2nd International EULAG Workshop

Beyond MPI – Exploring OpenMP and OpenCL Perspectives of EULAG Parallelization

Roman Wyrzykowski
Krzysztof Rojek
Łukasz Szustak
[roman, krojek, lszustak]@icis.pcz.pl

Czestochowa University of Technology

Agenda

- The scope of our research on EULAG model
- Motivations
- Architecture of GPU
  - Architecture of NVIDIA Tesla C1060
  - Architecture of ATI Radeon HD 5870
- OpenCL: emerging standard for multicore architectures
- Perspectives of EULAG parallelization
- Performance results
- Conclusions and future work

The scope of our research on EULAG model

- EULAG is a numerical solver for all-scale geophysical flows
- The underlying anelastic equations are either solved in an EULerian (flux form), or a LAGrangian (advective form) framework
- Our reasearch includes linear version of Multidimensional Positive Definite Advection Transport Algorithm (MPDATA)

Motivations

- Current GPUs are highly efficient, multi-core processors, which have the computing power of **several TFLOPS**
- GPUs offer a fast, inexpensive solution, but understanding the parallel tradeoffs is crucial
- GPU allows for creating of many thousands of threads, which has significant influence on performance of parallel codes
- Available software (OpenCL, CUDA) facilitates the implementation of general-purpose computation on GPU

Architecture of NVIDIA Tesla C1060 (1/2)

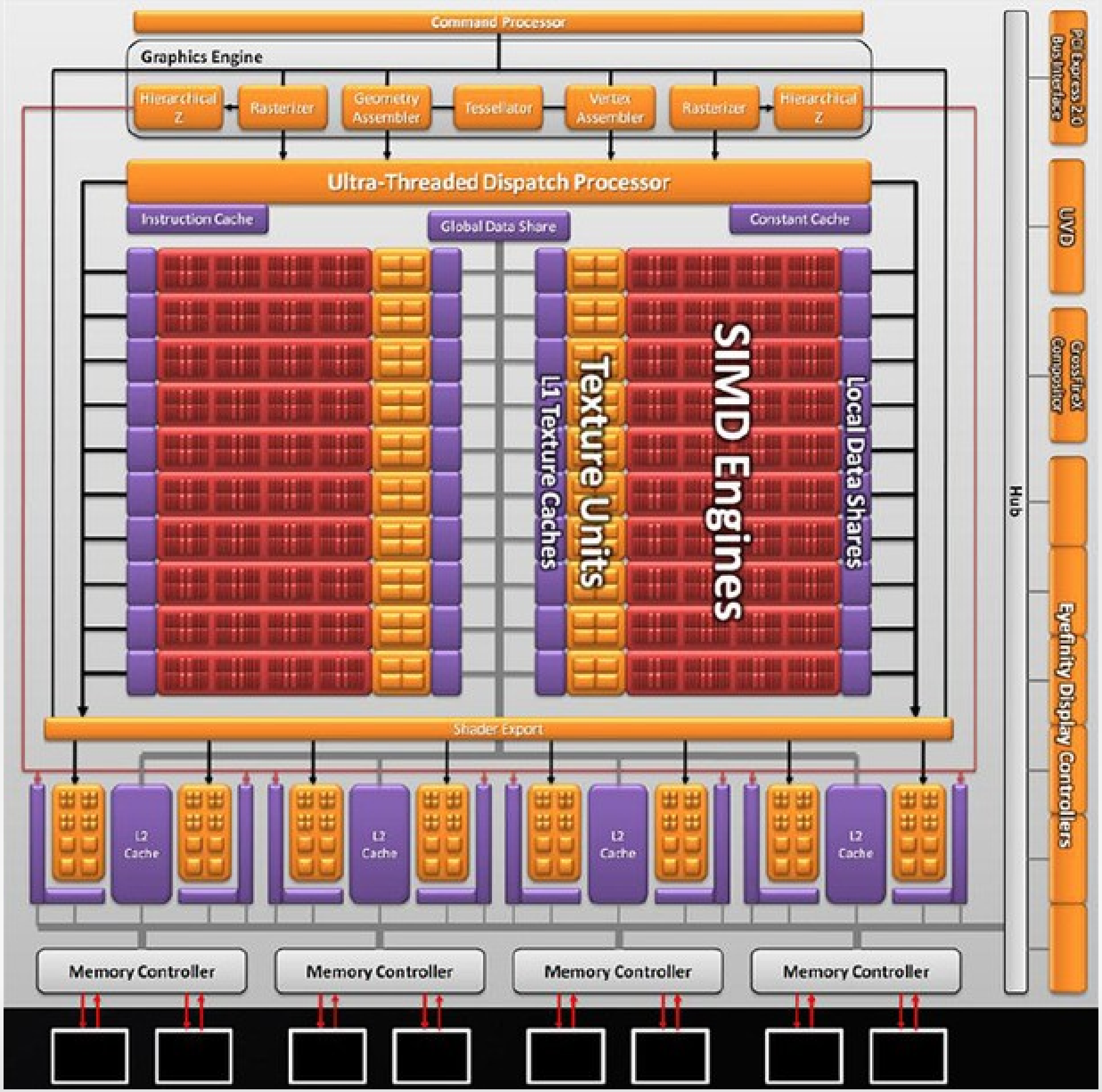
- 10 processing clusters (TPC)
- 3 compute units per processing cluster
- 8 processing elements per compute units = **240 processing elements**
- 1296 MHz – clock fruequency
- 16 KB of local memory
- 64 KB of constant buffer
- 4 GB of global memory
- 102.4 GB/s of global memory bandwidth
- It gives 240 * 1.296 * 2 (MADD) = **0.622 TFLOPS** in single precision

