

# Innovative Applications and Technology Pivots – A Perfect Storm in Computing



Wen-mei Hwu

Professor and Sanders-AMD Chair, ECE, NCSA

University of Illinois at Urbana-Champaign

with

Izzat El Hajj, Liwen Chang, Simon Garcia, and Carl Pearson

ECE ILLINOIS

The  
**IMPACT**  
Research Group

  
NCSA

 ILLINOIS

# Agenda

- Revolutionary paradigm shift in applications
- Post-Dennard technology pivot - heterogeneity
- An example of positive application-technology spiral
- Engineering high-efficiency software for heterogeneous computing

# A major paradigm shift

- In the 20th Century, we were able to understand, design, and manufacture what we can measure
  - Physical instruments and computing systems allowed us to see farther, capture more, communicate better, understand natural processes, control artificial processes...

# A major paradigm shift

- In the 20th Century, we were able to understand, design, and manufacture what we can measure
  - Physical instruments and computing systems allowed us to see farther, capture more, communicate better, understand natural processes, control artificial processes...
- **In the 21st Century, we are able to understand, design, and create what we can compute**
  - Computational models are allowing us to see even farther, going back and forth in time, learn better, test hypothesis that cannot be verified any other way, create safe artificial processes...

# Examples of Paradigm Shift

## 20<sup>th</sup> Century

- Small mask patterns
- Electronic microscope and Crystallography with computational image processing
- Anatomic imaging with computational image processing
- Teleconference
- GPS

## 21<sup>st</sup> Century

- Optical proximity correction
- Computational microscope with initial conditions from Crystallography
- Metabolic imaging sees disease before visible anatomic change
- Tele-emersion
- Self-driving cars

# Diving deeper into computational microscope

- Large clusters (scale out) allow simulation of biological systems of realistic space dimensions
  - 0.5Å (0.05 nm) lattice spacing needed for accuracy
  - Interesting biological systems have dimensions of mm or larger
  - Thousands of nodes are required to hold and update all the grid points.
- Fast nodes (scale up) allow simulation at realistic time scales
  - Simulation time steps at femtosecond ( $10^{-15}$  second) level needed for accuracy
  - Biological processes take milliseconds or longer
  - Current molecular dynamics simulations progress at about one day for each 10-100 microseconds of the simulated process.

# Blue Waters Science Breakthrough Example

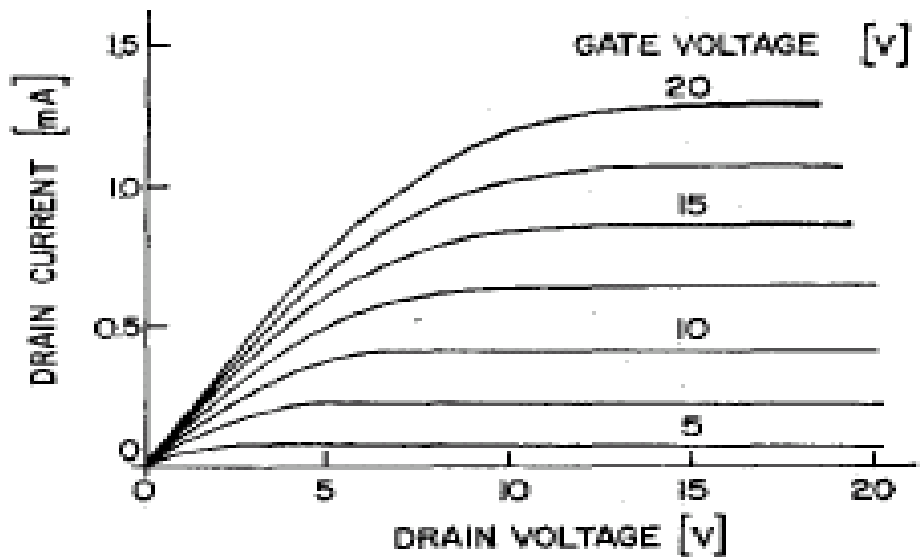
- Determination of the structure of the HIV capsid at atomic-level
- Collaborative effort of experimental groups at the U. of Pittsburgh and Vanderbilt U., and the Schulten's computational team at the U. of Illinois.
- 64-million-atom HIV capsid simulation of the process through which the capsid disassembles, releasing its genetic material
- a critical step in understanding HIV infection and finding a target for antiviral drugs.



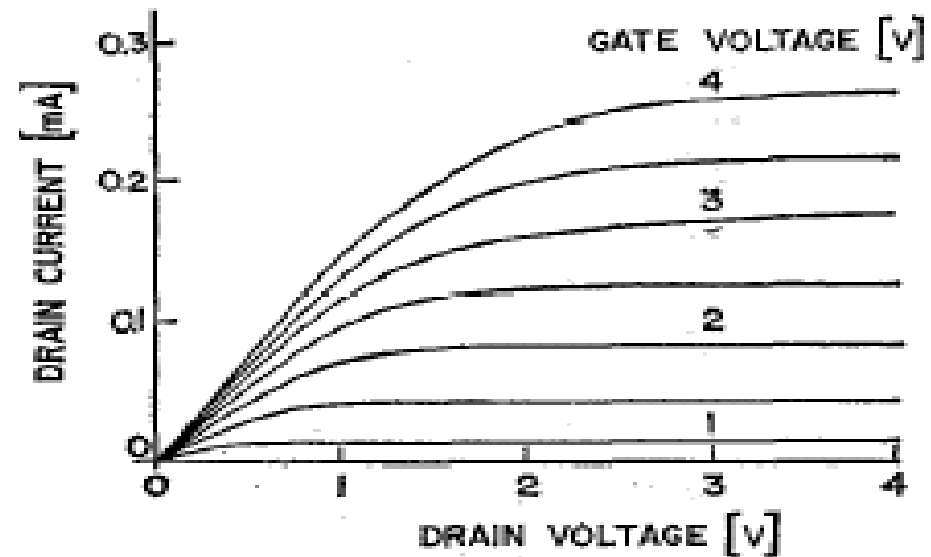
# Post-Dennard technology pivot - heterogeneity



# Dennard Scaling of MOS Devices



$t_{ox} = 1000 \text{ \AA}$   
 $L = W = 5 \mu\text{m}$   
 $V_{sub} = -7 \text{ V}$   
 $\psi_s = 0.65 \text{ V}$

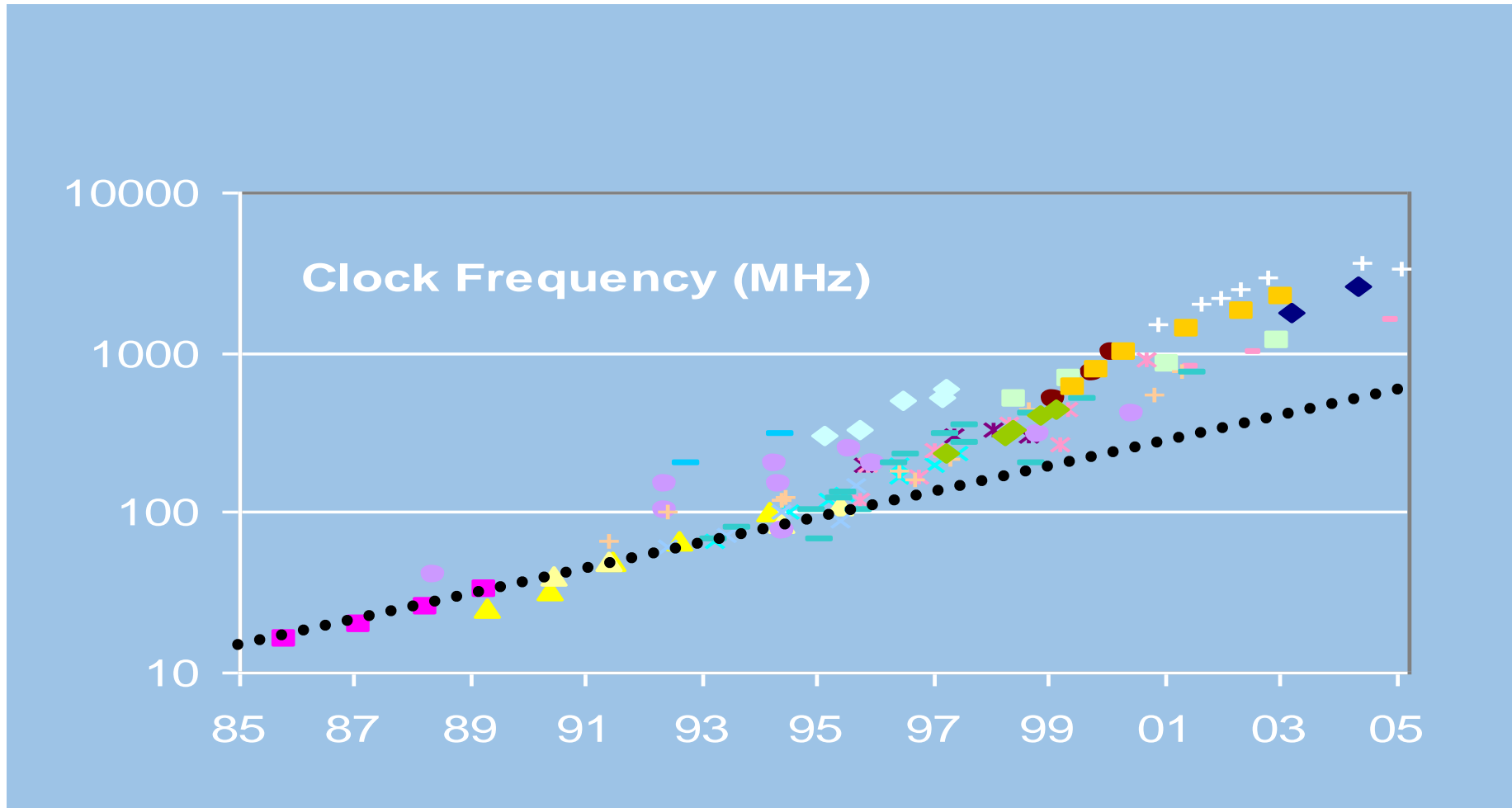


$t'_{ox} = 200 \text{ \AA}$   
 $L' = W' = 1 \mu\text{m}$   
 $V'_{sub} = -1 \text{ V}$   
 $\psi'_s = 0.73 \text{ V}$

JSSC Oct 1974, page 256

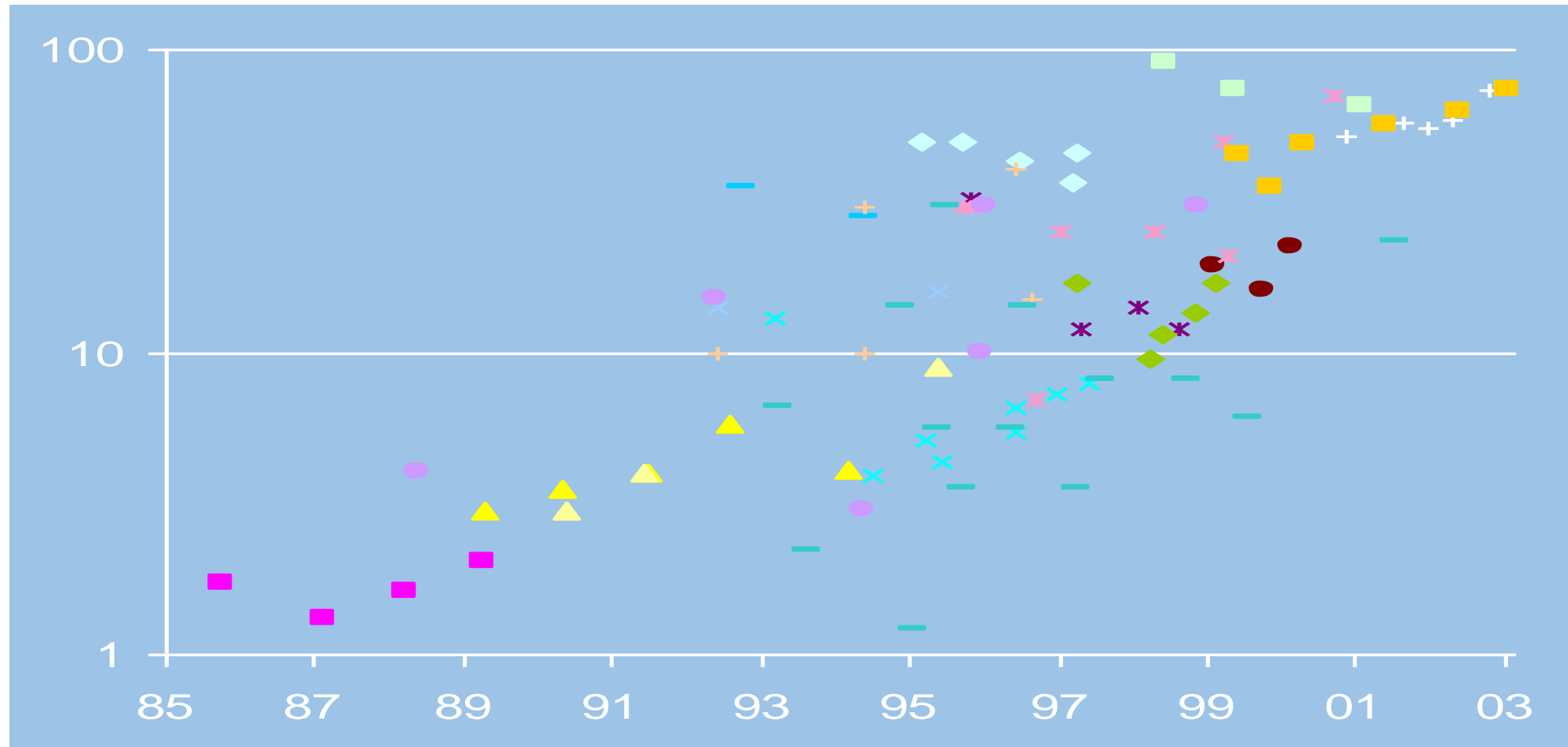
- In this ideal scaling, as  $L \rightarrow \alpha * L$ 
  - $V_{DD} \rightarrow \alpha * V_{DD}$ ,  $C \rightarrow \alpha * C$ ,  $i \rightarrow \alpha * i$
  - Delay =  $CV_{DD}/I$  scales by  $\alpha$ , so  $f \rightarrow 1/\alpha$
  - Power for each transistor is  $CV^2 * f$  and scales by  $\alpha^2$ 
    - keeping total power constant for same chip area

# Frequency Scaled Too Fast 1993-2003



# Total Processor Power Increased

(super-scaling of frequency and chip size)



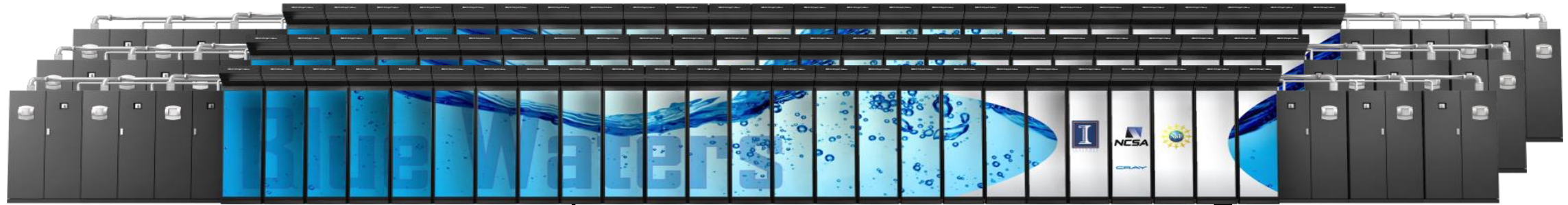
# Post-Dennard Pivoting

- Multiple cores with more moderate clock frequencies
- Heavy use of vector execution
- Employ both latency-oriented and throughput-oriented cores
- 3D packaging for more memory bandwidth

# Blue Waters Computing System

Operational at Illinois since 3/2013

49,504 CPUs -- 4,224 GPUs



12.5 PF  
1.6 PB DRAM  
\$250M

10/40/100 Gb  
Ethernet Switch

IB Switch

>1 TB/sec

120+ Gb/sec

100 GB/sec



WAN



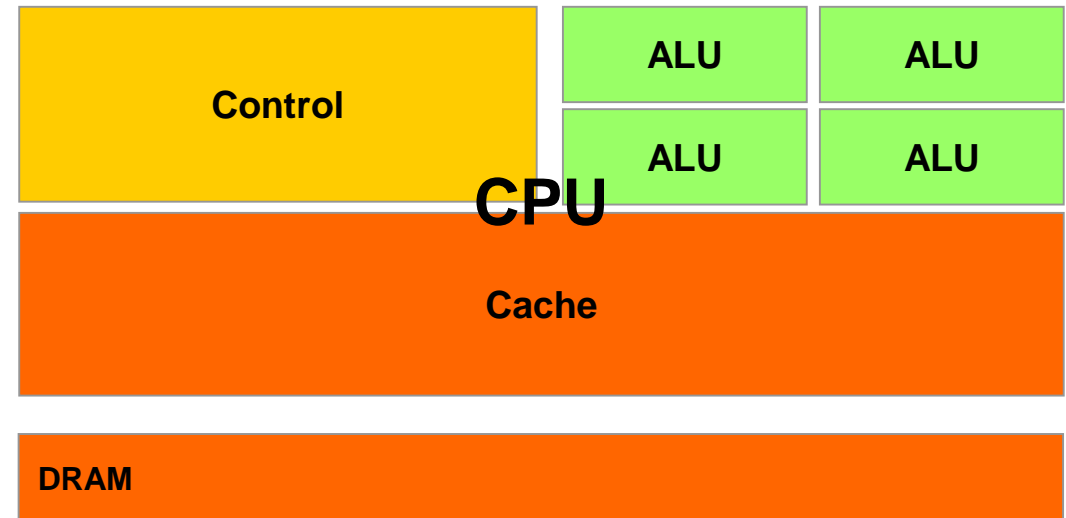
Spectra Logic: 300 PBs



Sonexion: 26 PBs

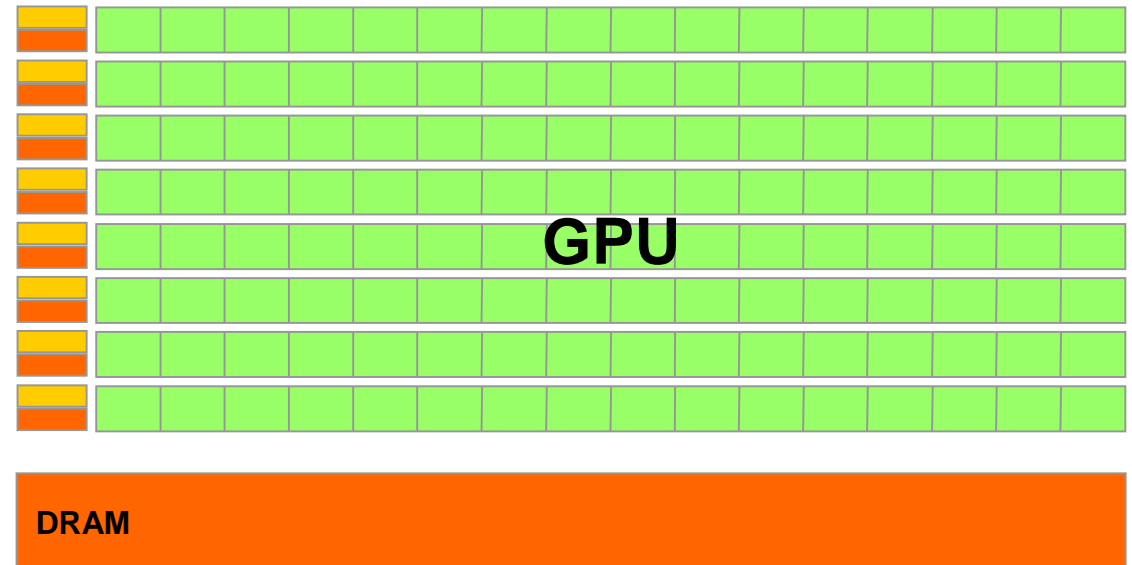
# CPUs: Latency Oriented Design

- High clock frequency
- Large caches
  - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency
- Powerful ALU
  - Reduced operation latency



# GPUs: Throughput Oriented Design

- Moderate clock frequency
- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies



# Applications Benefit from Both CPU and GPU

- CPUs for sequential parts where latency matters
  - CPUs can be 10+X faster than GPUs for sequential code
- GPUs for parallel parts where throughput wins
  - GPUs can be 10+X faster than CPUs for parallel code



# Initial Production Use Results

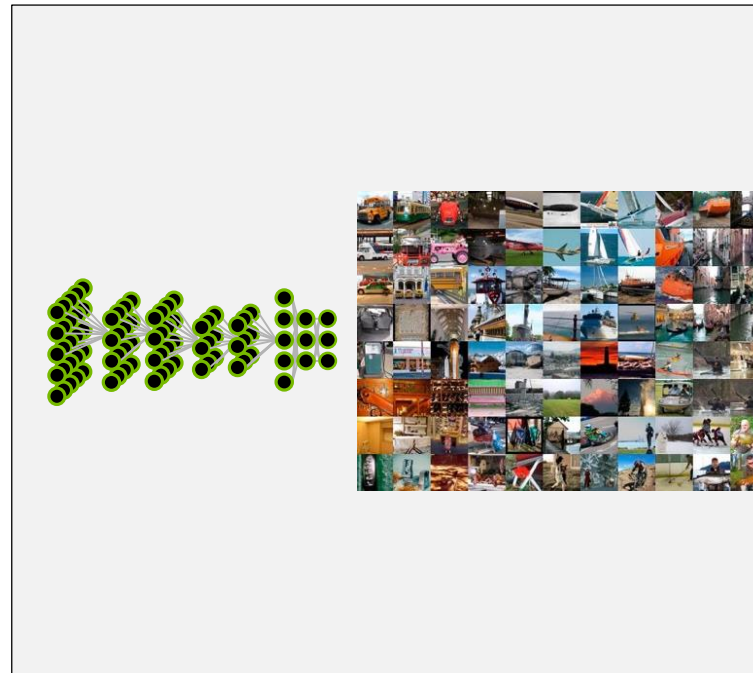
	Application Description	Application Speedup
<b>NAMD</b>	100 million atom benchmark with Langevin dynamics and PME once every 4 steps, from launch to finish, all I/O included	1.8
<b>Chroma</b>	Lattice QCD parameters: grid size of 483 x 512 running at the physical values of the quark masses	2.4
<b>QMCPACK</b>	Full run Graphite 4x4x1 (256 electrons), QMC followed by VMC	2.7
<b>ChaNga</b>	Collisionless N-body stellar dynamics with multipole expansion and hydrodynamics	2.1
<b>AWP</b>	Anelastic wave propagation with staggered-grid finite-difference and realistic plastic yielding	1.2

An example of positive  
application-technology spiral

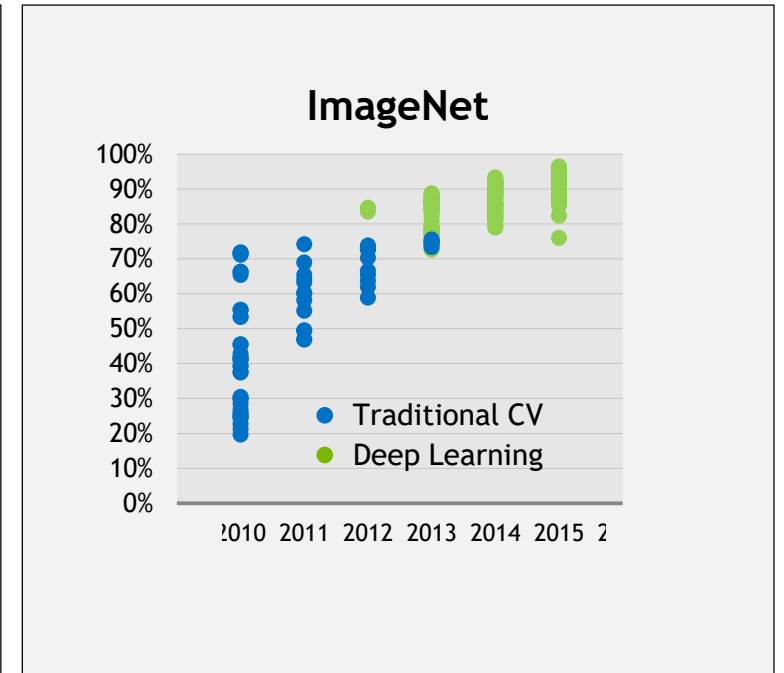
# DEEP LEARNING IN COMPUTER VISION



Traditional Computer Vision  
Experts + Time

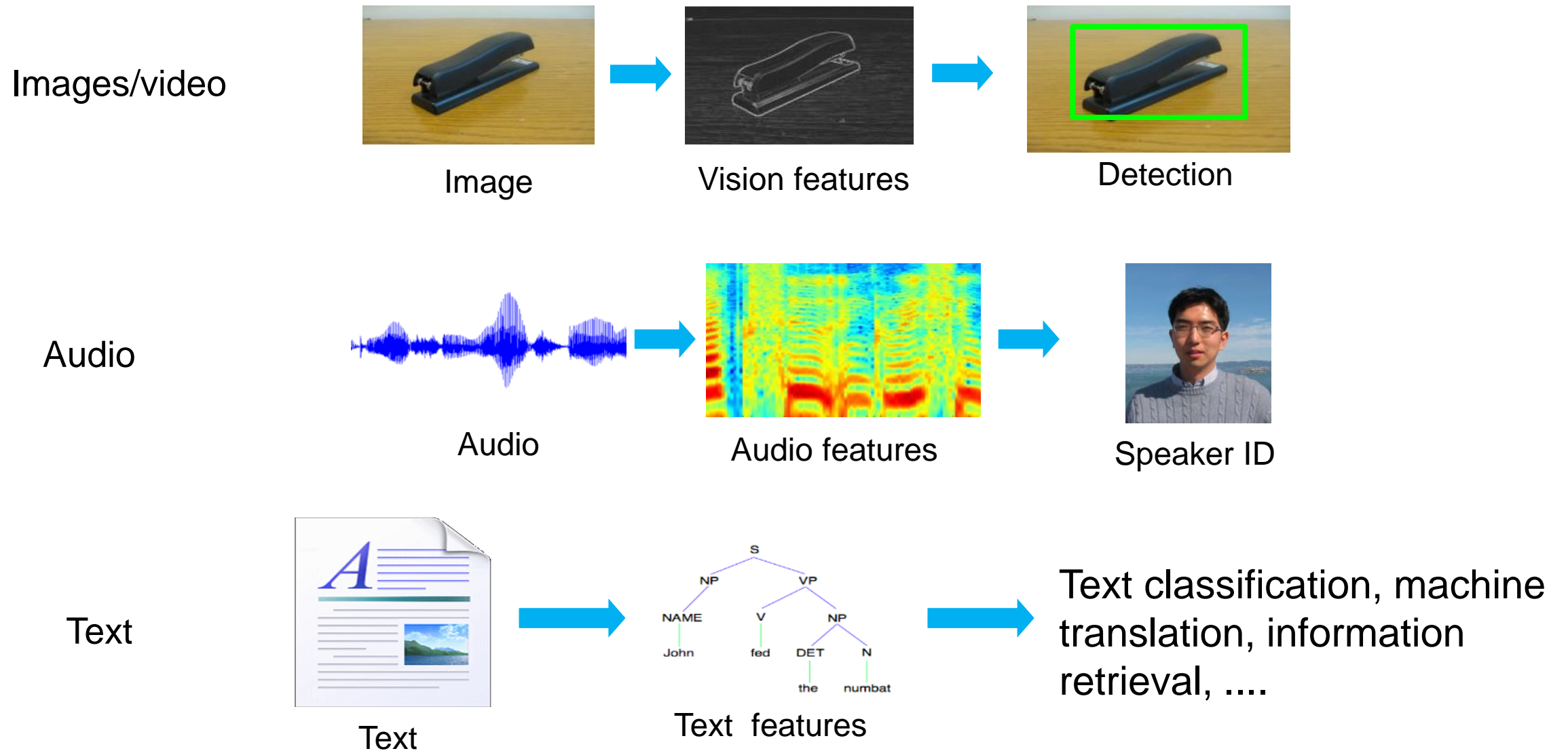


Deep Learning Object Detection  
DNN + Data + HPC



Deep Learning Achieves  
"Superhuman" Results

# DIFFERENT MODALITIES OF REAL-WORLD DATA



# A long way to go towards cognitive computing

## ▼ Social Sciences

Use the cartoon to answer the next TWO questions.



24

What economic condition motivated Phil to request a raise?

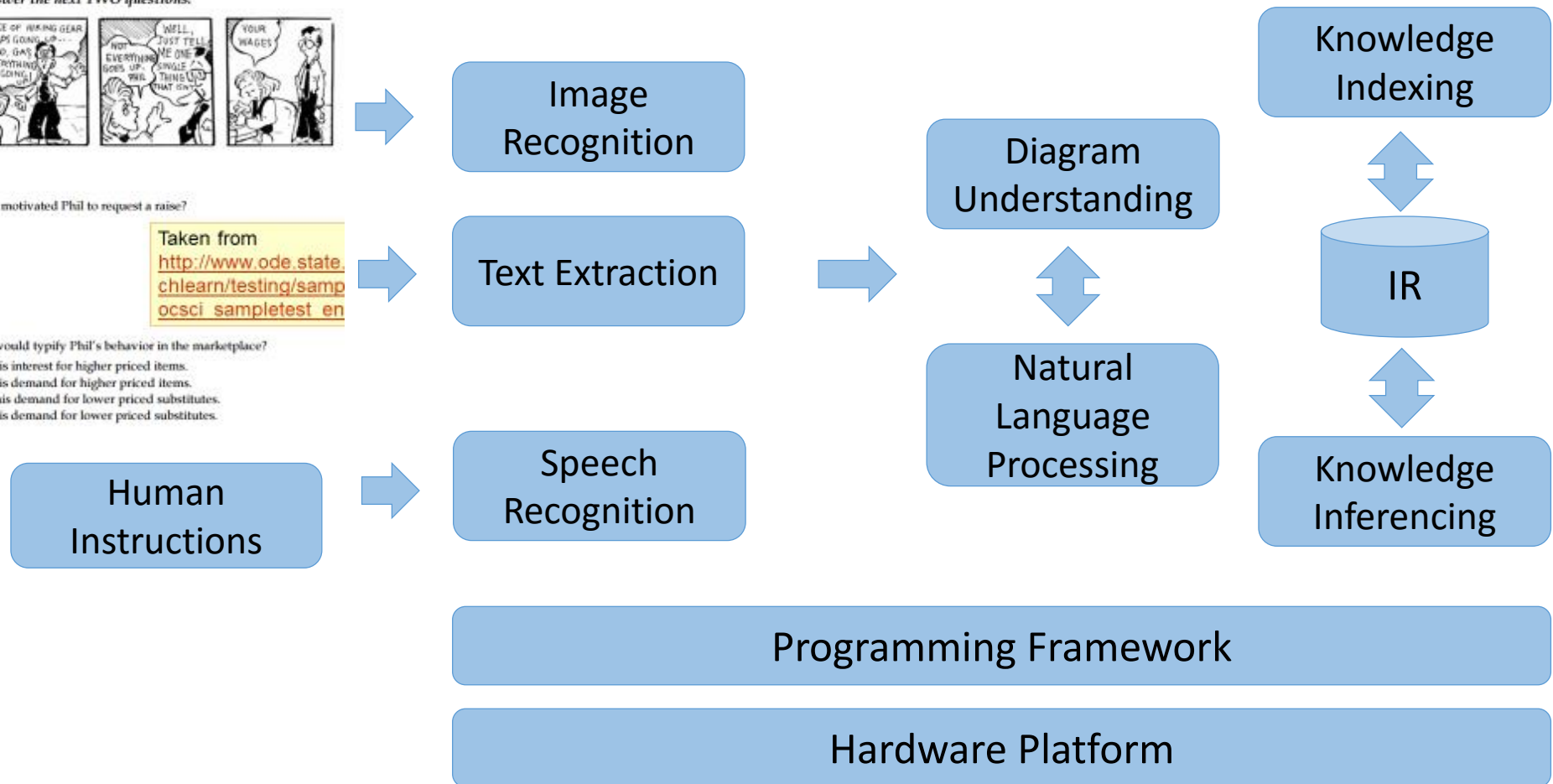
- A. Inflation
- B. Specialization
- C. Unemployment
- D. Embargo

Taken from  
<http://www.ode.state.ch/learn/testing/sampletest/en>

25

Without his raise, which would typify Phil's behavior in the marketplace?

