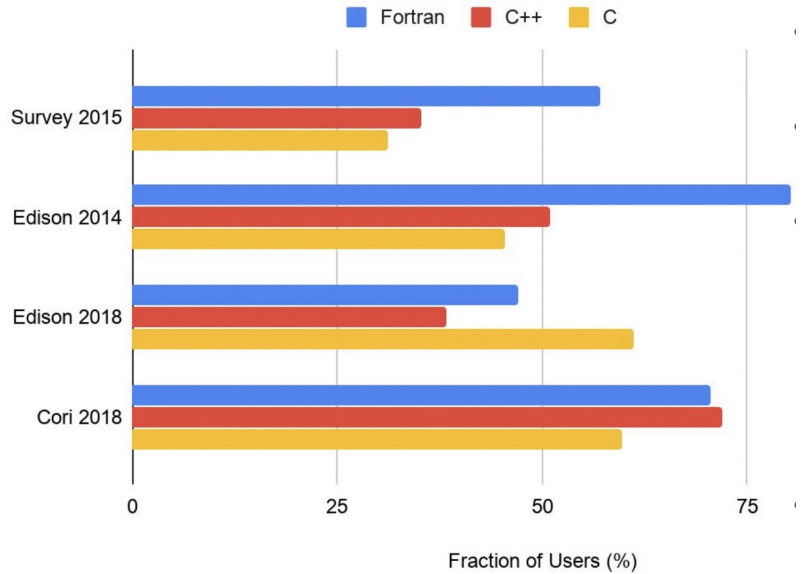
Benchmarking Fortran DO CONCURRENT on CPUs and GPUs Using BabelStream

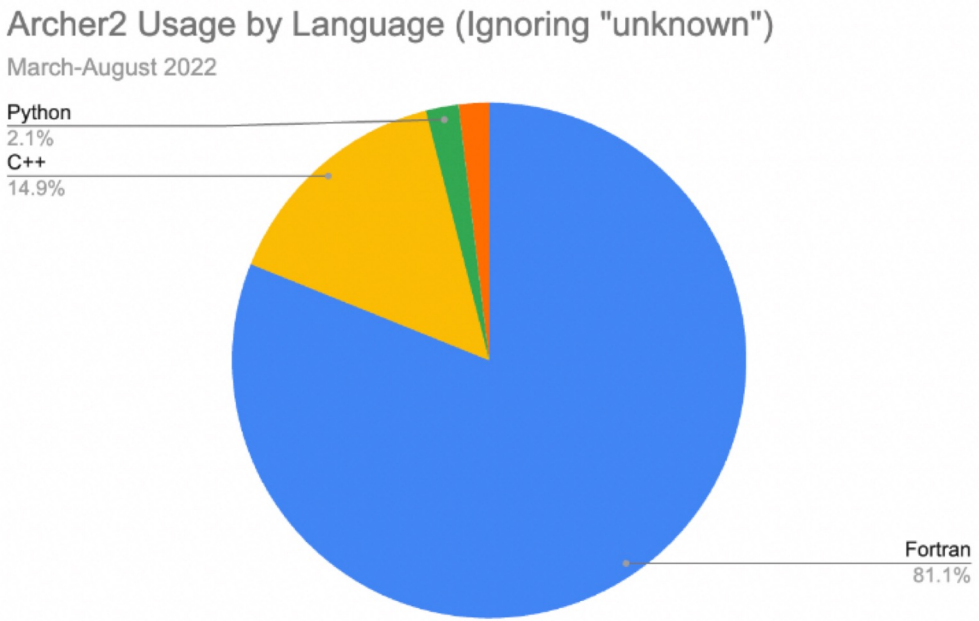
Jeff Hammond (NVIDIA, Finland)
Tom Deakin (Bristol, UK)
James (Jim) Cownie (Bristol, UK)
Simon McIntosh-Smith (Bristol, UK)

