

Heterogeneous platforms

- Systems combining main processors and accelerators
 - ▣ e.g., CPU + GPU, CPU + Intel MIC, AMD APU, ARM SoC

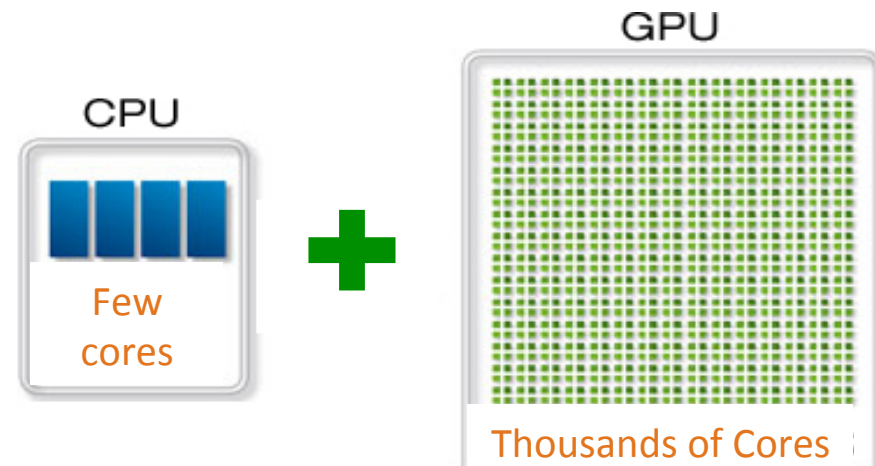


Any platform using a GPU is a heterogeneous platform!



Further in this talk ...

- A heterogeneous platform = 1 CPU + n GPUs
- Execution model = computation/kernel offloading
- An application workload = an application + its input dataset
- Workload partitioning = workload distribution among the processing units of a heterogeneous system

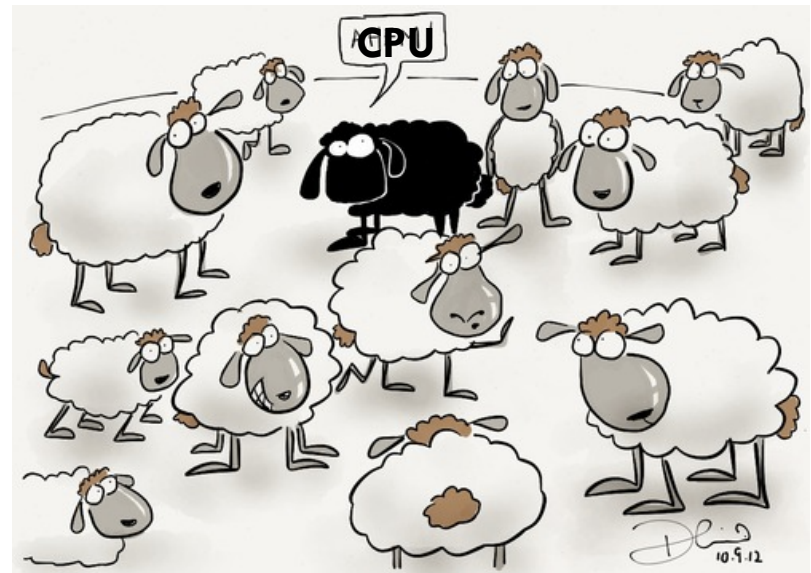


Heterogeneity vs. Homogeneity

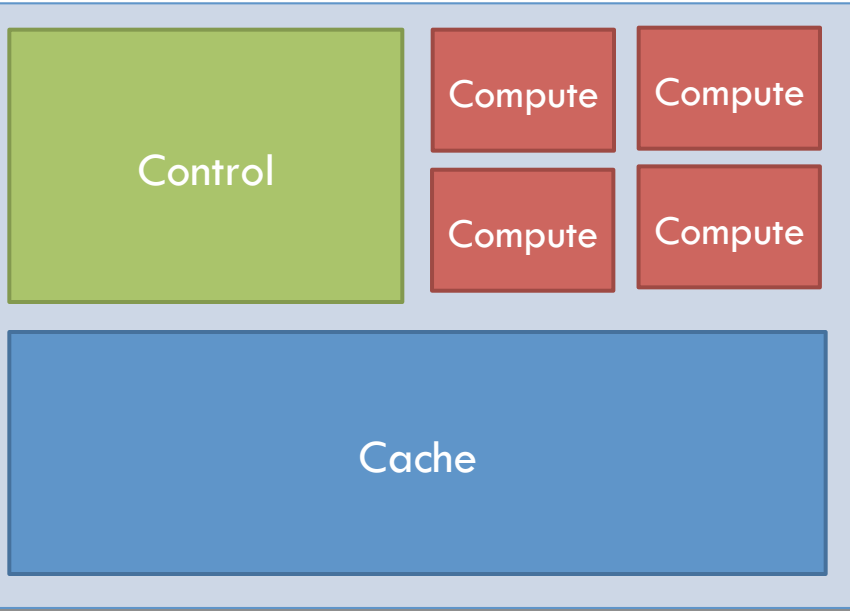
- Increase performance
 - ▣ Both devices work in parallel
 - ▣ (might) Decrease data communication
 - ▣ Different devices play different roles
- Increase flexibility and reliability
 - ▣ Choose one/all *PUs for execution
 - ▣ Fall-back solution when one *PU fails
- Increase power efficiency
- Cheaper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

Goals*

- Demonstrate heterogeneous computing is interesting ~~interesting~~ **challenging**
- Discuss the landscape of heterogeneous computing
 - ▣ Programming models
 - ▣ Partitioning models
- Tell some success stories
- Present open questions

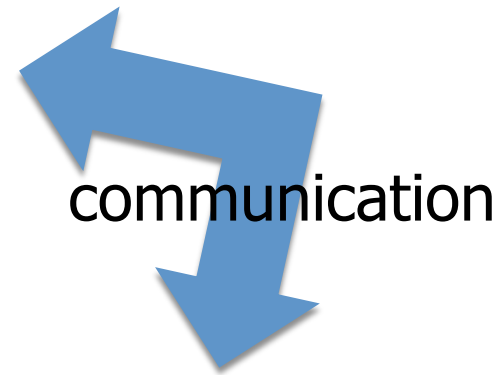


CPU vs. Accelerator (GPU)



