# Evolutionary Re-Engineering of Multi-Physics Industrial HPC Applications with OP-DSLs

**Gihan Mudalige**

Royal Society Industry Fellow

Associate Professor (Reader) in High Performance Computing

Department of Computer Science, University of Warwick

g.mudalige@warwick.ac.uk

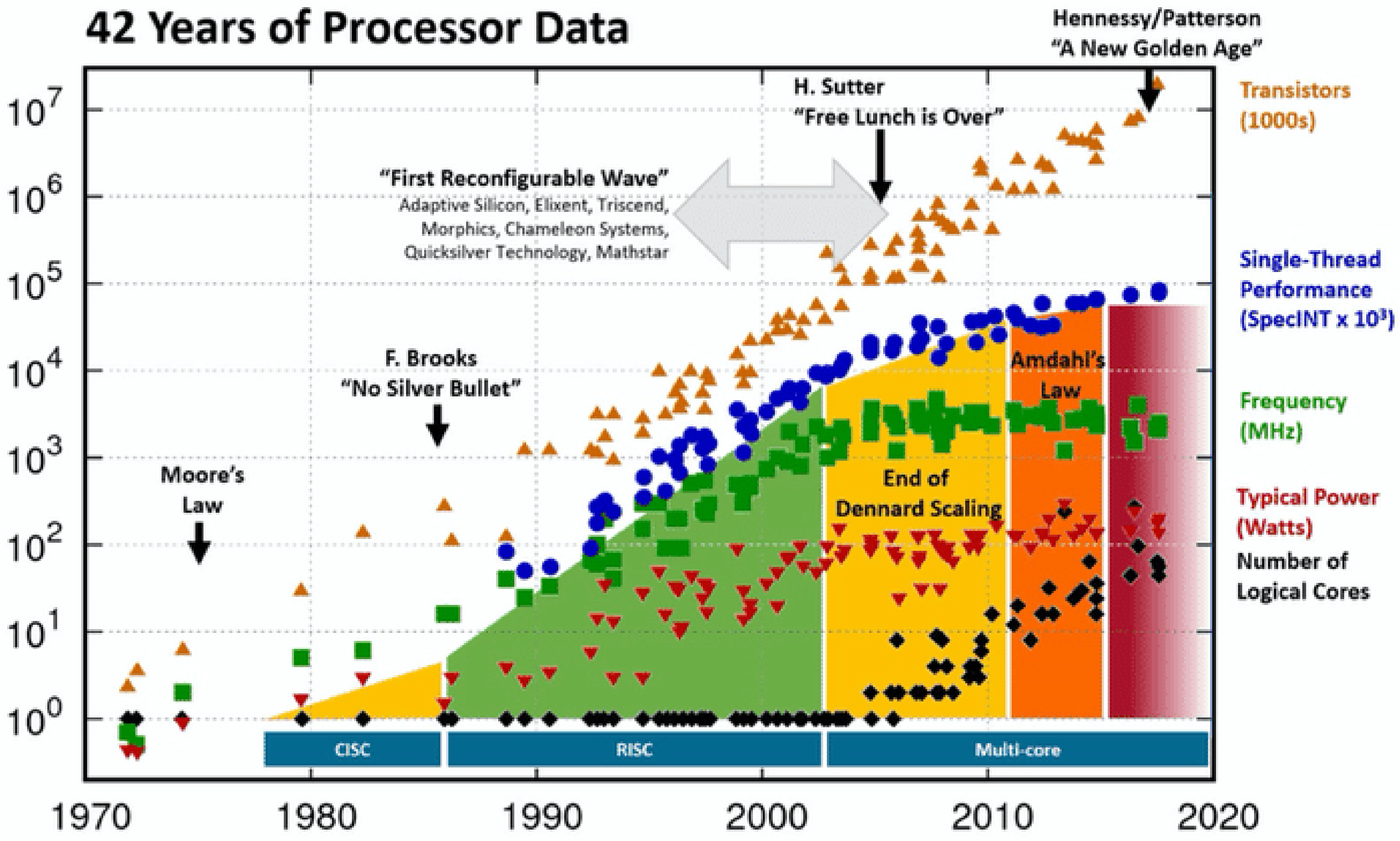
Joint work with:

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

26th Feb 2022 – Code Generation and Transformation in HPC on Heterogeneous Platforms - SIAM-PP22

