Tampere University

PARALLEL COMPUTING 2025

M.Sc. Embedded Systems

Report Part 2

NOV 2025

Tampere University

Contents

Tampere University

7. Part 2

7.1. PreReturn

Already DONE.

7.2. Porting the project to OpenCL

Summary of what we did:

- Added OpenCL support in CMake + code
- Created parallel.cl kernel for graphics part only
- Updated only parallelGraphicsEngine() for GPU execution
- CPU physics remains OpenMP
- OpenCL initialization in init(), cleanup in destroy()
- Verified correctness against sequential version
- Error checking with CL_CHECK

Codes related to OpenCL:

File	Content
CMakeLists.txt	Added find_package(OpenCL)
parallel.c	init(), parallelGraphicsEngine(), destroy()
parallel.cl	shade() kernel

Build & run (used for all measurements):

- Windows / Visual Studio (Release, x64)
  - CMake config: OpenMP + OpenCL + SDL2 (as in CMakeLists.txt)
  - Build: X64-Release configuration
  - The executable prints:
    - chosen platform/device (like in the Lab: NVIDIA CUDA / GeForce RTX 4070)
    - preferred WG multiple and max WG size (like 32 and …)

7.2.1. Benchmarking:

Include these results in the same TABLE that you used in Part 1:

1. **The Original C Version on CPU without optimizations?**
2. **The Original C version on CPU with best optimizations?**
3. **The OpenCL version on GPU, WG size 1x1?**
4. **The OpenCL version on GPU, WG size 4x4?**
5. **The OpenCL version on GPU, WG size 8x4?**
6. **The OpenCL version on GPU, WG size 8x8?**
7. **The OpenCL version on GPU, WG size 16x16?**

NO.	MODE	Benchmark		
		Physics	**Graphics**	**Total**
	Lab Setup			
1	CPU – No Optimizations – Debug	164 ms	1251 ms	1417 ms
2	CPU – No Optimizations – Release	117 ms	557 ms	676 ms
3	CPU – Best Optimizations	5 ms	16 ms	23 ms
	OpenCL			
1	WG size 1x1	5 ms	13 ms	21 ms
2	WG size 4x4	5 ms	2 ms	9 ms
3	WG size 8x4	5 ms	1 ms	8 ms
4	WG size 8x8	5 ms	1 ms	8 ms
5	WG size 16x16	5 ms	1 ms	8 ms
6	WG size 32x32	5 ms	1 ms	7 ms
	Our Setup			
1	CPU – No Optimizations – Debug	188 ms	1369 ms	1561 ms
2	CPU – No Optimizations – Release	130 ms	617 ms	750 ms
3	CPU – Best Optimizations	8 ms	24 ms	34 ms
	OpenCL			
1	WG size 1x1	7 ms	30 ms	40 ms
2	WG size 4x4	7 ms	3 ms	12 ms
3	WG size 8x4	7 ms	2 ms	11 ms
4	WG size 8x8	7 ms	2 ms	11 ms
5	WG size 16x16	7 ms	2 ms	11 ms
6	WG size 32x32	7 ms	2 ms	11 ms

* These benchmarks are only from the first 10 to 20 frames. While later frames show an increase in the delays. (Default dimension: 1920x1024)

7.2.2. WG Sizes:

Make your code so that you can change the WG size easily afterwards because you might be asked to demonstrate different sizes when showing the code to the assistant.

To make the work-group (WG) size adjustable for demonstrations, we defined the WG parameters as static variables in the source code:

```
static size_t              OCL_wgSizeX = …;
static size_t              OCL_wgSizeY = …;
```

This allows us to quickly change WG configurations during testing or demonstration without recompiling major parts of the code.

7.2.2.1 Bonus:

If some work group size or sizes did not work, try to explain why they did not work.

During our lab tests, all WG configurations up to 16x16 worked correctly, which we initially considered a positive sign. We then extended testing to 32x32, which also executed successfully and showed similar performance results to 16x16. However, increasing the WG dimensions further (32x33 or 64x64) caused program crashes with the following error: `OpenCL error -54 at…`

This corresponds to CL_INVALID_WORK_GROUP_SIZE (OpenCL error -54), indicating that the requested local size exceeded the maximum supported by the device or kernel. (We will tell which one!)

BUT there is a confusing point we faced. The “CL_KERNEL_WORK_GROUP_SIZE” in the source code is supposed to show the max WG size, and it shows 256:

```
OpenCL platform: NVIDIA CUDA | vendor: NVIDIA Corporation
OpenCL device  : NVIDIA GeForce RTX 4070 | type: GPU
Preferred WG multiple: 32 | Max WG size: 256
```

It seems normal, but the problem is that we tried 32x32 and the executable file still works! 32x32 is more than 256, which is mentioned as the max WG size.

It was confusing! 😵‍💫

To verify this, we searched a lot and we guess this is the only reason we could provide. We know that Device-wide max and Kernel-specific max are two different concepts.

Currently, “CL_KERNEL_WORK_GROUP_SIZE” shows the Kernel-specific max equal to 256. But we added the third factor, “CL_DEVICE_MAX_WORK_GROUP_SIZE”, to show what the real GPU max WG is. Here is the result:

```
OpenCL platform: NVIDIA CUDA | vendor: NVIDIA Corporation
OpenCL device  : NVIDIA GeForce RTX 4070 | type: GPU
Preferred WG multiple: 32 | Kernel Max WG size: 256 | Device max WG size: 1024
```

So, as we can see, the real max WG size supported by the device is 1024, which is also supported by “Preferred WG multiple: 32” (32x32=1024).

7.2.3. WG Performance Analysis:

Try to analyze the performance with different work group sizes. What might explain the differences?

