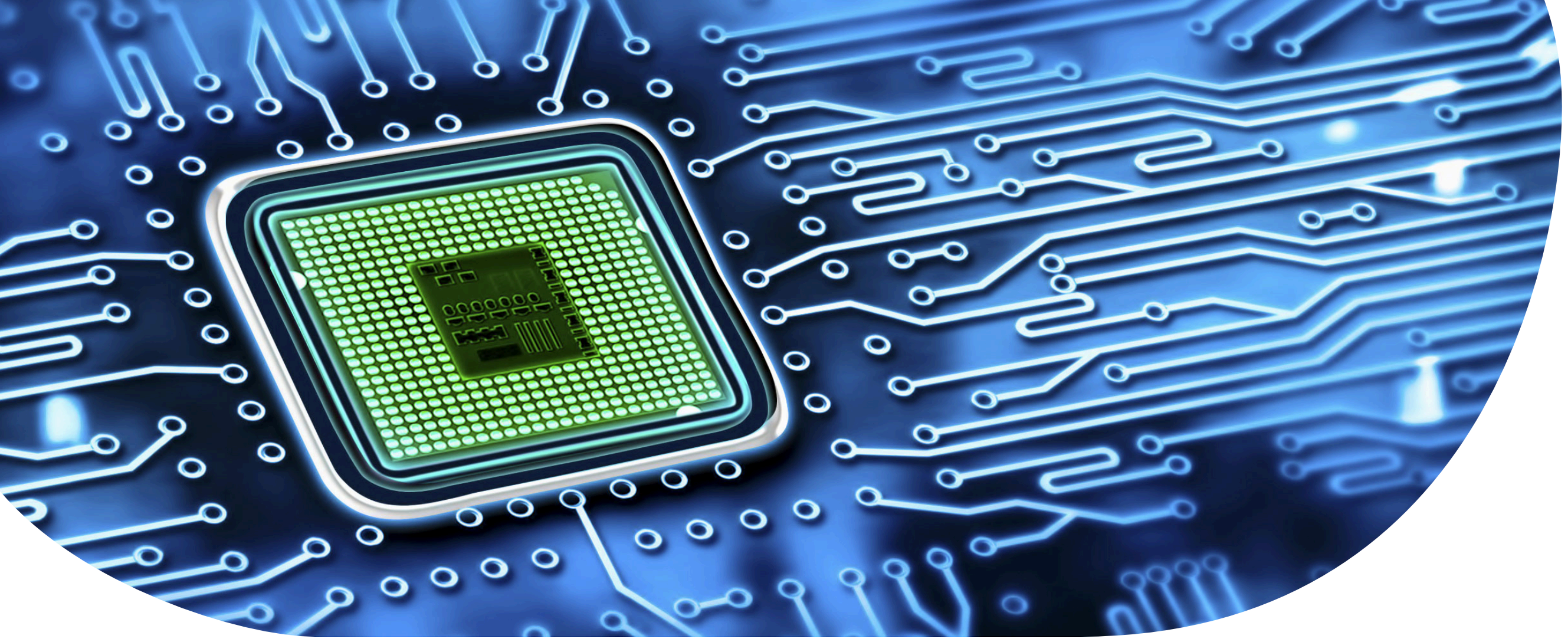


جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Stencils, Solvers and Sphere Decoders on FPGAs

H2RC 2022

Suhaib A Fahmy
suhaib.fahmy@kaust.edu.sa

FPGAs for High Performance Computing

- 8th edition of this workshop continues to stake a claim
- FPGAs' unique features continue to attract attention, primarily:
  - Efficiency: reducing energy footprint
  - Irregularity and precision: offering more tailored computing
- Evolution in the last decade:
  - Productive HLS tools

Three applications

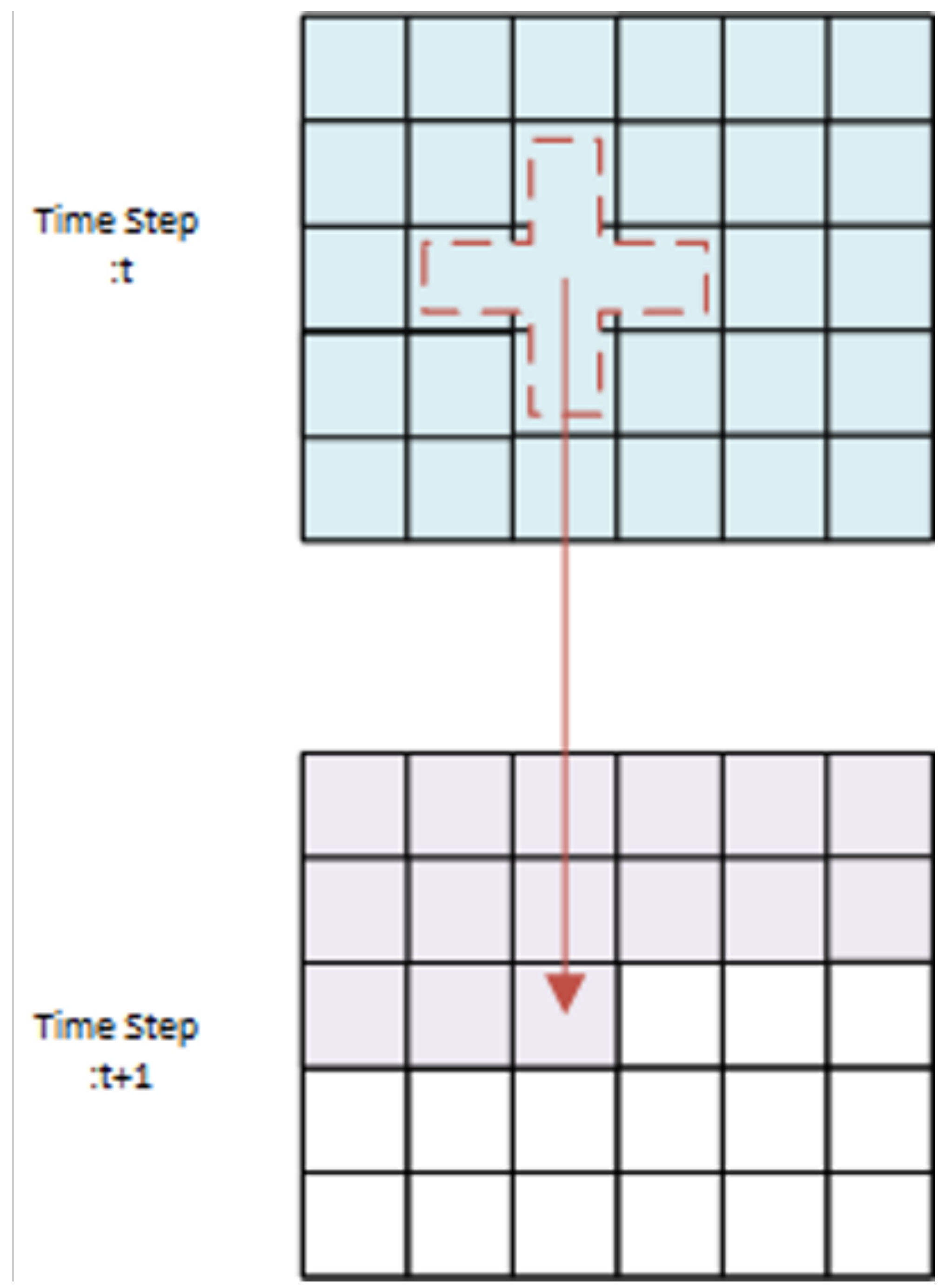
- Structured mesh stencils
- Tridiagonal system solvers
- Sphere decoding for Massive MIMO

Structured mesh based stencil solvers

- Finite Difference Methods (FDM) used to solve PDEs numerically, stencils are used to specify required points

```
for t in range(n_iter)
   for x in range(height)
      for y in range (width)
```

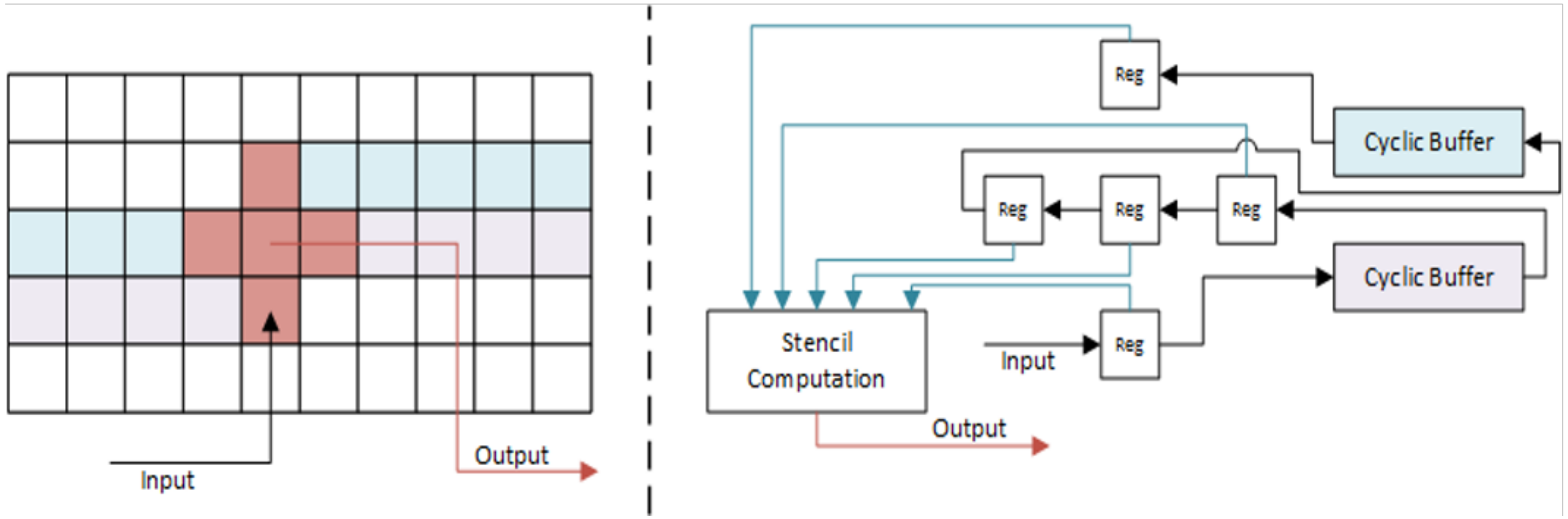
$$U^{t+1}_{x,y} = k_1 U^t_{x-1,y} + k_2 U^t_{x,y-1} + k_3 U^t_{x+1,y} + k_4 U^t_{x,y+1} + k_5 U^t_{x,y}$$

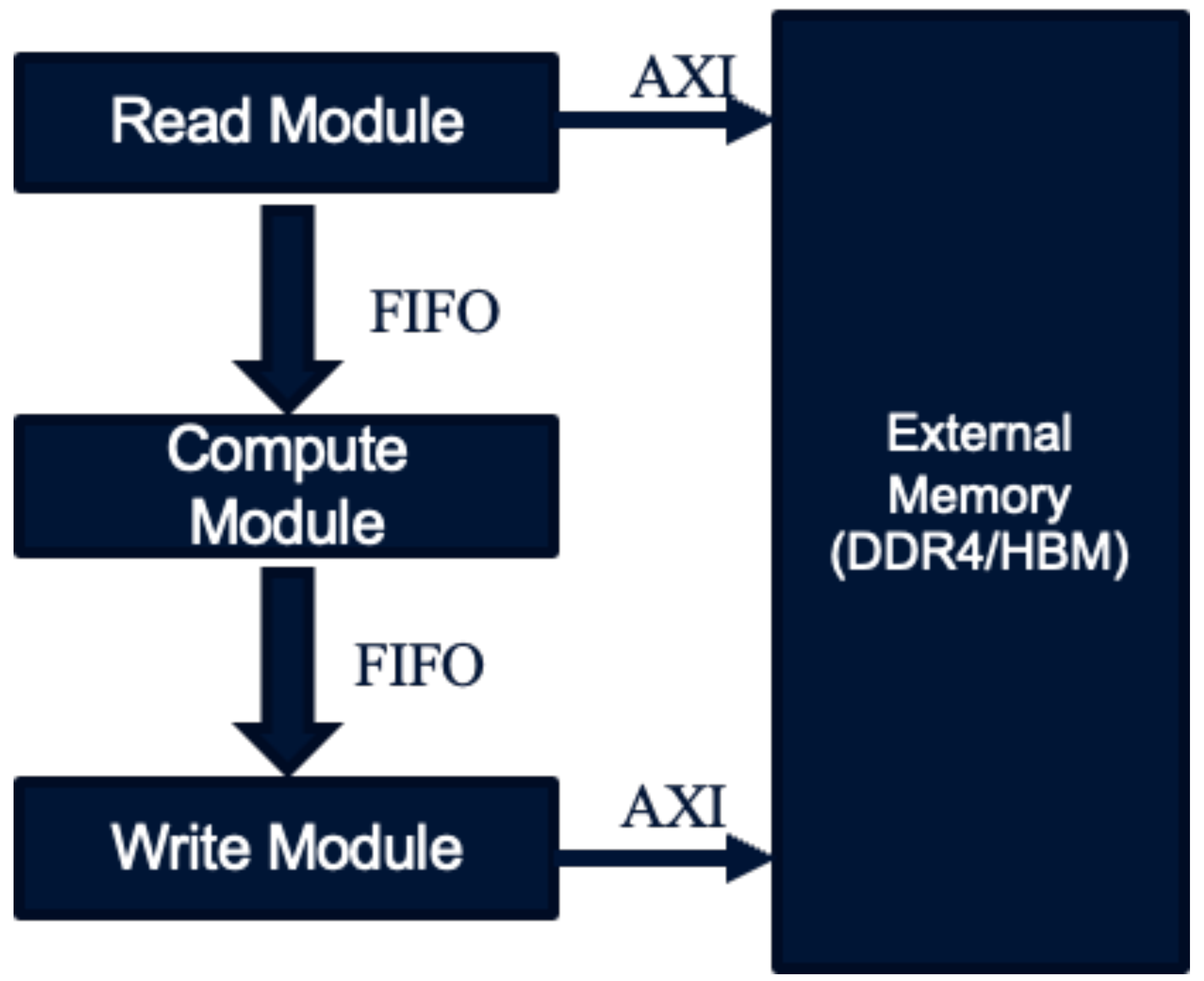


- Intrinsically parallel: all cells could be updated in parallel

Stencil processing element

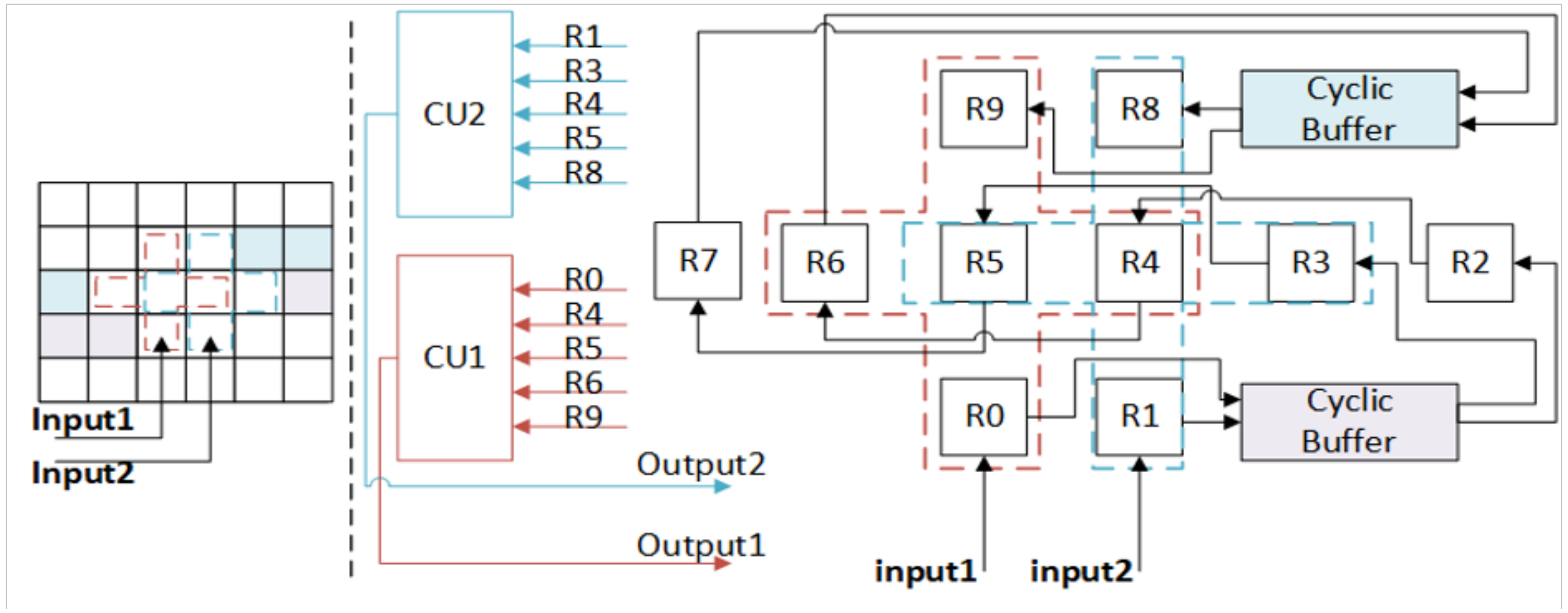
- We build a processing primitive using similar techniques to FIR filters:
  - Line buffers using BRAMs/URAMs
  - Window buffers using registers enabling parallel stencil operation
  - Perfect data reuse: each point is only read from memory once

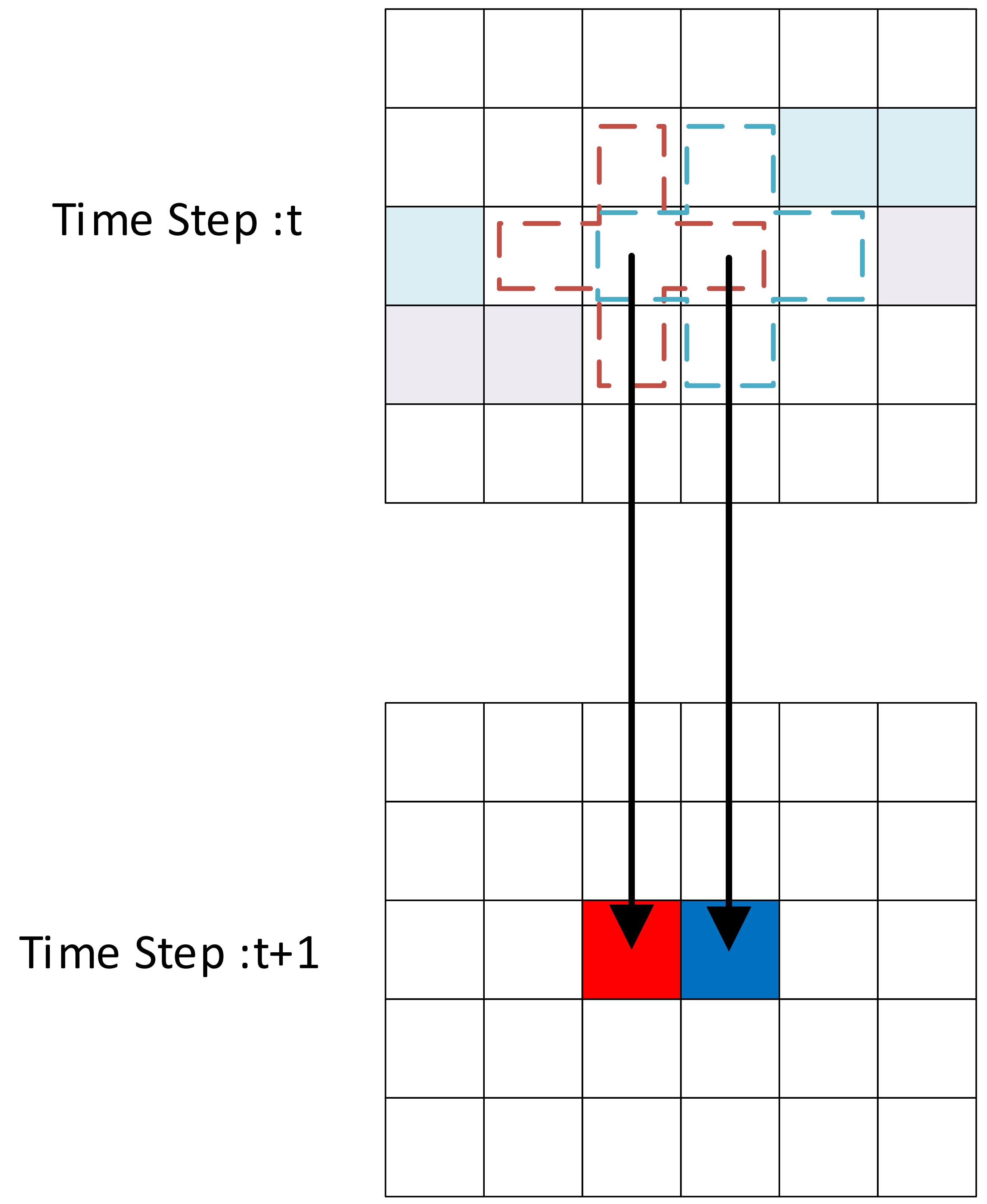




Vectorising stencil operations

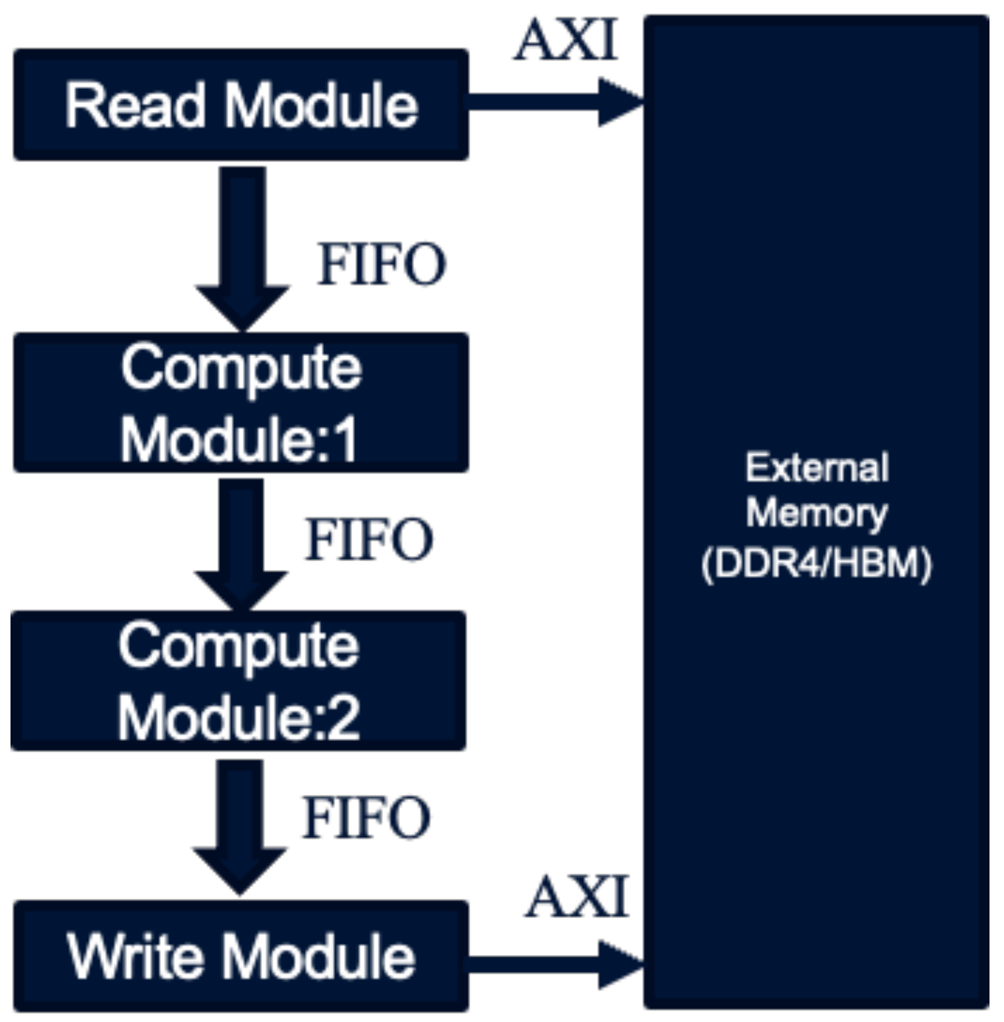
- It is possible to vectorise these operations such that we compute multiple stencil points at a time.
- Buffering remains the same, but bandwidth requirement proportional to vectorisation factor

