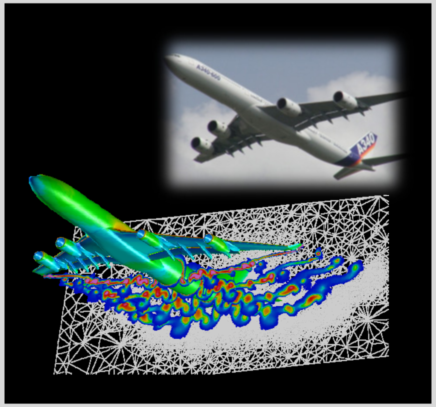


High Performance Computing

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

@Morris Riedel @MorrisRiedel @MorrisRiedel

PRACTICAL LECTURE 6.1

Understanding OpenMP Parallel Programming

October 14, 2019
Webinar

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

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

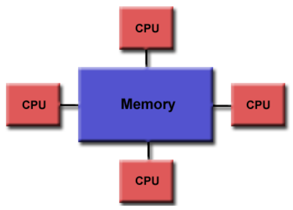
DEEP Projects

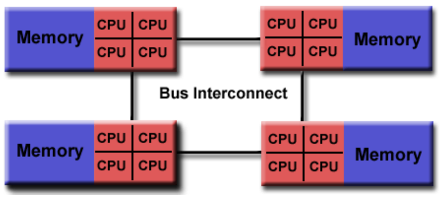
HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

HAICU HELMHOLTZ ARTIFICIAL INTELLIGENCE COOPERATION UNIT

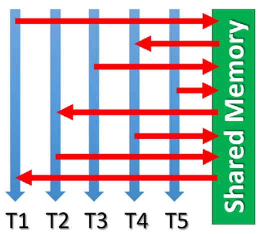
Review of Practical Lecture 6 – Parallel Programming with OpenMP

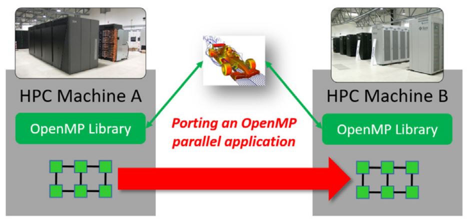
- Shared Memory Programming Approach

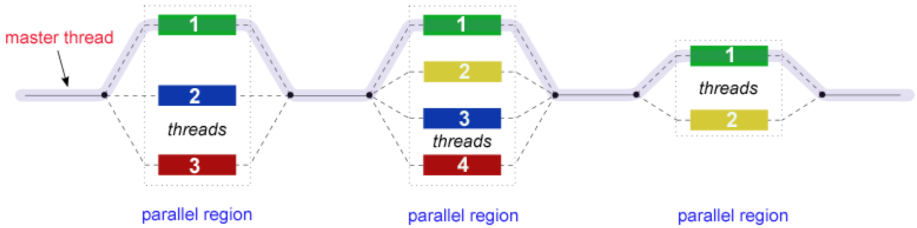




(uniform memory access) (non-uniform memory access)

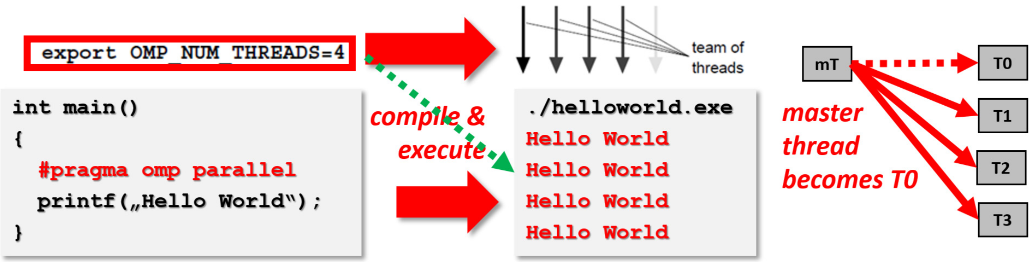


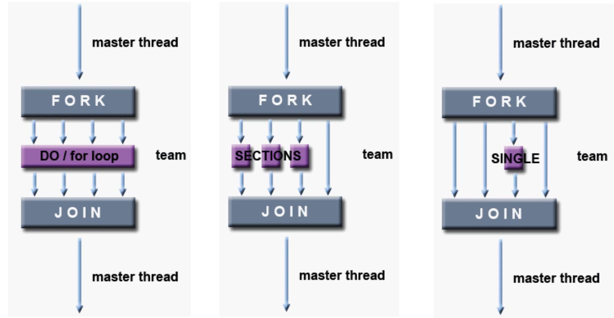




[1] LLNL OpenMP Tutorial *[2] OpenMP API Specification*

- Open Standard

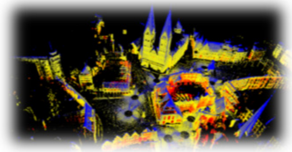




(work sharing constructs)

```
#!/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
```

(hybrid MPI & OpenMP job script for best peformance)



```
// hpdbscan.h file
...
#include <hdf5.h>
#include <omp.h>
...
// local DBSCAN run
#pragma omp parallel for schedule(dynamic, 32) private(neighboring_points)
    firstprivate(previous_cell) reduction(merge: rules)
   for (size_t point = lower; point < upper; ++point) {
...
Clusters cluster(Dataset& dataset, int threads=omp_get_max_threads()) {
```

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

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

- Programming & Compiling C-based OpenMP Programs
  - Shared Memory & Parallel Programming – Revisited
  - Step-Wise Walkthrough for Programming a C & OpenMP Program
  - Parallel Environment Setup & Number of Threads for Application
  - Simple Application Example with OpenMP Compiler Directives
  - Fine-grained Job Script Request & Allocation of Compute Resources

- Understanding OpenMP Work Sharing Constructs & Methods
  - OpenMP Work Sharing Constructs – Revisited
  - Simple Application Example with OpenMP For Loop Work Sharing
  - OpenMP Synchronization Construct & Simple Critical Region Example
  - Advanced Examples: ThreadPrivate & Persistence between Parallel Regions
  - HPDBSCAN Clustering with OpenMP Data Science Example

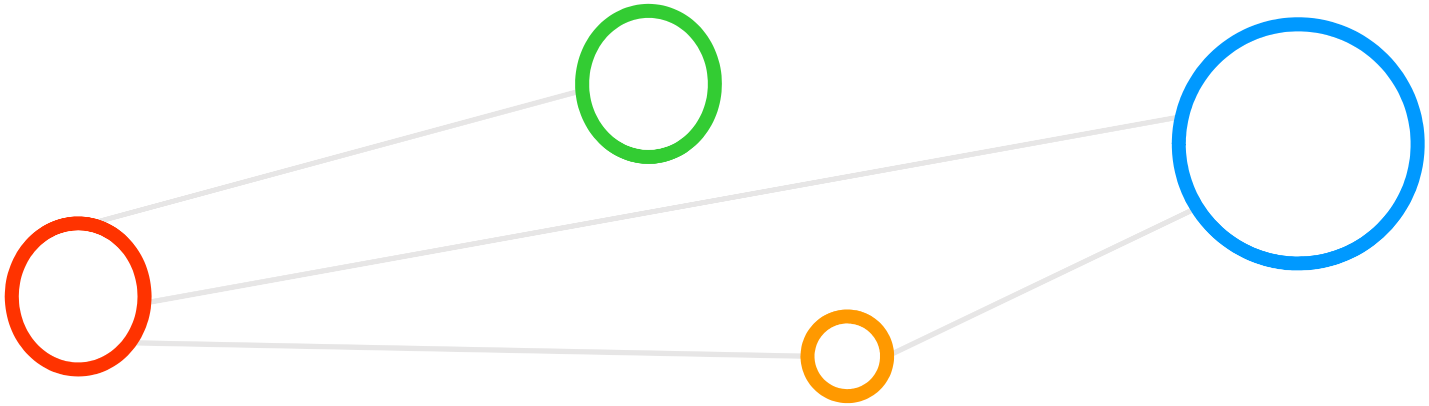
- **This lecture is not considered to be a full introduction to OpenMP programming and the approach of using shared memory and rather focusses on selected commands and concepts particularly relevant for our assignments such as OpenMP Sentinels and selected OpenMP functionality, e.g., for loops & work sharing constructs**
- **The goal of this practical lecture is to make course participants aware of the process of compiling simple C & OpenMP programs and the use of OpenMP that enable many scientific & engineering applications in data sciences & simulation sciences today**

