(MPI) SPD Lab Time

M. Coppola

Last Revision May 10, 2011

This document will collect the exercises and problems that were discussed in the Lab time for the SPD course 2010-2011. It cannot substitute attending the lab time, but if you did not follow the course or missed a specific lesson, here you will find a list of the examples discussed. Some of the exercises are followed by a list of questions that you should try to answer, maybe achieving a deeper understanding of the examples..

The document is a work-in-progress which will grow as more lab lessons are given.

1 Exercise 1 : 29/03/2011

2 Exercise 2 : 12 - 15/04/2011

Design and implement an MPI program with several stages, some of them parallel. Our main aim is modularity and potential software reuse: how would you reuse part of the program in different settings, or the whole program as a parallel subroutine of a bigger one? The emphasis is on proper use of communicators to implement parallel constructs within the program. The program has three parameters in input:

$n \geq 2$	denotes the size of the matrices
$k \geq 1$	parallelism degree in stage FNR
$c > 0$	overall stream length; c is an even integer

$$\boxed{\text{RAN}} \rightarrow \boxed{\text{MUL}} \rightarrow \boxed{\text{NRM}} \rightarrow \boxed{\text{OUT}}$$

- The whole program is a pipeline of four stages, each stage sending a stream of data to the following one. The type of the data which are sent from one stage to the following one are fixed during program execution, but are not known at compile time (i.e. they depend on the size of the matrix, n^2, which is only known at program launch).

- Stage RAN is a sequential process generating random matrices of size n with floating point elements (choose any precision you like). Each Matrix M is defined as

$$M_{i,j} = \delta_{i,j} + random(-0.5, 0.5)$$

that is, each M is the identity matrix where each element is added a different random number between -0.5 and 0.5 .

- Stage MUL is a parallel map multiplying each two consecutive matrices in its input stream. It computes ${}_{(x)}M \times {}_{(x+1)}M$ with the usual algorithm

$$A_{i,j} = \sum_{k \in [0,n]} B_{i,k} C_{k,j}$$

The parallelism degree in MUL is n. You have to define a parallel matrix multiplication which uses n processes.

- Stage NRM is a farm computing the norm-2 of each matrix in its input. The norm-2 of a matrix ${}_{x}M$ is defined as

$$l_x = \sqrt{\sum_{i,j \in [0,n]} ({}_{x}M_{i,j})^2}$$

The parallelism degree in FNR is k

- Stage OUT is a sequential process which simply prints on `stdout`all the norm values it receives.

2.1 Questions about the Exercise

- Discuss how many (minimum and maximum number) of communicators that it makes sense to specify in this exercise.
- Discuss the relationship between the number of processes in your implementation (`mpirun -n`) and the n and c parameters.
- Do you use communicators to hide your code from the outside?
- Does it make sense to isolate parallel code from the code it hosts (e.g. the farm implementation from the worker code)?
- Can we assume the code contained in a parallel pattern will always be sequential?

- Can we support workers which are in fact parallel subprograms?
- How do we avoid specifying the stream length c before execution?
- How do we propagate an end-of stream signal?
  - Can we propagate and end-of stream signal using the TAG value?
  - Can we propagate and end-of stream signal using the message value?
  - Can messages using a different TAG value overtake each other?
- What kind of MPI Datataype definitions are needed by the different program stages? (compare the number you choose with the absolute minimum number, in terms of ease of code maintenance).

3 Exercise 3: 10/05/2011

Implement the following parallel algorithm working on a 129x129 real-valued matrix.

1. initialize each point in the matrix whose x,y coordinates are both multiple o 32 by generating random numbers between 0,1. These points will never change their values; all the other cells will be initialized with 0 and will have their value computed by the algorithm;
2. distribute the matrix across a number of processes;
3. each cell in the matrix is computed as the weighed average of its previous value and that of its 8 neighbors; where the weight in the stencil is 2 for the old cell value, and is proportional to the relative distance (either 1 or $\sqrt{2}$) for the neighbours, that is

   $$m_{i,j} = \frac{1}{6+4\sqrt{2}} \sum \left\{ 2m_{i,j}, m_{i\pm1,j}, m_{i,j\pm1}, \sqrt{2}m_{i-1,j-1}, \sqrt{2}m_{i+1,j+1}, \sqrt{2}m_{i\pm1,j\mp1} \right\}$$

   each process will thus have to exchange the new values on the partition borders with his neighboring processes;
   assume that any cell needed by the stencil that is outside the matrix border has the same value as the cell $m_{i,j}$ in the center of the stencil;
4. iterate the parallel computation as long as it is needed for each cell of the matrix to stabilize (i.e. the value change is less than 10^{-4}).

3.1 Questions/comments about the Exercise

- It may be simpler to analyze the problem on an one-dimensional array, with a stencil composed of the cell to compute and the two direct neighbours.
- Pay attention to the immutable positions within the matrix. Do you need to take them into account in the parallelization, or may you deal with them only in the sequential part of the computation?
- State your assumptions on the possible parallelism degree.
  - Consider what is the fraction of data that needs to be exchanged between different MPI processes.
  - Devise a schema for exchanging only the needed part of data at each global iteration.
  - Can you avoid global synchronizations among all the processes in order to exchange the partition borders?
- How many copies of the matrix do you need to keep in each process' memory?
- Is it possible to write all the processes according to a common schema, regardless of the part of the matrix that they receive for computation?
- Set up a maximal number of iteration after which the program will stop with a warning.
- Discuss the advantages of partitioning the matrix by rows, columns, or blocks. What is the tradeoff between complexity and amount of transferred data? Does it depend on the size of the matrix?
- Are your data partitions always perfectly balanced?
- Do your MPI datatypes depend on the position in the matrix of the data your process is assigned?
- How do you detect that all cells in the matrix have stabilized?
  - Can you avoid a global synchronization to check for stabilization? How often do you need to check?
- What is the expected comparison among the cost of global synchronization, of local synchronization, and the computations assigned to a process?

- How this comparison is likely to affect a generic problem of size $N \times N$ an parallelism $p = i \times j$?