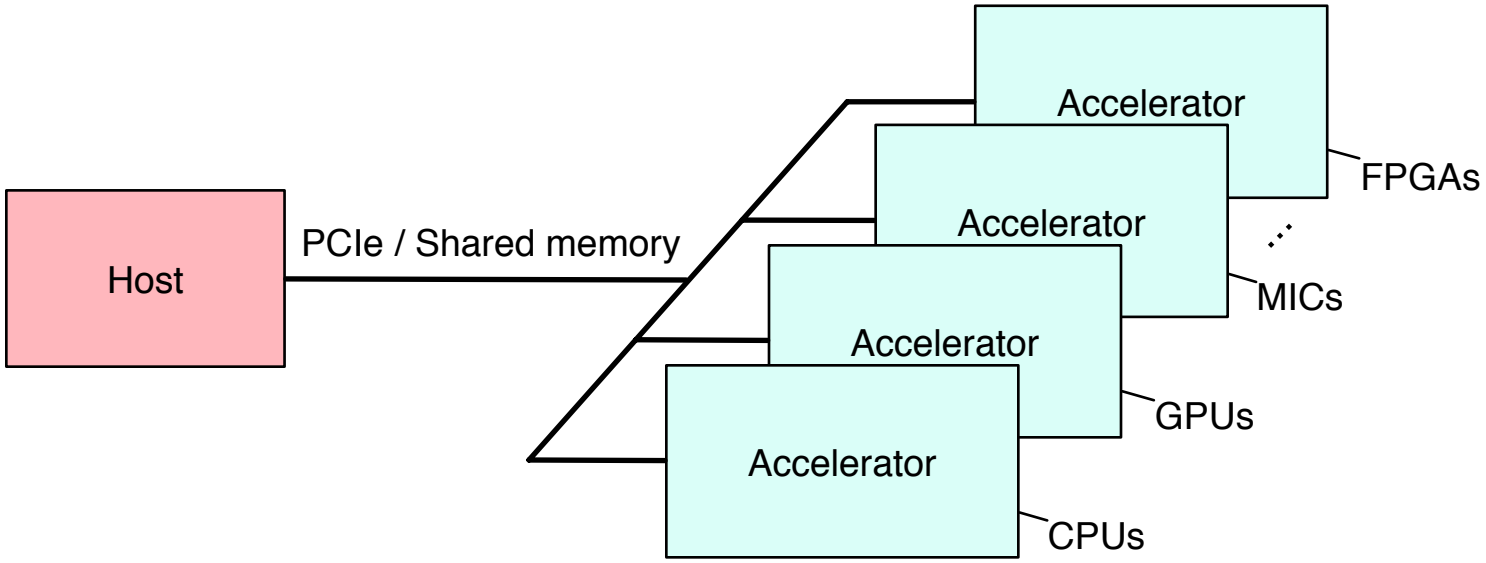
Heterogeneous platforms

- Systems combining main processors and accelerators
  - e.g., CPU + GPU, CPU + Intel MIC, AMD APU, ARM SoC
  - Everywhere from supercomputers to mobile devices

Heterogeneous platforms

- Host-accelerator hardware model



Heterogeneous platforms

- Top 500 (June 2015)

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	1,572,864	17,173.2	20,132.7	7,890
4	RI Co Ja					0
5	DOE/SC/Argonne National Laboratory United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945

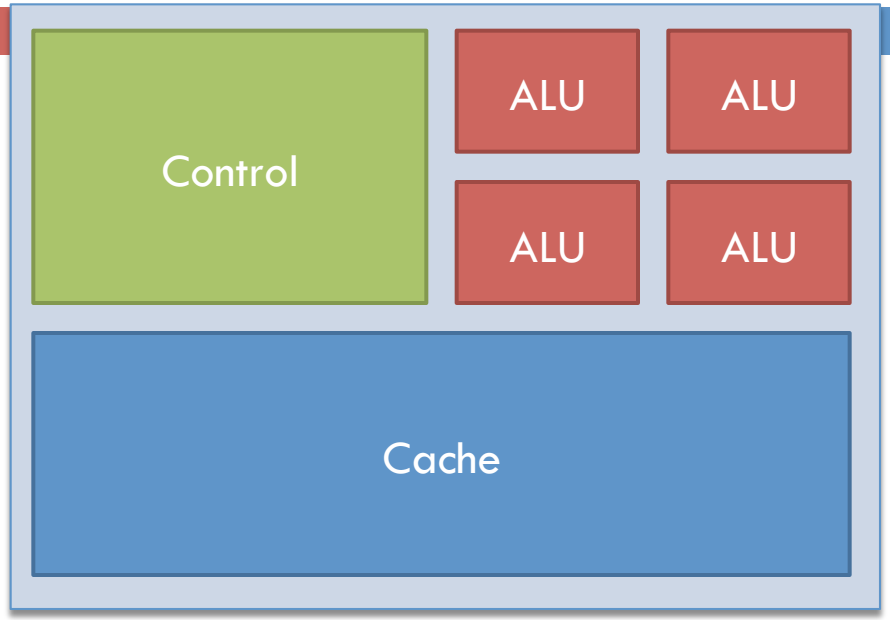
195 cores/node

Accelerated!

Accelerated!

All systems are based on multi-cores.
90 systems have accelerators (18%).
Of those, 50% are NVIDIA GPUs, 30% are Intel MICs (Xeon Phi).

CPU vs. Accelerator (GPU)

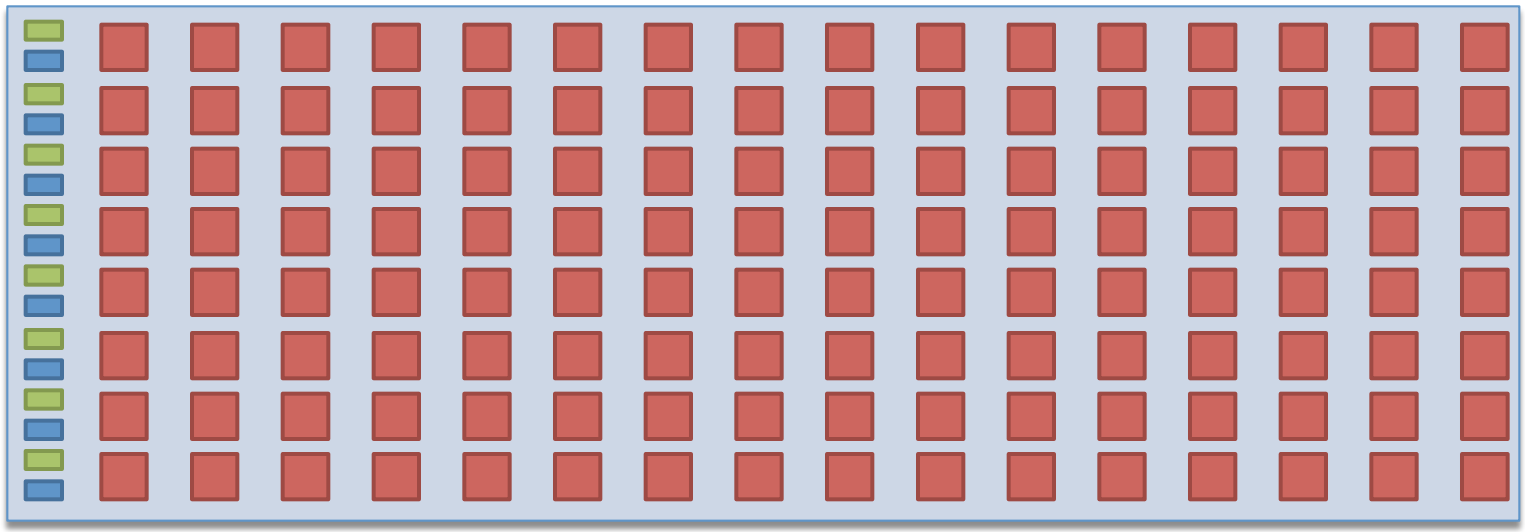


CPU

Low latency, high flexibility.
Excellent for irregular codes with limited parallelism.

GPU

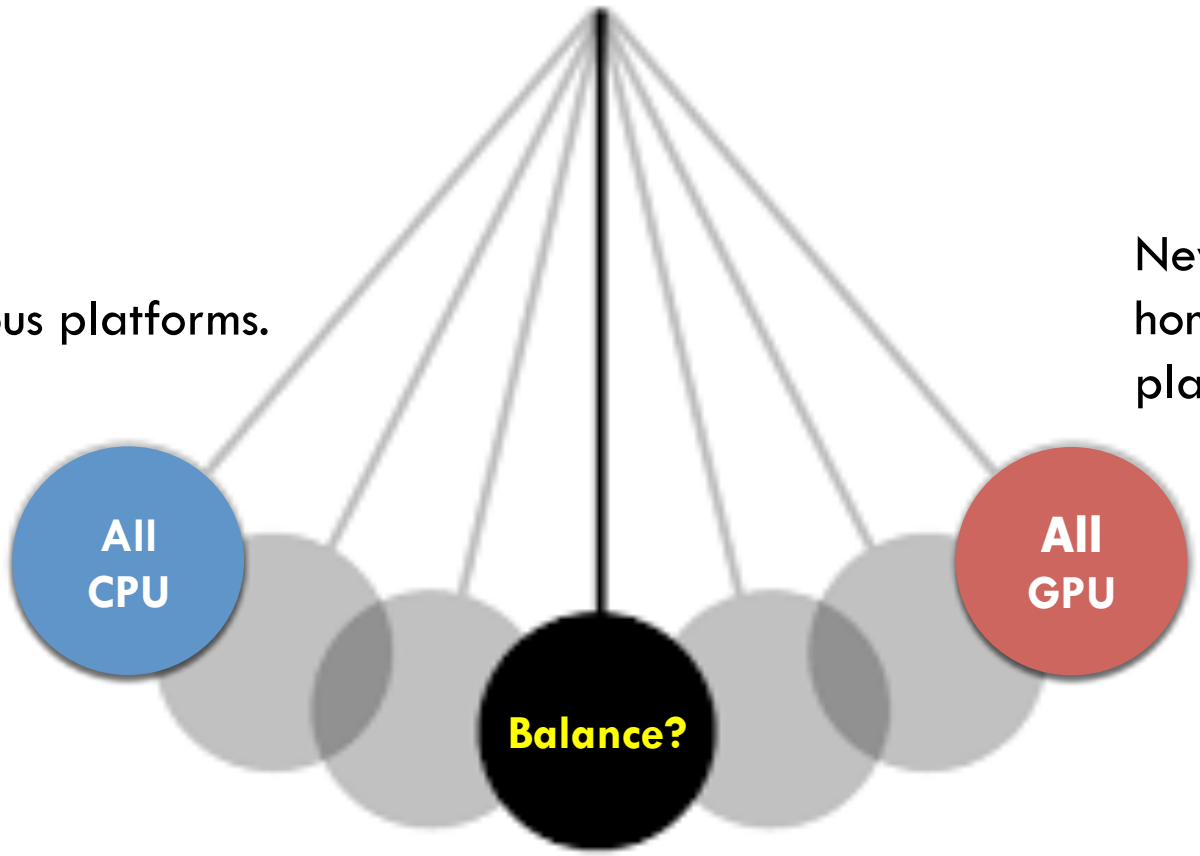
High throughput.
Excellent for massively parallel workloads.



A CPU-GPU pendulum

Traditional, homogeneous platforms.

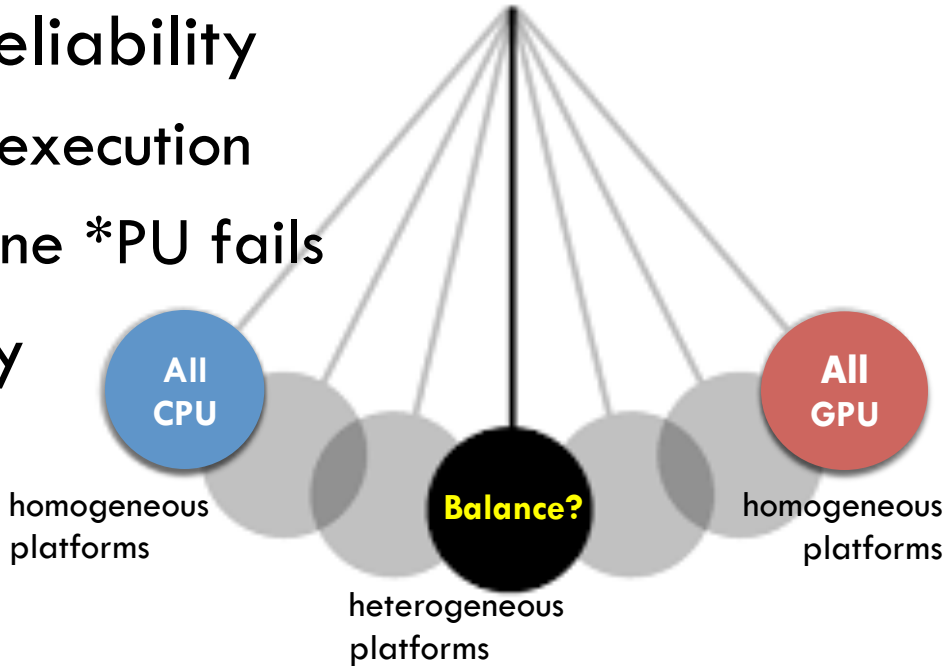
New…still homogeneous platforms.



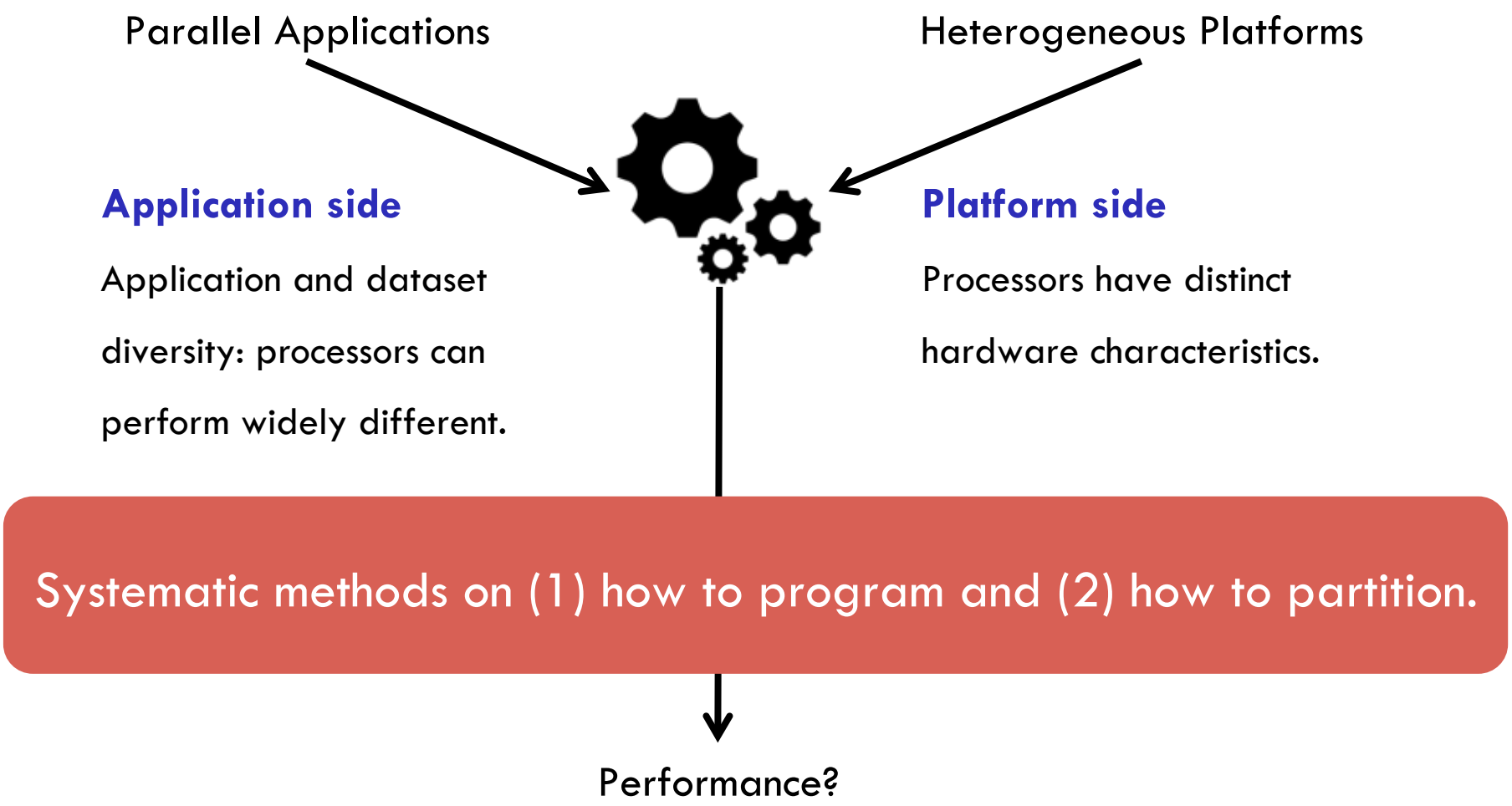
Heterogeneous systems for equilibrium.

Heterogeneity vs. Homogeneity

- Increase performance
  - Both devices work in parallel
  - Decrease data communication
  - Different devices for different roles
- Increase flexibility and reliability
  - Choose one/all *PUs for execution
  - Fall-back solution when one *PU fails
- Increase power efficiency
- Cheaper per flop



…But no free lunch!



Goal

- Introduce our methods for efficient heterogeneous computing
  - Programming (ask me more)
  - Partitioning
- Practical examples
- Current challenges and open research questions.

- Fair to others, but advertise our research ☺

Outline

- Introduction (and motivation)
- Partitioning strategies
- Static partitioning
- Dynamic partitioning

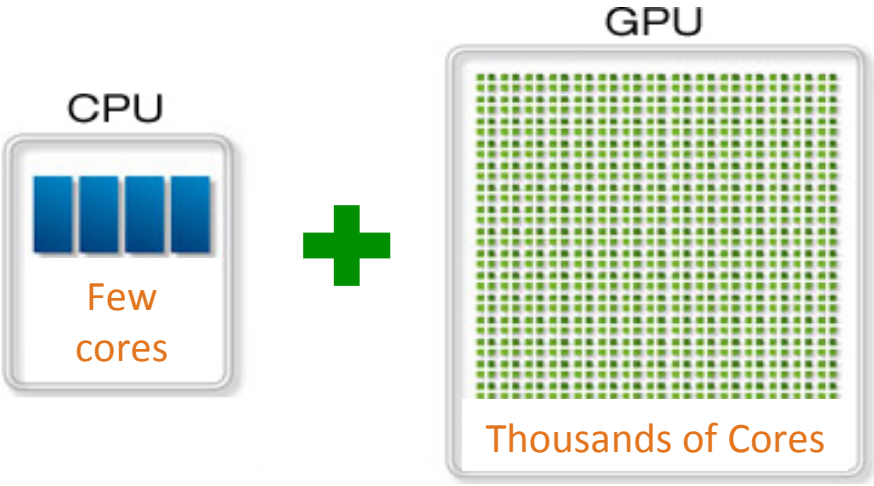
still static

- Putting it all together
- Take home message and open questions

to the office

Disclaimer …

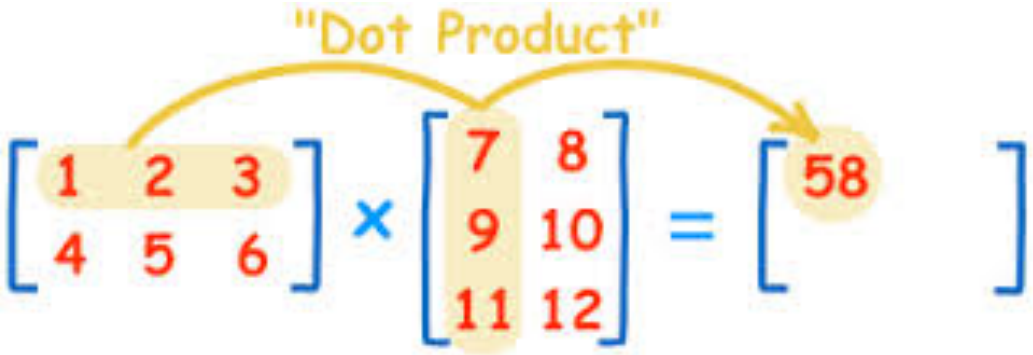
- A heterogeneous platform = a CPU + a GPU (the starting point)
- An application workload = an application + its input dataset
- Workload partitioning = workload distribution among the processing units of a heterogeneous system



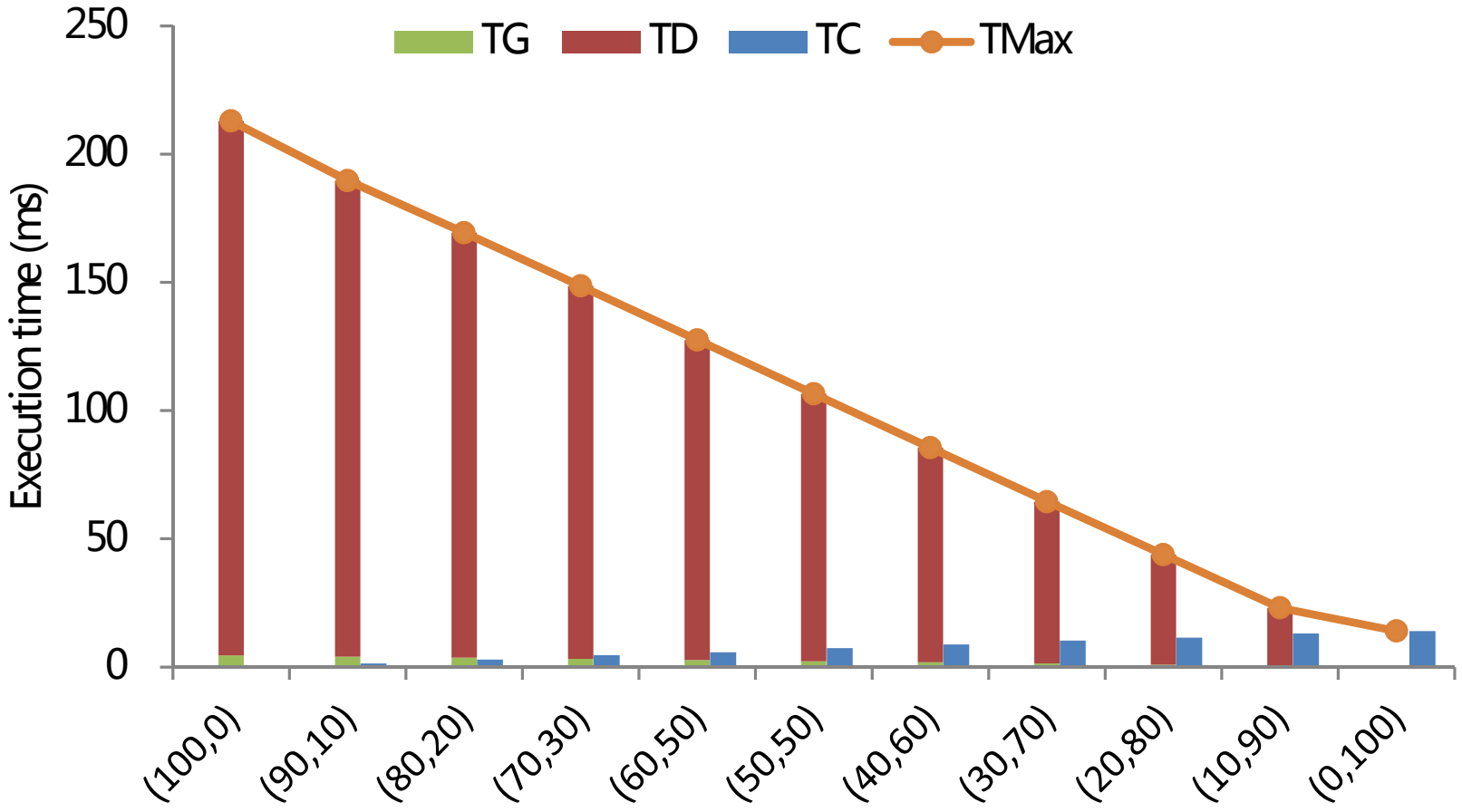
Example 1

- Dot product
  - Compute the dot product of 2 (1D) arrays
- Performance
  - T_G = execution time on GPU
  - T_C = execution time on CPU
  - T_D = data transfer time CPU-GPU

- GPU best or CPU best?

