Automatic parallelisation from high-level abstractions for mesh-based simulations

Gábor Dániel Balogh, Dr. István Reguly,
Prof. Mike Giles, Dr. Gihan Mudalige

Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

June 22, 2017

Outline

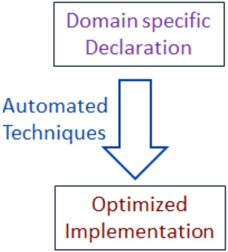
- Motivation
- Structured and unstructured grids
- OPS/OP2
  - Abstraction
  - API
- Optimizations performed inside OP2 and OPS
  - Data layout
  - Checkpointing
- Performance

Future proofing parallel HPC applications

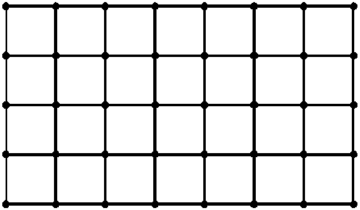
- Hardware is rapidly changing with ambitions to overcome exascale challenges
- There is considerable uncertainty about which platform to target
  - Not clear which architectural approach is likely to "win" in the long-term
  - Not even clear in the short-term which platform is best for each application
- Increasingly complex programming skills set needed to extract best performance for your workload on the newest architectures.
  - Need a lot of platform specific knowledge
  - Cannot be re-coding applications for each "new" type of architecture or parallel system.

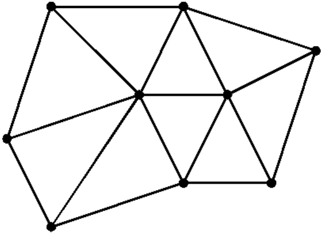
One approach to develop future proof HPC applications is the use of domain specific high-level abstractions (HLAs)

- Provide the application developer with a domain specific abstraction
  - To declare the problem to be computed
  - Without specifying its implementation
  - Use domain specific constructs in the declaration
- Create a lower implementation level
  - To apply automated techniques for translating the specification to different implementations
  - Target different hardware and software platforms
  - Exploit domain knowledge for better optimisations on each hardware system



Structured and unstructured grids





- Structured grids
  - Logical indexing with implicit connectivity
  - Easy to parallelize, including on GPUs
- Unstructured grids
  - A collection of nodes, edges, etc., with explicit connections - e.g. mapping tables define connections from edges to nodes
  - Much harder to parallelize
  - For many interesting cases, unstructured meshes are the only tool capable of delivering correct results

OP2/OPS

- Open Source project
- OP2 based on OPlus (**O**xford **P**arallel **L**ibrary for **U**nstructured **S**olvers), developed for CFD codes on distributed memory clusters
- OPS (**O**xford **P**arallel **S**tructured software) based on OP2, for structured mesh applications
- Support application codes written in C++ or FORTRAN
- Looks like a conventional library, but uses code transformations (source to source translator) to generate parallel codes

OP2 Abstraction

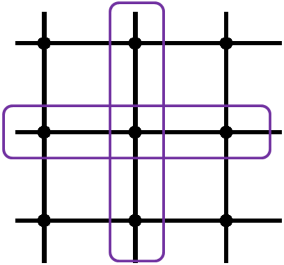
- Sets (e.g. nodes, edges, faces)
- Datasets on sets (e.g. flow variables)
- Mappings (e.g. from edges to nodes)

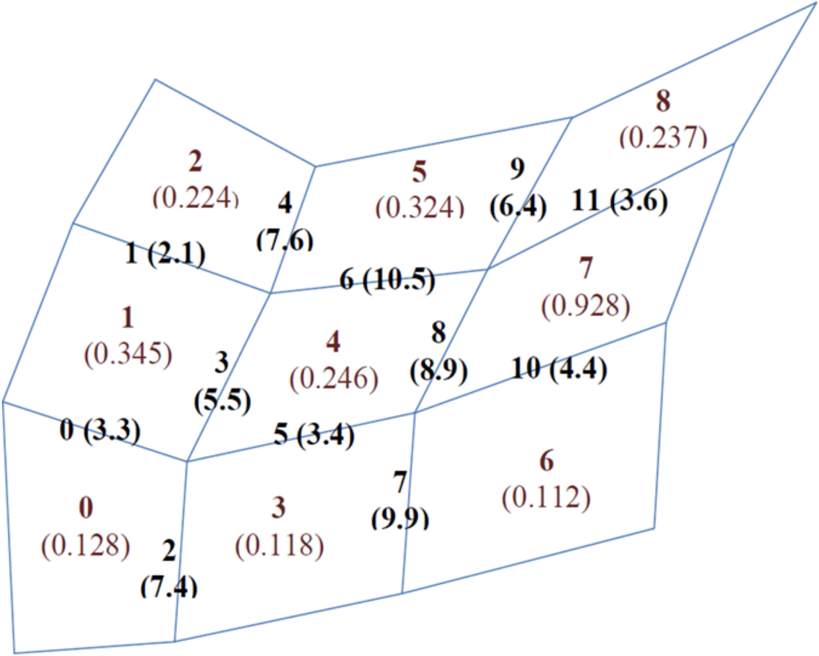
- Parallel loops
  - Operate over all members of one set
  - Datasets accessed at most one level of indirection
  - User specifies how data is used (e.g. Read-only, write-only, increment, read/write)

