BRIDGING THE COMPLEXITY GAP IN EXASCALE SIMULATION SOFTWARE DEVELOPMENT THROUGH HIGH-LEVEL DOMAIN SPECIFIC LANGUAGES

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

Joint work with:
Istvan Reguly (PPCU), Mike Giles (Oxford), Carlo Bertolli (IBM Research), Paul Kelly, (Imperial), Andrew Owenson, Arun Prabhakar and others ay the HPSC group (Warwick)
David Lusher, Sathya Jammy, Christian Jacobs and Neil Sandham (Southampton),
Rolls Royce plc., NAG, UCL, STFC, IBM and many more.

28th October 2020 - Workshop on Domain Specific Languages in High Performance Computing
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
“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**

---

**42 Years of Processor Data**

- **First Reconfigurable Wave**
  - Adaptive Silicon, Elxent, Tricend, Morphics, Chameleon Systems, Quicksilver Technology, Mailstar

- **Free Lunch is Over**
  - H. Sutter

- **No Silver Bullet**
  - F. Brooks

- **A New Golden Age**
  - Hennessy/Patterson

**Transistors** (1000s)

**Single-Thread Performance** (SpecINT x 10^3)

**Frequency** (MHz)

**Typical Power** (Watts)

**Number of Logical Cores**

---

Hennessy and Patterson, Turing Lecture 2018, overlaid over “42 Years of Processors Data”
https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/; “First Wave” added by Les Wilson, Frank Schirrmieister

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

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

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

- **Heterogeneous Processors**
  - NVIDIA Volta + POWER9 + NVLink
  - AMD APUs

- **FPGAs**
  - Various vendors / configurations
  - Low-level language
  - Significant energy savings

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

- **Quantum?**
Its “Exascale” Jim but not as we know it!

EXASCALE MAY BE ACHIEVED IN MANY DIFFERENT (ARCHITECTURAL) WAYS
OpenMP, SIMD, CUDA, OpenCL, OpenMP4.0, OpenACC, SYCL/OneAPI, ROCm, MPI, PGAS
Task-based (?) and more …

- Open standards (e.g. OpenMP, OpenMP4) – so far have not been agile to catch up with changing architectures
- Proprietary models (CUDA, OpenACC, ROCm, OneAPI ?) – restricted to narrow vendor specific hardware
- Need different code-paths/parallelization schemes to get the best performance
  - E.g. Coloring vs atomics vs SIMD vs MPI vs Cache-blocking tiling for unstructured mesh class of applications
- What about legacy codes? There is a lot of FORTRAN code out there!

But .. Even more diverse ways to programming them!
SOFTWARE CHALLENGE – A MOVING TARGET

- What would an Exa-scale machine architecturally look like?

- Each new platform requires new performance tuning effort
  - Deeper memory/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
Motivation
Raising the Level of Abstraction
Oxford Parallel Libraries – OP2 and OPS
Codes and Projects using OP2/OPS
Challenges
Future Plans
Lessons Learnt and Conclusions
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)
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)
DOMAINE SPECIFIC ABSTRACTIONS

- 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)
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
  - Build in safe guards so that user cannot write bad code!

- Implementation of the API left to a lower level
  - Target implementation to hardware – can use best optimizations
How do we raise the level of abstraction?

- 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

- Provide 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
//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
def 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));
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);
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

Mesh (hdf5)

Platform Specific Binary Executable

OP2/OPS Platform Specific Optimized Backend libraries

Link

Sequential

Vectorized

CUDA

OpenMP

MPI

OpenCL

SYCL

Hardware
CODE SYNTHESIS POSSIBILITIES

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

- **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

- **Load-balancing**
  - Across heterogeneous processor architectures

- **More ..**
  - Automatic checkpointing
  - Runtime compilation (JIT)
  - Insitu visualization?
  - Uncertainty quantification?
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 colour update the same node
  - Can also use atomics (performance?)

