Rolls-Royce Hydra on GPUs using OP2

I.Z. Reguly, G.R. Mudalige, M.B. Giles,
University of Oxford
C. Bertolli, A. Betts, P.H.J. Kelly
Imperial College London, (IBM TJ Watson)
David Radford
Rolls-Royce plc.

The Challenge

- HPC is undergoing an enormous change
  - New hardware architectures
  - New parallel programming abstractions, languages
- Flat (MPI) parallelism -> Multiple levels of parallel programming, heterogeneous systems (Titan, CORAL)
- Getting high performance means specialization for the hardware
- Code maintainability, longevity
- “Future proofing”

Domain Specific Languages

- Separate abstract specification of computations from the parallel implementation
- High productivity for the domain scientist
- High productivity for the library developer
  - Can experiment and validate on small benchmarks, results immediately apply to large-scale scientific codes
- As hardware changes, the library adopts the latest and greatest features, optimizations
  - “User” code doesn’t change

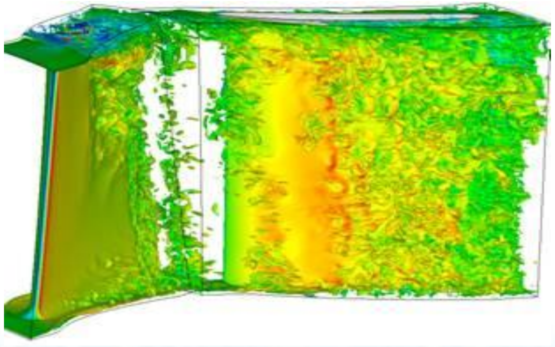
Domain Specific Languages

- Lots of research done on DSLs
  - Most of them wither away and die…
- What are the obstacles to widespread adoption?
  - Critical mass
  - Usually applied to simple, toy problems
  - Little evidence that DSLs can be applied to industrial scale applications

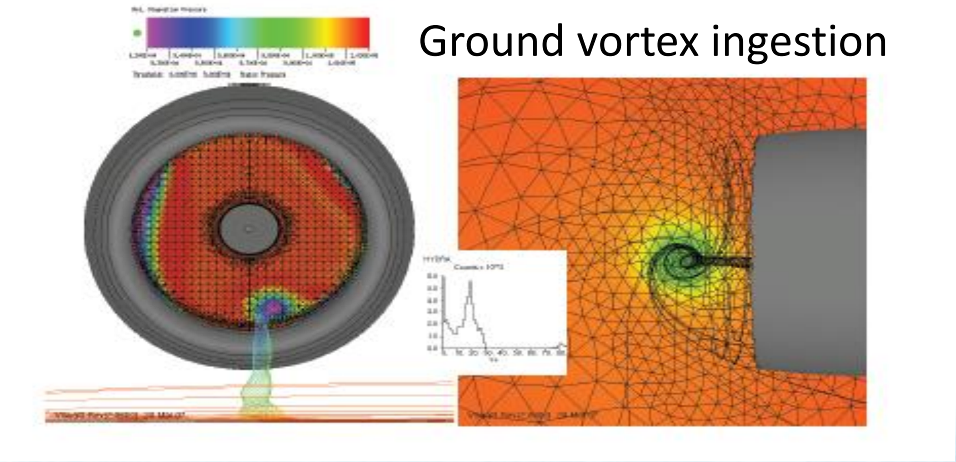
Unstructured Meshes

- For extremely complex cases, unstructured meshes are the only tool capable of delivering correct results.
- Large, very complicated codebase

Vorticity isosurface from a large Eddy simulation of a compressor

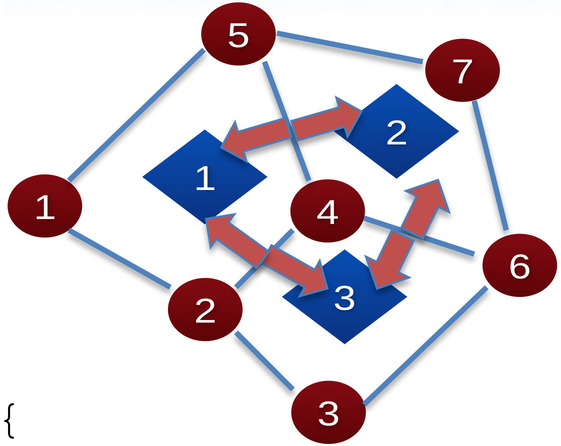


Ground vortex ingestion



OP2 for Unstructured Grids

- Abstraction:
  - Sets, maps, data
  - Loop over sets, describing access type



res.h :

```
void res(double *A, double *u, double *du) {
   (*du) += (*A) * (*u);
}
...
op_par_loop(res,"res", edges,
    op_arg_dat(A,-1,OP_ID, 1,”double",OP_READ),
    op_arg_dat(u, 0,col,1,”double",OP_READ),
    op_arg_dat(du,0,row,1,”double",OP_INC));
```

