Rolls-Royce Hydra CFD Code on GPUs using OP2 Abstraction

I. Z. Reguly[1], G. R. Mudalige[1], C. Bertolli[2], M. B. Giles[1], A. Betts[3], P. H. J. Kelly[3], D. Radford[4]
[1]Oxford e-Research Centre, University of Oxford, UK, [2]IBM TJ Watson Research Center, USA
[3]Department of Computing, Imperial College London, [4]Rolls-Royce plc.

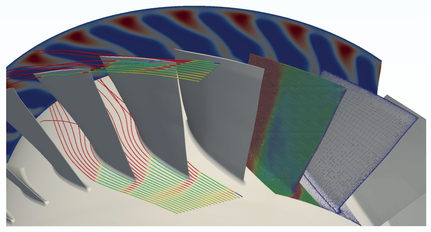
Abstract

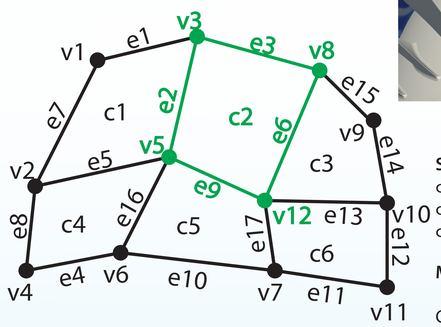
Developers of scientific codes face an important dilemma: in order to achieve high performance, it is increasingly necessary to implement low-level optimisations that are specific to a certain hardware. At the same time, there is considerable uncertainty about what programming approaches and hardware are best suited to different problems and how they will change in the future - it would be unfeasible to refactor large codes to every new generation of hardware. Domain Specific Languages address this problem by providing a high-level abstraction for a specific class of applications. OP2 defines such an abstraction for unstructured grid computations, that hides the details of parallelism and data movement, and enables efficient mapping of execution to various programming abstractions and hardware.

Rolls-Royce Hydra

- Complex and configurable full-scale industrial application used for the simulation and design of turbomachinery components.
- Equations solved are the Reynolds-Averaged Navier-Stokes second-order PDEs, using a 5-step Runge-Kutta method for time-marching, accelerated by multigrid and block-Jacobi preconditioning.
- Fortran code, uses the OPlus library, consists of 300+ parallel loops.
- OPlus library only supports distributed memory parallelism over MPI, but the latest hardware require the utilization of shared memory programming models (OpenMP, CUDA).
- Adopted Hydra to use OP2 and support modern heterogeneous architectures.

OP2 Abstraction for Unstructured Grid Computations





Sets:

```
op_decl_set(num_cells, "cells")
op_decl_set(num_edges, "edges")
op_decl_set(num_vertices, "vertices")
```

Mappings between sets:

```
                  c1  c2  c3
Cells2Edges: .....|2,3,6,9|....
op_decl_map(cells,edges,4, ...)

                  e1  e2  e3  e4
Edges2Vertices: ...|3,5|3,8|....
op_decl_map(edges,vertices,2, ...)
```

Data on sets:

Coordinates on vertices:

```
    v1  v2  v3
...|xyz|xyz|xyz|...
op_decl_dat(vertices,3,"double",...
```

Flow variables on cells:

```
     c2   c3   c4
...|uvwp|uvwp|uvwp|...
op_decl_dat(cells,4,"double",...
```

Parallel loops:

Iterate over all the elements in a set in parallel, executing a user-definded "kernel function" on each, passing a number of data arguments, either on the iteration set or through at most one level of indirection, describing the type of access (read / write / increment).

