SIEMENS
Ingenuity for life

Implémentation GPU d'une méthode DG en temps pour l'équation des ondes

30 Novembre 2022

Rose-Cloé Meyer

Hadrien Bériot, Gwénaël Gabard, Axel Modave, Thijs van Putten

Ensta - Équipe-Projet POEMS - Siemens

GPU

- Image processing
- High performance computing

NVIDIA QUADRO GV100

Parallelism :

- Shared Memory (threads) : OpenMP, Cuda
- Distributed memory : MPI, Multi-GPU

GPU:

- Advantages: high number of threads performing an elementary task.
- Limits: the memory of the chip is small, the memory transfers are slow.
- Simple/double precision
- Cuda

GPU organization

- GPU: set of N Stream Multiprocessors (SM)
- Threads
- Blocks of threads (max. 1024)
- Warps: 32 threads
- Grid (1D,2D,3D)

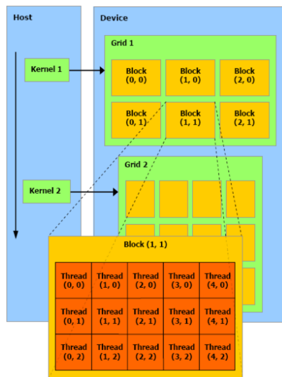


Image from [1]

Kernel Call: myKernel <<< Dg, Db >>> with
Dg the Grid size (int or dim3)
Db theBlock Size (int ou dim3)

Memory

- Global memory: accessible by all the threads
- Shared memory: The shared variables are only accessible by the threads in the same block.
- Register memory: visible only to the thread that wrote it and lasts only for the lifetime of that thread.
- Local, constant, texture memory

Structure of the code

- Initialization on the CPU
- Allocation of the memory on the GPU
- Copy the data from the CPU to the GPU
- Execution of the massively parallel kernels on the GPU
- Collect the results on the CPU: cudaMemcpyDeviceToHost
- Free the GPU memory
- Visualize the results

Data transfers

- Allocation of variables on GPU:

```
int N = 256;
float *A_cpu = (float*)malloc(N);
float *A_gpu;
cudaMalloc(&A_gpu, N*sizeof(float));
```

- Copy of variables from CPU to GPU

```
cudaMemcpy(A_gpu, A_cpu, sizeof(float)*N,
cudaMemcpyHostToDevice);
```

- Copy of variables from GPU to CPU

```
cudaMemcpy(A_cpu, A_gpu, sizeof(float)*N,
cudaMemcpyDeviceToHost);
```

- Release allocations

```
cudaFree(A_gpu);
```

- Non-blocking transfers: streams

Example

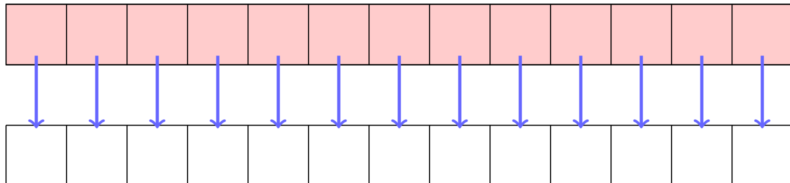
```
// CPU code
int main() {
  int N = 256;
  float *x = (float*)malloc(N);
  float *x_step = (float*)malloc(N);
  float *x_dot = (float*)malloc(N);
  // Initialization x, x_step, x_dot
  for (int n = 0; n < N; n++)
      x[n] = x_step[n] + a * x_dot[n];
  return 0;
}
```

```
// GPU code
int main() {
  int N = 256;
  float *x = (float*)malloc(N);
  float *x_step = (float*)malloc(N);
  float *x_dot = (float*)malloc(N);
  // Initialization x, x_step, x_dot
  // Allocation x, x_step, x_dot on the GPU
  cudaMalloc(&x_gpu, N*sizeof(float));
  cudaMalloc(&x_step_gpu, N*sizeof(float));
  cudaMalloc(&x_dot_gpu, N*sizeof(float));
  // Copy of x, x_step, x_dot on the GPU
  cudaMemcpy(x_step_gpu, x_step, sizeof(float)*N,
  cudaMemcpyHostToDevice);
  cudaMemcpy(x_dot_gpu, x_dot, sizeof(float)*N,
  cudaMemcpyHostToDevice);
  kernel<<<1, N>>>(a, x_gpu, x_step_gpu, x_dot_gpu);
  // Rapatriate the results
  cudaMemcpy(x, x_gpu, sizeof(float)*N,
  cudaMemcpyDeviceToHost);
  return 0;
}
```

