THE OXFORD PARALLEL DOMAIN SPECIFIC LIBRARIES FOR PERFORMANCE PORTABLE APPLICATION DEVELOPMENT

Gihan Mudalige
Royal Society Industry Fellow
Assistant Professor, Department of Computer Science, University of Warwick
g.mudalige@warwick.ac.uk

Joint work with:
Istvan Reguly, Attila Sulyok, Dániel Balogh (PPCU), Mike Giles (Oxford), Carlo Bertolli (IBM Research),
Sathya Jammy, Christian Jacobs and Neil Sandham (Southampton),
Paul Kelly, Adam Betts, Fabio Luporini (Imperial),
Richard Kirk (Warwick),
Rolls Royce plc., UCL, STFC and many more.
“The semiconductor industry threw the equivalent of a Hail Mary pass when it switched from making microprocessors run faster to putting more of them on a chip - doing so without any clear notion of how such devices would in general be programmed.”

David Patterson, University of California - Berkeley 2010
SINGLE THREAD SPEEDUP IS DEAD — MUST EXPLOIT PARALLELISM

Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp
DIVERSE HARDWARE LANDSCAPE

- **Traditional CPUs**
  - Intel, AMD, IBM, ARM
  - multi-core (> 20 currently)
  - Deep memory hierarchy (cache levels and RAM)
  - longer vector units (e.g. AVX-512)

- **GPUs**
  - NVIDIA, AMD
  - Many-core (> 1024 simpler SIMT cores)
  - CUDA cores, Tensor cores
  - Cache, Shared memory, HMB (3D stacked DRAM)

- **XeonPhi**
  - Many-core – based on simpler x86 cores
  - MCDRAM (3D stacked DRAM)
  - Have we seen the last of this?

- **FPGAs**
  - Various vendors / configurations
  - Low-level language

- **DSP Processors**
  - e.g. The Chinese Matrix2000 GPDSP accelerators (Top500 news 29/01/2018)

- **Quantum?**

- **Heterogeneous Processors**
  - NVIDIA Volta + POWER9 +NVLink
  - AMD APUs
SOFTWARE CHALLENGE – A MOVING TARGET

- Each new platform requires new performance tuning effort
  - Deeper cache hierarchies and/or shared-memory (non-coherent)
  - Multiple (heterogeneous) memory spaces (device memory/host memory)
  - Complex programming skills set needed to extract best performance on the newest architectures

- Not clear which architectural approach is likely to win in the long-term
  - Cannot be re-coding applications for each new type of architecture or parallel system
  - Nearly impossible for re-writing legacy codes

- Need to future-proof applications for their continued performance and portability
  - If not – significant loss of investment: applications will not be able to make use of emerging architectures
SOFTWARE CHALLENGE – DATA MOVEMENT

- Computing is cheap and massively parallel

- Data movement dominates performance costs
  - Bandwidth is the main bottleneck
  - Reduce communications to reduce energy

- Current programming environments
  - Not designed to take account of the cost of communication
  - Simply rely on the hardware cache coherency to virtualize data movement
  - Difficult to express data locality and affinity
  - Difficult to describe how to decompose and layout data in the memory

- Need easier ways to program for optimized data movement
  - Express information about data locality/affinity
  - A data-centric programming model?
The Level of Abstraction – Climbing the Analysis Hill and Generating Code

- Classical compiler have two halves: Analysis and Synthesis
- The higher you can get to (in analysis) the bigger the space of code synthesis possibilities

Adapted from: Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity? Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)
If you start at a lower level – climbing higher is a struggle
- Difficult to ensure optimizations are safe (e.g. data races, pointer aliasing)
- Sometimes, impossible to extract richer information (e.g. data partitioning/layouts, memory spaces)
- Limits the optimizations possible

Compounding the issue - the way code is written by (most) people will not be easy to analyse!

Adapted from: Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity? Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)
THE LEVEL OF ABSTRACTION

- If you can start higher
  - Results in a bigger space of code synthesis possibilities
  - Could they give the same (or better) performance as code written by hand?
  - Could these possibilities include targeting different (parallel) architectures?
- How can you start higher?