To better understand how work-group (WG) size affects execution time, we tested several configurations while keeping all other parameters constant. These are our the tested values:

NO.	WGX x WGY	Total	works?	Observation
1	1x1	1	Yes	Basic performance and GPU utilization
2	8x8	64	Yes	Stable execution, moderate GPU utilization
3	16x16	256	Yes	Better performance due to improved parallel occupancy
4	32x32	1024	Yes	Best observed performance, highest GPU occupancy
5	32x33	>1024	No	Exceeds device limit
6	64x64	>1024	No	Exceeds device limit

Performance improves as the work-group size increases up to the hardware limit.

The observed differences arise mainly from:

1. **Thread-level Parallelism**
   Larger wgSize allow more concurrent threads to run, hiding memory latency and better utilizing GPU cores.

2. **Warp Alignment:**
   Since NVIDIA warps are 32 threads wide, WG sizes that are multiples of 32 (32x16, 32x32) achieve perfect warp alignment and avoid idle lanes.

3. **Driver/Runtime Scheduling Overhead:**
   Too many small groups increase kernel launch and synchronization overhead, lowering performance.

4. **Hardware Limits:**
   When wgSizeX x wgSizeY > 1024, the GPU rejects the launch due to exceeding the maximum threads per work-group, triggering “CL_INVALID_WORK_GROUP_SIZE”.

Performance scaled positively with WG size up to 32x32, after which it crashed.

This matches NVIDIA’s architectural preference for warp-aligned groups and confirms that 1024 threads per workgroup is the practical and theoretical optimum for this kernel on our RTX 4070 GPU.

7.2.4. Window Sizes – OpenCL vs. C (CPU)

Try reducing the window size to a minimal size, such as 80x80. The WINDOW WIDTH and WINDOW HEIGHT are defined at the beginning of the code that controls this.

Fastest WG size for OpenCL version is 16x16 while 32x32 shows the same performance. So, we can continue with 32x32:

NO.	MODE	Benchmark		
	Window Size	Physics	Graphics	Total
Lab Setup				
OpenMP – CPU				
1	1920 x 1024	5 ms	16 ms	22 ms
2	1440 x 720	5 ms	8 ms	14 ms
3	720 x 384	5 ms	2 ms	7 ms
4	80 x 80	5 ms	0 ms	5 ms
5	20 x 20	5 ms	0 ms	5 ms
6	5 x 5	5 ms	0 ms	5 ms
OpenCL with WG Size 32x32				
1	1920 x 1024	5 ms	1 ms	8 ms
2	1440 x 720	5 ms	0 ms	6 ms
3	720 x 384	5 ms	0 ms	5 ms
4	80 x 80	5 ms	0 ms	5 ms
5	20 x 20	5 ms	0 ms	5 ms
6	5 x 5	5 ms	0 ms	5 ms
Our Setup				
OpenMP – CPU				
1	1920 x 1024	8 ms	24 ms	35 ms
2	1440 x 720	8 ms	12 ms	21 ms
3	720 x 384	8 ms	3 ms	12 ms
4	80 x 80	8 ms	0 ms	8 ms
5	20 x 20	8 ms	0 ms	8 ms
6	5 x 5	8 ms	0 ms	8 ms
OpenCL with WG Size 32x32				
1	1920 x 1024	8 ms	2 ms	11 ms
2	1440 x 720	8 ms	1 ms	9 ms
3	720 x 384	8 ms	0 ms	8 ms
4	80 x 80	8 ms	0 ms	8 ms
5	20 x 20	8 ms	0 ms	8 ms
6	5 x 5	8 ms	0 ms	8 ms

If the OpenCL version running on the GPU was slower than C on CPU with this image size, what is the reason for this?

In our measurements, total frame time is equal to = physics (CPU/OpenMP) + graphics (GPU/OpenCL)

For tiny images (5x5, 20x20, 80x80), the OpenCL graphics kernel is essentially negligible (~0 ms), but the physics step remains the same "ms" because it performs PHYSICSUPDATESPERFRAME Euler iterations on the CPU.

We left physics on the CPU; therefore, for small images, the CPU physics dominates, and GPU advantages appear only when the image is large enough to show its power.

In brief, CPU physics is constant-time and dominates when pixel count is small.

Tampere University

7.2.5. Bonus – Optimization

As a final (and the most significant) bonus question, you are now free to implement any optimizations to your program. How fast can you get it to run?

We worked diligently to achieve optimal performance, and all the benchmarks above related to OpenCL are derived from the optimized version of the code.

What we have done so far:

Part2 modifications:

- Implementing NVIDIA GPU selection
  - Verified GPU selection and capability by printing platform, device name, and vendor information.
- Optimized kernel and data flow:
  - Updated only position buffers each frame, while keeping satellite color data constant on the GPU
  - Used persistent OpenCL buffers to minimize memory allocations and transfers across frames.
- Optimized overall execution pipeline:
  - Overlapped CPU and GPU workloads where possible and ensured efficient synchronization using clFinish() only when needed.

We transformed the original CPU-bound implementation into a GPU-accelerated hybrid (CPU + GPU) system capable of high-resolution real-time simulation and rendering.

All the reported benchmarks and analyses were obtained from this optimized final version.

7.2.5.1. Bonus in Bonus

Add some more thoughts on how you would attempt to further improve the total performance of this system?

No idea how much these approaches would help the total performance (because they are just advices) but we can consider these:

1. Move physics to GPU:
GPU physics kernel updates positions and velocity in device memory, then SHADE kernel uses them in-place, which avoids position transfer between cl and c file, per frame.

2. other ways of using sqrt:
In our implementation we tried to avoid sqrt, but if we are forced to use this in the mathematics, then it's better to use **fsqrt** or **rsqrt**.

The rest ideas came to our mind were used in our solution.