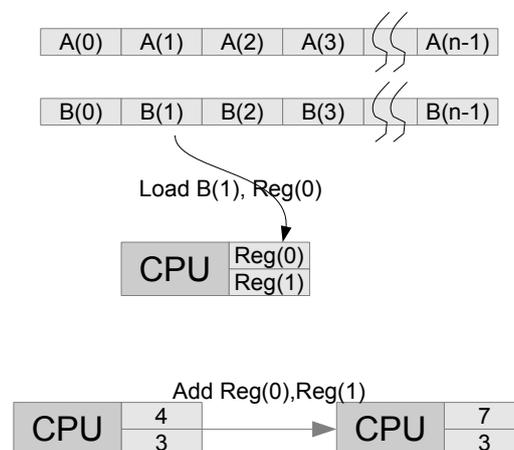


Parallel Numerics

Exercise Sheet 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 ($\text{Reg}(0)$ and $\text{Reg}(1)$) and is able to perform one operation (load, store, add, mult) per clock tick.

The following program computes $A(2) = A(3) + 2 \cdot B(5)$:

```

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

```

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

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

```

1      for i = 0...5
2          load      A(i), Reg(0)
3          load      2, Reg(1)
4          mult
5          load      B(i), Reg(1)
6          add
7          store     Reg(0), A(i)

```

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

c) 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:

```

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

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

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

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

9      add   Reg(0), Reg(1)
10     add   Reg(2), Reg(3) // replace by add2

11     store Reg(0), A(0)
12     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 for addition are saved. Further improvement could be an operation **mult2**. Real computers nowadays e.g. SSE, graphic cards, vector components. The Flynn type is **SIMD**.

d) 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):

```

1  for i = 0...2
2      see exercise a)

```

```

1 for i = 3...5
2     see exercise a)

```

- e) 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/operation. This means the combination

```

1     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

```

1     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:

```

1     load          A(0), Reg(0)
2     load          2, Reg(1)
3     mult / load  B(0), Reg(1)
4     add / load   2, Reg(1)    // already step 2 of second iteration (loop merge)
5     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 Parallel Programming Paradigms

- a) 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).
- b) In exercise 1, the simple computer was made a shared memory machine. Define the terms “shared memory” and “distributed shared memory”.

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 → Computational tasks on local data.

Hybrid: CPU's on node share memory. Nodes only know about own memory but may communicate with other nodes.

3 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.

```
1     int main ( int argc , char* argv[] ) {
2         int myrank , nproz;
3
4         MPI_Init( &argc , &argv );
5         MPI_Comm_size( MPI_COMM_WORLD , &nproz );
6         MPI_Comm_rank( MPI_COMM_WORLD , &myrank );
7
8         MPI_Barrier( MPI_COMM_WORLD );
9         MPI_Finalize();
10
11         return 0;
12     }
```

`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.

4 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 [source code to corresponding tutorial on webpage](#).