- A. He will increase his interest for higher priced items.
- B. He will increase his demand for higher priced items.
- C. He will decrease his demand for lower priced substitutes.
- D. He will increase his demand for lower priced substitutes.



# More Heterogeneity Is Coming

- Beyond traditional CPUs and GPUs
  - FPGAs (e.g., Microsoft FPGA cloud)
  - ASICs (e.g., Google's TPU)
- Beyond traditional DRAM
  - Stacked DRAM for more memory bandwidth
  - Non-volatile RAM for memory capacity
  - Near/in memory computing for reduced power used in data movement

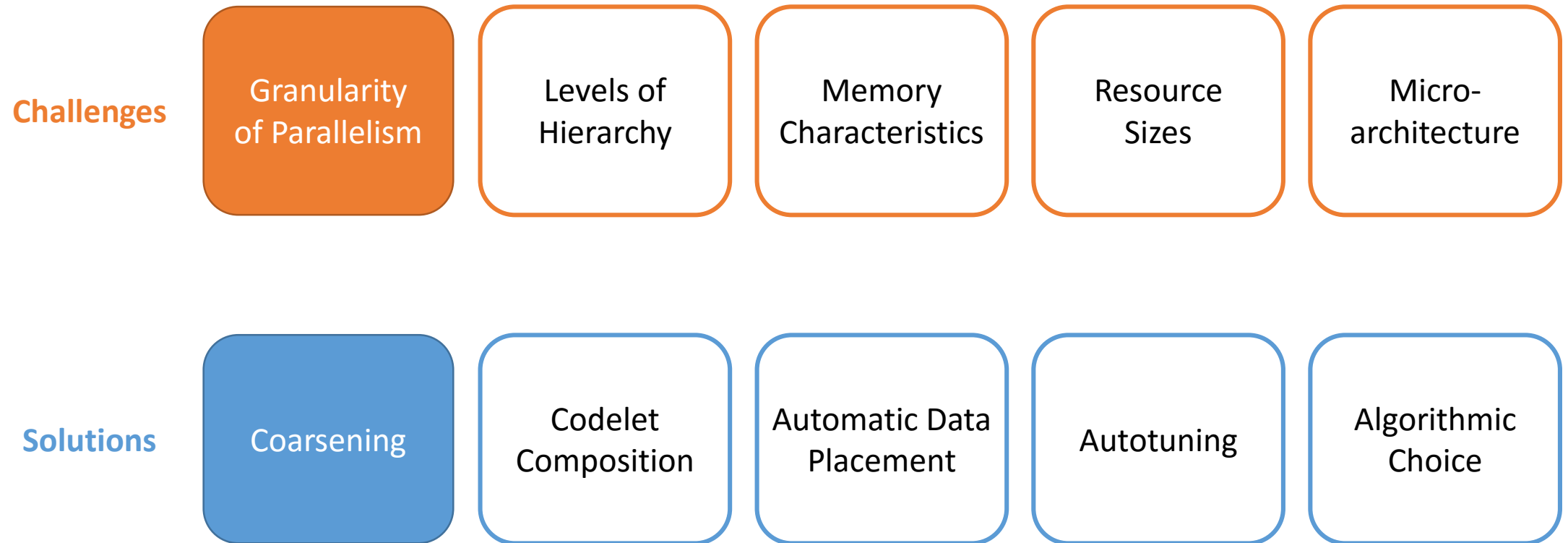
# Some Lessons Learned

- Throughput computing using GPUs can result in 2-3X end-to-end application-level performance improvement
- GPUs, big data and deep learning have formed a positive spiral for the industry
- GPU computing has so far had narrow but deep impact in the application space
  - Data movement overhead and small GPU memory
    - Unified memory, HBM, NVLink, and HSA-style systems will help
  - **Low-level programming interfaces with poor performance portability**

# Engineering high-efficiency software for heterogeneous computing

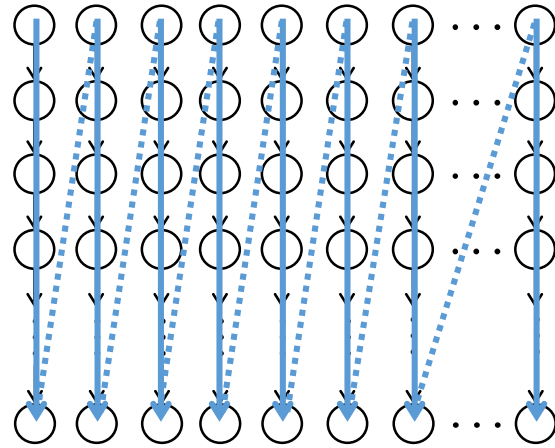


# Performance-Portability: One Source for All

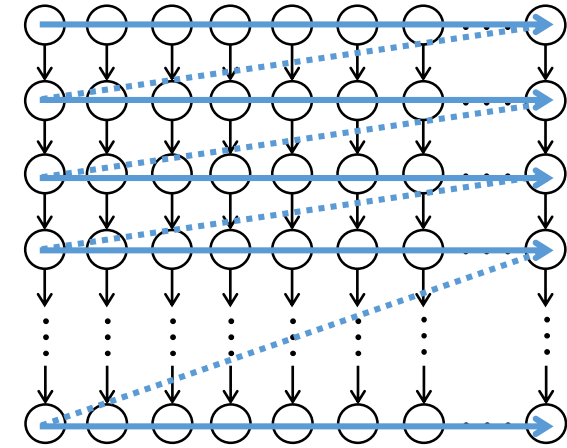


# Coarsening Scheduling Alternatives

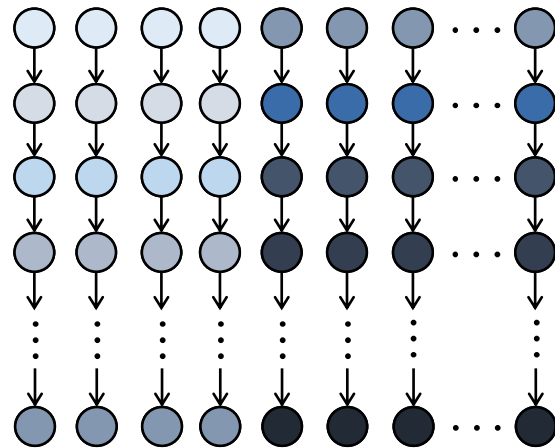
**Depth First Order (DFO) Scheduling**



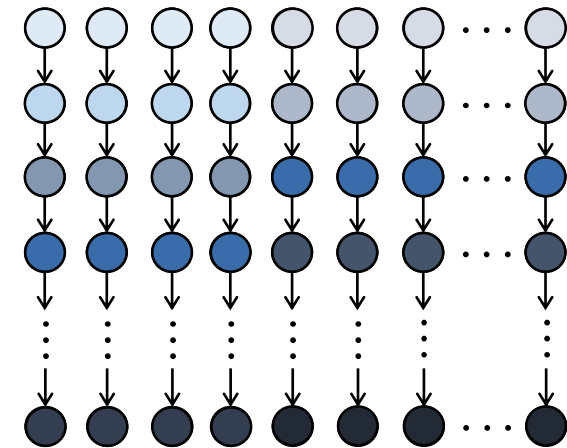
**Breadth First Order (BFO) Scheduling**



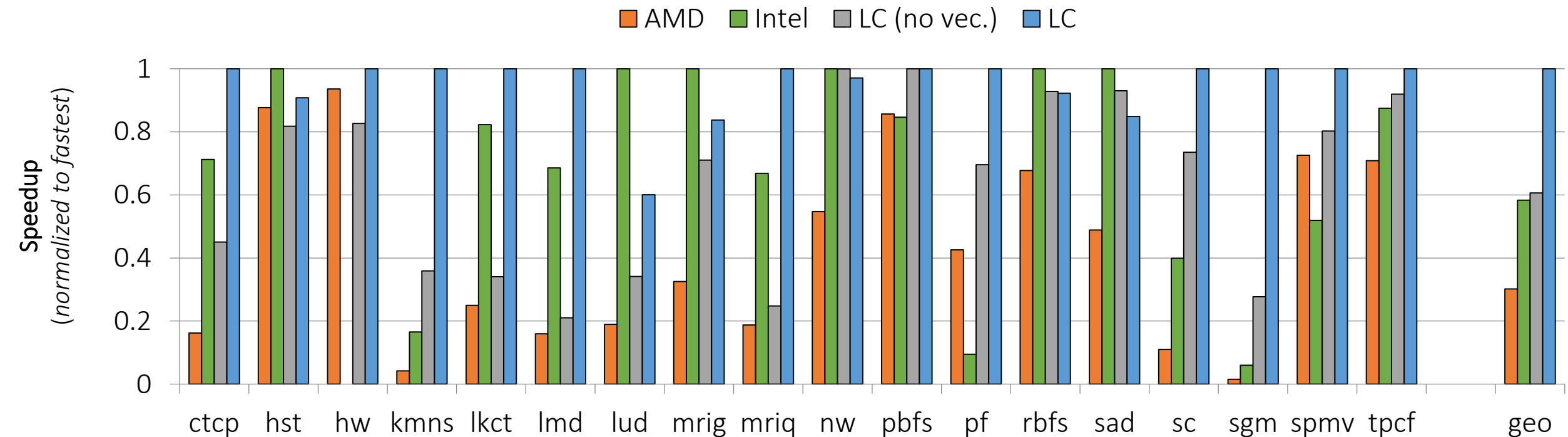
**DFO Scheduling with Vectorization**  
(time progresses as color gets darker)



**BFO with Vectorization**  
(time progresses as color gets darker)



# Performance Results

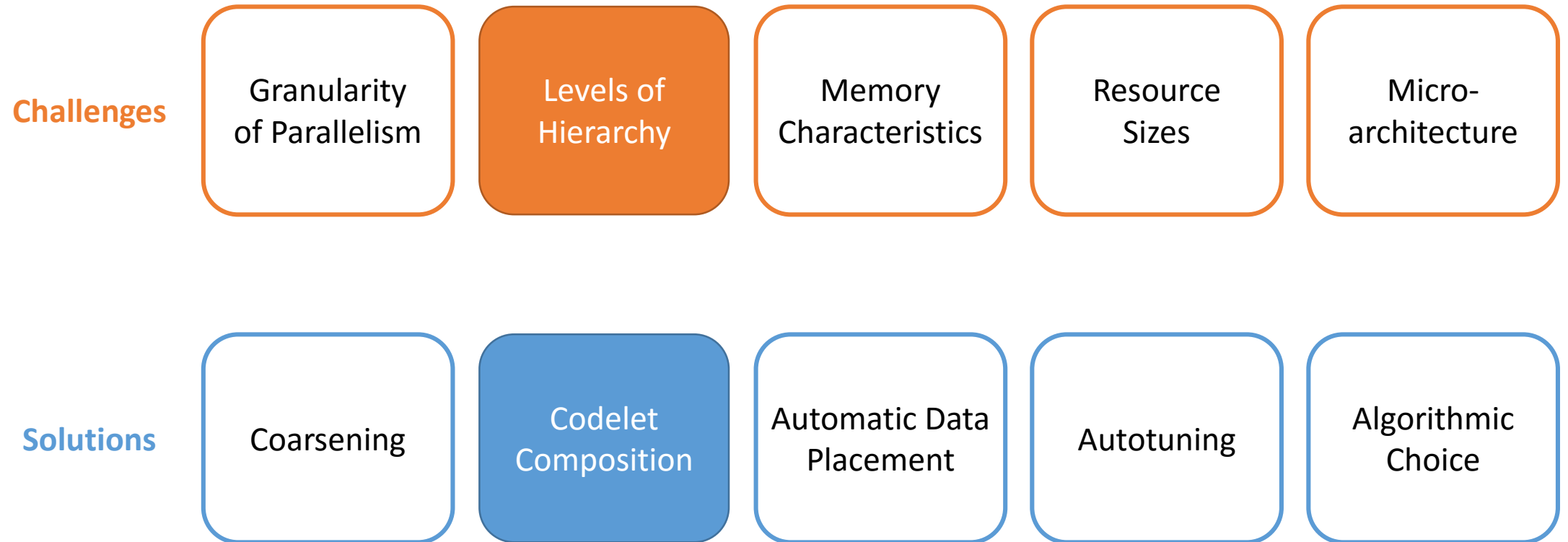


Speedups of 3.32x and 1.71x over AMD and Intel OpenCL implementations

Kim et al., CGO'15



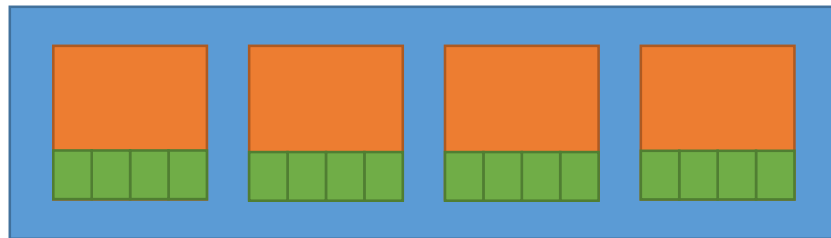
# Performance-Portability: One Source for All



# Hierarchical Compute Organization of Devices

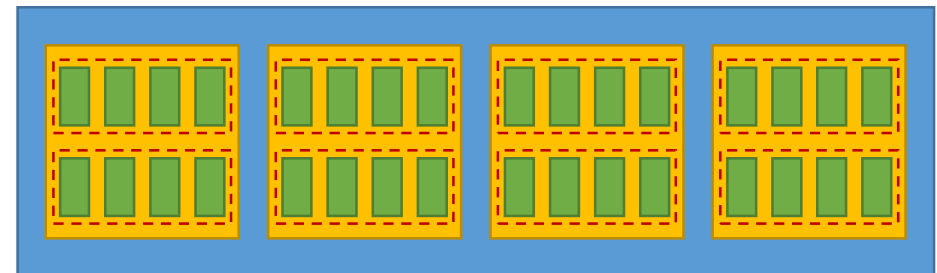
## CPU

1. Process
2. Thread (vector-capable)
3. Vector Lane
4. Instruction-level Parallelism



## GPU

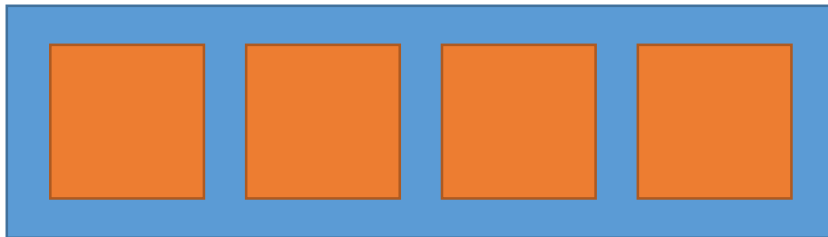
1. Grid
2. Block
3. Warp
4. Thread
5. Instruction-level Parallelism



# Hierarchical Compute Organization of Devices

## CPU

1. Process
2. Thread (vector-capable)
3. Vector Lane
4. Instruction-level Parallelism



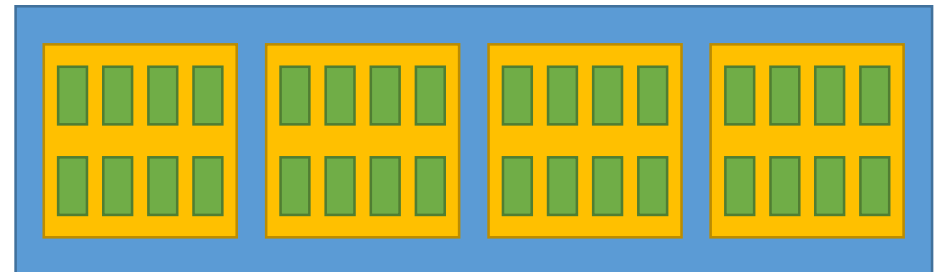
```
nt = omp_get_num_threads();
tile = (len + nt - 1)/nt;
#pragma omp parallel
{
    j = omp_get_thread_num();
    accum = 0;
    #pragma unroll
    for(int i = 0; i < tile; ++i) {
        accum += in[j*tile + i];
    }
    partial[j] = accum;
}
sum = 0;
for(int j = 0; j < nt; ++j) {
    sum += partial[j];
}
return sum;
```

# Hierarchical Compute Organization of Devices


## GPU

```
tile = (len + blockDim.x - 1)/blockDim.x;
sub_tile = (tile + blockDim.x - 1)/blockDim.x;
accum = 0
#pragma unroll
for(unsigned i = 0; i < sub_tile; ++i) {
    accum += in[blockIdx.x*tile
        + i*blockDim.x + threadIdx.x];
}
tmp[threadIdx.x] = accum;
__syncthreads();
for(unsigned s=1; s<blockDim.x; s *= 2) {
    if(id >= s)
        tmp[threadIdx.x] +=
            tmp[threadIdx.x - s];
    __syncthreads();
}
partial[blockIdx.x] = tmp[blockDim.x-1];
return; // Launch new kernel to sum up partial
```


1. Grid
2. Block
3. Warp
4. Thread
5. Instruction-level Parallelism



# Tangram: Codelet-based Programming Model

 `__codelet`  
`int sum(const Array<1,int> in) {`  
    `unsigned len = in.size();`  
    `int accum = 0;`  
    `for(unsigned i=0; i < len; ++i) {`  
        `accum += in[i];`  
    `}`  
    `return accum;`  
`}`

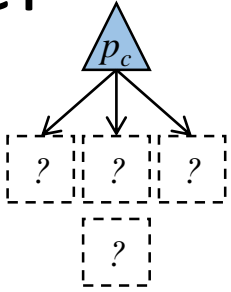
(a) Atomic autonomous codelet

 `__codelet __coop __tag(kog)`  
`int sum(const Array<1,int> in) {`  
    `__shared int tmp[coopDim()];`  
    `unsigned len = in.size();`  
    `unsigned id = coopIdx();`  
    `tmp[id] = (id < len)? in[id] : 0;`  
    `for(unsigned s=1; s<coopDim(); s *= 2) {`  
        `if(id >= s)`  
            `tmp[id] += tmp[id - s];`  
    `}`  
    `return tmp[coopDim()-1];`  
`}`

(b) Atomic cooperative codelet

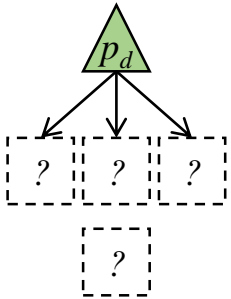
`__codelet __tag(asso_tiled)`  
`int sum(const Array<1,int> in) {`  
    `__tunable unsigned p;`  
    `unsigned len = in.size();`  
    `unsigned tile = (len+p-1)/p;`  
    `return sum( map( sum, partition(in,`  
        `p,sequence(0,tile,len),sequence(1),sequence(tile,tile,len+1))));`  
`}`

(c) Compound codelet using adjacent tiling



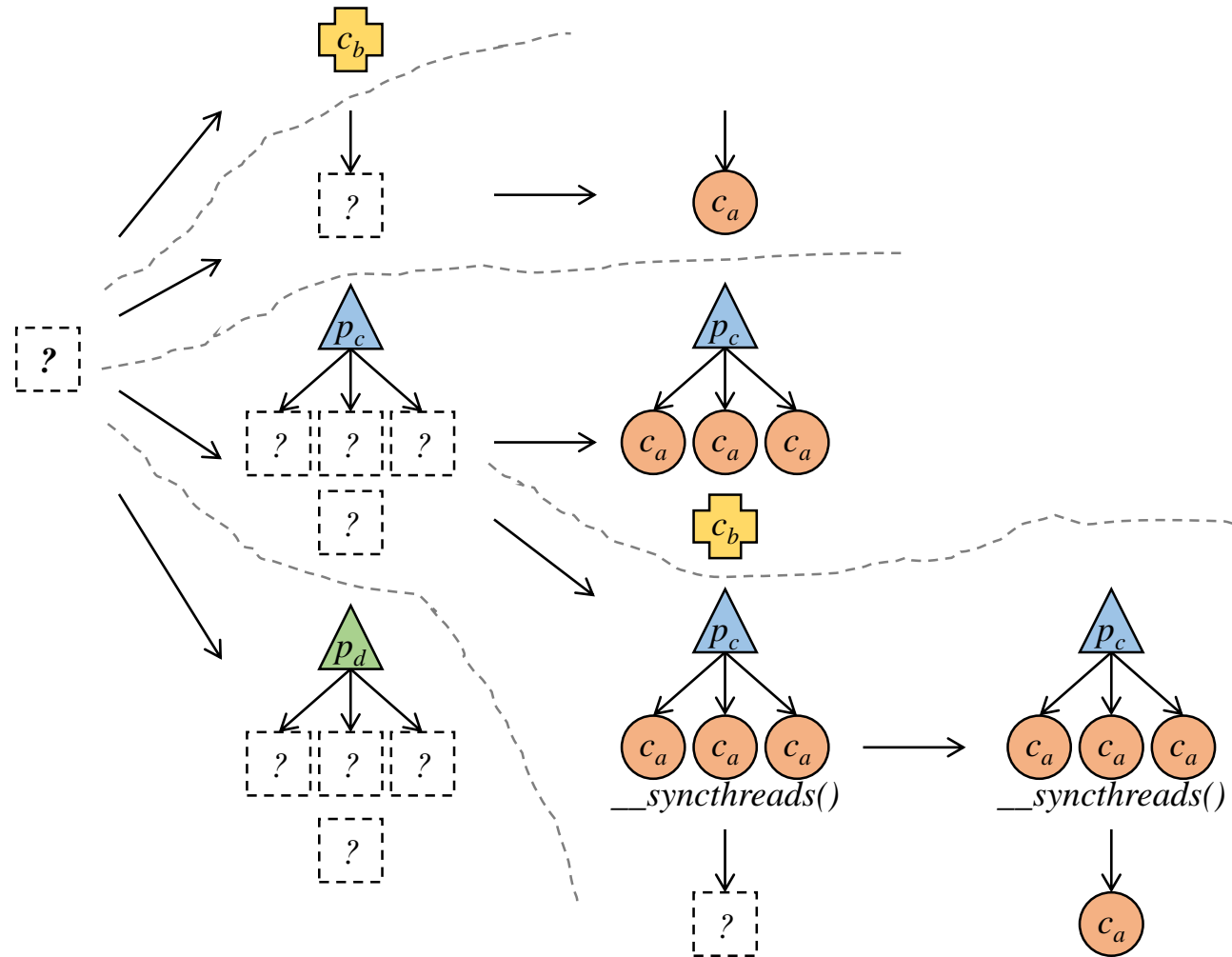
`__codelet __tag(stride_tiled)`  
`int sum(const Array<1,int> in) {`  
    `__tunable unsigned p;`  
    `unsigned len = in.size();`  
    `unsigned tile = (len+p-1)/p;`  
    `return sum( map( sum, partition(in,`  
        `p,sequence(0,1,p),sequence(p),sequence((p-1)*tile,1,len+1))));`  
`}`

(d) Compound codelet using strided tiling

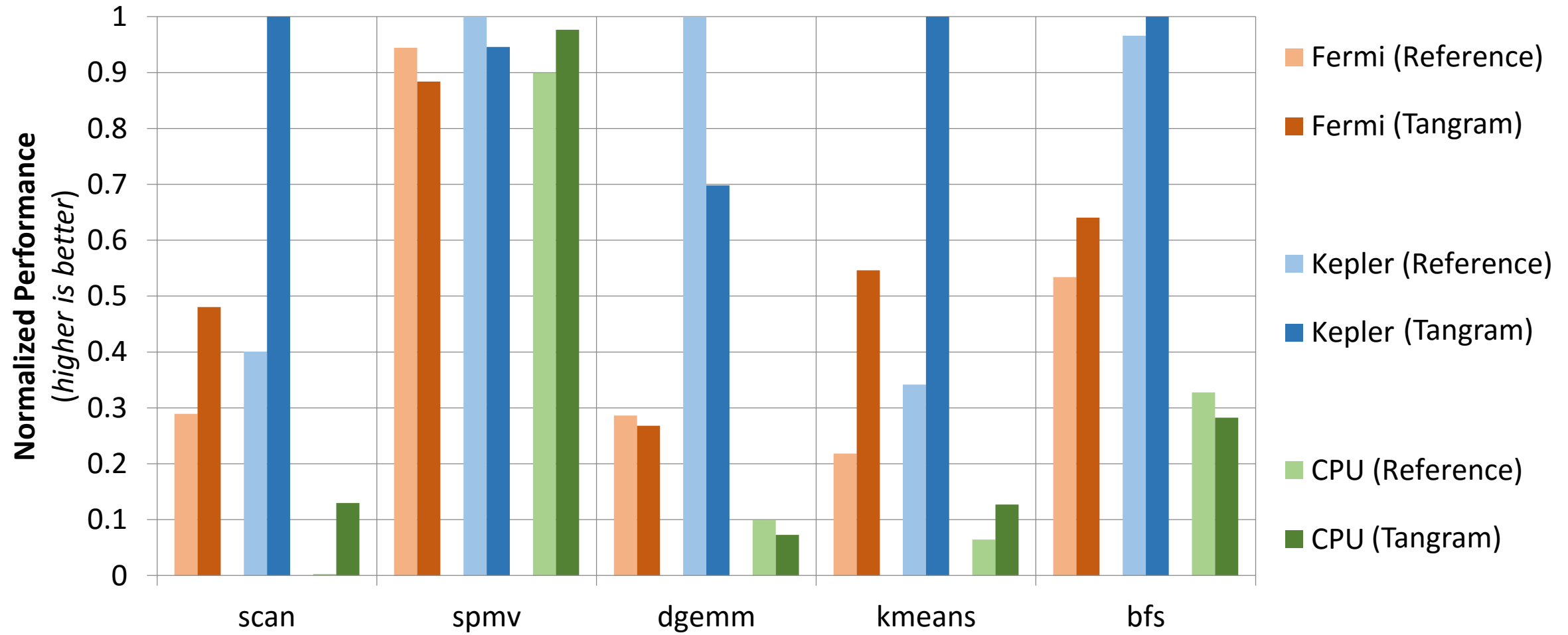




# Tangram: Composition Example



# Tangram Results



# Need for Run-time Selection

- Statically determining best algorithm could be difficult or infeasible
  - Sometimes it is input dependent
- Even a robust compiler or an expert could select suboptimal sequence of optimization
  - A catastrophic performance loss could happen

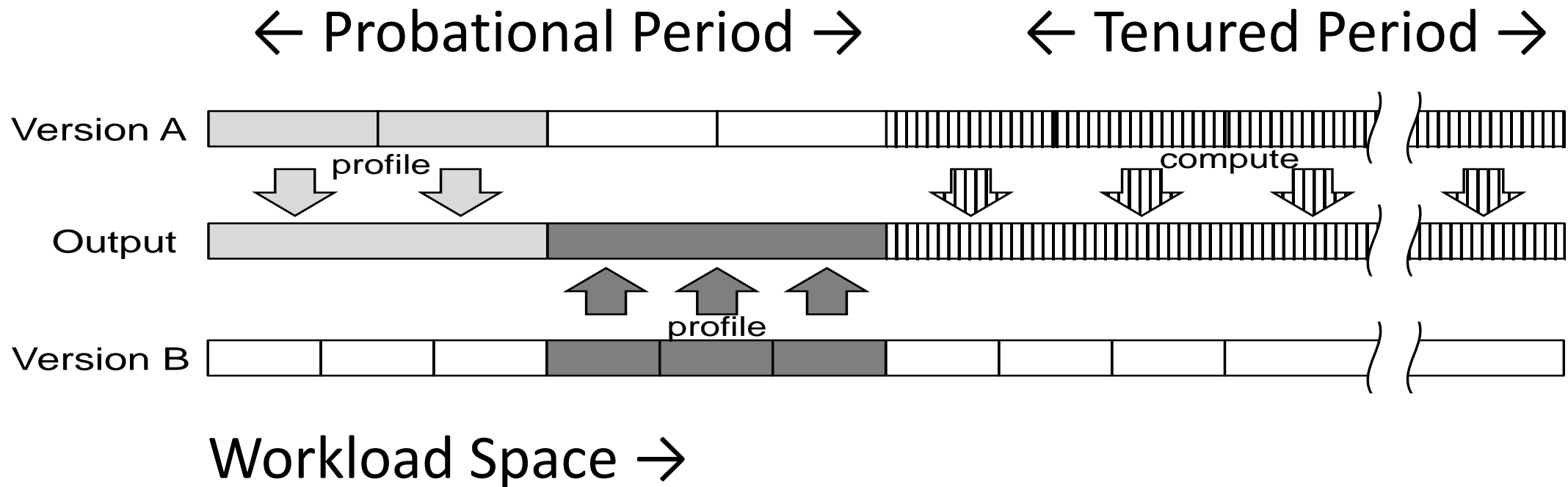
# DySel Runtime Selects the Best Version

- Application or compiler provides multiple versions
  - Typically 4-10
- Runtime performs the final selection
  - Apply micro-profiling to sample the performance of each candidate
  - Use a small subset of the actual workload per candidate
    - Contributes to final result
  - Profile candidates concurrently
    - Reduces profiling overhead
- Incurs less than 8% of overhead in the worst observed case



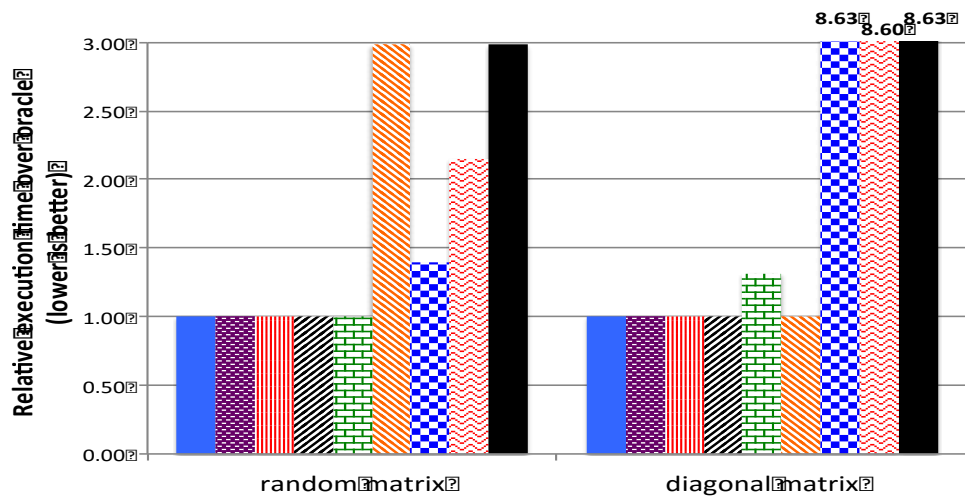
# Productive Profiling Mode

- Computation in profiling also contributes to the final output

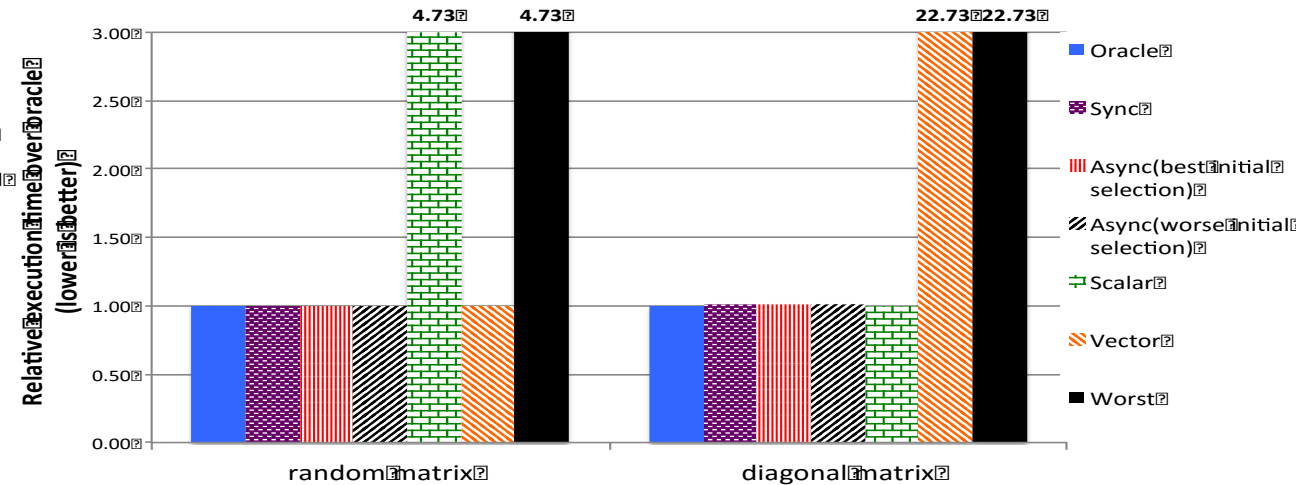


# Case Study: Input-dependent Optimizations

- Best optimizations could be input-dependent



(a) CPU



(b) GPU

# Conclusion and Outlook

- Applications have very large appetite for more computing power
  - Both larger scale clusters and faster devices
- Heterogeneity has become the norm for all hardware systems
  - HPC community are currently seeing about 2-3x application speedup
  - Recent positive spiral between deep learning and GPU computing
  - More positive spirals are yet to come
- Performance portability is critical for broad software adoption
  - There is critical need for programming systems with strong support for portability
  - Performance portability involves several dimensions of technical challenges
  - Unfortunately, vendors have not been interested in solving this problem.

