



Code Portability and Performance

26th and 28th April 2022



Dr Tom Deakin

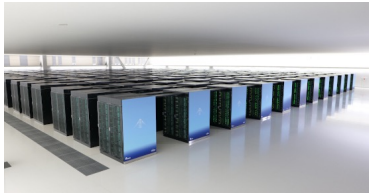
- Lecturer in Advanced Computer Systems
University of Bristol
- Khronos SYCL Outreach Officer
- Chair Khronos SYCL Advisory Panel
- Member of OpenMP ARB and Khronos SYCL Working Group
- Programming GPUs with OpenMP Tutorials
- Author of OpenMP for Computational Scientists Tutorial
- Co-Author of HandsOnOpenCL
- Twitter: @tjdeakin
- Email: tom.deakin@bristol.ac.uk
- Web: <https://hpc.tomdeakin.com>

Agenda

- Performance Portability
 - What is Performance Portability?
 - How to measure Performance Portability and efficiency
 - Performance Portability Metric
- Consistency of performance portability
 - Cascade Plots
- The Productivity Dimension
- Writing Performance Portable Applications
 - ... in SYCL and OpenMP

Performance Portability

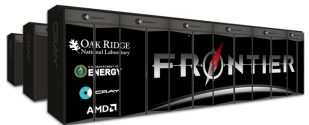
Processor diversity at (pre-)Exascale



At RIKEN: Fujitsu A64fx CPUs



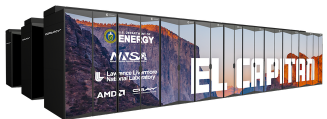
At NERSC: AMD EPYC Milan CPUs and NVIDIA A100 GPUs



At ORNL: AMD EPYC custom CPUs and Radeon Instinct GPUs (4 per node)



At ALCF: Intel Xeon Sapphire Rapids CPUs and Xe Ponte Vecchio GPUs (6 per node)



At LLNL: AMD EPYC Genoa CPUs and Radeon Instinct GPUs (4 per node)

Recent architectural trends



CPUs

- Many “complex” cores (80 per socket).
- Wide vectors (AVX-512, SVE 128-2048 bits).
- Chiplet manufacturing.
- Deep cache hierarchy. NUMA.
- Mainly DRAM, but...
 - Intel Xeon Phi MCDRM
 - Fujitsu A64FX HBM2
 - NVIDIA Grace LPDDR5x

GPUs

- Lots of “lightweight” cores.
- Very wide vector units (warp).
- Cores becoming more complex:
 - Specialised in-core accelerators.
- Interconnects (NVLink).
- Latest (specialised) memory technology:
 - GDDR
 - HBM
- Deepening memory hierarchy:
 - Caches, scratchpad (shared), ...

What is performance portability?

“A code is performance portable if it can achieve a similar fraction of peak hardware performance on a range of different target architectures”

- Needs to be a good fraction of best achievable (i.e., hand optimised).
- Range of architectures depends on your goal, but important to allow for future developments.

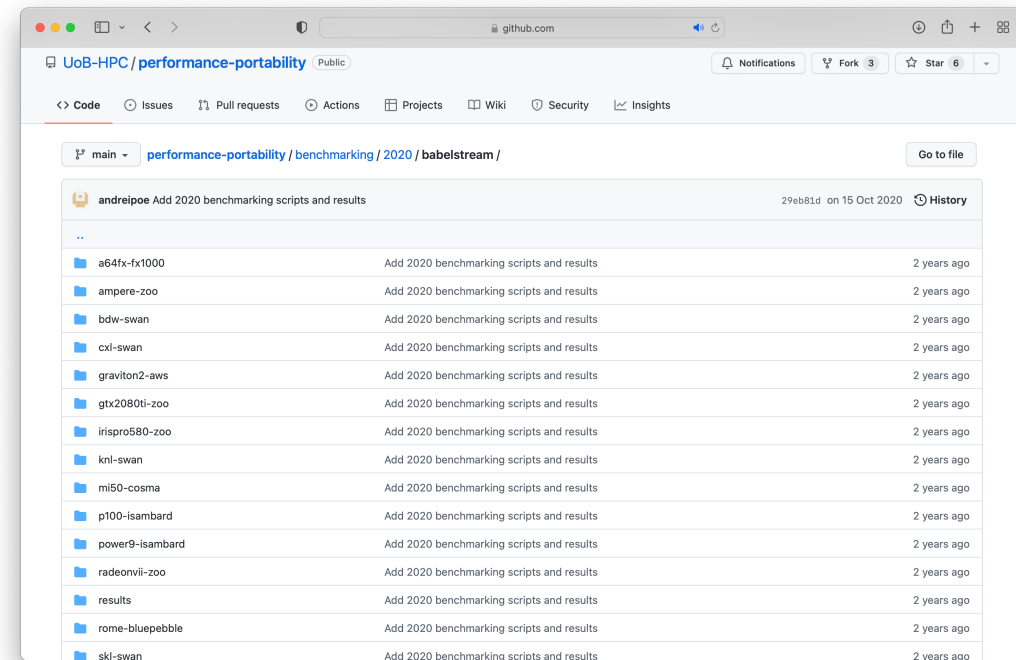


Measuring efficiency

- Compare relative application performance on different processors.
- Processors have different performance characteristics.
- Architectural efficiency:
 - Percentage of peak hardware performance.
 - E.g. achieved GB/s or FLOP/s vs theoretical tech sheet.
- Application efficiency:
 - Performance relative to specialised, hand-tuned, unportable, “best” version.
 - I.e. vs “World record”.

Collecting performance data

- Compiling codes on multiple systems crucial
- Compiling codes with multiple compilers crucial
- Document compile and execution steps in systematic and reproducible scripts



<https://github.com/UoB-HPC/performance-portability>

Core-bound, or not core-bound?

