Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Performance of DPC++ on Representative Structured/Unstructured Mesh Applications

István Reguly, PPCU ITK

reguly.istvan@itk.ppke.hu

DevSummit at SC21

Overview

OP2 and OPS abstractions, build systems

Shared memory parallelization of unstructured meshes

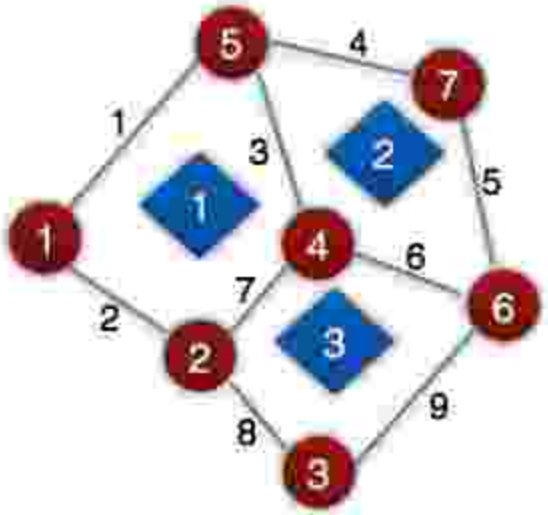
The applications: MG-CFD, Acoustic Wave solver, OpenSBLI

Performance & analysis

Conclusions

The OP2 eDSL for unstructured meshes

- OP2 (Oxford Parallel Library for Unstructured meshes)
- API for unstructured mesh computations
  - sets: edges, vertices, cells
  - maps: explicit lists, such as edges->vertices
  - data: variables defined on sets
  - operations: a parallel loop across a set, accessing data directly or indirectly via a mapping




```
op_par_loop(res,"res", edges,
  op_arg_dat(p_edge,-1,OP_ID     ,4,"double",OP_READ),
  op_arg_dat(p_cell, 0,edge2cell,4,"double",OP_INC ),
  op_arg_dat(p_cell, 1,edge2cell,4,"double",OP_INC));
```

Per-element kernel

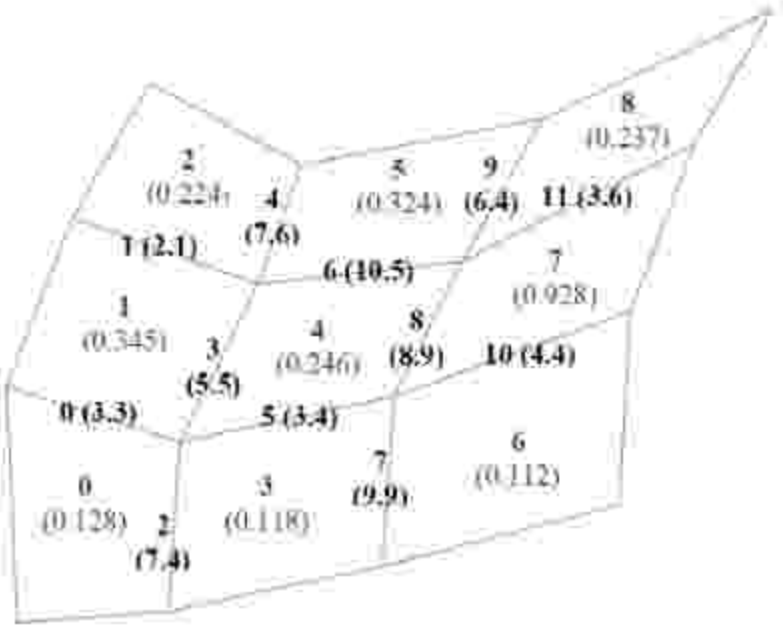
```
void res(const double *edge,
         double *cell0, double *cell1 ){
  //Computations, such as:
  cell0 += *edge; *cell1 += *edge;
}
```

OP2 FOR UNSTRUCTURED-MESH APPLICATIONS

```
//sets
op_set nodes  = op_decl_set(nnode,  "nodes");
op_set edges  = op_decl_set(nedge,  "edges");
op_set cells  = op_decl_set(ncell,  "cells");


//mapping between sets
op_map pedge  = op_decl_map(edges, nodes, 2, edge,  "pedge");
op_map pecell = op_decl_map(edges, cells, 2, ecell, "pecell");


//data on sets
op_dat p_x    = op_decl_dat(nodes , 2, "double", x   ,"p_x");
op_dat p_q    = op_decl_dat(cells , 4, "double", q   ,"p_q");
op_dat p_adt  = op_decl_dat(cells , 1, "double", adt ,"p_adt");
op_dat p_res  = op_decl_dat(cells , 4, "double", res ,"p_res");
```