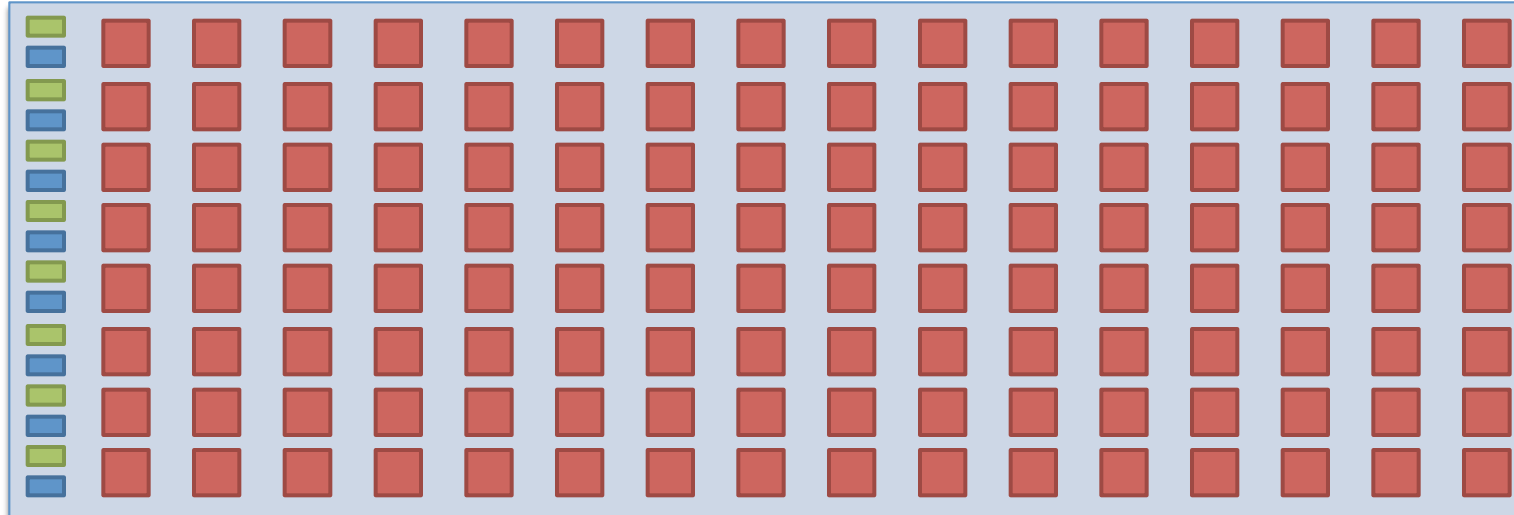
CPU

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



GPU

High throughput.
Excellent for massively parallel workloads.



Example 1: dot product

6

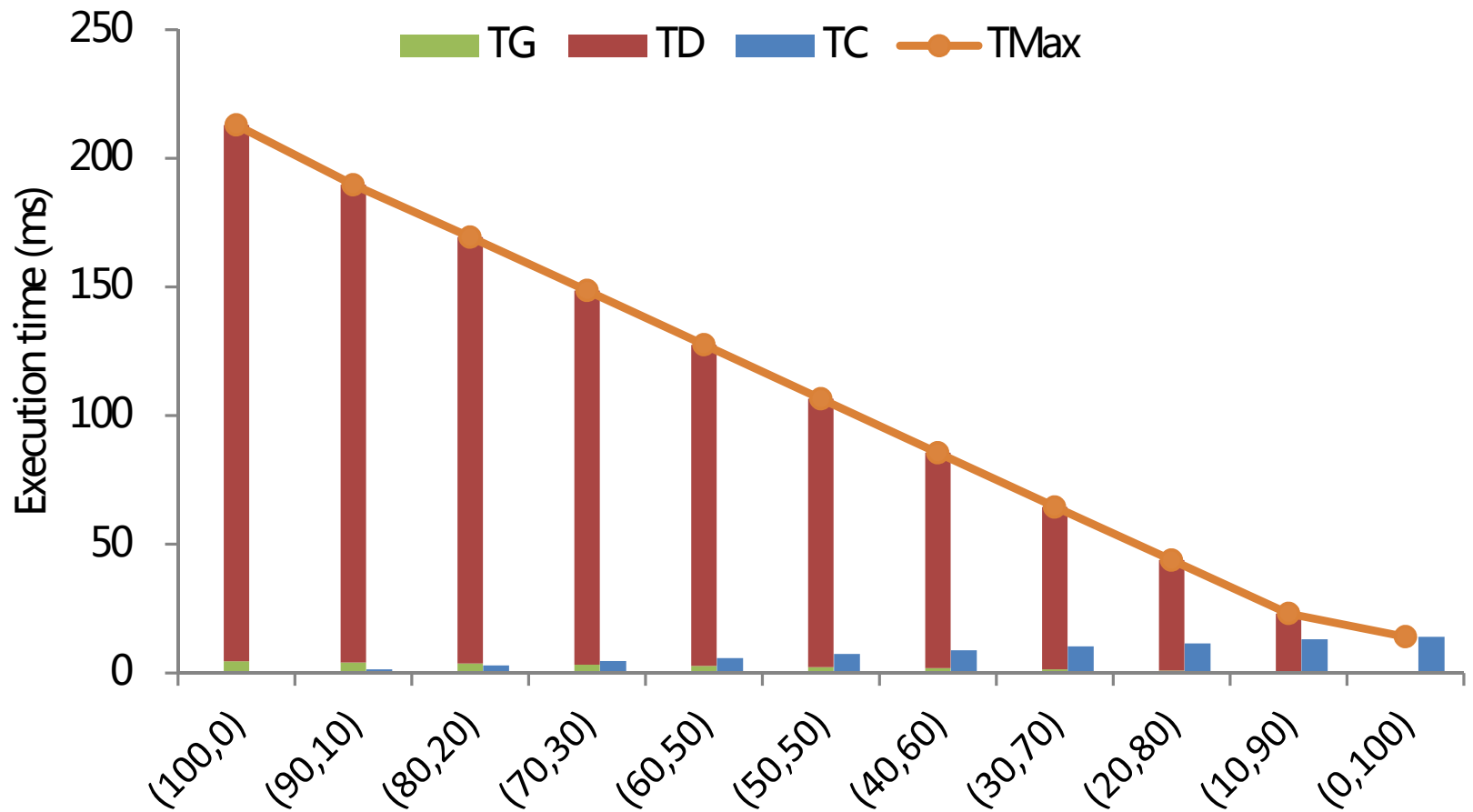
- Dot product
 - ▣ Compute the dot product of 2 (1D) arrays
- Performance
 - ▣ T_G = execution time on GPU
 - ▣ T_C = execution time on CPU
 - ▣ T_D = data transfer time CPU-GPU
- GPU best or CPU best?