Adapted from: Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity? Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)
- Rise the abstraction to a **specific domain** of variability
- Concentrate on a narrower range (class) of computations
  - Computation-Communications skeletons - Structured-mesh, Unstructured-mesh, ... 7 Dwarfs [Colella 2004]
  - (higher) Numerical Method - PDEs, FFTs, Monte Carlo ...
  - (even higher) Specify application requirements, leaving implementation to select radically different solution approaches

Adapted from: *Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity?* Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)
**DOMAIN SPECIFIC ABSTRACTIONS**

- If you get the abstraction right, then:
  - Can isolate numerical methods from mapping to hardware
  - Can reuse a body of optimizations/code generation expertise/techniques for this class (or numerical method) to match target hardware

---

Adapted from: *Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity?* Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)
HOW DO WE RAISE THE LEVEL OF ABSTRACTION?

- **Domain Specific API**
  - Get application scientists to pose the solution using domain specific constructs – provided by the API
  - Handling data done *only* using API – contract with the user

- **Restrict writing code that is difficult (for the compiler) to reason about and optimize**
  - “OP2 and OPS are a straightjacket” – Mike Giles

- **Implementation of the API left to a lower level**
  - Target implementation to hardware – can use best optimizations
OPS for Multi-block Structured-Mesh Applications

```c
#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
//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);
//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");
// 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));
APPLICATION DEVELOPMENT

Application

OP2 / OPS Application (Fortran/C/C++ API)

Source-to-Source translator (Python / Clang-LLVM)

Modified Platform Specific OP2/OPS Application

Platform Specific Optimized Application Files

Conventional Compiler (e.g. icc, nvcc, pgcc, clang, XL, Cray) + compiler flags

Link

Mesh (hdf5)

Platform Specific Binary Executable

Op2/OPS Platform Specific Optimized Backend libraries

Sequential
Vectorized
CUDA
OpenMP
MPI
OpenCL

Hardware
CODE SYNTHESIS POSSIBILITIES

- Full responsibility for data layout and movement
  - Data Layout – SoA - AoS, distributed memory partitioning, local block partitioning
  - Data movement – MPI halo creation and exchange, host/device data movement (memory spaces)
  - Communication avoidance – computation vs communication balance, cache-blocking tiling

- Auto-parallelization
  - Target different hardware and programming models (SIMD, SIMT, SPMD, Task parallelism?)
  - Sophisticated orchestration of parallelizations – handle data races to match the context

- Load-balancing
  - Across heterogeneous processor architectures

- More?
  - Automatic checkpointing
  - Runtime compilation (JIT)
HANDLING DATA RACES

- Distributed memory parallelization
  - Mesh partitioning
  - Standard halo exchange methods
  - Redundant computation

- Single node – Inter-thread-block
  - Coloring
  - No two blocks of the same color update the same memory location

- Single node – Intra-thread block
  - Coloring
  - No two edges of the same color update the same node
  - Can also use atomics (performance ?)

Diagram:
- Proc 0
- Proc 1
- Threads 0 and 2 can run in parallel
DATA REUSE ON THE GPU – GLOBAL COLORING VS HIERARCHICAL COLORING

- **Global Coloring** - Colour the whole mesh assigned to a GPU
  - Do multiple kernel launches corresponding to the colours
  - No concurrent writes between threads in the same kernel
  - Poor data reuse, low cache-line utilization

- **Hierarchical Coloring** - Colour the blocks and the threads within a block
  - Form mini-partitions of the mesh (1) reorder mesh or (2) use a partitioner e.g. Metis
  - Load mini-partitions from GPU global memory in to GPU shared memory
  - Compute on the mini-partition using a CUDA thread-block

---

**BookLeaf** - `getacc_scatter`, 4M edges. (block sizes in brackets)

Reordering algorithms: No reordering (NR), GPS reordering and Partitioning (part.)

**LULESH** – `IntegrateStressForElems` – 4913k cells.

Original LULESH code handle data races by using large temporary array, then gathers

A.A. Sulyok,, G.D. Balogh, I.Z. Reguly, G.R. Mudalige. *Improving Locality of Unstructured Mesh Algorithms on GPUs*. (under review) JPDC 2018
void op_par_loop_res_calc(char const* name, op_set set, op_arg arg0, 
  op_arg arg1, op_arg arg2, op_arg arg3, op_arg arg4)
{
  int nargs = 5; op_arg args[5] = {arg0, arg1, arg2, arg3, arg4};
  int set_size = op_mpi_halo_exchanges(set, nargs, args);

  if (set->size > 0) {
    for (int n = 0; n < set_size; n++) {
      if (n == set->core_size) op_mpi_wait_all(nargs, args);

      int map0idx = arg0.map_data[n * arg0.map->dim + 0];
      int map1idx = arg0.map_data[n * arg0.map->dim + 1];
      int map2idx = arg3.map_data[n * arg3.map->dim + 0];
      int map3idx = arg3.map_data[n * arg3.map->dim + 1];

      res_calc(
        &((double*)arg0.data)[2 * map0idx],
        &((double*)arg0.data)[2 * map1idx],
        &((double*)arg2.data)[4 * n],
        &((double*)arg0.data)[4 * map2idx],
        &((double*)arg0.data)[4 * map3idx]);
    }
  }
}
SIMD vectorization

