Evaluation of a Runtime Environment for a Discontinuous Galerkin Solver for General Hyperbolic Systems

Malcolm Roberts[*,1], Bruno Weber[1,2], Philippe Helluy[1], Emmanuel Franck[1]

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

HPC Days Lyon, 2016-05-06

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

Outline

- The discontinuous Galerkin method and `OpenCL`.
  - `schnaps`: `schnaps.gforge.inria.fr`
- Benchmarking the code on `CPU`, `GPU`, and comparing performance.
- Runtime environments:
  - StarPU-SOCL
  - StarPU
- Analysis of results.

Discontinuous Galerkin Method

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:

- Maxwell's equations
- MHD
- Vlasov equations

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

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

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

OpenCL

The DG formulation is good for conserving invariants and has other nice properties, but it is computationally expensive.

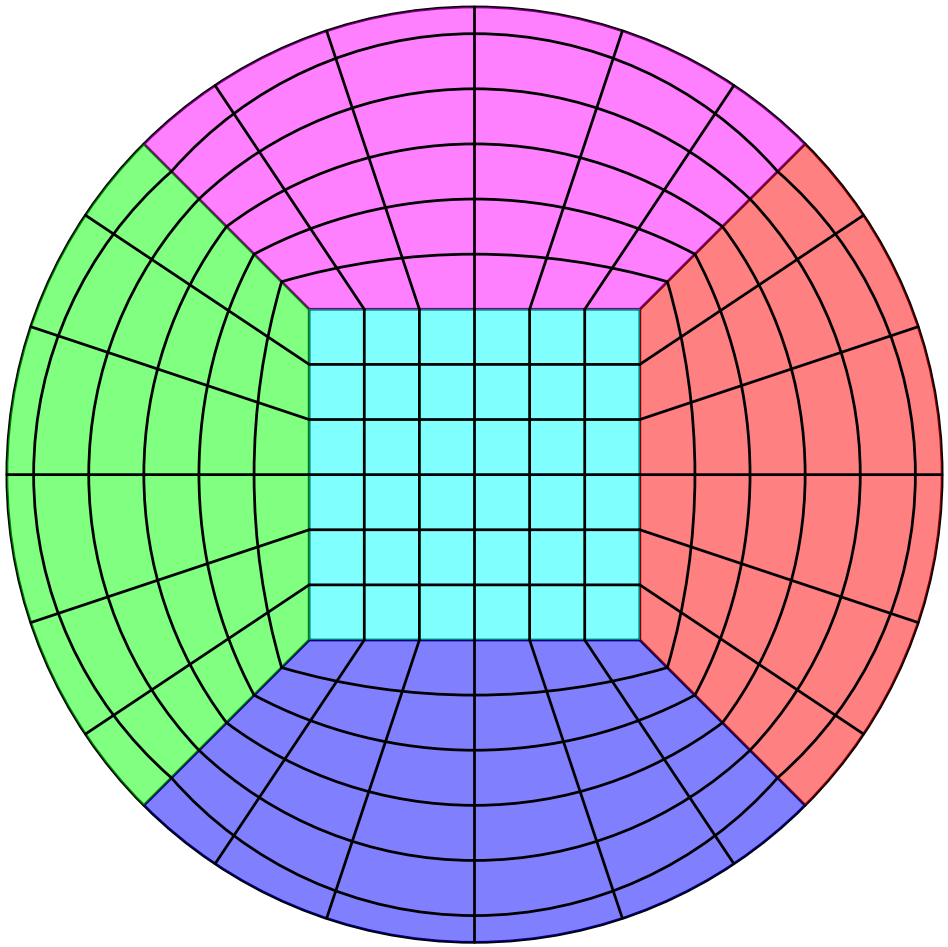
Using `OpencL` to program for `GPUs` can reduce this cost:

- Fast coalescent memory access.
- Events to control program flow.
- We can run on `GPUs`, but also `CPUs` and `MICs`.
- A string composed of `C`-like code is sent compiled for the device at run-time.