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

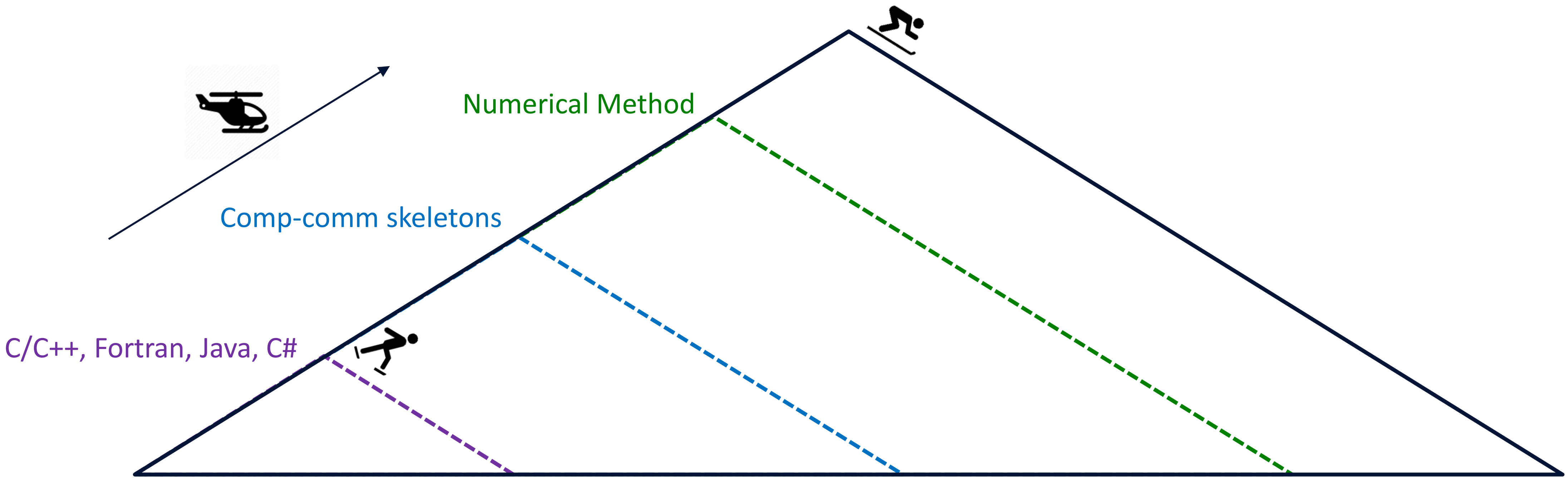
- Not clear which architectural approach is likely to *win* in the long-term
  - Cannot be re-coding applications for each *new* type of architecture or parallel system
  - Nearly impossible for re-writing legacy codes

- 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 ✓
- OP-DSLs
- Evolving Hydra to OP2 Hydra
- Challenges and Lessons Learnt
- Conclusions

# 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

# 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

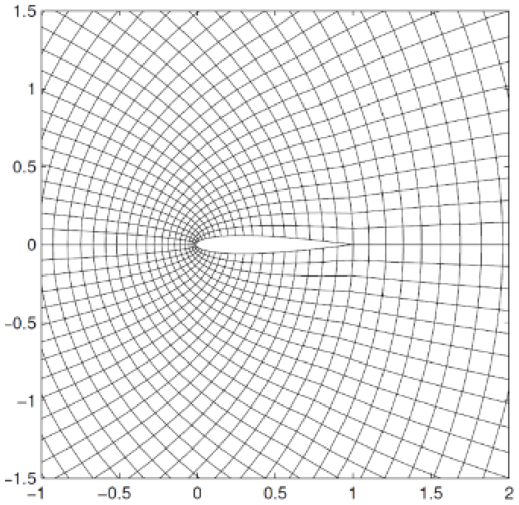
- Restrict writing code that is difficult (for the compiler) to reason about and optimize
  - *"OP2 and OPS are a straightjacket"* – Mike Giles
  - Build in safe guards so that user cannot write bad code !

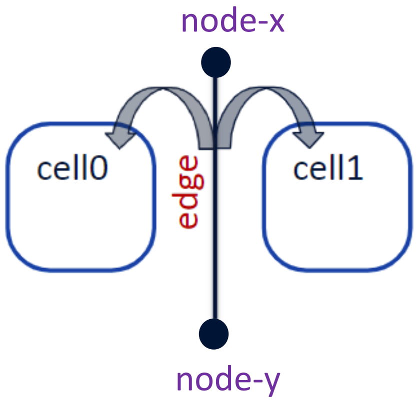
- Implementation of the API left to a lower level
  - Target implementation to hardware – can use best optimizations
  - Automatically generate implementation from specification for the context
  - Exploit domain knowledge for better optimisations - reuse what we know is best for each context

# OP2 API - Example

```fortran
! Declaring the mesh with OP2
! sets
call op_decl_set(nnode,nodes,'nodes')
call op_decl_set(nedge,edges,'edges')
call op_decl_set(ncell,cells,'cells')
! maps
call op_decl_map(edges,nodes,2,edge ,pedge ,'pedge' )
call op_decl_map(edges,cells,2,ecell,pecell,'pecell')
! data
call op_decl_dat(nodes,2,'real(8)',x,p_x,'p_x')
call op_decl_dat(cells,4,'real(8)',q,p_q,'p_q')
call op_decl_dat(cells,1,'real(8)',adt,p_adt,'p_adt')
call op_decl_dat(cells,4,'real(8)',res,p_res,'p_res')

