MULTI-LAYERED ABSTRACTIONS FOR PERFORMANCE PORTABILITY - LESSONS LEARNT AND CHALLENGES

Gihan Mudalige
Royal Society Industry Fellow
Reader (Associate Professor) in High Performance Computing
Department of Computer Science, University of Warwick
g.mudalige@warwick.ac.uk

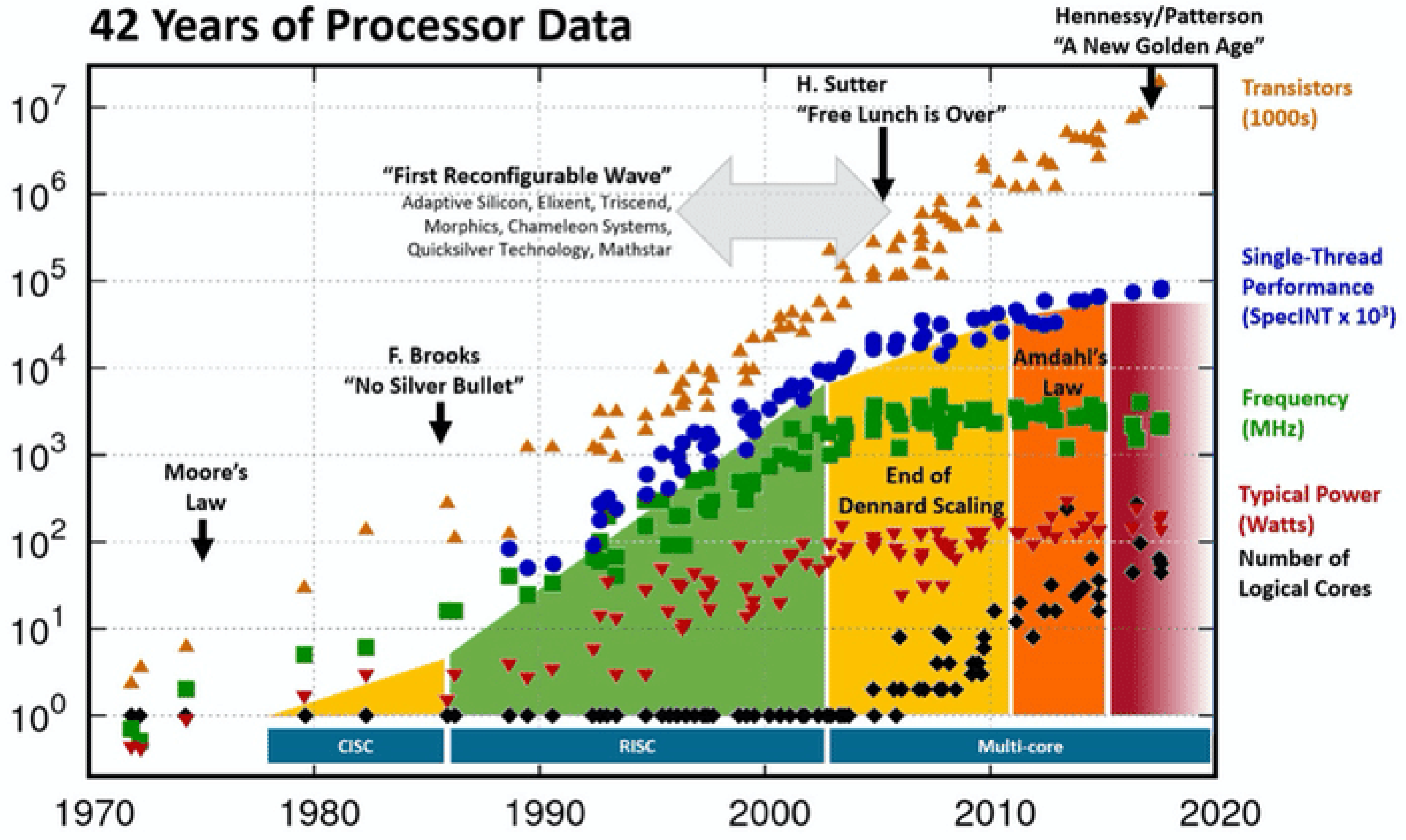
Joint work with:

Istvan Reguly @ PPCU,
Kamalavasan Kamalakannan, Arun Prabhakar and others at the HPSC group @ Warwick
Neil Sandham and team @ Southampton, Dario Amirante @ Surrey,
Mike Giles @ Oxford, Paul Kelly and many more @ Imperial,
Rolls-Royce plc., NAG, UCL, STFC, IBM and many more industrial and academic collaborators.

10th December 2021 – Computing Insight UK 2021

WARWICK
THE UNIVERSITY OF WARWICK

Single thread speedup is dead – Must exploit parallelism



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 Schirrmeister
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 Hail Mary pass !

http://www.theemike.com/mikes-free-football-comic-book-hail-mary-pass/

“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

Diverse Hardware Landscape – Compounded by the Race to Exascale !

- 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 (A100), AMD (MI200) , Intel (Xe GPUs)
  - Many-core (> 1024 simpler SIMT cores)
  - CUDA cores, Tensor cores
  - Cache, Shared memory, HBM (3D stacked DRAM)
- Heterogeneous Processors
  - Different core architectures over the past few years
  - ARM big.LITTLE
  - NVIDIA Grace.Hopper
- XeonPhi (discontinued)
  - Many-core – based on simpler x86 cores
  - MCDRAM (3D stacked DRAM)
- FPGAs
  - Dominated by Xilinx and Intel
  - Various configurations
  - Low-level language / HLS tools for programming
  - Significant energy savings
- DSP Processors
  - Phytium / The Chinese Matrix2000 GPDSP accelerator (Yet to be announced Chinese Exascale system ?)
- TPUs, IPUs ….
- Quantum ?

But .. Even more diverse ways to programming them !

OpenMP,
SIMD,
CUDA, OpenCL,
OpenMP4.0, OpenACC,
SYCL/OneAPI,
HIP/ROCm,
MPI, PGAS
Task-based (e.g Legion)
and others ….

- ❑ Open standards (e.g OpenMP, SYCL) – 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 !

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

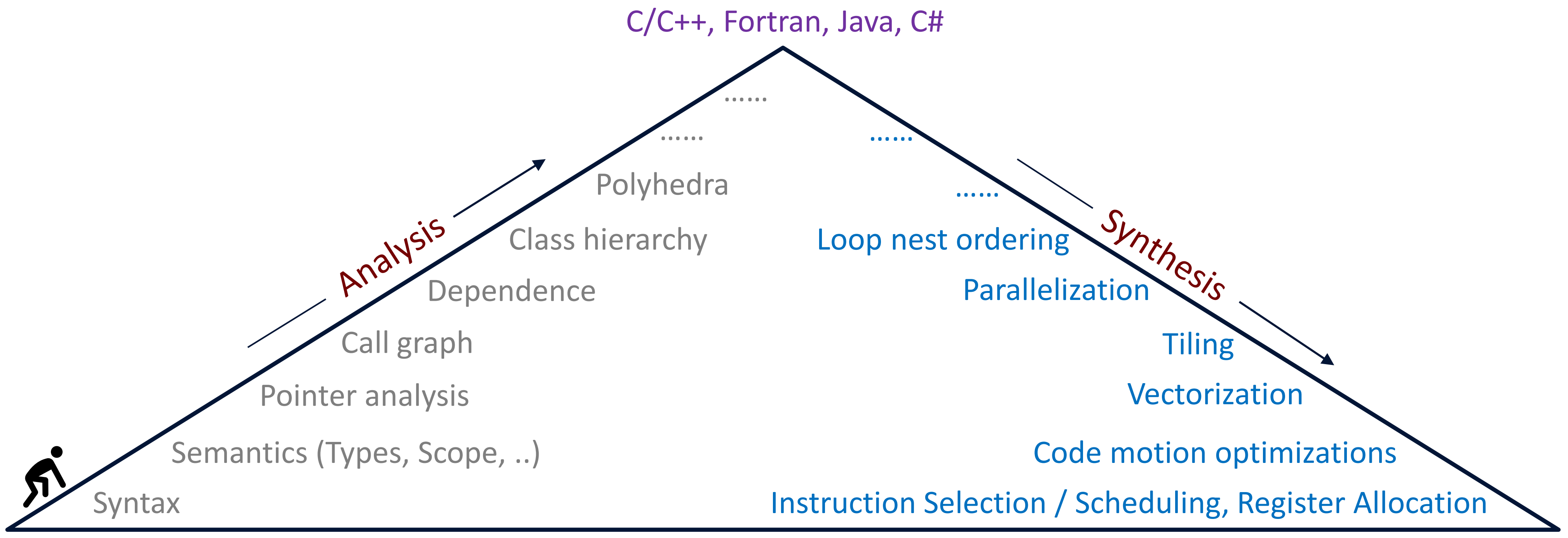
- Not clear which architectural approach is likely to *win* in the long-term
  - Cannot be re-coding applications for each *new* type of architecture or parallel system
  - Nearly impossible for re-writing legacy codes

- 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

OUTLINE

- ❑ Motivation ✓
- ❑ Raising the Level of Abstraction
- ❑ OP-DSLs
- ❑ Codes and Projects using OP2/OPS
- ❑ Future Plans – ExCALIBUR
- ❑ Challenges and Lessons Learnt
- ❑ Conclusions