Selected Learning Outcomes – Revisited

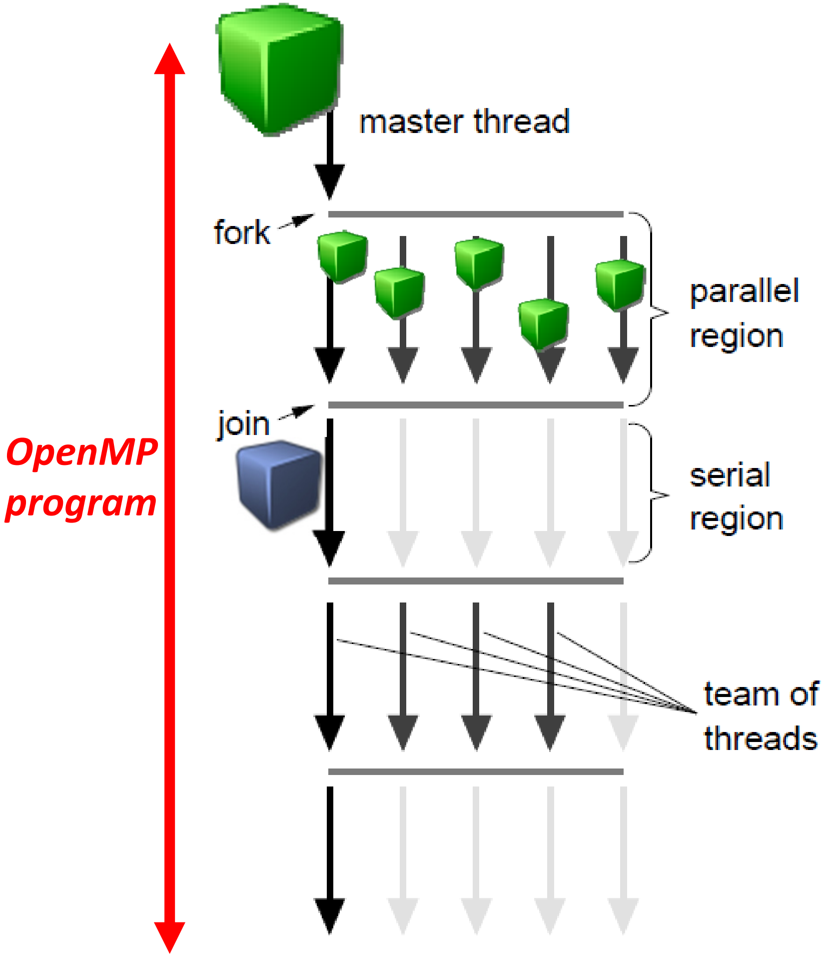
- Students understand…
  - Latest developments in parallel processing & high performance computing (HPC)
  - How to create and use high-performance clusters
  - What are scalable networks & data-intensive workloads
  - The importance of domain decomposition
  - Complex aspects of parallel programming → e.g., scheduling(!)
  - HPC environment tools that support programming or analyze behaviour
  - Different abstractions of parallel computing on various levels
  - Foundations and approaches of scientific domain-specific applications

- Students are able to …
  - Programm and use HPC programming paradigms
  - Take advantage of innovative scientific computing simulations & technology
  - Work with technologies and tools to handle parallelism complexity

Programming & Compiling C-based OpenMP Programs



Important Terminology



- **Thread: An execution entity with a stack and associated static memory, called thread private memory**
- **OpenMP Thread: A thread that is managed by the OpenMP runtime system**
- **Team: A set of one or more threads participating in the execution of a parallel region**
- **Task: A specific instance of executable code and its data environment that the OpenMP imlementation can schedule for execution by threads**
- **Base Language: A programming language that serves as the foundation of the OpenMP specification**
- **Base Program: A program written in the base language**
- **OpenMP Program: A program that consists of a base program that is annotated with OpenMP directives or that calls OpenMP API runtime library routines.**
- **Directive: In C/C++, a #pragma that specifies OpenMP program behavior**

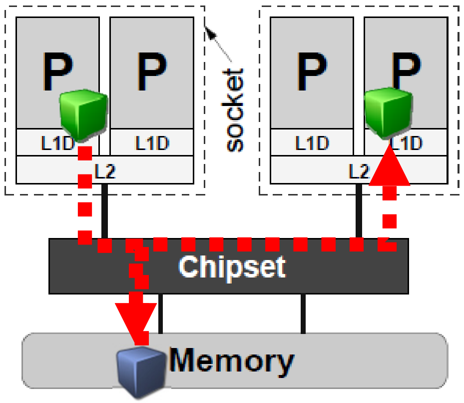
[2] OpenMP API Specification

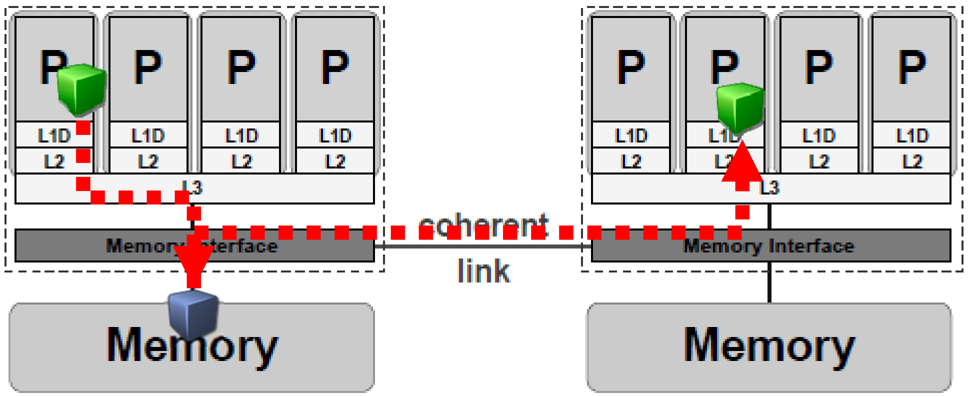
What means a ‘Shared Address Space’ (cf. Lecture 6)?

- Shared-memory programming enables immediate access to all data from all processors without explicit communication
- OpenMP is dominant shared-memory programming standard today
- OpenMP is a set of compiler directives to ‘mark parallel regions’

[2] OpenMP API Specification

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





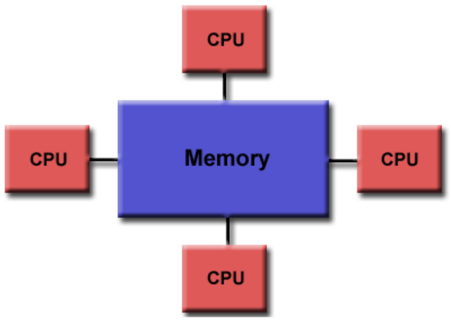
Modified from [3] Introduction to High Performance Computing for Scientists and Engineers

Programming with Shared Memory using OpenMP – Revisited (cf. Lecture 1)

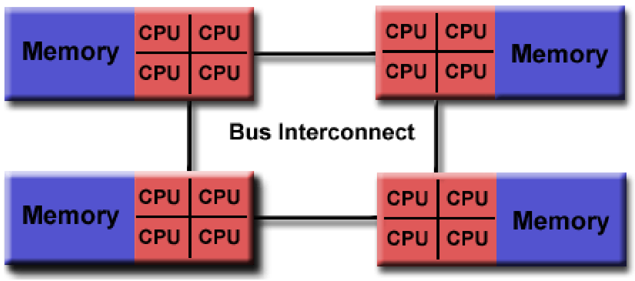
- **Shared-memory programming enables immediate access to all data from all processors without explicit communication**
- **OpenMP is dominant shared-memory programming standard today**
- **OpenMP is a set of compiler directives to 'mark parallel regions'**

[2] OpenMP API Specification

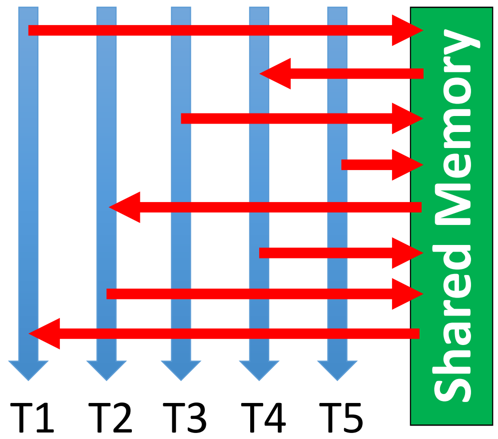
- Features
  - Bindings are defined for C, C++, and Fortran languages
  - Threads TX are 'lightweight processes' that mutually access data



(uniform memory access)