```

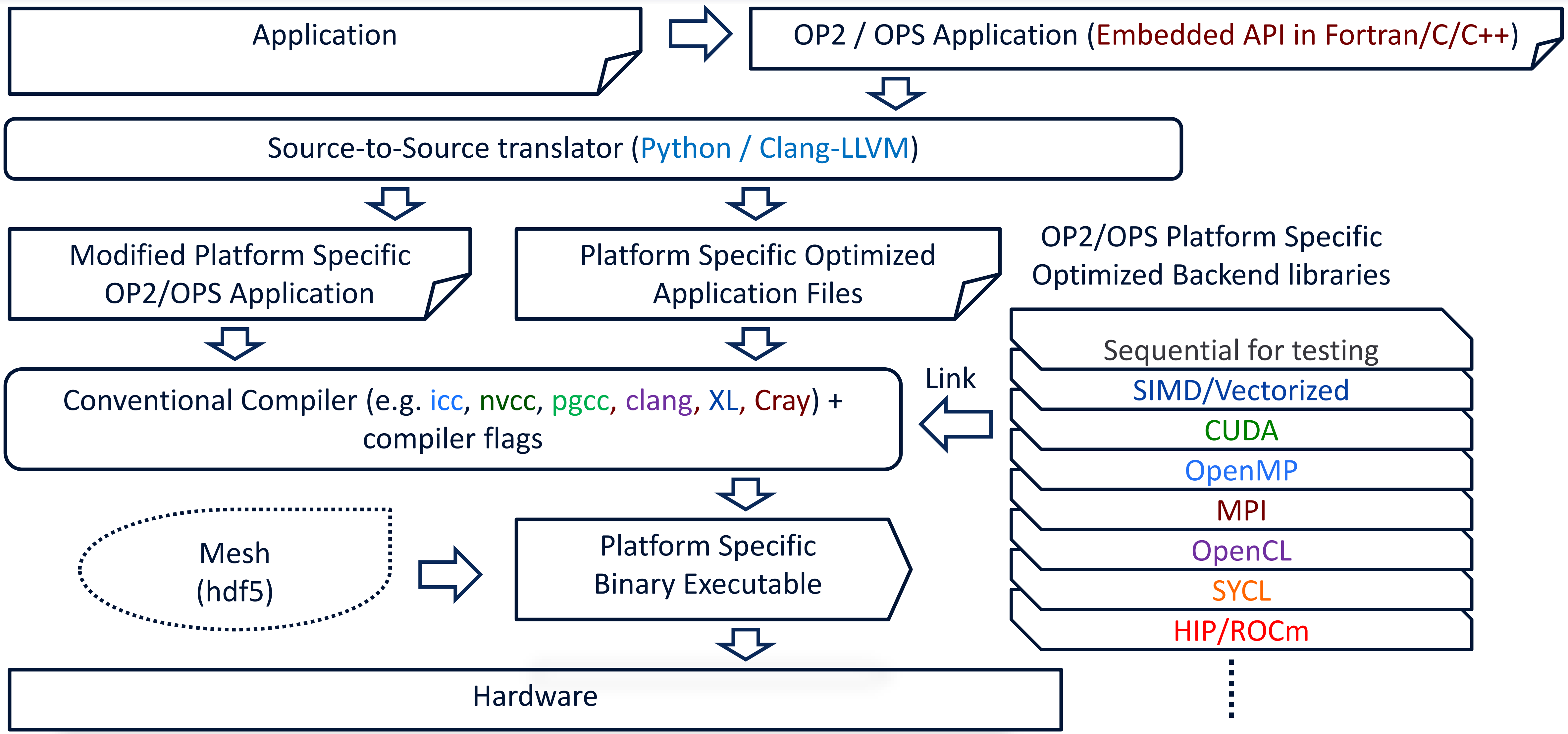


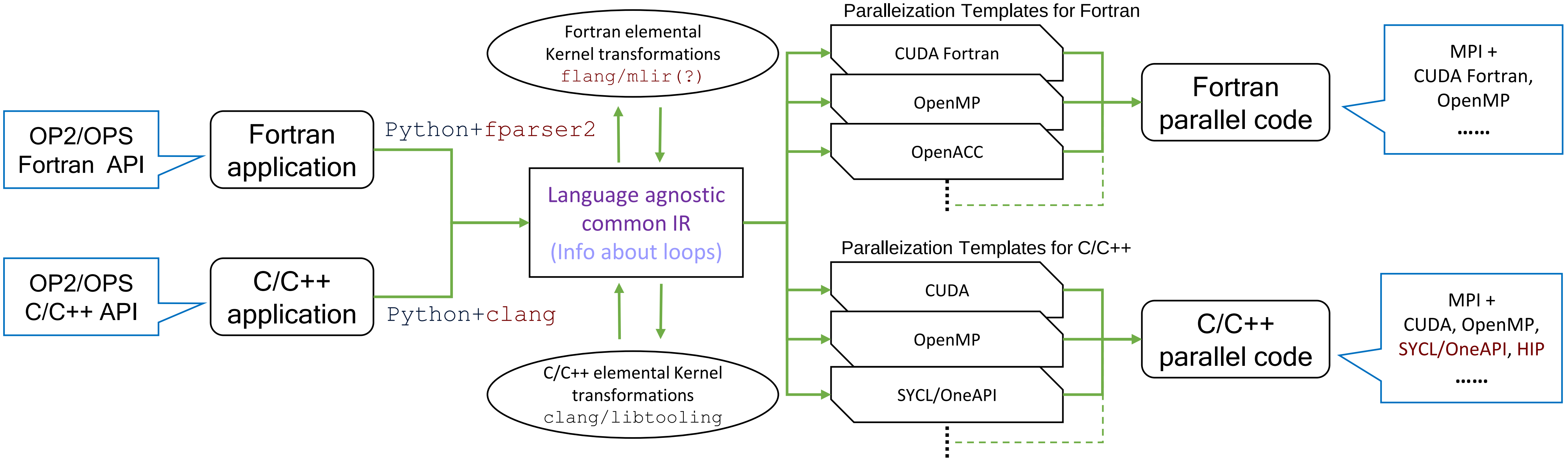

```fortran
! Elemental kernel
subroutine res_calc(x1,x2,q1,q2,adt1,adt2,res1,res2)
  IMPLICIT NONE
  REAL(kind=8), DIMENSION(2), INTENT(IN) :: x1
  REAL(kind=8), DIMENSION(2), INTENT(IN) :: x2
  ...
  REAL(kind=8) :: dx,dy,mu,ri,p1,vol1,p2,vol2,f
  dx = x1(1) - x2(1)
  dy = x1(2) - x2(2)
  ...
  f = 0.5 * (vol1 * q1(1) + vol2 * q2(1)) + &
  &    mu * (q1(1) - q2(1))
  res1(1) = res1(1) + f
  res2(1) = res2(1) - f
  ...
end subroutine
```

```fortran
! Calculate flux residual - parallel loop over edges
call op_par_loop_8 (res_calc, edges, &
& op_arg_dat(x,    1, edge,  2,"real(8)", OP_READ), &
& op_arg_dat(x,    2, edge,  2,"real(8)", OP_READ), &
& op_arg_dat(q,    1, ecell, 4,"real(8)", OP_READ), &
& op_arg_dat(q,    2, ecell, 4,"real(8)", OP_READ), &
& op_arg_dat(adt,  1, ecell, 1,"real(8)", OP_READ), &
& op_arg_dat(adt,  2, ecell, 1,"real(8)", OP_READ), &
& op_arg_dat(res,  1, ecell, 4,"real(8)", OP_INC ), &
& op_arg_dat(res,  2, ecell, 4,"real(8)", OP_INC ))
```