- Follow a procedure by Voysey (Met Office) to help discover performance limiting factors:
 1. Run on all cores of one socket. (e.g. 18 cores of one Broadwell socket)
 2. Run on half of cores of both sockets. (e.g. 2 x 9 cores)
- If performance improves, performance is bound in shared resources such as memory bandwidth.
 - E.g. Two sockets give you twice the main memory bandwidth of one socket.
- Otherwise, bound by on-core resources.
 - Same number of cores, so have same number of FLOPs, same cache bandwidth/size, etc.
- Warning! Sometimes see increase in clock speed for the two-socket run.

BabelStream

- Benchmarks achievable (main) memory bandwidth.
- Based on McCalpin STREAM, except:
 - Arrays allocated on the heap.
 - Problem size known only at runtime.
- Written in many programming models.
- Constructed of simple vector operations, e.g.:
 - Copy: $c[i] = a[i]$
 - Mul: $b[i] = \text{scalar} * c[i]$
 - Add: $c[i] = a[i] + b[i]$
 - Triad: $a[i] = b[i] + \text{scalar} * c[i]$

<https://github.com/UoB-HPC/BabelStream>



Modelling memory bandwidth

- Arrays of size N FP64 elements
- Read B and C: 2N
- Write A: N
- Total $3N * \text{sizeof}(\text{double})$
= $3 * N * 8$ bytes
= $24 * N$ bytes
- Divide by runtime to get
bytes/second
- Multiply by $1\text{E-}9$ to get GB/sec
(base 10)
- Compare to theoretical peak for
architectural efficiency

```
void triad() {  
  
    #pragma omp parallel for  
    for (int i = 0; i < array_size; i++) {  
        a[i] = b[i] + scalar * c[i];  
    }  
  
}
```

BabelStream heatmaps

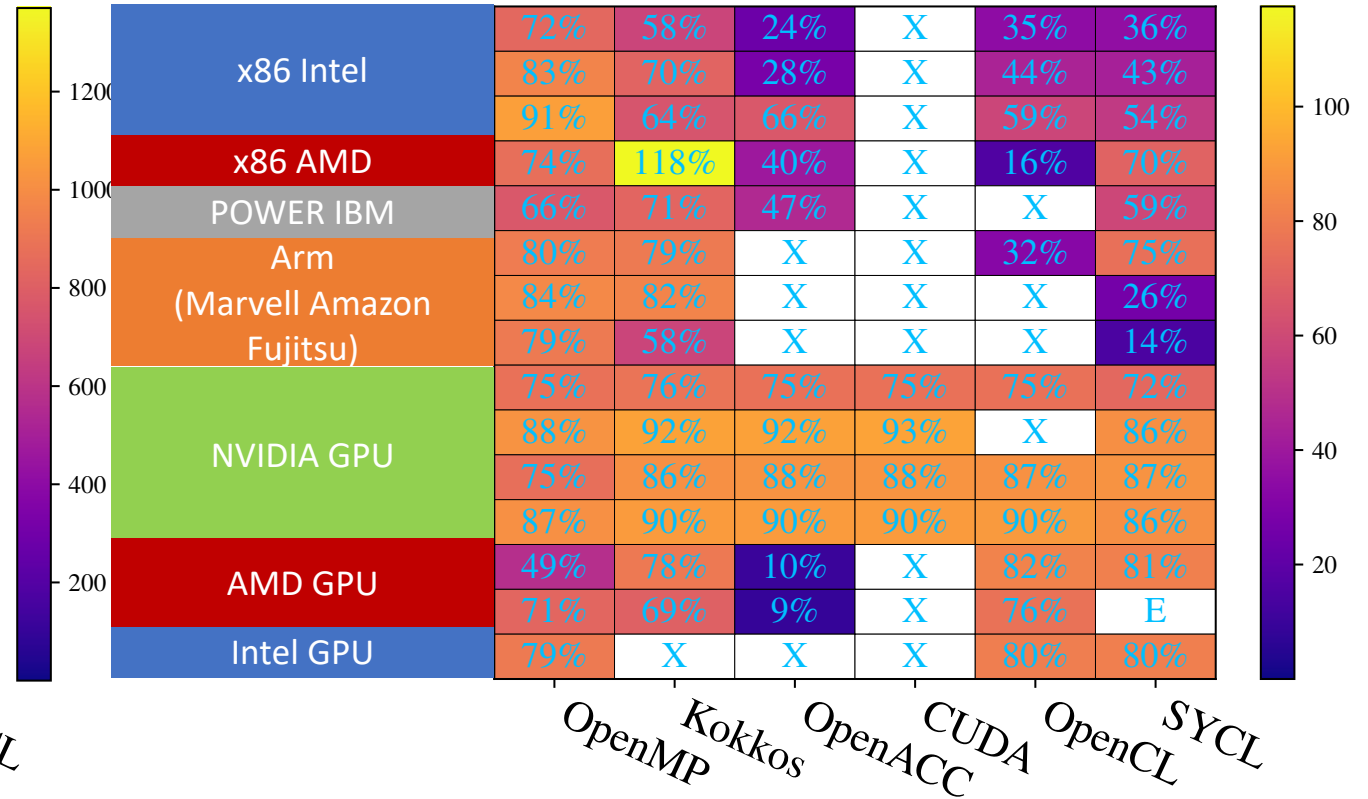
Peak performance

BabelStream Triad array size=2**25

| | | | | | | |
|-----------------|--------|--------|---------|------|--------|------|
| Cascade Lake | 203 | 163 | 68.5 | X | 98.0 | 100 |
| Skylake | 211 | 180 | 70.6 | X | 113 | 110 |
| Knights Landing | 448 | 315 | 322 | X | 287 | 263 |
| Rome | 305 | 481 | 162 | X | 64.9 | 287 |
| Power 9 | 224 | 241 | 158 | X | X | 200 |
| ThunderX2 | 229 | 226 | X | X | 93.5 | 217 |
| Graviton 2 | 173 | 169 | X | X | X | 54.3 |
| A64FX | 804 | 595 | X | X | X | 147 |
| P100 | 552 | 559 | 551 | 551 | 552 | 526 |
| V100 | 788 | 831 | 830 | 837 | X | 774 |
| A100 | 1161 | 1343 | 1361 | 1370 | 1356 | 1358 |
| Turing | 533 | 555 | 555 | 556 | 554 | 530 |
| Radeon VII | 488 | 781 | 99.0 | X | 821 | 808 |
| MI50 | 729 | 708 | 88.3 | X | 778 | E |
| IrisPro Gen9 | 26.8 | X | X | X | 27.3 | 27.4 |
| | OpenMP | Kokkos | OpenACC | CUDA | OpenCL | SYCL |