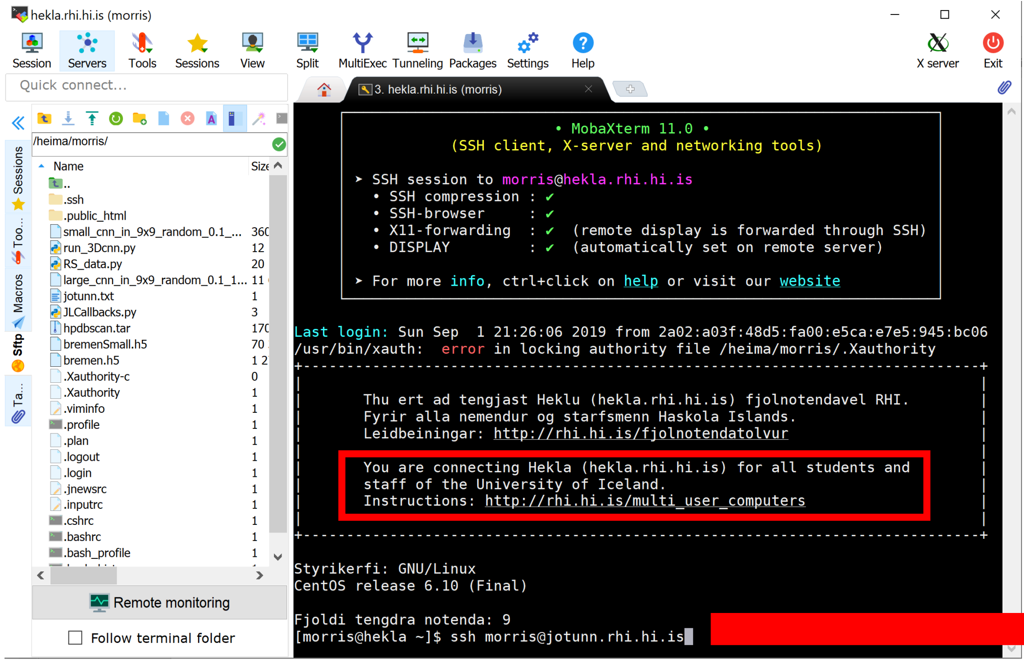


(non-uniform memory access)



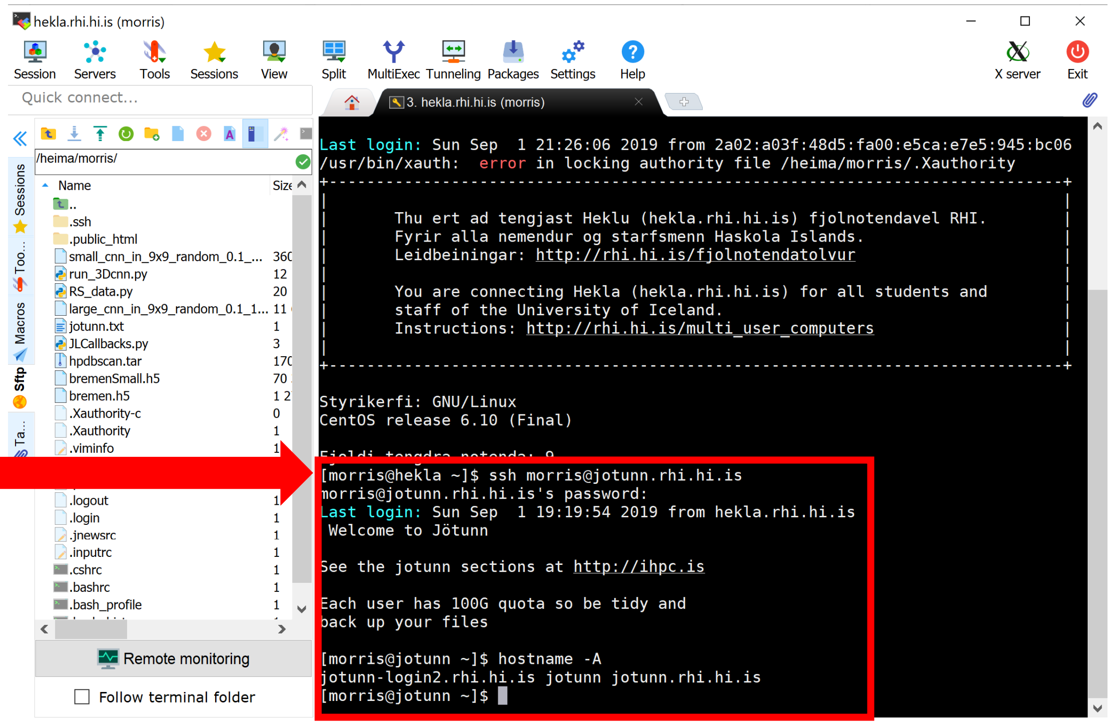
[1] LLNL OpenMP Tutorial

Step 1: SSH Access to HPC System – Jötunn HPC System Example



Hekla System

Jötunn HPC System



Step 2 & 3: C & OpenMP Basic Building Blocks: Hello World Example

```
#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);
  }
 }
}
```

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

- **Shared variable nthreads; local variable tid**

- **The Sentinel is a special string that starts an OpenMP compiler directive: '#pragma omp'**
- **private defines local variables for each thread**
- **Each thread works independently and thus needs space to 'store' private local results**

- **omp_get_thread_num() function provides unique Thread ID (0…n-1)**
- **Same code executed n times with n threads, but tid is unique and thus different for each thread**

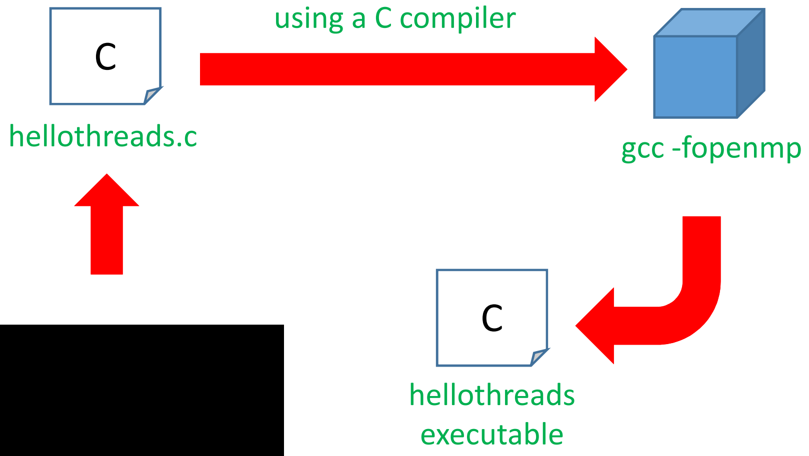
- **Similar like MPI ranks, here the Thread ID can be used to perform different executions per threads**
- **Only the master (tid=0) provides output of how many threads are existing in the parallel region**

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

Step 4 & 5: Load Modules (if needed) & Compilation

- Using basic gcc compiler
  - Load Modules (if needed)
  - Note: there are many C compilers available, we here pick one for our particular HPC course that works with OpenMP
  - Note: If there are no errors, the file hellothreads is now a full C program executable that can be started by an OS having OpenMP directives

```
[morris@jotunn hellothreads]$ gcc -fopenmp -o hellothreads hellothreads.c
[morris@jotunn hellothreads]$ ls -al
total 16
drwxrwxr-x 2 morris morris   46 okt  9 20:08 .
drwxrwxr-x 3 morris morris   25 okt  9 20:04 ..
-rwxrwxr-x 1 morris morris 8844 okt  9 20:08 hellothreads
-rw-rw-r-- 1 morris morris  275 okt  9 20:05 hellothreads.c
```




```c
#include <stdio.h>
#include <omp.h>
int main(void)
{
  printf("Hello from your main thread.\n");

  #pragma omp parallel
  printf("Hello from thread %d of %d.\n", omp_get_thread_num(), omp_get_num_threads());

  printf("Hello again from your main thread.\n");
  return 0;
}
```

[5] Icelandic HPC Machines & Community

Step 6: Parallel Processing – Executing an MPI Program with MPIRun & Script

- Submission using the Scheduler
  - Example: SLURM on Jötunn HPC system
  - Scheduler allocated 1 node as requested
  - MPIRun & scheduler distribute the executable on the right node (can be used with srun, sometimes performance differences)
  - Output consists of the amount of outputs given by number of threads in environment variable (OMP_NUM_THREADS)