Architecture of NVIDIA Tesla C1060 (2/2)

Architecture of ATI Radeon HD 5870 (1/2)

- 20 compute units
- 16 processing elements per compute unit
- 5 stream processors per processing element = **1600 stream processors**
- 850 MHz – clock frequency
- 32 KB of local memory
- 64 KB of constant buffer
- 1 GB of global memory
- 153.6 GB/s of bandwidth
- It gives 1600 * 0.850 * 2 (MADD) = **2.72 TFLOPS**

Architecture of ATI Radeon HD 5870 (2/2)



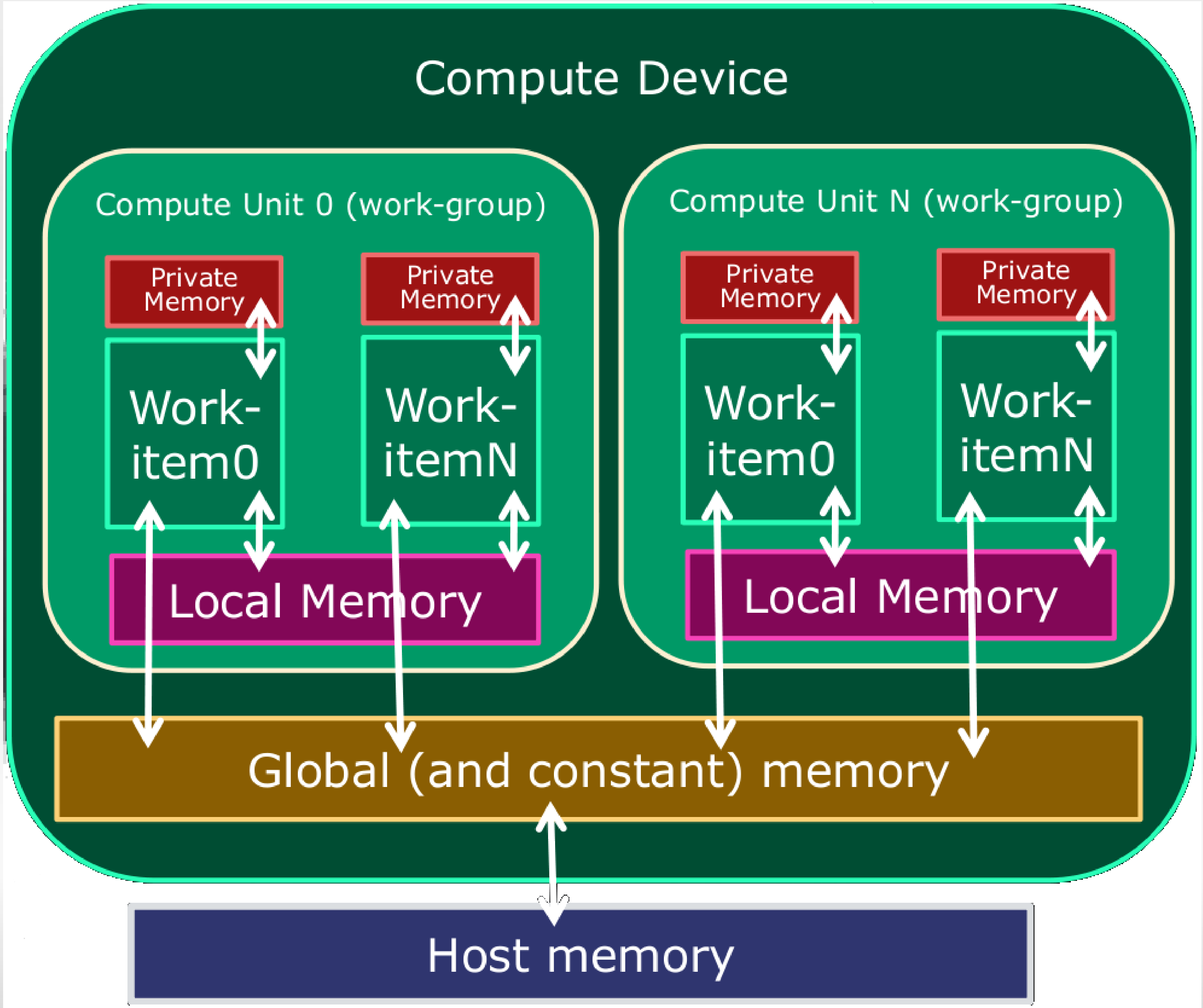
OpenCL: emerging standard for multicore architectures (1/2)

