CS257 Advanced Computer Architecture Coursework Assignment

1 Introduction

The purpose of this coursework is to give you some hands-on experience in code optimisation. By the time that you read this you will have encountered a variety of code optimisation techniques including loop unrolling and vectorisation.

2 Submission

Your submission will consist of two parts:

1. **Optimised Code (70%)**
   A piece of C code based on the initial implementation provided. This C code will be assessed with respect to your selection and understanding of optimisations, functional correctness, i.e., producing the right answer, and execution speed.

2. **Written Report (30%)**
   A report (3 pages maximum, excluding references) detailing your design and implementation decisions. Your report will be evaluated with respect to your understanding of code optimisation techniques and the optimisations you attempted. This means that your report should explain:

   (a) which optimisations you used/didn't use;

   (b) why your chosen optimisations improve performance; and

   (c) how your chosen optimisations affect floating-point correctness.

Given that you may apply many different optimisations, a sensible approach is to build your solution incrementally, saving each partial solution and documenting the impact of each optimisation you make. This means that it is in your interest to attempt as many different optimisations, or combinations of optimisations as you can.

You may discuss optimisation techniques with others but you are not allowed to collaborate on solutions to this assignment. Please remember that the University takes all forms of plagiarism seriously.

3 Introduction to ACACGS

ACSCGS is a conjugate gradient proxy application for a 3D mesh. The simulation will execute for either a fixed number of timesteps, or alternatively until the residual value falls below a given threshold. This is done for a given mesh size, which is passed in at runtime through command line arguments.

In this proxy application, a force is applied to each edge boundary of the cuboid, which is then propagated throughout the mesh. As each time step passes, the force is dissipated within the mesh, until the amount of residual is significantly small that the simulation stops (as there is no more calculations to perform), or a set number of time steps have passed.

In addition to providing numeric solutions, the code can also generate visuals which depict the pressure within the mesh throughout the simulation run. Creating the visualisations relies on two optional packages, Silo and VisIt, which are available on the DCS systems.

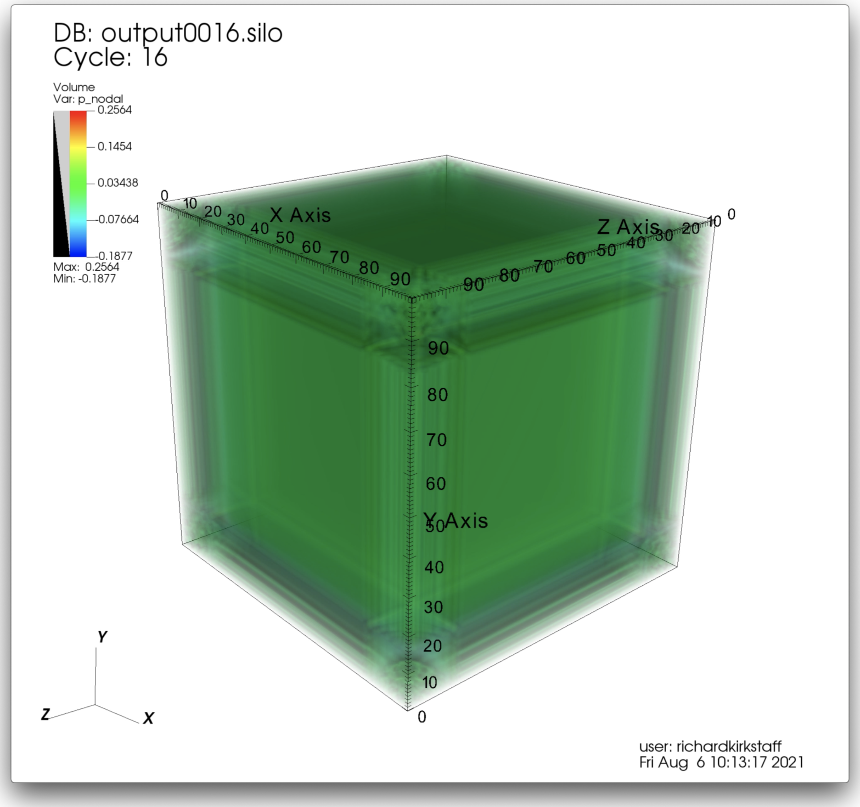

Figure 1: Pressure Matrix Visualisation

4 Compiling and Running the Code

The code includes a makefile to build the program. You can compile all of the code using the command make. You may modify the makefile if you wish to, but please ensure you include it with your submission. If you do not include a makefile, then the default one will be used.

While the DCS machines do include a version of gcc, it is preferable to use a more recent version. On the DCS systems, you can make version 9 the default by using the 'module load gcc9' command. Once this is loaded you can simply type 'make' to build the code, which will create an executable named 'acacgs' in the directory.

To run the code, you need to provide the three dimensions for the mesh as three parameters to the executable. For example to execute the provided code on a small 10x10x10 mesh you would enter './acacgs 10 10 10'. On my system the output for the code is below. This information is also stored in a file, which is named after the wallclock date and time of when the program was first executed (for example, `2022_01_26_12_00_00.txt`).

```
===== Final Statistics =====
Executable name:      ./acacgs
Dimensions:           10 10 10
Number of iterations: 149
Final residual:       2.226719e-92

=== Time ==
Total:           1.126600e-02 seconds
ddot Kernel:     8.390000e-04 seconds
waxpby Kernel:   1.087000e-03 seconds
sparsemv Kernel: 9.123000e-03 seconds

=== FLOP ==
Total:           9.536000e+06 floating point operations
ddot Kernel:     5.960000e+05 floating point operations
waxpby Kernel:   8.940000e+05 floating point operations
sparsemv Kernel: 8.046000e+06 floating point operations

=== MFLOP/s ==
Total:           8.464406e+02 MFLOP/s
ddot Kernel:     7.103695e+02 MFLOP/s
waxpby Kernel:   8.224471e+02 MFLOP/s
sparsemv Kernel: 8.819467e+02 MFLOP/s

Difference between computed and exact = 1.110223e-15
```

You will find more detailed instructions to build the code in the README.md file, including flags to turn on verbose mode, which will output details for each timestep in the simulation, and flags for enabling visualisation.