Architectural efficiency

BabelStream Triad array size=2**25



Performance Portability metric

Pennycook, Sewall and Lee: <https://doi.org/10.1016/j.future.2017.08.007>

$$\mathcal{P}(a, p, H) = \begin{cases} \frac{|H|}{\sum_{i \in H} \frac{1}{e_i(a, p)}} & \text{if, } \forall i \in H \\ & e_i(a, p) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

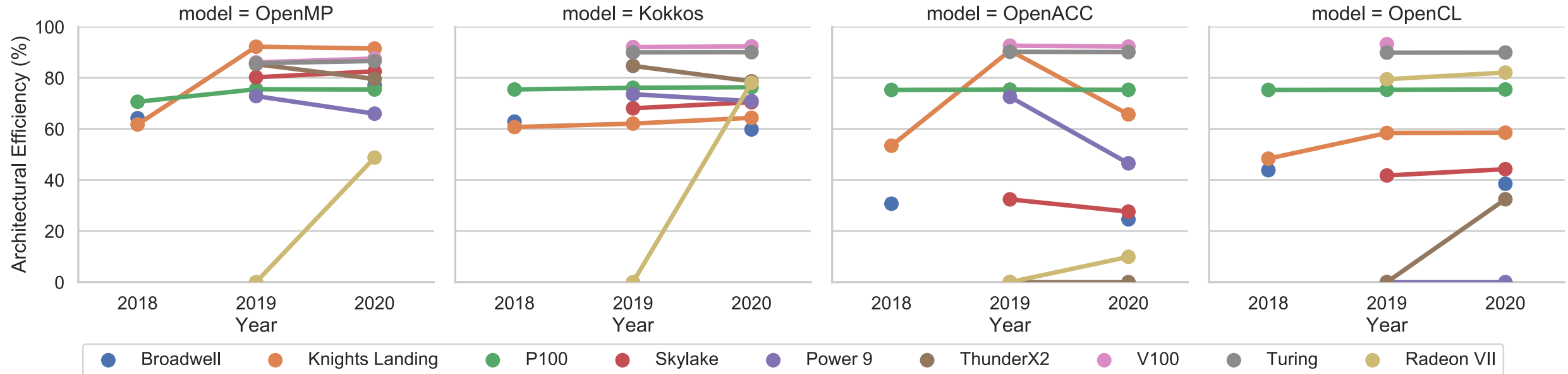
```
from statistics import harmonic_mean
def pp(n):
    if 0 in n:
        return 0
    return harmonic_mean(n)
```

Python scripts: <https://github.com/UoB-HPC/performance-portability/tree/main/metrics>

BabelStream Triad PP metric

| Platforms | OpenMP | Kokkos | OpenACC | CUDA | OpenCL | SYCL |
|------------------|---------------|---------------|----------------|-------------|---------------|-------------|
| All | 75.1 | 0 | 0 | 0 | 0 | 0 |
| All non-zero | 75.1 | 75.4 | 27.3 | 86.1 | 46.6 | 47.4 |
| Supported CPUs | 77.9 | 71.6 | 35.9 | 0 | 30.8 | 36.1 |
| Supported GPUs | 72.2 | 81.2 | 22.8 | 86.1 | 81.4 | 81.7 |

How far have we come?



How far has SYCL come?