```c
#define SIMD_VEC 4

/* user function -- modified for vectorisation */
SUBROUTINE res_calc_vec(x1,x2,q1,q2,
    ad1,ad2,ref1,ref2,ixd)
   !x,y attributes vector :: res_calc_vec
   IMPLICIT NONE
   real(8), DIMENSION(SIMD_VEC,2), INTENT(IN) :: x1, X2
   real(8), DIMENSION(SIMD_VEC,4), INTENT(IN) :: q1, q2
   real(8), DIMENSION(SIMD_VEC,4), INTENT(IN) :: ad1, ad2
   real(8), DIMENSION(SIMD_VEC,4) :: ref1, ref2
   INTEGER(4): ixd
!
computation such as:
   ref(idx,1) = ref(idx,1) + &
   & q1(idx,1) = (x1(idx,1) - x2(idx,1))
END SUBROUTINE

SUBROUTINE op_wrap_res_calc( opDat1Loc, opDat3Loc, &
    opDat5Loc, opDat7Loc, opDat1Map, opDat3MapDim, &
    opDat5Map, opDat7Map, bottom,top)
   ...!
   real(8) dat1(SIMD_VEC,2), dat2(SIMD_VEC,2), &
   & dat3(SIMD_VEC,4), dat4(SIMD_VEC,4), dat6(SIMD_VEC,1),
   & dat8(SIMD_VEC,4)
!
loop SIMD number of iterations at a time
DO i1 = bottom, ((top-1)/SIMD_VEC)*SIMD_VEC + 1, SIMD_VEC
   loop over SIMD_VEC
!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!
   /!```
PERFORMANCE

- AWE - CloverLeaf2D
  - 2 x 8-core Intel Xeon E5-2680 2.70GHz (Sandy bridge)
  - Cray XC30 (ARCHER) - Strong scaling 15360^2 mesh
  - Weak scaling 2x 3840 x 3840 mesh per node
  - Cray XK7 (TITAN) - Strong scaling 15360^2 mesh
  - Weak scaling 2x 3840 x 3840 mesh per node

- OpenSBLI – TGV problem (Uni of Southampton)
  - 2 x NVIDIA Tesla K20c
  - Plot showing performance evaluation of algorithms with varying amounts of computational and memory intensity.


- AWE CloverLeaf – ~6k LoC originally written in Fortran 90
- Original code - Multiple manually parallelized versions
- Re-engineered to use OPS – uses OPS C/C++ API

- OpenSBLI – higher level Python-based framework capable of expanding a set of differential equations written in Einstein notation
- Generates OPS C/C++ API code
  - ARCHER (Cray XC30) 2×12-core Intel Xeon E5-2697 2.70GHz (Ivy Bridge)
  - Titan (Cray XK7) – AMD Opteron 6274 (16 core) + NVIDIA K20X
OP2 Hydra NASA Rotor 37, Multi-/Many-core performance
2xTesla K20c GPUs + 2x6-core Intel Xeon E5-2640 2.50GHz (2.5M edges, 20 iterations)

Rolls-Royce Hydra – ~50k LoC originally written in Fortran 77 (over 20 years ago), over 300 parallel loops
- Re-engineered to use OP2 – uses OP2 Fortran API
- Automatically Parallelized with OP2 – MPI + (OpenMP 3.0, CUDA, OpenACC), OpenMP 4.0 (experimental)
- Royal Society Industrial Fellowship (2018) - Moving OP2-Hydra to production

LOAD BALANCING

Hydra full hybrid execution (NASA Rotor 37 problem, 2.5M mesh edges)

