

Second Programming Assignment

- Parallel Implementation of **Minimum Spanning Tree**
- Deadline: Monday May 9
- Implementation & Benchmark.
- Based on **Borůvka's** algorithm.
- Platform:
DAS-4 (16 nodes, 2 CPUs/node, 2 threads/core)

Second Programming Assignment

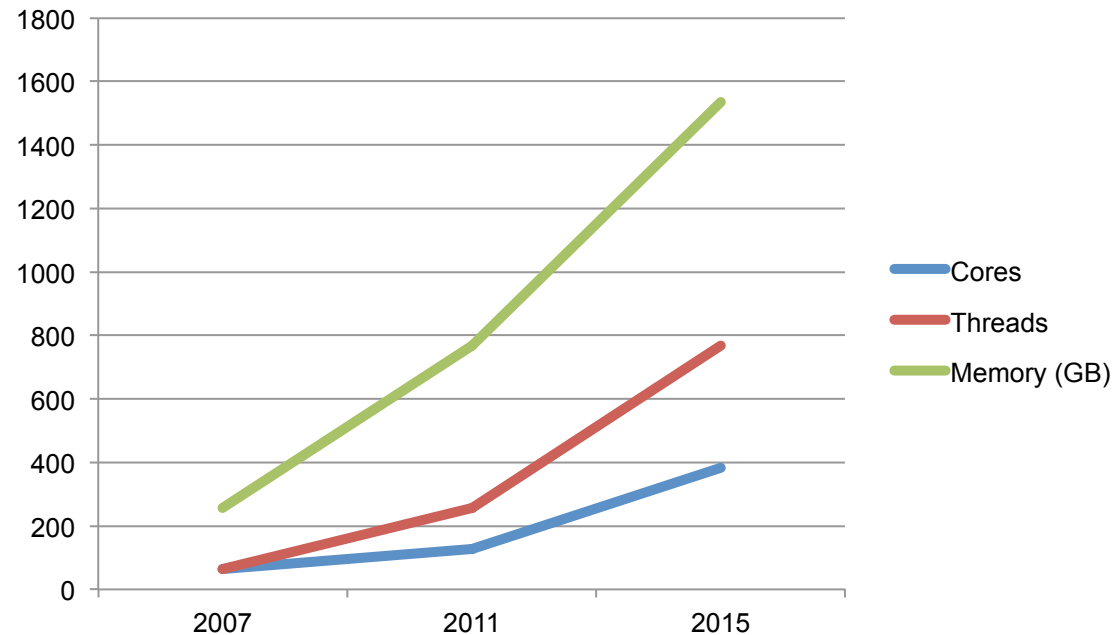
- Parallelization **only** to be done by distributing the work over different nodes. So, per node only **one CPU core** and **one thread** will be used. So, the maximum degree of parallelism is limited to **16 !!!**
- Test graphs are taken from the UF Matrix Collection. Three of which were selected:
 - mouse_gene (29M nnz),
 - ldoor (42M nnz),
 - nkpkkt240 (760M nnz).

A Short DAS History (1)

- DAS configurations at Leiden University.
- DAS3 (2007): 32 nodes
 - Each node: 2 CPUs (single core), 4GB RAM
 - Totals: 64 CPUs / 64 cores / 64 threads, 256GB RAM
- DAS4 (2011): 16 nodes
 - Each node: 2 CPUs (4 cores, 2 threads/core), 48GB RAM
 - Totals: 32 CPUs / 128 cores / 256 threads, 768GB RAM

A Short DAS History (2)

- DAS5 (2015): 24 nodes
 - Each node: 2 CPUs (8 cores, 2 threads/core), 64GB RAM
 - Totals: 48 CPUs / 384 cores / 768 threads, 1536GB RAM



MPI

- Communication between processes in a distributed program is typically implemented using MPI: **Message Passing Interface**.
- MPI is a generic **API** that can be implemented in different ways:
 - Using specific interconnect hardware, such as InfiniBand.
 - Using TCP/IP over plain Ethernet.
 - Or even used (emulated) on Shared Memory for inter process communication on the same node.

Some MPI basic functions

- `#include <mpi.h>`

- **Initialize library:**

```
MPI_Init(&argc, &argv);
```

- **Determine number of processes that take part:**

```
int n_procs;
```

```
MPI_Comm_size(MPI_COMM_WORLD, &n_procs);
```

(MPI_COMM_WORLD is the initially defined universe intracommunicator for all processes)

- **Determine ID of this process:**

```
int id;
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &id);
```

Sending Messages

```
MPI_Send(buffer, count, datatype, dest, tag, comm) ;
```

- buffer: pointer to data buffer.
- count: number of items to send.
- datatype: data type of the items (see next slide).

All items must be of the same type.

- dest: rank number of destination.
- tag: message tag (integer), may be 0.

You can use this to distinguish between different messages.

- comm: communicator, for instance MPI_COMM_WORLD.

Note: this is a blocking send!

MPI data types

You must specify a data type when performing MPI transmissions.

- For instance for built-in C types:
 - "int" translates to MPI_INT
 - "unsigned int" to MPI_UNSIGNED
 - "double" to MPI_DOUBLE, and so on.
- You can define your own MPI data types, for example if you want to send/receive custom structures.

Other calls

- `MPI_Recv()`
- `MPI_Isend()`, `MPI_Irecv()`
 - **Non-blocking send/receive**
- `MPI_Scatter()`, `MPI_Gather()`
- `MPI_Bcast()`
- `MPI_Reduce()`

Shutting down

- `MPI_Finalize()`

Example: Computing Pi

Source: <http://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiexmpl/src/pi/C/>

```
int main(int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
```

```

while (!done)
{
    if (myid == 0) {
        printf("Enter number of intervals: (0 quits)");
        scanf("%d", &n);
    }
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (n == 0) break;

    h = 1.0 / (double) n;
    sum = 0.0;
    for (i = myid + 1; i <= n; i += numprocs) {
        x = h * ((double)i - 0.5);
        sum += 4.0 / (1.0 + x*x);
    }
    mypi = h * sum;

    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
              MPI_COMM_WORLD);

    if (myid == 0)
        printf("pi = approximately %.16f, Error is %.16f\n",
              pi, fabs(pi - PI25DT));
}
MPI_Finalize();
return 0;
}

```

Compiling and Running

- Compile using "mpicc", this automatically links in necessary libraries.
- Run the program using "mpirun". On systems like DAS, mpirun is started from a "job script".
- The job script is submitted to the job scheduler, which will run your job once your resource reservation can be fulfilled.
- See assignment text for links to more detailed manuals.