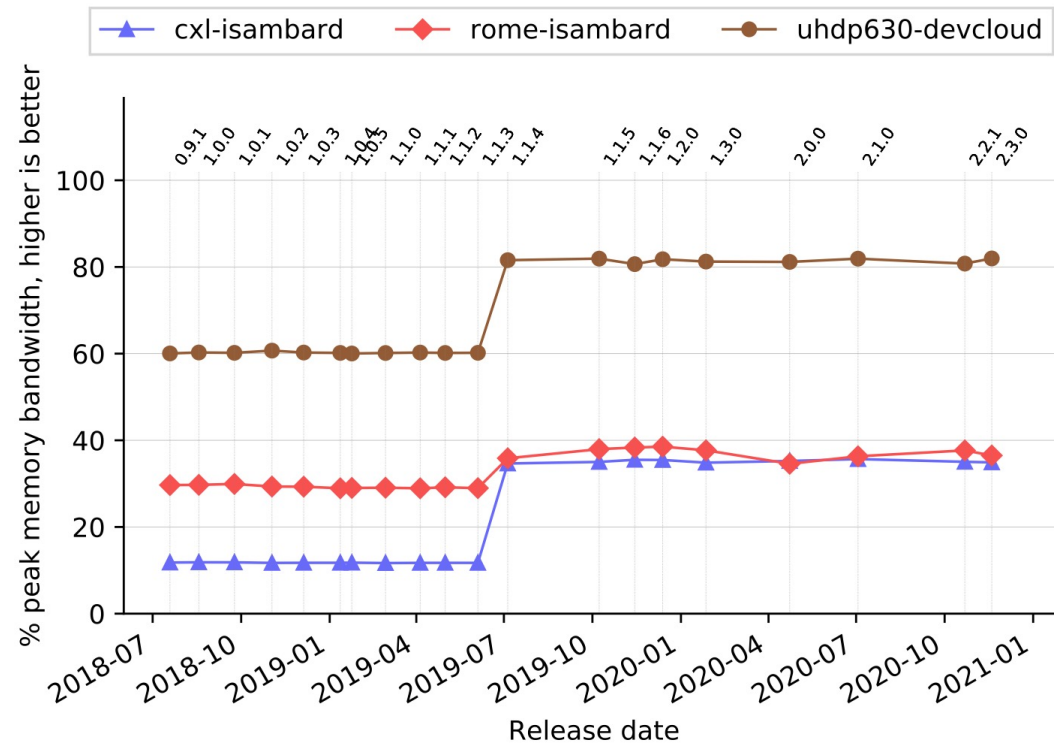


Figure 1: BabelStream on ComputeCpp results

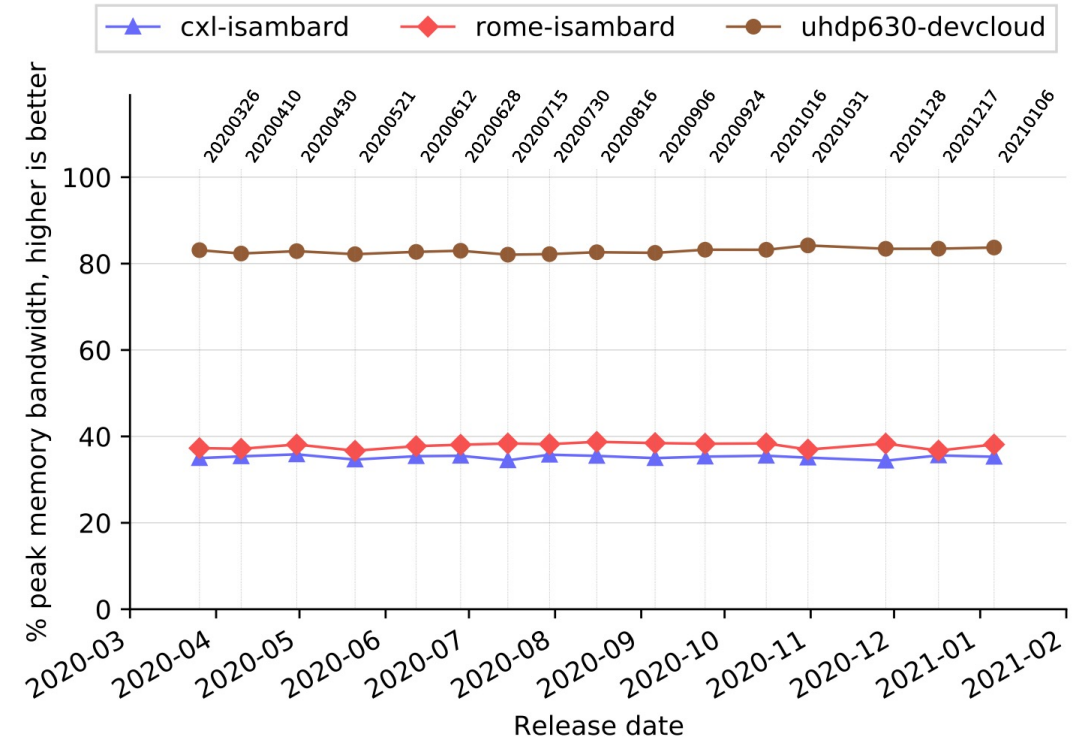


Figure 4: BabelStream on DPC++ results

Consistency of Performance Portability

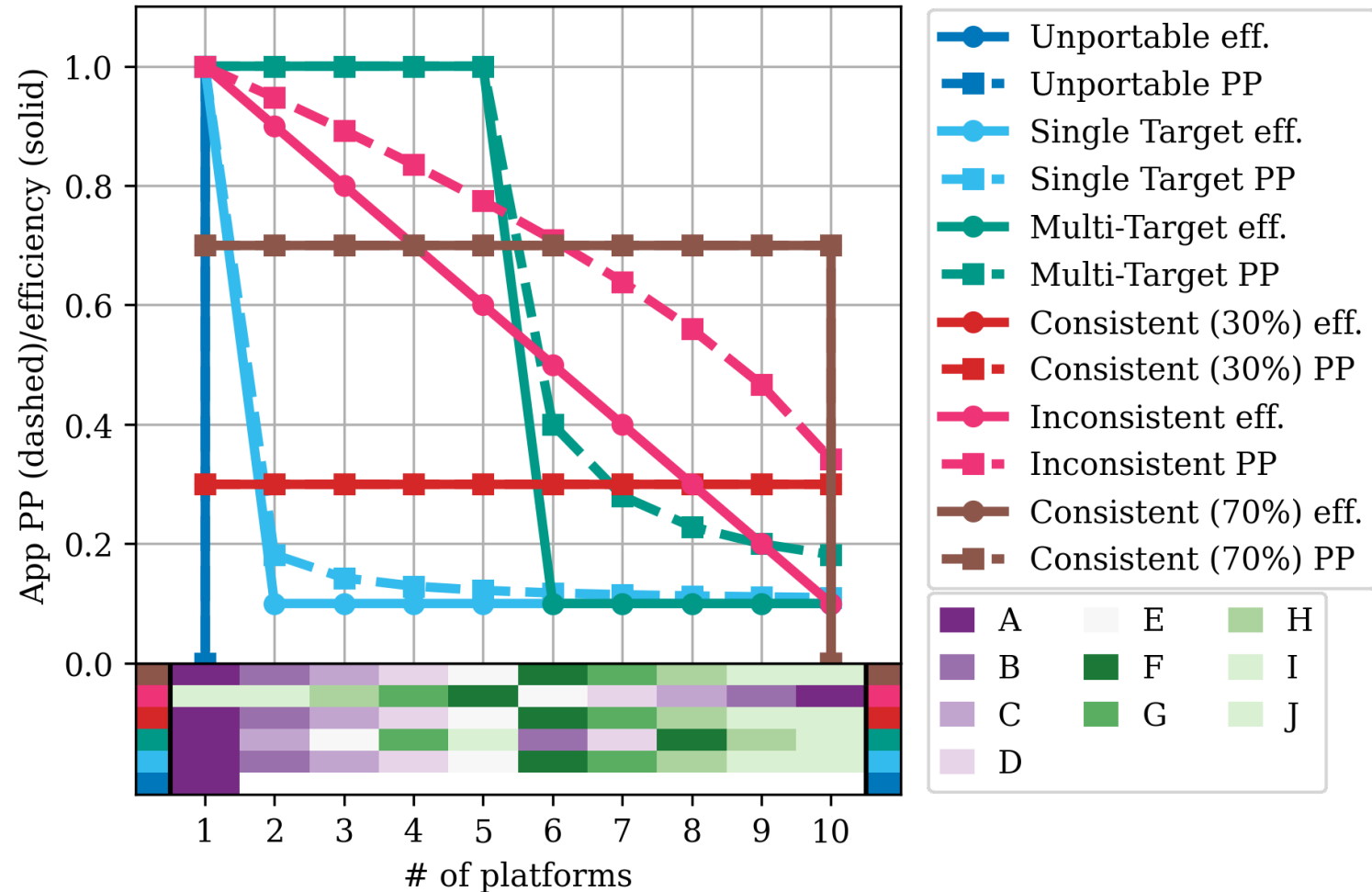
What is performance portability?

