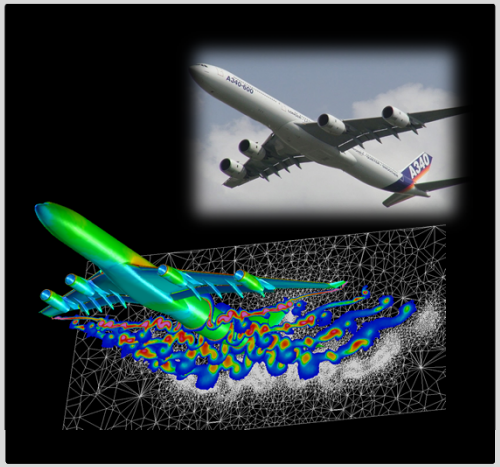


High Performance Computing

ADVANCED SCIENTIFIC COMPUTING

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

LECTURE 6

Parallel Programming with OpenMP

September 26th, 2017
Room TG-227

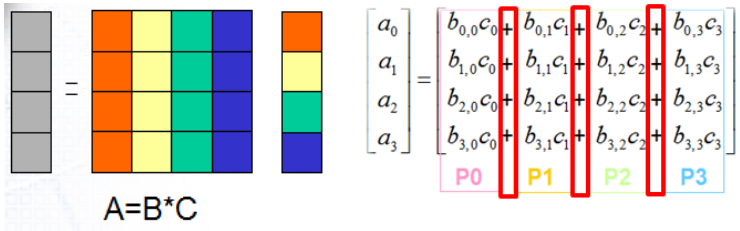
UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

JÜLICH
FORSCHUNGSZENTRUM

Review of Lecture 5 – Parallel Algorithms & Data Structures

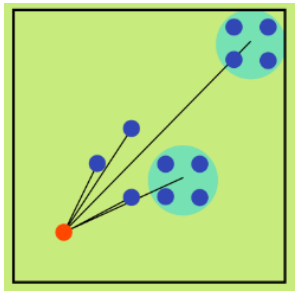
- Using MPI Collectives

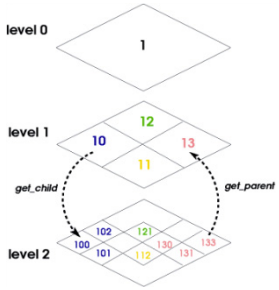

```
MPI_Scatter(x,CHUNK,MPI_DOUBLE,xpart,CHUNK,MPI_DOUBLE,0,MPI_COMM_WORLD);
MPI_Scatter(y,CHUNK,MPI_DOUBLE,ypart,CHUNK,MPI_DOUBLE,0,MPI_COMM_WORLD);

for (i=0; i<CHUNK; i++)
  zpart[i] = xpart[i] + ypart[i];

/* Collect the result */
MPI_Gather(zpart,CHUNK,MPI_DOUBLE,z,CHUNK,MPI_DOUBLE,0,MPI_COMM_WORLD);
```




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

/* Scatter vector C */
MPI_Scatter(C,1,MPI_FLOAT,Cpart,1,MPI_FLOAT, 0,MPI_COMM_WORLD);

/* Do the vector-scalar multiplication */
for(j=0; j < NCOLS; j++)
  Apart[j] = Cpart[0] * Bpart[j];

/* Reduce to matrix A */
MPI_Reduce(Apart,A,NCOLS,MPI_FLOAT,MPI_SUM, 0,MPI_COMM_WORLD);
```

- Tree-Code Approaches





- MPI Derived Datatypes

```
MPI_Type_contiguous( 3, oldtype, newtype );
```

```
MPI_Type_vector( 5, 2, 3, oldtype, newtype );
```

```
array_of_blocklengths[] = {2,3,1,2,2,2} /* below BL */
array_of_displacements[] = {3,9,12,15,18} /* below DIS */