Call “res” for each edge

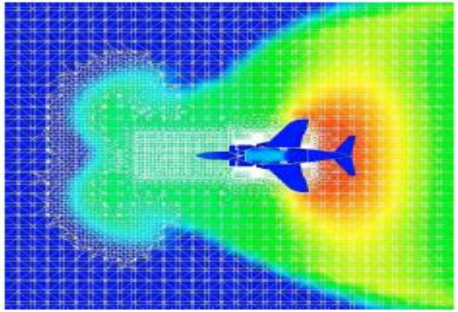
Iterate over edges

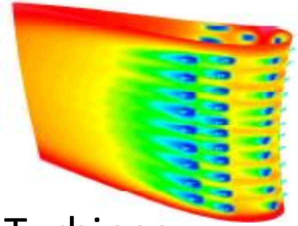
With the following arguments

Rolls-Royce Hydra

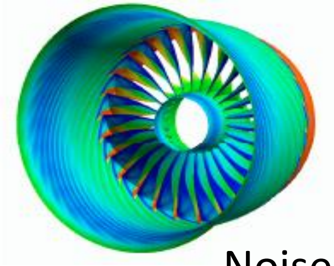
Hydra is an unstructured mesh production CFD application used at Rolls-Royce for simulating turbo-machinery of aircraft engines

Full aircraft

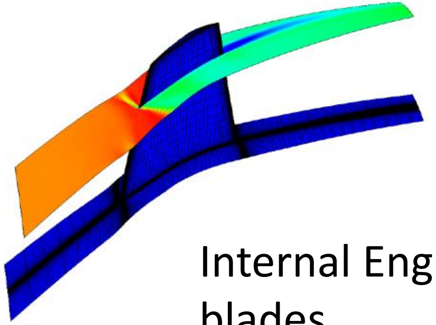




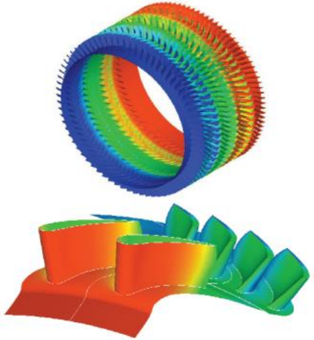
Turbines



Noise

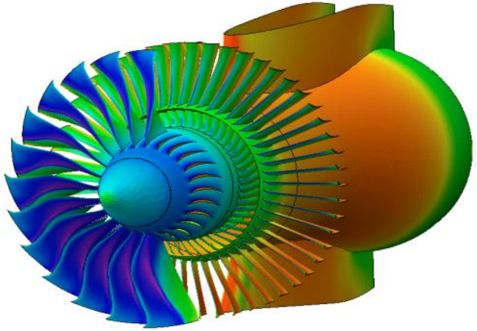


Internal Engine blades



Rolls-Royce Hydra

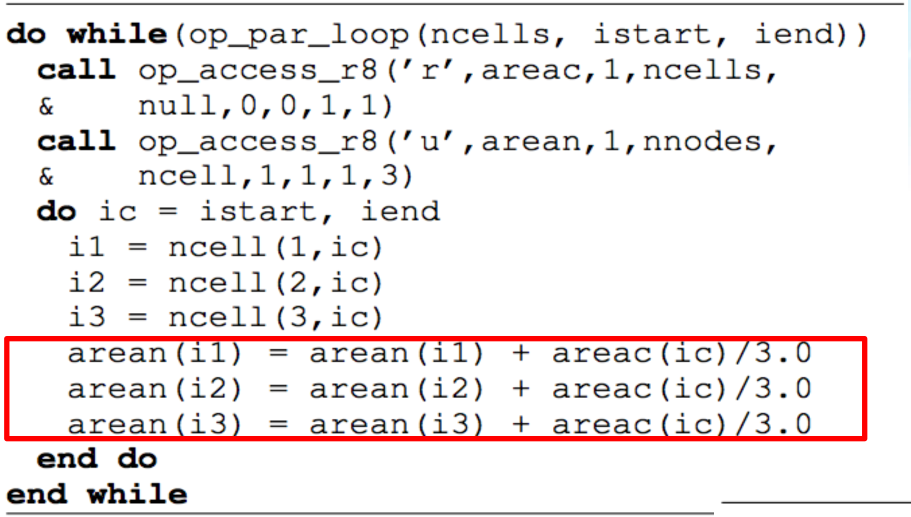
- Used for the design of turbomachinery
  - Key CFD production code
  - Steady and unsteady flow
  - Reynolds Averaged Navier-Stokes
- In development for >15 years
  - Fortran 77
  - 50k+ lines of source code
  - ~300 computational loops
- Written in OPlus – same notions of sets, maps, data and loops over sets
- Our goal is to evaluate the utility of OP2, when applied to Rolls-Royce Hydra