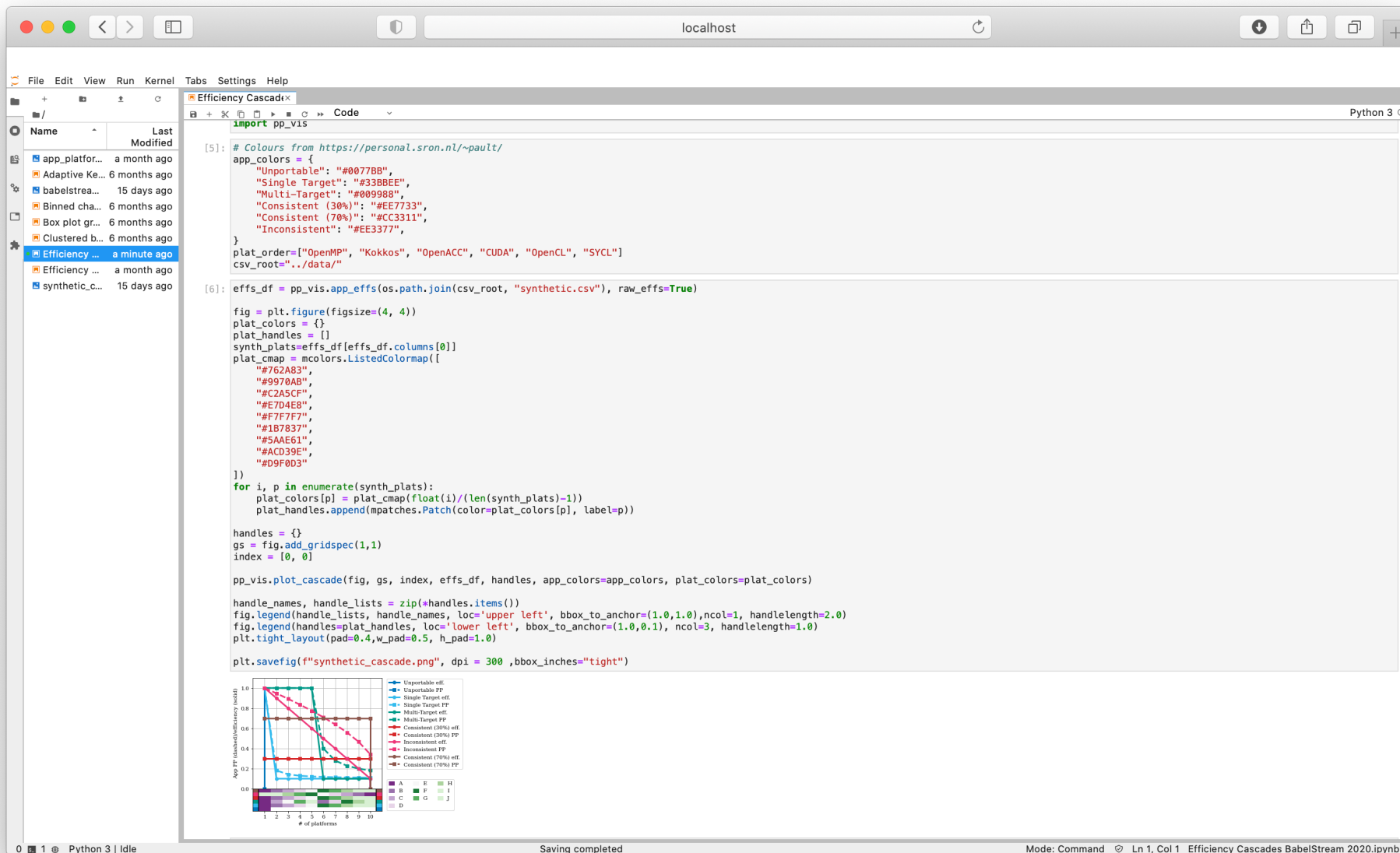
“A code is performance portable if it can achieve a similar fraction of peak hardware performance on a range of different target architectures”

- Needs to be a good fraction of best achievable (i.e., hand optimised).
- Range of architectures depends on your goal, but important to allow for future developments.



