ICS'25, Salt Lake City, UT, USA.

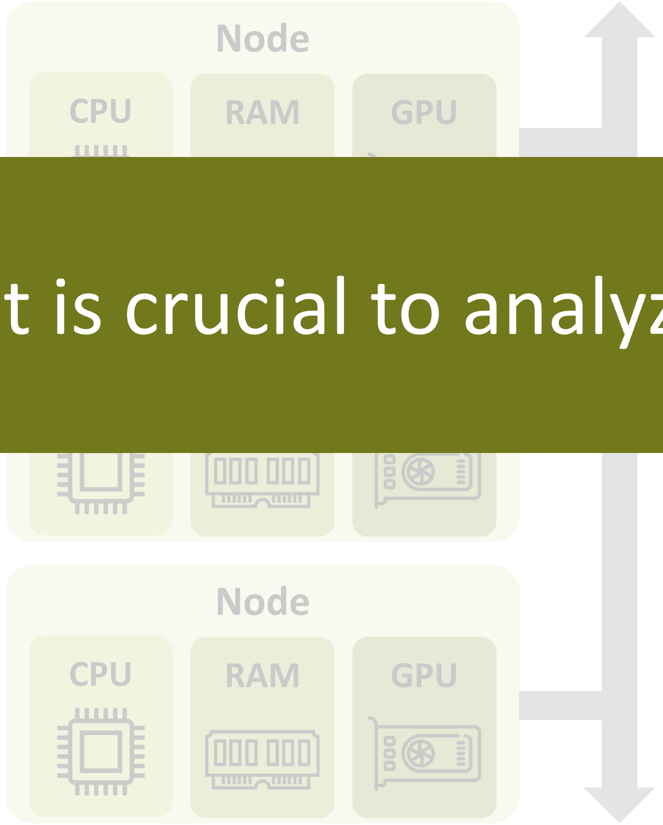
EDAN: Towards Understanding Memory Parallelism and Latency Sensitivity in HPC

Siyuan Shen[1], Mikhail Khalilov[1], Lukas Gianinazzi[1], Timo Schneider[1], Marcin Chrapek[1], Jai Dayal[2], Manisha Gajbe, Robert Wisniewski[3], Torsten Hoefler[1]

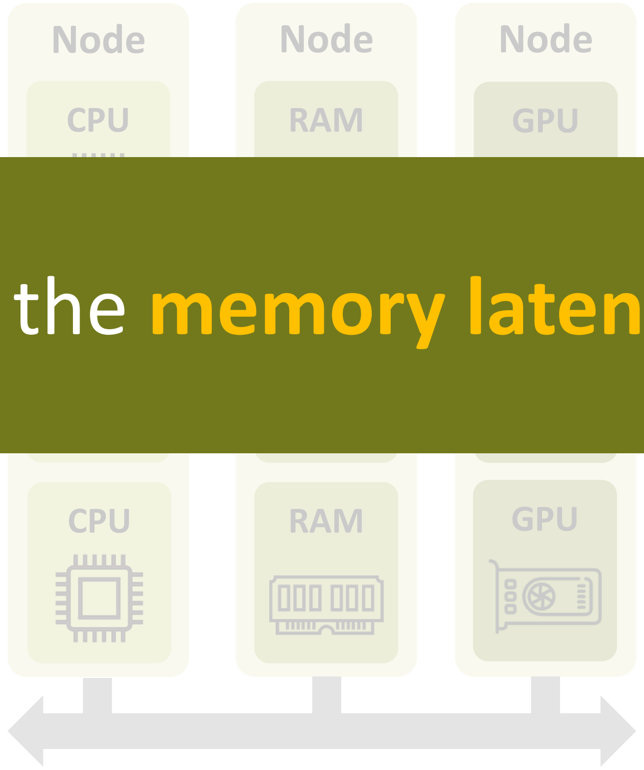
1. ETH Zurich, 2. Cerebras systems, 3. HPE

Motivation

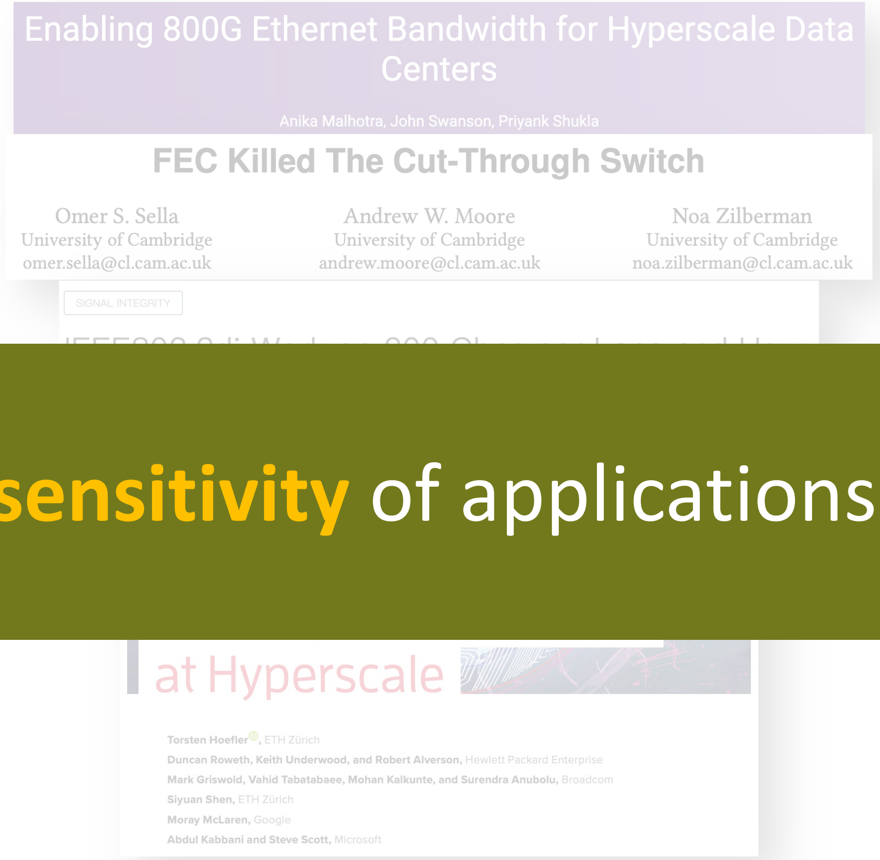
Standard Server
Aggregated resources using high-speed network between nodes