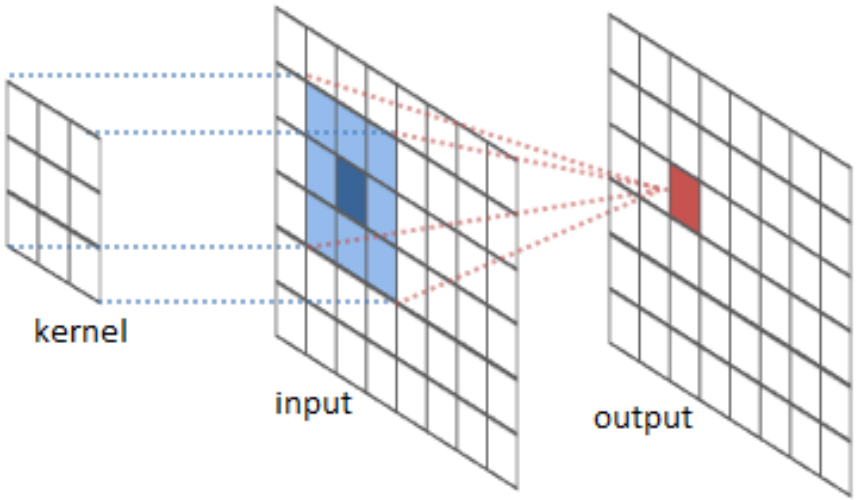


Example 1 : dot product

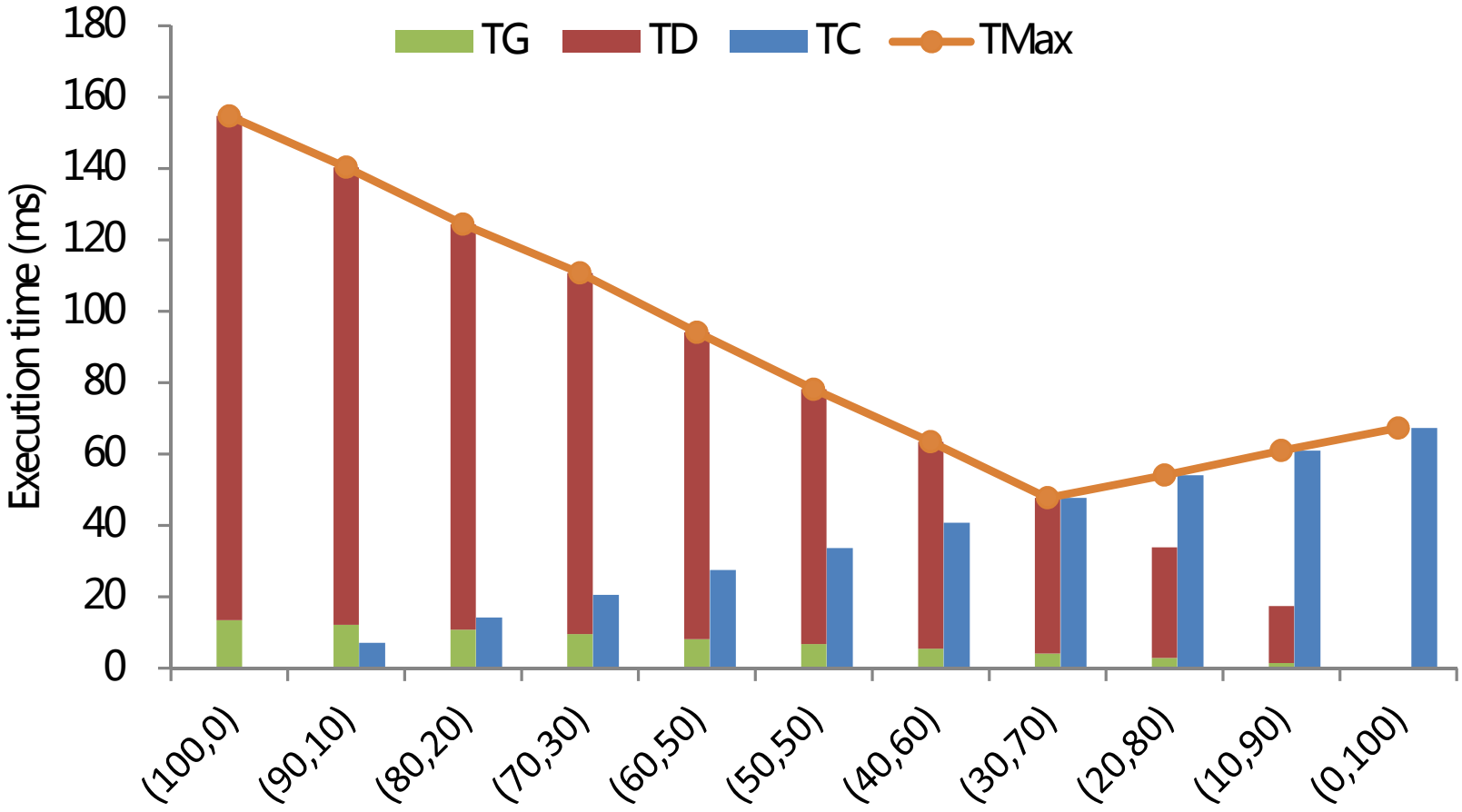


Example 2: separable convolution

- Separable convolution (CUDA SDK)
  - Apply a convolution filter (kernel) on a large image.
  - Separable kernel allows applying
    - Horizontal first
    - Vertical second
- Performance
  - T_G = execution time on GPU
  - T_C = execution time on CPU
  - T_D = data transfer time
- GPU best or CPU best?

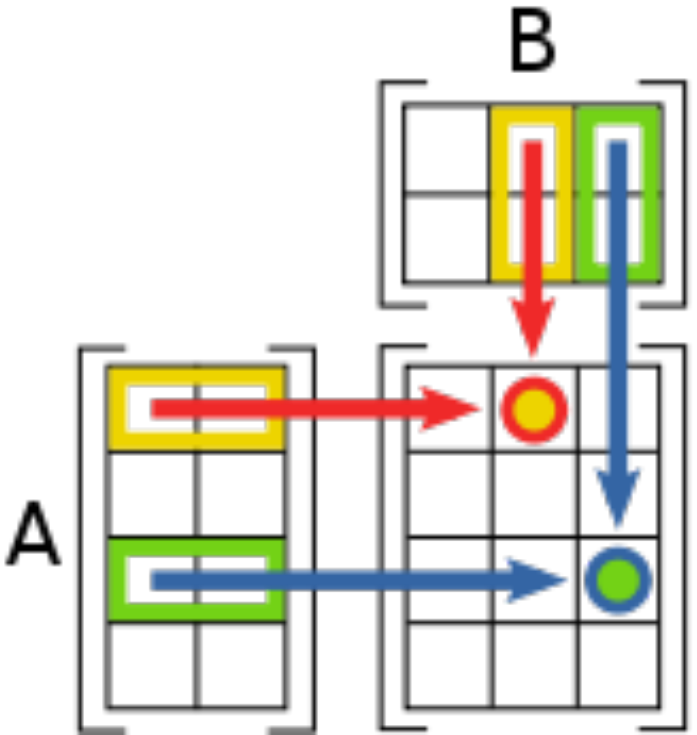


Example 2: separable convolution

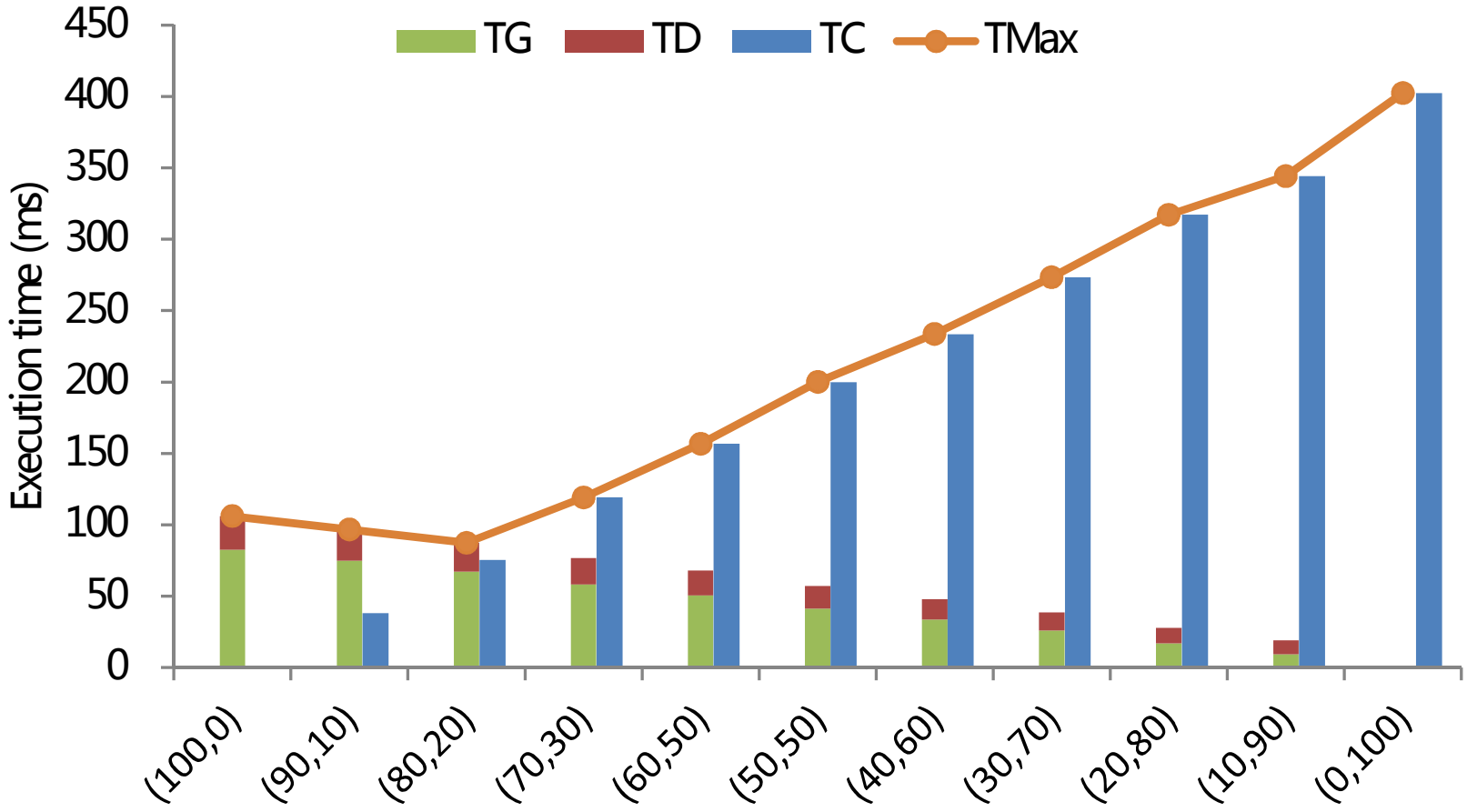


Example 3

- Matrix multiply
  - Compute the product of 2 matrices
- Performance
  - T_G = execution time on GPU
  - T_C = execution time on CPU
  - T_D = data transfer time CPU-GPU
- GPU best or CPU best?



Example 3 : matrix multiply



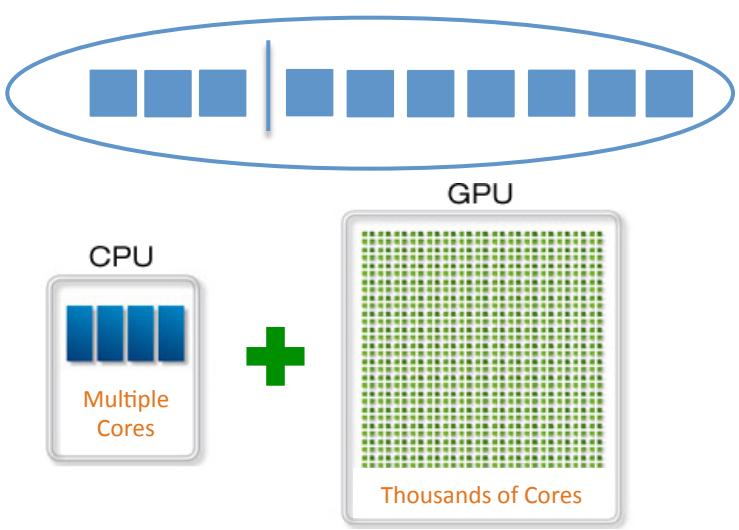
So …

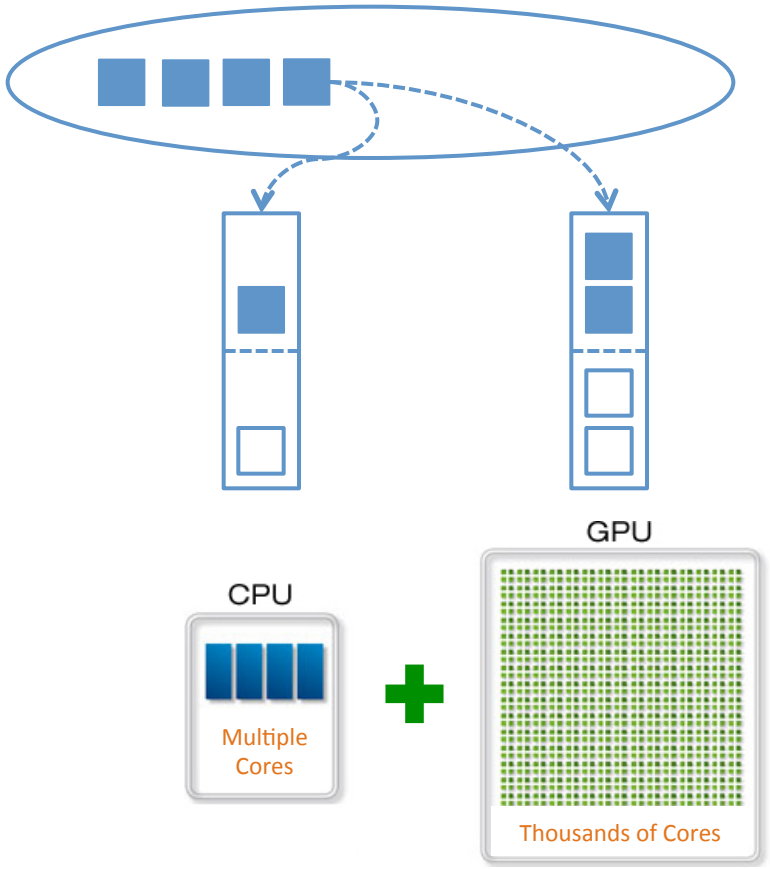
- There are **very** few GPU-only applications
  - CPU – GPU communication bottleneck.
  - Increasing performance of CPUs
- A part of the computation can be done by the CPU.
  - How to program? (ask me more)
  - Which part?

Main challenges: programming and workload partitioning!

Determining the partition

- Static partitioning (SP) vs. Dynamic partitioning (DP)



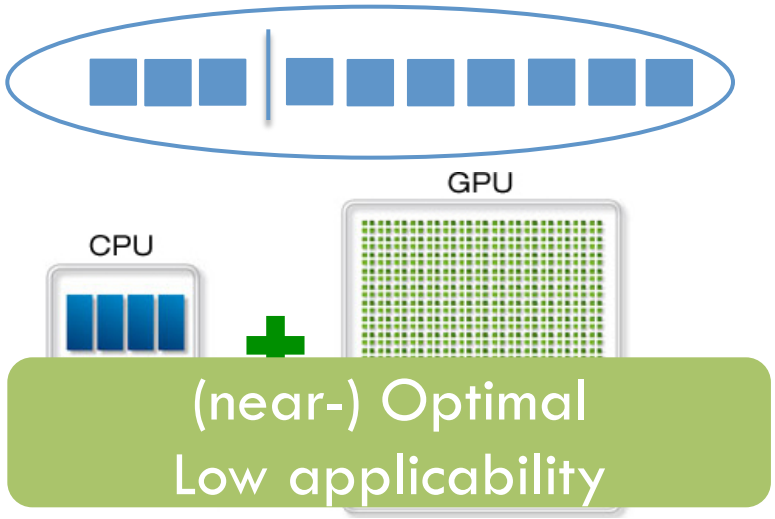


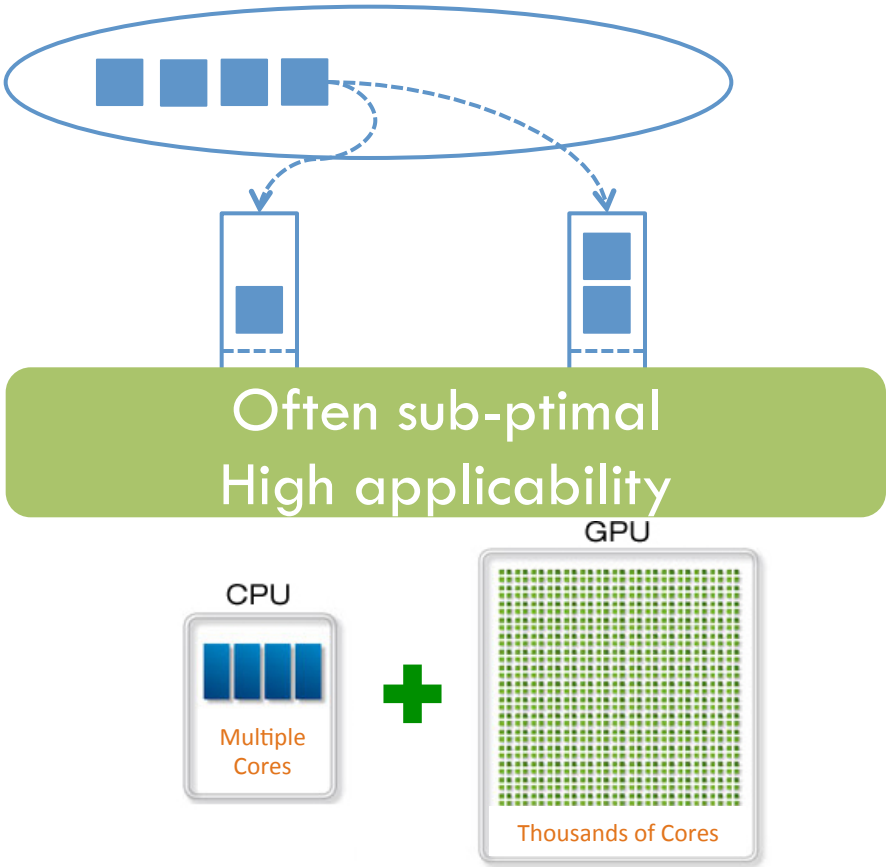
Static vs. dynamic

- Static partitioning
  - + can be computed before runtime => no overhead
  - + can detect GPU-only/CPU-only cases
  - + no unnecessary CPU-GPU data transfers
  - -- does not work for all applications
- Dynamic partitioning
  - + responds to runtime performance variability
  - + works for all applications
  - -- incurs (high) runtime scheduling overhead
  - -- might introduce (high) CPU-GPU data-transfer overhead
  - -- might not work for CPU-only/GPU-only cases

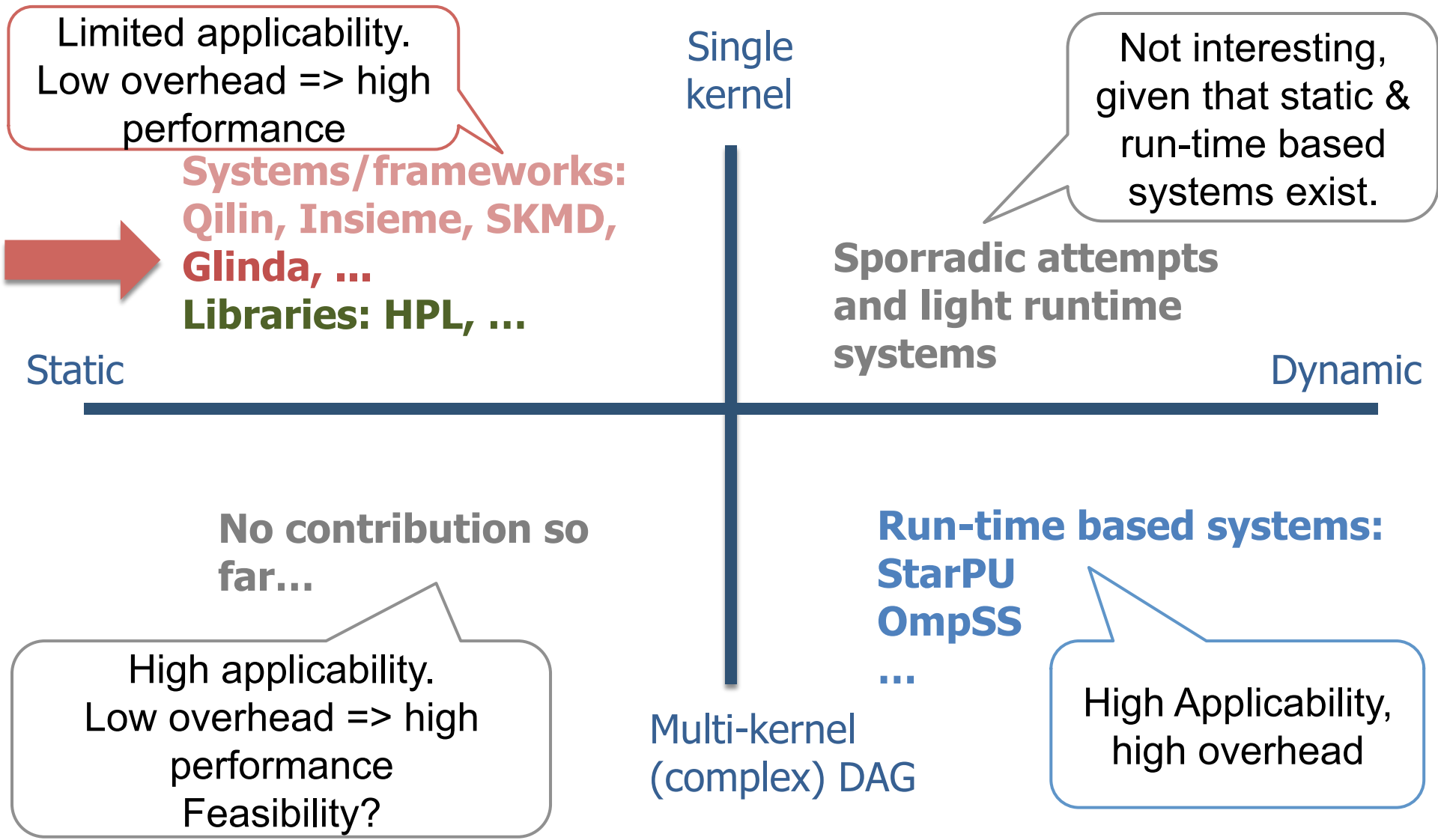
Determining the partition

- Static partitioning (SP) vs. Dynamic partitioning (DP)





Heterogeneous Computing today



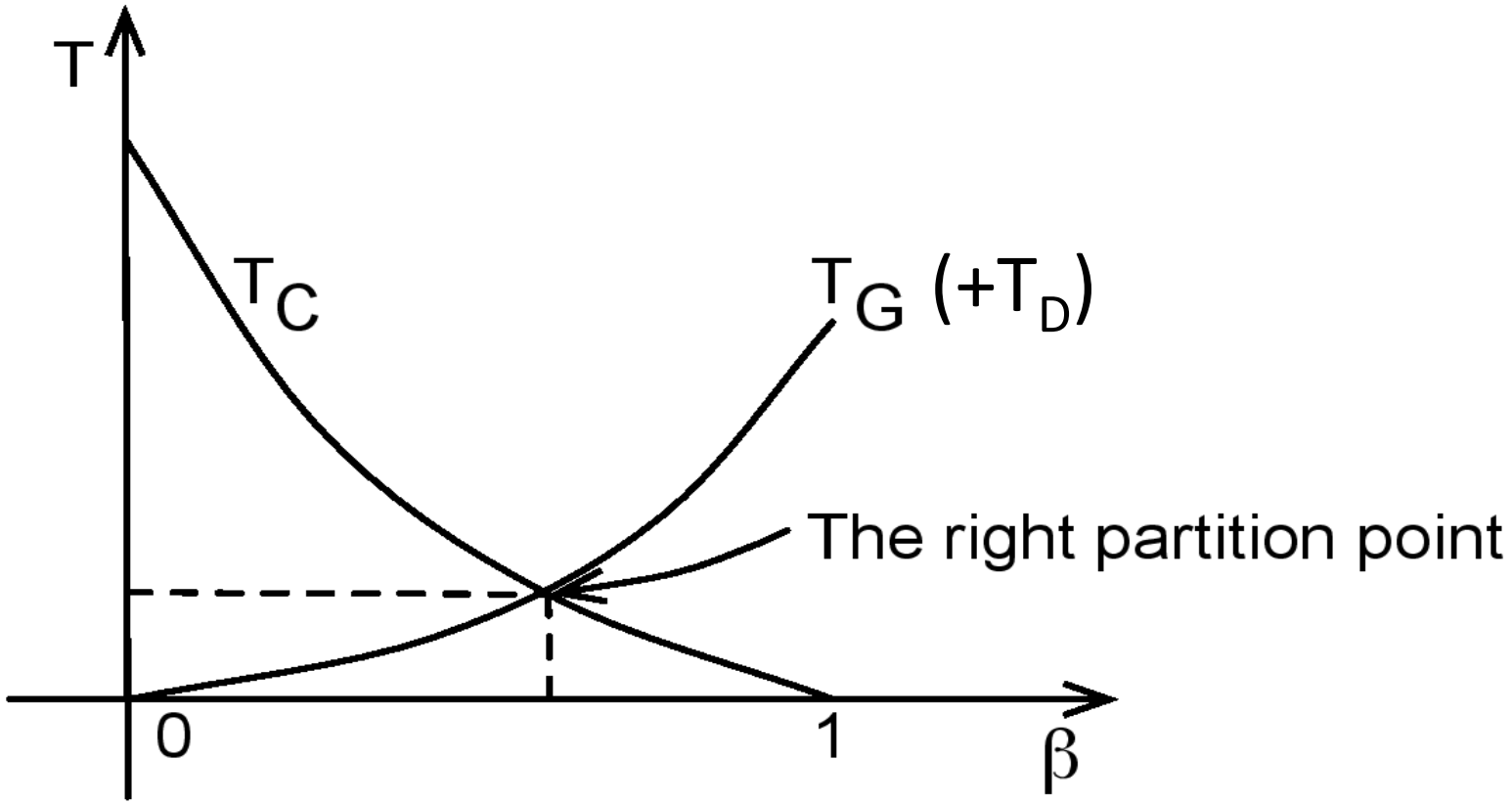
Glinda: our approach*

- Modeling the partitioning
  - The application workload
  - The hardware capabilities
  - The GPU-CPU data transfer
- Predict the optimal partitioning
- Making the decision in practice
  - Only-GPU
  - Only-CPU
  - CPU+GPU with the optimal partitioning

