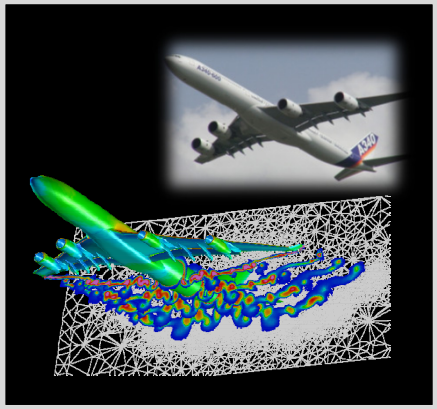


High Performance Computing

ADVANCED SCIENTIFIC COMPUTING

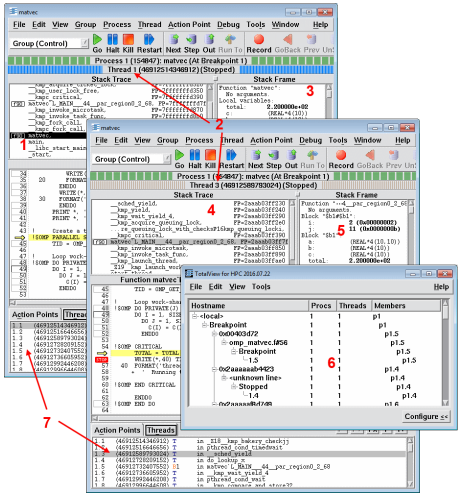
Prof. Dr. – Ing. Morris Riedel
Adjunct Associated Professor
School of Engineering and Natural Sciences, University of Iceland, Reykjavik, Iceland
Research Group Leader, Juelich Supercomputing Centre, Forschungszentrum Juelich, Germany

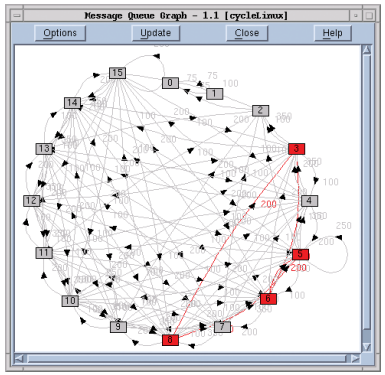
LECTURE 10

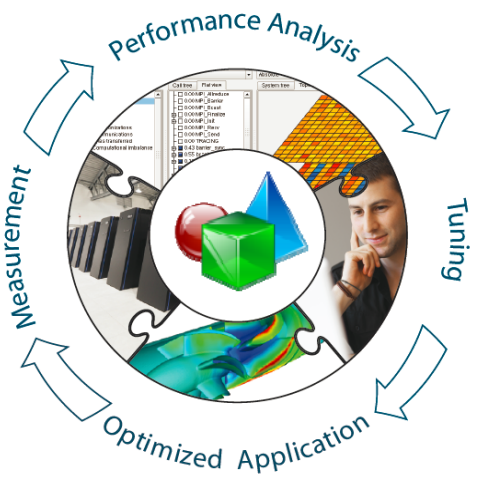
Hybrid Programming & Patterns

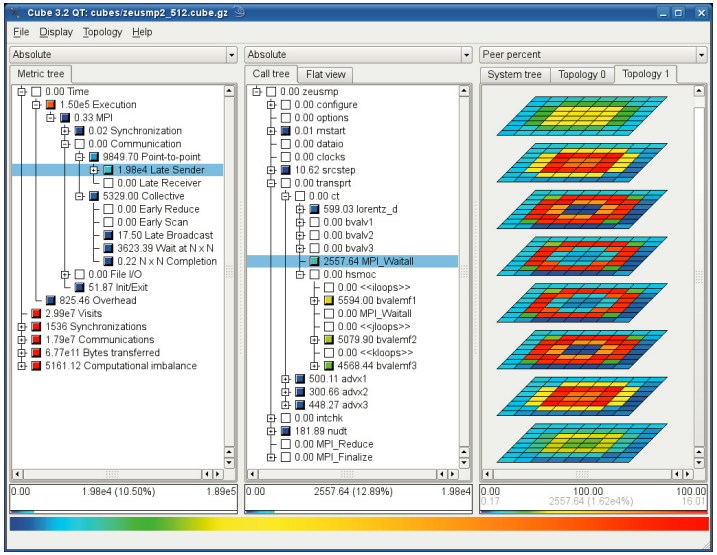
November 11, 2019
Room V02-156

Review of Lecture 9 – Debugging & Profiling & Performance Toolkits









```
...
MPI_Send()
...

/* program logic */

...
MPI_Bcast()
...
```

MPI program source code

MPI_Send()

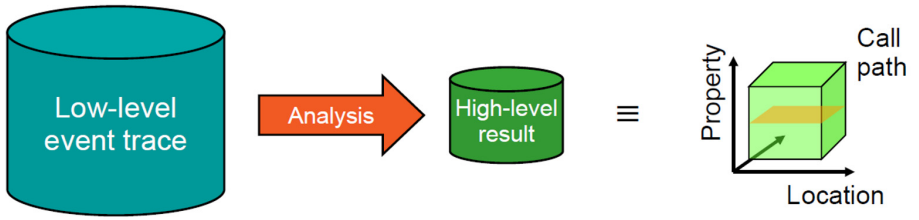
MPI profiling library

MPI_Send()
PMPI_Send()

MPI_Bcast()

MPI standard library

(Scalasca tool example using tracing: which performance problem – where in the program – which processes of the HPC machine are affected)



[1] Scalasca Flyer [2] TotalView Tool

Outline of the Course

1. High Performance Computing
2. Parallel Programming with MPI
3. Parallelization Fundamentals
4. Advanced MPI Techniques
5. Parallel Algorithms & Data Structures
6. Parallel Programming with OpenMP
7. Graphical Processing Units (GPUs)
8. Parallel & Scalable Machine & Deep Learning
9. Debugging & Profiling & Performance Toolsets
10. Hybrid Programming & Patterns
11. Scientific Visualization & Scalable Infrastructures
12. Terrestrial Systems & Climate
13. Systems Biology & Bioinformatics
14. Molecular Systems & Libraries
15. Computational Fluid Dynamics & Finite Elements
16. Epilogue

+ additional practical lectures & Webinars for our hands-on assignments in context

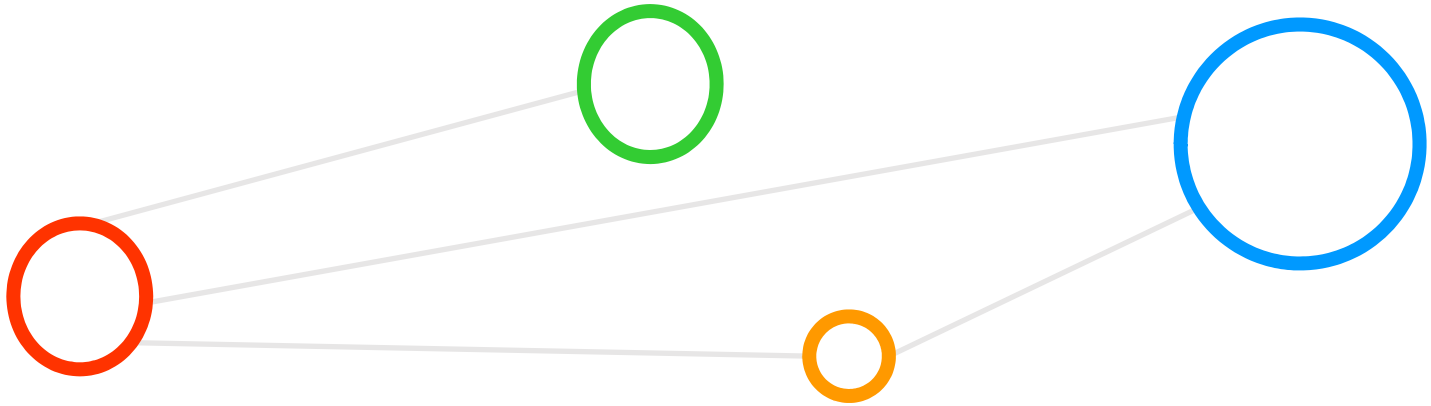
- Practical Topics
- Theoretical / Conceptual Topics

Outline

- Hybrid Programming
  - Motivation & Memory Benefits & Programming Complexity
  - Programming Hybrid Systems with Vector Mode & Task Mode
  - Lessons Learned & Performance of Hybrid Programs
  - Hybrid Programming using simultanously GPUs & CPUs
  - Simulation Sciences & Data Science Applications in Context
- Patterns
  - Neareast Neighbour Communication & Cartesian Communicators
  - Stencil-based Iterative Methods following a Regular Structure
  - Jacobi 2D Application Example & Working with Halo Regions
  - Numerical Methods & Role of Partial Differential Equations (PDEs)
  - Towards Realistic Simulations – Terrestrial Systems Example

- **Promises from previous lecture(s):**
- ***Lecture 1 & 6:*** **Lecture 10 will provide insights into hybrid programming models and introduces selected patterns used in parallel programming**
- ***Lecture 1:*** **Lecture 10 will introduce the programming of accelerators with different approaches and their key benefits for applications**
- ***Lecture 3 & 5:*** **Lecture 10 on Hybrid Programming and Patterns will offer more details on stencil methods & patterns in simulation science applications**
- ***Practical Lecture 5.1:*** **Lecture 10 shows how MPI non-blocking communication is used in Cartesian communicators for nearest neighbor communications**
- ***Lecture 6:*** **Lecture 10 will provide more details about stencil-based iterative methods & used patterns in many different HPC application examples**

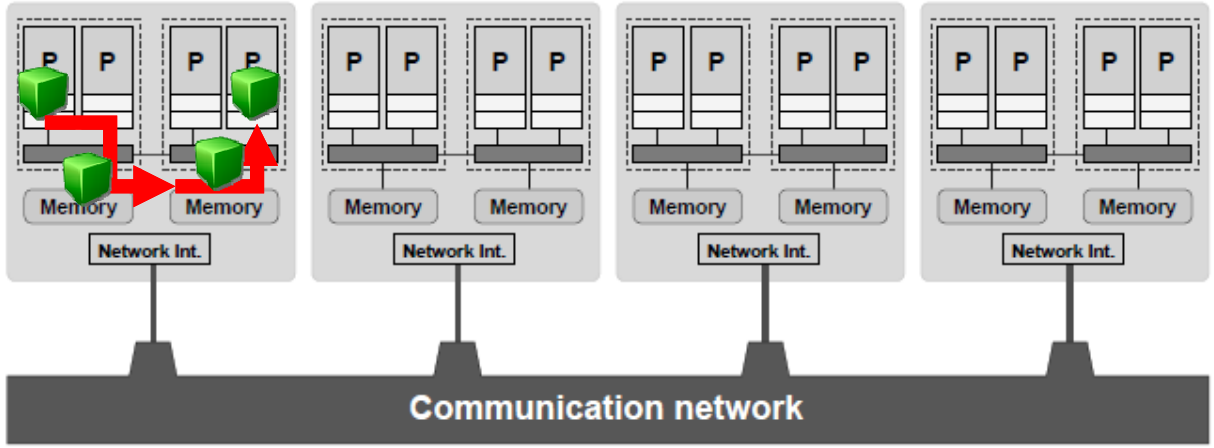
Hybrid Programming



Programming Hybrid Systems – Motivation

- Inefficient ‘on-node communications‘
  - MPI uses ‘buffering techniques‘ to transfer data (cf. Lecture 3 & 4)
  - Transfers may require ‘multiple memory copies‘ to get data from A to B
  - Comparable to a ‘memory copy‘ between different MPI processes
- Take advantage of shared memory techniques where feasible
  - OpenMP threads can read memory on the same node (cf. Lecture 6)

MPI ?