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 \end{bmatrix}$$

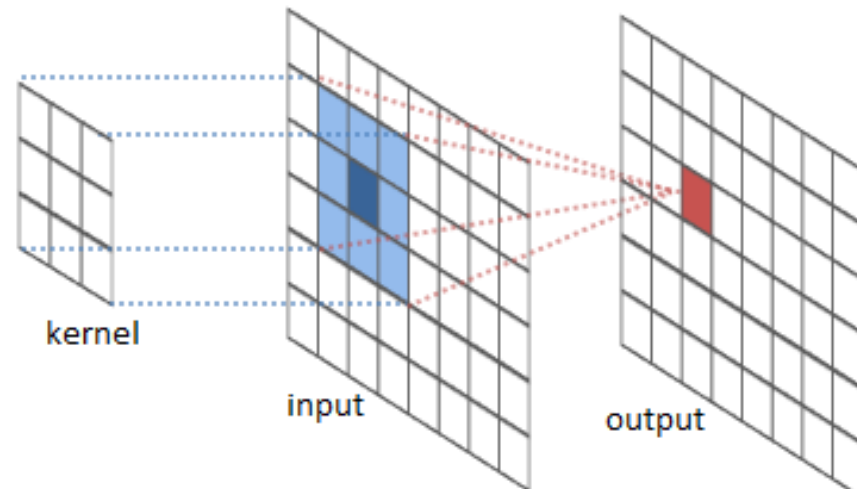
Example 1: dot product

7

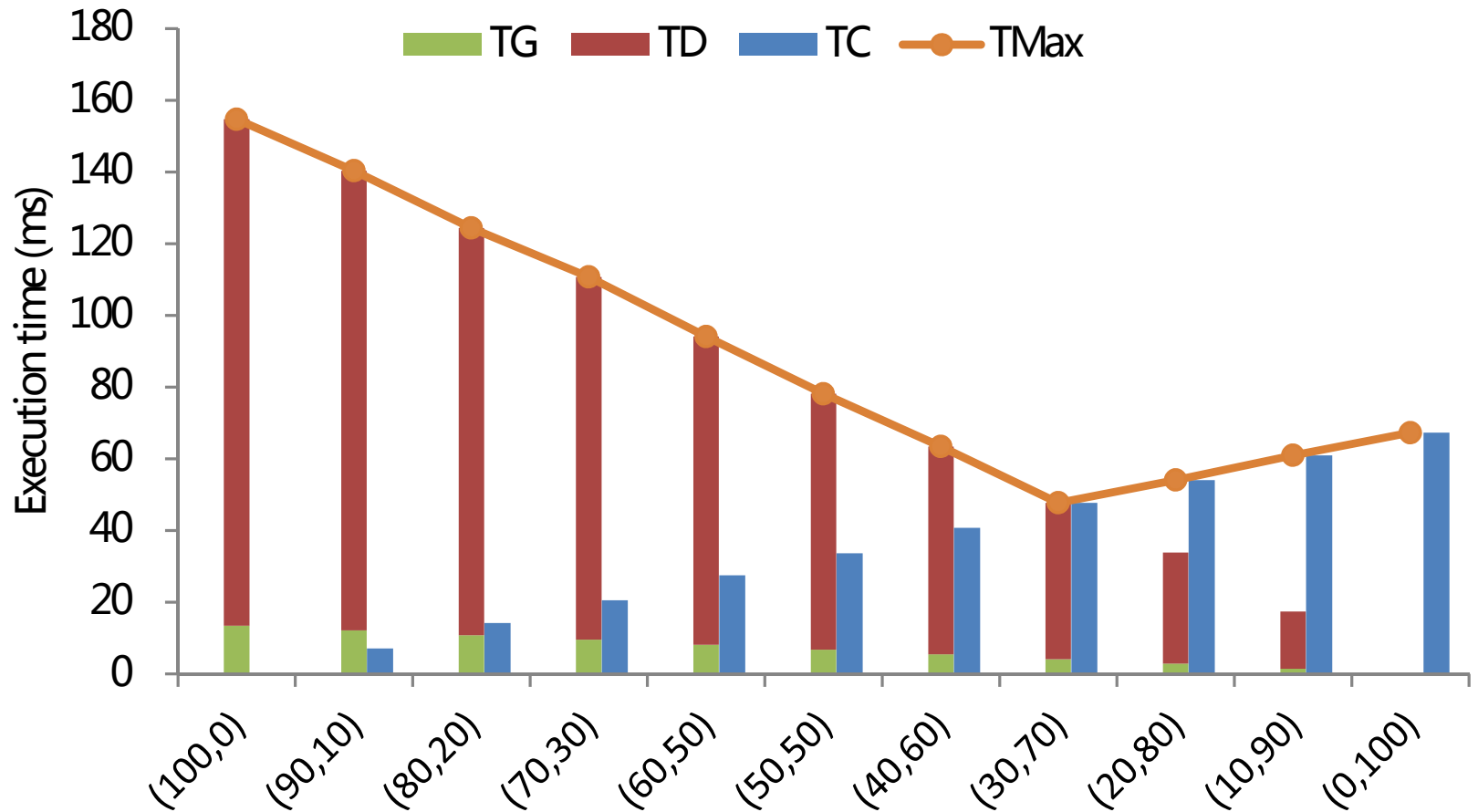


Example 2: separable convolution

- 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?



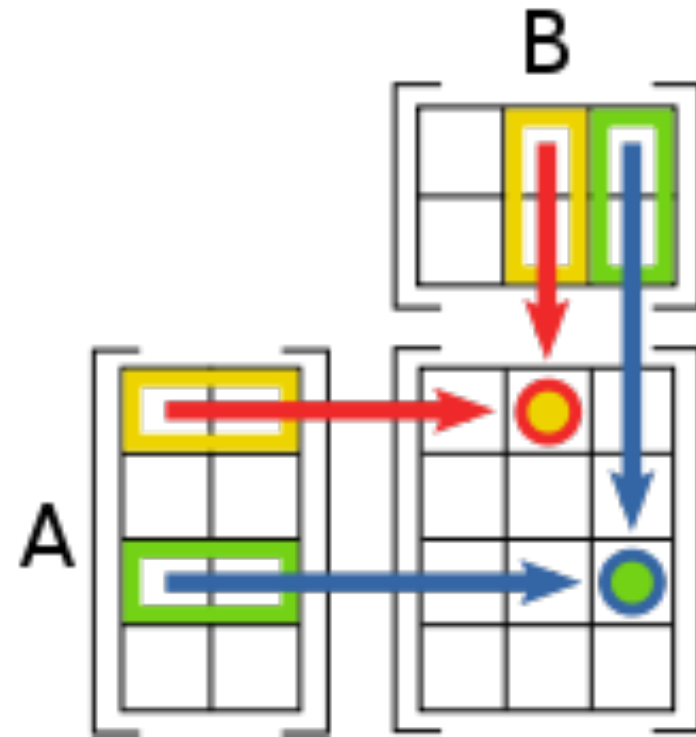
Example 2: separable convolution



Example 3: matrix multiply

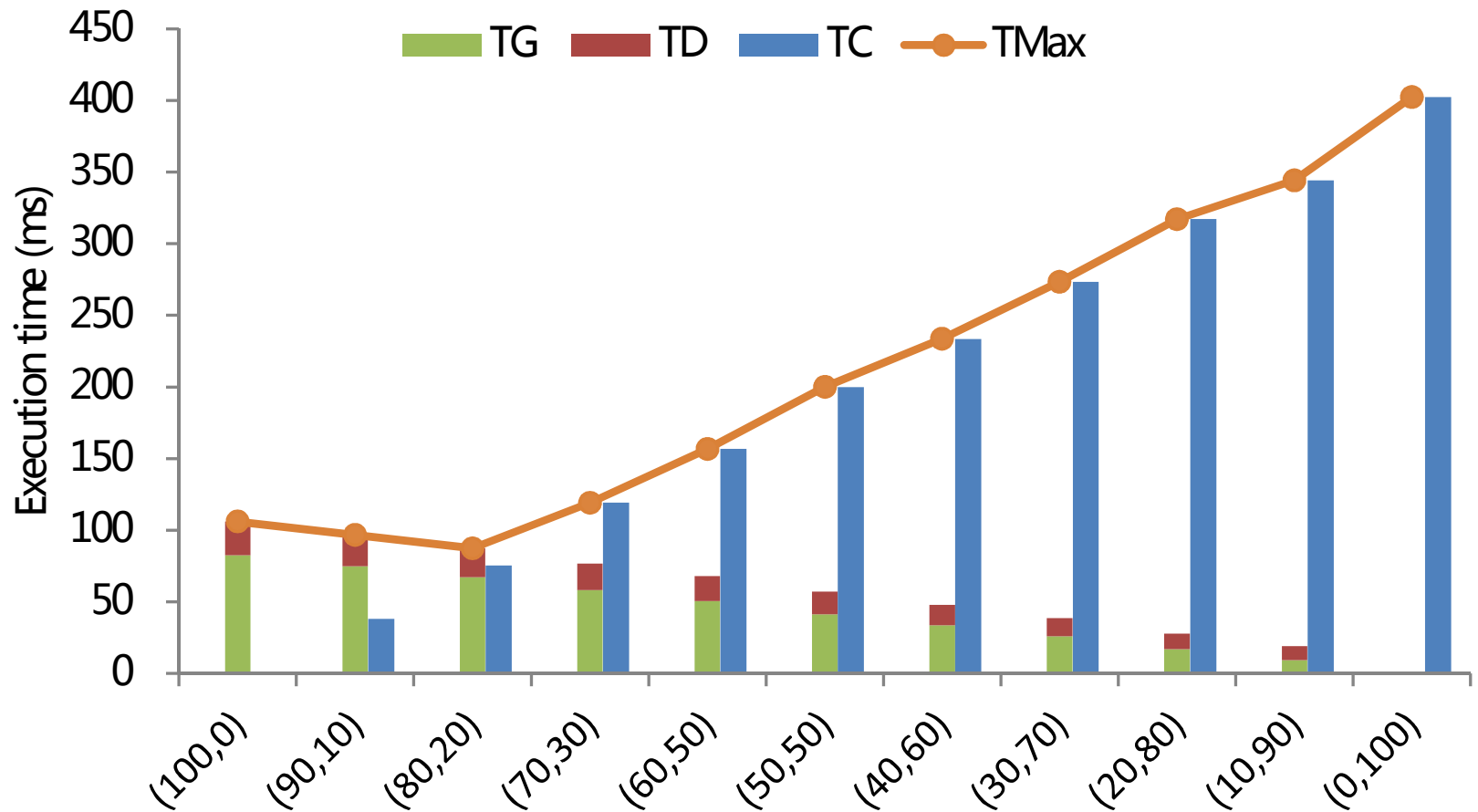
10

- 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

11



Findings

12

- There are **very few GPU-only applications**
 - ▣ CPU – GPU communication bottleneck.
 - ▣ Increasing performance of CPUs
- Optimal partitioning between *PUs is difficult
 - ▣ Load balancing depends on (platform, application, dataset)
 - ▣ Imbalance => performance loss versus original !
- Programming different platforms with a coherent model is difficult

