Computation of Lattice Boltzmann in CPU and GPU heterogeneous environment

Tomasz Rybak Tomasz.Rybak@unige.ch
Jonas Latt Jonas.Latt@unige.ch
Bastien Chopard Bastien.Chopard@unige.ch

ICMMES 2011, Lyon, 2011-07-07

Outline

Lattice Boltzmann

- Useful simulations are large
  - Require many nodes
  - and small time step
- Large memory and CPU requirements
- Easy to parallelize (usually)

CPU vs. GPU

CPU

- Few threads
- Large memory per core
- General-purpose
- Hidden hardware details
- Large multi-level cache

GPU

- Many threads
- Small memory per core
- Specialised
- Visible hardware details
- Small cache

GPUs

- Single core slower than CPU
- Many cores available
- Thousands of threads running simultaneously
- High cost of accessing memory
- Fasts switching of threads to hide delays
- Threads in a group should follow the same code execution path
- Memory hierarchy
- Threads hierarchy: thread, warp, block, grid

GPU hardware details

- Threads hierarchy
  - Thread
  - Warp
  - Block
  - Grid
- Memory types
  - Private (local) memory
  - Shared memory
  - Global memory
  - Constant memory
  - Texture memory
- Proper usage of those is needed for achieving acceptable performance

CUDA and OpenCL

- APIs to access special-purpose devices
- Assume programs consisting of many threads
- Data item-thread mapping
- Ability to group threads

CUDA

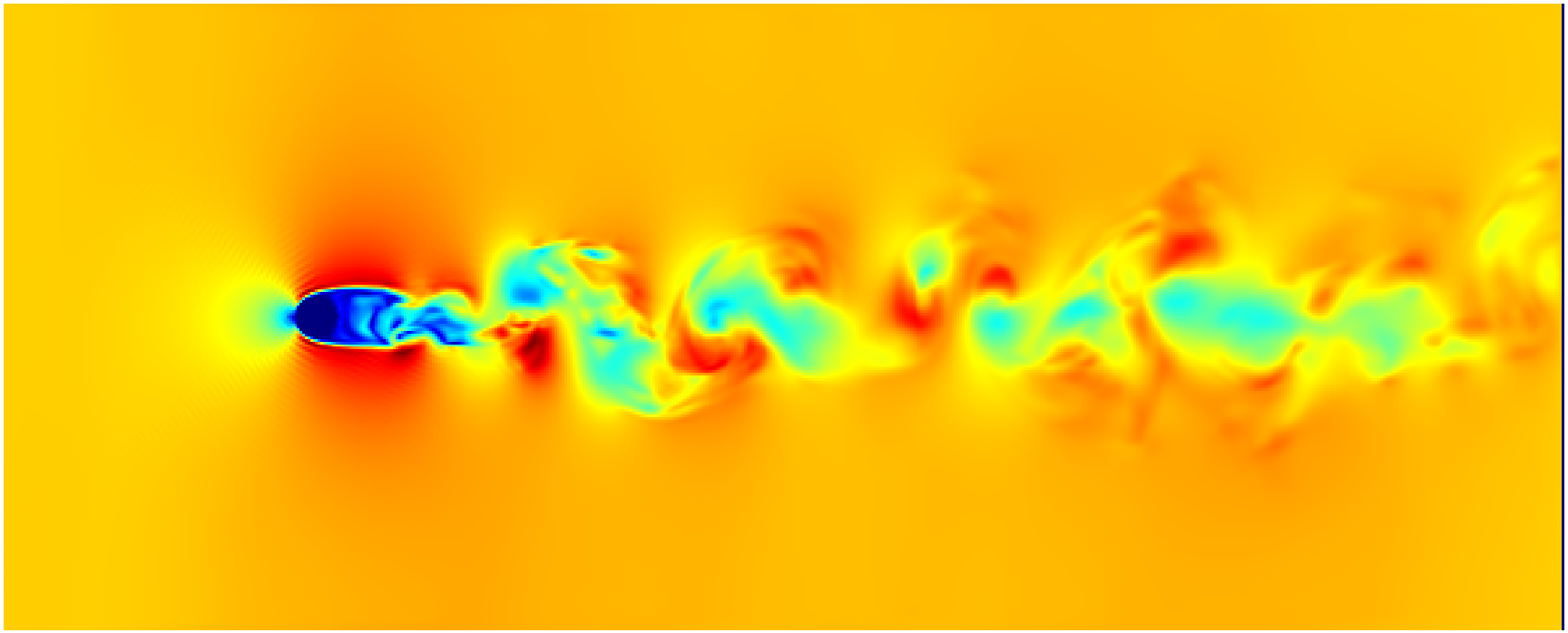
- Proprietary
- One vendor (risk of lock-in)
- Quick development of new features
- Additional libraries (CUFFT, CURAND, CUBLAS)
- One device type (GPU)