Resource Disaggregation
Disaggregated nodes with dedicated high-speed interconnect



Memory access relies on the **network fabric**


Enabling 800G Ethernet Bandwidth for Hyperscale Data Centers
Anika Malhotra, John Swanson, Priyank Shukla

FEC Killed The Cut-Through Switch
Omer S. Sella, University of Cambridge, omer.sella@cl.cam.ac.uk
Andrew W. Moore, University of Cambridge, andrew.moore@cl.cam.ac.uk
Noa Zilberman, University of Cambridge, noa.zilberman@cl.cam.ac.uk

SIGNAL INTEGRITY

at Hyperscale
Torsten Hoefler, ETH Zürich
Duncan Roweth, Keith Underwood, and Robert Alverson, Hewlett Packard Enterprise
Mark Griswold, Vahid Tabatabaee, Mohan Kalkunte, and Surendra Anubolu, Broadcom
Siyuan Shen, ETH Zürich
Moray McLaren, Google
Abdul Kabbani and Steve Scott, Microsoft

Higher network bandwidth leads to more complex **forward error correction (FEC)**, which will **increase latency**

It is crucial to analyze the memory latency sensitivity of applications

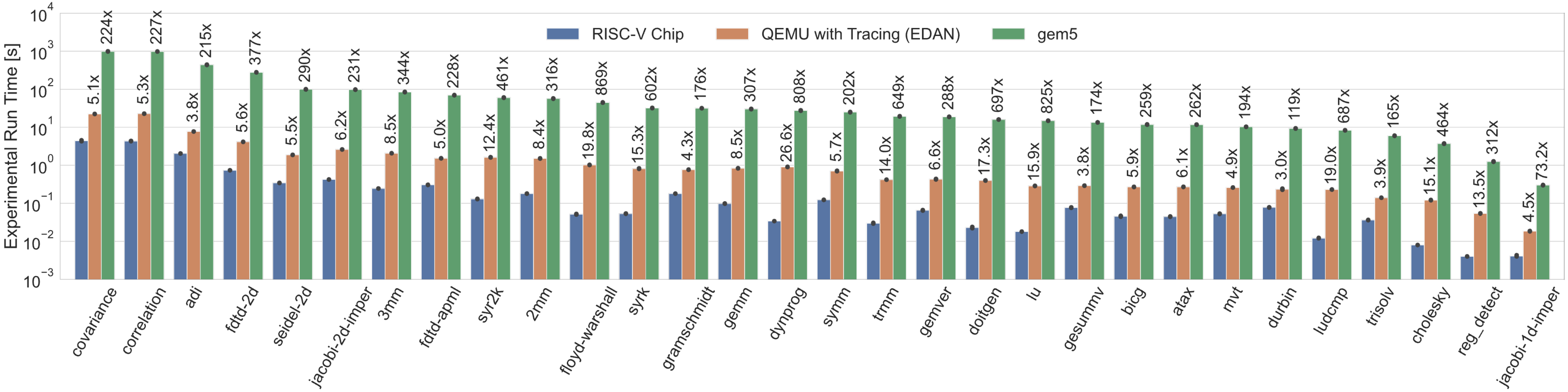
Challenges in Measuring Latency Sensitivity