Findings

13

- There are **very** few GPU-only applications
 - ▣ CPU – GPU communication bottleneck.
 - ▣ Increasing performance of CPUs
- Optimal partitioning between *PUs is difficult
 - ▣ Load balancing depends on (platform, application, dataset)
 - ▣ Imbalance \rightarrow performance loss versus original

We need systematic methods (1) to program and (2) to partition workloads for heterogeneous platforms.



Programming

Programming models (PMs)

- Variety of options
 - ▣ Platform-specific programming models
 - ▣ Unified programming models
 - ▣ Heterogeneous programming models (WiP)
- Taxonomy: abstraction level and generality

Low level



OpenCL

High level

OpenACC
Directives for Accelerators

OpenMP 4.0

Heterogeneous Programming Library



Heterogeneous Computing PMs

Higher level abstraction.
Dedicated APIs/pragma's.
Focus on ease of use.

Domain and/or
application specific.
Focus on: productivity
and performance

OpenACC, OpenMP 4.0
OmpSS, StarPU, ...
HPL

HyGraph (graph processing),
Cashmere (divide and conquer)
GlassWing (mapReduce)

Generic

Specific

OpenCL
OpenMP+CUDA

TOTEM (graph processing)

The most common atm.
Useful for performance,
difficult to use in practice

Low
level

Domain specific, focus
on performance.
Difficult to use.

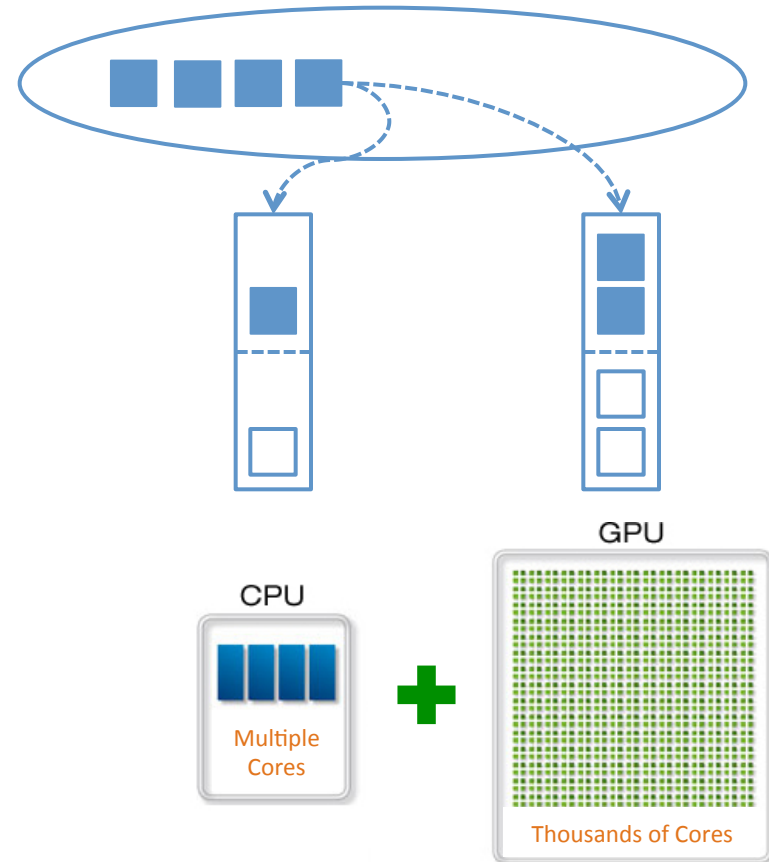
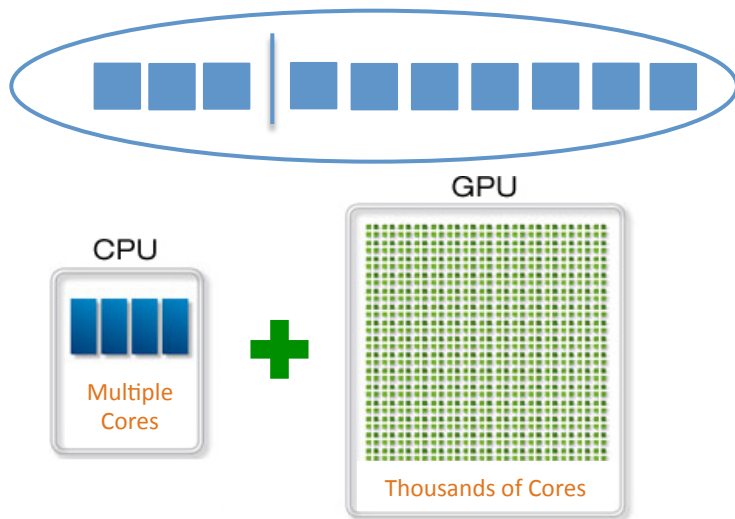


