**1 HPC**

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    - Often in state of flux/vendor-specific, quickly outdated.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

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

**2 Parallelization Fundamentals**

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

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

**3 HPC A Parallel Programming MPI**

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

SPMD: Single Processor, Multiple Data.

General:

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

### 3.1 HPC A Practical Lecture

Basic commands

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

MPI:

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

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

- MPI JobScript:

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

**4 HPC A Parallel Programming OpenMP**

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

### 4.1 HPC A Practical Lecture

- Int nthreads, tid;
  #pragma omp parallel private(tid)
  - Shared variable nthreads, local variable tid.

  tid = omp_get_thread_num();
    - Get the current thread id.

  If(tid == 0)

    - If the master thread.

    nthreads = omp_get_num_threads();

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

  #pragma omp for private(n)
    - Already in parallel so statement omp for is enough when creating new parallel region.
- Example program (helloloop):

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

int main (argc, argv)
{

  int nthreads, tid;
  int n;

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

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

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

  }

  return 0;
}
```

- OpenMP jobscript:

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

export OMP_NUM_THREADS=4

cd /home/morris/2014-HPC-A

./helloworldomp.exe
```

## 5 Algorithms and Data Structures

-