MPI_Type_indexed ( 6, array_of_blocklengths, array_of_displacements, oldtype, newtype);
```

[1] Parallel Algorithms *[2] PEPC Webpage*

Outline of the Course

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

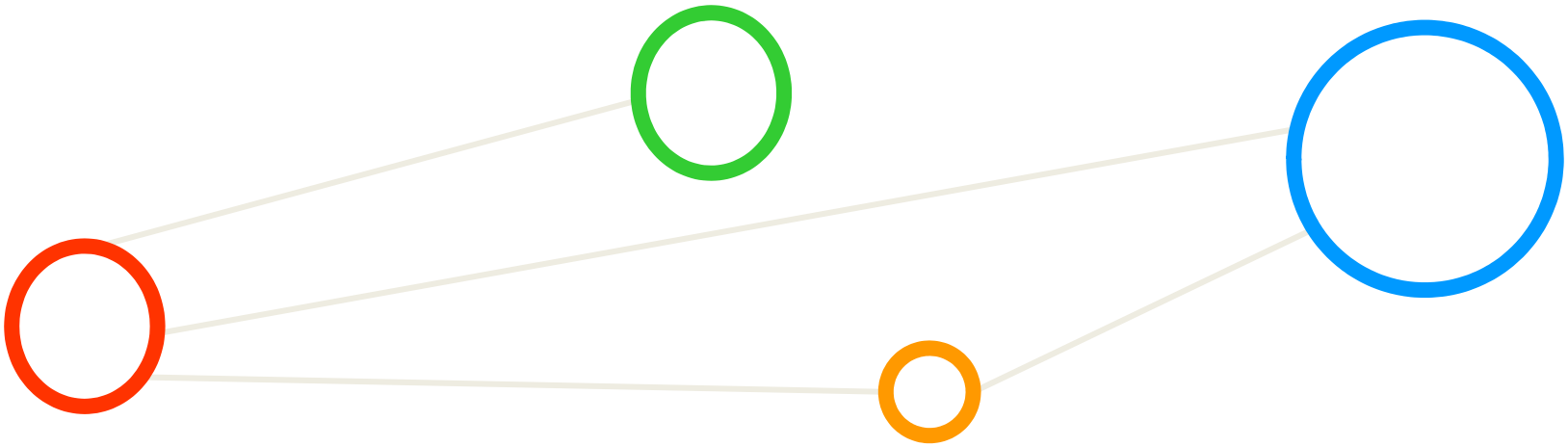
+ additional practical lectures for our hands-on exercises in context

Outline

- Shared-Memory Programming Concepts
  - Parallel and Serial Regions
  - Fork/Join & Master and Worker Threads
  - Standard & Portability
  - Hybrid Programming Motivation
  - Scientific Application Examples
- OpenMP Parallel Programming Basics
  - Basic building blocks
  - Local/shared variables & Loops
  - Synchronization & Critical Regions
  - Jacobi 2D Application Example
  - Selected Comparisons with MPI & Evolutions

- **Promises from previous lecture(s):**
- ***Lecture 1:* Lecture 6 will give in-depth details on the shared-memory programming model with OpenMP**
- ***Lecture 2:* Lecture 6 about OpenMP will include 'data parallelism on loops' methods that are useful here**
- ***Lecture 5:* Lecture 6 provides a detailed introduction to OpenMP used for shared memory programming**

Shared-Memory Programming Concepts



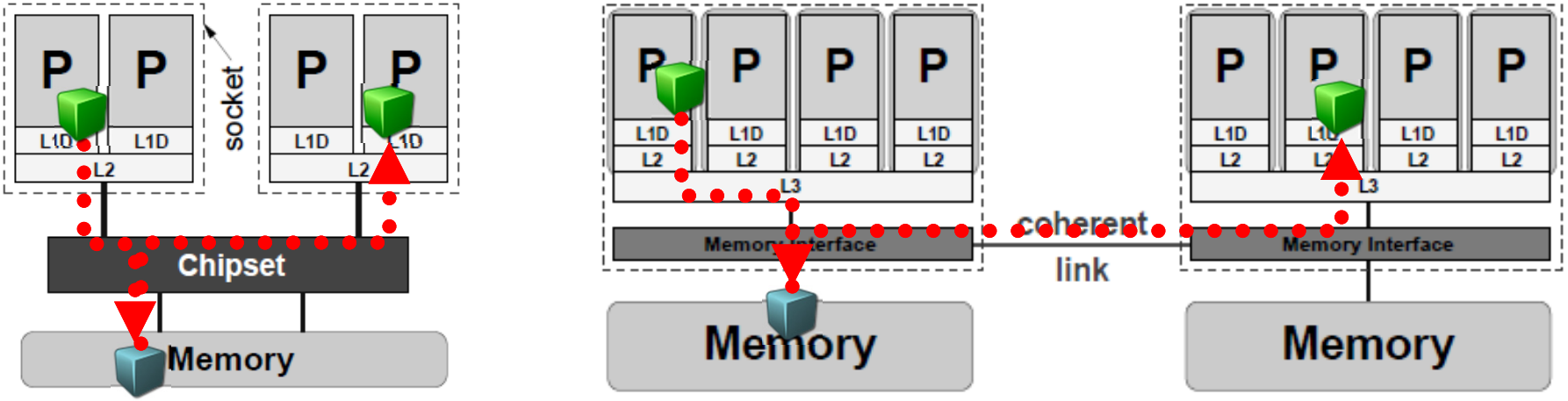
Shared-Memory Computers: Reviewed

- **A shared-memory parallel computer is a system in which a number of CPUs work on a common, shared physical address space**

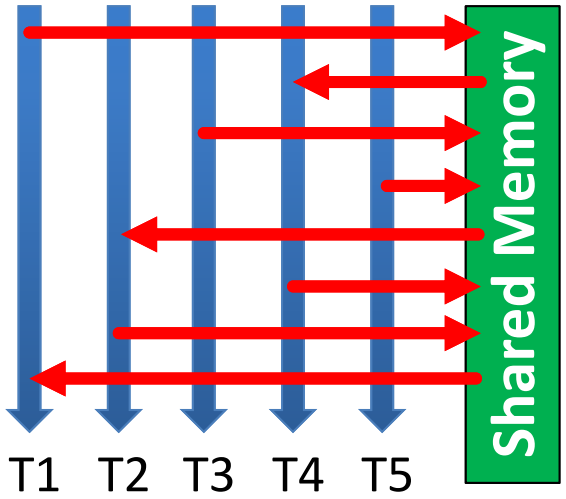
[3] Introduction to High Performance Computing for Scientists and Engineers

- Two varieties of shared-memory systems:
  1. Unified Memory Access (UMA)
  2. Cache-coherent Nonuniform Memory Access (ccNUMA)

(programming model: work on shared address space – 'local acess to memory')



Programming with Shared Memory using OpenMP



- **Shared-memory programming enables immediate access to all data from all processors without explicit communication**
- **OpenMP is dominant shared-memory programming standard today (v3)**

[4] OpenMP API Specification

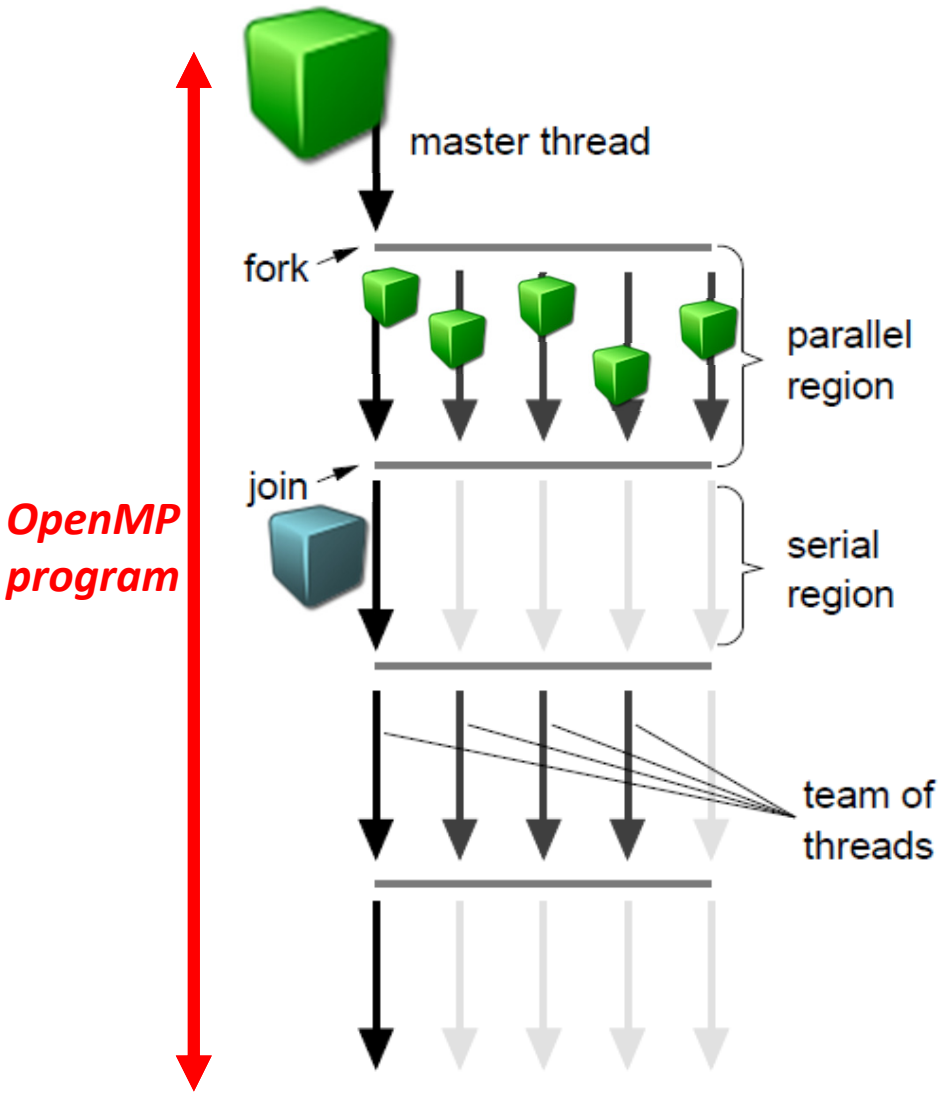
- OpenMP is a set of compiler directives to 'mark parallel regions'
- Bindings are defined for C, C++, and Fortran languages
- Threads TX are 'lightweight processes' that mutually access data