Fortran remains ubiquitous in HPC

NERSC



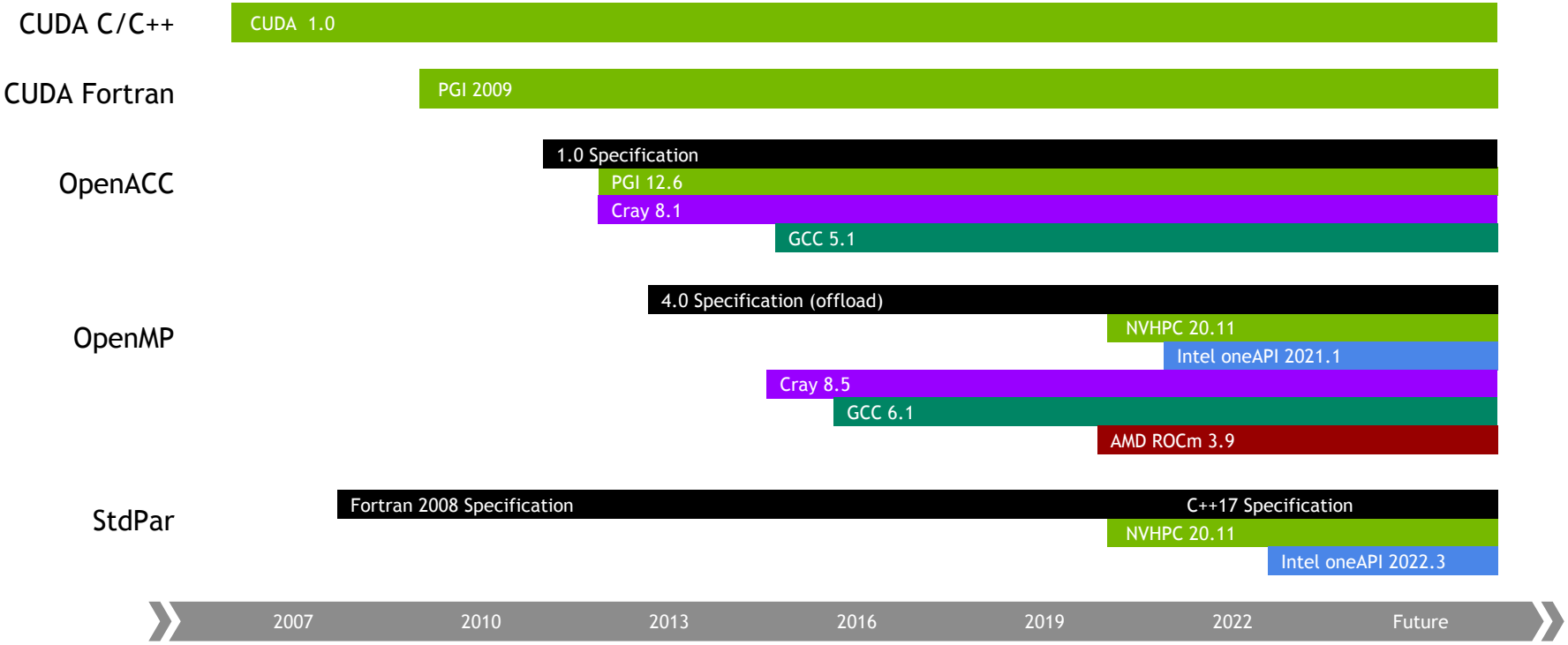
Totals exceed 100% because some users rely on multiple languages.



https://portal.nersc.gov/project/m888/nersc10/workload/N10_Workload_Analysis.latest.pdf

Programming Models for GPUs

A brief, possibly incomplete, history



“When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science, whatever the matter may be.”

Lord Kelvin, *Popular Lectures and Addresses* vol. 1 (1889)
‘Electrical Units of Measurement’, delivered 3 May 1883

Objectives

Create an idiomatic, complete and objective benchmark for Fortran programming models, based on BabelStream:

1. Use modern Fortran programming style.
2. Reproduce BabelStream C++, both in performance and functionality.
3. Support **all** Fortran parallel programming models for CPUs and GPUs.
4. Support as many platforms (compilers + hardware) as possible.

Then measure everything…

Evaluating attainable memory bandwidth of parallel programming models via BabelStream.
Deakin T, Price J, Martineau M, McIntosh-Smith S. https://dl.acm.org/doi/10.5555/3292750.329275

Summary of Experiments - Hardware

System Name	CPU	GPU
orin	Arm Cortex-A78AE (12x)	not included
nuclear	Intel Core i7-1165G7 (Tiger Lake)	Iris Xe Graphics (device=0x9a49), GeForce RTX 2060 (Turing, cc75)
perlmutter	AMD 7713 (Milan)	NVIDIA A100-40G
gorby	AMD 7742 (Rome)	NVIDIA A100-80G
brewster	Ampere Altra Q80-30	NVIDIA A100-40G
c6g16xlarge	AWS Graviton 2	-
c7g16xlarge	AWS Graviton 3	-
mi100	AMD 7502 (Rome)	AMD MI100
ice4	Intel Xeon 6338 (Ice Lake)	-
a64fx	Fujitsu A64fx	-

Summary of Experiments - Software

Programming motifs

- Sequential loops (not shown)
- Array notation
- Do Concurrent
- OpenMP classic
- OpenMP workshare @ Array
- OpenMP taskloop
- OpenMP target prescriptive
- OpenMP target descriptive
- OpenACC loop
- OpenACC kernels @ Array

Compilers

- AMD ROCm 5.1.6 (GPU)
- ARM 22.0.2
- Cray 10.0.3 (A64fx) or 14.0.1 (GPU)
- Fujitsu 4.3.1
- GCC 11 or 12
- Intel 2021.6 (LLVM-based on GPU)
- NVHPC 22.7

Implementation Design

1. Every implementation is contained in a module that implements 6 management procedures (e.g. allocation) and the 6 BS kernels.
2. The driver is implementation agnostic except for preprocessor selection of the different implementation modules.
3. The input parser and output match C++ essentially exactly.
4. The build system uses GNU Make, which works perfectly.

