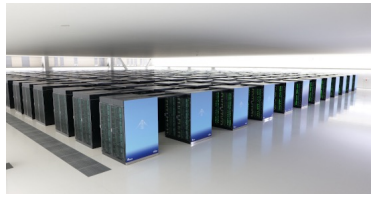
How to Measure, Visualise and Interpret Performance Portability

Dr Tom Deakin

Senior Research Associate
University of Bristol
tom.deakin@bristol.ac.uk

Processor diversity at (pre-)Exascale

The Next Platform, Jan 13th 2020: "HPC in 2020: compute engine diversity gets real" https://www.nextplatform.com/2020/01/13/hpc-in-2020-compute-engine-diversity-gets-real/

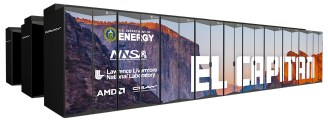


At RIKEN: Fujitsu A64fx CPUs.

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).

Processor diversity in the Cloud

- AWS: Amazon Graviton 2 CPUs, Intel Xeon, AMD EPYC.
- Oracle Cloud: Ampere Altra CPUs, Intel Xeon, AMD EPYC.
- Plus GPU offerings.

https://www.arm.com/solutions/infrastructure/cloud-computing

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.

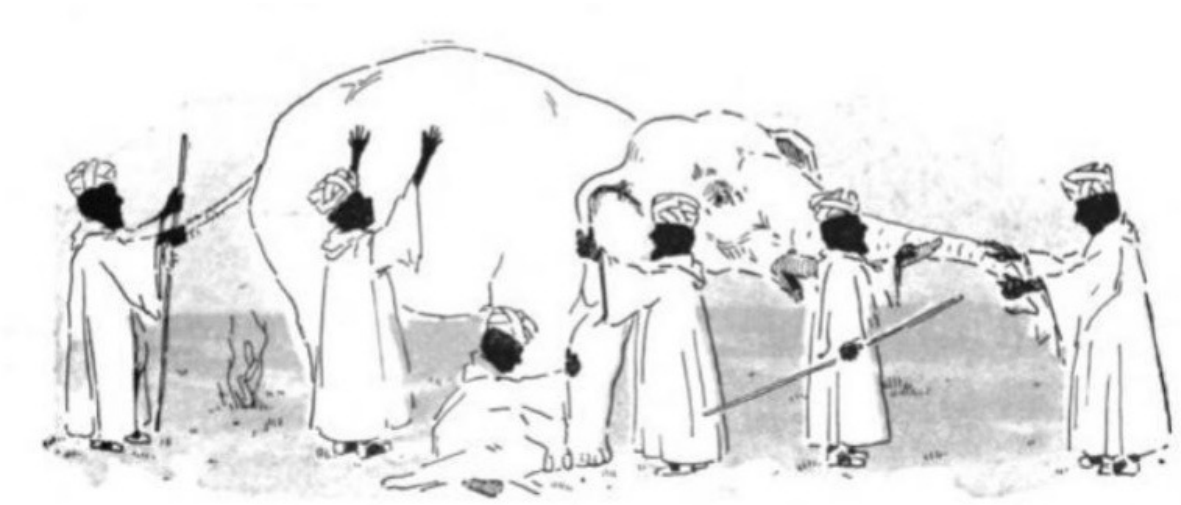


Image from Wikipedia, public domain

Enabling performance portability

Open standard parallel programming models

Open-source programming abstractions

Your favourite DSL and its compiler

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.:
  - Triad: a[i] = b[i] + scalar * c[i]