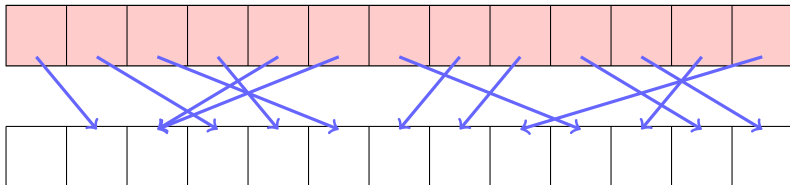
```
// Kernel
__global__ void kernel(float a, float * x,
 float * x_step, float * x_dot)
{
  int n = threadIdx.x;
  x[n] = x_step[n] + a * x_dot[n];
  return;
}
```

Coalescence

- Good coalescence



- Bad coalescence



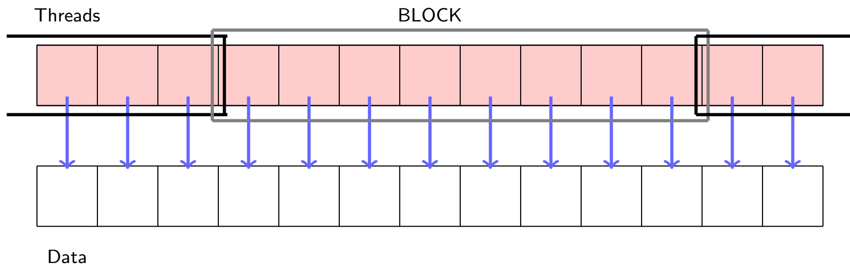
Example

```
// GPU code
int main() {
  int N = 256;
  float *x = (float*)malloc(N);
  float *x_step = (float*)malloc(N);
  float *x_dot = (float*)malloc(N);
  // Initialization x, x_step, x_dot
  // Allocation x, x_step, x_dot on the GPU
  cudaMalloc(&x_gpu, N*sizeof(float));
  cudaMalloc(&x_step_gpu, N*sizeof(float));
  cudaMalloc(&x_dot_gpu, N*sizeof(float));
  // Copy of x, x_step, x_dot on the GPU
  cudaMemcpy(x_step_gpu, x_step, sizeof(float)*N, cudaMemcpyHostToDevice);
  cudaMemcpy(x_dot_gpu, x_dot, sizeof(float)*N, cudaMemcpyHostToDevice);
  Dg = 4;
  kernel<<<Dg, N/Dg>>>(a, x_gpu, x_step_gpu, x_dot_gpu);
  // Rapatriate the results
  cudaMemcpy(x, x_gpu, sizeof(float)*N,
  cudaMemcpyDeviceToHost);
  return 0;
}
```

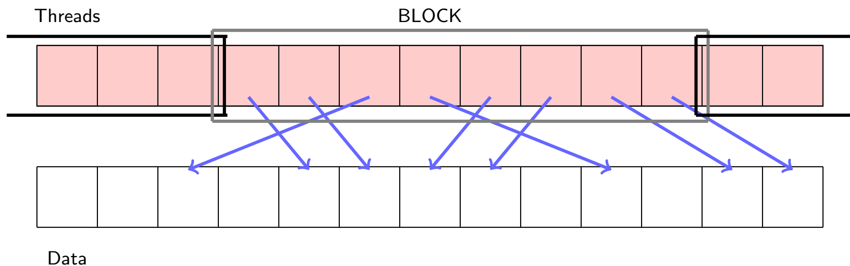
```
// Kernel
__global__ void kernel(float a, float * x,
 float * x_step, float * x_dot)
{
  int n = blockIdx.x * blockDim.x + threadIdx.x;
  x[n] = x_step[n] + a * x_dot[n];
  return;
}
```