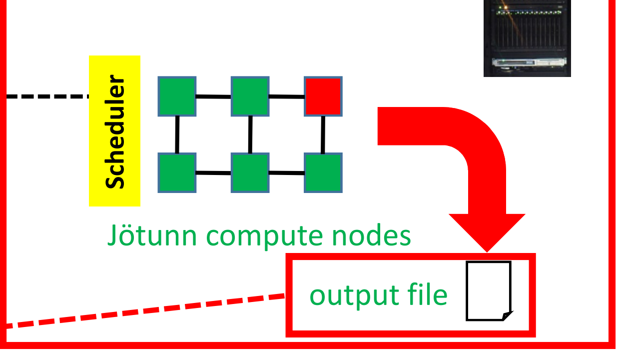
- **#SBATCH --ntasks-per-node=1 specifies just one process (e.g., 4 means four MPI ranks on one node)**
- **#SBATCH --nodes=1 specifies number of overall compute nodes**
- **#SBATCH --ntasks=1 specifies number of instances command is executed**
- **#SBATCH --cpus-per-task specifies how many CPUs each task can use**
- **Export OMP_NUM_THREADS specificies number of threads/process**

```
#!/bin/bash
#SBATCH --job-name=hellothreads
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4
#SBATCH --time=00:20:00

# load modules
module load gnu/5.3.0

# set openmp threads
export OMP_NUM_THREADS=4

srun /home/morris/2019-HPC-Course/openmp/hellothreads/hellothreads
```

```
[morris@jotunn hellothreads]$ sbatch submit-hellothreads.sh
Submitted batch job 200210
```

Jötunn login node

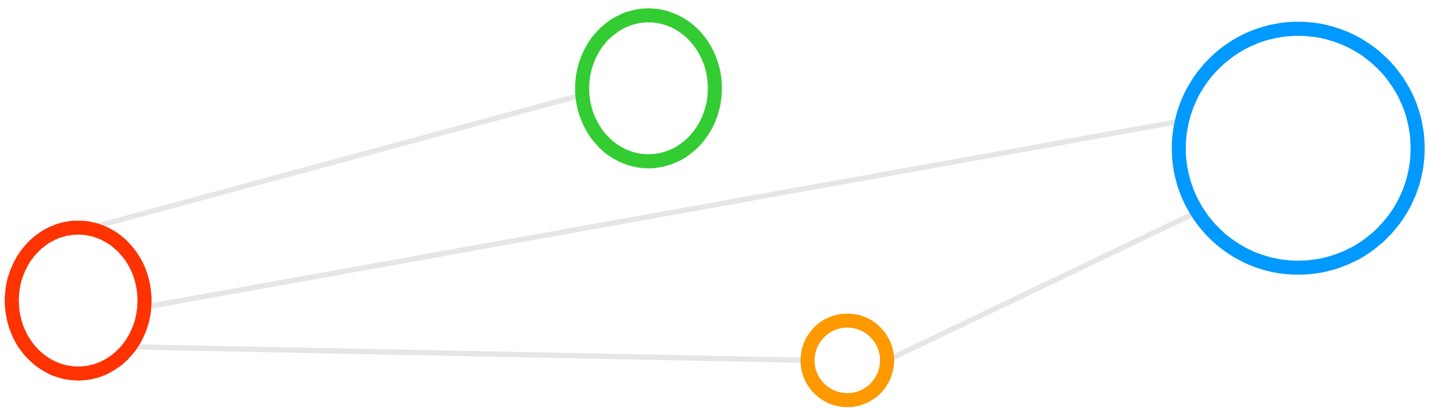
```
[morris@jotunn hellothreads]$ qstat
Job id              Name             Username        Time Use S Queue
------------------- ---------------- --------------- -------- - -------
199590              hello-mpi-exampl jfb3            00:00:00 Q normal
199591              hello-mpi-exampl jfb3            00:00:00 Q normal
199592              hello-mpi-exampl jfb3            00:00:00 Q normal
199593              hello-mpi-exampl jfb3            00:00:00 Q normal
199595              pingpong         jfb3            00:00:00 Q normal
199596              pingpong         jfb3            00:00:00 Q normal
199840              pingpong         jfb3            00:00:00 Q normal
199841              pingpong         jfb3            00:00:00 Q normal
200207              HPDBSCAN-openmp  morris          00:00:37 R normal
200210              hellothreads     morris          00:00:00 C normal
```



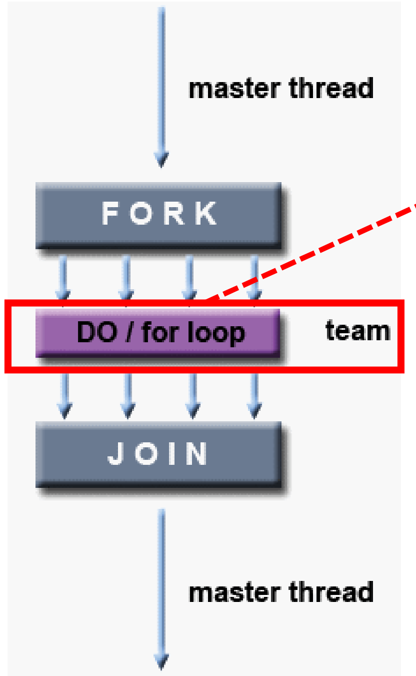

```
[morris@jotunn hellothreads]$ ls -al
total 28
drwxrwxr-x 2 morris morris  121 okt 14 08:51 .
drwxrwxr-x 9 morris morris  150 okt 14 08:08 ..
-rwxrwxr-x 1 morris morris 8844 okt 14 08:46 hellothreads
-rw-rw-r-- 1 morris morris  277 okt  9 20:16 hellothreads.c
-rw-rw-r-- 1 morris morris  168 okt 14 08:45 slurm-200209.out
-rw-rw-r-- 1 morris morris  168 okt 14 08:51 slurm-200210.out
-rwxr-xr-x 1 morris morris  313 okt 14 08:51 submit-hellothreads.sh
```

```
[morris@jotunn hellothreads]$ more slurm-200210.out
Hello from your main thread.
Hello from thread 0 of 4.
Hello from thread 1 of 4.
Hello from thread 3 of 4.
Hello from thread 2 of 4.
Hello again from your main thread.
```

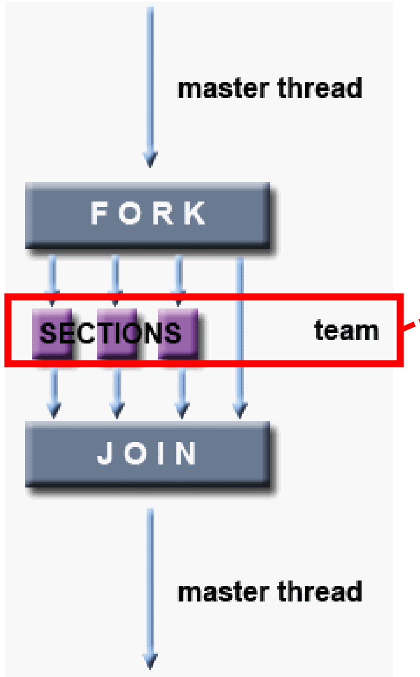
Understanding OpenMP Work Sharing Constructs & Methods



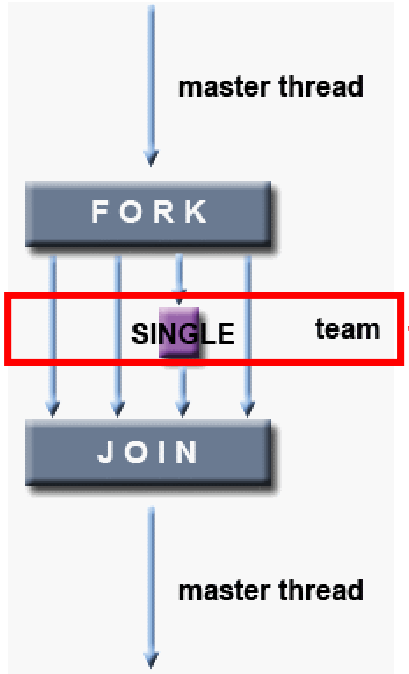
OpenMP Work Sharing Constructs (cf. Lecture 6) – Revisited



- for - shares iterations of a loop across the team. Represents a type of 'data parallelism' (cf. Lecture 3)



- Sections - breaks work into separate, discrete sections. Each section is executed by a thread. Can be used to implement a type of 'functional parallelism' (cf. Lecture 3)



