Asynchronous OpenCL/MPI Discontinous Galerkin Solver for Conservation laws

Malcolm Roberts[*,1], Michaël Gutnic[1], Philippe Helluy[1], Michel Massaro[1], Thomas Strub[2]

[1]University of Strasbourg and INRIA TONUS
[2]AxesSim

SMAI, 2015-06-11

[*]`malcolm.i.w.roberts@gmail.com`, `www.malcolmiwroberts.com`

Outline

- The DG method
  - Macrocell/subcell formulation
  - Hexahedral elements
  - Gauss-Legendre vs. Gauss-Lobatto points
- Implementation
  - The `OpenCL` programming language
  - Performance analysis
- Example simulation results

Evolution Equations

We consider the general hyperbolic equation

$$\partial_t w + \sum_{k=1}^{k=d} \partial_k F^k(w) = S, \tag{1}$$

in $d =$ dimensions. F is the flux and S the source term.

Examples:

- Navier–Stokes equations
- Maxwell's equations
- MHD
- Vlasov equations

We would like to numerically solve such equations in complex geometries with as general boundary conditions as possible.

Discontinuous Galerkin Method

The physical domain is divided into cells.

In each cell L, w is projected onto a finite set of basis functions $\psi_i^L(x)$:

$$w(x,t) \approx \sum_{i\in L} w_L^i(t)\psi_i^L(x). \tag{2}$$

The evolution equation is approximated by

$$\int_L \partial_t w \psi_i^L - \int_L F(w,w,\nabla\psi_i^L) + \int_{\partial L} F\left(w_L, w_R, \boldsymbol{n}_{LR}\right)\psi_i^L = S_i^L, \tag{3}$$

where $\boldsymbol{n}_{LR}$ is the normal vector from cell L to cell R.

The DG formulation is good for conserving invariants.

Elements can be curved and meshes can be non-conformal.

Macrocell/subcell

The physical domain is divided into an unstructured mesh of macrocells.

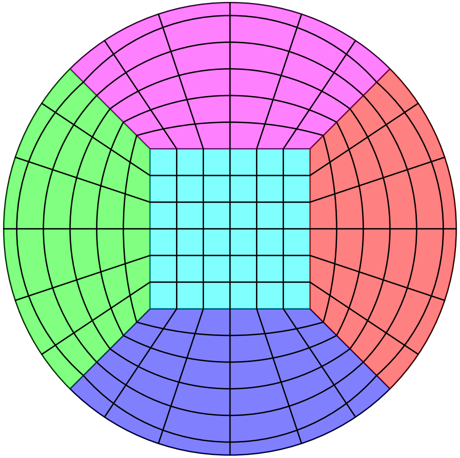
The macrocells are divided into subcells.

- The curvature is constant within a macrocell.
- The subcells are arranged in a Cartesian grid in the reference space.

Advantages:

- We are able consider complex geometries.
- Memory access is coalescent when looping over subcells in a macrocell.
- Macrocells provide coarse-grained parallelism.

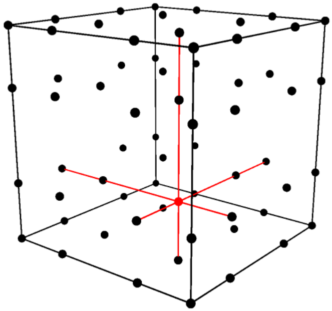
Macrocell/subcell



Hexahedral elements

We make use of hexahedral elements.

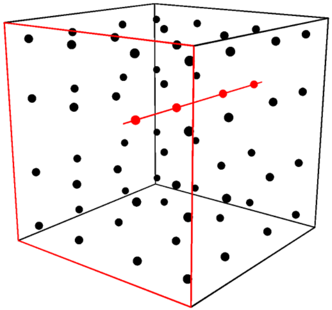
- Memory access is coalescent within an element.
- The volumic term is non-zero on a subdimensional set.



