Comparison of Parallelisation Approaches, Languages, and Compilers for Unstructured Mesh Algorithms on GPUs

G. D. Balogh[1] I. Z. Reguly[1] G. R. Mudalige[2]

[1]Faculty of Information Technology and Bionics, Pazmany Peter Catholic University, Budapest, Hungary,

[2]Department of Computer Science, University of Warwick, Coventry, United Kingdom

November 13, 2017

PMBS17

Motivation

CUDA gives best performance on GPUs with the low level SIMT abstraction

- Converting applications to use CUDA require significant effort

OpenACC and OpenMP gives a high level directive based abstraction

- Simplify the adoption of GPUs

Performance difference between CUDA and the directive based approaches depends on multiple factors

- Type of computation
- Language (C/C++ or Fortran)
- Compiler

Motivation

- Performance difference between C/C++ and Fortran?
- How the new support for OpenMP performs?

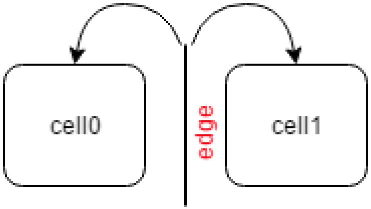
Our contribution

- Using Airfoil (C & Fortran) to compare CUDA, OpenMP 4, and OpenACC on a K40 and a P100 GPU
- We carry out analysis to identify differences between algorithms, languages, and compilers.
- Evaluate these parallelizations and compilers on Volna (C) and BookLeaf (Fortran)
- We compared the performance of
  - nvcc CUDA C/C++
  - clang CUDA and OpenMP C/C++
  - XL CUDA and OpenMP Fortran and for OpenMP C/C++
  - PGI for CUDA and OpenACC Fortran and for OpenACC C/C++
  - Cray

Ustructured mesh applications

- Operates on unstructured grids
  - A collection of nodes, edges, etc., with explicit connections - e.g. mapping tables define connections from edges to nodes
- Consists of parallel loops over some set in the mesh
- Accessing data directly on the iteration set or indirectly via a mapping
- Indirect data accesses make the parallelisation difficult
  - low-level models have advantage in orchestrating parallelism

```
cells[edgeToCells[2*i+0]]
    += edge[i];
cells[edgeToCells[2*i+1]]
    += edge[i];
```



Hierarchical coloring

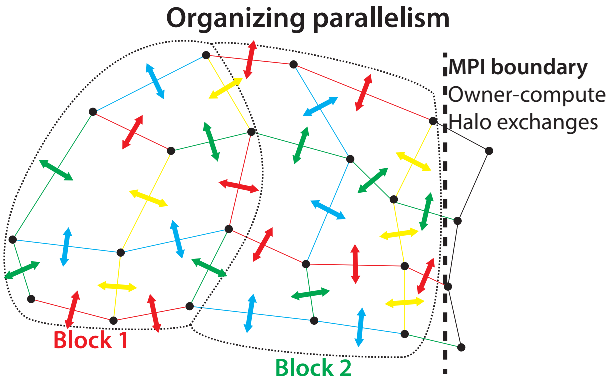


Figure: Illustration for hierarchical coloring on a computation on edges that write data on the cells. The blocks are colored so that there is no neighboring blocks with the same color and inside the blocks threads colored so that no two threads with the same color write the same data.

Global coloring

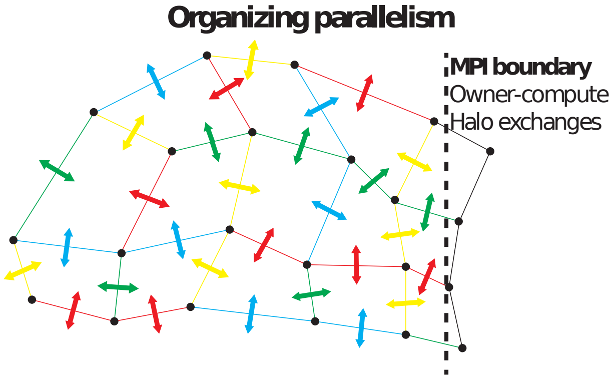
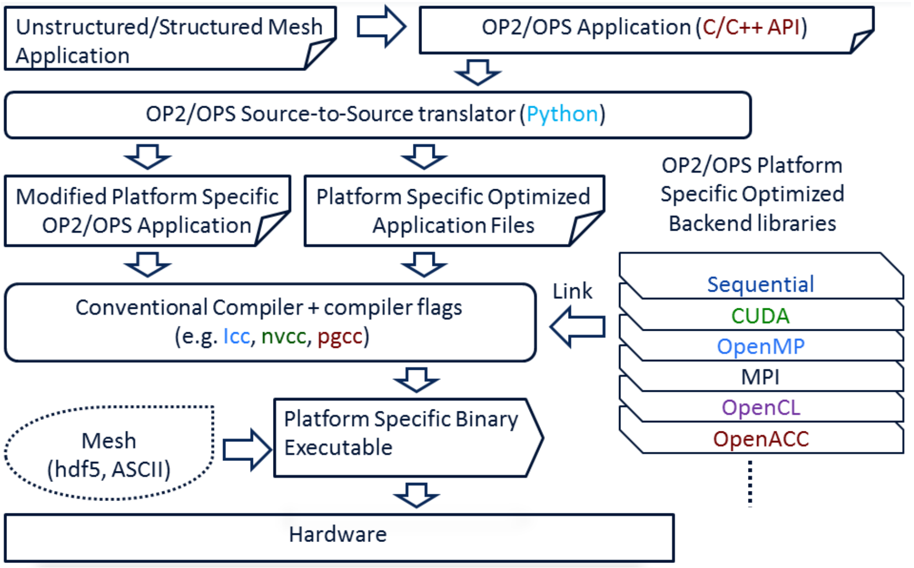