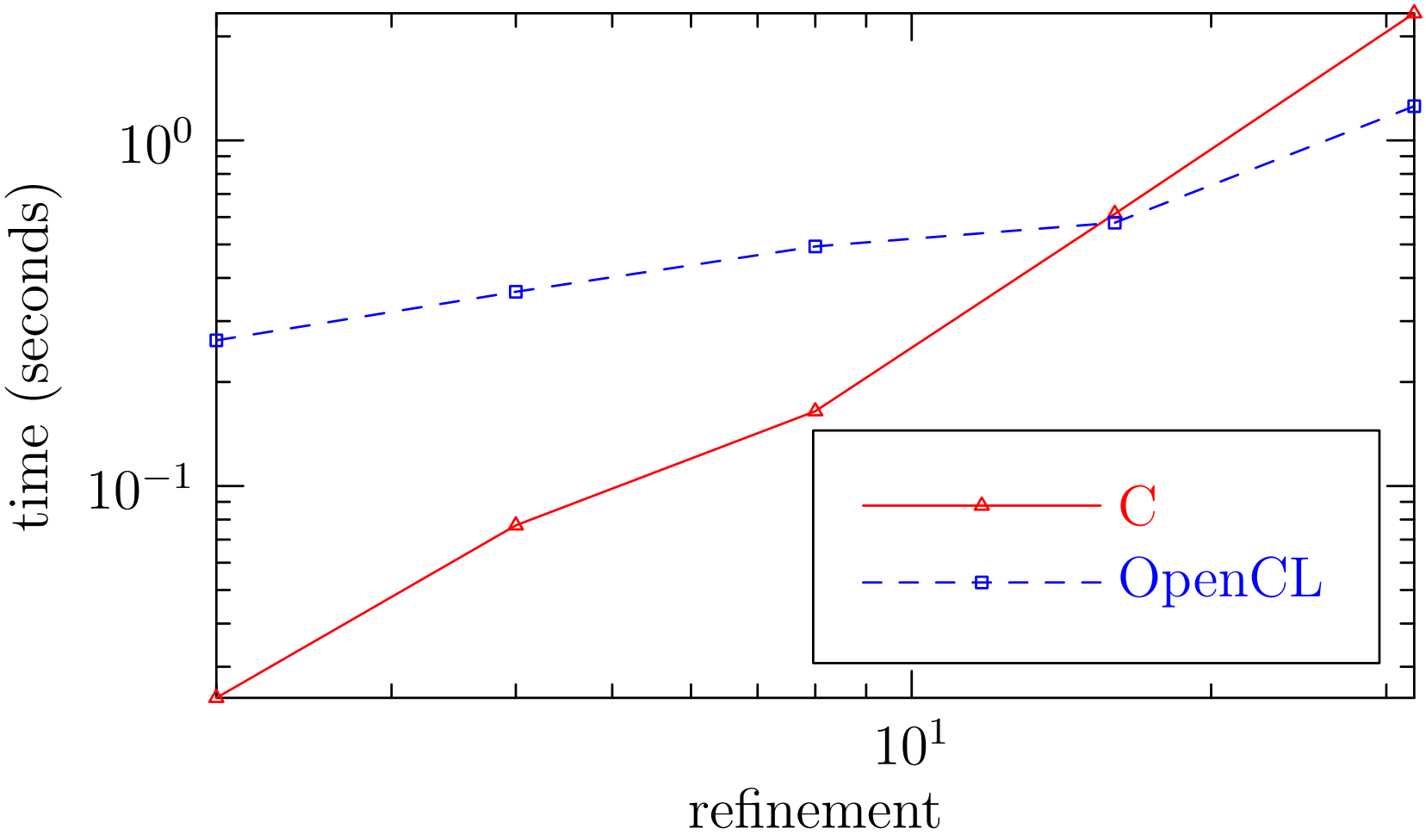
We use hexahedral elements to increase coalescence.

A macrocell/subcell structure further improves memory access.