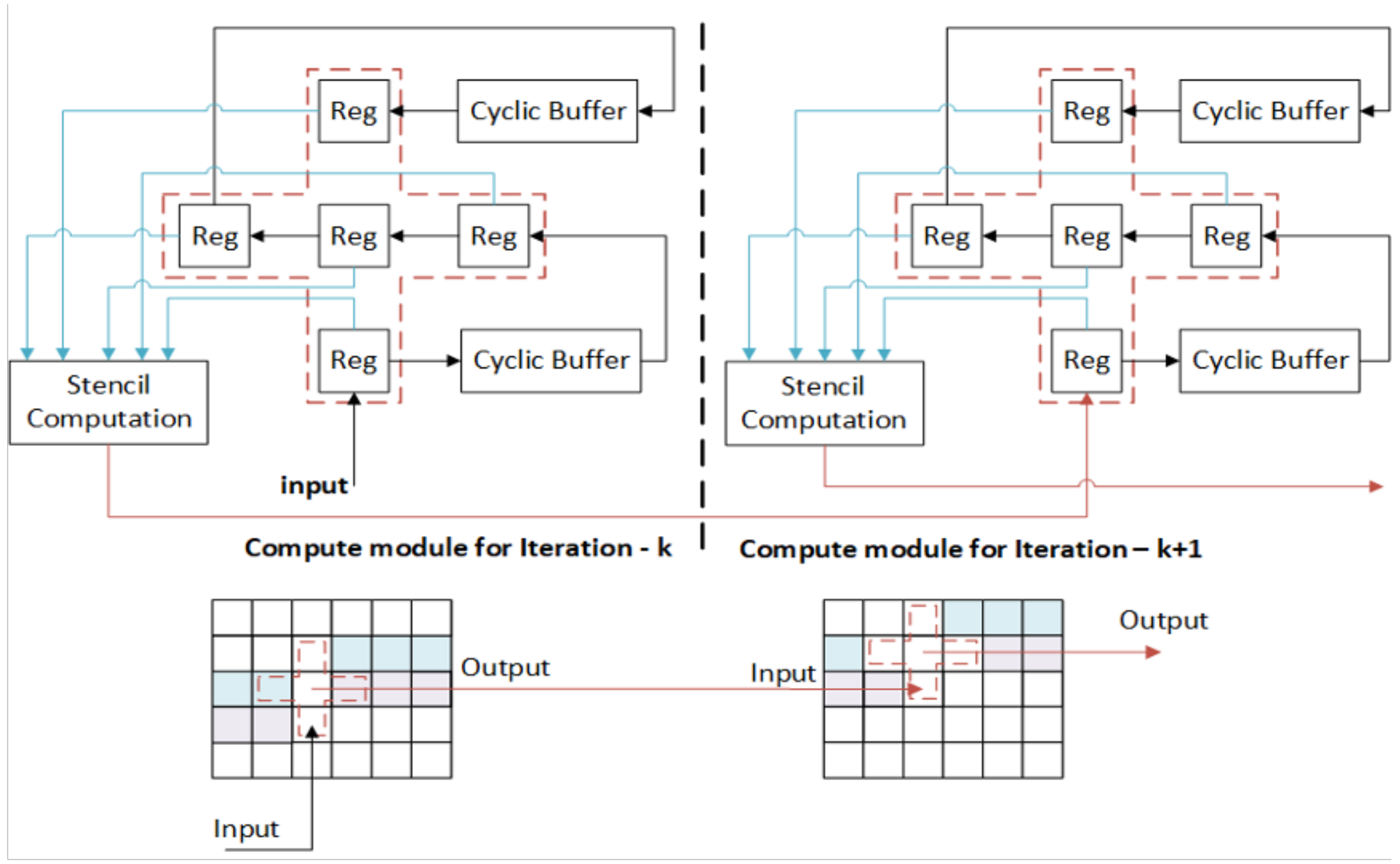




Iterative loop unrolling

- We can also unroll the iterative loop, that is, execute multiple iterations in a pipelined parallel manner
- Bandwidth requirement not affected
- On-chip buffering proportional to unroll factor





Reg
Cyclic Buffer
Stencil Computation
input
Compute module for Iteration - k
Compute module for Iteration – k+1
Output
Input

Performance Model

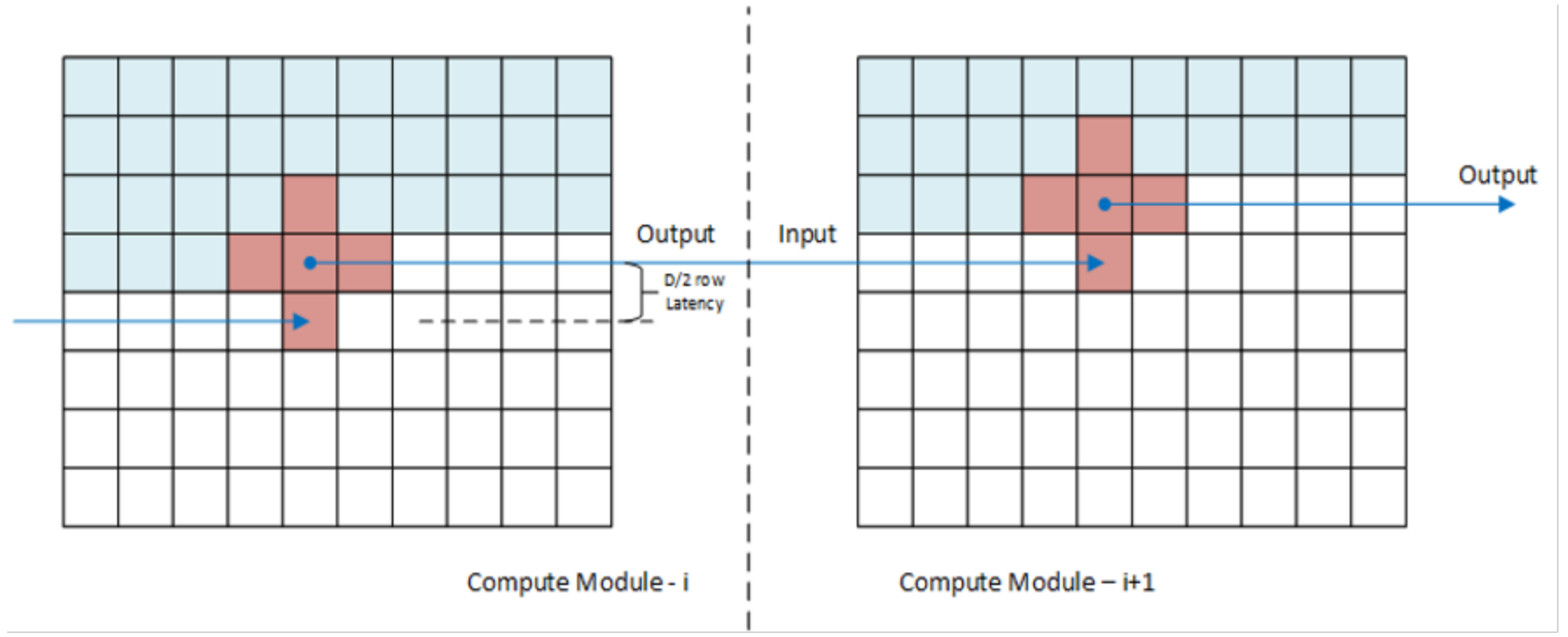
- We can compute the expected latency based on the loop latency and the pipeline latency
- 2D mesh

$$Lat_{2D} = \frac{n_{iter}}{p} \times \left(\left\lceil \frac{m}{V} \right\rceil \times \left(n + p + \frac{D}{2} \right) \right)$$

- 3D mesh

$$Lat_{3D} = \frac{n_{iter}}{p} \times \left(\left\lceil \frac{m}{V} \right\rceil \times n \times \left(l + p + \frac{D}{2} \right) \right)$$

- Model is 85% accurate



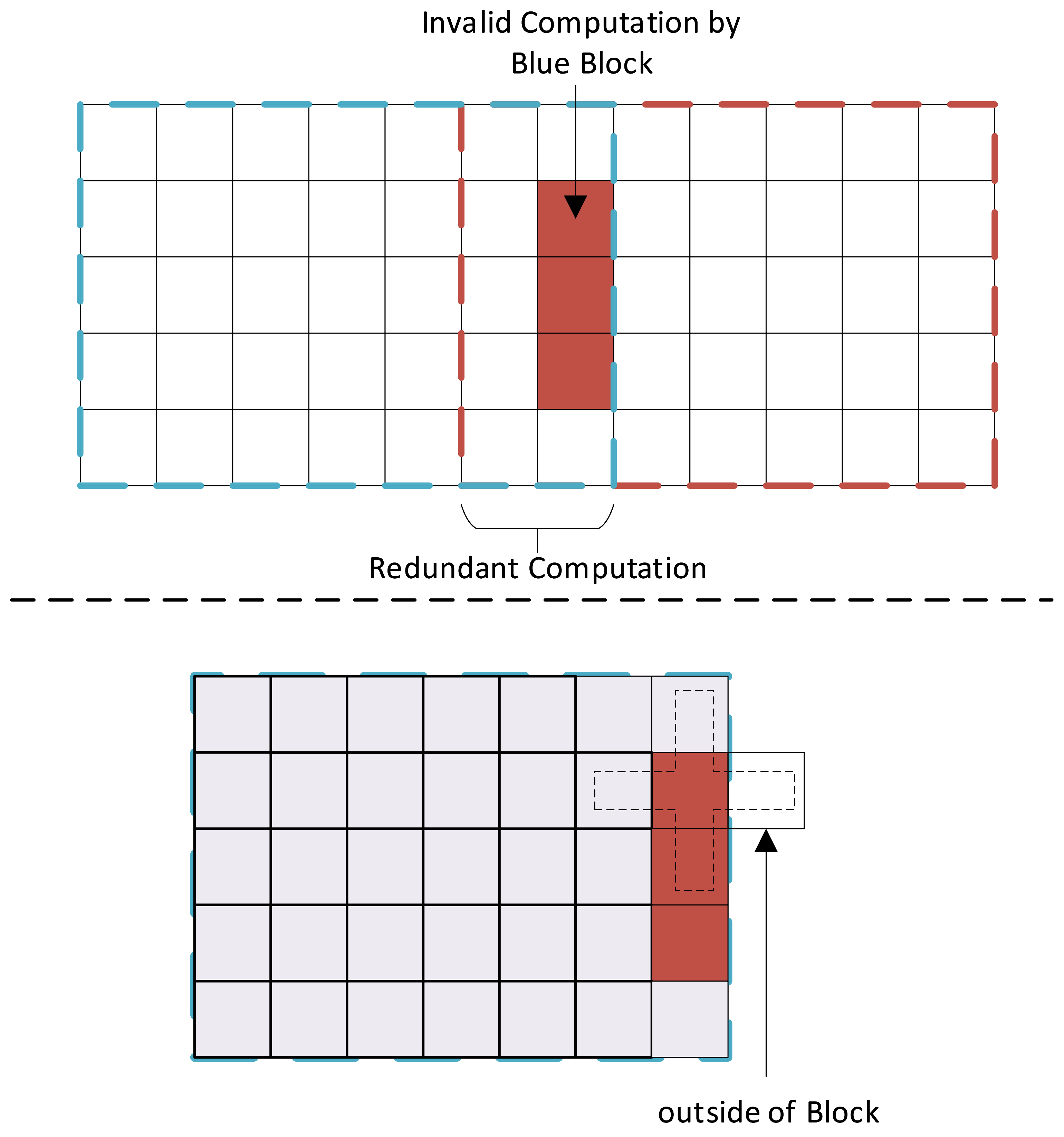
Symbol	Parameter
V	Vectorisation factor
p	Iterative loop unroll factor
D	Stencil order
m,n,l	x, y, z, dimensions of mesh
n_{iter}	Total number of iterations

Resource model

- We can compute a resource model to help guide us to the best parameters for an implementation
  - DSP blocks used by compute units
    - Depends on number of additions and multiplications
  - BRAM/URAM used for cyclic buffers
  - External memory bandwidth proportional to V

Optimising for mesh sizes

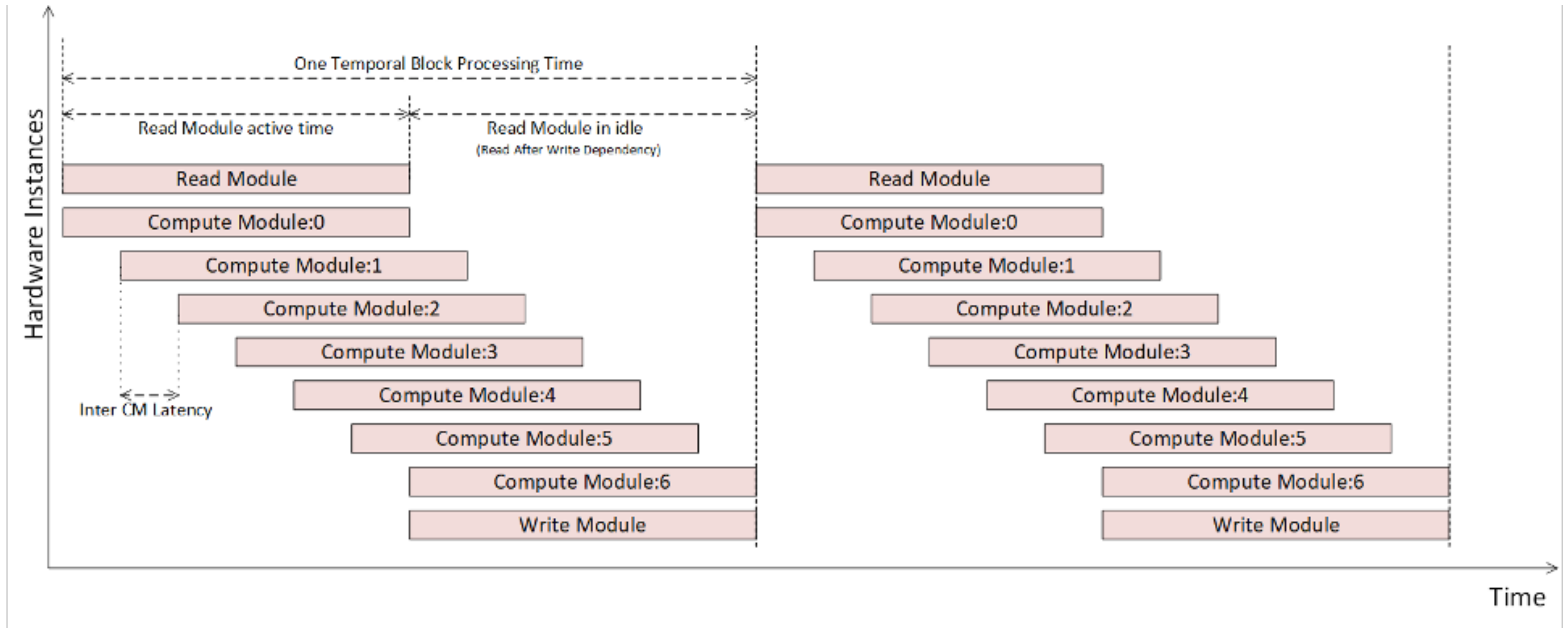
- Large meshes could exhaust on-chip memory for buffering
  - Solve overlapped smaller meshes
  - Some redundant computation:
    - Lower iterative loop unroll
    - Larger spatial blocks
    - Higher vectorisation

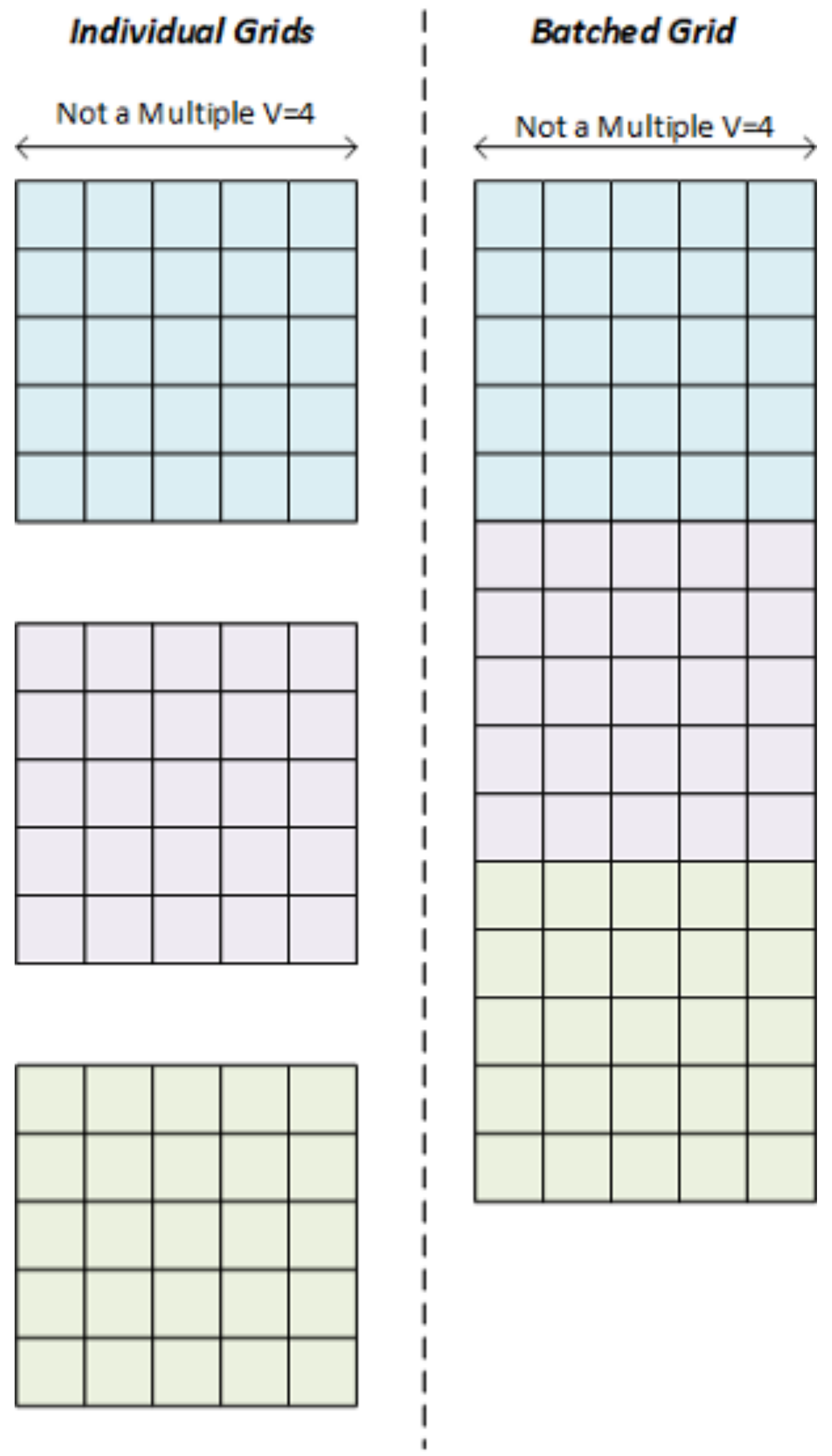


Optimising for mesh sizes

- Small meshes underutilise FPGA resources due to pipeline latency

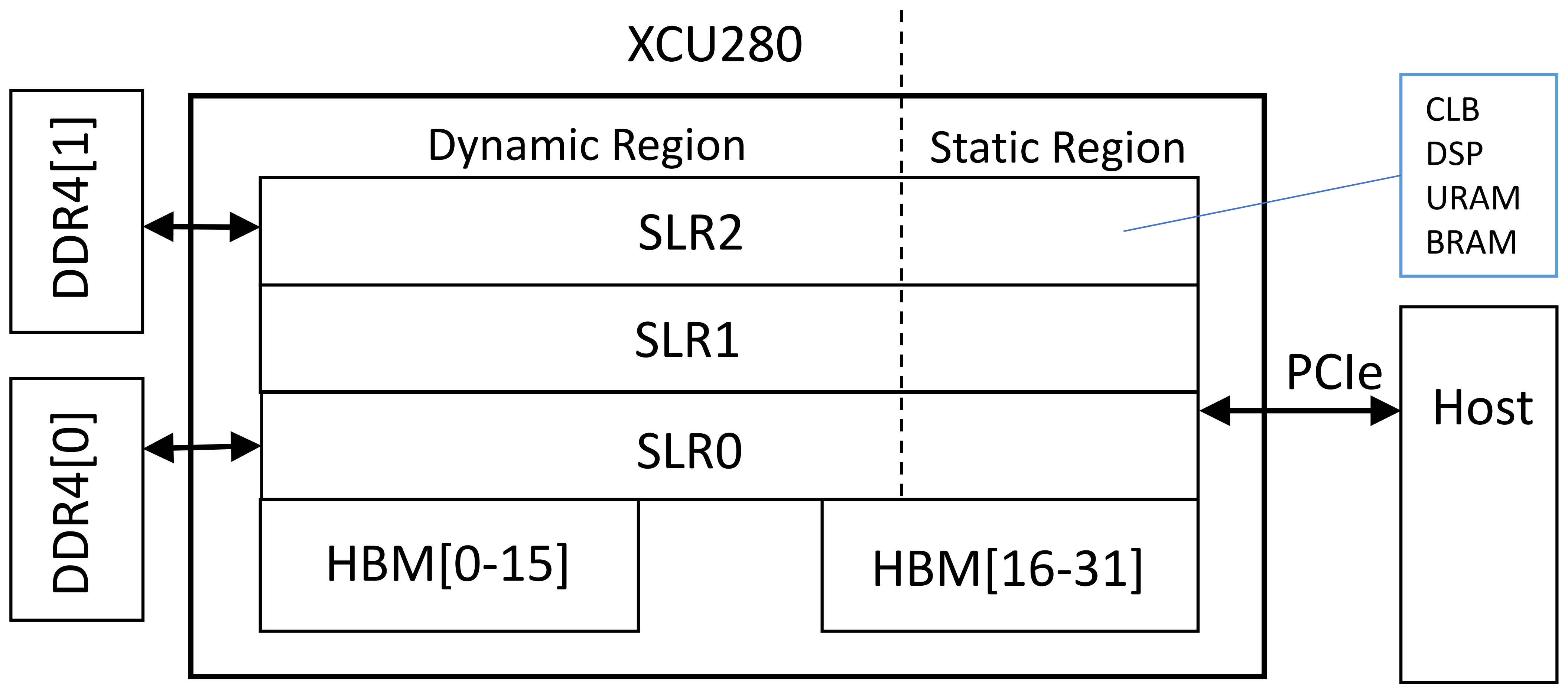
- We can batch smaller meshes into a larger mesh to overcome this latency, solving one dimension with larger size





Stencil evaluation

- Three representative applications
  - Poisson – 2D, 5-point Stencil
  - Jacobi7pt – 3D, 7-point stencil
  - RTM_forward – 3D, 25-point stencils, vector elements
- Implementation using Vitis 2019.2 in C++ targeting Alveo U280



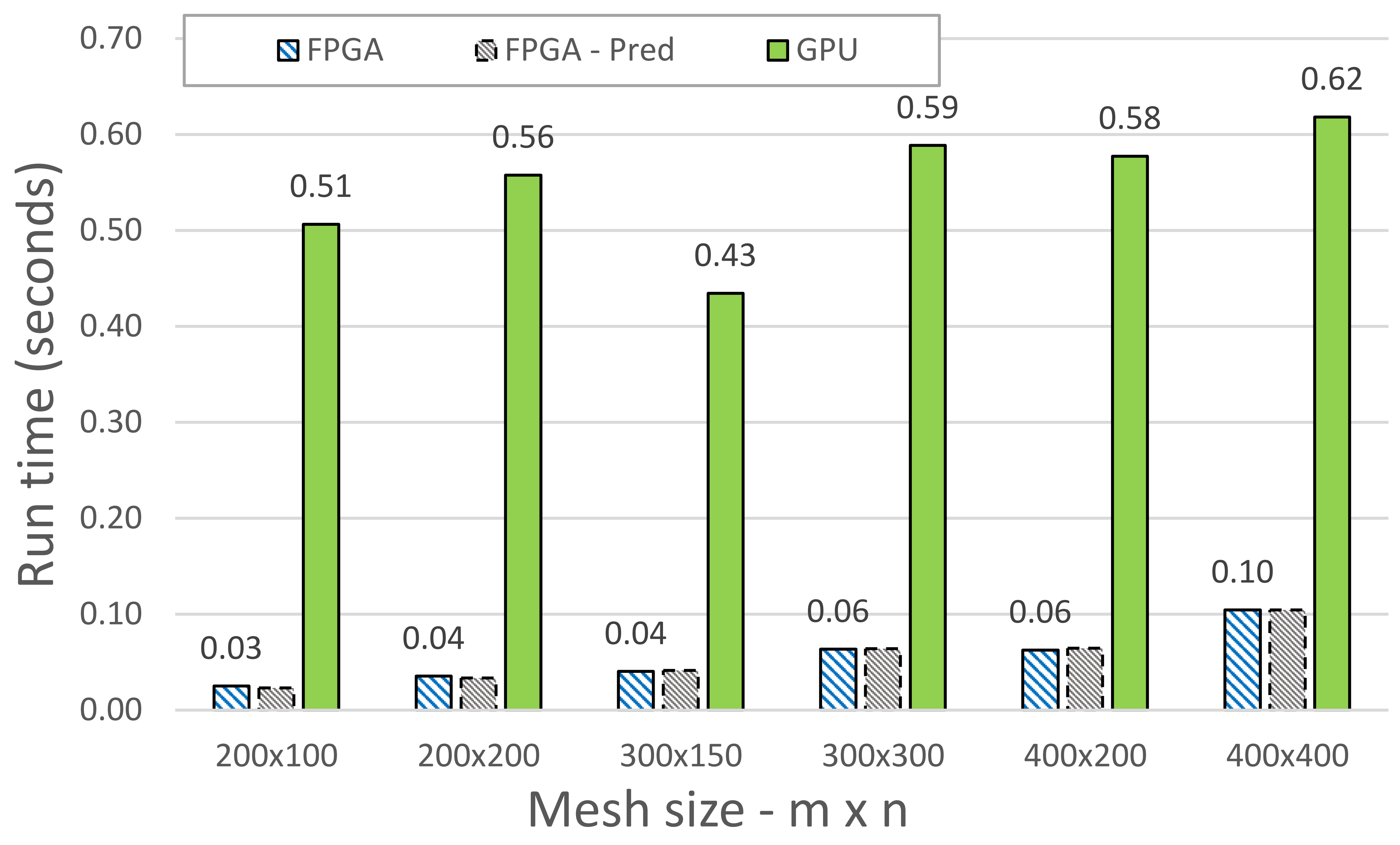
Determining design parameters

- Poisson: 4 add, 2 multiply
- Jacobi: 7 add, 6 multiply
- Determine DSP usage per compute unit then use the model to determine design parameters
- We lower V to support easier routing to memory ports (2 HBM)