Run time (seconds)

Partition size balance

2×Tesla K20c GPUs + 2×6-core Intel Xeon E5-2640 2.50 GHz
5GB/GPU Memory + 64GB RAM
CROSS-LOOP TECHNIQUES

- Loop descriptors and user contract allows to delay the execution of loops until API call to return data to user

- Now we have information about a sequence of loops to analyse/reason about together
  - Access descriptors provide precise dependence iteration-to-iteration information
  - Reason about a chain (DAG) of parallel loops at runtime

- Cross-loop optimizations
  - Cache-blocking Tiling
  - Communication avoidance
  - Automated checkpointing

- No changes to user code
  - OP2/OPS generates the required code
  - and carries-out the delayed-execution at runtime

- Applied to Production-grade applications
  - CloverLeaf 2D/3D mini-app: 150-600 loops,
  - OpenSBLI large scale CFD research code: 30-200 loops,
  - (TODO) Rolls-Royce Hydra CFD code: >300 loops
Data sets too large to fit on cache: limited data reuse

Improve reuse by considering multiple loops

Block iteration ranges of loops, reorganize them so that data accessed by a given block in the first loop nest stays in cache and gets accessed by blocks of subsequent loop nests

Need to make sure all data dependencies are satisfied

Parallelise within tiles

Single-socket Intel Xeon E5-2650 v3 (Haswell),
10 cores per socket – Hyperthreading On (20 threads total)
20 MB of L3 cache per socket
20 OpenMP threads (Run with `numactl` pinned to cores)
Intel Compilers 17.0.3 -fp-model fast + FMAs enabled

Tiling done over many loops spread across many compilation units
Many complex loops
Can’t be done by existing (compiler) technology
TILING – BEYOND 16GB

CloverLeaf 2D – KNL

CloverLeaf 3D – KNL

OpenSBLI – KNL

CloverLeaf 2D – P100

CloverLeaf 3D – P100

OpenSBLI – P100
TILING – COMMUNICATION AVOIDANCE / OVERLAPPING TILING

- Tiling in shared memory has a sequential dependency across tiles – there we only parallelize within tiles

- Apply an overlapped tiling approach over MPI
  - Replicate part of neighbour’s domain
  - Do redundant computation over them to satisfy dependencies

- Results in a deeper halo being exchanged over the chain of loops being tiled

- Exchange a larger message, but much less frequently
TILING – UNSTRUCTURED-MESH

- OP2 - Tiling on unstructured meshes
- Fabio Luporini, Paul Kelly (Imperial), Michelle Strout (Colorado State University) and others


ACCT - Average compute and communication time

Number of processes (Mesh elements)

\[ q - \text{polynomial order of the method} \]

\[ ~1.3x \text{ on large seismic application} \]

On 2x14-core Intel Xeon E5-2680 v4 2.40GHz cluster
AUTOMATIC-CHECKPOINTING

- Given a loop chain, reason about what data needs to be saved
- Create a checkpoint
  - Any data sets with READ : Saved
  - Anything data sets WRITE : Not saved
  - At any given loop, only a few datasets are touched: keep going and save/not save unseen datasets at later loops
- Use checkpointed data to automatic fast-forward after re-start
- Options on how to save the data
  - Parallel File I/O system
  - Each process writes its own checkpoint file
  - In memory checkpoints with redundancy
  - Local file system with redundancy or parallel file system
RUNTIME COMPILATION (JIT)

- Production application have large number of problem dependant parameters
  - E.g. NPDEs, mesh sizes, logicals selecting turbulence models
  - Unknown at compile time
  - If known, more optimizations can be applied - loop unrolling, vectorisation, or removing entire code paths

- Can code generate to `#include` parameters which can be then be compiled at runtime
- Currently (experimental) branch for OP2 – working with Rolls-Royce Hydra

---

### Performance Results

<table>
<thead>
<tr>
<th>System Configuration</th>
<th>Runtime (Seconds)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Xeon (SkyLake) – 2 x Sockets (total 24 cores)</td>
<td>NVIDIA P100</td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td>9.091</td>
<td></td>
</tr>
<tr>
<td>OpenMP</td>
<td>10.31</td>
<td></td>
</tr>
<tr>
<td>MPI+OpenMP</td>
<td>15.32</td>
<td></td>
</tr>
<tr>
<td>SIMD - AVX 512</td>
<td>8.48</td>
<td>1.28x</td>
</tr>
<tr>
<td>MPI + JIT</td>
<td>8.9404</td>
<td></td>
</tr>
<tr>
<td>MPI + NoJIT</td>
<td>8.48</td>
<td></td>
</tr>
<tr>
<td>CUDA</td>
<td>3.70</td>
<td></td>
</tr>
<tr>
<td>OpenACC</td>
<td>4.37</td>
<td></td>
</tr>
</tbody>
</table>