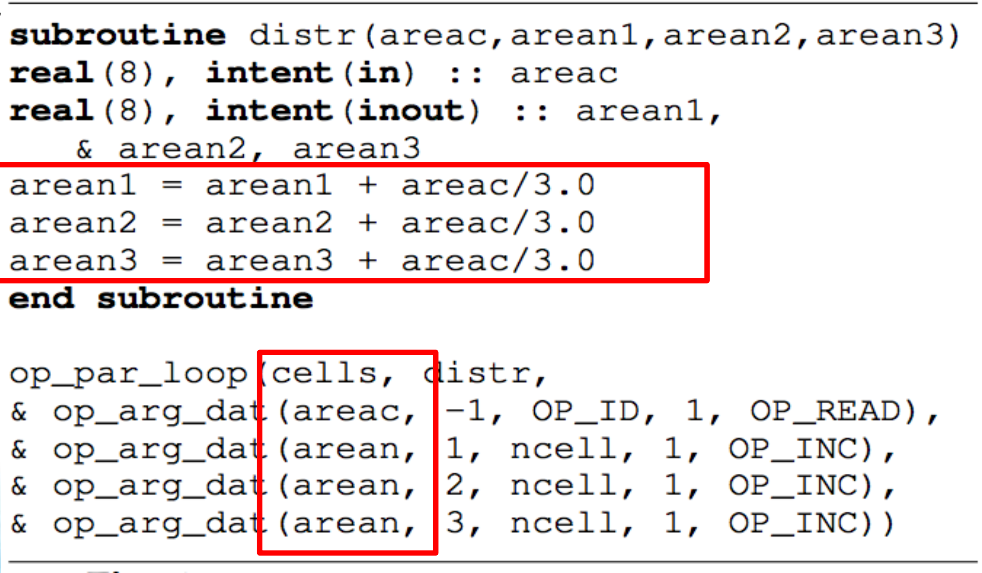
Conversion

- The original source code had to be converted to use the OP2 API, keeping the “science” intact
- Hydra was based on OPlus, the conversion was not difficult
    - Computations did not change, they were only outlined and described using the parallel loop API

From an application developer point of view, this is it – the rest is about the library



```
do while(op_par_loop(ncells, istart, iend))
  call op_access_r8(’r’,areac,1,ncells,
  &    null,0,0,1,1)
  call op_access_r8(’u’,arean,1,nnodes,
  &    ncell,1,1,1,3)
  do ic = istart, iend
    i1 = ncell(1,ic)
    i2 = ncell(2,ic)
    i3 = ncell(3,ic)
    arean(i1) = arean(i1) + areac(ic)/3.0
    arean(i2) = arean(i2) + areac(ic)/3.0
    arean(i3) = arean(i3) + areac(ic)/3.0
  end do
end while
```