OP2 FOR UNSTRUCTURED-MESH APPLICATIONS

```
//elemental kernel
void res_calc(const double* x1, const double* x2,
const double* q, double* res1, double* res2){
  //computations such as:
  res1[0] += q[0]*(x1[0]-x2[0]);
  ...
  ...
}

//Parallel loop
op_par_loop(res_calc,"residual_calculation", edges,
        op_arg_dat(p_x,  0, pedge,  2, "double", OP_READ),
        op_arg_dat(p_x,  1, pedge,  2, "double", OP_READ),
        op_arg_dat(p_q, -1,OP_ID,   4, "double", OP_READ),
        op_arg_dat(p_res, 0, pecell, 4, "double", OP_INC),
        op_arg_dat(p_res, 1, pecell, 4, "double", OP_INC));
```

Directly accessed

Iteration set

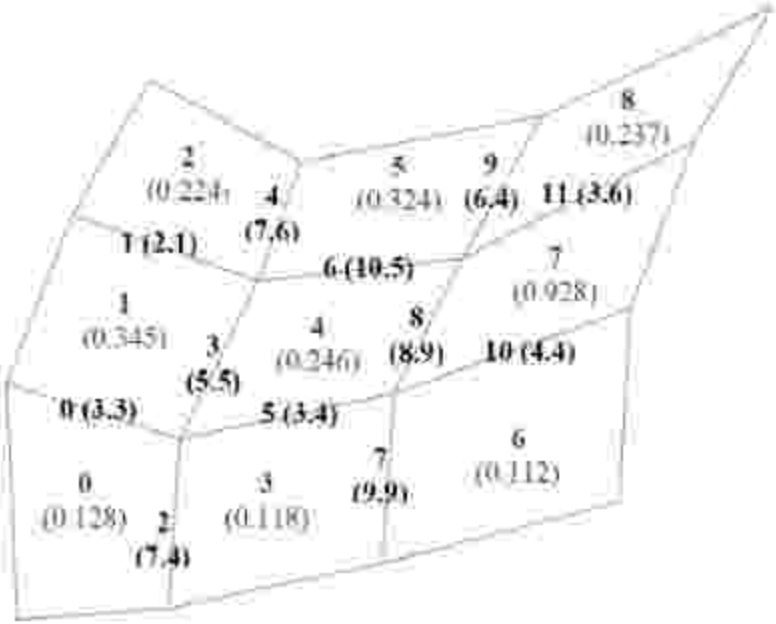
Access descriptors

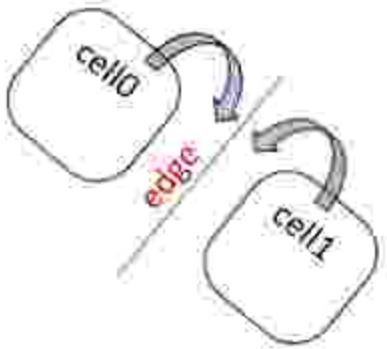
Indirection index

Indirectly accessed via mapping

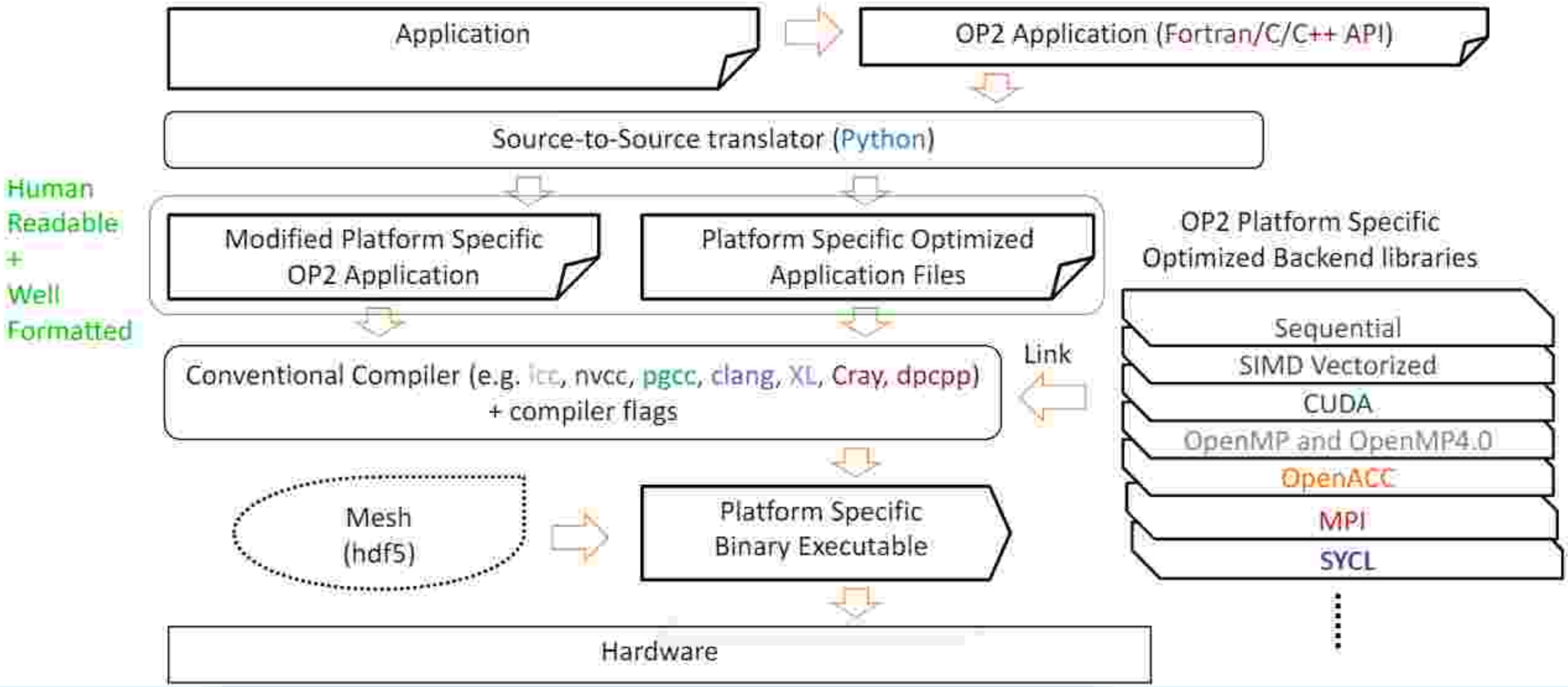
Data dimension

Data type (for checks)



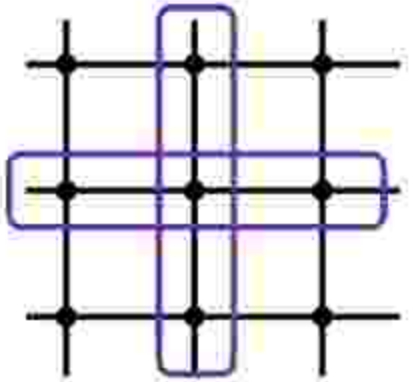


Application Development



The OPS eDSL for multi-block structured-mesh applications

- OPS (Oxford Parallel Library for Structured meshes)
- API for multi-block structured mesh computations
  - blocks
  - data on blocks, with specific size (staggered grids, multigrid)
  - stencils (possibly strided)
  - operations: a parallel loop on a block, accessing data with stencils



```
//elemental kernel
void poisson_kernel(const double* u, double* v) {
  v[OPS_ACC1(0,0)] = ((u[OPS_ACC0(-1,0)]-2.0f*u[OPS_ACC0(0,0)]+u[OPS_ACC0(1,0)])*0.125f
                   + (u[OPS_ACC0(0,-1)]-2.0f*u[OPS_ACC0(0,0)]+u[OPS_ACC0(0,1)])*0.125f
                   +  u[OPS_ACC0(0,0)]);
}
ops_par_loop(poisson_kernel, "poisson_kernel", block0, 2, range,
             ops_arg_dat(u, 1, S2D_00_P10_M10_0P1_0M1, "double", OPS_READ),
             ops_arg_dat(v, 1, S2D_00, "double", OPS_WRITE));
```

