Aalto University
School of Science
and Technology

Astaroth: A GPU API for computations in structured grids

Johannes Pekkilä

Department of Information and Computer Science
Aalto University, School of Science and Technology
johannes.pekkila@aalto.fi

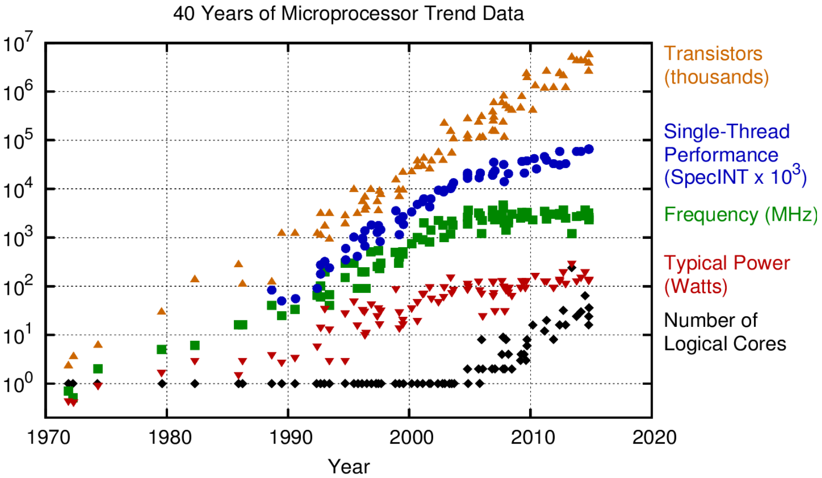
August 28, 2019

Problem

- We want to simulate magnetohydrodynamics
- Multiple interacting fields $\rightarrow$ difficult to optimize
- Must be able to turn equations on and off
- Someone may want to introduce new equations
- How can we get the best performance also in cases we did take into account?
- How can we optimize the program if we do not even know what the problem is?
- Common case: stencil computations in a structured grid

Parallel computing

- Single-processor performance stalled in the 2000s



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Parallel computing

- Single-processor performance stalled in the 2000s
  - Power wall: clock frequencies cannot be increased because of cooling becomes too expensive for mass-produced microprocessors

Parallel computing

- Single-processor performance stalled in the 2000s
    - Power wall: clock frequencies cannot be increased because of cooling becomes too expensive for mass-produced microprocessors
    - Memory wall: memory bandwidth increases at a slower rate than arithmetic performance
    - Instruction-level parallelism wall: branch-prediction is not perfect

Graphics processing units

- CPUs must perform relatively well in all tasks
- Memory systems of the CPU are optimized for low latency at the cost of throughput
- Jack of all trades but specialized architectures are generally faster

- GPUs are coprocessors specialized in data-parallel tasks
- High memory access latency but also high throughput
- Excellent if the problem can be decomposed into independent subproblems, but useless if the problem
  - must be solved sequentially (data dependencies) or
  - the program may fetch data from arbitrary memory locations

Parallel programming

- Many moving parts
- Requires understanding of the underlying architecture
- What if you do not care and just want to do your thing?

Astaroth API

- The implementation has been decoupled from the problem description
- We provide a high-level domain-specific language (DSL) for expressing physics
- We provide an optimizing source-to-source compiler for translating programs written in the DSL to efficient CUDA code
- The CUDA code is then embedded to a library that can be linked and used in other projects
- Astaroth is now essentially a GPU API for solving PDEs, much like what OpenGL and DirectX are for computer graphics

Astaroth API

- High-level general-purpose GPU APIs lack the expressivity to translate complex mathematical problems into efficient code
- Inherent tradeoff: simpler language - less room for optimization

Astaroth API

- High-level general-purpose GPU APIs lack the expressivity to translate complex mathematical problems into efficient code
- Inherent tradeoff: simpler language - less room for optimization
- Unless you focus on some specific problem domain!

