High-Performance Parallel and Distributed Systems – COM00036H
Y3878747

Parallelisation approach [35%]

Assumption: the parallelised ports required no manual optimisations of the code (e.g. loop fusion/unrolling and vector intrinsics such as Intel's AVX/SSE for the OpenMP port).

Describe any performance analysis undertaken prior to parallelisation [100 words] (5%)

I extended the application to use a timer function to measure timings of the main operations with a –countseconds, or -s, flag [Fig 1]. As it was limited to nanosecond resolution, I also used the GNU Profiler [Fig 4]. Both corroborated that the poisson() function took the longest. Intel Advisor gave finer-grained timing information for individual loops within the functions [Fig 6]. Counting cache misses via L1 and L2 performance counters available on Viking [Fig 2] and PAPI was enabled via a –countmisses, or -m, flag, showing high data misses for compute_tentative_velocity() and update_velocity() [Fig 3].

Describe the approach taken to parallelise the application with OpenMP [100 words] (5%)

If iterations of a loop were independent, I used `for` clause in its pragma. This employed the SPMD technique for worksharing a loop across threads i.e. thread-level parallelism. However, race conditions could now occur where multiple threads updated a shared value simultaneously. I could have used critical regions and barriers to enforce synchronisation, but to avoid deadlocking, I used `reduction` clauses. For perfect loops, I reduced load imbalance via the `collapse` clause. Vectorisation techniques exploiting data-level parallelism could have been used via `simd` clauses (where backward dependencies between iterations of loops didn't exist) with vector intrinsics, but would reduce portability.

Describe the approach taken to parallelise the application with MPI [150 words] (15%)

Describe the approach taken to parallelise the application with CUDA [100 words] (10%)

All 2D arrays were converted to 1D arrays for CUDA compatibility. I used vectorisation techniques to exploit data-level parallelism in areas where backward dependencies between iterations of loops didn't exist. Data parallelism existed at the thread block scale, whereas task parallelism existed at the kernel level between independent tasks. Synchronisation barriers were used between sequentially dependent kernels. Atomic operations to avoid race conditions was only used once, when the latency would be offset by the number of other computations within a loop. To ensure efficient utilisation of resources despite inevitable latency, at least two threads should concurrently run on core.

Validation [15%]

Provide a brief description of the validation process undertaken and provide a summary of the results [150 words] (15%)

I extended the application to allow a new parameter, –validate or -i, which prints "Validated!" or "Invalidated" upon verification of the code. Without the -i tag used, the application defaults to run the standard code to completion. The standard code still runs to completion with -i parameter, however the serial, original version of the code is also run. The values of u and v calculated from the default code are then compared against the values calculated via the serial code. If the difference is greater than a certain value (defined by the MAX_ERR macro in the validated() function in main.c), the defaulted code is deemed invalid. I extended the application by moving the main() function from vortex.c into a new main.c, and kept the computation shared by both serial and parallel code portions within the main function to allow for less branching if validating, and thus reduce effects on timings.

Performance Evaluation [25%]

Outline the results collection and analysis process [100 words] (5%)

Timing functions [Fig 2] and PAPI counters [Fig 3] were used to evaluate the ports. Visit was used to visualise the output.

Provide appropriate figures and analysis demonstrating the performance of your OpenMP port [150 words] (10%)

The timing function [Fig 2] and PAPI counters [Fig 3] show that there was a significant decrease in the amount of time taken. However, some functions took slightly longer as well. Serialisation was more efficient for the functions: . This could have been mitigated by using certain #pragma omp parallel for if (iters > 100), but only in areas where reductions were not used.

Provide appropriate figures and analysis demonstrating the performance of your MPI port [150 words] (5%)

Scalsca could have been used to evaluate the MPI port.

Provide appropriate figures and analysis demonstrating the performance of your CUDA port [150 words] (5%)

Nvprof was used to analyse the CUDA port. overhead when accessing elements of an array…

Comparative Analysis [25%]

Provide appropriate figures and provide an analysis of the performance and portability of your three ports [150 words] (10%)

The CUDA port is not that portable. MPI is portable.

