Università della Svizzera italiana

Faculty of Informatics

Institute of Computational Science

# Notes from learning CUDA by example of wave equation stencil

Dmitry Mikushin, Pietro Benedusi

(University of Lugano)

July 9, 2014

# 4th assignment of PDCLab course @ USI

Pick up your favourite test program from previous assignments (Game Of Life (2d), wave13pt (3d), or hidden parallelism kernel (2d)) and port it to:

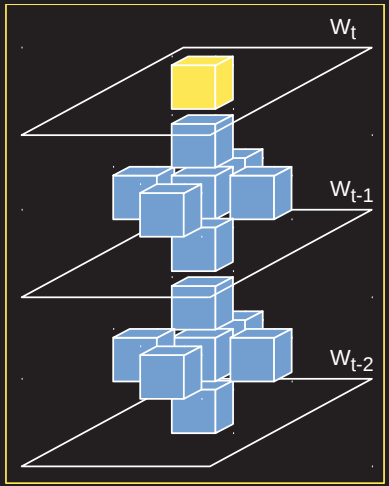
- CUDA
- Cell Broadband Engine
- **(optional)** Xeon Phi

# 4th assignment of PDCLab course @ USI

- Time iterations (“forget“ the oldest data and replace it with newest)
- 3D Loop on every iteration:

```
for (int k = 2; k < ns — 2; k++)
    for (int j = 2; j < ny — 2; j++)
        for (int i = 2; i < nx — 2; i++)
        {
                w2[k][j][i] =  m0 * w1[k][j][i] — w0[k][j][i] +

                m1 * (
                        w1[k][j][i+1] + w1[k][j][i—1]  +
                        w1[k][j+1][i] + w1[k][j—1][i]  +
                        w1[k+1][j][i] + w1[k—1][j][i]) +
                m2 * (
                        w1[k][j][i+2] + w1[k][j][i—2]  +
                        w1[k][j+2][i] + w1[k][j—2][i]  +
                        w1[k+2][j][i] + w1[k—2][j][i]);
}
```



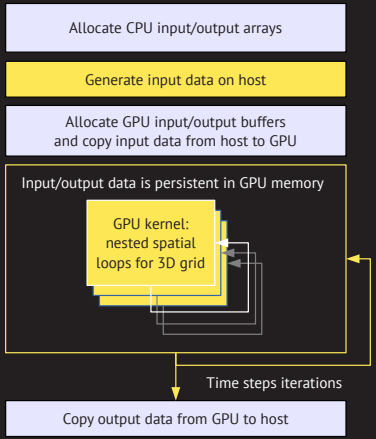
# Notes on understanding the algorithm

Familiarity with data flow is crucial:

- What are the inputs and the outputs
- How propagation moves from current level to the next one

Ways to explore the program behavior:

- Run the code in debugger line by line



# Notes on understanding CUDA

1. Study simple examples, e.g. cuda_gpu_test and compute_grid_3d
2. How to select the compute grid config?
   - Choose grid point : GPU thread mapping: 1:1 or N:1 (with jumping by blockDim.x increment)
   - Use blocks of size divisible by warpSize, better to have more smaller blocks, than less larger
3. How to handle multiple dimensions?
   - Either linearize into 1D and recover (i,j,k) from 1D index in GPU kernel:

```
wave13pt<<<szarray/THREADS_PER_BLOCK,THREADS_PER_BLOCK>>>(
    nx, ny, ns, m0, m1, m2, d_w0, d_w1, d_w2);
```

   - Or use multidimensional grid, that allows to map multidimensional problems naturally:

```
wave13pt<<< X, Y, Z >>>(...)
```

# Notes on bug hunting: basic checks

I've got the working version with wrong results. Where should I search for an error?

1. Check error status of every CUDA function call with e.g. `CUDA_ERR_CHECK` macro
2. Check error status of every CUDA kernel with `CUDA_ERR_CHECK(cudaGetLastError())`
3. Try to run only single time step
4. Use `printf()` to check if swapping of time levels is done correctly
5. You can also do `printf()` from GPU

# Notes on bug hunting: synchronization

- Check if the CUDA kernel is synchronized properly:

```
for (int it = 0; it < nt; it++)
{
    wave13pt<<<config.gridDim, config.blockDim>>>(
        nx, ny, ns,
        config,
        m0, m1, m2, w0p, w1p, w2p);
    CUDA_SAFE_CALL(cudaGetLastError());
    CUDA_SAFE_CALL(cudaDeviceSynchronize());

    real* w = w0p; w0p = w1p; w1p = w2p; w2p = w;
    int idx = idxs[0]; idxs[0] = idxs[1]; idxs[1] = idxs[2]; idxs[2] = idx;
}
```

- Pointer swapping does not involve CUDA functions $\Rightarrow$ dependency will not be detected $\Rightarrow$ incorrect result without explicit synchronization.

# Notes on bug hunting: cuda-memcheck

- Detect out-of-range memory accesses with cuda-memcheck ≈ valgrind for GPU

```
$ cuda-memcheck ./wave13pt 512 256 256 10
========= CUDA-MEMCHECK
m0 = 0.680375, m1 = -0.035206, m2 = 0.094366
initial mean = 0.000024
init time = 0.219432 sec
device buffer alloc time = 0.023246 sec
data load time = 0.062911 sec (5.960818 GB/sec)
========= Invalid __global__ read of size 4
=========     at 0x000012f0 in /home/marcusmae/apc/trainings/schools/↩
    cscs_summer_school_2014/slides/pietro/black/wave13pt_gpu/cuda/../wave13pt.cu:642:↩
    wave13pt
=========     by thread (95,0,0) in block (2,237,251)
=========     Address 0xb181409bc is out of bounds
```

# Notes on performance

1. CUDA code for GPU is compiled with full optimization by default (i.e. implicit -O3)
2. Initially the perf was poor, until the block size of {128, 1, 1} was selected
3. Always maintain memory coalescing