- Restrictions
  - Set elements can be processed in any order, doesn't affect results within machine precision
  - Static sets and mappings (no dynamic grid adaptation)

OPS Abstraction

- Blocks
- Datasets on blocks
- Stencils
- Parallel loops
  - Operate over elements of a block
  - Accessing data through stencils, describing type of access




2
(0.224)
4
(7.6)
5
(0.324)
9
(6.4)
8
(0.237)
11 (3.6)
1 (2.1)
6 (10.5)
7
(0.928)
1
(0.345)
3
(5.5)
4
(0.246)
8
(8.9)
10 (4.4)
0 (3.3)
5 (3.4)
7
(9.9)
6
(0.112)
0
(0.128)
2
(7.4)
3
(0.118)

OP2 declarations

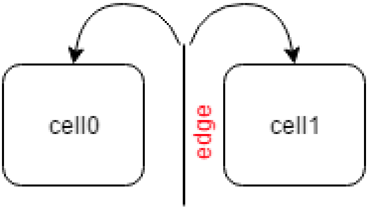
```
int nedges = 12; int ncells = 9;

int edge_to_cell[24] = {0,1, 1,2, 0,3, 1,4, 2,5,
3,4, 4,5, 3,6, 4,7, 5,8, 6,7, 7,8};

op_set edges = op_decl_set(nedges, "edges");
op_set cells = op_decl_set(ncells, "cells");
op_map pecell =
op_decl_map(edges, cells, 2, edge_to_cell,
        "edge_to_cell_map");

op_dat dcells =
op_decl_dat(cells, 1, "double", cell_data,
        "data_on_cells");
```

OP2 declarations

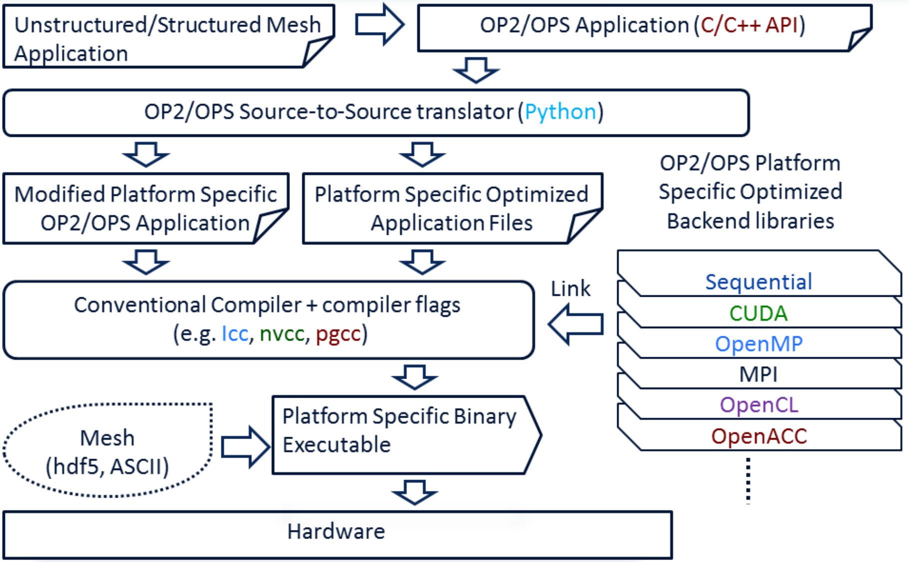
```
double cell_data[9] = {0.128, 0.345, 0.224, 0.118,
0.246, 0.324, 0.112, 0.928, 0.237};

double edge_data[12] = {3.3, 2.1, 7.4, 5.5, 7.6,
3.4, 10.5, 9.9, 8.9, 6.4, 4.4, 3.6};

op_dat dcells =
op_decl_dat(cells, 1, "double", cell_data,
        "data_on_cells");
op_dat dedges =
op_decl_set(edges, 1, "double",edge_data,
        "data_on_edges");
```