Workarounds for compiler bugs is incomplete.

We support both Fortran and OpenMP timers, because the former isn't always accurate.

Everything can be seen here:
https://github.com/UoB-HPC/BabelStream/pull/135

Fortran versus C++, Part 1

NVIDIA compilers show minimal differences between languages with OpenMP on an Ampere Altra Q80 CPU and CUDA on an A100-80G GPU, except when the CUDA C++ Dot product uses a suboptimal (processor-dependent) parameter.

	OpenMP			CUDA		
Function	C++	Fortran	Δ	C++	Fortran	Δ
Copy	156642	157308	+0.42%	1748027	1747360	-0.04%
Mul	159294	157741	-0.98%	1745605	1745195	-0.02%
Add	165328	164461	-0.52%	1772533	1791087	+1.05%
Triad	165713	164382	-0.80%	1773863	1793031	+1.08%
Dot[1]	188901	188504	-0.21%	1555400	1764934	+13.47%
Dot[2]	-	-	-	1743625	1764934	+1.22%

$$\Delta = B_{Fortran} - B_{C++}/B_{C++}$$

1. DOT_NUM_BLOCKS=256, which is the default.
2. DOT_NUM_BLOCKS=1024.

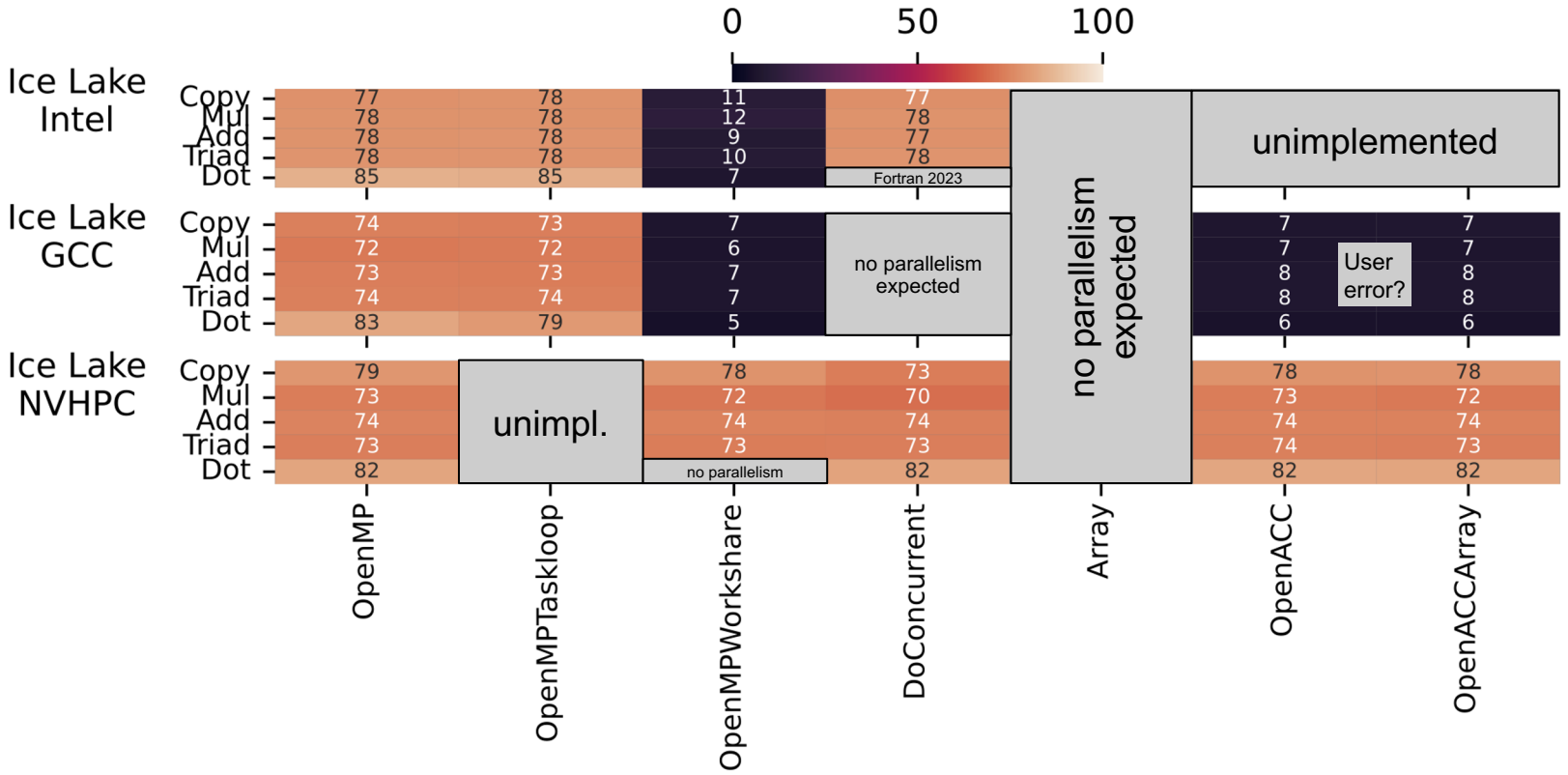
Fortran versus C++, Part 2

OpenMP target is the only model available for all GPUs. Unfortunately, the performance difference between languages is larger with some compilers.