Figure: Illustration for global coloring on a computation on edges that write data on the cells. Threads colored so that no two threads with the same color write the same data.

OP2[1]


Unstructured/Structured Mesh Application
OP2/OPS Application (C/C++ API)
OP2/OPS Source-to-Source translator (Python)
Modified Platform Specific OP2/OPS Application
Platform Specific Optimized Application Files
OP2/OPS Platform Specific Optimized Backend libraries
Sequential
CUDA
OpenMP
MPI
OpenCL
OpenACC
Link
Conventional Compiler + compiler flags (e.g. Icc, nvcc, pgcc)
Mesh (hdf5, ASCII)
Platform Specific Binary Executable
Hardware

Airfoil[2]

- Non-linear 2D inviscid airfoil code
- Both in C and Fortran
- Five kernels with different access patterns:
  - **save_soln** - simple kernel, only direct reads and writes
  - **adt_calc** - computationally expensive operations, indirect reads, direct increments
  - **res_calc** - complex computation, indirect reads and indirect increments
  - **bres_calc** - similar to **res_calc** but on the boundary edges
  - **update** - simple computation with a global reduction, only direct reads and writes

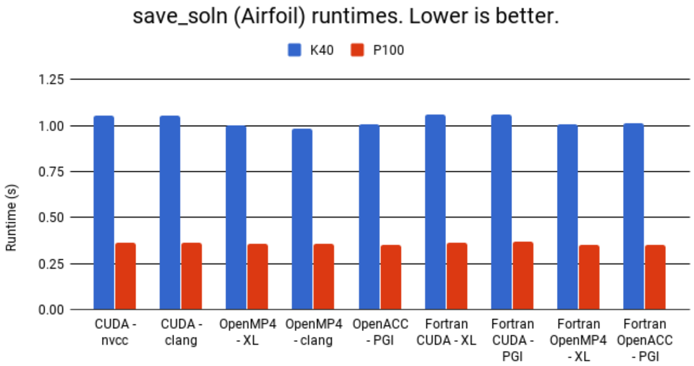
BookLeaf and Volna

- BookLeaf [3]
  - 2D Lagrangian hydrodynamics application
  - Written in Fortran 90
  - Most time consuming kernels:
    - **getacc_scatter** - indirect increments
    - **getq_christiensen1**, and **gather** - indirect reads and direct writes
- Volna [4]
  - Shallow water simulation capable of handling the complete life-cycle of a tsunami
  - Written in C++
  - Most time consuming kernels:
    - **SpaceDiscretization** - indirect reads and increments
    - **NumericalFluxes** - indirect reads and global reduction
    - **computeFluxes** - indirect reads

Simple kernels

save_soln - 3.6% of total runtime on K40 (nvcc global coloring), 3.8% on P100

- All versions perform within 2% of each other
- Fortran versions execute 10% more integer instructions

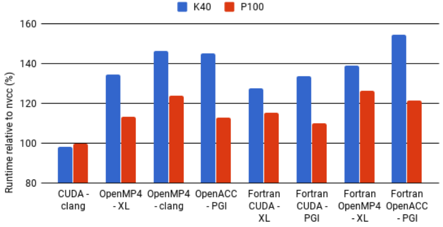


Computation intensive kernels

adt_calc - 9.5% of total runtime on K40 (nvcc global coloring), 9.1% on P100

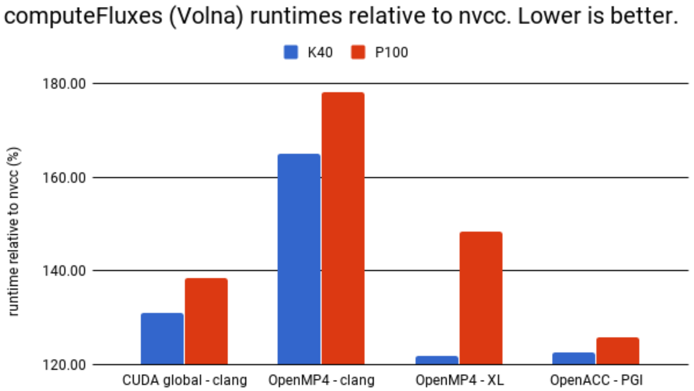
- CUDA clang executes 15-20% less integer and floating point operation than nvcc
- High register usage with high level models and Fortran
- OpenMP compilers struggle with texture cache

adt_calc (Airfoil) runtimes relative to nvcc. Lower is better.



