
Scientific Computing 2

Tutorial 1

Handout: Apr. 10th, 2019

Exercise 1:

We consider the following set of (pseudo-) algorithms from which the following properties are known with respect to the input data size n :

	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4
Number of Flops	$815n$	$2n^2 + n$	$6n^3$	$15n^3$
Percentage of parallelizeable code	90%	99%	15%	80%

- Compute the upper bounds for the speedup of all three algorithms.
- Consider an input of size $n = 10\,000$ and a CPU with a performance of 18 GFlop/s. Compute the theoretical runtime and the speedup if the work is distributed to 1, 2, 4 or 8 of these CPUs in parallel.

Exercise 2:

We consider the following pseudo code algorithms. For each of them count the number of necessary flops in dependence of the input data size. Identify parallelizable regions of the algorithms and determine the maximum theoretical speed up on a parallel machine. If the algorithm seems to be strictly sequential there is often a possibility to easily modify it such that parts of the algorithm can be done in parallel. Identify these parts and give an alternative formulation which is easier to parallelize.

- The scalar (dot) product:

Input: $x \in \mathbb{R}^n, y \in \mathbb{R}^n$

Output: $s = (x, y) = x^T y$

1: $s := 0$

2: **for** $i := 1, \dots, n$ **do**

3: $s := s + x_i y_i$

4: **end for**

- The general matrix-matrix product (GEMM):

Input: $A \in \mathbb{R}^{n \times k}, B \in \mathbb{R}^{k \times m}, C \in \mathbb{R}^{n \times m}$

Output: $C := C + AB$

1: **for** $i := 1, \dots, n$ **do**

2: **for** $j := 1, \dots, m$ **do**

3: **for** $l := 1, \dots, k$ **do**

4: $C_{ij} := C_{ij} + A_{il} B_{lj}$

5: **end for**

6: **end for**

7: **end for**

c.) The LU decomposition without pivoting:

Input: $A \in \mathbb{R}^{n \times n}$

Output: $LU := A$, A is overwritten by L and U

- 1: **for** $k = 1, \dots, n - 1$ **do**
- 2: $A_{k+1:n,k} := A_{k+1:n,k} / A_{k,k}$
- 3: $A_{k+1:n,k+1:n} := A_{k+1:n,k+1:n} - A_{k+1:n,k} A_{k,k+1:n}$
- 4: **end for**

d.) Forward substitution $Lx = b$ (Backward substitution has the identical properties.):

Input: $L \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$

Output: $x \in \mathbb{R}^n$, $x = L^{-1}b$, b is overwritten by x

- 1: $b_1 := b_1 / L_{1,1}$
- 2: **for** $i := 2, \dots, n$ **do**
- 3: $b_i := b_i - L_{i,1:i-1} b_{1:i-1}$
- 4: $b_i := \frac{b_i}{L_{i,i}}$
- 5: **end for**

e.) Solving a linear system $Ax = b$, $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^{n \times n}$ and $x \in \mathbb{R}^{n \times n}$. Use the LU -decomposition from c.) and the forward/backward substitution from d.).

Exercise 3:

The *vector-triad* benchmark is a small benchmark example to check the test the performance of a computer systems for bandwidth bound problems. For a given set of vectors $a \in \mathbb{R}^n$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, and $d \in \mathbb{R}^n$ it computes

$$a = b + c \otimes d,$$

where \otimes denotes the element-wise product.

a.) Compute the runtime of the vector-triad when we execute it sequentially on a CPU with a performance of 12.8 GFlops/s for different vector lengths n

Length n of a, b, c, d	Runtime in s
$n = 10\,000$	
$n = 100\,000$	
$n = 1\,000\,000$	
$n = 100\,000\,000$	

b.) Explain why the practical results shown in the exercise are much different from the theoretical ones. What is might be the problem when we analyze the theoretical performance of algorithms in this simple way?

c.) It is obvious that the vector-triad is perfectly parallelizable. Why does even employing a second CPU-core with an additional performance of 12.8 GFlops/s not result in a nice performance gain?