Accessed via stencil

Access descriptors

The OPS eDSL for multi-block structured-mesh applications

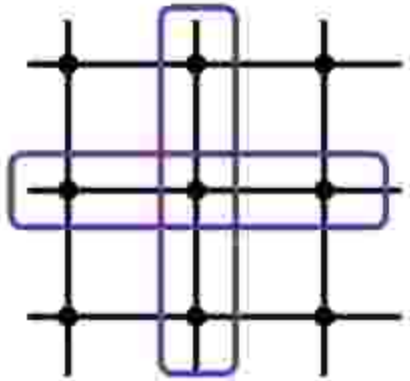
```
ops_block block0 = ops_decl_block(2, "poisson block");
ops_dat u = ops_decl_dat(block0, 1, size, base, d_m, d_p, temp, "double", bufu);
ops_dat v = ops_decl_dat(block0, 1, size, base, d_m, d_p, temp, "double", bufv);

int sten_0[] = {0,0}; int sten_1[] = {0,0, 1,0, -1,0, 0,1, 0,-1};
ops_stencil S2D_00_P10_M10_0P1_0M1 = ops_decl_stencil(2, 1,sten_1,"00,10,-10,01,0-1");
ops_stencil S2D_00 = ops_decl_stencil(2, 1, sten_0, "00");

int iter_range[] = {0,100,0,100};

#define OPS_ACC0(x, y) (x + xdim0 * (y))
#define OPS_ACC1(x, y) (x + xdim1 * (y))

//elemental kernel
void poisson_kernel(const double* u, double* v) {
  v[OPS_ACC1(0,0)] = ((u[OPS_ACC0(-1,0)]-2.0f*u[OPS_ACC0(0,0)]+u[OPS_ACC0(1,0)])*0.125f
                   + (u[OPS_ACC0(0,-1)]-2.0f*u[OPS_ACC0(0,0)]+u[OPS_ACC0(0,1)])*0.125f
                   +  u[OPS_ACC0(0,0)]);
}
ops_par_loop(poisson_kernel, "poisson_kernel", block0, 2, range,
             ops_arg_dat(u, 1, S2D_00_P10_M10_0P1_0M1, "double", OPS_READ),
             ops_arg_dat(v, 1, S2D_00, "double", OPS_WRITE));
```



Accessed via stencil

Access descriptors

The OPS eDSL for multi-block structured-mesh applications

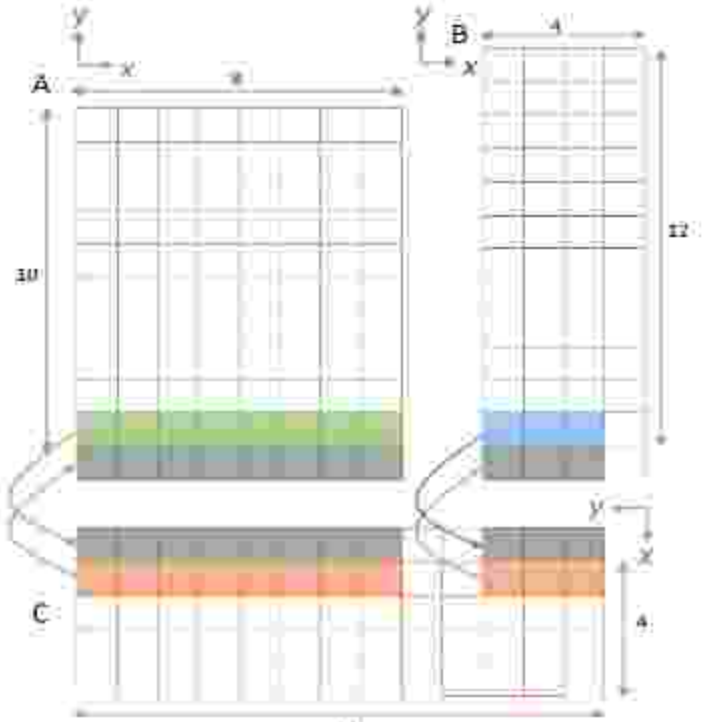
```
//halo from C to A
int iter_CA[] = {1,8}; //num of elems in each dim
int base_from[] = {0,5}; int base_to[] = {0,-1};
int axes_to[] = {-2,1}; int axes_from[] = {1,2};

ops_halo halo_C_A = ops_decl_halo(dat3, dat1, iter_CA
                        base_from, base_to,
                        axes_from, axes_to);

//halo from A to C
int iter_AC[] = {8,1};
int base_from[] = {0,0}; int base_to[] = {-1,5};
int axes_from[] = {1,2}; int axes_to[] = {-2,1};

ops_halo halo_A_C = ops_decl_halo(dat3, dat1, iter_AC
                        base_from, base_to,
                        axes_from, axes_to);

//create a halo group
ops_halo grp[] = {halo_C_A,halo_A_C};
ops_halo_group G1 = ops_decl_halo_group(2,grp);
```