Hardware Modification

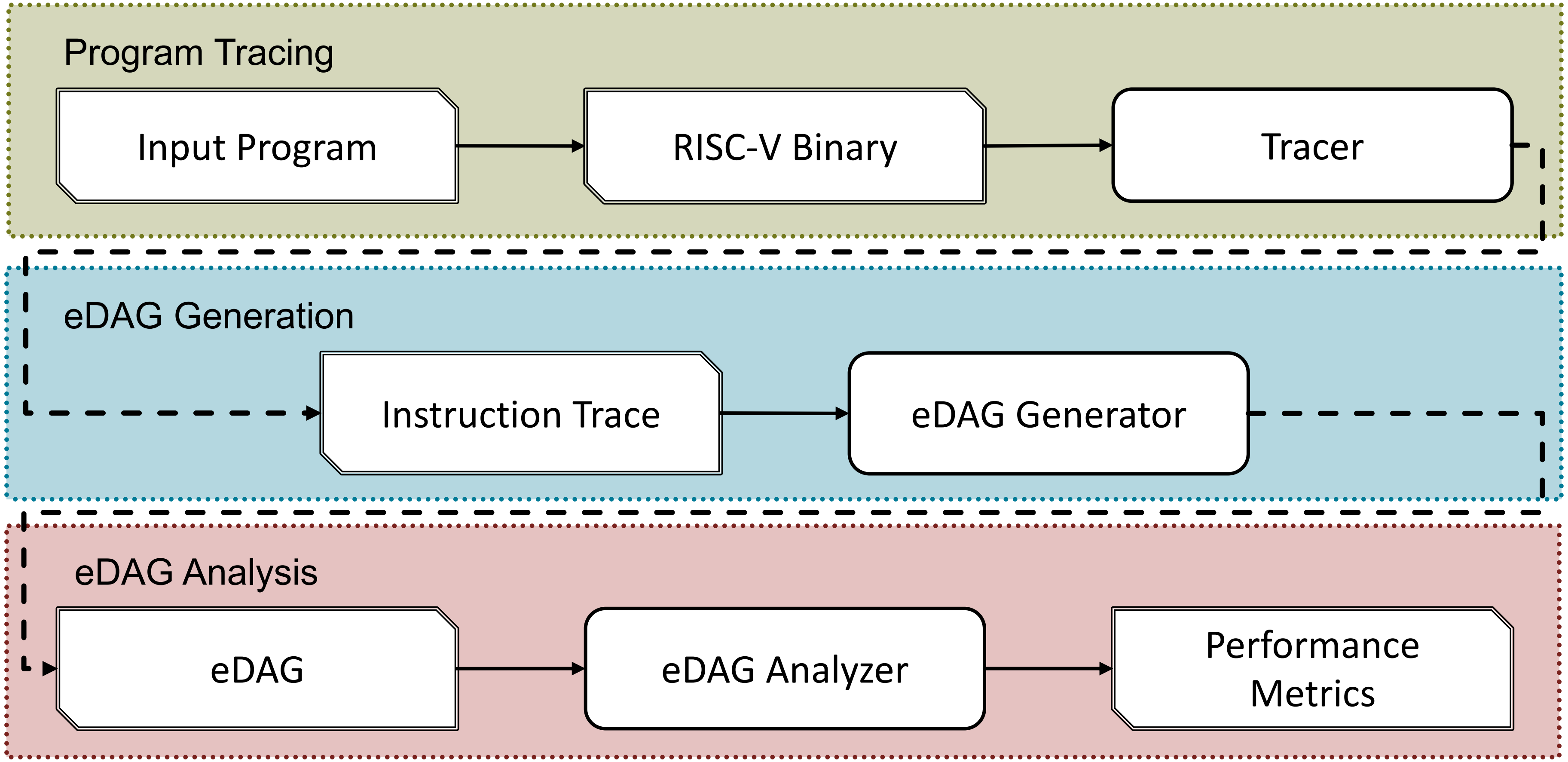
Difficult to procure and inflexible

Simulation

Execution is often very time-consuming



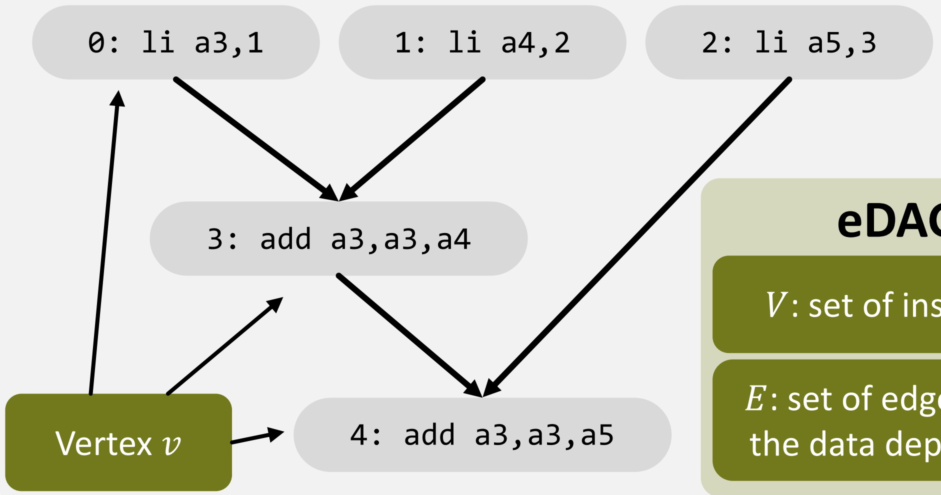
Introducing EDAN



Background: Execution DAG (eDAG) and Brent's Lemma

```
int a = 1;
int b = 2;
int c = 3;
int sum = a + b + c;
```

Execution DAG (eDAG)



eDAG G

V: set of instructions

E: set of edges defining the data dependencies

$t(v)$: execution time of instruction

$T_1 = 5, T_\infty = 3,$
$Parallelism = \frac{5}{3}$

Work

$$T_1 = \sum_{v \in V} t(v)$$

If all vertices take **unit time** to execute, T_1 equals the **total number of vertices** in G.

Depth/Span

T_∞: the time required to execute all instructions on the critical path.

Degree of Parallelism

$$Parallelism = \frac{Work}{Depth} = \frac{T_1}{T_\infty}$$

Bounds on Execution Time T_p

p: number of processors

$$max\left\{\frac{T_1}{p}, T_\infty\right\} \le T_p \le \frac{T_1 - T_\infty}{p} + T_\infty$$

Work and **depth laws**

Brent's lemma (assuming a greedy scheduler is used)

eDAG Generation Process

Input Program → RISC-V Binary → Tracer → Instruction Trace → eDAG Generator → eDAG

Compiled with GCC 12.2.0 with O3 optimization.

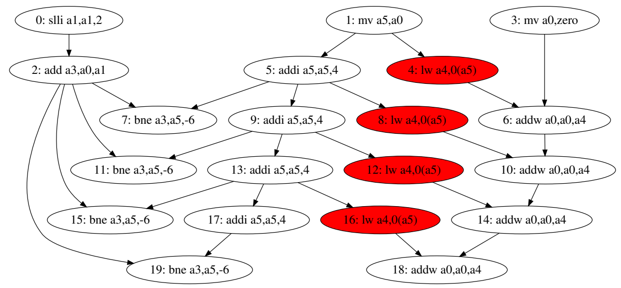
Cache model: simulate cache hits and misses

```
#define N 4
int kernel(int *arr, int n)
{
  int i, sum = 0;
  // Perform summation
  for (i = 0; i < n; ++i)
      sum += arr[i];
  return sum;
}
```