Domain-specific languages

- Domain-specific languages are tailored for solving a subset of problems
- Because of that, they can provide both a high-level language and be translated into efficient code
- Examples:
  - Graphics shading languages (GLSL, HLSL)
  - Image processing languages (Halide, PolyMage)
  - Intermediate languages to build DSLs upon (Delite, Lift)
  - Astaroth :-)

Results

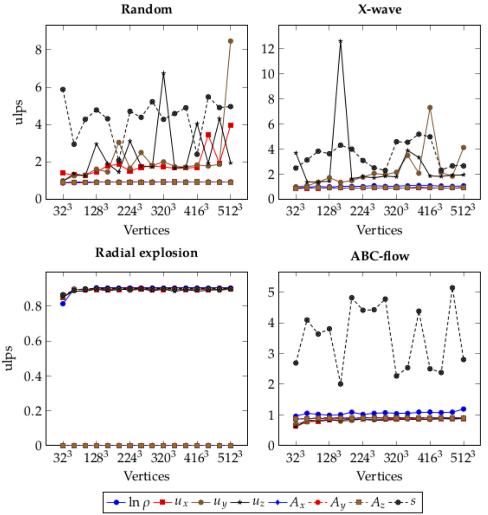


Figure 5.5: The maximum arithmetic error in terms of units in the last place for each field after a complete integration step using 64-bit precision. A single ulp was equal to $\beta^{e-(p-1)}$ as shown in Equation 5.2. The error in ulps at the point of the maximum absolute error of the field is shown.

Results

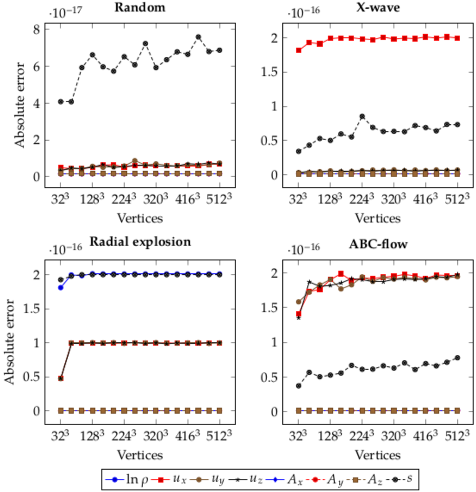
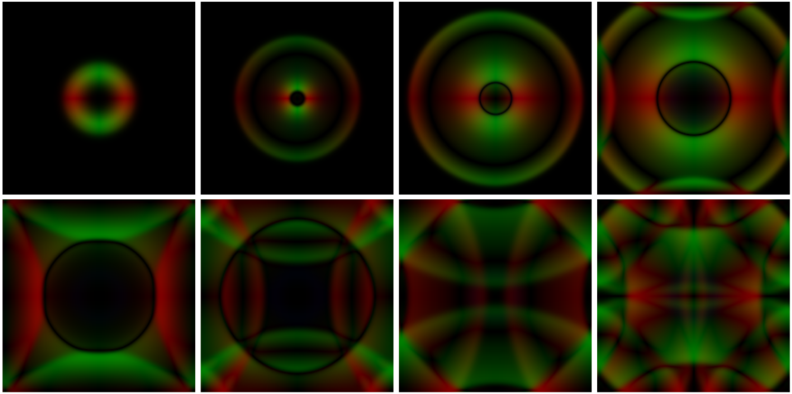
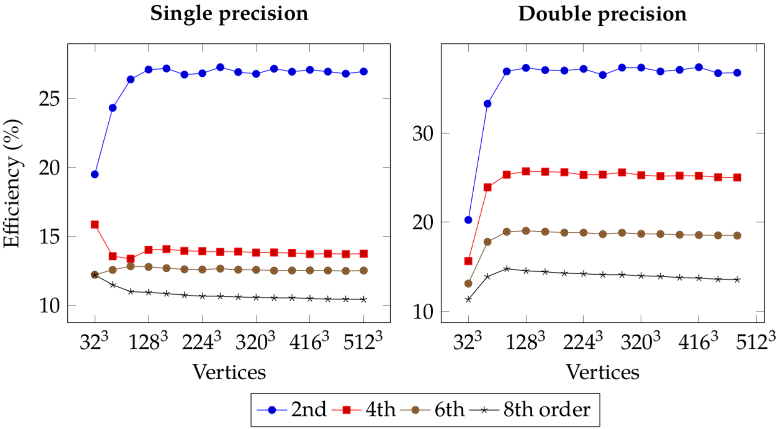


Figure 5.7: The absolute error of an integration step when comparing the output computed with 64-bit precision to a model solution computed with 80-bit precision.