- (Shared-Memory concept itself is very old, like POSIX Threads)

What is OpenMP?

- OpenMP is a library for specifying 'parallel regions in serial code'
  - Defined by major computer hardware/software vendors → portability!
  - Enable scalability with parallelization constructs w/o fixed thread numbers
  - Offers a suitable data environment for easier parallel processing of data
  - Uses specific environment variables for clever decoupling of code/problem
  - Included in standard C compiler distributions (e.g. gcc)
- Threads are the central entity in OpenMP
  - Threads enable 'work-sharing' and share address space (where data resides)
  - Threads can be synchronized if needed
  - Lightweight process that share common address space with other threads
  - Initiating (aka 'spawning') n threads is less costly then n processes (e.g. variable space)

- **Recall 'computing nodes' are <u>independent</u> computing processors (that may also have N cores each) and that are all part of one big parallel computer**
- **Threads are lightweight processes that work with data in memory**

Parallel and Serial Regions

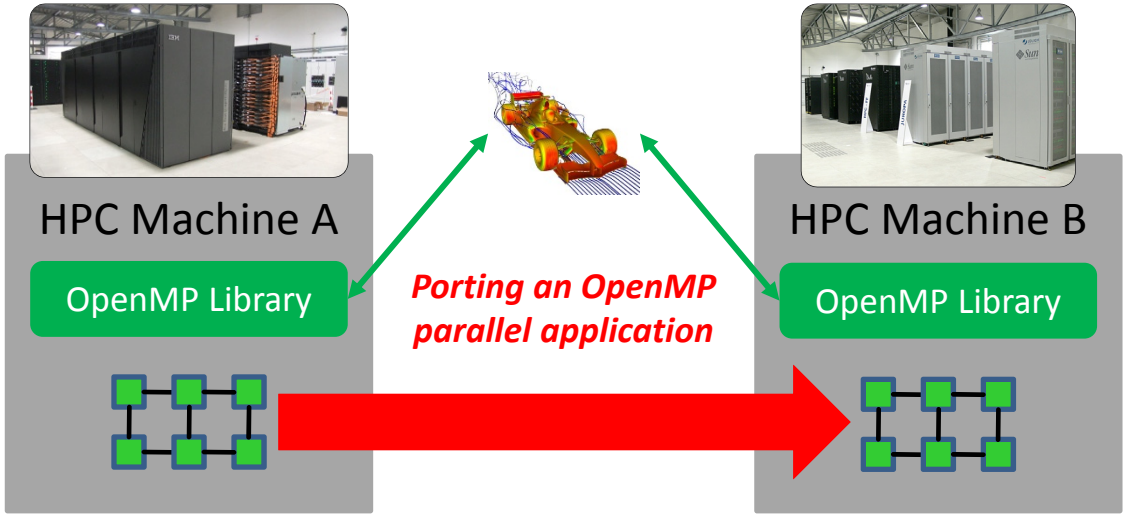


modified from [3] Introduction to High Performance Computing for Scientists and Engineers

- **fork() initiated by master thread (exists always) creates team of threads**
- **Team of threads concurrently 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 Standard enables Portability

- Key reasons for requiring a standard programming library
  - Technical advancement in supercomputers is extremely fast
  - Parallel computing experts switch organizations and face another system
- Applications using proprietary libraries where not portable
  - Create whole applications from scratch or time-consuming code updates
- OpenMP is parallel programming model for UMA and ccNUMA

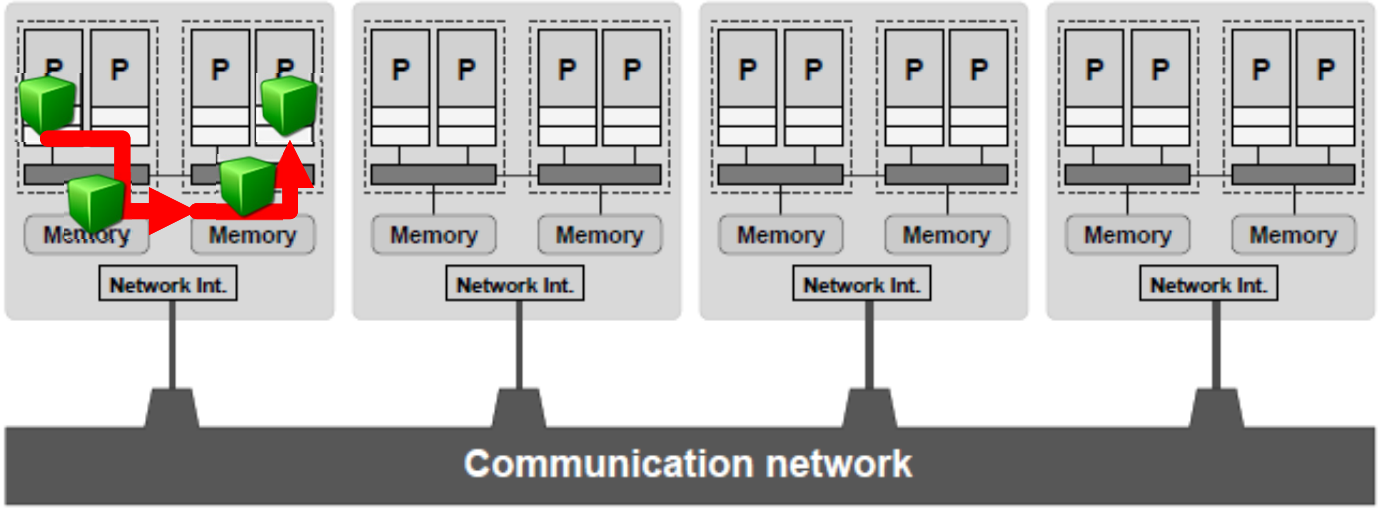


- **OpenMP is an open standard that significantly supports the portability of parallel shared-memory applications**
- **But different vendors might implement it differently**

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

MPI ?



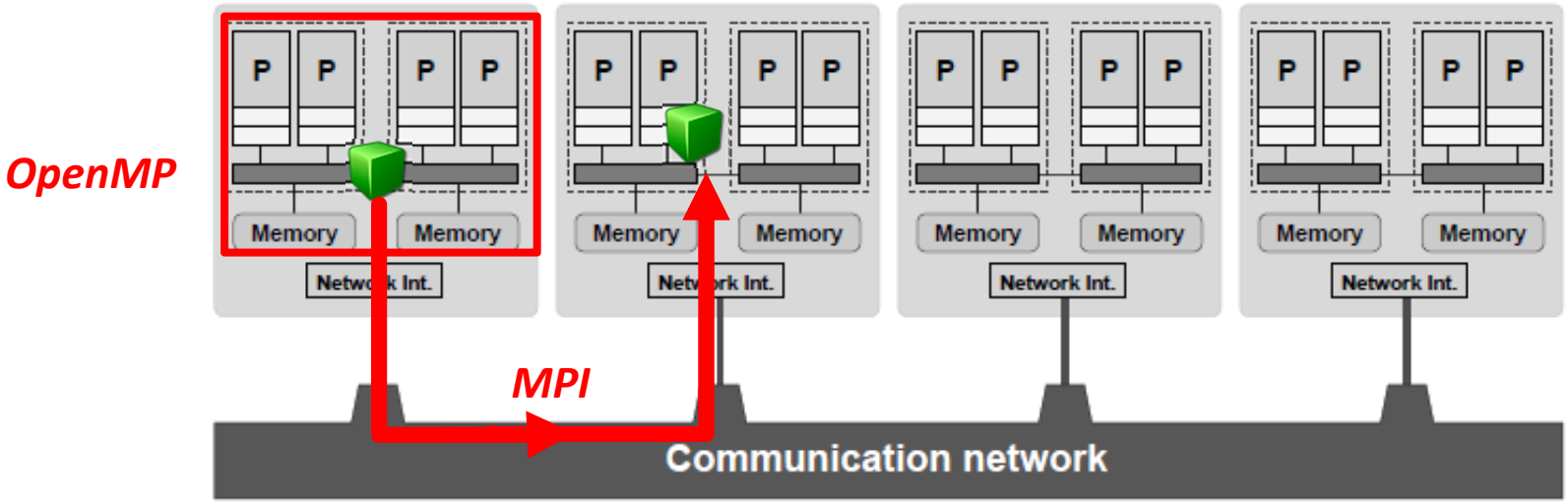