# Application Development



# OP2/OPS Code Generation

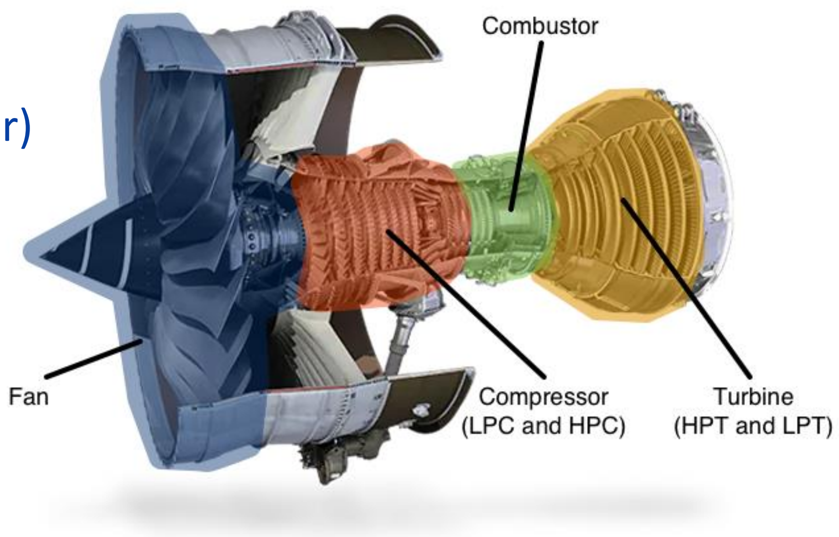



- Simplest Code generation / translation
  - Intermediate representation is simply the loop descriptions + elemental kernels
  - Generated parallel code can be viewed and understood by a human !

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

# EPSRC Prosperity Partnership – ASIMoV [ Hydra to OP2-Hydra ]

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



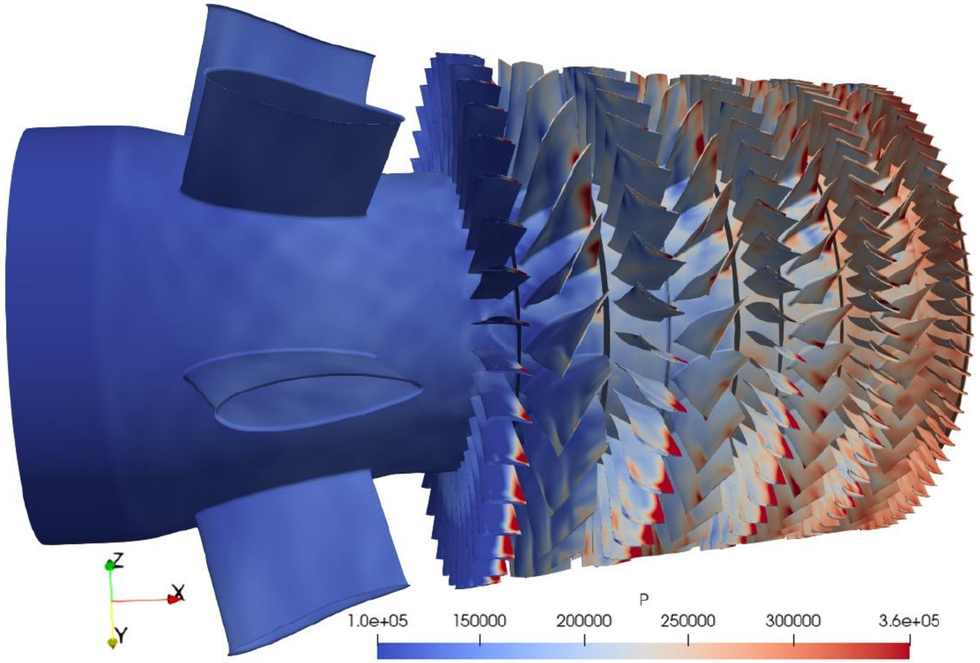

- Grand Challenge 1 – Sliding Planes model of Rig250 (DLR test rig compressor)
  - 4.5 stage rotor-stator (10-row full annulus) | 4.58B mesh nodes.
  - Need to obtain 1 revolution of compressor in less than 24 hours
  - Current production estimates at 7 days