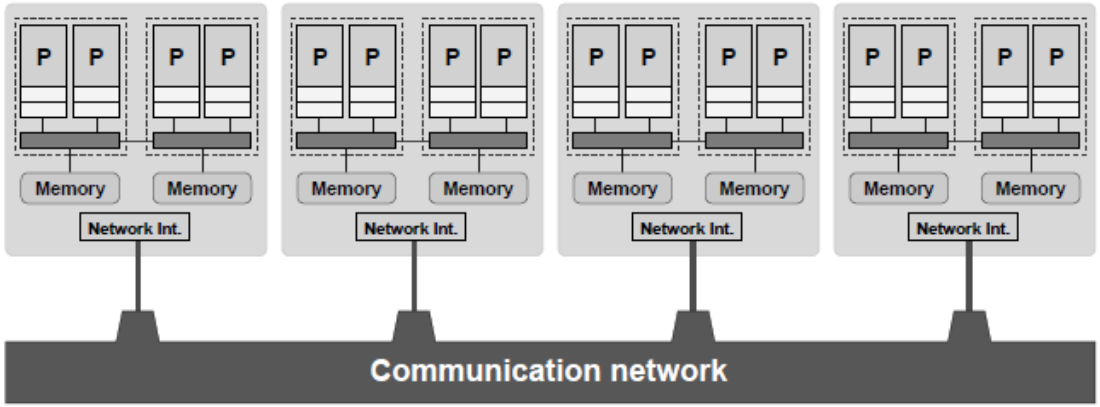


modified from [3] Introduction to High Performance Computing for Scientists and Engineers

Hierarchical Hybrid Computers – Revisited (cf. Lecture 1)

- **A hierarchical 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**

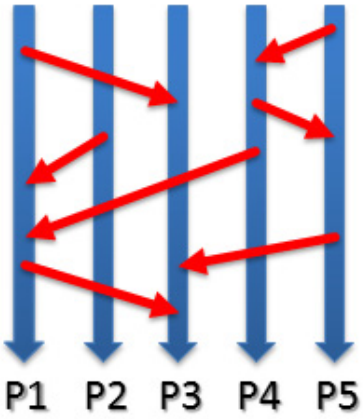
[3] Introduction to High Performance Computing for Scientists and Engineers



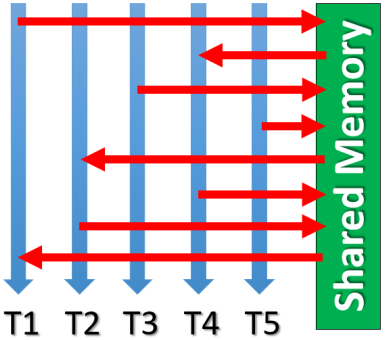
- Features
  - Shared-memory nodes (here ccNUMA) with local NIs
  - NI mediates connections to other remote 'SMP nodes'

Programming Hybrid Systems & Patterns – Revisited (cf. Lecture 1)

- **Hybrid systems programming uses MPI as explicit internode communication and OpenMP for parallelization within the node**
- **Parallel Programming is often supported by using 'patterns' such as stencil methods in order to apply functions to the domain decomposition**



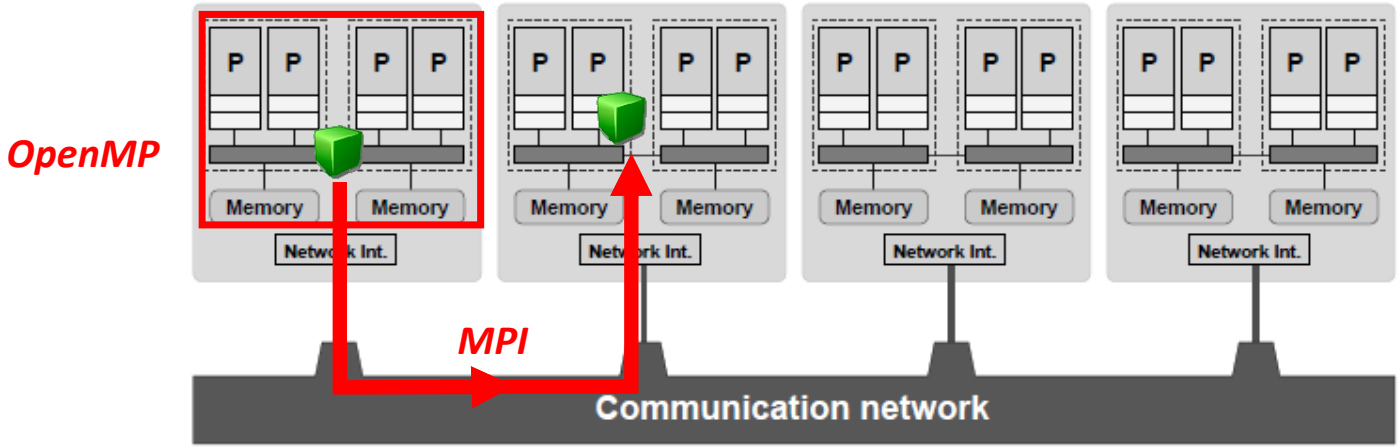
- Experience from HPC Practice
  - Most parallel applications still take no notice of the hardware structure
  - Use of pure MPI for parallelization remains the dominant programming
  - Historical reason: old supercomputers all distributed-memory type
  - Use of accelerators is significantly increasing in practice today
- Challenges with the 'mapping problem'
  - Performance of hybrid (as well as pure MPI codes) depends crucially on factors not directly connected to the programming model
  - It largely depends on the association of threads and processes to cores
  - Patterns (e.g., stencil methods) support the parallel programming



Programming Hybrid Systems & Patterns – Memory Benefits

- Using ‘OpenMP in combination with MPI’
  - Still one buffer, but shared with the threads (spawned from one process)
  - Complex programming, but rewards in good performance

(amount of computing remains constant)



modified from [3] Introduction to High Performance Computing for Scientists and Engineers

- **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 communication 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**

[4] MPI/OpenMP Hybrid Programming

Programming Hybrid Systems – Simple Example

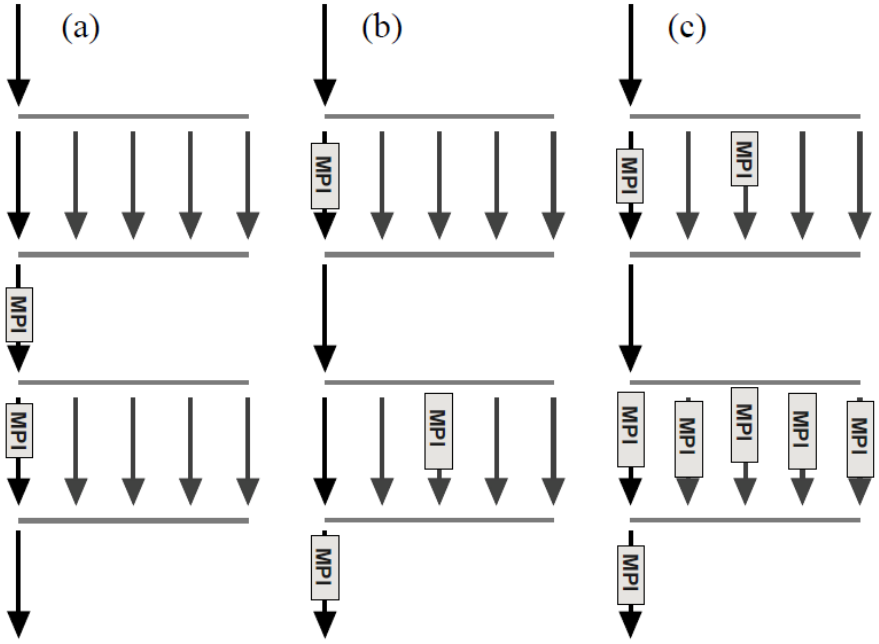
'simplified demo code'

```
#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

Programming Hybrid Systems – Thread Safety

- User specifies ‘guarantees‘ to the MPI library in initialization
  - 4 different options, (d) MPI_THREAD_SINGLE – MPI-only application



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

- **(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

- Any MPI process spawns n worker threads (‘fine-grained parallel’)
  - Augmenting a parallel MPI program with OpenMP compiler directives
  - MPI process takes the role of the OpenMP master thread (becomes T0)
  - Need to specify the maximum number of threads for a certain region
- Example
  - Useful for compute-intensive loops (cf. Lecture 6 specific loop support)
  - Consequence: some processes are in pure MPI parts, others in hybrid parts
- Two implementation approaches
  - Vector mode and Task mode
  - Differ in the degree of interaction between MPI and OpenMP

- **Exploiting an additional level of finer granularity with ‘multi-threading’ can be sometimes the only way to increase parallelism beyond MPI limits (e.g. application logic constraints)**

Hybrid Vector Mode Implementation

```
  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

- **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)**

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

Hybrid Task Mode Implementation

```
!$OMP PARALLEL PRIVATE(iteration,threadID,k,j,i,...)
  threadID  = omp_get_thread_num()
  do iteration=1,MAXITER
  ...
  if(threadID .eq. 0) then
  ...
! Standard 3D Jacobi iteration
! updating BOUNDARY cells
  ...
! After updating BOUNDARY cells
! do halo exchange

    do dir=i,j,k
      call MPI_Irecv( halo data from neighbor in -dir direction )
      call MPI_Send( data to neighbor in +dir direction )
      call MPI_Irecv( halo data from neighbor in +dir direction )
      call MPI_Send( data to neighbor in -dir direction )
    enddo

    call MPI_Waitall( )

  else ! not thread ID 0

! Remaining threads perform
! update of INNER cells 2,...,N-1
! Distribute outer loop iterations manually:

    chunksize = (N-2) / (omp_get_num_threads()-1) + 1
    my_k_start = 2 + (threadID-1)*chunksize
    my_k_end = 2 + (threadID-1+1)*chunksize-1
    my_k_end = min(my_k_end, (N-2))

! INNER cell updates

  endif ! thread ID
!$OMP BARRIER
  enddo
!$OMP END PARALLEL
```

OpenMP parallel region

- **OpenMP parallel environment is created and master threads performs MPI calls**
- **MPI calls can be performed INSIDE OpenMP parallel regions**
- **Useful for functional task decompositions**
- **Enable decoupling of communication and computation**

???