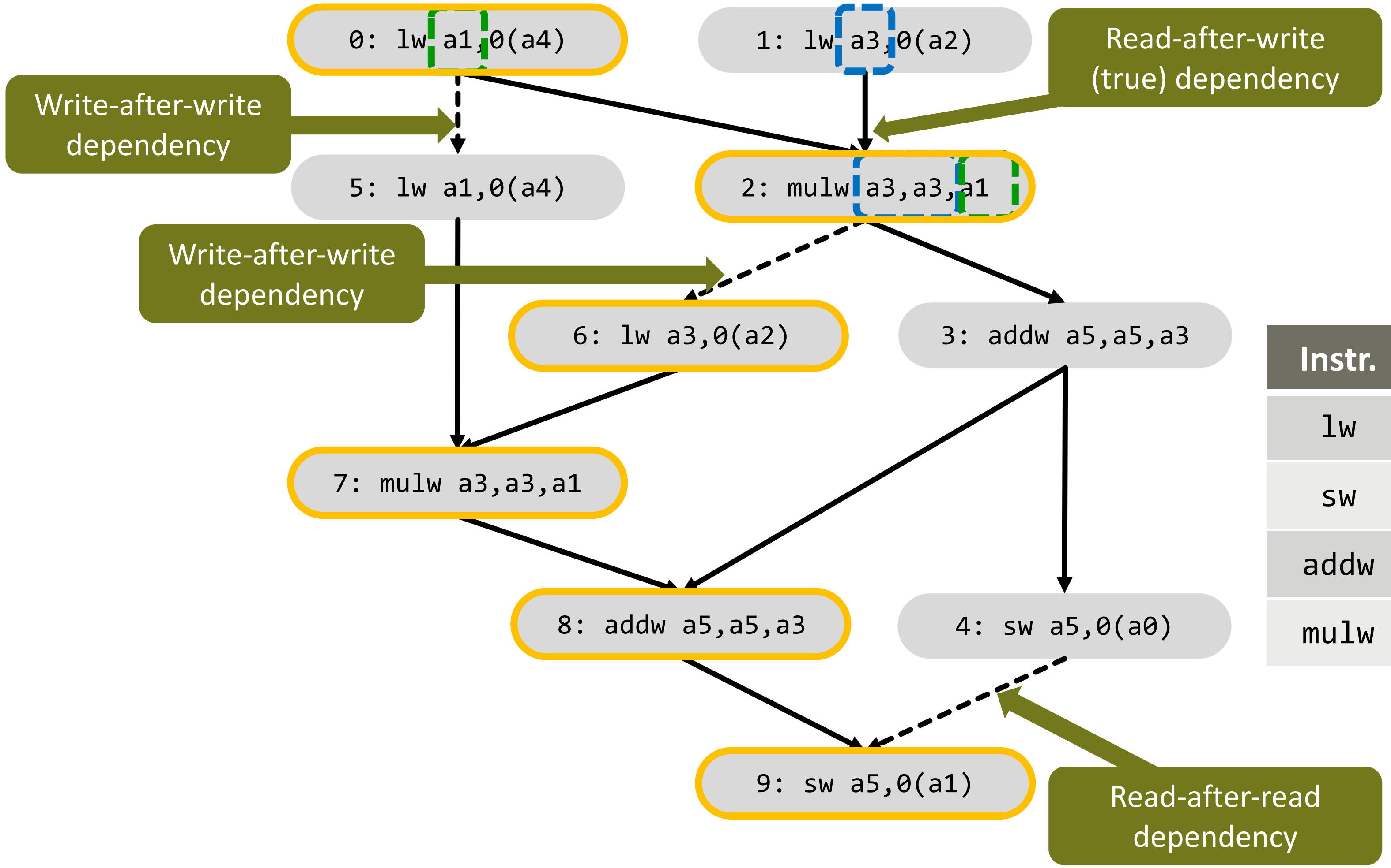
QEMU

- Custom Tiny Code Generator (TCG) plugin
- Fast tracing
- Usable for other ISAs

```
add a3,a0,a1
mv a0,zero
lw a4,0(a5);0x40080290
addi a5,a5,4
addw a0,a0,a4
bne a3,a5,-6
lw a4,0(a5);0x40080294
addi a5,a5,4
addw a0,a0,a4
```



