UNIVERSITY OF ALBERTA

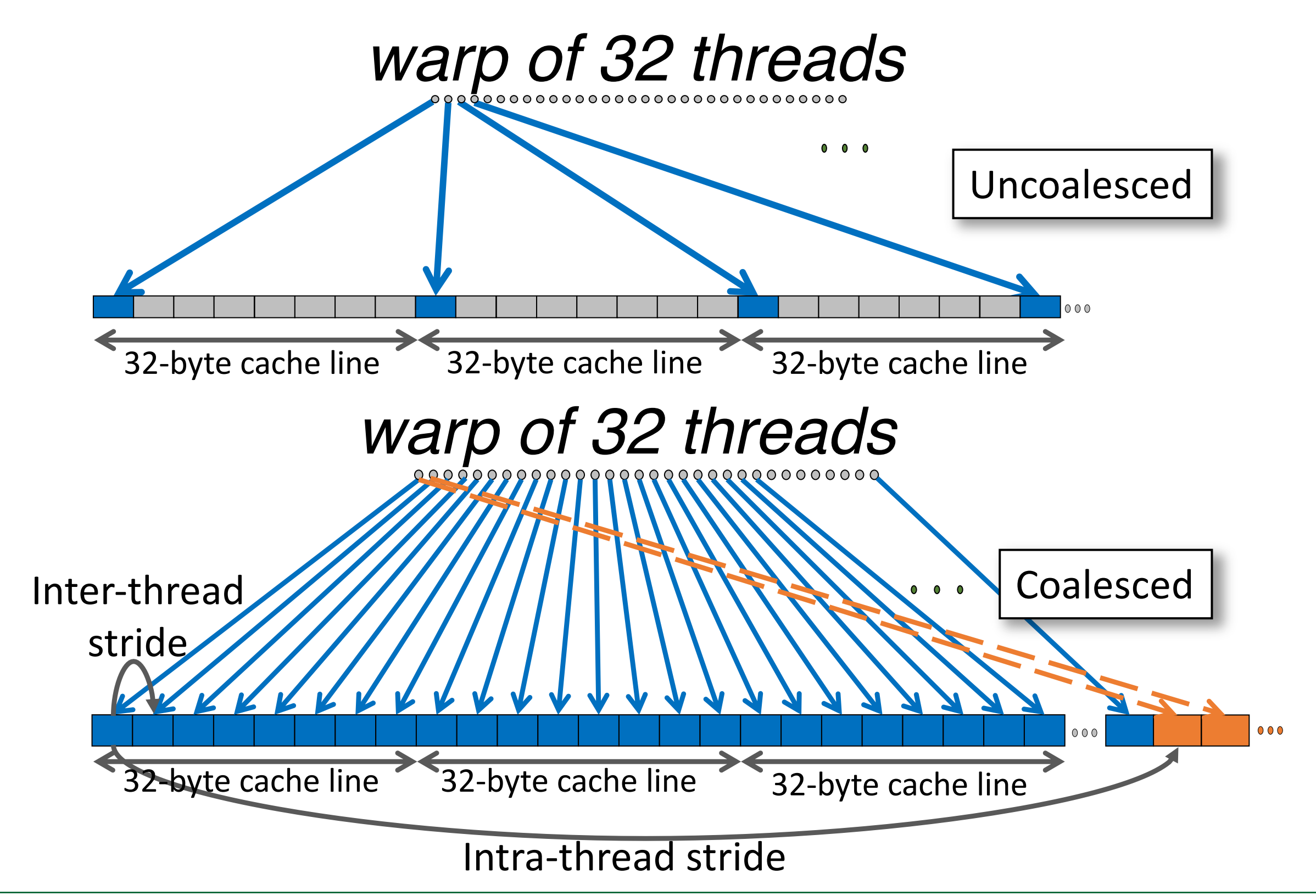
Memory-Access-Aware Loop Transformations of Accelerator-Bound OpenMP Loops

Artem Chikin[1], Taylor Lloyd[1], José Nelson Amaral[1], Ettore Tiotto[2]

[1]University of Alberta [2]IBM Canada

Portable Performance Demands Stronger Program Analyses

OpenMP supports architecture-agnostic accelerator programming. Computation on CPUs and GPUs has opposing performance demands to memory access patterns. CPUs must avoid false-sharing and achieve per-thread spatial locality. GPUs demand memory coalescing:



Iteration-Point Algebraic Difference (IPAD): A sophisticated static analysis framework that computes the inter-iteration memory access stride by calculating differences of addressing expressions' symbolic values that capture both data and control flow:

```
#define TSIZE 64
for(i = 1; i < N; ++i) {
  int idx = 0;
  if (I < (TSIZE / 2))
    idx = TSIZE + i;
  else
    idx = i;
  B[idx] = foo();
}
```

$$\mathrm{IPAD}(B[idx]) = ([i]<32)\times([\&B]+8\times(64+[i])) + ([i]>=32)\times([\&B]+8\times[i])$$

$$\begin{aligned}
&\mathrm{IPAD}_{t1}(B[idx]) - \mathrm{IPAD}_{t0}(B[idx]) \\
&= (1<32)\times([\&B]+8\times65) \\
&+ (1>=32)\times([\&B]+8\times1) \\
&- (0<32)\times([\&B]+8\times64) \\
&+ (0>=32)\times([\&B]+8\times0) \\
&= ([\&B]+520) - ([\&B]+512) \\
&= 520 - 512 \\
&= 8 \text{ (bytes)}
\end{aligned}$$

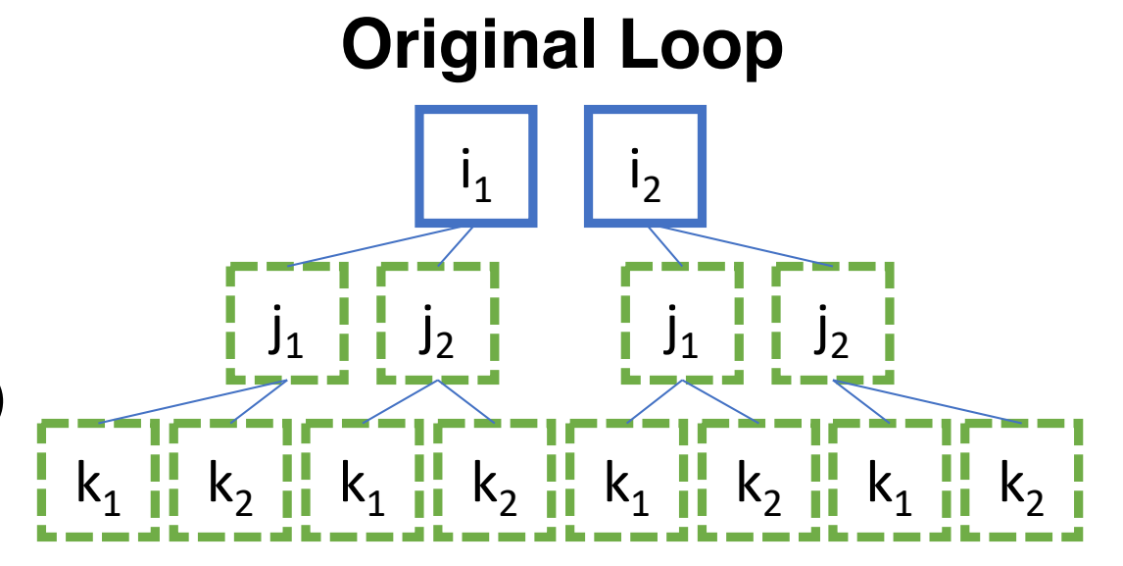
Parallelism Effects

Collapsing a loop nest with a parallel outer loop increases the overall number of iterations that can execute in parallel.

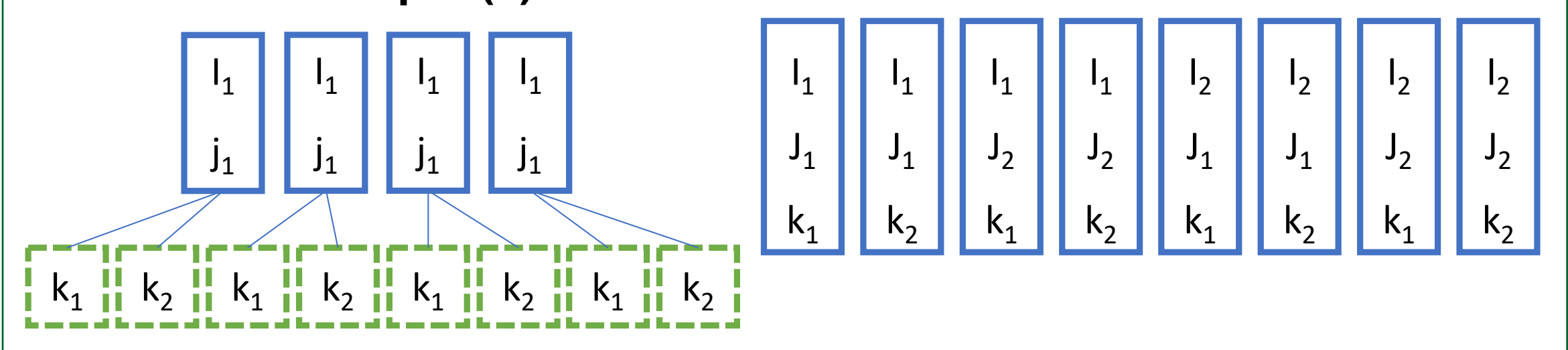
Original Loop

```
#pragma omp parallel for
for(i = 1; i < UI; ++i)
  for(j = 1; j < UJ; ++j)
    for(k = 1; k < UK; ++k)
      { … }
```