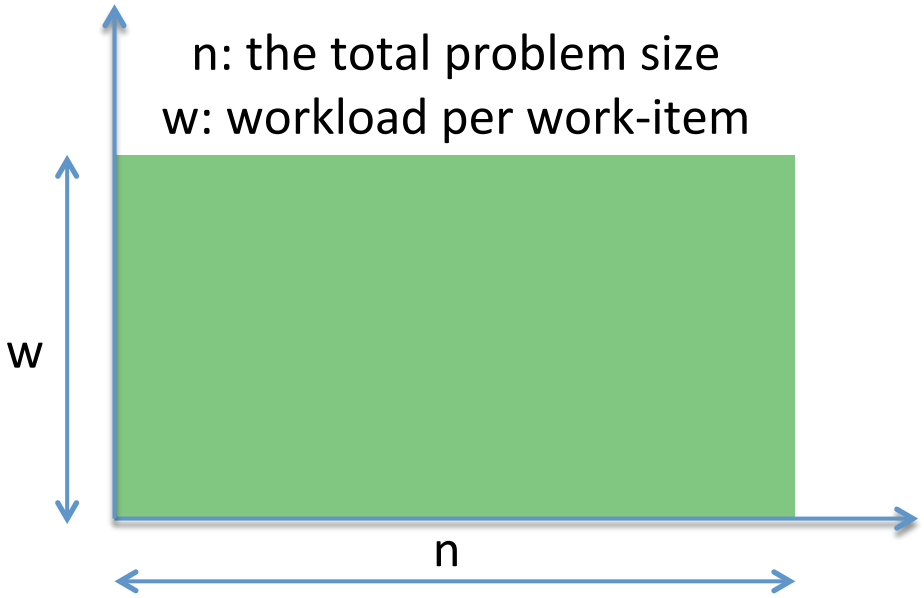
*Jie Shen et al., HPCC'14.
"Look before you Leap: Using the Right Hardware.
Resources to Accelerate Applications

Modeling the partitioning

- Define the optimal (static) partitioning $T_G + T_D = T_C$
  - β= the fraction of data points assigned to the GPU

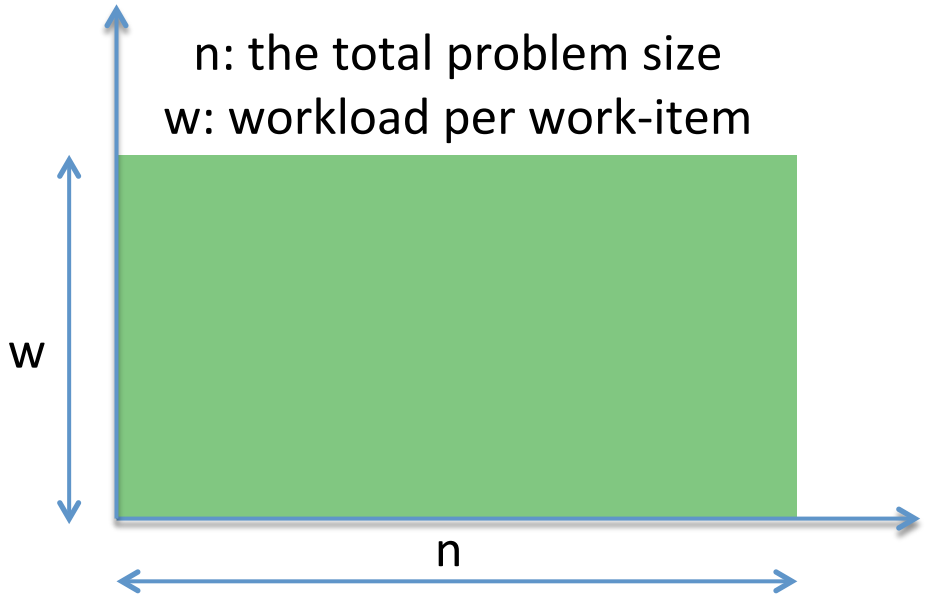


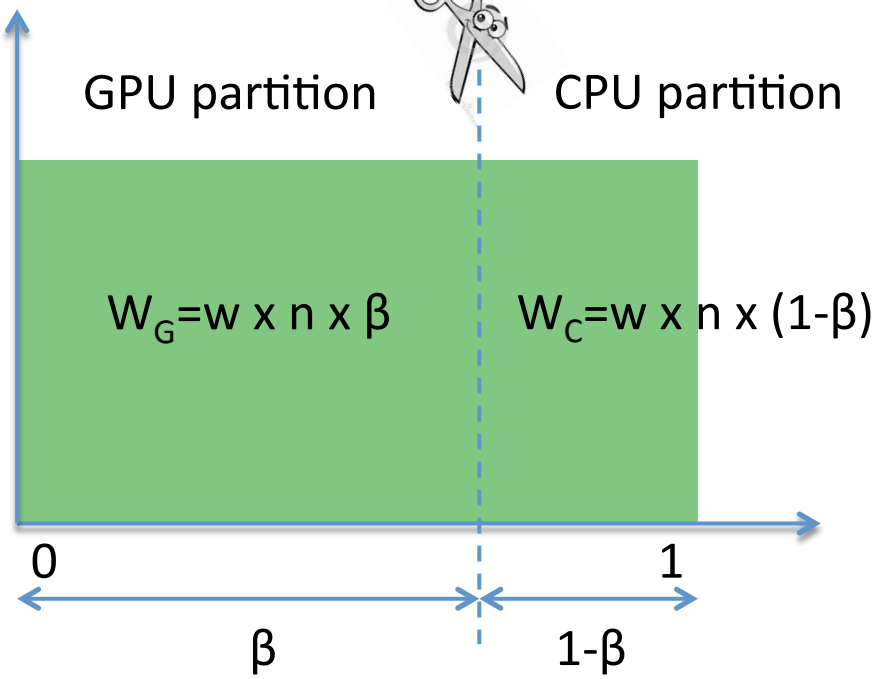
Model the app workload



Model the app workload

$$W = \sum_{i=0}^{n-1} w_i \approx \text{the area of the rectangle}$$





W (total workload size) quantifies how much work has to be done

Modeling the partitioning

$$T_G = \frac{W_G}{P_G} \quad T_C = \frac{W_C}{P_C} \quad T_D = \frac{O}{Q}$$

$$* W = W_G + W_C$$

Two pairs of metrics

***W*: total workload size**
***P*: processing throughput (*W*/second)**

O: data-transfer size
Q: data-transfer bandwidth (bytes/second)

$$T_G + T_D = T_C \quad \rightsquigarrow \quad \frac{W_G}{W_C} = \frac{P_G}{P_C} \times \frac{1}{1 + \frac{P_G}{Q} \times \frac{O}{W_G}}$$

Model the HW capabilities

- Workload: $W_G + W_C = W$
- Execution time: T_G and T_C
- P (processing throughput)
  - Measured as workload processed per second
  - P evaluates the hardware capability of a processor

GPU kernel execution time: $T_G = W_G / P_G$
CPU kernel execution time: $T_C = W_C / P_C$

Model the data-transfer

- O (GPU data-transfer size)
  - Measured in bytes
- Q (GPU data-transfer bandwidth)
  - Measured in bytes per second

Data-transfer time: TD=O/Q + (Latency)
Latency < 0.1 ms, negligible impact

Predict the partitioning …

$$T_G = \frac{W_G}{P_G} \quad T_C = \frac{W_C}{P_C} \quad T_D = \frac{O}{Q}$$

$$T_G + T_D = T_C \quad \rightsquigarrow \quad \frac{W_G}{W_C} = \frac{P_G}{P_C} \times \frac{1}{1 + \frac{P_G}{Q} \times \frac{O}{W_G}}$$

… by solving this equation in β

Predict the partitioning

- Solve the equation

$$\frac{W_G}{W_C} = \frac{P_G}{P_C} \times \frac{1}{1 + \frac{P_G}{Q} \times \frac{O}{W_G}}$$

β-dependent terms

Expression

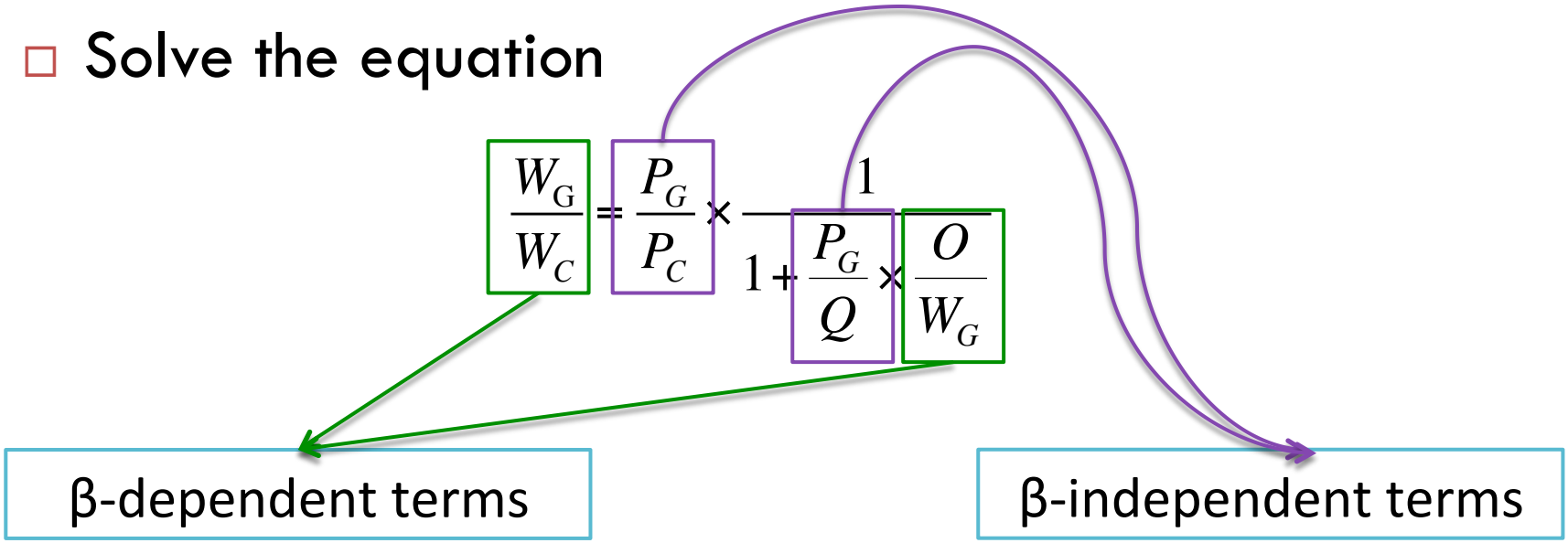
$W_G = w \times n \times \beta$

$W_C = w \times n \times (1-\beta)$

O=Full data transfer or
Full data transfer x β

Predict the partitioning

- Solve the equation



Expression

Estimation

W_G=w x n x β

W_C=w x n x (1-β)

O=Full data transfer or
Full data transfer x β

Modeling the partitioning

- Estimating the HW capability ratios by using profiling
  - The ratio of GPU throughput to CPU throughput
  - The ratio of GPU throughput to data transfer bandwidth

$$\frac{W_G}{W_C} = \frac{P_G}{P_C} \times \frac{1}{1 + \frac{P_G}{Q} \times \frac{O}{W_G}}$$

R_{GC}

CPU kernel execution time vs. GPU kernel execution time

R_{GD}

GPU data-transfer time vs. GPU kernel execution time

Predicting the optimal partitioning

- Solving β from the equation

Total workload size
HW capability ratios
Data transfer size

$$\frac{W_G}{W_C} = \frac{P_G}{P_C} \times \frac{1}{1 + \frac{P_G}{Q} \times \frac{O}{W_G}}$$

→ β predictor

- There are three β predictors (by data transfer type)

$$\beta = \frac{R_{GC}}{1 + R_{GC}}$$

No data transfer

$$\beta = \frac{R_{GC}}{1 + \frac{v}{w} \times R_{GD} + R_{GC}}$$

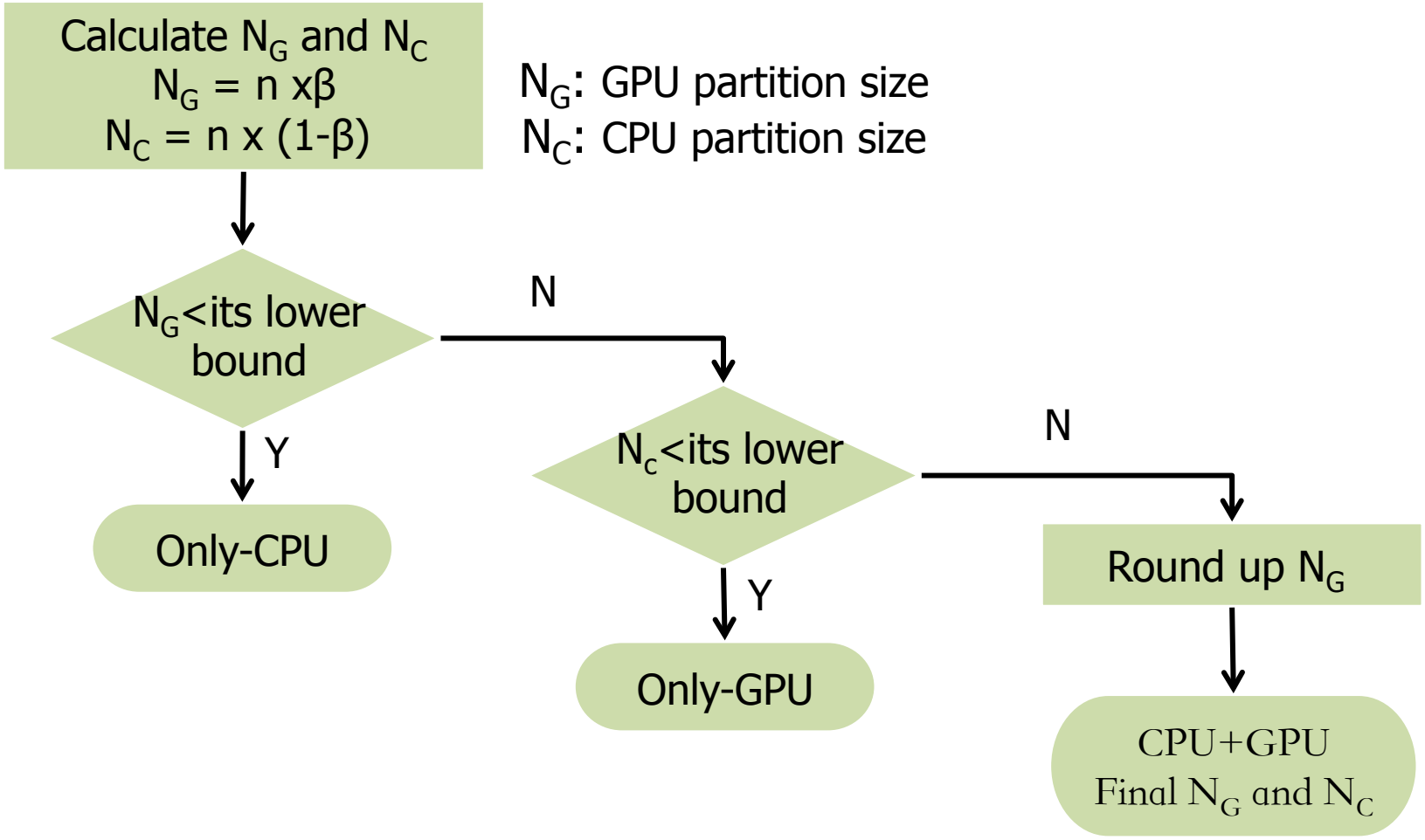
Partial data transfer

$$\beta = \frac{R_{GC} - \frac{v}{w} \times R_{GD}}{1 + R_{GC}}$$

Full data transfer

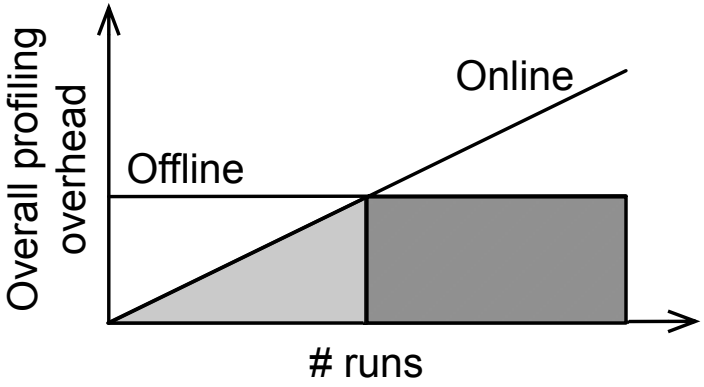
Making the decision in practice

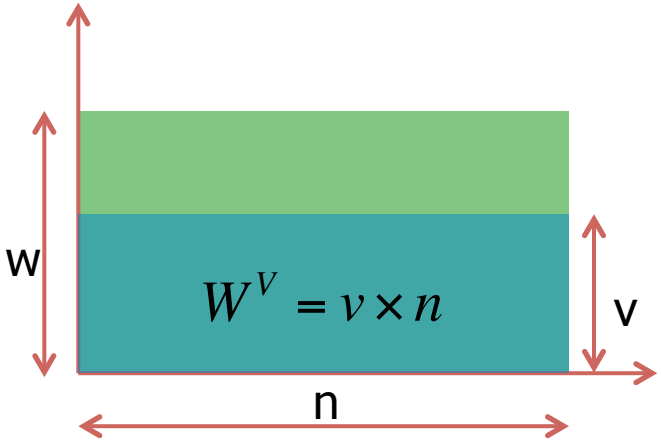
- From β to a practical HW configuration



Extensions

- Different profiling options
  - Online vs. Offline profiling
  - Partial vs. Full profiling



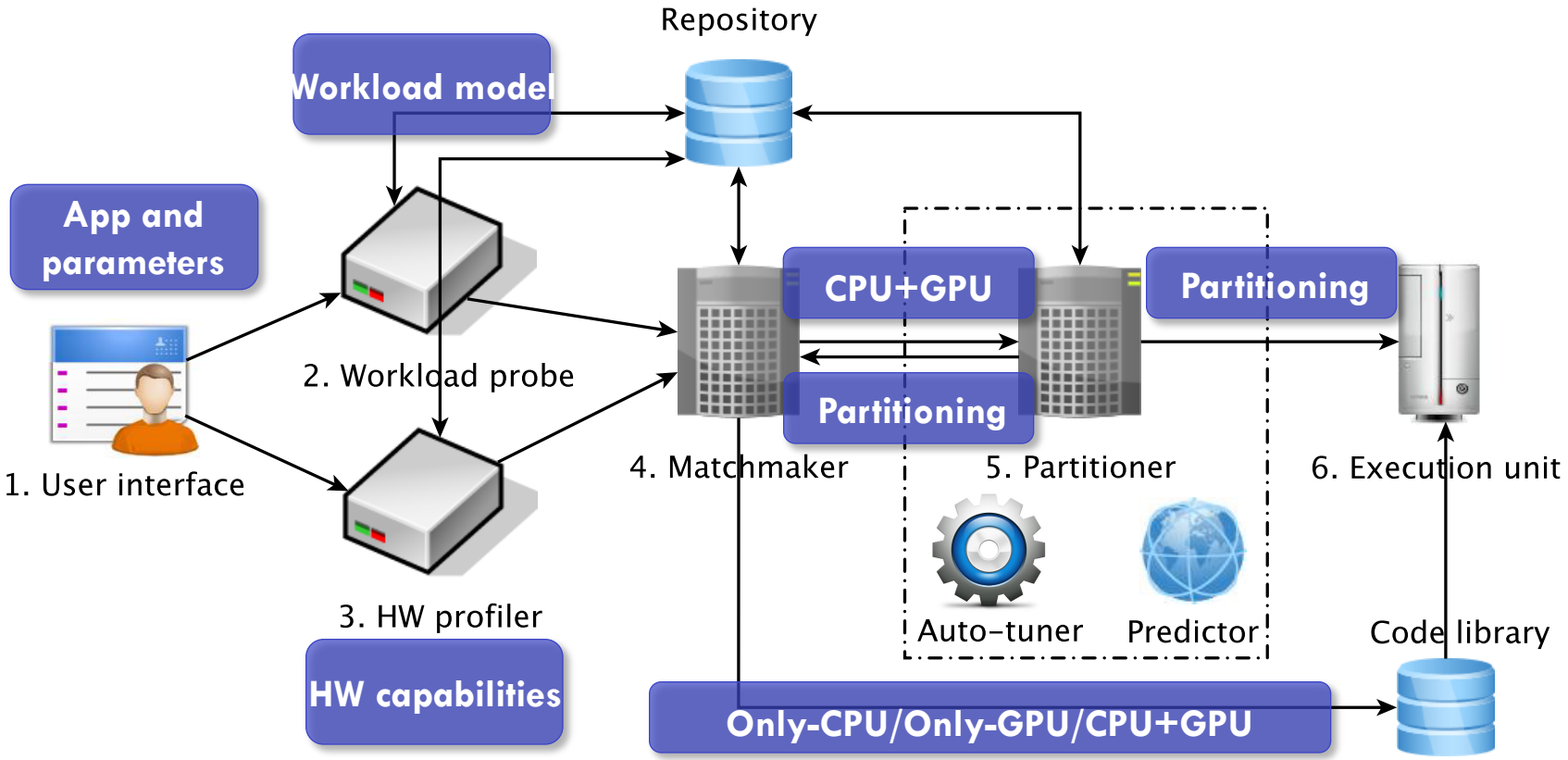


profile partial workload W^v

$$v_{\min} \le v \le w$$

- CPU+Multiple GPUs
  - Identical GPUs
  - Non-identical GPUs (may be suboptimal)

The Glinda framework



Glinda outcome

- (Any?) data-parallel application can be transformed to support heterogeneous computing

- A decision on the execution of the application
  - only on the CPU
  - only on the GPU
  - CPU+GPU
    - And the partitioning point

How to use Glinda?