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

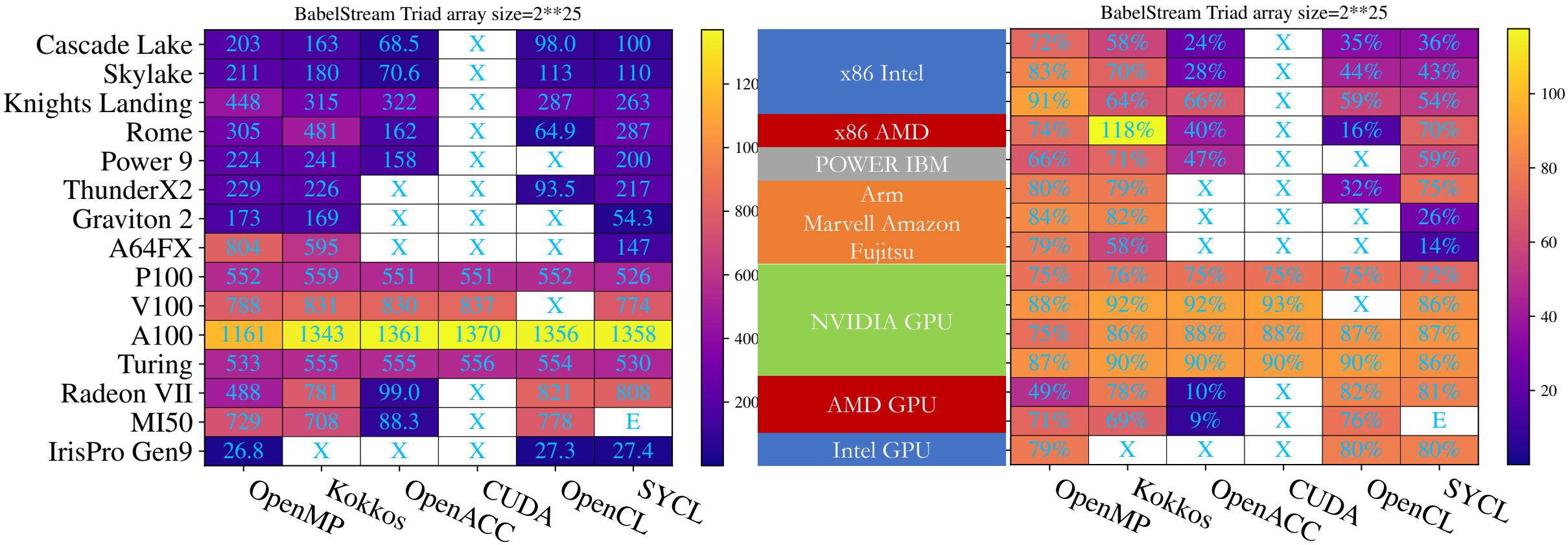
Measuring efficiency

- 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”.

