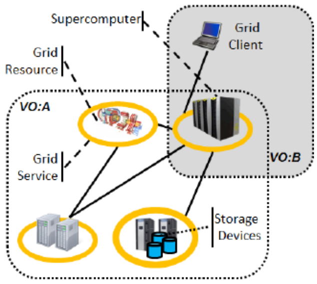
- Reducing the amount of halo regions with OpenMP in large-scale MPI applications can be useful.



  - The latter is better.

10 – Scalable Infastructures & GPGPUs

- A resource is a specific hardware or software system such as a parallel computer, a disk or tape storage, 3D display capabilities, or a (scientific) measurement instrument like a telescope.
  - Parallel computing infastructures enable the parallel use of such resources with many others.
- A High Performance Computing (HPC) – driven infastructure is based on computing resources that enable the efficient use of parallel computing techniques through specific support with dedicated hardware such as high performance cpu/core interconnections.
- A High Throughput Computing (HTC) – driven indastructure is based on commonly available computing resources such as commodity PCs and small clusters that enable the execution of ‚farming jobs' without providing a high performance interconnection between the cpu/cores.
- ‚e(enhanced)-Science is about collaboration in key areas of Science and the next generation infastructure that will enable it.'
- A virtual organization (VO) enables a secure sharing of a wide variety of geographically distributed resources across different organizational boundaries (e.g. time limited, dynamic add/remove).
- Grid MiddleWare is a technology that presents the Grid as a single system by hiding administrative and geographic boundaries.
  - Grid Middleware provides seamless, secure, and intuitive access by hiding complexities in such a way taht its infastructure appears transparently to its users.
- The ‚scalability of a Grid' refers to a use of more than one system if needed while the ‚scalability of a HPC application in a system' refers to an increased number of cores keeping a reasonable speed-up.
- Different types of Grid middleware implementations exist that vary in their architectures, approaches, and different levels of open Grid standard adoptions.
- Grid middleware implementations are often driven by the needs of key scientific communities, but used in many scientific domains like high energy physics, life sciences, neuroscience, etc.
- Grid as Service Oriented Architecture (SOA)
  - Grid services are implemented as state-ful Web services (the ‚rings')

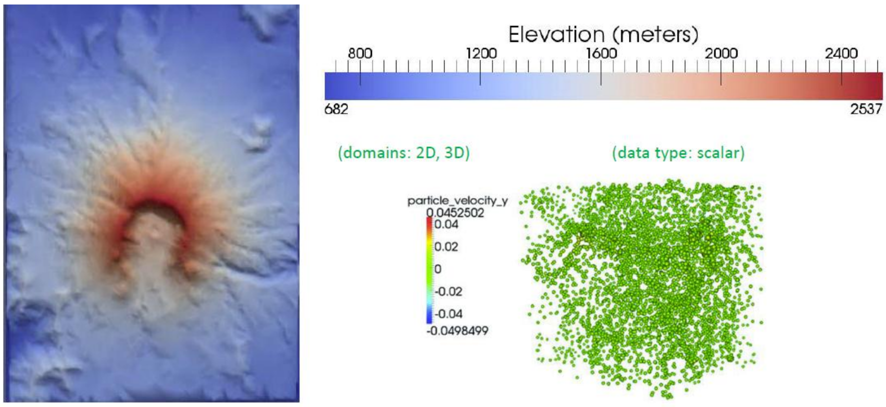


  - (State-ful) Grid/Web services rely on HTTP(S) protocol and XML message exchanges.
  - Data transfers for large volumes of data require more specialized protocols (e.g. gridFTP).
- ‚Results today only possible due to extraordinary performance of Accelerators – Experiments – Grid computing'.

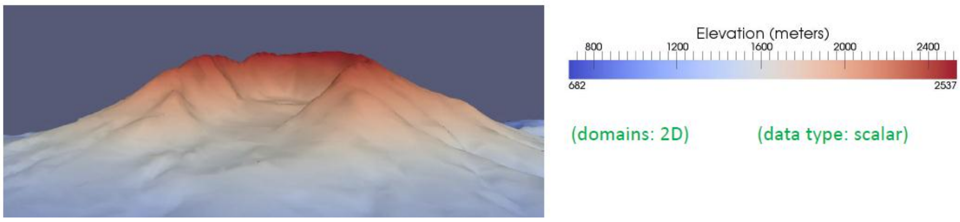
- Grid computing can be seen as the major percursor of inustry-driven Cloud computing : inspired many technology approaches used in clouds & virtualization (e.g. in their backend infastructures).
- The advancement of virtualization technologies and hypervisors is a key enabler for Cloud computing and cloud storage infastructures.
- Cloud computing infastructures can offer services on three different levels:
    - Infastructure as a Service (IaaS).
        - Provides specific ‚ready-to-run applications‘.
    - Platform as a Service (PaaS).
        - Virtual images ready to deploy your software.
    - Software as a Service (SaaS).
        - Provides ‚bare metal‘ infastructure/virtual images.
- Cloud computing infastructures have operational models that can be differentiated according public clouds, private clouds and hybrid clouds.
- A collaborative data infastructure combines the massive amount of unique resources of large multi-disciplinary data and computing centers with strong domain-specific centers (e.g. climate).
- GPGPUs
    - General-Purpose Computing On Graphics Processing Units (GPGPUs).
    - GPUs have been traditionally used to perform computing for computer graphics (e.g. games).
    - GPGPUs use GPUs to perform application computation instead or in addition to normal CPUs.
- GPUs have a parallel throughput architecture thatemphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly.
- In the context of GPUs, the Kernel is a function that runs on a GPU device.
- Rendering pipeline designed for massively parallelism and independant operations.
- General processing in science and engineering partly rely on independant operations & data.
- GPGPUs are very restrictive in operations and programming, but ideal for data parallel tasks.
- GPGPUs are very effective for a set of records that require similar computation names as streams.
- OpenCL is the open general-purpose GPU programming model approach that is vendor neutral.
- CUDA is the dominant propriety general-purpose GPU programming model that is very vendor-specific.