- Setup
  - Moving rotor-stator – sliding planes interfaces
  - Rotors and Stators modelled with Hydra CFD suite – URANS (360 degree models)
  - 10 rotor-stator interfaces
  - Code coupling for sliding planes – move from current monolithic (Hydra only) production code to coupling

- Challenges
  - Performance portability – run both CPUs and GPUs by multiple vendors
  - Preserve production code's scientific code and structure – cannot re-write, MUST "evolve" not overhaul !
  - Convince users to adopt ! (Ongoing for nearly 10 years now)

# OP2-Hydra Performance

| | | |
|---|---|---|
| System | ARCHER2<br>HPE Cray EX [6] | Cirrus<br>SGI/HPE 8600<br>GPU Cluster [4] |
| Processor | AMD EPYC 7742<br>@ 2.25 GHz | Intel Xeon Gold 6248<br>(Cascade Lake) @ 2.5 GHz<br>+ NVIDIA Tesla<br>V100-SXM2-16GN GPU |
| (procs×cores)<br>/node | 2×64 | 2×20 + 4×GPUs |
| Memory/node | 256 GB | 384 GB + 40GB/GPU |
| Interconnect | HPE Cray Slingshot<br>2×100 Gb/s<br>bi-directional/node | Infiniband<br>FDR, 54.5 Gb/s |
| OS | HPE Cray LE<br>(based on SLES 15) | Linux CentOS 7 |
| Compilers | GNU 10.2.0 | nvfortran (nvhpc 21.2) |
| Compiler Flags | `-O2 -eF -fPIC` | |
| Power/node | 660W | ≈ 900W |



# OP2-HYDRA PERFORMANCE *

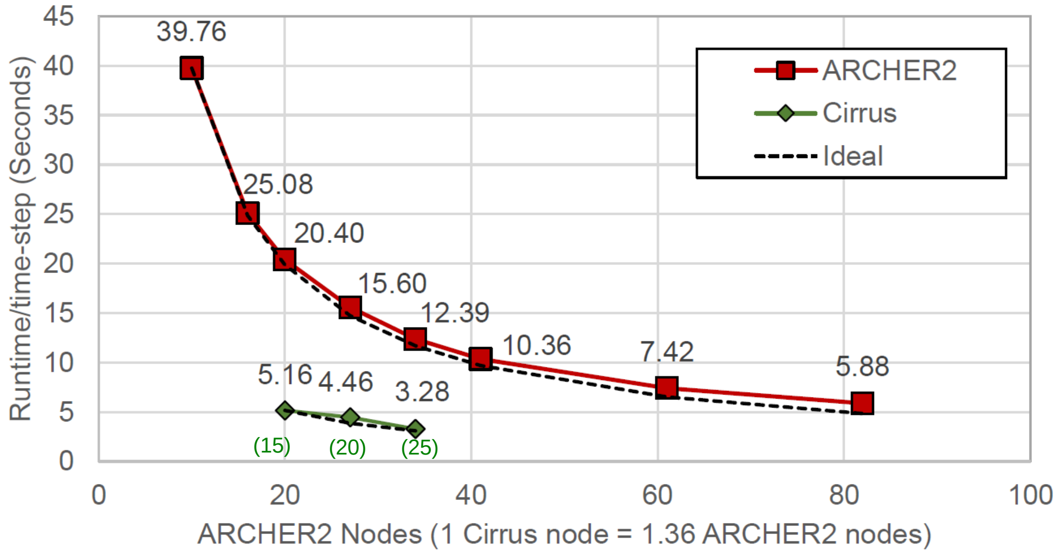
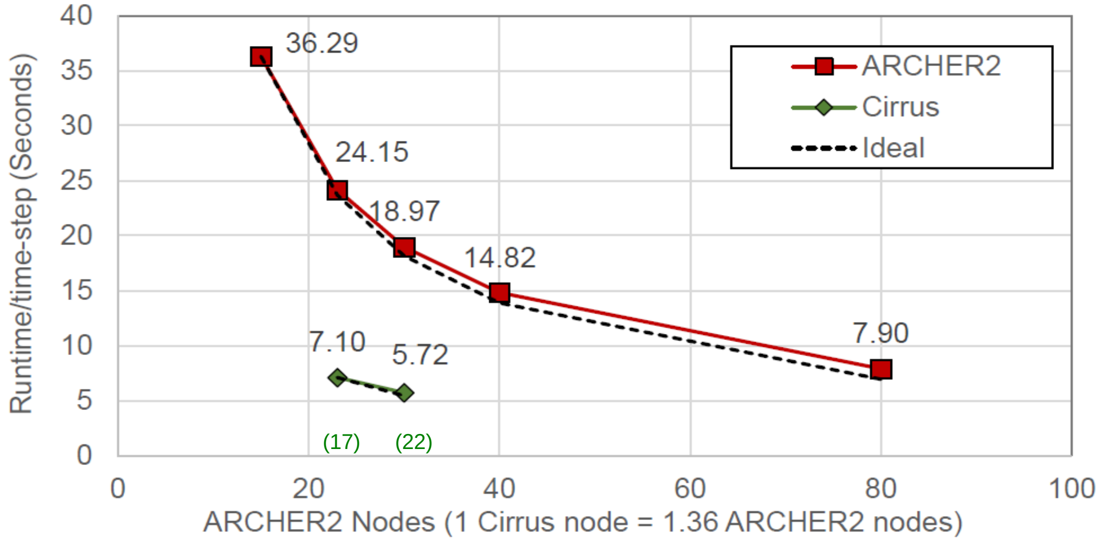