Coalescence

- Good coalescence



- Bad coalescence



DG method

- Acoustic equation in the time domain

$$\frac{\partial \mathrm{p}}{\partial t} + \rho\,\mathrm{c}^2\,\nabla\cdot\mathbf{v} = f(t)\,\delta(\mathbf{x}-\mathbf{x}_0)\,,$$

$$\rho\,\frac{\partial \mathbf{v}}{\partial t} + \nabla \mathrm{p} = 0\,,$$

$\mathrm{p}(\mathbf{x},t)$ the pressure, $\mathbf{v}(\mathbf{x},t)$ the particle velocity, ρ the density and c the phase velocity.

- Conservation equations

$$\mathbf{q} = \begin{pmatrix} \mathrm{p} \\ \rho\,\mathrm{c}\,\mathrm{v}_x \\ \rho\,\mathrm{c}\,\mathrm{v}_y \\ \rho\,\mathrm{c}\,\mathrm{v}_z \end{pmatrix},\quad \mathbf{f}_x = \begin{pmatrix} \mathrm{c}\,q_2 \\ \mathrm{c}\,q_1 \\ 0 \\ 0 \end{pmatrix},\quad \mathbf{f}_y = \begin{pmatrix} \mathrm{c}\,q_3 \\ 0 \\ \mathrm{c}\,q_1 \\ 0 \end{pmatrix},\quad \mathbf{f}_z = \begin{pmatrix} \mathrm{c}\,q_4 \\ 0 \\ 0 \\ \mathrm{c}\,q_1 \end{pmatrix},$$

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}_d}{\partial x_d} = \mathbf{s}\,.$$

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}_d}{\partial x_d} = \mathbf{s}\,.$$

- Variational formulation

$$\int_{D_k} \frac{\partial \mathbf{q}}{\partial t}\, \phi(\mathbf{x})\, d\Omega + \int_{D_k} \frac{\partial \mathbf{f}_d}{\partial x_d}\, \phi(\mathbf{x})\, d\Omega = \int_{D_k} \mathbf{s}\, \phi(\mathbf{x})\, d\Omega\,,$$

DG method

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}_d}{\partial x_d} = \mathbf{s}\,.$$

- Variational formulation

$$\int_{D_k} \frac{\partial \mathbf{q}}{\partial t}\,\phi(\mathbf{x})\,d\Omega + \int_{D_k} \frac{\partial \mathbf{f}_d}{\partial x_d}\,\phi(\mathbf{x})\,d\Omega = \int_{D_k} \mathbf{s}\,\phi(\mathbf{x})\,d\Omega\,,$$

$$\int_{D_k} \frac{\partial \mathbf{q}}{\partial t}\,\phi(\mathbf{x})\,d\Omega + \int_{D_k} \frac{\partial \phi(\mathbf{x})}{\partial x_d}\,\mathbf{f}_d\,d\Omega + \sum_{f=1}^{Nfaces} \int_{F} \mathbf{g}_f\,\phi(\mathbf{x})d\Gamma = \int_{D_k} \mathbf{s}\,\phi(\mathbf{x})\,d\Omega\,,$$

DG method

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}_d}{\partial x_d} = \mathbf{s}\,.$$