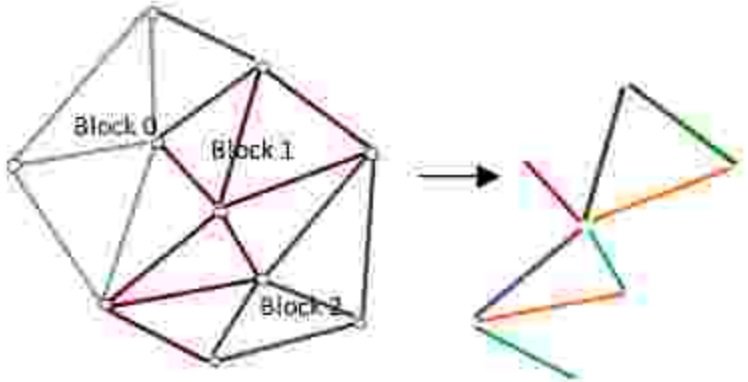


Unstructured mesh execution strategies

- Sequential execution – edge after edge, updating vertices. Good data locality
- Resolving race conflicts in shared memory parallel environments
  - Global coloring
    - Very simple, no data locality
  - Hierarchical coloring
    - Parallelism between blocks, sequential/parallel within block
    - Data locality within blocks, not across
  - Atomics (fp64)
    - Limited support/performance
    - Good data locality
- Not all can be used for different hardware, performance varies too

Global coloring

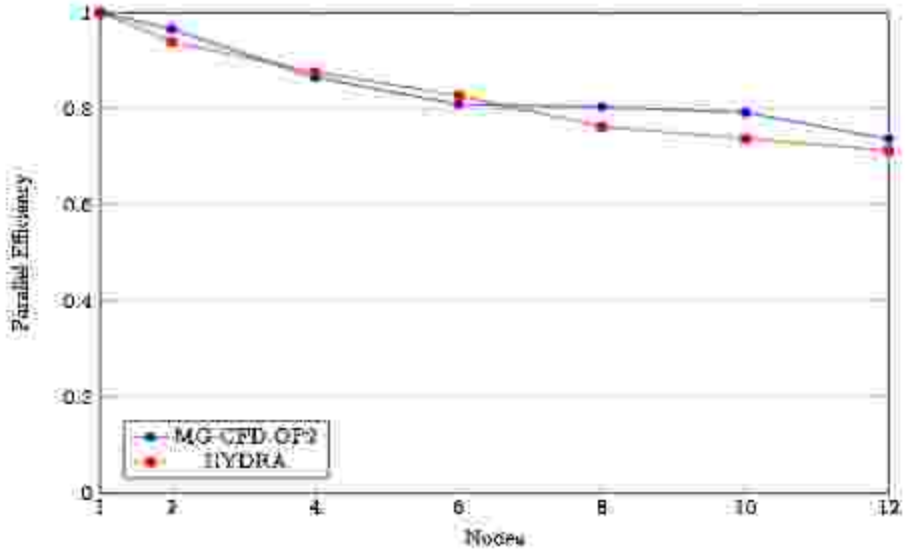
Hierarchical coloring



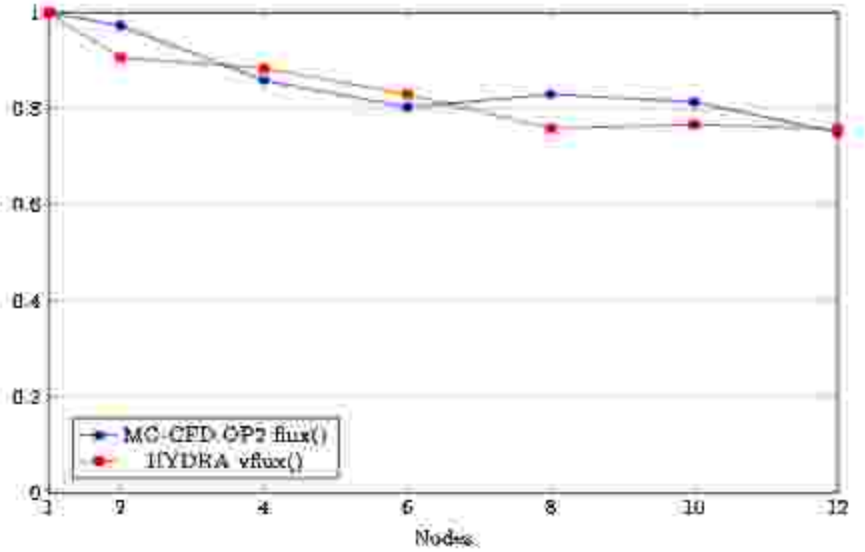
MG-CFD – OP2 UNSTRUCTURED MESH

- MG-CFD: A proxy application of Rolls-Royce Hydra
  - Open-source multigrid unstructured mesh mini-application – easy to use, modify, distribute
  - Capture key performance characteristics: CFD computation, FV, unstructured mesh, multigrid
  - Adapted to use single precision
- Representative:

Total walltime



Flux accumulation



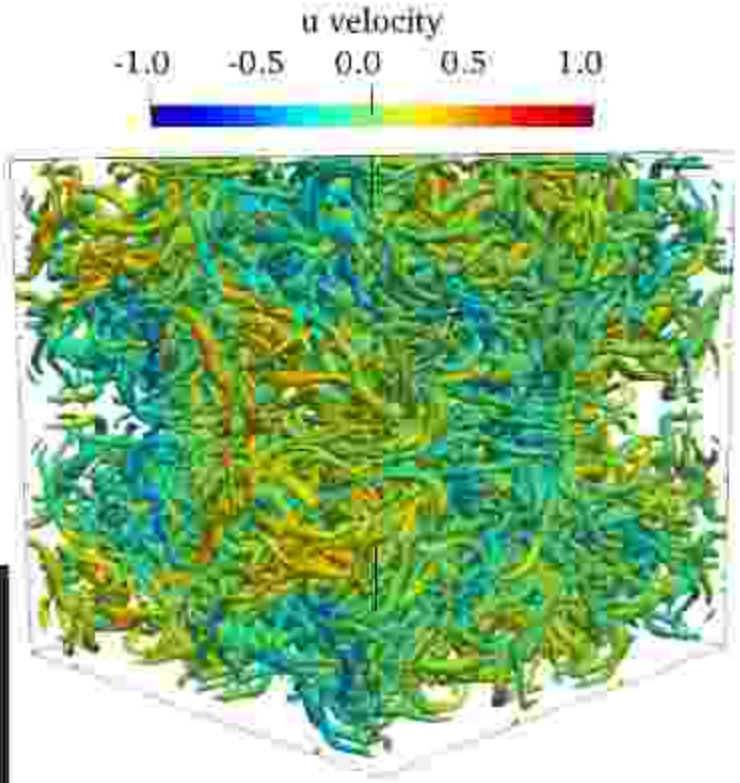
https://github.com/warwick-hpsc/MG-CFD-app-[plain, OP2]

https://github.com/warwick-hpsc/MG-CFD-performance-model

Owenson A.M.B., Wright S.A., Bunt R.A., Ho Y.K., Street M.J., and Jarvis S.A. (2019), An Unstructured CFD Mini-Application for the Performance Prediction of a Production CFD Code, Concurrency Computat. Pract Exper., 2019

OpenSBLI

- Compressible Navier-Stockes solver
  - With shock capturing WENO/TENO
  - 4th order Finite Difference, 256^3 grid, double precision
- OpenSBLI is a Python library
  - SymPy expressions
  - OPS code generated




```
ndim = 3
sc1 = "**{'scheme':'Teno'}"
mass = "Eq(Der(rho,t), - Conservative(rhou_j,x_j,%s))" % sc1
momentum = "Eq(Der(rhou_i,t) , -Conservative(rhou_i*u_j + KD(_i,_j)*p,x_j) , %s) + Der(tau_i_j,x_j) )" % sc1
energy = "Eq(Der(rhoE,t), - Conservative((p+rhoE)*u_j,x_j, %s) - Der(q_j,x_j) + Der(u_i*tau_i_j ,x_j) )" % sc1
stress_tensor = "Eq(tau_i_j, (mu/Re)*(Der(u_i,x_j)+ Der(u_j,x_i) - (2/3)* KD(_i,_j)* Der(u_k,x_k)))"
heat_flux = "Eq(q_j, (-mu/((gam-1)*Minf*Minf*Pr*Re))*Der(T,x_j))"

Avg = RoeAverage([0, 1])
LLF = LLFTeno(teno_order, averaging=Avg)
cent = Central(4)
rk = RungeKuttaLS(3, formulation='SSP')

boundaries[direction][side] = IsothermalWallBC(direction, 0, wall_eqns)

alg = TraditionalAlgorithmRK(block)
OPSC(alg)
```

Acoustic Wave solver

- 3D acoustic laplacian for isotropic wave equation
- 512^3 grid, single precision
- Space order adaptable 2-16, we use 8 here