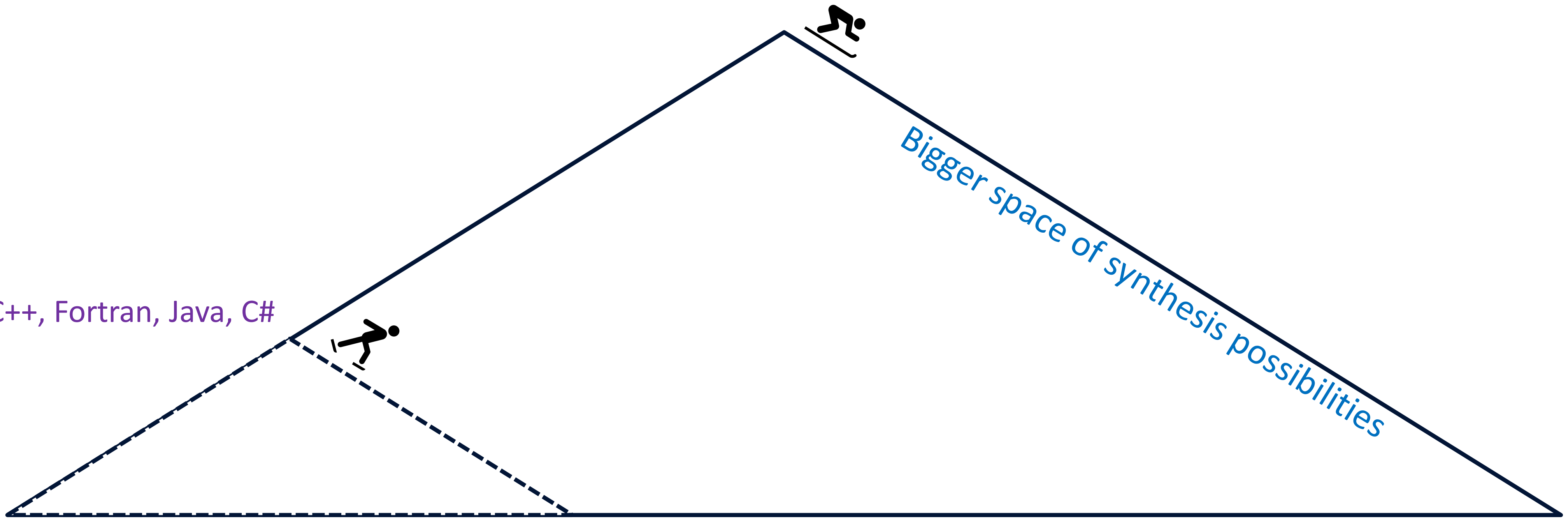
The Level of Abstraction - Climbing the Analysis Hill and Generating Code



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)

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

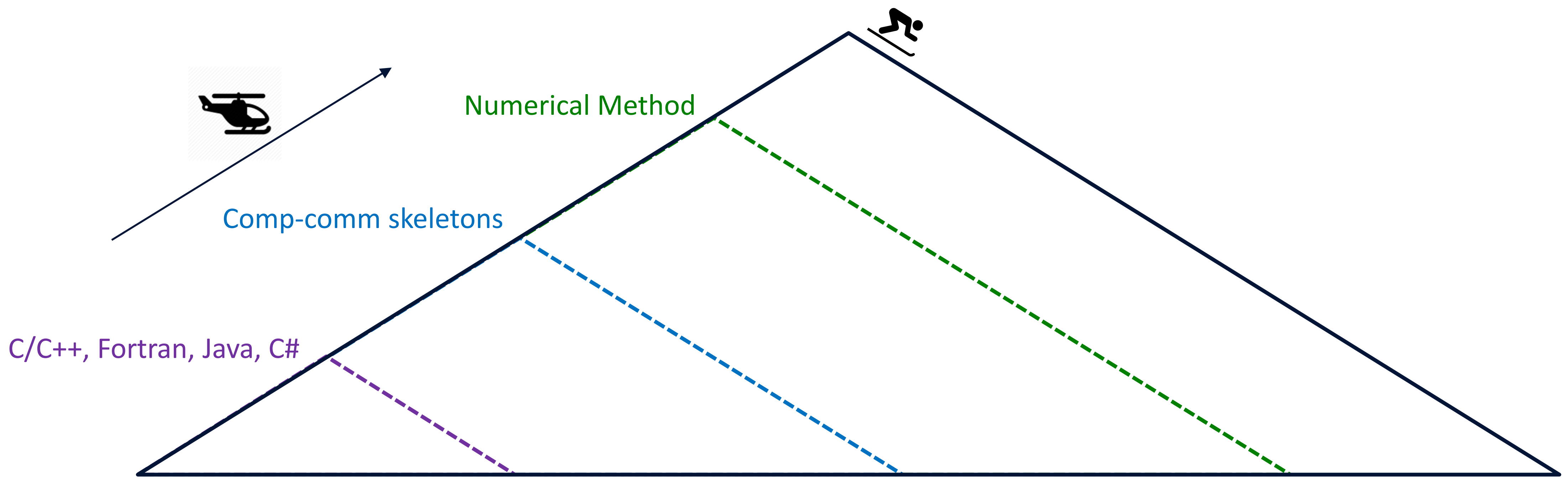
The Level of Abstraction



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 ?

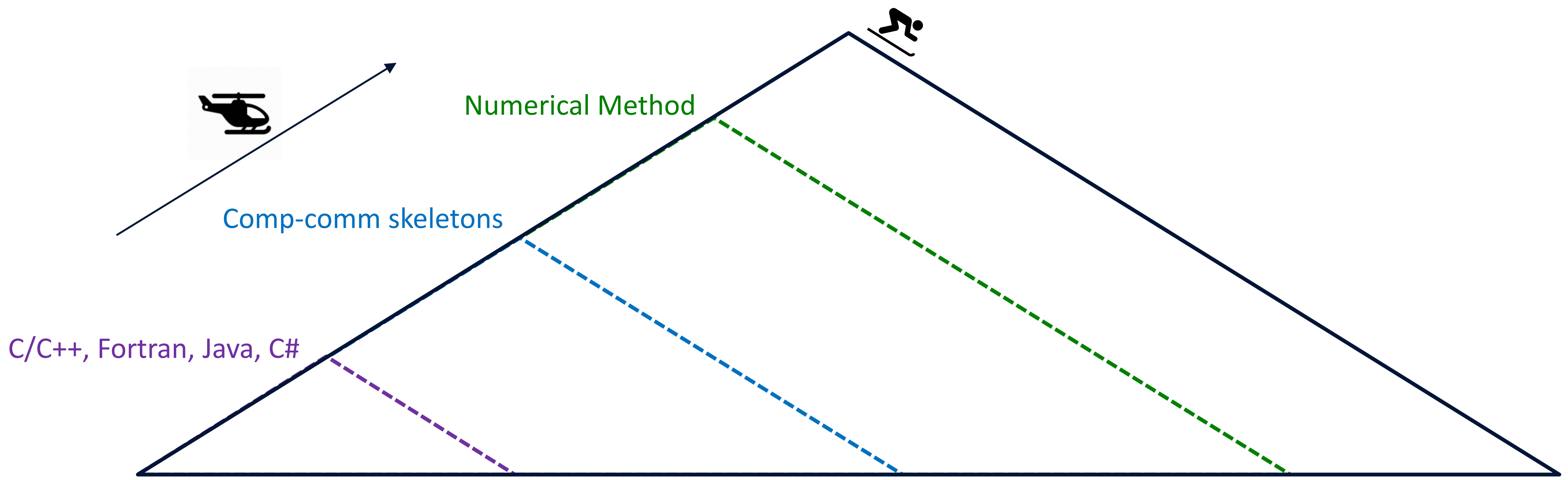
Domain Specific Abstractions



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

Domain Specific Abstractions



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

OP-DSL

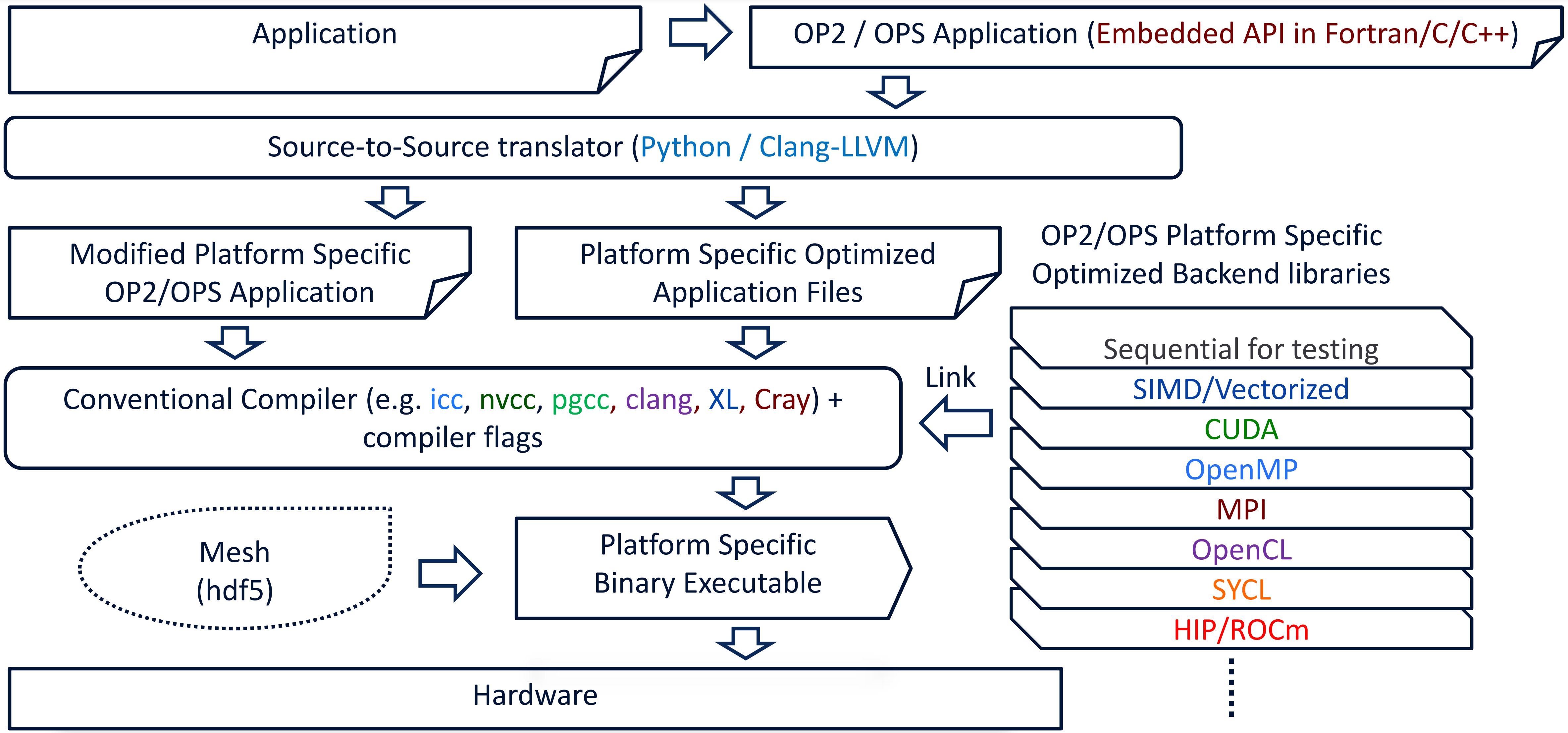
- Separation of Concerns (… back in 2010 !)
  - Specify the problem – not the implementation
  - Leverage the best implementation for the target context
  - Can be many contexts - hardware, programming model, parameters etc.