Exposing Potential Parallelism

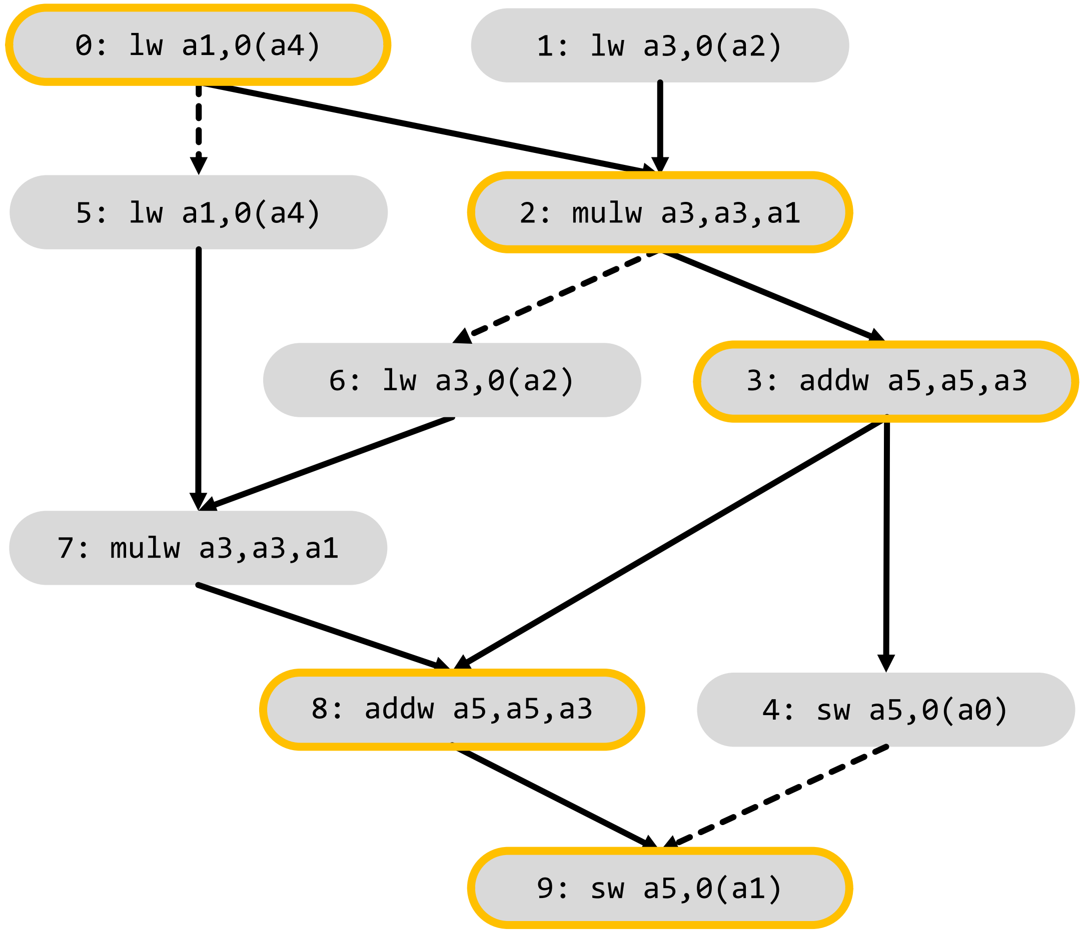


Depth: $T_\infty = 6$

$$\frac{T_1}{T_\infty} = \frac{10}{6} \approx 1.67$$

Instr.	Desc.	Source	Dest
lw	Load	op2	op1
sw	Store	op1	op2
addw	Addition	op2, op3	op1
mulw	Multiplication	op2, op3	op1

Exposing Potential Parallelism (cont.)



Depth: $T_\infty = 5$

$$\frac{T_1}{T_\infty} = \frac{10}{5} = 2$$

Removing **non-true dependencies** exposes potential parallelism

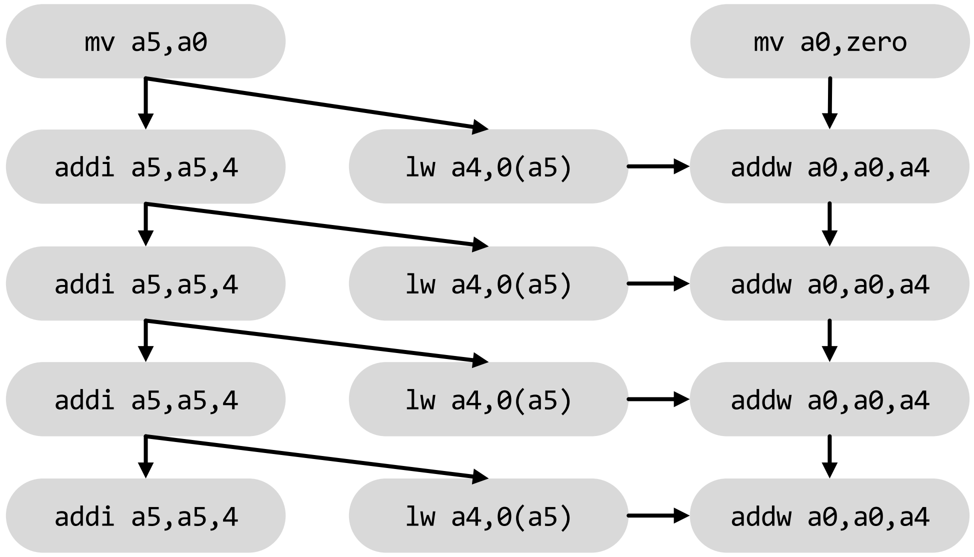
Memory Cost Model

Summation Kernel

```c
#define N 4
int kernel(int *arr, int n)
{
  int i, sum = 0;
  // Perform summation
  for (i = 0; i < n; ++i)
      sum += arr[i];
  return sum;
}
```

α: RAM access latency in cycles

m: number of memory accesses that can be issued in parallel



Bounds on Execution Time $T_{m,\alpha}$

$$max\left\{\frac{\mathcal{W}}{m}, \mathcal{D}\right\}\alpha + C \leq T_{m,\alpha} \leq \left(\frac{\mathcal{W}-\mathcal{D}}{m} + \mathcal{D}\right)\alpha + C$$

Derived from the **work & depth laws** and **Brent's lemma**

Memory Work

$\mathcal{W}$: total number of memory access instructions to RAM (i.e., cache misses)

$\mathcal{W} = 4$

Memory Depth

$\mathcal{D}$: maximum number of RAM access instructions on one critical path

$\mathcal{D} = 1$

Non-memory Access Instruction Cost

C: total cost of instructions that do not access the RAM (e.g., computation, cache hits)

$C = 10$

Memory Latency Sensitivity Metrics

m: Number of memory issue slots
α: RAM access latency
$\mathcal{W}$: Memory work
$\mathcal{D}$: Memory depth

Upper-bound of Execution Time $T_{m,\alpha}$

$$T_{m,\alpha} \le \left(\frac{\mathcal{W}-\mathcal{D}}{m} + \mathcal{D}\right)\alpha + C$$

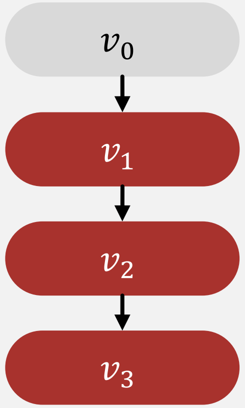
Based on the theory of **sensitivity analysis** in mathematics