modified from [3] Introduction to High Performance Computing for Scientists and Engineers

- Take advantage of shared memory techniques where feasible
  - OpenMP threads can read memory on the same node

Hierarchical Hybrid Computers – Revisited

- **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

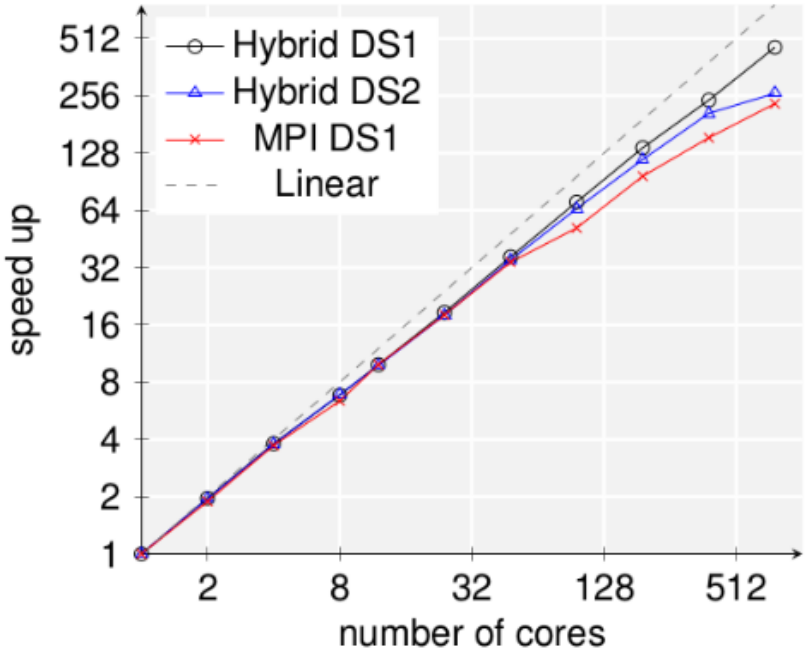


- Shared-memory nodes (i.e. ccNUMA) with local Network Int. (NI)
  - NI mediates connections to other remote 'SMP nodes'

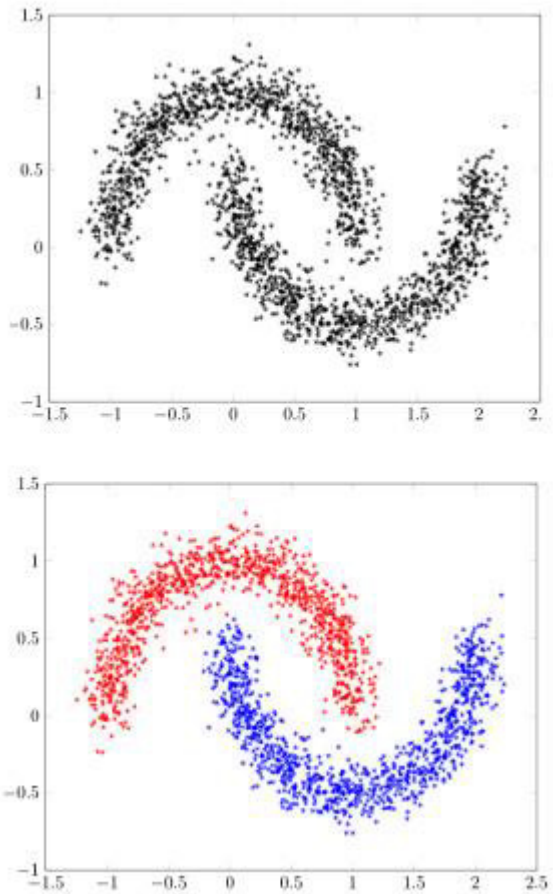
➢ **Lecture 7 offers more insights into hybrid programming jointly using both MPI and OpenMP**

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

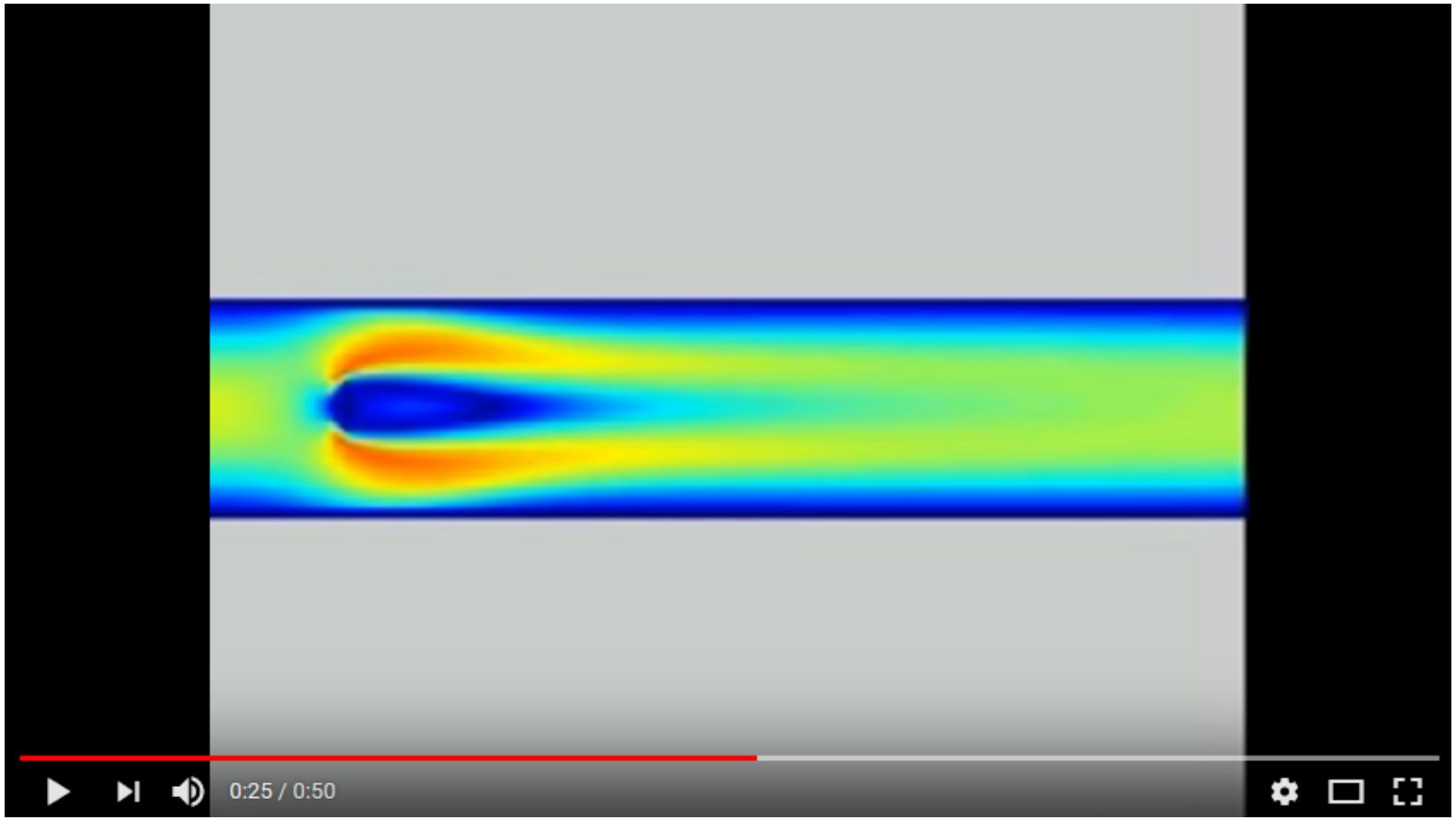


[5] M. Goetz & M. Riedel et al., 'Highly Parallel DBSCAN', MLHPC Workshop, SC2015



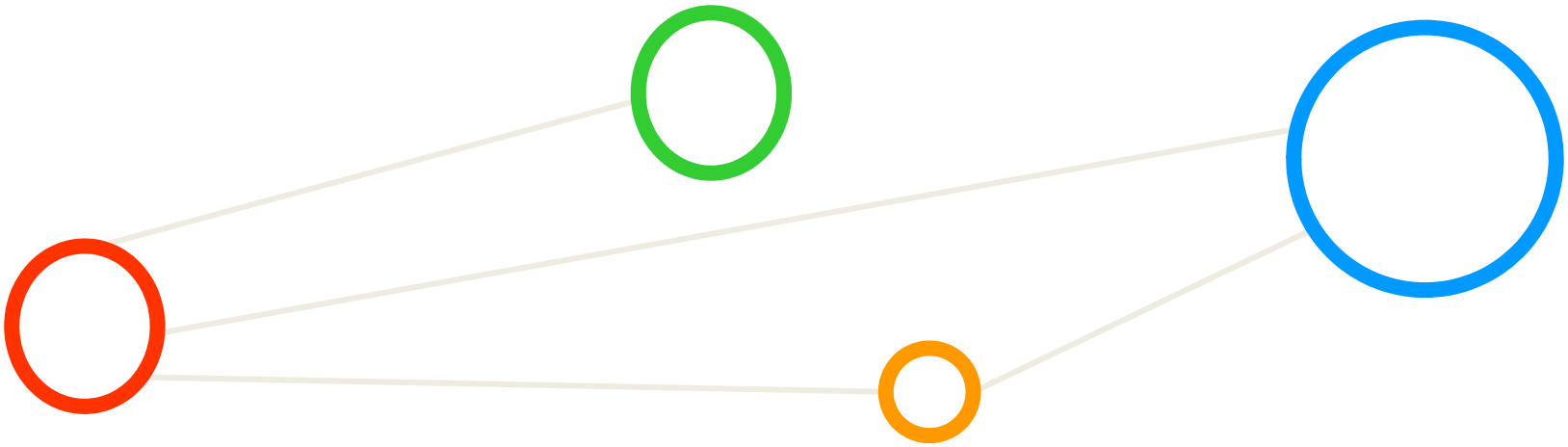
➢ **Lecture 17 gives insights into machine learning & data mining applications using OpenMP & MPI**

[Video] Scientific Application Example using OpenMP



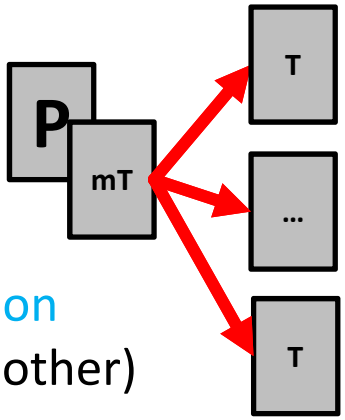
[6] Lattice Boltzmann – Flow past an obstacle, YouTube Video

OpenMP Parallel Programming Basics



Start 'Thinking' Parallel

- Parallel OpenMP programs know about the existence of a certain number of threads that all work togeter as part of a bigger picture
- OpenMP programs are written in a sequential programming language and some parts are executed in parallel
  - OpenMP programs run on a processor that 'spawns' numerous threads



- Parallelization of dedicated n parallel regions is key to the design in OpenMP (n = 1,2,3….)
  - E.g. loops/additions are good candidates for parallelization (if individual loop iterations are independent from each other)

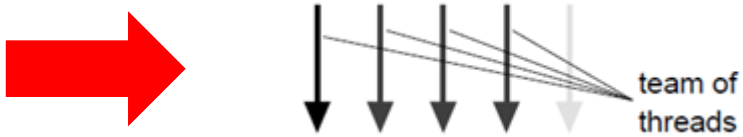
- Start with the basic building blocks using OpenMP
  - Defining code that enables 'parallel computing', step-by-step is possible

Number of Threads & Scalability

- **OpenMP programs should be always written in a way that it does not assume a specific number of threads → Scalable program**

- The real number of threads normally not known at compile time
  - (There are methods for doing it in the program → do not use them!)
  - Number is set in scripts and/or environment variable before executing

