CS 340 – Notes for Chapter 6.1 – Parallel Algorithms

[this topic is not covered in the textbook]

so far: serial algorithms, i.e., only one instruction executed at a time ⇝ suitable for uniprocessor computers

parallelism: <u>some</u> (not all!) algorithmic problems can be solved faster when using multiple processors in parallel.

cf. house construction contests: with teams of 200 people, a house can be constructed in a few hours

however, while a couple can have a baby in 9 months, you cannot use 9 couples to have a baby in 1 month.

approaches to parallelism:

- shared memory: each processor can directly access the whole memory
- distributed memory: each processor has its own private memory for which it can grant access to other processors

(• distributed shared memory: large shared memory, plus private memory section for each processor)

⇝ no consensus on a single model of formal study and analysis of parallel algorithms

we will assume shared memory and (unrealistically!) that

- memory access costs constant time
- unlimited number of processors are available

[note: one could easily teach a whole course on the design and analysis of parallel algorithms! We here just get a very superficial glimpse!]

simple examples of parallel processing:

<u>EXAMPLE 48</u> summing N numbers $a[0], a[1], \ldots, a[N-1]$

standard serial procedure:

$$((\ldots((a[0]+a[1])+a[2])+\ldots)+a[N-1])$$

$N-1$ additions in $\underline{N-1 \text{ stages}}$

parallelize:

(1) use $\lceil \frac{N}{2} \rceil$ processors to compute

$a[0]+a[1]$, $a[2]+a[3]$, $a[4]+a[5]$, ...

(2) use $\lceil \frac{N}{4} \rceil$ processors to compute pairwise sums of these $\lceil \frac{N}{2} \rceil$ numbers

(3) ...

$\leadsto$ $\underline{\lceil \log_2 N \rceil \text{ stages}}$

$\leadsto$ the parallel approach uses more operations in total (<u>more "work"</u>), but in <u>less time</u>.

<u>EXAMPLE 49.</u> parallel version of Mergesort

The two recursive calls of Mergesort can be executed on separate processors, as they do independent work ...

Can parallel algorithms solve unsolvable problems, e.g., the Halting Problem?

NO. Each parallel algorithm can be turned into a sequential one, i.e., simulated on a uniprocessor machine.

Can parallel algorithms turn intractable problems into tractable ones?

Using an <u>exponential number of processors</u>, each NP-complete problem can be solved in polynomial time, e.g., for the Traveling Salesman Problem: each processor generates one candidate witness and checks it.

$\rightsquigarrow$ still not a practically feasible solution, because of the required number of processors.

Famous problematic incidents caused by improper use of parallel algorithms:

- Therac-25: radiation therapy machine, developed by Atomic Energy of Canada Ltd., mid 1980s
  → killed 3 patients and seriously injured others by giving large overdoses of radiation
- Northeast blackout of 2003 (Ontario, Northeastern + Midwest US States)
  ~ 55 million people affected
  caused almost 100 deaths

both incidents happened due to "<u>race conditions</u>": two parallel instructions access the same memory, and at least one of them overwrites it.

→ does processor 2 access the value of x BEFORE or AFTER processor 1 incremented x?

```
x=0;
parallel for i=1 to 2
        x=x+1;
print x;
```

could result in x = 1 or in x = 2

→ extra caution needed in parallel computing!!!