Cascade plots





The screenshot shows a Jupyter Notebook window titled 'Efficiency Cascade' with the following Python code:

```
[5]: # Colours from https://personal.sron.nl/~pault/
app_colors = {
    "Unportable": "#0077BB",
    "Single Target": "#3388EE",
    "Multi-Target": "#009988",
    "Consistent (30%)": "#EE7733",
    "Consistent (70%)": "#CC3311",
    "Inconsistent": "#EE3377",
}

plat_order=["OpenMP", "Kokkos", "OpenACC", "CUDA", "OpenCL", "SYCL"]
csv_root="../../data/"

[6]: effs_df = pp_vis.app_effs(os.path.join(csv_root, "synthetic.csv"), raw_effs=True)

fig = plt.figure(figsize=(4, 4))
plat_colors = {}
plat_handles = []
synth_plats=effs_df[effs_df.columns[0]]
plat_cmap = mcolors.ListedColormap([
    "#762A83",
    "#9770AB",
    "#C2A5CF",
    "#E704EB",
    "#F7F7F7",
    "#1B7837",
    "#5AAE61",
    "#ACD39E",
    "#D9F0D3"
])

for i, p in enumerate(synth_plats):
    plat_colors[p] = plat_cmap(float(i)/(len(synth_plats)-1))
    plat_handles.append(mpatches.Patch(color=plat_colors[p], label=p))

handles = {}
gs = fig.add_gridspec(1,1)
index = [0, 0]

pp_vis.plot_cascade(fig, gs, index, effs_df, handles, app_colors=app_colors, plat_colors=plat_colors)

handle_names, handle_lists = zip(*handles.items())
fig.legend(handle_lists, handle_names, loc='upper left', bbox_to_anchor=(1.0,1.0), ncol=1, handlelength=2.0)
fig.legend(handles=plat_handles, loc='lower left', bbox_to_anchor=(1.0,0.1), ncol=3, handlelength=1.0)
plt.tight_layout(pad=0.4,w_pad=0.5, h_pad=1.0)

plt.savefig(f"synthetic_cascade.png", dpi = 300 ,bbox_inches="tight")
```

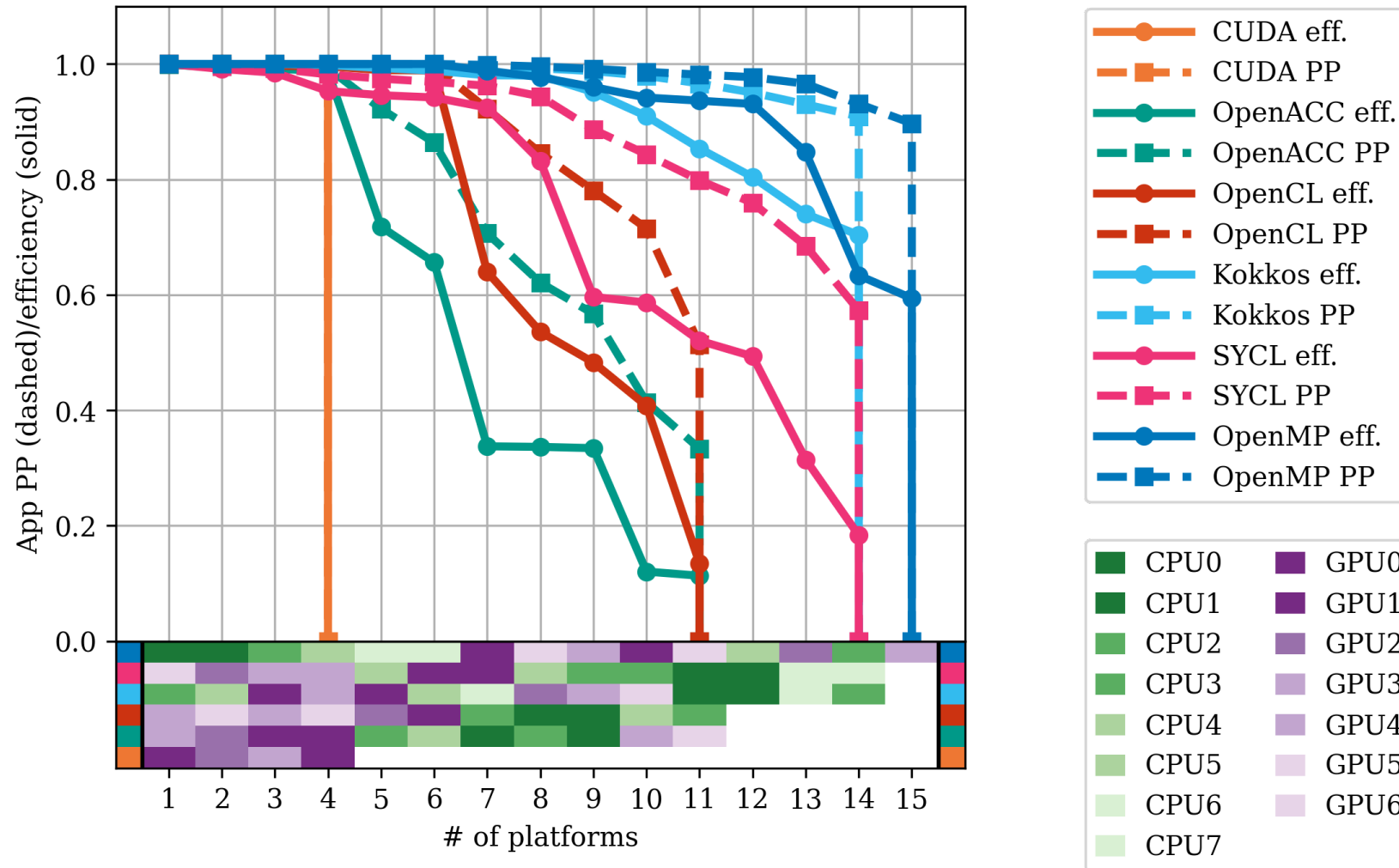
The plot shows the 'app PP (in % of reference)' on the y-axis (ranging from 0.0 to 1.0) versus the '# of platforms' on the x-axis (ranging from 1 to 10). The plot displays various performance metrics for different platforms, categorized by color and line style. The legend includes:

- Unportable eff.
- Unportable PP
- Single Target eff.
- Single Target PP
- Multi-Target eff.
- Multi-Target PP
- Consistent (30%) eff.
- Consistent (30%) PP
- Consistent (70%) eff.
- Consistent (70%) PP
- Inconsistent eff.
- Inconsistent PP
- Consistent (70%) eff.
- Consistent (70%) PP

The plot also includes a color-coded legend for platforms: A (purple), B (green), C (blue), D (red), E (orange), F (yellow), G (light green), H (light blue), I (light purple), J (light pink).