https://doi.org/10.1109/P3HPC51967.2020.00006

BabelStream heatmaps

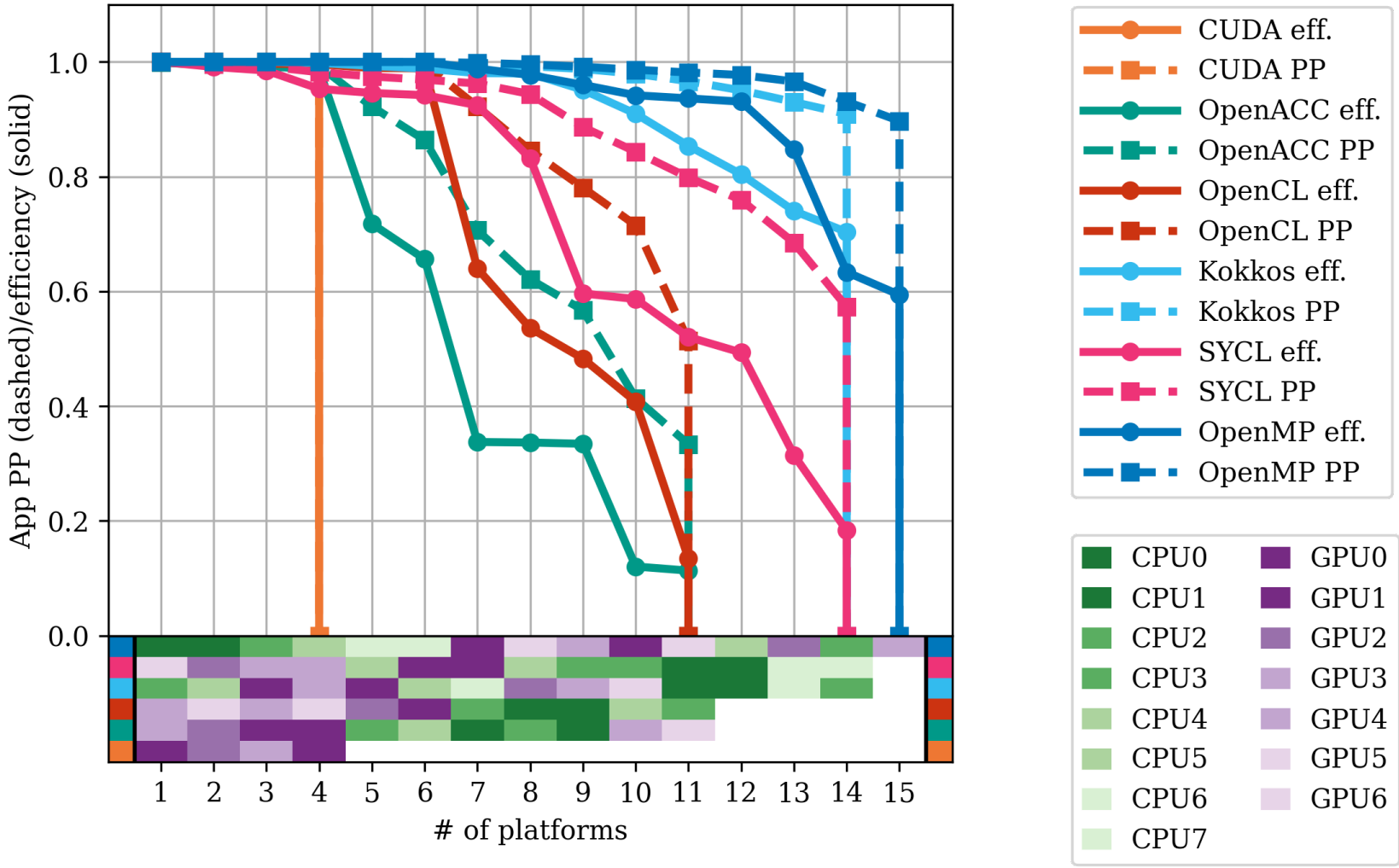
Peak performance



Architectural efficiency

Data from https://doi.org/10.1109/P3HPC51967.2020.00006

BabelStream Cascade plot



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

File Edit View Run Kernel Tabs Settings Help

Efficiency Cascad

Code

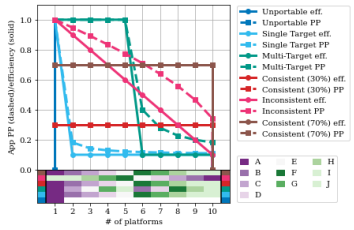
Name | Last Modified

app_platfor... a month ago
Adaptive Ke... 6 months ago
babelstrea... 15 days ago
Binned cha... 6 months ago
Box plot gr... 6 months ago
Clustered b... 6 months ago
Efficiency ... a minute ago
Efficiency ... a month ago
synthetic_c... 15 days ago

```
import pp_vis
```

[5]:
```
# Colours from https://personal.sron.nl/~pault/
app_colors = {
    "Unportable": "#0077BB",
    "Single Target": "#33BBEE",
    "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",
    "#9970AB",
    "#C2A5CF",
    "#E7D4E8",
    "#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")
```