- OpenMP with XL and OpenACC fail to optimize with multiple global writes
- High number of memory transactions with Fortran

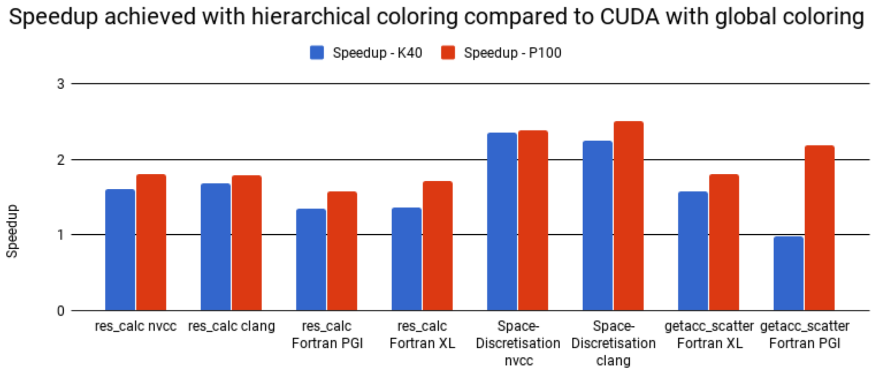
Computation intensive kernels



On large computations all versions but nvcc end up with spilled registers.

Complex kernels with indirections - Hierarchical coloring

res_calc (Airfoil) - 70% of total runtime (global coloring), **SpaceDiscretization** (Volna) - 60%, **getacc_scatter** (BookLeaf) - 12%



Complex kernels with indirections - Global coloring

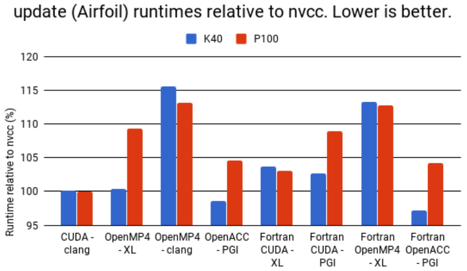
res_calc - 71% of total runtime on K40, 70% on P100

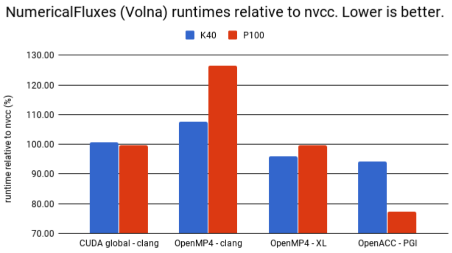
- All C versions within 5%, Fortran lagging with 10% behind C
- Main factors are occupancy and global memory usage
  - Directive based approaches and Fortran versions have high register counts
  - OpenMP doesn't use texture caches
  - OpenMP and Fortran version execute more memory transactions
- Same tendencies in **getacc_scatter** (BookLeaf)
- OpenMP XL have a 40% lower performance in **SpaceDiscretization** (Volna)

Reduction

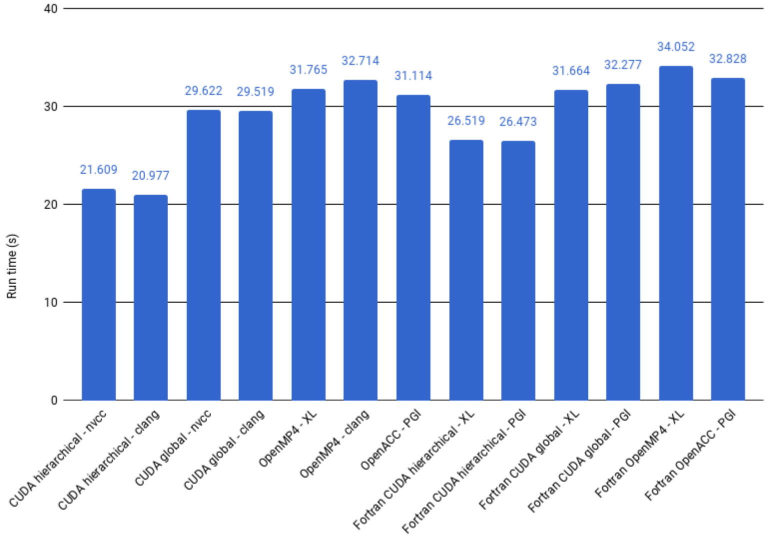
update - 15% of total runtime on K40 (nvcc global coloring), 16% on P100