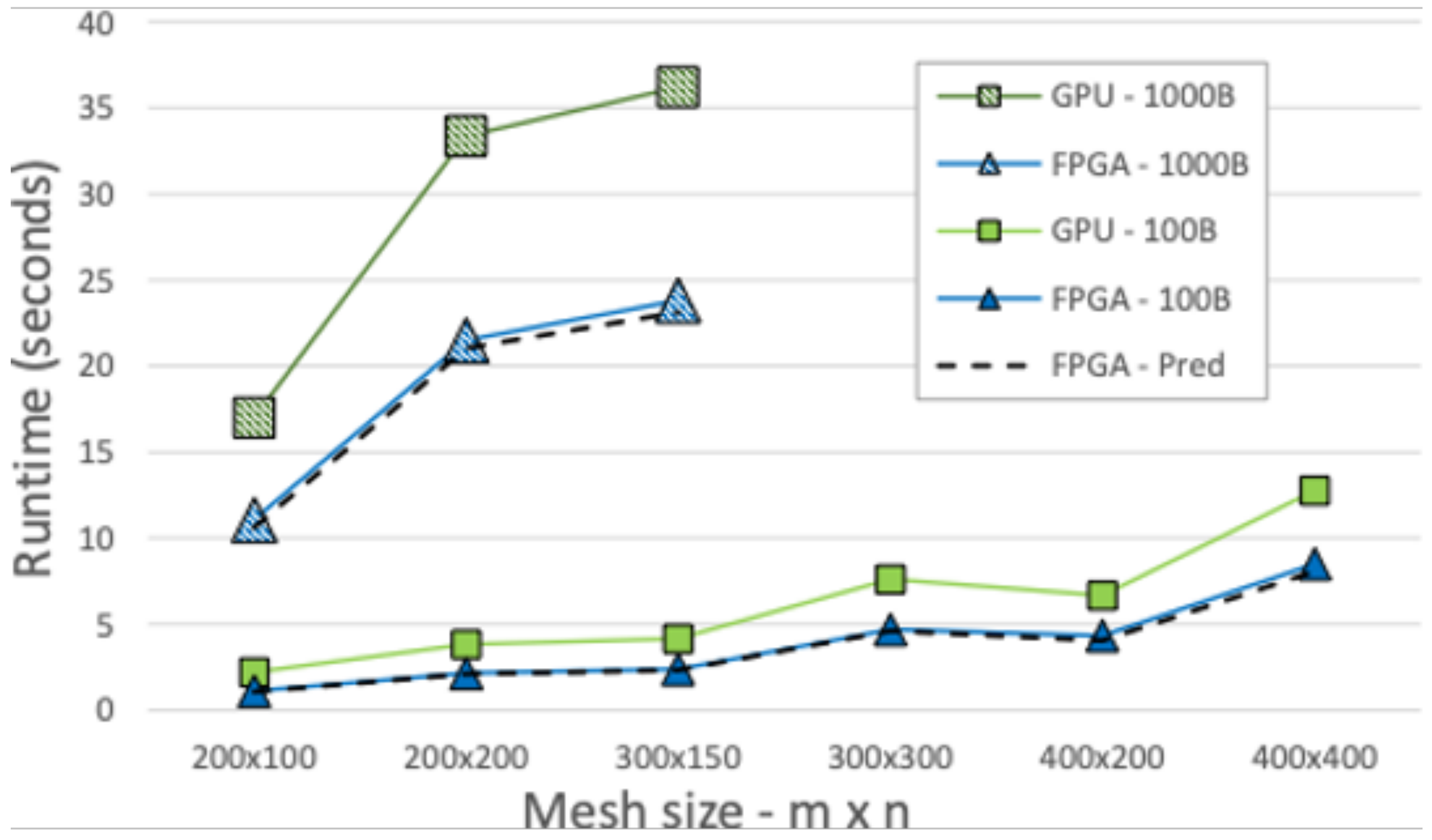
Stencil	Batched - V	Batched - P	Tiled - V	Tiled - P	Tile Size	Tiled:Valid ratio
Poisson	8	68	8	68	8192 x H	98.5
Jacobi	8	28	64	3	768 x 768 x Z	98.4

Results

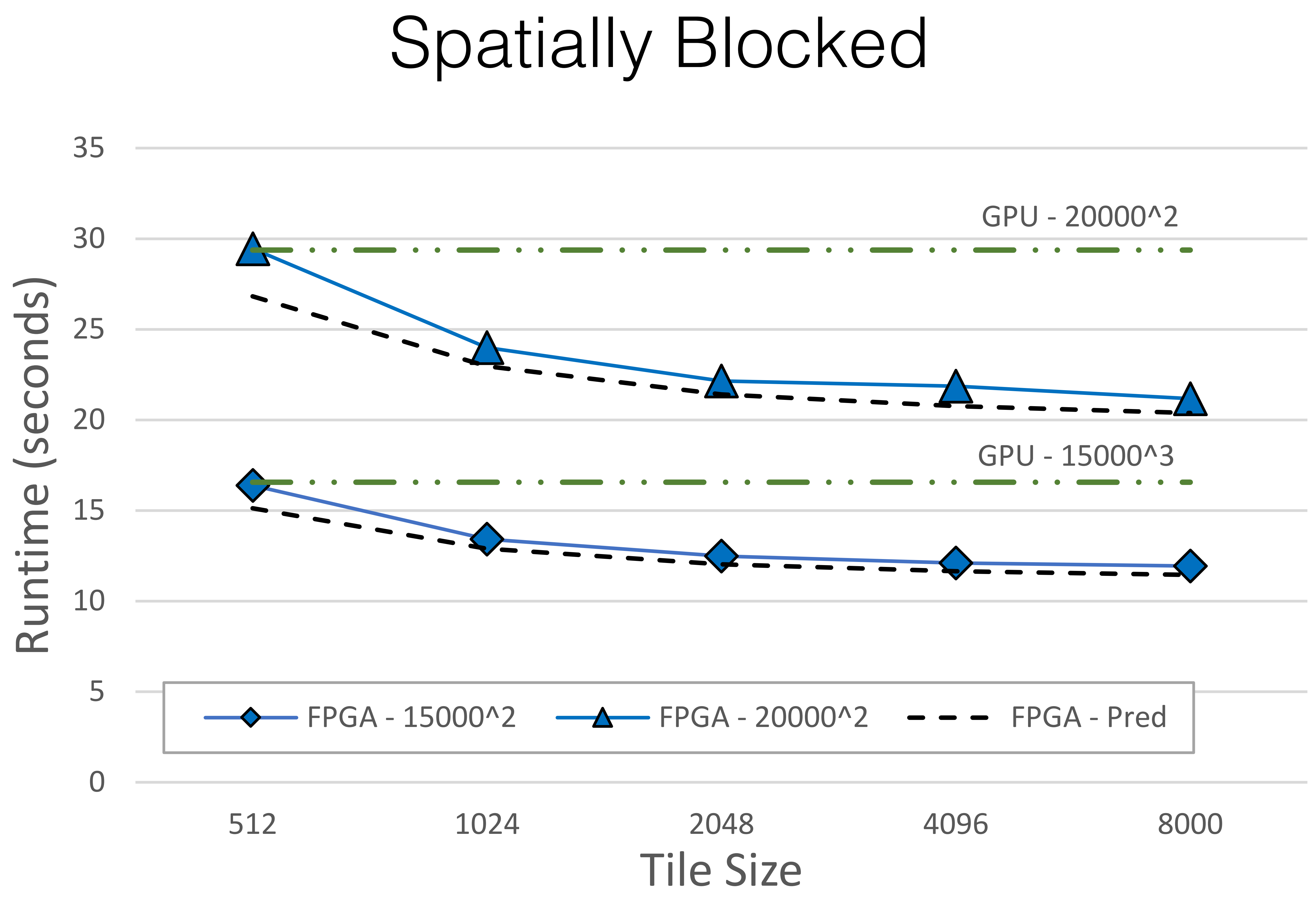
Baseline

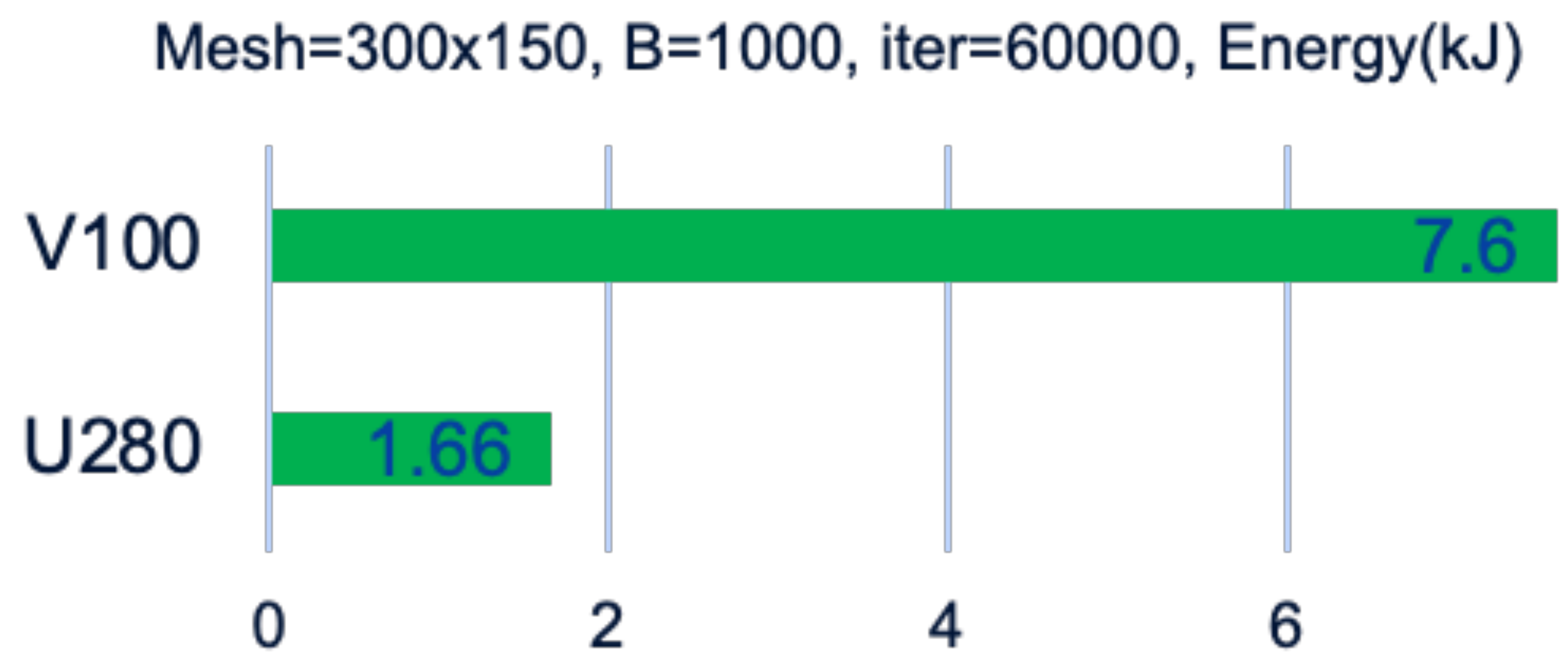


Batched



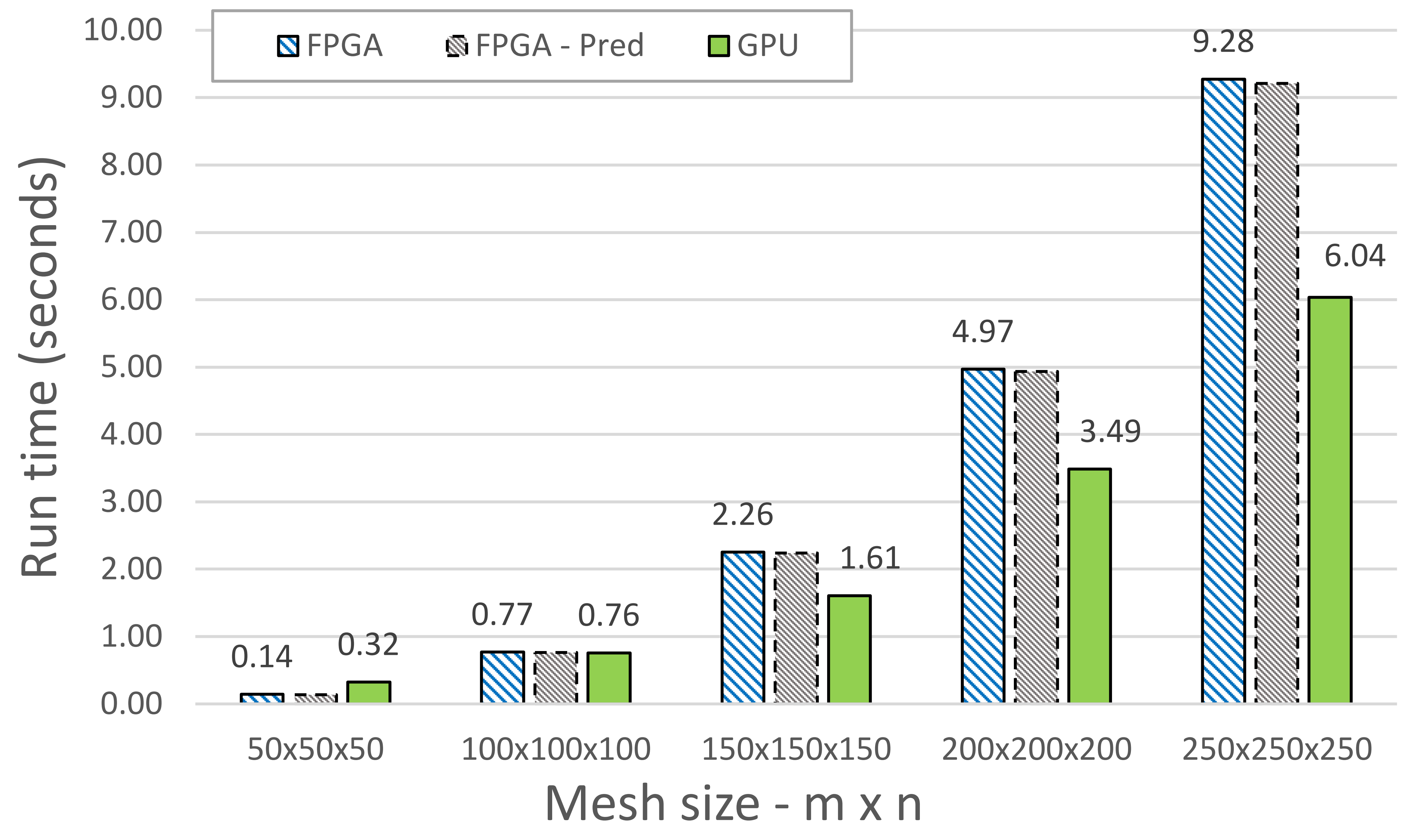
Results – Poisson



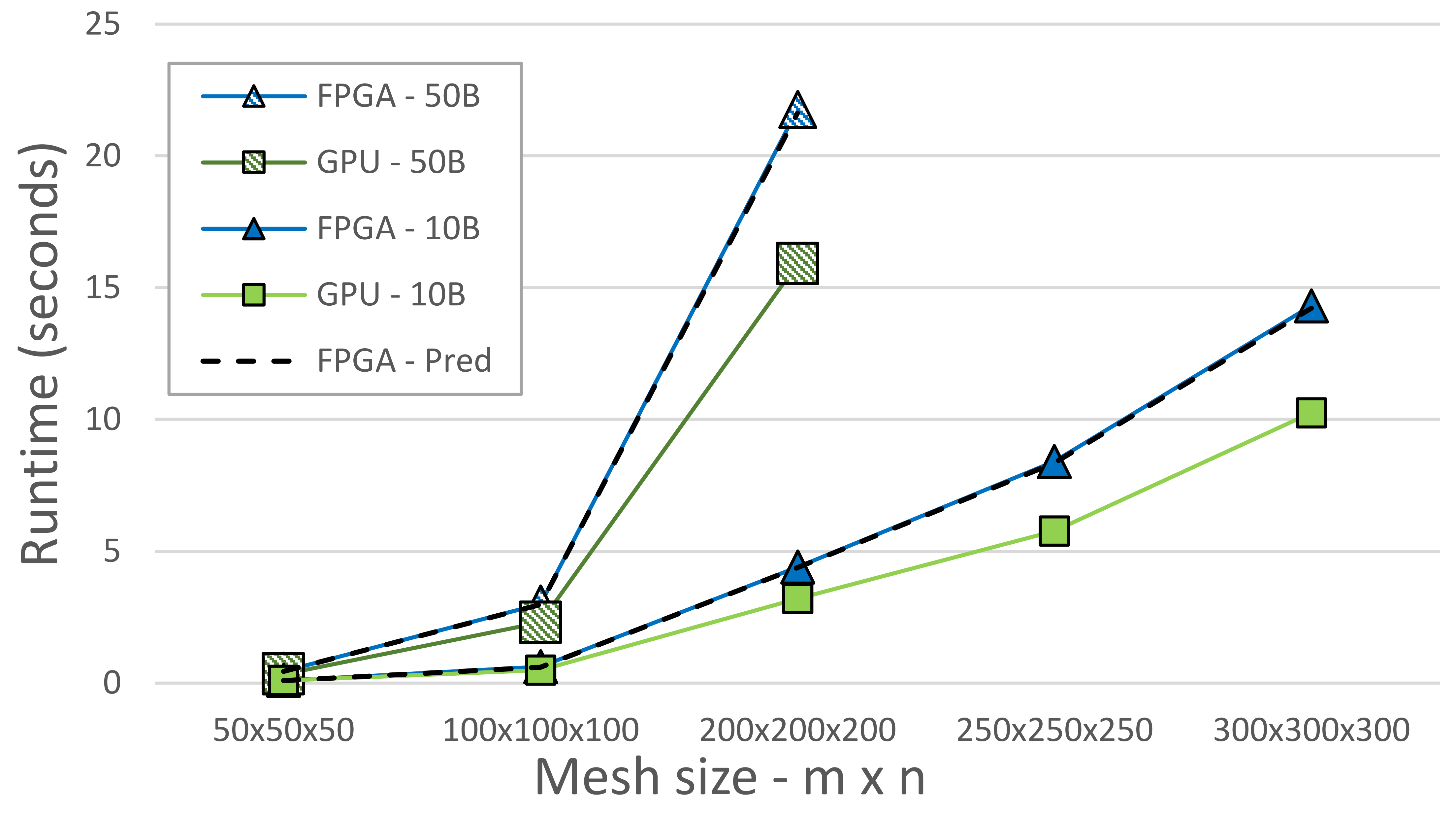


Results – Jacobi

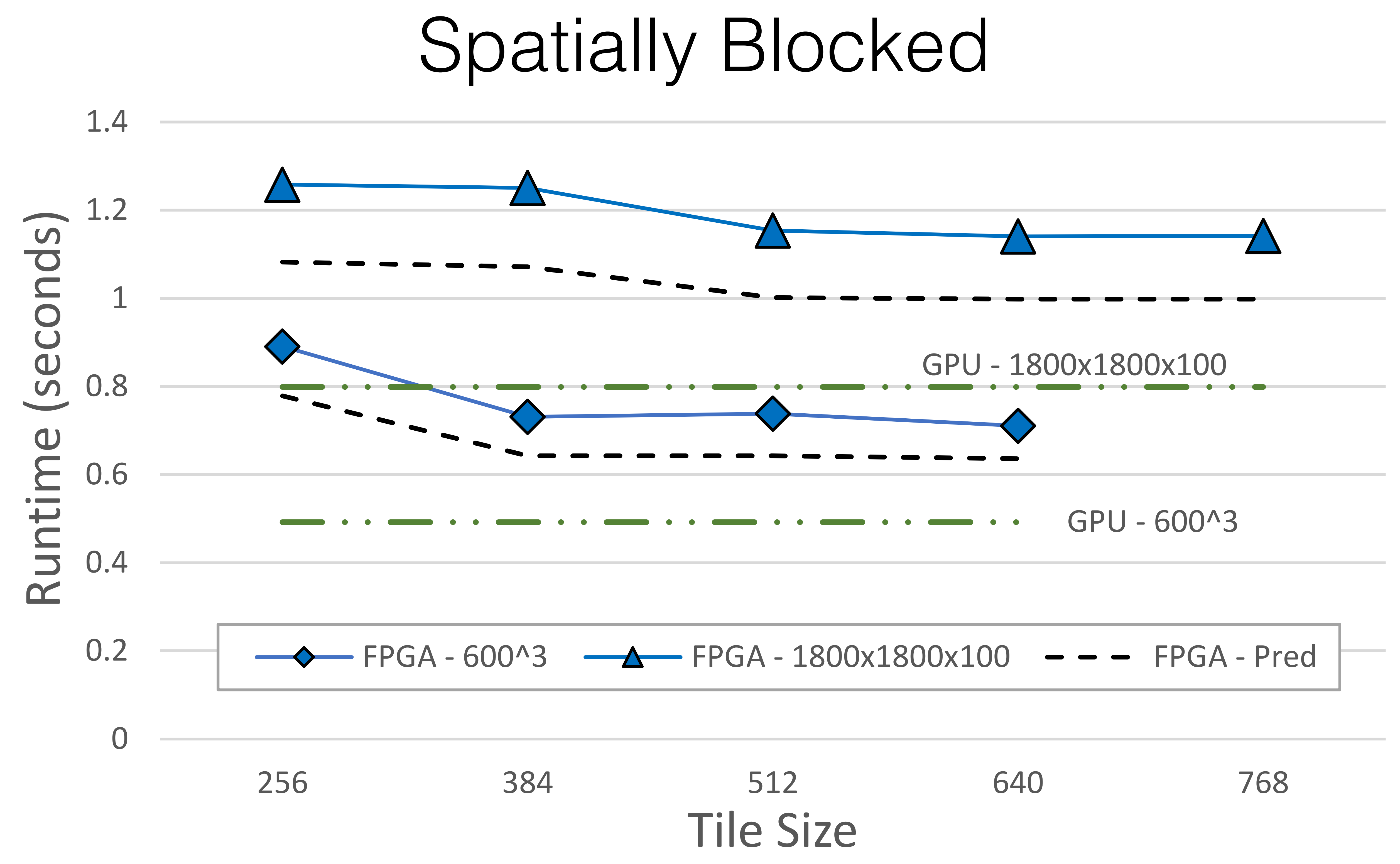
Baseline

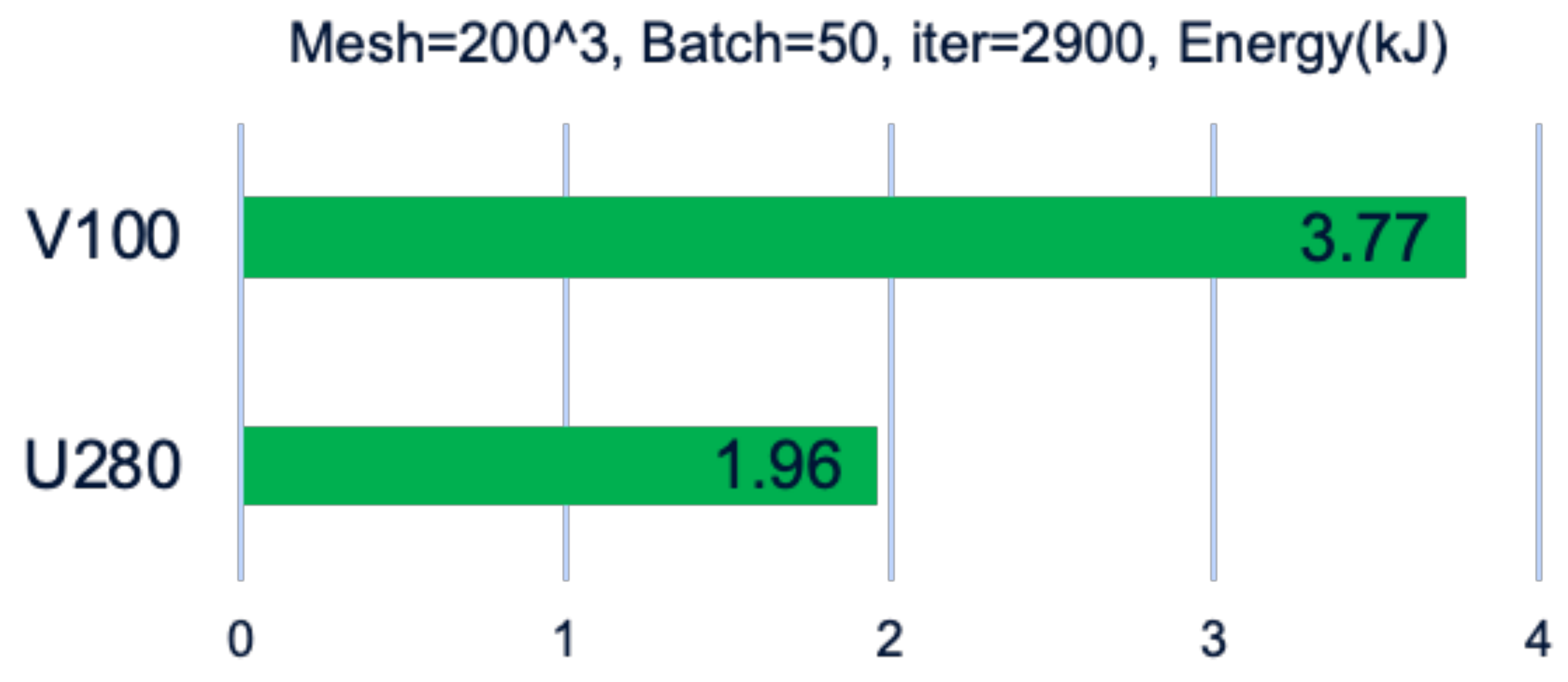


Batched



Results – Jacobi





More results for RTM in our IPDPS 2021 paper + open source release

Kamalakkannan, Mudalige, Reguly, and Fahmy,
High-Level FPGA Accelerator Design for Structured-Mesh-Based Explicit Numerical Solvers, IPDPS 2021