OP2-Hydra
NASA Rotor 37 problem
(2.8 million edges)
OTHER USERS, CODES AND PROJECTS - VOLNA
OTHER USERS, CODES AND PROJECTS – OPEN SBLI

Kelvin–Helmholtz instability

3D Taylor-Green vortex problem

Simulation of a three-dimensional Taylor-Green vortex using OpenSBLI

Christian T. Jacobs, Satya P. Jammy, Neil D. Sandham
University of Southampton, 2017
- ETH Zurich – BASEMENT code (Basic Simulation Environment for Computation of Environmental Flows and Natural Hazard Simulations)
  - Flood forecast and mitigation, River morphodynamics, Design of hydraulic structures
  - Finite volume discretisation, cell centred
  - Targeting OP2 for GPU and multi-core parallelisation

- STFC – HiLeMMS project (High-Level Mesoscale Modelling System):
  - high-level abstraction layer over OPS for the solution of the Lattice Boltzmann method
  - Adaptive mesh refinement - Chombo (Lawrence Berkeley National Labs)

- University of Nottingham – CFD code development with OPS
  - Simulation of Turbomachinery flows
  - Implicit solvers using OPS’s (experimental) Tridiagonal Solver API
Measuring Performance Portability

\[ \mathcal{P}(a, p, H) = \begin{cases} \frac{|H|}{\sum_{i \in H} e_i(a, p)} & \text{if } i \text{ is supported } \forall i \in H \\ 0 & \text{otherwise} \end{cases} \]

- $H$ - set of platforms
- $a$ - the application
- $p$ - the parameters for $a$
- $e$ - is the performance efficiency measure


TeaLeaf - Performance Portability
Xeon E5-2660 v4, KNL (MCDRAM) and a P100 card for the 4000x4000 mesh

<table>
<thead>
<tr>
<th>Version</th>
<th>Eff. (Xeon) (%)</th>
<th>Eff. (KNL) (%)</th>
<th>Eff. (P100) (%)</th>
<th>$P$(CPU ∪ GPU) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td>0.96</td>
<td>60.49</td>
<td>100.00</td>
<td>1.52</td>
</tr>
<tr>
<td>OPS</td>
<td>1.35</td>
<td>89.61</td>
<td>67.02</td>
<td>3.39</td>
</tr>
<tr>
<td>Kokkos</td>
<td>2.73</td>
<td>64.11</td>
<td>91.45</td>
<td>1.57</td>
</tr>
<tr>
<td>RAJA</td>
<td>0.91</td>
<td>53.13</td>
<td>80.73</td>
<td>1.60</td>
</tr>
</tbody>
</table>

(a) Architectural efficiency = 25.0%
(b) Application efficiency = 51.2%
CHALLENGES

- **Cost / Effort of Conversion**
  - Converting legacy code is time consuming (large code base, defunct 3rd party libs, Fortran 77 or older !)
  - Difficult to validate code – new code giving the same accurate scientific output?
  - Difficult to convince users to use new code
  - Incremental conversion – loop by loop
  - Simpler than CUDA, but more difficult than OpenACC/OpenMP
  - Automated conversion?

- **Code-generation**
  - Tools not entirely mature – currently source-to-source with Python
  - Pushing clang/LLVM source-to-source to do what we want - experimental
  - User kernel modification, Vectorization
  - Maintainable/long term source-to-source technologies (not the ROSE compiler !!)