- OpenACC runs separate kernels for reductions
  - Higher occupancy
- In OpenMP reductions lead to 4-6 times more control and integer instruction

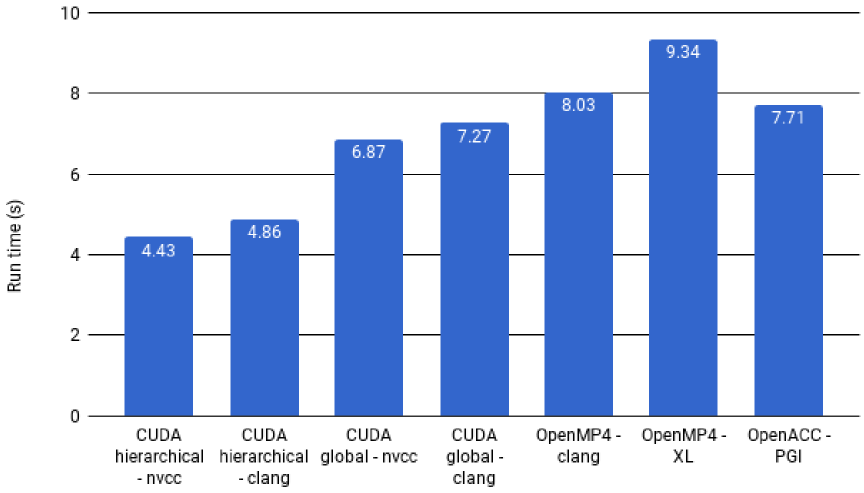




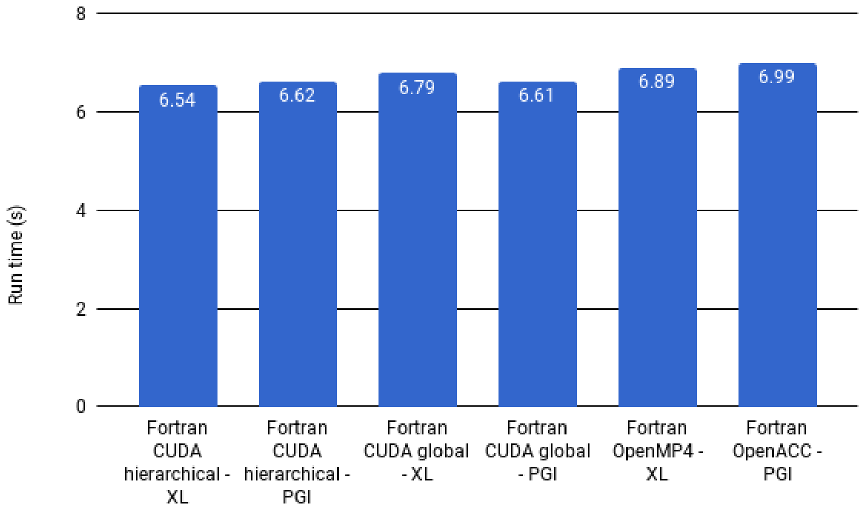
Airfoil (Total) runtimes on K40



Volna (Total) runtimes on K40



BookLeaf (Total) runtimes on K40



Conclusions

CUDA and the SIMT abstraction gives fine-grained control over GPU architectures.

- Significantly better performance with hierarchical coloring compared to global coloring
- New Clang support for CUDA
  - outperforms nvcc in arithmetic instruction counts
- On Fortran side there is support for CUDA in PGI and XL compilers
  - 5-20% performance gap compared to CUDA C/C++
  - Most cases end up with high register usage (and low occupancy)
  - High number of read transactions

Conclusions II

- OpenMP4 and OpenACC reach same performance as CUDA in simple kernels
- For complex kernels there is a 5-15% performance gap compared to CUDA
  - High register usage
  - Low support for texture caches
- Relatively new OpenMP4 support already within 5-10% of OpenACC's performance

References I

The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work http://dx.doi.org/10.5281/zenodo.22558

[1] "OP2 github repository." https://github.com/OP2/OP2-Common.

[2] M. Giles, G. Mudalige, and I. Reguly, "Op2 airfoil example," 2012.

[3] "Uk mini-app consortium." https://uk-mac.github.io.

[4] D. Dutykh, R. Poncet, and F. Dias, "The volna code for the numerical modeling of tsunami waves: Generation, propagation and inundation," *European Journal of Mechanics-B/Fluids*, vol. 30, no. 6, pp. 598–615, 2011.