- OpenCL (Open Computing Language) is open, royalty-free standard for parallel programming of heterogenous computing systems
- OpenCL standard defines the host API and the programming language
- OpenCL allows for creating portable code across different devices and architectures, including CPUs, GPUs and other processors like DSPs or Cell\B.E.

OpenCL: emerging standard for multicore architectures (2/2)

- **Host** is connected to one or more Compute Devices
- Compute Device is a collection of one or more **Compute Units**
- Compute Unit consist of **Processing Elements** that execute code as SIMD or SPMD
- **Kernel** – Equivalent to C function executed on Compute Device
- Kernels are instanced as **work-items** (”threads”) that are grouped in **work-groups**
  - No synchronization between work-groups, they are independent
  - Barriers for synchronizing work-items within work-group

OpenCL: Memory Model (1/2)



OpenCL: Memory Model (2/2)

- **Private memory** is assigned per every work-item
- **Local Memory:** At least 32KB split into blocks, each available to any work-item in a given work-group
- **Global/Constant Memory:** Not synchronized
- **Host Memory:** On the CPU
- Host Memory management is explicit
  - You must move data from host → global → local and back

Perspectives of EULAG Parallelization (1/2)

- Our implementation is based on the following part of MPDATA kernel:

```
if(j<m && i<n)
      for(k=0; k<l; ++k)
        x(i, j, k)-=
          ( f1(i+1, j, k)-f1(i, j, k)
           +f2(i, j+1, k)-f2(i, j, k)
           +f3(i, j, k+1)-f3(i, j, k) )/h(i, j, k);
```