- Variational formulation

$$\int_{D_k} \frac{\partial \mathbf{q}}{\partial t}\, \phi(\mathbf{x})\, d\Omega + \int_{D_k} \frac{\partial \mathbf{f}_d}{\partial x_d}\, \phi(\mathbf{x})\, d\Omega = \int_{D_k} \mathbf{s}\, \phi(\mathbf{x})\, d\Omega\,,$$

$$\int_{D_k} \frac{\partial \mathbf{q}}{\partial t}\, \phi(\mathbf{x})\, d\Omega + \int_{D_k} \frac{\partial \phi(\mathbf{x})}{\partial x_d}\, \mathbf{f}_d\, d\Omega + \sum_{f=1}^{Nfaces} \int_{F} \mathbf{g}_f\, \phi(\mathbf{x}) d\Gamma = \int_{D_k} \mathbf{s}\, \phi(\mathbf{x})\, d\Omega\,,$$

- Upwind numerical fluxes, calculated using a Riemann solver.

- Field approximations:

$$q_i(\mathbf{x}, t) \; = \; \sum_{n=1}^{N_p} \phi(\mathbf{x}) \, \underline{q(t)} \, ,$$

$$\mathbb{M}^K \frac{\partial \mathbf{q}}{\partial t} \; = \; \sum_{d=1}^{3} \mathbb{G}_d^K \, \mathbf{f}_d \; + \sum_{f=1}^{Nfaces} \mathbb{A}^K \mathbf{g}_f \; + \; \mathbb{M}^K \, \mathbf{s}^K \, ,$$

- Field approximations:

$$q_i(\mathbf{x},t) = \sum_{n=1}^{N_p} \phi(\mathbf{x})\, \underline{q(t)}\,,$$

$$\mathbb{M}^K \frac{\partial \mathbf{q}}{\partial t} = \sum_{d=1}^{3} \mathbb{G}_d^K\, \mathbf{f}_d + \sum_{f=1}^{Nfaces} \mathbb{A}^K \mathbf{g}_f + \mathbb{M}^K\, \mathbf{s}^K\,,$$

- System:

$$\frac{\partial \mathbf{q}}{\partial t} = \sum_{d=1}^{3} \mathbb{D}_d^K\, \mathbf{f}_d + \sum_{f=1}^{Nfaces} \mathbb{B}_f^K \mathbf{g}_f + \mathbf{s}^K\,,$$

DG method: discretization

- Field approximations:

$$q_i(\mathbf{x},t) = \sum_{n=1}^{N_p} \phi(\mathbf{x})\, \underline{q(t)}\,,$$

$$\mathbb{M}^K \frac{\partial \mathbf{q}}{\partial t} = \sum_{d=1}^{3} \mathbb{G}_d^K\, \mathbf{f}_d + \sum_{f=1}^{Nfaces} \mathbb{A}^K \mathbf{g}_f + \mathbb{M}^K\, \mathbf{s}^K\,,$$

- System:

$$\frac{\partial \mathbf{q}}{\partial t} = \sum_{d=1}^{3} \mathbb{D}_d^K\, \mathbf{f}_d + \sum_{f=1}^{Nfaces} \mathbb{B}_f^K \mathbf{g}_f + \mathbf{s}^K\,,$$

The matrices $\mathbb{D}_d^K$, $\mathbb{B}_f^K$ are stored contiguously.
The vectors $\mathbf{f}_d$, $\mathbf{g}_f$ are stored contiguously.

$$\frac{\partial \mathbf{q}}{\partial t} = \mathbb{K}^K\, \mathbf{F}^K + \mathbf{s}^K\,.$$

DG method: quadrature-free elements

$$\mathbb{M}^K \frac{\partial \mathbf{q}}{\partial t} = \sum_{d=1}^{3} \mathbb{G}_d^K \, \mathbf{f}_d + \sum_{f=1}^{Nfaces} \mathbb{A}^K \mathbf{g}_f + \mathbb{M}^K \, \mathbf{s}^K ,$$

