HPCSE II - Exercise 1

Anian Ruoss

February 23, 2019

All experiments were run on a node obtained by running the following command:

```
bsub -n 24 -R fullnode -R "select[model=XeonE5_2680v3]" -Is bash
```

Task 4

a) See the first row in Table 1.

b) The optimal number of grids is 6 as illustrated in Table 1.

Table 1: Iterations and running times for different numbers of grids for DownRelaxations = 1 and UpRelaxations = 1.

GridCount	Iterations	Running Time	Verification
1	199204	150.165s	Passed
2	43522	57.425s	Passed
3	32457	46.268s	Passed
4	32102	46.524s	Passed
5	30625	44.926s	Passed
6	22973	33.810s	Passed
7	33171	48.768s	Passed
8	51762	75.945s	Passed

c) The optimal combination of up and down relaxations is determined in a greedy way. The optimal number of down relaxations in terms of running time is 3 as illustrated in Table 2. Although, values of 5 and 7 produce less iterations, the time per iteration is significantly higher.

Table 2: Iterations and running times for different numbers of down relaxations given GridCount = 6 and UpRelaxations = 1.

DownRelaxations	Iterations	Running Time	Verification
1	22973	33.770s	Passed
2	37152	72.471s	Passed
3	13467	33.071s	Passed
4	20885	61.631s	Passed
5	12068	41.526s	Passed
6	14811	58.096s	Passed
7	10130	44.835s	Passed

The optimal number of up relaxations is 1 as illustrated in Table 3.

Table 3: Iterations and running times for different numbers of up relaxations with GridCount = 6 and DownRelaxations = 3.

UpRelaxations	Iterations	Running Time	Verification
1	13467	33.055s	Passed
2	13467	34.066s	Passed
3	13467	35.200s	Passed
4	13467	36.909s	Passed
5	13467	37.349s	Passed
6	13467	38.467s	Passed
7	13467	39.613s	Passed

d) The optimal configuration is given by GridCount = 6, DownRelaxations = 3, and UpRelaxations = 1.

e) The solution reaches its equilibrium state faster on coarser grids as every cell covers more area. Fine grids ensure that the discretization error remains small.

f) TODO

Task 5

a) *Loop Interchange* can be applied to a prevent bad memory access patterns, i.e. swapping the i- and j-loops. *Loop Fusion* can be applied to prevent the overhead of a for-loop, i.e. when two independent computations with the same iteration space are merged into a single for-loop.

b) Every grid should be allocated in its entirety before moving on to the next one as this prevents cache thrashing.

c) Omitted.

Task 6

a)
- Division operator replaced with multiplication of inverse.
- Multiplication by 1 removed.
- $pow\,(x, 2)$ replaced with $x \cdot x$.
- Constants pre-computed and moved out of loops.
- Expressions simplified with associativity.
- Copy of array avoided by pointer swap.

Task 7

a) Most loops cannot be vectorized due to vector dependence.

b) This can be fixed by adding **#pragma** ivdep before the loop if the vectors are in fact independent.

c) Unaligned SIMD operations need to gather from different locations, whereas aligned operations rely on the fact that the relevant data is stored contiguously in memory.

Table 4: Iterations and running times for the three problems after all optimizations have been performed.

Problem	Iterations	Running Time	Verification
1	7844	1.422s	Passed
2	13467	8.238s	Passed
3	12865	7.824s	Passed