**Figure 7: Rig250** $1 - 10_{430M}$ **Mesh Runtime**




**Rig250** $1 - 2_{653M}$ **Mesh Runtime**

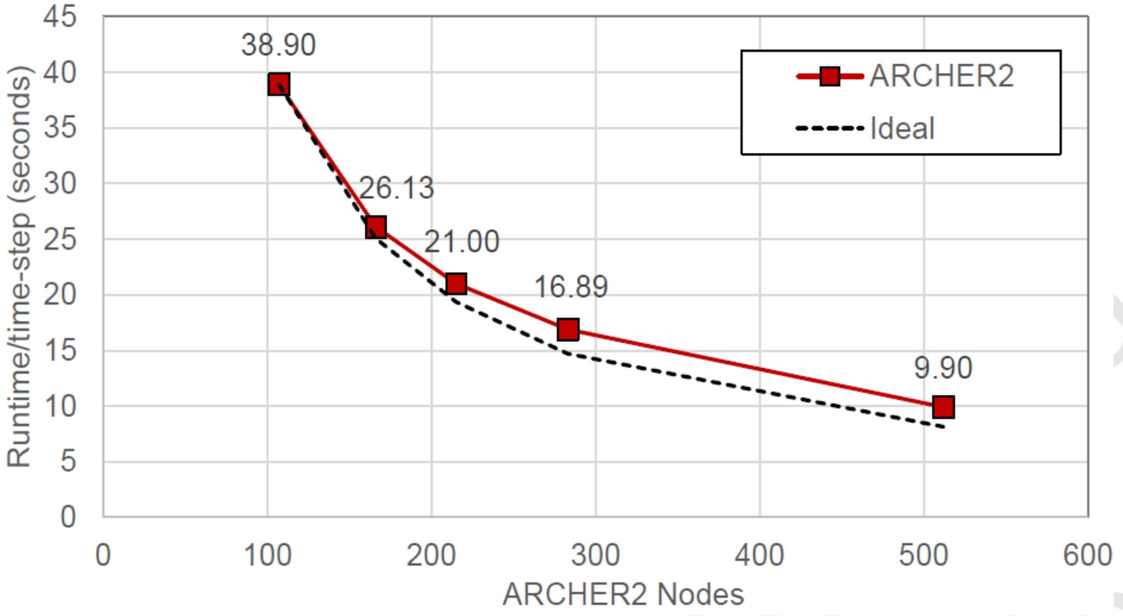
- ❑ ARCHER2 @ 34 nodes
  - 94% parallel efficiency
  - 10% coupling overhead
- ❑ Cirrus @ 25 nodes
  - 94% parallel efficiency
  - 20% coupling overhead
- ❑ ARCHER2 @ 82 nodes
  - 82% parallel efficiency
  - 20% coupling overhead

3.7- 4x speedup

- ❑ ARCHER2 @ 80 nodes
  - 88% parallel efficiency
  - 8% coupling overhead
- ❑ Cirrus @ 22 nodes
  - 94% parallel efficiency
  - 12% coupling overhead
- ❑ Cirrus @ 22 nodes
  - 94% parallel efficiency
  - 12% coupling overhead

3.3 - 3.4x speedup

* Results under review




**Rig250 $1-10_{4.58B}$ Mesh Runtime**

**Achieved (A) and *Projected (P)* times to solution (hours) : Rig250, 1 revolution**

| Rig250 Problem | ARCHER2 Runime | ARCHER2 #nodes | Cirrus Runtime | Cirrus #nodes |
|---|---|---|---|---|
| $1-10_{430M}$ - Monolithic | *93.0 (P)* | *8* | | |
| $1-10_{430M}$ - Coupled | *85.0 (P)* | *8* | *2.9 (P)* | *15* |
| $1-10_{430M}$ - Coupled | *3.3 (P)* | *80* | *1.8 (P)* | *25* |
| $1-2_{653M}$ - Monolithic | *110.0 (P)* | *8* | | |
| $1-2_{653M}$ - Coupled | *40.0 (P)* | *8* | *3.9 (P)* | *17* |
| $1-2_{653M}$ - Coupled | *8.2 (P)* | *40* | *3.2 (P)* | *22* |
| $1-10_{4.58B}$ - Coupled | *14.5 (A)* | *166* | *4.7 (P)* | *122* |
| $1-10_{4.58B}$ - Coupled | *9.4 (A)* | *256* | | |
| $1-10_{4.58B}$ - Coupled | *5.5 (A)* | *512* | | |

- ARCHER2 @ 512 nodes:
  - 82% parallel efficiency (vs 107 node run)
  - 15% coupling overhead

- 122 Cirrus nodes is power equivalent to 166 ARCHER2 nodes
- ARCHER2 needs just over 3x more number of *power equivalent* nodes (512) to match Cirrus's runtime (4.7 hours)

* Results under review

# OTHER PRODUCTION APPLICATIONS – OPENSBLI (UNI. OF SOUTHAMPTON)

- Compressible Navier-Stockes solver
  - With shock capturing WENO/TENO
  - 4th order Finite Difference
  - Single/double precision