OP2 loop over edges

```
void res(double* edge,
         double* cell0,
         double* cell1){
   *cell0 += *edge;
   *cell1 += *edge;
}
```




```
op_par_loop(res,"residual_calculation", edges,
  op_arg(dedges, -1, OP_ID, 1, "double", OP_READ,
  op_arg(dcells, 0, pecell, 1, "double", OP_INC),
  op_arg(dcells, 1, pecell, 1, "double", OP_INC));
```


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

- Array-of-structs (AoS) storage preferred to struct-of-arrays (SoA)
  - Better cache hits for indirect addressing
  - Data transfers on GPU still largely ”coalesced”

a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d

(a) Array-of-Structures (AoS)

a	a	a	a	b	b	b	b	c	c	c	c	d	d	d	d

(b) Structure-of-Arrays (SoA)

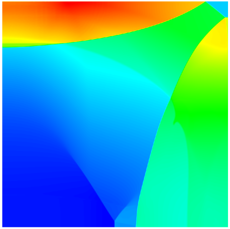
- Distributed memory parallelization using MPI makes use of non-blocking communications for overlapping computation with communications
  - First compute set elements that does not require referring halo elements while halo elements are fetched from corresponding MPI neighbour
  - On GPU clusters halos needs to be copied each time on to the GPU global memory from host via the PCIe bus
    - Tracking changes of data to minimize the number of MPI messages

- Abstraction makes it possible to implement automatic check-pointing
  - Save data every X minutes, if a crash happens, read back and continue

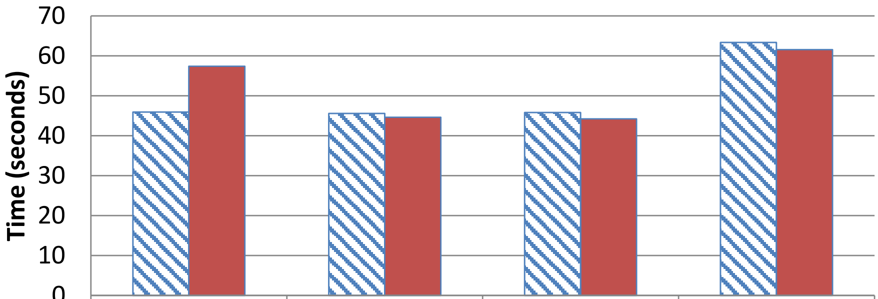
- We can implement lazy execution
  - Don't have to execute a loop immediately, unless it has a reduction that is used afterwards
  - Queue up a sequence of loops to be executed
  - Data dependency analysis at run-time
    - Find the optimum point for check-pointing
    - Communication avoidance

Cloverleaf

- Mini-app - Representative, but lightweight application-
  - 6K LoC
- 2D/3D Structured Hydrodynamics
- Explicit solution to the compressible Euler equations
- Single material
- Finite volume predictor/corrector Lagrangian step followed by an adaptive remap

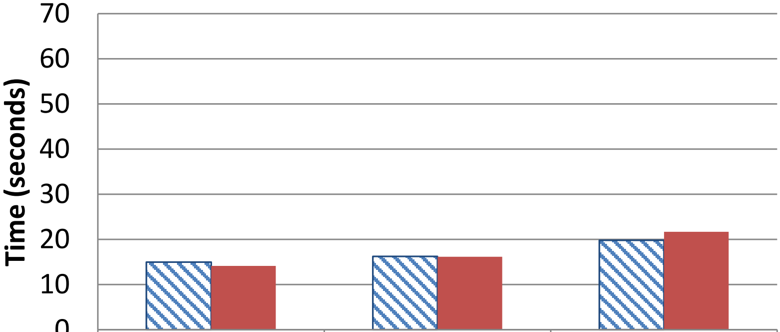


Single Node Performance CPU (2x8-core Intel CPU) (3840 x 3840 mesh, 87 iterations)



	32 OMP	32 MPI	2OMPx16MPI	OpenCL
OPS	45.92	45.55	45.82	63.35
Original	57.39	44.60	44.22	61.54

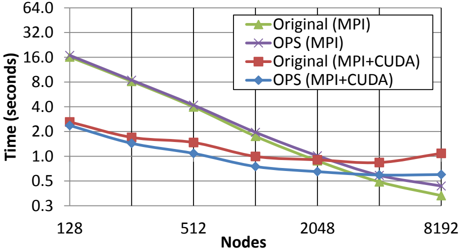
Single Node Performance GPU (K20) (3840 x 3840 mesh, 87 iterations)



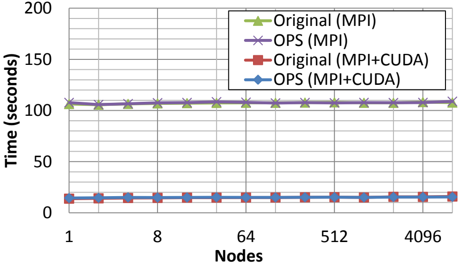
	CUDA	OpenCL	OpenACC
OPS	15.01	16.27	19.82
Original	14.14	16.19	21.67