- 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

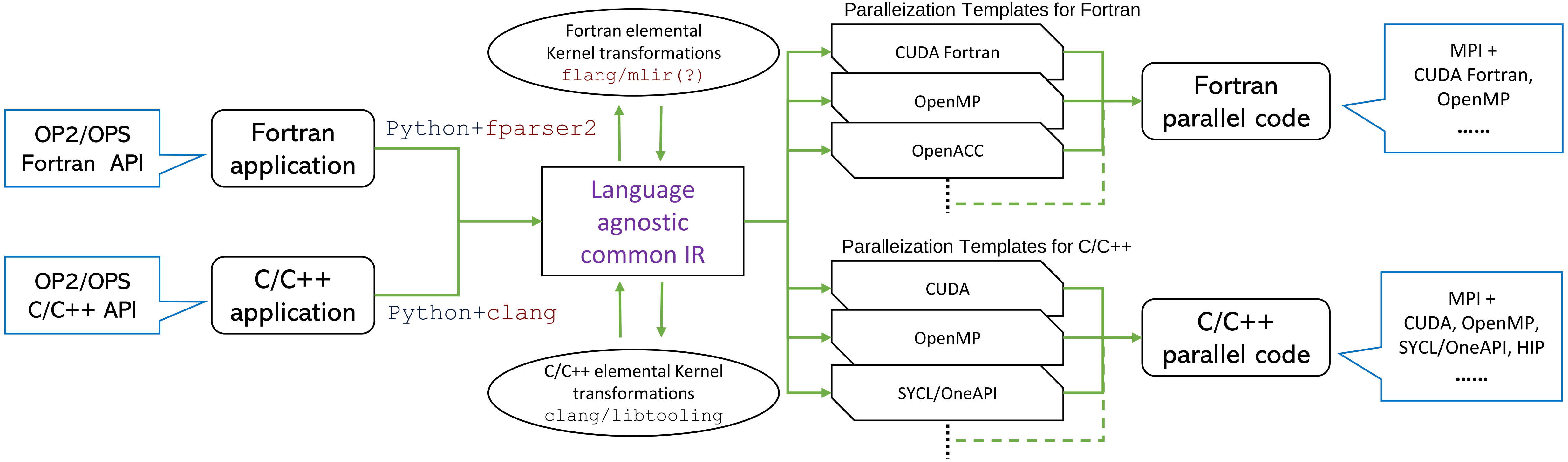
- Restrict writing code that is difficult (for the compiler) to reason about and optimize
  - *"OP2 and OPS are a straightjacket"* – Mike Giles
  - Build in safe guards so that user cannot write bad code !

- Implementation of the API left to a lower level
  - Target implementation to hardware – can use best optimizations
  - Automatically generate implementation from specification for the context
  - Exploit domain knowledge for better optimisations - reuse what we know is best for each context

Application Development


Application
OP2 / OPS Application (Embedded API in Fortran/C/C++)
Source-to-Source translator (Python / Clang-LLVM)
Modified Platform Specific OP2/OPS Application
Platform Specific Optimized Application Files
OP2/OPS Platform Specific Optimized Backend libraries
Sequential for testing
SIMD/Vectorized
CUDA
OpenMP
MPI
OpenCL
SYCL
HIP/ROCm
Link
Conventional Compiler (e.g. icc, nvcc, pgcc, clang, XL, Cray) + compiler flags
Mesh (hdf5)
Platform Specific Binary Executable
Hardware

OP2/OPS Code Generation



OP2/OPS Fortran API
Fortran application
OP2/OPS C/C++ API
C/C++ application
Python+fparser2
Python+clang
Fortran elemental Kernel transformations flang/mlir(?)
Language agnostic common IR
C/C++ elemental Kernel transformations clang/libtooling
Paralleization Templates for Fortran
CUDA Fortran
OpenMP
OpenACC
Paralleization Templates for C/C++
CUDA
OpenMP
SYCL/OneAPI
Fortran parallel code
C/C++ parallel code
MPI + CUDA Fortran, OpenMP ……
MPI + CUDA, OpenMP, SYCL/OneAPI, HIP ……

- ❑ Simplest Code generation / translation
  - Intermediate representation is simply the loop descriptions + elemental kernels
  - Generated parallel code can be viewed and understood by a human !

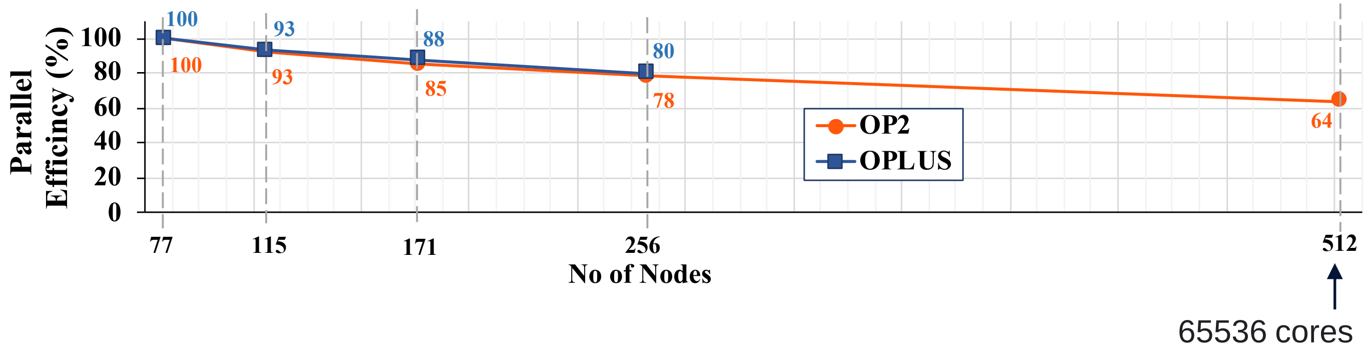
- ❑ Multi-layered – no opaque / black box layers
- ❑ Built with well supported / long-term technologies - Python, Clang/libtooling, [flang, mlir]

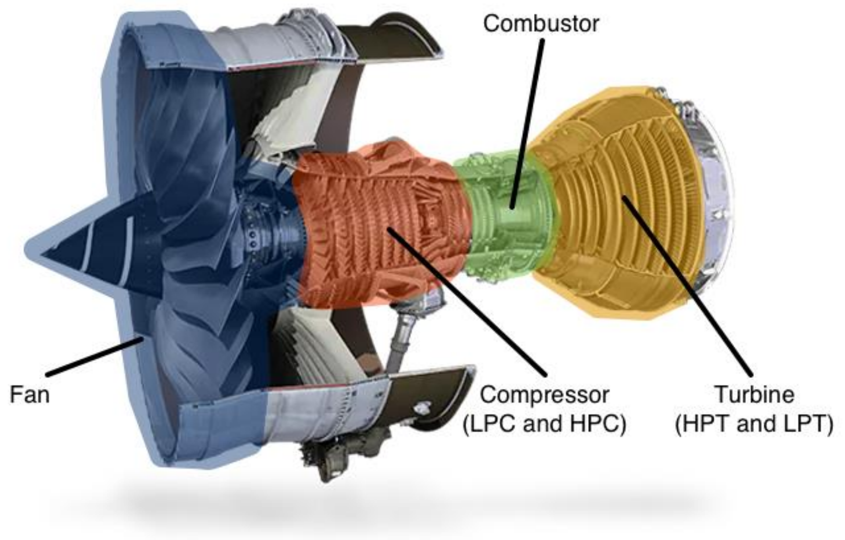
Code Synthesis possibilities

- 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 ?

Production Apps - EPSRC Prosperity Partnership – ASIMoV Project

- Virtual certification of Gas Turbine Engines
- Main consortium with partners – EPCC, Warwick, Oxford, Cambridge, Bristol and Rolls-Royce plc.





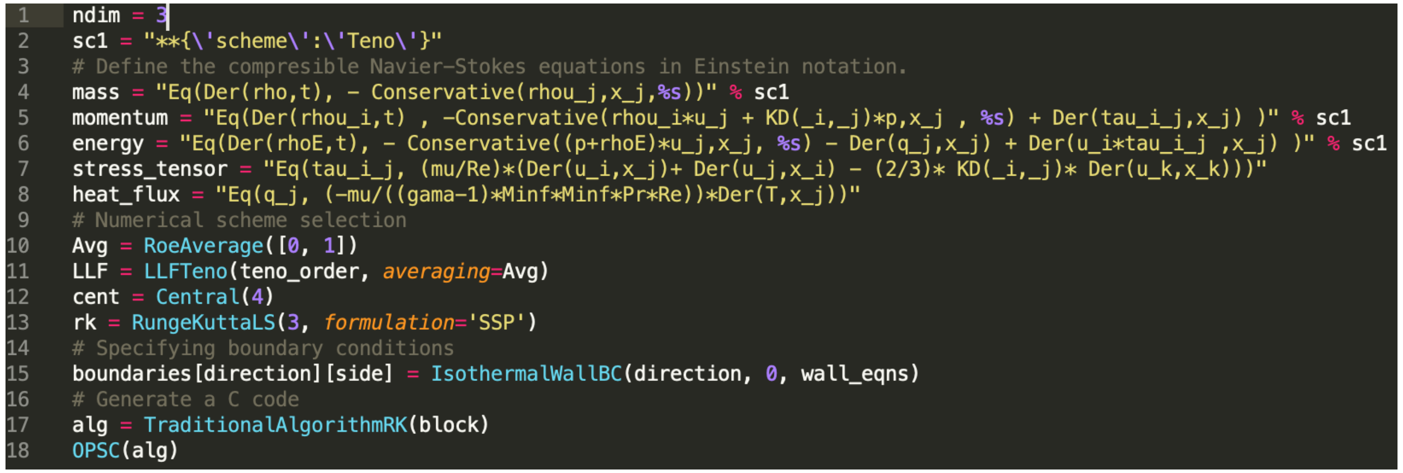
Grand Challenge : Rig250 – Compressor from DLR