The OpenMP code is the most portable, as it targets multiple architecture types, such as x86 or ARM CPUs. However, my port does not target heterogenous architectures like CUDA port can. MPI can be used in message passing between multiple CPUs on different nodes in a cluster enabling wider computation, however there is a larger communication overhead on a greater scale.

The block size and grid size of the cuda port is applicable to GPUs

Discuss the differences between your ports, highlighting any advantages or disadvantages that may exist [150 words] (15%)

OpenMP targets multiple architecture types, such as x86 or ARM CPUs, but newer versions can be heterogenous. However, my port does not target heterogenous architectures like CUDA port can. MPI can be used in message passing between multiple CPUs on different nodes in a cluster enabling wider computation, however there is a larger communication overhead on a greater scale. Programming models that targets heterogenous are the most performant and energy efficienct. MPI targets distributed systems. But accelerator programming means lots of reengineering.

Appendix:

```
Timing Performance Summary
=======================================
 Allocate Arrays Time: 0.000216
 Problem Set Up Time: 0.003097
 Average Compute Tentative Velocity Time: 0.001223
 Average Poisson Time: 0.061926
 Average Compute RHS Time: 0.000231
 Average Update Velocity Time: 0.000276
 Average Apply Boundary Conditions Time: 0.000054
 Average Checkpoint Write Time: 0.000000

 Total Problem Time: 132.713945
=======================================
```

[Fig 1: Pre-parallelisation timing summary, using configuration where timestep is 0.1 and end time 1 with file I/O disabled]

```
================================================================================
  PAPI Preset Events
================================================================================
    Name        Code    Avail Deriv Description (Note)
PAPI_L1_DCM  0x80000000  Yes   No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No   Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002  Yes   Yes  Level 2 data cache misses
PAPI_L2_ICM  0x80000003  Yes   No   Level 2 instruction cache misses
PAPI_L3_DCM  0x80000004  No    No   Level 3 data cache misses
PAPI_L3_ICM  0x80000005  No    No   Level 3 instruction cache misses
PAPI_L1_TCM  0x80000006  Yes   Yes  Level 1 cache misses
PAPI_L2_TCM  0x80000007  Yes   No   Level 2 cache misses
PAPI_L3_TCM  0x80000008  Yes   No   Level 3 cache misses
```

[Fig 2: Pre-parallelisation performance counters available on Viking that I used in vortex_papi.c]

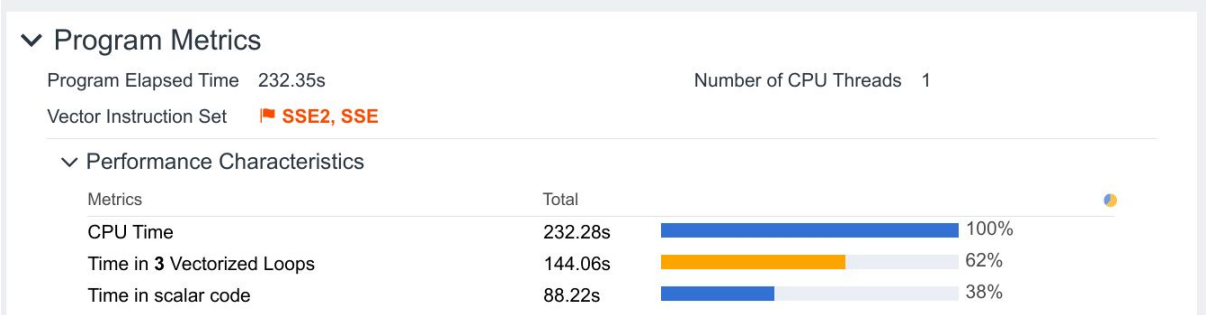
```
Step        0, Time: 1.00000000e-01 (del_t: 1.00000000e-01), Residual: 1.80316675e+01
Step        1, Time: 2.00000000e-01 (del_t: 1.00000000e-01), Residual: 1.37953704e+03
Step        2, Time: 3.00000000e-01 (del_t: 1.00000000e-01), Residual: 6.15407455e+05
Step        3, Time: 4.00000000e-01 (del_t: 1.00000000e-01), Residual: 4.32084181e+09
Step        4, Time: 5.00000000e-01 (del_t: 1.00000000e-01), Residual: 1.86754878e+17
Step        5, Time: 6.00000000e-01 (del_t: 1.00000000e-01), Residual: 3.79000681e+32
Step        6, Time: 7.00000000e-01 (del_t: 1.00000000e-01), Residual: 1.48040990e+63
Step        7, Time: 8.00000000e-01 (del_t: 1.00000000e-01), Residual:            inf
Step        8, Time: 9.00000000e-01 (del_t: 1.00000000e-01), Residual:           -nan
Step        9, Time: 1.00000000e+00 (del_t: 1.00000000e-01), Residual:           -nan
Step       10, Time: 1.10000000e+00 (del_t: 1.00000000e-01), Residual:           -nan
Step       11, Time: 1.10000000e+00, Residual:           -nan
Simulation complete.
Cache Miss Summary:

Allocate Array Events:
Level 1 Cache Misses:   Data - 570, Instruction  - 418, Total 988
Level 2 Cache Misses:   Data - 339, Instruction  - 38, Total 377


Problem Set Up Events:
Level 1 Cache Misses:   Data - 12971, Instruction  - 214, Total 13185
```

