HAVUGIMANA Leonce pers.n.: 860512_5494 **2017 august, 28th**

Viktor Lunken Lundkvist pers.n.: 920628

Performance Optimization (DV1463)

Project 2 Report

Introduction

This is a brief description of the work done in project2 of performance optimization course (DV1463). The main objective is to implement pthreads(POSIX threads) in given template versions of three different algorithms namely: Mandelbrot fractal, 2-dimensional matrix-matrix multiplication and quick sort. Each of the three codes is compiled by gcc compiler in linux ubuntu by passing command options through the terminal as follows:

Sequential code: gcc -<source_code_file.c> -o <output_file>

Threaded code: gcc <source_code_file.c> -o <outpu_file> -lpthread

Task1: **Fractal code**

In 8-threads code, we have split mandelbrot function into 3 functions. The first function "mandelbrot" is basically useful to increment the value of "ii" variable and test "while(sxsy<=64.0f)" loop. Depending on the value of "ii" one of the other two functions, mandelbrot2 or mandelbrot3, will be called. The first 6 threads will work on "mandelbrot" function to speed up the loop. The seventh thread will work on "mandelbrot2" function and "mandelbrot3" will be performed by the eighth thread.

Sequential version execution time: 2.824 sec

One threaded version execution time: 2.808 sec

8 threaded version execution time: 1.706sec

Based on time execution results we can find that the speed up is promising to be good enough; seq_time is almost 2X execution time of 8_threads program: 2.824 sec/1.706 sec = 1.655

We hope the speedup can be approximately 2 if the application is run on a computer with good cpu performance.

Task2: **Matrix multiplication**

In 8_threaded code we have set array of eight threads. Each thread is going to execute matmul() function. The work is distributed equitably (load balance) among all threads. That is each thread has 1024/8 rows to calculate for product matrix. It is very useful to underline importance of load balance in thread programming; it allows to maximize the usage of available resources (cores).

Sequential version: 1min 2.448sec

One threaded version: 1min 1.372sec

8 threads version: 36.73 sec

Given data above, the speedup is 1.7 when we compare performance of 1 thread code to 8 threads code. Again, this is a good performance, it can gradually increase if the application is run on high speed machine.

Task3: **Quick sort**

Quick sort algorithm is favorable for thread programming. Its the algorithm can be decomposed into chunks of small routines which can run independently from the rest of the application. We have tried to parallelize quick_sort() function because it is the function called by main() which has to do much computations compared to other functions called from main().

Sequential version: 39.720sec

One threaded version: 36.976 sec

8threads code: 24.8sec

For this application, the attained speedup is 1.65 which is not bad.

Conclusion

This project exercise has been a great practice to put some hands on pthreads implementation in three different applications candidates to parallelization. It was inevitable to do a few alterations in some functions to make them compatible with threads architecture. The focus was on functions which seem to encompass much computational work required to produce the output.

The codes files accompany this report.