- 1-10 passage full annulus
- Aim: 1 Rev in 24 hours (ARCHER2)
- Achieved : 1 Rev in 11 hours (ARCHER2 32k cores / 256 nodes)
- Predicted : 1 Rev in less than 5 hours (408 V100 GPUs / ~100 nodes)
  1 Rev in less than 6 hours (168 A100 GPUs / ~ 64 nodes)

less than ½ or ¼ th of the ARCHER2 machine size

Production Apps – OpenSBLI (Uni. of Southampton)

- Compressible Navier-Stockes solver
  - With shock capturing WENO/TENO
  - 4th order Finite Difference
  - Single/double precision
- OpenSBLI is a Python framework
  - Write equations in SymPy expressions
  - OPS code generated

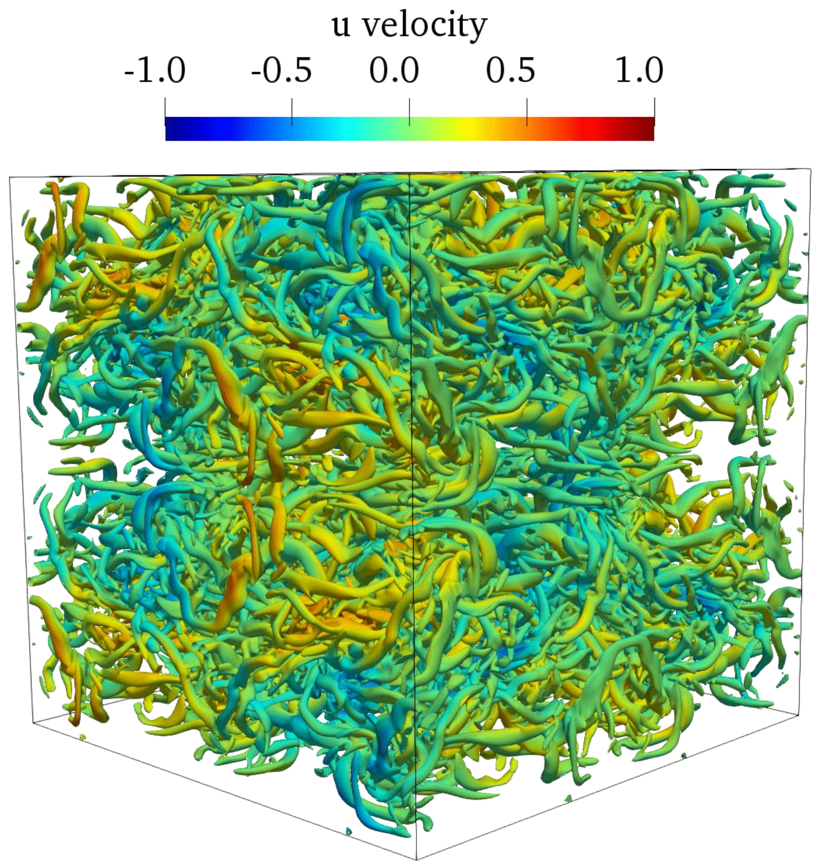


```python
ndim = 3
sc1 = "**{\'scheme\':\'Teno\'}"
# Define the compresible Navier-Stokes equations in Einstein notation.
mass = "Eq(Der(rho,t), - Conservative(rhou_j,x_j,%s))" % sc1
momentum = "Eq(Der(rhou_i,t) , -Conservative(rhou_i*u_j + KD(_i,_j)*p,x_j , %s) + Der(tau_i_j,x_j) )" % sc1
energy = "Eq(Der(rhoE,t), - 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/((gama-1)*Minf*Minf*Pr*Re))*Der(T,x_j))"
# Numerical scheme selection
Avg = RoeAverage([0, 1])
LLF = LLFTeno(teno_order, averaging=Avg)
cent = Central(4)
rk = RungeKuttaLS(3, formulation='SSP')
# Specifying boundary conditions
boundaries[direction][side] = IsothermalWallBC(direction, 0, wall_eqns)
# Generate a C code
alg = TraditionalAlgorithmRK(block)
OPSC(alg)
```

Jacobs, C. T., Jammy, S. P., Sandham N. D. (2017). OpenSBLI: A framework for the automated derivation and parallel execution of finite difference solvers on a range of computer architectures. *Journal of Computational Science*, 18:12-23, DOI: 10.1016/j.jocs.2016.11.001

OpenSBLI

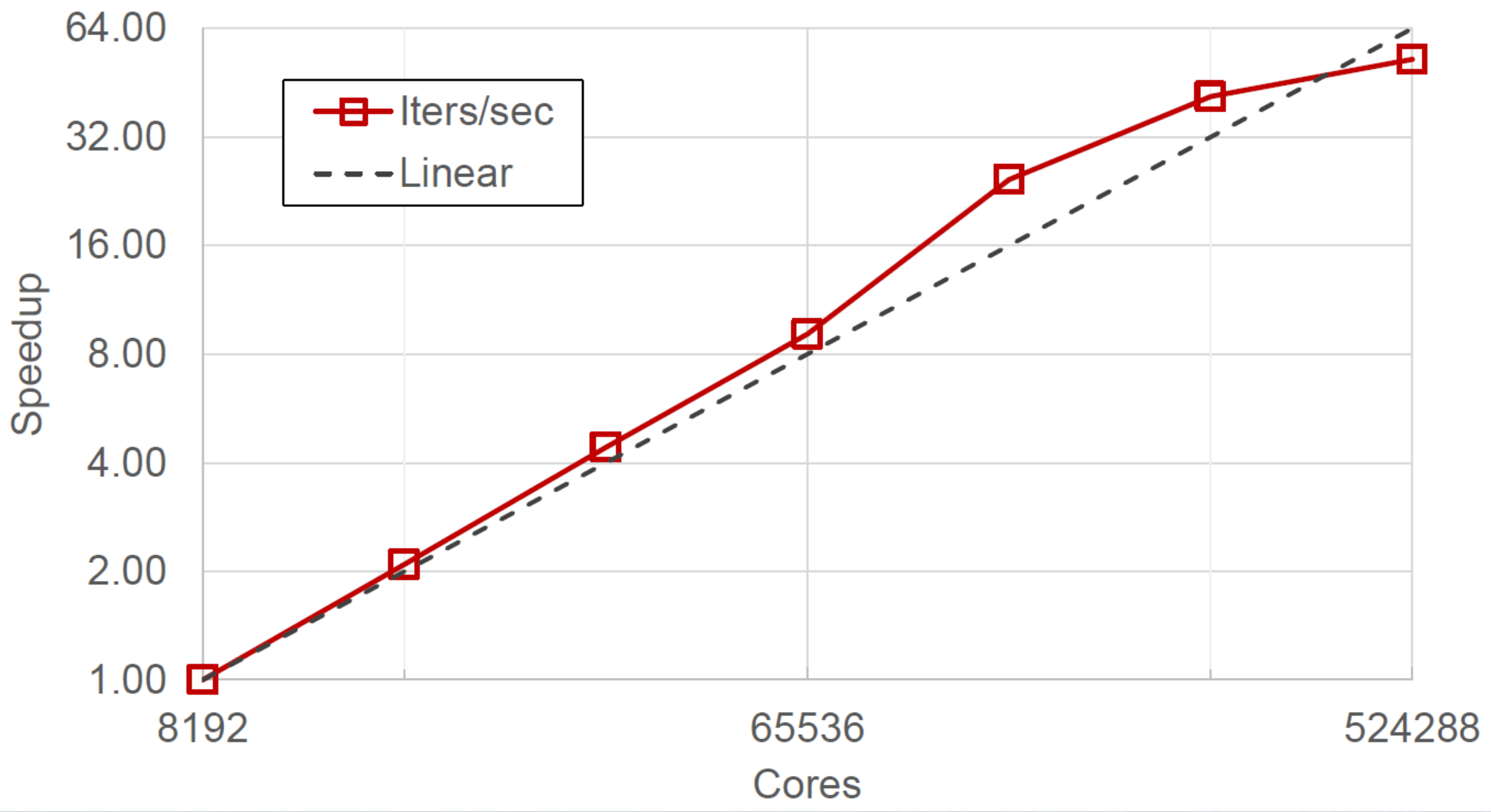
https://opensbli.github.io/



OPENSBLI ON ARHCER2

- Taylor – Green Vortex Problem – ARCHER2 benchmark
  - Strong Scaling - 1024^3 Mesh
  - Double precision
  - Speedup calculated from 1000 iterations – includes start up time.