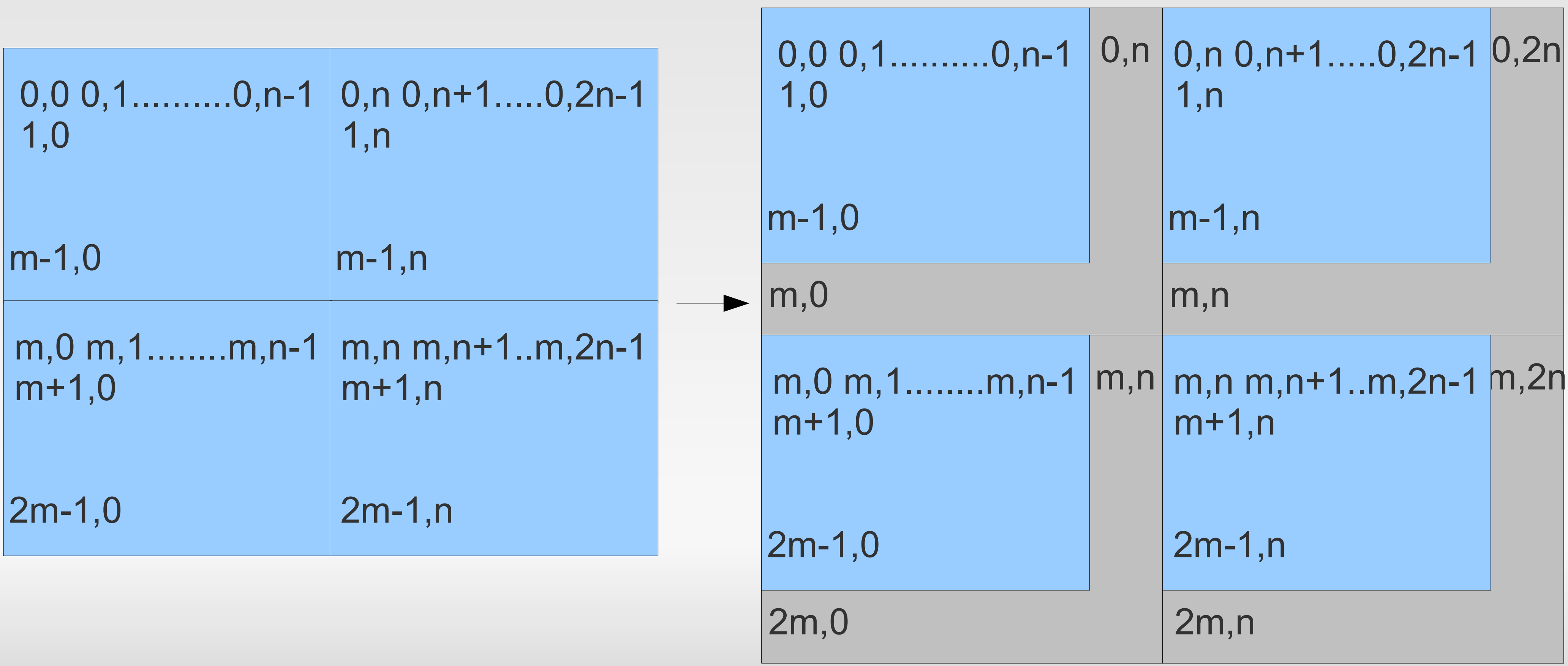
- Where `f1,f2,f3` are computed using donnor-cell scheme:

```
#define donor(y1,y2,a)(fdim(a,0.0f)*(y1)-fdim(0.0f,a)*(y2))
```

  - `dim` returns `x - y` if x > y, `+0` if x is less than or equal to y

Perspectives of EULAG Parallelization (2/2)

- 2D grid decomposition with group size of **n** x **m**
- To avoid dependencies between work-groups, additional work-items are required



Code autotuning (1/2)

- Optimizations of code on different GPUs architectures is based on **autotunig** technique
- **Autotuning** is a technique of self-adaptation of algorithm to some features of architecture like:
  - Number of compute units (number of work-groups)
  - Number of processing elements per compute unit (size of work-group)
  - Preffered vector width (number of floats)
  - Size of private and local memory

Code autotuning (2/2)

- Autotuning is based on two methods:
  - getting some informations about architecture using OpenCL API and generating compiler directives – **results are generated immediately**
    - Preffered vector size, informations about available resources...
  - searching a space of possible solutions and generating the best setup of algorithm – **time consuming optimization**
    - Size of work-group, size of local memory...

Performance results (1/2)

- The algorithm was tested on the following hardware:
  - AMD Phenom(tm) II X4 955 Processor – single-core implementation
  - NVIDIA Tesla C1060
  - ATI Radeon HD 5870

Performance results (2/2)

	CPU	NVIDIA Tesla	ATI Radeon
Kernel time [s]	0.75	0.041	0.039
Speedup	1	18.29	**19.23**
Bandwidth [GB/s]	-	**2.57092**	1.35215
Kernel + data reciving time [s]	-	0.06	0.08
Speedup	-	**12.5**	9.38
Kernel + data sending + data reciving time [s]	-	0.16	0.27
Speedup	-	**4.68**	2.78
Memory usage [MB]	**514.016**	584.543	584.543

Conclusions

- NVIDIA was tested with Linux operating system, while ATI used Windows7
- On ATI we achieved beter performance of computing but worse bandwith than on NVIDIA
- Our code can run on different GPUs
- Performance on GPUs was higher than on CPU
- The implementation is optimized on the very basic level

Future Work

- GPU+CPU implementation (OpenCL)
- GPUs+CPU (OpenCL)
- GPUs+CPUs + shared memory (OpenCL, OpenMP)
- GPUs+CPUs + distributed memory (OpenCL, MPI)
- Exploring innovative heterogenous technologies like AMD Fusion (GPUs+CPUs in a single processor)
- Load balancing between GPUs and CPUs
- Implementation of other parts of EULAG code using GPUs
- Porting our code to Fortran

Thank YOU for your attention!