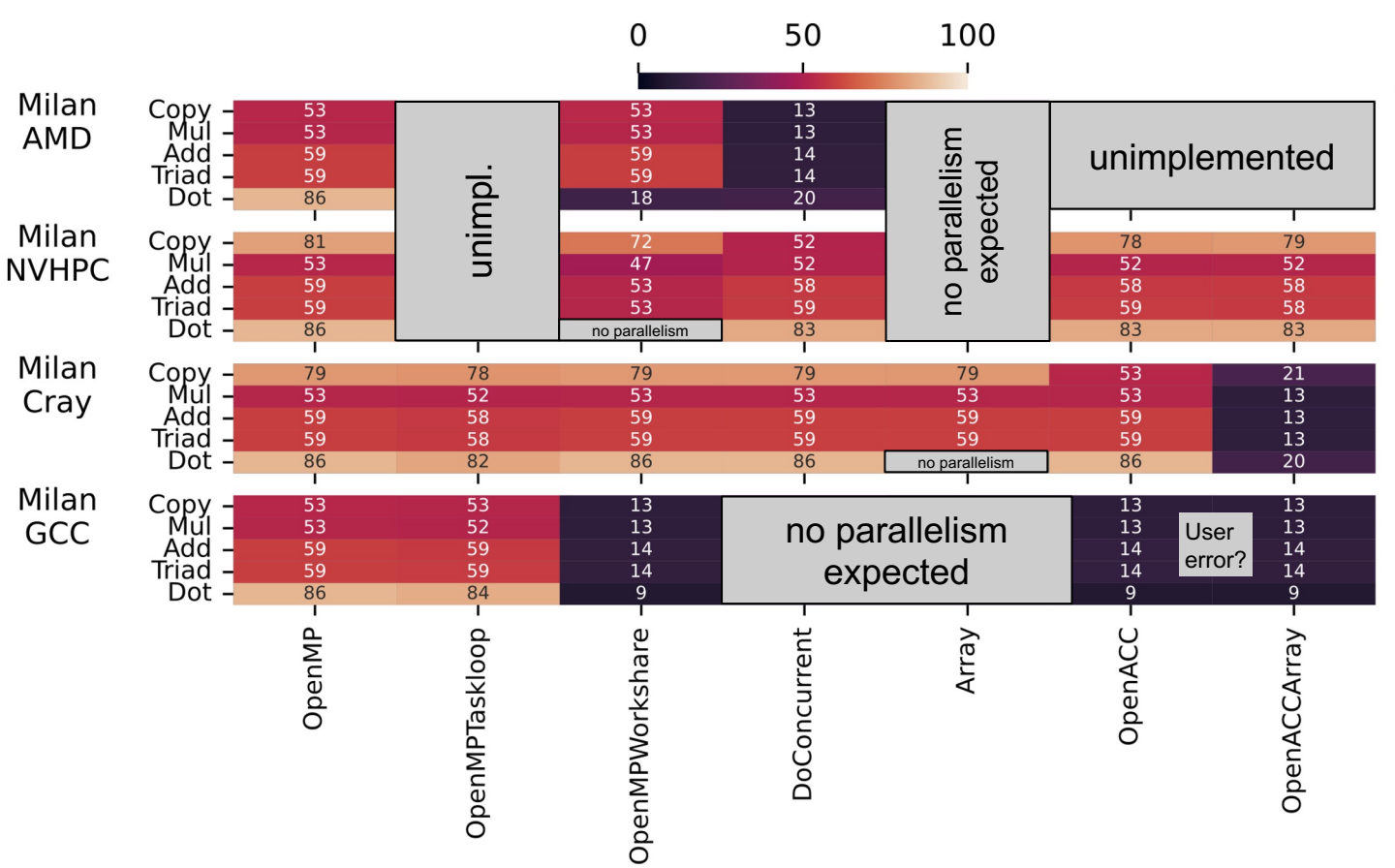
The Dot product kernel is particularly affected, sometimes profoundly, but these are compiler issues, not flaws in the languages themselves.