Original Loop



Collapse(2) **Collapse(3)**



Memory Access Effects

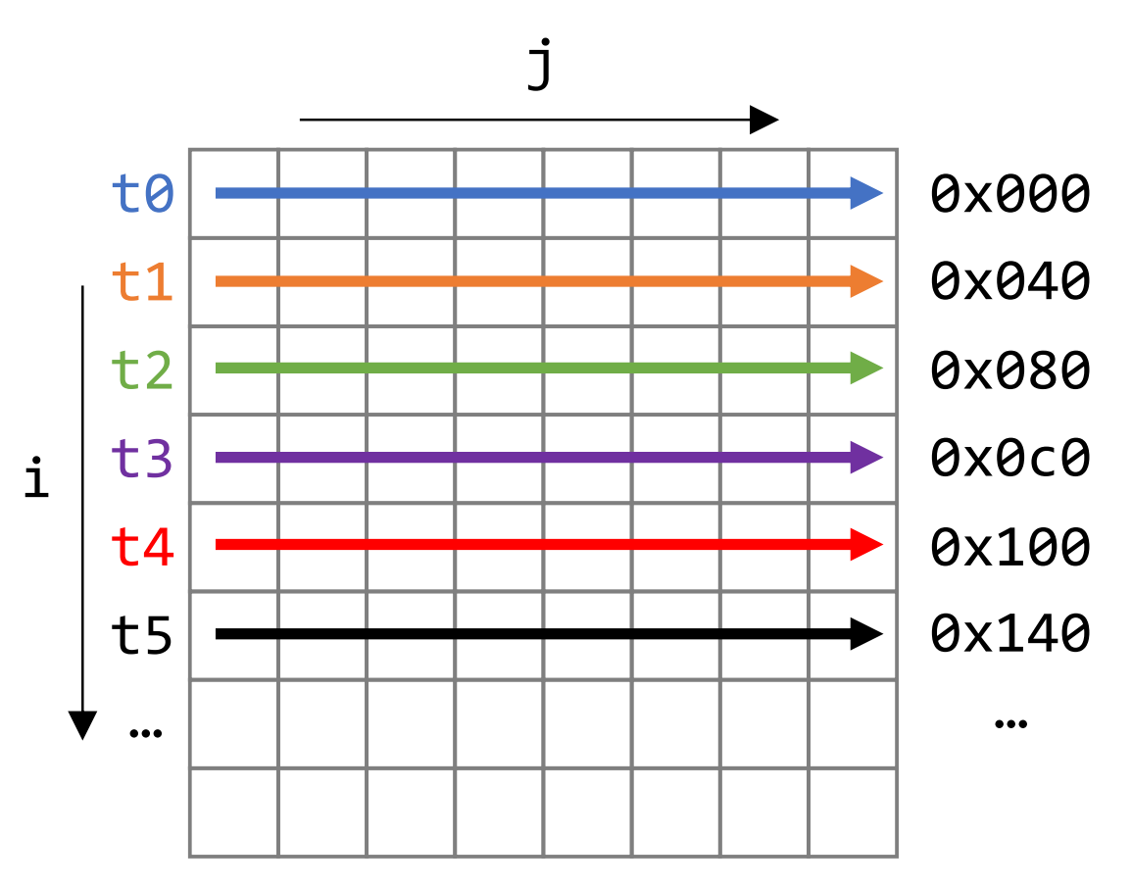
A collapsed nest's combined iteration space changes which threads execute which iteration points, affecting memory access patterns.