Array of structs of arrays

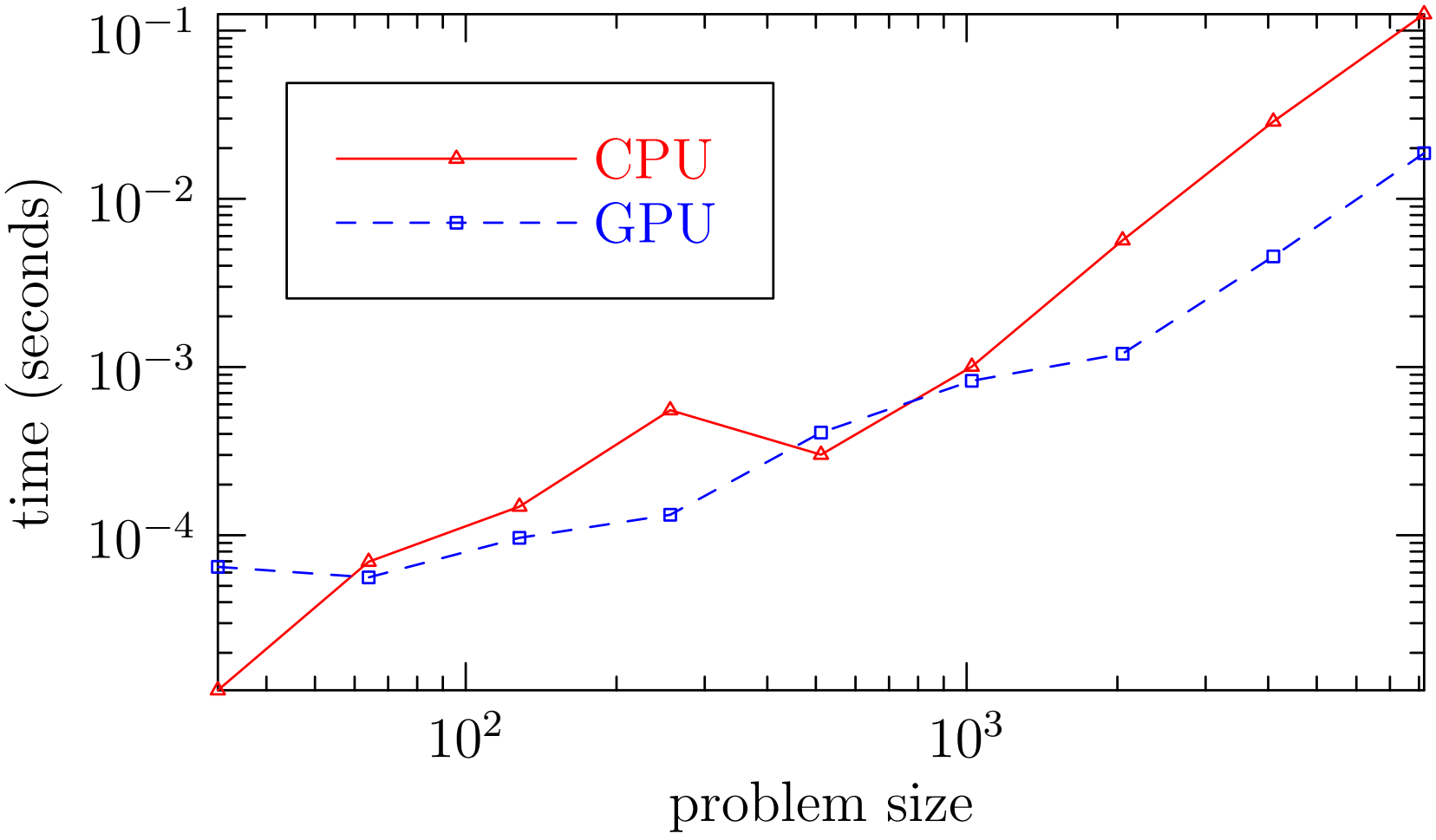


Performance analysis of OpenCL implementation

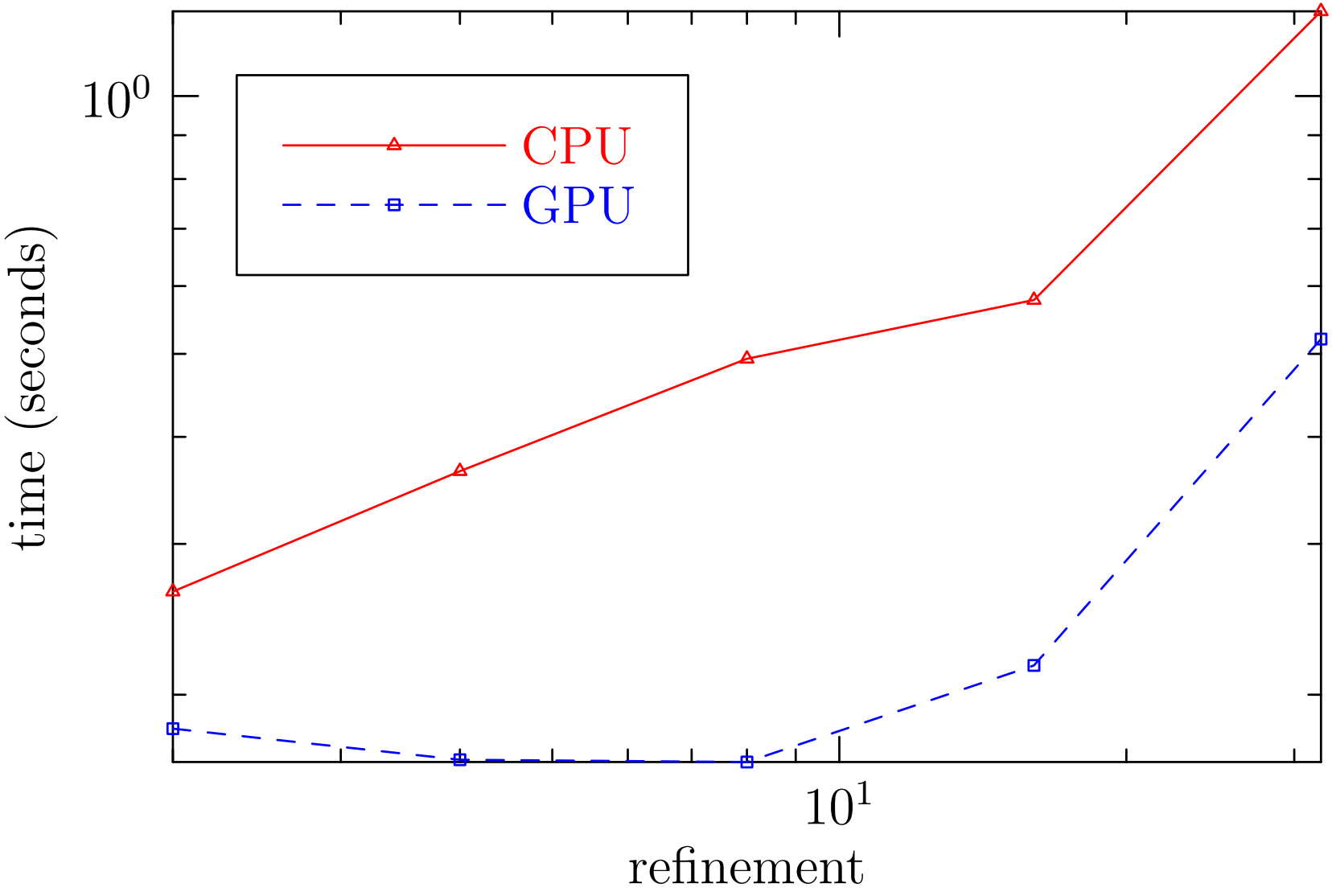


Performance analysis of OpenCL implementation

`clFFT`, an FFT library written in `OpenCL` by AMD.