Function	ROCM/MI100	NVHPC/A100	Intel/Xe iGPU	Cray/A100
Copy	-15.98%	-1.77%	-4.56%	-3.12%
Mul	-15.62%	-1.62%	-1.05%	-3.71%
Add	-15.84%	-1.01%	-0.20%	-0.11%
Triad	-15.98%	-1.07%	-0.43%	-0.71%
Dot	165.82%	-6.43%	-10.9%	-0.97%

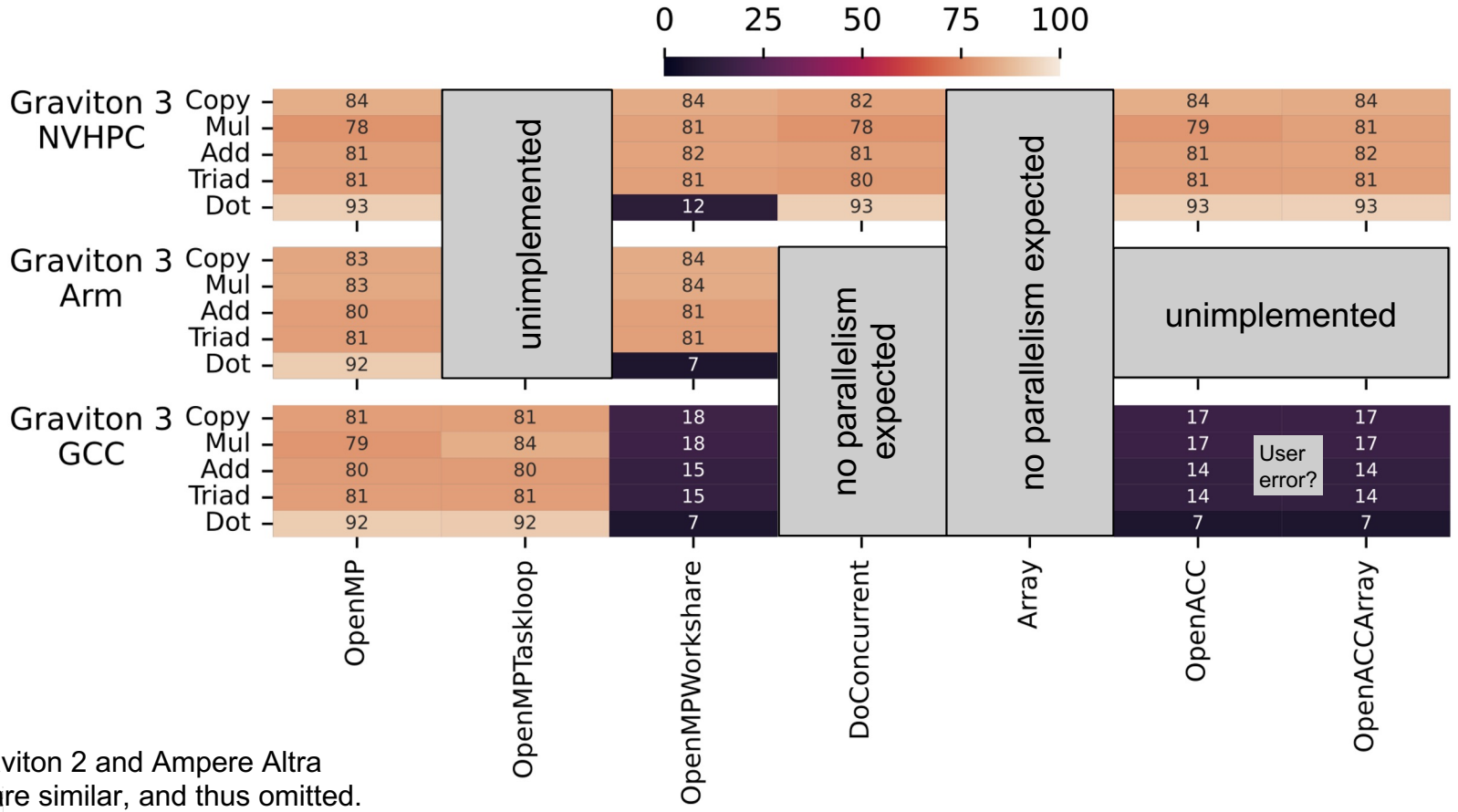
Intel Ice Lake Xeon



AMD Milan

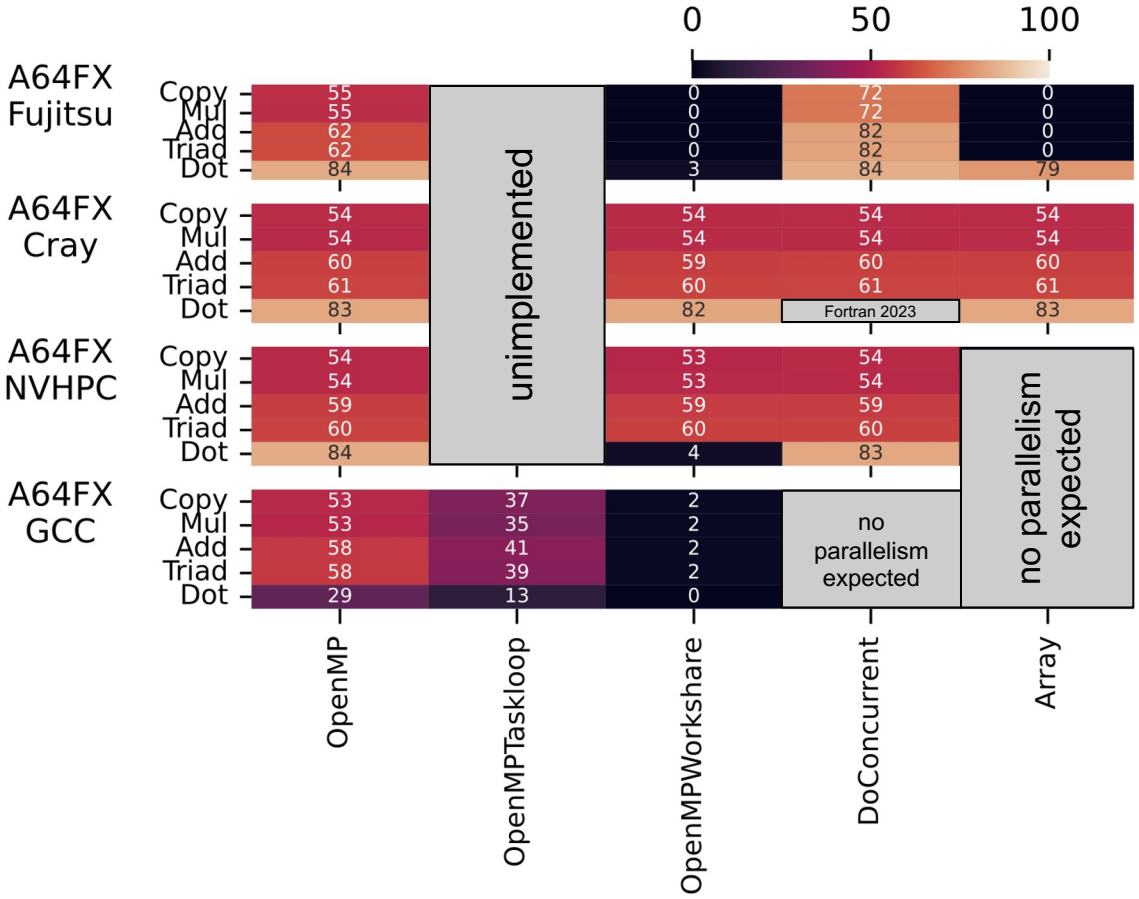

0
50
100
Milan AMD
Copy 53 53 53 13
Mul 53 53 53 13
Add 59 59 59 14
Triad 59 59 59 14
Dot 86 18 20
unimpl.
no parallelism expected
unimplemented
Milan NVHPC
Copy 81 72 52 78 79
Mul 53 47 52 52 52
Add 59 53 58 58 58
Triad 59 53 59 59 58
Dot 86 no parallelism 83 83 83
Milan Cray
Copy 79 78 79 79 79 53 21
Mul 53 52 53 53 53 53 13
Add 59 58 59 59 59 59 13
Triad 59 58 59 59 59 59 13
Dot 86 82 86 86 no parallelism 86 20
Milan GCC
Copy 53 53 13 13 13
Mul 53 52 13 13 13
Add 59 59 14 14 14
Triad 59 59 14 14 14
Dot 86 84 9 9 9
no parallelism expected
User error?
OpenMP
OpenMPTaskloop
OpenMPWorkshare
DoConcurrent
Array
OpenACC
OpenACCArray