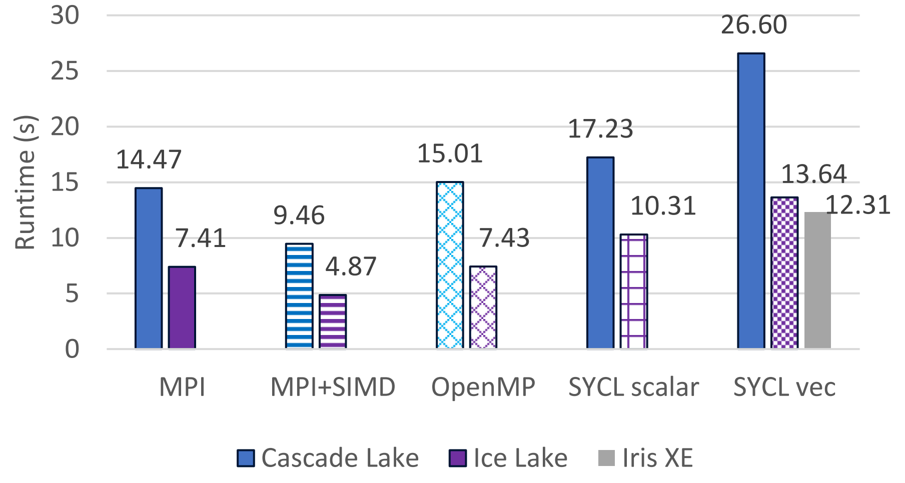
From recent benchmarking runs done by Andrew Turner and the ExCALIBUR Benchmarking team (Oct 2021)

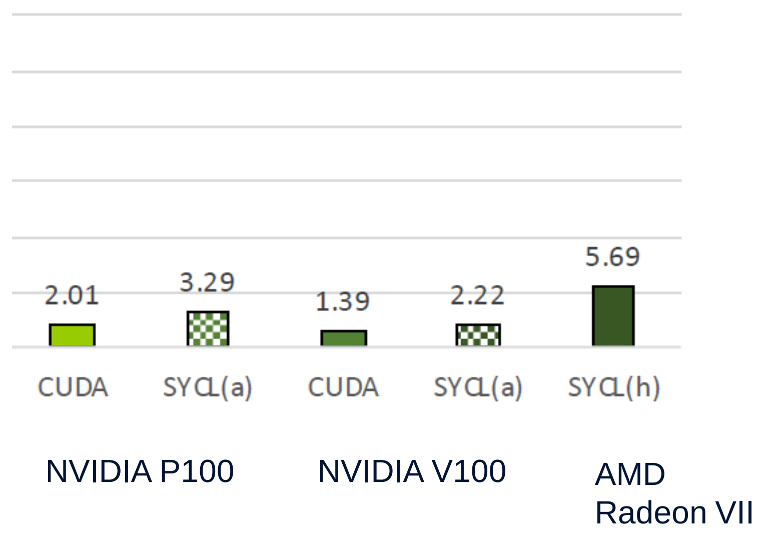


New Parallel Programming Models / Languages - OP2 Generating SYCL

- MG-CFD – Multigrid CFG MiniAPP:
  - NASA Rotor37, 4 multigrid levels, 8M edges
  - Generate Parallelization using OP2
  - Intel compilers - from oneAPI
  - Intel MPI - for MPI, SIMD, OpenMP, MPI+OpenMP

- GPUs – NVIDIA P100 and V100, AMS Radion VII, Intel Iris XE MAX
- CPUs – single socket only to avoid NUMA issues:
  - Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz, 16 cores
  - Intel(R) Xeon(R) Platinum 8360Y @ 2.40 GHz, 36 cores
- SYCL compilers - Intel OneAPI 2021.4 and HipSYCL





HipSYCL

I.Z. Reguly, A.M.B. Owenson, A. Powell, S.A. Jarvis, and G.R. Mudalige, *Under the Hood of SYCL – An Initial Performance Analysis With an Unstructured-mesh CFD Application*, International Supercomputing Conference (ISC 2021), June 2021
I.Z. Reguly. *Performance of DPC++ on Representative Structured/Unstructured Mesh Applications.* Intel DevSummit at SC21

OneAPI/SYCL

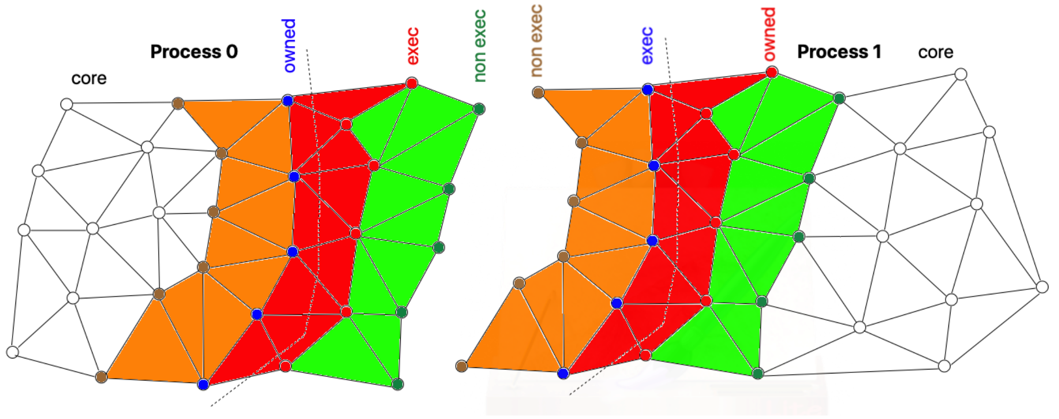
- OP2 and OPS can generate SYCL paralleizations
  - Structured-mesh / Regular applications have good performance portability
  - But various execution strategies needed for unstructured-mesh (irregular) applications to avoid data races

- 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 improving.

- SYCL a much more productive alternative to OpenCL, and performance is improving rapidly
  - But the challenges in terms of performance productivity remain
  - Need multiple code paths for different architectures – e.g. Coloring vs Atomics

Exotic Optimizations - Communication Avoiding Algorithms

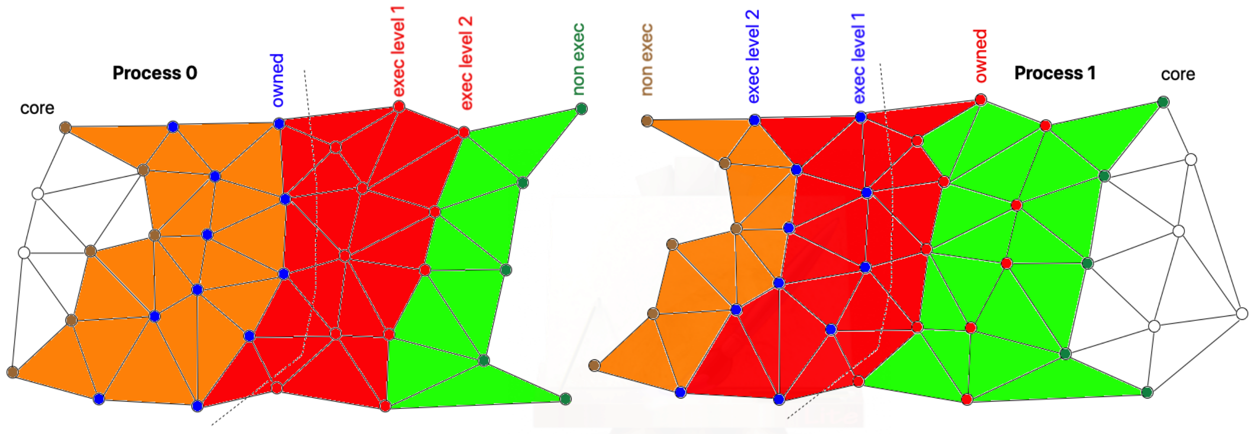
❑ With MPI communication



- Standard OP2 redundant execution over one halo level
- Less computations per node

```
op_par_loop(adt_calc, "adt_calc", cells, …) // MPI Comm
op_par_loop(res_calc, "res_calc", edges, …) // MPI Comm
op_par_loop(bres_calc,"bres_calc", bedges,… ) // MPI Comm
op_par_loop(update, "update", cells,… ) // MPI Comm
```

❑ Without MPI Communication



- Extend halo by one further level
- Redundant compute over both levels
- MPI Comm now avoided – but more computations per node

```
loop_chain_start {
// do all the MPI comms here – with 1 large message per neighbour
op_par_loop(adt_calc, "adt_calc", cells, …);
op_par_loop(res_calc, "res_calc", edges, …);
op_par_loop(bres_calc,"bres_calc", bedges,… );
op_par_loop(update, "update", cells,… );
} loop_chain_end
```

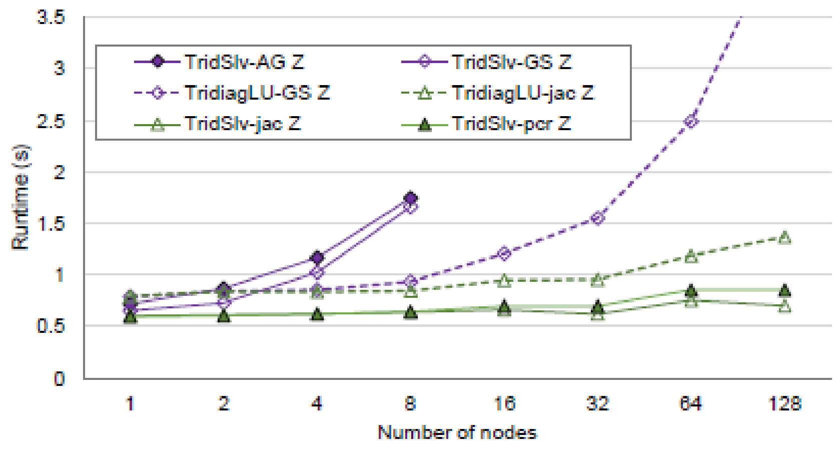
Direct Solvers - Multi-Dim Tridiagonal Solvers on Clusters of GPUs

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d, \quad i = 0, 1, ..., N-1$$

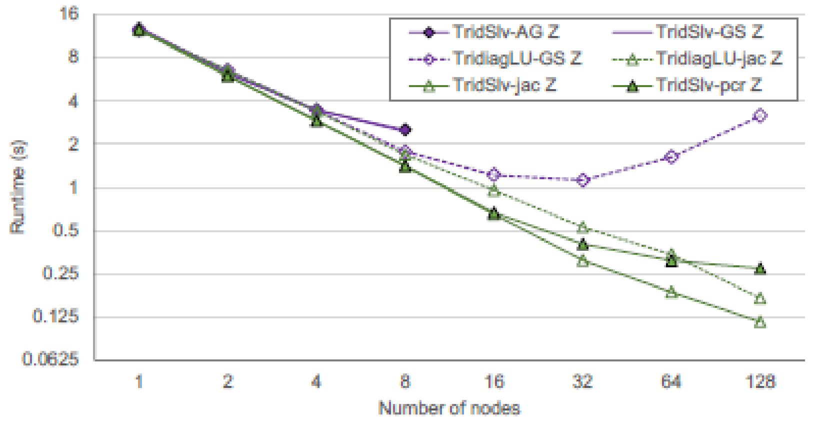
$$\begin{bmatrix} b_0 & c_0 & 0 & \dots & 0 \\ a_1 & b_1 & c_1 & \dots & 0 \\ 0 & a_2 & b_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{N-1} & b_{N-1} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{N-1} \end{bmatrix}$$

pcr – parallel cyclic reduction (PCR) solver
Jac – Jacobi iterative solver
AG – All Gather
GS – Gather Scatter
HC – host copy
GD – GPU direct