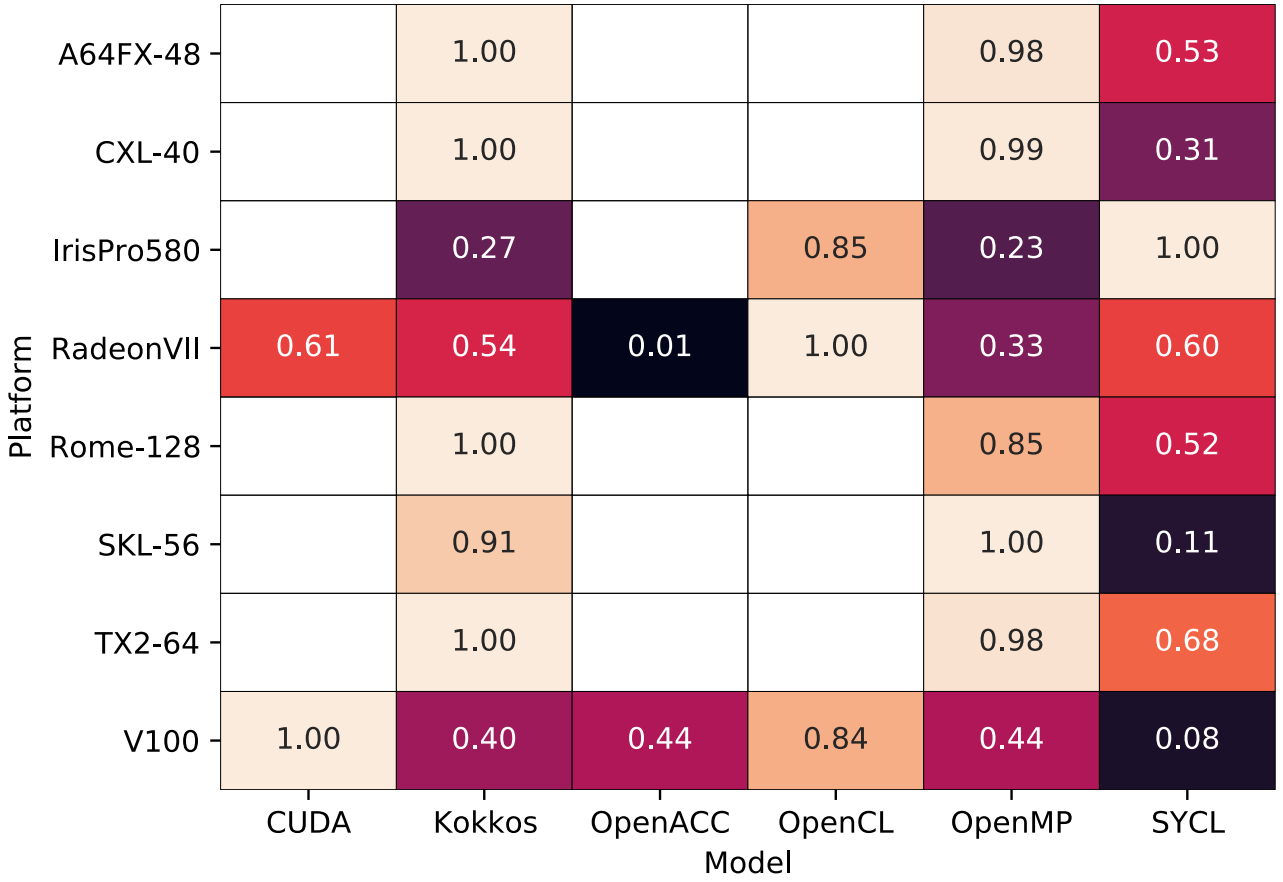


0 1 Python 3 | Idle Saving completed Mode: Command Ln 1, Col 1 Efficiency Cascades BabelStream 2020.ipynb

Figures from: Poenaru, Lin and McIntosh-Smith.
https://doi.org/10.1007/978-3-030-78713-4_18