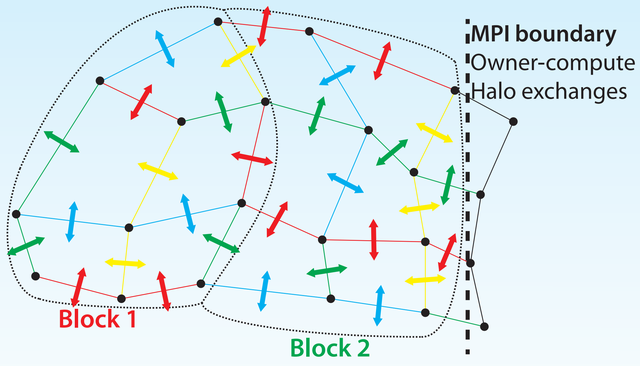
```
void res(const double *edge_flux,
         double *left_cell,
         double *right_cell) {
  //Computational code
  double f_x = edge_flux[0];
  right_cell[0] += f_x * 0.25;
  ...
}
...
op_par_loop(res,"res", edges,
   op_arg_dat(flux,-1,OP_ID,3,"double",OP_READ),
   op_arg_dat(flow,0,edges2cells,4,"double",OP_INC),
   op_arg_dat(flow,1,edges2cells,4,"double",OP_INC));
```

Based on this information, OP2 can automatically handle data dependencies and generate code for parallel execution on a range of heterogeneous architectures

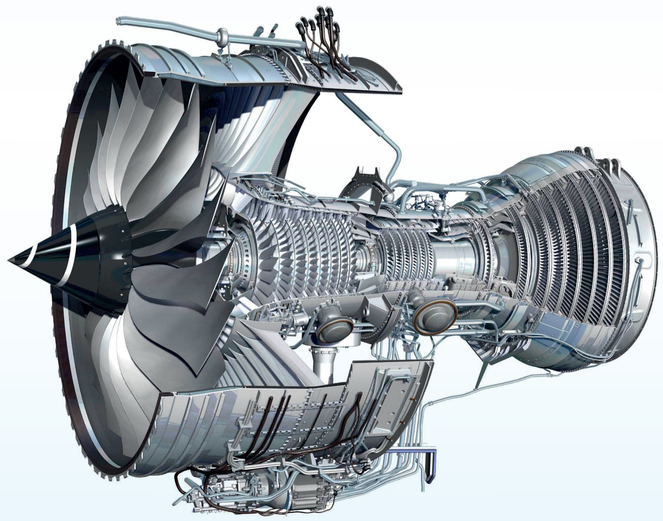
Acknowledgements

This research has been funded by the UK Technology Strategy Board and Rolls-Royce plc. through the Siloet project, the UK Engineering and Physical Sciences Research Council projects EP/I006079/1, EP/I00677X/1 on "Multi-layered Abstractions for PDEs" and the "Algorithms, Software for Emerging Architectures" (ASEArch) EP/J010553/1 project. The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work.

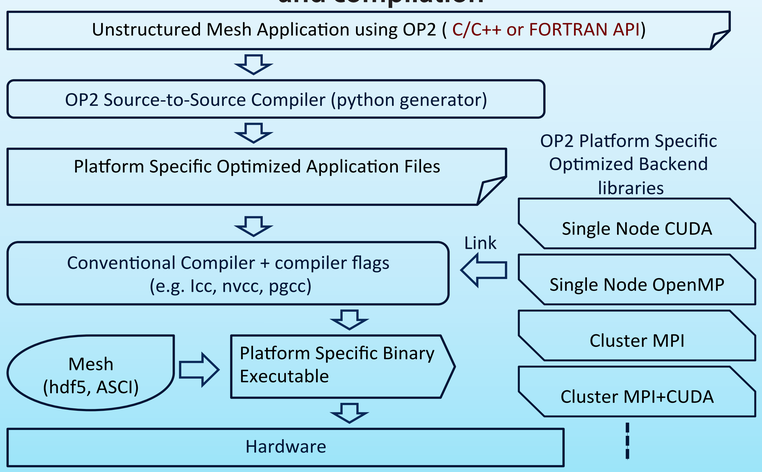
Organizing parallelism



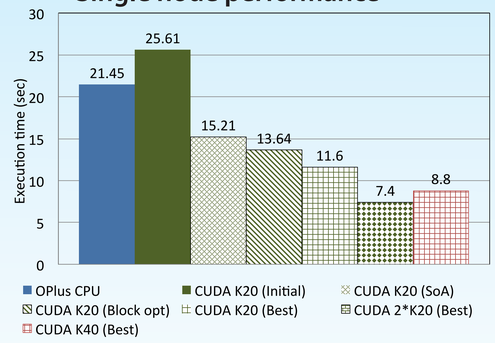
Parallelization happens on three levels:
(1) **distributed memory** MPI and the standard owner-compute approach with halo exchanges,
(2) **coarse grained shared memory**, for OpenMP threads and CUDA thread blocks, where a partition is broken up into blocks which are colored based on potential data races and then executed by color,
(3) **fine grained shared memory**, for CUDA threads in the same thread block, where set elements are colored based on potential data races, accesses are then serialized by color.