(will lead to massive complexity in a large HPC application program)

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

Comparison of Vector Mode and Task Mode

- Vector Mode (recommended)
  - Basically no real disadvantages, just less flexible as Task Mode
  - Independent programming of OpenMP & MPI ('simplicity')

- Vector mode implementation is straightforward to program and keeps clean code
- Programming hybrid like this means programming MPI/OpenMP parts independently
- 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)
  - Many disadvantages and thus only for experts
  - E.g. blows up sourcecode and increases code complexity significantly
  - E.g. impacts on thread safety and specific support is available in libraries
  - E.g. incremental hybrid parallelization impossible, MPI parts need rewrite

- Task mode is the most flexible option for programming hybrid but also most difficult
- Programming hybrid like this means having MPI calls as part of OpenMP parallel regions
- Convenient OpenMP worksharing parallelization directives not used to differentiate threads

Comparison of Vector Mode and Task Mode – Hybrid Benefits

	Vector mode	Task mode
Improved/easier load balancing	✓	✓
Additional levels of parallelism	✓	✓
Reliable overlapping of communication and computation	✗	✓
Improved rate of convergence	✓	✓
Re-use of data in shared caches	✓	✓
Reduced MPI overhead	✓	✓

[23] G. Hager

Comparison of Vector Mode and Task Mode – Hybrid Drawbacks

	Vector mode	Task mode
OpenMP overheads	✓	✓
Node-level bulk-synchronous communication	✓	(✓)
Possible deficiencies in code optimization by compiler	✓	✓
ccNUMA placement problems	✓	✓ ✓
Nonability to saturate network interface	✓	(✓)
Complexities in thread/core affinity	✓	✓ ✓

[23] G. Hager

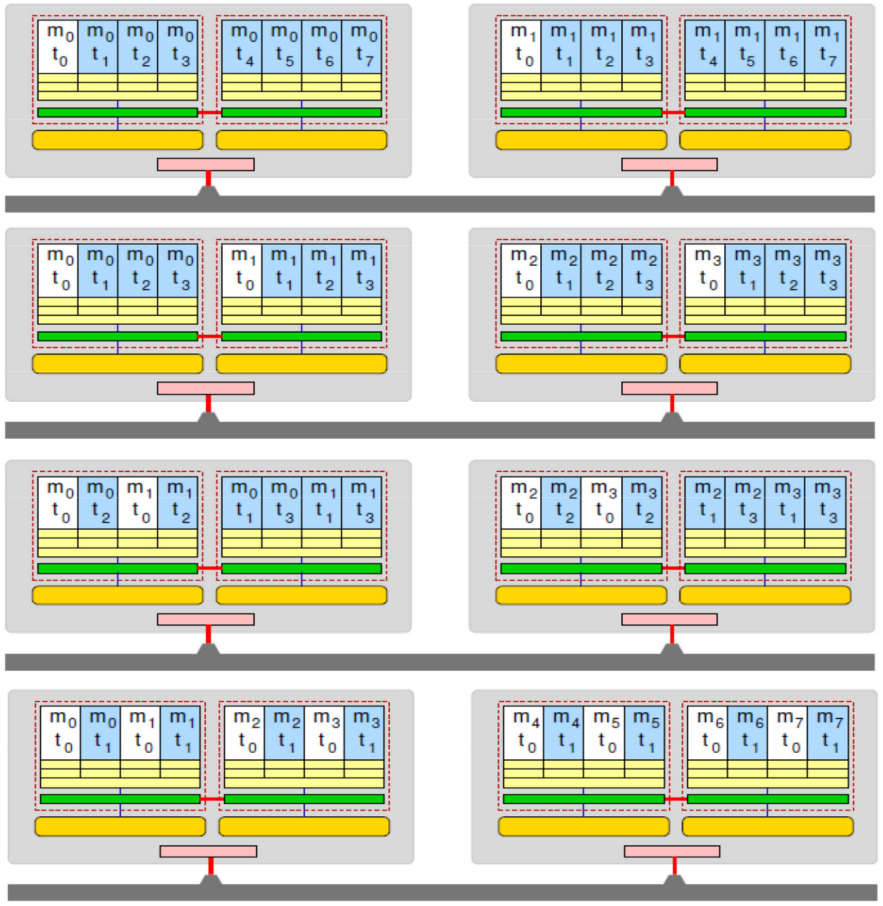
Mapping Challenges – Different Options for Hybrid Programming

One MPI process per node

One MPI process per socket

OpenMP threads pinned "round robin" across cores in node

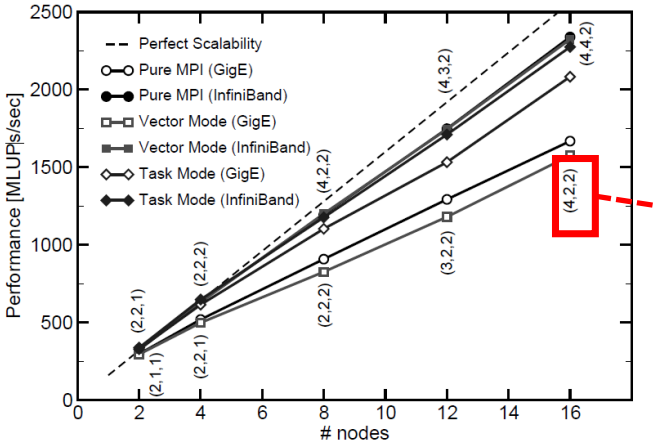
Two MPI processes per node



[23] G. Hager

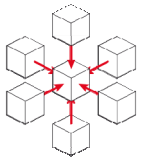
Application Example & Performance Considerations

- Lessons Learned: hybrid MPI/OpenMP vs. plain MPI programming
  - Example: 3D Jacobi application over Gigabit Ethernet & Infiniband (often)
  - Measurement: MLUPs (mega lattice side updates per second)
  - Network: Infiniband shows rarely benefit from hybrid programming



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

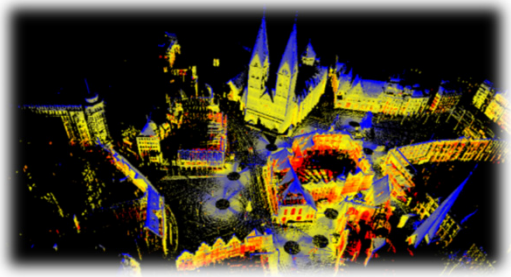
3D domain decomposition topology: number of processes in each Cartesian direction

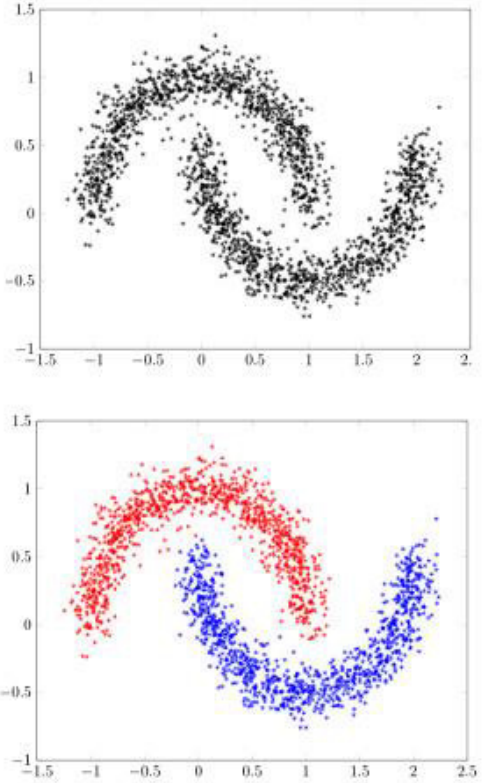


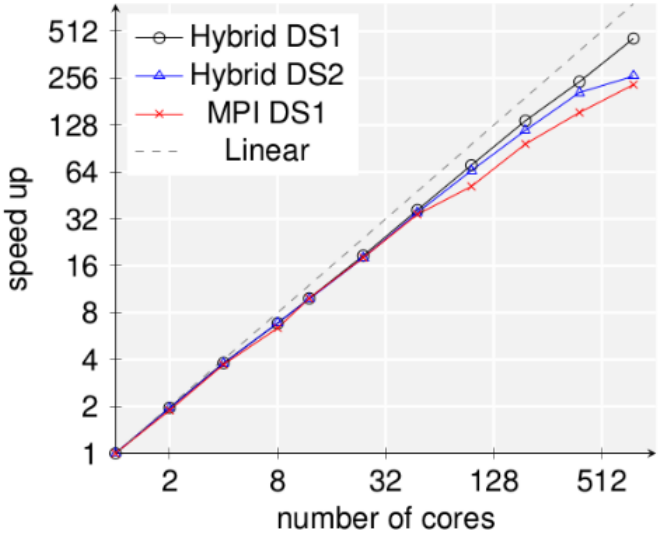
- **Do hybrid programming only if pure MPI scalability is not satisfactory (i.e. often Infiniband in HPC)**
- **Working hard on hybrid programming makes less sense, rather work on perfectly scaling MPI code**
- **Since multi-core systems are expected to grow, above statements need to be reviewed every year**

Scientific Application Example: Data Mining & Clustering

- Hybrid data mining algorithm example
  - Parallel Density-based Spatial Clustering for Applications with Noise (DBSCAN)
  - Using MPI and OpenMP to scale better
  - Standalone OpenMP is also possible to use





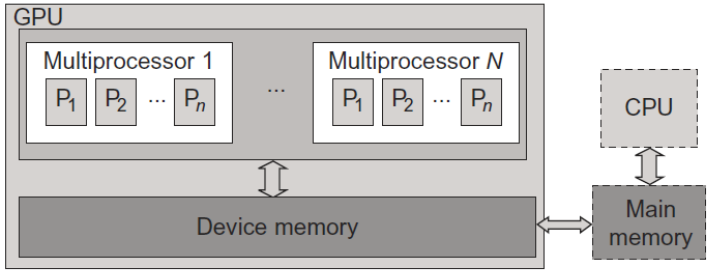



```
#!/bin/bash
#SBATCH --job-name=HPDBSCAN
#SBATCH -o HPDBSCAN-%j.out
#SBATCH -e HPDBSCAN-%j.err
#SBATCH --nodes=2
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=4
#SBATCH --time=00:20:00
#SBATCH --cpus-per-task=4
#SBATCH --reservation=ml-hpc-1

export OMP_NUM_THREADS=4

# location executable
HPDBSCAN=/homea/hpclab/train001/tools/hpdbscan/dbscan

# your own copy of bremen small
BREMENSMALLDATA=/homea/hpclab/train001/bremenSmall.h5

# your own copy of bremen big
BREMENBIGDATA=/homea/hpclab/train001/bremen.h5

srun $HPDBSCAN -m 100 -e 300 -t 12 $BREMENSMALLDATA
```

[5] M. Goetz and M. Riedel et al, Proceedings IEEE Supercomputing Conference, 2015

Many-core GPGPUs – Revisited (cf. Lecture 1)

- Use of very many simple cores
  - High throughput computing-oriented architecture
  - Use massive parallelism by executing a lot of concurrent threads slowly
  - Handle an ever increasing amount of multiple instruction threads
  - CPUs instead typically execute a single long thread as fast as possible



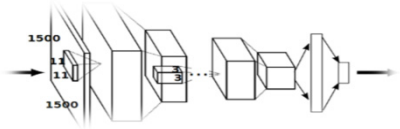
[6] Distributed & Cloud Computing Book

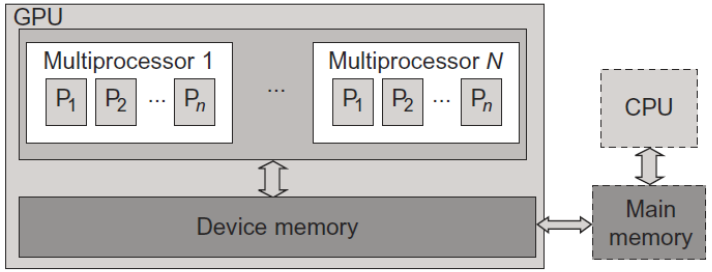
- Many-core GPUs are used in large clusters and within massively parallel supercomputers today
  - Named General-Purpose Computing on GPUs (GPGPU)
  - Different programming models emerge