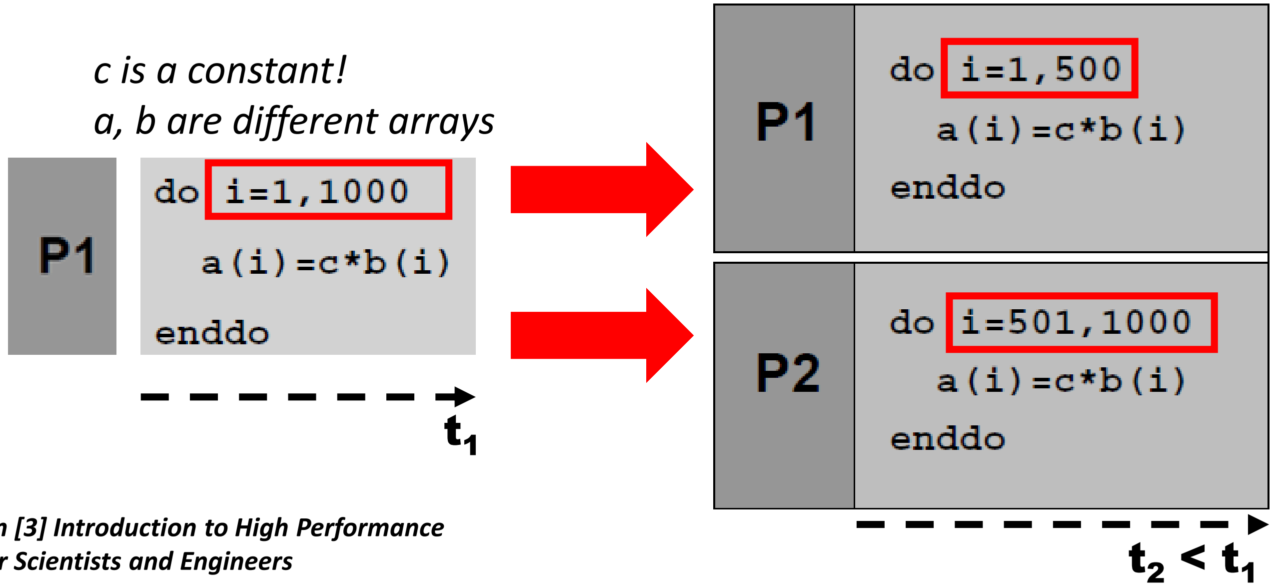
- Single - serializes a section of code

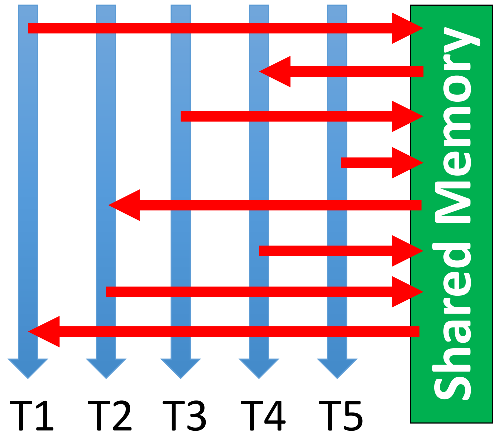
[1] LLNL OpenMP Tutorial

Data Parallelism: Medium-grained Loop Parallelization (cf. Lecture 3)

- Idea: Computations performed on individual array elements are independent of each other
  - Good for parallel execution by N processors (e.g., using shared memory parallel programming)

c is a constant!
a, b are different arrays

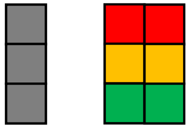




Modified from [3] Introduction to High Performance Computing for Scientists and Engineers

OpenMP Work Sharing Construct: Advanced For Loop Example with Printout

```
#include <omp.h>
#define N 1000
#define CHUNKSIZE 100
main(int argc, char *argv[]) {
  int i, chunk, nthreads;
  float a[N], b[N], c[N];
  /* Some initializations */
  for (i=0; i < N; i++)
    a[i] = b[i] = i * 1.0;
  chunk = CHUNKSIZE;
  #pragma omp parallel shared(a,b,c,chunk) private(i)
  {
    nthreads = omp_get_num_threads();
    printf("Number of threads in parallel region = %d\n", nthreads);
    #pragma omp for schedule(dynamic,chunk) nowait
    for (i=0; i < N; i++)
      c[i] = a[i] + b[i];
  }  /* end of parallel region */
  return 0;
}
```



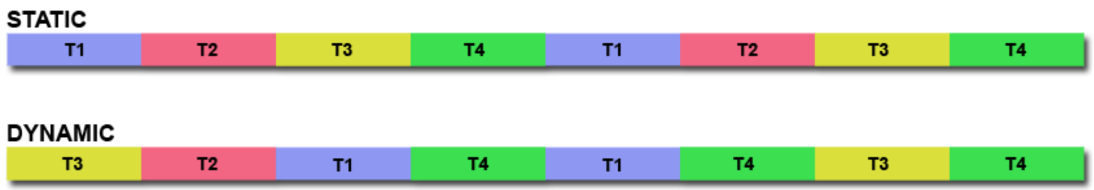
C = A+B

(we show many threads work on addition)

(we can also play with the i iterator here and do printouts and individual thread ids)

- Arrays a, b, c, and chunk will be shared by all threads
- Variable i is private to each thread: each thread will have its own unique copy of i

- Schedule: Describes how iterations of the loop are divided among the threads in the team; the default schedule is implementation dependent
- The iterations of the loop will be distributed dynamically in CHUNK sized pieces: when a thread finishes one chunk, it is dynamically assigned another
- Threads will not synchronize upon completing their individual pieces of work (nowait)



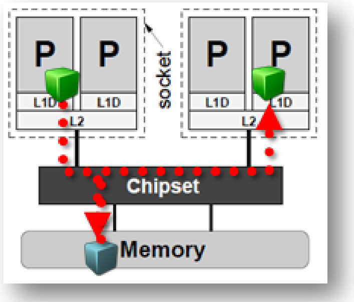
[1] LLNL OpenMP Tutorial

OpenMP Synchronization Construct: Critical Region Example

```
#include <omp.h>
main(int argc, char *argv[]) {
  int x;
  x = 0;

  #pragma omp parallel shared(x)
  {
   #pragma omp critical
   x = x + 1;

  }  /* end of parallel region */
  printf("Number of x after parallel region = %d\n", x);
  return 0;
}
```

- If a thread is currently executing inside a critical region and another thread reaches that critical region and attempts to execute it, it will block until the first thread exits that critical region
- All threads in the team will attempt to execute in parallel, however, because of the critical construct surrounding the increment of x, only one thread will be able to read/increment/write x at any time
- Note the 'race conditions' of variable x otherwise: Race Condition in shared-memory: shared variable x will be set concurrently by the different threads – not with critical regions

(we can show what happens if the critical region is removed)



[1] LLNL OpenMP Tutorial

OpenMP ThreadPrivate Directive – Persistence between Parallel Regions (1)

```
#include <omp.h>
int  a, b, i, tid;
float x;
#pragma omp threadprivate(a, x)
main(int argc, char *argv[]) {
  omp_set_dynamic(0);
  printf("1st Parallel Region:\n");
  #pragma omp parallel private(b,tid)
  {
    tid = omp_get_thread_num();
    a = tid;
    b = tid;
    x = 1.1 * tid +1.0;
    printf("Thread %d:   a,b,x= %d %d %f\n",tid,a,b,x);
  }  /* end of parallel region */
  printf("Master thread doing serial work here\n");
  ...
}
```

- **Threadprivate() directive specifies that variables are replicated, with each thread having its own copy**
- **Can be used to make global file scope variables (C/C++/Fortran local and persistent to a thread through the execution of multiple parallel regions**
- **Threadprivate() variables differ from private variables because they are able to persist between different parallel regions of a code**

- **Explicitly turn off dynamic threads**

[1] LLNL OpenMP Tutorial

OpenMP ThreadPrivate Directive – Persistence between Parallel Regions (2)

```
#include <omp.h>
int  a, b, i, tid;
float x;
#pragma omp threadprivate(a, x)
main(int argc, char *argv[]) {
  ...
  printf("Master thread doing serial work here\n");
  ...
  printf("2nd Parallel Region:\n");
  #pragma omp parallel private(tid)
  {
    tid = omp_get_thread_num();
    printf("Thread %d:   a,b,x= %d %d %f\n",tid,a,b,x);
  }  /* end of parallel region */
}
```

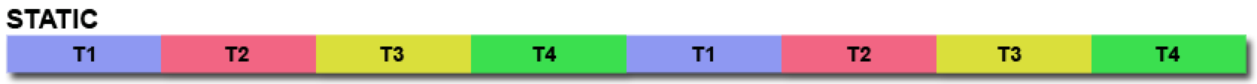
```
Output:

1st Parallel Region:
Thread 0:   a,b,x= 0 0 1.000000
Thread 2:   a,b,x= 2 2 3.200000
Thread 3:   a,b,x= 3 3 4.300000
Thread 1:   a,b,x= 1 1 2.100000
************************************
Master thread doing serial work here
************************************
2nd Parallel Region:
Thread 0:   a,b,x= 0 0 1.000000
Thread 3:   a,b,x= 3 0 4.300000
Thread 1:   a,b,x= 1 0 2.100000
Thread 2:   a,b,x= 2 0 3.200000
```