- Profile the platform: RGC, RGD
- Configure and use the solver: β
- Take the decision: Only-CPU, Only-GPU, CPU+GPU (and partitioning)
  - if needed, apply the partitioning

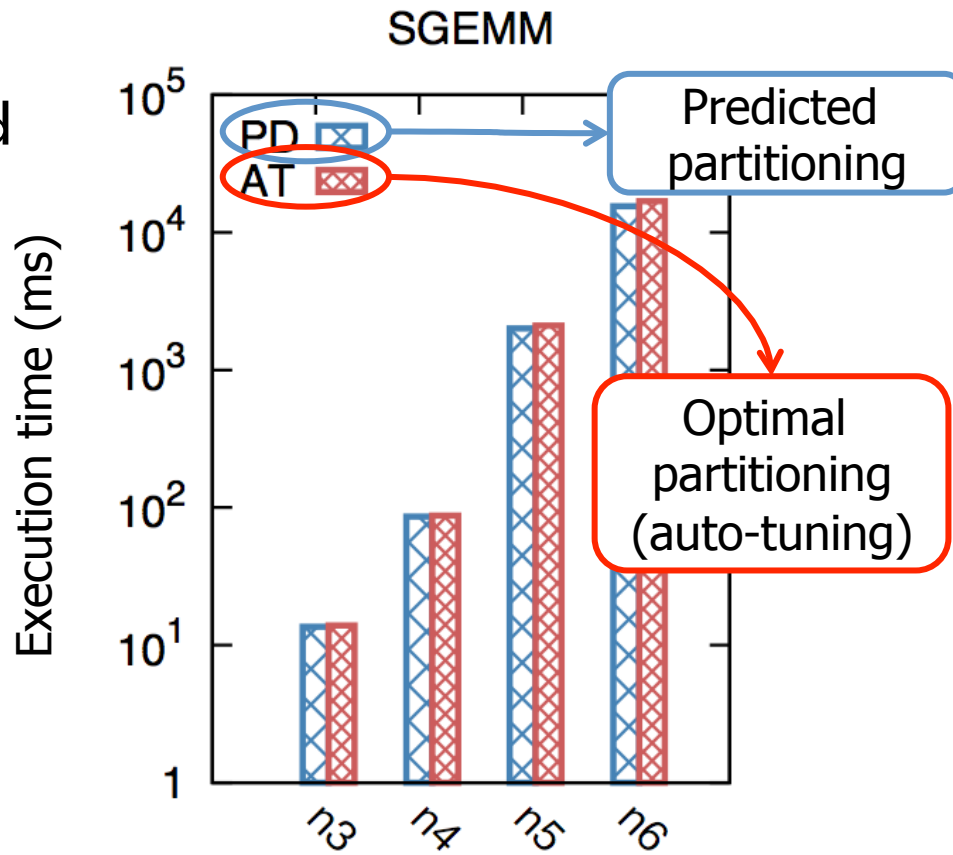
- Code preparation
  - Parallel implementations for both CPUs and GPUs
  - Enable profiling and partitioning
- Code reuse
  - Single-device code and multi-device code are reusable for different datasets and HW platforms

DEMO

Results [1]

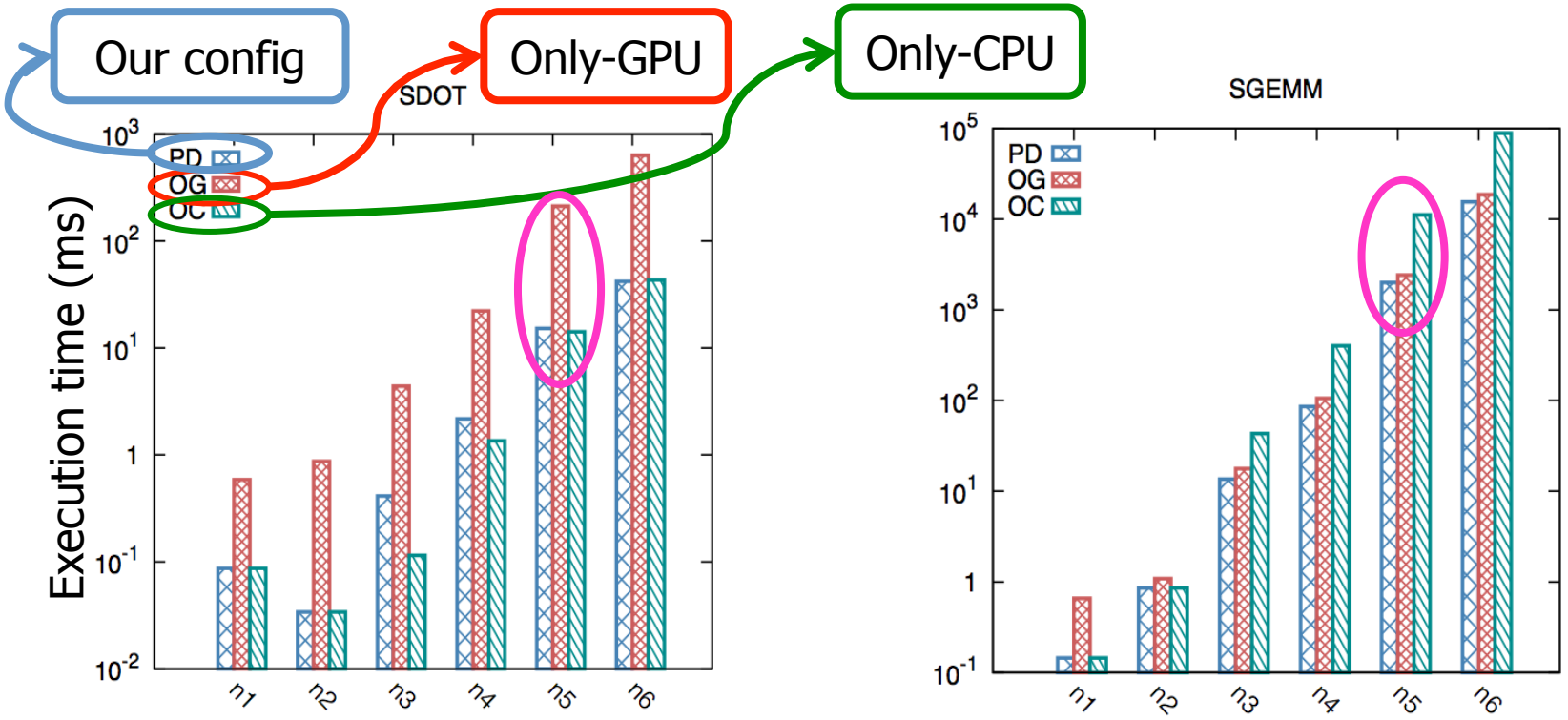
- Quality (compared to an oracle)
  - The right HW configuration in 62 out of 72 test cases
  - Optimal partitioning

  when CPU+GPU is selected



Results [2]

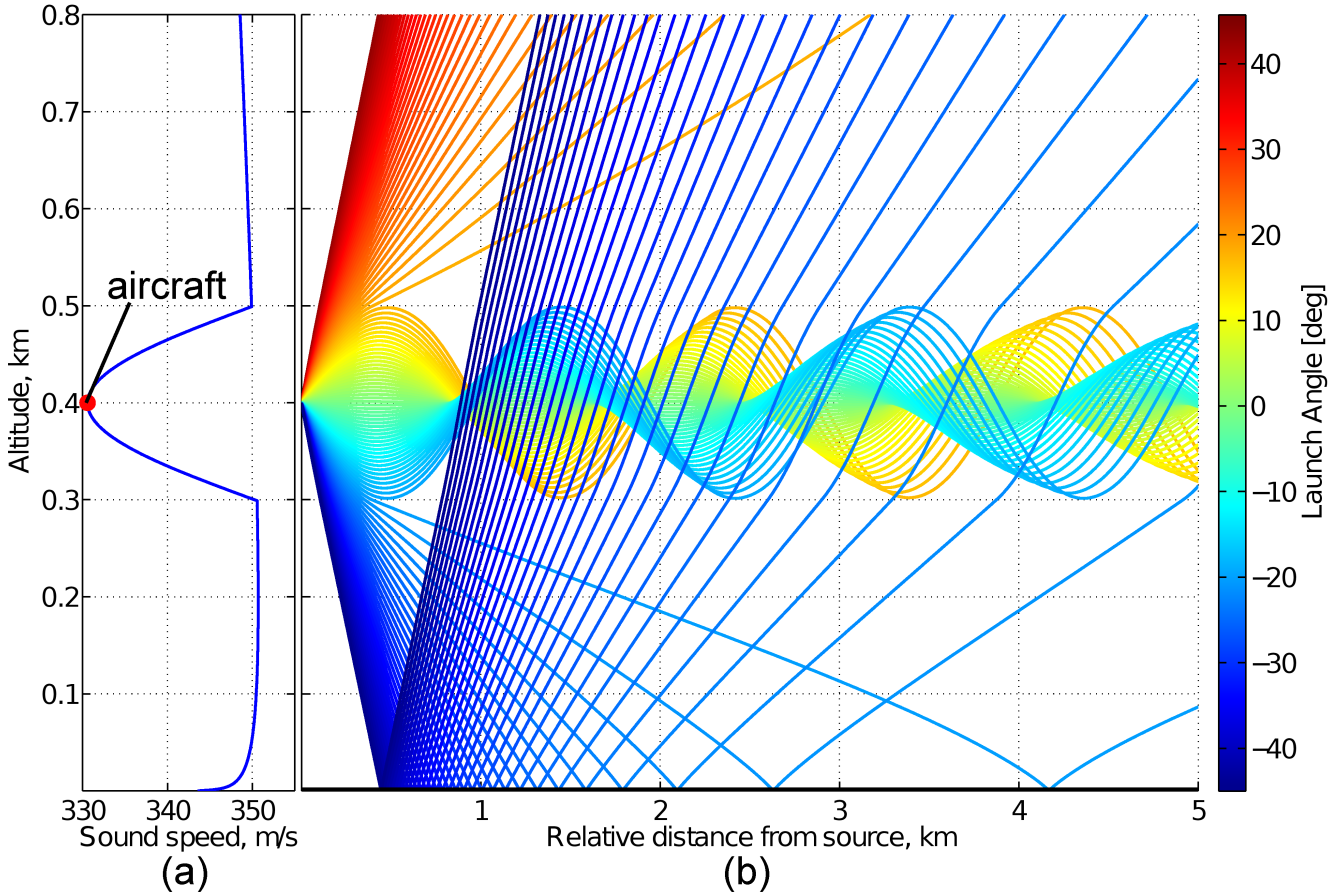
- Effectiveness (compared to Only-CPU/Only-GPU)
  - 1.2x-14.6x speedup
  - If taking GPU for granted, up to 96% performance will be lost



BONUS: imbalanced applications

Sound ray tracing

Sound ray tracing



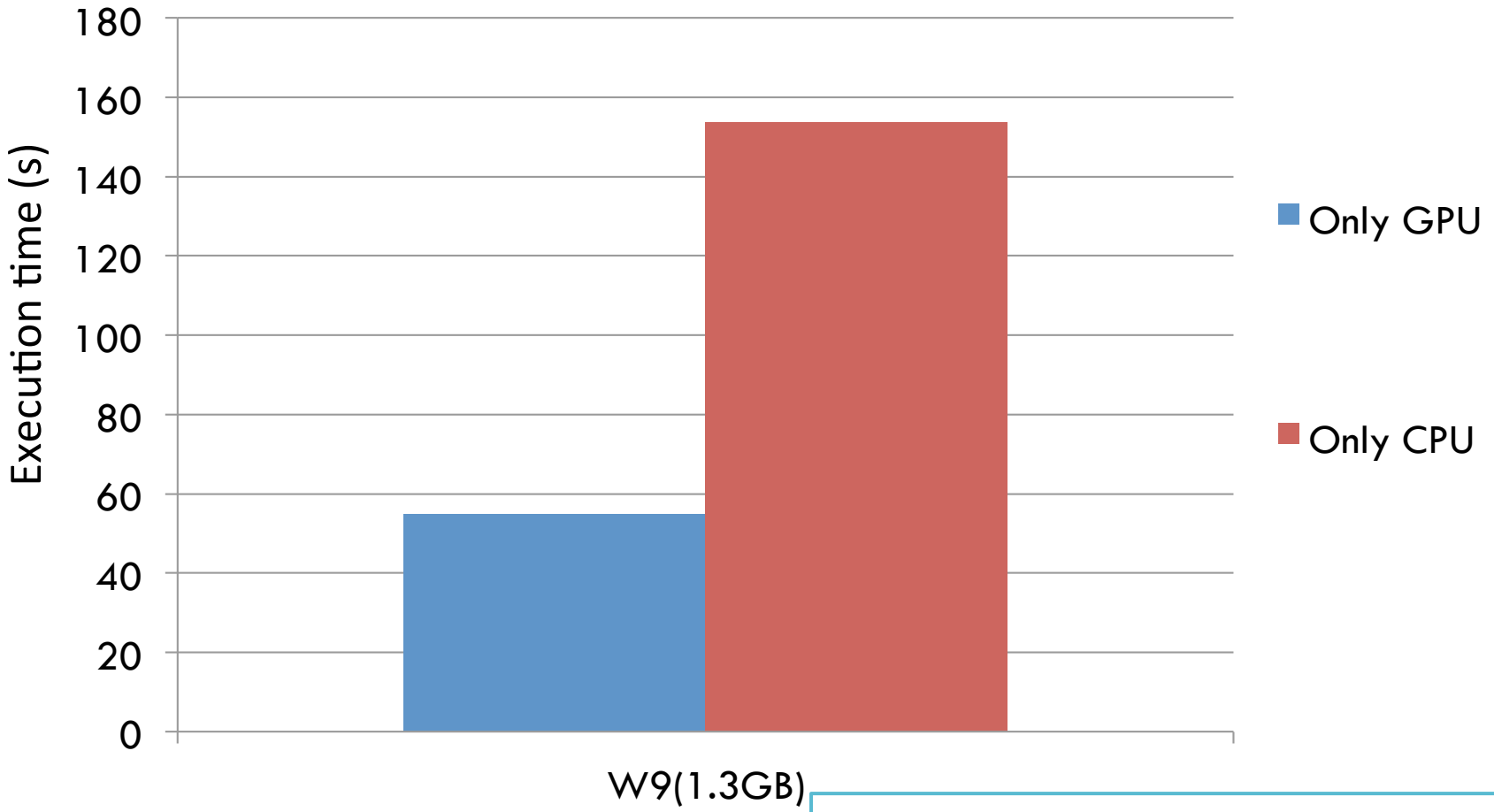
Which hardware?

Our application has …

- Massive data-parallelism …
- No data dependency between rays …
- Compute-intensive per ray …

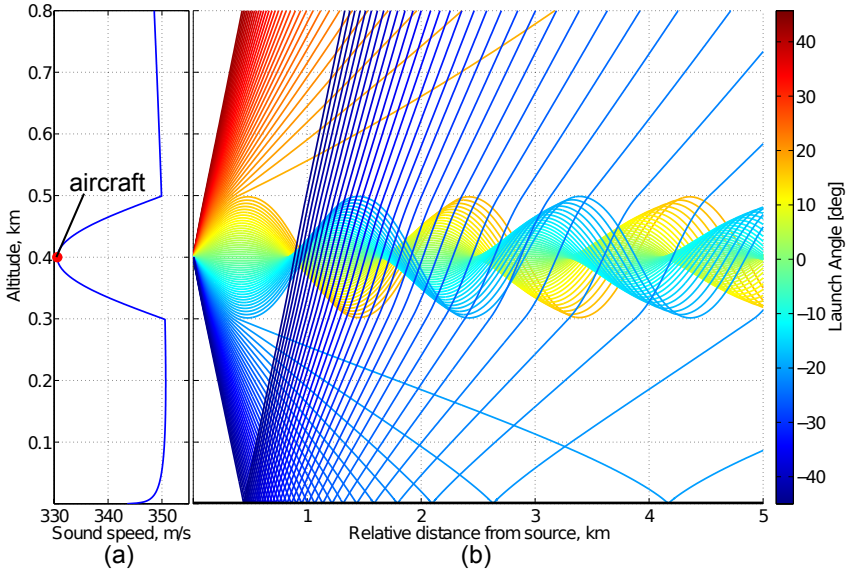
… clearly, this is a perfect GPU workload !!!

Results [1]

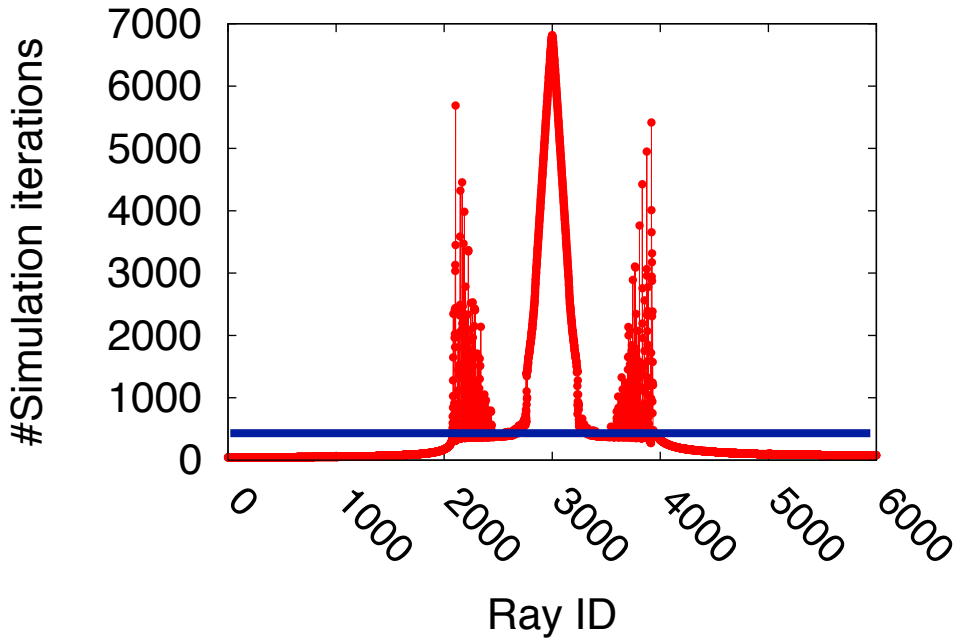


Only 2.2x performance improvement!
We expected 100x …

Workload profile

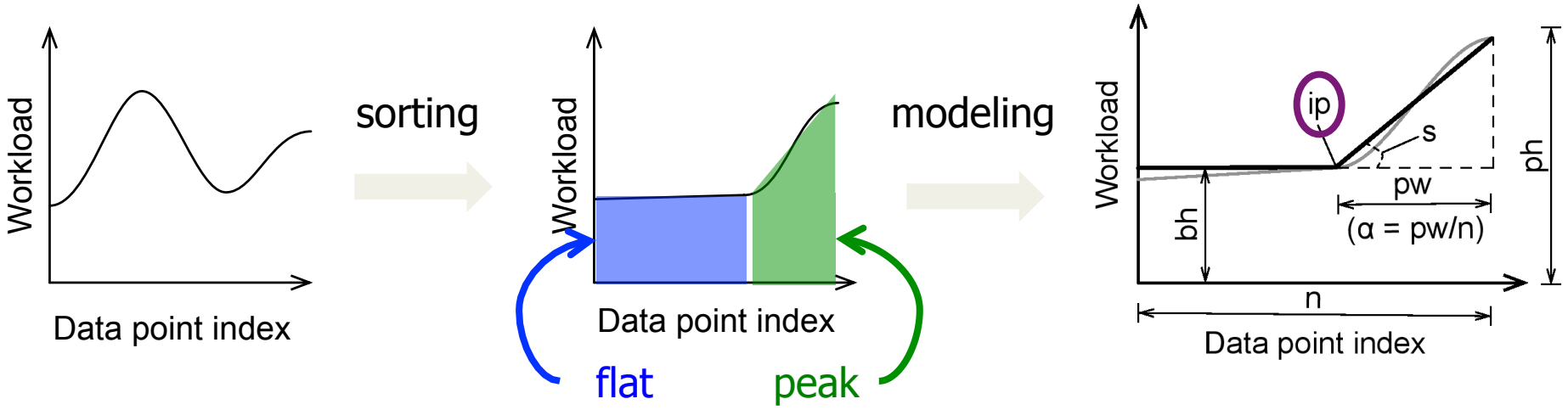


Peak
Processing iterations: ~7000

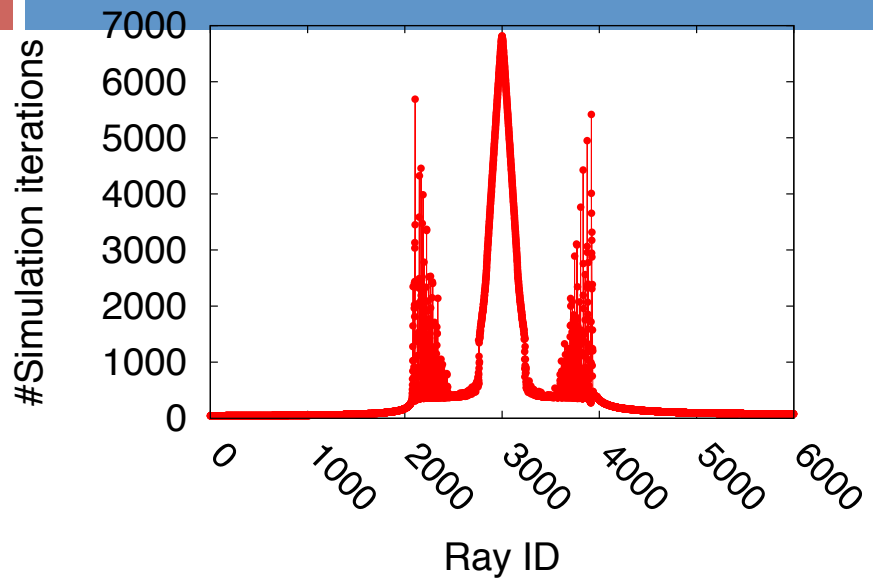


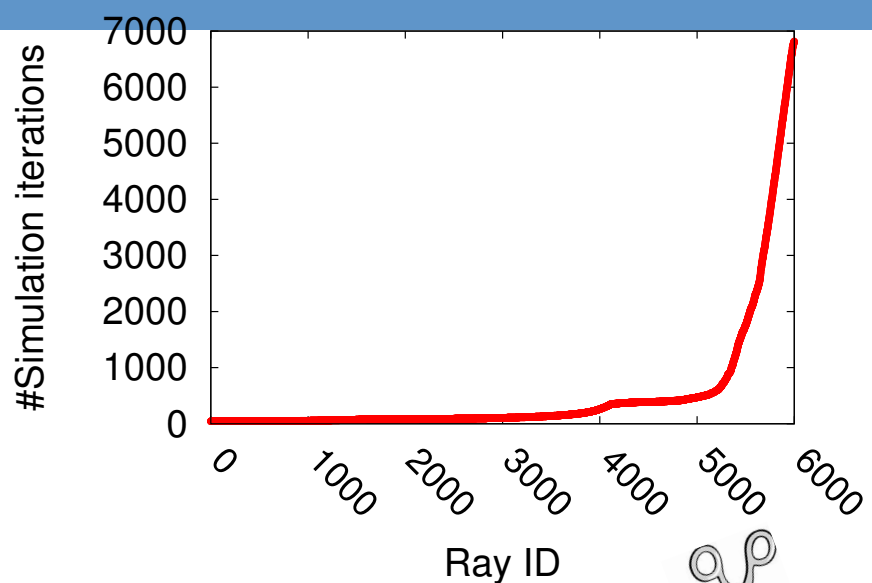
Bottom
Processing iterations: ~500

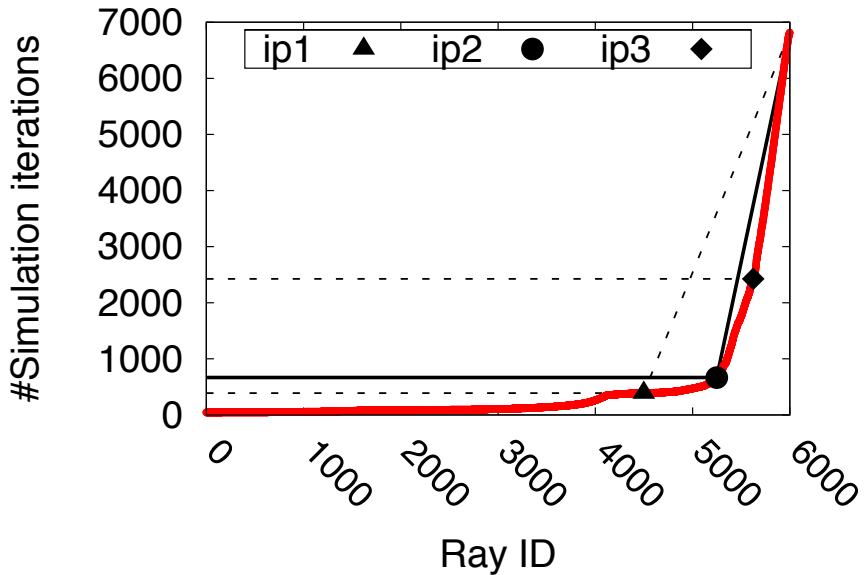
Modeling the imbalanced workload

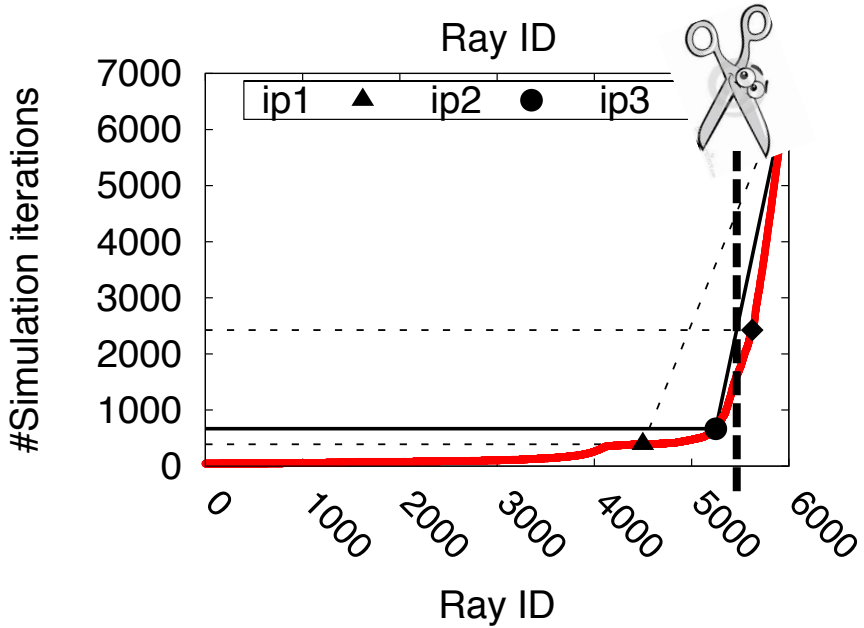


Glinda for imbalanced workload

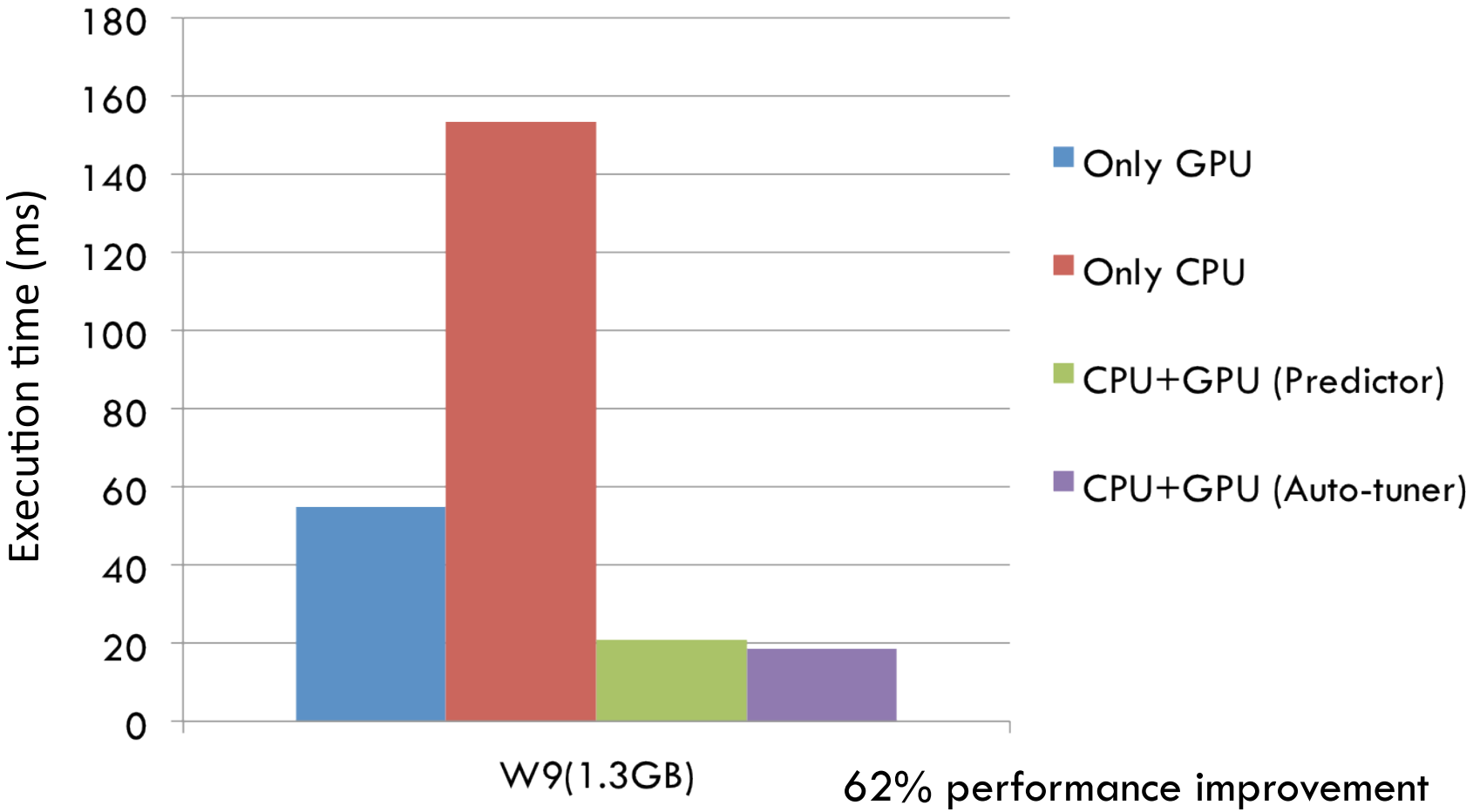








Results [2]



62% performance improvement compared to “Only-GPU”

Summary: single-kernel static partitioning

- It targets single-kernel data parallel applications
- It computes a static partitioning before runtime
- The challenge is to determine the optimal partitioning by building prior knowledge
  - We are not the only one
    - Online-profiling + analytical modeling: Ginda
    - Offline-training + analytical modeling (curve fitting): Glinda, Qilin
    - Offline-training + machine learning: Insieme, the work from U. Edingburgh

Related work

		Machine learning [1,2]	Qilin [3]	Ours	
				Online	Offline
Cost (in relative	Collection	+++	++/+ (depending on *m*)	~	+
			(depending on *m*)		
Performance		[1]: 85% [2] with SVM: 83.5% [2] with ANN: 87.5% (of the approximated optimal)	94% (of the approximated optimal)	91% (of the optimal)	90% (of the optimal)

Achieve similar or better performance than the other partitioning approaches with less cost

More advantages …

- Support different types of heterogeneous platforms
  - Multi-GPUs, identical or non-identical
- Support different profiling options suitable for different execution scenarios
  - Online or offline
  - Partial of full
- Determine not only the optimal partitioning but also the best hardware configuration
  - Only-GPU / Only-CPU / CPU+GPU with the optimal partitioning
- Support both balanced and imbalanced applications

Dynamic partitioning?