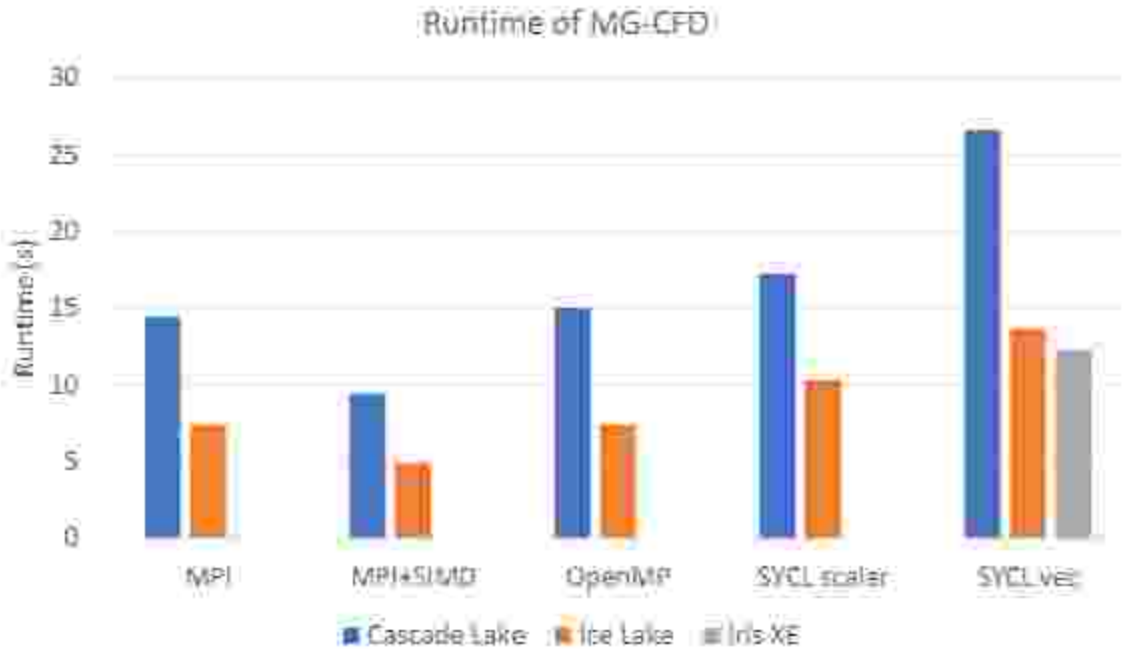
Testing environment

- GPUs:
  - Intel Iris XE MAX
- CPUs – single socket only to avoid NUMA issues:
  - Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz, 16 cores (single socket)
  - Intel(R) Xeon(R) Platinum 8360Y @ 2.40 GHz, 36 cores (single socket)
- SYCL compilers
  - Intel OneAPI 2021.4
- Baseline implementations
  - Intel compilers (from oneAPI) and Intel MPI (for MPI, OpenMP, MPI+OpenMP)
- MG-CFD: NASA Rotor37, 4 multigrid levels, 8M vertices on finest level
- OpenSBLI: 256^3
- Acoustic: 512^3

Performance – MG-CFD

- Variety of different parallelizations
- MPI
  - "Sequential" execution in each process
  - Indirect loops do not vectorize
- MPI+SIMD
  - Each process gathers into vectors, executes a vectorized kernel, then serially scatters
  - All loops vectorize
- OpenMP and SYCL scalar
  - Threads/workgroups process one block serially (1 workitem) – no vectorization for indirect loops
- SYCL vec
  - Workgroup of 16 workitems (1 subgroup) – vectorization and colored scatters with SG barriers
  - For Iris XE, 32 workitems

Platform	STREAM Bandwidth
Cascade Lake	67 GB/s
Ice Lake	148 GB/s
Iris XE	58 GB/s



Performance MG-CFD

- Key kernel: compute flux
  - >50% of runtime
  - 150 flops/iteration, 14 sqrt/div ops