```
export OMP_NUM_THREADS=4
```

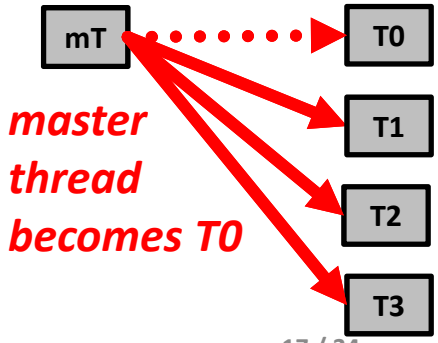


- Parallel programming is done without knowing number of threads

```
int main()
{
  #pragma omp parallel
  printf(„Hello World“);
}
```

compile & execute

```
./helloworld.exe
Hello World
Hello World
Hello World
Hello World
```



OpenMP Basic Building Blocks: Library & Sentinel

'standard fortran programming…'

```
  use omp_lib

  print *,'I am the
!$OMP PARALLEL
  call do_work_package(                                    )
!$OMP END PARALLEL
```

'standard C/C++ programming…'

```
  #include <omp.h>

  std::cout << "I am the master, and I am alone";
#pragma omp parallel
{
  do_work_package                                          );
}
```

- **The OpenMP library contains OpenMP API definitions**

- **The Sentinel is a special string that starts an OpenMP compiler directive**

- **Practice view: programming OpenMP in C/C++ and fortran is slightly different, but providing the same basic concepts (e.g. no end of parallel region in C/C++, local variables, etc.)**

OpenMP Basic Building Blocks: Unique Thread IDs

'standard fortran programming…'

```
  use omp_lib

  print *,'I am the
!$OMP PARALLEL
  call do_work_package(omp_get_thread_num(),omp_get_num_threads())
!$OMP END PARALLEL
```

'standard C/C++ programming…'

```
  #include <omp.h>

  std::cout << "I am the master, and I am alone";
#pragma omp parallel
{
  do_work_package(omp_get_thread_num(),omp_get_num_threads());
}
```

- **omp_get_thread_num() function provides unique Thread ID (0…n-1)**

- **omp_get_num_threads() function obtains number of <u>active</u> threads in the current parallel region**

- **do_work_package() routine code is now executed in parallel by each thread**
- **BUT also sub-routines of that routine are now executed in parallel**

OpenMP Basic Building Blocks: Private Variables (Fortran)

```
  integer :: bstart, bend, blen, numth, tid, i
  integer :: N
  double precision, dimension(N) :: a,b,c
  ...
!$OMP PARALLEL PRIVATE(bstart,bend,blen,numth,tid,i)
  numth = omp_get_num_threads()
  tid = omp_get_thread_num()
  blen = N/numth
  if(tid.lt.mod(N,numth)) then
    blen = blen + 1
    bstart = blen * tid + 1
  else
    bstart = blen * tid + mod(N,numth) + 1
  endif
  bend = bstart + blen - 1
  do i = bstart,bend
    a(i) = b(i) + c(i)
  enddo
!$OMP END PARALLEL
```

- **PRIVATE defines local variables for each thread**
- **Each thread <u>works independently</u> and thus needs space to 'store' local results**

- **Same code executed n times with n threads, BUT tid is unique and thus different for each thread**

- **Practice view: the real parallelization idea is here in the loop: the simple sum of two arrays**
- **For each value of i we can compute and store array values independently from each other**

Traditional HelloWorld Example (C/C++)