ARCHER2



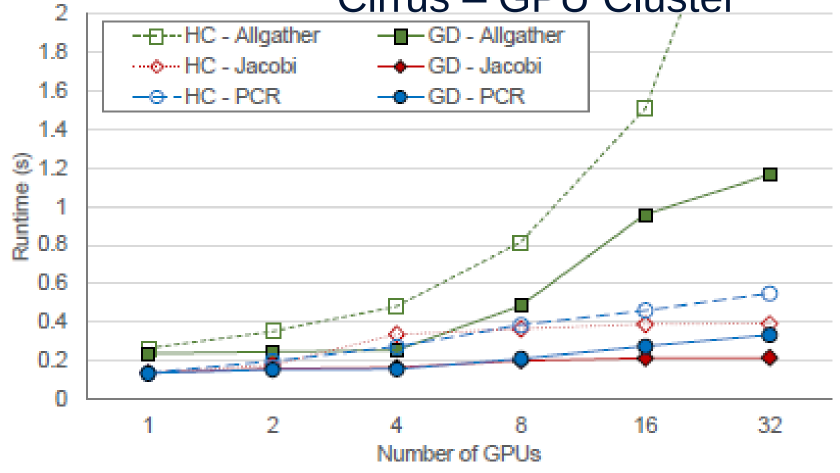
(b) TridSlv vs TridiagLU weak scaling



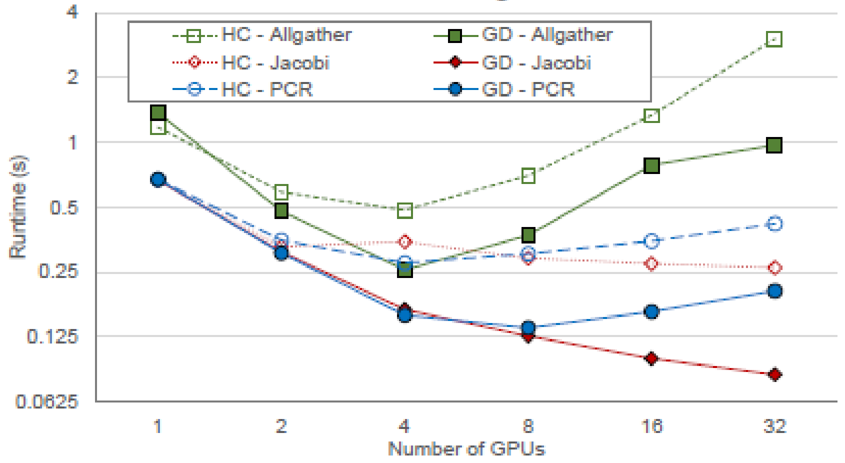
(d) TridSlv vs TridiagLU strong scaling

Weak Scaling : 512^3 per Node
Strong Scaling : 512× 512 × 8192

Cirrus – GPU Cluster



(b) TridSlv, weak scaling, HC vs GD

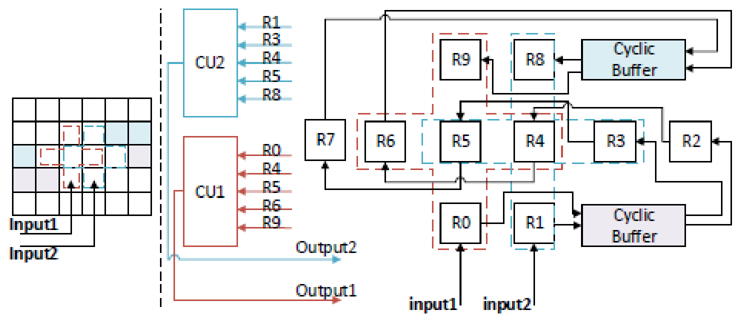


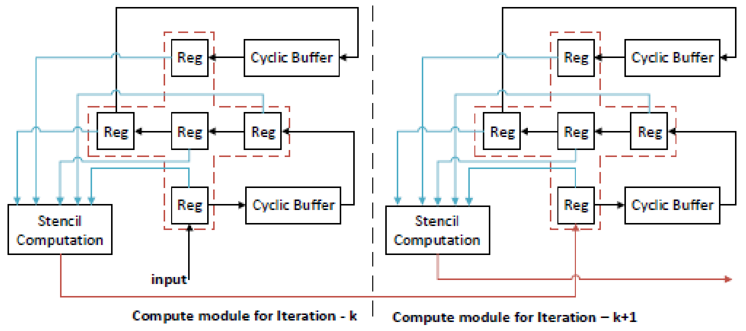
(d) TridSlv, strong scaling, HC vs GD

Weak Scaling : 512^3 per GPU
Strong Scaling : 512× 512 × 2048

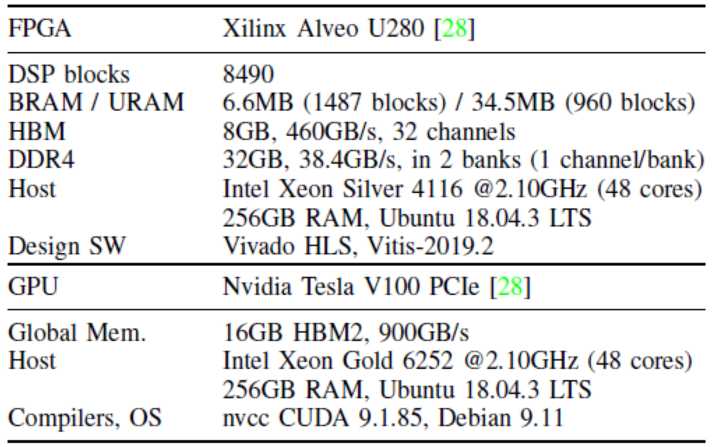
G.D. Balogh, T. Flynn, S. Laizet, G.R. Mudalige, I.Z. Reguly. *Scalable Many-core Algorithms for Tridiagonal Solvers,* in 2021 Computing in Science & Engineering, vol. , no. 01, pp. 1-1, 5555.
doi: 10.1109/MCSE.2021.3130544

STENCILS ON FPGAS



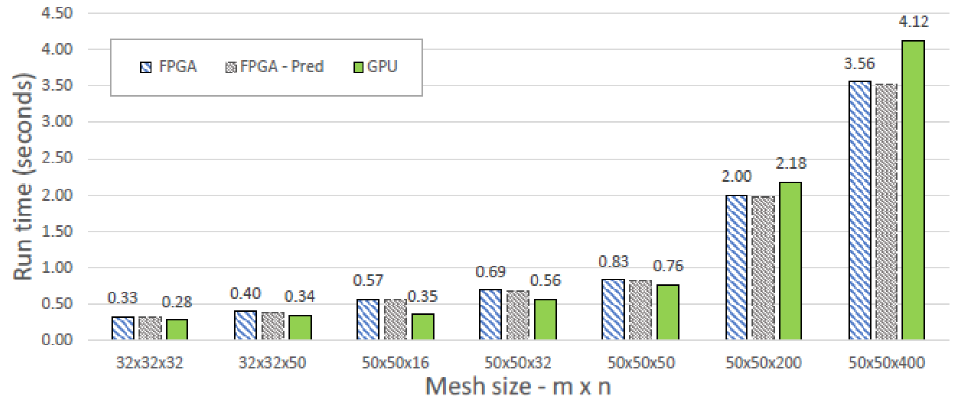


Reg
Cyclic Buffer
Reg
Reg
Reg
Reg
Cyclic Buffer
Stencil Computation
input
Compute module for Iteration - k
Compute module for Iteration – k+1

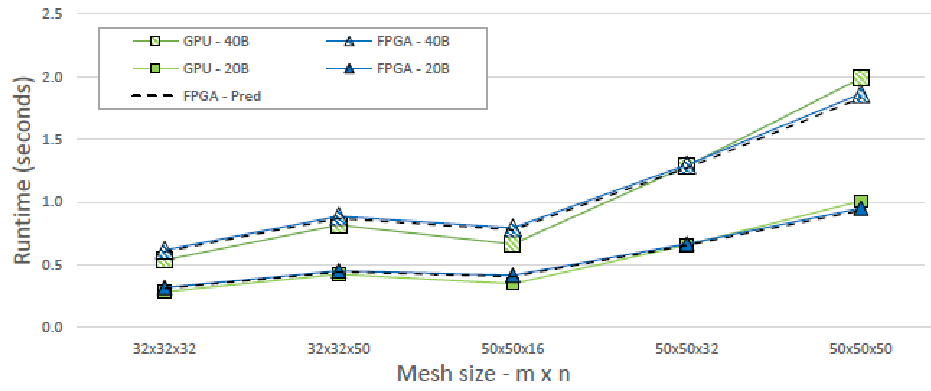


FPGA	Xilinx Alveo U280 [28]
DSP blocks	8490
BRAM / URAM	6.6MB (1487 blocks) / 34.5MB (960 blocks)
HBM	8GB, 460GB/s, 32 channels
DDR4	32GB, 38.4GB/s, in 2 banks (1 channel/bank)
Host	Intel Xeon Silver 4116 @2.10GHz (48 cores) 256GB RAM, Ubuntu 18.04.3 LTS
Design SW	Vivado HLS, Vitis-2019.2
GPU	Nvidia Tesla V100 PCIe [28]
Global Mem.	16GB HBM2, 900GB/s
Host	Intel Xeon Gold 6252 @2.10GHz (48 cores) 256GB RAM, Ubuntu 18.04.3 LTS
Compilers, OS	nvcc CUDA 9.1.85, Debian 9.11