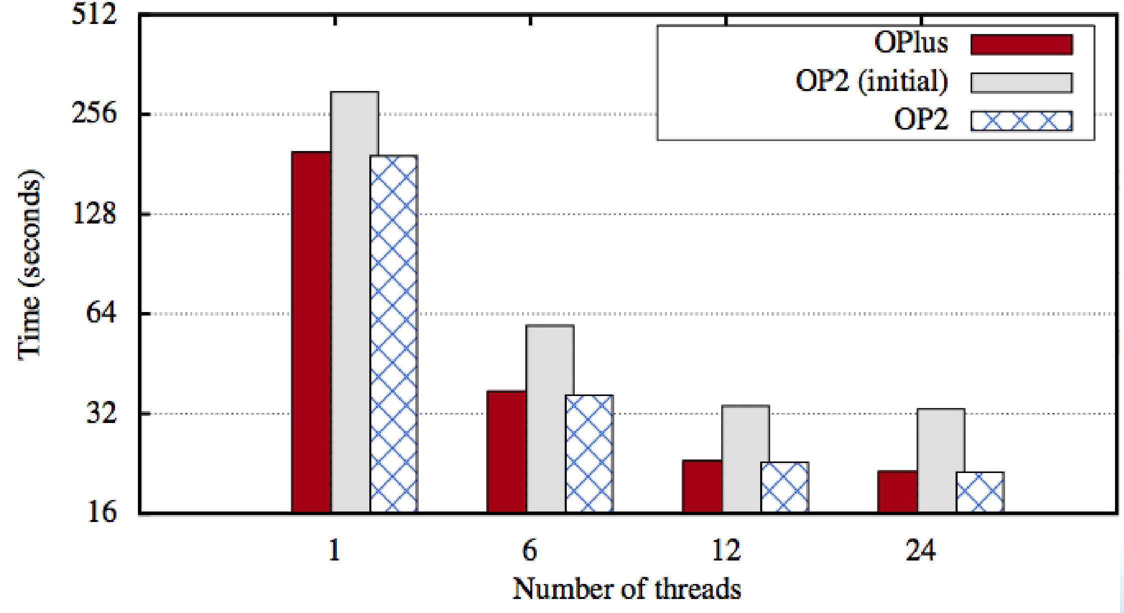
```
subroutine distr(areac,arean1,arean2,arean3)
real(8), intent(in) :: areac
real(8), intent(inout) :: arean1,
    & arean2, arean3
arean1 = arean1 + areac/3.0
arean2 = arean2 + areac/3.0
arean3 = arean3 + areac/3.0
end subroutine

op_par_loop(cells, distr,
& op_arg_dat(areac, -1, OP_ID, 1, OP_READ),
& op_arg_dat(arean, 1, ncell, 1, OP_INC),
& op_arg_dat(arean, 2, ncell, 1, OP_INC),
& op_arg_dat(arean, 3, ncell, 1, OP_INC))
```

Code generation

- OP2-Hydra can do pure MPI right away, but performance is poor due to loss of optimizations (function pointers, outlined code, going through Fortran to C bindings)
- Code generation for MPI can recover these optimizations
- Python script parses op_par_loop calls in high-level files, replaces them with calls to generated code
  - Why not compilers?

Baseline performance

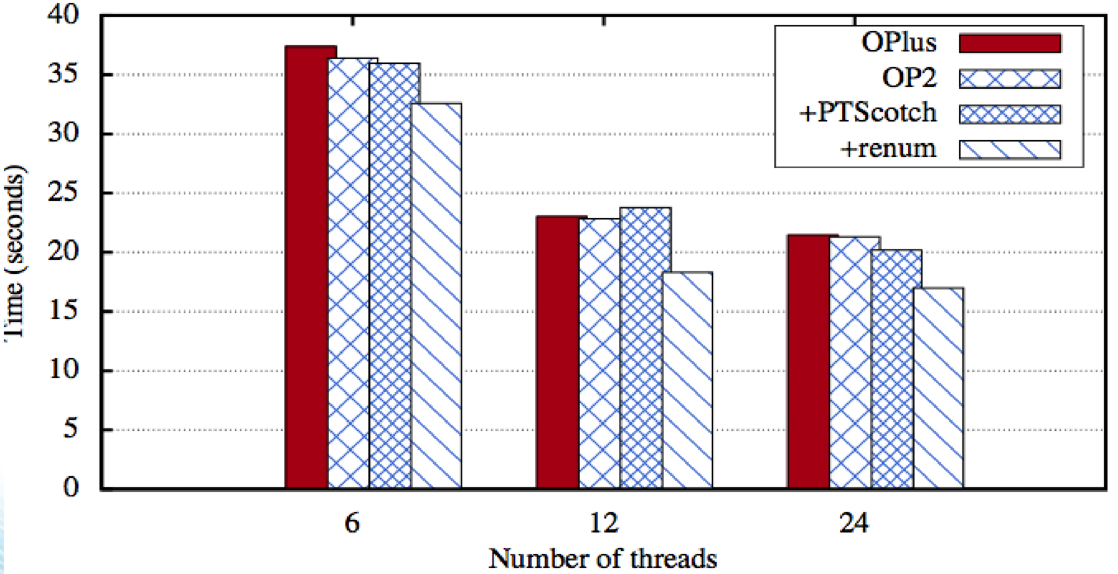
OPlus PP vs. OP2 perfectly match, down to instruction count being within 5%.