Performance Scaling (Titan)

Strong Scaling
15360 x 15360 mesh
(87 iterations)

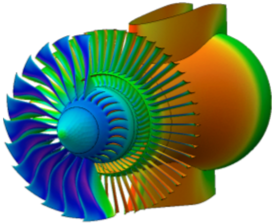


Weak Scaling
3840 x 3840 mesh per node
(87 iterations)

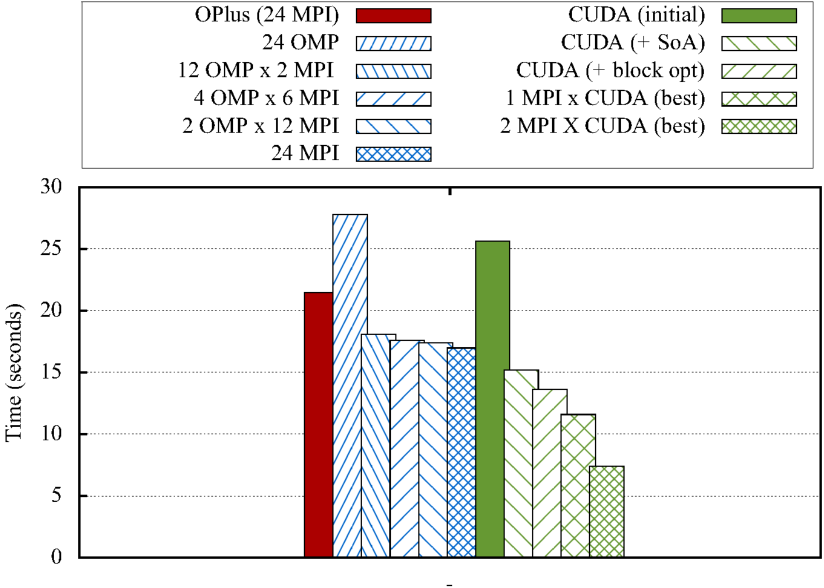


Rolls Royce - Hydra

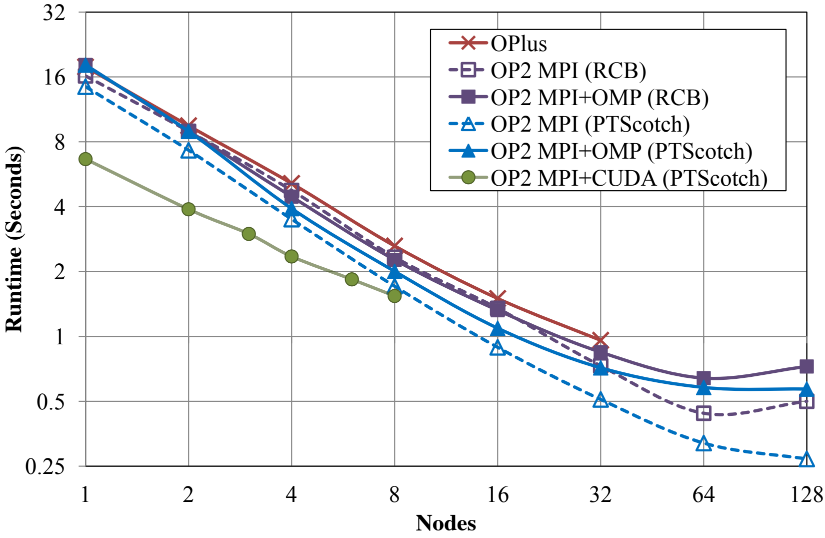
- Hydra is an unstructured mesh production CFD application used at Rolls-Royce for simulating turbo-machinery of Aircraft engines
- Production code is written in FORTRAN 77
  - 50K lines with 1000 parallel loops
- Originally using the OPlus library (predecessor of OP2)
- For real production problems, simulation time is in the order of hours, up to days for large calculations



Hydra Oplus - OP2 performance



Hydra performance scaling



Other apps with OP2 and OPS

OP2

- Airfoil mini-app
  - non-linear 2D inviscid airfoil code
  - solves 2D Euler equations
- Volna
  - numerical modelling of tsunami waves

OPS

- Tealeaf mini-app
  - linear heat conduction equation
  - solves a sparse system of linear equations, without explicitly forming the sparse matrix
- OpenSBLI
  - compressible navier-stokes solver

Summary

- OP2/OPS abstraction facilitate the development of application for parallel execution
- Nearly optimal performance
  - but the optimization is done automatically, not by the developer
- Automatic support for different parallelization models and features
- Future proof maintainable application source
  - Support future parallel systems based on the back-ends