Thank you!



# Backup Slides

# Performance-Portability: One Source for All

## Challenges

Granularity  
of Parallelism

Levels of  
Hierarchy

Memory  
Characteristics

Resource  
Sizes

Micro-  
architecture

## Solutions

Coarsening

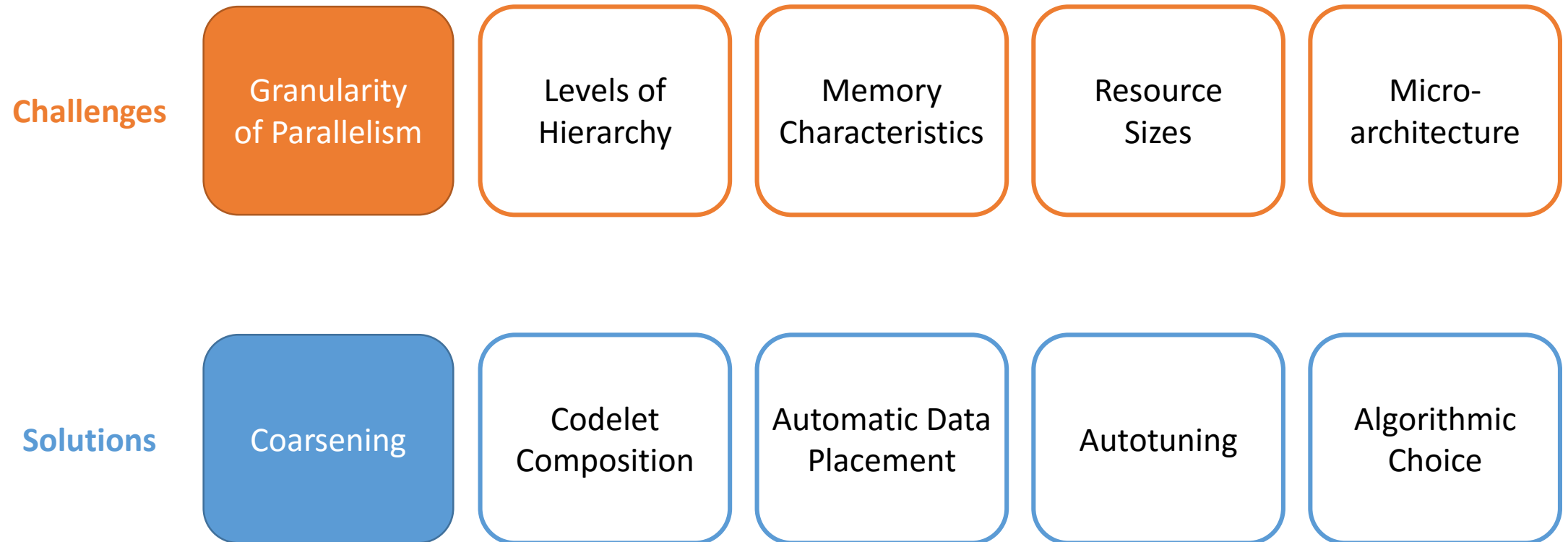
Codelet  
Composition

Automatic Data  
Placement

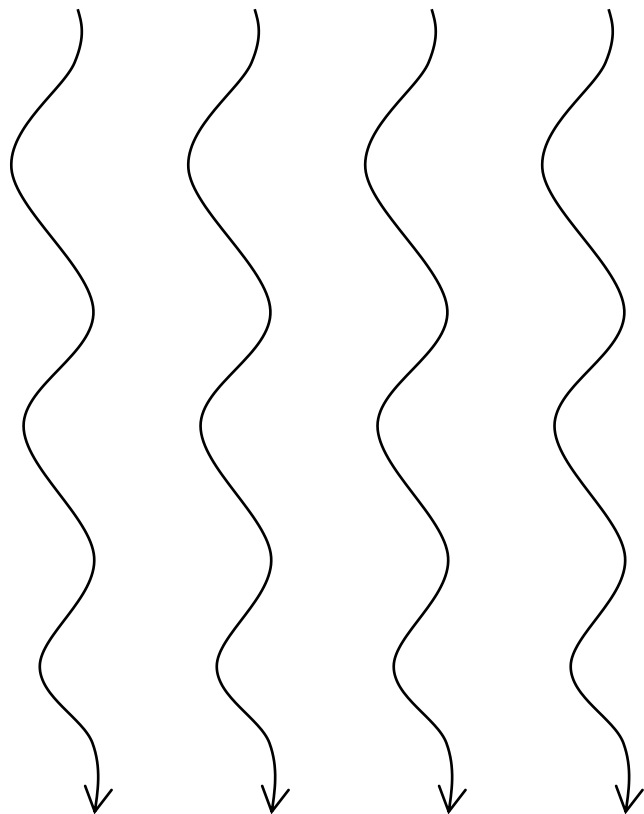
Autotuning

Algorithmic  
Choice

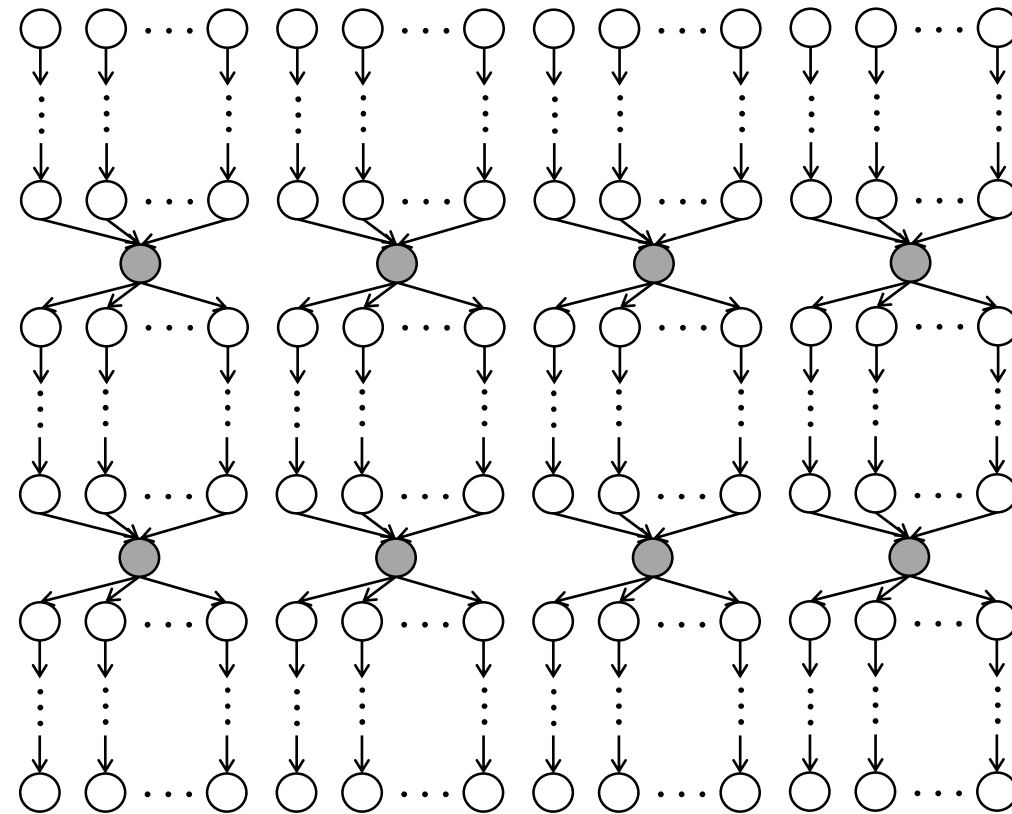
# Performance-Portability: One Source for All



Coarse-grain CPU threads

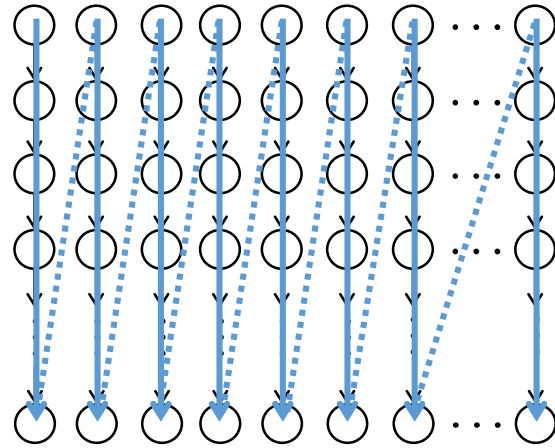


Fine-grain GPU threads

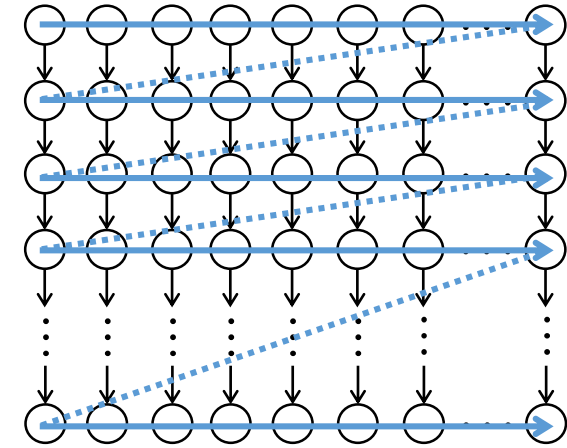


# Coarsening Scheduling Alternatives

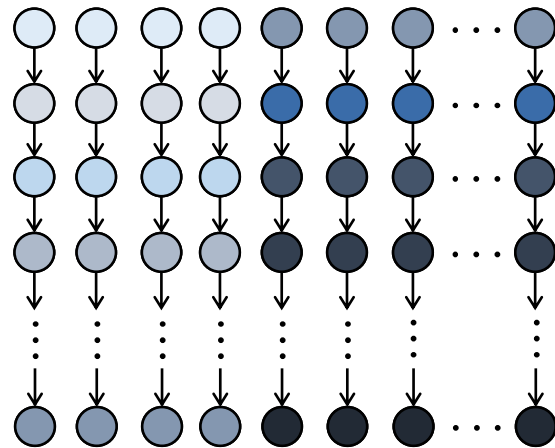
**Depth First Order (DFO) Scheduling**



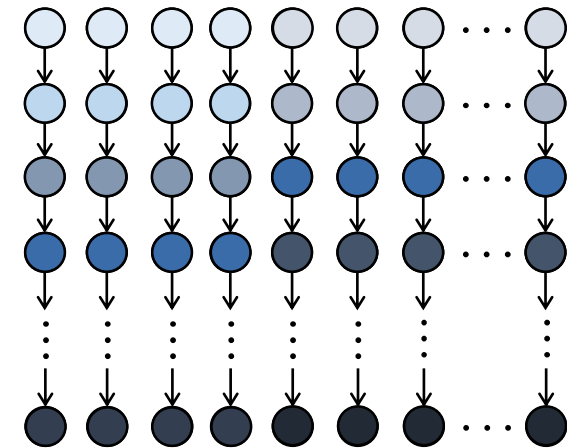
**Breadth First Order (BFO) Scheduling**



**DFO Scheduling with Vectorization**  
(time progresses as color gets darker)



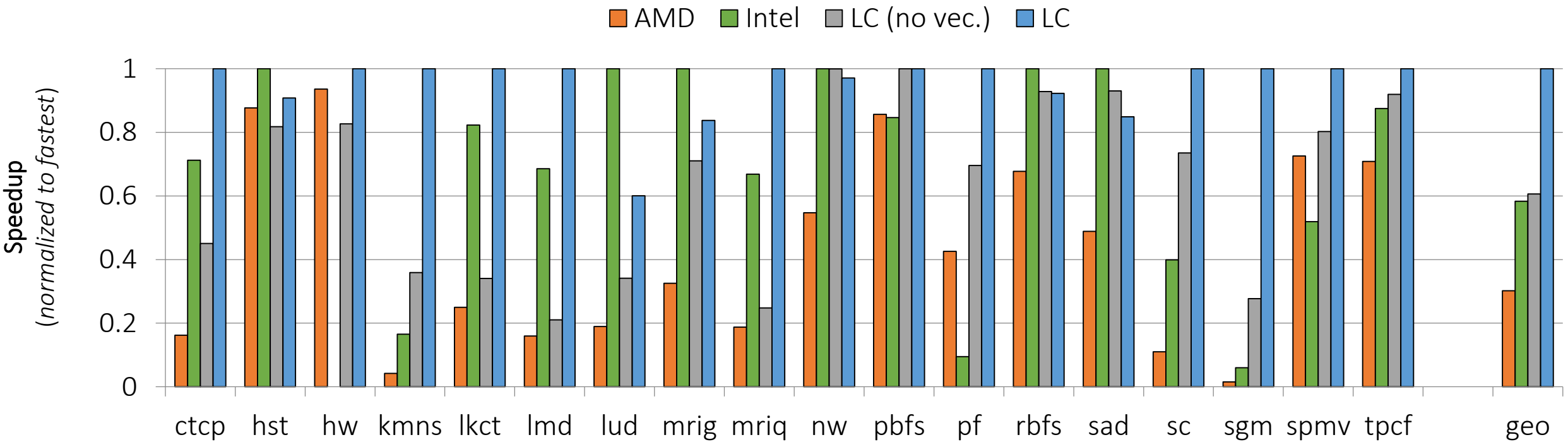
**BFO with Vectorization**  
(time progresses as color gets darker)



# OpenCL/CUDA to CPU Compilers

	Basic Coarsening (DFO)	Vectorization	Locality-aware Scheduling (DFO vs. BFO)
AMD	No	No	No
MCUDA	Yes	No	No
SnuCL	Yes	No	No
Karrenberg & Hack	Yes	Yes	No
pocl	Yes	Yes	No
Intel	Yes	Yes	No
MxPA	Yes	Yes	Yes

# Performance Results

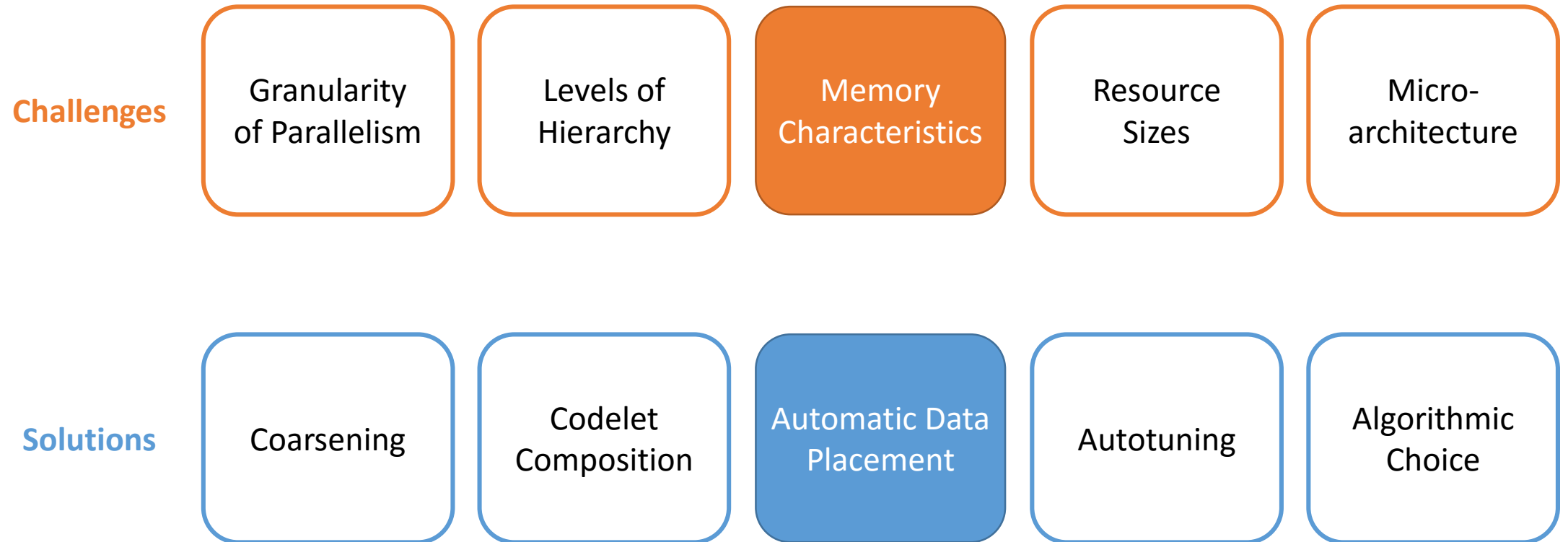


Speedups of 3.32x and 1.71x over AMD and Intel OpenCL implementations

Kim et al., CGO'15



# Performance-Portability: One Source for All





# Data Placement Options

## CPU

- Global memory
- Caches (data tiling)
- Registers

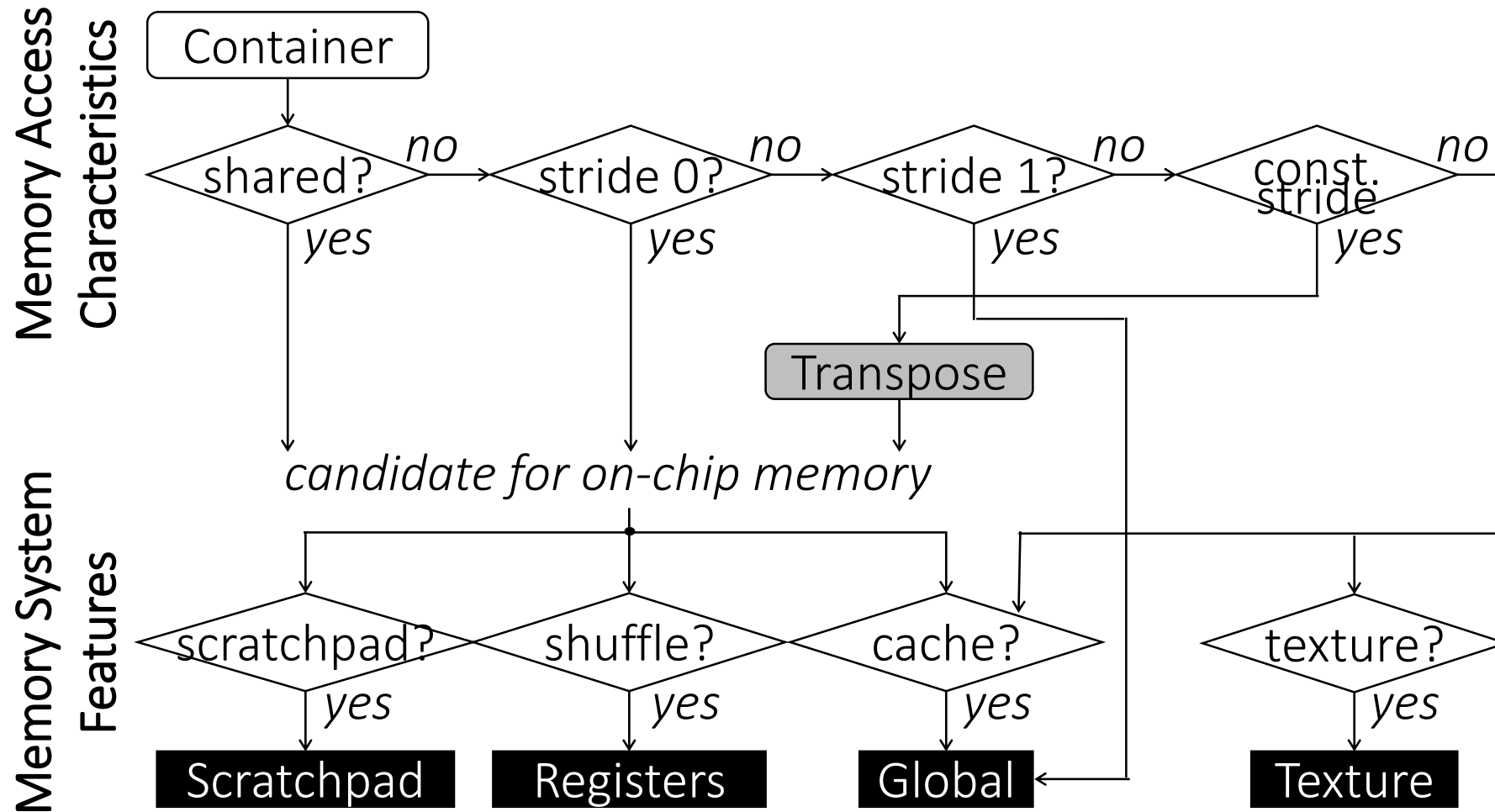
## GPU

- Global memory
  - Caches (data tiling)
  - Registers
- +
- Scratchpad memory
  - Constant memory
  - Texture memory

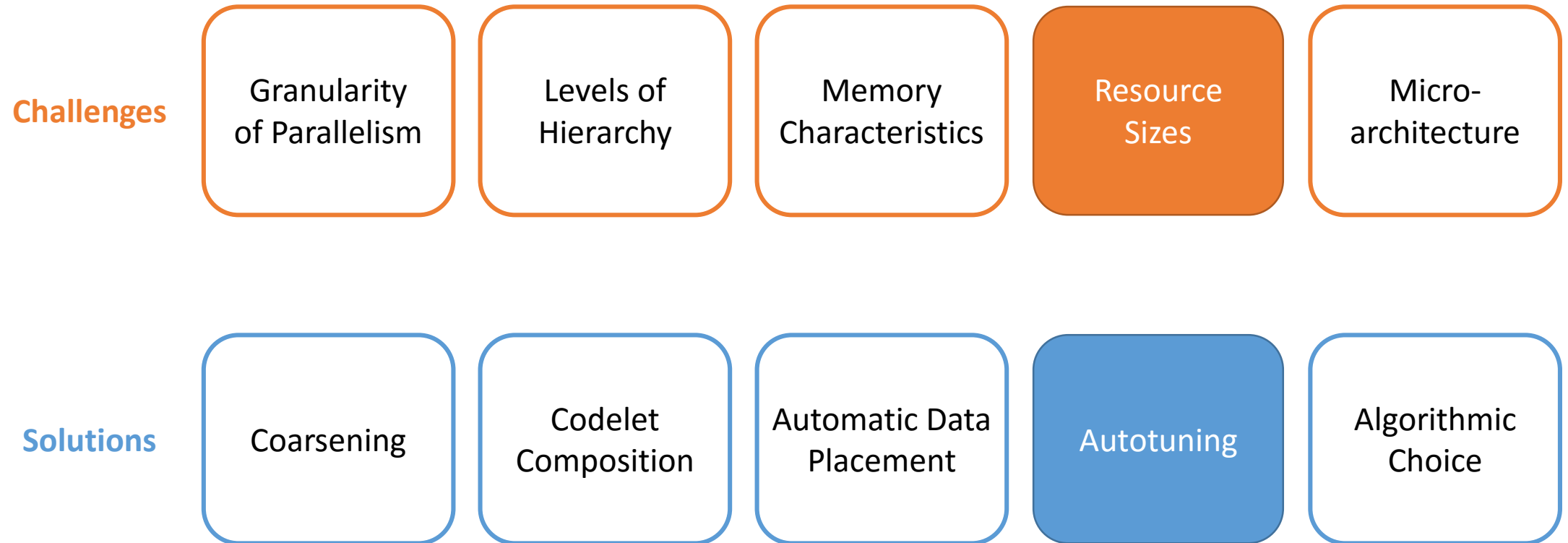
# Rule-based vs. Model-based

- Rule-based (e.g., Jang et al.)
  - Heuristics on the memory access pattern
- Model-based (e.g., PORPLE)
  - Create a model the memory subsystem
  - Slower but more accurate

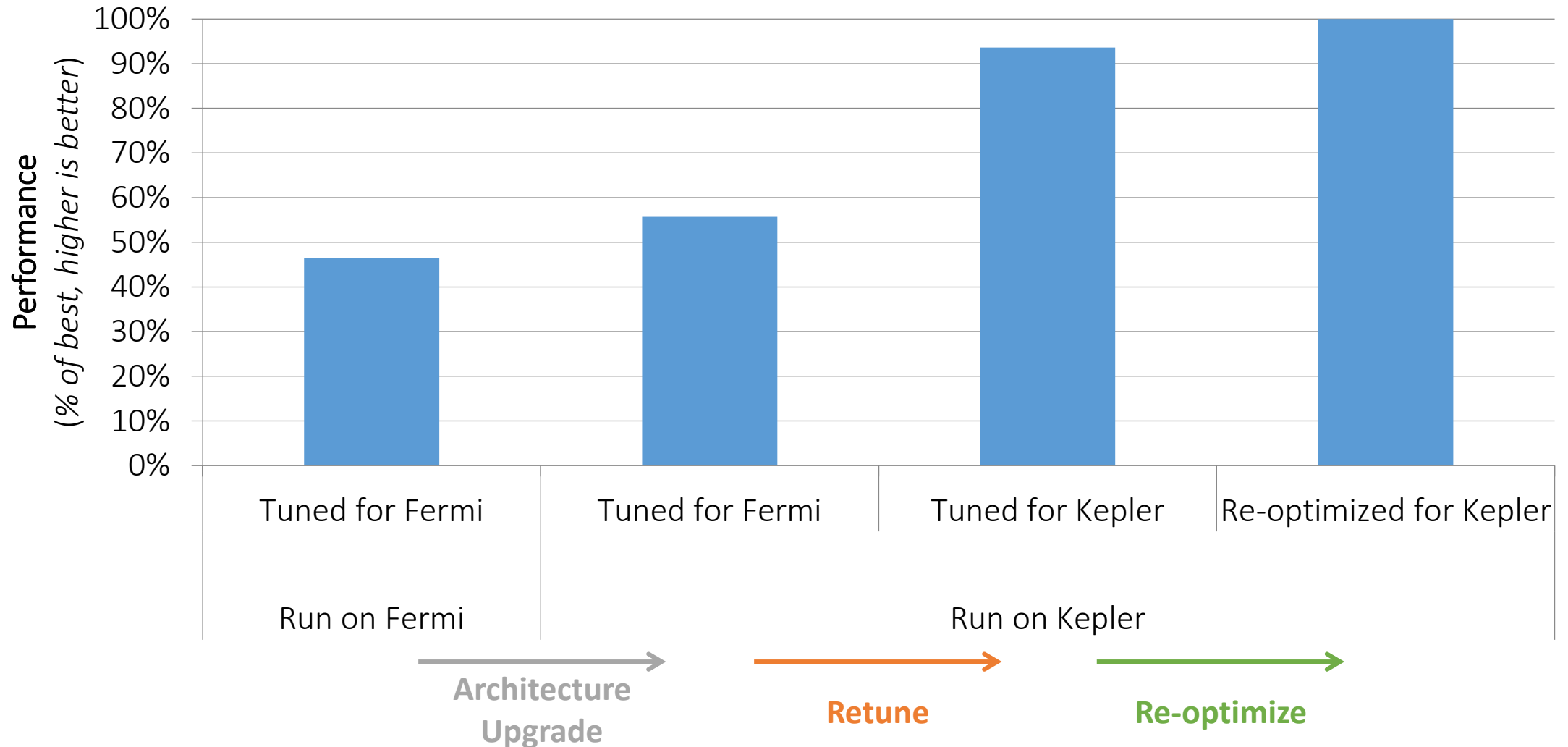
# Tangram's Rule-based Data Placement



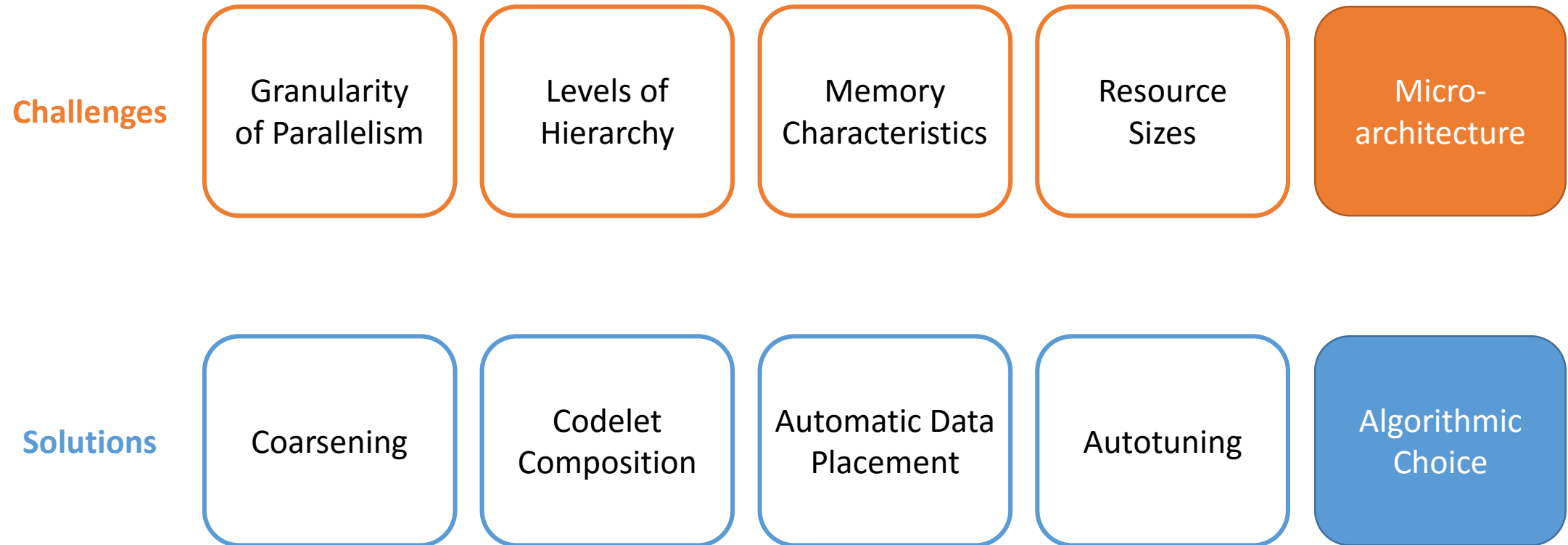
# Performance-Portability: One Source for All