- **Graphics Processing Unit (GPU) is great for data parallelism and task parallelism**
- **Compared to multi-core CPUs, GPUs consist of a many-core architecture with hundreds to even thousands of very simple cores executing threads rather slowly**

GPU Acceleration – Revisited (cf. Lecture 7)

- GPU accelerator architecture example (e.g. NVIDIA card)
  - GPUs can have 128 cores on one single GPU chip
  - Each core can work with eight threads of instructions
  - GPU is able to concurrently execute 128 * 8 = 1024 threads
  - Interaction and thus major (bandwidth) bottleneck between CPU and GPU is via memory interactions
  - E.g. applications that use matrix – vector/matrix multiplication (e.g. deep learning algorithms)



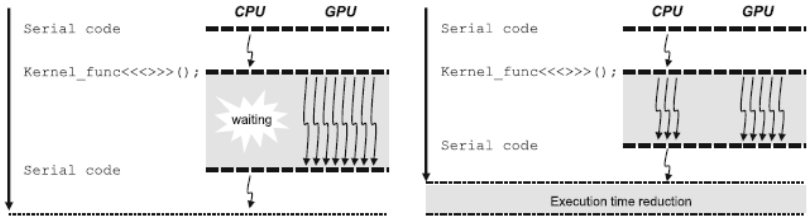


[6] Distributed & Cloud Computing Book

- **CPU acceleration means that GPUs accelerate computing due to a massive parallelism with thousands of threads compared to only a few threads used by conventional CPUs**
- **GPUs are designed to compute large numbers of floating point operations in parallel**

Another Type of Hybrid Programming: CPUs & GPGPUs

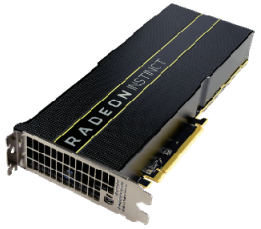
- Emerging ‘hybrid programming model‘
  - Using General-purpose computing on graphics processing units (GPGPUs)
  - Combine with traditional CPUs to accelerate elements of processing
  - Idea: exploit parallelism across host CPU cores in addition to the GPU cores



[7] ‘Boosting CUDA Applications with CPU-GPU Hybrid Computing’

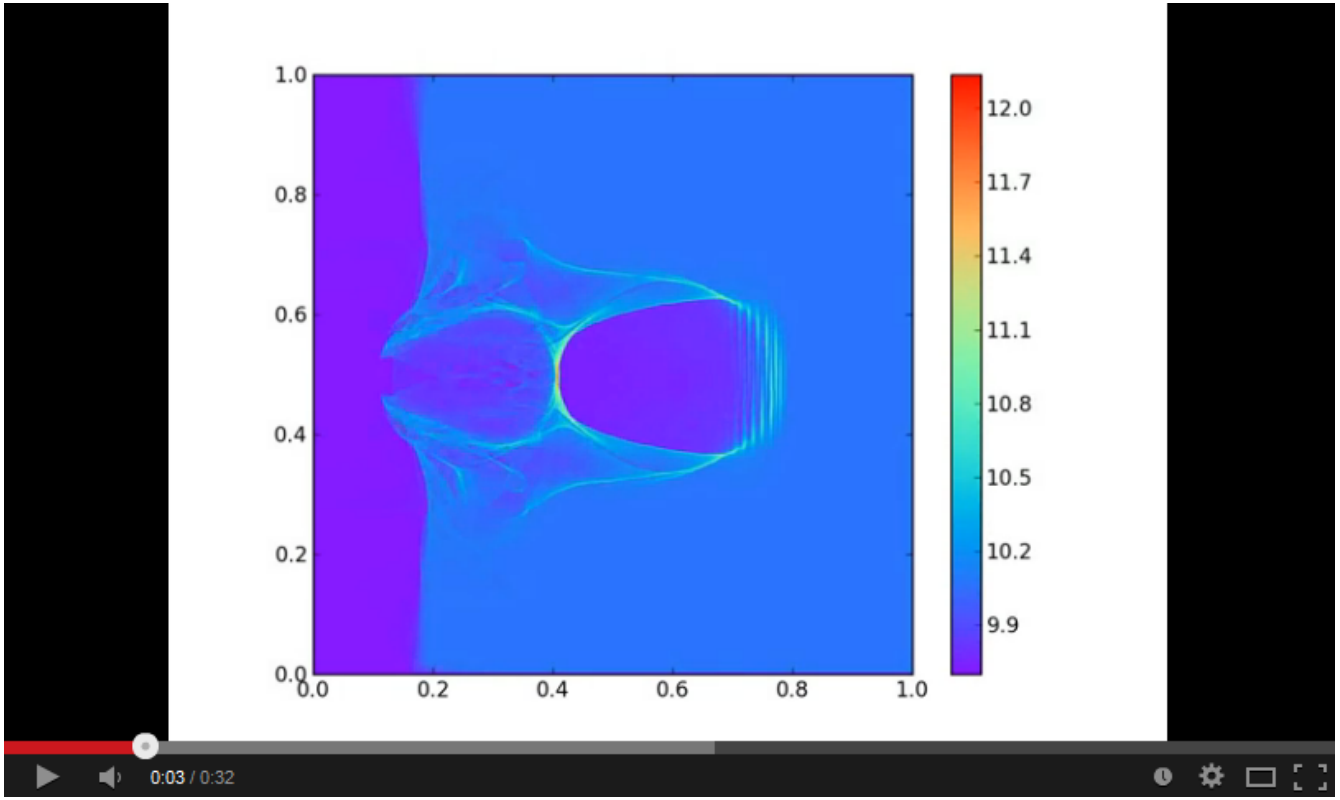
[8] NVidea Tesla

- Programming
  - NVidea Compute Unified Device Architecture (CUDA) as dominant propriety framework (cf. Lecture 7)
  - GPU-accelerated scientific applications increasing
  - AMD Radeon and other accelerators with new programming languages



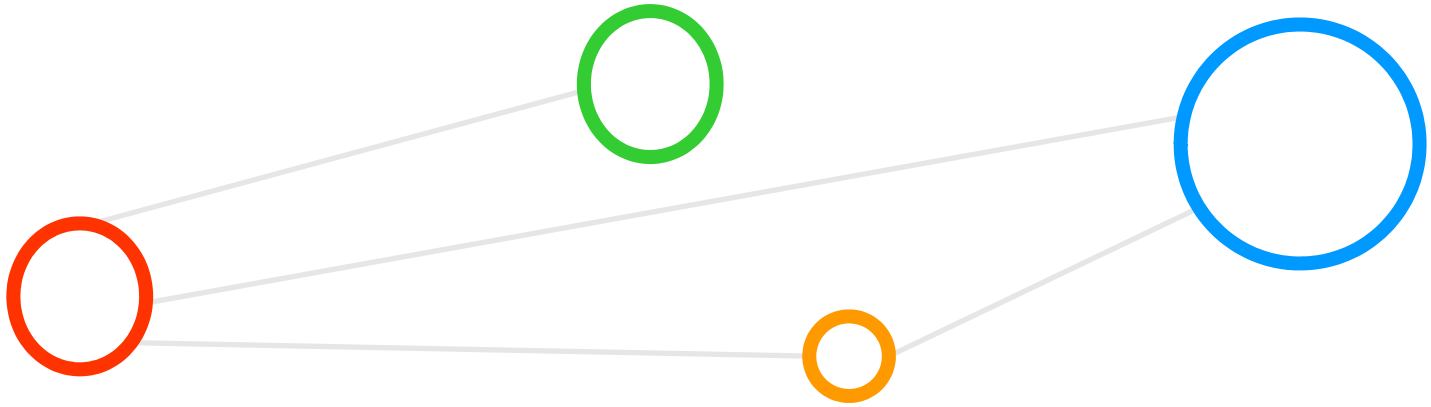
[9] AMD Radeon Instinct

[Video] Application Example



[10] Jasmine Particle in Cell codes Framework

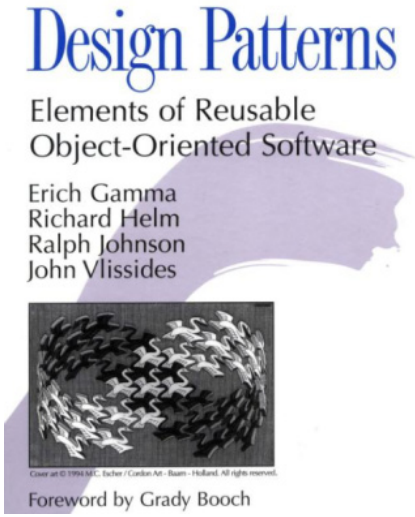
Patterns



Meaning of (Common) Patterns

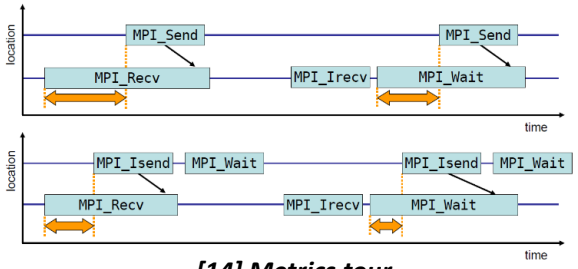
- Not ‘Software Design Patterns‘
  - Often used in software engineering for repeating patterns in programming
  - Only rarely used in scientific computing, major reason: ‘physics rule code‘
  - Tried a number of times in HPC , e.g. object oriented frameworks / libraries / papers / reference models, …

[11] E. Gamma et al., 1994

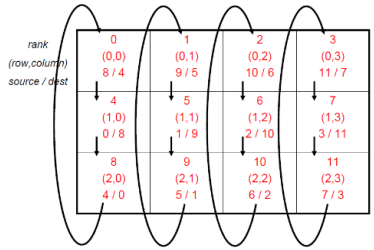


- Common Patterns in HPC
  - Similiar thinking as ‘design patterns‘, but more ‘teaching common practice‘
  - Refer rather to commonly used methods again used often in parallel codes
- Various impacts & usage
  - Patterns affect how to organize data structures and domain decomposition
  - Common patterns are able to reduce communication or computation
  - Increases often also code readability

Blocking vs. Non-blocking Communication – Parallel Algorithms & Patterns

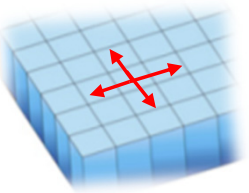


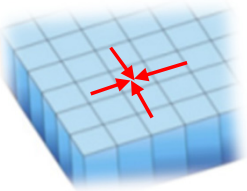
[14] Metrics tour



[15] German MPI Lecture

```
// do some work with MPI communication operations...
// e.g. exchanging simple data with all neighbours

outbuf = rank;

for (i=0; i<4;i++) {
  dest=nbrs[i];
  source=nbrs[i];

  // perform non-blocking communication
  MPI_Isend(&outbuf, 1, MPI_INT, dest, tag, MPI_COMM_WORLD, &reqs[i]);
  MPI_Irecv(&inbuf[i], 1, MPI_INT, source, tag, MPI_COMM_WORLD, &reqs[i+4]); // 4 as a kind of offset
}

// wait for non-blocking communication to be completed for output
MPI_Waitall(8, reqs, stats);

printf("rank= %d has received (u,d,l,r)= %d %d %d %d \n", rank,
       inbuf[UP], inbuf[DOWN], inbuf[LEFT], inbuf[RIGHT] );

MPI_Finalize();

return 0;
```