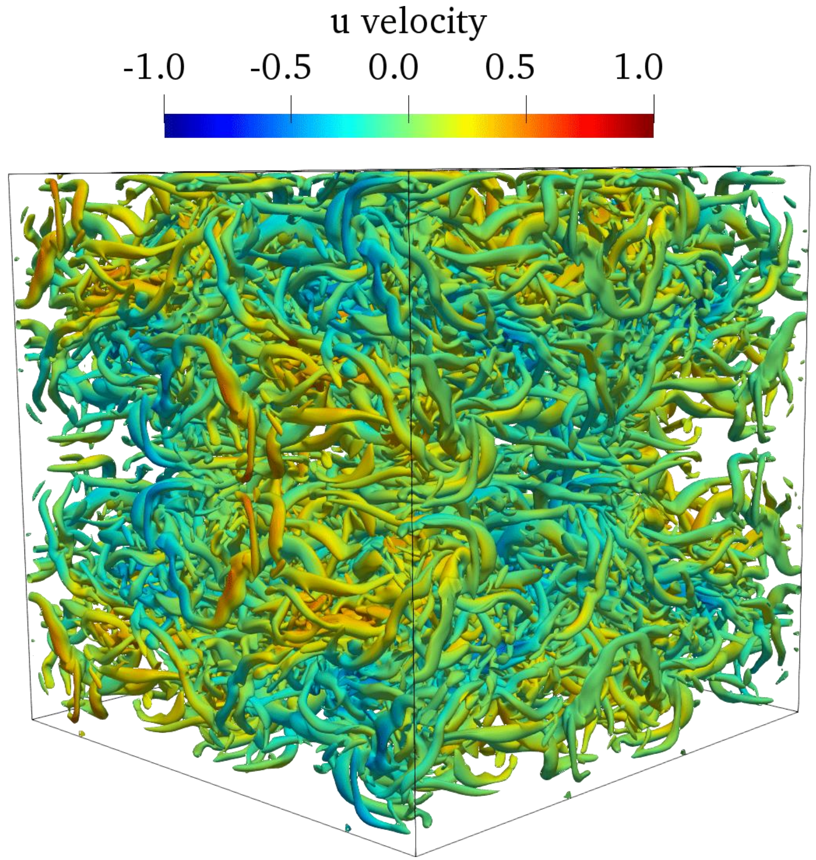
- OpenSBLI is a Python framework
  - Write equations in SymPy expressions
  - OPS code generated

```
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**

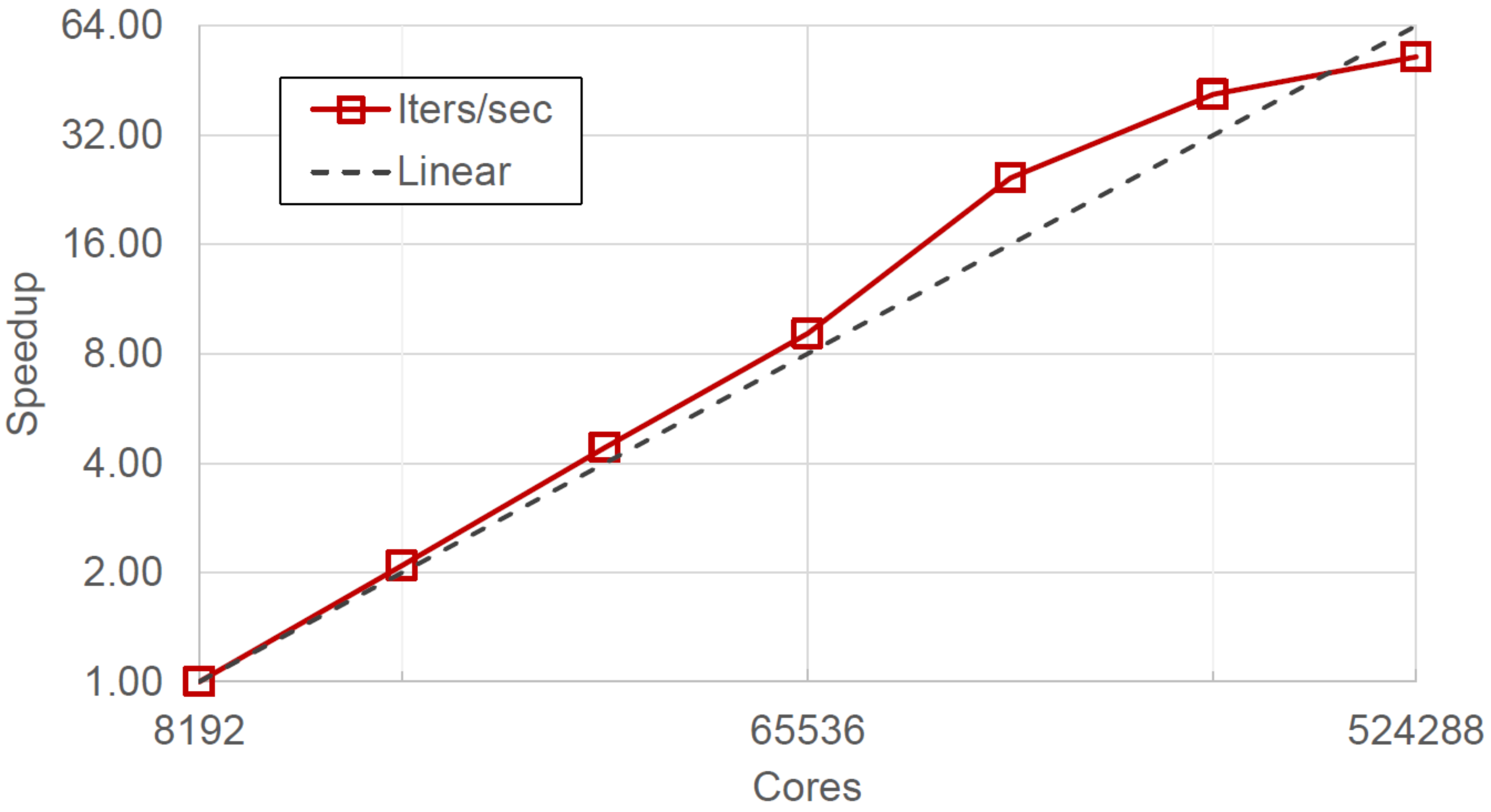
https://opensbli.github.io/



# 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)



# 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

# 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 –cross loop, loop fusion and unrolling to create longer and longer pipelines !
- 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

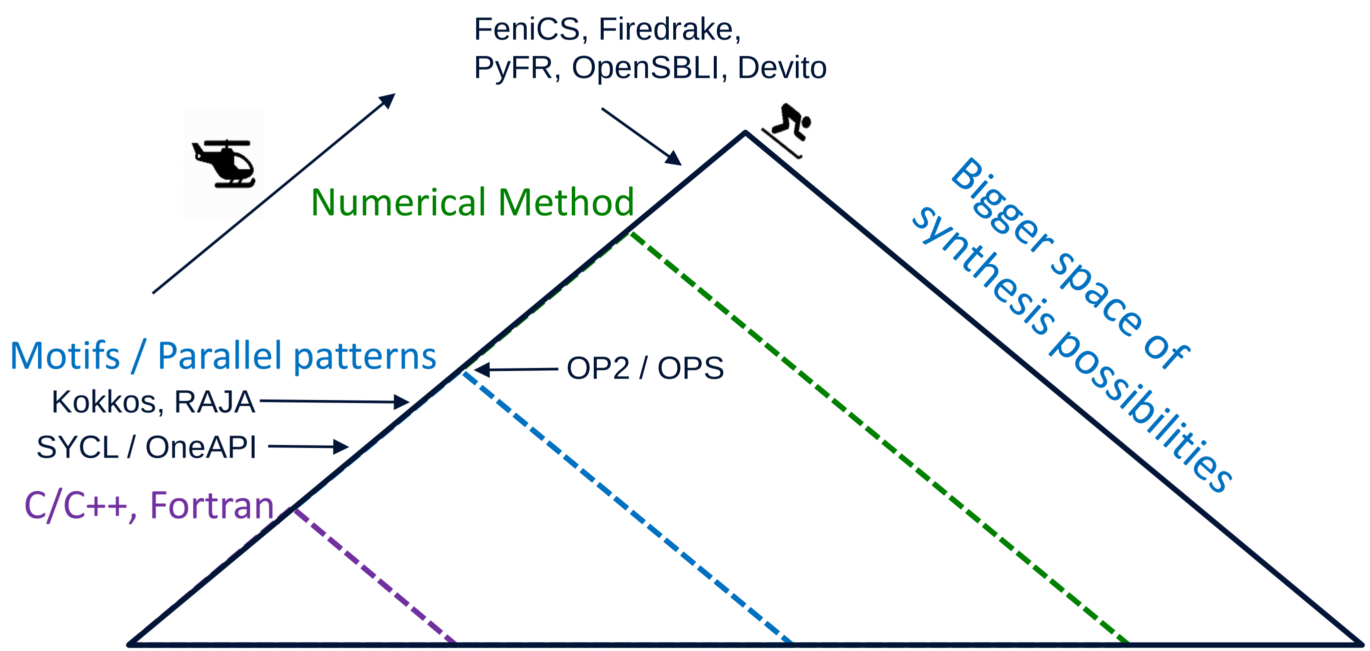