```
#include <omp.h>
#include <stdio.h>
int main(argc,argv)
int argc; char *argv[]; {

  int nthreads, tid;

  #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);
    }
  }
}
```

- Shared variable nthreads
- Local variable tid

- Simple Parallel Program
- Only the master (tid=0) provides output of how many threads are existing in the parallel region

➢ Practical lectures focus on MPI and not OpenMP, but both are important programming models

OpenMP Basic Building Blocks: Loops (do, for in C/C++)

```
  double precision :: pi,w,sum,x
  integer :: i,N=1000000

  pi = 0.d0
  w = 1.d0/N
  sum = 0.d0
!$OMP PARALLEL PRIVATE(x) FIRSTPRIVATE(sum)
!$OMP DO
  do i=1,n
    x = w*(i-0.5d0)
    sum = sum + 4.d0/(1.d0+x*x)
  enddo
!$OMP END DO
```

local sum exists, but where is the global sum?

```
!$OMP END PARALLEL
```

$$\pi = \int_0^1 dx \frac{4}{1+x^2}$$

- **FIRSTPRIVATE() copies initial value of shared variable to local variable (simple init here otherwise problems in loop)**

- **DO loop (in front of usual do) distributes (automatically) loop iterations among threads as specifically supported 'work-sharing' construct)**

- **Smart programming support by OpenMP: Loops are very often part of scientific applications**
- **Less burden for programmer: no manual definition of local variables (e.g. i automatically localized)**

OpenMP Basic Building Blocks: Critical Regions

```
  double precision :: pi,w,sum,x
  integer :: i,N=1000000

  pi = 0.d0
  w = 1.d0/N
  sum = 0.d0
!$OMP PARALLEL PRIVATE(x) FIRSTPRIVATE(sum)
!$OMP DO
  do i=1,n
    x = w*(i-0.5d0)
    sum = sum + 4.d0/(1.d0+x*x)
  enddo
!$OMP END DO
!$OMP CRITICAL
  pi= pi + w*sum
!$OMP END CRITICAL
!$OMP END PARALLEL
```

- Local sum exists in each of the different threads
- We have n times the local variable value for sum now

- Race Condition in shared-memory: shared variable pi will be set concurrently by the different threads
- Value of pi depends on the exact order the threads access pi and assign wrong values

- Critical regions define a region within a parallel region where at most one thread at a time executes code (e.g. sum of new pi based on pi)

OpenMP Basic Building Blocks: Reduction

```
  double precision :: r,s
  double precision, dimension(N) :: a

  call RANDOM_SEED()
!$OMP PARALLEL DO PRIVATE(r) REDUCTION(+:s)
  do i=1,N
    call RANDOM_NUMBER(r)   ! thread safe
    a(i) = a(i) + func(r)   ! func() is thread safe
    s = s + a(i) * a(i)
  enddo
!$OMP END PARALLEL DO

  print *,'Sum = ',s
```

- Several operations are common in scientific applications
- +, *, -, &, |, ^, &&, ||, max, min

- REDUCTION() with operator + on variable s enables here …
- Starting with a local copy of s for each thread
- During progress of parallel region each local copy of s will be accumulated seperately by each thread
- At the end of the parallel region automatically synchronized and accumulated with resulting master thread variable

- Reduction operations are a smart alternative to manual critical regions defintions around operations of variables
- Reduction operation automatically localizes variable

Vector Addition in OpenMP – Revisited (cf. Lecture 5)

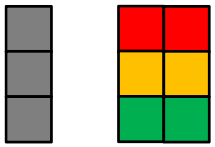
```
#include <omp.h>

int main(int argv, char **argv)
{
  int n, i;
  double *x, *y;

  /* Get input size */
  n = atoi(argv[1]);
  x = (double *)malloc(n*sizeof(double));
  y = (double *)malloc(n*sizeof(double));

  #pragma omp parallel for private(i) shared (x,y)
  for (i=0; i<n; i++)
  {
    x[i] = x[i] + y[i];
  }

  /* x containts the result for all vector elements */
}
```

'simplified demo code'

- The Sentinel 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 <u>works independently</u> and thus needs space to 'store' local results – here i as index

- SHARED defines global variables that exist only one time
- Each thread <u>works independently</u> but SHARED variables can be written and read from all threads



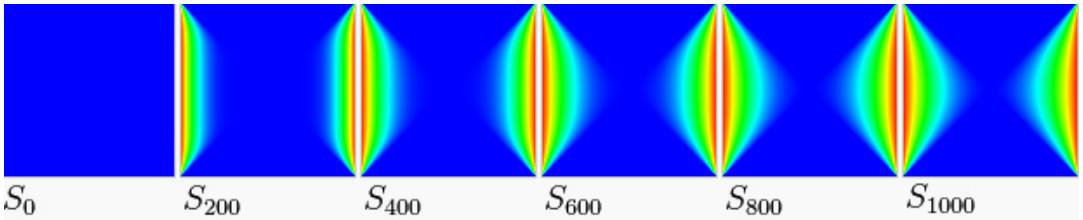
X = X+Y

Jacobi 2D Application Example – Revisited (cf. Lecture 2)

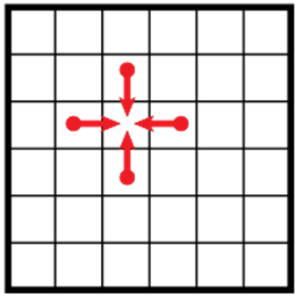
- **The Jacobi iterative method is a stencil-based iterative method used in numerical linear algebra**
- **Algorithm for determining the solutions of diagonally dominant system of linear equations**

- 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)

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



[7] Wikipedia on 'stencil code'



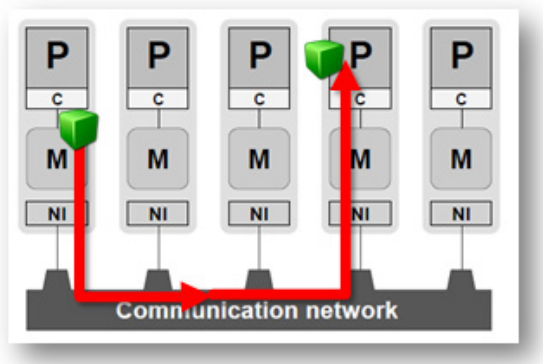
➢ **Lecture 7 will provide more details about stencil-based iterative methods and used patterns**

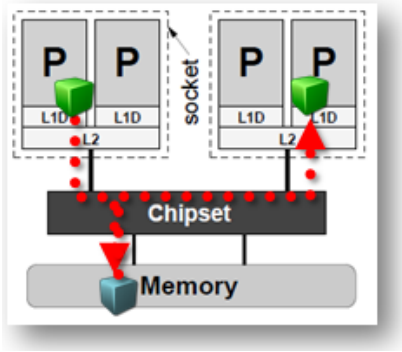
Advanced Example: 2D Jacobi Algorithm with OpenMP

