TDDC78 - Programming of Parallel Computers

Lab3 - Heat Equation (OpenMP)

Authors:
David Tran - Davtr766
Jakob Bertlin - Jakbe457

I. Introduction

The focus of the third lab in the course TDDC78 is OpenMP. The aim of this lab is to solve a stationary heat conduction problem on a shared memory computer and parallelize it using OpenMP. A serial code for solving this problem is given in the file *laplsolv.f90* which is implemented with FORTRAN. The iterative method is known as the Jacobi method. The aim is to reproduce the same result as the given file using our own implementation.

II. Approach

A decision was made to use *C++* since the standard library available is convenient for this problem. Therefore, the steps taken was to rewrite the given file from FORTRAN to *C++*. The first step was to define the data structures to initialize the matrix and the temporary vectors. The temperature data matrix was defined as a vector of vectors using the data structure *std::vector* from the standard library. The second step was to initialize the boundaries for the matrix. The matrix was flipped by 90 degrees because the FORTRAN code handles the matrix column-wise in memory and C++ does it row-wise.

The next step is to implement the nested iterative loops from the file *laplsolv.f90* in C++. Standard library functions are heavily used and the time complexity of the operations are *O(n)*.

The final step is to use OpenMP in order to parallelize the inner loop where the computation are made. OpenMP uses the fork-join model for parallel execution. This is done by adding the #pragma directive to specify where the parallelization will start.

III. Result

In order to parallelize the code, the #pragma directive was used:

```
#pragma omp parallel private(varlist) shared(varlist)
```

Where *varlist* in the *private* clause are the variables specified privately for each thread and the *varlist* for *shared* clause are variables that are shared among the threads. Before the inner loop declaration, the *for* directive was used with the *schedule* clause:

```
#pragma omp for schedule(static)
```

This will divide the iteration work into chunks based on the amount of threads and assign each thread to one chunk.

The code was tested on Triolith varying the number of cores. The linear plot can be seen in figure 1.

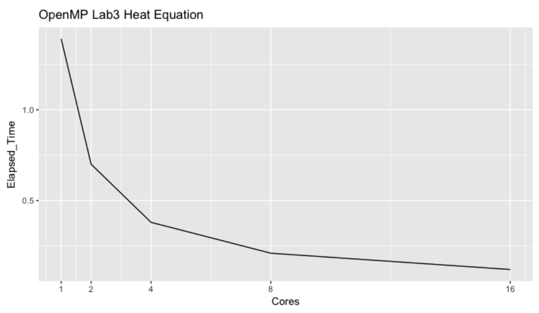

Fig. 1: A linear plot showing the dependence between number of cores and the elapsed time.

IV. Discussion

The results given from Fig. 1 shows a good linear scaling over the number of cores/threads used. However since OpenMP is using shared memory we can not scale above 16 cores in this case since Triolith only contains 16 cores per node.

APPENDIX

Lab3 code

```cpp
#include <ctime>
#include <iostream>
#include <vector>
#include <omp.h>
#include <algorithm>
#include <cmath>
#include <iterator>
#include <cfloat>
#include <chrono>
#include <numeric>

int main(int argc, char* argv[]){
    /*Variable initialization*/
    int n{1000}, maxiter{3000}, k{1}, j, t, nt;
    double tol{1.0e-3}, error{DBL_MAX}, x{0.0};
    std::vector<std::vector<double>> T(n+2,std::vector<double>(n+2));
    std::vector<double> tmp1, tmp2, vec1, vec2, vec3;
    std::string str;
    std::vector<double> temp;

    for(int i = 0; i < n+2; i++) {
        T[0][i] = 1.0;
        T[n+1][i] = 1.0;
        T[i][n+1] = 2.0;
    }
    T[0][n+1] = 2.0;

    omp_lock_t writelock;
    omp_init_lock(&writelock);

    //auto t_start = std::chrono::high_resolution_clock::now();
    double t_start = omp_get_wtime();


    for(k = 1; k < maxiter && error > tol;k++){

        error = 0.0;
        /*Set boundaries and initial values for the unknowns*/
        //Heat conduction

        /* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(tmp1, tmp2, vec1, vec2, vec3, j, temp) shared(k, T, n, error)
        {

            tmp1.resize(n);
            temp.resize(n);
            tmp2.resize(n);
            vec1.resize(n);
            vec2.resize(n);
            vec3.resize(n);
            tmp1.assign(T[0].begin()+1, T[0].end()-1);

#pragma omp for schedule(static)
            for(j = 1; j <= n; j++){
                tmp2.assign(T[j].begin()+1, T[j].end()-1);

                vec1.assign(T[j].begin(), T[j].end()-2);
                vec2.assign(T[j].begin()+2, T[j].end());
                vec3.assign(T[j+1].begin()+1, T[j+1].end()-1);

                std::transform(vec1.begin(), vec1.end(), vec2.begin(), vec1.begin(), std::plus<double>());
                std::transform(vec1.begin(), vec1.end(), vec3.begin(), vec1.begin(), std::plus<double>());
                std::transform(vec1.begin(), vec1.end(), tmp1.begin(), vec1.begin(), std::plus<double>());

                std::for_each(vec1.begin(), vec1.end(), [](double& val){
                    val/=4.0;
                });

                std::copy(vec1.begin(), vec1.end(), T[j].begin()+1);
                std::transform(tmp2.begin(), tmp2.end(), T[j].begin()+1, temp.begin(), [&temp](double a, double b){
                    return std::abs(a-b);
                });

                omp_set_lock(&writelock);
                error = std::max(error, *std::max_element(temp.begin(), temp.end()));
                omp_unset_lock(&writelock);

                //std::cout << error << "\n";
                tmp1.assign(tmp2.begin(), tmp2.end());

            } // Inner for end

        } /* Parallel end, All threads join master thread and disband */

    } // Outer for end

    //auto t_end = std::chrono::high_resolution_clock::now();
    double t_end = omp_get_wtime();
    //double time_elapsed_ms = std::chrono::duration<double, std::milli>(t_end-t_start).count();
    double time_elapsed_s = t_end-t_start;
    printf("Number of iterations: %d \n", k);
    printf("CPU time used: %g ms \n", time_elapsed_s);

    double avg{0.0};

    for(auto r : T) {
        avg = std::accumulate(r.begin(), r.end(), avg);
        //std::copy(r.begin(), r.end(), std::ostream_iterator<double>(std::cout, " "));
        //std::cout << "\n";
    }

    printf("Average temp: %g C \nError: %g \n", avg/(n*n), error);



    return 0;
}
```