Tridiagonal Solvers

- Many HPC applications rely on solving systems of PDEs
  - Computational fluid dynamics
  - Computational finance

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

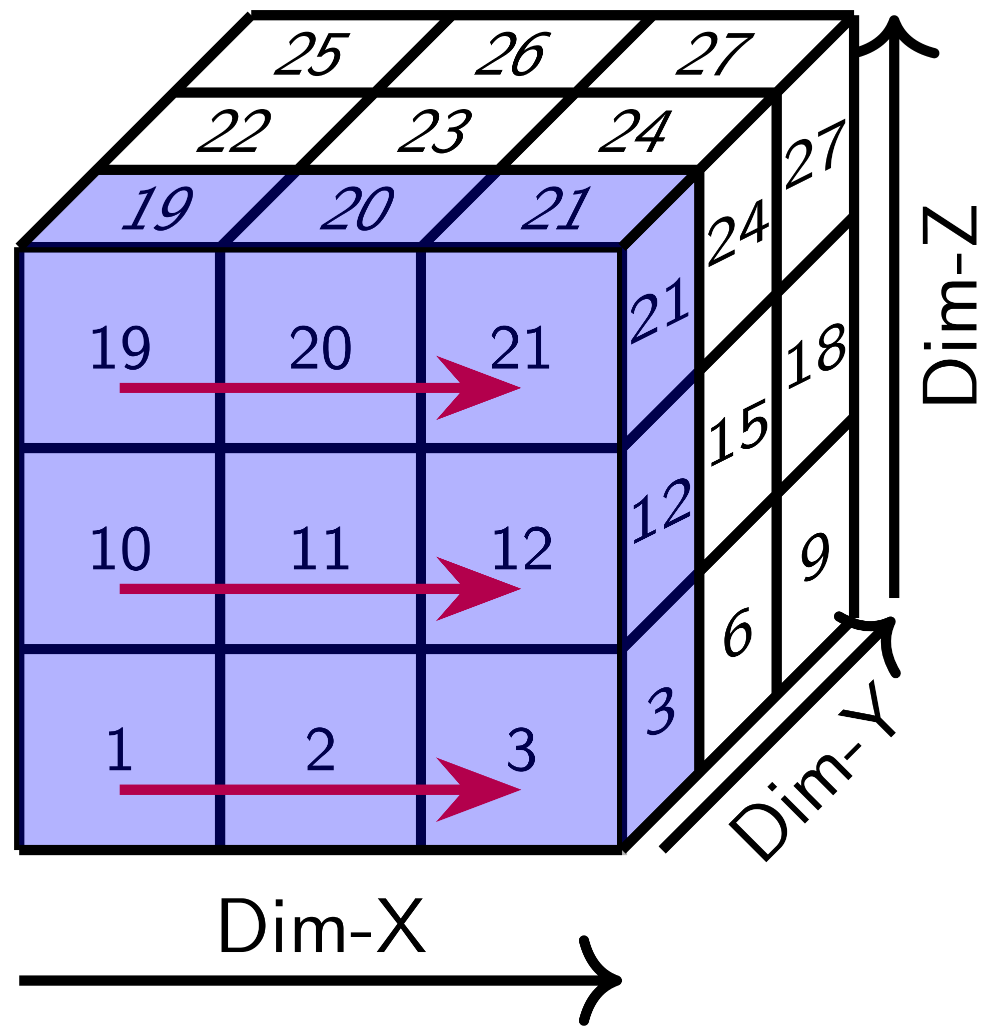
- Many methods use tridiagonal system solver kernels, e.g. Alternating Direction Implicit , Multigrain
- ADI breaks time steps and solves simpler tridiagonal matrices

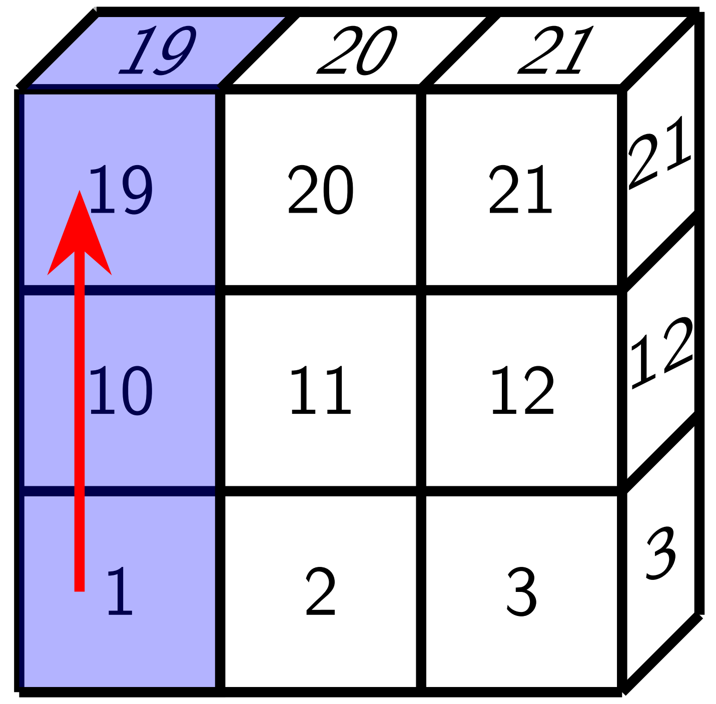
$$\begin{pmatrix} x & x & & \\ x & x & x & \\ & x & x & x \\ & & x & x \end{pmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

- Many libraries on CPU and GPU

On-chip memory flexibility

- Consider an application where you are required to solve in the z-dimension
- We can read XZ planes from DRAM and cache on-chip
- Read the z-lines from this on-chip buffer





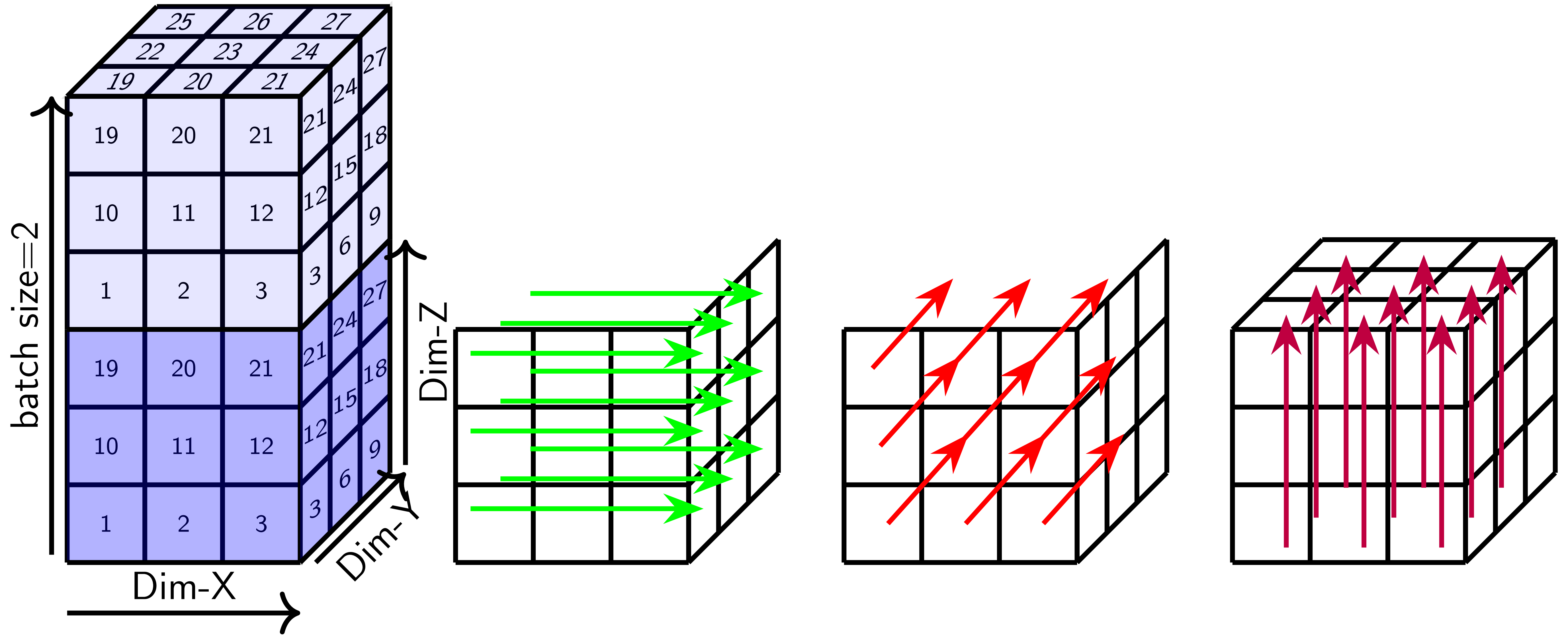
Multi-dimensional tridiagonal solvers

- A is banded matrix and b values are non-zero
- Solve for the unknowns u in multiple dimensions iteratively
- Popular iterative solvers: Thomas, PCR, SPIKE

$$Au = d$$

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d_i, \quad i = 0, 1, \ldots, N-1$$

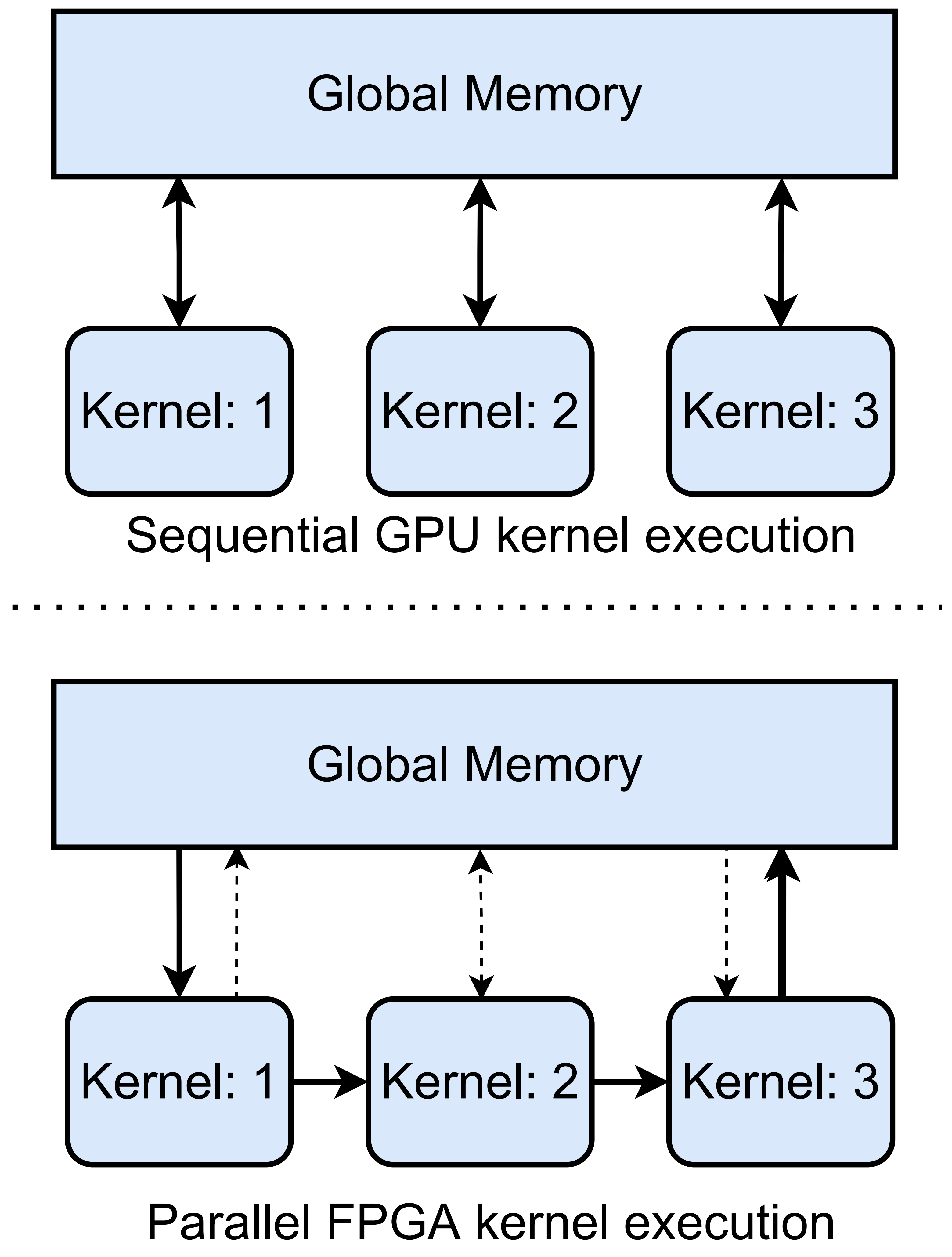
$$\begin{bmatrix} b_0 & c_0 & 0 & \cdots & 0 \\ a_1 & b_1 & c_1 & \cdots & 0 \\ 0 & a_2 & b_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{N-1} & b_{N-1} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{N-1} \end{bmatrix}$$



(a) Batched mesh (b) X-solve (c) Y-solve (d) Z-solve

FPGAs for iterative kernels

- Individual GPU kernels receive and push data to global memory
- Fusing, pipelining, or unrolling of iterative loops require global memory synchronisation which degrades performance
- FPGAs enable kernel to kernel communication and allow for on-chip data reuse



Approach

- Re-evaluated tridiagonal solver algorithms on FPGAs considering problem size, dimensionality, number of systems, and compute data path
- Created a new FPGA tridiagonal solver library for batched multi-dimensional solves, exploiting High Bandwidth Memory
- Created an analytical performance model to aid design space exploration, achieving 85% accuracy
- Implementing two non-trivial applications and compared against GPU

Solver algorithms

- Thomas solver has O(N) complexity but has serial dependencies within loops and between iterations
- PCR has O(N log N) complexity but inner loop can be partitioned and parallelised

Algorithm 1: `thomas`(a, b, c, d, u)

```
1: $d_0^* \leftarrow d_0/b_0$
2: $c_0^* \leftarrow c_0/b_0$
3: for $i = 1, 2, ..., N-1$ do
4:     $r \leftarrow 1/(b_i - a_i c_{i-1}^*)$
5:     $d_i^* \leftarrow r(d_i - a_i d_{i-1}^*)$
6:     $c_i^* \leftarrow r c_i$
7: end for
8: $u_{N-1} \leftarrow d_{N-1}$
9: for $i = N-2, ..., 1, 0$ do
10:     $u_i \leftarrow d_i^* - c_i^* u_{i+1}$
11: end for
12: return $u$
```

Algorithm 2: `pcr`(a, b, c, d, u)

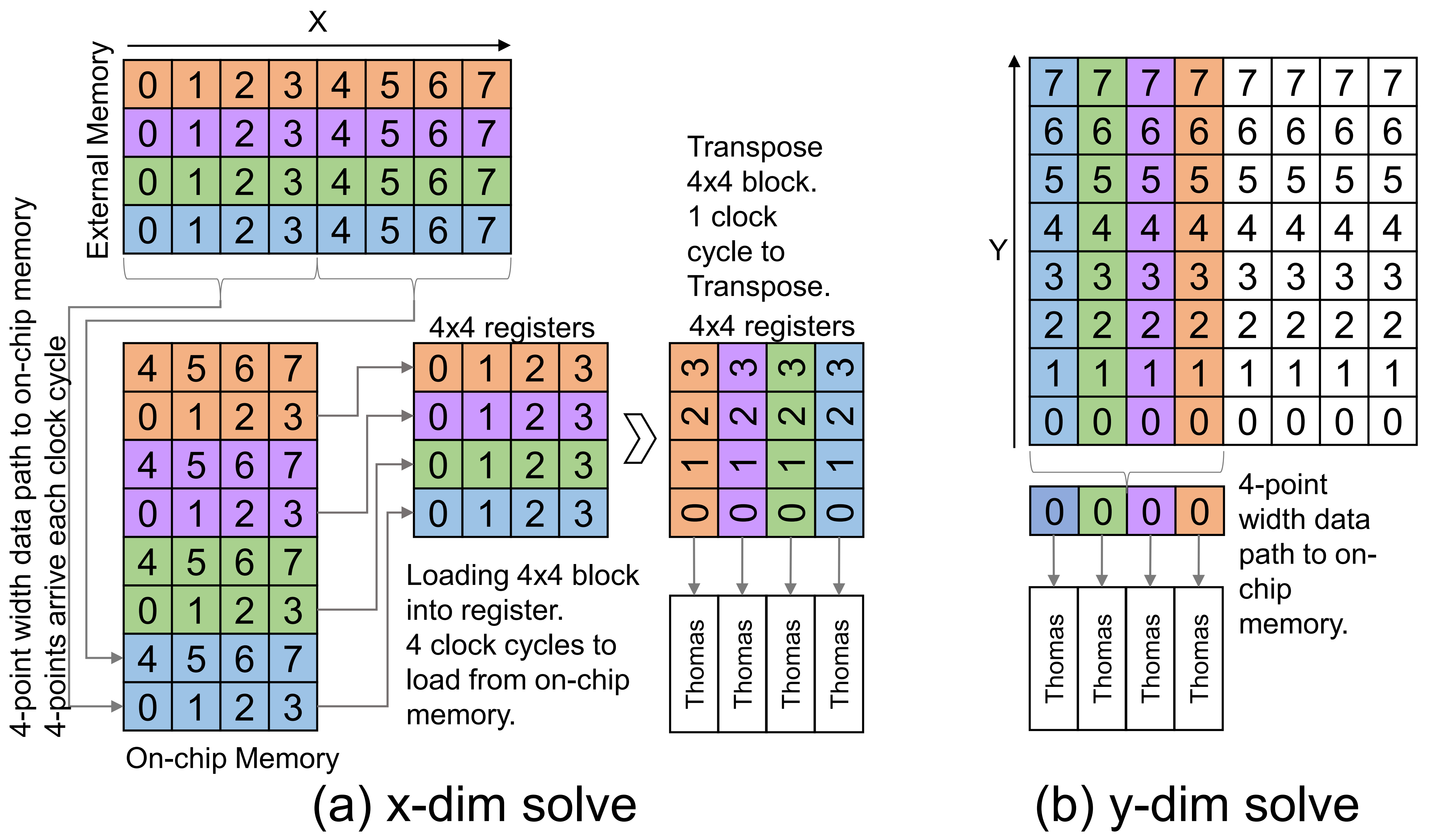
```
1: for $p = 1, 2, ..., P$ do
2:     $s \leftarrow 2^{p-1}$
3:     for $i = 0, 1, ..., N-1$ do
4:         $r \leftarrow 1/(1 - a_i^{(p-1)} c_{i-s}^{(p-1)} - c_i^{(p-1)} a_{i+s}^{(p-1)})$
5:         $a_i^{(p)} \leftarrow -r(a_i^{(p-1)} a_{i-s}^{(p-1)})$
6:         $c_i^{(p)} \leftarrow -r(c_i^{(p-1)} c_{i+s}^{(p-1)})$
7:         $d_i^{(p)} \leftarrow$
           $r(d_i^{(p-1)} - a_i^{(p-1)} d_{i-s}^{(p-1)} - c_i^{(p-1)} d_{i+s}^{(p-1)})$
8:     end for
9: end for
10: $u \leftarrow d^{(P)}$
11: return $u$
```

Optimising Thomas on FPGA

- Inter iteration loop dependency: single iteration loop latency means pipeline is idle for some time → group multiple systems and interleave their solves
    - On-chip memory enables interleaving
- Forward loop and backward loop cannot operate in parallel → apply forward and backward to different groups
    - Double buffers for continuous data movement
- Thomas is then more efficient than PCR due to the lower resource usage

Parallelising Thomas

- Parallel Thomas solvers each solve different systems
- However, when solving in the x-dimension, the parallel solvers require non-coalesced memory access
- Instead, a block transpose can be done on-chip, which is highly efficient


X
External Memory
4-point width data path to on-chip memory
4-points arrive each clock cycle
On-chip Memory
4x4 registers
Loading 4x4 block into register. 4 clock cycles to load from on-chip memory.
Transpose 4x4 block. 1 clock cycle to Transpose.
4x4 registers
Thomas
(a) x-dim solve
Y
4-point width data path to on-chip memory.
Thomas
(b) y-dim solve

Tiling for large systems

- Interleaving for larger systems can consume significant on-chip memory
- Instead, the systems can be solved in tiles, each solved interleaved with others
- Back substitution used to produce final result

$$\left[\begin{array}{cccc|cccc} b_0 & c_0 & & & & & & \\ a_1 & b_1 & c_1 & & & & & \\ & a_2 & b_2 & c_2 & & & & \\ & & a_3 & b_3 & c_3 & & & \\ \hline & & & a_4 & b_4 & c_4 & & \\ & & & & a_5 & b_5 & c_5 & \\ & & & & & a_6 & b_6 & c_6 \\ & & & & & & a_7 & b_7 \end{array}\right] \left[\begin{array}{c} u_0 \\ u_1 \\ u_2 \\ u_3 \\ \hline u_4 \\ u_5 \\ u_6 \\ u_7 \end{array}\right] = \left[\begin{array}{c} d_0 \\ d_1 \\ d_2 \\ d_3 \\ \hline d_4 \\ d_5 \\ d_6 \\ d_7 \end{array}\right]$$

$$\left[\begin{array}{cccc|cccc} 1 & c_0 & & & & & & \\ a_1^* & 1 & c_1 & & & & & \\ a_2^* & & 1 & c_2 & & & & \\ a_3^* & & & 1 & c_3 & & & \\ \hline & & & a_4^* & 1 & c_4 & & \\ & & & & a_5^* & 1 & c_5 & \\ & & & & a_6^* & & 1 & c_6 \\ & & & & a_7^* & & & 1 \end{array}\right] \left[\begin{array}{c} u_0 \\ u_1 \\ u_2 \\ u_3 \\ \hline u_4 \\ u_5 \\ u_6 \\ u_7 \end{array}\right] = \left[\begin{array}{c} d_0^* \\ d_1^* \\ d_2^* \\ d_3^* \\ \hline d_4^* \\ d_5^* \\ d_6^* \\ d_7^* \end{array}\right]$$

$$\left[\begin{array}{cccc|cccc} 1 & & & c_0^* & & & & \\ a_1^* & 1 & & c_1^* & & & & \\ a_2^* & & 1 & c_2^* & & & & \\ a_3^* & & & 1 & c_3^* & & & \\ \hline & & & a_4^* & 1 & & & c_4^* \\ & & & & a_5^* & 1 & & c_5^* \\ & & & & a_6^* & & 1 & c_6^* \\ & & & & a_7^* & & & 1 \end{array}\right] \left[\begin{array}{c} u_0 \\ u_1 \\ u_2 \\ u_3 \\ \hline u_4 \\ u_5 \\ u_6 \\ u_7 \end{array}\right] = \left[\begin{array}{c} d_0^* \\ d_1^* \\ d_2^* \\ d_3^* \\ \hline d_4^* \\ d_5^* \\ d_6^* \\ d_7^* \end{array}\right]$$