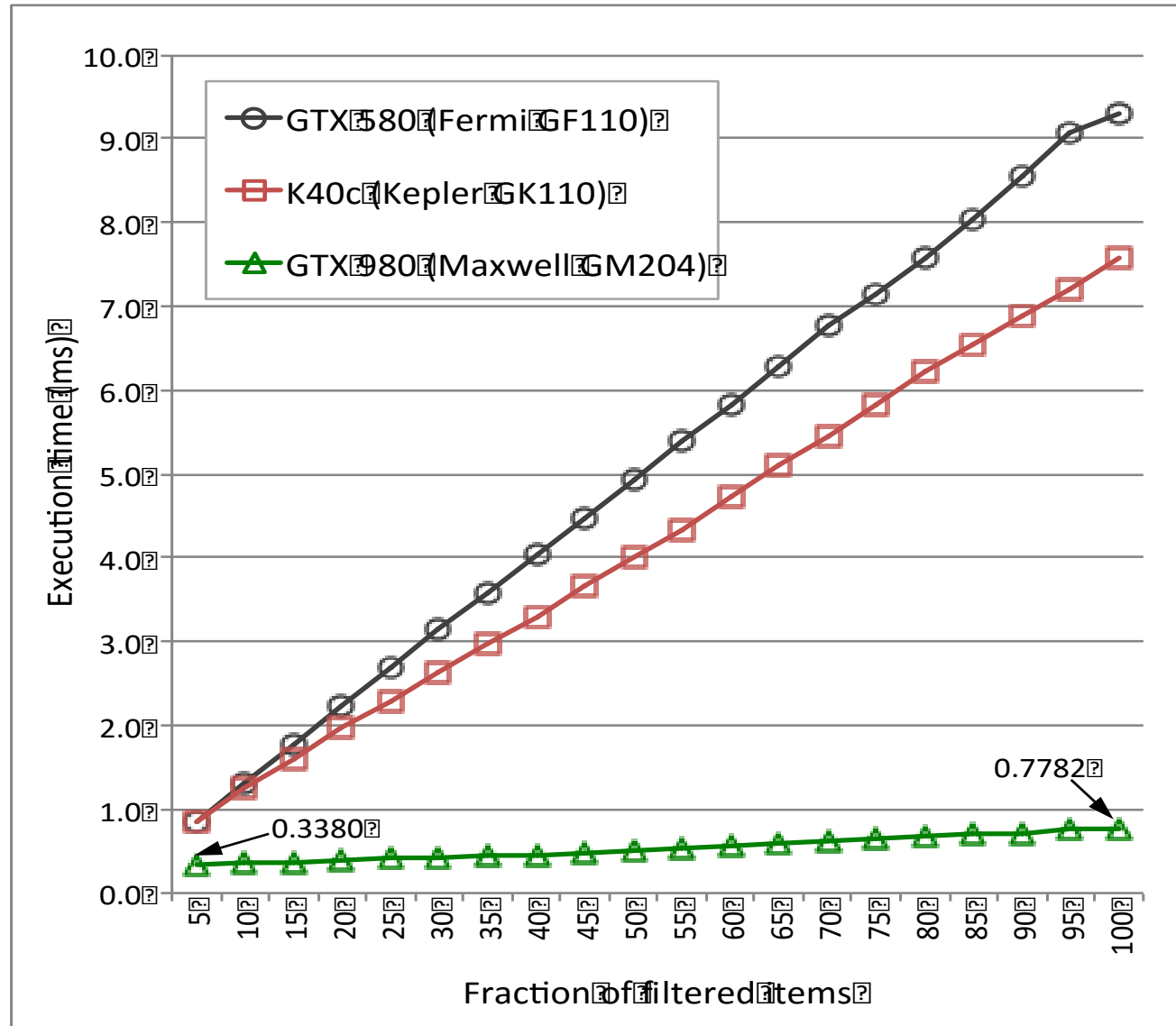
# GPU Tuning: Scan Case Study



# Performance-Portability: One Source for All



# Scratchpad atomics performance (stream compaction)

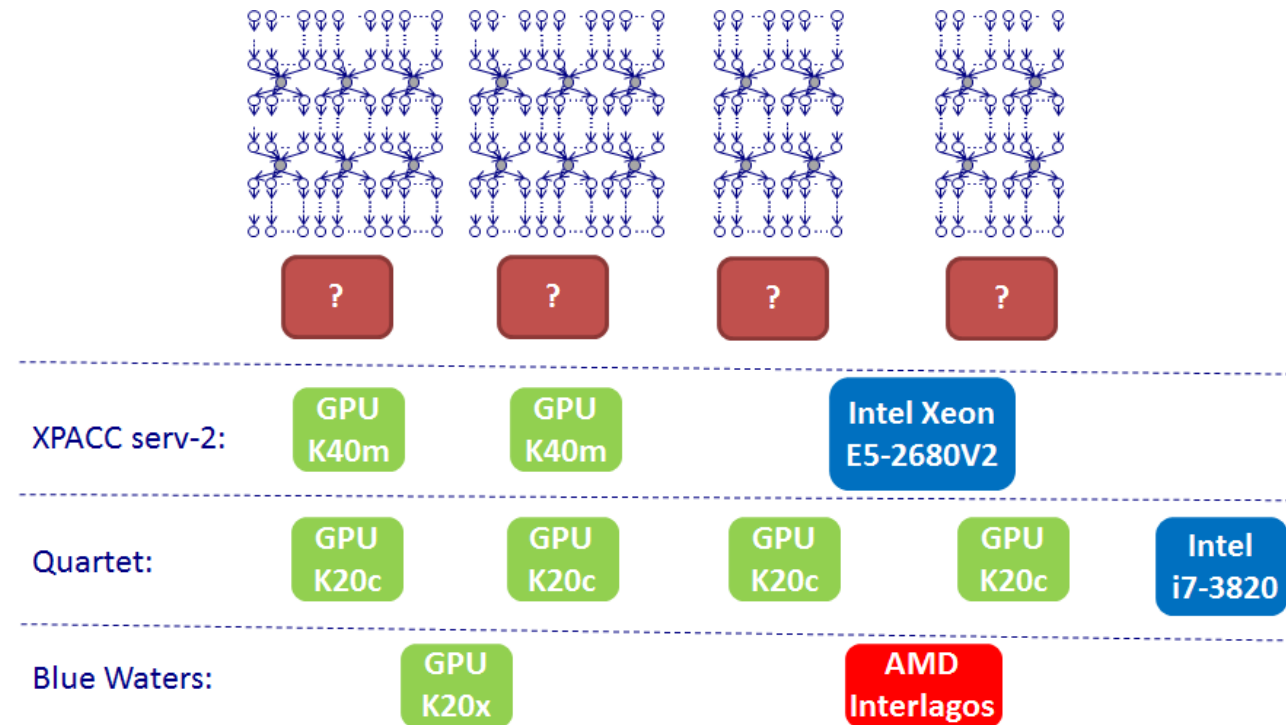


# Motivation Backup



# XPACC: THE CENTER FOR EXASCALE SIMULATION OF PLASMA-COUPLED COMBUSTION

- Codesign among diverse areas will be required to reach exascale
  - Every level of the computational stack is a potential bottleneck.
- XPACC code will need to run efficiently and portably on next-generation heterogeneous platforms (CPUs, GPUs, Xeon-Phis)



# Initial Production Use Results

- NAMD

- 100 million atom benchmark with Langevin dynamics and PME once every 4 steps, from launch to finish, all I/O included
- 768 nodes, Kepler+Interlagos is 3.9X faster over Interlagos-only
- 768 nodes, XK7 is 1.8X XE6

- Chroma

- Lattice QCD parameters: grid size of 483 x 512 running at the physical values of the quark masses
- 768 nodes, Kepler+Interlagos is 4.9X faster over Interlagos-only
- 768 nodes, XK7 is 2.4X XE6

- QMCPACK

- Full run Graphite 4x4x1 (256 electrons), QMC followed by VMC
- 700 nodes, Kepler+Interlagos is 4.9X faster over Interlagos-only
- 700 nodes, XK7 is 2.7X XE6

# Blue Waters Science Production Applications

- Work with science teams to effectively use GPUs in their production code.
  - ChaNGa – cosmological simulation, University of Washington
  - AWP – earthquake simulation, Southern California Earthquake Center
- Significant speedup by tuning kernels to specific GPU characteristics
  - Real-world opportunities for performance portability

## GPU Kernel Optimizations

		Running Time (ms)	Speedup
ChaNGa	Baseline	1.35	2.11
	Optimized	1.16	
AWP	Baseline	61.6	1.33
	Optimized	43.3	

# Levels of GPU Programming Interfaces

## Prototype & in development

X10, Chapel, Nesi, Delite,  
Par4all, Tangram...

Implementation manages GPU threading and synchronization invisibly to user

## Next generation

OpenACC, HCC++, Thrust, Bolt

Simplifies data movement, kernel details and kernel launch  
Same GPU execution model (but less boilerplate)

## Current generation

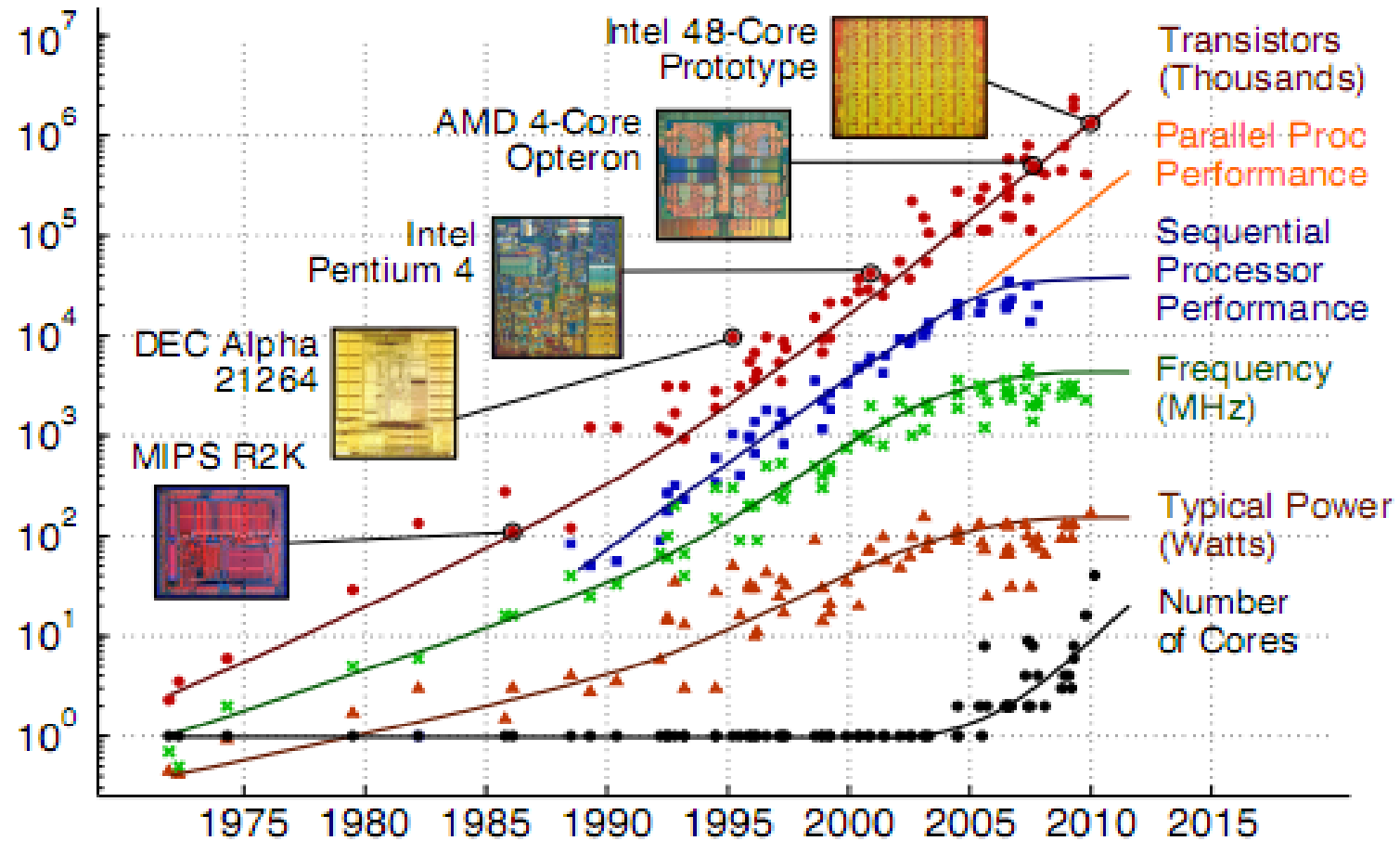
CUDA, OpenCL, DirectCompute

# Portability- CPU vs. GPU Code Versions

- Maintaining multiple code versions is extremely expensive
- Most CUDA/OpenCL developers maintain original CPU version
- Many developers report that when they back ported the CUDA/OpenCL algorithms to CPU, they got better performing code
  - Locality, SIMD, multicore
- MxPA is designed to automate this process (*John Stratton, Hee-Seok Kim, Izzat El Hajj*)

# Performance Library

- A major qualifying factor for new computing platforms
  - MKL, BLAS, CUSPARSE, Trust, FFT, OpenCV, CUDNN, etc.
  - Currently redeveloped and hand-tuned for each HW type/generation
- Exa-scale HW expected to have increasing levels of heterogeneity, parallelism, and hierarchy
  - Increasing levels of memory heterogeneity and hierarchy
  - Increase SIMD width and types/number of cores
- Performance library development process must keep up with the HW evolution and diversification
  - Performance portability



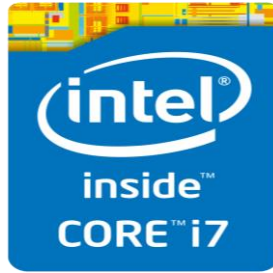
Data partially collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond

Prepared by C. Batten - School of Electrical and Computer Engineering - Cornell University - 2005 - retrieved Dec 12 2012 - <http://www.csl.cornell.edu/courses/ece5950/handouts/ece5950-overview.pdf>

1 core

4 cores

6 cores



2003

2006

2010

2005



2 cores



1 core



2003

4 cores



2006

SoC (1 core)



2008

6 cores



2010

SoC (2 cores)



many-core



2012


SoC (6 cores)



2014



2005




2 cores

2007

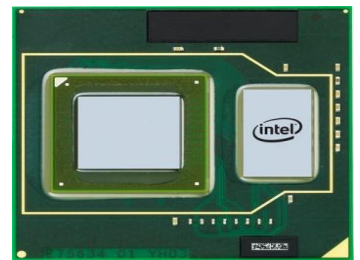


many-core

2010



many-core



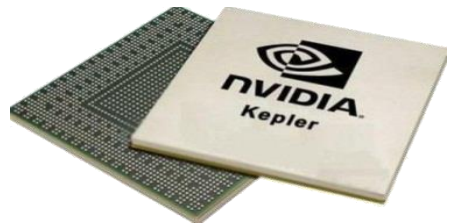
Stellarton  
CPU+FPGA

2011



APU (1<sup>st</sup> gen)

2012



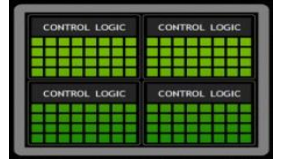
many-core

ACCELERATED  
A-SERIES  
PROCESSOR

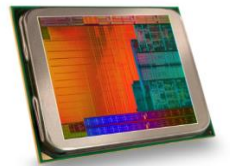


APU (2<sup>nd</sup> gen)

2014



NVIDIA  
Maxwell  
many-core

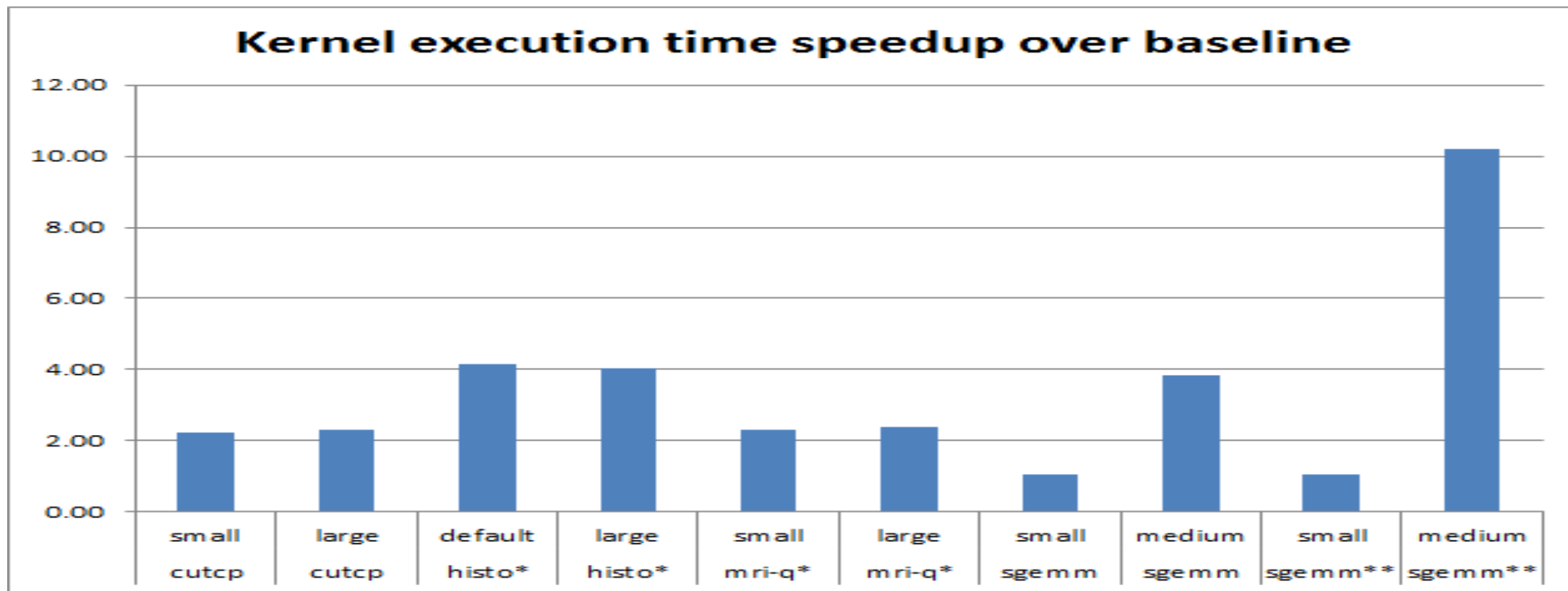


Kaveri  
APU (3<sup>rd</sup> gen)

# Portability Backup

# Granularity Tuning (OpenCL)

Results of thread coarsening for Parboil benchmarks(written for NVIDIA SIMT GPUs) on AMD Radeon HD6990 (VLIW-5)



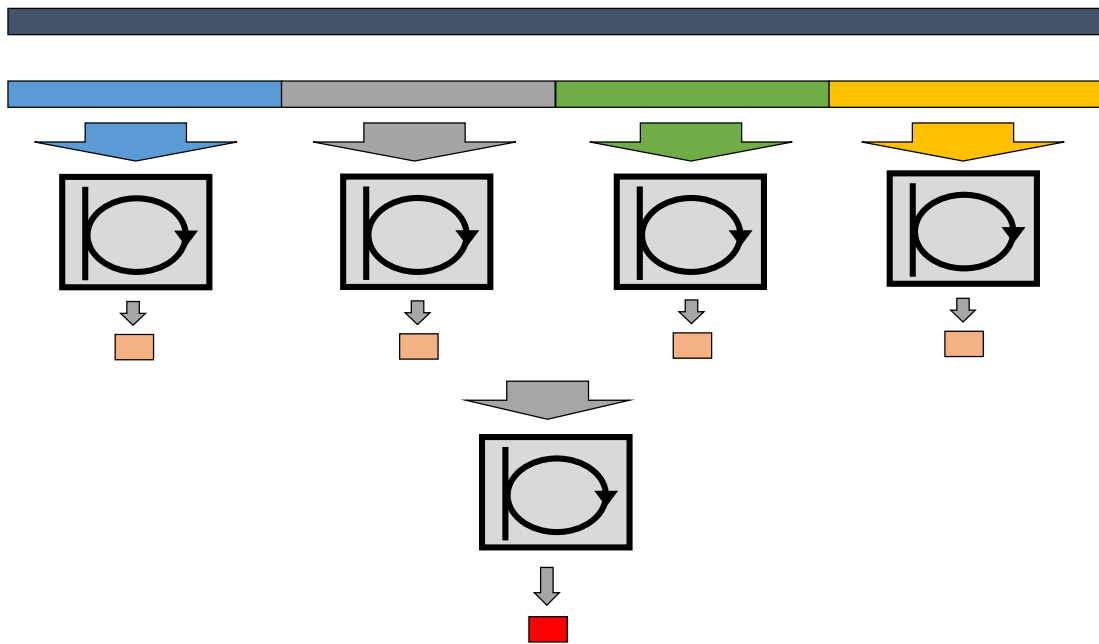
\* Not a single kernel

\*\* Results from more than one dimension coarsening

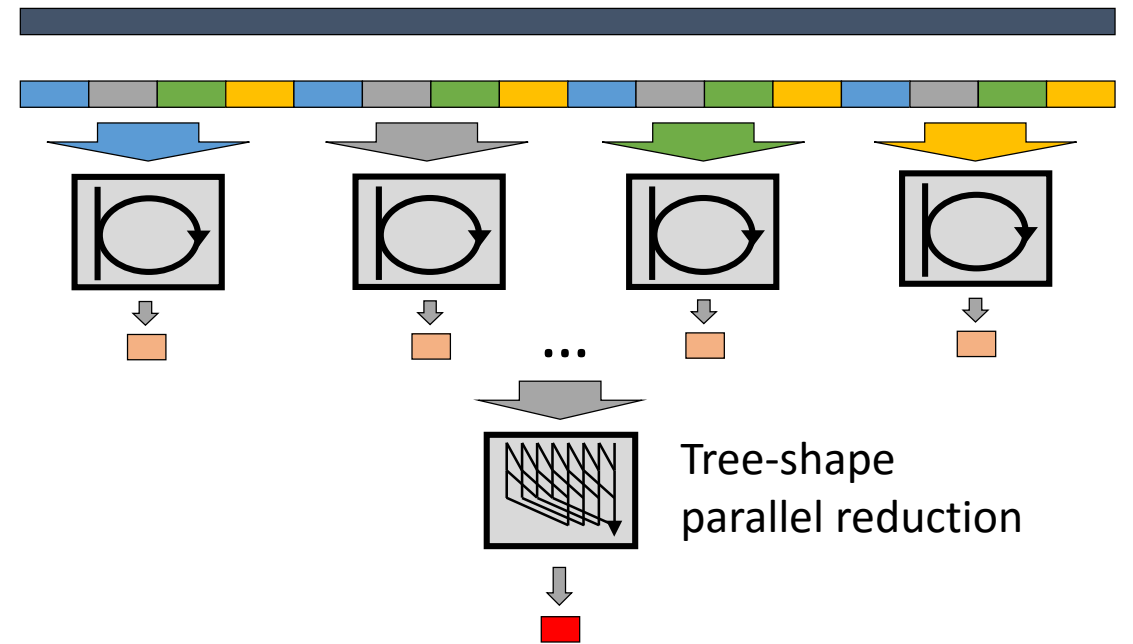
Results compiled using MulticoreWare's SlotMaximizer

- Reduction – CPU vs. GPU (Part 1)

CPUs favor intra-thread locality

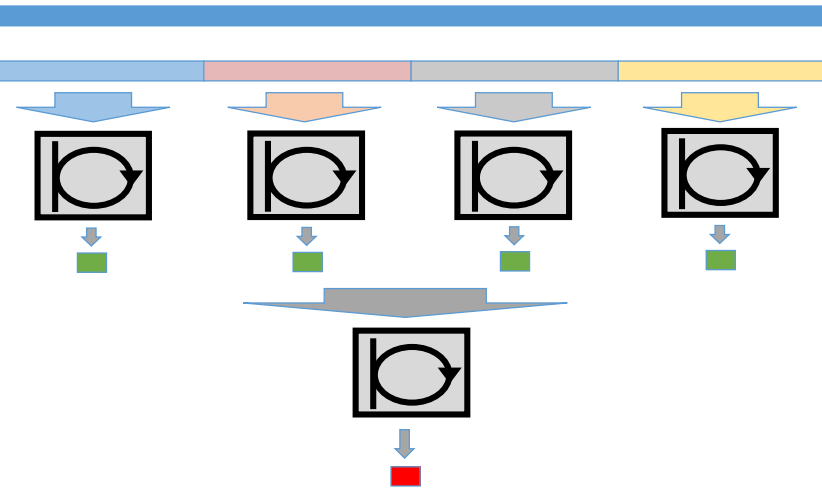


GPUs favor inter-thread locality (within Work Groups)

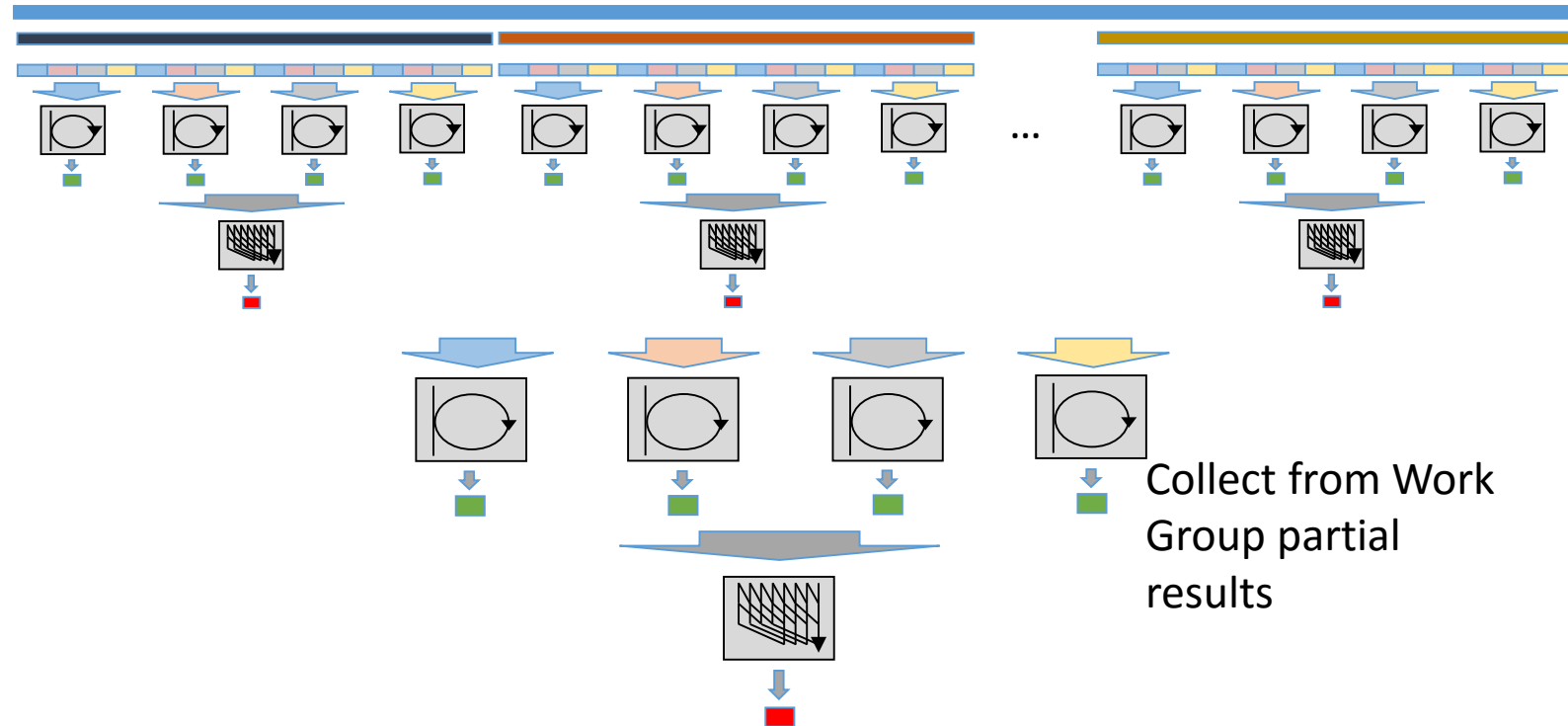


- Reduction – CPU vs. GPU (Part 2)

