PRAC5 : GPU Computing

DIETZ T., JABER A., DONNENFELD T.

December 5, 2025

1 Q1

1.1 A

```
[ahpc-4@aolin-login session5]$ ./compile.sh
main:
     53, Loop is parallelizable
     54, Loop is parallelizable
     59, Loop is parallelizable
     60, Loop is parallelizable
     64, Loop is parallelizable
     65, Loop is parallelizable
```

According to this, all loops within the main loop are parallelizable (Lines 53, 54, 59, 60, 64, 65).

1.2 B

```
[ahpc-4@aolin-login session5]$ perf stat LCPU
Jacobi relaxation Calculation: 4096 x 4096 mesh, maximum of 100 iterations
   10, 0.246094
   20, 0.176197
   ...
```

Processor	Configuration	Elapsed Time	#Instructions	#Cycles	IPC
CPU - Nehalem	N=4096, Iter=100	33.48s	18.8G	143.8G	0.13

1.3 C

```
[ahpc-4@aolin-login session5]$ ./compile_minfoall.sh
main:
     29, Loop not fused: dependence chain to sibling loop
     ...
```

Assuming the *hotspot* code section is within the main loop of the program (starting at line 50), following optimizations are applied:

- Line 53/54: Nested loop is optimized by reordering the loops (optimizing data access), vectorizing the operations with SIMD code and performing unrolling.
- Line 60: The inner loop of the second nested loop is unrolled 4 times.
- Line 64/65: Similar to line 53/54, the nested loop is optimized by reordering the loops, vectorizing the operations with SIMD code and performing unrolling.

The other loops are parallelizable but not fused together due to data dependencies.

Device (GPU) Name	NVIDIA GeForce RTX 3080	NVIDIA GeForce RTX 3080
number of SMs	68	68
device clock rate	1740 MHz	1740 MHz
size of the device L2 cache	5.242 MB	5.242 MB
size of the device memory	10.495 GB	10.495 GB
Max bandwidth from device to device memory	380 GB/s	380 GB/s
CPU target architecture	nehalem	sandybridge

2 Q2

3 Q3

3.1 A

With the following commands, we compile and run the code for the two different CPU architectures identified in the previous question.

```
nvc -fast -acc -gpu=cc80 -tp=nehalem -Minfo=all laplace.c -o LCPUnehalem
nvc -fast -acc -gpu=cc80 -tp=sandybridge -Minfo=all laplace.c -o LCPUsandy
sbatch -p cuda-ext.q -w aolin-gpu-3 --gres=gpu:1 -o Res3.txt runGPU.sh LCPUnehalem 100
sbatch -p cuda-int.q -w aolin-gpu-4 --gres=gpu:1 -o Res4.txt runGPU.sh LCPUsandy 100
```

```
     53, Loop is parallelizable
         Generating implicit copyout(Anew[1:4094][1:4094]) [if not already present]
         Generating implicit copyin(A[:][:]) [if not already present]
         Loop interchange produces reordered loop nest: 54,53
         Generated vector simd code for the loop
         Residual loop unrolled 2 times (completely unrolled)
     54, Loop is parallelizable
         Generating Tesla code
         53, #pragma acc loop gang, vector(128) collapse(2) /* blockIdx.x threadIdx.x collapsed-innermost */
         54,   /* blockIdx.x threadIdx.x auto-collapsed */
     54, Loop not fused: no successor loop
     59, Loop is parallelizable
         Generating implicit copyin(A[1:4094][1:4094]) [if not already present]
         Generating implicit copy(error) [if not already present]
         Generating implicit copyin(Anew[1:4094][1:4094]) [if not already present]
         Loop not fused: no successor loop
     60, Loop is parallelizable
         Generating Tesla code
         59, #pragma acc loop gang, vector(128) collapse(2) /* blockIdx.x threadIdx.x collapsed-innermost */
             Generating implicit reduction(max:error)
         60,   /* blockIdx.x threadIdx.x auto-collapsed */
     60, Loop unrolled 4 times
     64, Loop is parallelizable
         Generating implicit copyin(Anew[1:4094][1:4094]) [if not already present]
         Generating implicit copyout(A[1:4094][1:4094]) [if not already present]
         Loop interchange produces reordered loop nest: 65,64
         Generated vector simd code for the loop
         Residual loop unrolled 2 times (completely unrolled)
     65, Loop is parallelizable
         Generating Tesla code
         64, #pragma acc loop gang, vector(128) collapse(2) /* blockIdx.x threadIdx.x collapsed-innermost */
         65,   /* blockIdx.x threadIdx.x auto-collapsed */
     65, Loop not fused: no successor loop
```

3.2 B

We provide here only console output for the nehalem architecture as the sandybridge ouput is almost identical. The difference the sandybridge output has, is that the residual loops for the loops at lines 53 and 54 are unrolled 2 times in nehalem and 6 times in sandybridge.

