Umeå University
Department of Computing Science

Parallel Programming 7.5 p
5DV152

Exercises, Chapter/Topic 1

Submitted 2017-02-16
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 3.2 - Generalization of algorithm for trapezoidal rule

Two functions to adapt the *trapezoidal rule* for `calc\_local\_a` and `calc\_local\_b` were written and tested with the source code from the book (*mpi_trap.c*).

```
double calc_local_a(int my_rank, double a, double b, int n, int comm_sz){
  double local_a = 0;
  double h = 0;
  int local_n = 0;
  int rest_n  = 0;

  h = (b-a)/n;
  local_n = n/comm_sz;

  rest_n = n%comm_sz;

  if(my_rank < rest_n){
    local_a = a + my_rank*local_n*h + my_rank*h;
  } else {
    local_a = a + my_rank*local_n*h + rest_n*h;
    local_a += (my_rank-rest_n) * h;
  }

  return local_a;

}


double calc_local_b(int my_rank, double a, double b, int n, int comm_sz){
  double h;
  int local_n;

  h = (b-a)/n;
  local_n = n/comm_sz;

  if (my_rank == (comm_sz-1)){
    return a + my_rank+1*local_n*h;
  } else {
    return calc_local_a(my_rank+1, a, b, n, comm_sz);
  }
}
```

3 3.6 - Array distributions

Block distribution

Block distribution can be obtained by $b = \lfloor i \div p \rfloor$ where b is the block number, i the index of n and p is the number of processes. This solution seems reasonable altough it is not the optimum solution in every case.

Cyclic distribution

Cyclic distribution is described by $b = imodp$ with b as block number i as index of n and p as number of processes.

Block cyclic distribution

Block cyclic distribution can be expressed as $b = \lfloor i \div l \rfloor modp$ where b is block index, i index of n, l block length and p number of processes.

4 3.8 - Tree-structured algorithms for scatter and gather

5 3.9 - Vector scaling and dot product

takes a while to solve, requires programming

6 3.11 - Prefix sums

takes a qhile to solve requires programming

7 3.13 - Generalization of vector scaling and dot product

8 3.16 - Diagram for a butterfly implementation of allgather

9 3.18 - Derived data types

takes a while to solve requires programming

10 3.20 - Pack and unpack

requires programming

11 3.21 - Matrix-vector multiplication

takes a while to solve requires programming requires testing

12 3.22 - Timing the trapezoidal rule

takes a while to solve Requires programming requires testing

13 3.27 - Speedup and efficienciy of odd-even sort