CPU 2-level hierarchy

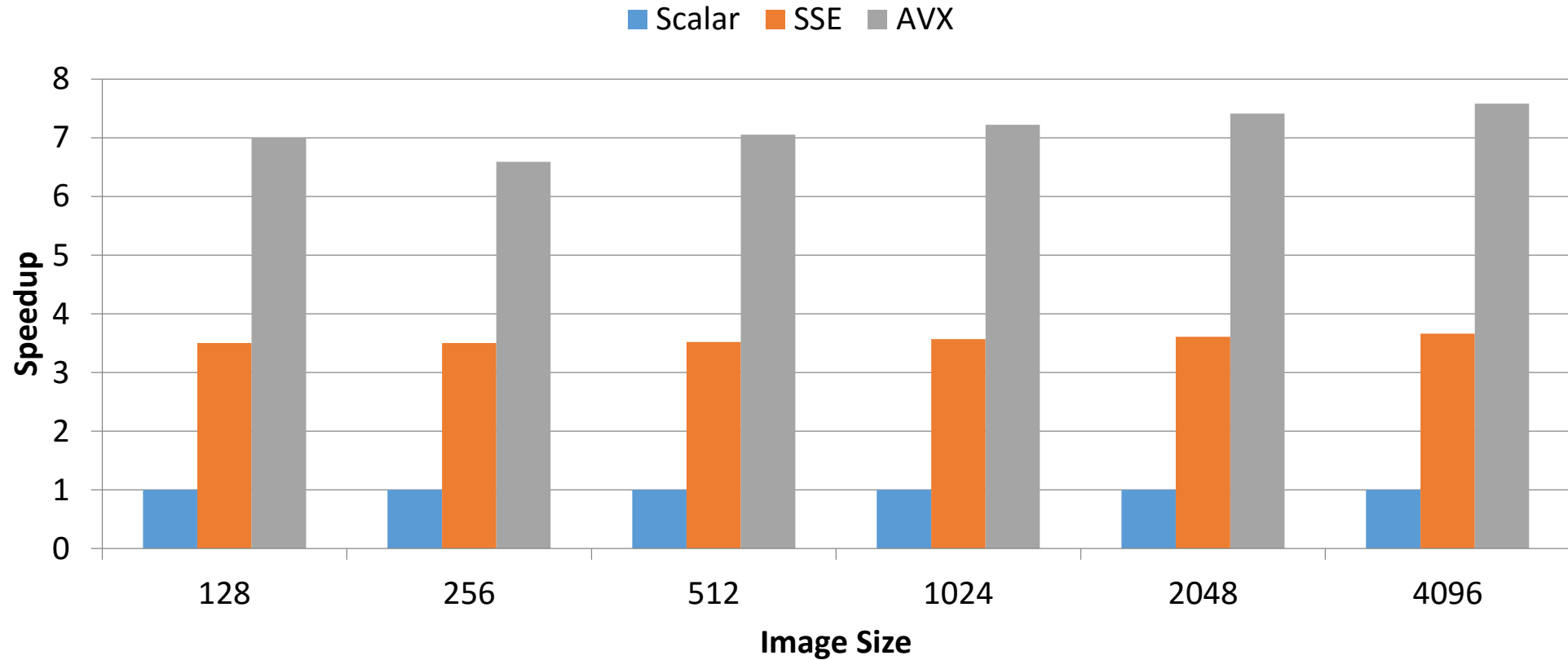


GPU 4-level hierarchy



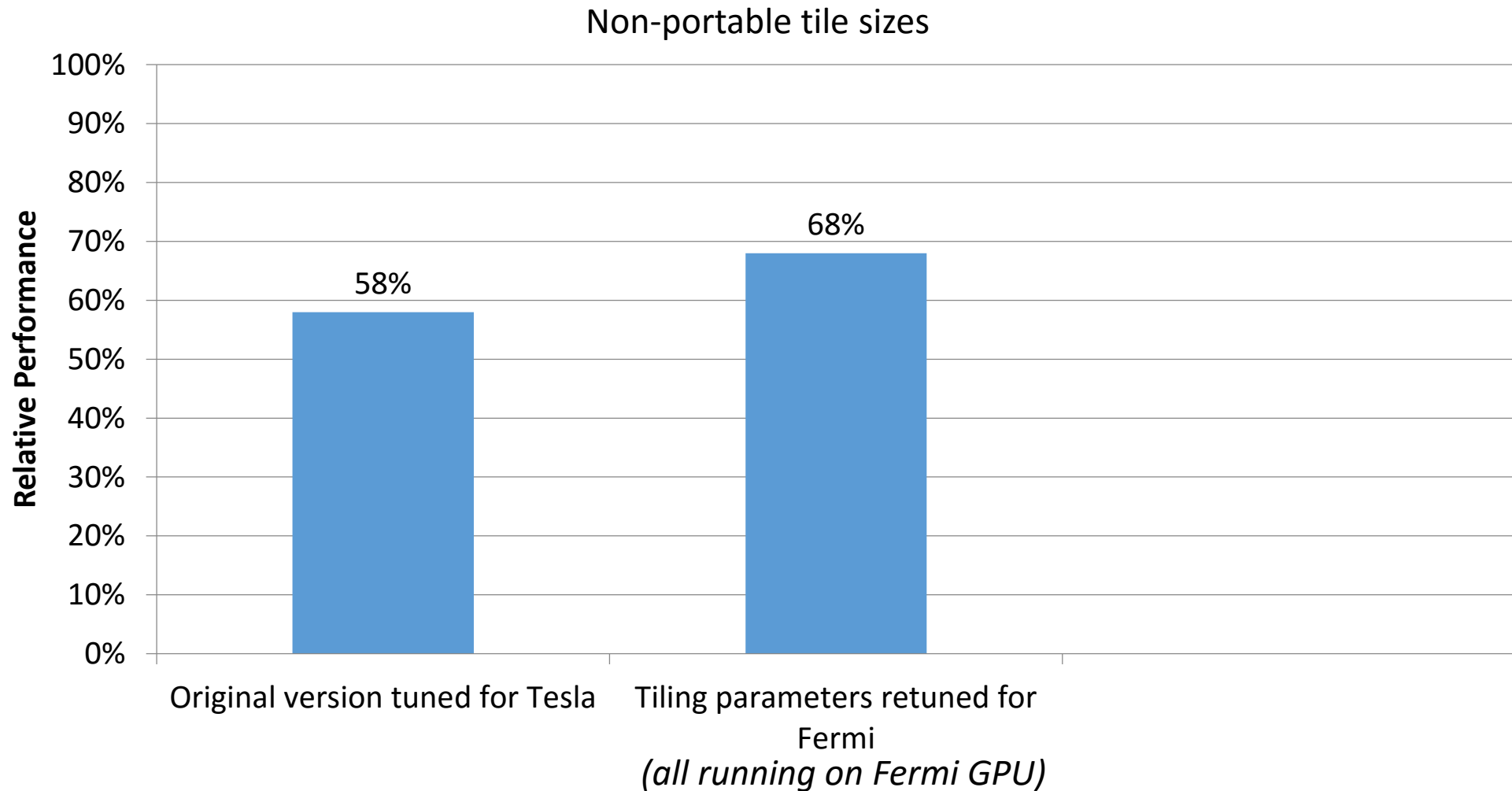
- CPU Parameter Tuning

Mandelbrot performance with vector width

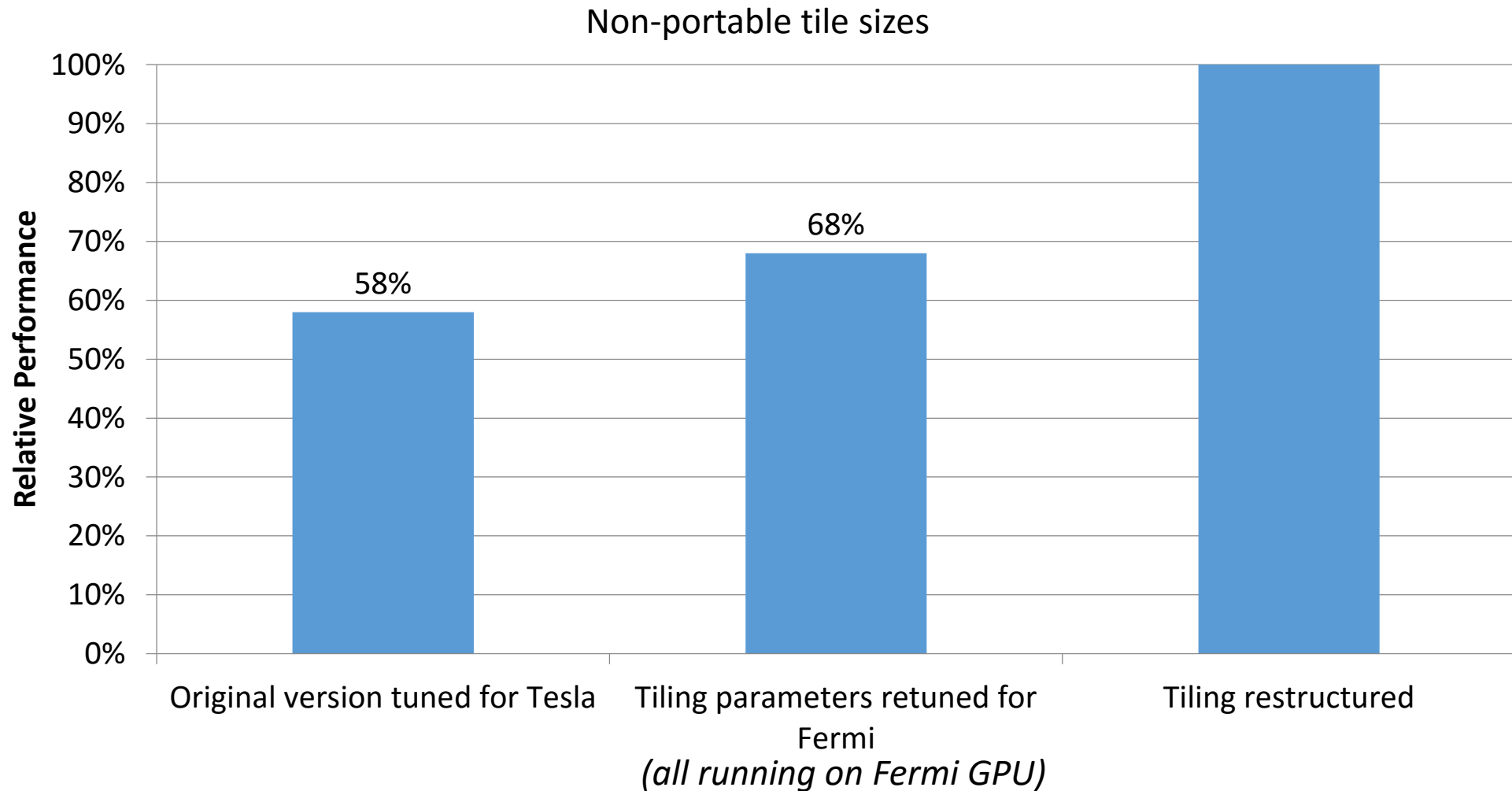


Results courtesy of intel.com

# GPU Parameter Tuning



# GPU Parameter Tuning





# Bitonic Sort

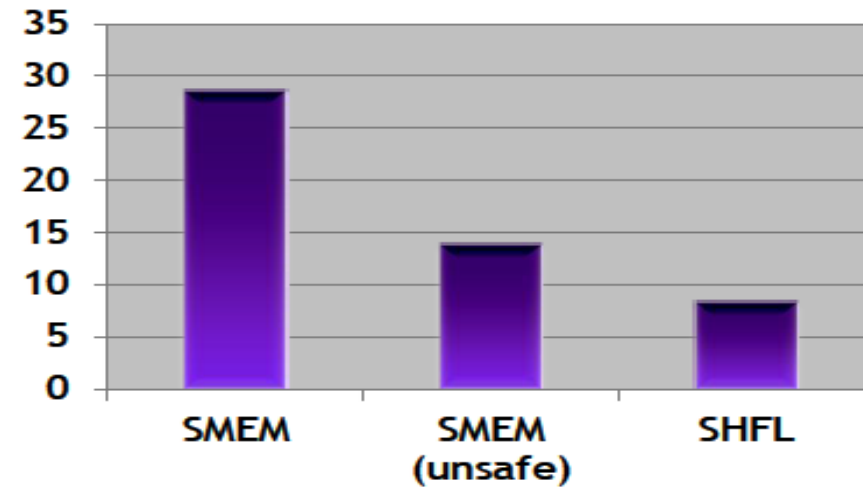
```
int swap(int x, int mask, int dir)
{
    int y = __shfl_xor(x, mask);
    return x < y == dir ? y : x;
}
```

```
x = swap(x, 0x01, bfe(laneid, 1) ^ bfe(laneid, 0)); // 2
x = swap(x, 0x02, bfe(laneid, 2) ^ bfe(laneid, 1)); // 4
x = swap(x, 0x01, bfe(laneid, 2) ^ bfe(laneid, 0));
x = swap(x, 0x04, bfe(laneid, 3) ^ bfe(laneid, 2)); // 8
x = swap(x, 0x02, bfe(laneid, 3) ^ bfe(laneid, 1));
x = swap(x, 0x01, bfe(laneid, 3) ^ bfe(laneid, 0));
x = swap(x, 0x08, bfe(laneid, 4) ^ bfe(laneid, 3)); // 16
x = swap(x, 0x04, bfe(laneid, 4) ^ bfe(laneid, 2));
x = swap(x, 0x02, bfe(laneid, 4) ^ bfe(laneid, 1));
x = swap(x, 0x01, bfe(laneid, 4) ^ bfe(laneid, 0));
x = swap(x, 0x10, bfe(laneid, 4) ^ bfe(laneid, 4)); // 32
x = swap(x, 0x08, bfe(laneid, 3));
x = swap(x, 0x04, bfe(laneid, 2));
x = swap(x, 0x02, bfe(laneid, 1));
x = swap(x, 0x01, bfe(laneid, 0));
```

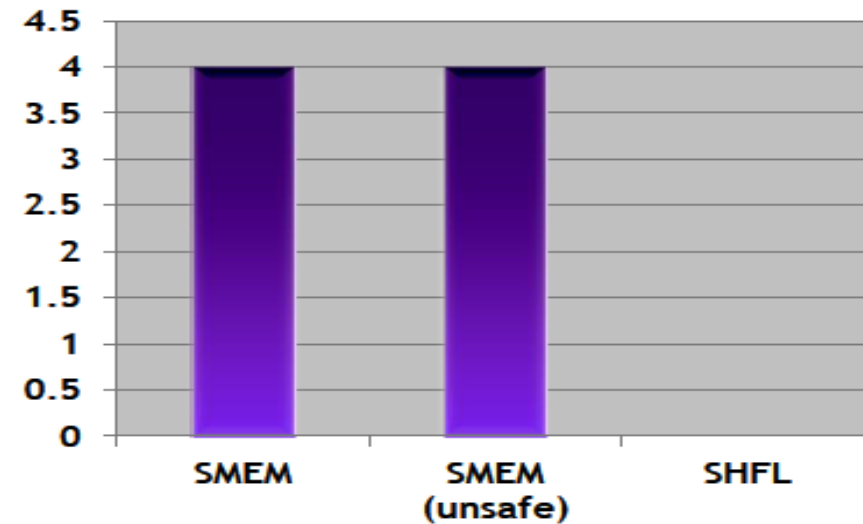
```
// int bfe(int i, int k): Extract k-th bit from i
```

```
// PTX: bfe dst, src, start, len (see p.81, ptx_isa_3.1)
```

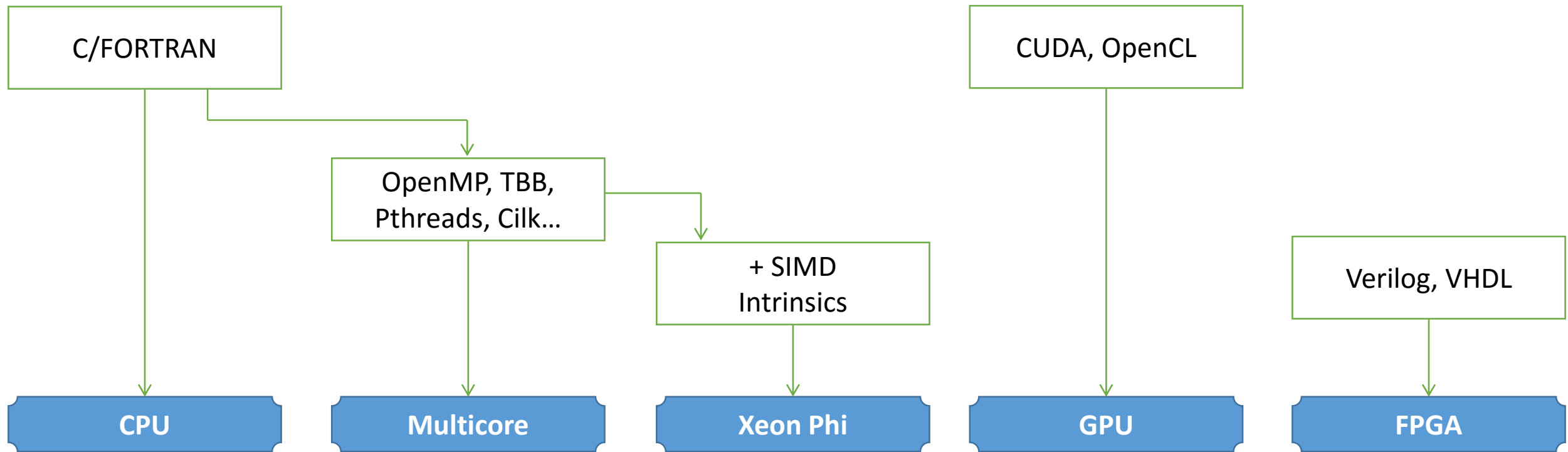
Execution Time int32 (ms)

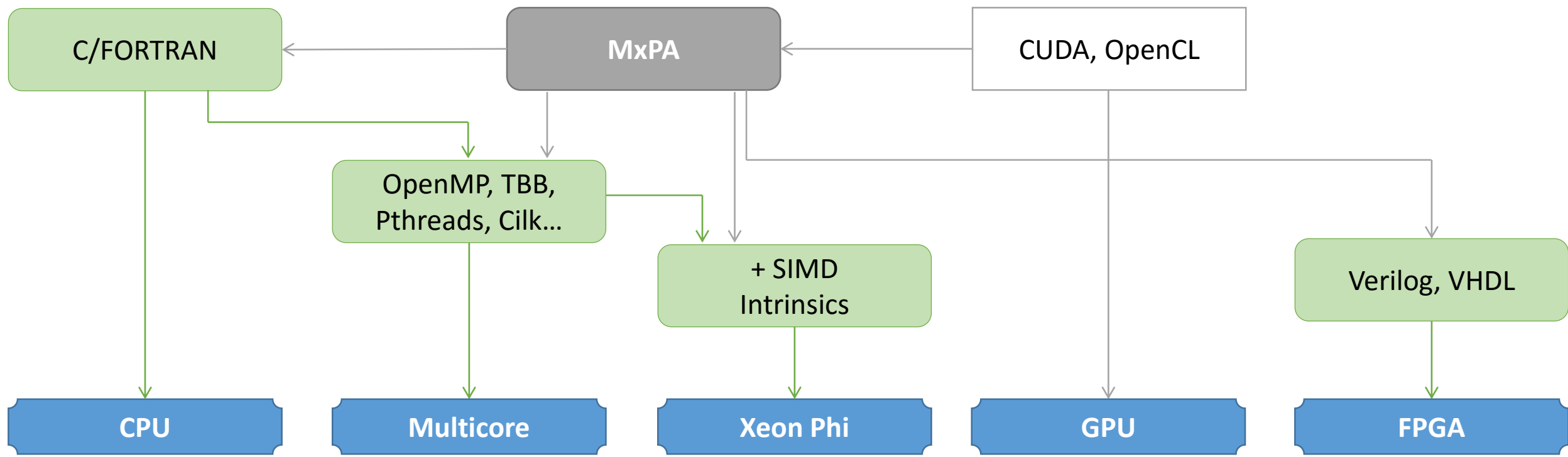


SMEM per Block (KB)

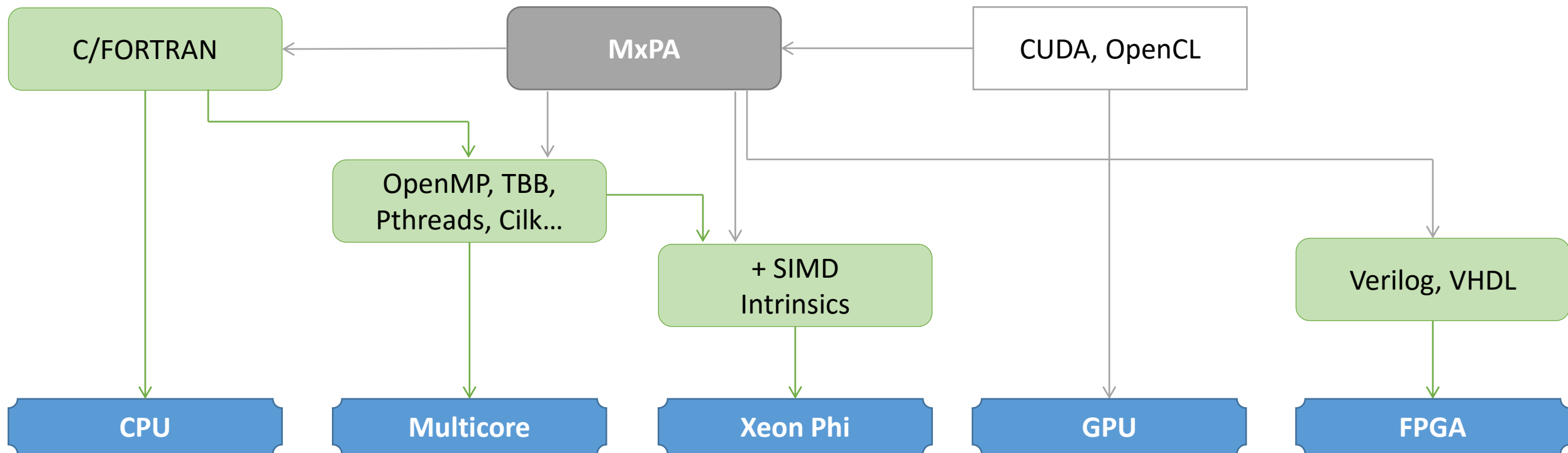


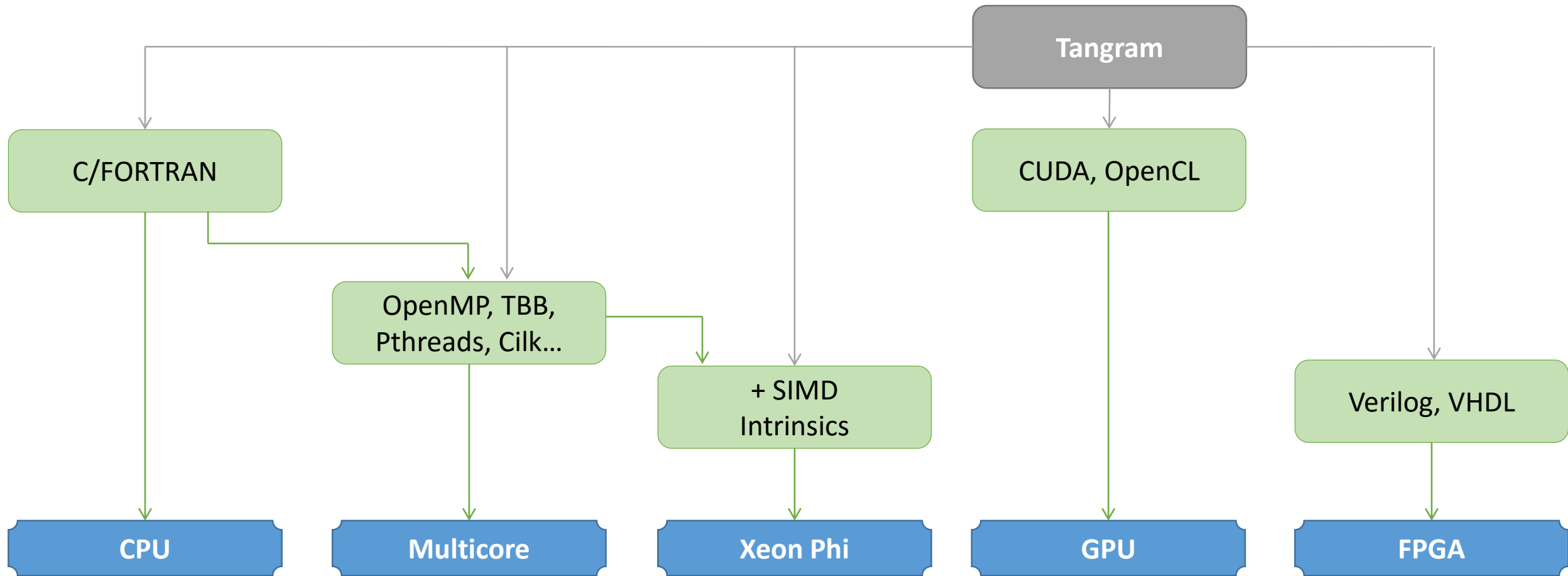
Slide courtesy of nvidia.com





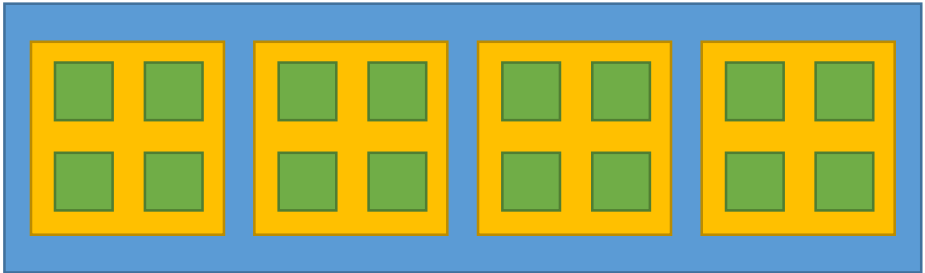
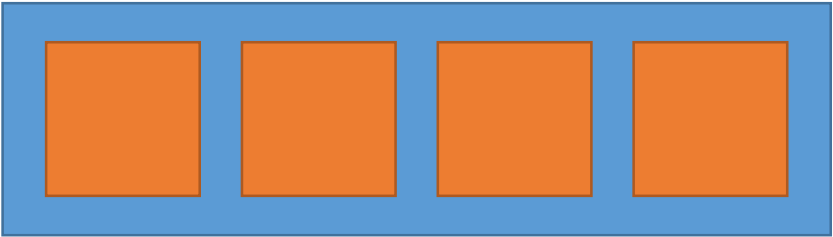
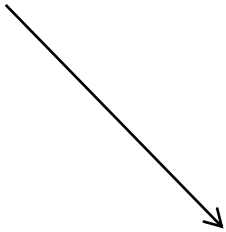
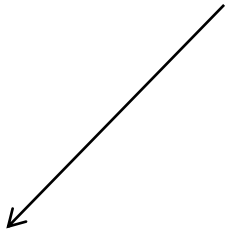
- Locality-centric work-item scheduling
- Speedups of 3.32x and 1.71x over AMD and Intel OpenCL implementations





# Tangram Backup

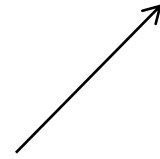
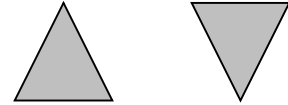
Devices have different architectural hierarchies



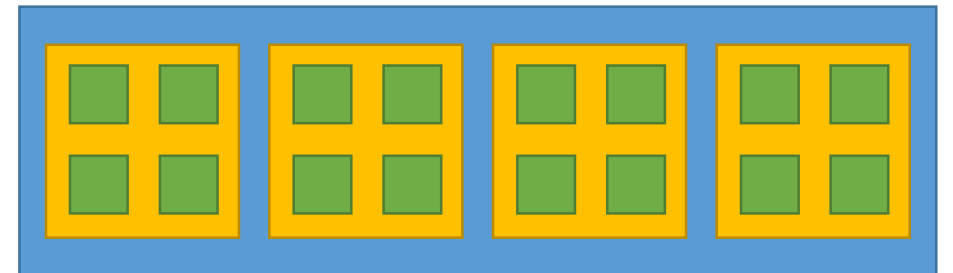
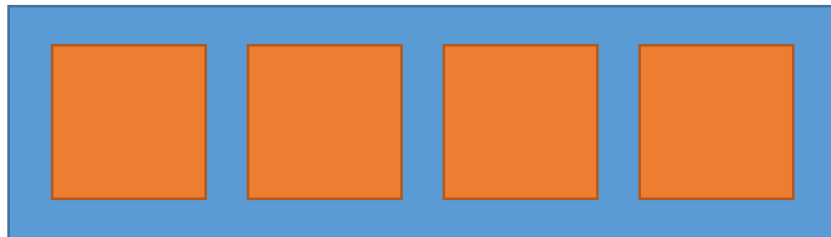
Computation Codelets



Decomposition Codelets



Programmer writes  
architecture-neutral  
computations and  
decomposition rules

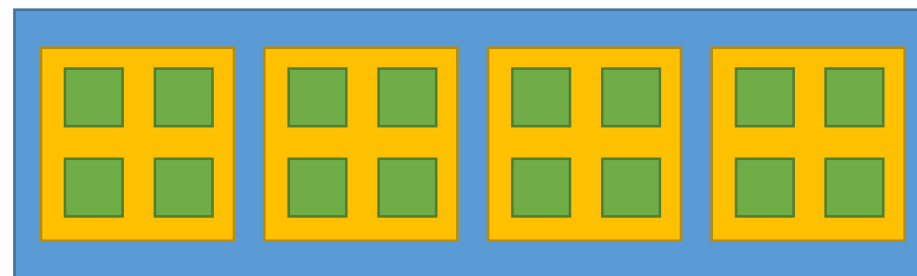
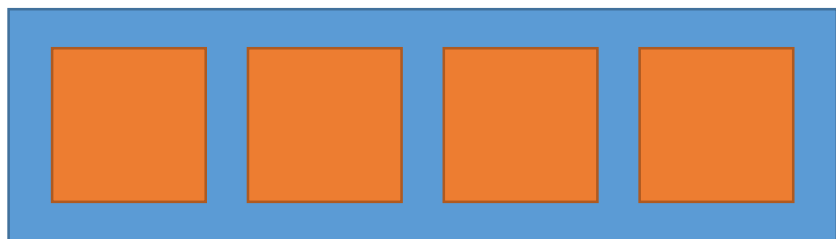
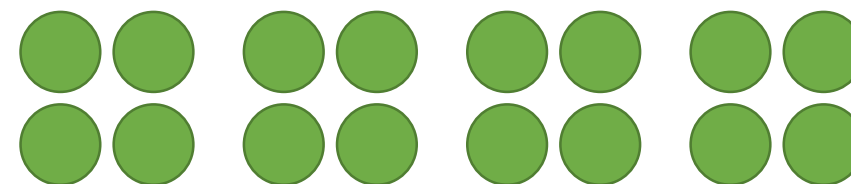
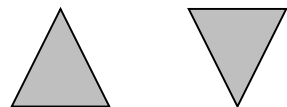




Computation Codelets



Decomposition Codelets

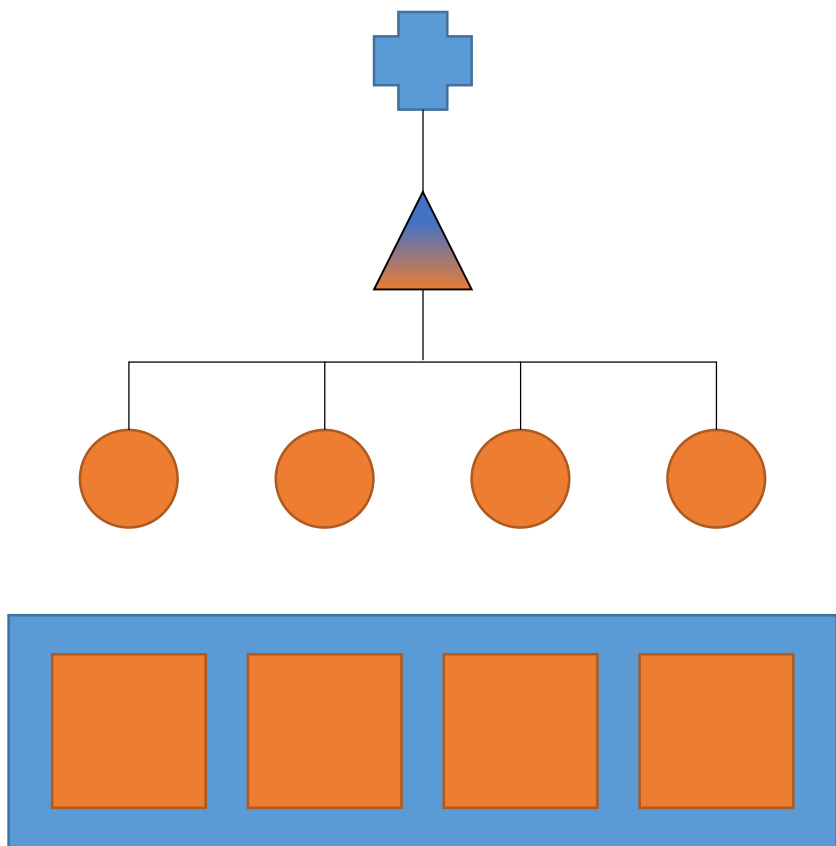
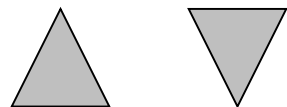


Compiler maps computations to each level of the hierarchy...