2 socket
Xeon E5-2640
2*12 cores
2.4GHz

Basic optimizations in OP2

- Support for ParMetis and PT-Scotch partitioning
- Partial halo exchanges for boundary loops
- Mesh renumbering to improve cache locality

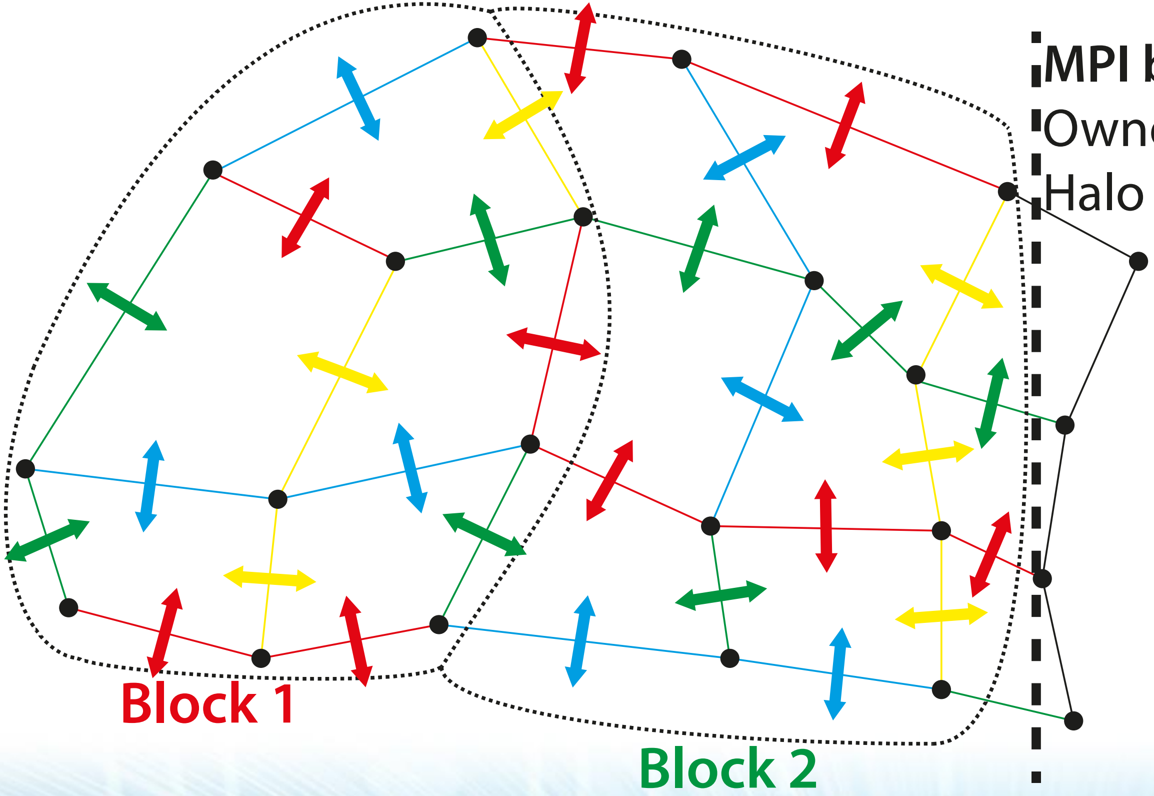


2 socket
Xeon E5-2640
2*12 cores
2.4GHz

We can match and outperform the original under the same circumstances

That alone is great, but what else can OP2 do?
Enable GPU execution of course...

Heterogeneous execution



- Fine grain parallelism with CUDA or OpenMP
- Code generation + pre-processing to support shared memory parallelism via coloring

Generating CUDA Fortran

- A Fortran module for each “kernel”
  - Set up pointers, reductions on the host
  - CUDA kernel where threads set up the parameters, call the user function, do memory movement
- Slight modifications to user kernel
  - Qualifiers, global constants

Challenges

- Large number of computational kernels
  - Direct, Indirect read, Indirect Increment
- Huge kernels
  - Datasets have up to 18 components (double precision values per set element)
  - Some kernels move up to 120 double precision values for each set element
- It’s all about bandwidth utilization and occupancy

