Centre de Thermique de Lyon
CETHIL UMR 5008

EDF R&D

Lip

Global Memory Access Modelling for Efficient Implementation of the Lattice Boltzmann Method on Graphics Processing Units

Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau, and Jean-Jacques Roux

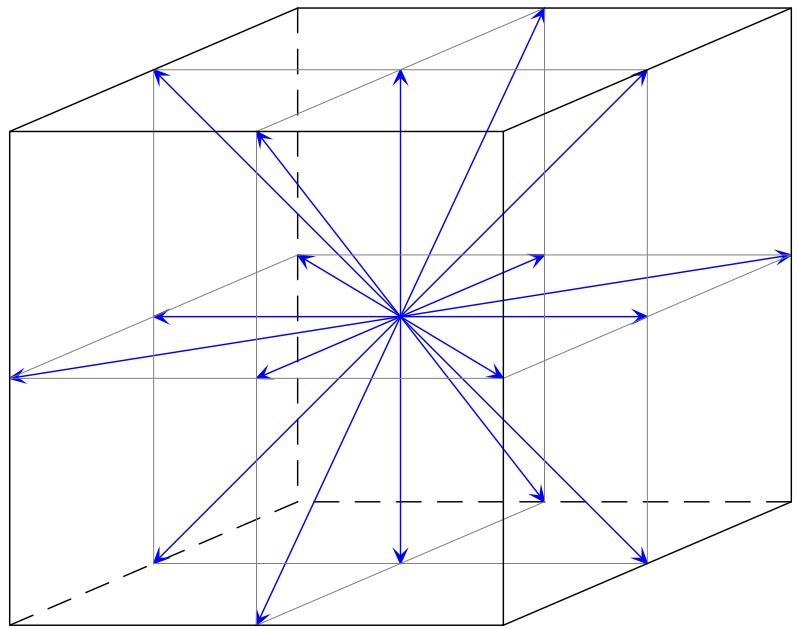
June 25, 2010

Motivations

- Due to computational cost, fluid dynamics is often neglected in building physics. Modelling of energy efficient buildings requires to take these effects into account.
- The lattice Boltzmann method (LBM) is an innovative approach in CFD. Besides other advantages, parallel implementations of LBM are rather straightforward.
- GPUs, e.g. CUDA capable hardware, provide an inexpensive and efficient way to perform parallel computations.
- The global memory maximum throughput is the limiting factor for CUDA implementations of the LBM, which currently achieve 70 to 80% of the maximum sustained throughput.

Lattice Boltzmann Method

Mass transfer is performed in discrete time and space using a finite set of velocities, as the D3Q19 stencil:



Lattice Boltzmann Equation

The fluid is represented by a discrete distribution f_i associated to the velocities $\mathbf{e}_i$, and obeying to the following equation:

$$f_i(\mathbf{x} + \delta t\, \mathbf{e}_i,\ t + \delta t) - f_i(\mathbf{x},\ t) = \Omega_i(f(\mathbf{x},\ t))$$

where Ω_i is a collision operator.

The macroscopic quantities are given by:

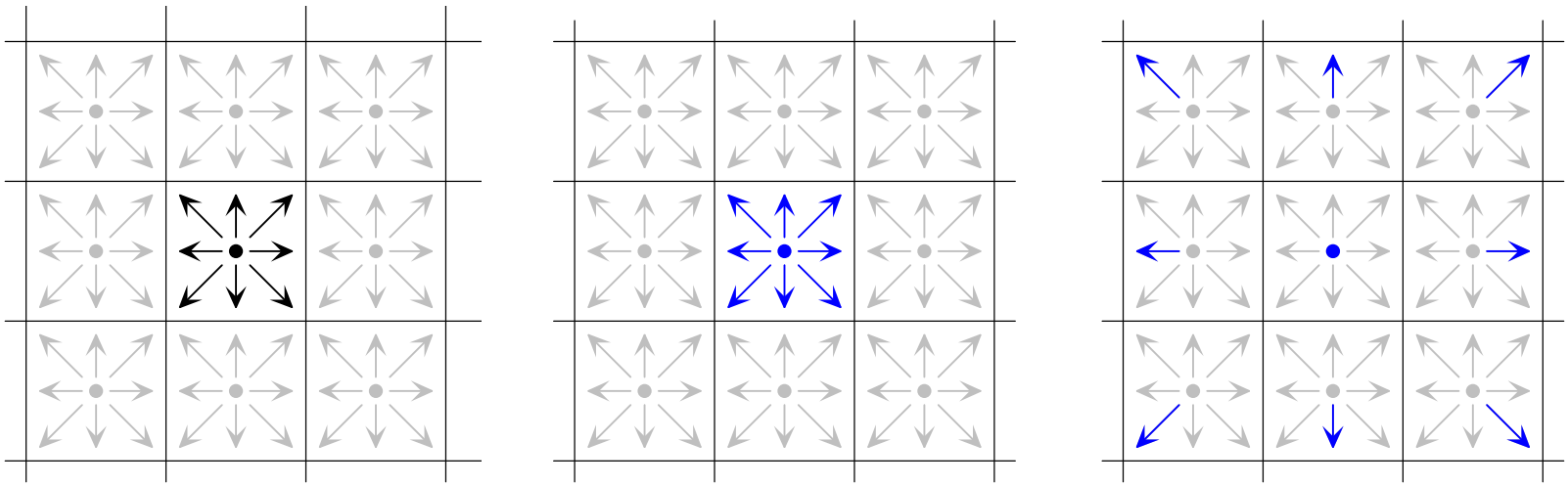
$$\rho = \sum_i f_i \qquad\qquad \mathbf{u} = \frac{1}{\rho} \sum_i f_i \mathbf{e}_i$$

Algorithmic Aspect

The LBM breaks up in two elementary steps, i.e. collision and propagation:

$$\tilde{f}_i(\mathbf{x},\ t) = f_i(\mathbf{x},\ t) + \Omega_i(f(\mathbf{x},\ t))$$

$$f_i(\mathbf{x} + \delta t\, \mathbf{e}_i,\ t + \delta t) = \tilde{f}_i(\mathbf{x},\ t)$$



Algorithm

for each time step t **do**
  for each lattice node **x** **do**
    read velocity distribution $f_i(\mathbf{x}, t)$
    if node **x** is on boundaries **then**
      apply boundary conditions
    end if
    compute updated distribution $\tilde{f}_i(\mathbf{x}, t)$
    propagate to neighbouring nodes $\mathbf{x} + \delta t \mathbf{e}_i$
  end for
end for

CUDA implementations of the LBM assign one thread to each node, store the velocity distribution in global memory, and ensure global synchronisation using one kernel launch for each time step.

Methodology

To study transactions between global memory and registers, we used kernels performing the following operations :

1. Store time t_0 in a register.
2. Read N words from global memory, with possibly L misalignments.
3. Store time t_1 in a register.
4. Write N words to global memory, with possibly M misalignments.
5. Store time t_2 in a register.
6. Write t_2 to global memory.

Time is accurately determined using the CUDA `clock()` function which uses per TPC counters.

Methodology (continued)

- The parameters of our measurements are N, L, M, and k, the number of warps concurrently affected to each SM.
- We used a one-dimensional grid and one-dimensional blocks containing one single warp.
- We created a script generating the kernels rather than using runtime parameters and loops.

Example of Kernel

```
__global__ void kernel(uint* t)
{
    int i = blockIdx.x;
    int j = threadIdx.x;
    uint t0 = clock();
    uint r0 = t[_(i, j, 0)];
    uint r1 = t[_(i, j, 1)];
    uint r2 = t[_(i, j, 2)];
    uint t1 = clock();
    t[_(i, j, 0)] = r0 + t0;
    t[_(i, j, 1)] = r1 + t1;
    t[_(i, j, 2)] = r2 + t1;
    uint t2 = clock();
    t[_(i, j, T)] = r2 + t2;
}
```

Measurements

We carried out our measurements on one CUDA device of a GeForce GTX 295 graphics board (featuring two GT200).