```
double precision, dimension(0:N+1,0:N+1,0:1) :: phi
double precision :: maxdelta,eps
integer :: t0,t1
eps = 1.d-14        ! convergence threshold
t0 = 0 ; t1 = 1
maxdelta = 2.d0*eps
do while(maxdelta.gt.eps)
  maxdelta = 0.d0
!$OMP PARALLEL DO REDUCTION(max:maxdelta)
  do k = 1,N
    do i = 1,N
      ! four flops, one store, four loads
      phi(i,k,t1) = (  phi(i+1,k,t0) + phi(i-1,k,t0)
                     + phi(i,k+1,t0) + phi(i,k-1,t0) ) * 0.25
      maxdelta = max(maxdelta,abs(phi(i,k,t1)-phi(i,k,t0)))
    enddo
  enddo
!$OMP END PARALLEL DO
  ! swap arrays
  i = t0 ; t0=t1 ; t1=i
enddo
```

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

Selected Comparisons with MPI





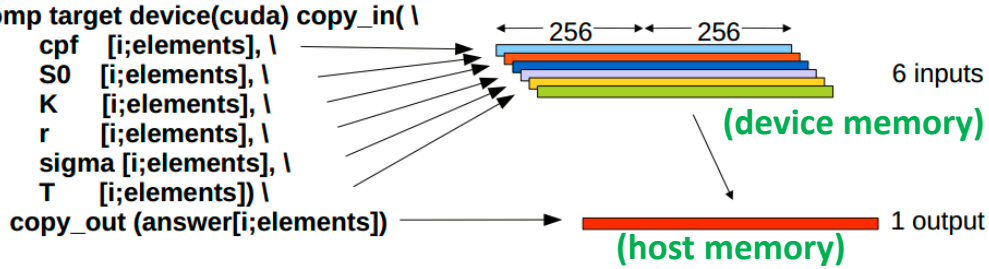
- Some aspects are similar, because both enable parallel computing
  - Obtaining unique IDs: MPI ranks vs. OpenMP thread-num
  - Master-worker approach (if rank==0 vs. if tid ==0)
- No explicit communication constructs to enable inter-process communication in OpenMP → assuming shared-memory
  - Data exchange: Message exchanges between processes vs. shared variable
  - Synchronization functions nevertheless exist in both: e.g. barriers
  - Clever automatisms for usual problems: MPI reduce vs. OpenMP reduction

DEEP-EST EU Project – OmpSs & OpenMP Evolutions

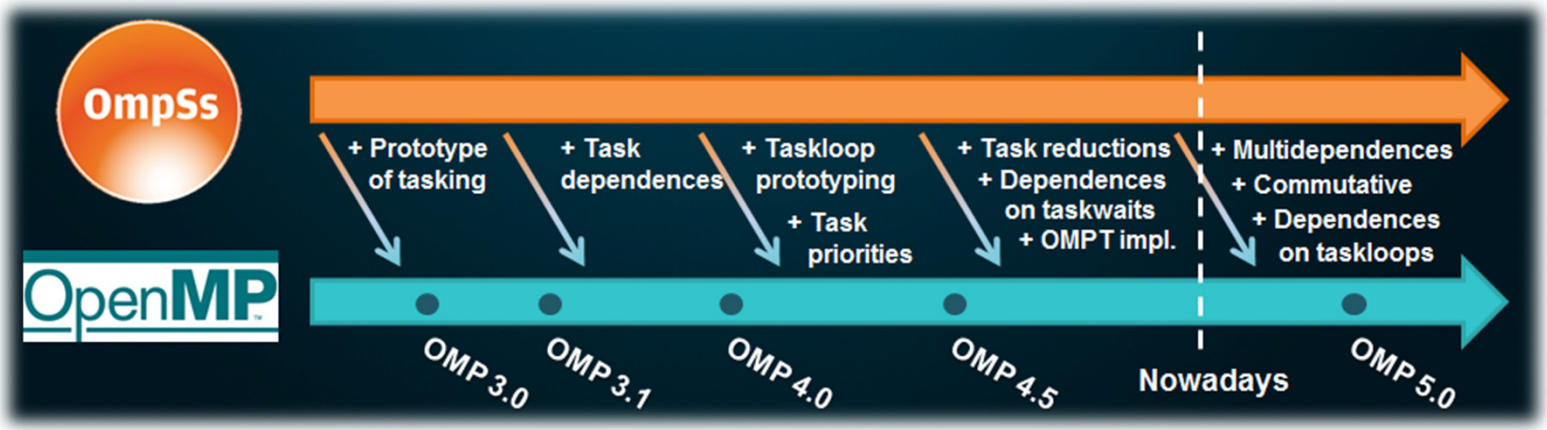
[9] DEEP-EST EU Project

[10] OmpSs Web page