- **Blocking vs. non-blocking: MPI_Send() blocks until data is received; MPI_Isend() continues**
- **The use of these functions can cause different performance problems (e.g. here 'late sender')**
- **MPI_Wait() does wait for a given MPI request to complete before continuing**
- **MPI_Waitall() does wait for all given MPI requests (e.g. waiting for message) to complete before continuing**

MPI_Waitall

Waits for all given MPI Requests to complete

Synopsis

```
int MPI_Waitall(int count, MPI_Request array_of_requests[],
                MPI_Status array_of_statuses[])
```

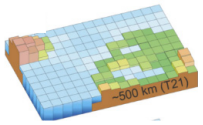
Input Parameters

count
list length (integer)

array_of_requests
array of request handles (array of handles)

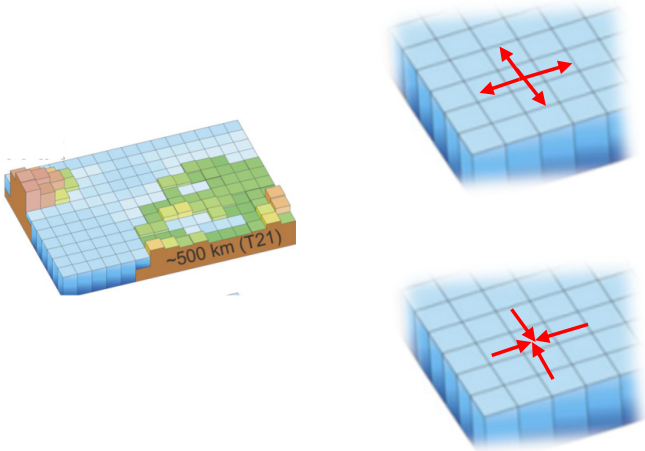
Output Parameters

array_of_statuses
array of status objects (array of Statuses). May be MPI_STATUSES_IGNORE.



Example: Cartesian Communicators – Practice & Experience (cf. Lecture 4)

- Methods for creating new communicators
  - Create a cartesian communicator out of existing communicator
  - Splitting an existing communicator
  - Duplicating an existing ecommunicator
  - Modifying a group of processes
  - Reordering

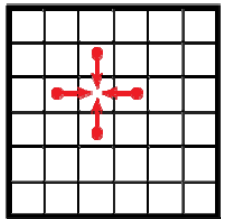


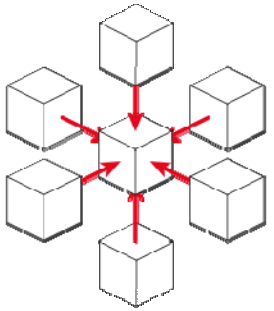
- Cartesian Communicators – ‘MPI virtual topology‘
  - NOT directly related with physical topology of hardware (network)
  - Implementations of MPI might perform corresponding mapping (network)
  - Describes the topological interrelation between processes
  - Enable nearest neighbour communication patterns in a simple form

- **Cartesian communicators are useful methods to implement nearest neighbour communication patterns that are used in many applications in scientific computing and simulation sciences**

Stencil-based Iterative Methods

- Simulation sciences & numerical methods
  - Stencil-based iterative methods
  - Applicable with exceptions with other methods: Finite element method (selected codes on regular grids can use stencil codes)
- Selected application examples
  - Computational Fluid Dynamics (CFD) codes
  - Partial differential equations (PDE) solver
  - Jacobi method
  - Gauss-Seidel method
  - Image processing





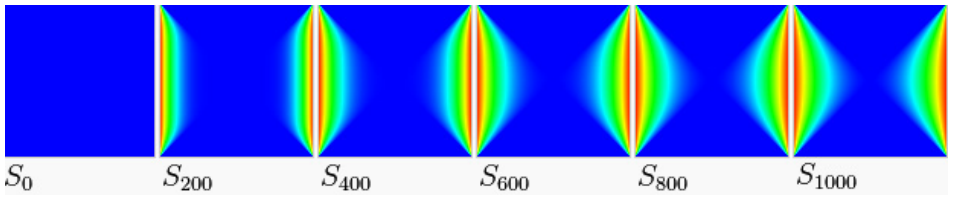
- **Stencil-based iterative methods update array elements according to a fixed pattern called 'stencil'**
- **The key of stencil methods is its regular structure mostly implemented using arrays in codes**
- **Method is often used in computational science as part of scientific and angineering applications**

[12] Wikipedia on 'stencil code'

Jacobi 2D Application Example – Shared Memory with OpenMP not Enough?

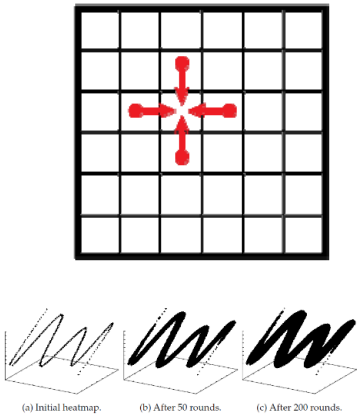
- Solver
  - Each diagonal element is solved and approximate value is plugged in
  - The process is iterated until it converges
- Update function 2D Jacobi iterative method example
  - E.g. computes the arithmetic mean of a cell‘s four neighbours
  - E.g. solving diffusion equations (heat dissipation example)

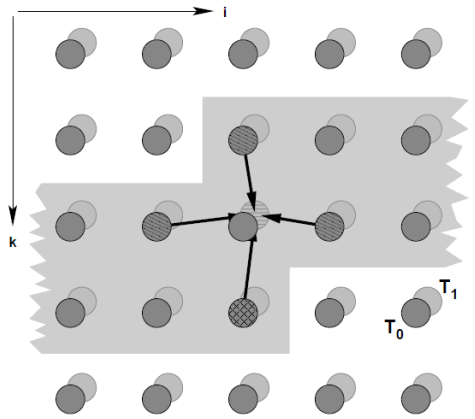
- **The Jacobi iterative method is a stencil-based iterative method used in numerical linear algebra**
- **Algorithm for determining the solutions of diagonally dominant system of linear equations**



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

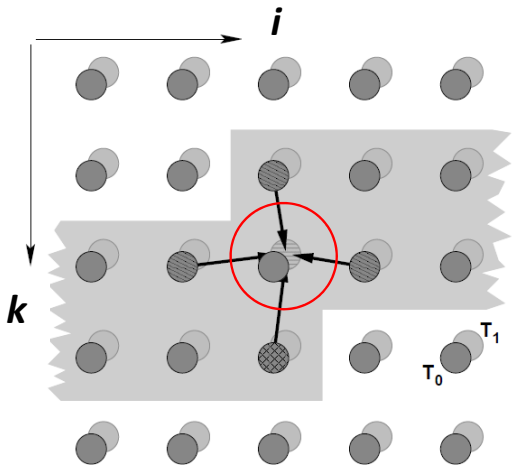
[12] Wikipedia on ‘stencil code’



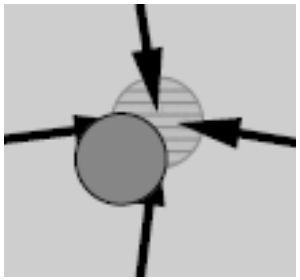


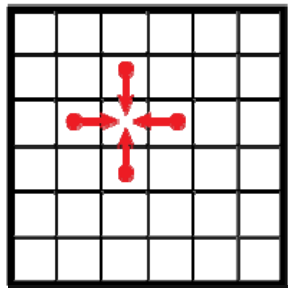
Jacobi 2D Application Example – Diffusion Equation

- Iterative (time) step → a ‘stencil update‘
  - A correction at coordinate (x_i, y_i) is calculated using a diffusion equation
  - Calculation needs the ‘old’ values from the four next neighbouring points
  - ‘Old‘ means: the values from the previous iteration!
  - After all points have been updated (a ‘sweep’) repeat & next time step
  - Updated values must be written to a second array



‘shaded’ area is cache contents





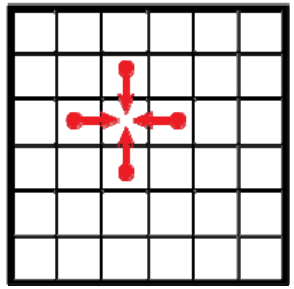
[3] Introduction to High Performance Computing for Scientists and Engineers

Jacobi 2D Application Example – Arithmetic Mean & Neighbouring Cells

- From the problem to computational data structures
  - Apply an 'isotropic lattice' technique

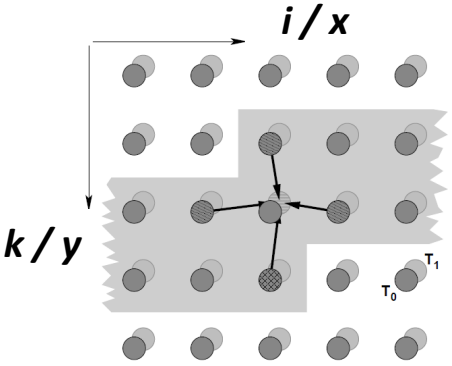
```
do k = 1,kmax
  do i = 1,imax
     ! four flops, one store, four loads
     phi(i,k,t1) = (  phi(i+1,k,t0) + phi(i-1,k,t0)
                    + phi(i,k+1,t0) + phi(i,k-1,t0) ) * 0.25
  enddo
enddo
```

arithmetic mean (¼) from four neighbouring isotropic cells



Modified from [3] Introduction to High Performance Computing for Scientists and Engineers

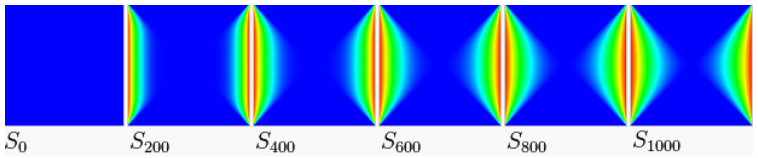
[12] Wikipedia on 'stencil code'

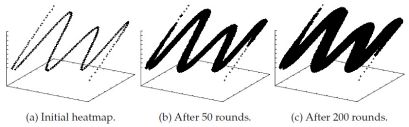


$$\frac{\delta\Phi(x_i,y_i)}{\delta t} = \frac{\Phi(x_{i+1},y_i)+\Phi(x_{i-1},y_i)-2\Phi(x_i,y_i)}{(\delta x)^2} + \frac{\Phi(x_i,y_{i-1})+\Phi(x_i,y_{i+1})-2\Phi(x_i,y_i)}{(\delta y)^2}$$

'change over time' diffusion equation

$$\frac{\partial\Phi}{\partial t} = \Delta\Phi$$

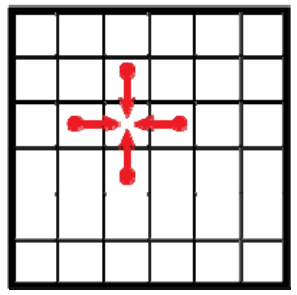




Jacobi 2D Application Example – Algorithm

- Time step: calculation from t_0 to t_1
- Performance considerations
  - Compute view: Floating point operations per second (FLOPs)
  - Data view: Stores & loads from cache (or memory if cache misses occur)