Performance analysis of OpenCL implementation



Performance analysis of OpenCL implementation

We can conclude that:

1. The `C` code makes use of all the cores. Note that the `C` code is not vectorized.
2. The `C` and `OpenCL` code speeds on the `CPU` are close for large problem sizes.
3. The performance difference of `schnaps` between the `CPU` and `GPU` is near what we should expect.
4. Thus, we claim that our code makes effective use of the `GPU`.

We can further improve the code by profiling and improving the costly steps.

Run-time environments

Should we run on the `CPU` or the `GPU`?

Why not both?

Also, how can we write code that suits all of these platforms?

Solution: run-time environments.

- Each task is associated with one or more codelets:
  - Test performance and then use the fastest!
  - Examples: FFTW, Atlas.
- Distribute tasks to devices.
- Synchronize task execution and manage memory transfers.

In particular, we look at `StarPU`.

Run-time environments: StarPU-SOCL

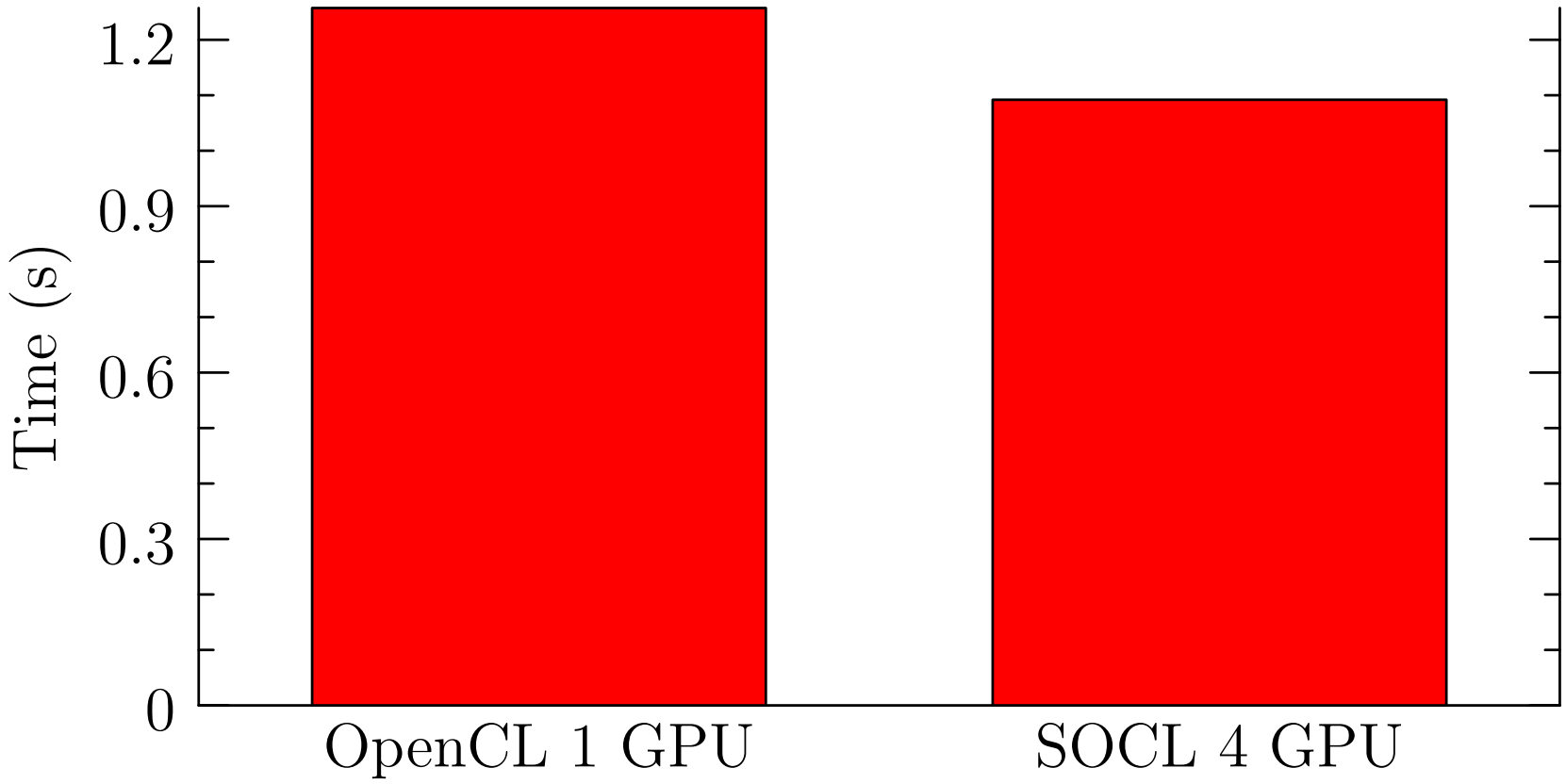
SOCL: StarPU On OpenCL.

The idea is that StarPU creates a fictitious OpenCL device.