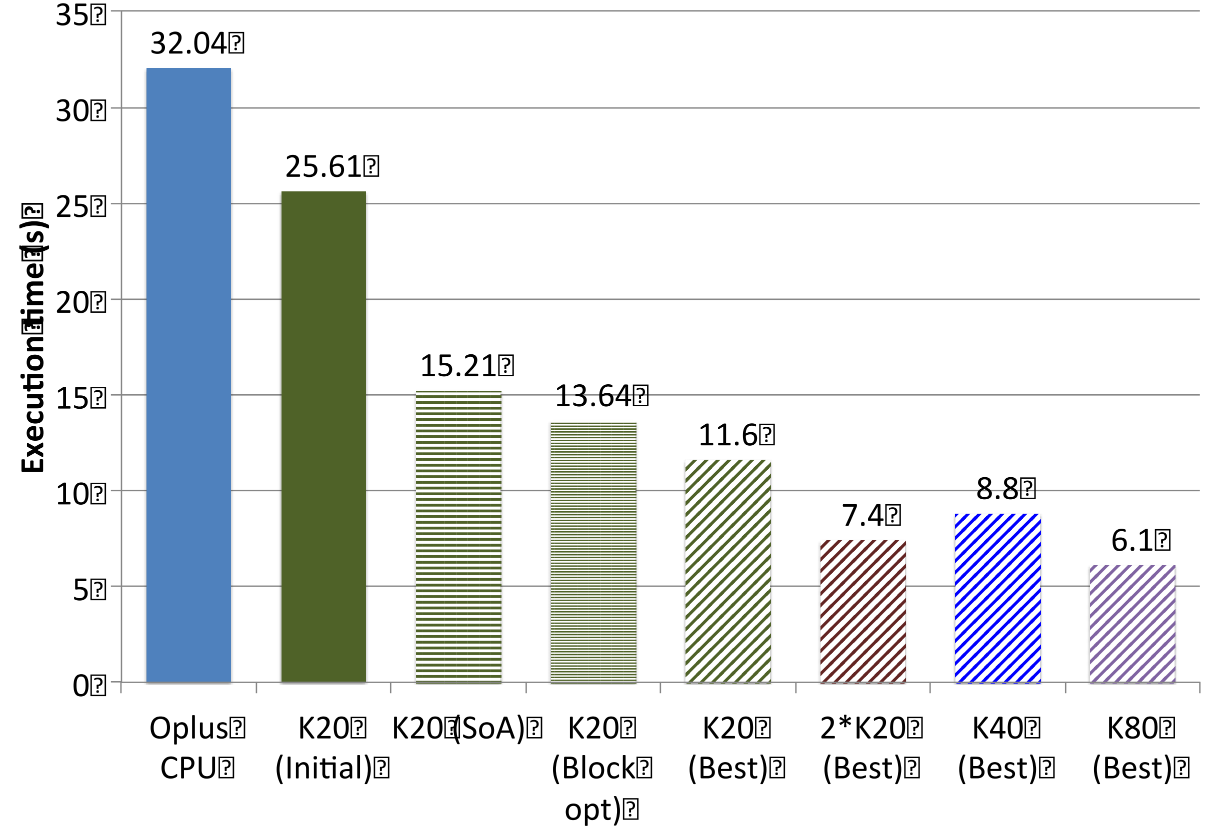
GPU optimizations

- Through the code generator
  - Replace device constants (regexp)
  - Change to SoA access (regexp)

var(m) -> var(nodes_stride*(m-1)+1), through OP2_SOA(var, nodes_stride,m)

- Manually
  - Add intent(in) to variables to enable caching loads
- Auto-tuning
  - Block sizes, register counts