- Quadrature-free elements: Straight elements

$$\mathbb{M}_{ij}^K = \int_{D_k} \phi_i(\mathbf{x}) \, \phi_j(\mathbf{x}) \mathbf{J}(\mathbf{x}) d\Omega ,$$

$$\mathbb{M}^K = \mathbf{J}^K \, \mathbb{M}^{ref} ,$$

DG method: quadrature-free elements

$$\mathbb{M}^K \frac{\partial \mathbf{q}}{\partial t} = \sum_{d=1}^{3} \mathbb{G}_d^K \, \mathbf{f}_d + \sum_{f=1}^{Nfaces} \mathbb{A}^K \mathbf{g}_f + \mathbb{M}^K \, \mathbf{s}^K ,$$

- Quadrature-free elements: Straight elements

$$\mathbb{M}_{ij}^K = \int_{D_k} \phi_i(\mathbf{x}) \, \phi_j(\mathbf{x}) \mathbf{J}(\mathbf{x}) d\Omega ,$$

$$\mathbb{M}^K = \mathbf{J}^K \, \mathbb{M}^{ref} ,$$

- Global system

$$\frac{\partial \mathbf{q}}{\partial t} = \mathbb{K}^K \, \mathbf{F}^K + \mathbf{s}^K .$$

$$\Rightarrow \frac{\partial \mathbf{Q}}{\partial t} = \mathbb{K}^{ref} \, \mathbf{F} + \mathbf{S} .$$

GPU for DG method

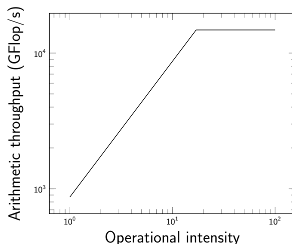
$$\frac{\partial \mathbf{q}}{\partial t} = \sum_{d=1}^{3} \mathbb{D}_d^K \, \mathbf{f}_d + \sum_{f=1}^{Nfaces} \mathbb{B}_f^K \mathbf{g}_f + \mathbf{s}^K ,$$

Initialization on the CPU
Loop in time:

- Computation of the source terms $\mathbf{s}^K$
- Computation of the volume terms $\mathbf{f}_d$
- Computation of the surface terms $\mathbf{g}_f$
- Multiplication (cublas)
- Update of the time scheme (RK6)
- Synchronization of the fluxes

GPU performances

- Operational intensity $= \dfrac{\#(FLOP)}{\#(Byte)}$.
- The arithmetic throughput is the number of float operations by second (GFLOP/s).
- The memory bandwidth is the number of bytes transferred from and to the global memory (GB/s).



Optimization

- Reduce the computational time.
- Increase the arithmetic throughput.
- Increase the memory bandwidth.

Optimizations:

- Change the granularity of the arrays to optimize the coalescence of the threads.
- Change the size of thread blocks.
- Reduce the global memory transfer: use shared memory instead of global memory.
- Reduce the number of floating points.

Results

GPU Characteristics:

- Memory Bandwidth Up to 870 GB/s
- 5,120 NVIDIA CUDA Cores
- Double-Precision Performance: 7.4 TFLOPS
- Single-Precision Performance: 14.8 TFLOPS
- 32GB of GPU Memory

NVIDIA QUADRO GV100

Test:

- 3D Test, Cube 1m
- Point-source
- Impedance condition
- 20 million nodes
- Quadrature-free elements
- Single-Precision
- Profiler: nvprof

