Umeå University
Department of Computing Science

Parallel Programming 7.5 p
5DV152

Exercises, Chapter/Topic 1

Submitted	2017-01-26
Author:	Lorenz Gerber (`dv15lgr@cs.umu.se` `lozger03@student.umu.se`)
Instructor:	Lars Karlsson / Mikael Ränner

Contents

1 Introduction 1
2 2.4 - Counting pages 1
3 2.8 - Hardware multithreading and caches 1
4 2.10 - Communication overhead 1
5 2.16 - Speedup efficiency 1
6 2.19 - Scalability 3
7 2.20 - Linear speedup and strong scalability 3
8 2.23 - Alternative algorithm for computing histogram 3
References 3

1 Introduction

This report is part of the mandatory coursework. It describes the solutions for several chosen exercises from the course book [1].

2 2.4 - Counting pages

2^{20}

3 2.8 - Hardware multithreading and caches

Caching operates on whole cache lines. Hence if 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: $10^9(1000-1) \times 10^{-9} = 99'900sec$, together with the actual calculation, $100'900sec = 28.03h$.

b) if sending one message takes $^{-3}$ seconds , the communication with 1000 cores will take: $10^9(1000-1) \times 10^{-3} = 99'900'000'000sec$, together with the actual calculation, $99'900'001'000sec = 3'167.8years$.

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 [2].

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#define START_N 10
#define START_P 1
#define N_LENGTH 6
#define P_LENGTH 8

int main (void) {

  FILE *f = NULL;
  if((f = fopen("data.txt", "w"))<0){
    perror("fopen");
    exit(EXIT_FAILURE);
  }

  double n = START_N;
  double p = START_P;

  double speedUp = 0;

  for(int i = 0 ; i < N_LENGTH; i++){
    for(int j = 0; j < P_LENGTH; j++){

      speedUp = n*n / ((n * n) /  p  + (log(p)/log(2)));
      fprintf(f, "%lf\t", speedUp);
      printf("SpeedUp for n = %lf p = %lf is %lf\n", n, p, speedUp);
      p*=2;
    }

    fprintf(f, "\n");
    n*=2;
    p = START_P;
  }

  fclose(f);
  return 0;
}
```

a) In table 1, the rawdata is presented, while 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 of 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.

Table 1 *shows the speed ups in relation to p (vertical) and n (horizontal)*

p \n	1	2	3	4	5	6	7	8
1	1	1.96	3.70	6.45	9.76	12.31	13.22	12.85
2	1	1.99	3.92	7.55	13.79	22.86	32.65	39.51
3	1	2	3.98	7.88	15.38	29.09	51.61	82.05
4	1	2	4	7.97	15.84	31.22	60.38	112.28
5	1	2	4	7.99	15.96	31.80	63.05	123.67
6	1	2	4	8	15.99	31.95	63.76	126.89

6 2.19 - Scalability

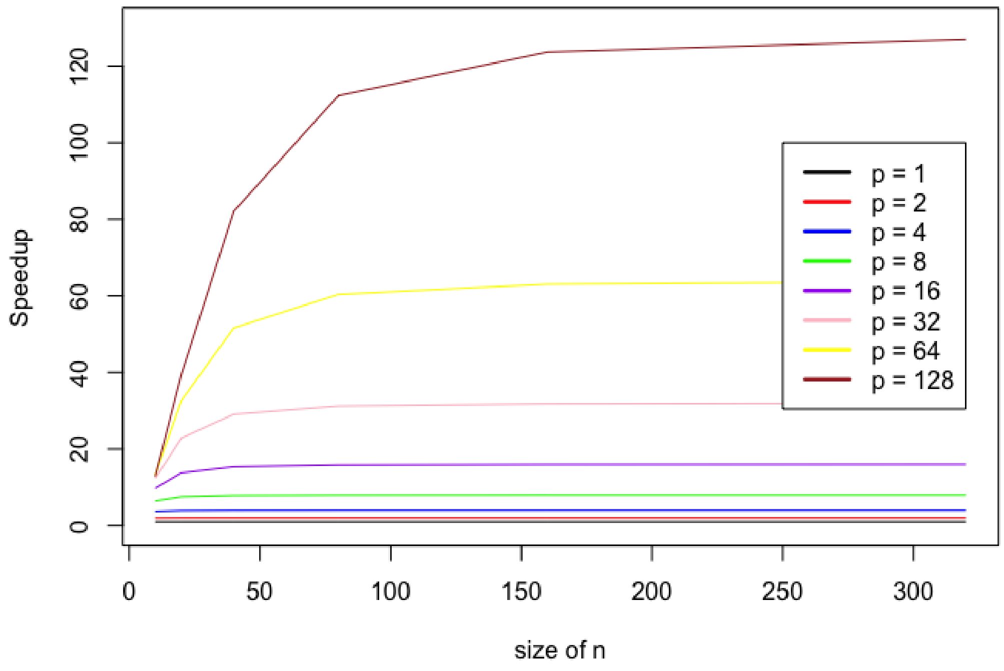
7 2.20 - Linear speedup and strong scalability

8 2.23 - Alternative algorithm for computing histogram

References

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

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



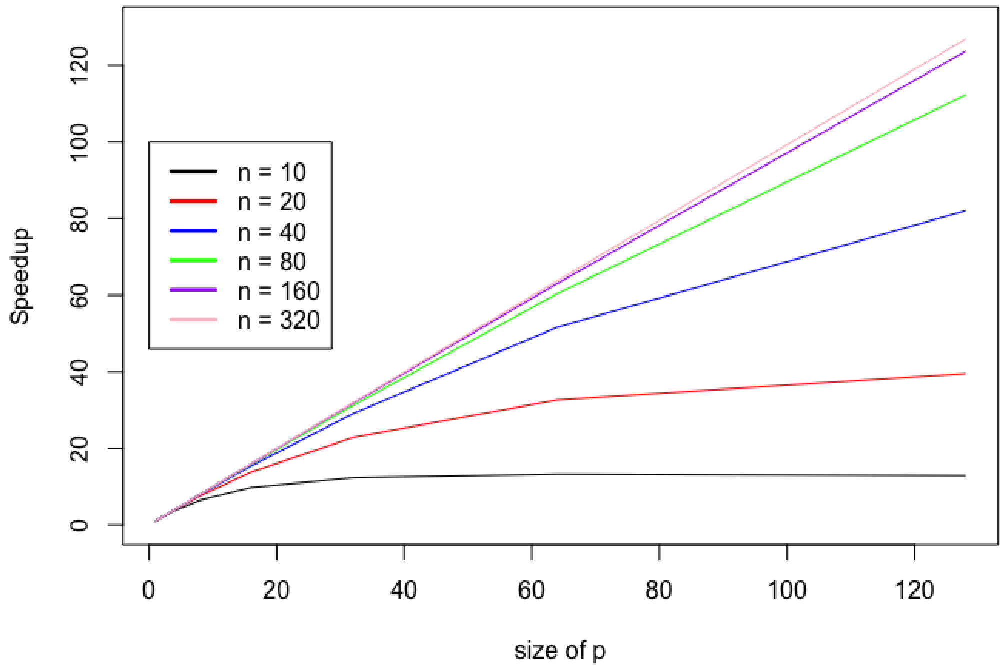


Figure 1: This figure shows the speedup for constant p, respectively constant n.