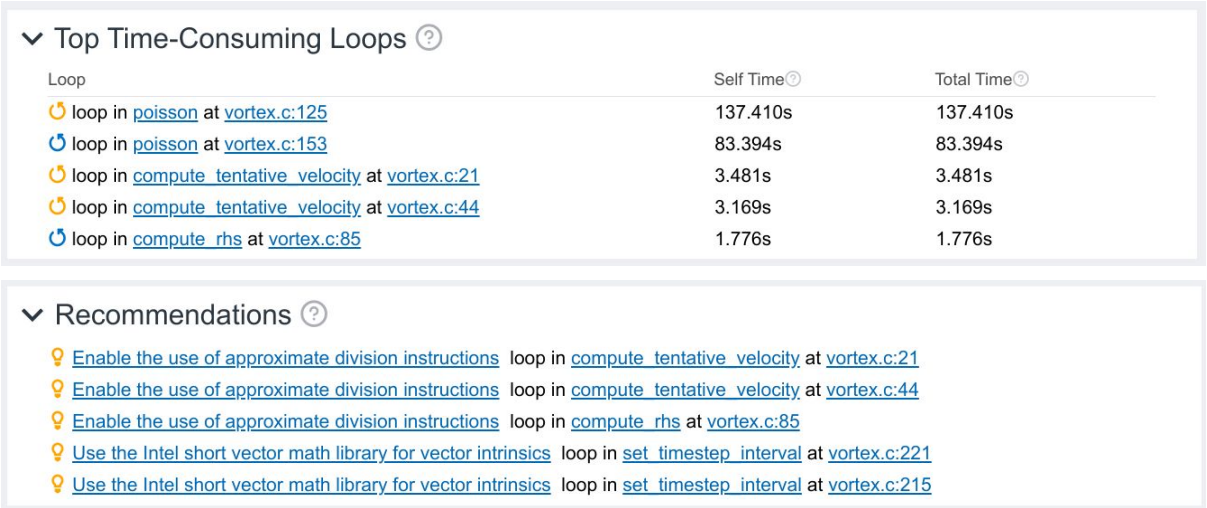
```
Level 1 Cache Misses:   Data - 12971, Instruction  - 214, Total 13185
Level 2 Cache Misses:   Data - 8602, Instruction  - 102, Total 8704


Average Compute Tentative Velocity Events:
Level 1 Cache Misses:   Data - 54611, Instruction  - 175, Total 54787
Level 2 Cache Misses:   Data - 51616, Instruction  - 163, Total 51779


Average Compute RHS Events:
Level 1 Cache Misses:   Data - 26597, Instruction  - 73, Total 26671
Level 2 Cache Misses:   Data - 25452, Instruction  - 71, Total 25524


Average Update Velocity Events:
Level 1 Cache Misses:   Data - 52675, Instruction  - 64, Total 52739
Level 2 Cache Misses:   Data - 50934, Instruction  - 63, Total 50998


Average Apply Boundary Conditions Events:
Level 1 Cache Misses:   Data - 2947, Instruction  - 80, Total 3028
Level 2 Cache Misses:   Data - 1972, Instruction  - 68, Total 2040


Total Write Events:
Level 1 Cache Misses:   Data - 0, Instruction  - 0, Total 0
Level 2 Cache Misses:   Data - 0, Instruction  - 0, Total 0


Remaining Events:
Level 1 Cache Misses:   Data - 2950, Instruction  - 7515, Total 10465
Level 2 Cache Misses:   Data - 2536, Instruction  - 5958, Total 8494
```