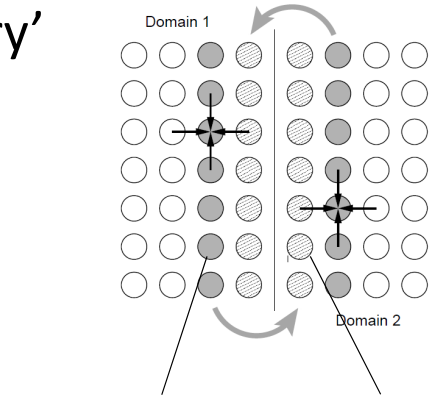
```
double precision, dimension(0:imax+1,0:kmax+1,0:1) :: phi
integer :: t0,t1
t0 = 0 ; t1 = 1
do it = 1,itmax    ! choose suitable number of sweeps
  do k = 1,kmax
    do i = 1,imax
       ! four flops, one store, four loads
       phi(i,k,t1) = (  phi(i+1,k,t0) + phi(i-1,k,t0)
                      + phi(i,k+1,t0) + phi(i,k-1,t0) ) * 0.25
    enddo
  enddo
  ! swap arrays
  i = t0 ; t0=t1 ; t1=i
enddo
```



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

Jacobi 2D Application Example – Halo Regions

- Two-dimensional Jacobi solver
- Shared-memory and complete domain fits into memory
  - Relatively easy: all grid sites in all domains can be updated before the processors have to synchronize at the end of the sweep (i.e. time step)
- Distributed-memory with no access to 'neighbours memory'
  - Complex: updating the boundary sites of one domain requires data from adjacent domain(s) → maybe out of memory
  - Idea: before a domain update (next step), all boundary values needed for the upcoming sweep must be communicated to the relevant neighboring domains
  - Store this data somewhere, so extra grid point(s) introduced (*halo/ghost layers*)

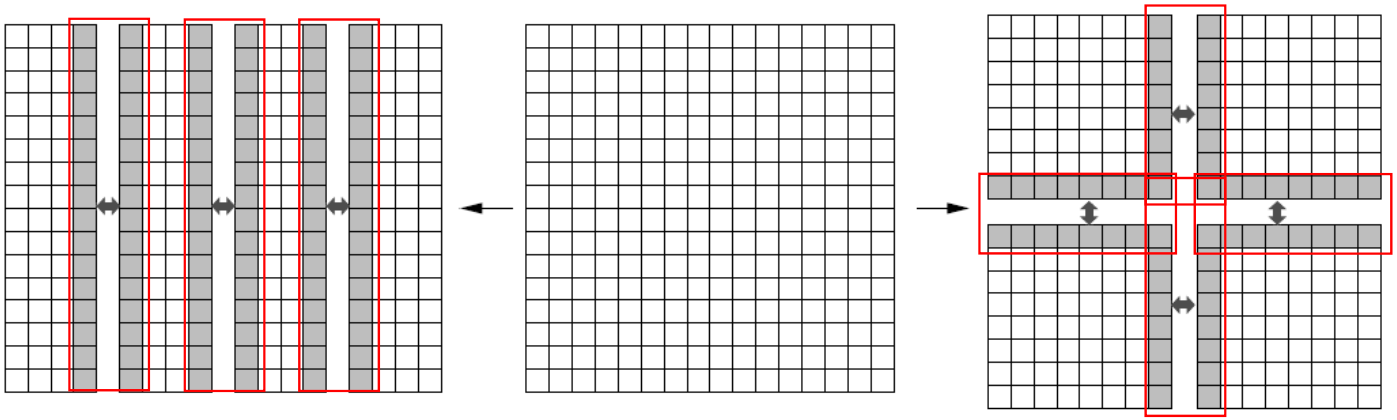


(a wider halo than 'one' is also possible)

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

Jacobi 2D Application Example – Halo Regions & Communication Costs

- Two-dimensional Jacobi solver in context of communication cost:
  - Often choosing the optimal domain decomposition is application-specific
  - Next neighbour interactions needed and can vary (more/less shaded cells)
  - Simple: Cutting in four stripes domains (left) incurs more communication
  - Optimal decomposition: four domains (right) incurs less communication



3 * 16 = 48

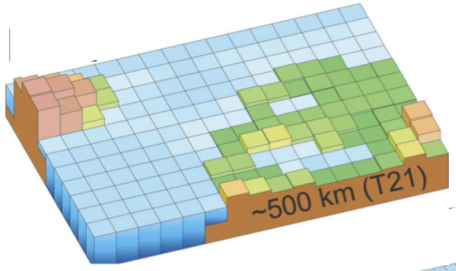
[3] Introduction to High Performance Computing for Scientists and Engineers

4 * 8 = 32

- **Halo regions are needed for local computation 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**

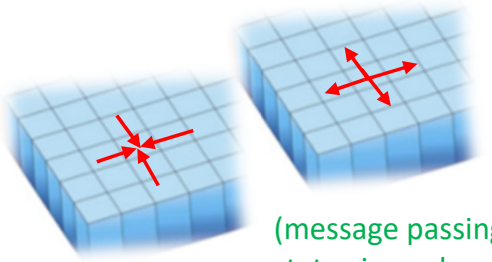
Terrestrial Systems Example – Towards Realistic Simulations – Granularity

- Scientific computing with HPC simulates ‘ ~realistic behaviour ‘
  - Apply common patterns over time & simulate based on numerical methods
  - Increasing granularity (e.g. domain decomposition) needs more computing

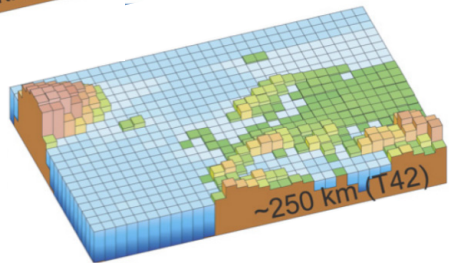


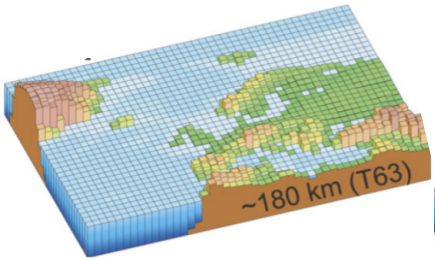
(introduce more and more physical parameters over time…)

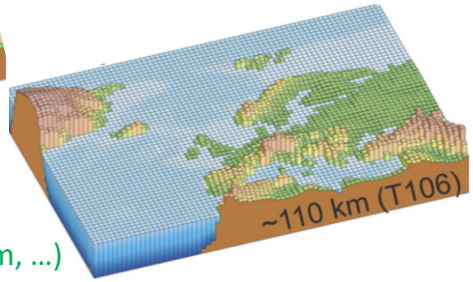
(compute more physical laws…)



(message passing of status in each cell)







(add scientific domain studies:
e.g. rainfall, ocean waves, wind, oil, storms…)

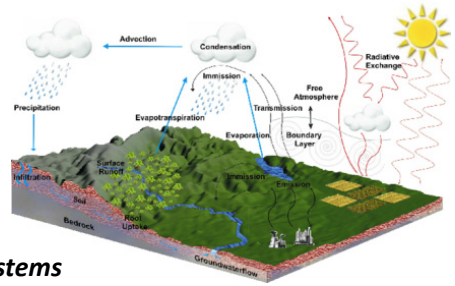
(add objects to study: boats, fish, birds, people, oil platform, …)

➢ **Lecture 12 will provide more details on using different domain decompositions for terrestrial system and climate simulations on HPC**

Terrestrial Systems Example – Need for Numerical Methods in HPC

- Behaviour ‘governed by equations‘ are computed
  - Nature is (too) complex & interconnected: simplification
- Behaviour governed by ‘difference equations‘
  - System state only change at discrete instants of time
  - System state ‘not change in time continously‘
- Behaviour governed by ‘differential equations‘
  - System state evolves ‘continously in time‘
- Selected ‘scientific questions‘ for simulations
  - Under what circumstances will a system evolve into an ‘equilibrium–state’ (state which does not change)
  - Under what circumstances will the system evolve into a ’periodic state’ (states the system return to over time)

[19] Introduction to SC

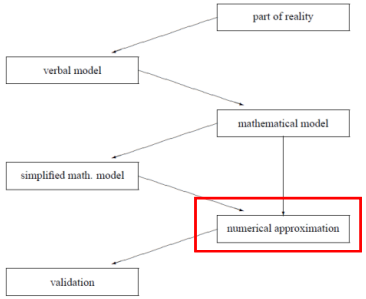


[21] SimLab Terrestrial Systems

(solutions can be computed simply by applying definitions iteratively)

- **Solving some mathematical problems & equations is too computational intensive → approximate**
- **Numerical methods are methods that obtain numerical approximation solutions to problems**

(harder to solve, e.g. initial value problem)



➢ **Lecture 12 will provide more details on using different domain decompositions for terrestrial system and climate simulations on HPC**

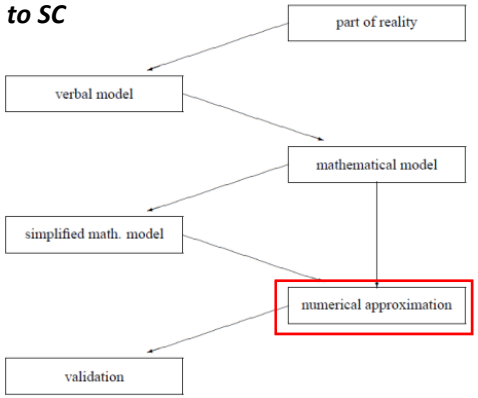
Terrestrial Systems – Role of Partial Differential Equations (PDEs)

- HPC simulation modelling
  - PDEs enable rates of change (of continous variables)
  - PDEs used to formulate problems involving functions of several variables
  - PDEs describe a wide variety of phenomena (e.g. sound, heat, electrostatics, fluid flow, etc.)
  - PDEs model multi-dimensional dynamical systems
- Differences to 'ordinary differential equations'
  - Ordinary differential equations deal with functions of a single variable and their derivatives
  - Ordinary differential equations model one-dimensional dynamical system

modified from [20] Wikipedia on 'Partial Differential Equation'

Solving those equations is often too complicated computationally expensive or impossible to analytically compute driving the need for numerical approximation