- The device encompasses the CPU, GPUs, etc.
- SOCL manages kernel compilation for all the device.
- SOCL distributes tasks and manages memory.
- Little modification is required to an existing OpenCL 1.0 program.

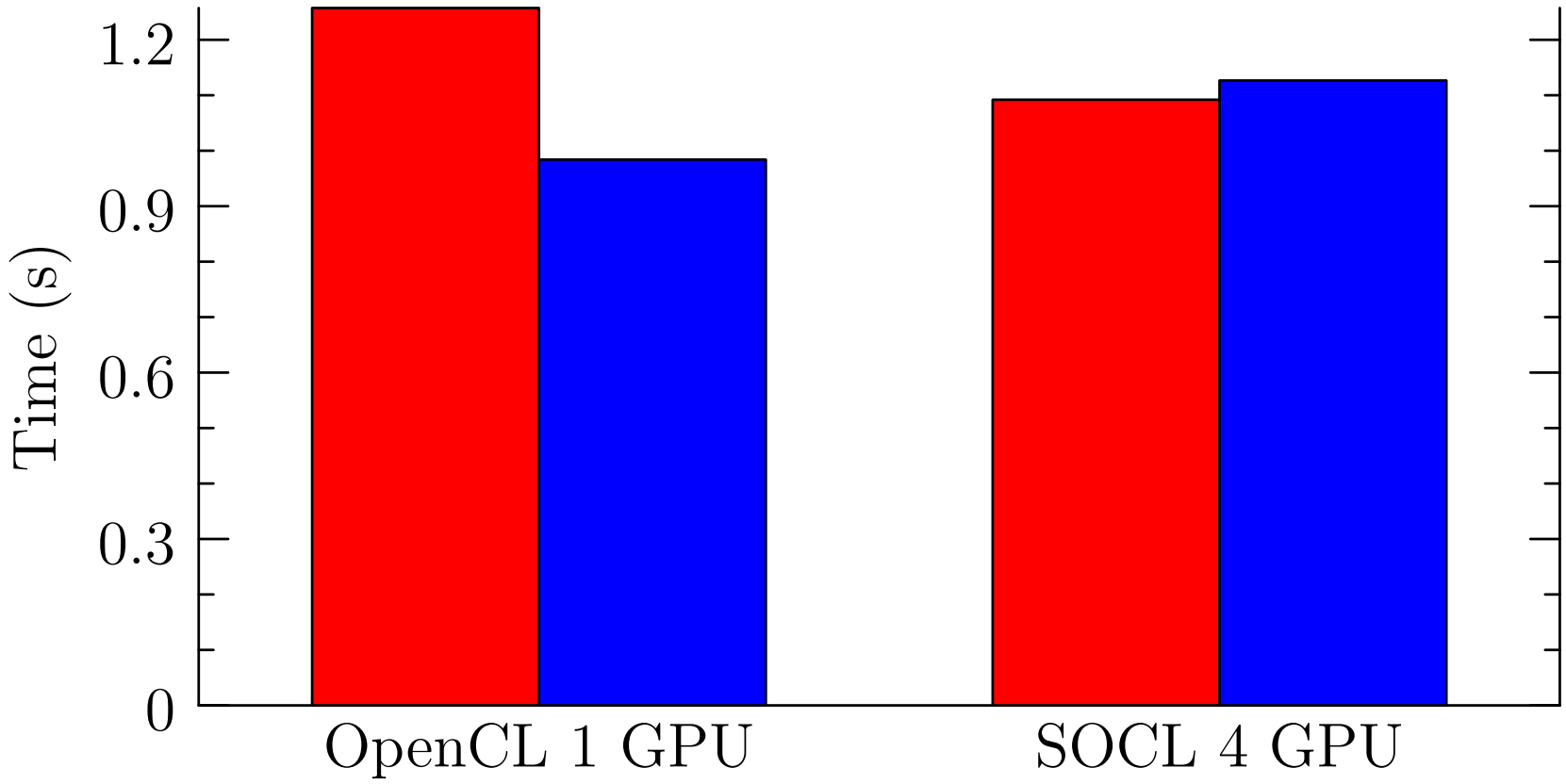
SOCL performance, without extraction

Comparison of OpenCL (1 $\times$ K80) and SOCL (4 $\times$ K80) performance, without interface extraction.