[1] LLNL OpenMP Tutorial

OpenMP Reduction Clause Example – Vector Dot Product Example

```c
#include <omp.h>
main(int argc, char *argv[])  {
  int   i, n, chunk;
  float a[100], b[100], result;
  n = 100;
  chunk = 10;
  result = 0.0;
  for (i=0; i < n; i++) {
    a[i] = i * 1.0;
    b[i] = i * 2.0;
  }
  #pragma omp parallel for       \
   default(shared) private(i)   \
   schedule(static,chunk)       \
   reduction(+:result)
   for (i=0; i < n; i++)
     result = result + (a[i] * b[i]);
  printf("Final result= %f\n",result);
}
```

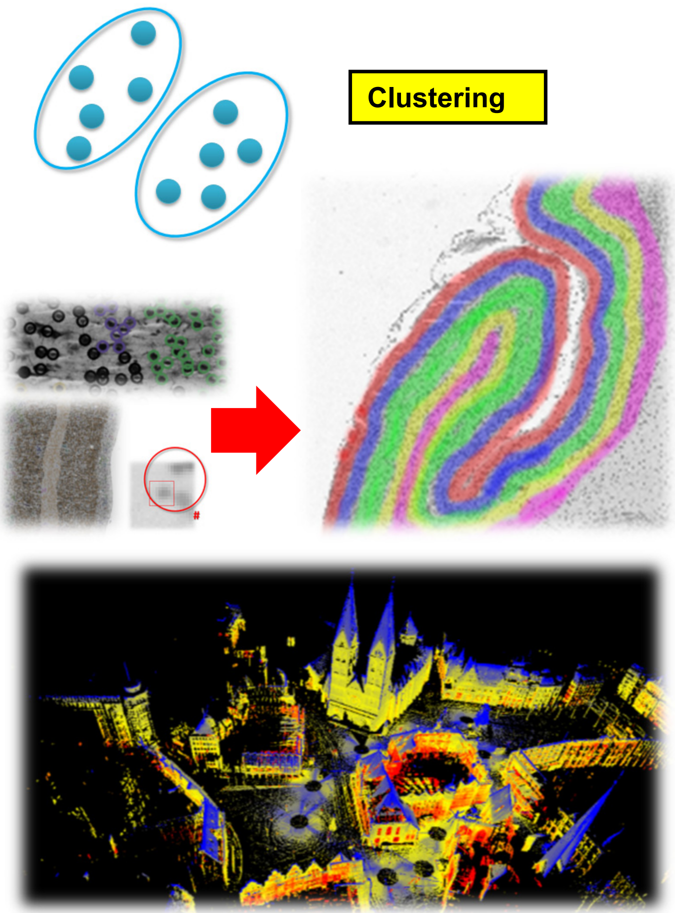
- Reduction clause performs a reduction operation on the variables that appear in its list
- A private copy for each list variable is created and initialized for each thread
- At the end of the reduction, the reduction variable is applied to all private copies of the shared variable, and the final result is written to the global shared variable
- Reduction operations are a smart alternative to manual critical regions definitions around operations of variables
- Reduction operation automatically localizes variable
- 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

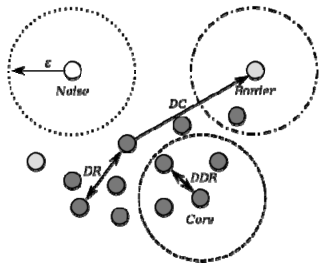


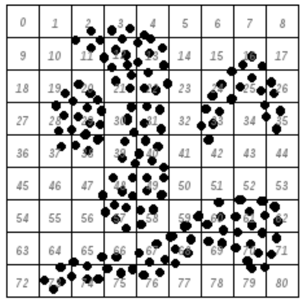
- Vector Dot Product Example: Result is a scalar!
- Iterations of the parallel loop will be distributed in equal sized blocks to each thread in the team (schedule static)
- At the end of the parallel loop construct, all threads will add their values of "result" to update the master thread's global copy

[1] LLNL OpenMP Tutorial

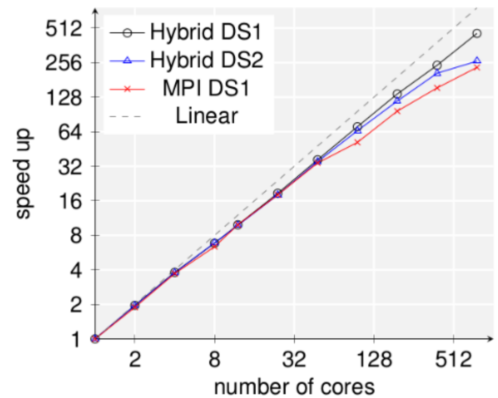
‘Big Data‘ Science Example – Parallel & Scalable Clustering Algorithm – Revisited

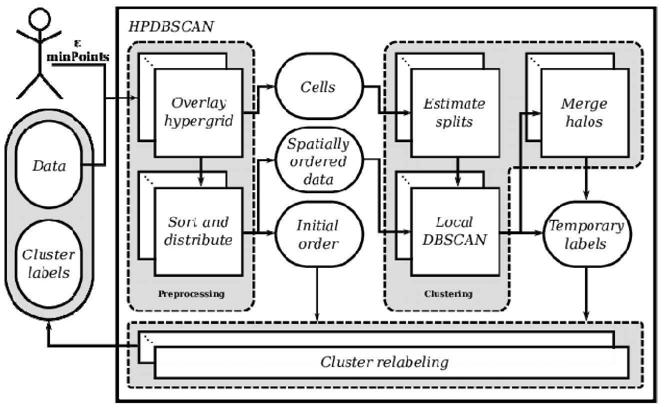


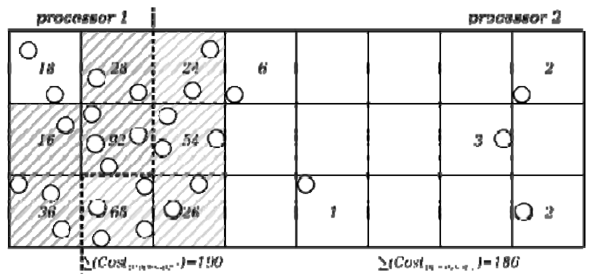




Overlayed spatial grid







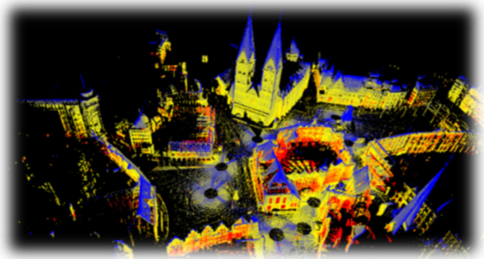
[4] M. Goetz and M. Riedel et al,
Proceedings IEEE Supercomputing Conference, 2015

HPDBSCAN Clustering OpenMP Application Example in Data Sciences

```
// hpdbscan.h file
...
#include <hdf5.h>
#include <omp.h>
...
// local DBSCAN run
#pragma omp parallel for schedule(dynamic, 32) private(neighboring_points)
    firstprivate(previous_cell) reduction(merge: rules)
   for (size_t point = lower; point < upper; ++point) {
...
Clusters cluster(Dataset& dataset, int threads=omp_get_max_threads()) {
#ifdef WITH_OUTPUT
double execution_start = omp_get_wtime();
#endif
// set the number of threads
omp_set_num_threads(threads);
...
```

- **How Many Threads within a parallel region?**
- **omp_get_max_threads() : generally reflects the number of threads as set by the OMP_NUM_THREADS environment variable**
- **omp_set_num_threads(threads) routine affects the number of threads to be used for subsequent parallel regions**

```
#!/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
```



