Approaches to Developing Performance Portable Scientific Software

Steven A. Wright, Christopher Ridgers
University of York, York, UK

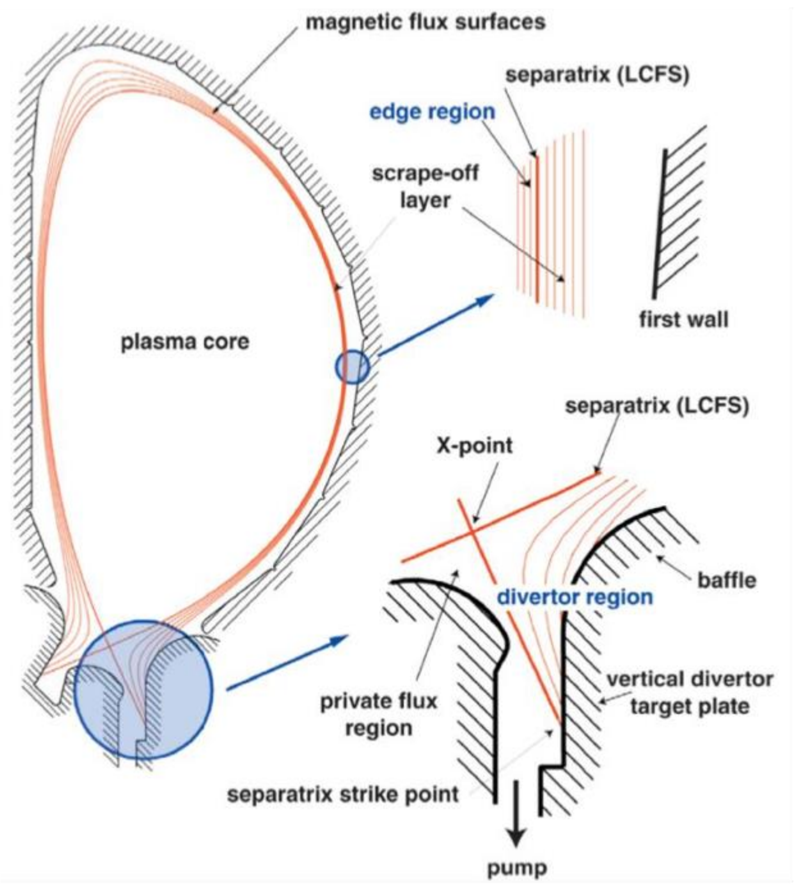
Gihan Mudalige, Zaman Lantra
University of Warwick, Coventry, UK

Josh Williams, Andrew Sunderland, Sue Thorne
Hartree Centre, STFC Daresbury Laboratory, Warrington, UK

Context

Project NEPTUNE (NEutrals & Plasma TUrbulence Numerics for the Exascale)

- ***Fusion Modelling System*** **use case of ExCALIBUR**
- **Develop software to make efficient use of current Petascale and future Exascale hardware**
  - **in order to draw insights from ITER**
  - **to guide and optimise the design of the UK demonstration nuclear fusion power plant STEP**
- **Initial focus on the edge and divertor regions**
- **Our work is on investigating approaches to developing a *performance portable* code**