[Fig 3: Pre-parallelisation results yielded via PAPI in vortex_papi.c]

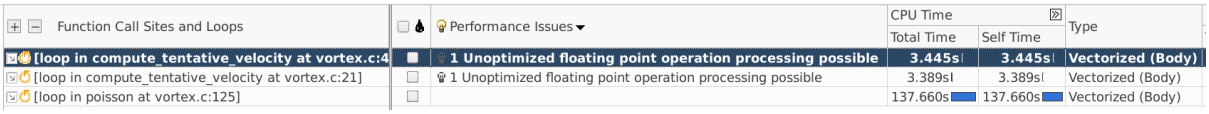
```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 97.14    126.91   126.91                             poisson
  1.80    129.26     2.35                             compute_tentative_velocity
  0.35    129.72     0.46                             update_velocity
  0.34    130.16     0.44                             compute_rhs
  0.31    130.57     0.41                             set_timestep_interval
  0.06    130.65     0.08                             apply_boundary_conditions
  0.00    130.65     0.00        6     0.00     0.00  alloc_2d_array
  0.00    130.65     0.00        6     0.00     0.00  free_2d_array
  0.00    130.65     0.00        1     0.00     0.00  alloc_2d_char_array
```

[Fig 4: Pre-parallelisation GNU Profiler results]