OpenCL

- Open
- Many vendors (NVIDIA, AMD, Apple)
- Design by committee
- Less libraries
- Many device types (CPU, GPU, accelerator)
- Interesting hardware (AMD Fusion)

Simulations

- Lid driven cavity
- Cylinder



Initial architecture

- Use the devices' features
  - GPU works on nodes of the same type
  - CPU works on the boundary conditions
- (Currently) One GPU subdomain

Implementation

- Two libraries
  - Palabos base library
  - Sailfish GPU-based library written using PyCUDA
- Avoid NIH syndrome
- Focus on core features of each library
- Let the Sailfish to focus on GPU performance

PyCUDA and PyOpenCL

- Python wrappers around CUDA and OpenCL
- Asynchronous calls to devices
- Fast prototyping
- Metaprogramming:
  - Code writing code
  - Dynamic kernel creation
  - Generating code depending on performance and available features

Sophisticated solution

- Palabos memory and threads
- Sailfish memory
- PyCUDA memory (GPUArray)
- Raw GPU mamory
- We are subverting some layers to inject Palabos state into Sailfish
- Sailfish is supposed to work on its own
- We are tying to steer its behaviour from Palabos
- Sailfish must not overwrite changes made by us

Cooperation between libraries

- Sailfish steered using parameters
- Differences in behaviour
  - Singularities in cavity

Implementation problems

1. Numerical types: double vs. float
2. Memory layout:
   - Population
   - Order of indices: x, y, z vs. z, y, x
   - Array stride (on GPU)
3. Asynchronous PyCUDA calls
   - implicit
   - explicit
4. Lack of support for some functions (OpenCL NVIDIA 1.1)
5. Problems with having many domains on one device or in one process

Implementation problems

1. Numerical types: double vs. float
2. Memory layout:
   - Population
   - . . . possible solution?
   - Order of indices: x, y, z vs. z, y, x
   - Array stride (on GPU)
3. Asynchronous PyCUDA calls
   - implicit
   - explicit
4. Lack of support for some functions (OpenCL NVIDIA 1.1)
5. Problems with having many domains on one device or in one process

Implementation problems

1. Numerical types: double vs. float
2. Memory layout:
   - Population
   - . . . possible solution?
   - Order of indices: x, y, z vs. z, y, x
   - Array stride (on GPU)
3. Asynchronous PyCUDA calls
   - implicit
   - explicit
   - using streams
4. Lack of support for some functions (OpenCL NVIDIA 1.1)
5. Problems with having many domains on one device or in one process

Deployment

- Might be complex:
- CUDA or OpenCL libraries
- Python
- PyCUDA or PyOpenCL

- Sailfish

Deployment

- Might be complex:
- CUDA or OpenCL libraries
- Python
- PyCUDA or PyOpenCL
- . . . and their dependencies
- Sailfish

Deployment

- Might be complex:
- CUDA or OpenCL libraries
- Python
- PyCUDA or PyOpenCL
- . . . and their dependencies
- Sailfish
- . . . and its dependencies

Results

- Cooperation of Palabos and Sailfish:
  - two-way data transfer
  - running simulation on both CPU and GPU
- So far gives improper physical results

Near future plans

- Finish implementation and test it throughly
- Support for many subdomains
- Copy envelope instead of full subdomain
  - New CUDA functions
  - Lack of support on NVIDIA OpenCL
- Multi-GPU support
  - new CUDA library and hardware
  - Sailfish support

Long-term plans

- Better GPUArray class in PyCUDA and PyOpenCL (compyte)
- Usage of new GPU architectures
  - New device types
  - New architecture: AMD Fusion
- Metaprogramming
- Injection of geometry-describing class into Sailfish

Thanks

Palabos team.

Thanks

Palabos team.

Michal Januszewski, Sailfish author.

Thanks

Palabos team.

Michal Januszewski, Sailfish author.

Andreas Kloeckner, author of PyCUDA and PyOpenCL.

Q & A

Thank you for your attention.

Questions?

Tomasz Rybak
tomasz.rybak@unige.ch