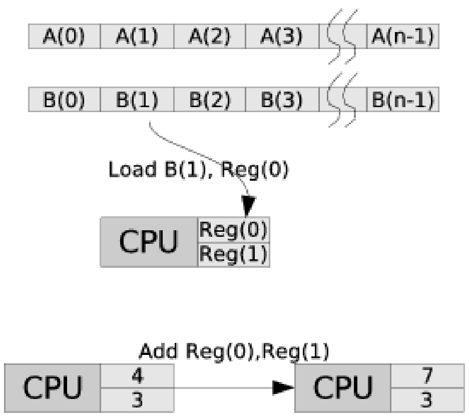
Technische Universität München
Institut für Informatik
Dr. rer. nat. Tobias Neckel
Matous Sedlacek

WT 2012/2013

Parallel Numerics

Exercise 1: Flynn's Taxonomy & MPI Basics

1) Flynn's Taxonomy



Given is a simple computer with a main memory that is capable to hold two arrays A and B with n entries each. The CPU has two registers and is able to perform one operation `load,store,add,mult` per clock tick. The following program computes $A(2) = A(3) + 2B(5)$:

```
load B(5),Reg(0)
load    2,Reg(1)
mult
load A(3),Reg(1)
add
store Reg(0),A(2)
```

Please note, that this exercise has a more theoretical character!

i) The memory holds the vectors $A, B \in \mathbb{R}^6$, $n = 6$. Write a program that computes $2A+B$ and stores the result in A.

```
for i = 0...5
    load      A(i),Reg(0)
    load      2,Reg(1)
    mult
    load      B(i),Reg(1)
    add
    store     Reg(0),A(i)
```

ii) To what type of computer does the architecture belong according to Flynn's taxonomy?
SISD.

iii) The CPU is added a second pair of registers (*Reg*(2) and *Reg*(3)). Furthermore, there is an operation `add2` computing "*Reg*(0) added *Reg*(1)" and "*Reg*(2) added *Reg*(3)" in one clock cycle. The result values are stored in *Reg*(0) and *Reg*(2). Rewrite the application using `add2` and compare the number of clock cycles required. Is there a computer available nowadays that supports such operations? To what type of computer does such an architecture belong according to Flynn's taxonomy?
Program sequence for first two iterations:

```
load   A(0),Reg(0)
load   A(1),Reg(2)

load   2,Reg(1)
load   2,Reg(3)

mult   Reg(0),Reg(1)
mult   Reg(2),Reg(3)

load   B(0),Reg(1)
load   B(1),Reg(3)

add    Reg(0),Reg(1)
                       > replaced by add2
add    Reg(2),Reg(3)

store  Reg(0),A(0)
store  Reg(2),A(1)
```

This way, using ***add2****, only* 33 *cycles are required instead of* 36 *cycles. Hence,* $\frac{1}{2}$ *cycles per iteration are saved. Further improvement could be an operation* ***mult2****. Real computers nowadays e.g. SSE, graphic cards, vector components. The Flynn type is SIMD.*

iv) The original computer is added a second CPU with two registers. Rewrite the program. To what type of computer does such an architecture belong according to Flynn's taxonomy?
Real parallel architecture/program. Flynn type is MIMD. Split up the program into 2 loops (operation sequences):

```
for i = 0...2         |    for i = 3...5
    see part i)       |        see part i)
```

v) Suppose, the CPU (the original one with two registers) is able to handle large instructions, i.e., two operations per cycle as long as they do not use the same resources. This means the combination

```
load B(5),Reg(0) and load A(2),Reg(1)
```

is not supported since both operations would use the main memory. On the other hand

```
mult and load B(4),Reg(1)
```

is allowed. Rewrite your program and count the number of clock cycles required. What standard computer architectures support such operations?
Again program sequence for first two iterations:

```
load         A(0),Reg(0)
load         2,Reg(1)
mult / load  B(0),Reg(1)
add / load   2,Reg(1)      already step 2 of second iteration (loop merge)
store        Reg(0),A(0)
```

5 *cycles first iteration,* 4 *cycles following iterations. Overall:* $5 \cdot 4 + 5 = 25$ *cycles. Modern architectures implement pipeline concept. Splitting up into substeps or executing substeps simultaneously. E.g. superscalar architectures, Very Large Instruction Words.*

2) Single Program Multiple Data

Define the term Single Program Multiple Data (SPMD). To what type of computer architecture according to Flynn do such applications fit? Compare SPMD to SIMD.
All processors use the same program but different data. Flynn type is MIMD: Although all processors run on the same instruction stream, the operations are not synchronized (difference to SIMD).