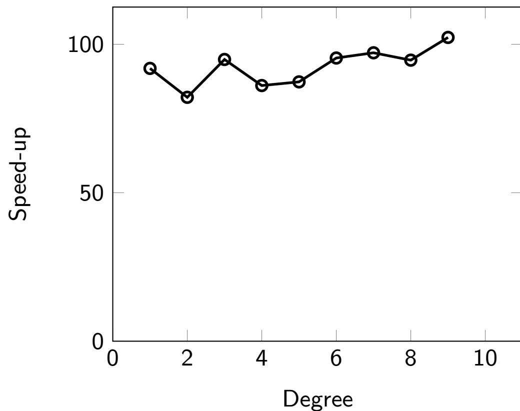


Figure: Speed-up

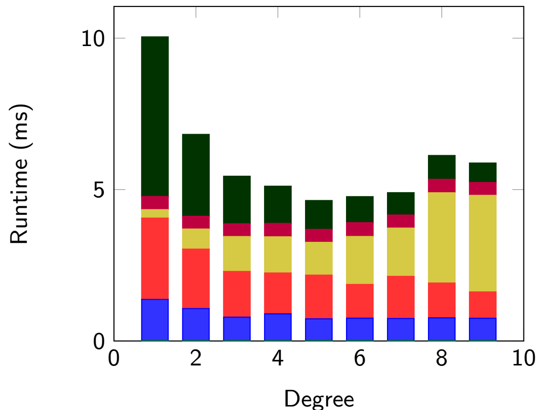
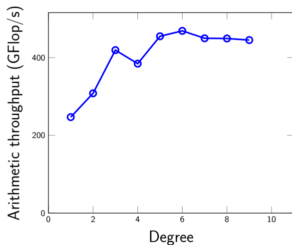
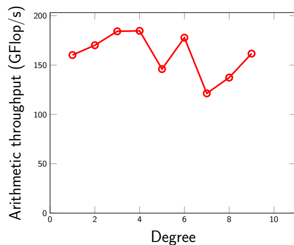


Figure: Runtime for one local time step.