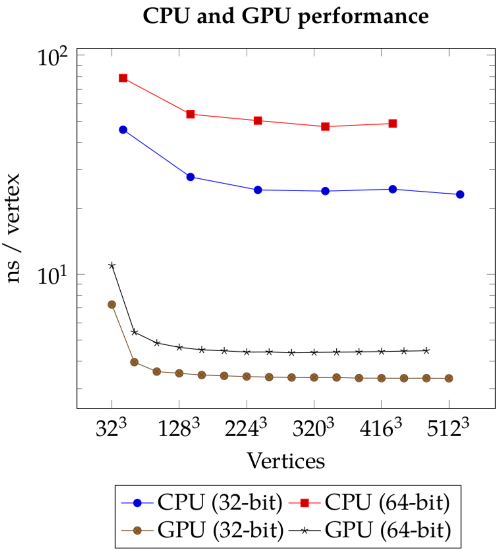
Results



Hardware utilization



CPU vs GPU (6th order FD)



Conclusion

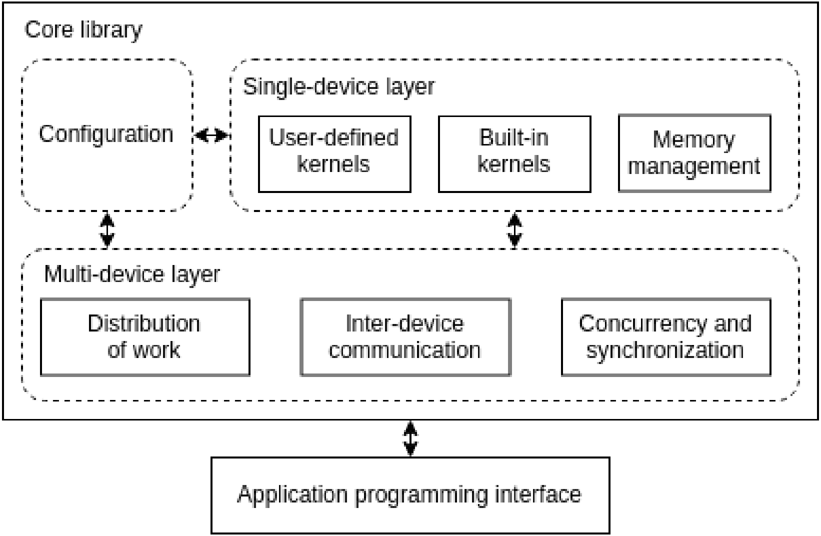
- Astaroth is no longer a proof-of-concept MHD solver, but rather a multi-GPU API for computations in structured grids
- We still do MHD though: Currently supported with the DSL
  - Hydro
  - Magnetic
  - Entropy
  - Helical forcing
  - Upwinding
  - Non-uniform grids (WIP, Miikka)
- Double precision: $10\times$ speedup with a single P100 over Pencil Code run on 24 CPU cores on Taito
- Single-GPU performance is very close to the practical hardware limits (bound by cache bandwidth)
- Near perfect scaling to multiple GPUs within a node.

Conclusion

- Technical details: `http://urn.fi/URN:NBN:fi:aalto-201906233993`
- Latest version: `https://bitbucket.org/jpekkila/astaroth/src/master/`