AWS Graviton 3

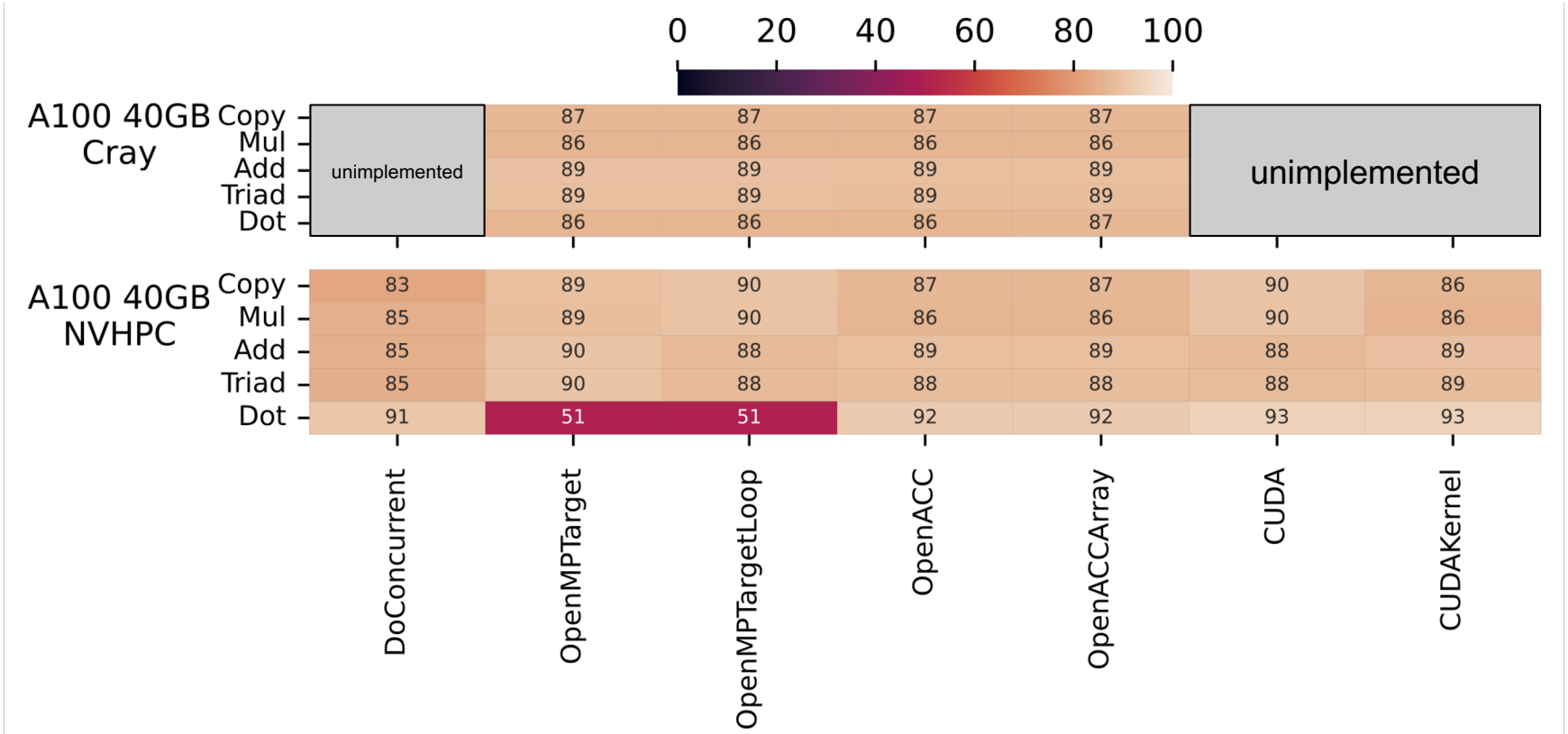


The Graviton 2 and Ampere Altra results are similar, and thus omitted.

