General marking feedback

Marks

The mark shown in your feedback does not consider late submission penalties. Com Teaching will update the marks. All submissions before midnight April 20 will not be penalised.

Command line arguments

The expected arguments are:

```
"nbody.exe N D M [-i I] [-f input_file]"
```

Where:

- N Is the number of bodies to simulate
- D Is the floating-point dimension size of the activity map
- M Is the operation mode, either 'CPU', 'OPENMP' or 'CUDA
- [-i I] Optionally specifies the number of simulation iterations 'I' to perform. No value will use visualisation mode
- [-f input_file] Optionally specifies an input file with an initial N bodies of data. If not specified random data will be created

Brackets only indicate that the arguments are optional. They should not be included to specify those arguments. Example of valid arguments are:

- `"nbody.exe 100 10 CPU -i 1000 -f example.nbody”`
- `"nbody.exe 100 10 OPENMP -i 1000”`
- `"nbody.exe 100 10 CPU -f example.nbody”`
- `"nbody.exe 100 10 CUDA”`

Code

The use of structure of arrays (SoA) rather than array of structures (AoS) improves coalesced memory access of data structures. This should lead to less memory bandwidth usage from better use of cache lines.

Consider use of double buffering to reduce memory writes, coupled with pointer swapping to reduce memory copying. This is implemented by initializing two data arrays. One array is used by the visualiser, while the other array updates its values through calculations. Once calculations are complete, pointer swap so the visualisation now points to the updated variables

Considered mathematical function performance compared with other operations. For example, pow(x,2) compared to (x*x).

Be sure to not allocate memory (malloc) within a function that is called each step. It should be possible to allocate all memory during the initialisation of the simulation.

File input should be able to inform/warn when number of particles in file don’t match command line arguments. Similarly, it should handle badly formatted files elegantly (i.e. without crashing) and read comment lines (#).

Inefficient activity map calculation involved iterating through D*D*N variables. Effective calculation involves finding the associated grid position for N variables.

For those who did not correct calculate the nBody simulation here is some help. The force experienced by a body i is

$$\boldsymbol{F}_i = gm_i \sum_{j}^{i \neq j} \frac{m_j(\boldsymbol{x_j} - \boldsymbol{x_i})}{\left(\left\|\boldsymbol{x_j} - \boldsymbol{x_i}\right\|^2 + \epsilon^2\right)^{3/2}}$$

Pseudocode where input and output are arrays of nbody structures:

```
for i{
	//force experienced by particle i in x direction
	output[i].f.x = 0;
	//force experienced by particle i in y direction
	output[i].f.y = 0;
	for j{
		if (i == j) continue;

		//x vector difference in positions
		xDiff = input[j].x - input[i].x;
		//y vector difference in positions
		yDiff = input[j].y - input[i].y;

		//Bottom part of the fraction within the summation
		denominator = pow(xDiff ^ 2 + yDiff ^ 2 + e ^ 2, 3 / 2);

		//update the force of particle i
		output[i].f.x += input[j].m * xDiff / denominator;
		output[i].f.y += input[j].m * yDiff / denominator;
	}
	//scale force by constants
	output_bodies[i].f.x *= g * input[i].m;
	output_bodies[i].f.y *= g * input[i].m;
}

for i{
	//update velocity
	output[i].v.x += input[i].f.x / input[i].m * dt;
	output[i].v.y += input[i].f.y / input[i].m * dt;

	//update position
	output[i].x += input[i].v.x * dt;
	output[i].y += input[i].v.y * dt;
}
```

If you still do not understand or feel like you cannot implement this please ask during lab time, or through the course mailing list. *It is possibly *more* correct to update position first *then* velocity but should not make much of a numerical difference and will not be judged either way.

Report

The ideal report would explain what regions were the most compute/bandwidth heavy and what optimizations could be done, and which would be most suitable. This would be iteratively applied until

it was deemed suitable that there was no more need for optimisations. Optimizations should have been done in CPU and OpenMP.

The OpenMP analysis must include parallelising outer and inner loop, scheduling and race condition solutions for the force calculation loop and the activity map.

On average code was to a better standard than the report, even though they carried equal weighting. Grades were given back as integers though were calculated through real numbers; this should explain any grade total that is 1 value away from the integer total.

Next submission

- Upload the entire solution (including the project files after performing a build clean), not just the .c/.cu/.h files
- The report should be a pdf or doc/docx file.