![Diagram showing distributed memory parallelization and data race handling](image)
inline void compute_flux_edge_kernel(
  const double *variables_a, const double *variables_b,
  const double *edge_weight, double *fluxes_a,
  double *fluxes_b) {
  double ewt = sqrt(edge_weight[0]*edge_weight[0] +
                      edge_weight[1]*edge_weight[1] +
                      edge_weight[2]*edge_weight[2]);
  double p_b = variables_b[VAR_DENSITY];
  ...
  ...
}

op_par_loop(compute_flux_edge_kernel,
  "compute_flux_edge_kernel",  op_edges,
  op_arg_dat(vars, 0, en, 5, "double", OP_READ),
  op_arg_dat(vars, 1, en, 5, "double", OP_READ),
  op_arg_dat(edwgts, -1, OP_ID, 3, "double", OP_READ),
  op_arg_dat(fluxes, 0, en, 5, "double", OP_INC),
  op_arg_dat(fluxes, 1, en, 5, "double", OP_INC));
**OP2 - Generated Code for the CPU**

```c
void op_par_loop_compute_flux_edge_kernel(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};

    // MPI halo exchange -- start
    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);

            // prepare indirect accesses
            int map0idx = arg0.map_data[n * arg0.map->dim + 0];
            int map1idx = arg0.map_data[n * arg0.map->dim + 1];

            // set pointers and call elemental kernel
            compute_flux_edge_kernel(
                (double*)arg0.data)[5 * map0idx],
                (double*)arg0.data)[5 * map1idx],
                (double*)arg2.data)[3 * n],
                (double*)arg3.data)[5 * map0idx],
                (double*)arg3.data)[5 * map1idx]);
        }
    }
}
```
OP2 - SIMD VECTORIZATION

```c
inline void compute_flux_edge_kernel{
    const double *variables_a, const double *variables_b,
    const double *edge_weight, double *fluxes_a,
    double *fluxes_b) {
    double ewt = sqrt(edge_weight[0] * edge_weight[0] +
                      edge_weight[1] * edge_weight[1] +
                      edge_weight[2] * edge_weight[2]);
    double p_d = variables_b[VAR_DENSITY];
    ...
    ...
}

void op_par_loop_compute_flux_edge_kernel(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 arg[5] = {arg0, arg1, arg2, arg3, arg4};
    //create aligned pointers for data
    ALIGN_2_DOUBLE const double __restrict__ ptre =
        (double *)arg0.data;
    //assume aligned (ptr0, double_ALIGN);
    //MPI halo exchange -- start
    int set_size = op_mpi_halo_exchanges(set, nargs, args);
    if (exec_size > 0) {
        #ifdef VECTORIZ
        #pragma novector
        for (int h = 0; h < (exec_size/SIMD_VEC) * SIMD_VEC;n++SIMD_VEC) {
            if (n+SIMD_VEC >= set->core_size)
                op_mpi_wait_all(nargs, args);
            ALIGN_DOUBLE double data0[SIMD_VEC];
            ALIGN_DOUBLE double data1[SIMD_VEC];
            ALIGN_DOUBLE double data2[SIMD_VEC];
            ALIGN_DOUBLE double data3[SIMD_VEC];
            ALIGN_DOUBLE double data4[SIMD_VEC];
        }
        #endif
    }

    //pragma omp simd simaline(SIMD_VEC)
    for (int i=0; i<SIMD_VEC; i++) {
        int idx0_5 = 5*arg0.map_data[(n+1)*arg0.map->dim0+0];
        int idx1_3 = 5*arg0.map_data[(n+1)*arg0.map->dim1+1];
        data0[0][1] = (ptr0)[idx0_5 + 0];
        data1[1][1] = (ptr0)[idx0_5 + 1];
        data2[2][1] = (ptr0)[idx0_5 + 2];
        data3[3][1] = (ptr0)[idx0_5 + 3];
        data4[4][1] = (ptr0)[idx0_5 + 4];
        ...
        ...
    }
}
```

OP2 - SYCL PARALLIZATION - GLOBAL COLORING

/* Cast OP2 data, maps and coloring plan as SYCL buffers */
arg0_buffer = ...; // vars - indirectly accessed
arg3_buffer = ...; // fluxes - indirectly accessed
map0_buffer = ...; // on - mapping table
arg2_buffer = ...; // edwghts - directly accessed

col_reord_buffer = ...; // coloring array

for (int col=0; col<Plan->ncolors; col++) { // for each color
    int start = Plan->col_offsets[0][col];
    int end = Plan->col_offsets[0][col+1];
    int nblocks = (end - start - 1)/nthread + 1;

    // enqueue arguments and elemental kernel
    op2_queue->submit([&](cl::sycl::handler& cgh) {
        ind_arg0 = (*arg0_buffer) ...; // enqueue vars
        ind_arg1 = (*arg3_buffer) ...; // enqueue fluxes
        opDat0Map = (*map0_buffer) ...; // enqueue mapping en
        arg2 = (*arg2_buffer) ...; // enqueue edwghts

        // enqueue coloring array
        col_reord = (*col_reord_buffer) ...; // enqueue any global constants used in the kernel ...