Multi-kernel applications?

- Use dynamic partitioning: OmpSs, StarPU
  - Partition the kernels into chunks
  - Distribute chunks to *PUs
  - Keep data dependencies

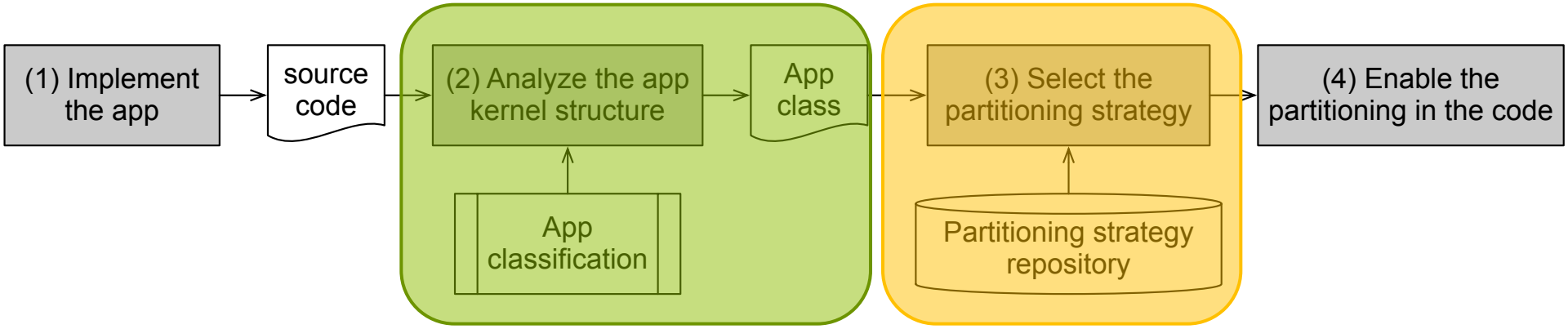
Static partitioning: low applicability, high performance.
Dynamic partitioning: low performance, high applicability.

  - Respond automatically to runtime performance variability

Can we do both?

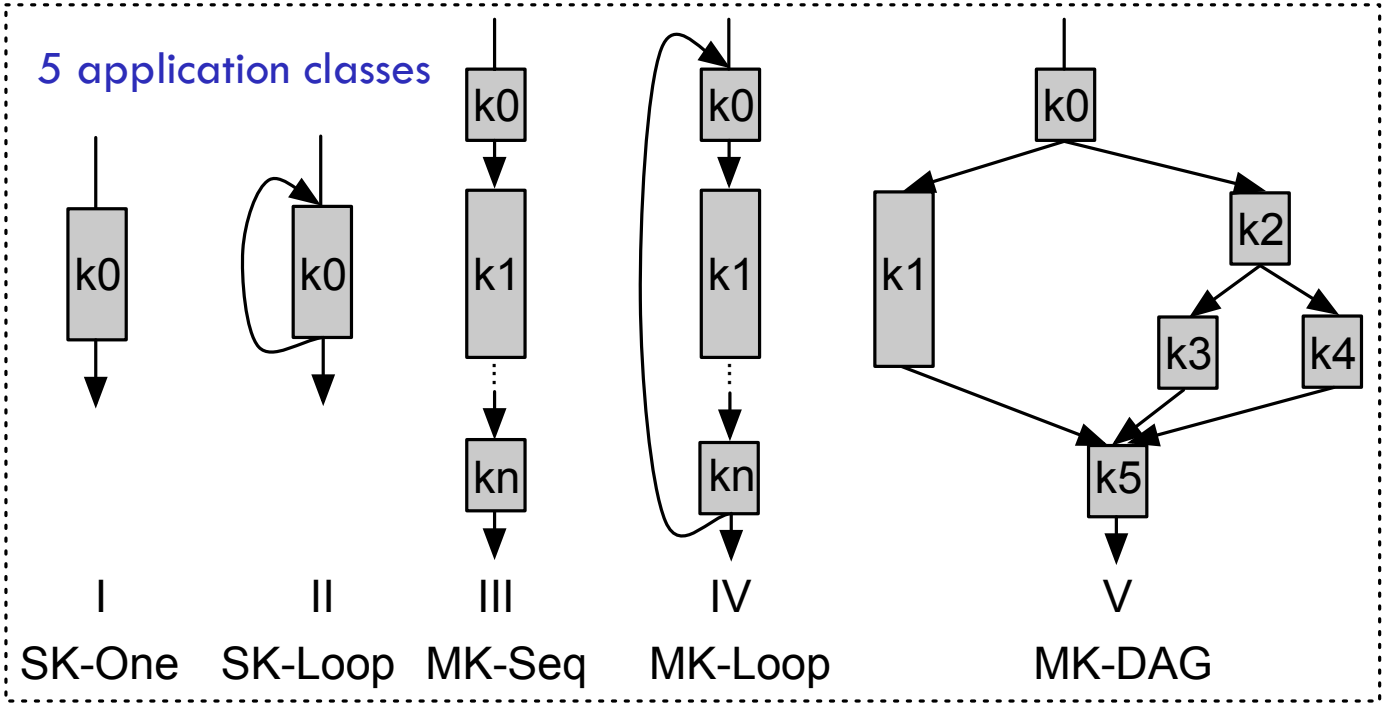
How to satisfy both requirements?

- We combine static and dynamic partitioning
  - We design an application analyzer that chooses the best performing partitioning strategy for any given application



Implement/get the source code → Analyze application's class → Select a partitioning strategy → Enable the partitioning

Application classification

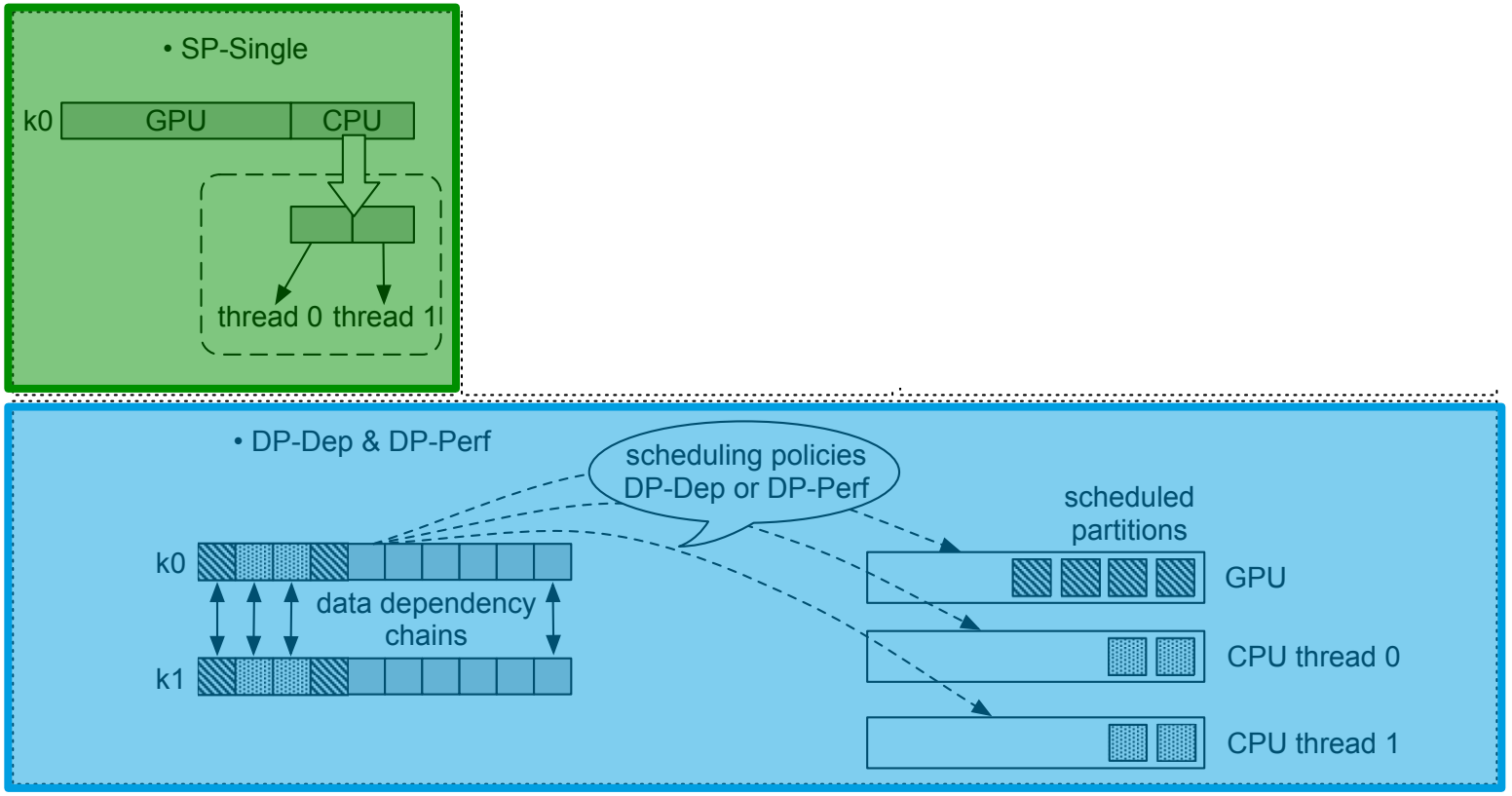


Implement/get the source code → Analyze application's class → Select a partitioning strategy → Enable the partitioning

Partitioning strategies

Static partitioning:
single-kernel applications

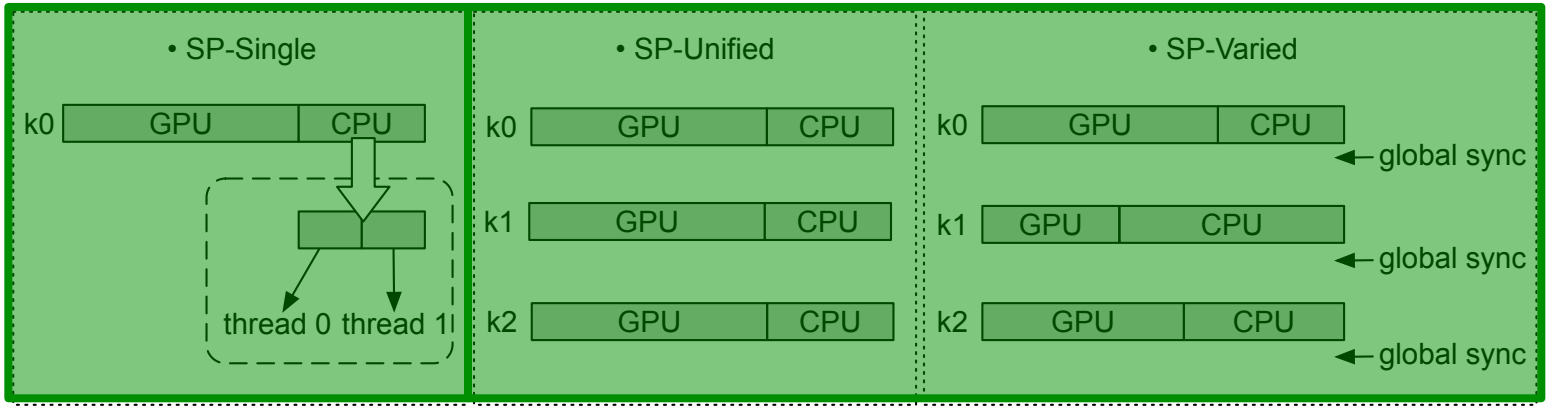
Dynamic partitioning:
multi-kernel applications

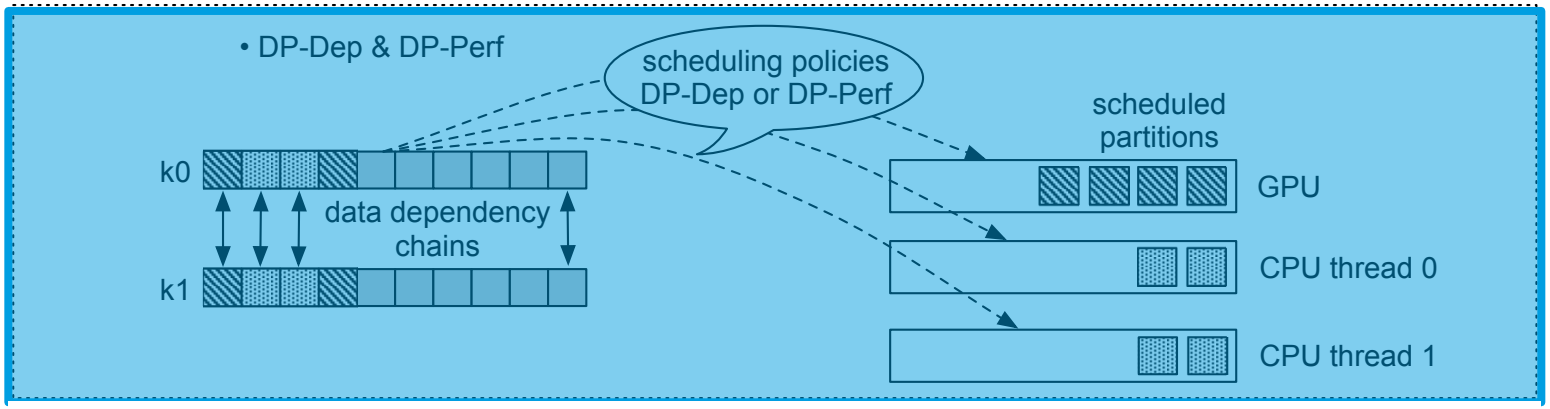


Implement/get the source code → Analyze application's class → Select a partitioning strategy → Enable the partitioning

Partitioning strategies

Static partitioning: in Glinda
single-kernel + multi-kernel applications

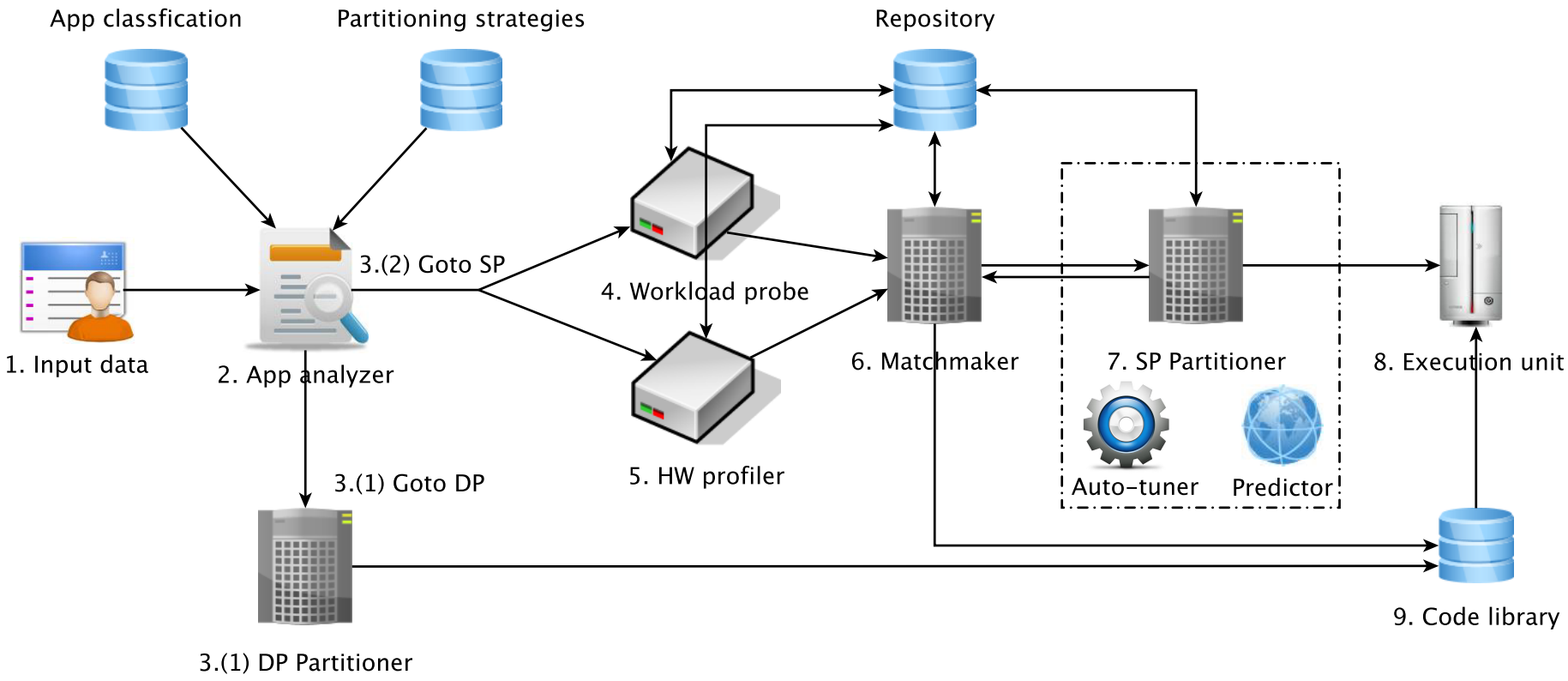




Dynamic partitioning: in OmpSs
multi-kernel applications (fall-back scenarios)

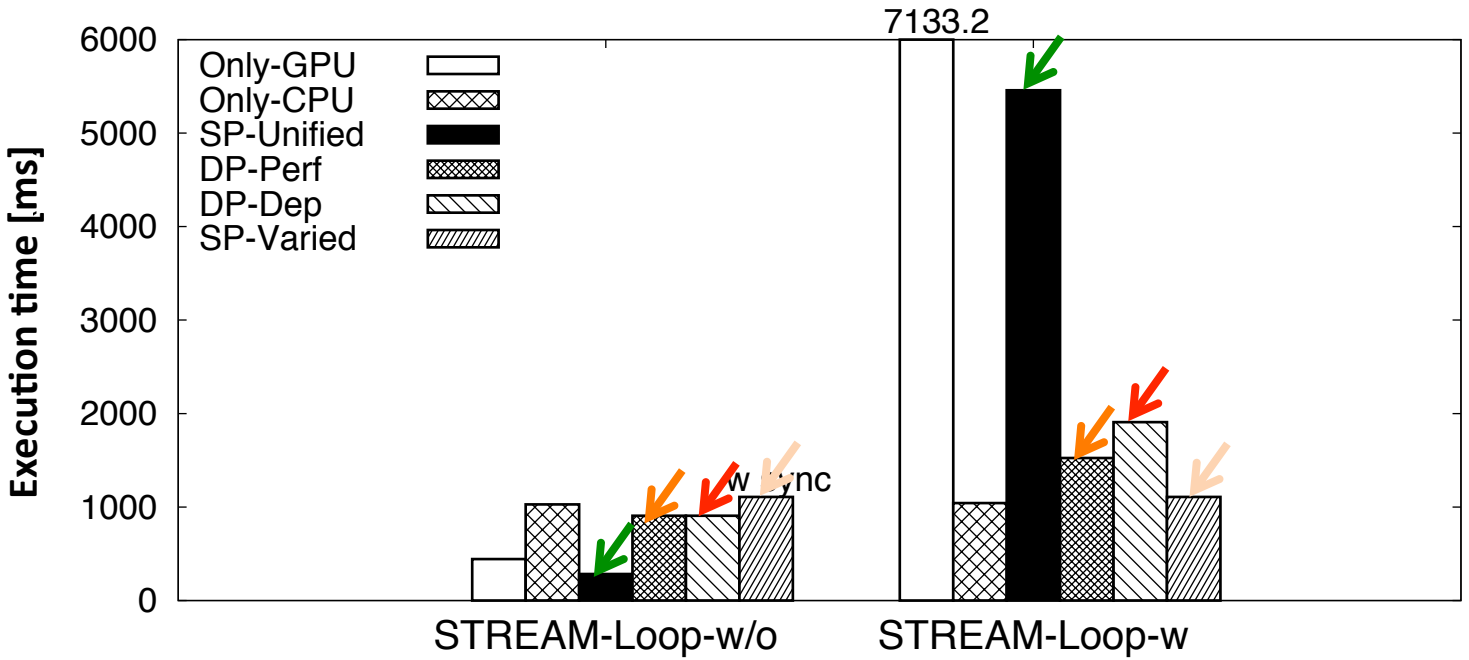
Implem
the source code
application's class
partitioning strategy
the partitioning

Putting it all together: Glinda 2.0



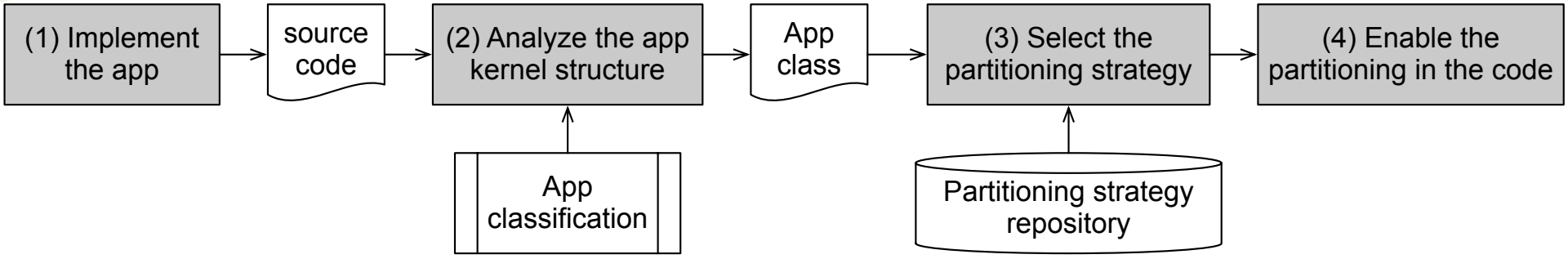
Results [4]

