Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences
Fachbereich Informatik
Lehrgebiet Programmierung komplexer Systeme
Prof. Dr. Rudolf Berrendorf

Parallel Systems
WS 19/20
given out : 16.12.2019
to deliver : 12.01.2020

Assignment 6 : Scheduling II

Assignment 1 (20 points) :
In the supplement for this assignment you find a simple MPI implementation for the Mandelbrot set. The program computes in an iterative process for every point (x,y) a value that is colored according to the number of iterations needed to compute this value. The computation for every point (x,y) is independent of the computation for any other point, and the time to compute that value for different points may vary significantly.

The MPI program works as follows. It assigns the first processor (master processor = rank 0) the task of collecting all computed pixels from other processors (slave processors) and displays these pixels (in our case just compute a checksum). All slave processors work together to compute all points (x,y). This is done in such a way that all rows for the `NROW` x `NROW` pixel area are distributed blockwise to available slaves. You will see that the computational effort to compute pixel values varies significantly such that the block distribution used in the program is very inefficient. Run the original program for 1, 2, 4, 8, 15, 31 slave processors (plus one master processor) and remember the time needed for that computation.

In order to optimize the overall computation time, your task is to schedule the mandelbrot rows (tasks) to processors in a more effective way with the help of graph partitioning techniques. To do that, proceed as follows:

1. In a first phase determine the computational effort for every row (there are `NROW` rows in the example) in seconds.
2. After that, in a second phase use graph partitioning techniques to find a good schedule of rows (corresponding computational tasks) to processors.
3. And lastly, in a third program phase use this information to compute the mandelbrot set again more efficiently.

Compare the time results you get for the original version and the version you scheduled (only program phase 3).

Therefore the steps to do:

1. Run as described the unmodified MPI program on 1, 2, 4, 8, 16, 32 compute processors (plus one master processor) and remember as a reference the time needed for the execution with this block distribution. For that you have to modify the job script in the supplement.
2. Then, copy the original program and modify it in a way descibed as follows.
   (a) Measure/gather the time needed to compute every row of the mandelbrot set (i.e. one parallel task) for each of the `NROW` rows on the processor executing this row. You can do that sequentially or parallel. Gather this information on processor 0.
   (b) Use this timing information to partition an appropriate graph for n partitions where n is the number of slave processor in your program run. The measured task times are vertex weights, edge weights are 0 as you have no communication costs (but see remark at the end). Use the graph partitioning package Metis to do the graph partitioning for the generated graph (see below). The partitioning process gives you as a result a vector of length n that tells you at position i what processor should execute row i (or in other words task i).
   (c) Call an appropriately modified copy of the `mandelbrot_client` function again using the partitioning you found in the last step and schedule the computation involved with a row to the corresponding processor. Measure the total time just for this program phase (ignoring the measurement phase and the partitioning phase) for different number of slave processors as done in the unmodified program before and compare the results.

You find in the supplement for this assignment a small example program how to use the Metis graph partitioning package for this task. A rough desciption on Metis is given below, a detailled reference manual is available on the web page. As the installed Metis version is a sequential library, it is easiest for you to do all administrative work (generate the graph,

call the partition function) on the master processor. Therefore you have to collect (with appropriate MPI calls) all relevant data on the master processor, call on this processor the Metis partition function, and then redistribute (with appropriate MPI calls) the resulting scheduling vector to slave processors.

Graph Partitioning with Metis

Metis (**http//www.cs.umn.edu/~metis/**) is a software library for graph parititioning. You find the reference manual on our web page. Please be aware that we use Metis version 5, the API changed with version 5.

Metis has a command line interface and a library interface to be called inside other programs. To use the library, you must load the module once a session **module load metis/5.1.0-32** and include the appropriate include file in your program (see the files supplied with the Metis example):

```
#include <metis.h>
```

and link your program with the Metis library:

```
cc ... -lmetis -lm
```

Use the **METIS_PartGraphKway()** function of the Metis Library to partition your graph into p partitions. In chapter 5.5 of the reference manual you find a description on how parameters are passed to the partition function and how results are given back. Mainly you have to describe the graph structure with 3 vectors and call the partition function with some additional information. In the supplement for this assignment you find a small example program how to do that in practice. You can leave all settings and just adapt the graph data structure and the number of partitions to your needs.

Edge weights 0: please be aware that Metis needs a connected graph with edge weights not equal 0, otherwise the Metis functions return an error code. If a graph does not have these properties (as in our case) you can modify your graph easily in adding dummy edges with edge costs ε. Therefore it is sufficient in our case to connect vertex i with vertex $i+1$ with edge weight ε for all $i = 0, ..., n-1$.