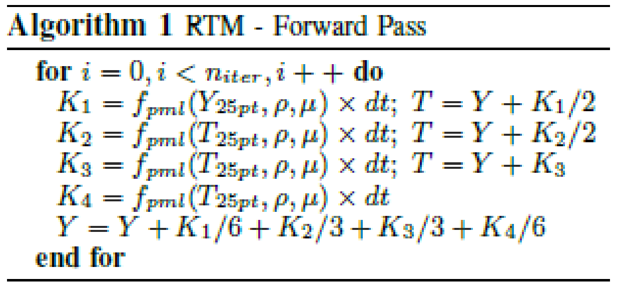
3D - Reverse Time Migration (RTM) – Forward Pass



(a) Baseline - 1800 iterations



(b) Batching - 180 iterations



Algorithm 1 RTM - Forward Pass

```
for $i = 0, i < n_{iter}, i++$ do
    $K_1 = f_{pml}(Y_{25pt}, \rho, \mu) \times dt;\ T = Y + K_1/2$
    $K_2 = f_{pml}(T_{25pt}, \rho, \mu) \times dt;\ T = Y + K_2/2$
    $K_3 = f_{pml}(T_{25pt}, \rho, \mu) \times dt;\ T = Y + K_3$
    $K_4 = f_{pml}(T_{25pt}, \rho, \mu) \times dt$
    $Y = Y + K_1/6 + K_2/3 + K_3/3 + K_4/6$
end for
```

- ❑ Competitive runtimes with GPUs
- ❑ Even when runtime is inferior to the GPU we get significant energy savings (e.g. over 2x for for the RTM app)

K. Kamalakkannan, **G.R. Mudalige**, I.Z. Reguly, S.A. Fahmy, *High-Level FPGA Accelerator Design for Structured-Mesh-Based Explicit Numerical Solvers*, in 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2021), Portland Oregon, USA, 2021 pp. 1087-1096. doi: 10.1109/IPDPS49936.2021.00117

Multi-Dim Tridiagonal Solvers on FPGAs

$$Ax = d$$

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d, \quad i = 0, 1, ..., N-1 \qquad (1)$$

$$\begin{bmatrix} b_0 & c_0 & 0 & \dots & 0 \\ a_1 & b_1 & c_1 & \dots & 0 \\ 0 & a_2 & b_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{N-1} & b_{N-1} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{N-1} \end{bmatrix} \qquad (2)$$

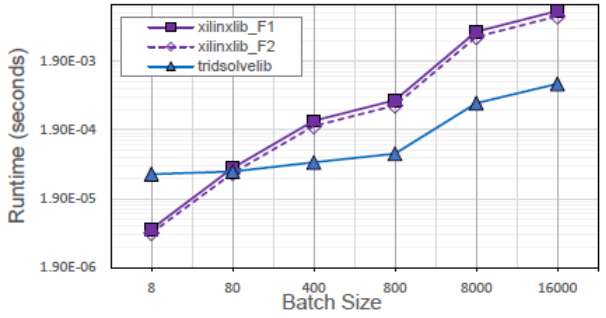
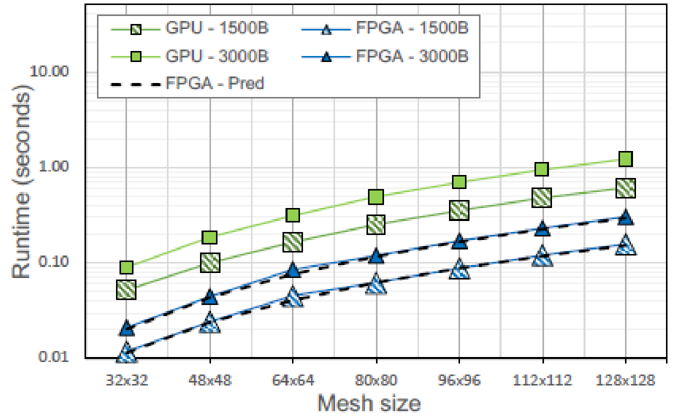


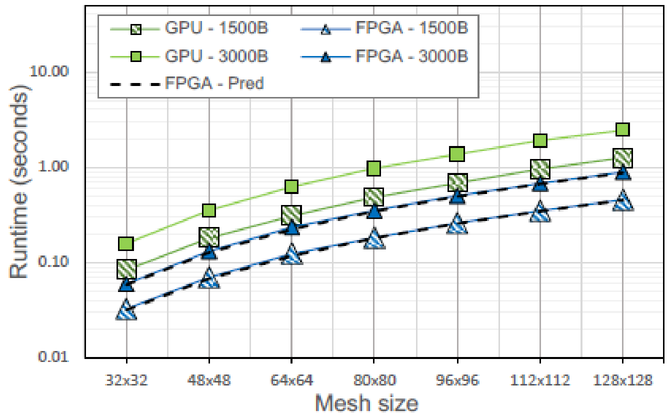
Fig. 4: `tridsolvlib` vs `xilinxlib`, System size-128

Algorithm 3: `3D ADI Heat Application`

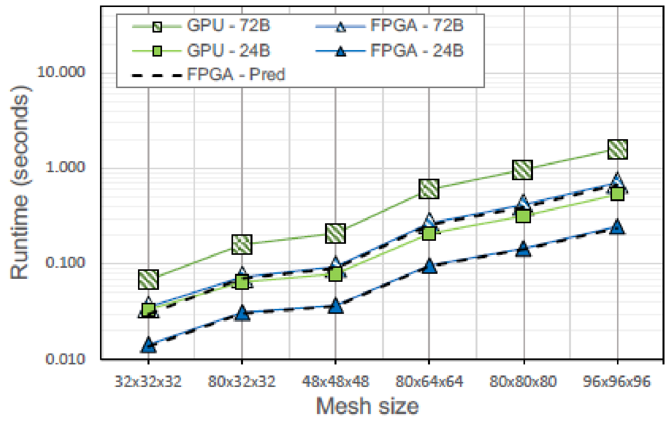
```
1: for $i = 0, i < n_{iter}, i++$ do
2:     Calculate RHS :
       $d = f_{7pt}(u), a = \frac{-1}{2}\gamma, b = \gamma, c = \frac{-1}{2}\gamma$
3:     Tridslv(x-dim), update $d$
4:     Tridslv(y-dim), update $d$
5:     Tridslv(z-dim), update $d$
6:     $u = u + d$
7: end for
```



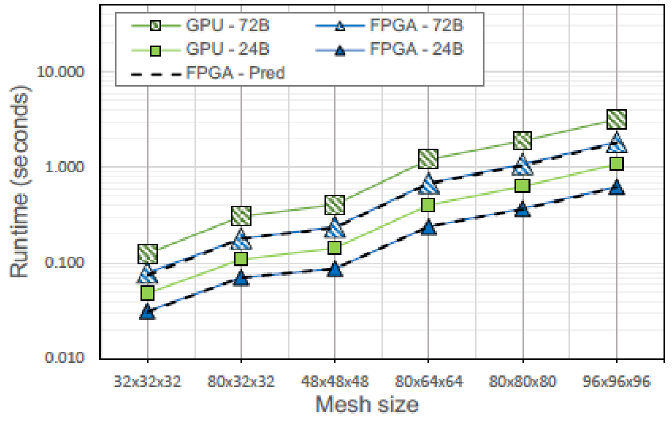
(a) 2D FP32 - 120 iter.$v = 8$, $f_U = 3$, $N_{CU} = 3$



(b) 2D FP64 - 120 iter.$v = 8$, $f_U = 2$, $N_{CU} = 3$



(c) 3D FP32 - 100 iter.$v = 8$, $N_{CU} = 6$



(d) 3D FP64 - 100 iter. $v = 8$, $N_{CU} = 3$

Fig. 5: ADI Heat Diffusion Application Performance

Kamalavasan Kamalakkannan, Istvan Z. Reguly, Suhaib A. Fahmy, and Gihan R. Mudalige. *High Throughput Multidimensional Tridiagonal Systems Solvers on FPGAs* (2021) – Under Review

Non-Traditional Architectures - Multi-Dim Tridiagonal Solvers on FPGAs

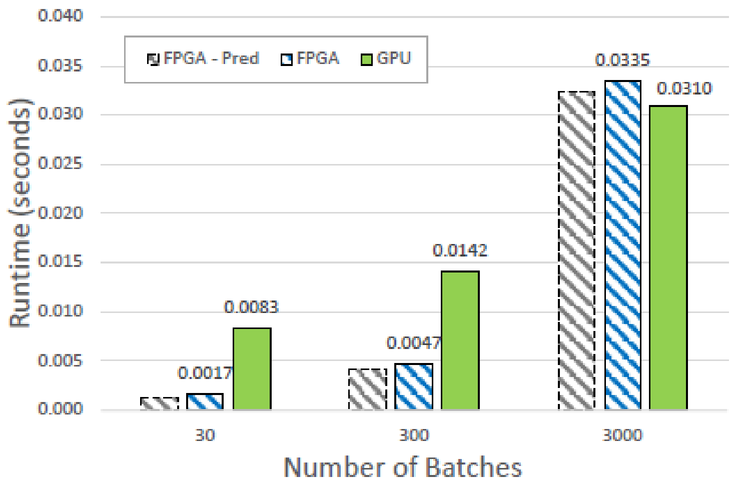
- Stochastic Local Volatility (SLV) model application - high throughput batched implementation on Xilinx FPGAs

Algorithm 4: `2D Heston SLV Backward`

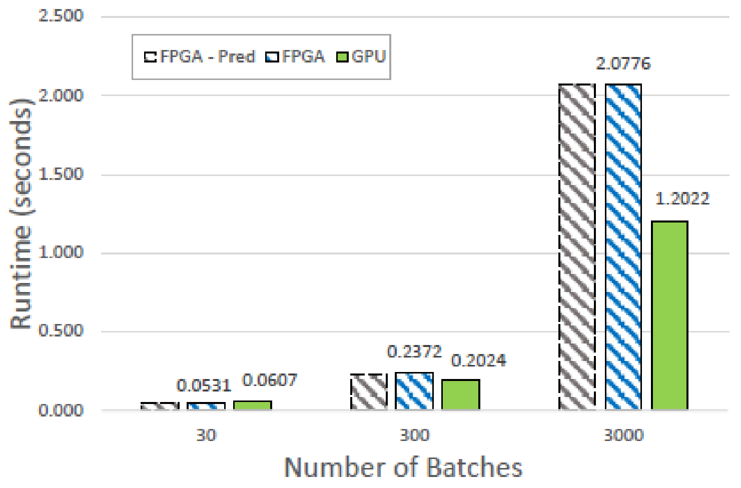
```
1: for $i = 0, i < n_{iter}, i++$ do
2:     hv_pred0(), hv_matrices()
3:     Tridslv(x-dim)
4:     hv_pred1(), Tridslv(y-dim)
5:     hv_pred2(), Tridslv(x-dim)
6:     hv_pred3(), Tridslv(y-dim)
7: end for
```

TABLE 5: SLV : Bandwidth, *BW* (GB/s) and Energy, *E*, (J)

Batch	*BW* F	*BW* Gx	*BW* Gy	*E* F	*E* G
40×20 mesh					
30	55.24	3.04	28.01	0.13	0.45
300	202.31	16.48	176.51	0.35	1.02
3000	281.06	123.84	327.65	2.51	4.75
100×50 mesh					
30	124.63	51.28	109.65	3.98	3.76
300	278.87	235.22	238.34	17.79	22.26
3000	318.36	421.77	429.21	155.82	216.40



(a) 40×20 mesh, $v = 1, f_U = 1, N_{CU} = 3$



(b) 100×50, $v = 1, f_U = 1, N_{CU} = 3$

Fig. 7: SLV application performance.