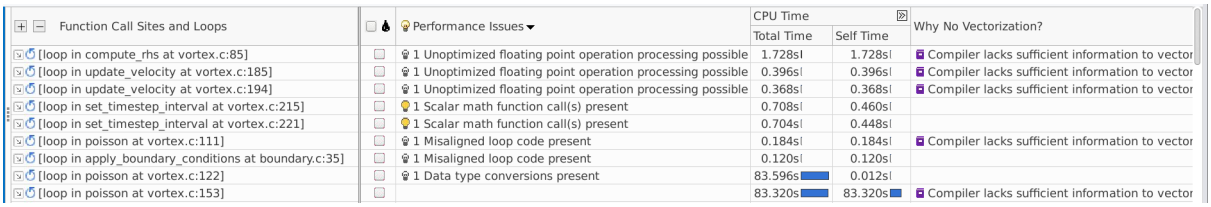




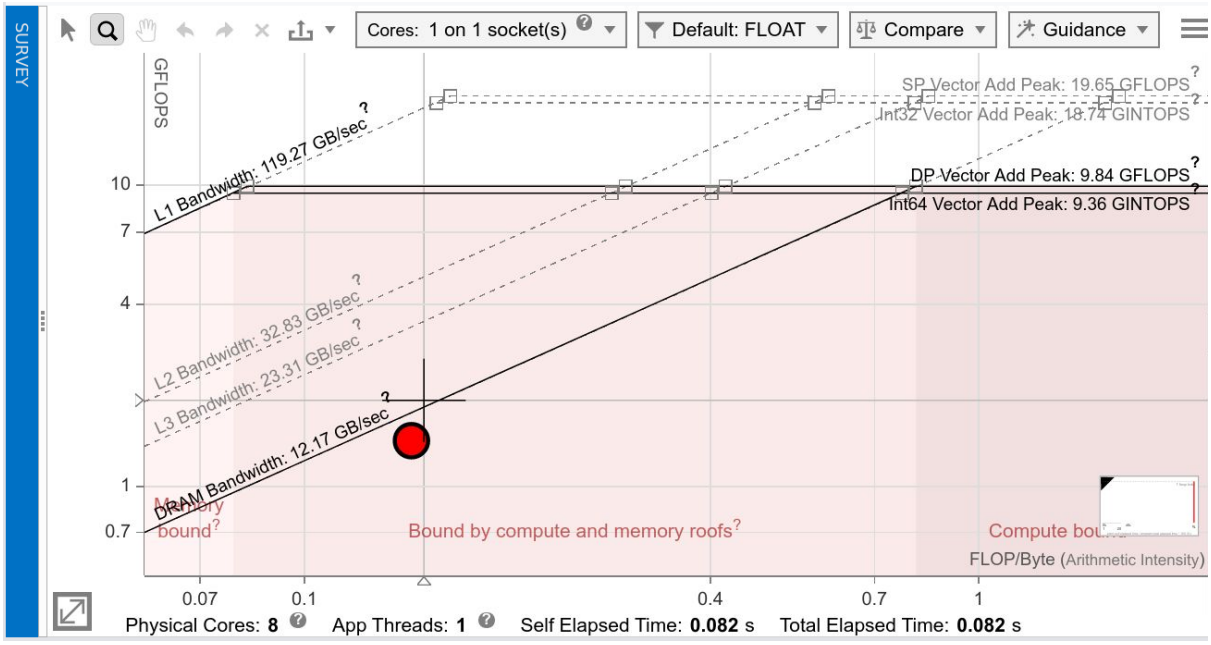
[Fig 6: Pre-parallelisation Intel Advisor results summary]



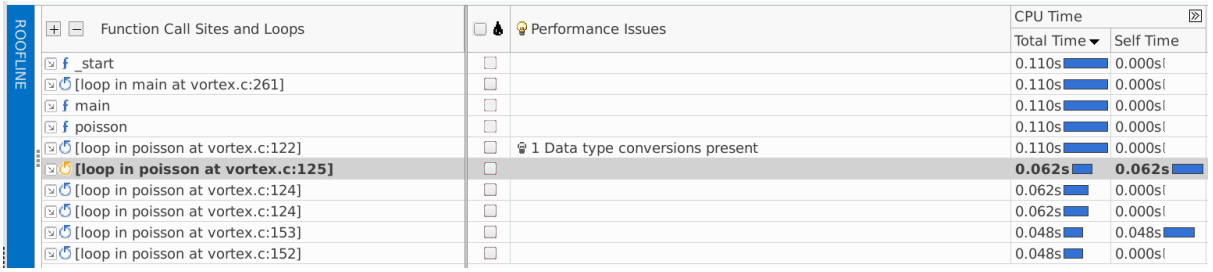
[Fig 7: Pre-parallelisation Intel Advisor Vectorization and Results' vectorisations]



[Fig 8: Non-vectorised code with performance issues via Intel Advisor Vectorization and Results pre-parallelisation]



[Fig 9: Pre-parallelisation Intel Advisor CPU/Memory Roofline Insights' roofline model]



Function Call Sites and Loops	Performance Issues	CPU Time	
		Total Time	Self Time
_start		0.110s	0.000s
[loop in main at vortex.c:261]		0.110s	0.000s
main		0.110s	0.000s
poisson		0.110s	0.000s
[loop in poisson at vortex.c:122]	1 Data type conversions present	0.110s	0.000s
[loop in poisson at vortex.c:125]		**0.062s**	**0.062s**
[loop in poisson at vortex.c:124]		0.062s	0.000s
[loop in poisson at vortex.c:124]		0.062s	0.000s
[loop in poisson at vortex.c:153]		0.048s	0.048s
[loop in poisson at vortex.c:152]		0.048s	0.000s

[Fig 10: Pre-parallelisation Intel Advisor Threading' survey]

Recommendations

Use the smallest data type loop in poisson at vortex.c:122

Top Time-Consuming Loops

Consider adding parallel site and task annotations around these time-consuming loops found during Survey analysis.