Compute bound: miniBUDE

>56% of peak compute performance on CXL and Rome



https://doi.org/10.1109/P3HPC.2018.00006

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

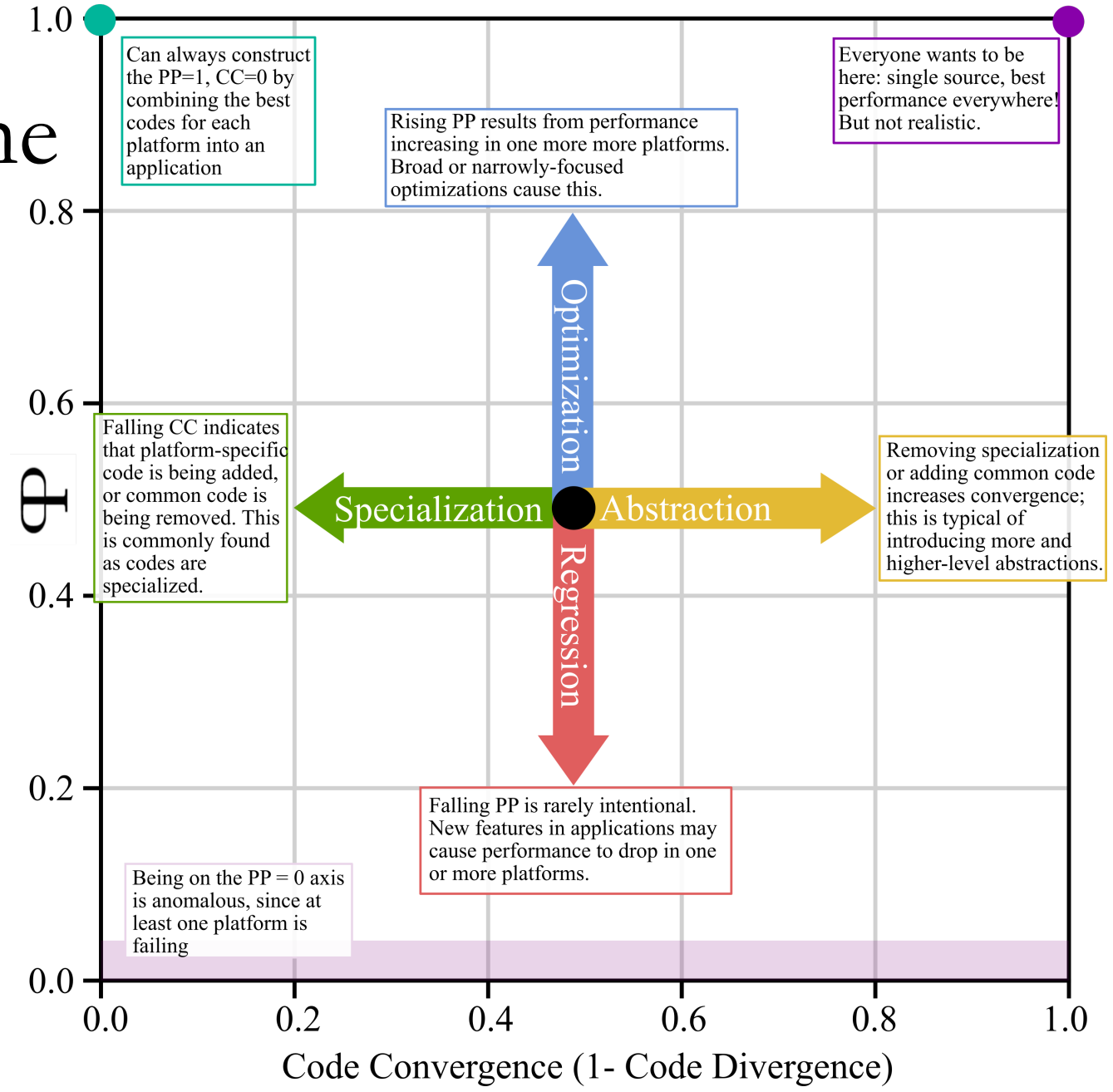


Figure by J. Sewall and J. Pennycook from upcoming article from Sewall, Pennycook, Jacobsen, Deakin and McIntosh-Smith

Top Tips for Performance Portability

- Measure and track performance and portability.
- Use Open Standards.
- Expose maximum concurrency/parallelism at all levels.
- Optimise for the diverse landscape - avoid over-optimising for one platform.
- Build in paths for specialisation and auto-tuning.

http://uob-hpc.github.io/publications/