Challenges in Developing Modern Parallel Applications

Pre- and Post-Exascale Hardware

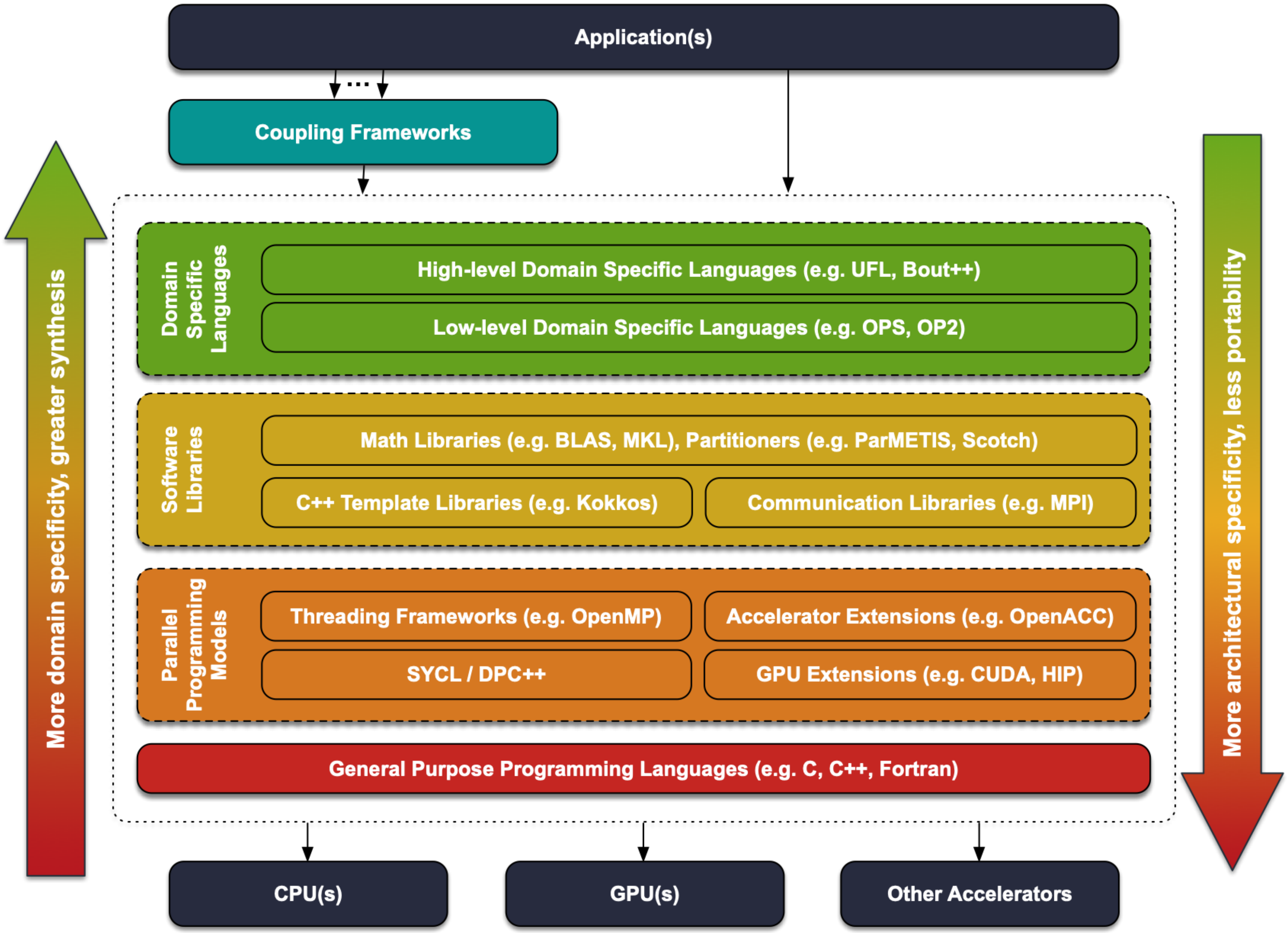
- All pre- and post-Exascale systems will be (or are) heterogenous (… except Fugaku)
- Most of the FLOP/s will be provided by GPU accelerators
  - NVIDIA Hopper
  - AMD Instinct
  - Intel Xe
- Most systems will use x86_64 hosts from Intel and AMD
  (+ perhaps some NVIDIA Grace)

Challenges in Developing Modern Parallel Applications

Developing Applications for Exascale

- How do we achieve *Performance*, *Portability*, and *Productivity* on Exascale systems?
- MPI+*X* likely for Exascale systems
  - MPI for inter-node
  - *X* for intra-node and accelerators

Challenges in Developing Modern Parallel Applications



- Review has focussed on
  - Programming languages
  - Parallel programming models
  - Software libraries
  - Domain specific languages
  - Coupling frameworks
- Assessment of 3Ps

General Purpose Programming Languages

Traditional programming languages with established history in scientific computing

- Fortran and C/C++ dominate HPC
  - Fortran codes account >50% ARCHER2 time, C/C++ >30%
- Python not traditionally "HPC", but often a glue language
- Julia promising with some "best-in-class" libraries

Considerations:

- Languages very prescriptive, optimisation may reduce portability and maintainability
- Multiple code paths may be required, duplicating development and maintenance
- Parallelism typically explicit, significantly increasing complexity

