Scientific Computing - Exercise Sheet 4

Jonathan Hellwig, Jule Schütt, Mika Tode, Giuliano Taccogna

May 11, 2021

1 Exercise

(a) Noch nichts gemacht:

```
**Input**: $\mathbf{A} \in \mathbb{R}^{n \times n} \quad b, x_0 \in \mathbb{R}^n$
1: $h_0 = \mathbf{A}x_0$
2: $r_0 = b - h_0$
3: $p_0 = r_0$
4: $\beta_0 = r_0^T \cdot r_0$
5: **for** $k = 1, 2, \dots$ **do**
6:     $h_{k-1} = \mathbf{A}p_{k-1}$    (matrix multiplication)
7:     $\gamma_{k-1} = p_{k-1}^T \cdot h_{k-1}$    (1. loop)
8:     $\alpha_{k-1} = \frac{\beta^{k-1}}{\gamma^{k-1}}$
9:     $x_k = x_{k-1} + \alpha_{k-1}p_{k-1}$    (2 loop)
10:    $r_k = r_{k-1} - \alpha_{k-1}h_{k-1}$    (3. loop)
11:    $\beta_k = r_k^T \cdot r_k$    (4. loop)
12:    $p_k = r_k + \frac{\beta_k}{\beta_{k-1}}p_{k-1}$    (5. loop)
13: **end for**
```

(b) The idea is to split the vectors such that the first coordinates were computed by one processor and the last coordinates were computed by the second processor. Therefore i denotes the counter, which is needed for the loop operation. Notice that for the Vectormultiplications in the loops 1 and 4 normally also a **reduce** operation should be included.

```
**Input**: $\mathbf{A} \in \mathbb{R}^{n \times n} \quad b, x_0 \in \mathbb{R}^n$
1: $h_0 = \mathbf{A}x_0$
2: **begin parallel private**$(i, r_0)$ **shared**$(b, h_0)$
3: $r_0 = b - h_0$
4: **end parallel**
5: $p_0 = r_0$
6: **begin parallel private**$(i, \beta_0)$ **shared**$(r_0)$ **end parallel**
7: $\beta_0 = r_0^T \cdot r_0$
8: **for** $k = 1, 2, \dots$ **do**
9:     $h_{k-1} = \mathbf{A}p_{k-1}$    (matrix multiplication)
```

```
10:     **begin parallel private**$(i, \gamma_{k-1})$ **shared**$(p_{k-1}, h_{k-1})$
11:     $\gamma_{k-1} = p_{k-1}^T \cdot h_{k-1}$        (1. loop)
12:     **end parallel**
13:     $\alpha_{k-1} = \frac{\beta^{k-1}}{\gamma^{k-1}}$
14:     **begin parallel private**$(i, x_k, p_k)$ **shared**$(r_{k-1}, r_k, x_{k-1}, h_{k-1}, \alpha_{k-1}, \beta_k, \beta_{k-1}, p_{k-1})$
15:     $x_k = x_{k-1} + \alpha_{k-1} p_{k-1}$        (2 loop)
16:     $r_k = r_{k-1} - \alpha_{k-1} h_{k-1}$        (3. loop)
17:     $\beta_k = r_k^T \cdot r_k$                   (4. loop)
18:     $p_k = r_k + \frac{\beta_k}{\beta_{k-1}} p_{k-1}$        (5. loop)
19:     **end parallel**
20: **end for**
```

