Scientific Computing - Exercise Sheet 4

Jonathan Hellwig, Jule Schütt, Mika Tode, Giuliano Taccogna

May 17, 2021

1 Exercise

(a) We have the following CG algorithm this time:

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** $k = 1, 2, \ldots$ **do**
6: $\quad h_{k-1} = \mathbf{A}p_{k-1}$ (matrix multiplication)
7: $\quad \gamma_{k-1} = p_{k-1}^T \cdot h_{k-1}$ (1. loop)
8: $\quad \alpha_{k-1} = \frac{\beta^{k-1}}{\gamma^{k-1}}$
9: $\quad x_k = x_{k-1} + \alpha_{k-1}p_{k-1}$ (2. loop)
10: $\quad r_k = r_{k-1} - \alpha_{k-1}h_{k-1}$ (3. loop)
11: $\quad \beta_k = r_k^T \cdot r_k$ (4. loop)
12: $\quad p_k = r_k + \frac{\beta_k}{\beta_{k-1}}p_{k-1}$ (5. loop)
13: **end for**

All sections marked as a loop can be parallelised as well as the lines 2 and 4 (see b)). When considering the CREW-P-RAM machine, we cannot write at the same value with more than one processor at the same time (exclusive writing). Therefore, we have to make sure that in the first and fourth loop (where a scalar is computed!) we only write with one processor.
Data dependencies occur in the second, third and fifth loop, as a previous value of the one currently calculated is needed for calculation (e.g. in the second loop x_{k-1} is needed for computation of x_k). One has to make sure not to overwrite the old value before computing the next one.

(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, β_0) **shared**(r_0)
7: $\beta_0 = r_0^T \cdot r_0$
8: **end parallel**
9: **for** $k = 1, 2, \ldots$ **do**
10: $h_{k-1} = \mathbf{A}p_{k-1}$ (matrix multiplication)
11: **begin parallel private**(i, γ_{k-1}) **shared**(p_{k-1}, h_{k-1})
12: $\gamma_{k-1} = p_{k-1}^T \cdot h_{k-1}$ (1. loop)
13: **end parallel**
14: $\alpha_{k-1} = \frac{\beta^{k-1}}{\gamma^{k-1}}$
15: **begin parallel private**(i, x_k, p_k) **shared**$(r_{k-1}, r_k, x_{k-1}, h_{k-1}, \alpha_{k-1}, \beta_k, \beta_{k-1}, p_{k-1})$
16: $x_k = x_{k-1} + \alpha_{k-1}p_{k-1}$ (2 loop)
17: $r_k = r_{k-1} - \alpha_{k-1}h_{k-1}$ (3. loop)
18: $\beta_k = r_k^T \cdot r_k$ (4. loop)
19: $p_k = r_k + \frac{\beta_k}{\beta_{k-1}}p_{k-1}$ (5. loop)
20: **end parallel**
21: **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.
If the processor is not specified in a calculation, the calculation is executed on both processors.
The command **barrier** in brackets is optional since the send/recv synchronizes the processors automatically.
For fulfilling the scalar product one has to add the addition of every summand which were computed on different processors command after the parallelized computation.
Pseudocode for the processor with index $i \in \{0, 1\}$:
Input: $\mathbf{A} \in \mathbb{R}^{n \times n} \quad b, x_0 \in \mathbb{R}^n$

1: $h_0 = \mathbf{A}x_0$ (matrix multiplication parallel with processor $i - 1$)
2: $r_0^{(i)} = b^{(i)} - h_0^{(i)}$
3: (**barrier**)
4: **send**$(r_0^{(i)})$
5: **recv**$(r_0^{(1-i)})$
6: $p_0 = r_0$
7: $\beta_0^{(i)} = r_0^{(i)T} \cdot r_0^{(i)}$
8: (**barrier**)

9: **send**$(\beta_0^{(i)})$
10: **recv**$(\beta_0^{(1-i)})$
11: $\beta_0 = \beta_0^{(i)} + \beta_0^{(1-i)}$
12: **for** $k = 1, 2, \ldots$ **do**
13: $h_{k-1} = \mathbf{A}p_{k-1}$ (matrix multiplication parallel with processor $i-1$)
14: $\gamma_{k-1}^{(i)} = (p_{k-1}^{(i)})^T \cdot h_{k-1}^{(i)}$ (1. loop)
15: (**barrier**)
16: **send**$(\gamma_{k-1}^{(i)})$
17: **recv**$(\gamma_{k-1}^{(1-i)})$
18: $\gamma_{k-1}^{(i)} + \gamma_{k-1}^{(1-i)}$
19: $\alpha_{k-1} = \frac{\beta_{k-1}}{\gamma_{k-1}}$
20: $x_k^{(i)} = x_{k-1}^{(i)} + \alpha_{k-1}^{(i)} p_{k-1}^{(i)}$ (2 loop)
21: $r_k^{(i)} = r_{k-1}^{(i)} - \alpha_{k-1}^{(i)} h_{k-1}^{(i)}$ (3. loop)
22: $\beta_k^{(i)} = (r_k^{(i)})^T \cdot r_k^{(i)}$ (4. loop)
23: (**barrier**)
24: **send**$(\beta_k^{(i)})$
25: **recv**$(\beta_k^{(1-i)})$
26: $\beta_l^{(i)} + \beta_k^{(1-i)}$
27: $p_k^{(i)} = r_k^{(i)} \frac{\beta_k^{(i)}}{\beta_{k-1}^{(i)}} p_{k-1}$ (5. loop)
28: (**barrier**)
29: **send**$(p_k^{(i)})$
30: **recv**$(p_k^{(1-i)})$
31: **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

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

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

The second algorithm needs a smaller computing time (total run time) throughout the different numbers of processors compared to the first algorithm, which aligns with our expectation and the fact that a larger fraction of the second algorithm can be parallelized. Furthermore, the effective speedup and efficieny for the second algorithm is larger than the

effective speedup and efficiency values for Algorithm 1 for each number of processors p.
As we have seen on the second exercise sheet, adding more and more processors does not always increase effective speedup and efficiency infinitely, e.g. for $p = 50$ we have a higher speedup in both algorithms compared to their respective speedup and efficiency values for $p = 100$.