Loop	Self Time	Total Time	Trip Counts
loop in poisson at vortex.c:122	<0.001s	0.110s	100
loop in main at vortex.c:261	<0.001s	0.110s	1
loop in poisson at vortex.c:125	0.062s	0.062s	128
loop in poisson at vortex.c:124	<0.001s	0.062s	512
loop in poisson at vortex.c:124	<0.001s	0.062s	2

Function Call Sites and Loops	Trip Counts		CPU Time	
	Average	Call Count	Total Time	Self Time
_start			0.110s	0.000s
main			0.110s	0.000s
poisson			**0.110s**	**0.000s**
[loop in poisson at vortex.c:122]	100	1	0.110s	0.000s
[loop in main at vortex.c:261]	1	1	0.110s	0.000s
[loop in poisson at vortex.c:124]	2	100	0.062s	0.000s
[loop in poisson at vortex.c:124]	512	200	0.062s	0.000s
[loop in poisson at vortex.c:125]	128	102400	0.062s	0.062s
[loop in poisson at vortex.c:152]	512	100	0.048s	0.000s
[loop in poisson at vortex.c:153]	128	51200	0.048s	0.048s

Loop at 122 is called once but has iteration of 100

```
Timing Performance Summary
=========================================
 Allocate Arrays Time: 0.000193
 Problem Set Up Time: 0.002893
 Average Compute Tentative Velocity Time: 0.001325
 Average Poisson Time: 0.049781
 Average Compute RHS Time: 0.000250
 Average Update Velocity Time: 0.000300
 Average Apply Boundary Conditions Time: 0.000058
 Average Checkpoint Write Time: 0.000000

 Total Problem Time: 67.801544
=========================================
```

[Timing summary using OpenMP port, using default configuration with file I/O disabled]

```
=========================================
Cache Miss Summary=========================================


Allocate Array Events:
Level 1 Cache Misses:   Data - 717, Instruction  - 467, To
tal 1184
Level 2 Cache Misses:   Data - 368, Instruction  - 58, Tot
al 426


Problem Set Up Events:
Level 1 Cache Misses:   Data - 14469, Instruction  - 261,
Total 14730
Level 2 Cache Misses:   Data - 9360, Instruction  - 158, T
otal 9518


Average Compute Tentative Velocity Events:
Level 1 Cache Misses:   Data - 53939, Instruction  - 199,
Total 54138
Level 2 Cache Misses:   Data - 50437, Instruction  - 178,
Total 50615


Average Compute RHS Events:
Level 1 Cache Misses:   Data - 26336, Instruction  - 106,
Total 26442
Level 2 Cache Misses:   Data - 25405, Instruction  - 104,
Total 25510
```

```
Average Update Velocity Events:
Level 1 Cache Misses:   Data - 52698, Instruction  - 108,
Total 52806
Level 2 Cache Misses:   Data - 50650, Instruction  - 105,
Total 50756


Average Apply Boundary Conditions Events:
Level 1 Cache Misses:   Data - 2956, Instruction  - 54, To
tal 3011
Level 2 Cache Misses:   Data - 1919, Instruction  - 46, To
tal 1966


Average Checkpoint Write Events:
Level 1 Cache Misses:   Data - 0, Instruction  - 0, Total
0
Level 2 Cache Misses:   Data - 0, Instruction  - 0, Total
0


Remaining Looping Events:
Level 1 Cache Misses:   Data - 1944, Instruction  - 4880,
Total 6824
Level 2 Cache Misses:   Data - 1690, Instruction  - 3842,
Total 5532
=======================================
```

[Cache miss summary using OpenMP port, using configuration where timestep is 0.1 and end time 1 with file I/O disabled]

[Intel Advisor CPU/Memory Roofline Insights]

[Threading Survey and Roofline results]