4.1 Visualisation Generation

To enable visualisation outputs, you must build your code using 'make SILO=1'. This will then compile your code in a way which produces files suitable for visualisation in VisIt. If you are working remotely and want to visualise the coursework, it will be quicker and easier for you to copy the files to your local machine, then utilise VisIt on the local machine to visualise the cuboid. Before you make the program, make sure you load the SILO module (`module load cs257-silo`).

When the program is ran with visualisations, each timestep will produce a SILO file within a directory named after the wallclock date and time (for example: `2022_01_26_12_00_00`). In this directory will be a collection of .silo files, each named `outputXXXX.silo`, where `XXXX` represents the timestep it relates to.

Once the program has finished, these can be utilised in Visit. To do so, load the VisIt module (`module load cs257-visit`) and open VisIt using the command `visit`. From here, you will get 2 windows. The smaller, skinner one is the control window and is used to manage everything that will be displayed. The larger window is the display window. In the control window, select *Open*, and navigate to the directory with the SILO files. You should then be able to select these SILO files.

Now that the SILO files have been loaded, we can now draw some given variables. To do this, click on the *Add* and select a mode and a variable that should be viewed. One of the nicest ones to use is `Volume` and either `x_nodal` or `p_nodal`. When you have finished adding elements, click on `Draw`. This will generate an image in the display window, that can be dragged around so that the cuboid can be viewed from different angles. The control window has a play button, which will run through each timestep.
Visualisations are nice to have, but for performance purposes we turn them off as they write a significant amount fo data to disk.

Table 1: Visualisation Data File Sizes

x	y	z	Cells	Approximate Data Size
10	10	10	1000	4MB
25	25	25	15,625	39MB
50	50	50	125,000	301MB
100	100	100	1,000,000	2.4GB
200	200	200	8,000,000	19.3GB

There is the potential to go significantly over your DCS disk quota with large meshes. I recommend that you do not exceed 30x30x30 for producing visualisations on the DCS machines. If you are developing your solution on your personal machine then you may wish to produce larger visualisations.

5 Hardware Details

On a linux system, you can read the processor information using the command "cat /proc/cpuinfo". This will provide full details on the cpu in the machine, including the CPU model, number of cores, the clock frequency and supported extensions. I strongly recommend taking a look at this on your development machine.

For the purposes of assessment, your code will be run on a DCS machine with 6 cores. The output from /proc/cpuinfo for a single core can be seen below:

```
processor       : 5
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz
stepping        : 10
microcode       : 0xca
cpu MHz         : 799.987
cache size      : 9216 KB
physical id     : 0
siblings        : 6
core id         : 5
cpu cores       : 6
apicid          : 10
initial apicid  : 10
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology
nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx
smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe
popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
3dnowprefetch invpcid_single intel_pt ssbd ibrs ibpb stibp tpr_shadow vnmi
flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid
rtm mpx rdseed adx smap clflushopt xsaveopt xsavec xgetbv1 dtherm ida arat
pln pts hwp hwp_notify hwp_act_window hwp_epp md_clear spec_ctrl
intel_stibp flush_l1d
bogomips        : 6000.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

Machines matching this specification are available in the cs257 queue of the

Batch Compute System in the Department. You will learn how to use this system during the lab sessions, so there will be time to get used to it.

6 How will my code be tested for performance?

Your submission will be tested on a range of input sizes to evaluate how robust your performance improvements are. It is recommended that you try testing your solution on inputs that are not cubes to see if there are any weaknesses in your optimisation strategies. The 7-pt stencil option will not be used for testing your code.

Your code will be executed five times for each problem size on the target hardware. The highest and lowest runtimes will be discarded, and the mean of the three remaining values will be taken as your runtime for that problem size.

7 Rules

Your submitted solution **must**:

- Compile on the DCS workstations.

Your submitted solution **must not**:

- Use instruction sets not supported by the DCS machines.
- Require additional hardware e.g., GPUs
- Add relaxed math options to the compile line, e.g., -ffast-math. Note: Manual use of approximate math functions is acceptable.

8 Where do I start?

This can seem like a daunting project, but we can break it down into a number of steps.

1. Compile and run the code as provided. This is a quick easy check to make sure your environment is setup correctly.
2. Read the code. Start in main.c and follow it through. The functions are well documented with Doxygen comments. Don't panic - you are not expected to understand the physics in the code.
3. Measure the runtime of the code for reference purposes.
4. Figure our where the most intensive sections of code are.
5. Develop a small optimisation.
6. Run the code and review the impact of your changes.
7. Repeat steps 5 and 6 until you have exhausted your performance ideas.

9 Instructions for Submission

Your solution should be submitted using Tabula. Please ensure that your code works on DCS machines prior to submission.

Submission deadline: Noon, Wednesday 16th March 2022.

Files required: A single file named coursework.zip which should contain all of your code at the top-level (i.e. no subdirectories) and the report file as a PDF.

10 Support

Support can be found from Richard Kirk (R.Kirk.1@warwick.ac.uk) via email or Teams.