Platform-specific code generation and compilation



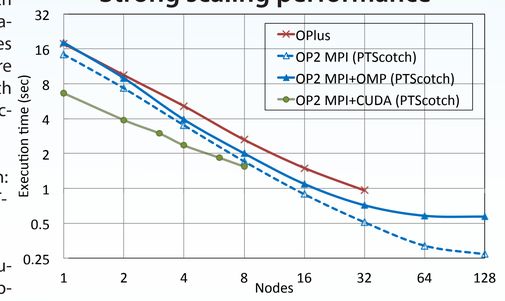
Single node performance



Optimizations can be enabled in the code generator and parameters can be automatically tuned:
- Transposing data to Structure of Arrays (SoA)
- Moving read-only data through the texture cache
- Tuning thread block size and register count

Timings from a machine with a dual-socket Intel Xeon E2640 CPU and 2 NVIDIA Tesla K20c GPUs.

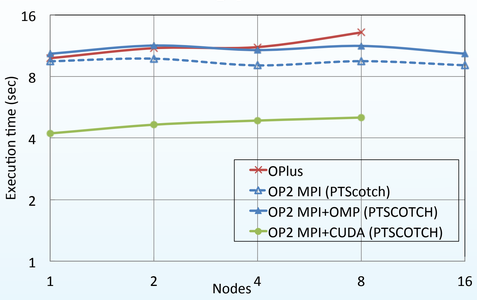
Strong scaling performance



SoA is especially advantageous with high dimensional datasets - gives coalesced memory accesses and decreases cache contention. Using the texture cache helps maximize the bandwidth and gives data reuse for indirect accesses.

Fully automated distributed execution:
- Partitioning using ParMetis or PT-Scotch
- Latency hiding
- Halo exchages and redundant execution to facilitate owner-compute approach

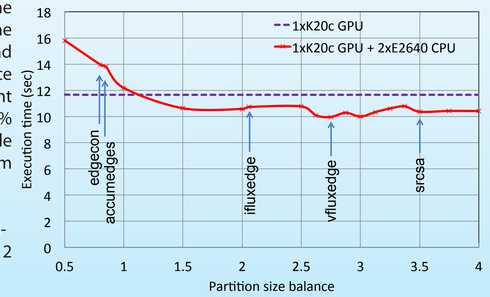
Weak scaling performance



With good utilization up to 2 times performance gain over fully utilized HECToR nodes (32 cores), at low node counts when strong scaling, and maintaned when weak scaling - doubling problem size when doubling node count.

Timings for the CPU from HECToR (32 AMD Opteron cores per node) and for the GPU from Jade (2 Tesla K20m cards per node), strong scaling with a 800k vertex mesh, weak scaling with 500k vertex per node

Hybrid CPU-GPU execution



With OP2, it is possible to utilize the CPU and the GPU at the same time. The key challenge is to find the right load balance when there are performance differences between loops on different hardware: with a good balance, 15% speedup can be achieved over a single GPU by utilizing the CPUs in the system as well.

Timings from a machine with a dual-socket Intel Xeon E2640 CPU and 2 NVIDIA Tesla K20c GPUs.

Conclusions

- A highly complex industrial application, such as Rolls-Royce Hydra, can be adopted to use the OP2 Domain Specific High-Level abstraction framework.
- Using OP2 does not compromise performance compared to the hand-coded original; in fact OP2 matches and outperforms it.
- By using OP2, Hydra is now capable of exploting the latest hardware, such as GPUs.
- OP2 provides code maintainability and future-proofing to the application developers.