ENEL464 Embedded Software and Advanced Computing 2020
Group Assignment 2

1 Introduction

The purpose of this assignment is to implement a numerical algorithm on a desktop computer to make the most efficient use of its caches and multiple cores.

The algorithm is Jacobi relaxation. This is an iterative algorithm used to approximate differential equations, for example, Poisson's and Laplace's equations. Poisson's equation can be used to find the electric potential given a specified charge distribution or temperature given a specified heat source. There are more efficient ways to solve this problem using Green's functions and Fourier transforms but that is not the purpose of this assignment.

2 Jacobi relaxation

The discrete form of Poisson's equation,

$$\nabla^2 V_{i,j,k,n} = f_{i,j,k,n}, \tag{1}$$

can be solved iteratively, at each time-step n, using Jacobi relaxation, where

$$V_{i,j,k,n+1} = \frac{1}{6}\left(V_{i+1,j,k,n} + V_{i-1,j,k,n} + V_{i,j+1,k,n} + V_{i,j-1,k,n} + V_{i,j,k+1,n} + V_{i,j,k-1,n} - \Delta^2 f_{i,j,k}\right). \tag{2}$$

Here $\Delta = \Delta x = \Delta y = \Delta z$ is the spacing between voxels in metres, $0 \le i < N$, $0 \le j < N$, and $0 \le k < N$. Voxels on the boundary ($i = -1$, $i = N$, $j = -1$, $j = N$, $k = -1$, $k = N$) have a value V_{bound}. This is a Dirichlet boundary condition, equivalent to enclosing the problem in a metal box at potential V_{bound}. You might recognise (2) as 3-D convolution at each time-step.

3 Implementation

Implement an algorithm for Jacobi relaxation in either C or C++. Your goal is to find a fast implementation that will run on your (or a CAE lab) computer, making best use of the caches and multiple cores.

Your implementation needs to work with an arbitrary 3-D source distribution f and use the specified API below:

```
/// Solve Poisson's equation for a rectangular box with Dirichlet
/// boundary conditions on each face.
/// \param source is a pointer to a flattened 3-D array for the source
    function
/// \param potential is a pointer to a flattened 3-D array for the
    calculated potential
/// \param xsize is the number of elements in the x-direction
/// \param ysize is the number of elements in the y-direction
/// \param zsize is the number of elements in the z-direction
/// \param delta is the voxel spacing in all directions
/// \param numiters is the number of iterations to perform
/// \param numcores is the number of CPU cores to use.  If 0, an optimal
    number is chosen
void poisson_dirichlet (double * __restrict__ source,
                        double * __restrict__ potential,
                        unsigned int xsize, unsigned int ysize, unsigned
                            int zsize, double delta,
                        unsigned int numiters, unsigned int numcores)
{
    // source[i, j, k] is accessed with source[((k * ysize) + j) * xsize +
        i]
}
```

4 Testing

Test your algorithm with a single point charge in the centre of the volume, i.e.,

$$f_{i,j,k} = \begin{cases} 1 & i = N/2, j = N/2, k = N/2, \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where the volume is comprised of $N \times N \times N$ voxels. Note, you algorithm must work with arbitrary source distributions.

5 Support

Only questions submitted via the ENCE464 assignment forum will be answered. Emails will be quietly ignored.

6 Reports

The reports are group reports and are to be submitted as PDF documents through the ENCE464 Learn page. They will be submitted to TurnItIn for plagiarism checking.

Guidelines for writing a report are available at
`https://eng-git.canterbury.ac.nz/mph/report-guidelines/blob/master/report-guidelines.pdf`.

Each report is to use a 12 point font and be no longer than five pages, including appendices. Please use margins of at least 2 cm.

7 Assessment

Your report will be marked in terms of:

- Written style. The writing should be concise technical writing.
- Presentation. The key here is consistency and clear diagrams and graphs.
- Architecture overview. This should describe your computer's architecture (such as the cache organisation, memory size, CPU, etc.).
- Cache analysis. This should discuss how your program takes advantage of the cache.
- Multithreading analysis. This should discuss how your program takes advantage of multiple threads.
- Profiling analysis. This should show which parts of your program take the most time.
- Optimisation analysis. This should discuss the affects of some of the compiler optimisations, such as loop unrolling, on your program.
- Overall excellence.

Each section is marked out of 5, giving a total of 40 marks. Five bonus marks will be awarded to any group who can beat our program when running on a CAE lab computer and give the correct results.

Your report should present the statistics for the time your program takes to run for the following problem sizes: $N = 101, 201, 301, 401, 501, 601, 701, 801, 901$. For $N = 801$ and $N = 901$, do not worry about performing many trials. If you have a really old computer, you can skip $N = 901$.

Your analyses should be backed up with experimental evidence, such as the output from profiling tools.

8 Code

You must submit your code as a `.zip` file so it can be checked for numerical accuracy and plagiarism. Your fastest implementation must be able to built by running `make`.