<https://github.com/UoB-HPC/performance-portability/tree/main/metrics/notebooks>

BabelStream Cascade plot





Conferences > 2020 IEEE/ACM International W...

Interpreting and Visualizing Performance Portability Metrics

Publisher: IEEE [Cite This](#) [PDF](#)

Jason Sewall ; S. John Pennycook ; Douglas Jacobsen ; Tom Deakin ; and Simon McIntosh-Smith [All Authors](#)

3 Paper Citations 177 Full Text Views

[R](#) [Share](#) [CC](#) [Folder](#) [Bell](#)

Abstract

Abstract: Recent work has introduced a number of tools and techniques for reasoning about the interplay between application performance and portability, or "performance portability". These tools have proven useful for setting goals and guiding high-level discussions, but our understanding of the performance portability problem remains incomplete. Different views of the same performance efficiency data offer different insights into an application's performance portability (or lack thereof): standard statistical measures such as the mean and standard deviation require careful interpretation, and even metrics designed specifically to measure performance portability may obscure differences between applications. This paper offers a critical assessment of existing approaches for summarizing performance efficiency data across different platforms, and proposes visualization as a means to extract useful information about the underlying distribution. We explore a number of alternative visualizations, outlining a new methodology that enables developers to reason about the performance portability of their applications and how it might be improved. This study unpicks what it might mean to be "performance portable" and provides useful tools to explore that question.

Document Sections

- I. Introduction
- II. Background / Motivation
- III. Single Number Metrics
- IV. Distribution Across Platforms
- V. Impact of Platform Selection

[Show Full Outline](#)

Need Full-Text
access to IEEE Xplore for your organization?
[REQUEST A FREE TRIAL >](#)

More Like This

[Is software aging related to software metrics?](#)
2010 IEEE Second International Workshop on Software Aging and Rejuvenation
Published: 2010

[Uncovering Causal Relationships between Software Metrics and Bugs](#)
2012 16th European Conference on Software Maintenance and Reengineering
Published: 2012

[Feedback](#)

<https://doi.org/10.1109/P3HPC51967.2020.00007>

The Productivity Dimension

Measuring Productivity

- “Ideal” application has one version that is Performant, Portable and Productive.
- Significant specialisation for Performance and/or Portability can impact Productivity.
- Intel Code Base Investigator measures code divergence.
 - Specialisation using C pre-processor.

