Umeå University
Department of Computing Science

Parallel Programming 7.5 p
5DV152

Exercises, Chapter/Topic 2

Submitted	2017-02-02
Author:	Lorenz Gerber (`dv15lgr@cs.umu.se` `lozger03@student.umu.se`)
Instructor:	Lars Karlsson / Mikael Ränner

Contents

1 Introduction

This report is part of the mandatory coursework. It describes the solutions for several chosen exercises from the course book [2].

2 2.4 - Counting pages

The 12 lower bits are used for within page adressing and the 20 upper bits of the 32 bit address are used for pages. With 20 bits, 2^{20} pages (1'046'576) can be addressed. 12 bits corresponds to a page size of 4Kb. Hence the total virtual memory is 4Gb.

3 2.8 - Hardware multithreading and caches

Caching operates on whole cache lines. Hence, the chance that another process/thread changes something in a specific cache line increases with cache size and number of processes/threads. The specific situation that can happen is called 'false sharing': When one process changes data in a cache line, there is no possibilty to check or know for another process which data exactly was changed. It can very well be the case that to the current process unrelated data was changed and a cache reload would not be needed. However, cache has to be reloaded.

4 2.10 - Communication overhead

The calculations can be split in two parts: the instructions and the communication. For both a) and b) the instructions will take the same time:
`instructions / cores / instructions_per_second`
Hence $10^{12} \div 1000 \div 10^{6} = 1000sec$. For communications, the time is calculated as
`messages_to_send * time_per_message`
while messages to send is described as $10^{9}(p-1)$.

a) if sending one message takes 10^{-9} seconds, the communication with 1000 cores will take: $10^{9}(1000-1) \times 10^{-9} = 999sec$, together with the actual calculation, $1'999sec = 33.32min$.

b) if sending one message takes 10^{-3} seconds , the communication with 1000 cores will take: $10^{9}(1000-1) \times 10^{-3} = 9.99 \times 10^{8}s$, or 31.7 years...

ah

5 2.16 - Speedup efficiency

To calculate the hypothetical speed up between a serial and parallel program with the run time formulas $T_{serial} = n^2$ and $T_{parallel} = \frac{n^2}{p} + log_2(p)$ where $T = [ms]$, a short program was written that calculates the speedup $S = \frac{T_{serial}}{P_{parallel}}$. The obtained data was then plotted (*figure 1*) using the R statistical environment [3]. Source code for the C program to generate data and R script for the graphs is accessible on github [1].

a) Figure 1 shows graphs for constant `n` (upper graph) and constant `p` (lower graph) respectively. From the upper graph, it can be seen that for constant `p`, the growth of speedup for larger and larger problem instances `n` decreases and eventually closes in to a horizontal asymptote. The lower graph shows, that for large problem sizes `n`, increasing the number of cores `p` leads to almost linear speedups, while when we increase the number of cores for small problem instances, the speed up levels off very soon.

b) Efficency is defined as $E = \frac{T_{serial}}{pT_{parallel}}$. In the current question, $T_{parallel}$ is defined as $\frac{T_{serial}}{p} + T_{overhead}$. Hence efficency E translates here to $\frac{T_{serial}}{p(\frac{T_{serial}}{p}+T_{overhead})} = \frac{T_{serial}}{T_{serial}+pT_{overhead}}$. From the last expression, it can be seen, that if p is held constant and T_{serial} grows faster than $T_{overhead}$, the ratio will increase, hence the efficency increases. The opposite follows, that if $T_{overhead}$ grows monotoically faster than T_{serial}, it will sooner or later dominate the denominator, the ratio will decrease, hence the efficency decrease.

6 2.19 - Scalability

The efficiency for the given T_{serial} and $T_{parallel}$ can be expressed as:

$$E = \frac{n}{p(\frac{n}{p} + log_2(p))} = \frac{n}{n + p \times log_2(p)}$$

Now we are looking for a factor x such that E stays constant while we increase p with a factor k:

$$\frac{n}{n + p \times log_2(p)} = \frac{n \times x}{n \times x + k \times p \times log_2(k \times p)}$$

This equation can be solved for x:

$$x = \frac{k \times log_2(k \times p)}{log_2(p)}$$

The found expression is the sought factor for x:

$$\frac{n\frac{k \times log_2(k \times p)}{log_2(p)}}{n\frac{k \times log_2(k \times p)}{log_2(p)} + k \times p \times log_2(k \times p)}$$

a) The factor for n is $\frac{k \times log_2(k \times p)}{log_2(p)}$

b) If $p = 8$ then $16 = p \times 2$, hence $k = 2, p = 8$. Filling these values into the expression from *a)*, the numeric result is $\frac{8}{3}$.

