# Serie 1 - Solution

**Exercise 1** (Big O notation)**.**

(a) Define in one sentence the time and space complexities for an algorithm.

- *Time complexity : The impact of more data on the time it takes to complete the algorithm (total execution time)*
- *Space complexity : The impact of more data on the memory requirements to complete the algorithm*

(b) Assume you have two algorithms $A_1$ (with time complexity $\mathcal{O}_t(n\,log\,n)$ and space complexity $\mathcal{O}_s(n)$) and $A_2$ (with time complexity $\mathcal{O}_t(n^3)$ and space complexity $\mathcal{O}_s(n^3)$). Which one is worth to be parallelized ?

*Algorithm $A_1$ is close to linear in terms of time and its memory requirements is linear. The second $A_2$ presents a cubic complexity in terms of time and memory requirements. Thus the second ($A_2$) is probably a better candidate.*

**Exercise 2** (Vocabulary)**.**

(a) Why a DAG (Direct Acyclic Graph) is useful for a theoretical analysis of a parallelisation strategy for a particular algorithm ?

*A DAG contains all the tasks involved in the parallel algorithm. Once all dependencies (relations between tasks) have been defined, it is straightforward to point the independent tasks that can be executed in parallel.*

(b) What is the critical path in a DAG of tasks with their dependencies ?

*It is the longest execution time path through the tasks.*

(c) What are the two main kind of parallelism ?

- *Data parallelism : any kind of parallelism that grows with the data set size*
- *Functional decomposition : assign tasks to distinct functions in the algorithm*

(d) What is one of the main solutions to solve the problem of "irregular parallelism" ?

*Introducing load balancing : all the tasks involved in the parallel computation should have the same amount of work to perform.*

**Exercise 3** (Machine model)**.**

(a) What ILP, TLP and DLP stands for ?

- *ILP : Instruction-level parallelism (how many instructions can be executed simultaneously on a CPU)*
- *TLP : Thread-level (or Task-level) parallelism (how many tasks can be executed simultaneously on the same data)*
- *DLP : Data-level parallelism (or vector parallelism : one instruction is executed on a vector of data in one clock cycle)*

(b) What is a vector intrinsic ?

*An instruction of the processor (CPU) that performed on a vector of data. For instance, the Intel intrinsic* `_mm_fmadd_pd(a,b,c)` *performs a fused multiply-add ($a = a + (b \times c)$) on 128 bits long vectors (2 double precision floating points)*

(c) Classify the memories according to their speed of access : DRAM, L2,L1,L3,SSD ?

*L1, L2, L3, DRAM, SSD*

(d) Give the 4 types of parallel architectures according to Flynn's taxonomy and an example of each.

| | *Single Data* | *Multiple Data* |
|---|---|---|
| *Single Instruction* | *SISD* | *SIMD* |
| *Multiple Instruction* | *MISD* | *MIMD* |

- *SISD : one instruction in one core of a CPU*
- *SIMD : vector registers (like Intel AVX))*
- *MISD : very rare !!*
- *MIMD : Distributed memory machines like clusters*

**Exercise 4** (Threads, processes and the OS)**.**

(a) What are the main differences between a thread and a process ?

| *Process* | *Thread* |
|---|---|
| *each process has its own memory space* | *Each thread uses the process' memory* |
| *heavyweight operations* | *leightweight operations* |
| *context switching is expensive* | *context switching is cheaper* |
| *No memory sharing between processes* | *Each thread uses the process' memory* |

**Exercise 5** (Performance aspects)**.**

(a) What is the definition of the speedup ?

*If $T_1$ is the (best) sequential execution time and $T_p$ the execution time on $p$ processes, then the speedup is defined by $S_p = \frac{T_1}{T_p}$*

(b) What is the definition of the parallel efficiency ?

*If $T_1$ is the (best) sequential execution time and $T_p$ the execution time on $p$ processes, then the parallel efficiency is defined by $E = \frac{T_1}{p\ T_p} = \frac{S_p}{p}$*

(c) What is the definition of the Amdahl's law (with the serial part $f$ and the serial execution time $T_1$) and what does this law measures ?

*The Amdahl's law defines the maximum speedup as $S = \frac{T_1}{f\ T_1 + (1-f)\ \frac{T_1}{p}} = \frac{p}{(1+(p-1)f)}$. It measures the maximum speedup a parallelized code can achieve by keeping the amount of work constant and by increasing the number of processes. This is called strong scaling.*

(d) What is the definition of the Gustafsson's law (with the non-parallelizable part $\alpha$ and the portion of parallelized part $P$) and what does this law measures ?

*The Gustafsson's law defines the maximum speedup as $S(P) = P - \alpha(P-1)$. It measures the maximum speedup a parallelized code can achieve by keeping the amount of work constant PER PROCESS and by increasing the number of processes. This is called weak scaling.*

**Exercise 6** (Performance aspects on 2D Poisson solver)**.**

Introducing remark : the communication between processors (steps 2 and 3) is a mechanism to transfer data from one processor to/from another : namely parallel processing. We'll have much insight into several libraries and tools in the next weeks to learn how to program that.

(a) compute the computational complexity ("big O" notation) of the algorithm (in **time**)

*The complexity in time is $\mathcal{O}(n^2)$. If the size of the grid is doubled, the number of iterations is 4 times bigger.*

(b) compute the computational complexity ("big O" notation) of the algorithm (in **space**)

*The complexity in space is $\mathcal{O}(n^2)$. If the size of the grid is doubled, the memory requirements is 4 times bigger.*

(c) is this problem worth to be parallelized ?

*Due to the computational complexities in $\mathcal{O}(n^2)$, the 2D Poisson problem is worth to be parallelized.*

**Exercise 7** (Theoretical analysis : Amdhal's law)**.**

(a) give an estimation of $f$ (the sequential part of the code that can not be parallelized).

*$f$ is defined by the sequential part of the code. In this example, we assume it is only representing the initialization phase. Thus, By computing $\frac{t_{init}}{t_{total}}$ we find approximatively $f = 1\%$ which is the non-parallelizable (serial) part of the code.*

(b) What is the upper bound of the speedup in the case of Amdahl's law ?

*With approximatively 1 % of serial part, the speedup seems to be close to the perfect case.*

**Exercise 8** (Theoretical analysis : Gustafsson's law)**.**

(a) What would be the maximum efficiency of this parallel 2D poisson solver at 128 processors ?

*Close to 100 % .*