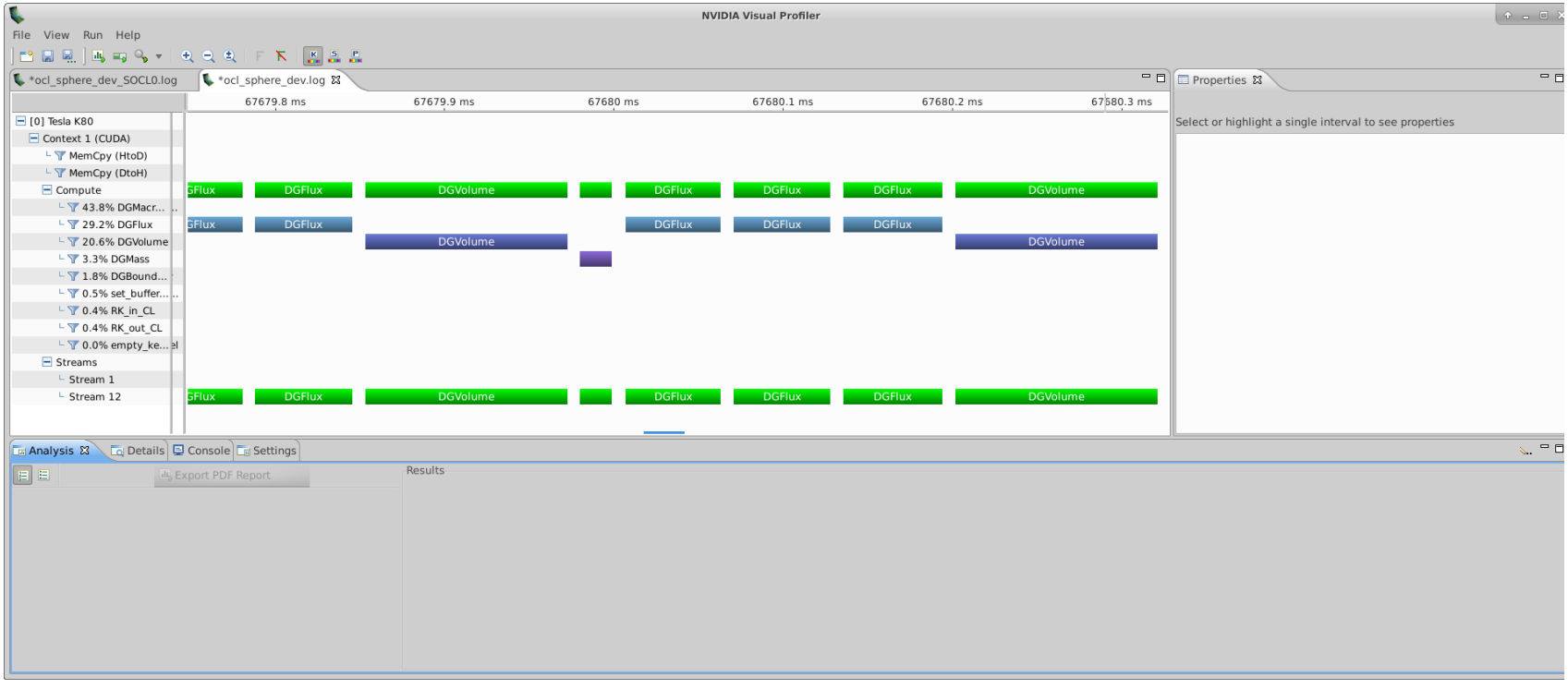
SOCL performance, with extraction

Comparison of OpenCL (1 × K80) and SOCL (4 × K80) performance with interface extraction.