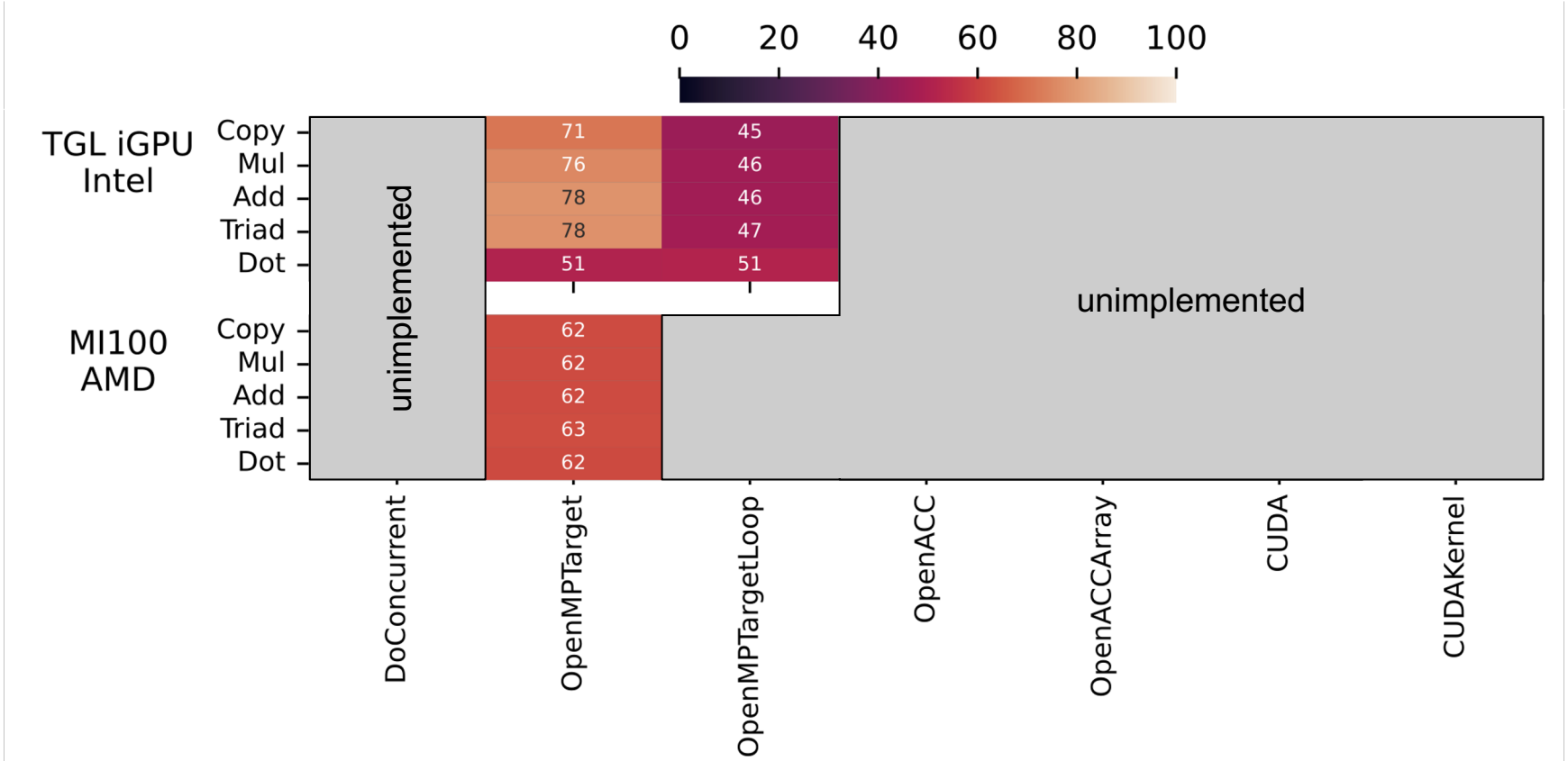
Fujitsu A64fx



NVIDIA A100


0
20
40
60
80
100
A100 40GB Cray
Copy
Mul
Add
Triad
Dot
unimplemented
87
86
89
89
86
87
86
89
89
86
87
86
89
89
86
87
unimplemented
A100 40GB NVHPC
Copy
Mul
Add
Triad
Dot
83
85
85
85
91
89
89
90
90
51
90
90
88
88
51
87
86
89
88
92
87
86
89
88
92
90
90
88
88
93
86
86
89
89
93
DoConcurrent
OpenMPTarget
OpenMPTargetLoop
OpenACC
OpenACCArray
CUDA
CUDAKernel

Other GPUs


0
20
40
60
80
100
TGL iGPU
Intel
Copy
Mul
Add
Triad
Dot
71
76
78
78
51
45
46
46
47
51
MI100
AMD
Copy
Mul
Add
Triad
Dot
62
62
62
63
62
unimplemented
unimplemented
DoConcurrent
OpenMPTarget
OpenMPTargetLoop
OpenACC
OpenACCArray
CUDA
CUDAKernel

Summary of Results - Good News

1. There are no less than 7* different Fortran compilers for HPC users that together support a wide range of parallel programming models for CPUs and GPUs.
2. OpenMP 4 with prescriptive loop parallelism works well essentially everywhere.
3. Most compilers show negligible performance differences between C++ and Fortran, despite the massive difference in compiler investment.
4. Do Concurrent is implemented effectively for CPUs in Cray, Fujitsu, Intel and NVHPC compilers, and for GPUs with NVHPC and Intel** compilers.

* 5 if one considers the shared ancestry (PGI) of NVHPC, ARM and AMD.

** Intel support for Do Concurrent on GPUs was released too recently to evaluate here.

Summary of Results - Bad News

1. OpenMP workshare is poorly implemented in many compilers, which discourages adoption of more expressive Fortran constructs.
2. OpenMP taskloop and target loop implementations require more investment from compilers to be useful.
3. The HPC community continues to get what it pays for from the GCC Fortran compiler.
4. Compiler support for AMD and Intel GPUs is immature.
   The Cray compiler is going to be essential for AMD GPUs.
5. Do Concurrent reductions are a Fortran 2023 (draft) feature only implemented by the NVHPC compiler. We expect Cray and Intel to support it very soon, once the standard is published.

What Now?

- **BabelStream Fortran** should be a useful tool for HPC users to evaluate the quality of their Fortran compilers. It might even be a good procurement benchmark for HPC centers.
- Our experiments involved no processor or compiler specialization for performance. This is a good topic those with the relevant compiler and hardware expertise to investigate.
- We couldn't evaluate AMD MI200 GPUs for this paper, and both Intel and NVIDIA will have interesting high-bandwidth hardware in 2023.
- Find the code here: https://github.com/UoB-HPC/BabelStream/pull/135
- **Evaluating attainable memory bandwidth of parallel programming models via BabelStream**. Deakin T, Price J, Martineau M, McIntosh-Smith S. https://dl.acm.org/doi/10.5555/3292750.329275