- MK-Loop (STREAM-Loop)
  - w/o sync: SP-Unified > DP-Perf >= DP-Dep > SP-Varied
  - with sync: SP-Varied > DP-Perf >= DP-Dep > SP-Unified



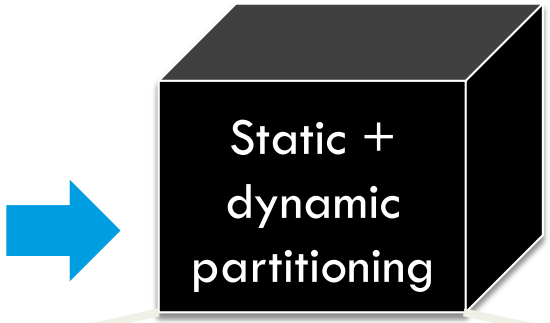
Where we are …

Static + dynamic partitioning

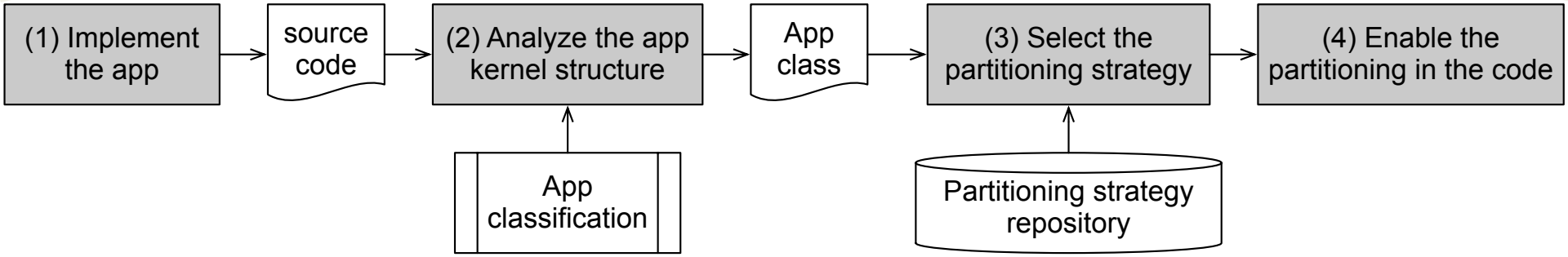


Where we are heading …

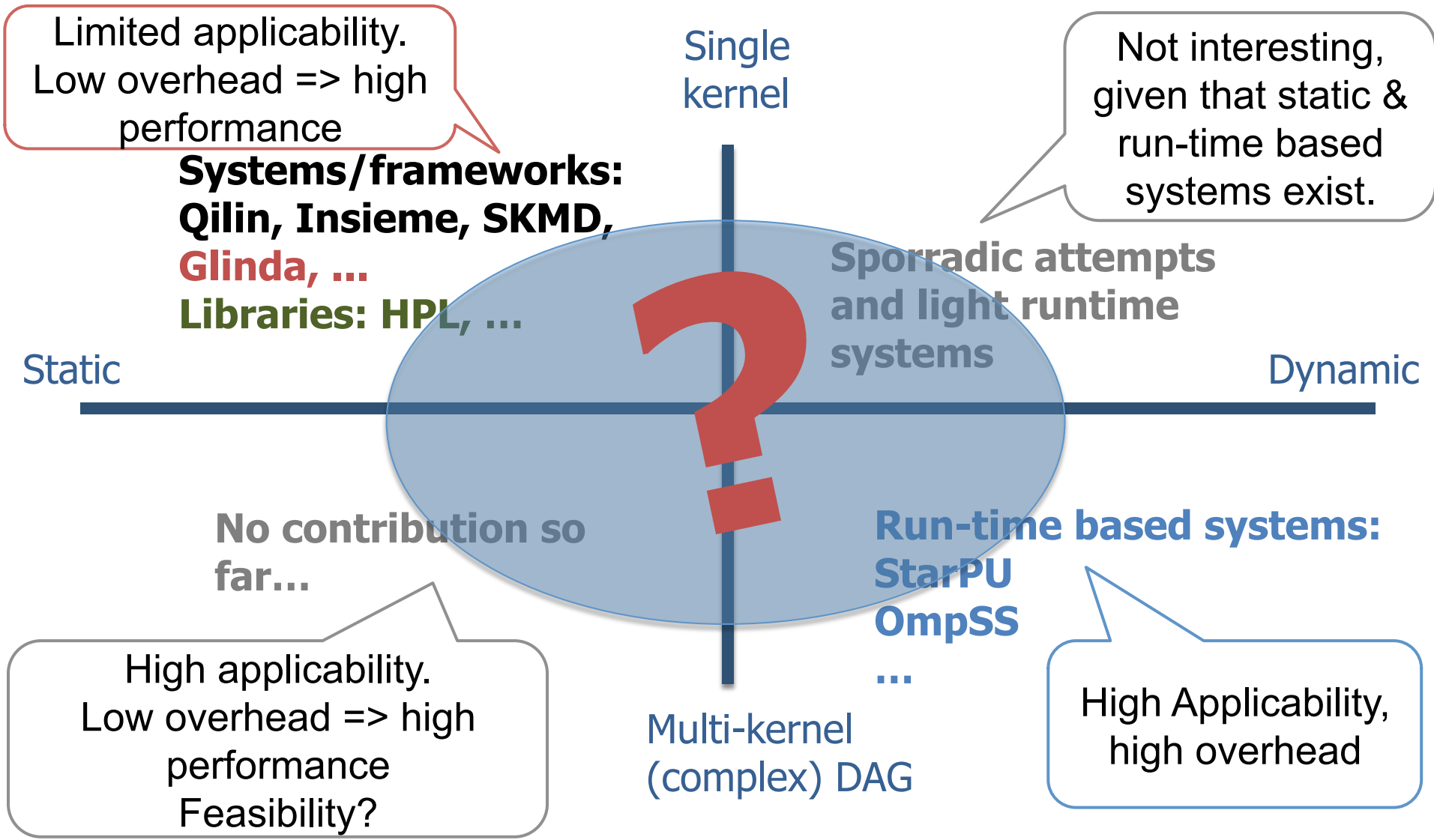
Any application, Any hardware platform



High Performance Computing.



Heterogeneous Computing next?



Instead of conclusion …

Take home message [1]

to the office

- Heterogeneous computing works!
  - *More resources.*
  - *Specialized resources.*
- Most efforts in static partitioning and run-time systems
  - Glinda = static partitioning for single-kernel, data parallel applications
    - Now works for multi-kernel applications, too
  - StarPU, OmpSS = run-time based dynamic partitioning for multi-kernel, complex DAG applications
- Domain-specific efforts, too
  - Totem – graph processing
  - GlassWing – MapReduce

Take home message [2]

to the office

- Choose a system based on your application scenario:
  - Single-kernel vs. multi-kernel
  - Massive parallel vs. Data-dependent
  - Single run vs. Multiple run
  - Programming model of choice
- There are models to cover combinations of these choices!
  - No framework to combine them all – food for thought?

Future research directions

- Porting Glinda 2.0 to more heterogeneous platforms
- Extension to more application classes
  - Multi-kernel with complex DAGs
  - (streaming applications, graph-processing applications)
- Integration with distributed systems
  - Intra-node workload partitioning + inter-node workload scheduling
- Extension of the partitioning model
  - Energy consumption

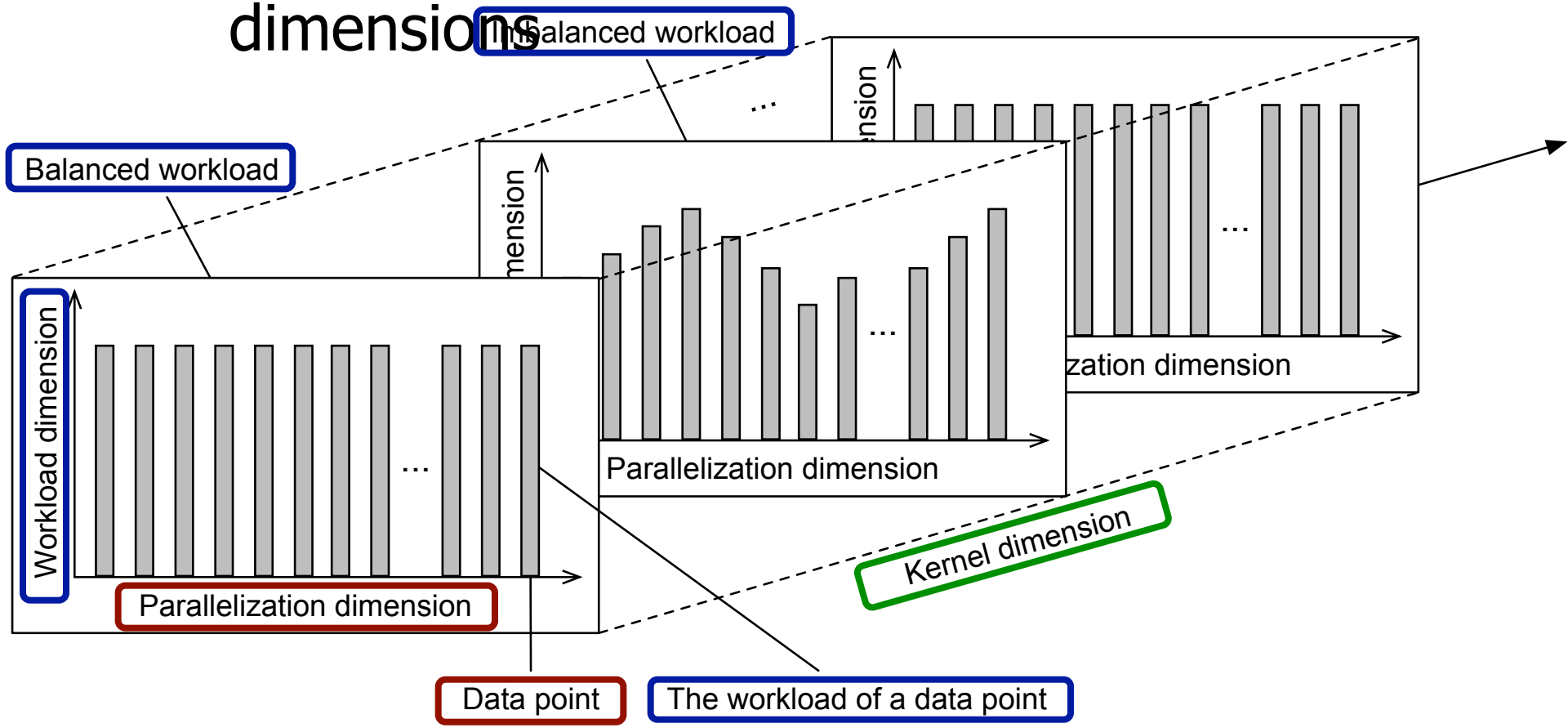
Open questions ?

- Analytical modeling instead of profiling
  - Statistical modeling – ask me more!
- Extending to other type of workloads
  - Graph processing – ask me more!
- Performance portability
  - HPL/specialized OpenCL – ask me more!

Backup Slides

An application centric approach

- Focuses on data parallel applications on three dimensions



Parallel programming models

- An abstraction that expresses application parallelism over multi-core and ma e processors

- Portability and abstraction level

Low level | High level

- Dedicated programming models

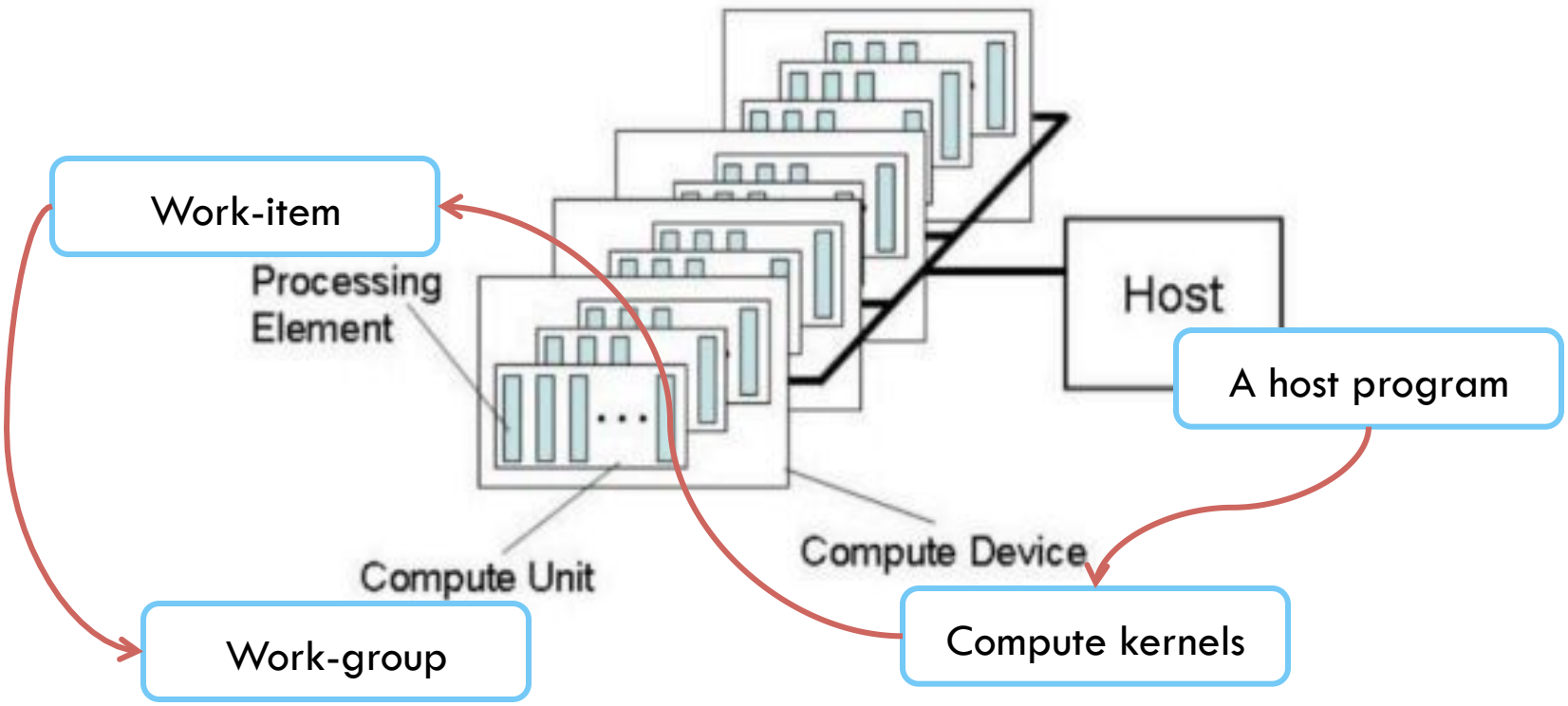
OpenCL

prog

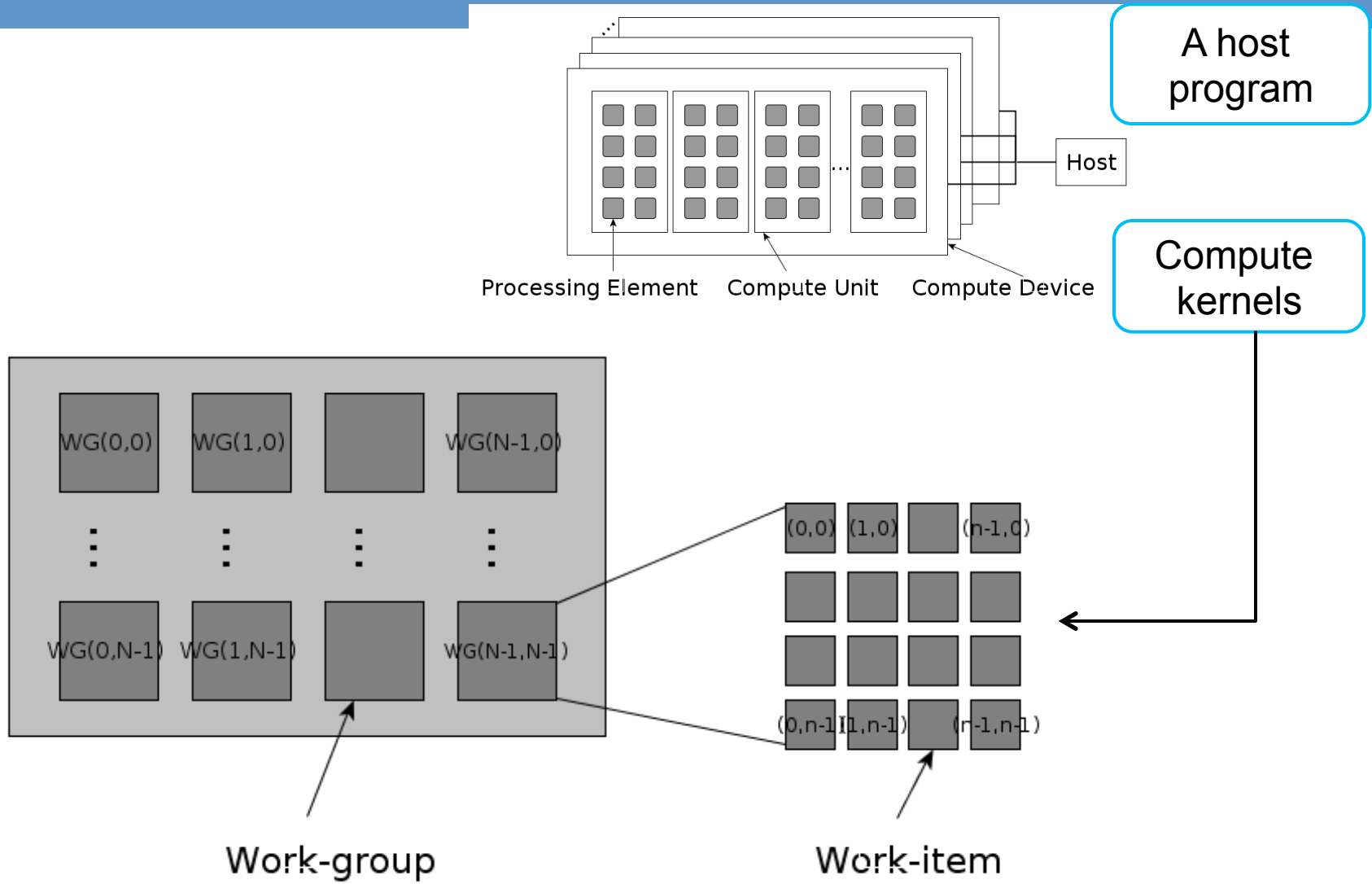
del

OmpSs

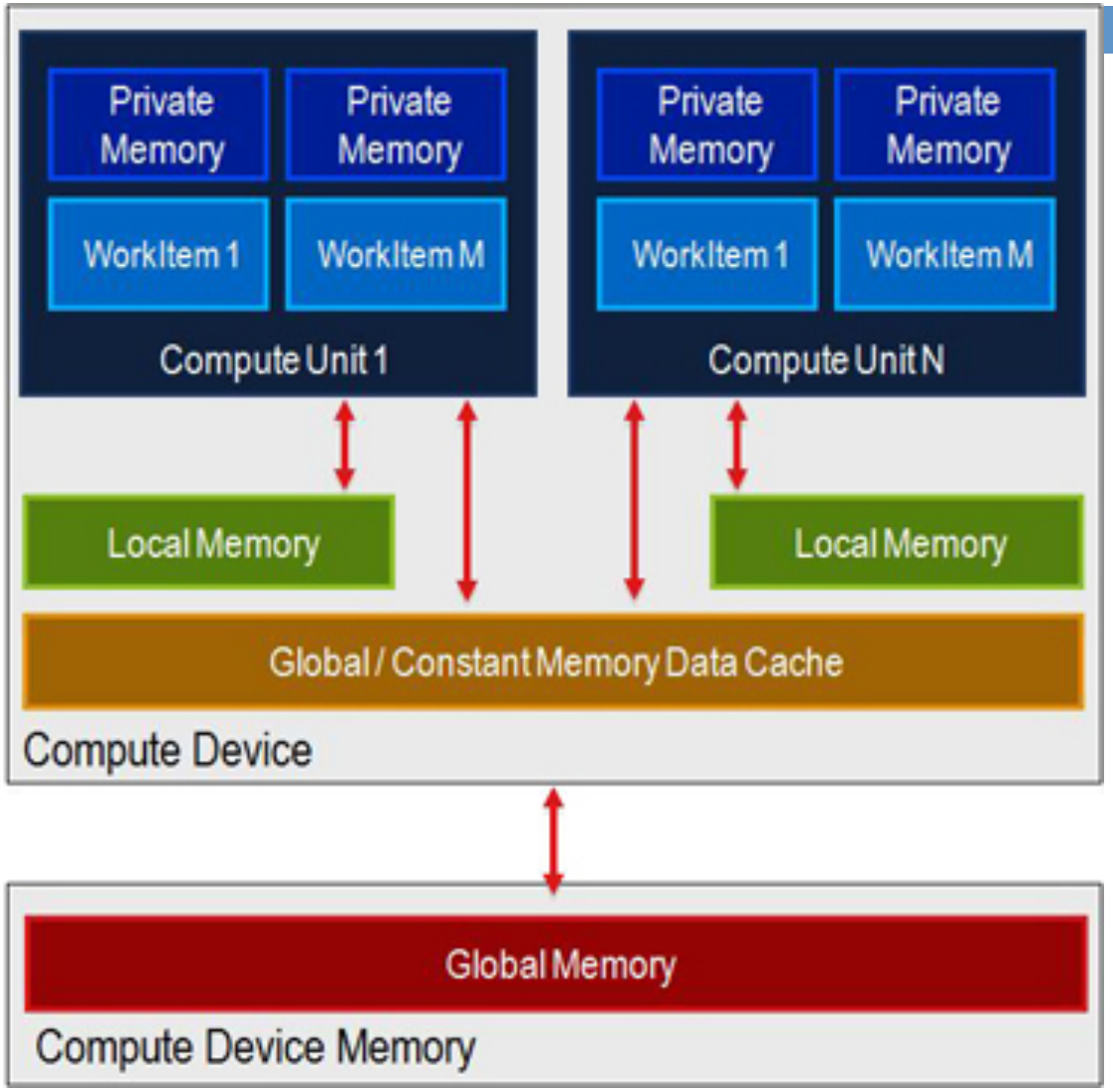
OpenCL programming model



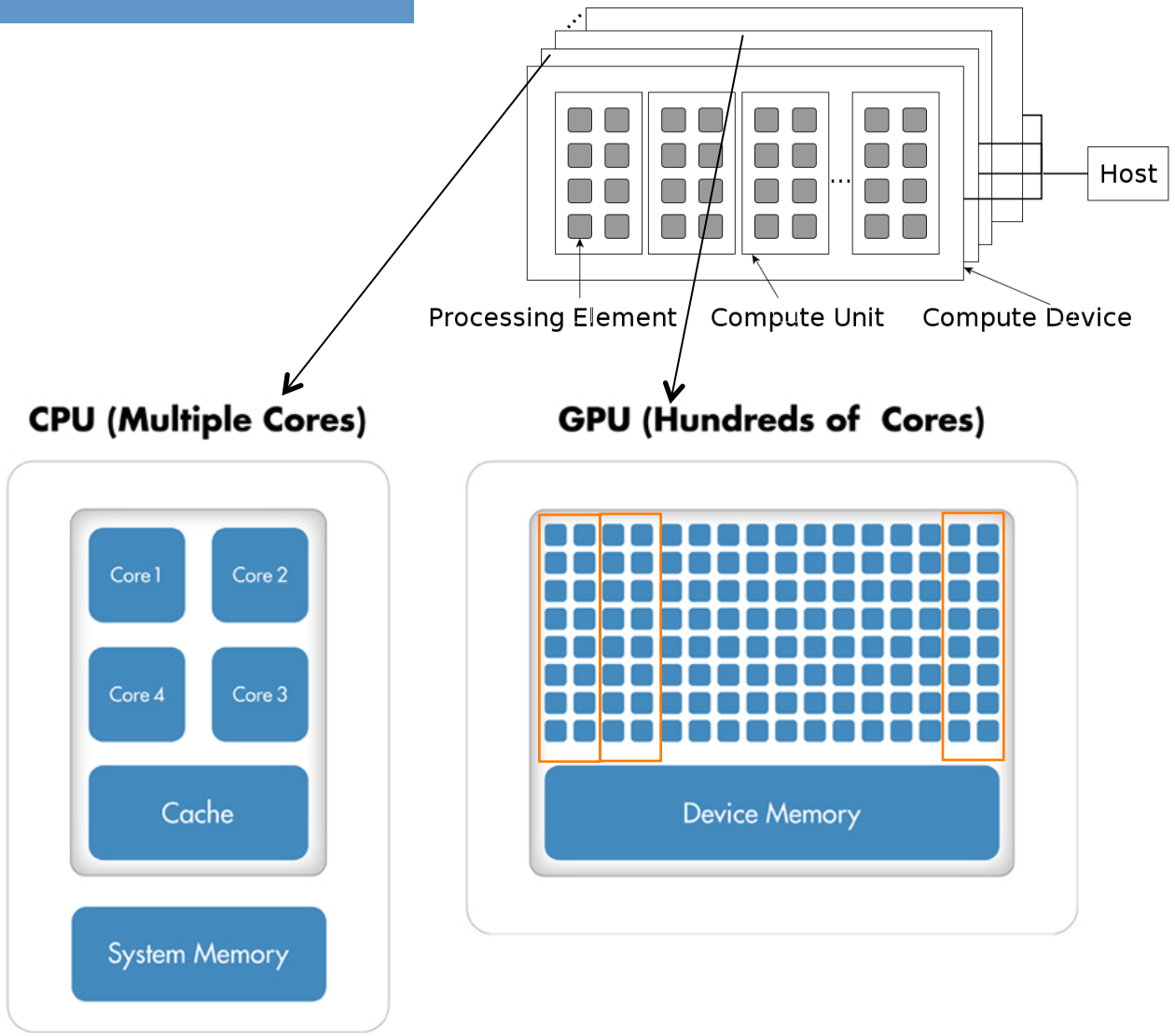
OpenCL programming model



OpenCL memory model



OpenCL virtual platform



OpenCL for heterogeneous platforms

- OpenCL has many similarities with CUDA and performs well on GPUs. **How about CPUs?**
- Three categories of performance impact factors on CPUs
  - GPU-like programming styles
    - Memory access patterns, data transfer, (local memory) => **Remove**
  - Parallelism granularity
    - Architectural mismatches with CPUs => **Tune**
  - OpenCL compilers
    - Still under development => **Test**