Partitioning

Determining the partition

18

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



Static vs. dynamic

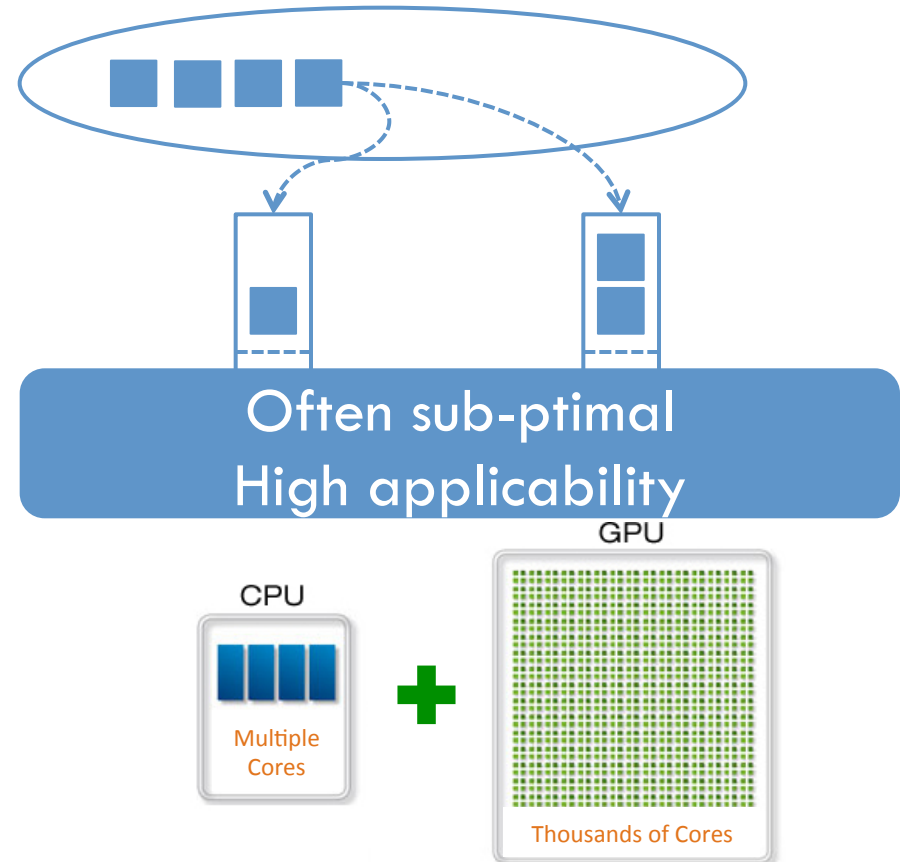
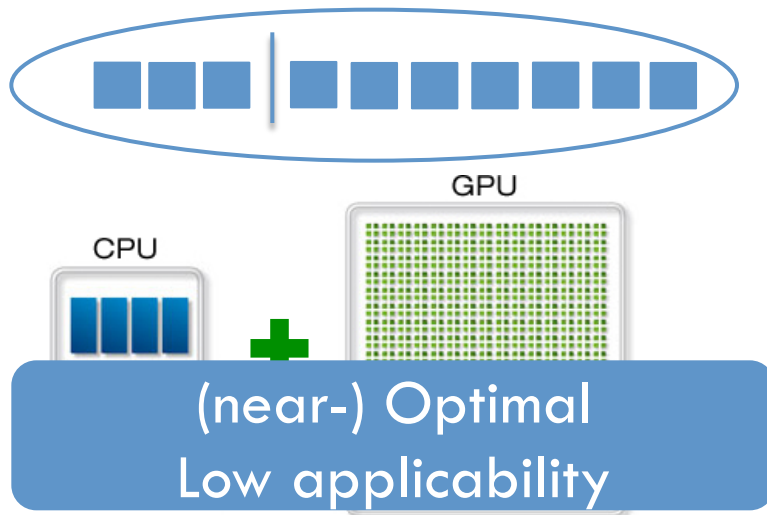
19

- 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

20

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



A simple taxonomy

Limited applicability.
Low overhead => high performance

**Qilin, Insieme, SKMD,
Glinda, ...**

Single
kernel

?

Static

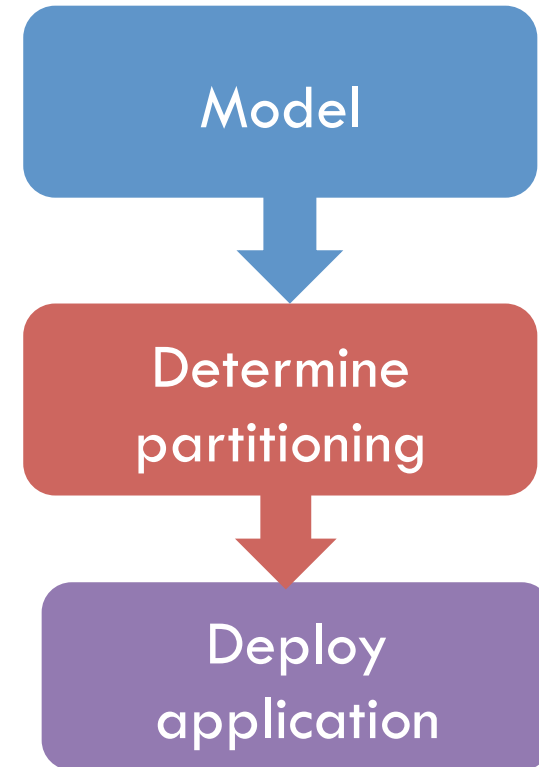
Dynamic

Our goal is to extend the use of static partitioning for as many applications as possible.

Dynamic partitioning is an excellent fallback scenario !

Static partitioning: Glinda*

- Model
 - ▣ 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

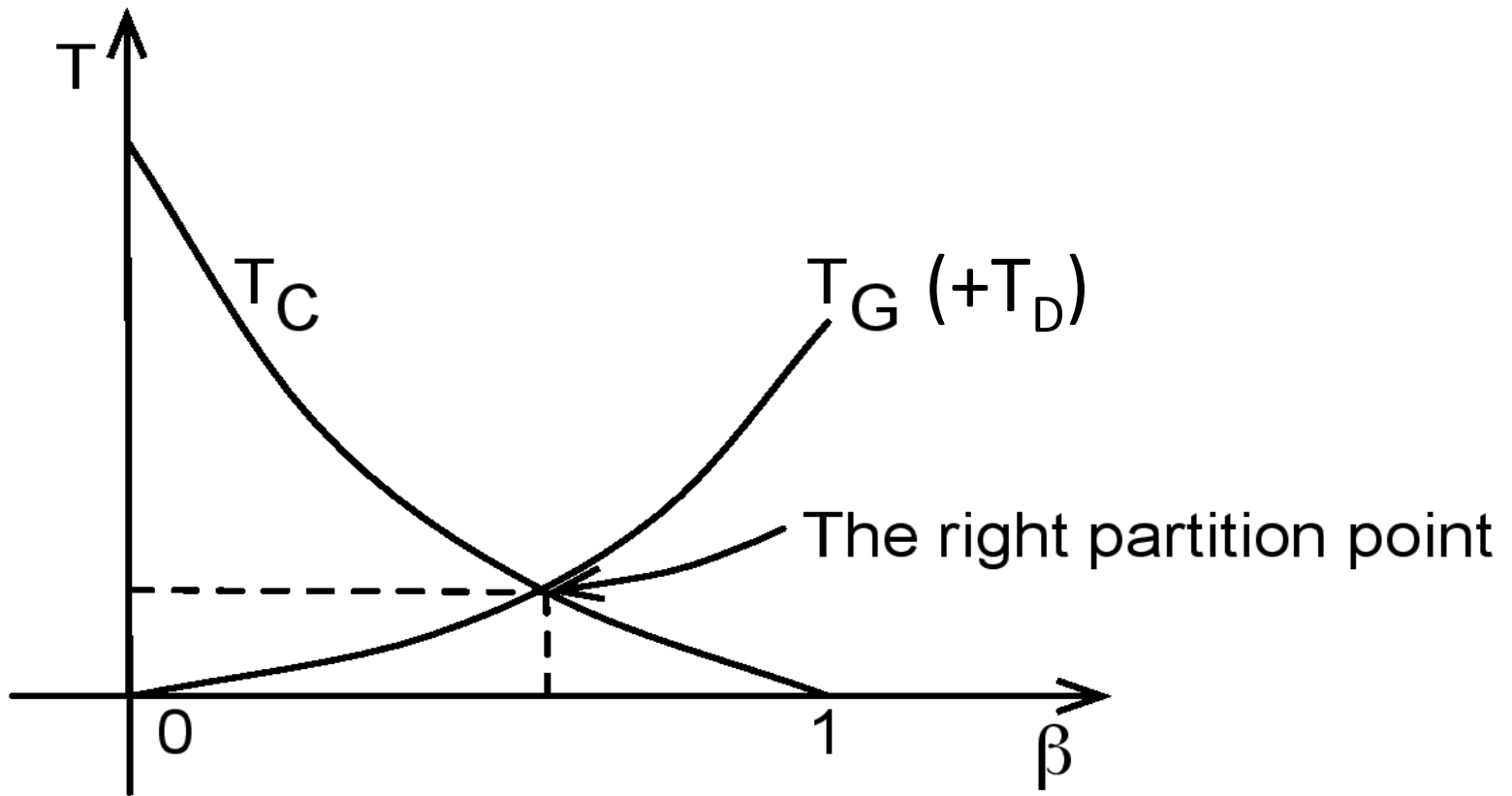


*Jie Shen et al., HPC'14.
"Look before you Leap: Using the Right Hardware.
Resources to Accelerate Applications