- At kernel launch, blocks are dispatched to the TPCs one by one up to k blocks per SM.
- Since the GT200 contains ten TPCs, blocks affected to the same TPC have identical `blockIdx.x` unit digit.
- In order to compare the measurements, we shifted the origin of the time scale to the minimal t_0.

We noticed that the timings are coherent on each of the TPCs.

Modelling, $N \leq 20$

For $N \leq 20$, we observed that:

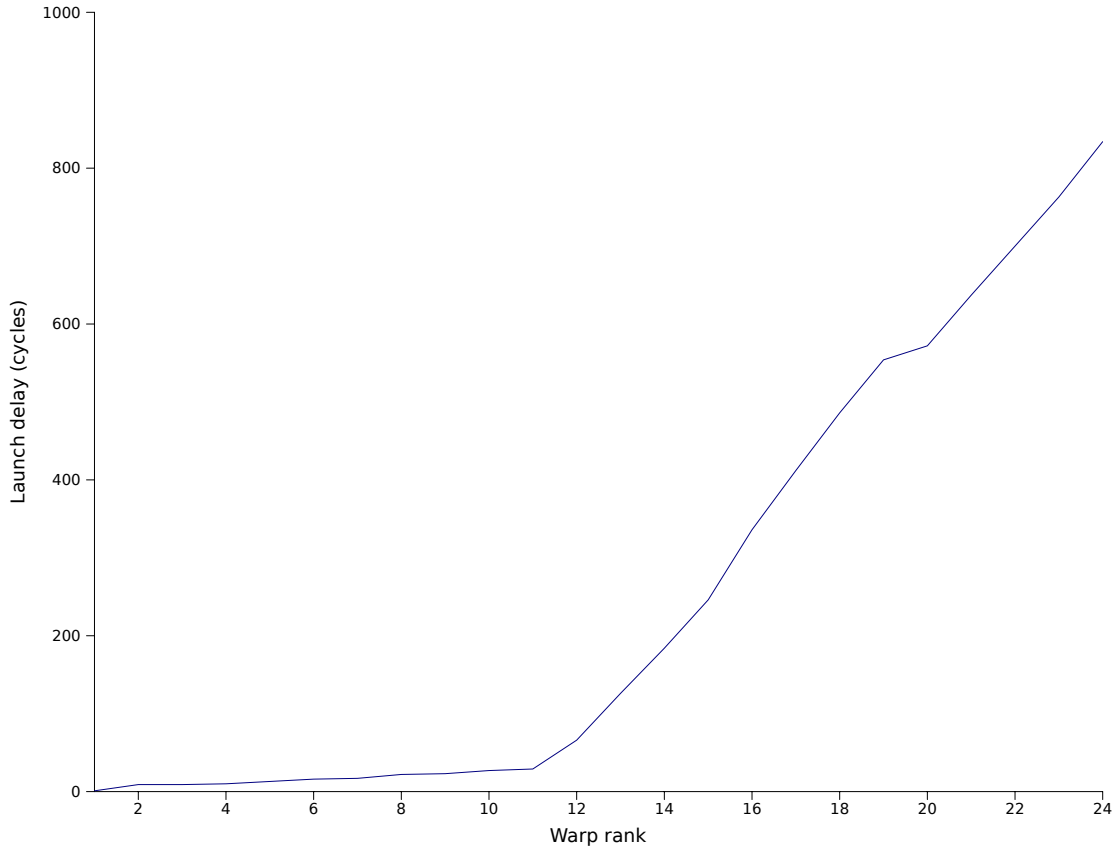
- Reads and writes are performed in one stage, hence storing of t_2 has no noticeable influence.
- Warps 0 to 8 are launched at once (in a determined but apparently incoherent order).
- Subsequent warps are launched one after the other every ~ 63 clock cycles.