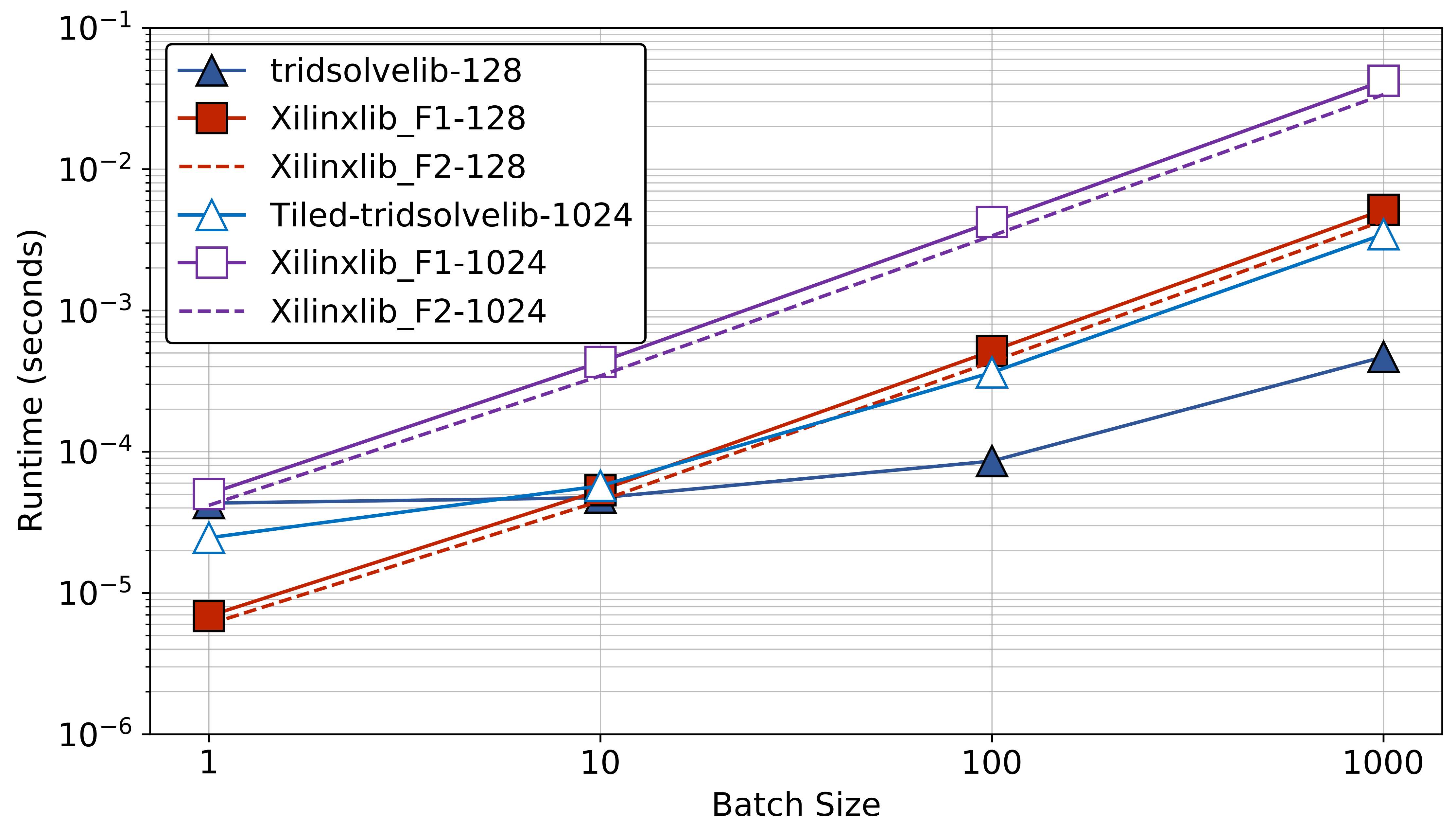
Experimental setup

- Used comparable FPGA and GPU platforms
  - Nvidia V100 is 12nm with 900GB/s global memory bandwidth
  - Xilinx U280 is 16nm with 460GB/s global memory bandwidth

FPGA	Xilinx Alveo U280
DSP blocks	8490
BRAM/URAM	6.6MB (1487 blocks)/34.5MB (960 blocks)
HBM	8GB, 460GB/s, 32 channels
DDR4	32GB, 38.4GB/s, in 2 banks
Host	AMD Ryzen Threadripper PRO 3975WX (32 cores) 512GB RAM, Ubuntu 18.04.6 LTS
Design SW	Xilinx Vivado HLS, Vitis 2019.2
Run-Time	Xilinx XRT 202020.2.9.317
GPU	**Nvidia Tesla V100 PCIe**
Global Mem.	16GB HBM2, 900GB/s
Host	Intel Xeon Gold 6252 @2.10GHz (48 cores) 256GB RAM, Ubuntu 18.04.3 LTS
Compilers, OS	nvcc CUDA 10.0.130, Debian 9.11

Xilinx library comparison

- Xilinx provide a PCR-based library for batched solves
- Compared for different batch sizes with 128 and 1024 systems
- Order of magnitude improved performance for larger batch sizes
- (F2 includes unrolling the inner loop of PCR for the Xilinx library)

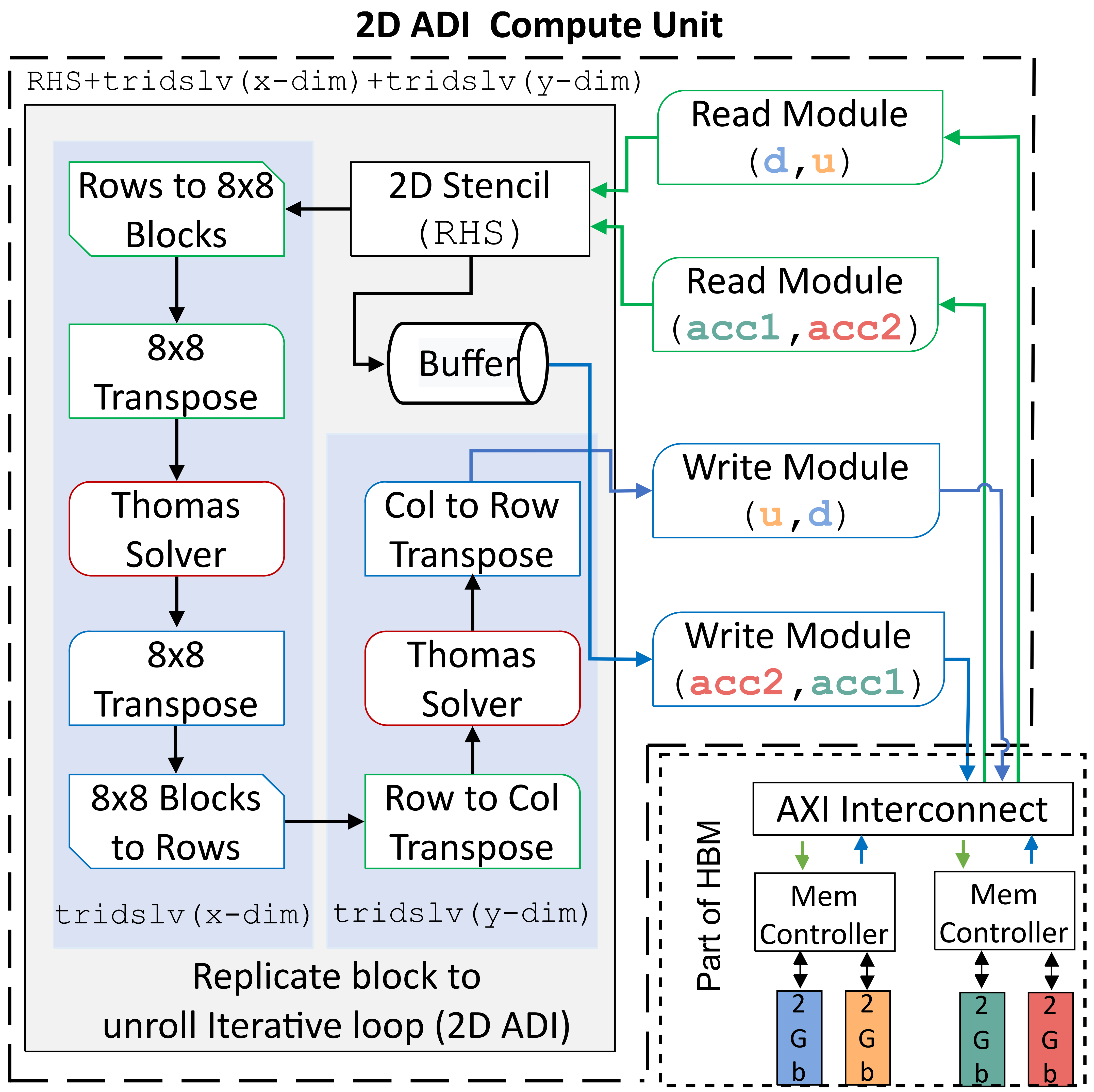


Complex applications

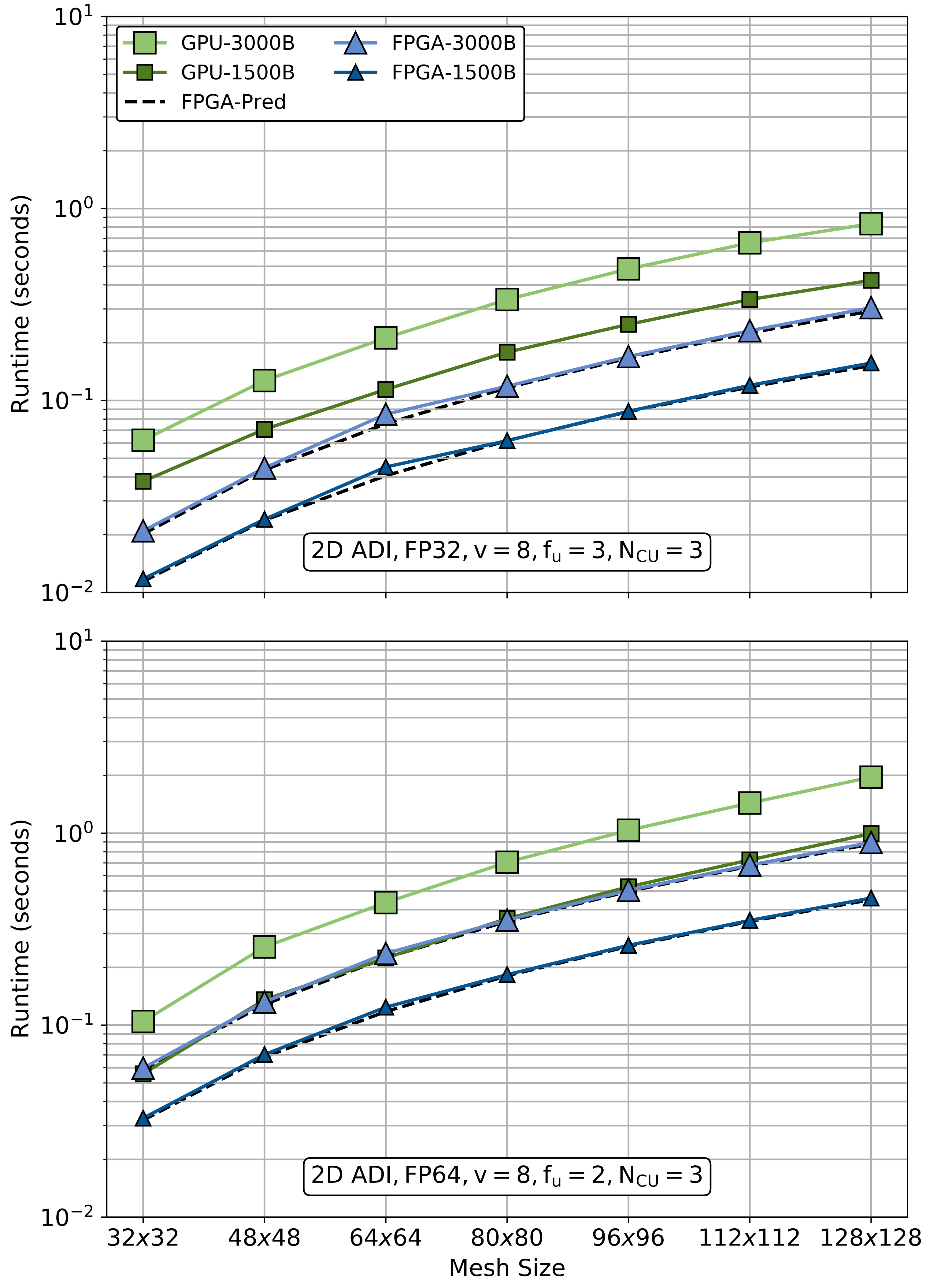
- Alternating Direction Implicit Heat Diffusion
  - 2D and 3D, FP32 and FP64
- 2D Stochastic Local Volatility
  - 2D FP64

Alternating Direction Implicit

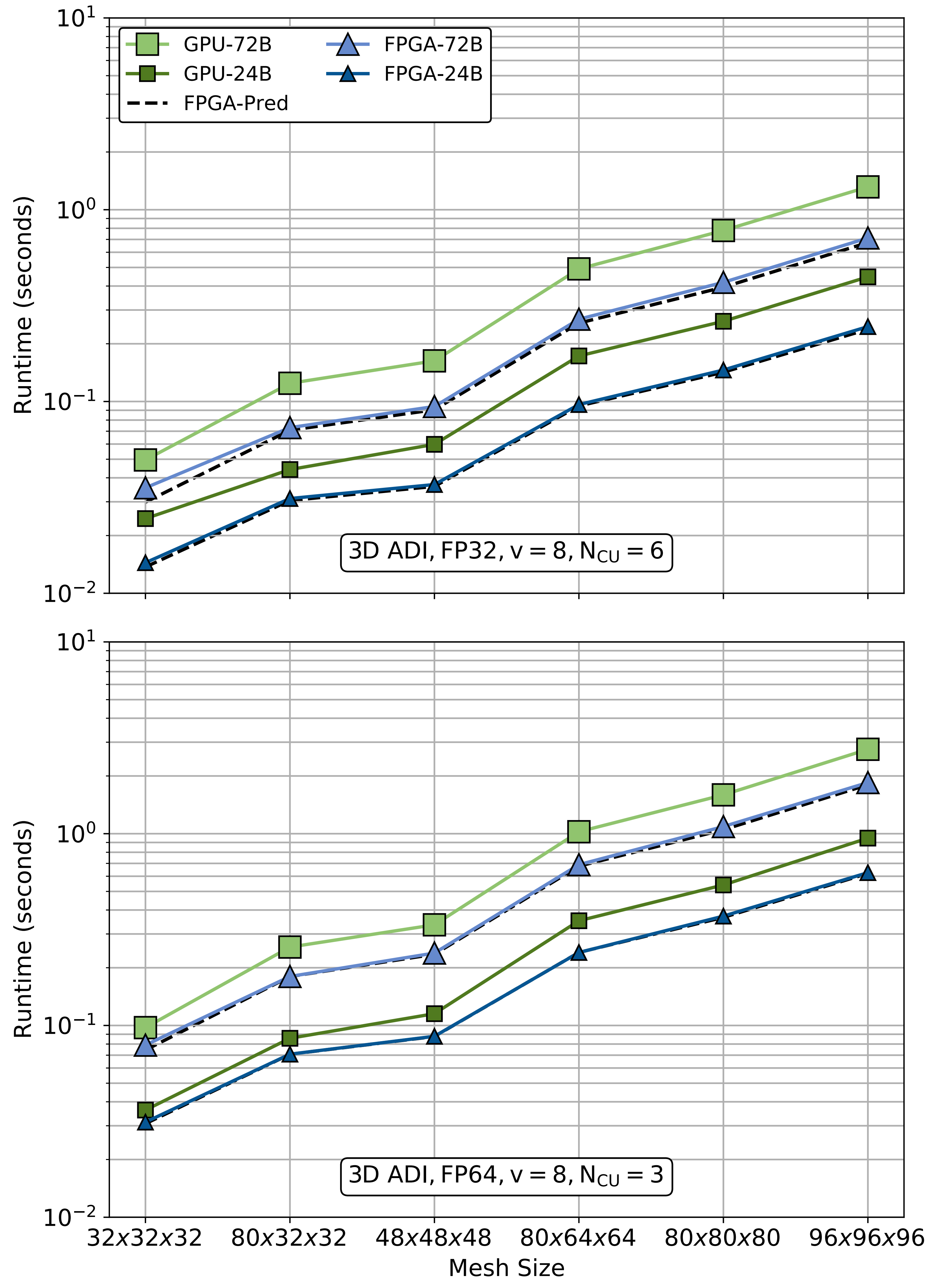
- For 2D application, possible to fully pipeline kernels
- Iterative loop can be unrolled
- To create the delay buffer, spare HBM interfaces are used
- Results in high bandwidth and pipeline is better utilised



ADI heat diffusion results



(a) 2D ADI: 120 iter.



(b) 3D ADI: 100 iter.

ADI heat diffusion results

2D FP32 (120 iterations, $f_U = 3$)

Batch Size	1500					3000				
	Bandwidth			Energy		Bandwidth			Energy	
Mesh	F	Gx	Gy	F	G	F	Gx	Gy	F	G
32^2	501	375	276	1	5	563	435	377	2	9
64^2	524	428	449	3	16	556	447	512	6	29
128^2	602	416	539	12	60	620	418	554	23	115

2D FP64 (120 iterations, $f_U = 2$)

Batch Size	1500					3000				
	Bandwidth			Energy		Bandwidth			Energy	
Mesh	F	Gx	Gy	F	G	F	Gx	Gy	F	G
32^2	360	396	501	2	7	395	441	535	4	14
64^2	380	401	492	9	30	399	417	506	18	61
128^2	411	286	512	34	141	422	281	548	67	284

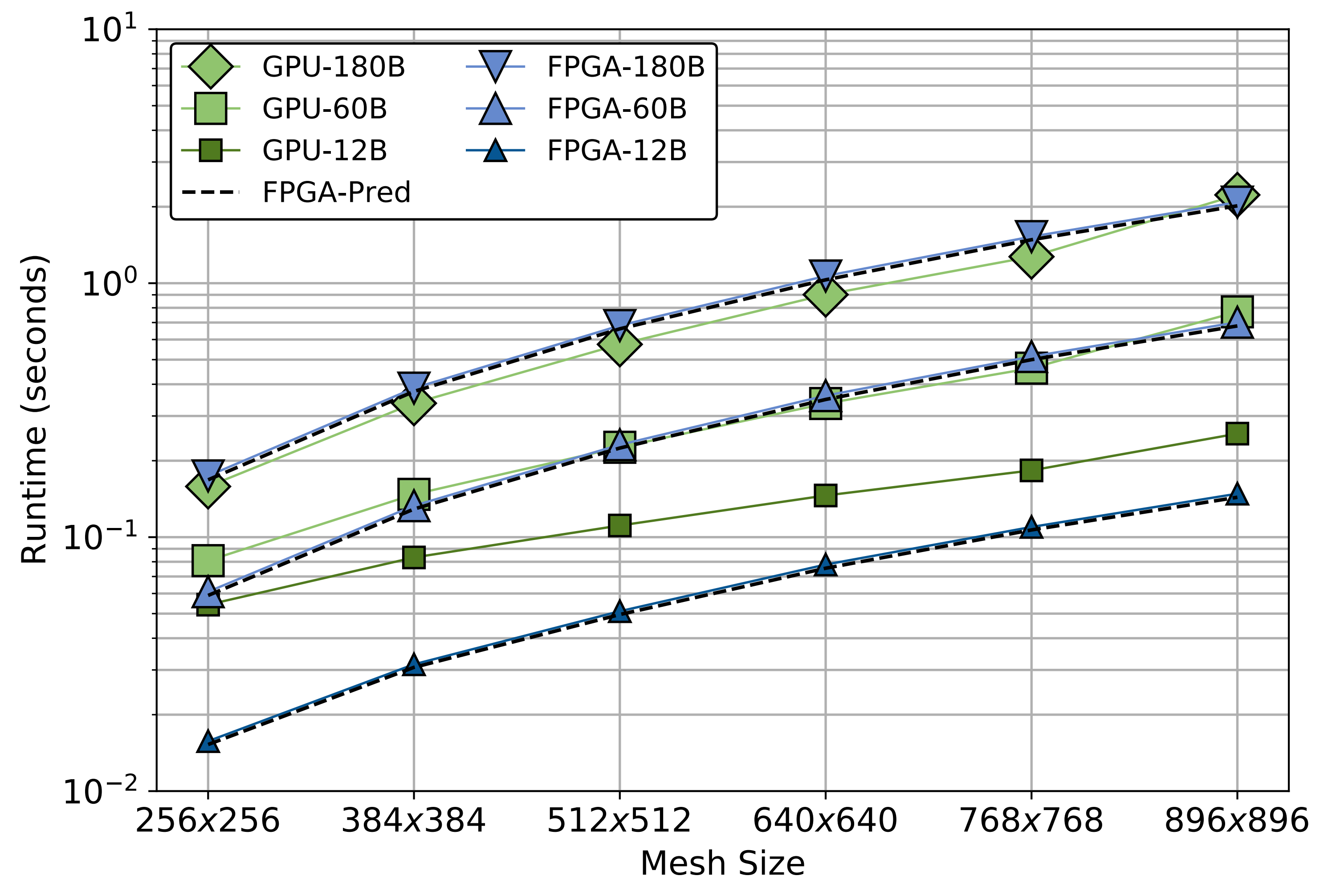
3D FP32 (100 iterations)

Batch Size	24						72					
	Bandwidth				Energy		Bandwidth				Energy	
Mesh	F	Gx	Gy	Gz	F	G	F	Gx	Gy	Gz	F	G
$32 \times 32 \times 32$	218	380	229	283	1	3	266	449	390	537	3	8
$48 \times 48 \times 48$	288	426	354	477	3	9	338	459	408	553	7	25
$96 \times 96 \times 96$	346	401	401	568	18	65	358	415	419	563	53	197

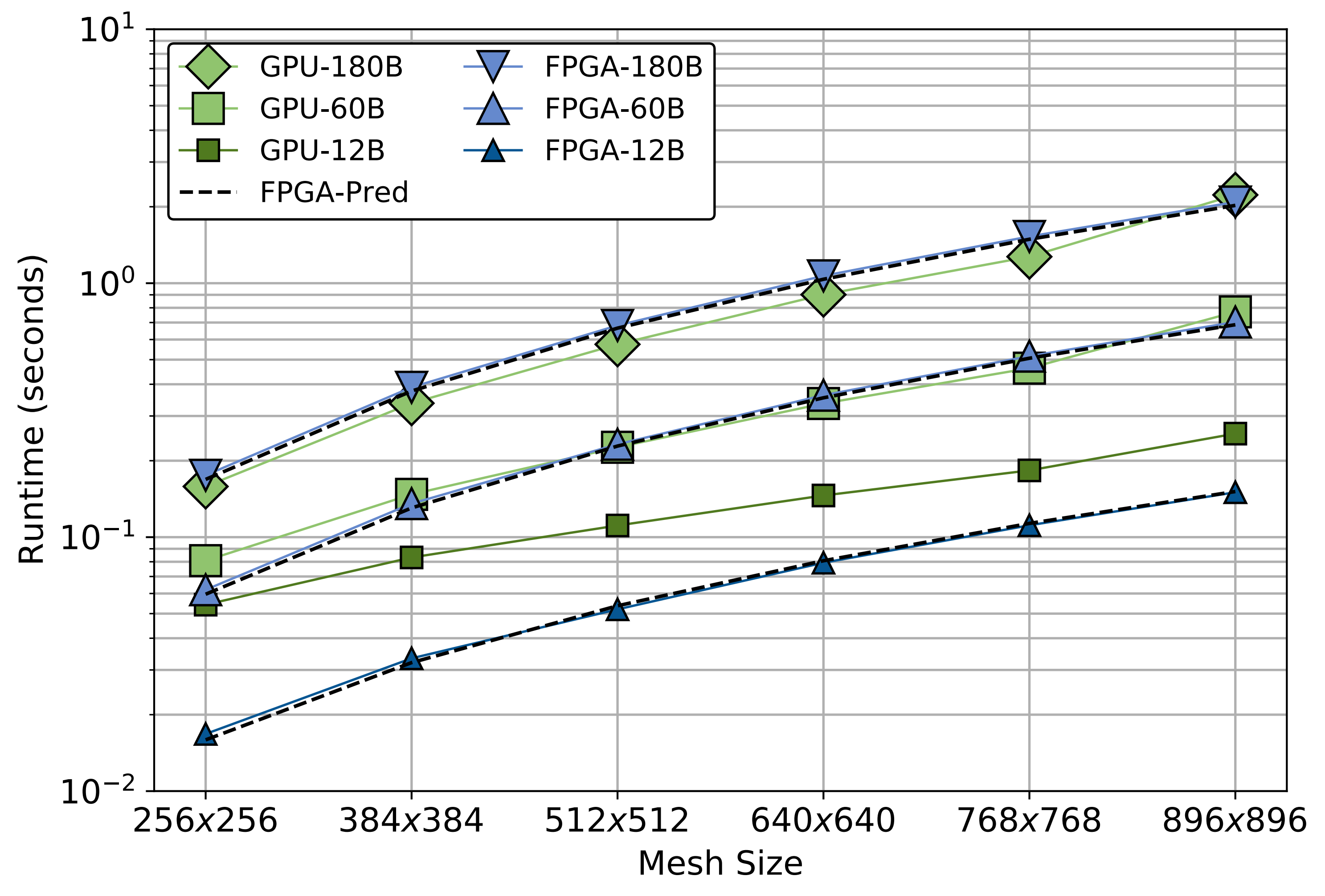
3D FP64 (100 iterations)

Batch Size	24						72					
	Bandwidth				Energy		Bandwidth				Energy	
Mesh	F	Gx	Gy	Gz	F	G	F	Gx	Gy	Gz	F	G
$32 \times 32 \times 32$	201	405	365	445	2	4	239	439	424	527	6	13
$48 \times 48 \times 48$	242	388	407	536	7	16	267	408	421	554	18	48
$96 \times 96 \times 96$	271	324	412	550	47	135	276	338	436	565	139	399

Tiled ADI heat diffusion



(c) 2D ADI-Thomas-PCR: 100 iter.