- Long term funding and maintenance
  - Once established probably will not be different to any other classical library
  - Will require compiler expertise to maintain code generation tools

- 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
  - [ In the UK ] Salary ☹
  - [ In the UK ] 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)

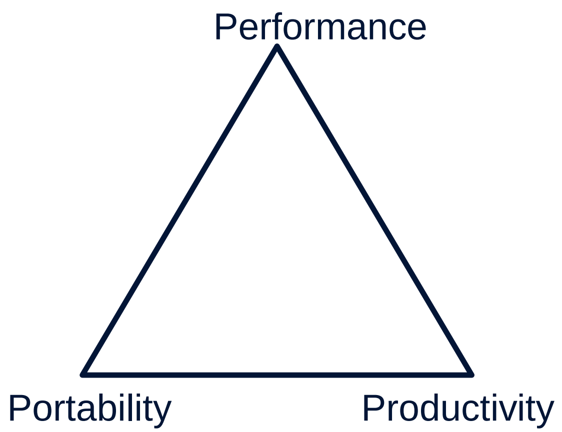
Separation of Concerns – One of the four pillars of ExCALIBUR

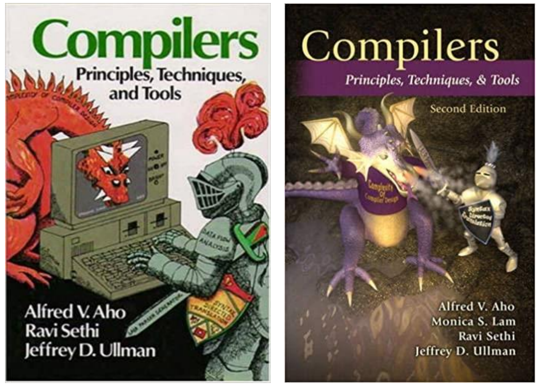
# Lessons Learnt and Conclusions

- 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/

OP-DSL

- 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