# Exercises for the Lecture: “Architecture and Programming Models for GPUs and Coprocessors” Exercise Sheet № 3

Priv.-Doz. Dr. rer. nat. Stefan Zellmann

SS 2021

## 3 Performance and Parallel Algorithms

### Ex. 3.1

We are given a hypothetical CPU with the following instruction set:

| **Instruction** | **Description** | **Latency** |
|---|---|---|
| `ADD $R1` $C$ | Add the konstant value $C$ to the content of register `$R1` | 4 cc |
| `LD $R1` $[S]$ | Load the content at memory cell $[S]$ into register `$R1` | 10 cc |
| `ST` $[S]$ `$R1` | Store the content of register `$R1` to memory cell $[S]$ | 10 cc |
| `JNZ $R1 label1` | If content of register `$R1` $\neq 0$, jump to label `label1` | 1 cc |

Determine the performance of the following program that runs on a CPU whose clock frequency is 100 Hz. For that we assume that the contents of `$R1` and `$R2` are 0 initially and that the memory cell at address `0x80` initially holds the value 12.

```
LD $R2 0x80
L1:
ADD $R1 3
ADD $R2 -4
JNZ $R2 L1
ST 0x80 $R1
```

## Ex. 3.2

We are assigned the task of optimizing a C program for multi-core architectures. The function that dominates the execution time of the program is the following one.

```
void func(double* x, double* y, double* a, double* b, int N)
{
    assert(N > 2);

    int i;
    for (i = 2; i < N; ++i)
        x[i] = x[i-1] + x[i-2];

    for (i = 0; i < N; ++i)
        y[i] = x[i] * a[i] / b[i];
}
```

We know that the pointers that are passed to `func()` will not *alias*—i.e., the memory regions that they are pointing to are guaranteed to not overlap. This knowledge and our thorough analysis of the program lead us to the conclusion that the first loop cannot be parallelized due to an *inter iteration* data dependency. The second loop can however be parallelized using a simple OpenMP `parallel for` clause so that multiple threads can concurrently execute the loop on a multi-core processor. We have analyzed the program using a profiler and real-world input data and found out that—with serial execution—the same amount of time is spent in the first and the second loop.

Assume for simplicity that a parallelization of the second loop would result in a speedup of $N$ when being executed on $N$ processor cores ("perfect scaling"). Under those circumstances, determine an upper bound for the *theoretical speedup* of the function `func()`, for $N = 2$, $N = 4$, and $N = 8$ processor cores.

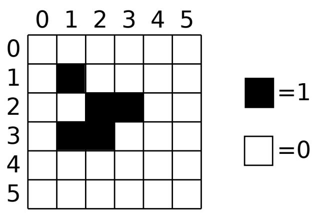
## Ex. 3.3

*Cellular automata* are grid-based models where the grid cells are defined by their *state*. That state changes over time and based on the states of the cell's neighbors. State transitions occur due to a set of rules.

We are given a cellular automaton on a uniform cartesian grid of size $N \times M$, $N, M \in \mathbb{N}$, where $N$ denotes the width and $M$ denotes the height of the grid $\Rightarrow$ each cell can be identified uniquely based on its position $(x, y) \in [0, N) \times [0, M)$, $x, y \in \mathbb{N}$. Cells store their state $s(x, y, t) \in \{0, 1\}$ for every time step $t \in \mathbb{N}$. Let $\mathfrak{S}(x, y, t)$ be the number of adjacent cells at $(x, y)$ and time $t$ for which $s = 1$. In order to determine $s(x, y, t)$ for this cell, we apply the following set of rules:

$$s(x,y,t) = \begin{cases} 1 \text{ if } (s(x,y,t-1) = 1 \wedge \mathfrak{S}(x,y,t-1) = 2) \vee \mathfrak{S}(x,y,t-1) = 3 \\ 0 \text{ otherwise.} \end{cases}$$

**a.)** Determine and draw the next four time steps for the following grid.



**b.)** Formulate a PRAM algorithm that uses an infinite loop to compute the state of the automaton. For that you can make use of $P = S \times T$ processors, where $N \bmod S := 0$ and $M \bmod T := 0$. Assume that all the *boundary cells* have value 0. You should use two separate grids $G_1$ and $G_2$ to maintain the states of two consecutive time steps. $G_1$ will store the state of the previous time step and $G_2$ that of the current one. Name the memory model ("EREW", "CREW", "CRCW") of the PRAM that is required.

**c.)** Formulate the same algorithm using the *work-time paradigm* for an infinite number of processors.

The exercise sheet will be discussed on May 20, 2021.