Thus we approximate the processing time T of k warps per SM with:

$$T \approx \ell + T_R + T_W$$

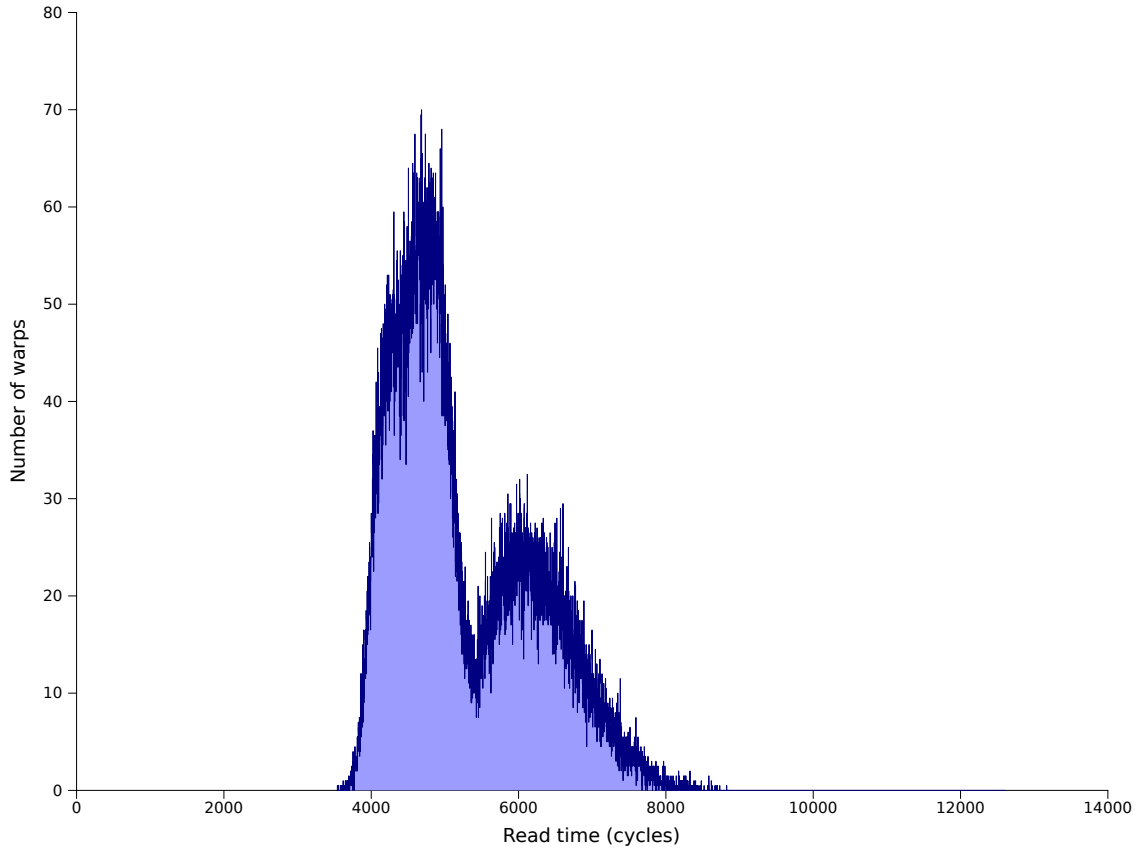
ℓ being start time of the last warp, T_R read time, T_W write time.