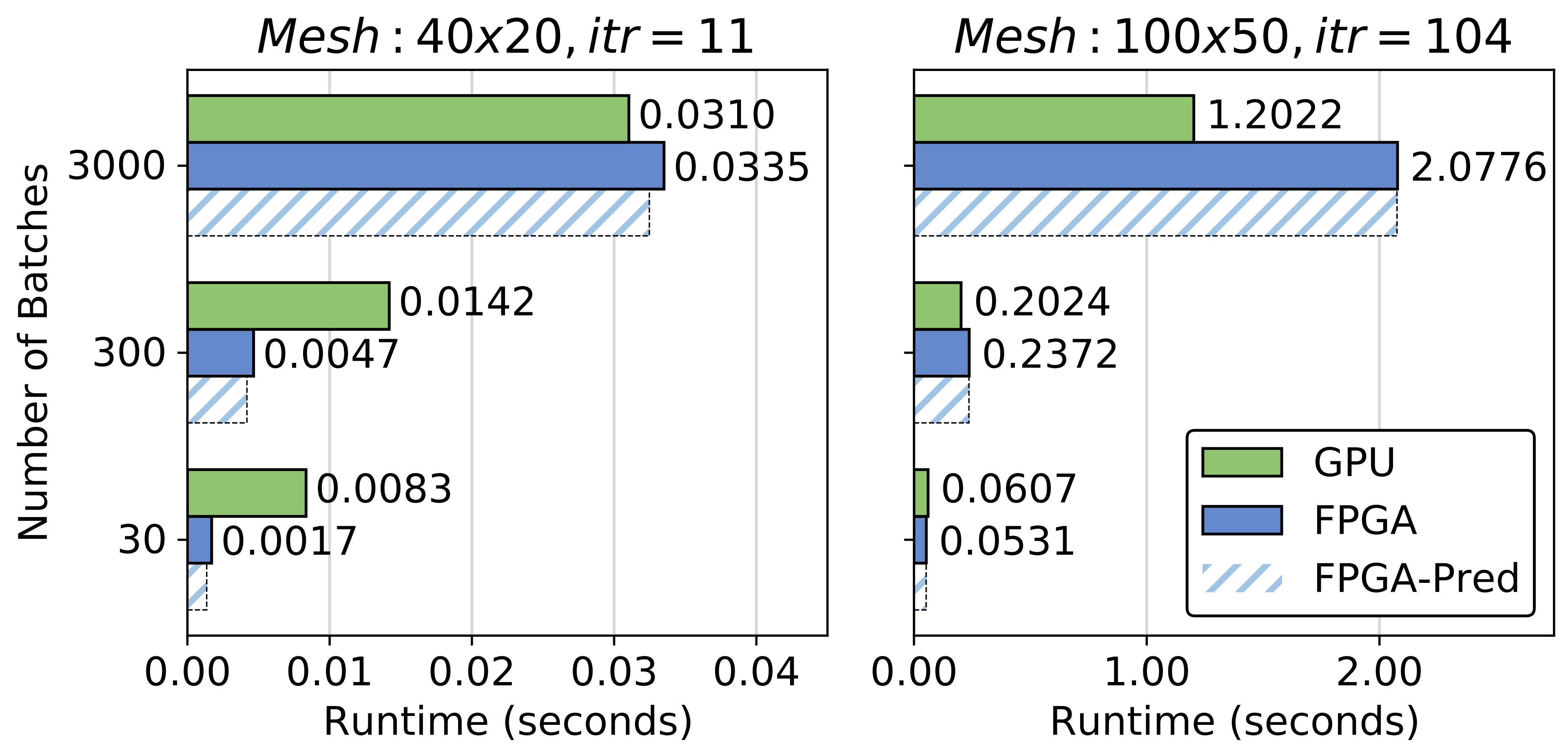


(d) 2D ADI-Thomas-Thomas: 100 iter.

Batch Size	60					180				
	Bandwidth			Energy		Bandwidth			Energy	
Mesh	F	Gx	Gy	F	G	F	Gx	Gy	F	G
256^2	692	213	238	5	8	217	766	437	13	21
512^2	218	768	379	17	28	222	797	534	51	75
896^2	220	345	495	53	104	222	342	562	156	312

Stochastic local volatility application

- 2D computational finance application based on ADI method
- Multiple loops including higher order stencils, complex data structures, and FP64 arithmetic
- Energy is the main win for FPGAs here
- Must consider the bandwidth differences between the two platforms



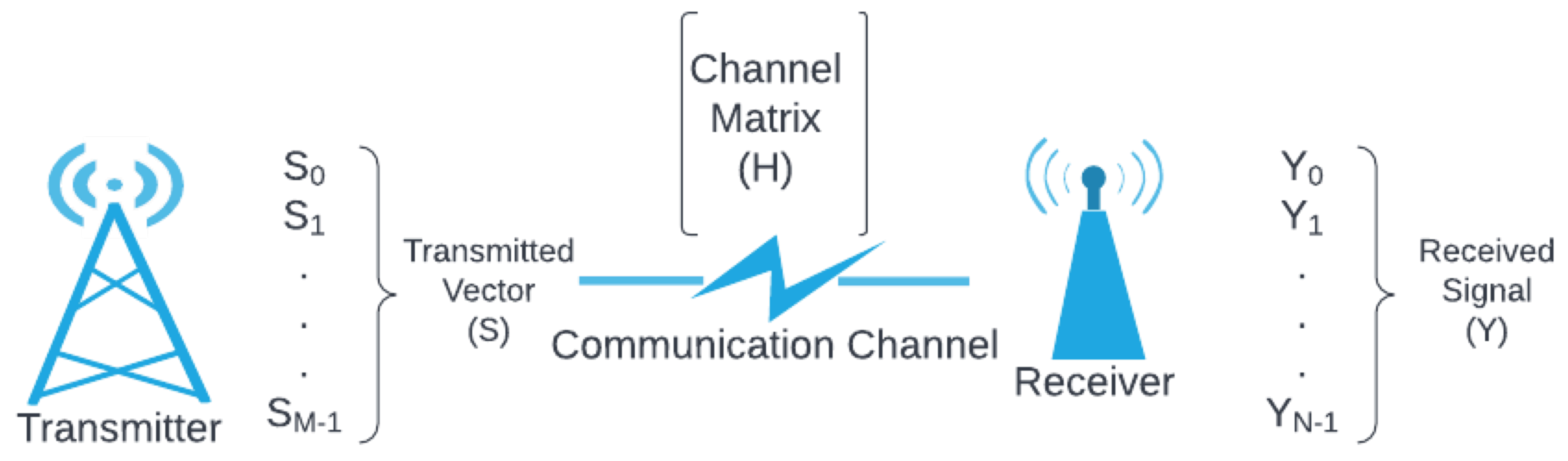
Batch	Bandwidth FPGA	Bandwidth GPU-x	Bandwidth GPU-y	Energy FPGA	Energy GPU
40×20 mesh: 11 iterations					
30	55.24	3.04	28.01	0.13	0.45
300	202.31	16.48	176.51	0.35	1.02
3000	281.06	123.84	327.65	2.51	4.75

More detail and results in our ICS 2022 paper + open source release

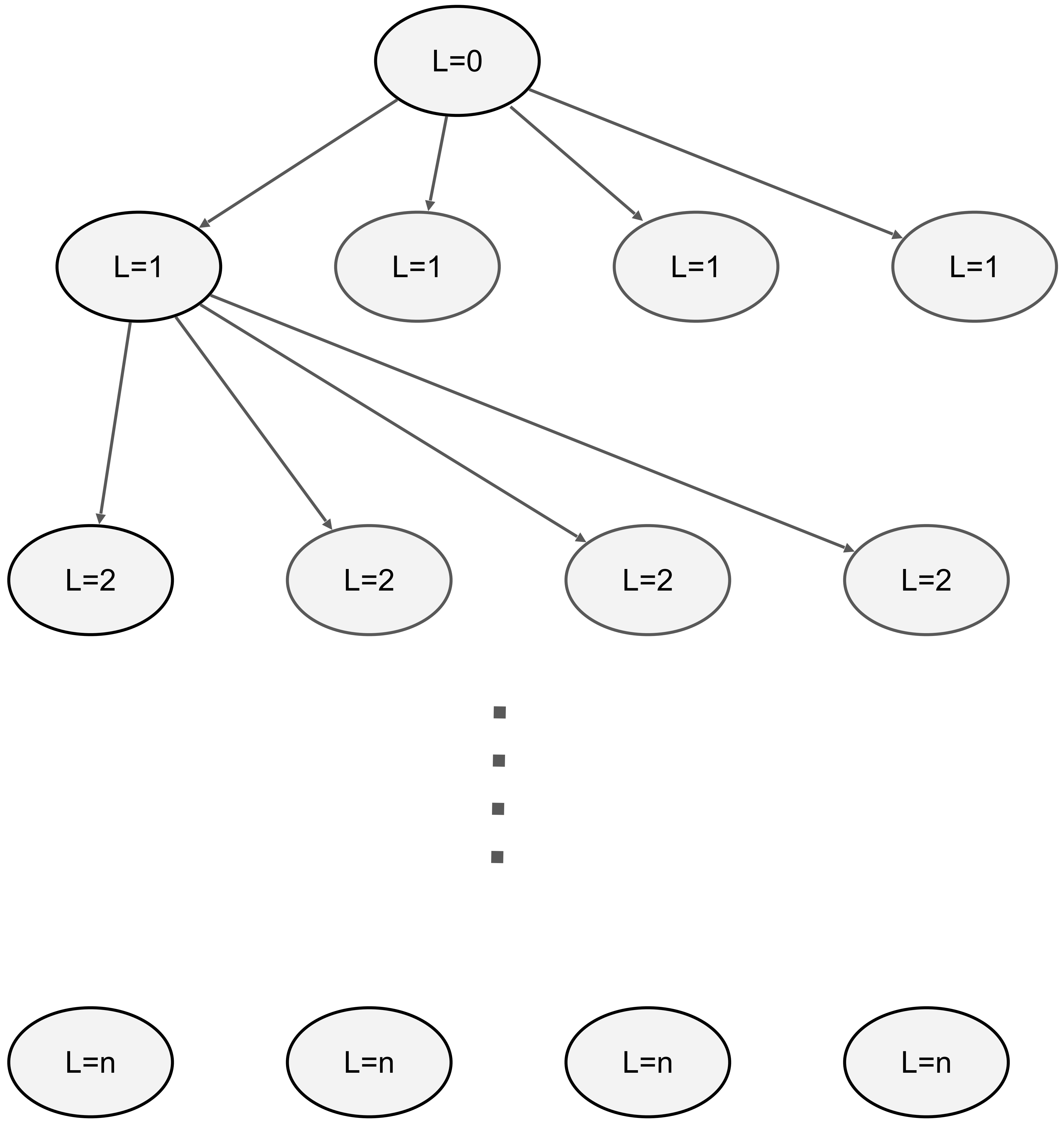
Kamalakkannan, Mudalige, Reguly, and Fahmy,
High Throughput Multidimensional Tridiagonal System Solvers on FPGAs, ICS 2022

Massive MIMO

- Wireless technique with large number of antennas in Multi-Input Multi-Output arrangement
- Increases channel capacity through transmission of multiple sub streams
- High computational complexity

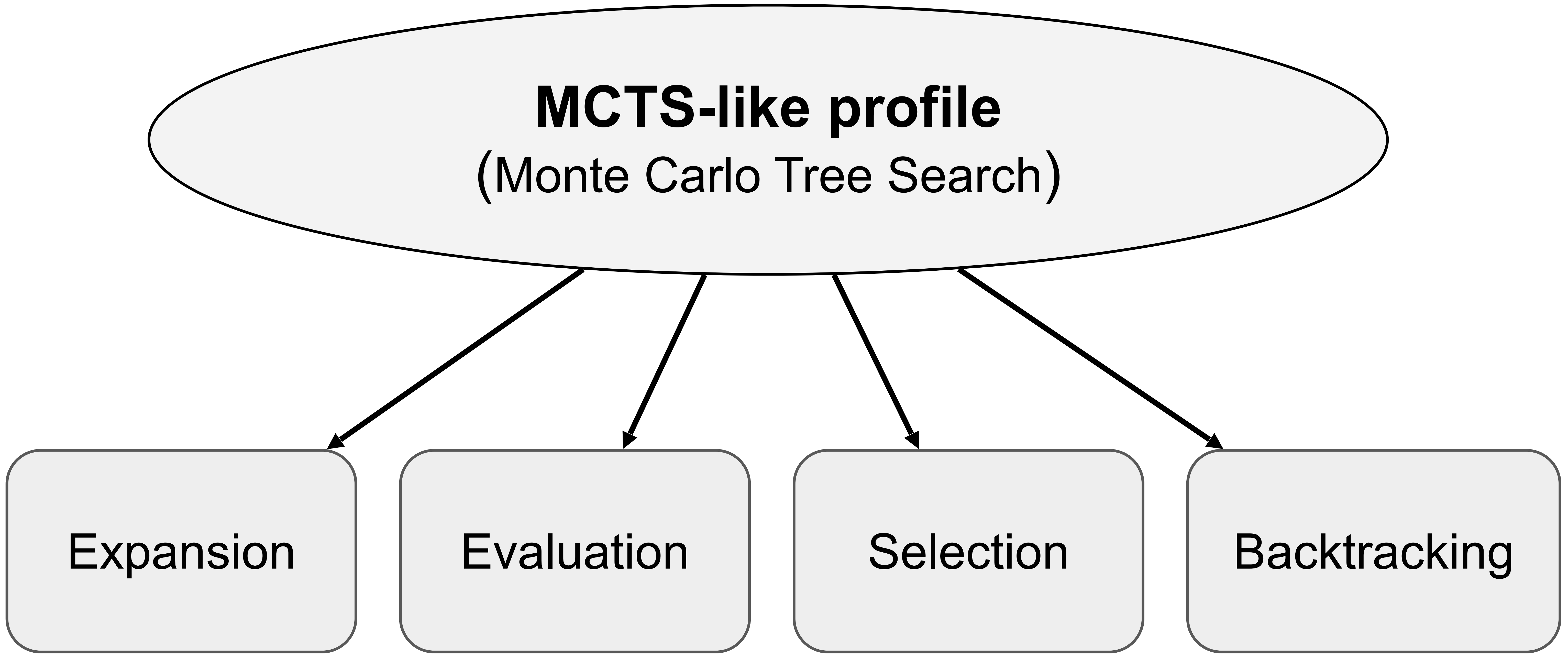


Non-linear Decoders



- Non-linear decoding is a tree search problem
  - Number of children based on modulation factor
- Maximum Likelihood (ML) decoder considered the optimal algorithm– explores whole tree
- Sphere Decoding (SD) is computationally simpler
  - Search space restricted by defining a sphere radius
  - Enumerate candidate solutions inside the sphere
  - Trade-off: *BER (Bit Error Rate)* vs *decoding time*

SD Computational Profile



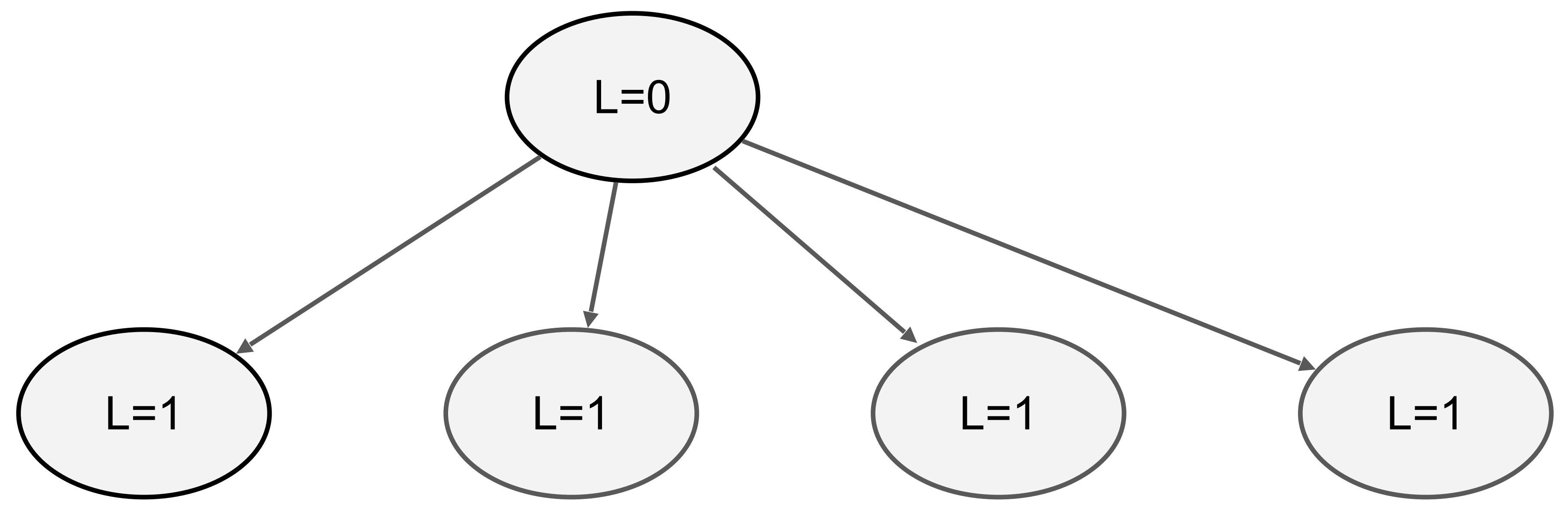
- Inputs
  - Received signal (Y)
  - Channel matrix (H)
  - Radius (r)

$$\| Y - Hs \| \leq r^2$$

Where (s) is the recovered decoded signal

SD Computational Profile

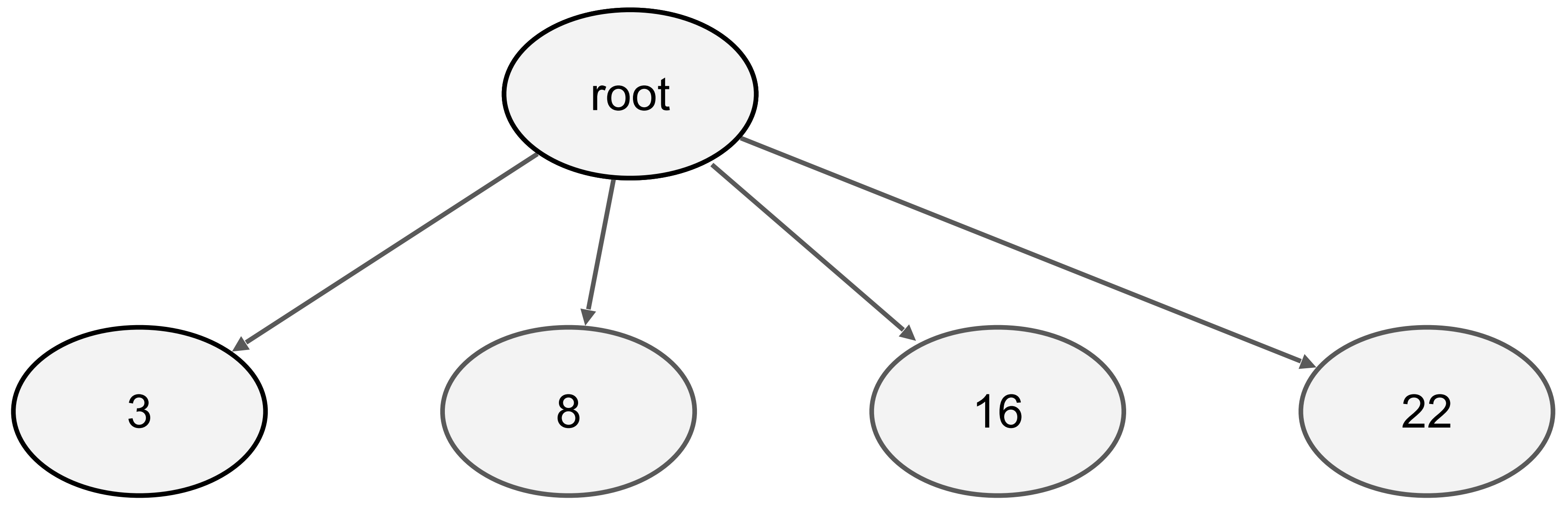
Expansion



- Expand the root node

SD Computational Profile

Evaluation

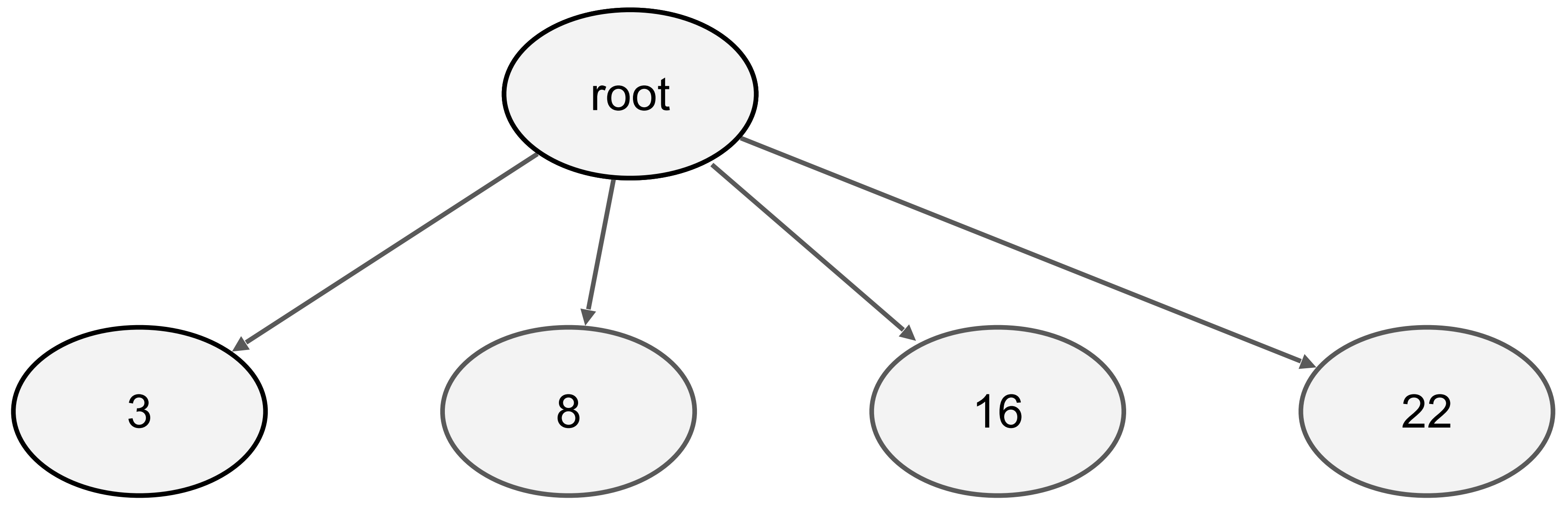


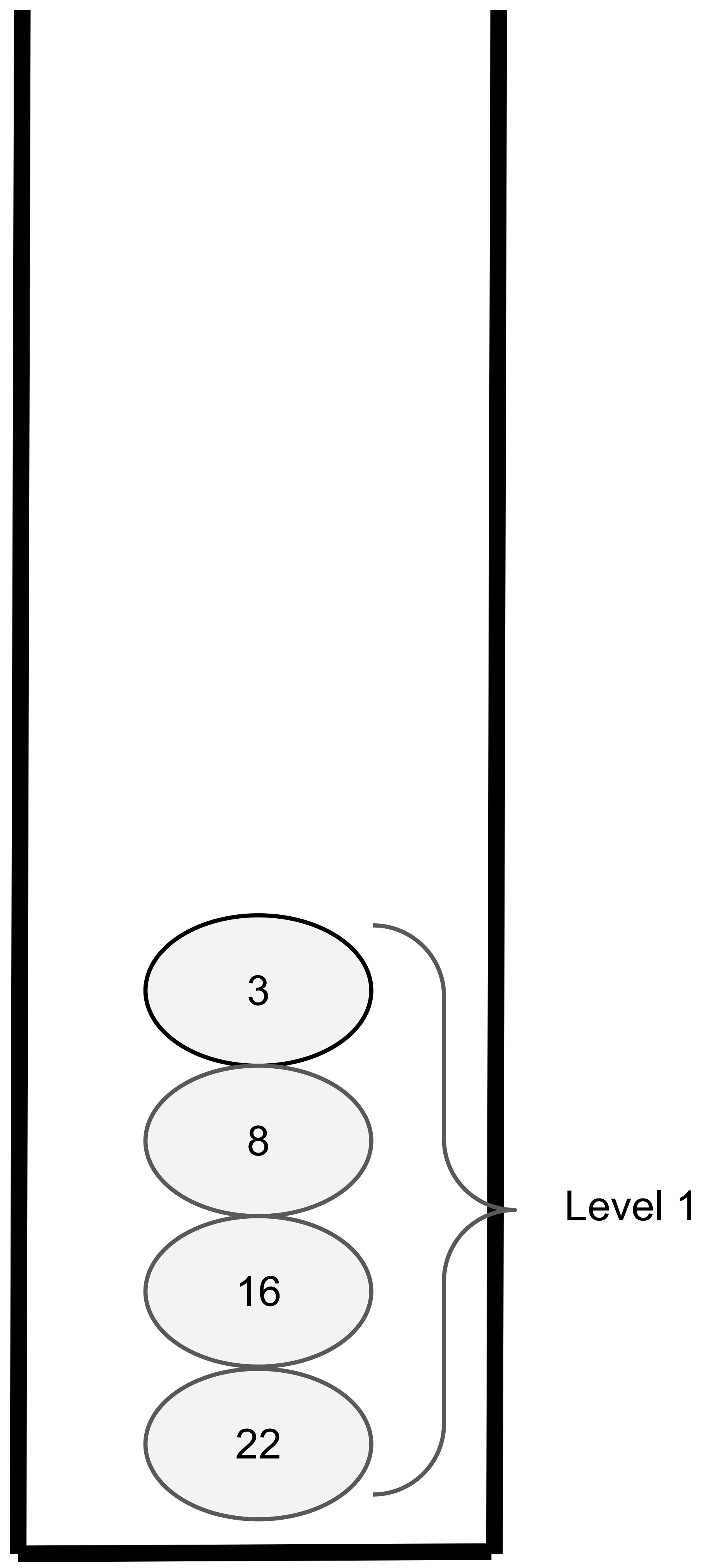
- Expand the root node
- Evaluate PD

(Partial Distance)