<https://github.com/intel/code-base-investigator>

PP-CC plane

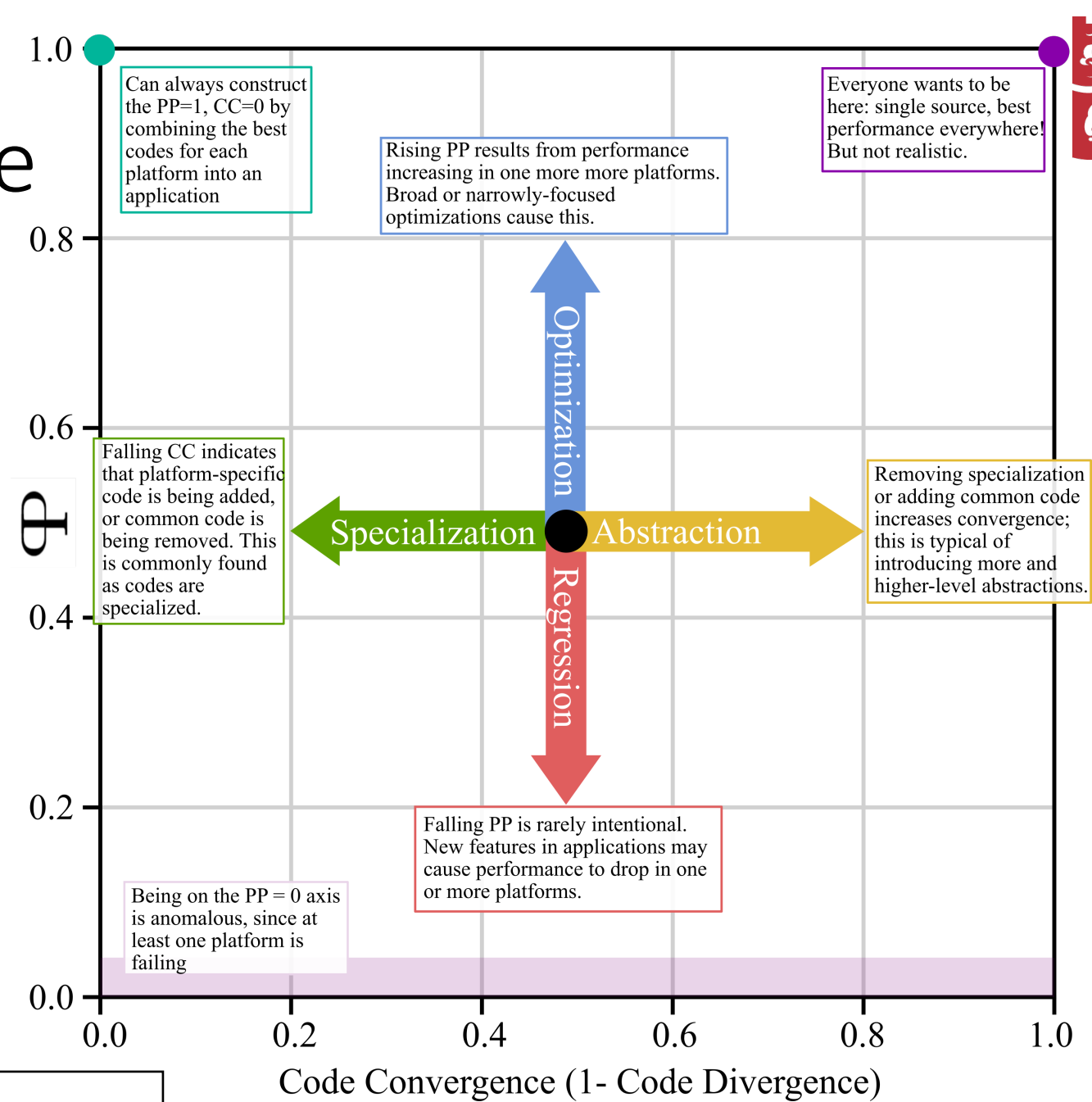


Figure from
<https://doi.org/10.1109/MCSE.2021.3097276>

Writing Performance Portable Applications

Enabling performance portability

Open standard parallel programming models



Open-source programming abstractions



Your favourite
DSL and its
compiler

Descriptive vs Prescriptive

- Descriptive: describe computation, but implementation freedom to decide how
- Prescriptive: all details provided of how to perform computation
- Descriptive is productive, but need the flexibility to prescribe where needed for performance
- Collapse() clause:
 - Good on GPUs for providing lots of parallel work.
 - But care needed with auto-vectorising compilers on CPUs – check the report!
- Avoid the more prescriptive clauses: num_threads, num_teams, thread_limit, etc
 - Let the runtime decide
 - Use environment variables to specialise on platforms if needed

Expressing Parallelism in SYCL

- Data parallel loop:
 - All iterations independent, no barriers
 - `parallel_for(range<1>{1024}, [=] (id<1> it) {...});`
- NDRange:
 - 2 (or 3) level hierarchy: work-items collected into work-groups (and sub-groups)
 - Work-group barriers allowed “anywhere”
 - `parallel_for(nd_range<1>{{1024}, {16}}, [=] (nd_item<1> it) {...});`
- NDRange must allow work-items reach barriers
 - Execution of work-items must “yield” control to allow other work-items to reach barrier
 - On CPUs, work-items need C++ fibers, threads, etc, work-groups are threaded
 - On GPUs: maps naturally to underlying models (CUDA, HIP, OpenCL, ...)

Coalescence

- **Coalesce** - to combine into one
- Coalesced memory accesses are key for high bandwidth
- Simply, it means, if "thread" i accesses memory location n then "thread" $i+1$ accesses memory location $n+1$
- In practice, it's not quite as strict...
- Stride one memory access often maps well to the underlying hardware:
 - SIMD lanes, GPU threads, ...

```
for (int id = 0; id < size; id++)
{
    // ideal
    float val1 = memA[id];

    // still pretty good
    const int c = 3;
    float val2 = memA[id + c];

    // stride size is not so good
    float val3 = memA[c*id];

    // terrible
    const int loc =
        some_strange_func(id);

    float val4 = memA[loc];
}
```

Achieving performance portability

1. **Use open (standard) parallel programming languages** supported by multiple vendors across multiple hardware platforms
 - E.g. OpenMP, SYCL, Kokkos, Raja, ...
2. **Expose maximal parallelism** at all levels of the algorithm and application
 - E.g. target teams distribute parallel for simd
3. **Keep data close** to the processing elements for as long as possible
 - Avoid host/device copies
4. **Avoid over-optimising** for any one platform
 - Optimise for at least two different platforms at once
5. **Multi-objective autotuning** can significantly improve performance
 - **Autotune for more than one target at once**
 - See: **Exploiting auto-tuning to analyze and improve performance portability on many-core architectures**, J.Price and S. McIntosh-Smith, P³MA, ISC'17

Register for free now!



**10th International
Workshop on OpenCL and
SYCL**

May 10-12, 2022 - VIRTUAL

<https://www.iwocl.org>

More information

S. J. Pennycook, J. D. Sewall, D. W. Jacobsen, T. Deakin and S. McIntosh-Smith, "Navigating Performance, Portability, and Productivity," in Computing in Science & Engineering, vol. 23, no. 5, pp. 28-38, 1 Sept.-Oct. 2021, doi: 10.1109/MCSE.2021.3097276.

T. Deakin, S. McIntosh-Smith, S. J. Pennycook and J. Sewall, "Analyzing Reduction Abstraction Capabilities," 2021 International Workshop on Performance, Portability and Productivity in HPC (P3HPC), 2021, pp. 33-44, doi: 10.1109/P3HPC54578.2021.00007.

T. Deakin, S. McIntosh-Smith, A. Alpay and V. Heuveline, "Benchmarking and Extending SYCL Hierarchical Parallelism," 2021 IEEE/ACM International Workshop on Hierarchical Parallelism for Exascale Computing (HiPar), 2021, pp. 10-19, doi: 10.1109/HiPar54615.2021.00007.

T. Deakin, J. Price, M. Martineau, S. McIntosh-Smith, "Evaluating attainable memory bandwidth of parallel programming models via BabelStream," in International Journal of Computational Science and Engineering (IJCSE), Vol. 17, No. 3, 2018.

T. Deakin, A. Poenaru, T. Lin and S. McIntosh-Smith, "Tracking Performance Portability on the Yellow Brick Road to Exascale," 2020 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC), 2020, pp. 1-13, doi: 10.1109/P3HPC51967.2020.00006.

J. Sewall, S. J. Pennycook, D. Jacobsen, T. Deakin and a. S. McIntosh-Smith, "Interpreting and Visualizing Performance Portability Metrics," 2020 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC), 2020, pp. 14-24, doi: 10.1109/P3HPC51967.2020.00007.

Wei-Chen Lin, Tom Deakin, and Simon McIntosh-Smith. 2021. On measuring the maturity of SYCL implementations by tracking historical performance improvements. In International Workshop on OpenCL (IWOCCL'21). Association for Computing Machinery, New York, NY, USA, Article 8, 1–13. DOI:<https://doi.org/10.1145/3456669.3456701>

Tom Deakin and Simon McIntosh-Smith. 2020. Evaluating the performance of HPC-style SYCL applications. In Proceedings of the International Workshop on OpenCL (IWOCCL'20). Association for Computing Machinery, New York, NY, USA, Article 12, 1–11. DOI:<https://doi.org/10.1145/3388333.3388643>

T. Deakin et al., "Performance Portability across Diverse Computer Architectures," 2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC), 2019, pp. 1-13, doi: 10.1109/P3HPC49587.2019.00006.

<https://uob-hpc.github.io/>