Launch Delay



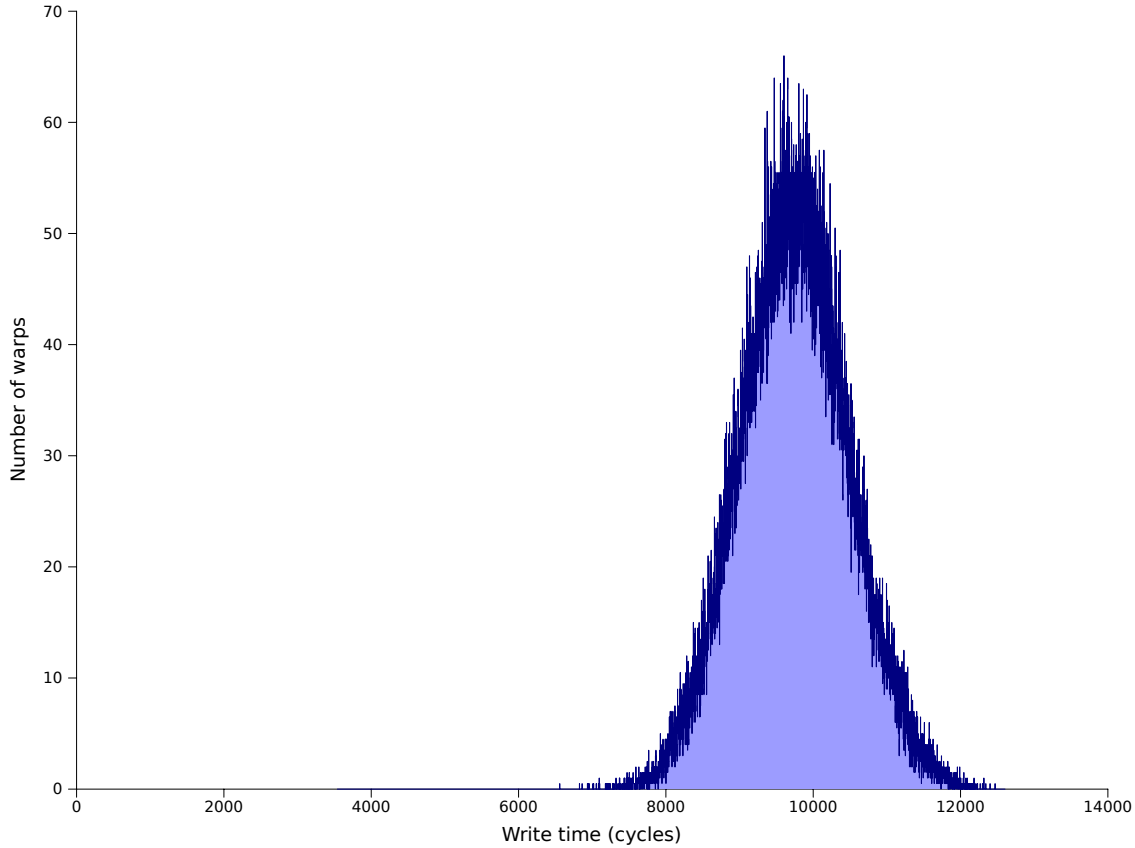
$\ell(i) \approx 0$ for $i \leq 9$ and $\ell(i) \approx 63(i - 10) + 13$ otherwise.

Read time distribution (96,000 warps with $N = 19$)



Bimodal shape of the distribution is most likely due to TLB misses.

Write time distribution



Average read and write times depend linearly of N. For instance with, $k = 8$, we obtained:

$$T_R \approx 317(N-4) + 440 \qquad T_W \approx 562(N-4) + 1178$$

Modelling, $N > 20$

For $N > 20$, reads and writes are performed in two stages.