OpenCL for heterogeneous platforms

- OpenCL has many similarities with CUDA and performs well on GPUs. **How about CPUs?**
- Three categories of performance impact factors on CPUs

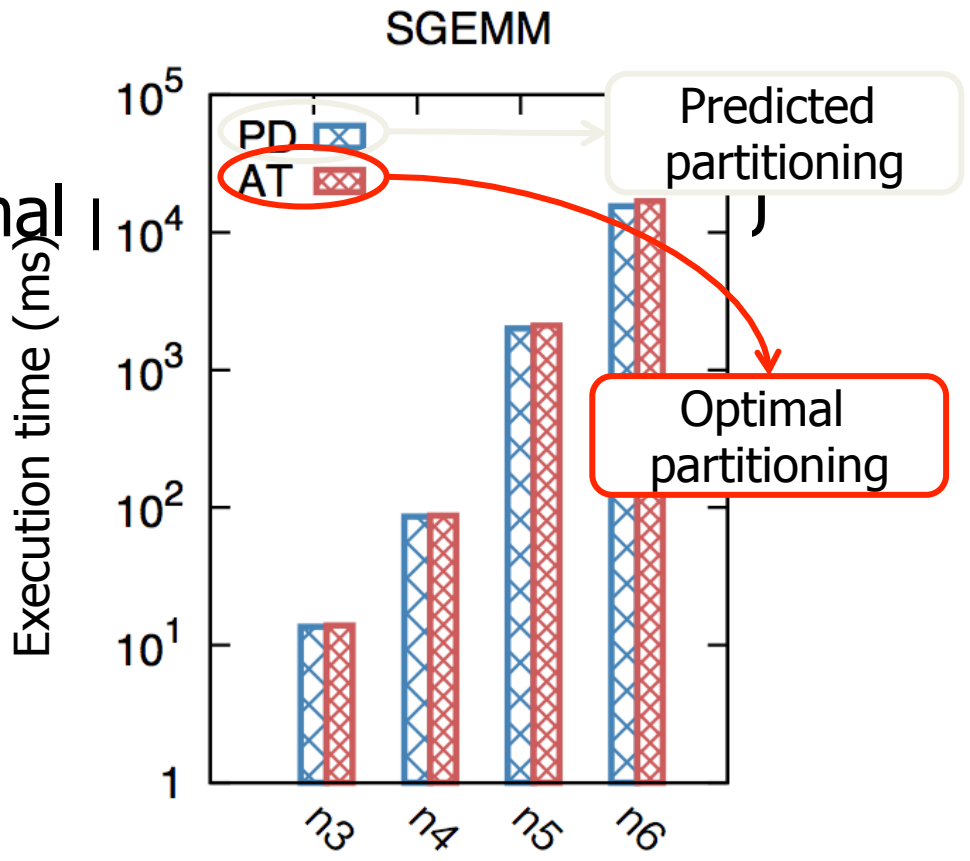
OpenCL is an efficient programming model for heterogeneous platforms if we keep processor architectural differences in mind and specialize the OpenCL code to fit different processors.

  - OpenCL compilers
    - Still under development => **Test**

Experimental evaluation

- Quality (compared to an *oracle*)
  - The right HW configuration in 62 out of 72 test

	Application	SGEMM
$n1$	R_{GC}, R_{GD}	6.04, 26.54
	β	0.1799
	Decision	Only-CPU
$n2$	R_{GC}, R_{GD}	4.78, 5.07
	β	0.4406
	Decision	Only-CPU
$n3$	R_{GC}, R_{GD}	4.15, 0.70
	β	0.7090
	Decision	CPU+GPU
$n4$	R_{GC}, R_{GD}	4.88, 0.29
	β	0.7916
	Decision	CPU+GPU
$n5$	R_{GC}, R_{GD}	5.02, 0.09
	β	0.8220
	Decision	CPU+GPU
$n6$	R_{GC}, R_{GD}	4.99, 0.04
	β	0.8271
	Decision	CPU+GPU

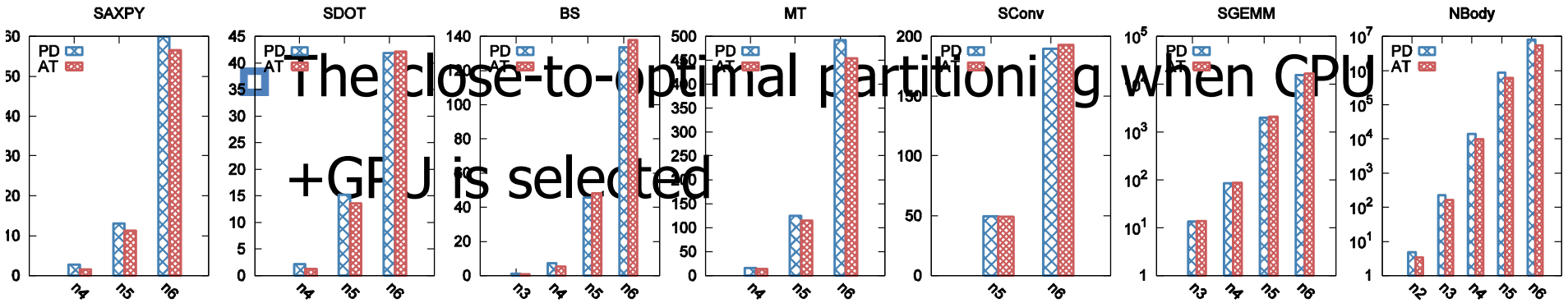


Experimental evaluation

- Impact of data transfer (assuming no data transfer)
  - Up to 80% of the work is shifted from the CPU to the GPU, up to 87% change in performance
  - HW solution
    - Use integrated CPU+GPU architecture (e.g., Intel Sandy-bridge and AMD APU)
  - SW solution
    - Use pinned memory to attain high data-transfer bandwidth
    - Hide data transfer by overlapping it with computation

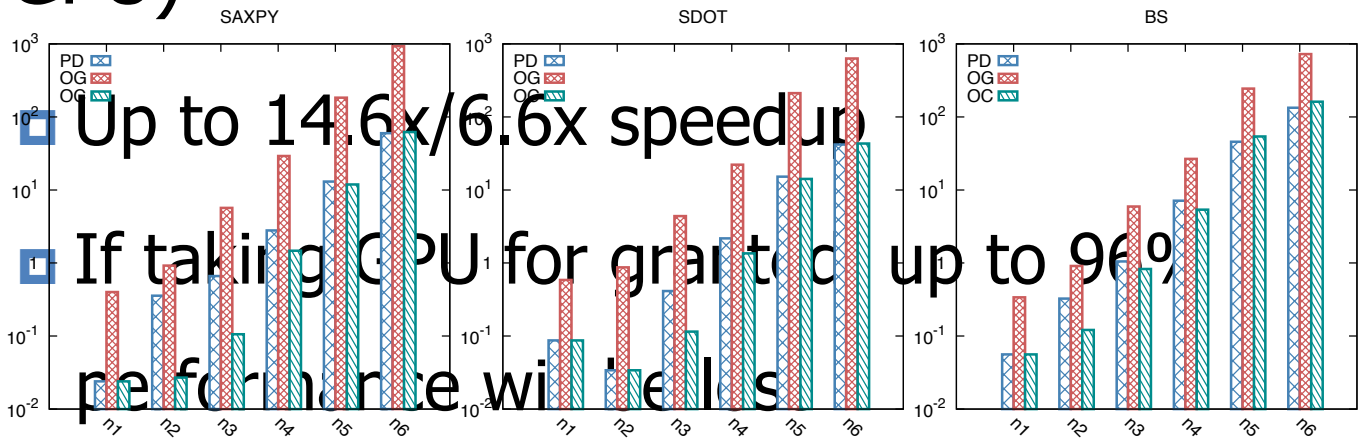
Experimental evaluation

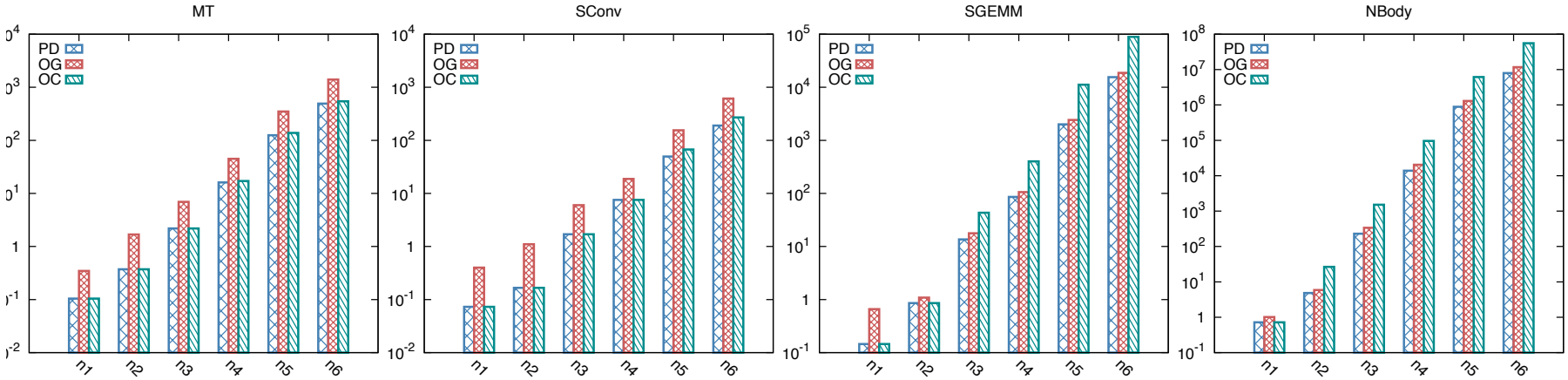
- Quality (compared to an *oracle*)
  - The right HW configuration in 62 out of 72 test cases
  - The close-to-optimal partitioning when CPU +GPU is selected



Experimental evaluation

- Effectiveness (compared to Only-CPU/Only-GPU)
  - Up to 14.6x/6.6x speedup
  - If taking CPU for granted, up to 96% performance with 1 core

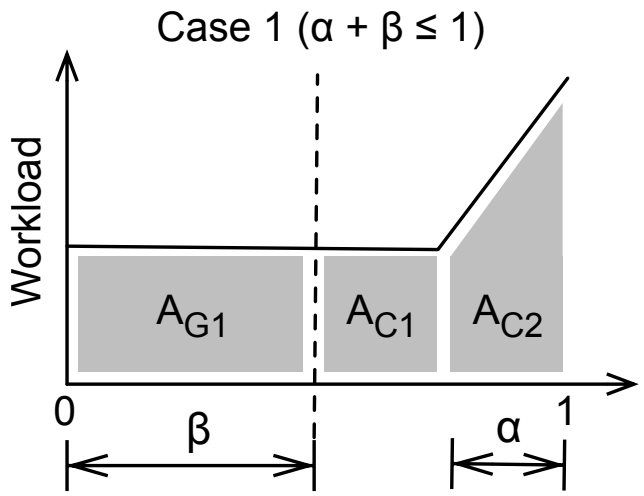




Model the Imbalanced Workload

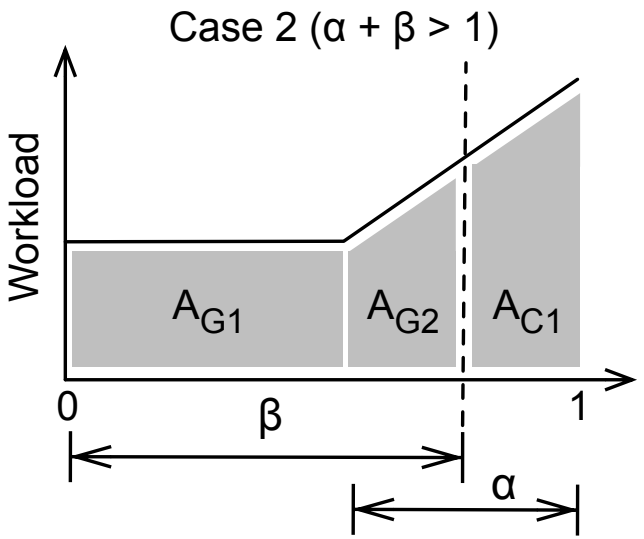
- Quantify the total workload size

$$W = \sum_{i=0}^{n-1} w_i \approx \text{the area of the partition}$$



$W_G = A_{G1}$, $W_C = A_{C1} + A_{C2}$

Partitioning point lies in the flat part

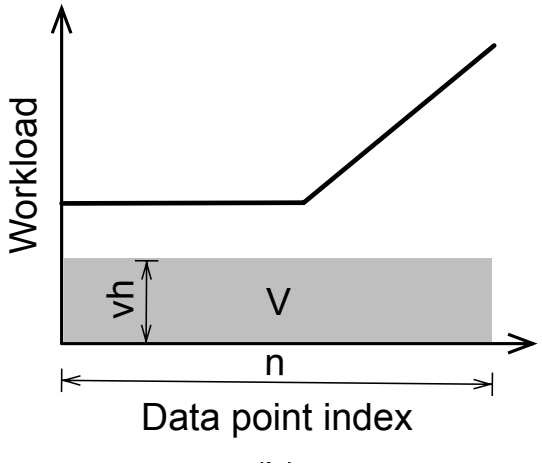


$W_G = A_{G1} + A_{G2}$, $W_C = A_{C1}$

Partitioning point lies in the peak part

Model the Hardware Capabilities

- Estimate the capability ratios by using profiling



partial workload

$$vh_{\min} \le vh \le bh$$

$$\frac{W_G}{W_C} = \frac{P_G}{P_C} \times \frac{1}{1 + \frac{P_G}{Q} \times \frac{O}{W_G}}$$

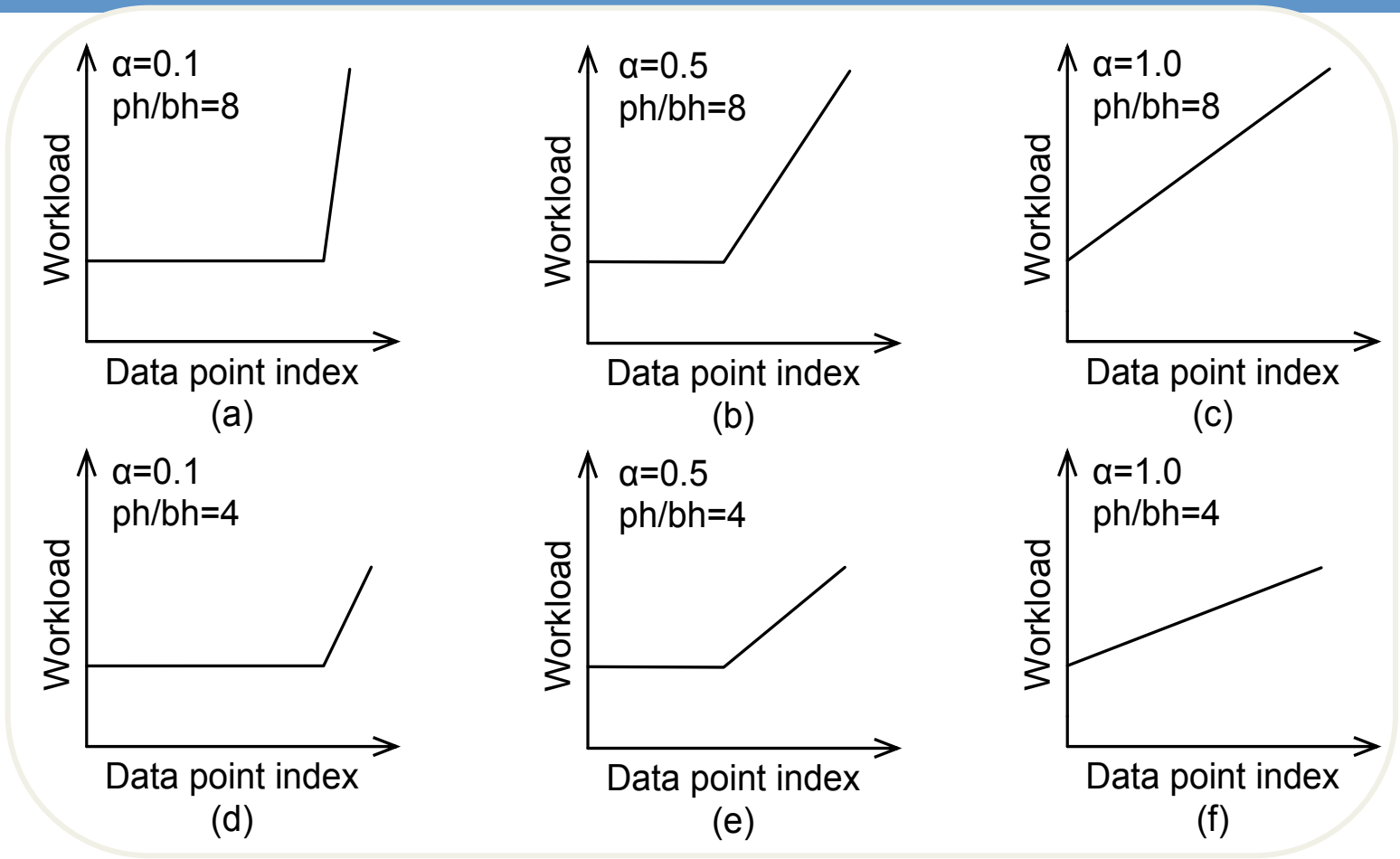
R_{GC}

CPU kernel execution time vs.
GPU kernel execution time

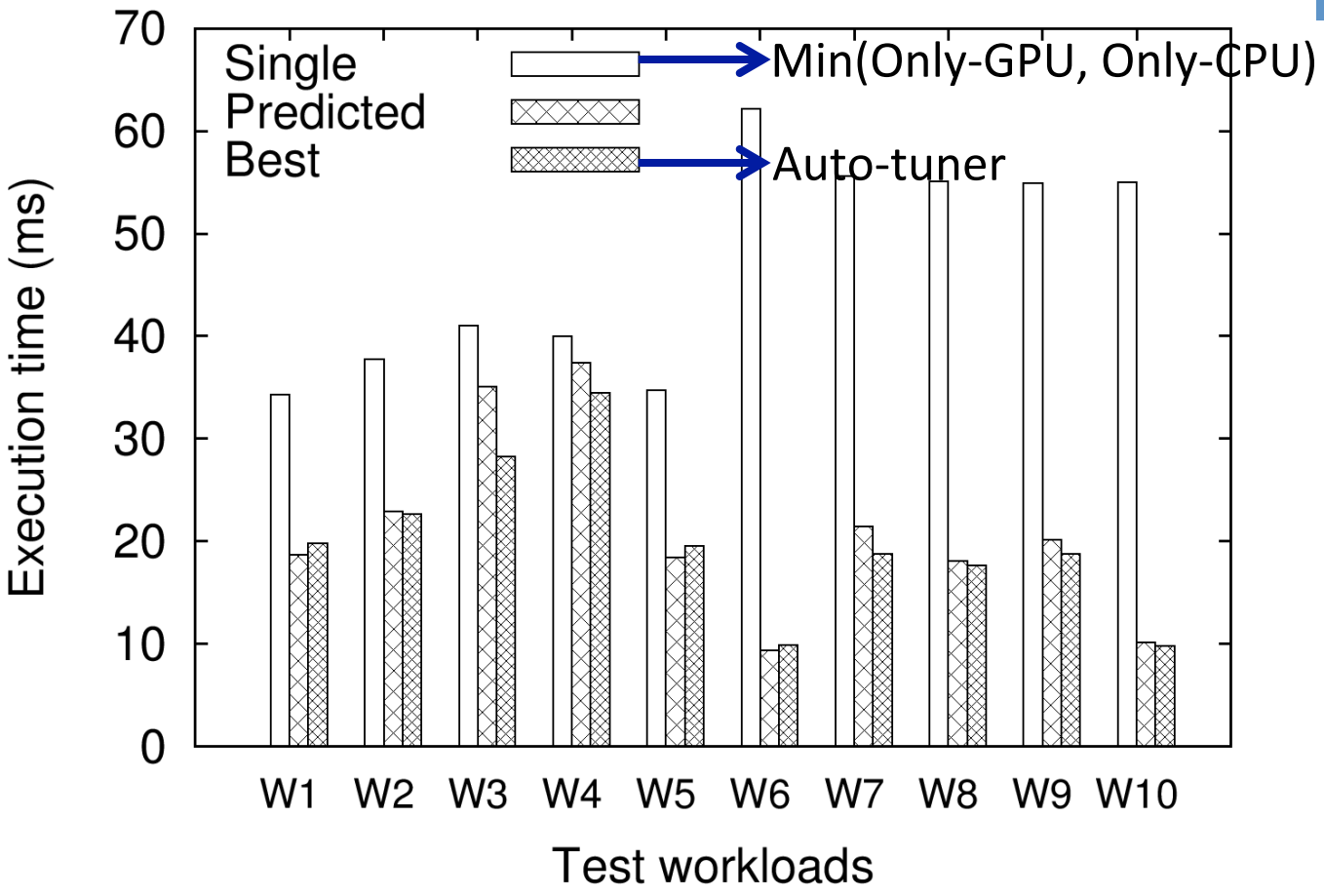
R_{GD}

GPU data-transfer time vs.
GPU kernel execution time

Synthetic imbalanced workloads



Results

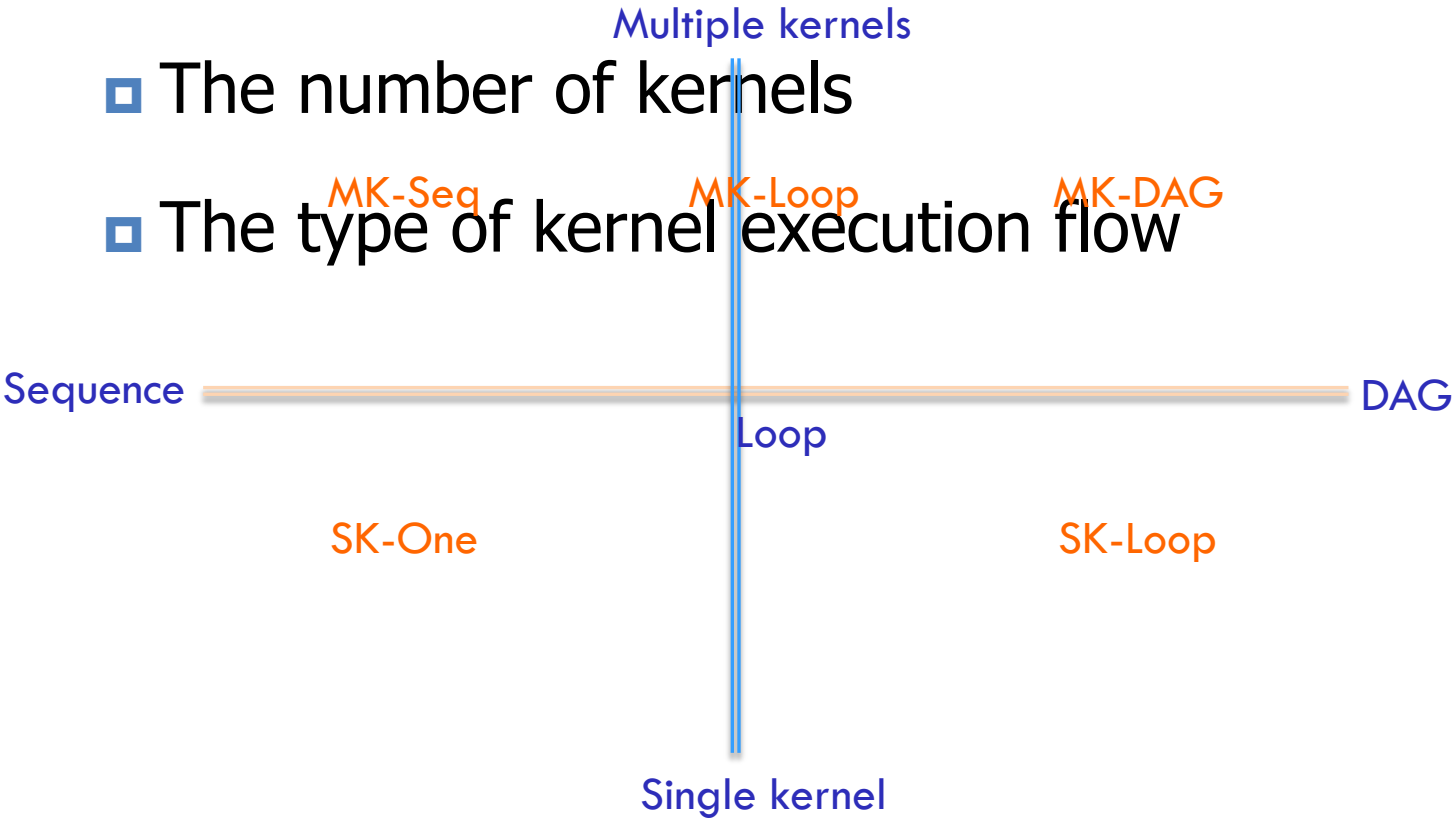


Average performance improvement: 51%

The more imbalance, the larger speedup!