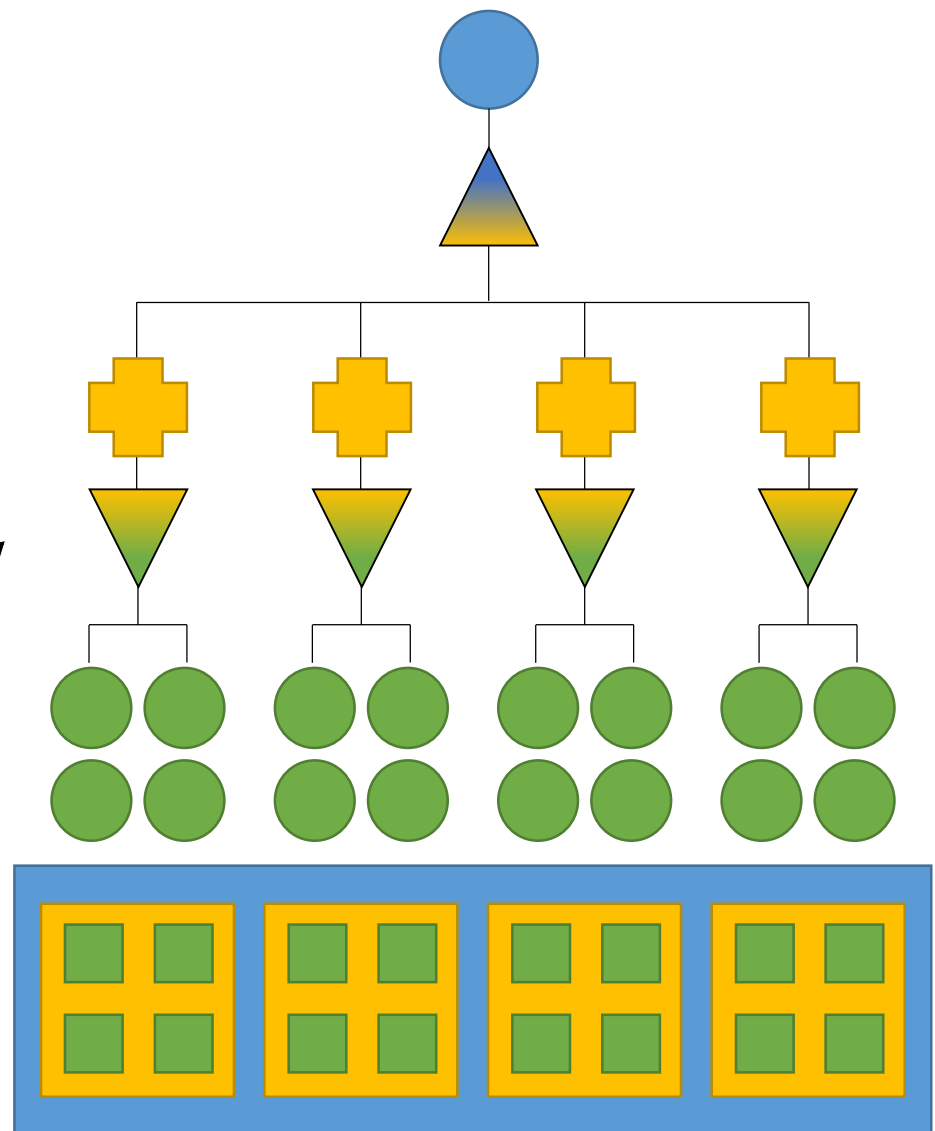
Computation Codelets



Decomposition Codelets



...and  
decomposition  
rules between  
each level



# DySel Backup



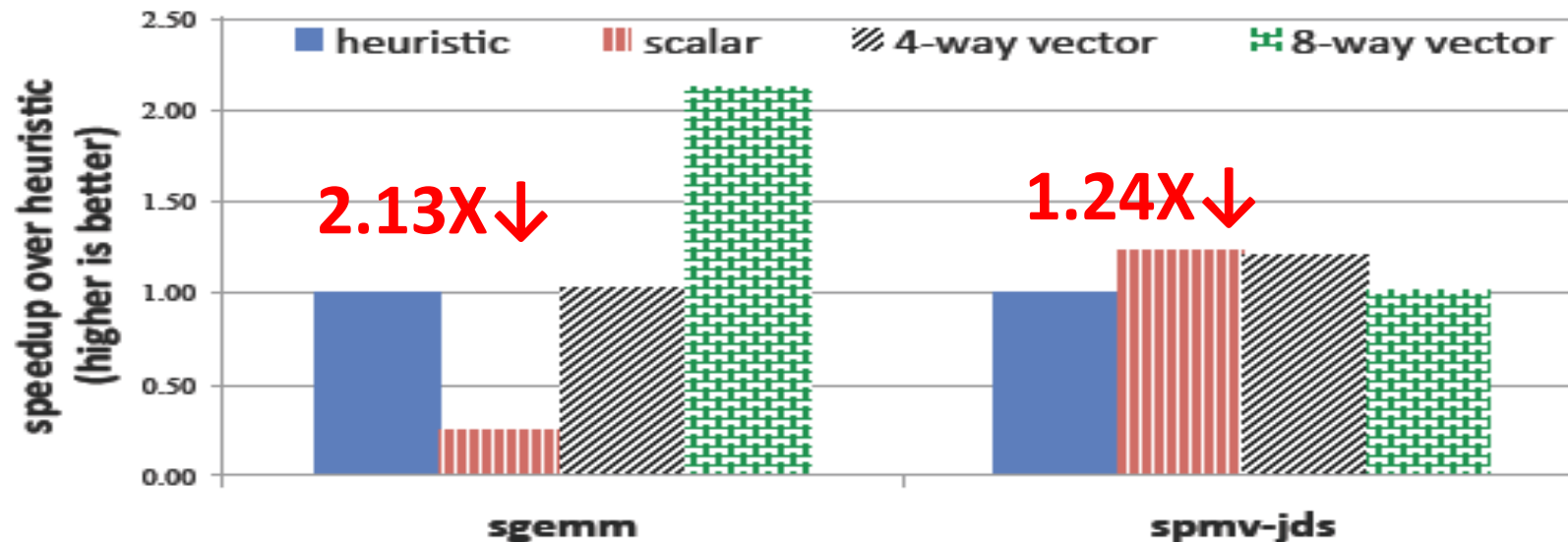
- Pronounced as diesel/'di:zəl/
- Imply low-cost and high-efficiency
  - Diesel was cheaper than regular gas, when we submitted the paper... :v
- A small but useful tool to save compiler optimization developers

# Motivation

- Statically determining the optimal code could be default or even infeasible
  - Sometimes it is input dependent
- Even a robust compiler or an expert could select suboptimal sequence of optimization
  - A catastrophic performance loss could happen

# Example: Intel OpenCL Vectorization for CPU

- Suboptimal heuristic for vectorization in sgemm and spmv-jds



# Relax the Constraints

- Instead of asking a compiler for an optimized version which it thought is the best
- Ask a compiler for multiple versions which are competitive
  - A typical number is around 4-10
  - Let the runtime to do the final selection

# Version Selection on Runtime

- We propose **DySel** for dynamic version selection on runtime
- Apply *micro-profiling* to sample the performance of each candidate

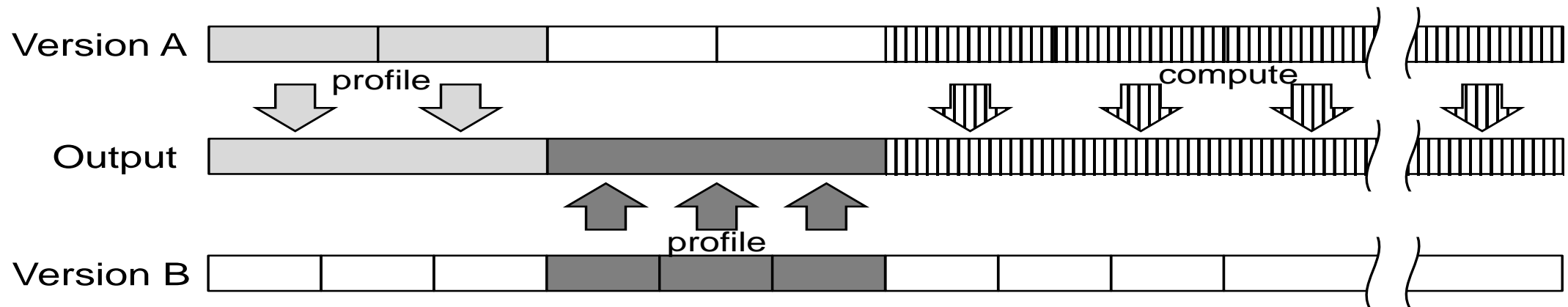


# Micro-Profiling

- Profile a kernel with smaller workload
  - A smaller number of work-group/thread block
  - Avoid large impact of performance
- Multiple micro-profiling can be scheduled and even executed concurrently

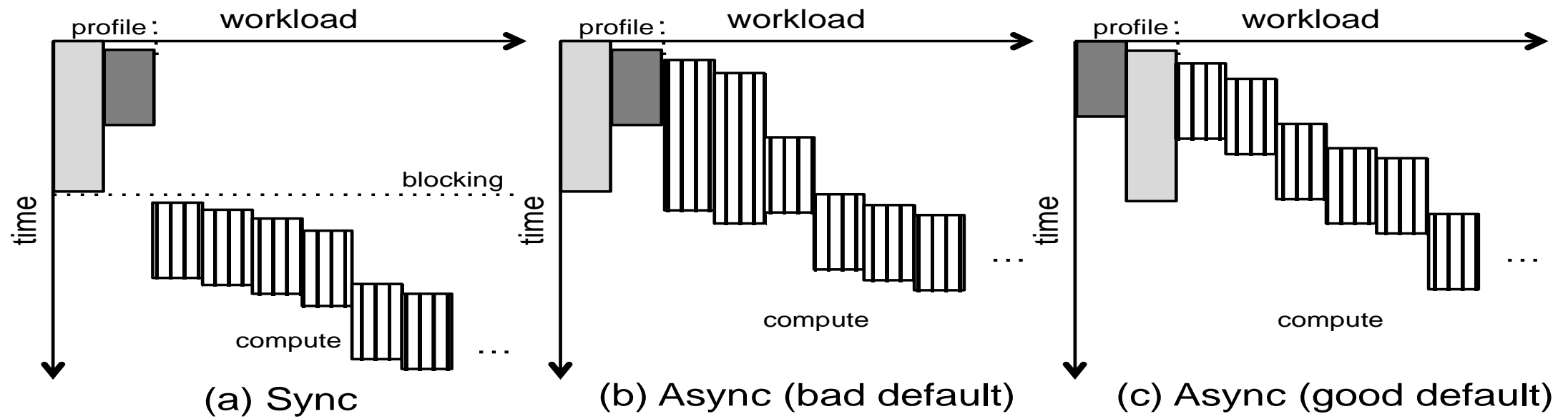
# Productive Profiling Mode

- Computation in profiling also contributes to the final output



# Synchronous vs Asynchronous Scheduling

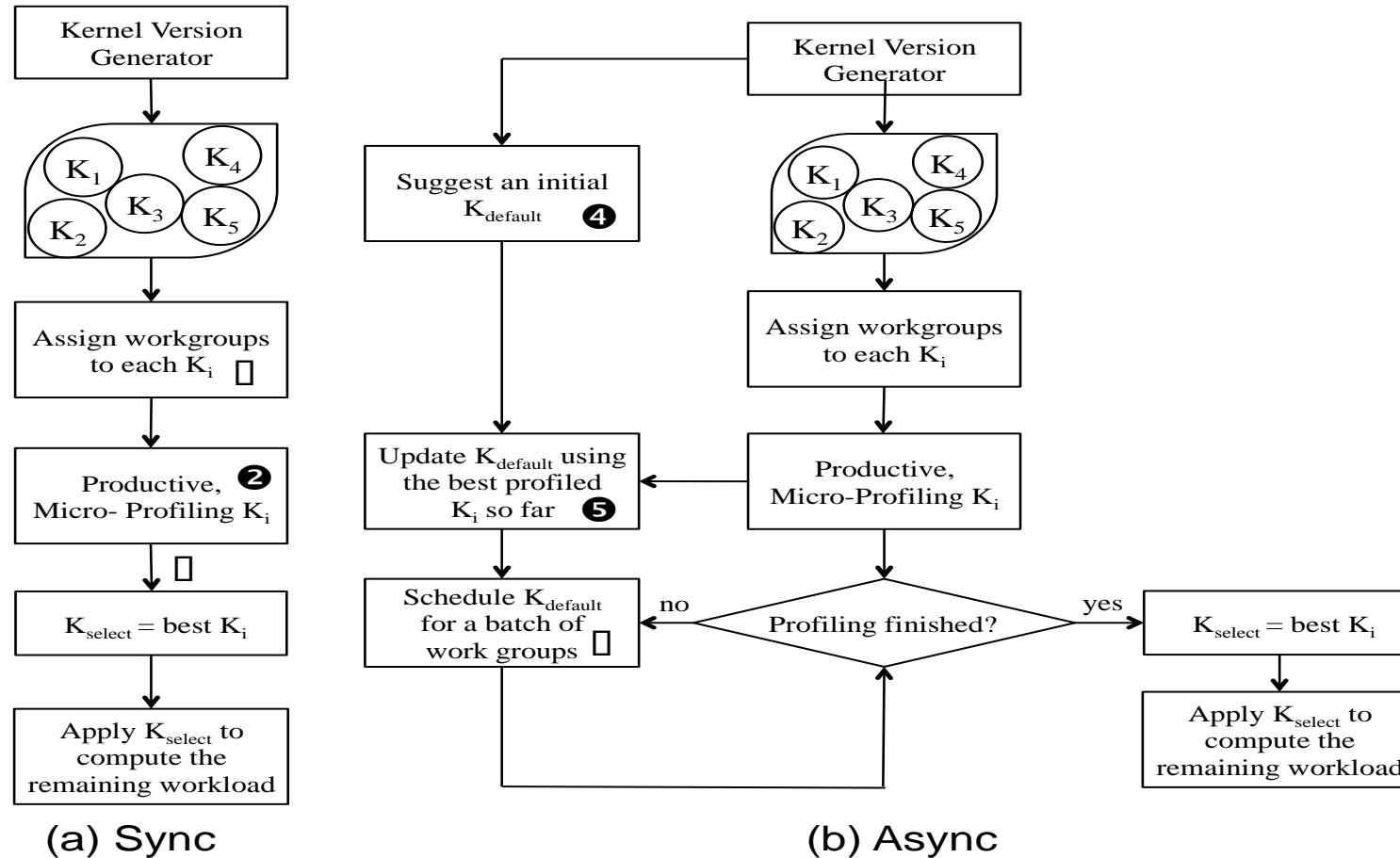
- **Synchronous**: Schedule the remaining workload after the best version is finalized
- **Asynchronous**: Schedule remaining workload eagerly in a batch using the current best candidate



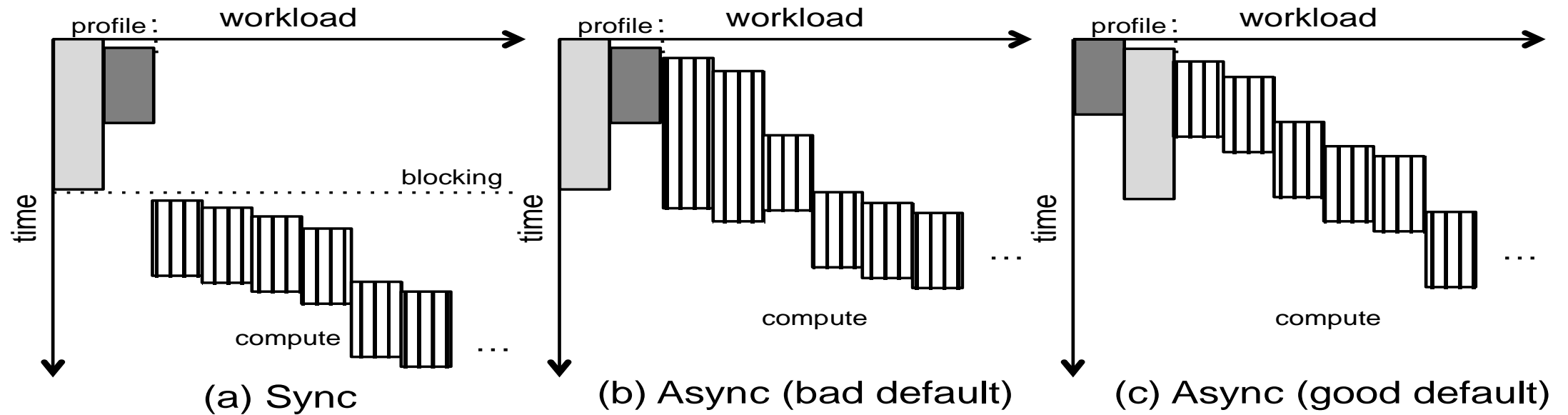
# Sync vs Async Scheduling

- Sync
  - Schedule the remaining workload after the best version is finalized
- Async
  - Schedule remaining workload eagerly in a batch using the current best candidate

# Sync vs Async Scheduling



# Sync vs Async Scheduling



# Things I skipped

- The two extra profiling modes
- Applicability and resource requirement of each mode
- What kind of compiler analyses needed for different modes
- Where compilers add profiling code in both CPU and GPU
- More details about DySel runtime using TBB and CUDA

# DySel Interface

```
DySelAddKernel(  
    string kernel_sig, // kernel name  
    func_ptr implementation, // kernel implementation  
    dim3 wa_factor, // work assignment factor  
    vector<int> sandbox_index= [ ] // argument offsets for  
    // sandboxes/private outputs  
);
```

(a) Kernel Implementation Registration API

```
DySelLaunchKernel(  
    string kernel_sig, // kernel name  
    bool profiling=true, // profiling activation flag  
    enum mode=fully_async // profiling mode  
);
```

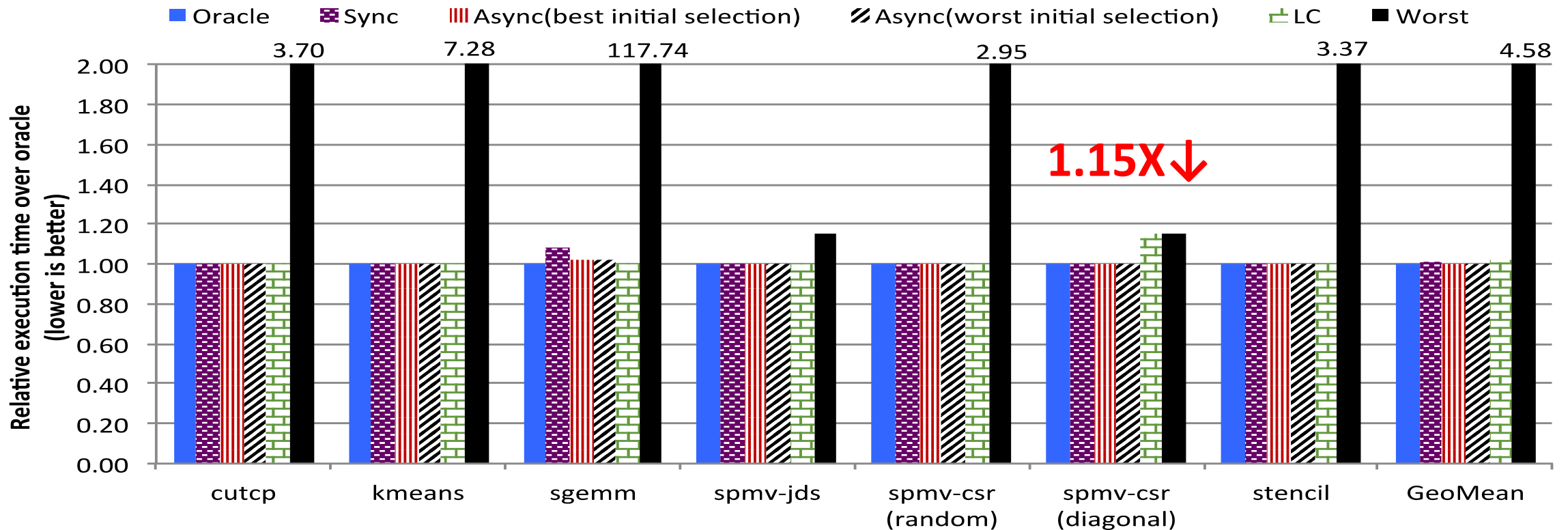
(b) Kernel Launch API



# Case Study: Locality-centric Scheduling for CPU OpenCL

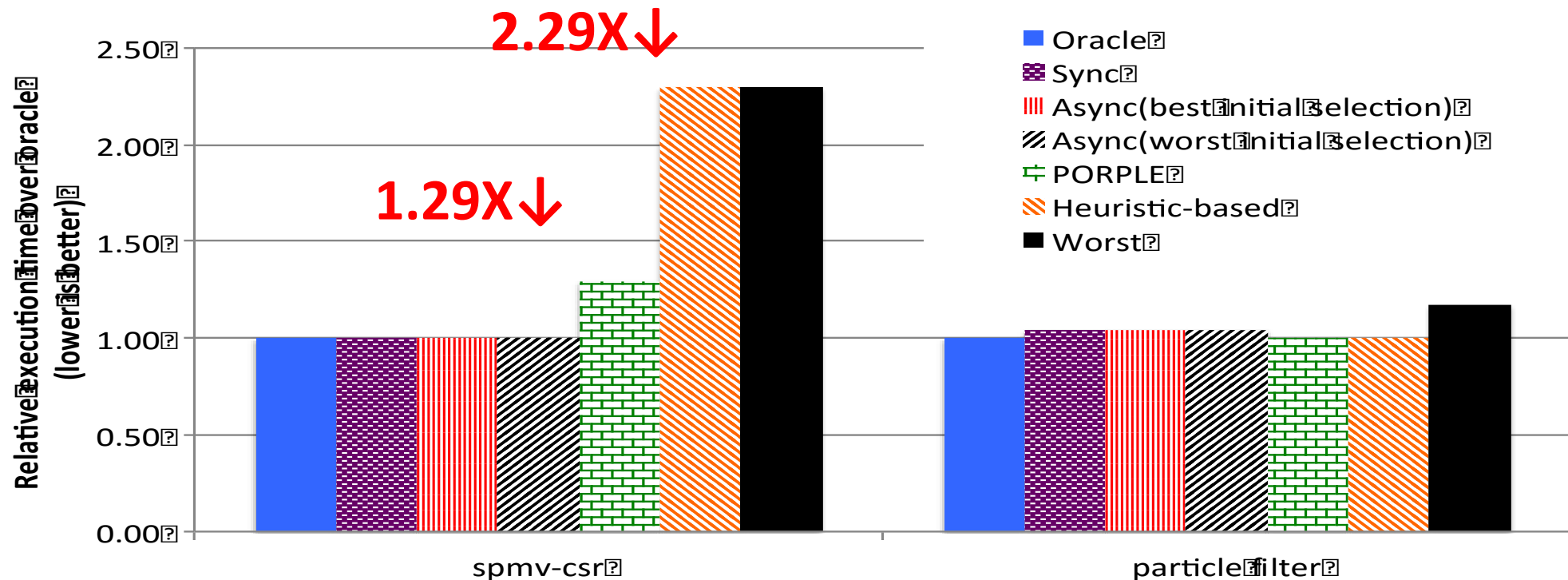
- Iterate in-kernel loops first or work-item loops for OpenCL on CPU (CGO'15) using MxPA
  - Through analyzing access patterns
- It is open-source, and robust
  - “3.32x over AMD, 1.71x over Intel OpenCL stacks”

# Case Study: Locality-centric Scheduling for CPU OpenCL



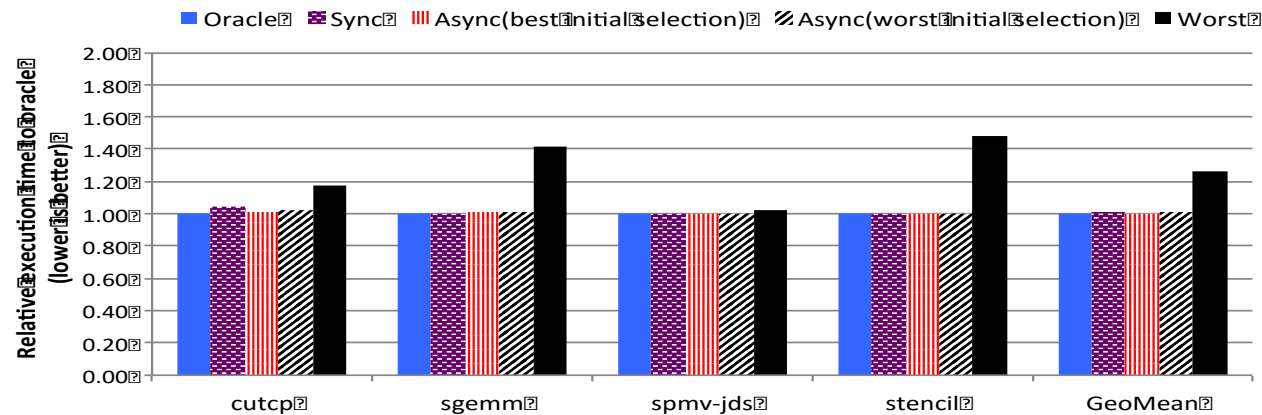
# Case Study: Data Placement for GPU

- Data placement optimizations are crucial for performance on GPUs (TPDS 2011 & MICRO 2014)
  - Although they are not open-source, they did show the transformed results
- Suboptimal decisions due to inaccurate model or improper heuristic

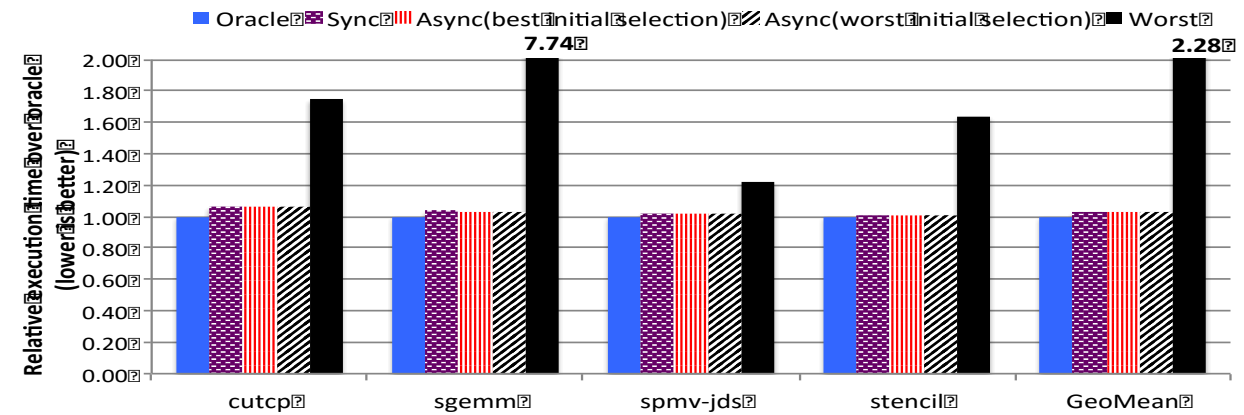


# Case Study: Experts' Mixed Optimizations

- Parboil provides multiple versions with different optimization strategies
  - Optimized versions usually run better
  - Some Optimizations are improper or redundant
  - E.g. loop unrolling and prefetching in spmv-jds on Kepler



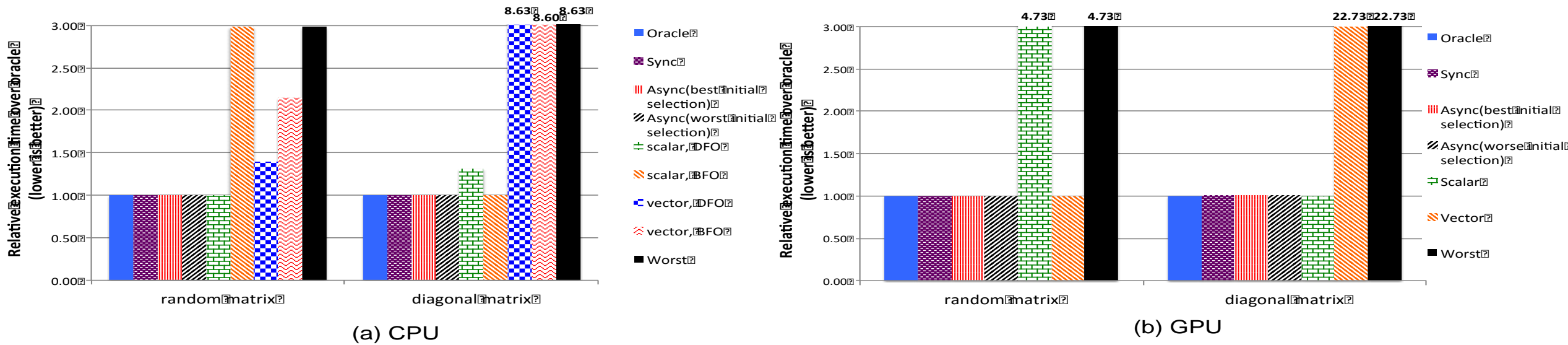
(a) CPU



(b) GPU

# Case Study: Input-dependent Optimizations

- Best optimizations could be input-dependent



# Conclusion

- DySel can deliver **high accuracy** and **low overhead** for dynamic version selection in data-parallel programming model
  - Incur less than **8%** of overhead in the worst observed case
- Using DySel is like buying an insurance...