We observe following optimizations for the hotspot code within the main loop (starting at line 50):

- Existing Optimizations: All previously identified optimizations (loop reordering, vectorization, unrolling) are still applied
- OpenACC directives: The compiler adds OpenACC pragmas that define how the loops are parallelized on the GPU.
- Implicit Data Copies: For each nested loop, the compiler generates implicit copyin and copyout commands to manage data transfer between host and GPU device memory.

Processor	Configuration	Elapsed Time	#Instructions	#Cycles	IPC
CPU - Nehalem	N=4096, Iter=100	33.481s	18.8G	143.8G	0.13
GPU - Nehalem	N=4096, Iter=100	12.609s	27.7G	38.5G	0.72
GPU - Sandy	N=4096, Iter=100	7.874s	21.1G	29.2G	0.72

4 Q4

	nehalem	sandybridge
Kernel Function names	main_54_gpu, main_65_gpu, main_60_gpu, main_60_gpu___red	same
Total Time Kernels	64 Mns	64 Mns
Memory Function Names	memcpyHtoD, memcpyDtoH, memset	same
Total Time Memory	7575.1 Mns	4180.1 Mns
Relative Computation Time	0.84%	1.51%

The main bottleneck we identify is the memory transfer between host and device, as most of the time is spent on the copying data from and to the GPU.

5 Q5

The single difference between the 2 implementations laplace.c and laplace2.c is following code in line 50:

```
1     #pragma acc data copy(A) create(Anew)
```

This results in the below exerpt from the compiler output. One can see that in line 51 the compiler generates the directives to create Anew at the GPU and copy A from host to device memory. By providing this, the compiler avoids all implicit copy commands it added in previous implementations: This reduces the memory transfer overhead and thus improves performance.

```
51, Generating create(Anew[:][:]) [if not already present]
    Generating copy(A[:][:]) [if not already present]
    Loop not vectorized/parallelized: potential early exits
```

The performance improvements are quantified in the table below. With the single CPU version taking 33s, the first GPU version taking 12.6s/7.9s and the optimized GPU version taking only 0.5s/???(TODO)s, We can confirm speed ups of first 2.6x/4.2x and then 66x/???x respectively **(always nehalem/sandybridge)**.

	nehalem (old)	sandybridge (old)	nehalem (new)	sandybridge (new)
Total Time Kernels	64 Mns	64 Mns	64 Mns	64 Mns
Total Time Memory	7575.1 Mns	4180.1 Mns	21 Mns	12 Mns
Relative Computation Time	0.84%	1.51%	85.29%	84.21%
#Memory Ops (DtoH/HtoD)	1600/900	1600/900	105/4	105/4
Avg Bytes (DtoH/HtoD)	16.4/14.5 MiB	16.4/14.5 MiB	0.6/16.4 MiB	0.6/16.4 MiB

6 Q6

```
CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances   Average   Minimum  Maximum        Name
```

29,0	1.968.006.058	10.000	196.800,0	195.490	198.754	main_55_gpu
29,0	1.940.668.853	10.000	194.066,0	193.762	195.970	main_61_gpu
29,0	1.939.356.955	10.000	193.935,0	192.706	195.458	main_66_gpu
11,0	732.085.764	10.000	73.208,0	71.968	78.976	main_61_gpu__red

CUDA Memory Operation Statistics (by time):

Time(%)	Total Time (ns)	Operations	Average	Minimum	Maximum	Operation
50,0	21.760.113	10.005	2.174,0	1.087	2.624.695	[CUDA memcpy DtoH]
24,0	10.778.309	10.000	1.077,0	1.024	1.856	[CUDA memset]
24,0	10.772.160	4	2.693.040,0	2.684.312	2.700.344	[CUDA memcpy HtoD]

The above GPU profiling is taken exemplary from the nehalem architecture implementation of laplace2.c. It acts similarly to the sandybridge code. The largest performance bottleneck in this setup changed from being memory transfer to kernel execution. More precisely, the 3 of the 4 main kernels invoked take up almost 90% of the total kernel execution time.