GPU optimizations



Node:
Xeon E5-1650 @ 3.2 GHz
2x Tesla K20m cards

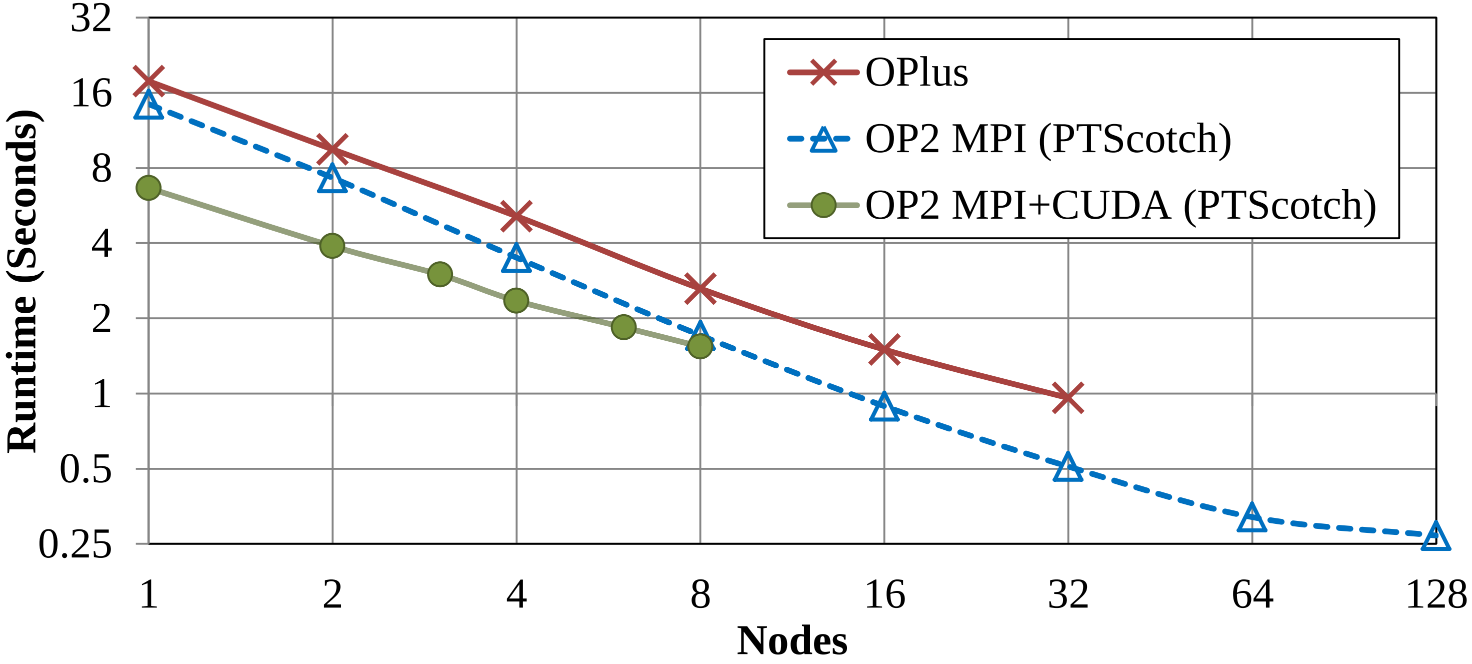
1x Tesla K40 @ 875 MHz

1x Tesla K80 @ 875 MHz

PGI 14.7

Strong scaling

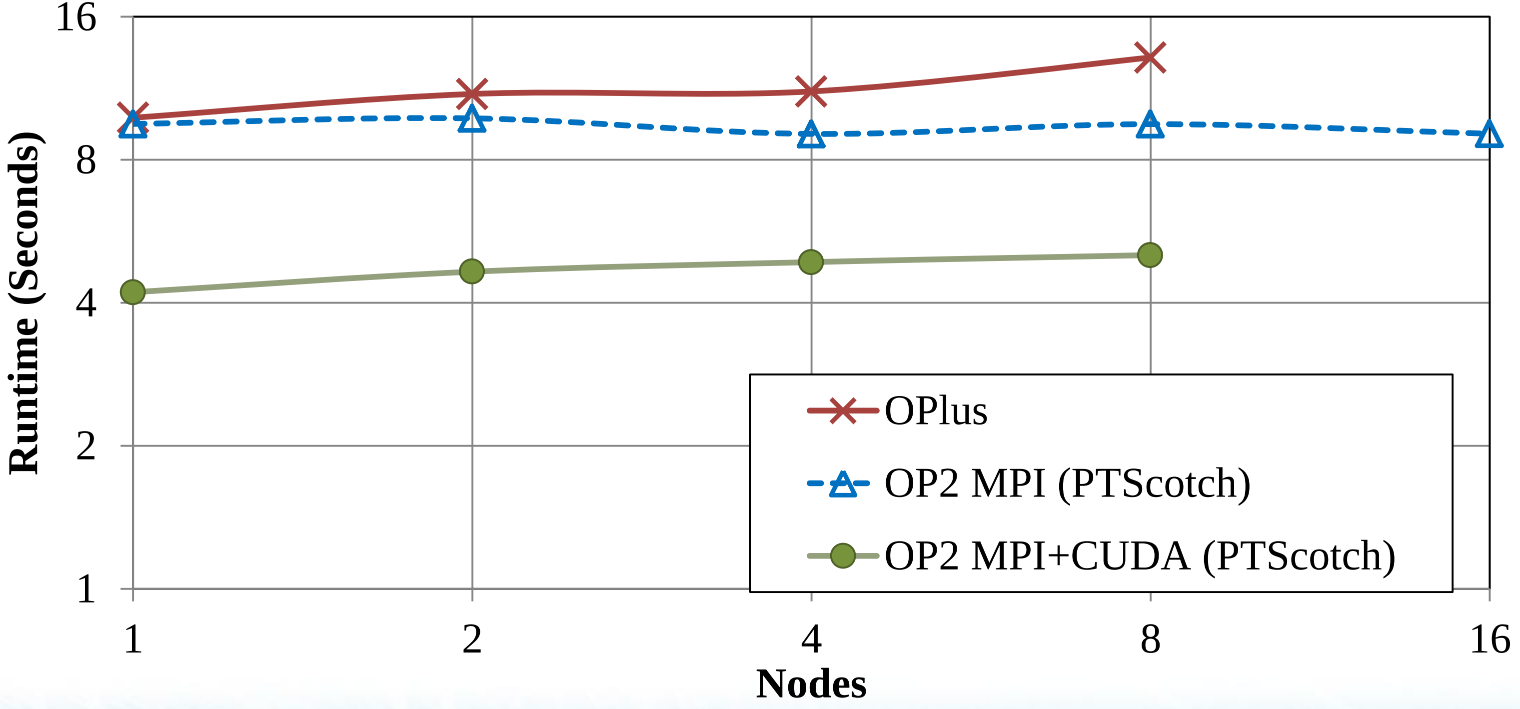
800K vertices, 2.5M edges. 1 Hector node (32 cores) and 1 Jade node (2 K20 GPUs)