What is an appropriate application classification?

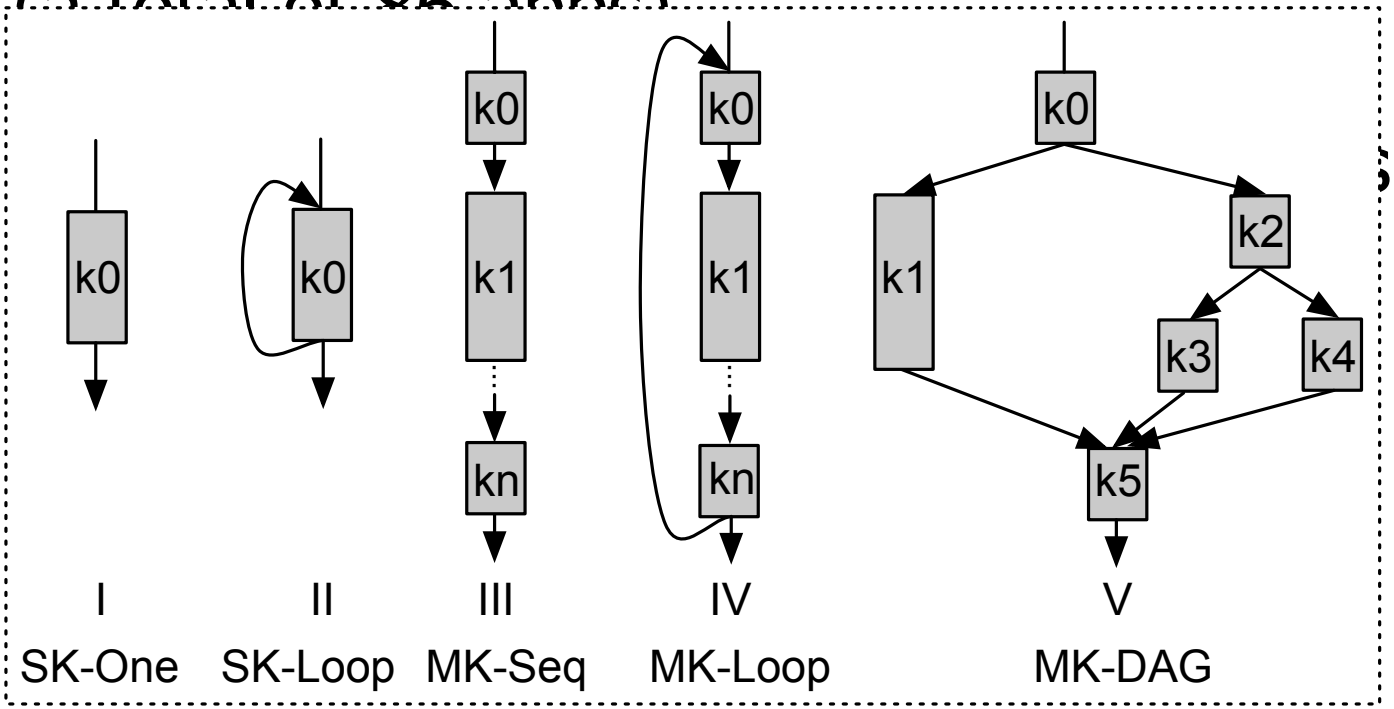
- We use **application kernel structure** to classify applications.
  - The number of kernels
  - The type of kernel execution flow



We examine five benchmark suites

Jie Shen et al, "A Study of Application Kernel Structure for Data Parallel Applications", in TUDelft technical report.

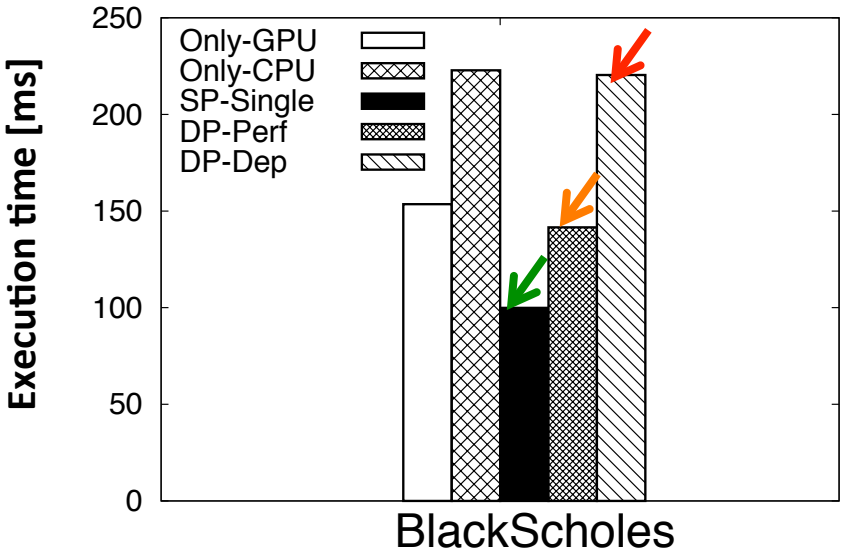
- Parboil, SHOC, Rodinia, Nvidia SDK, Mont-Blanc
- (a total of 86 apps)
-

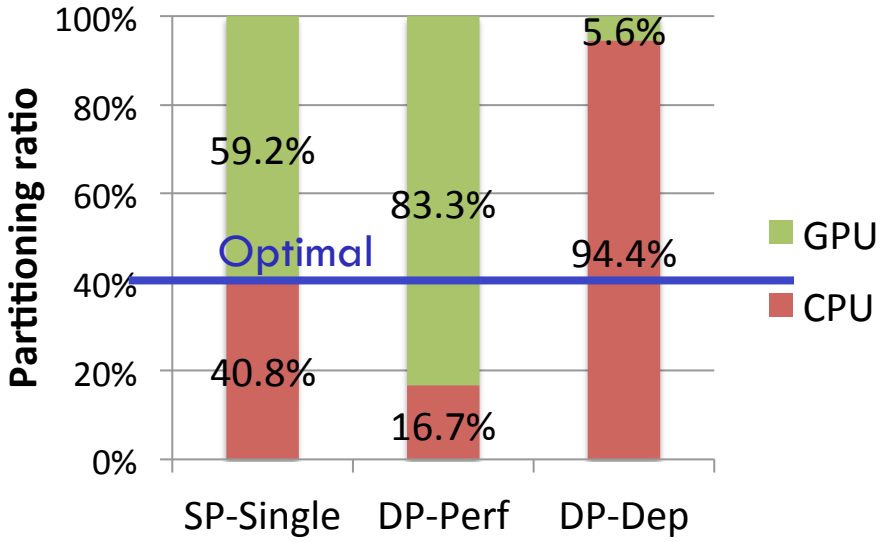


Count	31	15	25	8	7
%	36%	17%	29%	9%	8%

Results: performance ranking

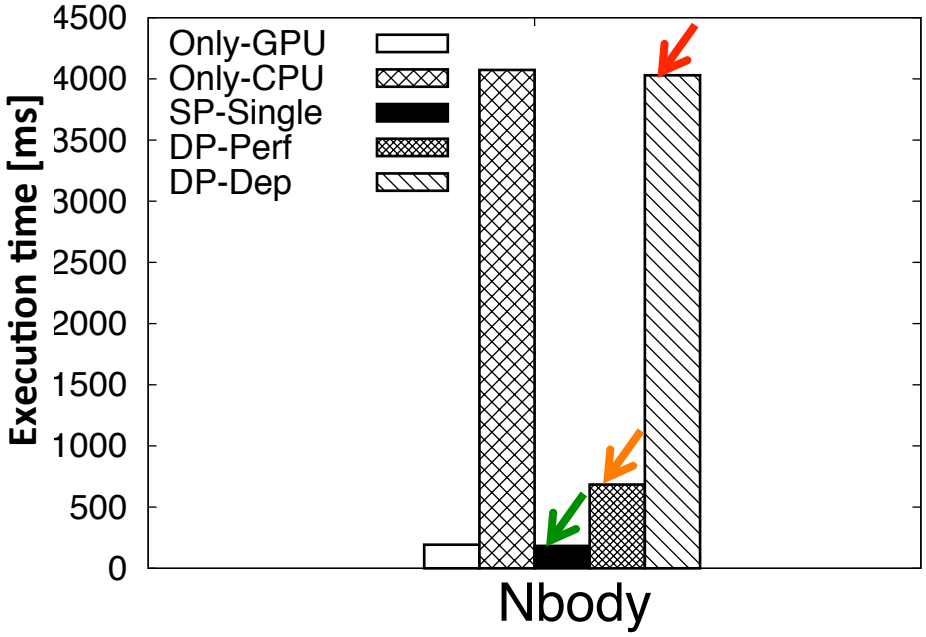
- SK-One (BlackScholes)
  - SP-Single > DP-Perf > DP-Dep

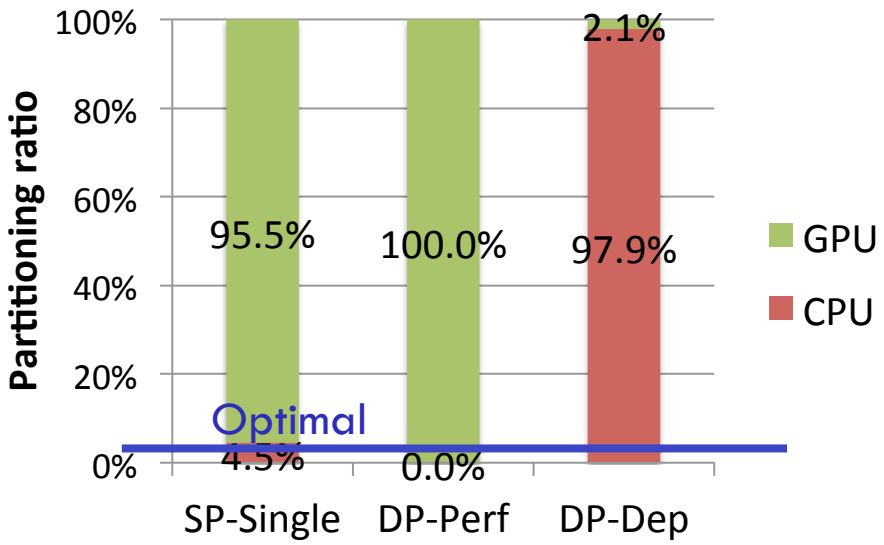




Results: performance ranking

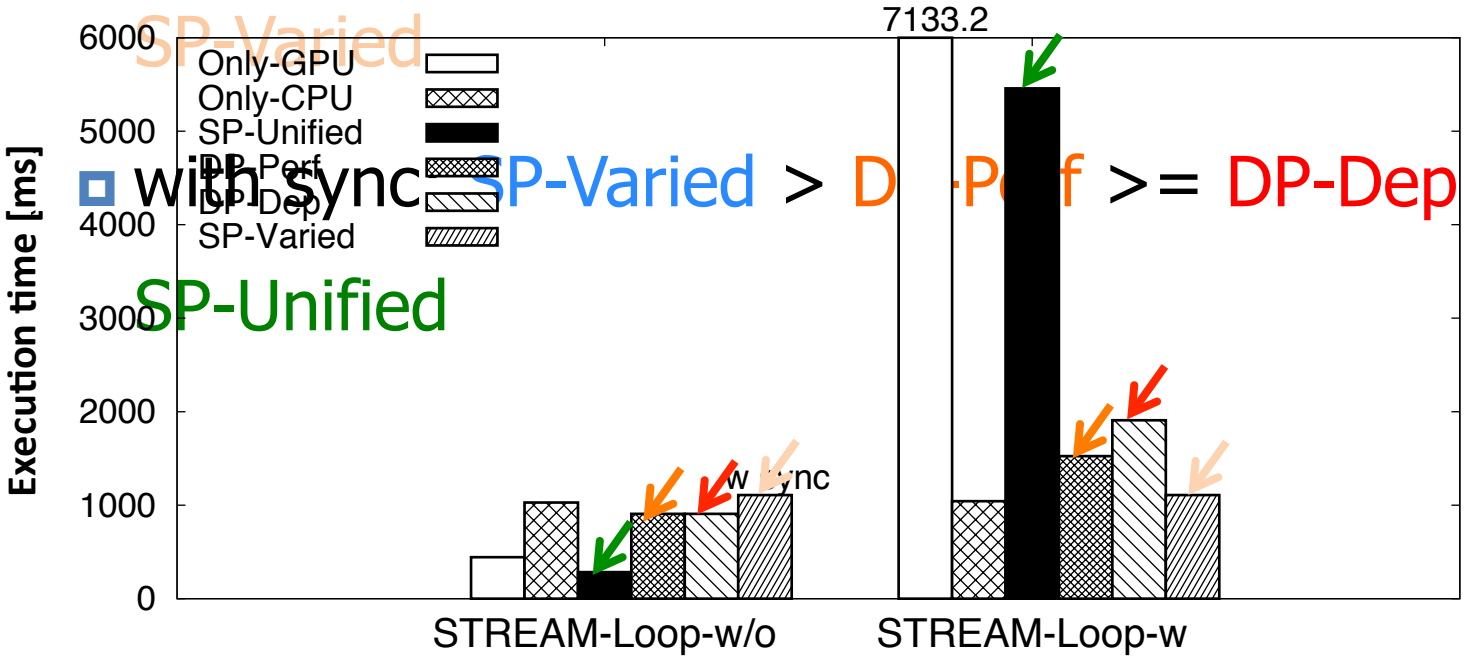
- SK-Loop (Nbody)
  - SP-Single > DP-Perf >





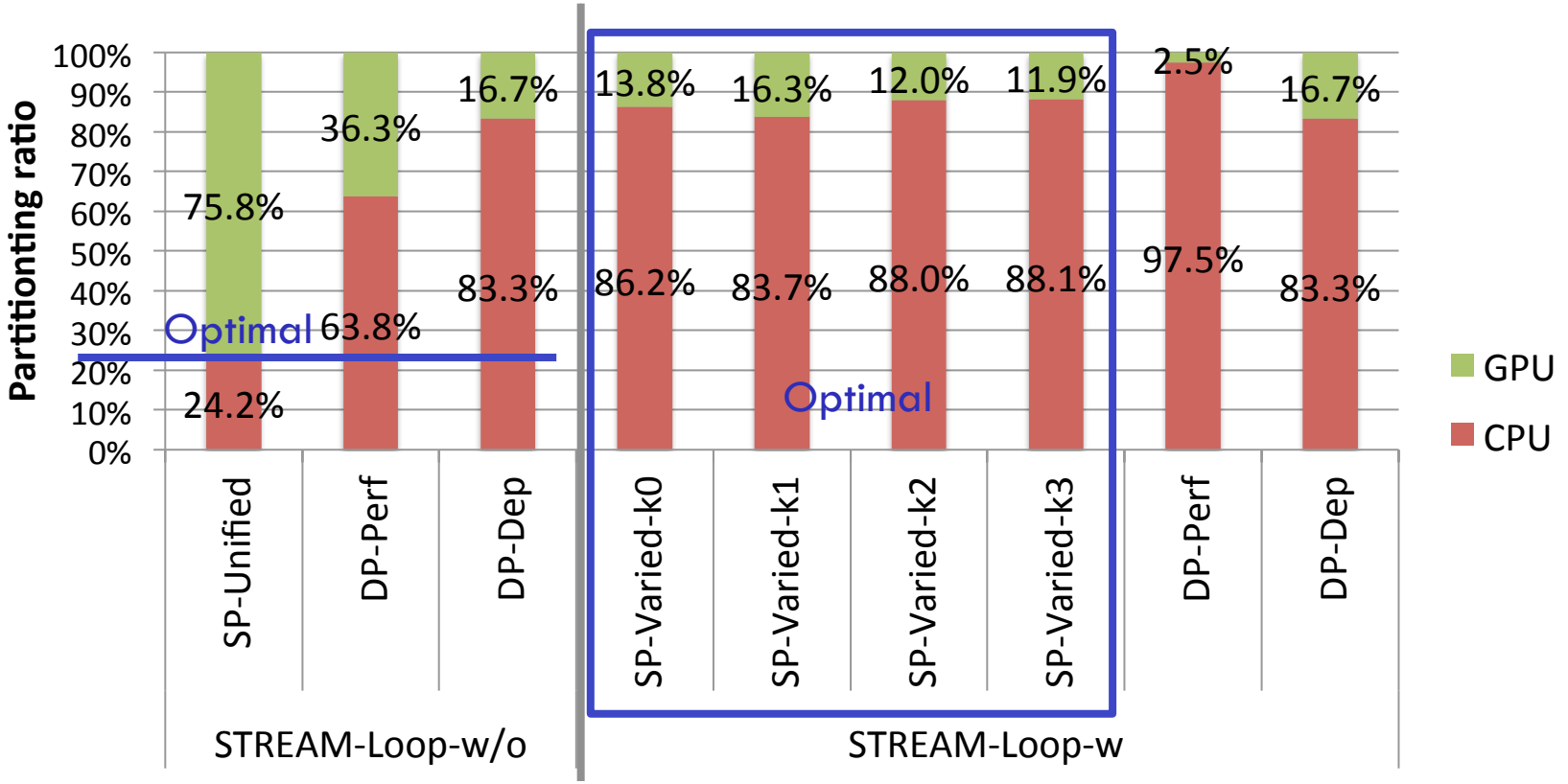
Results: performance ranking

- MK-Loop (STREAM-Loop)
  - w/o sync: SP-Unified > DP-Perf >= DP-Dep > SP-Varied
  - with sync: SP-Varied > DP-Perf >= DP-Dep > SP-Unified



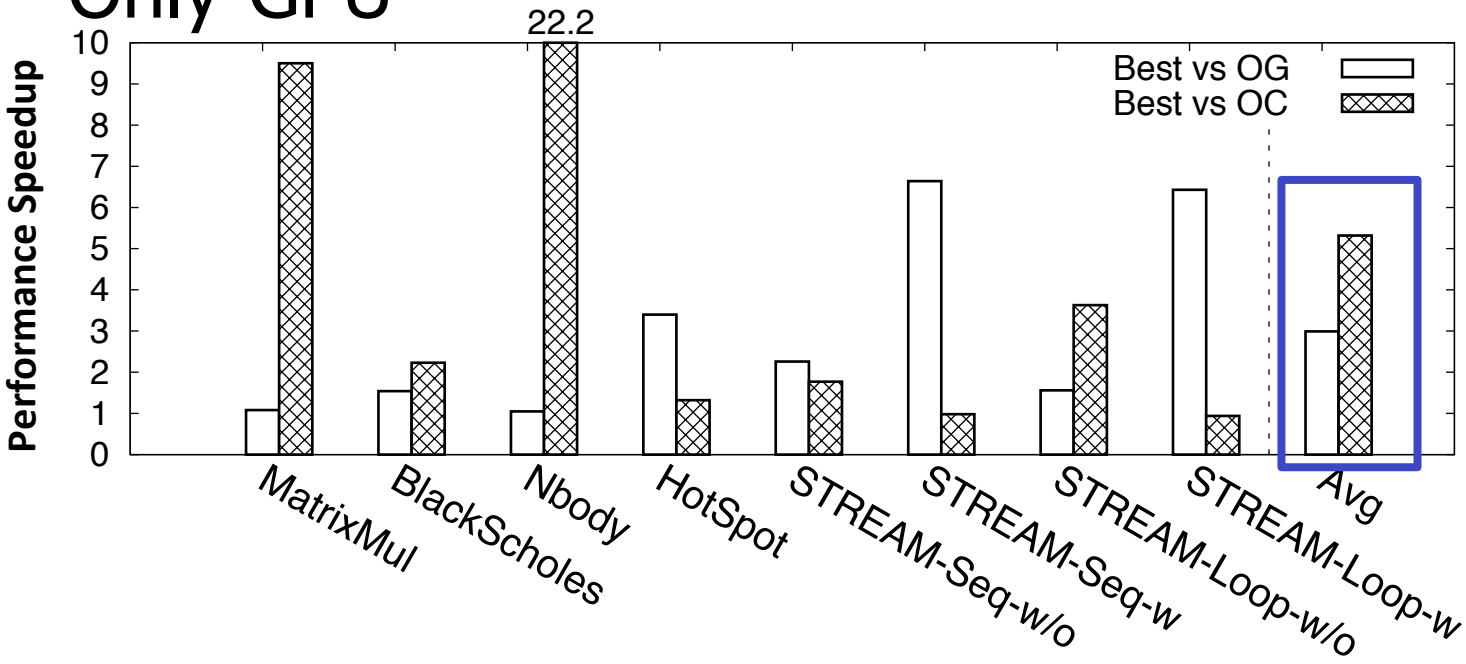
Results: performance ranking

- MK-Loop (STREAM-Loop)
  - w/o sync: SP-Unified > DP-Perf >= DP-Dep >



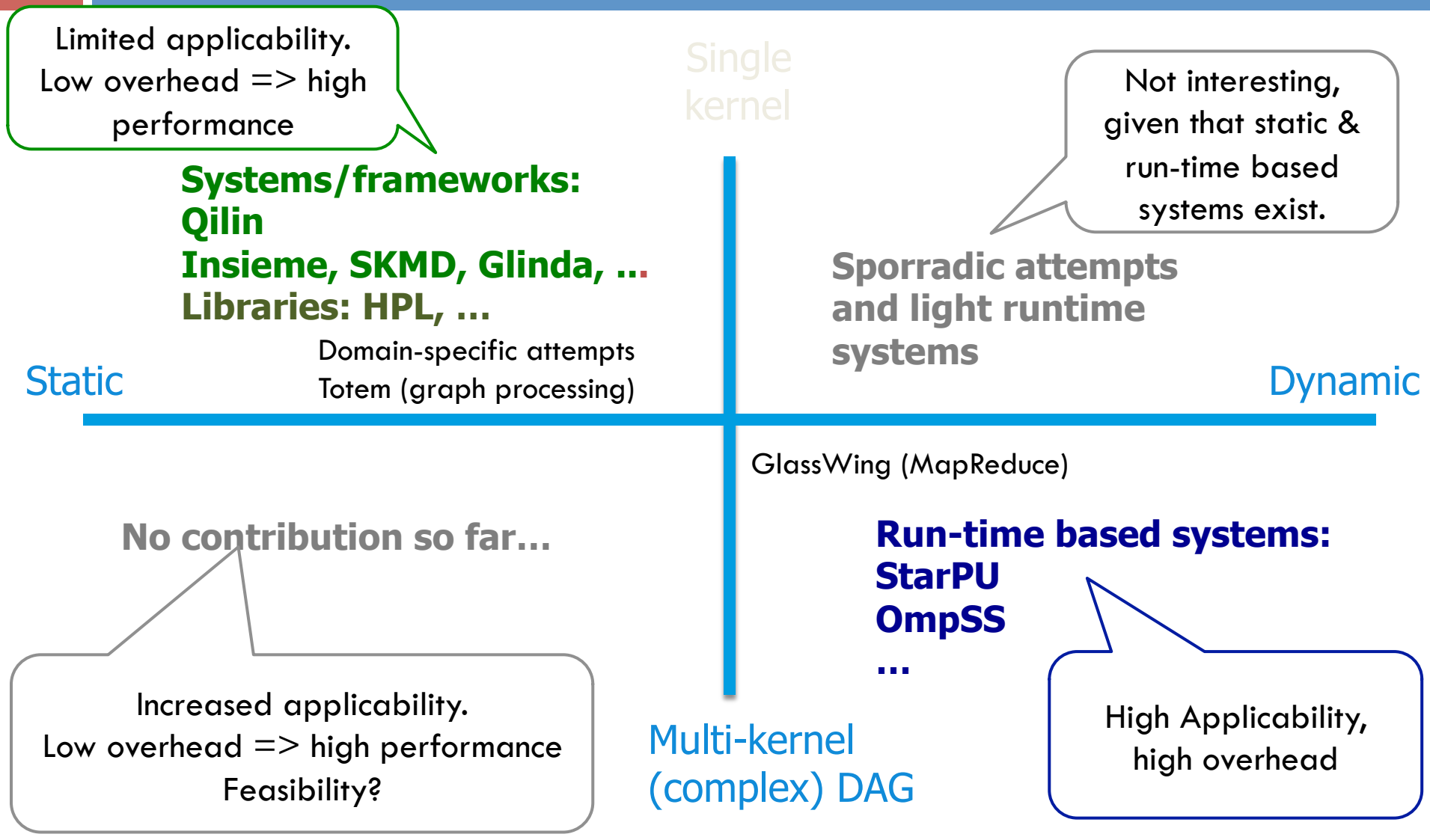
Results: performance gain

- Best partitioning strategy vs. Only-CPU or Only-GPU



Average:
3.0x (vs. Only-GPU)
5.3x (vs. Only-CPU)

Heterogeneous Computing today



Heterogeneous Computing next