# MxPA Backup

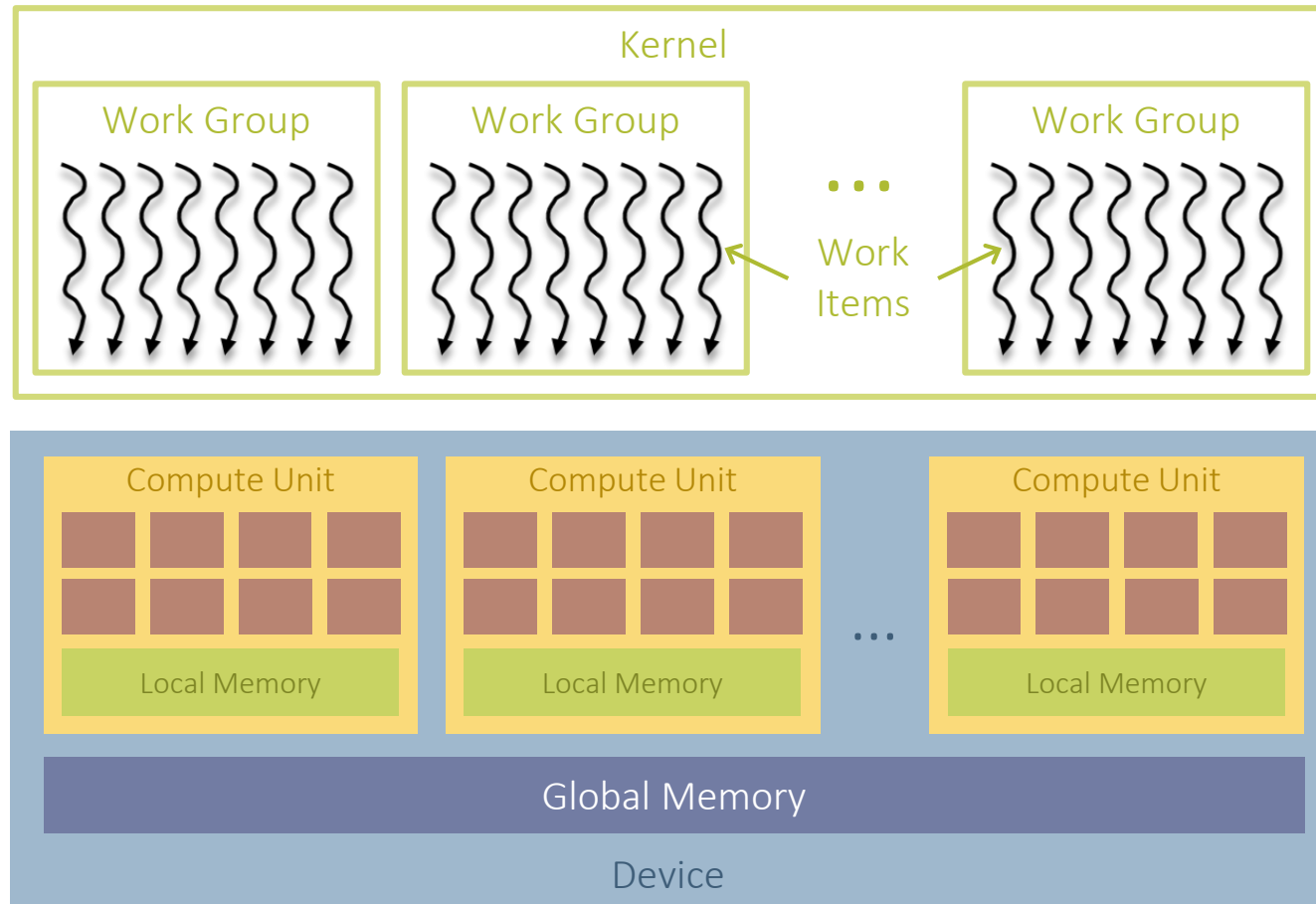
# Contributions

- Exploiting **data locality** in scheduling work-items for performance
- **Real system and measurement** demonstrates speedups of **3.32x** and **1.71x** over AMD and Intel OpenCL implementations
  - 18 benchmarks from Parboil and Rodinia
- Nominated for best paper award at CGO'15
- AE certified





# OpenCL Programming Model



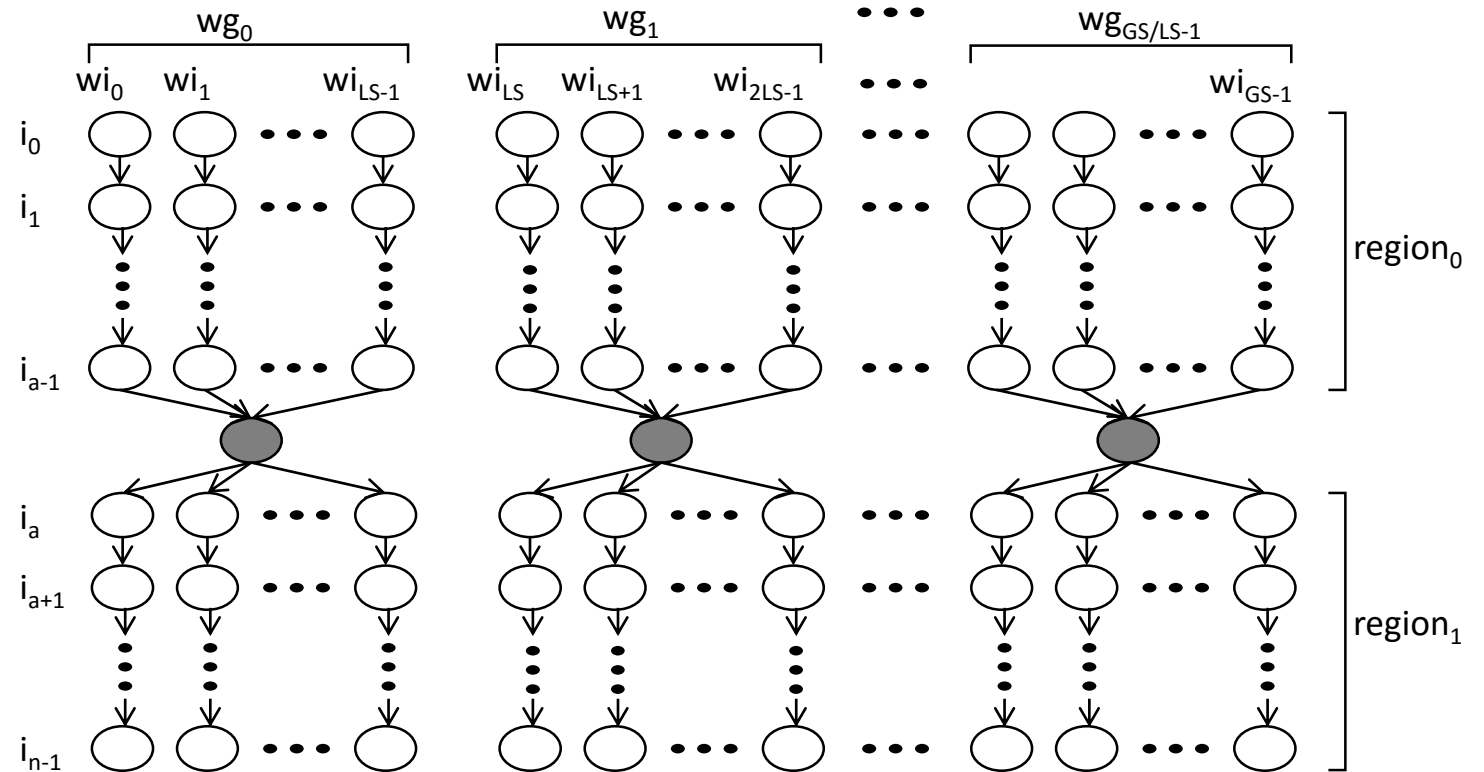
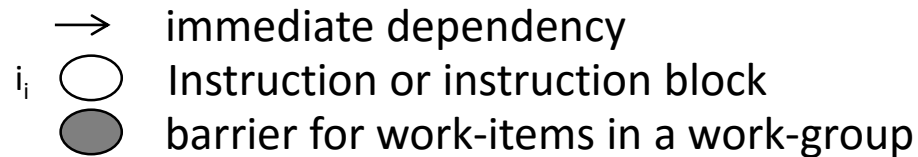
# OpenCL Execution Model

```

void kernel(...) {
    i0;
    i1;
    ...
    ia-1;
    barrier();
    ia;
    ia+1;
    ...
    ib-1;
}
    
```

**kernel code**

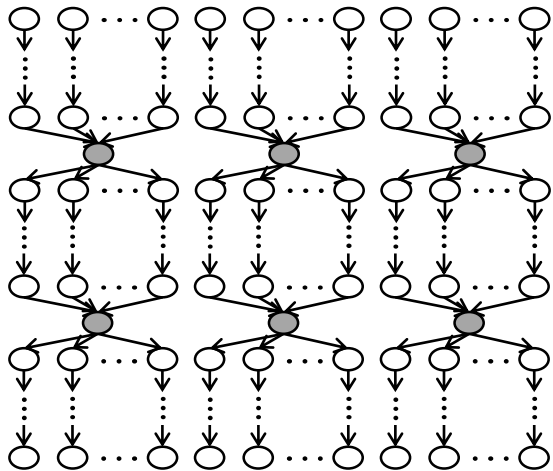
wi = work-item  
 wg = work-group  
 LS = local size  
 GS = global size



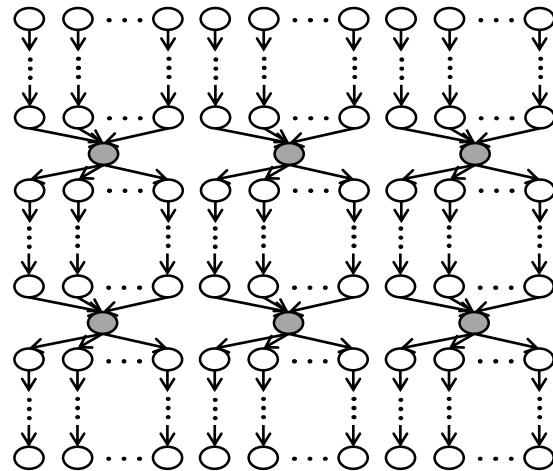
**How to schedule this execution graph on a multicore CPU?**

# Work-group Scheduling

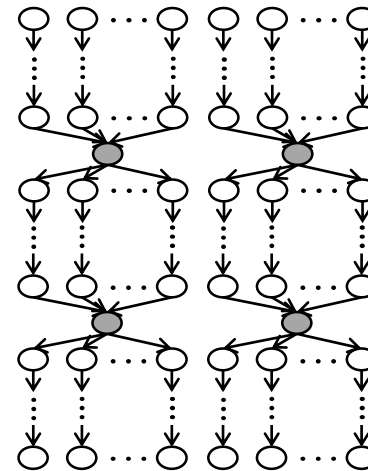
- Assign work-groups in whole to different cores
  - Considerations: Locality, Load balance



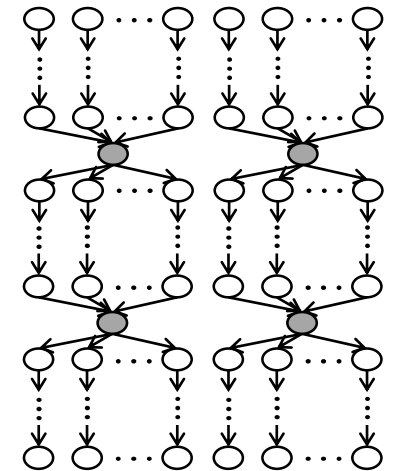
CPU Core



CPU Core



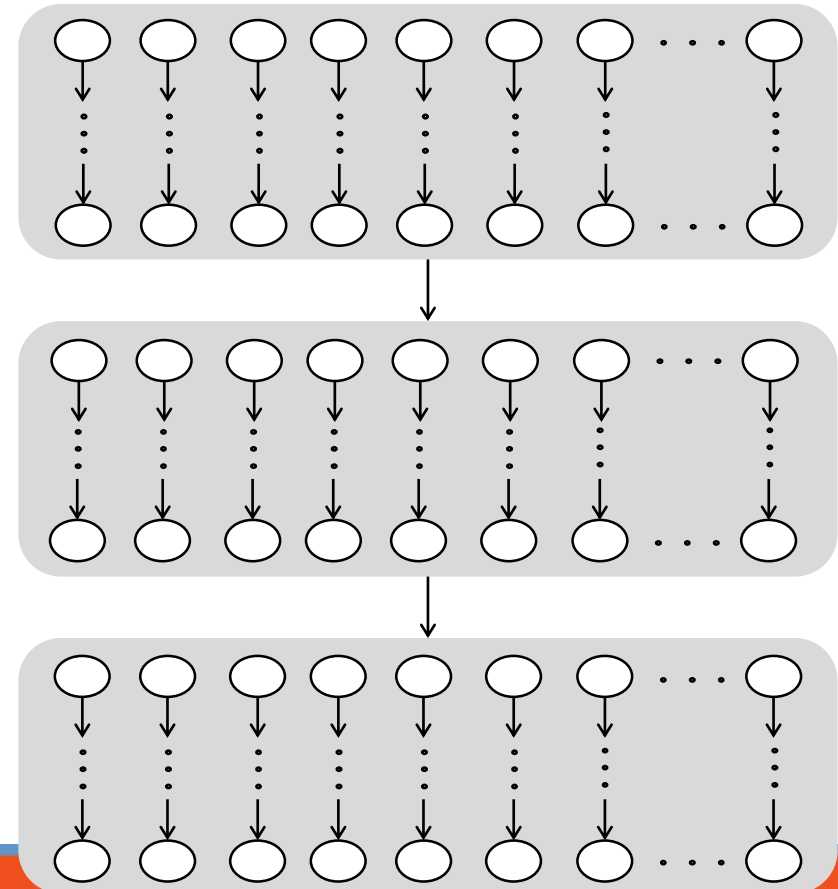
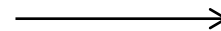
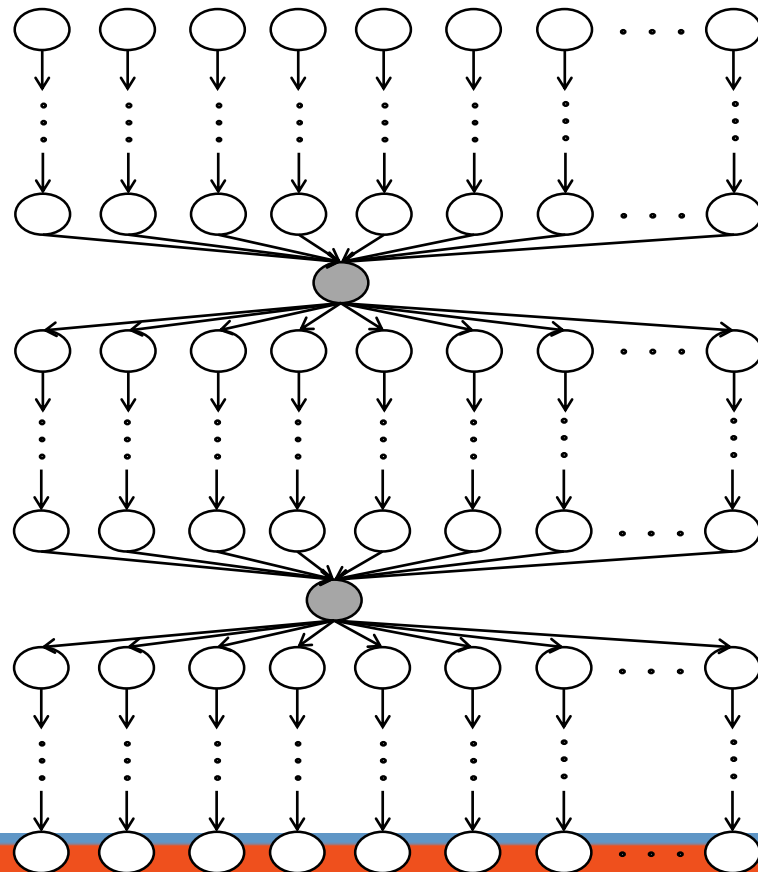
CPU Core



CPU Core

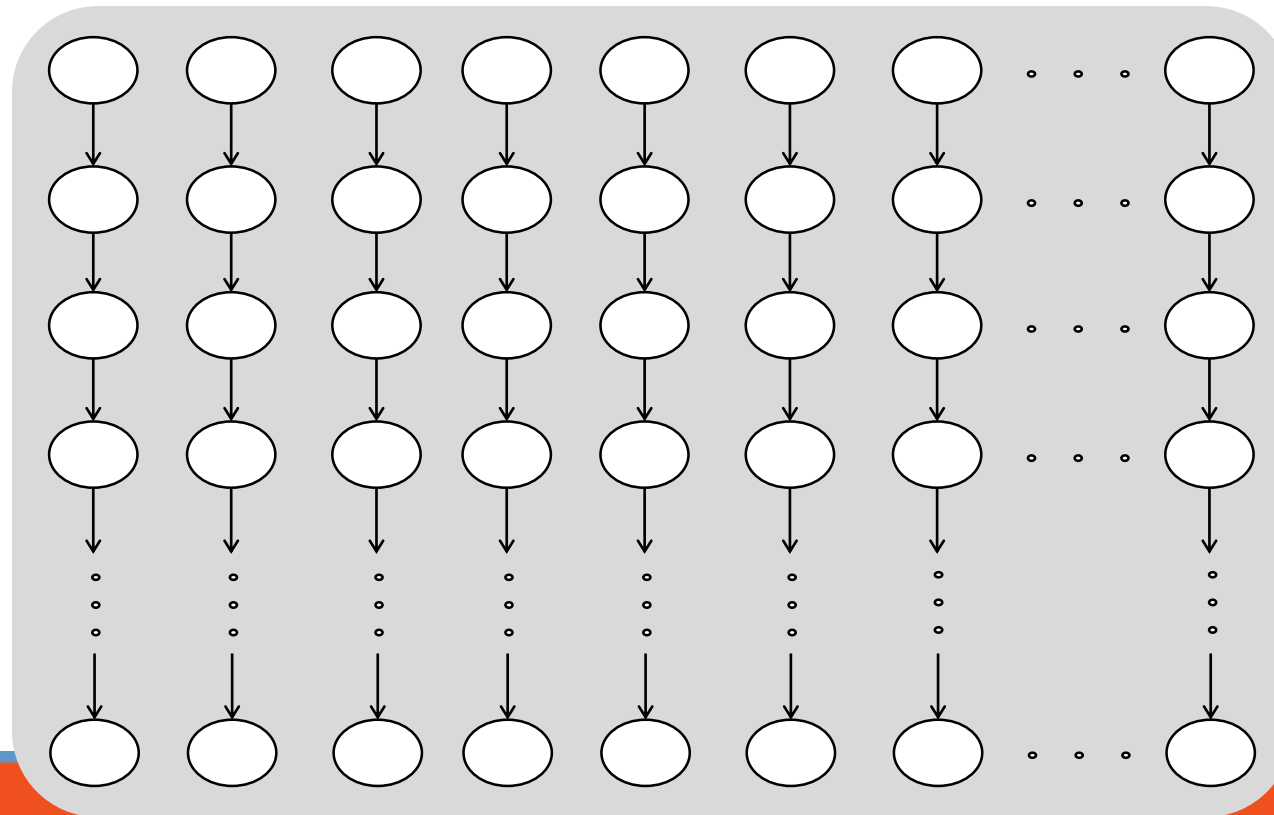
# Region Scheduling

- Serialize barrier-separated regions



# Work-item Scheduling

- How to schedule work-items within a region?
  - Different approaches by different compilers



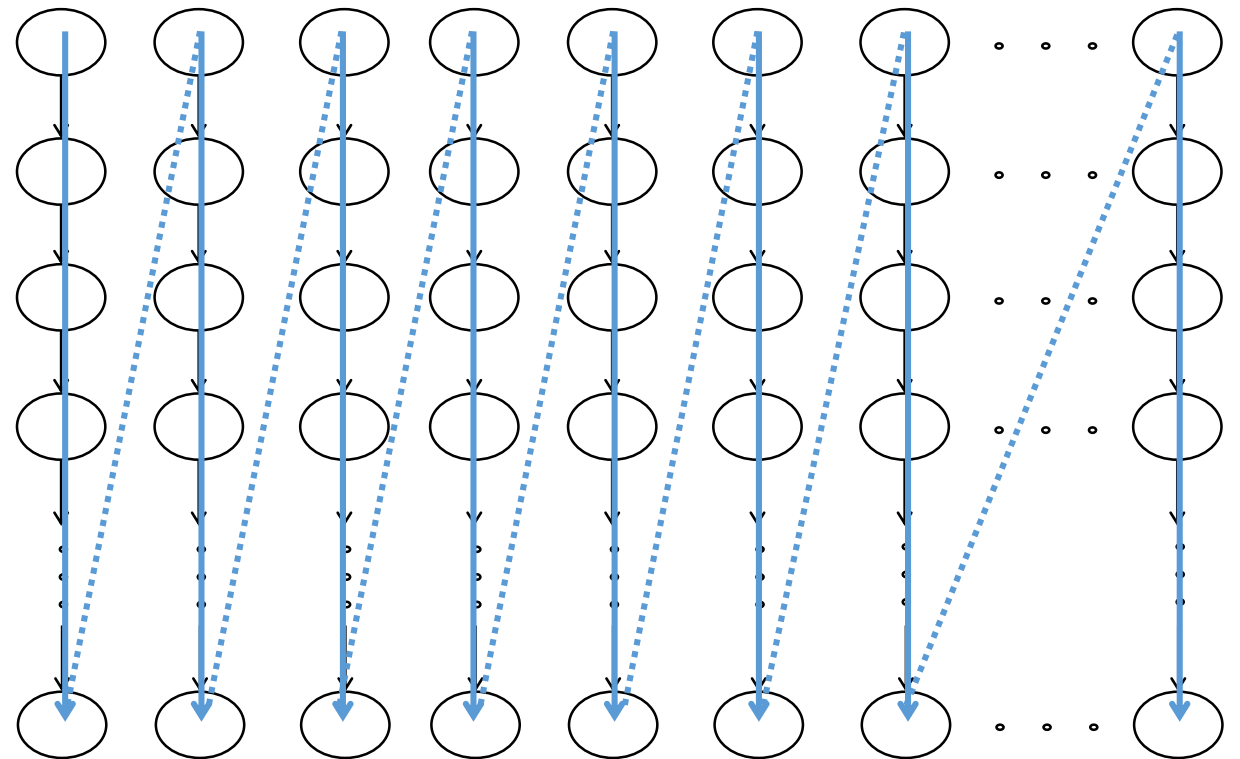
# Existing Approaches

- Industry

- Intel
- AMD (Twin Peaks)

- Academia

- Karrenberg & Hack
- SnuCL
- pocl



Depth First Order (DFO) Scheduling

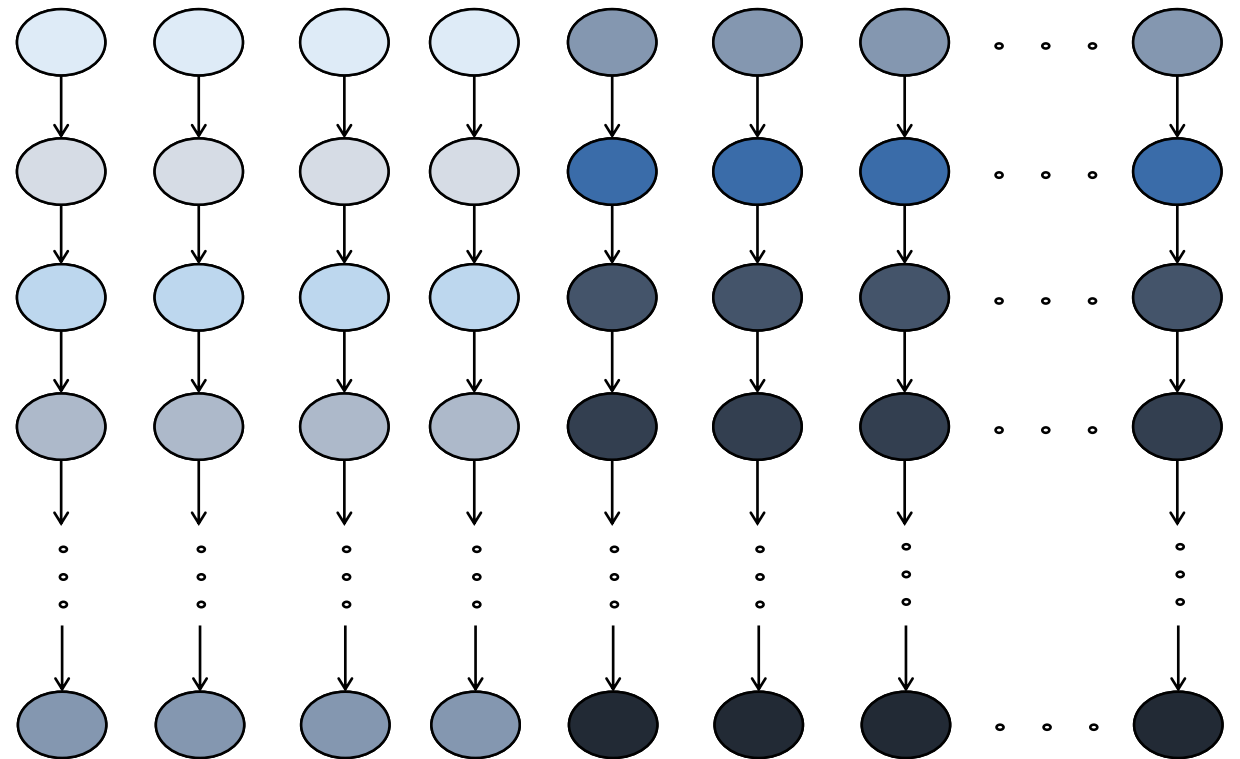
# Existing Approaches

- Industry

- Intel
- AMD (Twin Peaks)

- Academia

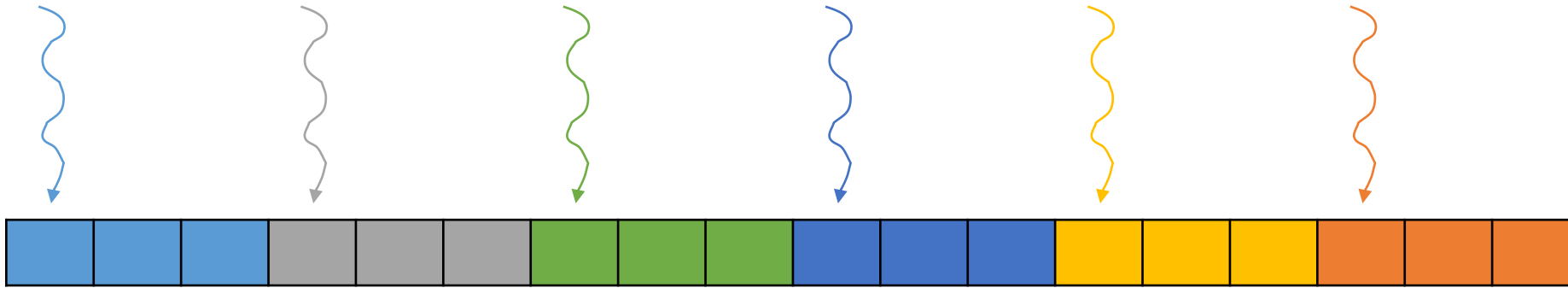
- Karrenberg & Hack
- SnuCL
- pocl



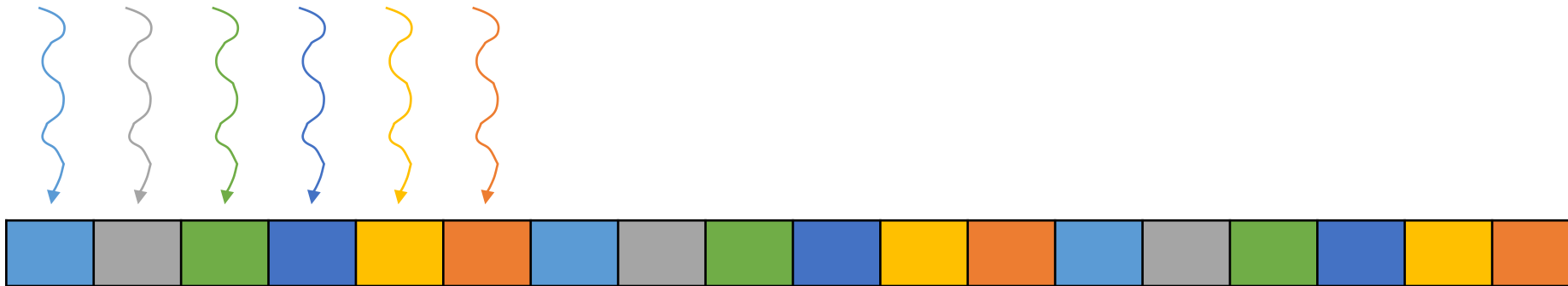
DFO Scheduling with Vectorization

(time progresses as color gets darker)

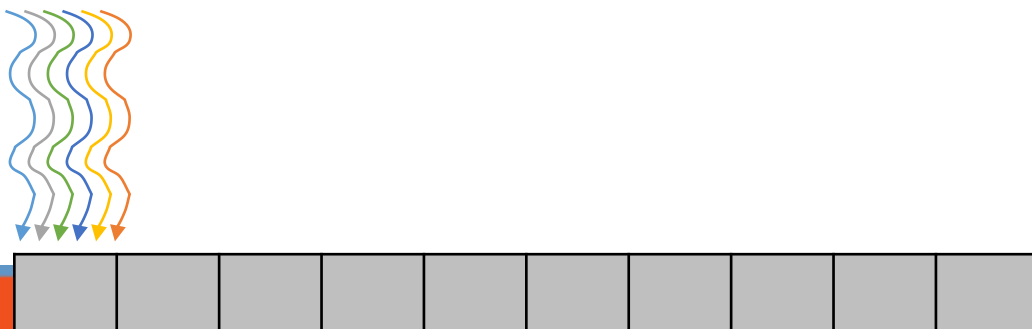
# Memory Access Patterns



e.g. bfs  
*(each thread traverses a list of neighbors)*



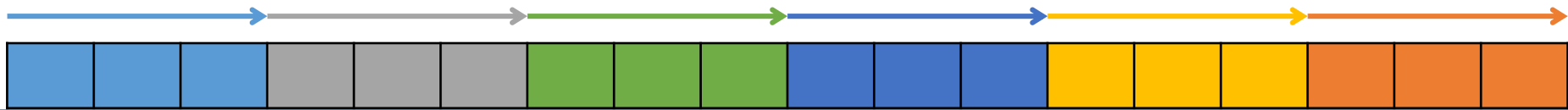
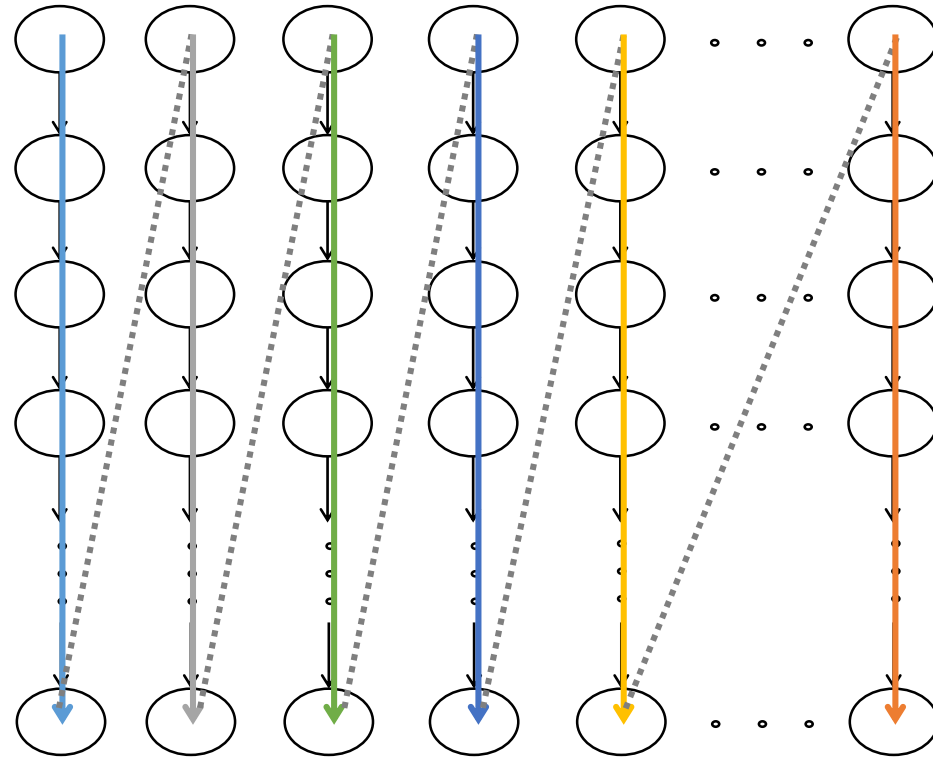
e.g. sgemm  
*(threads computing adjacent outputs access adjacent inputs)*