Gauss Legendre Quadrature

Gauss Legendre collocation points gives convergence

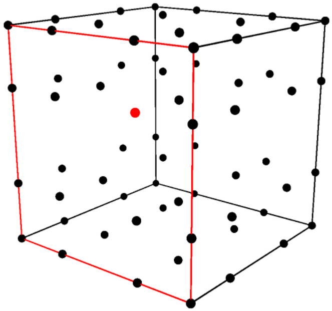
- degree n gives $\mathcal{O}\left(dx^{n+1}\right)$ spatial convergence,
- but the flux terms are expensive.



Gauss Lobatto Quadrature

Gauss Lobatto collocation points gives convergence

- degree n gives $\mathcal{O}(dx^n)$ spatial convergence,
- but the flux terms are cheaper.



Source-term computation

The computation of the source term is divided into the following steps:

- Volume terms
- Subcell interface flux terms
- Macrocell interface flux terms
- Mass division
- Computation of source terms

We time-step using either RK2 or RK4.

Implementation

We use `OpenCL` for our implementation.

- `OpenCL` is a `C`-like programming language.
- It works on `CPU`s, `GPU`s, and `MIC`s.
- No restriction to a specific vendor (unlike `CUDA`).

`GPU`s and `MIC`s are interesting computing platforms:

- Many cores (fine-grained parallelism).
- Fast memory access on board (less bottleneck).
- `CPU`-to-`GPU` transfers are slow (so stay on the board if possible).

Implementation

We would like to use multiple `GPU`s.

- Macrocells allow for coarse-grained parallelism
- Inter-macrocell flux computation doesn't need a lot of bandwidth.
- We can use `MPI` between nodes with different macrocells.

The `StarPU` runtime environment is an interesting possibility for managing data transfers and a heterogeneous computing environment.

Performance analysis

How do we compare performance between platforms?

Roofline: Count FLOPs, count i/o, compare with manufacturer specs.

Questions:

1. is $a * x + b$ two FLOPs or one FLOP?
2. Do multiplication, division, and addition count equally?
3. What about integer operations?
4. Is the listed speed a specification or an advertisement?

Comparison is difficult

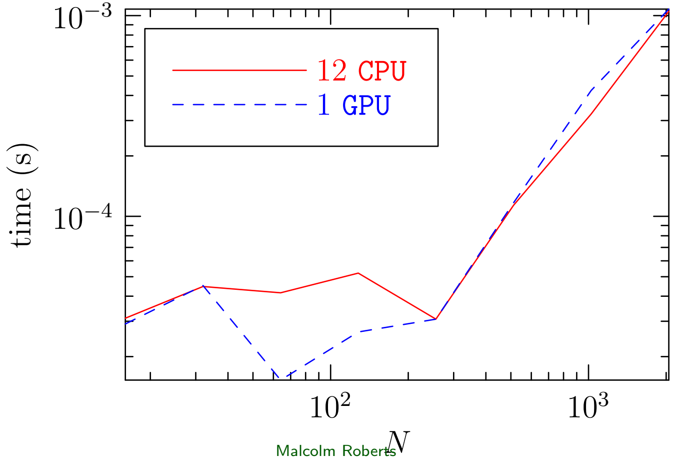
- `CPU` manufacturers do not publish GFLOP/s

We get around 200 GFLOP/s on a 1TFLOP/s `GPU`.

Comparison of CPU and GPU performance

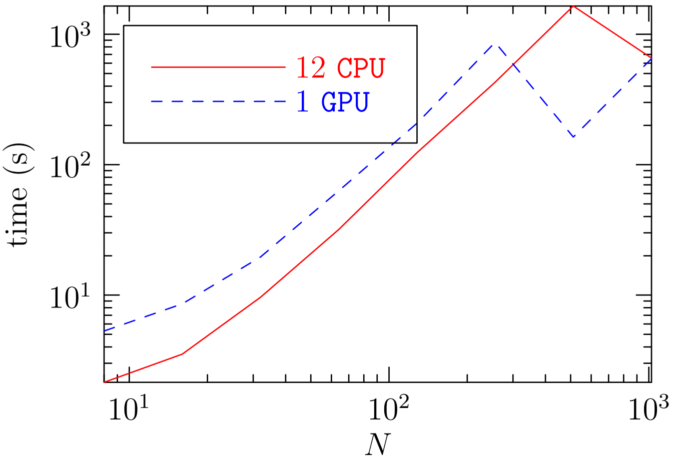
We can compare platforms by comparing execution speed.