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

Investigating Computatoinal Job Details – Scontrol Command of Scheduler

```
[morris@jotunn hpdbscan-openmp]$ scontrol show jobid -dd 200207
JobId=200207 JobName=HPDBSCAN-openmp
   UserId=morris(30017) GroupId=morris(30017)
   Priority=4294893959 Nice=0 Account=(null) QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   DerivedExitCode=0:0
   RunTime=00:00:31 TimeLimit=01:00:00 TimeMin=N/A
   SubmitTime=2019-10-14T08:14:56 EligibleTime=2019-10-14T08:14:56
   StartTime=2019-10-14T08:14:56 EndTime=2019-10-14T09:14:56
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=normal AllocNode:Sid=jotunn:5911
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=compute-2-0
   BatchHost=compute-2-0
   NumNodes=1 NumCPUs=4 CPUs/Task=4 ReqB:S:C:T=0:0:*:*
   TRES=cpu=4,node=1
   Socks/Node=* NtasksPerN:B:S:C=1:0:*:* CoreSpec=*
     Nodes=compute-2-0 CPU_IDs=0-3 Mem=0
   MinCPUsNode=4 MinMemoryNode=0 MinTmpDiskNode=0
   Features=(null) Gres=(null) Reservation=(null)
   Shared=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/morris/2019-HPC-Course/openmp/hpdbscan-openmp/submit-clustering-bremen.sh
   WorkDir=/home/morris/2019-HPC-Course/openmp/hpdbscan-openmp
   StdErr=/home/morris/2019-HPC-Course/openmp/hpdbscan-openmp/HPDBSCAN-200207.err
   StdIn=/dev/null
   StdOut=/home/morris/2019-HPC-Course/openmp/hpdbscan-openmp/HPDBSCAN-200207.out
   BatchScript=
#!/bin/bash
#SBATCH --job-name=HPDBSCAN-openmp
#SBATCH -o HPDBSCAN-%j.out
#SBATCH -e HPDBSCAN-%j.err
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4
#SBATCH --time=01:00:00

# load modules
module load gnu/5.3.0
module load hdf5/1.8.17
module load openmpi/1.10.2
module load HPDBSCAN/mpi

# set openmp threads
export OMP_NUM_THREADS=4

# executable
HPDBSCAN=dbscan

# your own copy of bremen small
BREMENSMALLDATA=/home/morris/2019-HPC-Course/openmp/hpdbscan-openmp/bremenSmall.h5

# your own copy of bremen big
BREMENBIGDATA=/home/morris/2019-HPC-Course/openmp/hpdbscan-openmp/bremen.h5

mpirun $HPDBSCAN -m 300 -e 800 -t 4 $BREMENSMALLDATA
```

```
[morris@jotunn hellothreads]$ qstat
Job id              Name             Username        Time Use S Queue
------------------- ---------------- --------------- -------- - -------
199590              hello-mpi-exampl jfb3            00:00:00 Q normal
199591              hello-mpi-exampl jfb3            00:00:00 Q normal
199592              hello-mpi-exampl jfb3            00:00:00 Q normal
199593              hello-mpi-exampl jfb3            00:00:00 Q normal
199595              pingpong         jfb3            00:00:00 Q normal
199596              pingpong         jfb3            00:00:00 Q normal
199840              pingpong         jfb3            00:00:00 Q normal
199841              pingpong         jfb3            00:00:00 Q normal
200207              HPDBSCAN-openmp  morris          00:00:53 R normal
```

```
[morris@jotunn hpdbscan-openmp]$ more HPDBSCAN-200207.err
slurmstepd: *** JOB 200207 ON compute-2-0 CANCELLED AT 2019-10-14T09:15:09 DUE TO TIME LIMIT ***
mpirun: Forwarding signal 18 to job
```

- Runs against the wall on Jötunn (01:00:00 walltime)
  - Number of threads = 4 too low (was only bremensmall data)
  - Potentially also parameter setting can be problematic (e.g., e=800 in this example)
  - Experiment with parameters and adding more threads (or going hybrid with MPI)

```
[morris@jotunn hpdbscan-openmp]$ more submit-clustering-bremen.sh
```

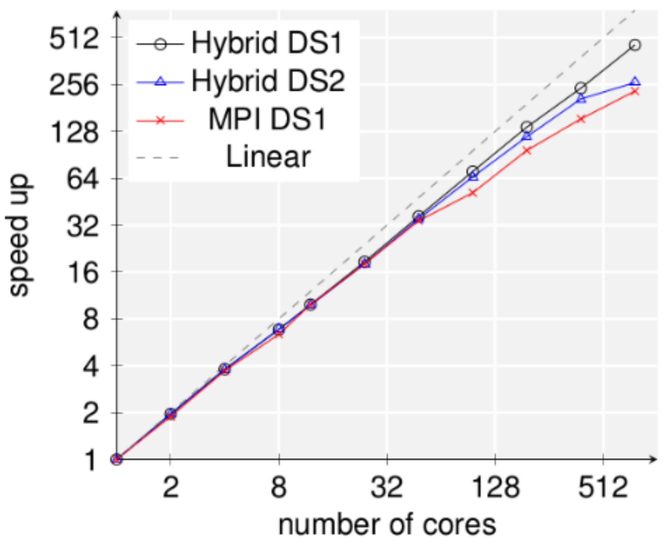

```
#!/bin/bash
#SBATCH --job-name=HPDBSCAN-openmp
#SBATCH -o HPDBSCAN-%j.out
#SBATCH -e HPDBSCAN-%j.err
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4
#SBATCH --time=01:00:00

# load modules
module load gnu/5.3.0
module load hdf5/1.8.17
module load openmpi/1.10.2
module load HPDBSCAN/mpi

# set openmp threads
export OMP_NUM_THREADS=4

# executable
HPDBSCAN=dbscan

# your own copy of bremen small
BREMENSMALLDATA=/home/morris/2019-HPC-Course/openmp/hpdbscan-openmp/bremenSmall.h5

# your own copy of bremen big
BREMENBIGDATA=/home/morris/2019-HPC-Course/openmp/hpdbscan-openmp/bremen.h5

mpirun $HPDBSCAN -m 300 -e 800 -t 4 $BREMENSMALLDATA
```

Scientific Application Example: Data Mining & Clustering of Big Data → Hybrid!

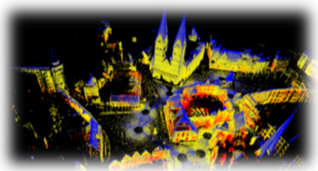
- Hybrid data mining algorithm example
  - Parallel Density-based Spatial Clustering for Applications with Noise (DBSCAN)
  - Using MPI and OpenMP to scale better
  - Standalone OpenMP is also possible to use

```
[morris@jotunn criticalexample]$ qstat
Job id             Name            Username       Time Use S Queue
------------------ --------------- -------------- -------- - -------
199590             hello-mpi-exampl jfb3          00:00:00 Q normal
199591             hello-mpi-exampl jfb3          00:00:00 Q normal
199592             hello-mpi-exampl jfb3          00:00:00 Q normal
199593             hello-mpi-exampl jfb3          00:00:00 Q normal
199595             pingpong        jfb3           00:00:00 Q normal
199596             pingpong        jfb3           00:00:00 Q normal
199840             pingpong        jfb3           00:00:00 Q normal
199841             pingpong        jfb3           00:00:00 Q normal
200211             HPDBSCAN-openmp  morris        00:00:31 C normal
```

(Bremen Small Data → 16 threads worked ~ 30 min, big data?)




```bash
#!/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
```

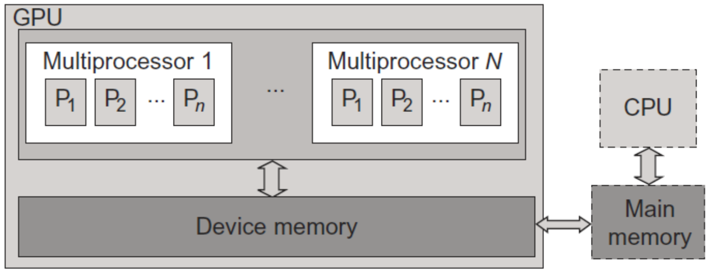


```
[morris@jotunn hpdbscan-openmp]$ more HPDBSCAN-200211.out
Calculating Cell Space...
        Computing Dimensions... [OK] in 0.019950
        Computing Cells...      [OK] in 0.304335
        Sorting Points...       [OK] in 0.422754
        Distributing Points...  [OK] in 0.000000
DBSCAN...
        Local Scan...           [OK] in 1888.362077
        Merging Neighbors...    [OK] in 0.000000
        Adjust Labels ...       [OK] in 0.026210
        Rec. Init. Order ...    [OK] in 0.600609
        Writing File ...        [OK] in 0.233867
Result...
        21      Clusters
        2974394 Cluster Points
        25606   Noise Points
        2949094 Core Points
Took: 1890.412036s
```

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

Recent Support of OpenMP for Programming GPUs with Directives

```
...
#pragma omp target map (tofrom:y), map(to:x)
#pragma omp teams num_teams(10) num_threads(10)
#pragma omp distribute
for (...) {
  ...
  #pragma omp parallel for
  for (...) {
  }
  ...
}
...
```

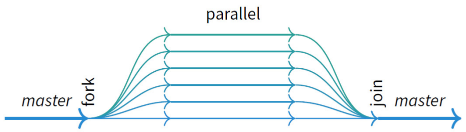


[6] Distributed & Cloud Computing Book

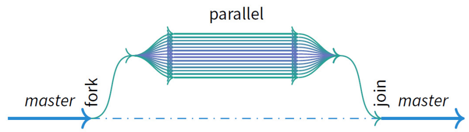
- **OpenACC is similar to OpenMP, because it is modeled after OpenMP, but for accelerators**

```
#pragma acc data copyout(y[0:N]) create(x[0:N])
{
double sum = 0.0;
#pragma acc parallel loop
for (int i=0; i<N; i++) {
    x[i] = 1.0;
    y[i] = 2.0;
}
#pragma acc parallel loop
for (int i=0; i<N; i++) {
    y[i] = i*x[i]+y[i];
}
}
```