(c) We determine that for all parallelizable parts Processor 0 works on the for-loop for $i = 1, ..., n/2$ and Processor 1 works on the for-loop for $i = n/2 + 1, ..., n$, with $n/2$ possibly rounded.
Accordingly $v^{(j)}$ denotes a scalarproduct or a vector which is calculated by processor $j = 0, 1$. If $v^{(j)}$ denotes a Vector we mean $v^{(0)} = (v(1), v(2), ..., v(n/2))^T$ and $v^{(1)} = (v(n/2+1), v(n/2+2), ..., v(n))^T$. If $v^{(j)}$ denotes a scalarproduct, we mean $v^{(j)} = H^{T(j)} \cdot K^{(j)}$ for $H, K \in \mathbb{R}^n$.
If a processor recives data through the **recv** command, we implicitly include in **recv** that the recieved data is combined with the data that was calculated on the processor itself in the right way.
That means if one half of a vector $v^{(i)}$ is recieved **recv** creates the whole vector $v = v^{(0)} + v^{(1)}$. If "one half" of a scalarproduct is recived **recv** adds the corresponding scalarproducts to the final scalarproduct. If the processor is not specified in a calculation, the calculation is executed on both processors.
For the first processor (index 0):
Input: $\mathbf{A} \in \mathbb{R}^{n \times n}$ $\quad b, x_0 \in \mathbb{R}^n$

```
1: $h_0 = \mathbf{A} x_0$
2: $r_0^{(0)} = b^{(0)} - h_0^{(0)}$
3: **barrier**
4: **send**$(r_0^{(0)})$
5: **recv**$(r_0^{(1)})$
6: $p_0 = r_0$
7: $\beta_0^{(0)} = r_0^{(0)T} \cdot r_0^{(0)}$
8: **barrier**
9: **send**$(\beta_0^{(0)})$
10: **recv**$(\beta_0^{(1)})$
11: **for** $k = 1, 2, \ldots$ **do**
12:     $h_{k-1} = \mathbf{A} p_{k-1}$        (matrix multiplication)
13:     $\gamma_{k-1}^{(0)} = (p_{k-1}^{(0)})^T \cdot h_{k-1}^{(0)}$        (1. loop)
14:     **barrier**
15:     **send**$(\gamma_{k-1}^{(0)})$
```

```
    **recv**($\gamma_{k-1}^{(1)}$)
    $\alpha_{k-1} = \frac{\beta^{k-1}}{\gamma^{k-1}}$
    $x_k^{(0)} = x_{k-1}^{(0)} + \alpha_{k-1}^{(0)} p_{k-1}^{(0)}$        (2 loop)
    $r_k^{(0)} = r_{k-1}^{(0)} - \alpha_{k-1}^{(0)} h_{k-1}^{(0)}$        (3. loop)
    $\beta_k^{(0)} = (r_k^{(0)})^T \cdot r_k^{(0)}$        (4. loop)
    **barrier**
    **send**($\beta_k^{(0)}$)
    **recv**($\beta_k^{(1)}$)
    $p_k^{(0)} = r_k^{(0)} + \frac{\beta_k^{(0)}}{\beta_{k-1}^{(0)}} p_{k-1}$        (5. loop)
    **barrier**
    **send**($p_k^{(0)}$)
    **recv**($p_k^{(1)}$)
**end for**
```

(d) To compare the performance of the algorithm of exercise sheet 3 and 4 we need to specify the paradigm. We are looking at the message passing model, so we have to consider exercise c) of both sheets.
The CG algorithm runs for 100 seconds on one processor, i.e.

$$t_1^N = 100.$$

Thus,

$$t_p^N = \frac{100}{p}.$$

Further the communication time is given by

$$t = 0.01$$

per processor and send/rec pair. We consider $p \in \{10, 50, 100\}$.
We are mainly looking on the for loop in both algorithm since we don't know how many times we have to do the for loop we can not compute the serial program fraction ν in a suitable way while looking at the start part. Anyway, the for loop needs much more time and we can disregard the other part.
For the first algorithm on exercise sheet 3 we count 2 of 9 instructions which can not be parallelized, so

$$\nu^1 = \frac{2}{9}.$$

For the second algorithm we count 1 of 7 parts which can not be parallelized, so

$$\nu^2 = \frac{1}{7}.$$

Since $\nu^1 > \nu^2$, we expect that the second algorithm is faster and more effective.
The amount of pairs of send/rec can be counted: $x^1 = 2,\ x^2 = 3$. Hence, the total communication overhead is given by

$$t_c^i = tx^ip \qquad i \in \{1,2\}.$$

Finally we can compute the total run time and the speedup and efficiency by the following formulas:

$$\begin{aligned}(t_{total}^{\nu})^i &= \nu^i t_1^N + (1-\nu^i)t_p^N \\ S_p^i &= \frac{t_1^N}{(t_{total}^N)^i + t_c^i} \\ E_p^i &= \frac{t_1^N}{p((t_{total}^{\nu})^i + t_c^i)}.\end{aligned}$$

We approximate the numbers to the second decimal place:

p	$(t_{total}^{\nu})^1$	S_p^1	E_p^1
10	30	3.31	$0,33$
50	$23,78$	4.04	0,08
100	23	4	0,04

p	$(t_{total}^{\nu})^2$	S_p^2	E_p^2
10	$22,86$	$4,32$	0,43
50	16	$5,71$	0,11
100	$15,14$	$5,51$	0,06