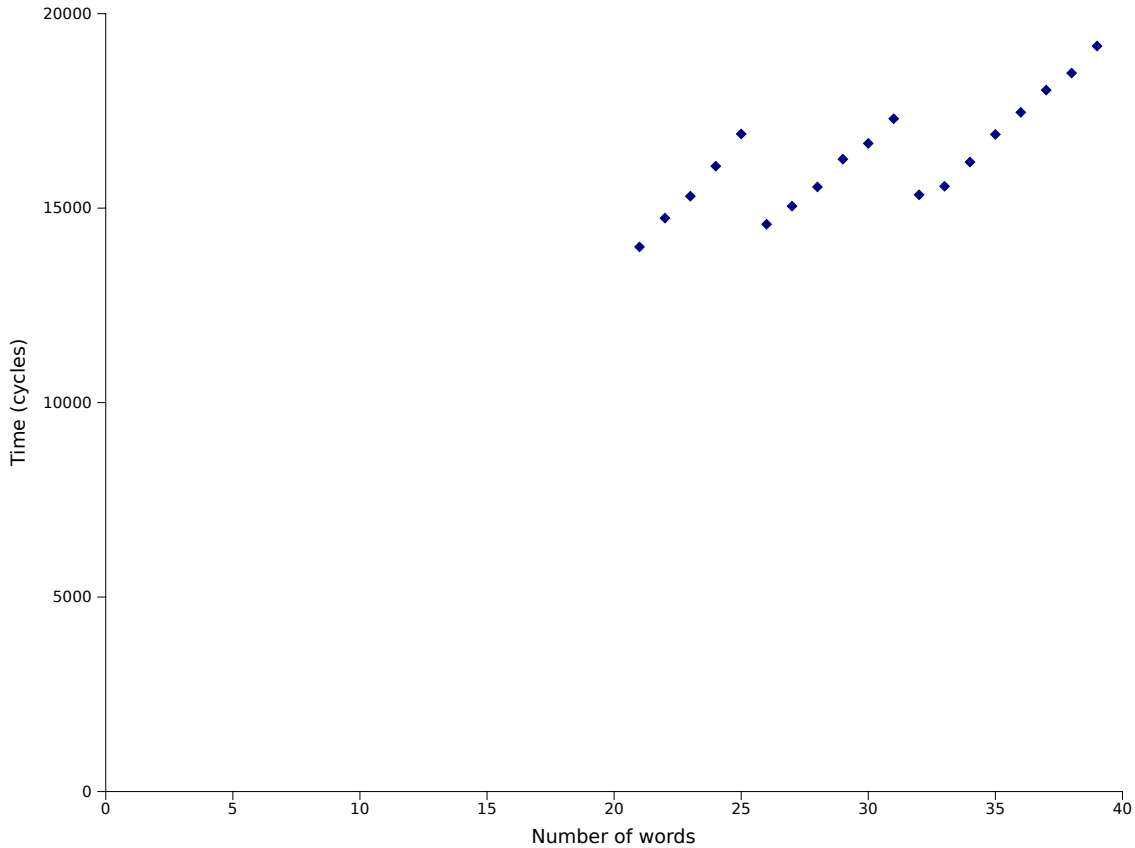
- If the first n warps in a SM read at least 4,096 words, where $n \in \{4, 5, 6\}$, then the processing of the subsequent warps is postponed.
- Time t_0 for the first $3n$ warps of a TPC follow the same pattern as in the first case.
- There is a slight overlapping of the two stages but we may consider that they are performed sequentialy.

Again, we can approximate the processing time T of k warps per SM with:

$$T \approx T_0 + \ell' + T'_R + T'_W$$

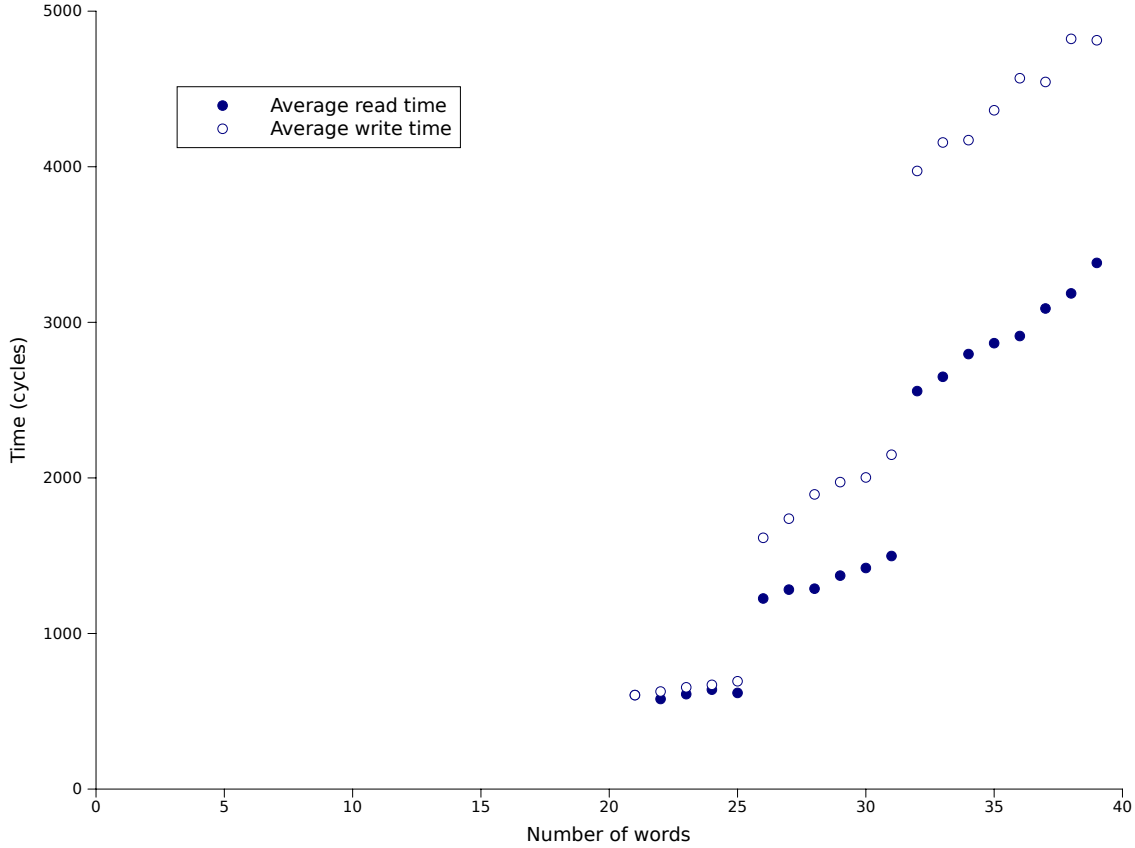
T_0 being the processing time for the first stage.

First stage duration



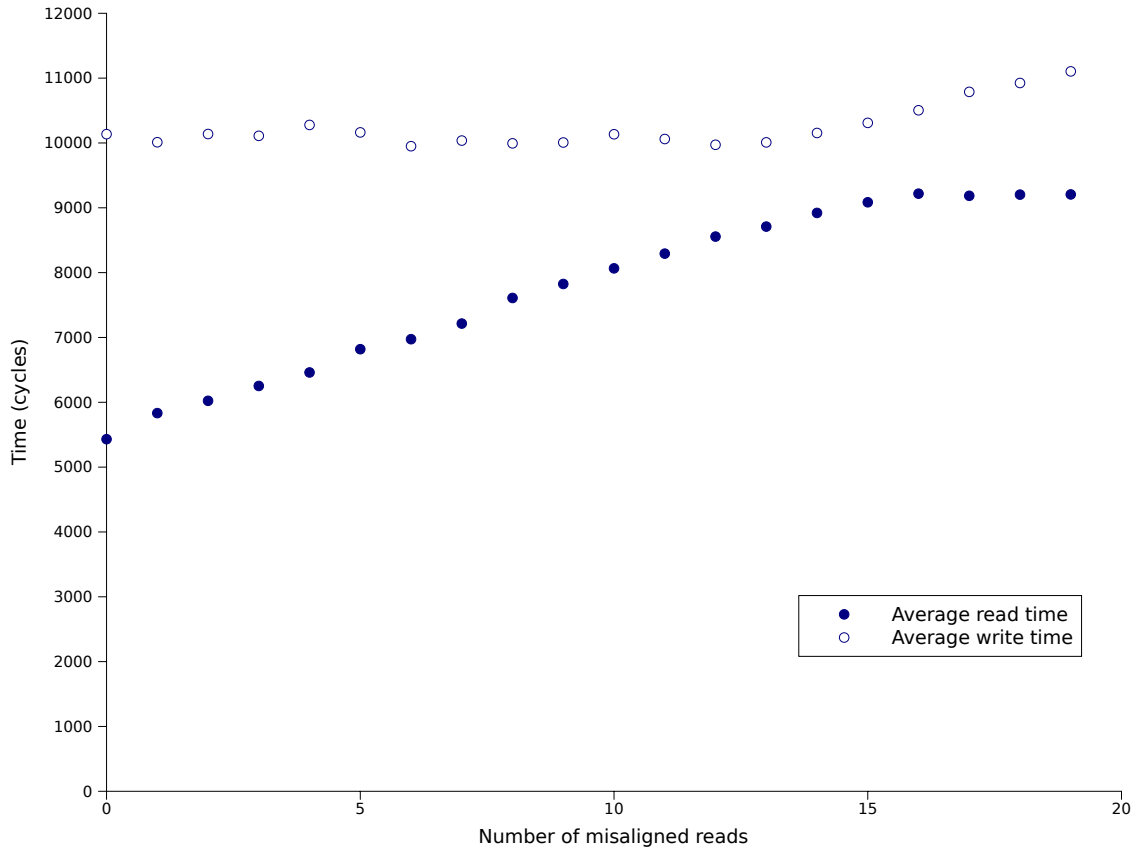
T_0 depends linearly of N in the three intervals $\{21, \ldots 25\}$, $\{26, \ldots 32\}$, and $\{33, \ldots 39\}$.