Parallel Programming Models

Extensions providing parallelism on- and off-node, or to accelerators

- **Loop-level parallelism often achieved with OpenMP**
- **MPI is de facto standard for distributed memory parallelism**
  - **Alternatives include Co-array Fortran, UPC**
- **Task-level parallelism available in OpenMP, or Charm++, LEGION, etc.**
- **Extensions targeting accelerators**
  - **CUDA, ROCm/HIP, SYCL/DPC++, OpenCL, OpenACC, OCCA**

Considerations:

- **Open standards sometimes lag hardware development**
- **Complete implementations of standards sometimes slow**
- **Low-level control over parallelism may lead to code specialisation**

Software Libraries

Scientific and mathematical libraries, and libraries that facilitate data- and task-parallelism

- Mathematical libraries provide common mathematical routines
  - Most based on BLAS, LAPACK, FFTW, optimised by vendors (e.g. MKL)
- Data libraries provide partitioning, data structures
  - Common examples include PETSc, METIS, Scotch
- C++ template libraries as parallel programming models
  - Kokkos, RAJA, Thrust

Considerations:

- Standard interfaces restrict use, but encourages vendor optimisation
- Library functions often work in lock-step, restricting fusing of operations
- Template libraries restrict use to modern C++
- Templates can increase compilation time and obfuscate errors
- But, platform specific code can be easily integrated into templated code

Domain Specific Languages

Languages and libraries limited to a particular application or algorithmic domain

- Sacrificing generality perhaps makes it feasible to achieve all 3 *P*s
- Low-level DSLs focus on parallel computation patterns
  - Mesh-based DSLs: Halide, YASK, OP-DSLs, PSyclone
  - Particle-based DSLs: PPML, PPME, OpenFPM, PPMD
- High-level DSLs focus on specific numerical methods
  - Finite differences, finite volume, finite element: FEniCS, Firedrake, ExaStencils, Bout++

Considerations:

- Debugging may be more difficult because of hidden layers
- Extensibility and customisability requires additional expertise
- There may be escape hatches, but this breaks the abstraction

Coupling Frameworks

Libraries acting as interfaces to enable communication between applications

- **Multiscale problems require different models that can interact (e.g. fluid and particle models)**
- **Typically flexible and lightweight**
  - **Minimal effect on performance and portability**
- **Examples include preCICE, CWIPI, MUI**

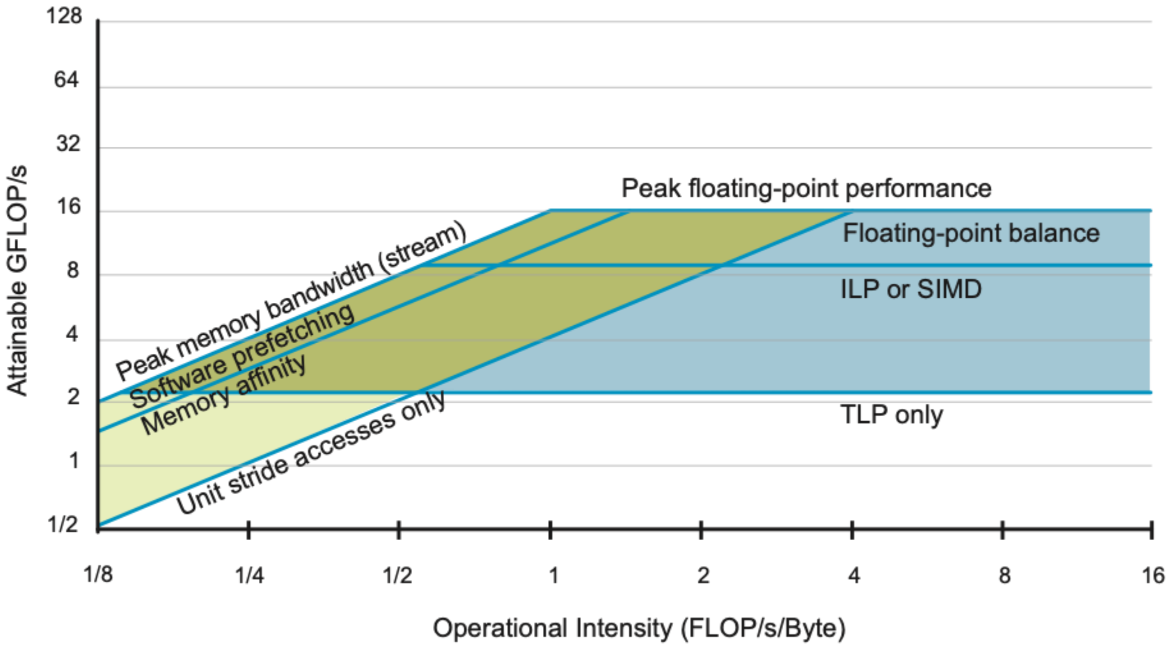
Considerations:

- **Performance of communication and coupling numerics**
- **Ease of use (and minimal intrusion)**

Evaluating Performance, Portability and Productivity

Metrics and heuristics for measuring the 3 *Ps*

- **Performance typically measured by metrics or proxies for "time-to-science"**
  - **Runtime, FLOP/s, Memory bandwidth, Energy, etc.**
- **Roofline model [1] helps us reason about performance compared to potential**



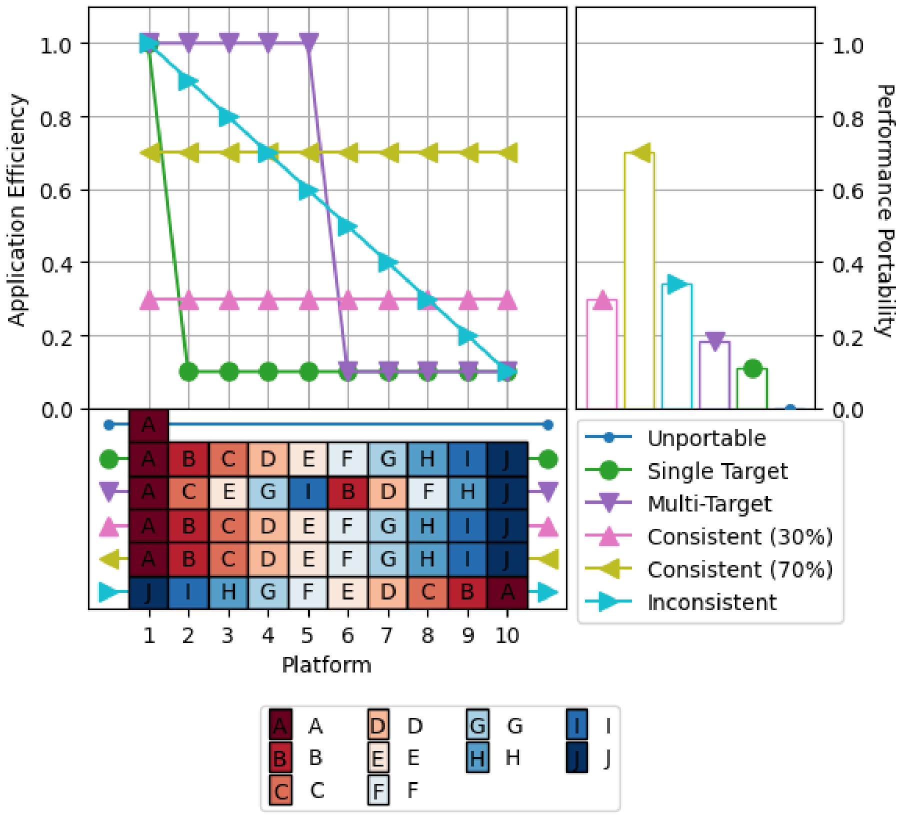
[1] S. Williams, A. Waterman, D. Patterson, Roofline: An Insightful Visual Performance Model for Multicore Architectures, Commun. ACM 52 (2009) 65–76.

Evaluating Performance, Portability and Productivity

Metrics and heuristics for measuring the 3 *Ps*

- **Although portability is a binary measure, we care about *portable performance***
- **One such metric and visual heuristic from Pennycook et al. [2] and Sewall et al. [3]**

$$\Phi(a,p,H) = \begin{cases} \dfrac{|H|}{\sum_{i \in H} \dfrac{1}{e_i(a,p)}} & \text{if } i \text{ is supported } \forall i \in H \\ 0 & \text{otherwise} \end{cases}$$



[2] S.J. Pennycook, J.D. Sewall, and V.W. Lee. Implications of a metric for performance portability. Future Generation Computer Systems, 92:947 –958, 2019.
[3] J.D. Sewall, S.J. Pennycook, D. Jacobsen, T. Deakin, and S. McIntosh-Smith. Interpreting and visualizing performance portability metrics. In 2020 P3HPC Workshop, pages 14–24, 2020.