- **OpenMP is the de-facto standard for multi-threaded programming on CPU**
- **OpenMP includes since version 4.0 (better since 4.5) also capabilities for programming GPUs**



OpenMP



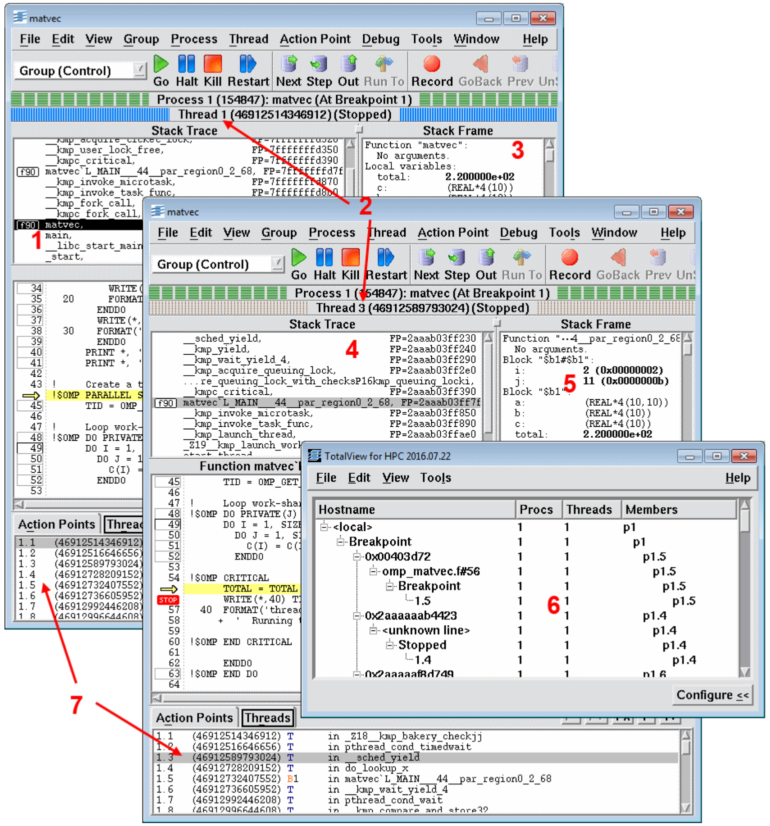
OpenACC

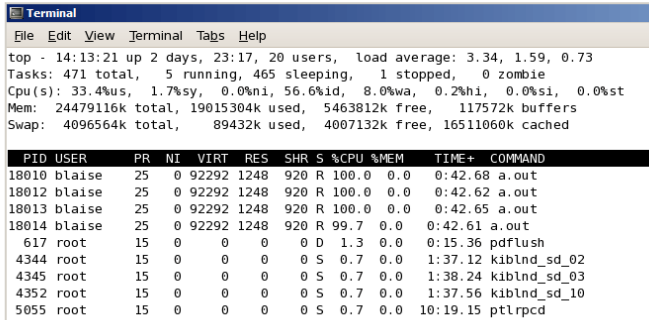
➢ **Lecture 7 will offer more details on OpenMP relationships of programming GPUs and similiarites to GPU programming using OpenACC**

Monitoring, Debugging and Performance Analysis Tools for OpenMP

- Different Tools exist
  - E.g. TotalView Debugger
  - E.g. Linux top command
  - E.g. Linux ps command

```
% ps -Lf
UID          PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
blaise     22529 28240 22529  0    5 11:31 pts/53   00:00:00 a.out
blaise     22529 28240 22530 99    5 11:31 pts/53   00:01:24 a.out
blaise     22529 28240 22531 99    5 11:31 pts/53   00:01:24 a.out
blaise     22529 28240 22532 99    5 11:31 pts/53   00:01:24 a.out
blaise     22529 28240 22533 99    5 11:31 pts/53   00:01:24 a.out

% ps -T
  PID  SPID TTY          TIME CMD
22529 22529 pts/53   00:00:00 a.out
22529 22530 pts/53   00:01:49 a.out
22529 22531 pts/53   00:01:49 a.out
22529 22532 pts/53   00:01:49 a.out
22529 22533 pts/53   00:01:49 a.out

% ps -Lm
  PID   LWP TTY          TIME CMD
22529     - pts/53   00:18:56 a.out
    - 22529 -        00:00:00 -
    - 22530 -        00:04:44 -
    - 22531 -        00:04:44 -
    - 22532 -        00:04:44 -
    - 22533 -        00:04:44 -
```

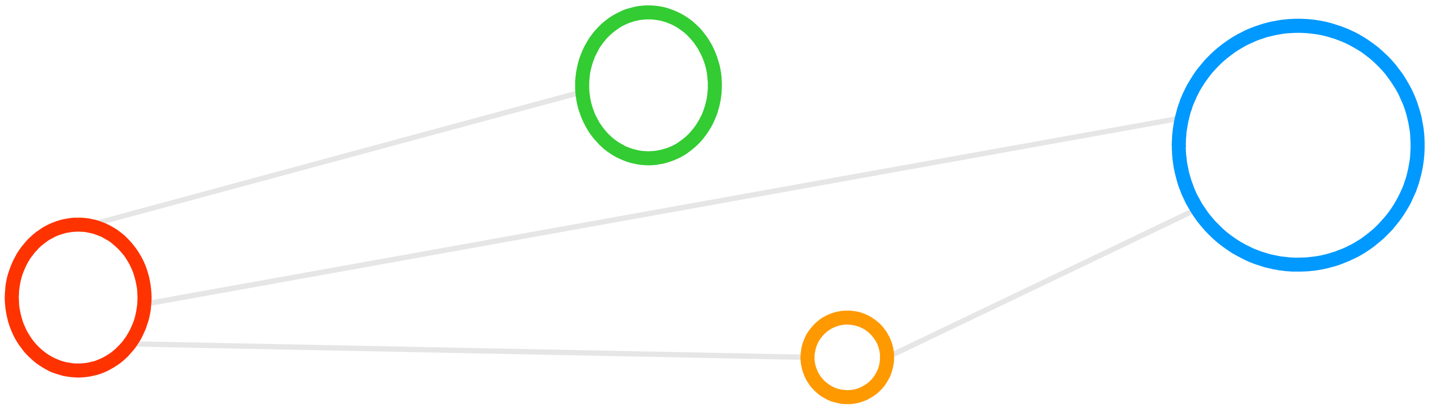



```
Terminal
File  Edit  View  Terminal  Tabs  Help
top - 14:13:21 up 2 days, 23:17, 20 users,  load average: 3.34, 1.59, 0.73
Tasks: 471 total,   5 running, 465 sleeping,   1 stopped,   0 zombie
Cpu(s): 33.4%us,  1.7%sy,  0.0%ni, 56.6%id,  8.0%wa,  0.2%hi,  0.0%si,  0.0%st
Mem:  24479116k total, 19015304k used,  5463812k free,   117572k buffers
Swap:  4096564k total,    89432k used,  4007132k free, 16511060k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
18010 blaise    25   0 92292 1248  920 R 100.0  0.0   0:42.68 a.out
18012 blaise    25   0 92292 1248  920 R 100.0  0.0   0:42.62 a.out
18013 blaise    25   0 92292 1248  920 R 100.0  0.0   0:42.65 a.out
18014 blaise    25   0 92292 1248  920 R 99.7  0.0   0:42.61 a.out
  617 root      15   0     0    0    0 D  1.3  0.0   0:15.36 pdflush
 4344 root      15   0     0    0    0 S  0.7  0.0   1:37.12 kiblnd_sd_02
 4345 root      15   0     0    0    0 S  0.7  0.0   1:38.24 kiblnd_sd_03
 4352 root      15   0     0    0    0 S  0.7  0.0   1:37.56 kiblnd_sd_10
 5055 root      15   0     0    0    0 S  0.7  0.0  10:19.15 ptlrpcd
```

[7] LLNL OpenMP Tutorial

➢ **Lecture 9 will provide a set of tools that can be used for monitoring, debugging, and performance analysis of MPI and OpenMP**

Lecture Bibliography



Lecture Bibliography

- [1] LLNL OpenMP Tutorial, Online:
  https://computing.llnl.gov/tutorials/openMP/
- [2] The OpenMP API specification for parallel programming, Online:
  http://openmp.org/wp/openmp-specifications/
- [3] Introduction to High Performance Computing for Scientists and Engineers, Georg Hager & Gerhard Wellein, Chapman & Hall/CRC Computational Science, ISBN 143981192X
- [4] 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
- [5] Icelandic HPC Machines & Community, Online:
  http://ihpc.is
- [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