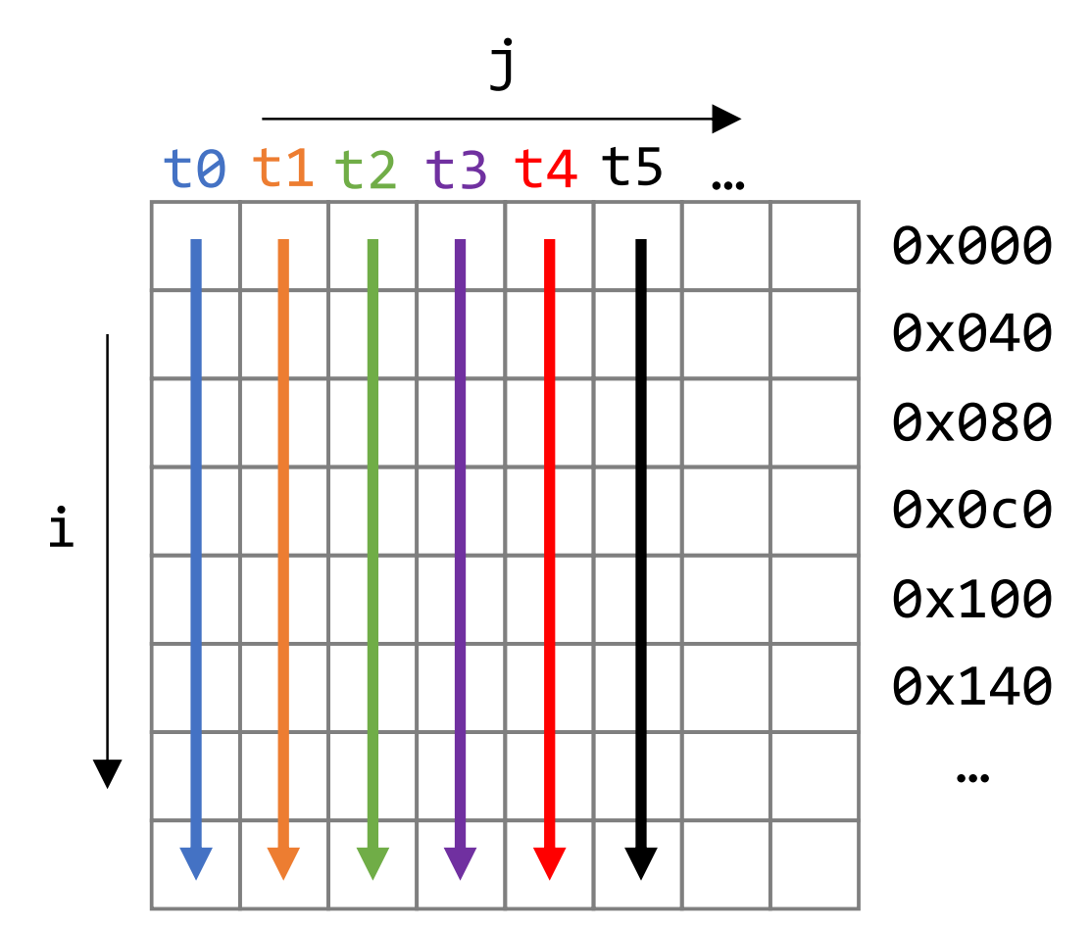
Original Loop

```
#pragma omp parallel for
for(i = 1; i < UI; ++i)
  for(j = 1; j < UJ; ++j)
    A[i * UI + j] = …
```

Collapse(2)

```
#pragma omp parallel for
for(c = 1; c < UI * UJ; ++c)
  i = c / UJ; j = c % UJ;
  A[i * UI + j] = …
```