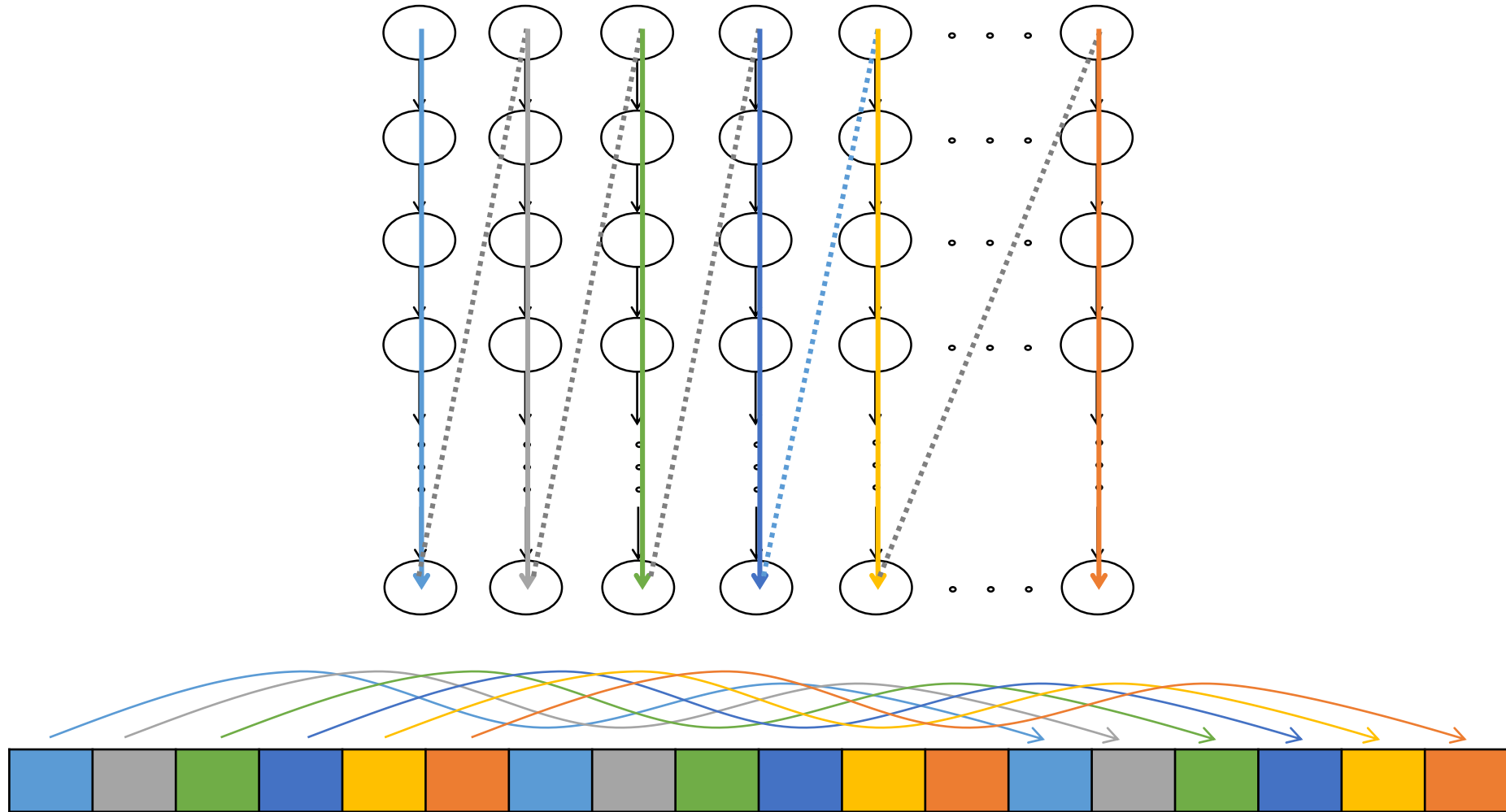
e.g. kmeans  
*(all threads loop over the same mean values)*



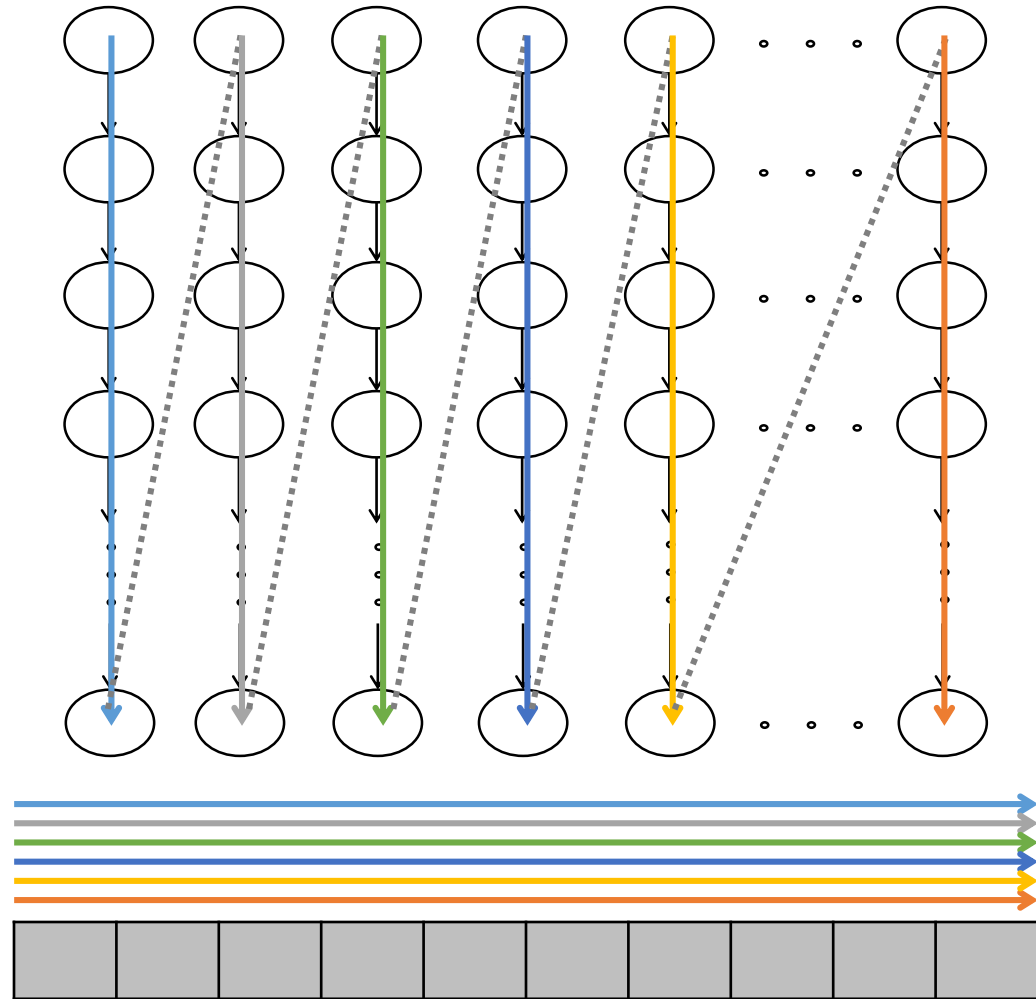
# DFO and Locality



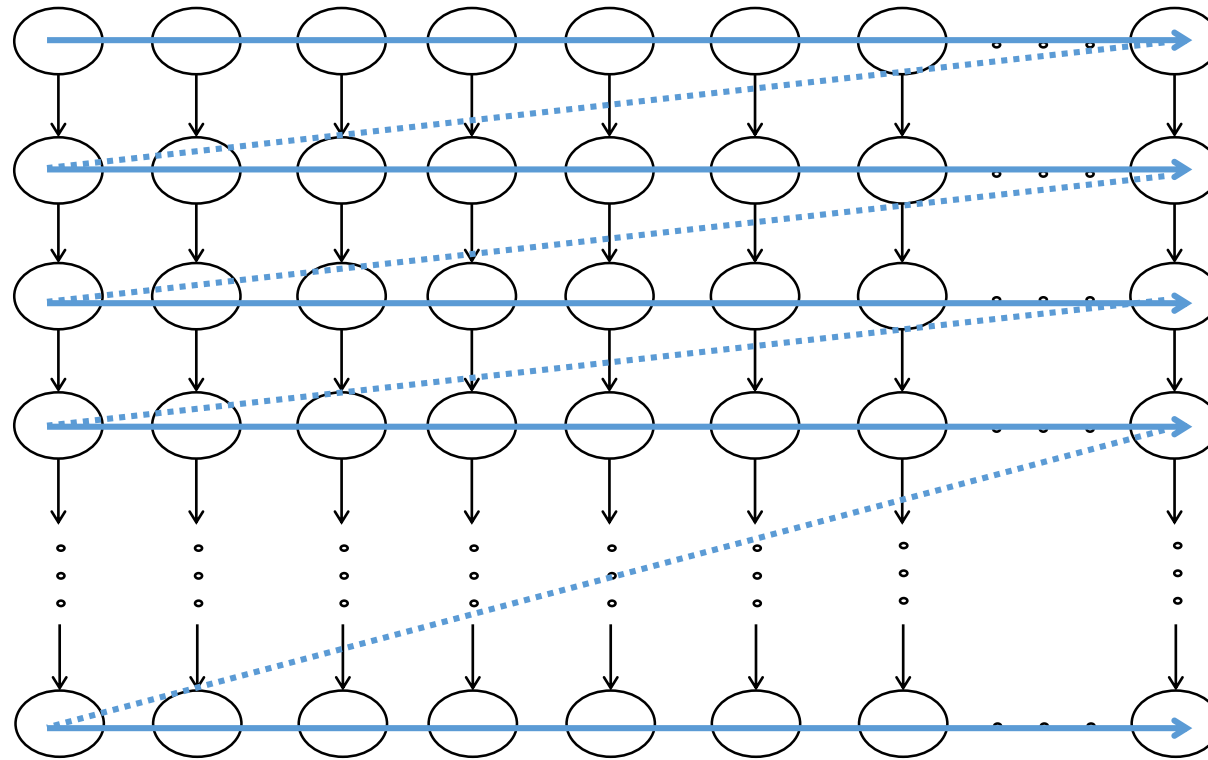
# DFO and Locality



# DFO and Locality

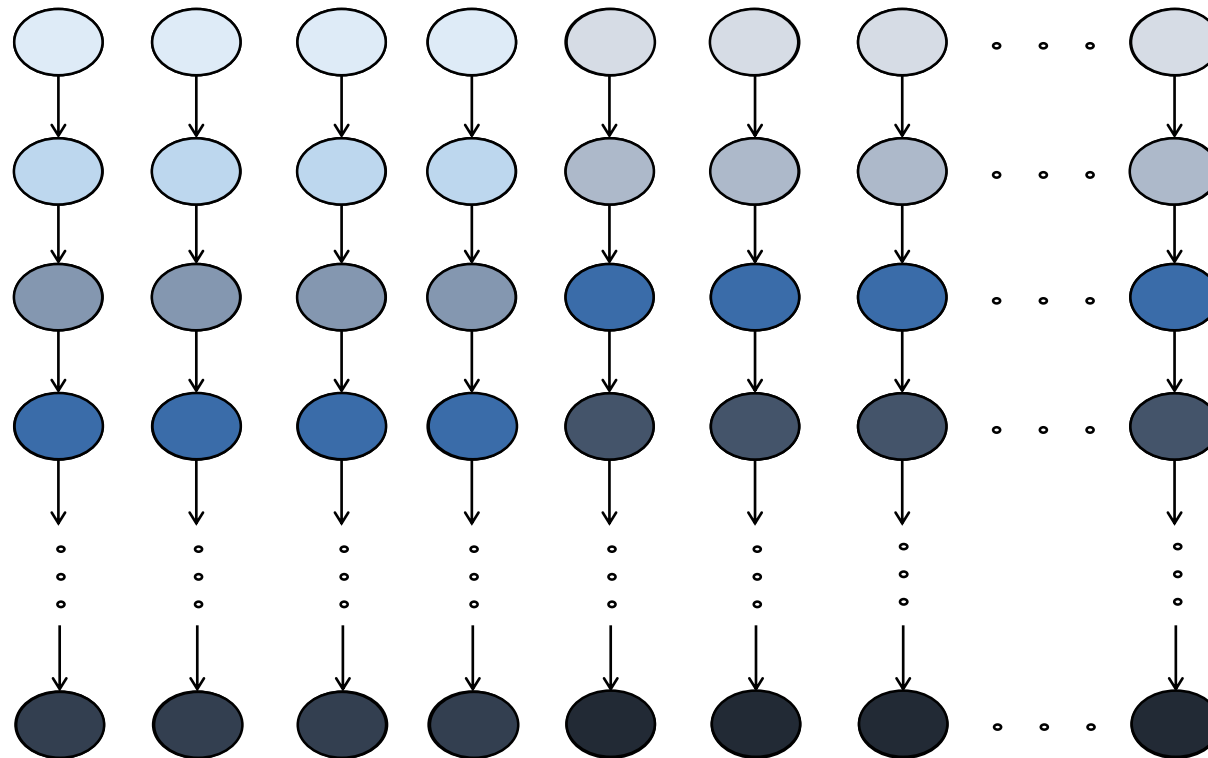


# Alternative Schedule: BFO



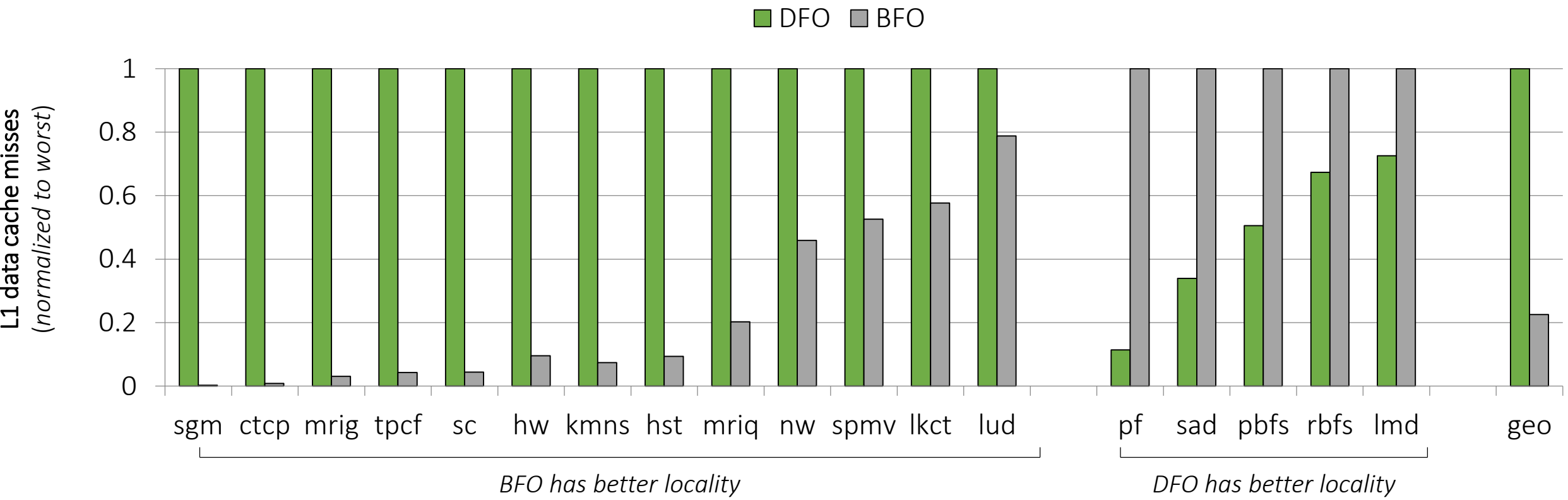
Breadth First Order (BFO) Scheduling

# Alternative Schedule: BFO



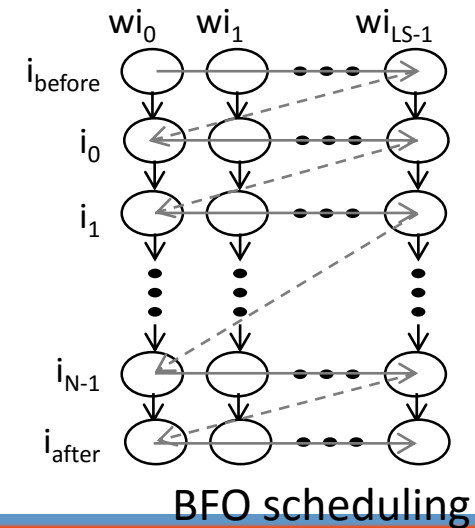
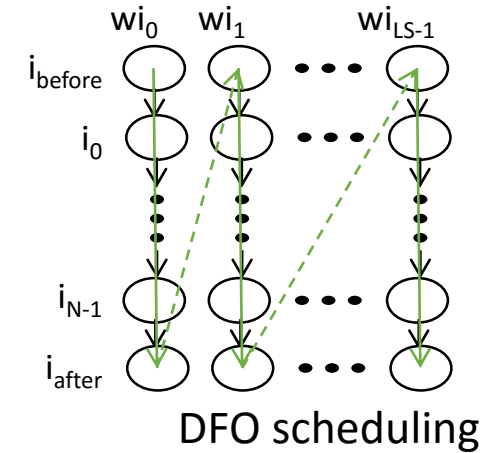
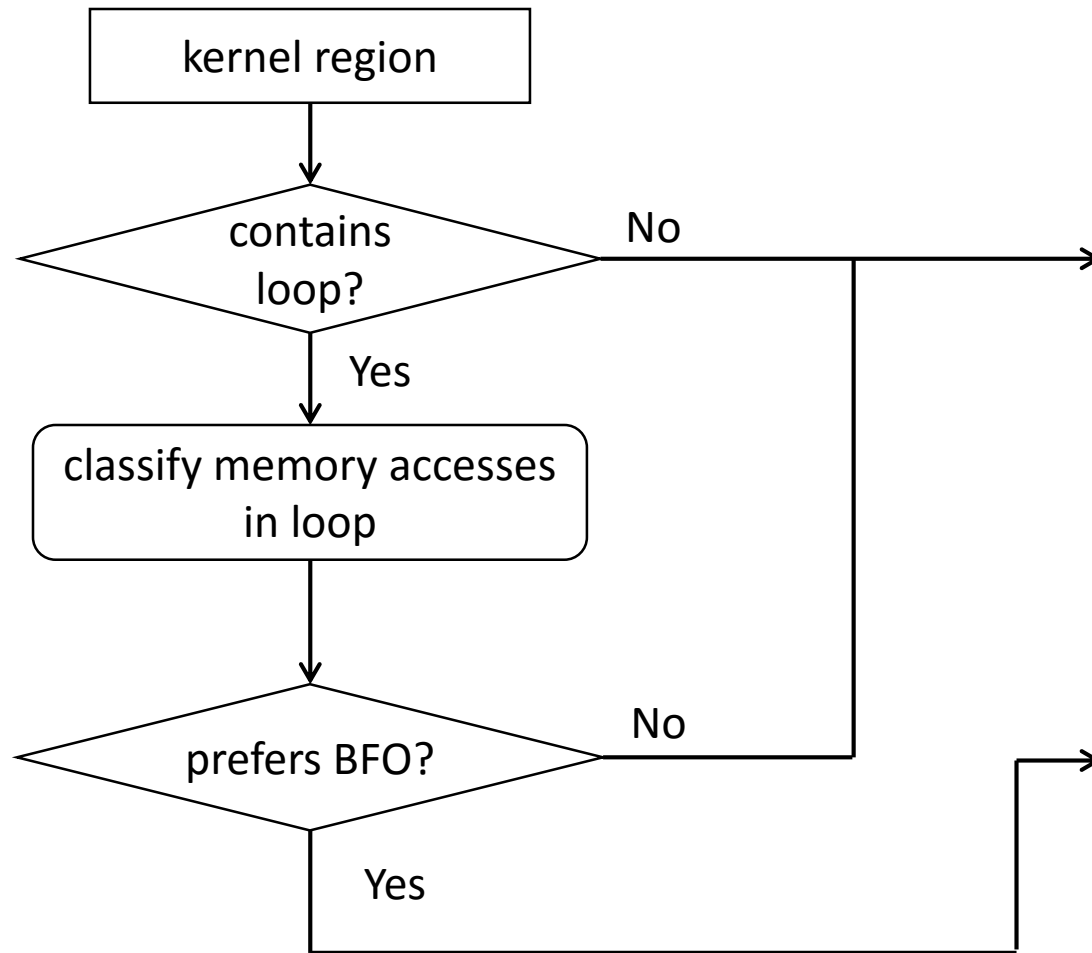
**BFO with Vectorization**  
(time progresses as color gets darker)

# DFO's vs. BFO's Impact on Locality



BFO has better locality for 13 benchmarks, DFO has better locality for 5 benchmarks. **No schedule is always the best.**

# Locality Centric (LC) Scheduling



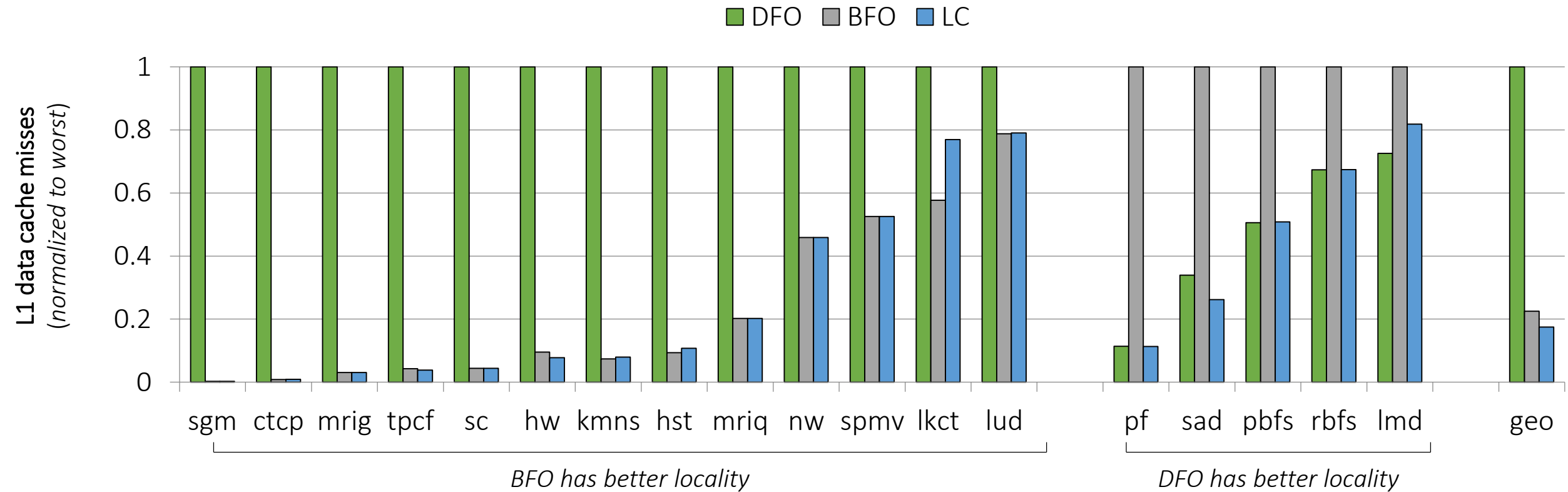
# Locality Centric (LC) Scheduling

		Work-item Stride		
		0	1	Other
Loop Iteration Stride	0	-	DFO	DFO
	1	BFO	-	DFO
	Other	BFO	BFO	-

Classify memory accesses per loop body and tally which schedule has greater popularity

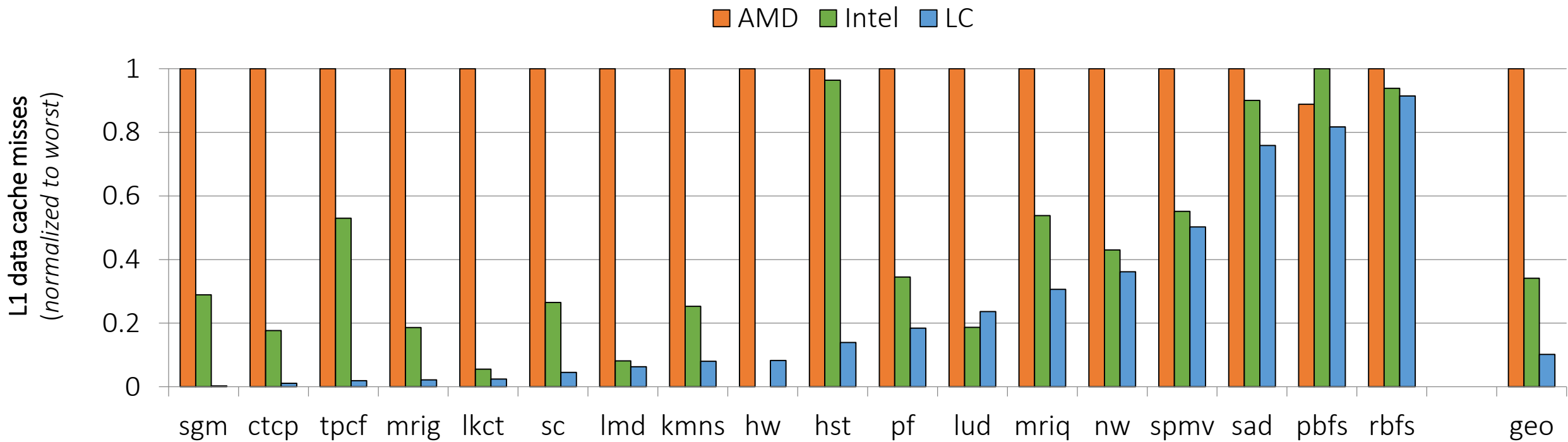


# LC's Impact on Locality



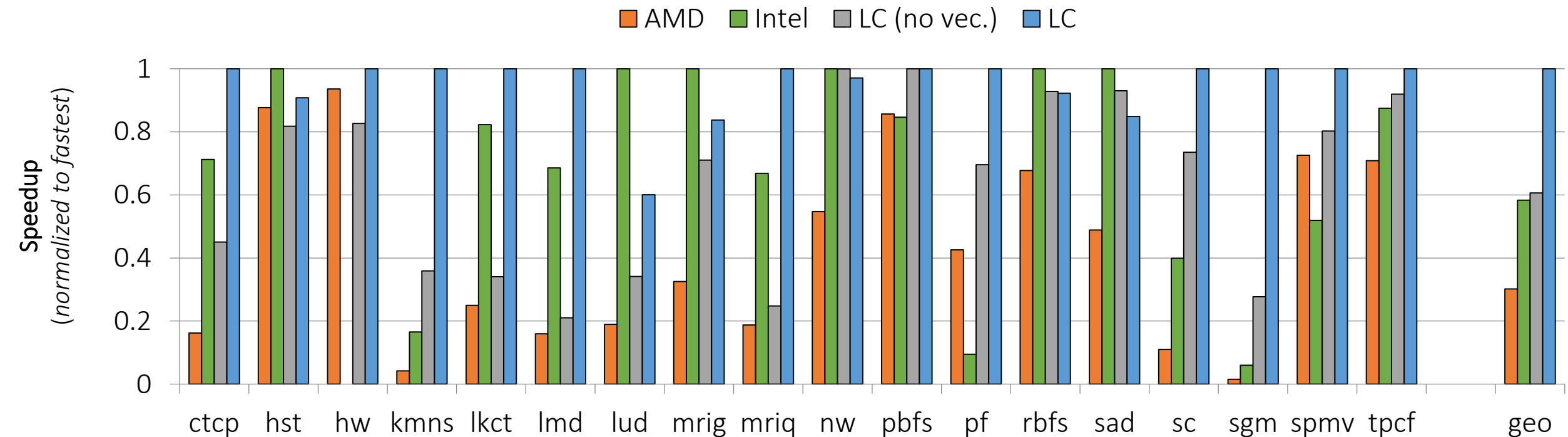
LC captures the best of both schedules

# Locality Results



LC has best locality for most benchmarks

# Performance Results



LC (with vec.) outperforms AMD (without vec.) and Intel (with vec.) by 3.32x and 1.71x

LC (without vec.) is faster than Intel (with vec.) by 1.04x

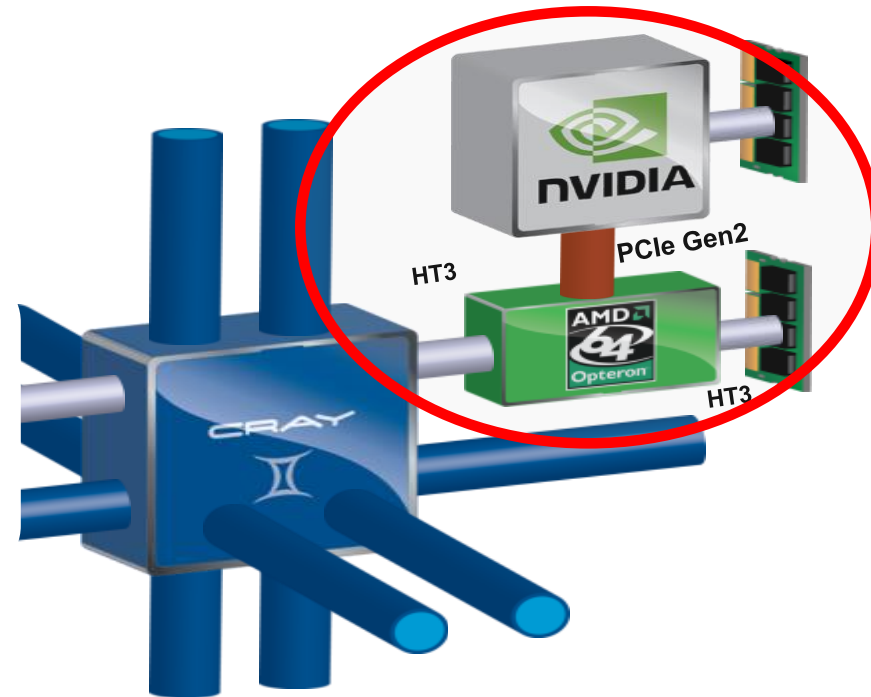
# Summary

- Proposed an alternative scheduling approach to the state-of-the-art
- Demonstrated that no schedule is always best and proposed a static schedule selection
- Outperformed industry implementations in memory system efficiency and performance

# Heterogeneous Computing in Blue Waters

- Dual-socket Node
  - One AMD Interlagos chip
    - 8 core modules, 32 threads
    - 156.5 GFs peak performance
      - Consumes 2,504 GB of data per second
    - 32 GBs memory
      - 51 GB/s bandwidth
  - One NVIDIA Kepler chip
    - 1.3 TFs peak performance
      - Consumes 20,800 GB of data per second
    - 6 GBs GDDR5 memory
      - 250 GB/sec bandwidth

**CRAY**  
THE SUPERCOMPUTER COMPANY



**Blue Waters contains 4,224 Cray XK7  
compute nodes.**