[19] Introduction to SC

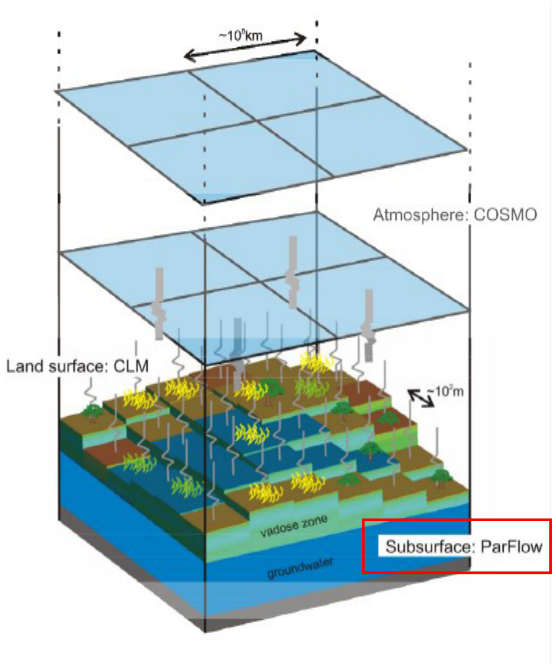


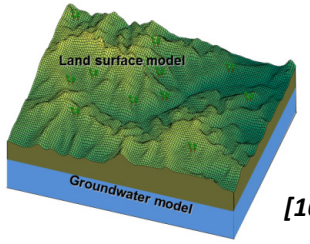
- **HPC models often use toolkits (e.g. PETSc) for Partial Differential Equations (PDEs) that are differential equations that contains unknown multivariable functions and their partial derivatives**
- **A general method in HPC modelling use parallel PDEs tools to approximate solutions to problems**

➢ **Lecture 12 will provide more details on using different domain decompositions for terrestrial system and climate simulations on HPC**

Terrestrial Systems – ParFlow Model Parallel Application Example

- Modelling ‘hydrology‘ processes
  - Parallel watershed flow model (ParFlow)
  - Simulate surface and subsurface fluid flow
  - Use in the assessment and management of groundwater and surface water
  - Investigate system physics and feedbacks
  - Understand interactions at a range of scales
  - Suitable for large scale & high resolution
- Parallel ‘numerical‘ application
  - Developed over 10 years (aka stable code)
  - Offers advanced numerical solvers for massively parallel HPC systems





[16] R. Maxwell

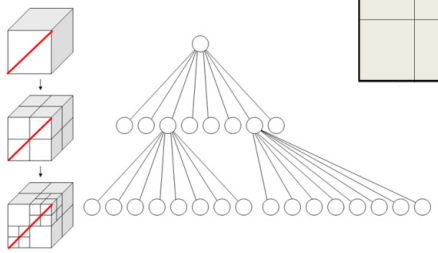
- **ParFlow enables the parallel simulation of hydrology processes with (sub-)surface fluid flows**

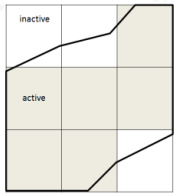
[17] ParFlow Web page

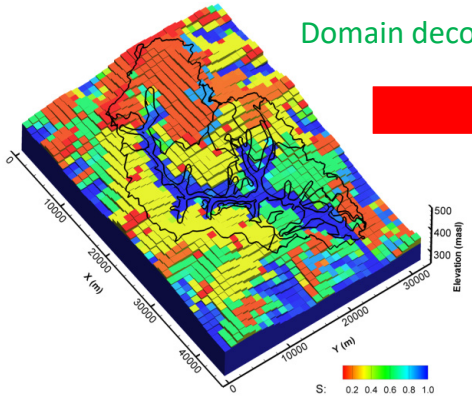
➢ **Lecture 12 will provide more details on using different domain decompositions for terrestrial system and climate simulations on HPC**

Terrestrial Systems – ParFlow Model Example using Parallel Programming

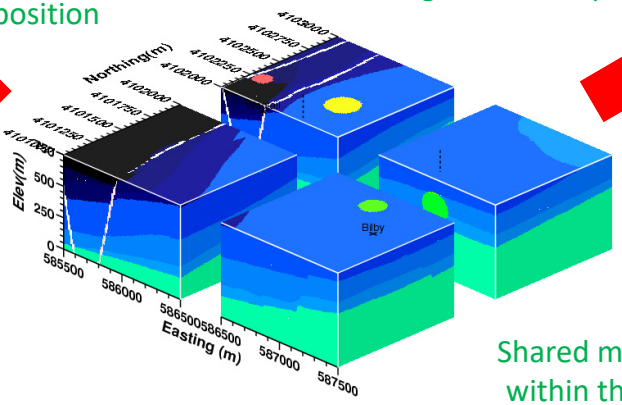
- Parallelization Techniques
  - 3D Grid domain decomposition
  - 3D code (use octree-space partitioning algorithm)
  - Implements hybrid programming
  - Requirement of ‘halo regions‘ for numerical equations

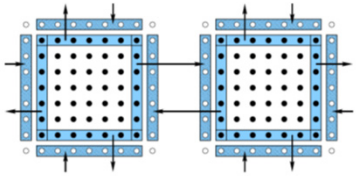




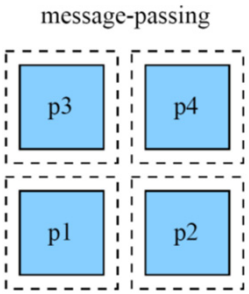


Domain decomposition

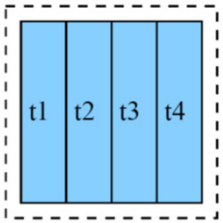




Distributed memory across the grid & halo updates



Shared memory within the grid

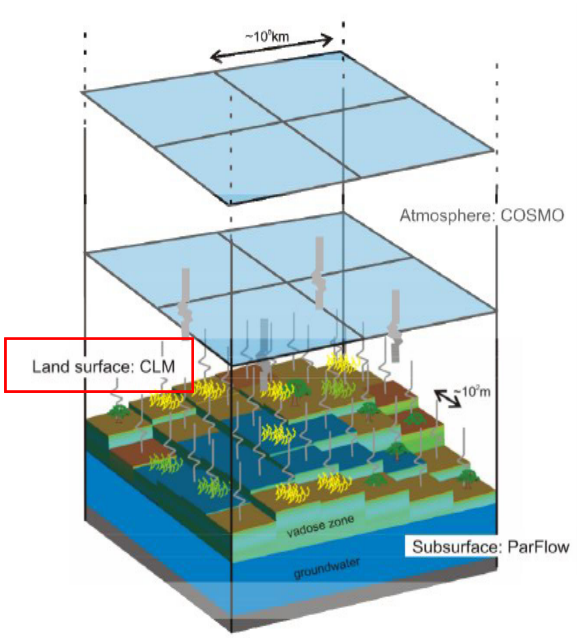


[17] ParFlow Web page

➢ **Lecture 12 will provide more details on using different domain decompositions for terrestrial system and climate simulations on HPC**

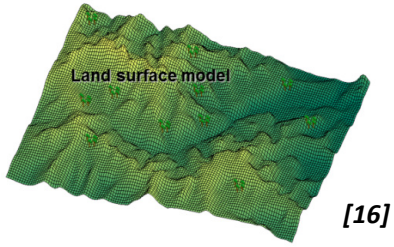
Terrestrial Systems – CLM Model Parallel Application Example

- Modelling ‘land surface‘ processes
  - Community land model (CLM)
  - Simulates concepts of ecological climatology
  - Understand how natural & human changes in vegetation affect the climate
  - Examine physical, chemical, and biological processes that affect (or are affected by climate across spatial / temporal scales
  - Investigate terrestrial ecosystems through their cycling of energy, water, chemical elements, and trace gases
  - Explore impact of terrestrial ecosystems as important determinants of climate



[18] CLM Web page

- **CLM enables the parallel simulation of land-surface with physical & chemical & biological processes**

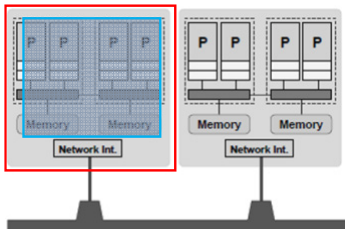


[16] R. Maxwell

➢ **Lecture 12 will provide more details on how to couple scientific simulation codes that simulate parts of a domain with different physics**

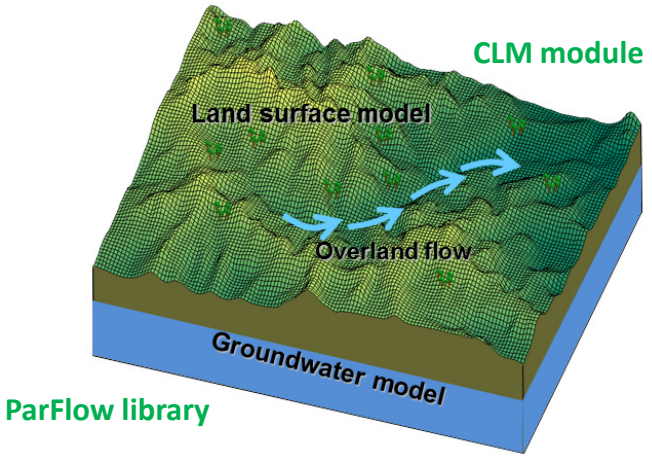
Terrestrial Systems – CLM Model Application – Parallel Programming

- Parallelization Techniques
  - Implements ‘hybrid programming‘
  - OpenMP within a node (cf. Lecture 6)
  - MPI routines for parallelism across nodes (cf. Lecture 3)



[16] R. Maxwell

- Coupled as module
  - Code is often fully coupled with ParFlow
  - Coupling is performed in a way that CLM is incorporated into ParFlow as a module (full coupled, fully parallel)
  - E.g. flow of water on land-surface affects groundwater model



➢ **Lecture 12 will provide more details on how to couple scientific simulation codes that simulate parts of a domain with different physics**

Systems Biology – Parallel Neuroscience Application Example

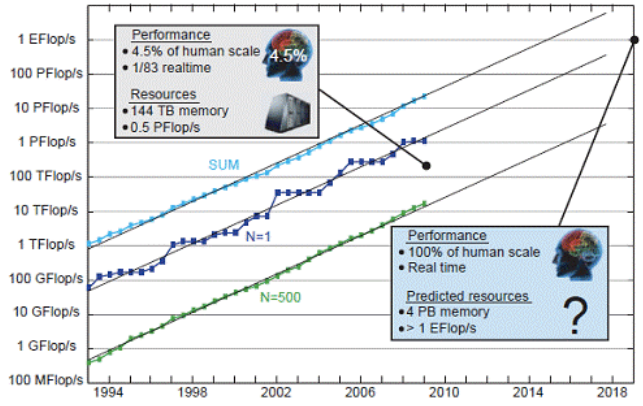
- Scientific case: understanding the function of the human brain
- Neuron/NEST code:
  - Parallel application codes to simulate biologically realistic neural networks (neurons + synapses)

(the biggest supercomputers today just reach ~4.5% of human scale)

- Simulate models of the brain at different levels
  - Different 'granularity': Molecular, cellular, network level
  - Simulated brain will reach up to ~900 TB (Big Data!)