Linear scaling up to 16 nodes (512 cores)

Weak scaling

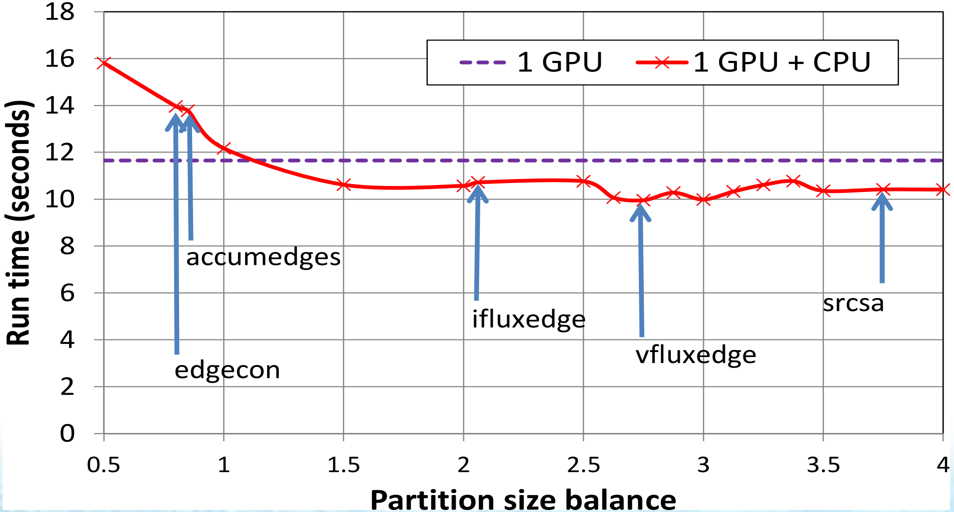
0.5M vertices per node



GPU node has 2* over HECToR node

Hybrid CPU-GPU execution

- Using the CPU and the GPU at the same time
- Some processes use the CPU, some the GPU
- How to load balance? Some loops are faster on the GPU, some on the CPU



Conclusions

- DSLs can be applied to industrial-scale codes
- Early version was slow: cost of a high-level API
  - Had to understand these limitations, code generate to circumvent them
- Matching & increased performance on the same HW
  - By using OP2, some improved techniques come for “free” (renumbering, better partitioning, better MPI, etc.)
- Enabled OpenMP, CUDA and CPU+GPU Hybrid execution
  - On such complicated code, the performance advantage is not huge – but the option is there!
- All of these optimizations apply with no (or very little) change to the user code

Thank you!

Questions?

istvan.reguly@oerc.ox.ac.uk

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.

Special thanks to: Brent Leback (PGI), Maxim Milakov (NVIDIA), Leigh Lapworth, Paolo Adami, Yoon Ho (Rolls-Royce), Endre László (Oxford), Graham Markall, Fabio Luporini, David Ham, Florian Rathgeber (Imperial College), Lawrence Mitchell (Edinburgh)