Memory Latency Sensitivity λ

$$\lambda \le \frac{\partial\left(\left(\frac{\mathcal{W}-\mathcal{D}}{m} + \mathcal{D}\right)\alpha + C\right)}{\partial \alpha} = \frac{\mathcal{W}-\mathcal{D}}{m} + \mathcal{D} = \frac{1}{m}\mathcal{W} + \left(1 - \frac{1}{m}\right)\mathcal{D}$$

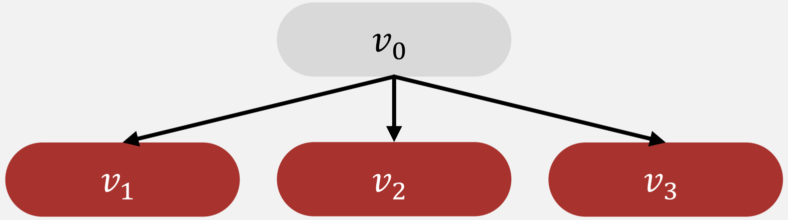
Weighted sum of $\mathcal{W}$ and $\mathcal{D}$ based on the number of memory issue slots

Latency Sensitive Application



$m = 3$
$\mathcal{W} = 3$
$\mathcal{D} = 3$
$\lambda = 3$

Latency Insensitive Application



$m = 3$ $\quad$ $\mathcal{W} = 3$, $\mathcal{D} = 1$ $\quad$ $\lambda = 1$

Validation of Memory Latency Sensitivity

Server configuration:

- AMD Ryzen 5 CPU
- 16 GB DDR3 RAM

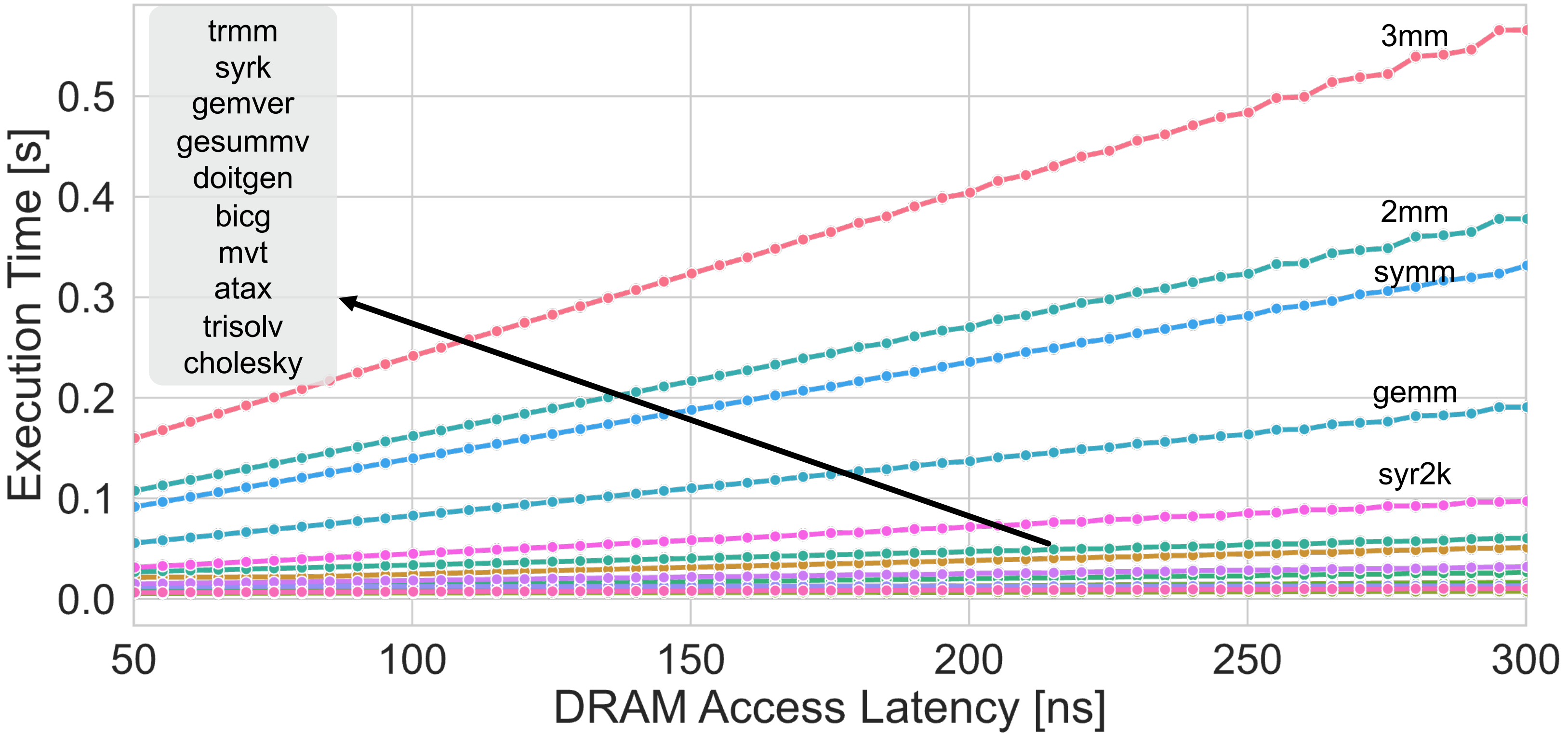
Varied DRAM latency for all memory access instructions in gem5 across **PolyBench** kernels.

gem5: 24 hours
EDAN: 1 hour

gem5 configuration:

- 1 GHz RiscvO3CPU
- 16 GB DRAM
- 16 KB L1i, 64 KB L1d cache
- 256 KB L2 cache

Parameter sweep of DRAM access latency from 50 to 300 ns at 5 ns increment (51 data points)

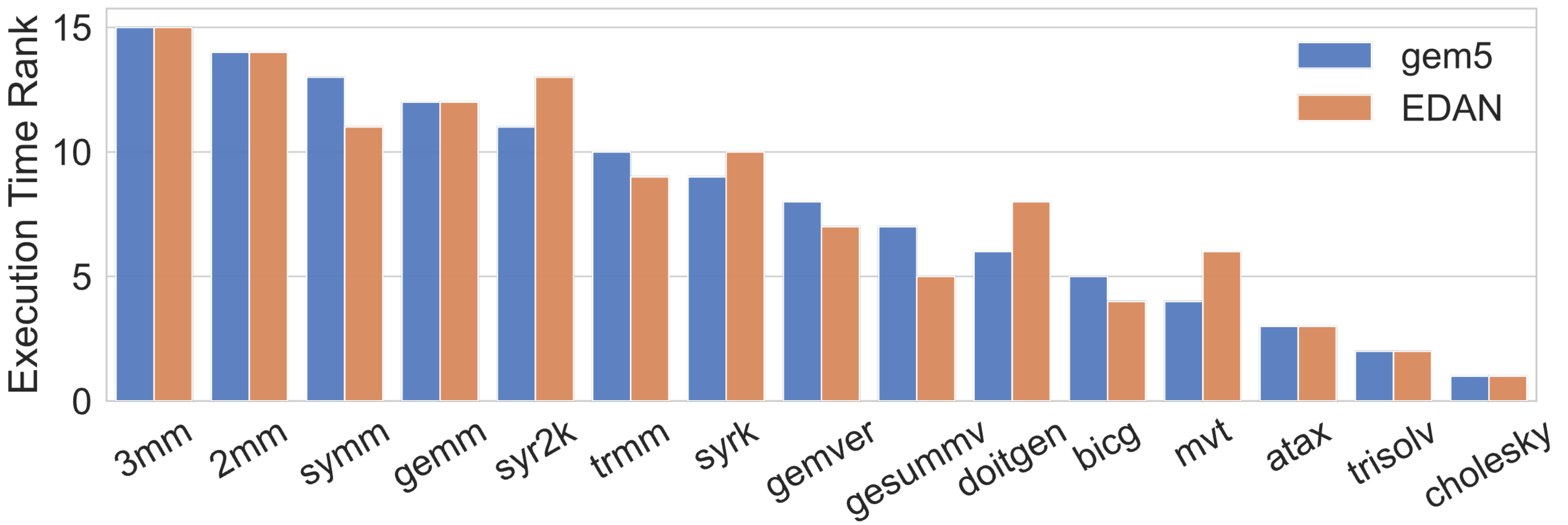


Validation of Memory Latency Sensitivity (cont.)

Server configuration:
- AMD Ryzen 5 CPU
- 16 GB DDR3 RAM

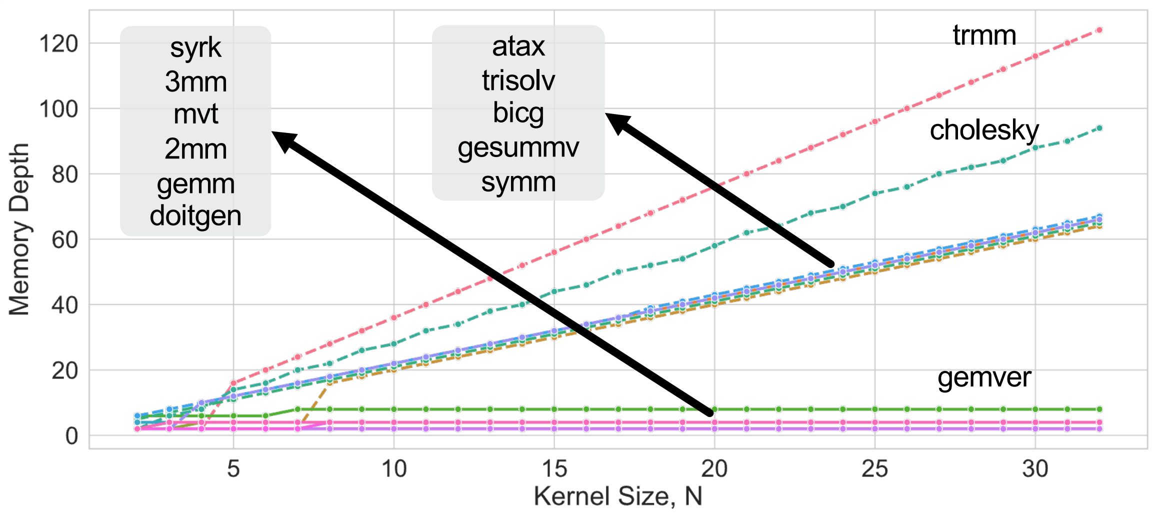
gem5 configuration:
- 1 GHz RiscvO3CPU
- 16 GB DRAM
- 16 KB L1i, 64 KB L1d cache
- 256 KB L2 cache

Rank the benchmarks based on the impact of memory latency and compare with the λ generated from EDAN by analyzing application eDAGs.



! The ranks differ by a maximum of 2

Case Studies: PolyBench



Data-oblivious applications should exhibit **constant memory depths**

Snippet from trmm Kernel

```
/* trmm: B := alpha*A'*B, A triangular */
for (i = 1; i < _PB_NI; i++)
  for (j = 0; j < _PB_NI; j++)
    for (k = 0; k < i; k++)
      B[i][j] += alpha * A[i][k] * B[j][k];
```

Too many values to keep in distinct registers between the first and last access to `B[i][j]`.