Model the partitioning

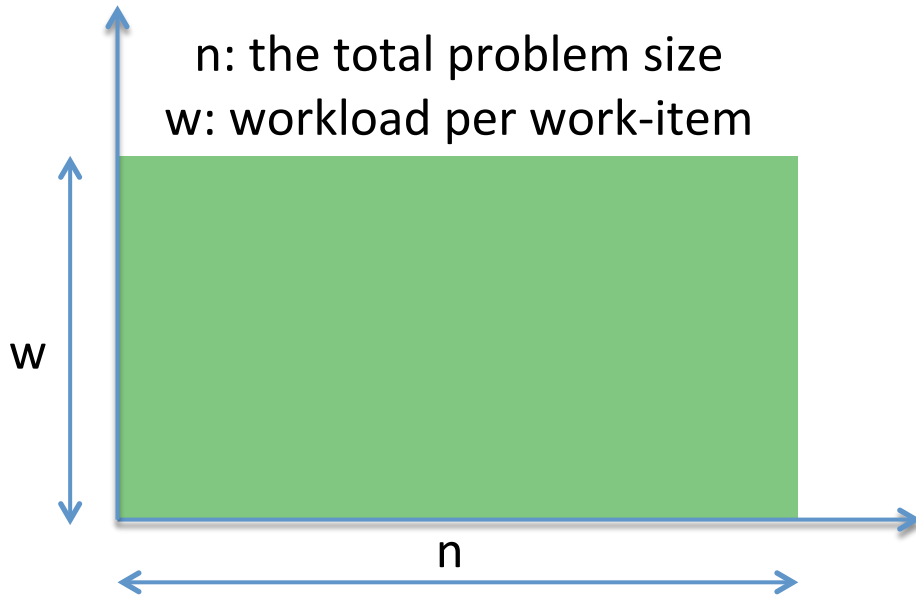
23

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



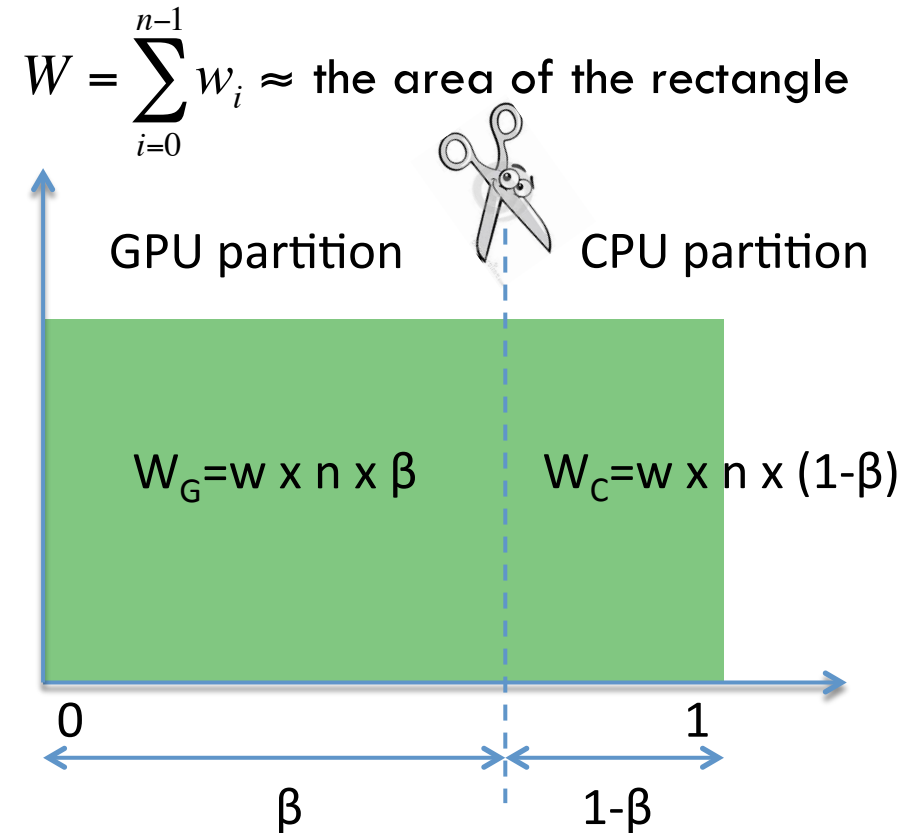
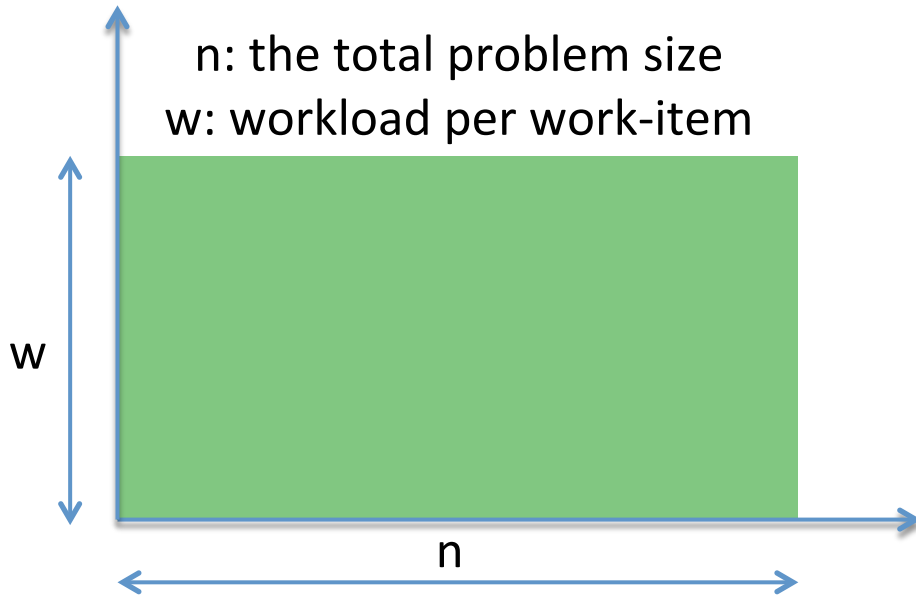
Model the workload

24



Model the workload

25



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

Model the hardware

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

Two pairs of metrics

W: total workload size

P: processing throughput (W/second)

O: data-transfer size

Q: data-transfer bandwidth (bytes/second)