11 – Scientific Visualization and Steering

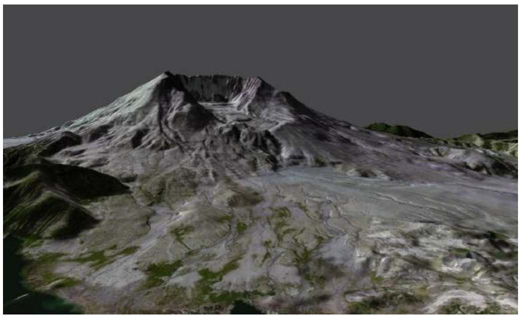
- Scientific Visualization is an interdisciplinary branch of science and a research field of its own.
- It is primarily concerned with the visualization of multi-deminsional phenomena where the emphasis is on realistic rendering of volumes, surfaces, etc. with a dynamic time component.
- Key objectives of scientific visualization in HPC are to (a) analyse/explore & (b) present and communicate scientific data.
- Simulation data can be simple points or connected structured & unstructured grids.
- Pseudocolor Mapping:
  - Pseudocolor mappings map scalar data to color table (copormap).
  - Enables investigation of range of data (temperature, pressure, elevation, velocity, etc.)
  - Offers fast and great mechanism for error diagnostic and visual validation.



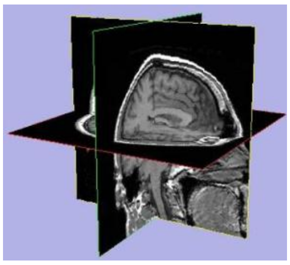
- Surface View:
  - Surface view takes advantage of scalar values to be used as 2 component (e.g. height).
  - Enables a 2D representation to become 3D thus more realistic (e.g. geographic data).
  - Offers a quick understanding of different intensity of the scalar values.



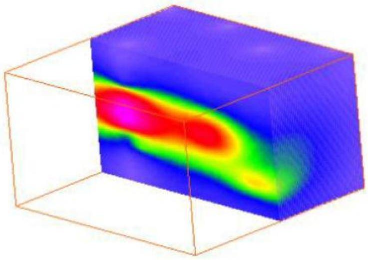
- Texture Mapping:
  - Texture mapping applies a 2D image on a surface by specifying the correspondence among some data points of the image and some data points of the surface.
  - Eanbles detailed and realistic visualizations and contextualizes the visualiation.



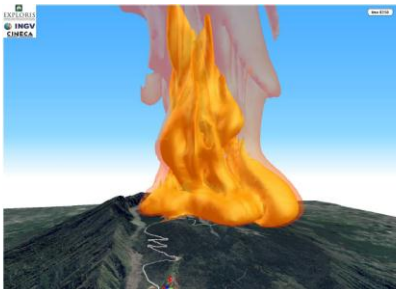
- Slicing:
  - Slicing defines a cutting surface that cuts the 3D data in order to visualize the intersection of the plane with the data being visualized in 2D.
  - Enables investiation of scalar values inside of a volume (i.e. inner view of a 3D object).



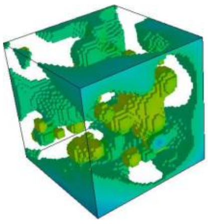
- Cropping/Clipping:
  - Cropping/clipping defines a cutting surface that cuts the 3D data and visualizes everything inside the cutting plane.
  - Enables to remove a part of the dataset and offers step-wise walkthrough (iterations).



- Iosurface/Isoline:
  - An isourface represents points of a constant value (e.g. pressure, temperature, velocity, density) within a scalar volume (named osiline in a 2D domain).
  - Identifies how scalars with constant values are distributed (temperature, pressure, etc.)

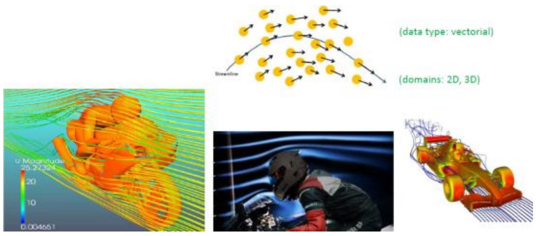


- Threshold:
  - Threshold techniques are used to only visualize scalar values higher (lower) of a defined value, or inside a specifically chosen interval of values.
  - Enables data filtering, emphasizes parts of the data, or used to remove unused data.



- Volume Rendering:
  - Volume rendering does not use intermediate surface representations.
  - The techniques is computing 2D projections of a colored semitransparent volume (typically a 3D scalar field).
  - View a 3D datasets as a whole to investigate interior/density of scalar volumentric data.

- Streamlines:
  - A streamline is a parth traced out by a massless particle as ait moves with the flow and the velocity is tangent to streamline at every given point.
  - Enables the investigation of the nature of flow (e.g. fluids, aero dynamics, etc.)



- A wide variety of tools & techniques exists for scientific visualization starting from low-level programming languages support and customizable GUIs to high level GUIs and HD vizualization.
- Computational steering:
    - Computational steering is the technique of manually intervening with an HPC simulation in order to change its outcome by the manipulation of certain parameters computed.
    - It requires the visualization during runtime (online) in order to properly steer parameters.
    - Computational steering is an old term, recently more used is ‚interactive simulations‘.

12 – Coming soon

- Coming soon.