Evaluating Performance, Portability and Productivity

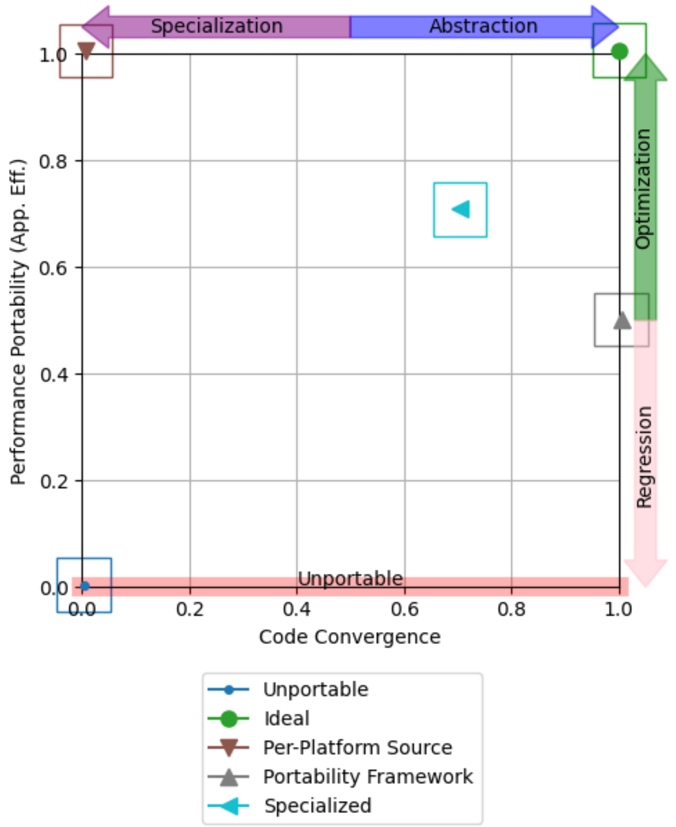
Metrics and heuristics for measuring the 3 *Ps*

- **Developer productivity is perhaps the most difficult to assess objectively**
  - **Proxies: LoC, Dev time, Code complexity**
- **Harrell et al. [4] propose Code Divergence**

$$\mathrm{CD}(a,p,H) = \binom{|H|}{2}^{-1} \sum_{\{i,j\}\in H\times H} d_{i,j}(a,p)$$

$$d_{i,j}(a,p) = 1 - \frac{|c_i(a,p) \cap c_j(a,p)|}{|c_i(a,p) \cup c_j(a,p)|}$$

- **Can be combined with *Performance Portability* on a Navigation Chart [5]**



[4] S. L. Harrell, J. Kitson, R. Bird, S. J. Pennycook, J. Sewall, D. Jacobsen, D. N. Asanza, A. Hsu, H. C. Carrillo, H. Kim, R. Robey, Effective performance portability, in: 2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC), 2018, pp. 24–36
[5] S. J. Pennycook, J. D. Sewall, D. W. Jacobsen, T. Deakin, S. McIntosh-Smith, Navigating Performance, Portability, and Productivity, Computing in Science & Engineering 23 (2021) 28–38

Summary

- New simulation software likely to employ software and DSLs at many different levels of software development stack
    - High-level DSLs for users to express equations directly
    - Low-level DSLs and programming models targeting different architectures
- Targeting high performance, portability and productivity from a single code base is challenging!
- There are a number of metrics, tools and visual heuristics to guide developers and measure success

Approaches to Developing Performance Portable Scientific Software

Steven A. Wright, Christopher Ridgers
University of York, York, UK

Gihan Mudalige, Zaman Lantra
University of Warwick, Coventry, UK

Josh Williams, Andrew Sunderland, Sue Thorne
Hartree Centre, STFC Daresbury Laboratory, Warrington, UK