Timings in second stage



T'_R and T'_W also depend linearly of N in the mentionned intervals.

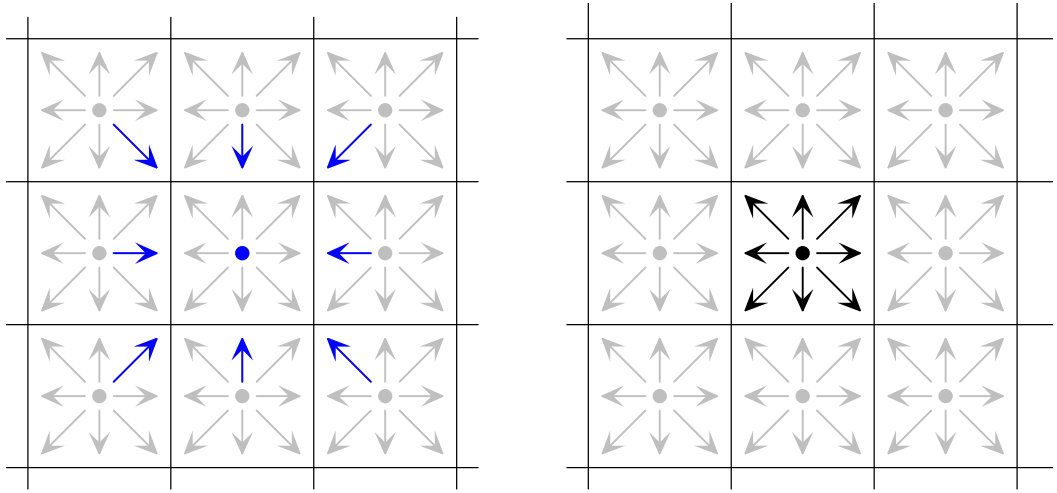
Misalignment impact



For misaligned reads, the average write time remains approximatively constant. Read time increases linearly until some threshold is reached. Similar conclusion can be drawn for misaligned writes.

Implementation

For each implementation, we used a SoA like data layout, and a two-dimensional grid of one-dimensional blocks. Since misaligned writes are more expensive than misaligned reads, we experimented several propagation schemes in which misalignments are deferred to the read phase of the next time step.



Performance and Estimation (MLUPS)

Model	Occupancy	Actual	Estimated	Rel. error
D3Q19 LBGK	25%	481	492	2.3%
D3Q19 MRT	25%	516	492	4.6%
Thermal LBM	12.5%	195	196	1.0%

Summary

Our work provides:

- An extensive study of the global memory access mechanism between global memory and GPU for the GT200.
- A description of the scheduling of global memory accesses at hardware level.
- A model which allows to estimate the global execution time of a regular data-parallel application on GPU. This model is likely to apply to similar GPU applications.

Thank you for your attention!