3) Parallel Programming Paradigms

In Problem 1 the simple computer was made a shared memory machine. Define the terms "shared memory" and "distributed shared memory". What type of architecture is implemented in the HLRB II (Höchstleistungsrechner Bayern II) at LRZ?
Shared memory: Memory that may be simultaneously accessed by multiple programs/processors.
Distributed memory: Refers to a multiple-processor computer system in which each processor has its own private memory $\rightarrow$ *Computational tasks on local data.*
Hybrid: CPU's on node share memory. Nodes only know about own memory but may communicate with other nodes.

The HLRB-II architecture is a Numalink4 (enhanced ccNUMA) distributed-shared memory architecture.

4) Setting up an MPI Environment

Last summer semester you attended the lecture on “Parallel Programming”. Part of the course dealt with MPI (Message Passing Interface) used to parallelise programs on supercomputers or workstation clusters. This term, we will use MPI for programming a couple of numerical codes. Therefore, a recapitulation of the material about MPI for those who are familiar with the interface is subject of this exercise. For all the others an introduction will be given throughout the tutorials.

Make yourself familiar with working with MPI and the terms NFS, public key authorization and the command `mpdboot`.

i) Write a machine file for your personal experiments.
A machine file specifies all the machines you want to use. Entries are seperated by line breaks.

ii) To avoid to be asked for your password every time MPI starts up, execute the following commands in your home-directory:

- Create a public dsa key, but do not give it a passphrase:

```
> ssh-keygen -t dsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/login/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/login/.ssh/id_dsa.
```

- Add the new key stored in `.ssh/id_dsa` to the public key file:

```
> cd ~/.ssh
> cat id_dsa.pub >> authorized_keys
```

- Log in once on every computer you want to use.

iii) Make yourself familiar with the commands `mpdboot`, `mpirun` and `mpdallexit`.
Example: `mpdboot -f mfile -n 25 --ncpus=1 -1`

- *`mpdboot` starts up MPI environment.*
- *`-f` passes machinefile.*
- *`-n` number of parallel SPMD instances (number of entries within machinefile should be greater).*
- *`--ncpus` number of processes to be started on local machine.*
- *`-1` allows more than one processor/node. Other duplicates in machinefile are neglected.*

Starting parallel application with: `mpirun -n * ./executable`*.*
`mpdallexit` *closes the MPI environment.*

5) MPI Application Structure

Below is a very simple MPI application. Try to run this example and identify the semantics and the syntax conventions of the five different MPI commands.

```
int main(int argc, char* argv[]) {
   int i, myrank, nproz;
   int neighbourID_le, neighbourID_ri;
   int recv_neighbourID_le, recv_neighbourID_ri;
   double pi;
   MPI_Status status;

   MPI_Init( &argc, &argv );
   MPI_Comm_size( MPI_COMM_WORLD, &nproz );
   MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

   MPI_Barrier( MPI_COMM_WORLD );
   MPI_Finalize();
   return 0;
}
```

`MPI_Init, MPI_Comm_size, and MPI_Comm_rank` build the header. `MPI_Finalize` is the trailer. All commands start with `MPI_`. MPI always uses pointers to pass variables. MPI enumerates all nodes starting with 0 (rank of a node). Size gives you the number of nodes available.

6) A First MPI Application

Write a simple application that defines a constant π on the first node in a cluster and, afterwards, sends this constant to all other nodes one by one. Use only the MPI operations `MPI_Send` and `MPI_Recv`.
See sourcecode to corresponding tutorial on webpage.