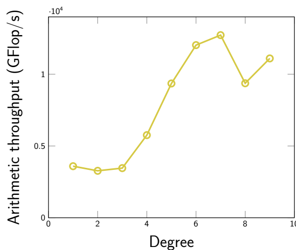
Optimization



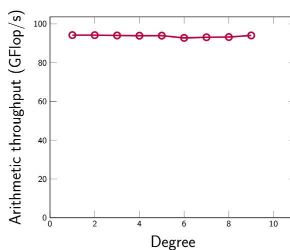
(a) Volum Terms



(b) Surface Terms



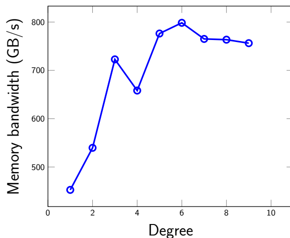
(c) Multiply



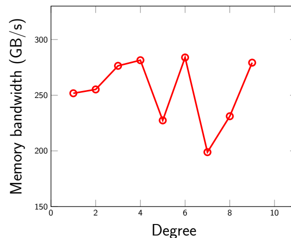
(d) Time Scheme

Figure: Arithmetic throughput

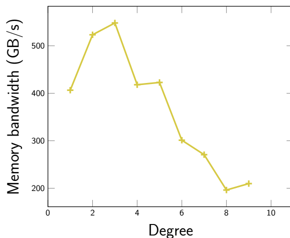
Optimization



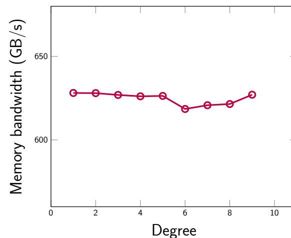
(a) Volum Terms



(b) Surface Terms



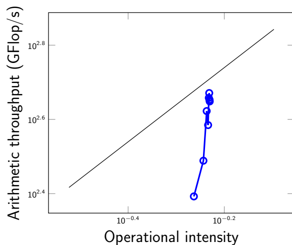
(c) Multiply



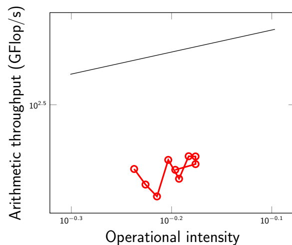
(d) RK

Figure: Memory bandwidth

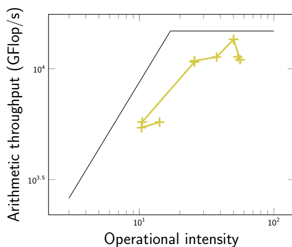
Optimization



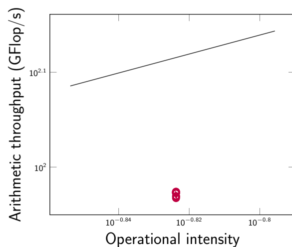
(a) Volum Terms



(b) Boundary Flux



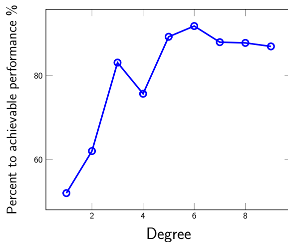
(c) Multiply



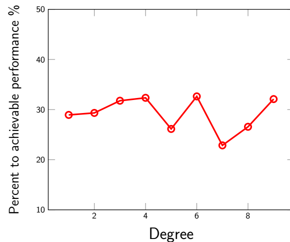
(d) Time Scheme

Figure: Arihmetic throughput as a function of the operational intensity

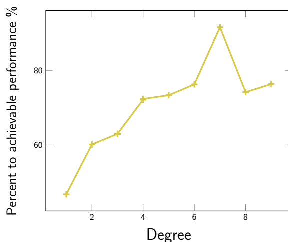
Optimization



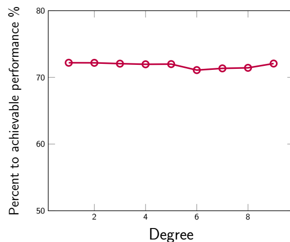
(a) Volum Terms



(b) Surface Terms



(c) Multiply



(d) Time Scheme

Figure: Percentage to achievable performance.

Conclusion

- Implementation of DG method on GPU in C++ with Cuda
- Optimization and performance assessment
- Multi-GPU
- Coupling of DG methods

[1] *http://www.metz.supelec.fr/metz/personnel/vialle/course/PPS-5A-GPGPU/index.htm*

[2] *https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html*

[3] Modave, A., St-Cyr, A., Mulder, W. A., Warburton, T. (2015). *A nodal discontinuous Galerkin method for reverse-time migration on GPU clusters.* Geophysical Journal International, 203(2), 1419-1435.

[4] G. Gabard, *Discontinuous Galerkin method*, 2020.

Conclusion

- Implementation of DG method on GPU in C++ with Cuda
- Optimization and performance assessment
- Multi-GPU
- Coupling of DG methods

Thank you!

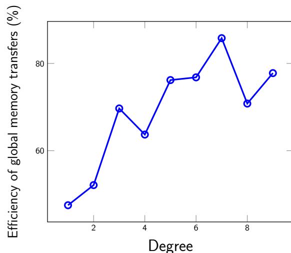
[1] *http://www.metz.supelec.fr/metz/personnel/vialle/course/PPS-5A-GPGPU/index.htm*

[2] *https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html*

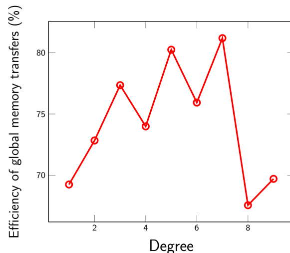
[3] Modave, A., St-Cyr, A., Mulder, W. A., Warburton, T. (2015). *A nodal discontinuous Galerkin method for reverse-time migration on GPU clusters.* Geophysical Journal International, 203(2), 1419-1435.

[4] G. Gabard, *Discontinuous Galerkin method*, 2020.

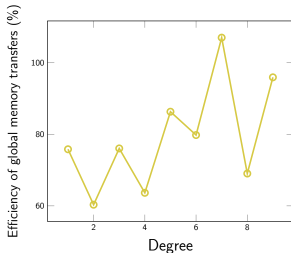
App. Global memory transfers



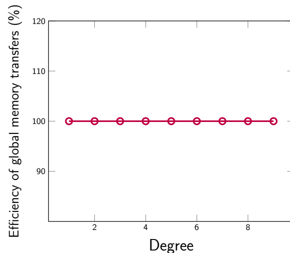
(a) Volum Terms



(b) Surface Terms



(c) Multiply



(d) Time Scheme

Figure: Efficiency of global memory transfer.