Simulated ~2 billion neurons

- 1 second biological time
- 40 minutes compute time (on K supercomputer)



[22] HBP Project

➢ **Lecture 13 will provide more details on various systems biology & bioinformatics application codes that use parallel computing**

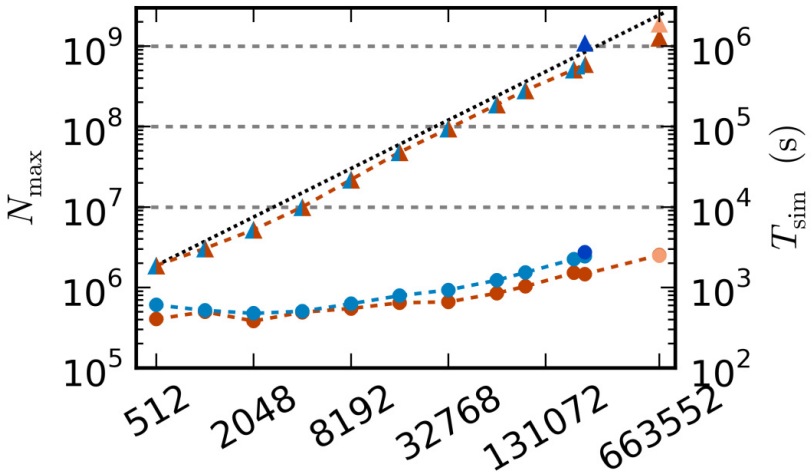
Systems Biology – Parallel Neuroscience Application – Parallel Programming

- Simulations of spiking – parallel neural network models
  - Use parallelization (e.g. MPI cf. Lecture 3 and hybrid programming)

Human Brain Project

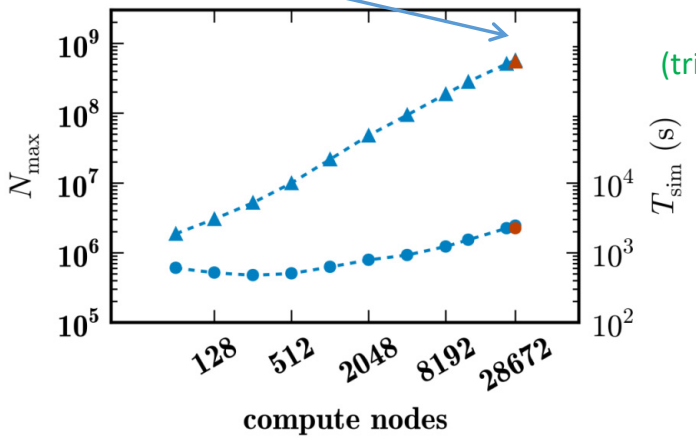
nest::

Largest spiking neural network simulation to date:
$1.86 \cdot 10^9$ neurons, $11.1 \cdot 10^{12}$ synapses



(11.250 synapses/neuron – light blue JUQUEEN, red: K computer)

550 Mio. Neurons, $5.5 \cdot 10^{12}$ synapses on 458752 cores



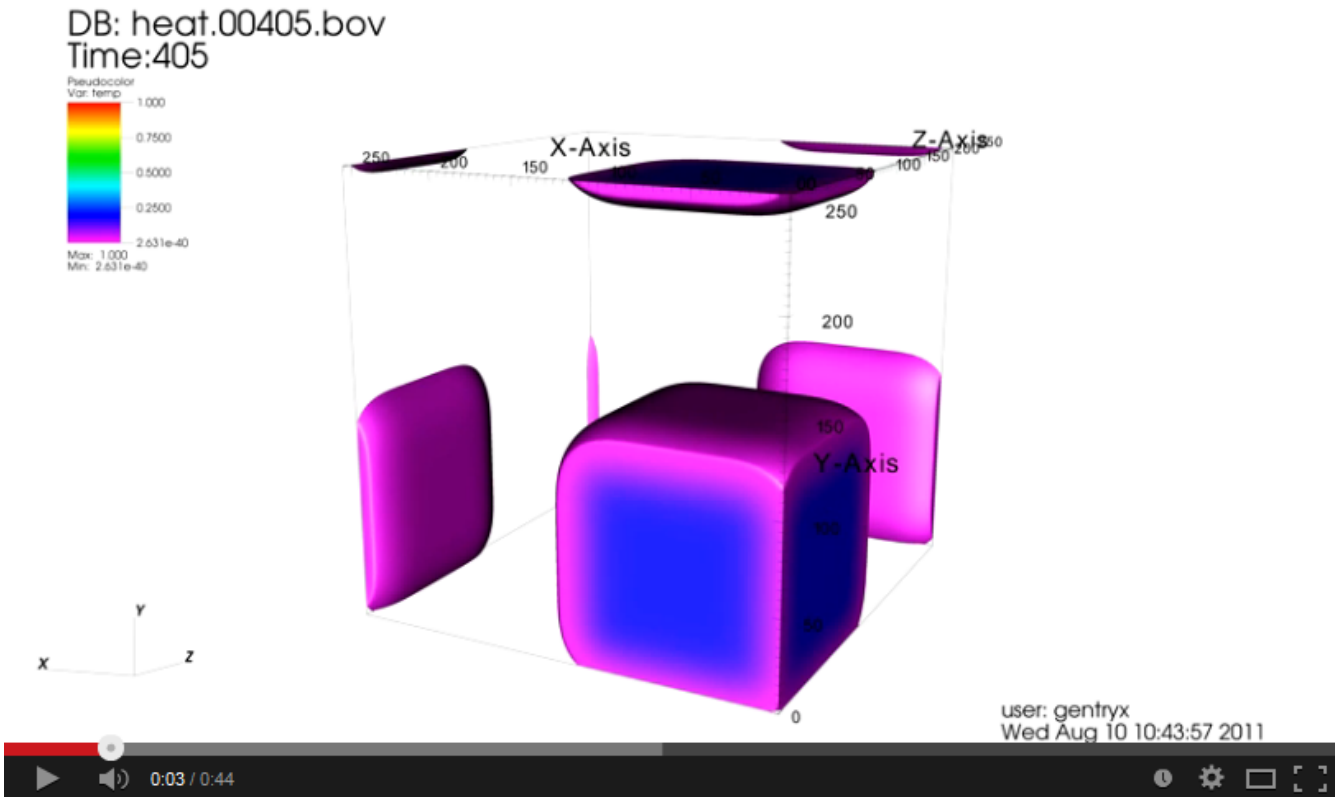
(blue: 8 threads, red: 16 threads per node)

(triangles: maximum network size, dots: simulation time)

[22] HBP Project

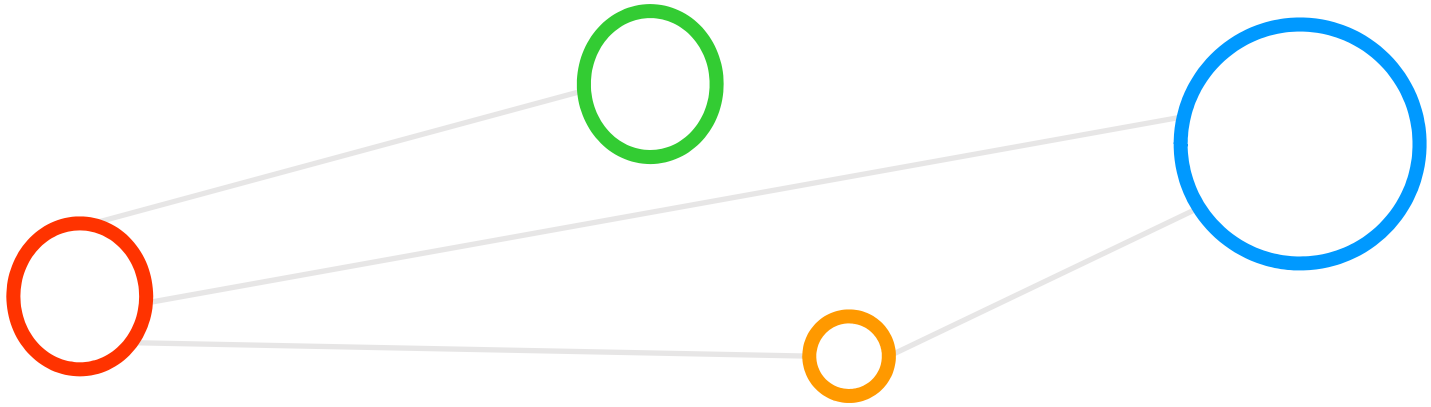
➢ **Lecture 13 will provide more details on various systems biology & bioinformatics application codes that use parallel computing**

[Video] Jacobi 3D Heat Dissipation Simulation



[13] LibGeoDecomp - Jacobi Solver (Heat Dissipation)

Lecture Bibliography



Lecture Bibliography (1)

- [1] Scalasca Flyer – Scalasca Performance Analysis Tool, Online: http://www.scalasca.org/
- [2] TotalView Debugger, Online: http://www.roguewave.com/products/totalview.aspx
- [3] Introduction to High Performance Computing for Scientists and Engineers, Georg Hager & Gerhard Wellein, Chapman & Hall/CRC Computational Science, ISBN 143981192X
- [4] YouTube Video, 'MPI/OpenMP Hybrid Programming – Getting the most from multi-core', Online: http://www.youtube.com/watch?v=TiQRPMBBmDs
- [5] M. Goetz, C. Bodenstein, M. Riedel, 'HPDBSCAN – Highly Parallel DBSCAN', in proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC2015), Machine Learning in HPC Environments (MLHPC) Workshop, 2015, Online: https://www.researchgate.net/publication/301463871_HPDBSCAN_highly_parallel_DBSCAN
- [6] K. Hwang, G. C. Fox, J. J. Dongarra, 'Distributed and Cloud Computing', Book, Online: http://store.elsevier.com/product.jsp?locale=en_EU&isbn=9780128002049
- [7] Changmin Lee, Won Woo Ro, Jean-Luc Gaudiot, 'Boosting CUDA Applications with CPU–GPU Hybrid Computing', Int J Parallel Prog (2014) 42:384–404, DOI 10.1007/s10766-013-0252-y
- [8] NVidea Tesla, Online: http://www.nvidia.de/object/tesla-high-performance-computing-de.html
- [9] AMD Radeon Instinct for HPC, Online: https://www.amd.com/en/products/servers-hpc-accelerators
- [10] YouTube Video, 'Sample jasmine 2D bubble simulation', Online: http://www.youtube.com/watch?v=pQYi9LKyI0I

Lecture Bibliography (2)

- [11] Erich Gamma et al., 'Design Patterns Elements of Reusable Object-Oriented Software', ISBN 0201633612, Prentice Hall, 1994
- [12] Wikipedia on 'stencil code', Online: http://en.wikipedia.org/wiki/Stencil_code
- [13] YouTube Video, 'LibGeoDecomp - Jacobi Solver (Heat Dissipation)', Online: http://www.youtube.com/watch?v=jBbanIGoIhE
- [14] M. Geimer et al., 'SCALASCA performance properties: The metrics tour'
- [15] German Lecture 'Umfang von MPI 1.2 und MPI 2.0'
- [16] Reed Maxwell, 'The ParFlow Hydrologic Model: HPC Highlights and Lessons Learned'
- [17] ParFlow Project, Online: http://computation.llnl.gov//casc/parflow/parflow_home.html
- [18] Community Land Model (CLM), Online: http://www.cgd.ucar.edu/tss/clm/
- [19] Lecture notes Introduction to Scientific Computing, TU Braunschweig, Online: https://www.tu-braunschweig.de/wire/lehre/skripte/index.html;jsessionid=TRIFORK661360156949
- [20] Wikipedia on 'Partial Differential Equation', Online: http://en.wikipedia.org/wiki/Partial_differential_equation

Lecture Bibliography (3)

- [21] Terrestrial Systems Simulation Lab, Online: http://www.hpsc-terrsys.de/simlab
- [22] Human Brain Project, Online: https://www.humanbrainproject.eu/de
- [23] G.Hager, MPI+OpenMP hybrid computing (on modern multicore systems), Online: http://www.speedup.ch/workshops/w39_2010/slides/hager.pdf


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