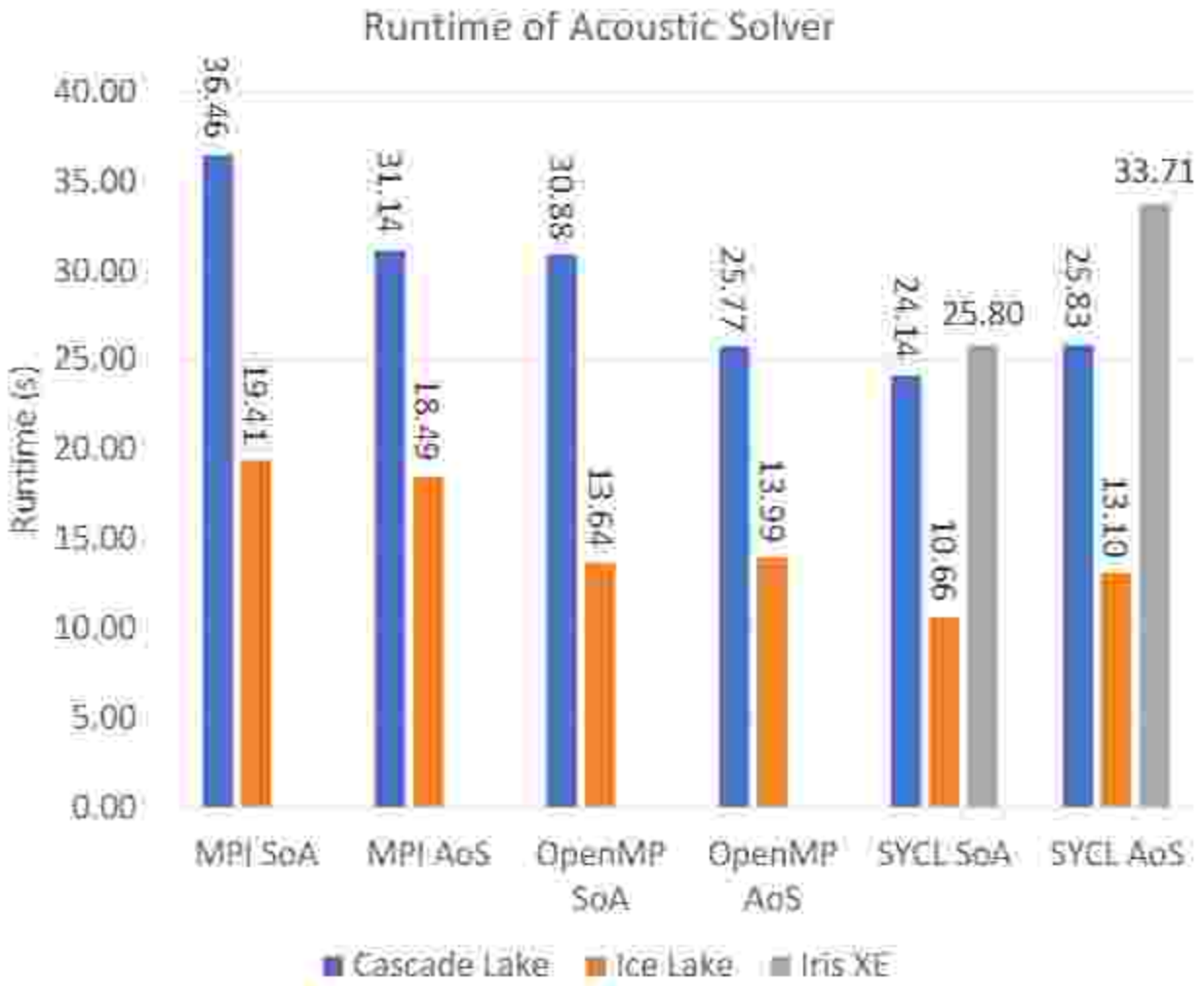
	MPI	OpenMP small	OpenMP large
DRAM traffic	332 GB	432 GB	685 GB

Cascade Lake

	MPI	MPI+SIMD	OpenMP	SYCL scalar	SYCL vec	SYCL GPU
Time (s)	9.275	4.55	10.0629	9.7632	22.1123	12.55
GFLOP	883.3	1185	883.23	897.18	850	1035
Self memory traffic (GB)	1099	2006	1076.2	889	3557	457
Instructions (1e9)	888	423	965	896	623	1024
Ratio of FLOP instructions	68%	22%	66%			

Acoustic wave solver

- 2 datasets, each with 6 values per gridpoint in key loop
  - Structure of Arrays vs. Array of Structs
- MPI
  - Extra explicit comms overhead – no latency hiding
- OpenMP & SYCL
  - Both vectorize on CPU
  - SYCL uses subgroup size of 64 for best results



OpenSBLI

- Single/Double precision (no double on Iris XE)
- SYCL reports vectorizing most expensive kernel on CPU, but runs scalar

Conclusions

- Extended OP2 and OPS to support SYCL
  - Various execution strategies for OP2 to avoid data races
- Benchmarking MG-CFD, Acoustic Wave solver, OpenSBLI
- Key challenge: understanding mapping from SYCL code (SIMT abstraction) to the hardware
  - Reasonably trivial for GPU architectures, where the hardware is a good fit for SIMT
  - Still problematic for SIMD architectures (such as CPUs)
    - OneAPI is quite aggressive about vectorization, and the sub-group API really helps with mapping to SIMD. Performance getting quite close
- SYCL a much more productive alternative to OpenCL, and performance is improving rapidly
  - But the challenges in terms of performance productivity remain

- Thanks to Intel for help through the OneAPI Innovator program