SD Computational Profile

Evaluation





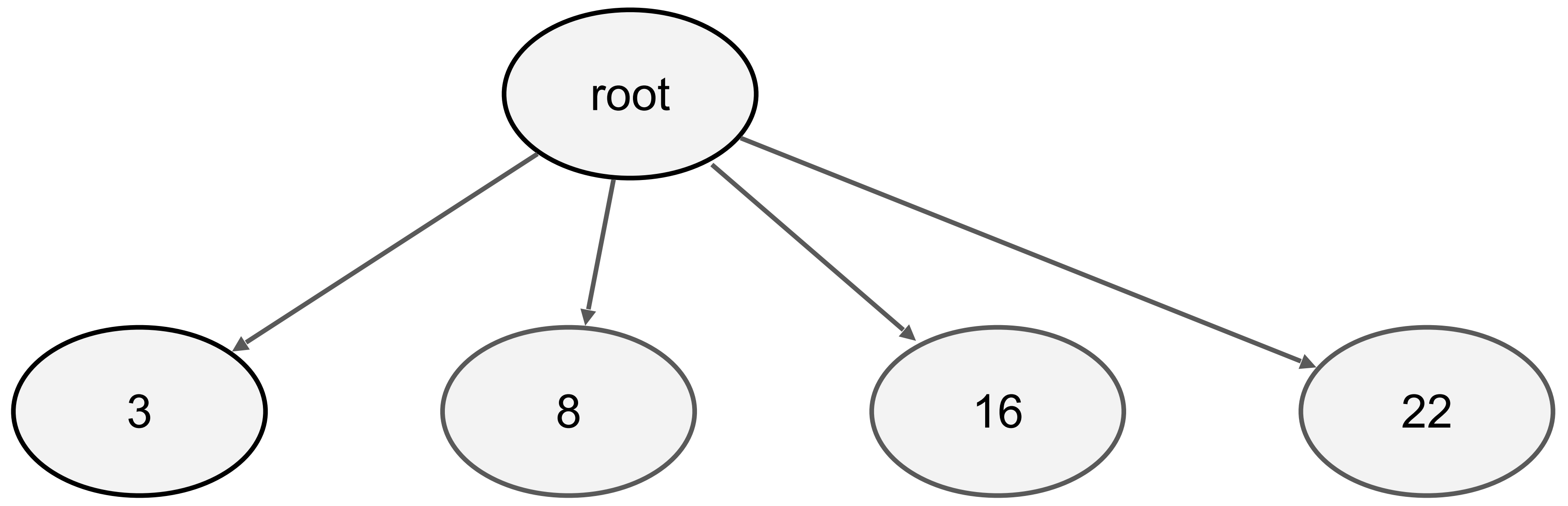
- Expand the root node
- Evaluate PD

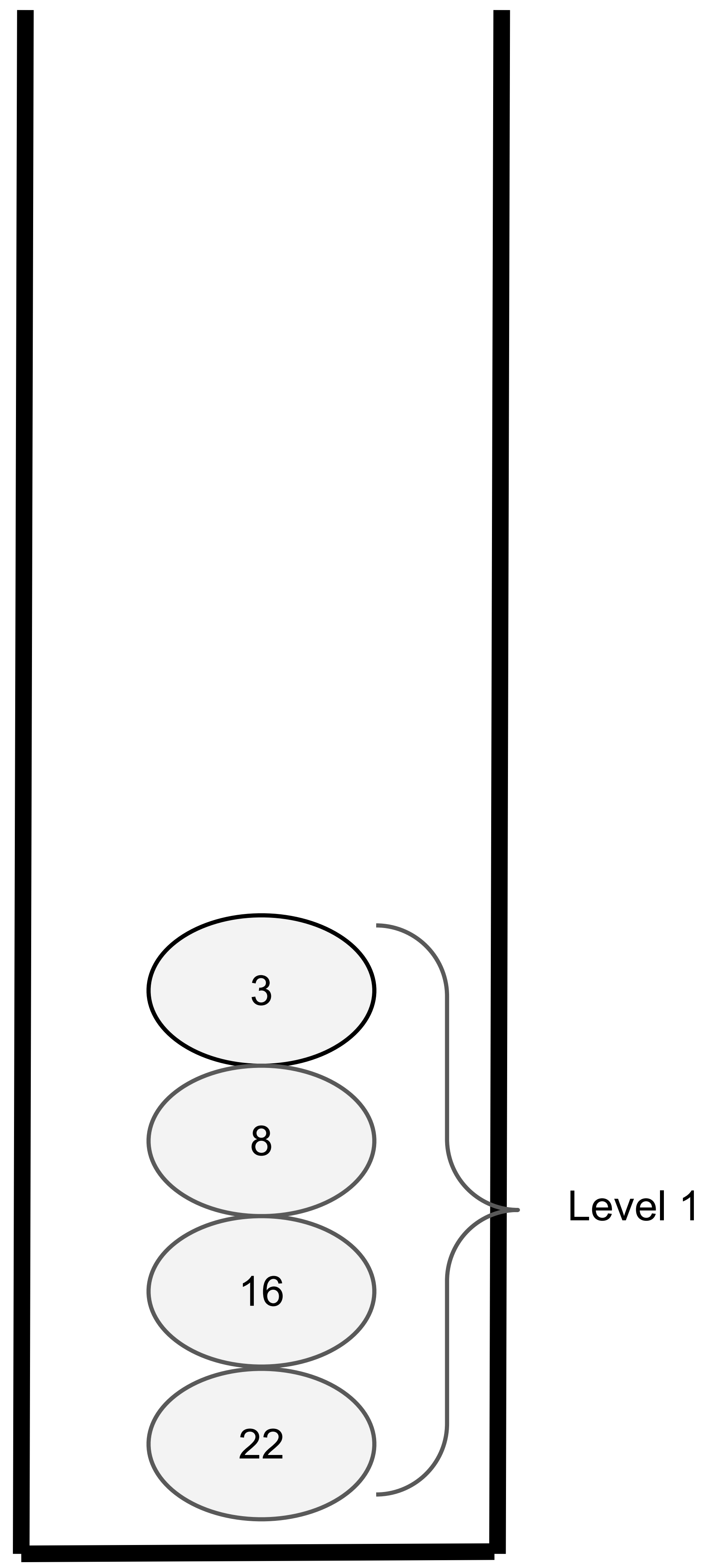
(Partial Distance)

- Sort based on PD
- Insert in active queue

SD Computational Profile

Evaluation



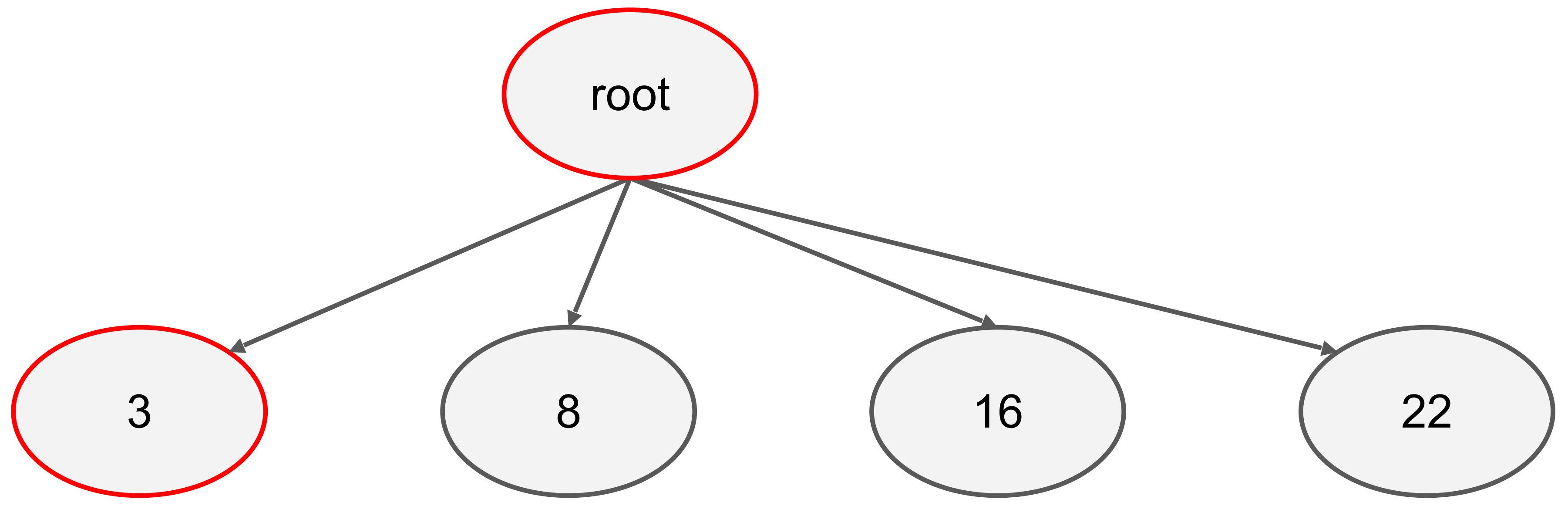


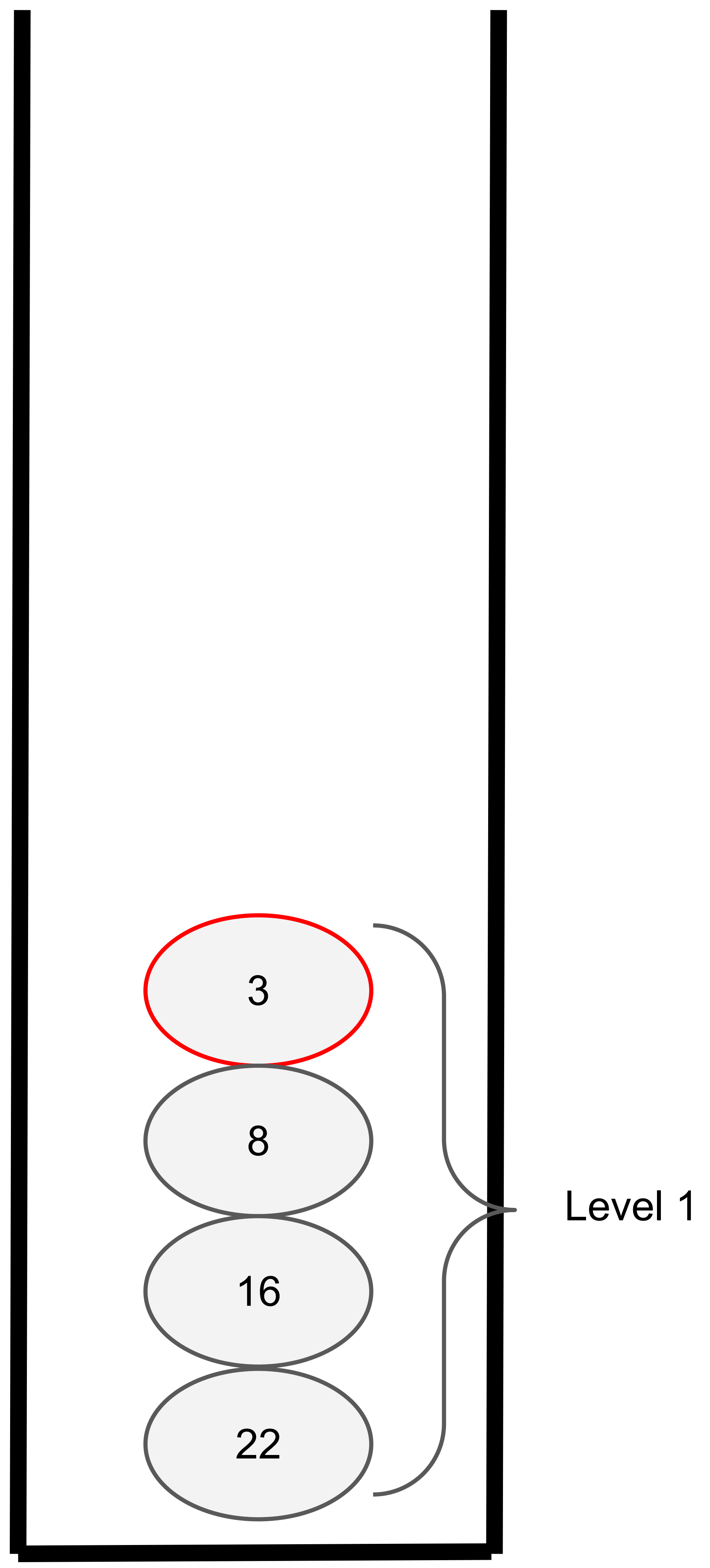
- Queue is LIFO

(Last In First Out)

SD Computational Profile

Selection





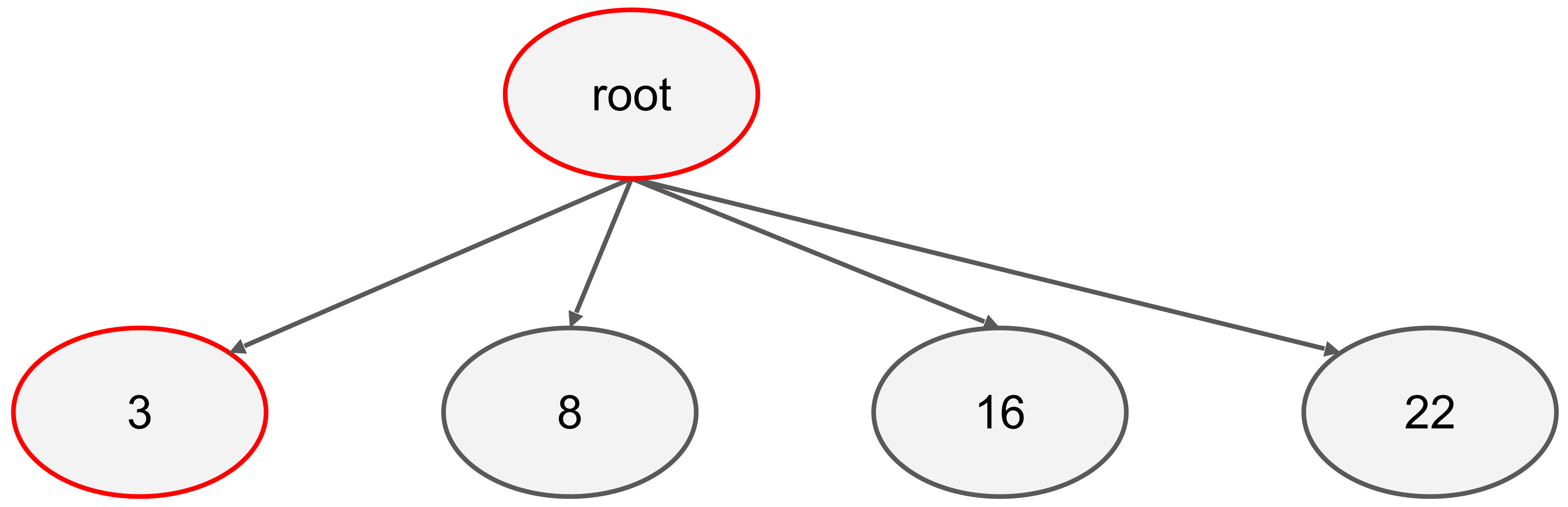
- Queue is LIFO

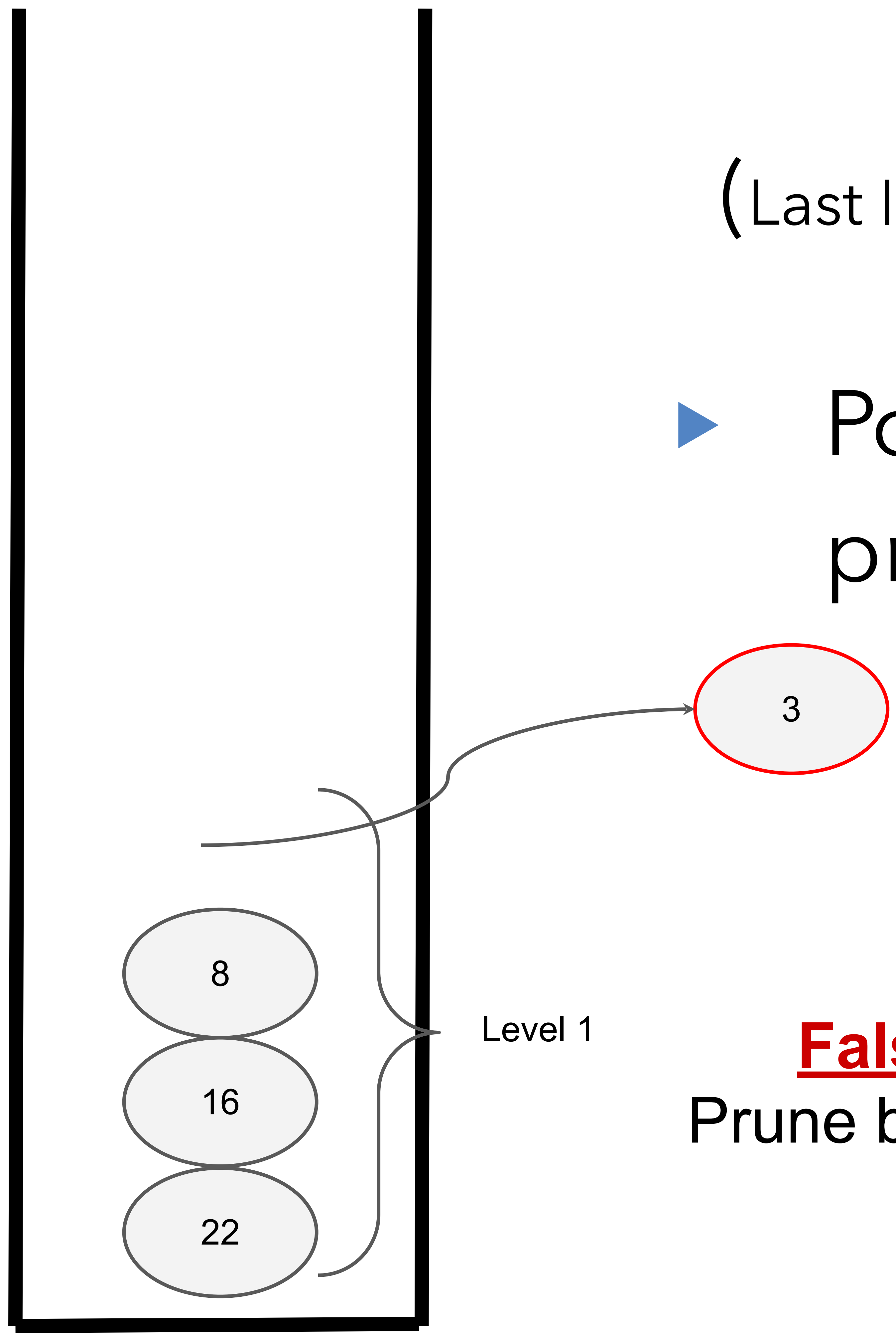
(Last In First Out)

- Pop most promising node

SD Computational Profile

Selection



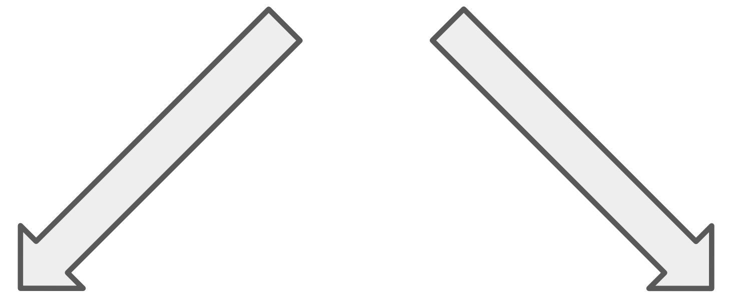


- Queue is LIFO

(Last In First Out)

- Pop most promising node

if(PD < r)