Register spilling causes some benchmarks to have a linear memory depth

Case Studies: HPCG

Server configuration:
- Intel Xeon X7550 CPU
- 1 TB DDR3 RAM
- PERC H700 hard disk

- 210 million lines of instructions
- 5.5 GB trace file generated in 35 seconds

EDAN cache configuration:
- Write-through
- 64-byte cache line
- 2-way associative
- LRU eviction strategy

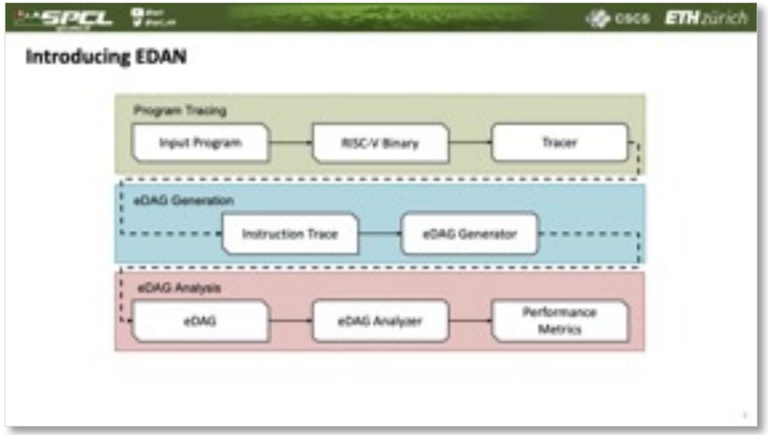
HPCG

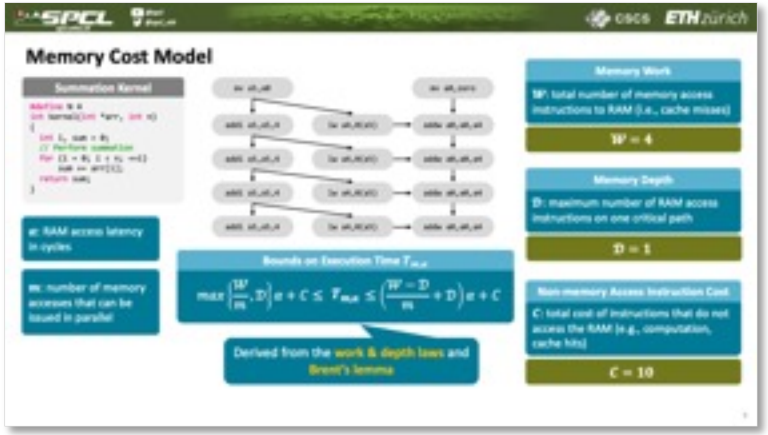
- Solve a large sparse linear system with the pre-conditioned conjugate gradient (PCG) method. Adopted by the **TOP500** list
- Low arithmetic intensity

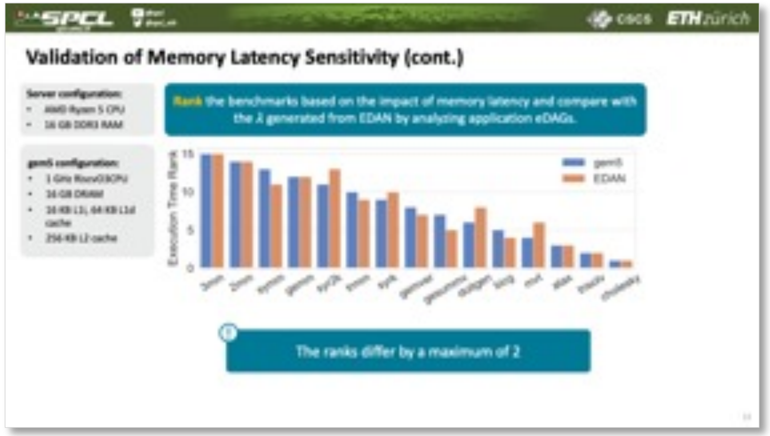
Cache Size	$\mathcal{W}$	$\mathcal{D}$	λ
No Cache	106151255	73703	26593091
32 KB	11200012 (89.4%)	45102 (38.8%)	2833830 (89.3%)
64 KB	10833505 (89.8%)	43502 (41.0%)	2741003 (89.7%)

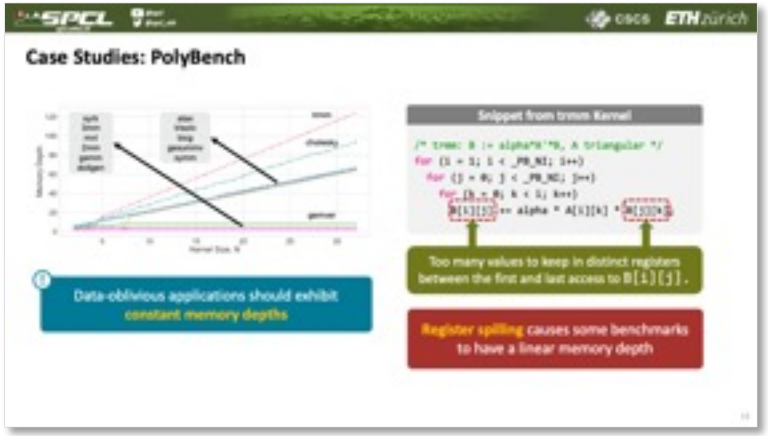
EDAN: 7 hours
gem5: 4 days (estimated)

Conclusions









More of SPCL's research:

youtube.com/@spcl — 210+ Talks

twitter.com/spcl_eth — 1.6K+ Followers

github.com/spcl — 5.6K+ Stars

… or spcl.ethz.ch

Relative memory latency sensitivity, bandwidth utilization visualization, and many **more results in the paper**

https://github.com/ss16118/sc23-edan