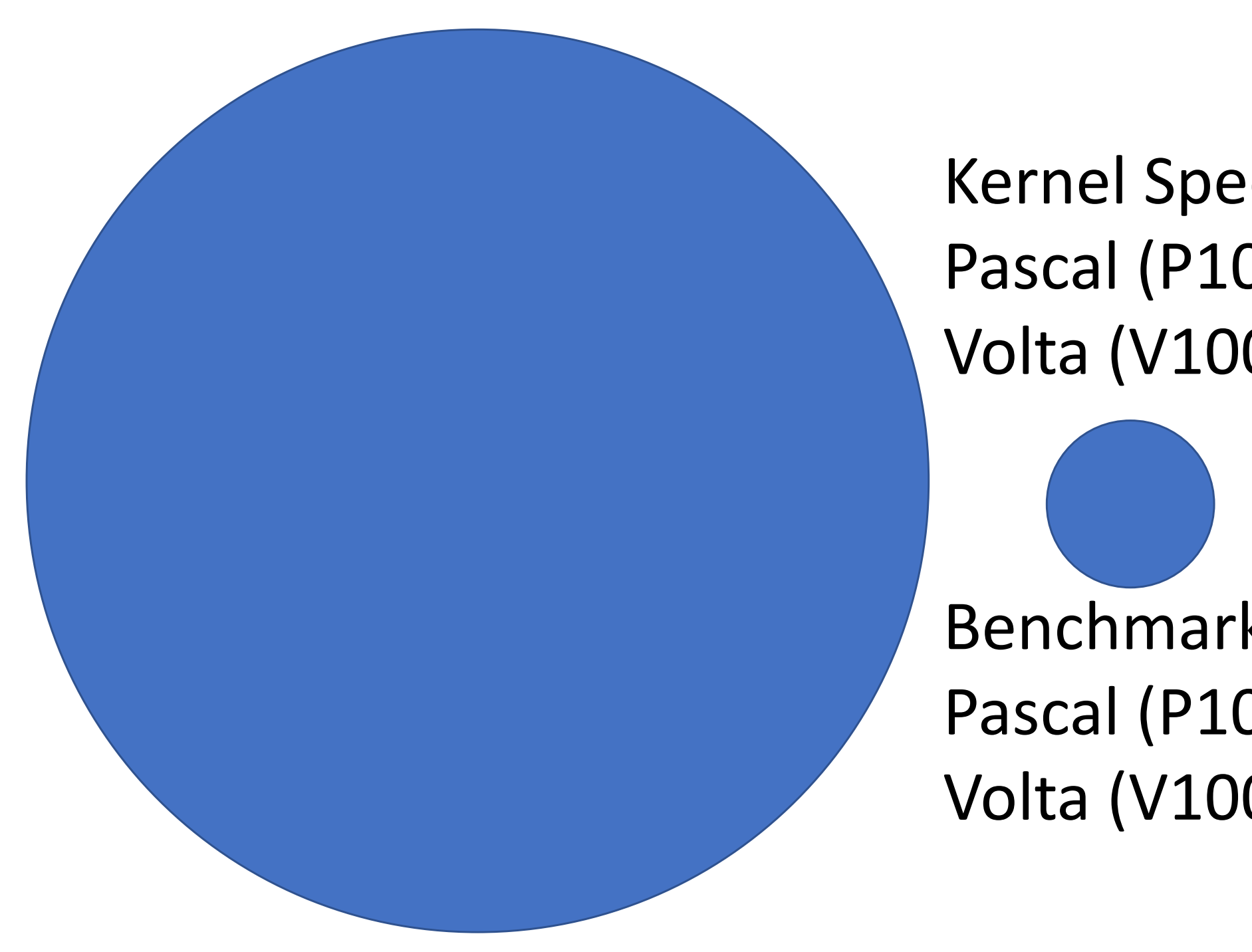




Performance

IPAD-enabled safety and profitability analyses power transformations that improve performance:

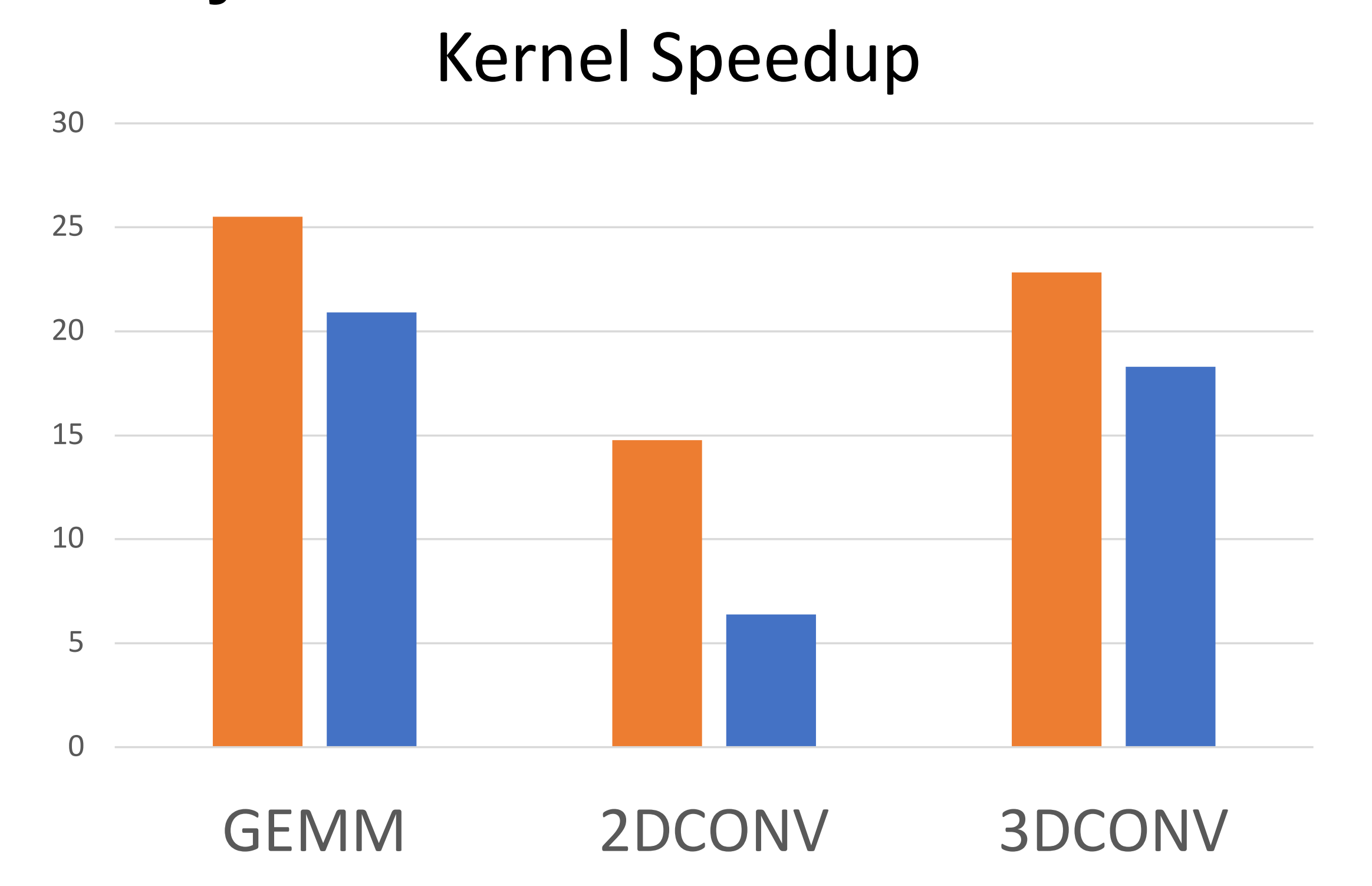
SPEC ACCEL: 557.pcsp

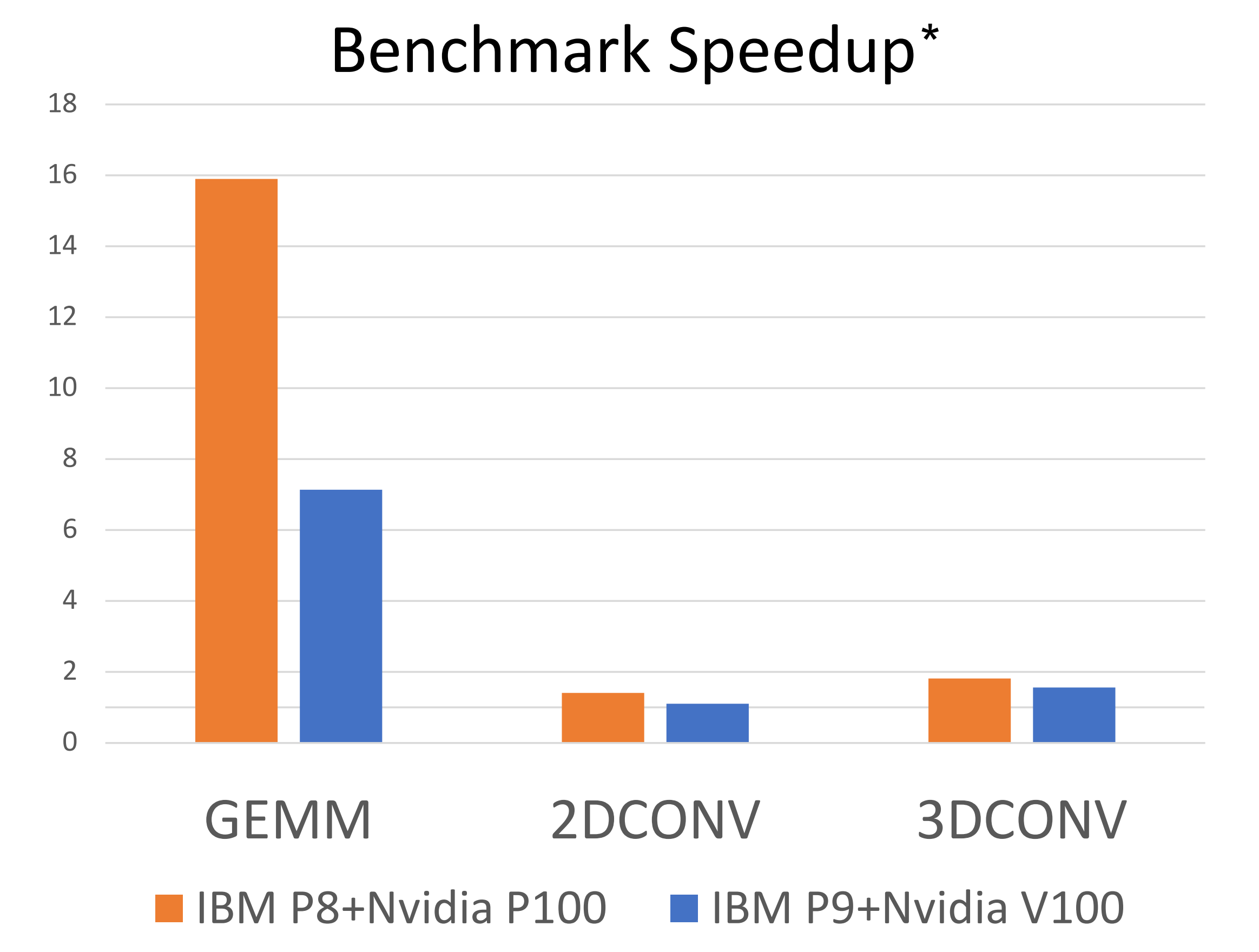


Kernel Speedup:
Pascal (P100): **29.4x**
Volta (V100): **16.4x**

Benchmark Speedup:
Pascal (P100): **1.53x**
Volta (V100): **1.26x**

Polybench

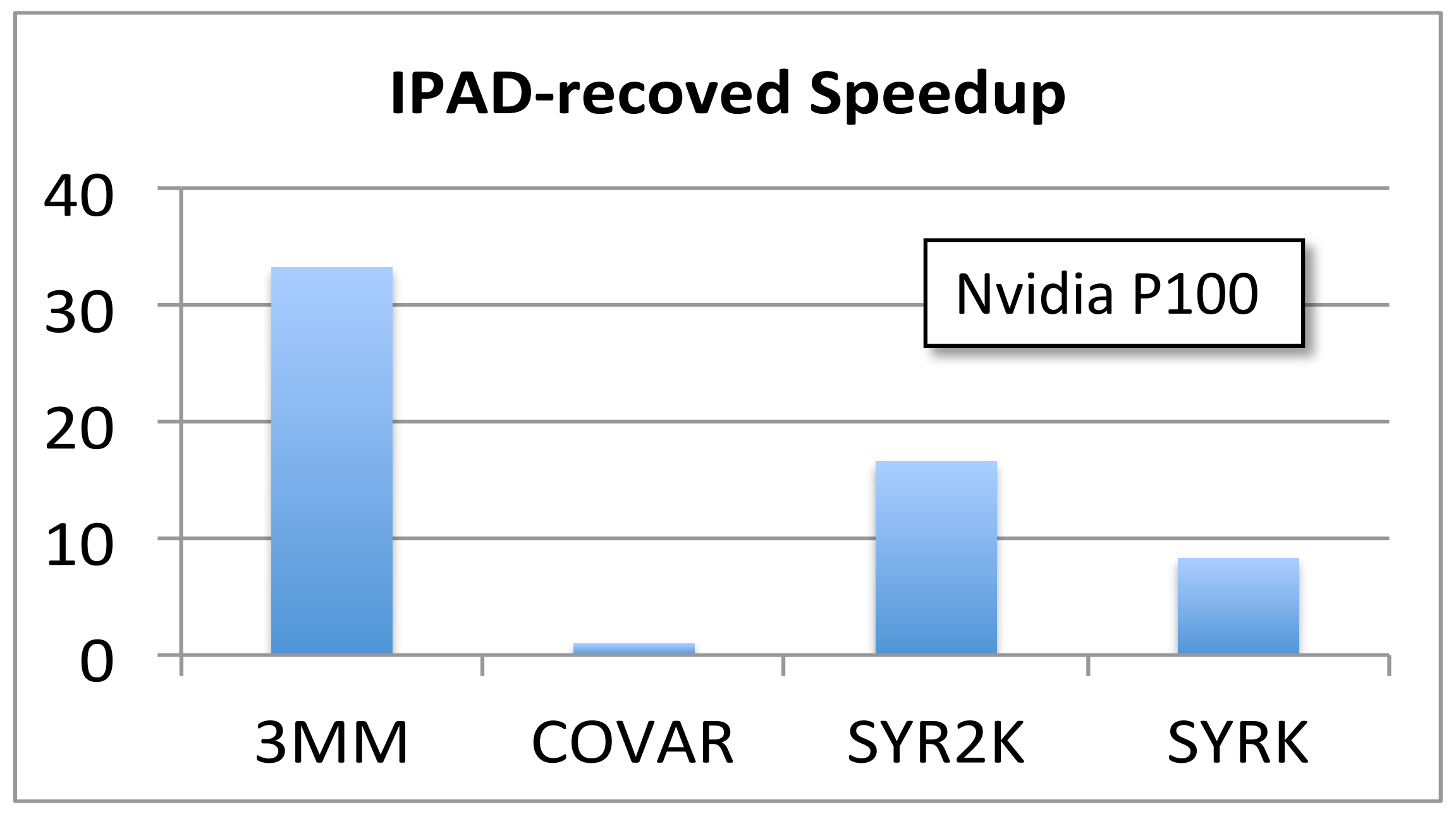




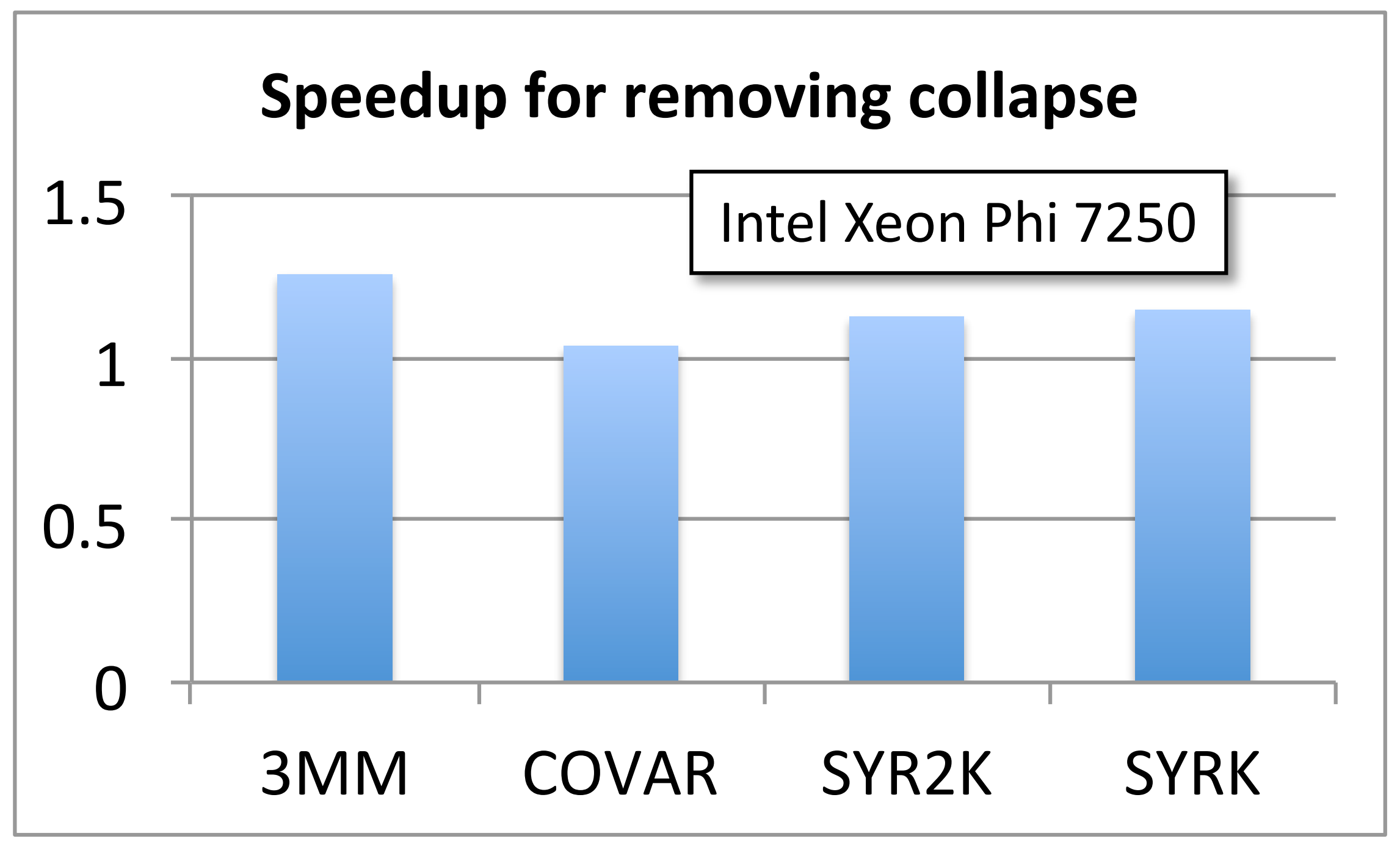
*High-throughput benchmark versions used

Portability

IPAD analysis framework can automatically discover optimization opportunities that would otherwise have to be specified by the programmer, like collapse(n).



Baseline: Existing collapse clauses had been removed from the benchmark source.
Comparison: IPAD discovered that they were needed and inserted them.



Baseline: Unaltered benchmark with collapse clauses.
Comparison: Portable version with removed collapse clauses.

Takeaway

Architecture-aware compilers must employ strong program analyses to generate higher-performing code without portability-reducing annotations.

CASTLE 2018

IBM Advanced Studies
Academic Applied Research Outcomes