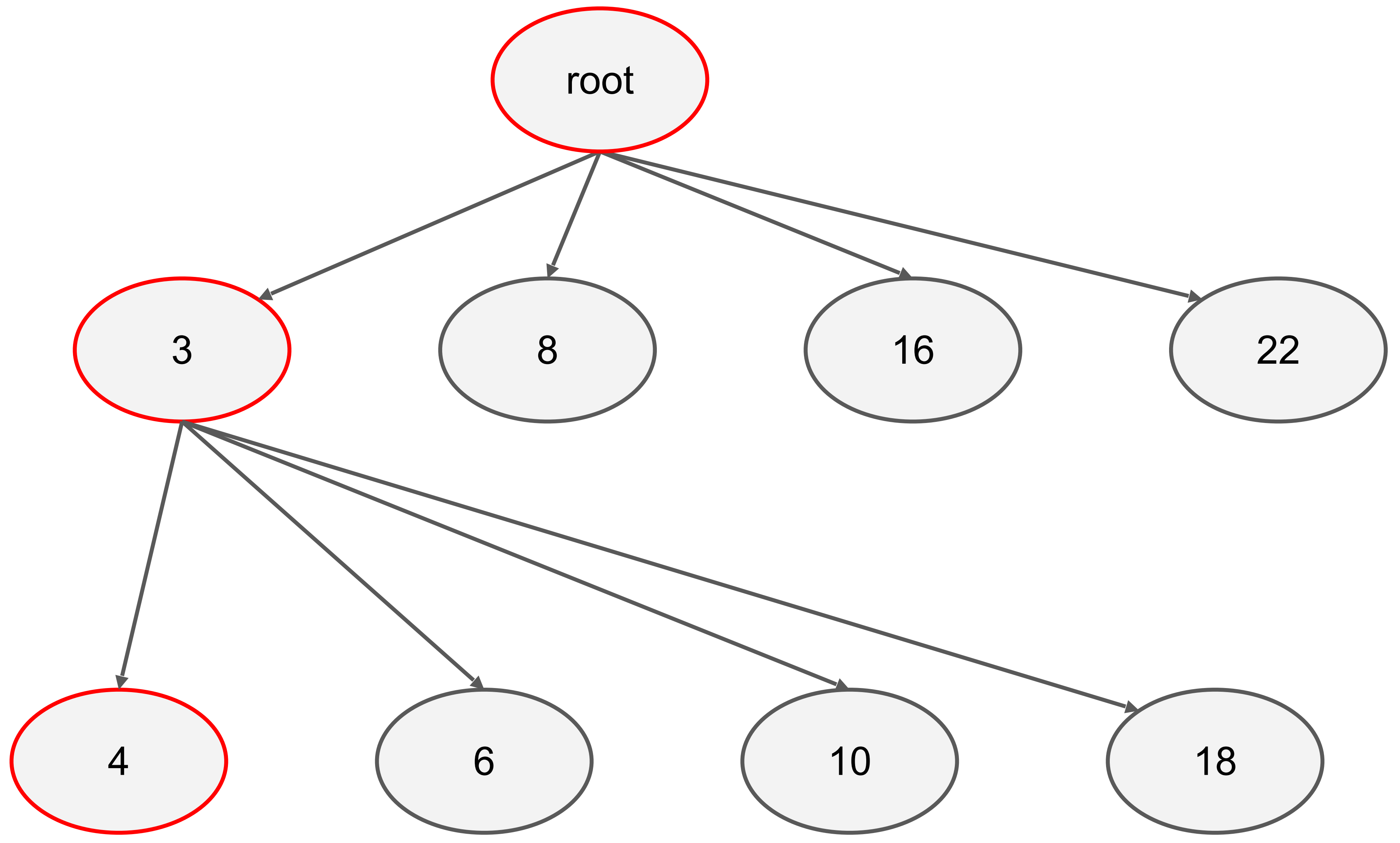


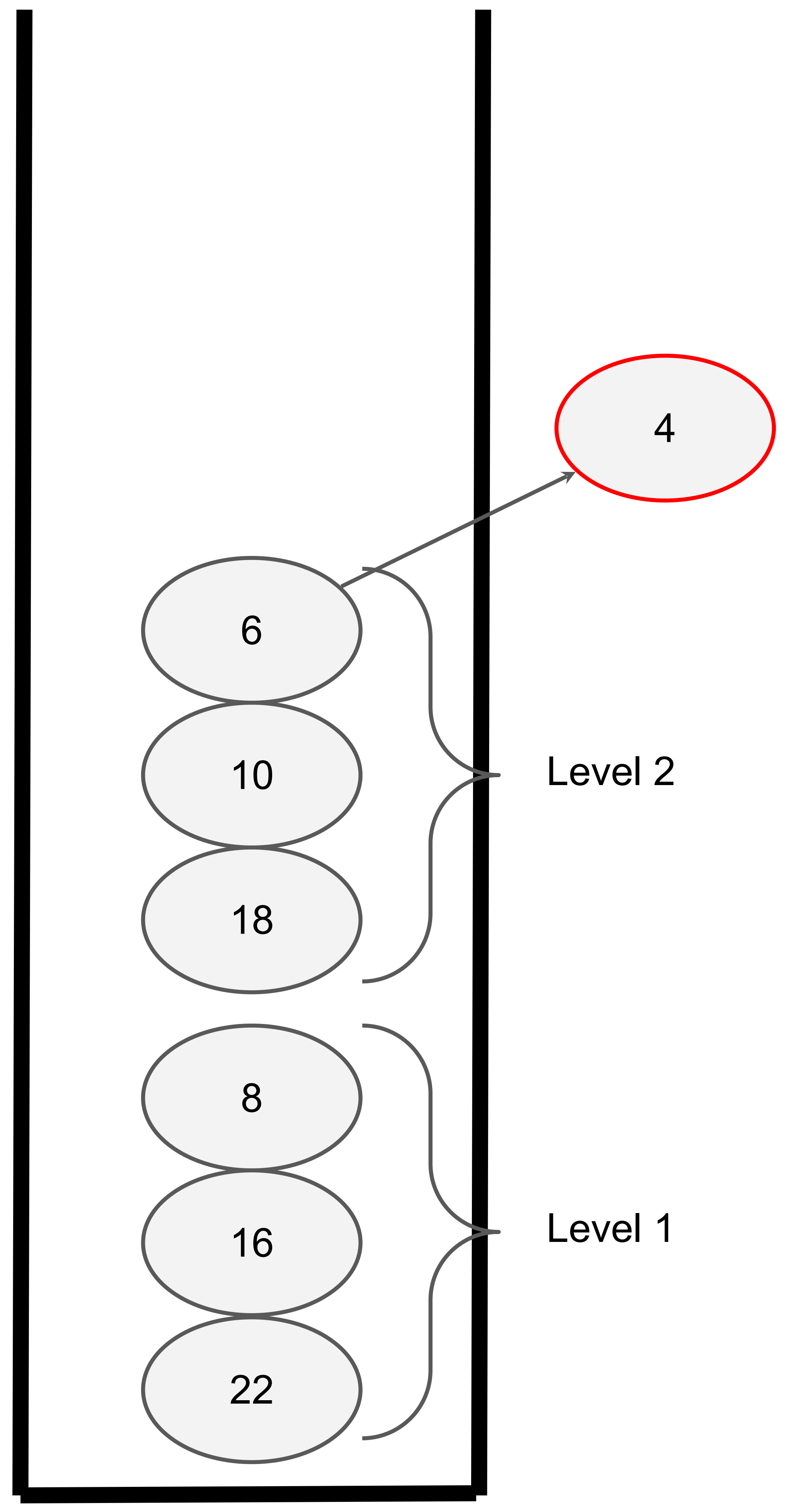
False
Prune branch

TRUE
Expand node

SD Computational Profile

Selection





- Queue is LIFO

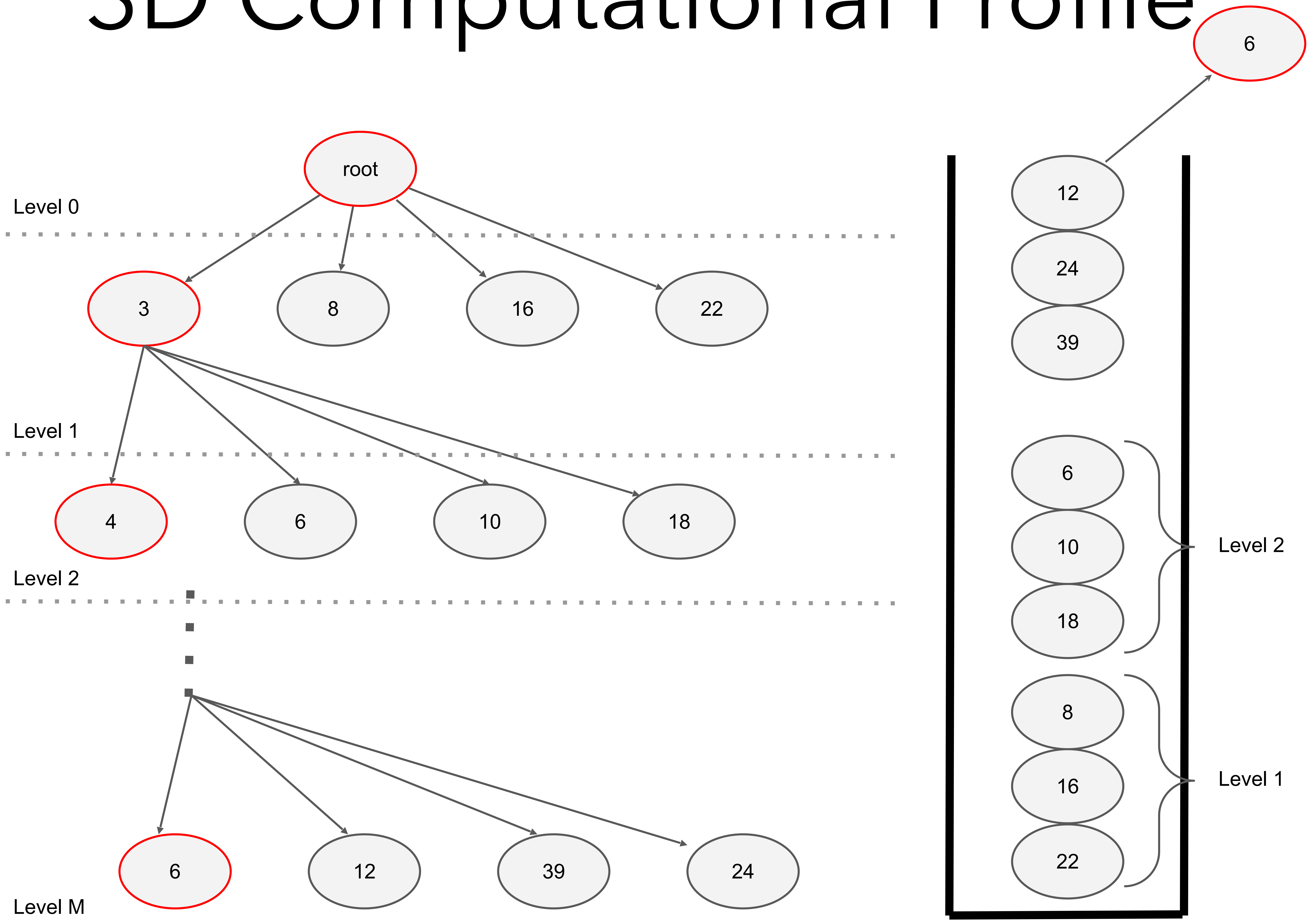
  (Last In First Out)

- Pop most promising node

- Expand next level

SD Computational Profile

Backtracking



Leaf node:

- Global synchronization
- Radius (r) updated

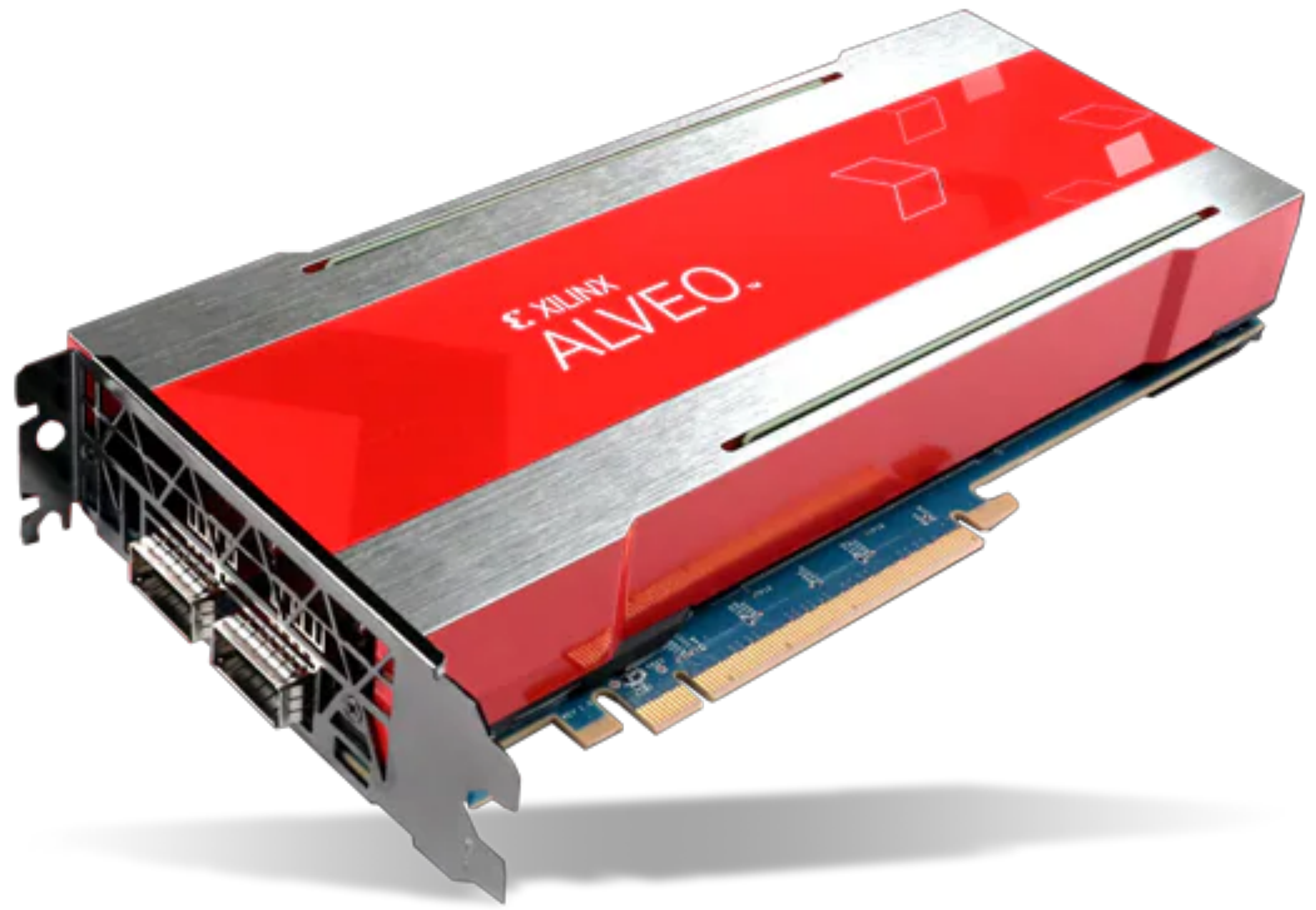
Backtrack to previous level

Evaluating Sphere Decoding

- Significant search space reduction compared to Maximum Likelihood
  - Fewer than 1% of nodes evaluated
- GEMM operation at each node is hardware amenable
- However, inherently sequential: evaluations depend on predecessors
- Global synchronisation step to update the Sphere Radius

Baseline FPGA Implementation

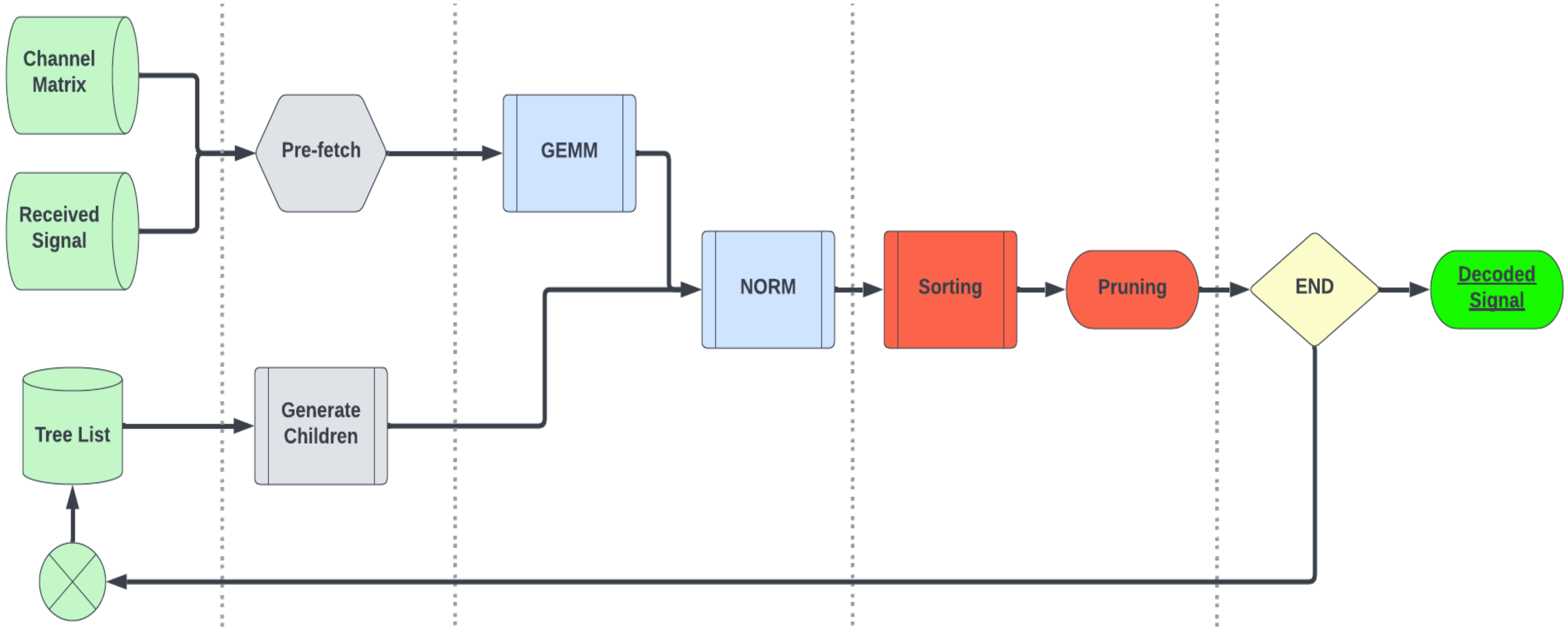
- Xilinx Alveo U280 accelerator board
- Xilinx Vitis HLS 2020.2 using C++
- Architecture design:
  - Channel matrix and received signal are sent to the FPGA once to be stored in HBM
  - The search tree is built dynamically in the configurable fabric
- Results in comparable performance to an optimised CPU implementation



Optimising Hardware Architecture

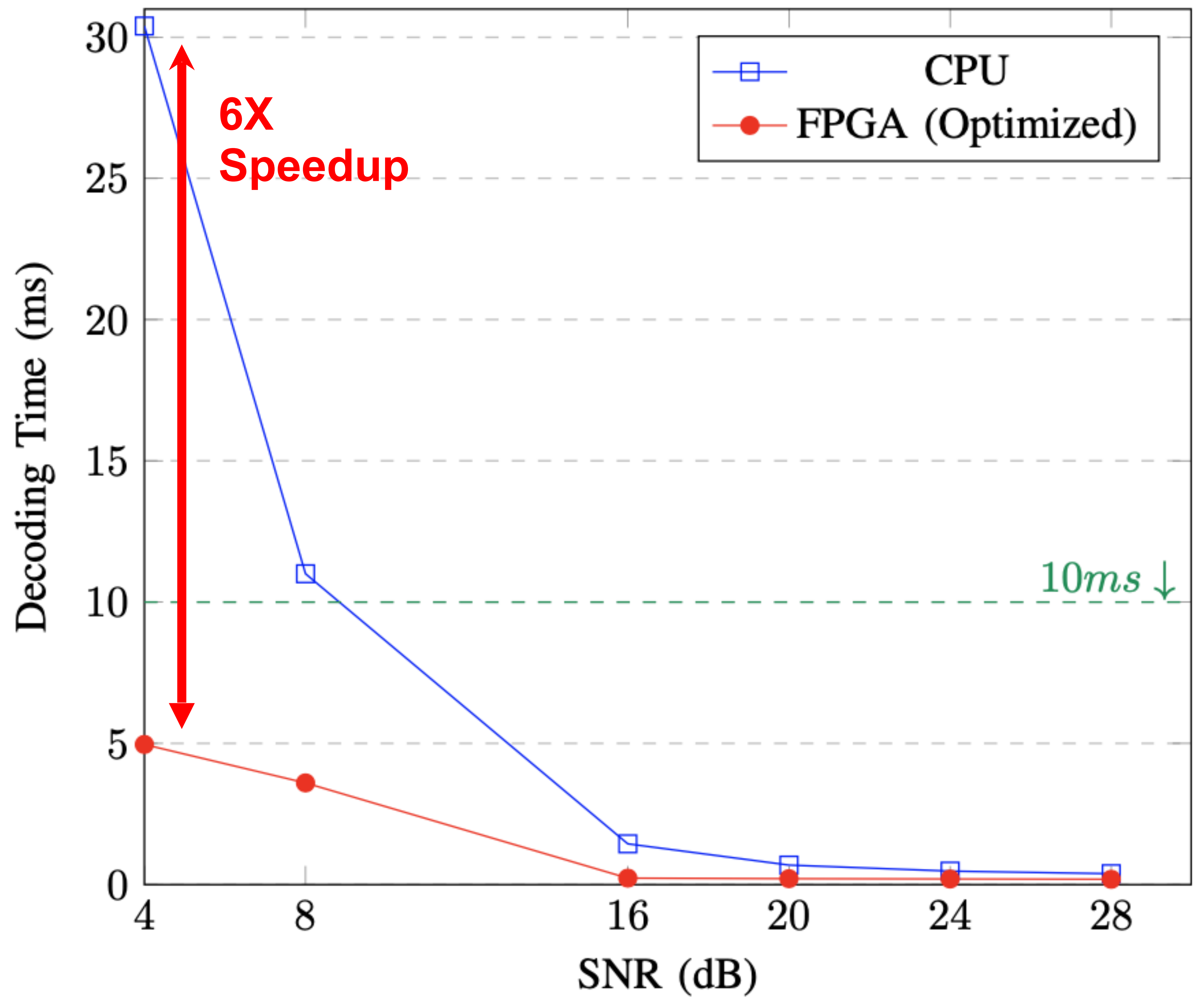
- Optimised GEMM Engine
  - Motivation: all nodes must perform GEMM
  - Systolic array implementation for high throughput
- Pre-fetching unit
  - Motivation: memory access pattern is data dependent
  - Prefetch GEMM operands for next iteration
- Metastate table
  - Motivation: Pointer based addressing not feasible in hardware
  - Indirection using meta state table improves performance

Optimising Hardware Architecture



Optimising Hardware Performance

- Shown for 15x15 MIMO using 4-QAM demodulation
- Realtime constraint of 10ms
- FPGA satisfies this for a lower SNR, which is better



Further Optimisations

- Move away from Depth-First-Search
  - Partition tree into M levels
  - Keep track of minimum PD for each level
  - Sort each level's queue with minimum PD explored first
- Further reduction in nodes explored and further 3–4× performance gain

Further Optimisations

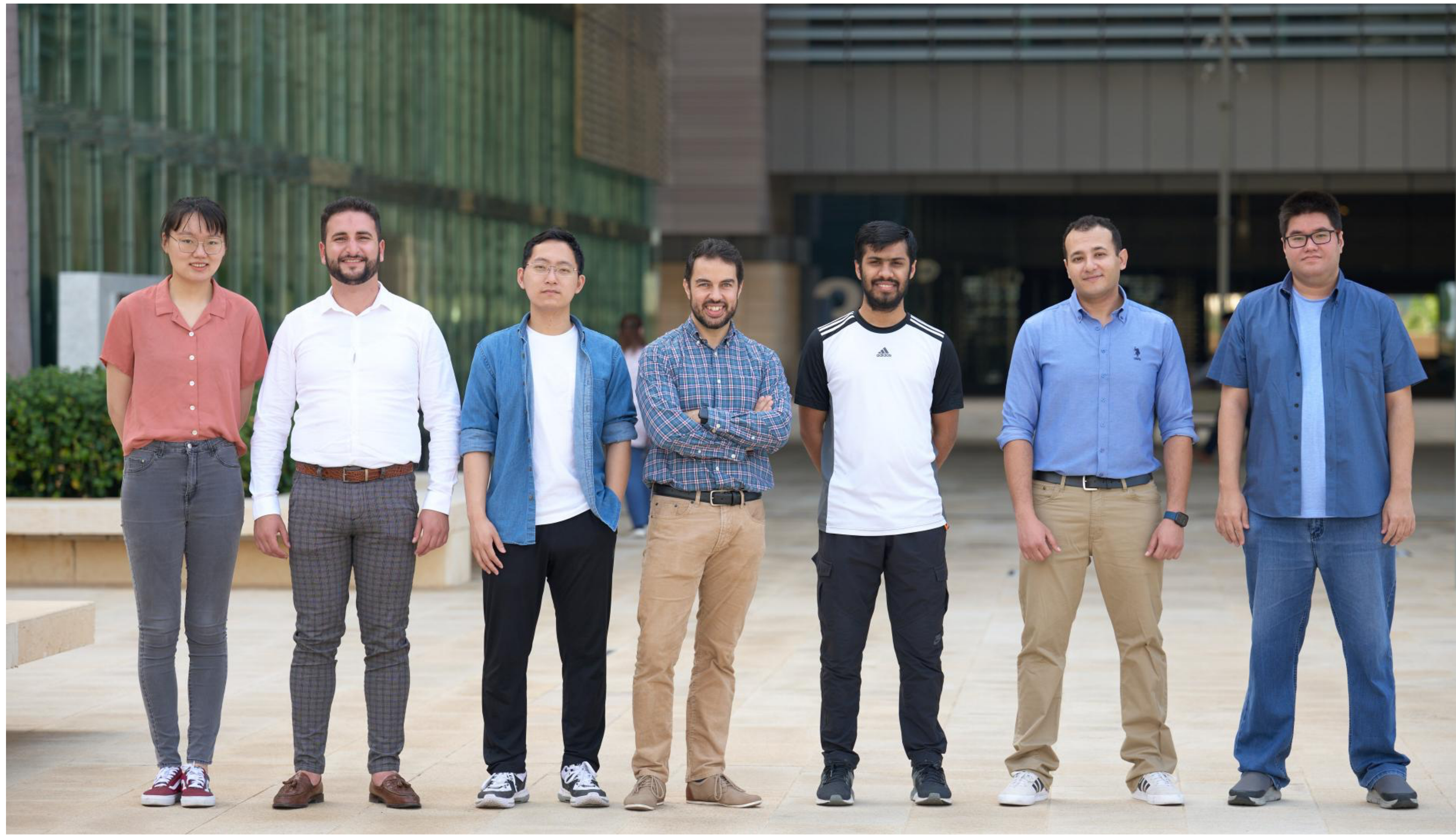
- Rather than backtracking one level, backtrack to the level with the lowest PD
- Track the global minimum encountered so far in a distributed manner
- Combine this global tracking with depth-first-search
- Preliminary results show an additional 10× performance improvement
- However, higher memory requirement and sorting becomes the bottleneck

About KAUST

- One the shores of the Red Sea, about an hour from the port city of Jeddah
- A graduate-only research university: MS and PhD programs
- All students receive scholarships
- Access to world-class facilities

About ACCL

- The Accelerated Connected Computing Lab looks at all aspects of compute acceleration and connecting accelerators
  - Networking and systems
  - Adaptive computing
  - High performance computing
- **We are hiring!**



FPGA HPC Research at KAUST

- New FPGA research cluster
  - 8 nodes
  - 6 Alveo U280
  - 12 Alveo U55C
  - 8 VCK5000 (Versal ACAP)
  - 8 NVIDIA A100 GPU
  - 4 Samsung SmartSSD