$$T_G + T_D = T_C$$

$$W = W_G + W_C$$



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

Determine 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

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

β dependent terms

R_{GC}
CPU kernel execution time vs.
GPU kernel execution time

R_{GD}
GPU data-transfer time vs.
GPU kernel execution time

Determine the 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

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

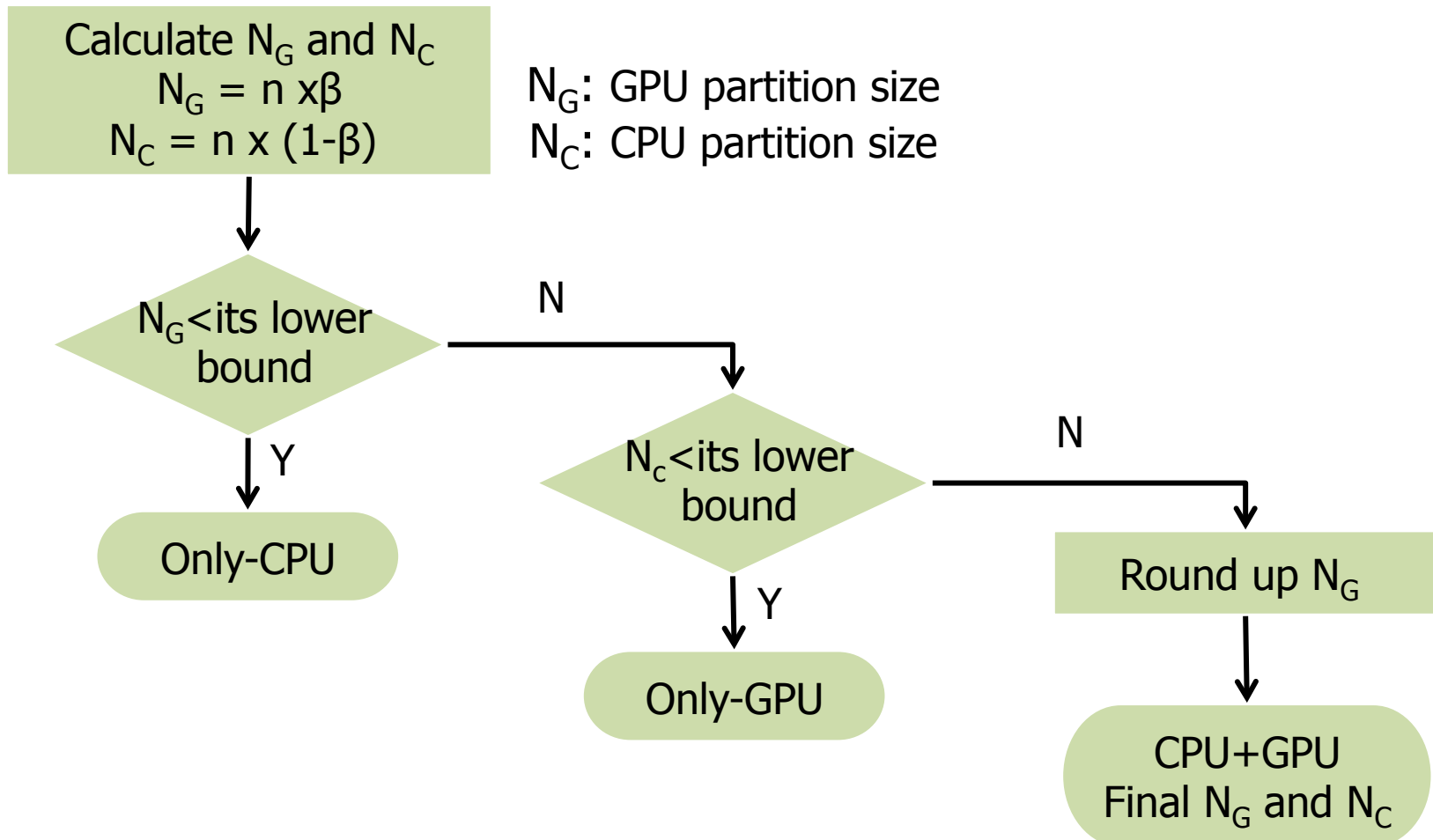
Partial data transfer

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

Full data transfer

Making the decision in practice

□ From β to a practical HW configuration



Glinda outcome

30

- A 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

Success story #1

- Applied Glinda for 7 (single-kernel) applications x 6 datasets per application
 - 42 tests
- 38/42 Glinda selected the best configuration
 - 14 CPU-only
 - 4 CPU+GPU incorrect
 - 20 CPU+GPU correct
- In all cases Glinda gains speed-up over GPU-only
 - 1.2x-14.6x speedup

How to use Glinda?

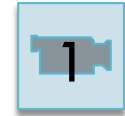
- Profile the platform to determine R_{GC} , R_{GD}
- Use the Glinda solver and determine β
- Take the decision: Only-CPU, Only-GPU, CPU+GPU (and partitioning)
 - ▣ if needed, apply the partitioning
- Code preparation
 - ▣ Parallel implementations for both CPUs and GPUs
 - ▣ Code templates for partitioning
 - ▣ Instrumentation for profiling
- Code reuse
 - ▣ Single-device code and multi-device code are reusable for different datasets and HW platforms

*HPL supports
Glinda.



Success story #2

Sound ray tracing

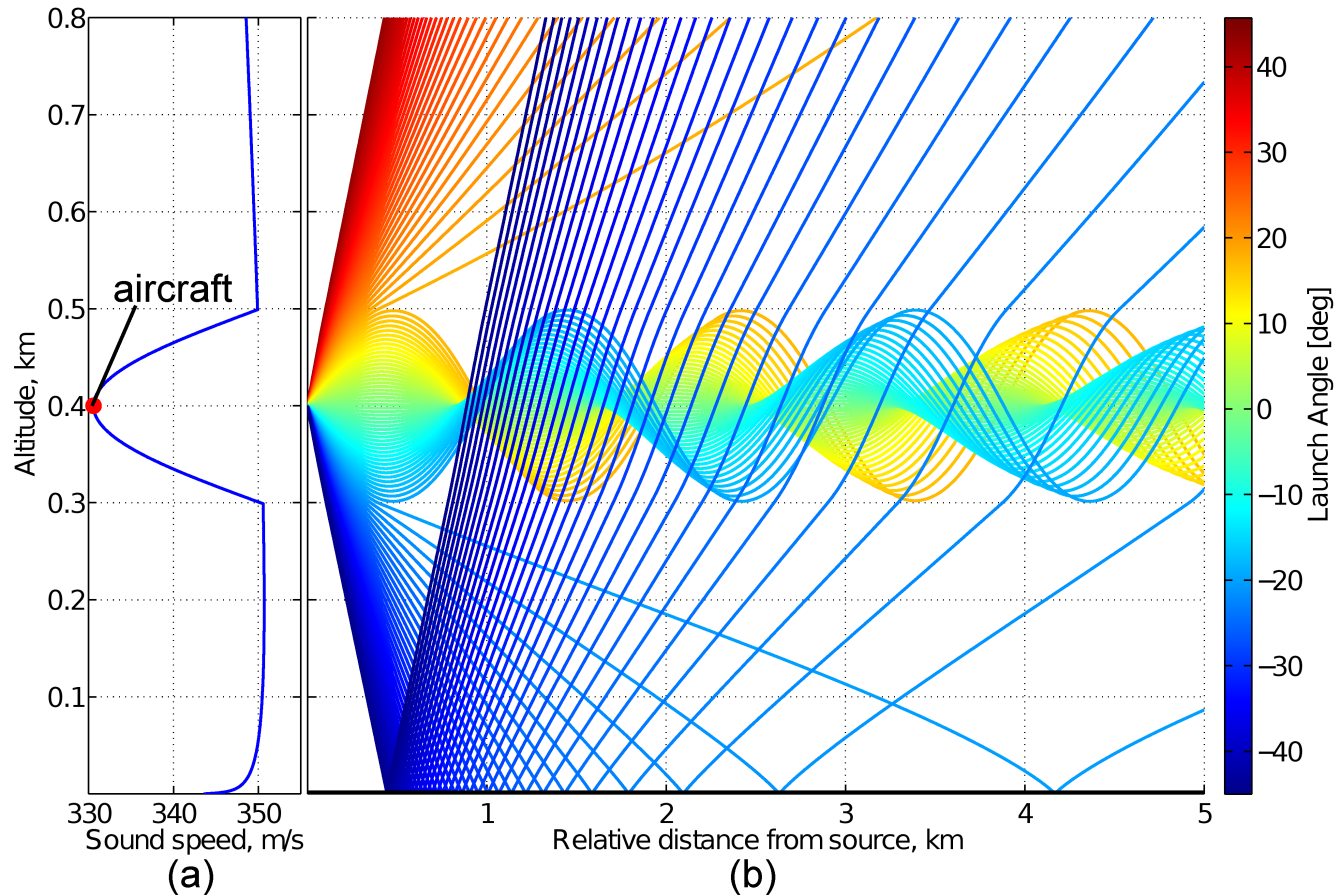


34



Sound ray tracing

35



Which hardware?