BONUS

Library architecture

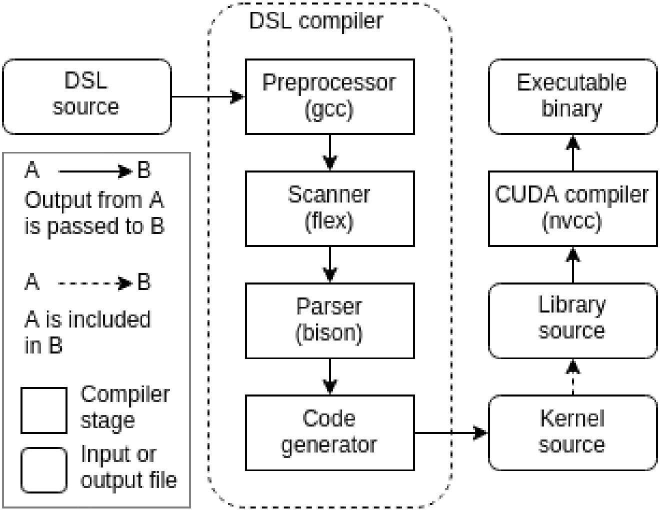


Astaroth domain-specific language

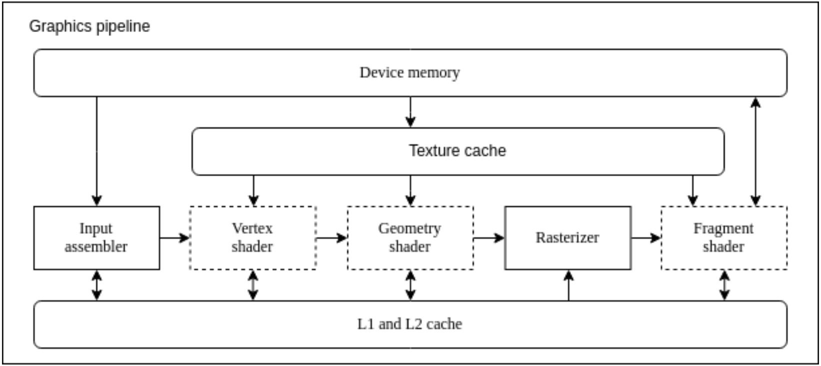
```
uniform Scalar alpha;

Vector
laplacian(in Vector T)
{
    return (Vector){laplacian(T.x), laplacian(T.y), laplacian(T.z)};
}

Vector
heat_equation(in Vector T)
{
    return alpha * laplacian(T);
}

in  Vector T_in  = (int3){0, 1, 2};
out Vector T_out = (int3){0, 1, 2};

Kernel
solve(Scalar dt)
{
    T_out = T_in + heat_equation(T_in) * dt;
}
```

Listing 4.4: Sample code for generating the stencil processing stage.

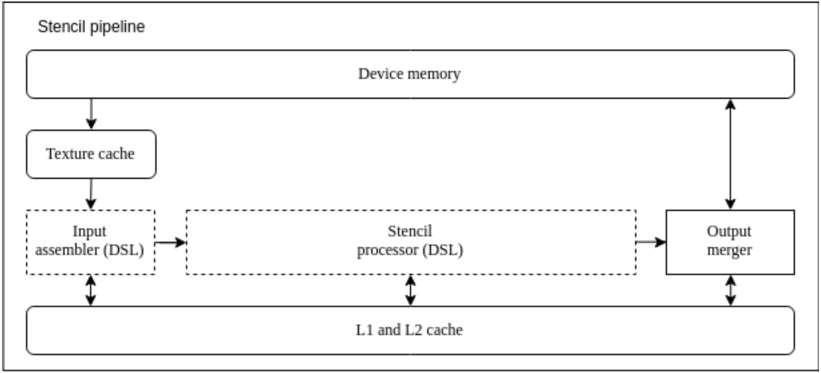
Code generation



Code generation (Graphics pipeline)



Code generation (Astaroth's Stencil pipeline)



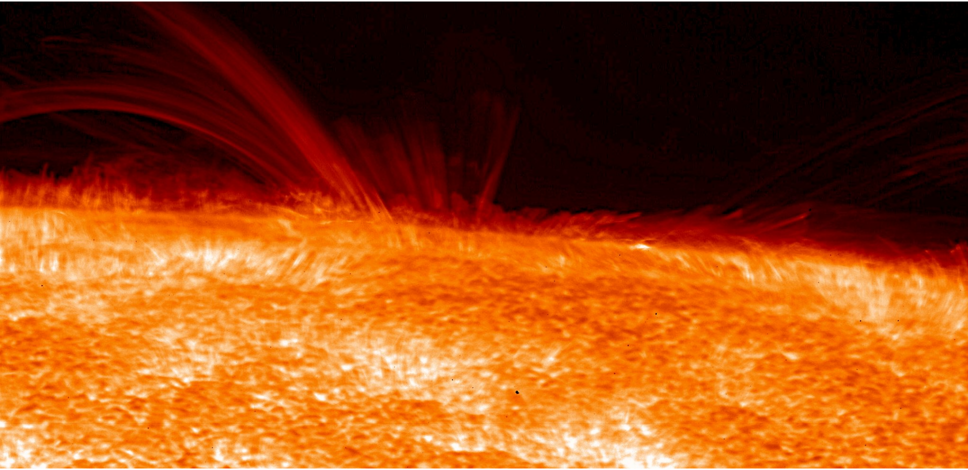


Image: *JAXA/NASA*