7 2.20 - Linear speedup and strong scalability

A linear speed-up is when a parallel program using p cores runs p times faster than the serial one. So yes, by definition, if a parallel program has a linear speed-up, it is strongly scalable. However, in practice it will hardly be obtainable to have a perfect linear speed-up. In this case, amdahl's law shows that if not 100% of the code can be made parallel, it can be at most weakly scalable.

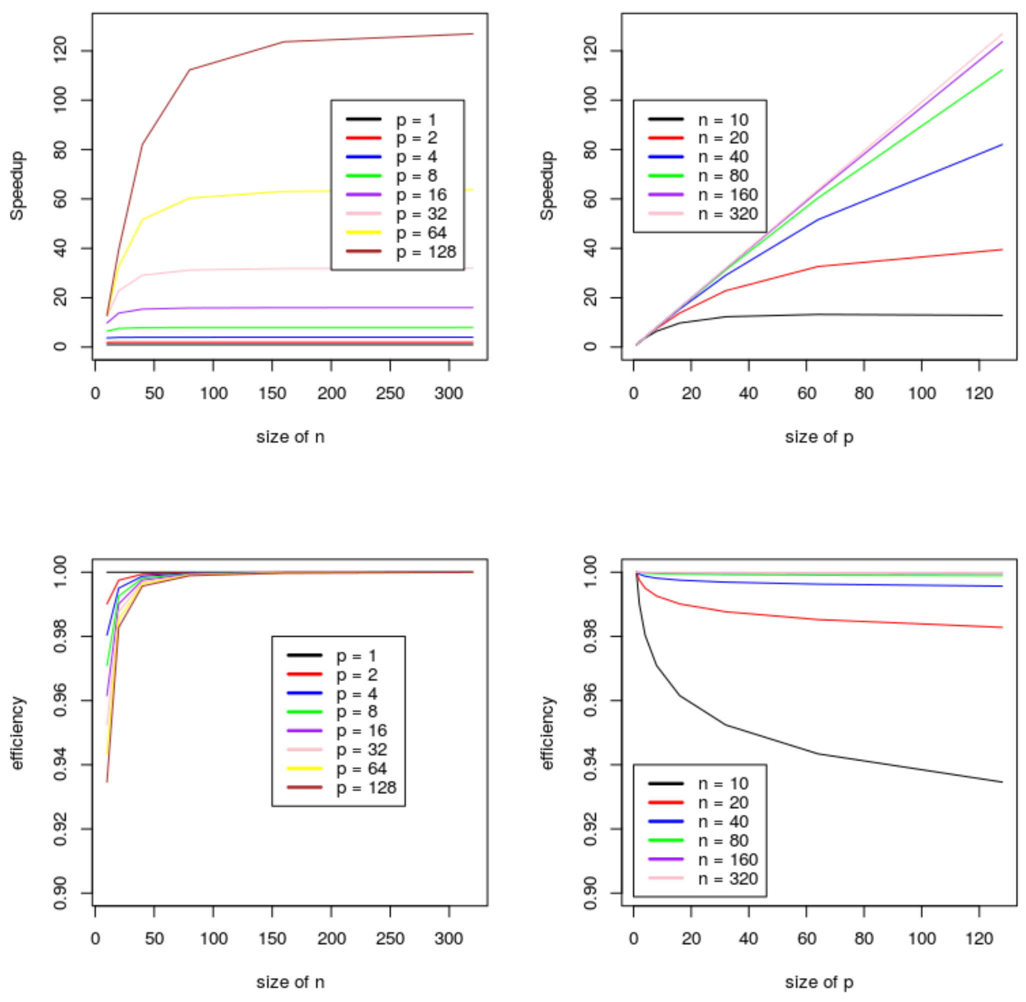

Figure 1: This figure shows the data for exercise 2.16. Upper left, shows the speedup in dependency of problem size n at constant core number p. Lower left, shows efficiency in dependecy of core problem size n at contant core number p. Upper right shows speedup in dependency of core number p at constant problem size n and lower right shows the efficiency in dependency of core numbers for sonstant problem size n.

8 2.23 - Alternative algorithm for computing histogram

Assuming `data_count` c and `bin_counts` b to be $c >> b$, agglomerating over elements of `bin_counts` instead of `data` would result in a suboptimal communication structure: To each core k, the whole set of `data` has to be sent. Scaling up the number of cores p will not reduce this and scaling up c will result in $c \times p$ data to be sent around. In the prior solution, when agglomerating over `data`, the amount of data to be sent to each core decreases when scaling up the core number. Scaling up the data, the amount of data to be sent around is never more than the total `data_count`.

References

[1] Github page of lorenz gerber, project parallel programming reports. `http://github.com/lorenzgerber/ParallelProgrammingReports`. accesses: 2017-02-01.

[2] P.S. Pacheco. *An Introduction to Parallel Programming*. Morgan Kaufman, 2011.

[3] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.