- Competitive runtimes with GPUs
- FPGA solution is over 30% more energy efficient for large batch solves over the GPU

Kamalavasan Kamalakkannan, Istvan Z. Reguly, Suhaib A. Fahmy, and Gihan R. Mudalige. *High Throughput Multidimensional Tridiagonal Systems Solvers on FPGAs* (2021) – Under Review

OTHER PROJECTS USING OP2/OPS

- 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

CURRENT WORK AND FUTURE WORK - EXCALIBUR PROJECT

- CCP – Turbulence
  - Direct solver libraries – Tri-, penta-, 7-, 9-, 11 diagonal, multi-dimensional solvers
  - Integrate directsolver libraries to be called within OPS
  - OpenSBLI type high-level (Python) framework for XCompact3D – High Order FD framework

- ExCALIBUR Phase 1B – Turbulence at the Exascale
  - Imperial, Warwick, Newcastle, Southampton, Cambridge, STFC collaboration | UKTC and UKCTRF Communities
  - Xcompact3D and Wind Energy, OpenSBLI and Green Aviation, uDALES and Air Quality, SENGA+ and Net-Zero Combustion
  - Extending OPS capability – robust code-gen tools and parallel transformations | support future-proof code development
  - UQ, I/O, Coupling and Visualization
  - Machine Learning Algorithms for Turbulent Flow

- UK AEA Mini-Apps Project
  - Collaboration with University of York
  - Developing Prototype miniApps for UKAEA workload
  - Investigate / advise on performance portability techniques and current state-of-the-art.

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 ?
  - What code should I certify ? High-level code/generated code ?
  - Difficult to convince users to use new code - fear of an opaque compiler / intermediate representation / black box !
- Incremental conversion – loop by loop
  - Simpler than CUDA, but more difficult than OpenACC/OpenMP
  - Automated conversion ?
- Changing user requirements
  - Wanting to use a DSL for doing things beyond what it was intended for !
  - Asking for "back-doors" / "escape hatches" -- leads to poor performance

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 ?
  - MLIR appearing to give some advance capabilities – see ExCALIBUR xDSL project (Tobias Grosser, Paul Kelly et al.)

- Code-generation for more exotic architectures – e.g. FPGAs
  - Large design space
  - Complex source transformations

- Maintainable/long term source-to-source technologies
  - Domain Scientists not having expertise to understand / maintain DSLs

Challenges – Who Maintains the DSL, What DSL to choose ?

- Currently purely done via academic and (small/short term) industrial funding

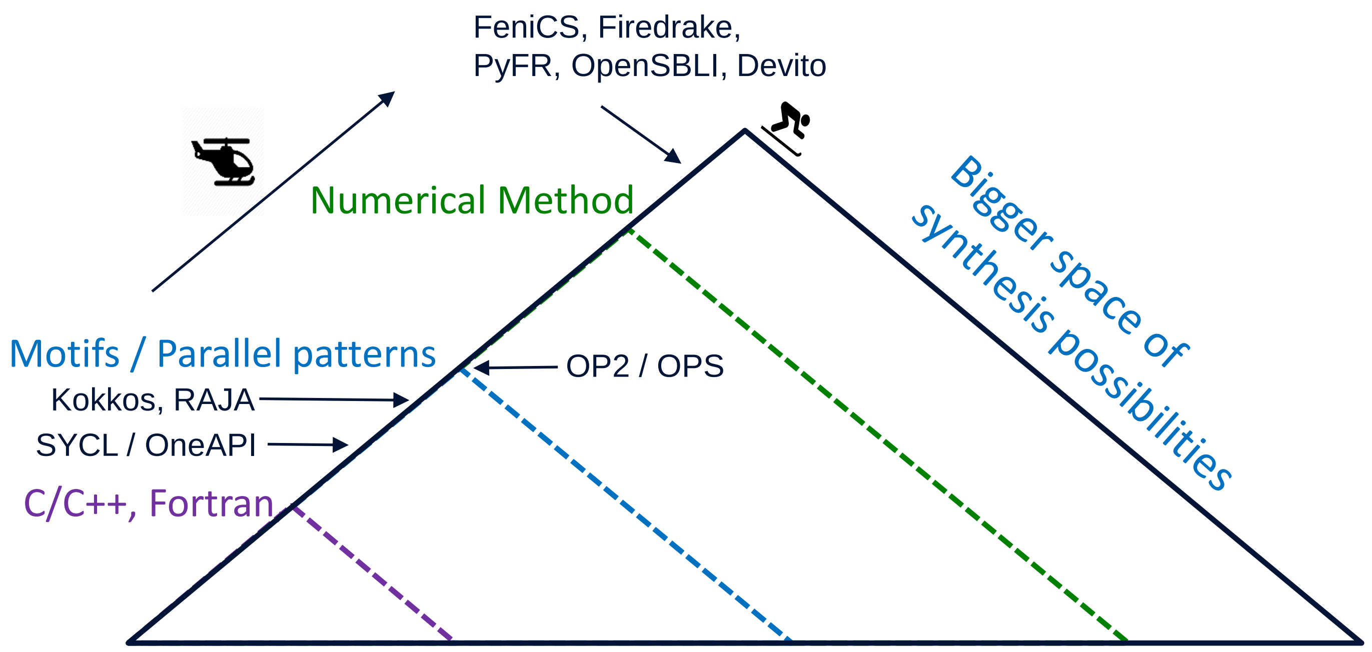
- Long term funding and maintenance
  - Once established probably will not be different to any other classical library
  - Will require compiler expertise to maintain code generation tools

- What DSL to choose ?
  - Re-use technologies / DSLs – especially code-gen tools (best not to reinvent !)

- Skills Gap
  - Programme in C/C++/Fortran (at a minimum)
  - Knowledge of compilers / code-generation
  - Compete for applicants – Communicate what we do better | impact of HPC / Computational Sciences
  - Salary ☹
  - Contracts ☹

DSLs / High-level Abstractions Gaining Traction !

- FEniCS - PDE solver package - https://fenicsproject.org/
- Firedrake - automated system for the portable solution of PDEs using the finite element method https://www.firedrakeproject.org/
- PyFR - Python based framework for solving advection-diffusion type problems on streaming architectures using the Flux Reconstruction approach - http://www.pyfr.org/
- 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
- GungHO project - Weather modelling codes (MetOffice)
- STELLA – DSL for stencil codes, for solving PDEs (Metro Swiss)
- Liszt – Stanford University : DSL for solving mesh-based PDEs - http://graphics.stanford.edu/hackliszt/
- Kokkos – C++ template library – SNL
- RAJA - C++ template libraries - LLNL

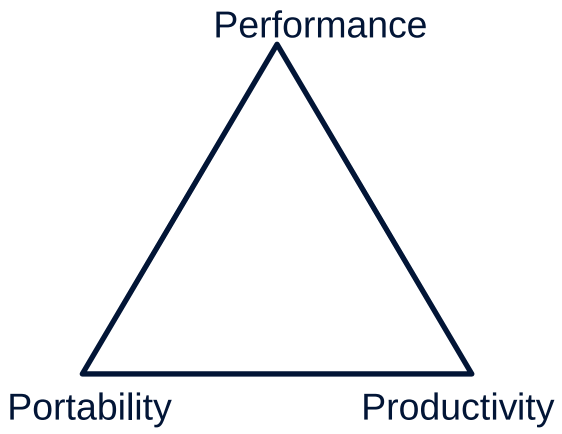


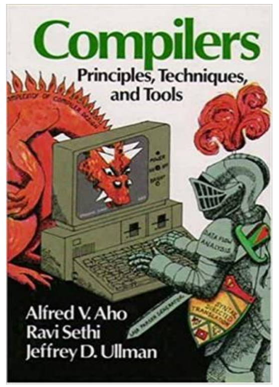
Adapted from: *Synthesis versus Analysis: What Do We Actually Gain from Domain-Specificity?* Keynote talk at the LCPC 2015. Paul H. J. Kelly (Imperial College London)

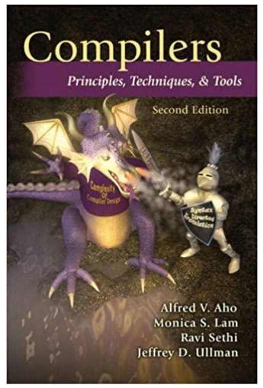
Separation of Concerns – One of the four pillars of ExCALIBUR

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







"Fundamentals and abstractions have more staying power than the technology of the moment"
Alfred Aho and Jeffrey Ullman (Turing Award Recipients 2020)

Downloads and More Information

- GitHub Repositories
  - OP2 – https://github.com/OP-DSL/OP2-Common
  - OPS – https://github.com/OP-DSL/OPS
  - OP-DSL Webpage - 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.

- ❑ This research is supported by Rolls-Royce plc., and by the UK EPSRC (EP/S005072/1) Strategic Partnership in ComputationalScience for Advanced Simulation and Modelling of Engineering Systems (ASiMoV).

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