```
#pragma omp target device(cuda) copy_in( \
        cpf    [i;elements], \
        S0     [i;elements], \
        K      [i;elements], \
        r      [i;elements], \
        sigma  [i;elements], \
        T      [i;elements]) \
copy_out (answer[i;elements])
```

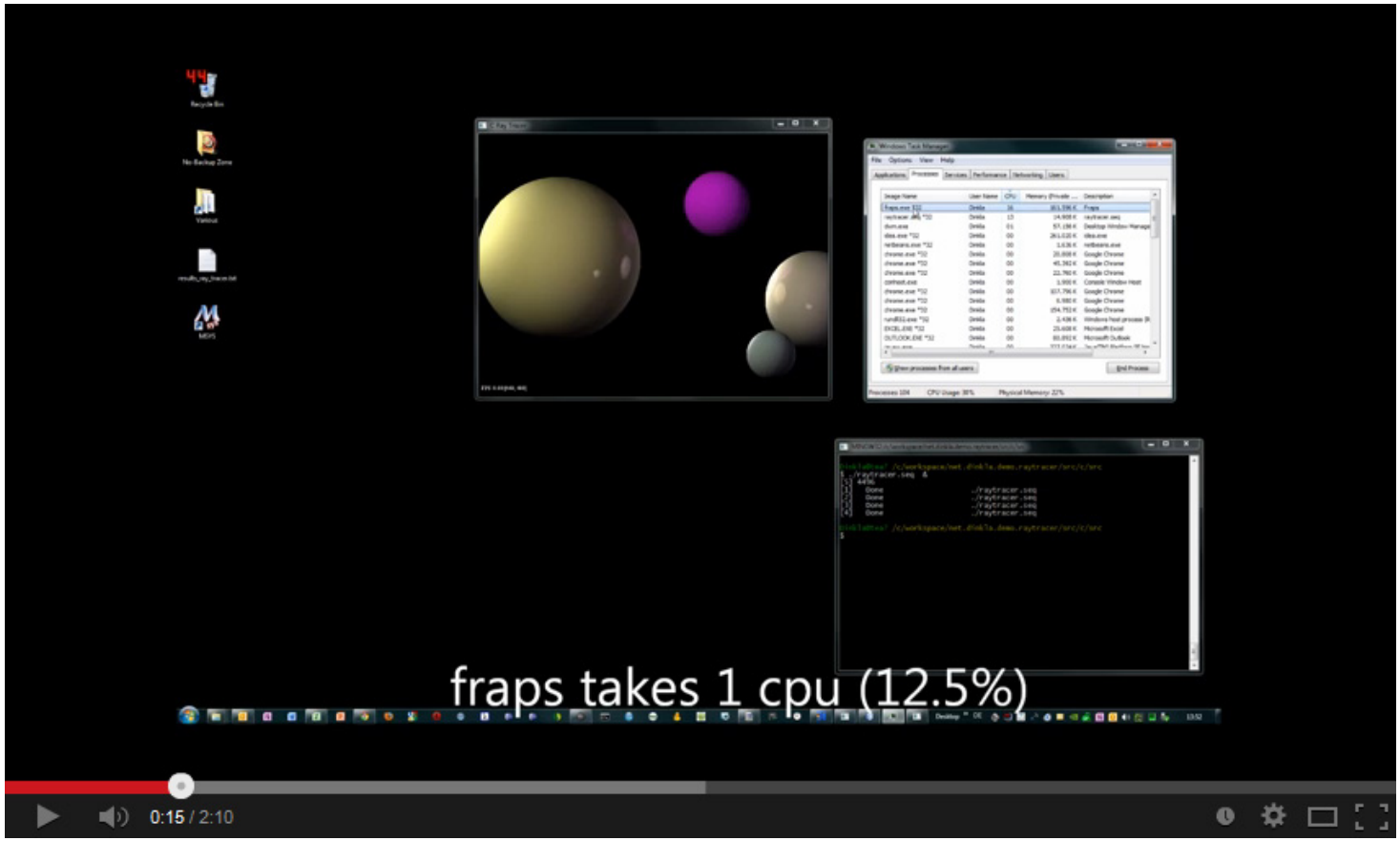


- OmpSs is an innovative programming model influencing OpenMP
  - Based on tasks and (data) dependencies – tasks as elementary unit of work
  - Extend OpenMP model: better data-flow & heterogenity (e.g. GPGPUs)



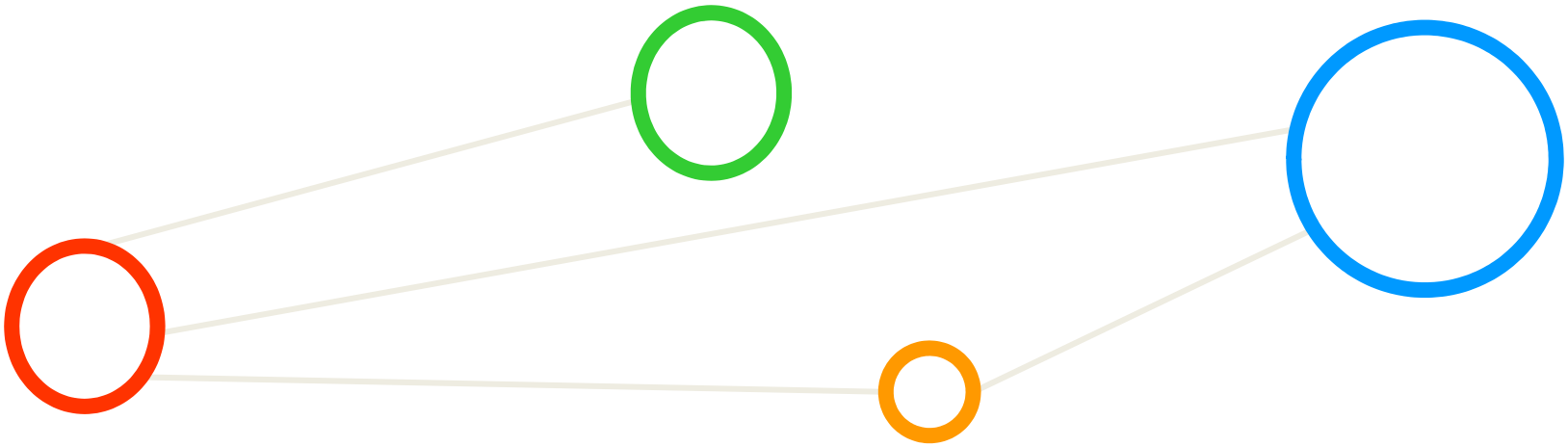
➢ **Lecture 10 will provide more details about the use of accelerators and the benefits of GPGPUs**

[Video] Raytracing Application with OpenMP



[8] Speeding up a Ray tracer with OpenMP, YouTube Video

Lecture Bibliography



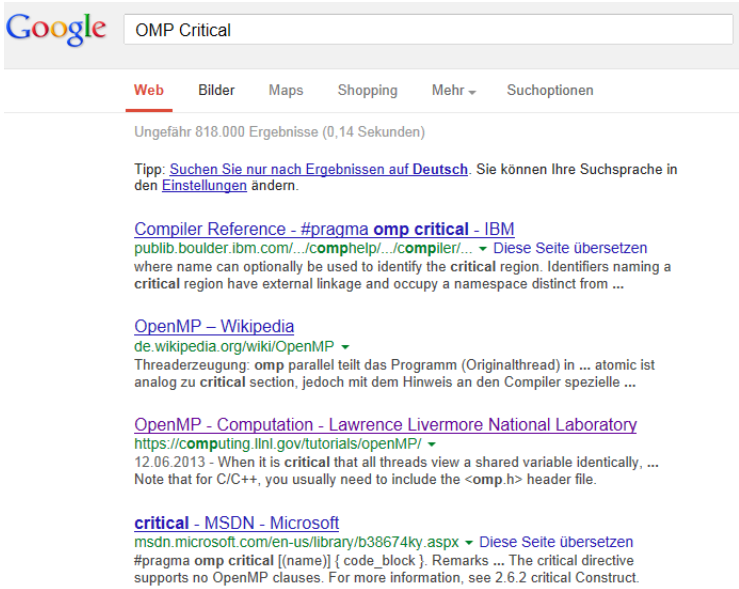
Lecture Bibliography

- [1] Parallel Algorithms Underlying MPI Implementations, Online: http://www.slidefinder.net/p/parallel_algorithms_underlying_mpi_implementations/13-parallelalgorithmsunderlyingmpiimplementations/17131238/p2
- [2] PEPC, FZ Juelich, Online: http://www.fz-juelich.de/ias/jsc/EN/AboutUs/Organisation/ComputationalScience/Simlabs/slpp/SoftwarePEPC/_node.html
- [3] Introduction to High Performance Computing for Scientists and Engineers, Georg Hager & Gerhard Wellein, Chapman & Hall/CRC Computational Science, ISBN 143981192X
- [4] The OpenMP API specification for parallel programming, Online: http://openmp.org/wp/openmp-specifications/
- [5] M. Goetz, C. Bodenstein, M. Riedel, '*HPDBSCAN – Highly Parallel DBSCAN',* Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC2015), Machine Learning in HPC Environments (MLHPC) Workshop, USA, Online: https://dl.acm.org/citation.cfm?doid=2834892.2834894
- [6] Lattice Boltzmann – Flow past an obstacle, Online: https://www.youtube.com/watch?v=fspGcBpxguo
- [7] Wikipedia on 'stencil code', Online: http://en.wikipedia.org/wiki/Stencil_code
- [8] Speeding up a Ray tracer with OpenMP, YouTube Video, Online: http://www.youtube.com/watch?v=S9Z5MeQS_LU
- [9] DEEP-EST EU Project, Online: http://www.deep-projects.eu/
- [10] OmpSs Programming Model, Online: https://pm.bsc.es/ompss-docs/specs/01_introduction.html

Useful Collection on OpenMP Tutorials/Materials

- YouTube – ‘PRACE Video Tutorial – Introduction to OpenMP’, Online: http://www.youtube.com/watch?v=LyKA77PEM3o
- Introduction to OpenMP, HLRS Stuttgart, Online: https://fs.hlrs.de/projects/par/par_prog_ws/pdf/openmp-intro7.pdf
- Using OpenMP – Portable Shared Memory Parallel Programming, B. Chapman, G. Jost, R. Van der Pas (2008)
- Parallel Programming in OpenMP R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon (2001)
- LLNL OpenMP Tutorial, Online: https://computing.llnl.gov/tutorials/openMP/

- **And if you don‘t find something:**
  ‚just google the OpenMP compiler directive or environment variable in question – good resources are out there‘:




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