- **Maintenance**
  - Currently purely done via academic and (small/short term) industrial funding
  - Long term funding – once established probably will not be different to any other classical library
  - Will require compiler expertise to maintain code generation tools
Future Work

- Tridiagonal solver capabilities for OPS
- Extending JIT compilation for all parallelizations - OP2 and OPS
- Adaptive Mesh Refinement in OPS
- Clang (libtooling) source-to-source translator/compiler for code generation
- Coupling with different parallel software, particularly over MPI (e.g. sparse linear solver, FFT)
- Multi-material data structures
- Coupling with visualization software
- Targeting Task-based parallelism – E.g. Legion (Stanford)
- Automatic conversion tool for Fortran
**Related Work**

- **FEniCS** - PDE solver package - [https://fenicsproject.org/](https://fenicsproject.org/)
- **Firedrake** - automated system for the portable solution of PDEs using the finite element method (FEM) - [https://www.firedrakeproject.org/](https://www.firedrakeproject.org/) (Imperial College and others)
- **Devito** - prototype DSL and code generation framework based on SymPy for the design of highly optimised finite difference kernels for use in inversion methods - [http://www.opesci.org/devito-public](http://www.opesci.org/devito-public) (Imperial College)
- **GungHO project** - Weather modelling codes - STFC and Metoffice
- **STELLA** – DSL for stencil codes, for solving PDEs - Metro Swiss
- **Kokkos** – C++ template library – SNL
- **RAJA** - C++ template libraries - LLNL
LESSONS LEARNT AND CONCLUSIONS

User application

How difficult is it to use / convert?  
Is the abstraction general enough?  

Domain Specific API

Source-to-source translation

Back-end library

Target-specific high-performance app

Does it deliver performance?  

CPU (AVX, SSE, OpenMP 3.0, OpenMP 4.0/4.5)

GPUs (CUDA, OpenCL, OpenACC, OpenMP 4.0/4.5)

Supercomputers (MPI + X)

human-readable and debuggable code?

Level of abstraction  
Cost of conversion  
Easy to debug  
Maintainability  
Easily extensible  
Human-readable / Validating  
Coupling with other libs and workflows

EC, Level 0.1
LESSONS LEARNT AND CONCLUSIONS

- Utilizing domain knowledge will expose things that the compiler does not know
  - Iterating over the same mesh many times without change
  - Mesh is partitioned and colourable

- Compilers are conservative
  - Force it to do what you know is right for your code!

- Let go of the conventional wisdom that higher abstraction will not deliver higher performance
  - Higher abstraction leads to a bigger space of code synthesis possibilities
  - We can automatically generate significantly better code than what (most) people can (reasonably) write
  - Do not destroy performance portability by (hand-) tuning at a very low level to a specific platform
OP2 was part-funded by the UK Technology Strategy Board and Rolls-Royce plc. through the SILOET project, and the UK EPSRC projects EP/I006079/1, EP/I00677X/1 on Multi-layered Abstractions for PDEs.

OPS was part-funded by the UK Engineering and Physical Sciences Research Council projects EP/K038494/1, EP/K038486/1, EP/K038451/1 and EP/K038567/1 on “Future-proof massively-parallel execution of multi-block applications” and EP/J010553/1 “Software for Emerging Architectures” (ASEArch) project.

CloverLeaf re-engineering research is funded by the UK AWE plc. under project “High-level Abstractions for Performance, Portability and Continuity of Scientific Software on Future Computing Systems”.

CloverLeaf development is supported by the UK AWE plc. under grants CDK0660 (The Production of Predictive Models for Future Computing Requirements) and CDK0724 (AWE Technical Outreach Programme).

Research was part-supported by the Janos Bolyai Research Scholarship of the Hungarian Academy of Sciences.

The research has been carried out within the project Thematic Research Cooperation Establishing Innovative Informatic and Info-communication Solutions, which has been supported by the European Union and co-financed by the European Social Fund under grant number EFOP-3.6.2-16-2017-00013.

OpenSBLI was part-funded by EPSRC grants EP/K038567/1 and EP/L000261/1, and European Commission H2020 grant 671571 “ExaFLOW: Enabling Exascale Fluid Dynamics Simulations.

Research presented used the UK National Supercomputing Service – ARCHER and resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

We acknowledge PRACE for awarding us access to resource Marconi based in Italy at Cineca.

This research utilized the systems provided by Advanced Research Computing (ARC) at the University of Oxford in carrying out this work.
DOWNLOADS AND MORE INFORMATION

- GitHub Repositories
  - OPS – https://github.com/OP-DSL/OPS

- Contact
  Gihan Mudalige (Warwick) - g.mudalige@warwick.ac.uk
  Istvan Reguly (PPCU – Hungary) - reguly.istvan@itk.ppke.hu