We use clFFT, an FFT library written in OpenCL by AMD.



Comparison of `CPU` and `GPU` performance

We perform the same test with `schnaps`



Comparison of `CPU` and `GPU` performance

`schnaps`:

- Fully uses all 12 `CPUs`.
- Is faster than our `C` implementation.
- Has similar performance on 12 `CPU` or on `GPU`.

So we claim that `schnaps` is well-optimized for both for `CPUs` and `GPUs`

Who's on the MIC?

MIC: Many integrated cores, Intel's Xeon Phi.

- A combination of a GPU and a CPU
- Same advertised performance as Tahiti GPU we're using.
- Works with OpenCL.

Requires optimization:

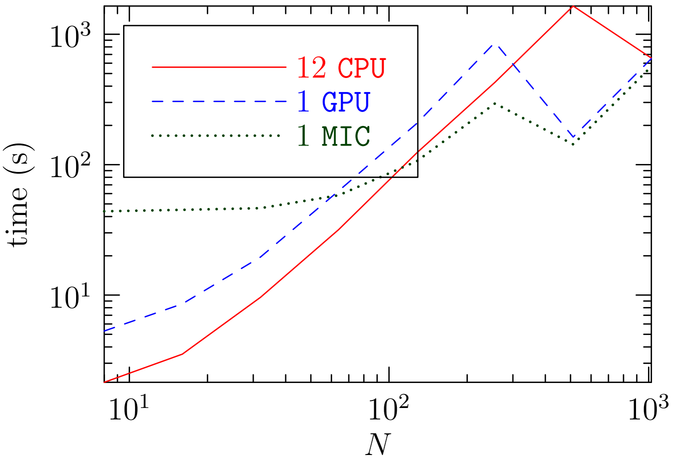
"There is no free lunch."

Evaluating kernels on Xeon Phi to accelerate Gysela application by G Latu, M Haefele, J Bigot, V Grandgirard, T Cartier-Michaud, F Rozar

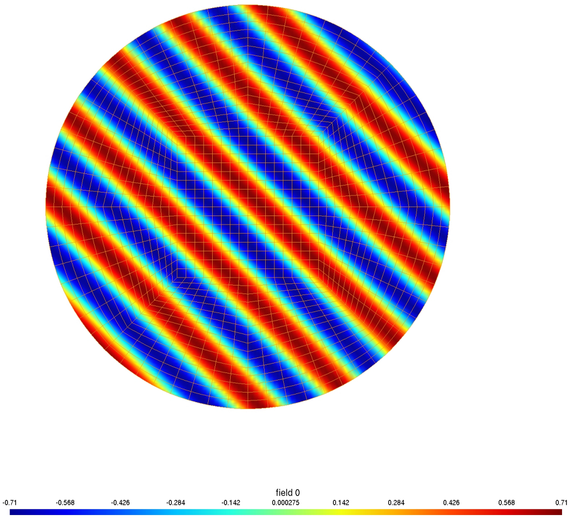
"There may be no lunch even if you pay."

Who's on the MIC?

schnaps works well on the Xeon Phi.



Example simulation: Maxwell's equations



Conclusion

`schnaps` is a new general-purpose DG implementation in `OpenCL`.

- For the DG method we use:
  - An unstructured mesh of macrocells.
  - A structured mesh of subcells within each macrocell.
  - hexahedral elements.
  - Gauss-Lobatto collocation points.
- We can run on `CPUs`, `GPUs`, and `MICs`
  - and we can do so efficiently.
  - Performance metric is based on comparison with established libraries.

`schnaps` is avilable at `schnaps.gforge.inria.fr`

Thank you for your attention!
Merci pour votre attention!