        // elemental kernel function as lambda - enqueue it
        auto compute_flux_edge_kernel_gpu = [=](
            const double *var_a, const double *var_b,
            const double *edwghts, double *fluxes_a,
            double *fluxes_b)
            { .../*body of kernel*/ ... });
    });
}

// setup kernel work items
auto kern = [=](cl::sycl::item<1> item) {
    int tid = item.get_id(0);
    if (tid + start < end) {
        int n = col_reord[tid + start];
        int map0idx; int maplidx;

        // get the indirect index via mapping
        map0idx = opDat0Map[n+set_size*0];
        maplidx = opDat0Map[n+set_size*1];

        // user-supplied kernel call
        compute_flux_edge_kernel_gpu(
            &ind_arg0[map0idx*5], &ind_arg0[maplidx*5],
            &arg2[n*3],
            &ind_arg1[map0idx*5], &ind_arg1[maplidx*5]);
    }
};

// execute kernel
cl::sycl::parallel_for
<class compute_flux_edge_kernel>(
    cl::sycl::range<1>(nthread * nblocks), kern);
}); // end of enqueue arguments and elemental kernel
```cpp
// Setup kernel work items
auto kern = [=](cl::sycl::nd_item<l> item) {
    // Local variables for holding indirect increments
    double arg3_l[5], arg4_l[5];
    for (int d = 0; d < 5; d++) { arg3_l[d] = 0.0; }
    for (int d = 0; d < 5; d++) { arg4_l[d] = 0.0; }
    int tid = item.get_global_linear_id();
    if (tid + start < end) {
        int n = tid + start;
        map0idx; int maplidx;
        // Get the indirect index via mapping
        map0idx = opDat0Map[n + set_size * 0];
        maplidx = opDat0Map[n + set_size + 1];
        // Elemental kernel call
        compute_flux_edge_kernel_gpu(
            &ind_arg0[map0idx+5],&ind_arg0[maplidx+5],
            &arg2[n+3],
            arg3_l, arg4_l);
        // Apply indirect increments using atomics
        (cl::sycl::atomic<double>* a =
            (cl::sycl::global_ptr<
            &ind_arg1[0+map0idx+5])));
        a.fetch_add(arg3_l[0]);
        ...
        (cl::sycl::atomic<double>* a =
            (cl::sycl::global_ptr<
            &ind_arg1[4+maplidx+5]));
        a.fetch_add(arg4_l[4]));
    }
    // Execute kernel
    cgh.parallel_for
    <class compute_flux_edge_kernel>
    (cl::sycl::nd_range<l>(nthread, nblocks, nthread), kern);
}; // end of enqueue arguments and elemental kernel
```
**PERFORMANCE**

**Strong Scaling (2.5M mesh edges)**
- OP2 Hydra NASA Rotor 37, Scaling on HECToR (MPI, MPI+OpenMP) and Jade (MPI+CUDA) : 20 iterations
- HECToR (Cray XE6) – 2 x 16-core AMD Opteron 6276 (Interlagos) 2.3GHz
- Jade (NVIDIA GPU Cluster) – 2 x Tesla K20m GPUs + Intel Xeon E5-1650 3.2GHz

**Weak Scaling (0.5M mesh edges per node)**
- OP2 Hydra NASA Rotor 37, Scaling on HECToR (MPI, MPI+OpenMP) and Jade (MPI+CUDA) : 20 iterations
- HECToR (Cray XE6) – 2 x 16-core AMD Opteron 6276 (Interlagos) 2.3GHz
- Jade (NVIDIA GPU Cluster) – 2 x Tesla K20c GPUs + 2x6-core Intel Xeon E5-2640 2.50GHz

- 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
PERFORMANCE

- **AWE - CloverLeaf2D**
  2 x 8-core Intel Xeon E5-2680 2.70GHz (Sandy bridge)