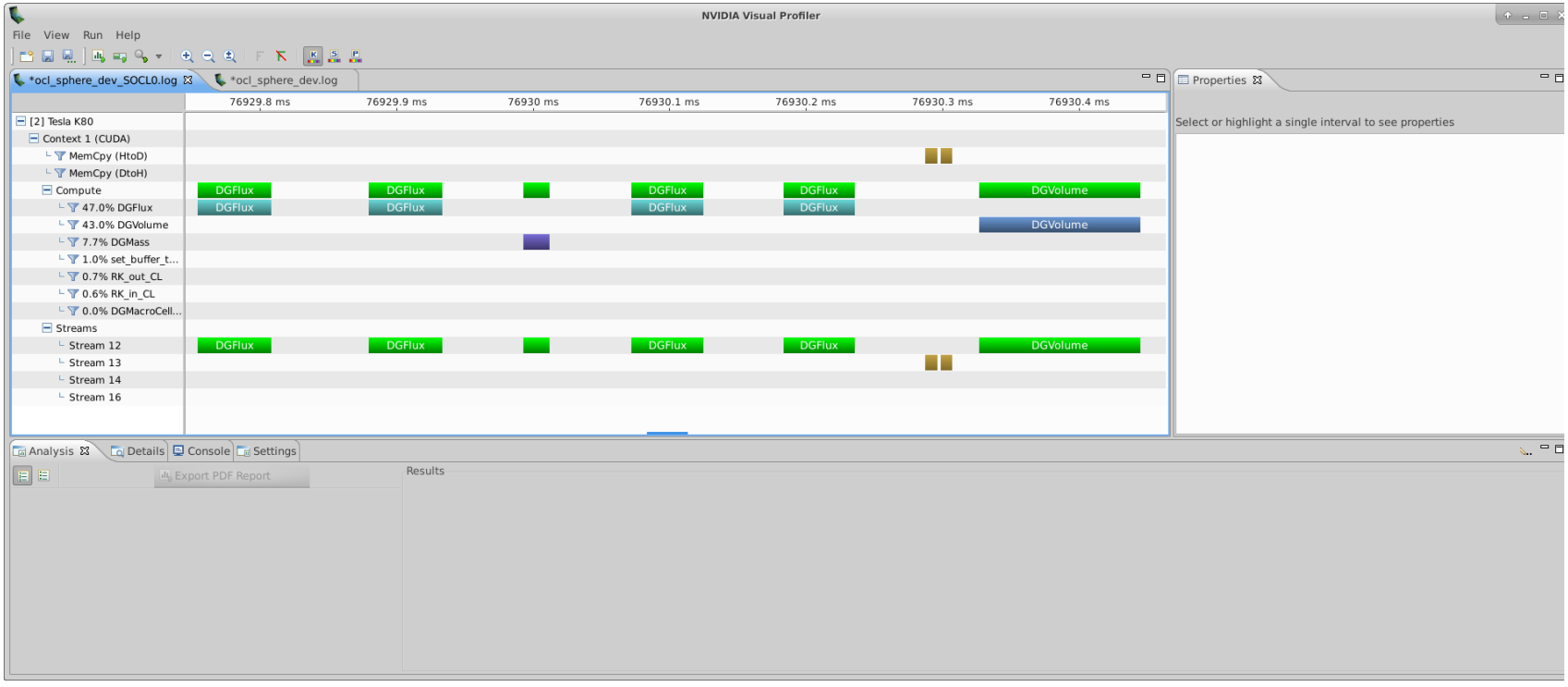
Run-time environments: StarPU-SOCL

NVidia Visual Profiler (nvvp) for `OpenCL`:



Run-time environments: StarPU-SOCL

NVidia Visual Profiler (nvvp) for `SOCL`:



SOCL performance

Analysis:

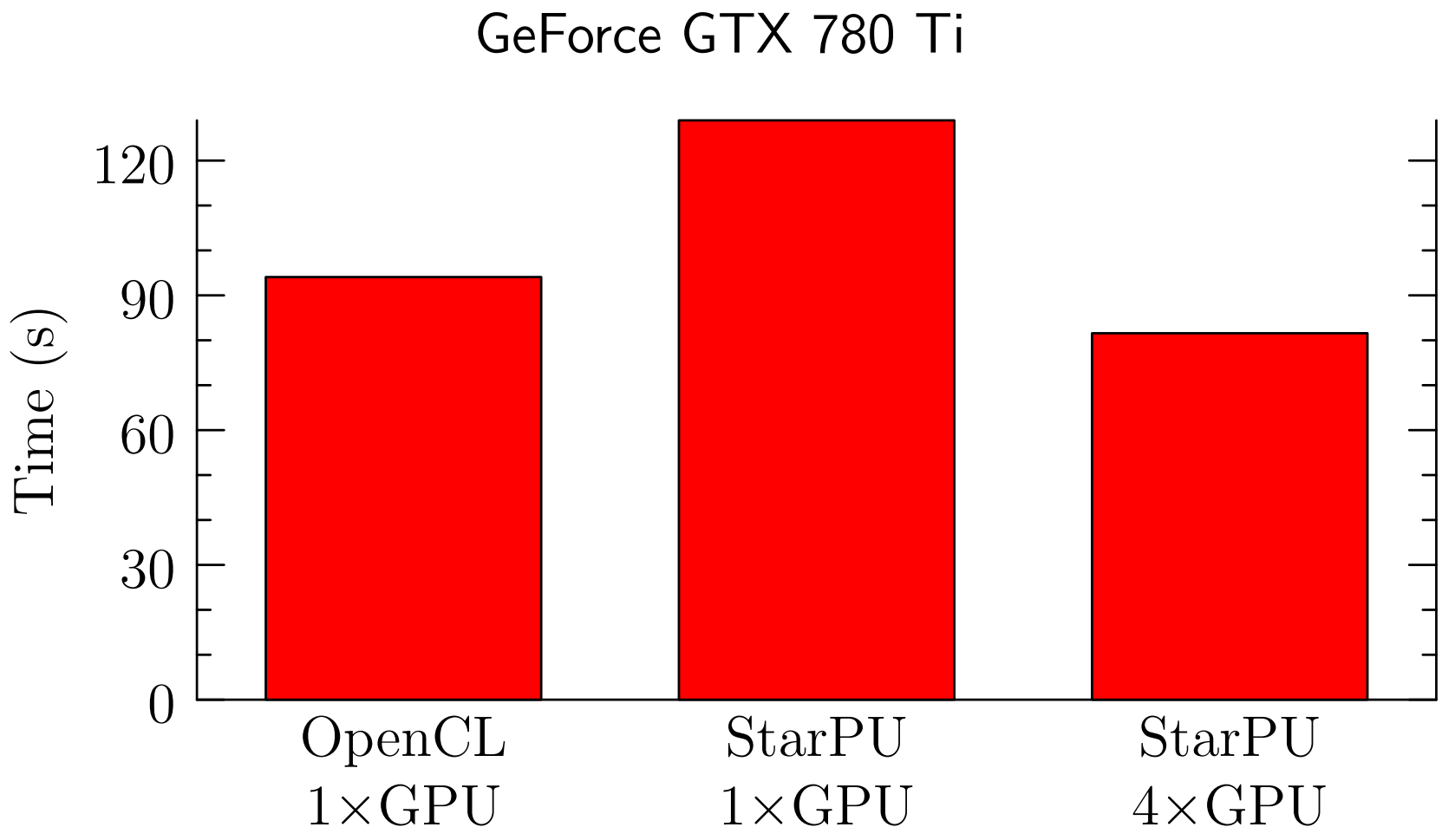
1. Kernel launch latency is an issue.

   Possible solutions:

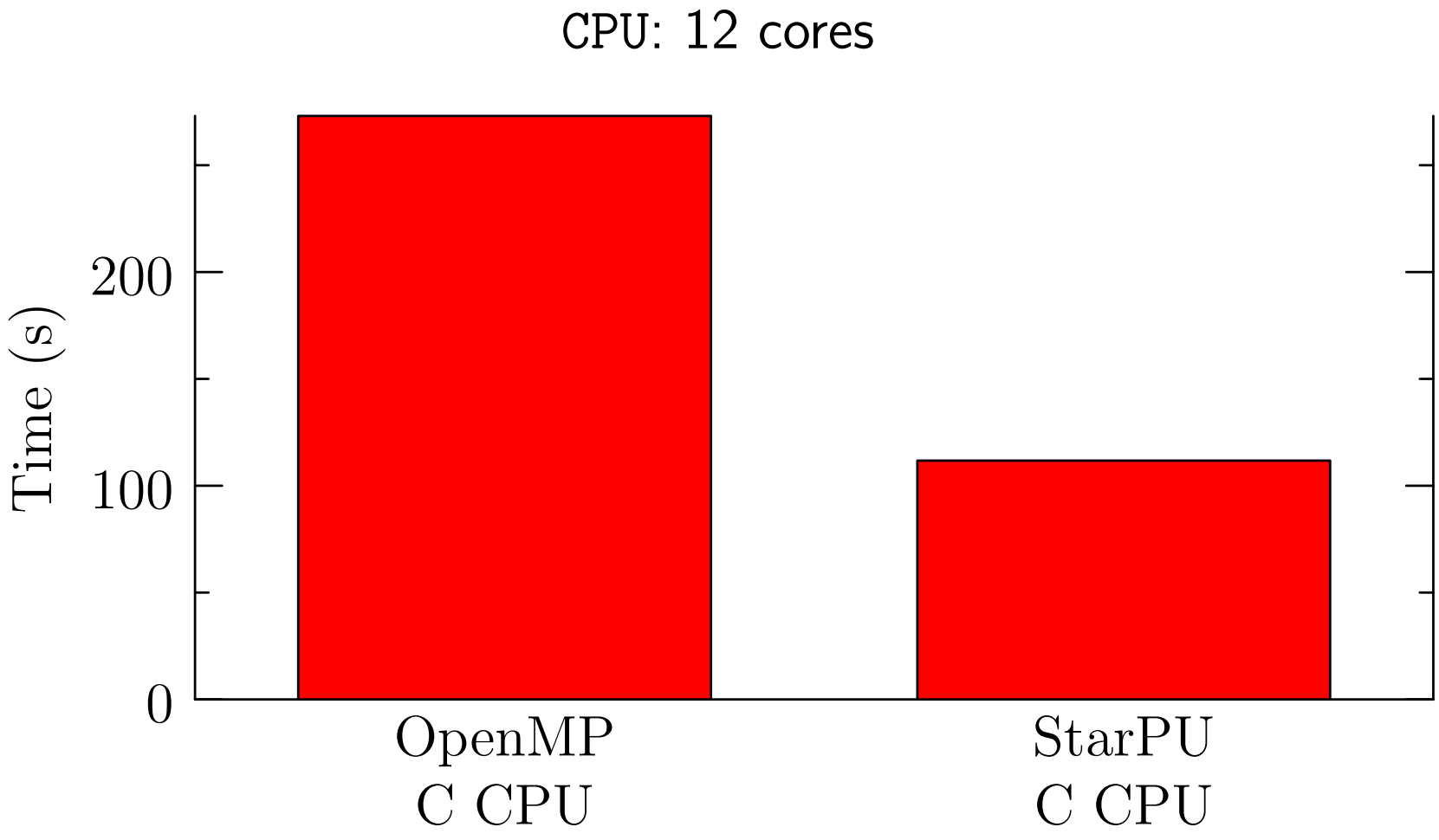
   - Reduce length of event wait list.
   - Move to an on-device queue, as in `OpenCL` 2.0.

2. There is also a large amount of memory transfer, whose origin is unknown.

StarPU


GeForce GTX 780 Ti
Time (s)
120
90
60
30
0
OpenCL
1×GPU
StarPU
1×GPU
StarPU
4×GPU

StarPU



StarPU

Analysis:

- We were not able to get good performance with StarPU-SOCL or StarPU using multiple GPUs.
- StarPU was able to parallelize a small grid which our OpenMP-C implementation didn't handle well.

For GPU performance, there are a variety of possible solutions to consider:

- Is there an issue with OpenCL events?
- Is CUDA a better choice for StarPU?
- Can we group tasks to reduce overhead? (Would the reduction in granularity be a problem?)

Conclusions

- Presented schnaps, a C/OpenCL implementation of the discontinuous Galerkin method.
  - schnaps.gforge.inria.fr
- Our implementation is efficient on CPU and GPUs.
- Presented initial results using the StarPU runtime environment.
- Using multiple GPUs with OpenCL codelets hasn't been effective so far.
- StarPU was effective at parallelising the C code on the CPU.
- We are looking at how we can improve multi-GPU performance with StarPU.

Thank you for your attention!