- **Cray XC30 (ARCHER) - Strong scaling 15360² mesh**

- **Cray XK7 (TITAN) - Strong scaling 15360² mesh**

- **2 x NVIDIA Tesla K20c**

- **Weak scaling 2x 3840 x 3840 mesh per node**

- **OpenSBLI – TGV problem (Uni of Southampton)**

- **Cray XC30  (ARCHER) 2×12-core Intel Xeon E5-2697 2.70GHz (Ivy Bridge)**

- **Cray XK7 (TITAN) – AMD Opteron 6274 (16 core) + NVIDIA K20X**

- **AWE CloverLeaf – ~6k LoC originally written in Fortran 90**

- **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

---

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,
  - 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

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

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 – 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
- Lots of work by Fabio Luporini, Paul Kelly (Imperial), Michelle Strout (Colorado State University)
- How to apply these exotic optimizations to real-world codes?

Work with S. Ekanayake (see his talk later)
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

![Strong scaling (384^3) on Titan](image1)

![Weak scaling (192^3) on Titan](image2)
USERS, CODES AND PROJECTS - VOLNA
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
Production CFD suite

Hydra (MPI) vs OP2-Hydra (MPI, MPI+OpenMP, and CUDA)
2 x socket Xeon Gold 6252 CPUs and an NVIDIA V100 GPU
ASIMOV (ADVANCED SIMULATION AND MODELLING OF ENGINEERING SYSTEMS)

- Coupling simulations (CFD-to-CFD)

- Simulate an entire aircraft jet engine in operation at very high fidelity - virtual certification
  - Couple CFD-to-Combustion
  - Couple CFD-to-FEM etc.

JM76 Coupler with Hydra and Op2-Hydra, 2 x socket Xeon Gold 6252 CPUs
The Numerical Algorithms Group (NAG) developing a new PDE solver package based on OPS

Initially targeting financial industry with further plans for Oil and Gas

Initial phase OPS only, latter phases may include OP2

Better software engineering / testing / maintenance
OTHER USERS

- **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 Tridiagonal Solver API
What about OneAPI/SYCL?

Work with Archie Powell, Istvan Reguly, Andrew Owenson (see Archie’s talk on this later)
WHAT ABOUT ONEAPI/SYCL?

Work with Archie Powell, Istvan Reguly, Andrew Owenson (see Archie’s talk on this later)
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?
- What code should I certify? High-level code/generated code?
- Difficult to convince users to use new code

Incremental conversion – loop by loop
- Simpler than CUDA, but more difficult than OpenACC/OpenMP
- Automated conversion?
## CHALLENGES – CODE-GENERATION

- **Tools not entirely mature**
  - Currently source-to-source with Python
  - Pushing clang/LLVM source-to-source to do what we want
  - What about Fortran - may be F18/Flang?

- **Elemental kernel modification – Vectorization**

- **Code-generation for more exotic architectures – FPGAs?**
  - Large design space
  - Complex source transformations

- **Maintainable/long term source-to-source technologies**
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

- What DSL to choose?
Future Work

- Implicit Solvers - Multi-diagonal solver capabilities for OPS
- Automatic code generation from OPS/OP2 to FPGAs
- Clang (libtooling) source-to-source translator/compiler for code generation
- Adaptive Mesh Refinement (Difficult)
- Coupling with different parallel software, particularly over MPI (e.g. sparse linear solver, FFT)
- Multi-material data structures
- Insitu visualization
- Targeting Task-based parallelism – E.g. Legion (Stanford)
- Code-generation tools / techniques 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)

- PyFR - Python based framework for solving advection-diffusion type problems on streaming architectures using the Flux Reconstruction approach - [http://www.pyfr.org/](http://www.pyfr.org/) (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

Domain Specific API

- Source-to-source translation
- Back-end library

Target-specific high-performance app

- CPUs (AVX, SIMD, OpenMP 3.0, OpenMP 4.0/4.5, SYCL)
- GPUs (CUDA, OpenCL, OpenACC, OpenMP 4.0/4.5, SYCL)
- Supercomputers (MPI + X)

How difficult is it to use / convert?

Is the abstraction general enough?

Does it deliver performance?

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

human-readable and debuggable code?
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
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

“This is the way”
DOWNLOADS AND MORE INFORMATION

- GitHub Repositories
  - OP-DSL Webpage - [https://op-dsl.github.io/](https://op-dsl.github.io/)

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

- 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.
- Gihan Mudalige was supported by the Royal Society Industrial Fellowship Scheme (INF/R1/180012)
- 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.