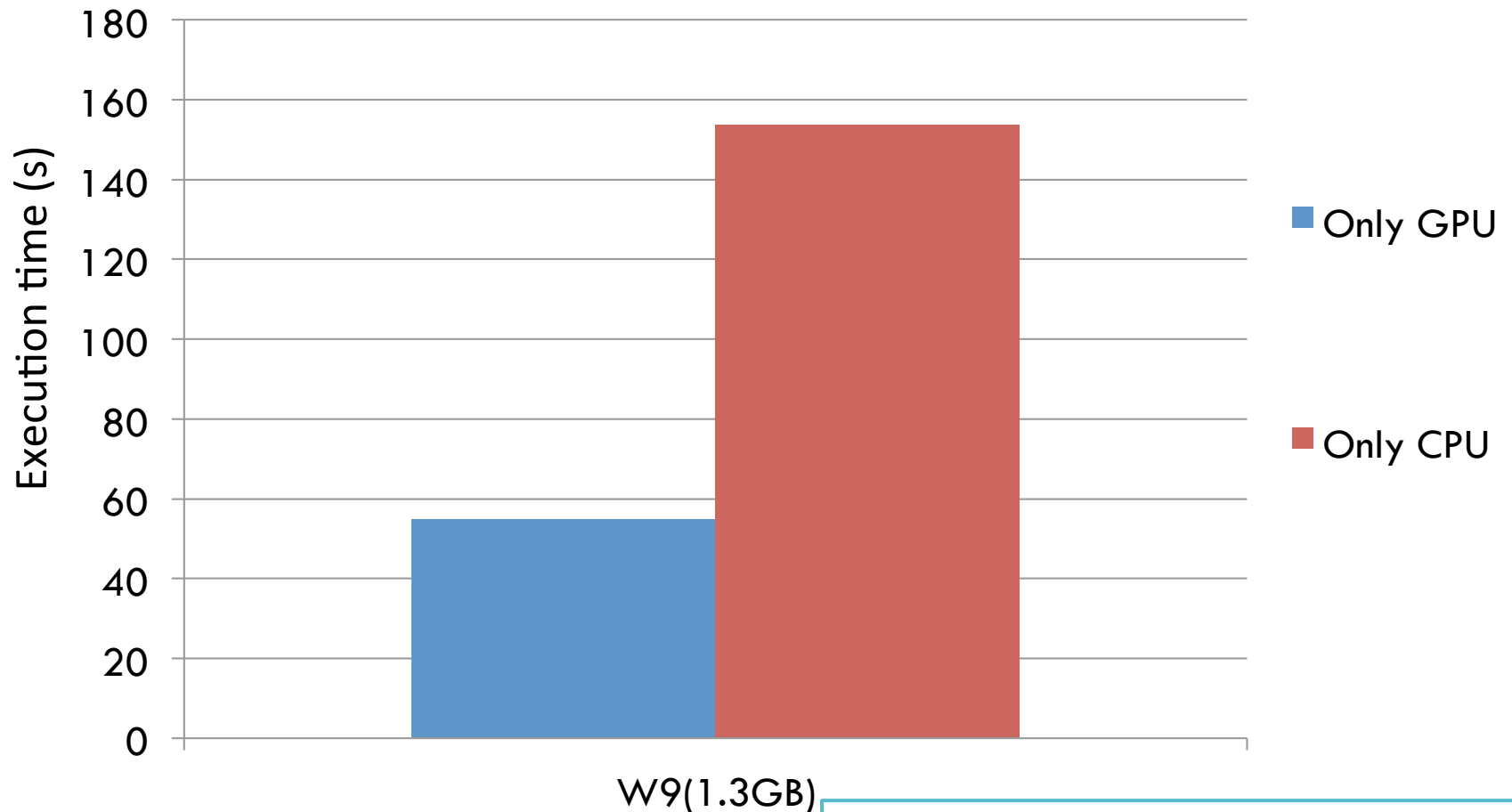
Our application has ...

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

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

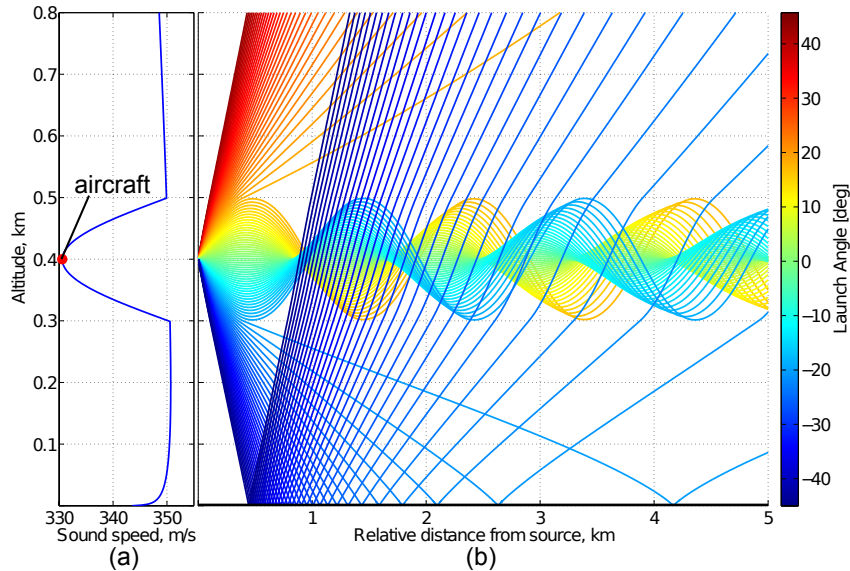
Initial Results

37

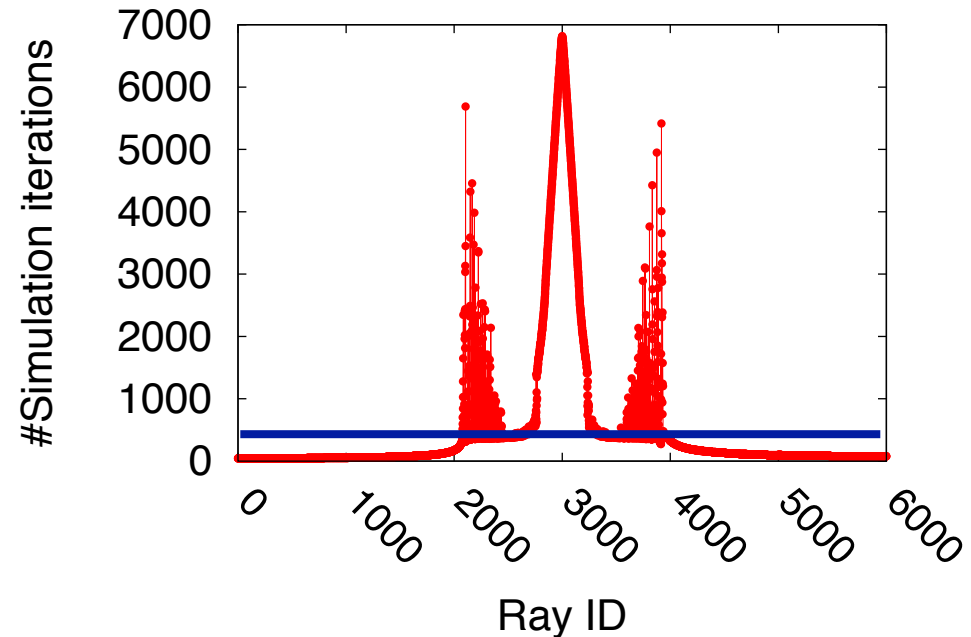


Only 2.2x performance improvement!
We expected 100x ...

Workload profile

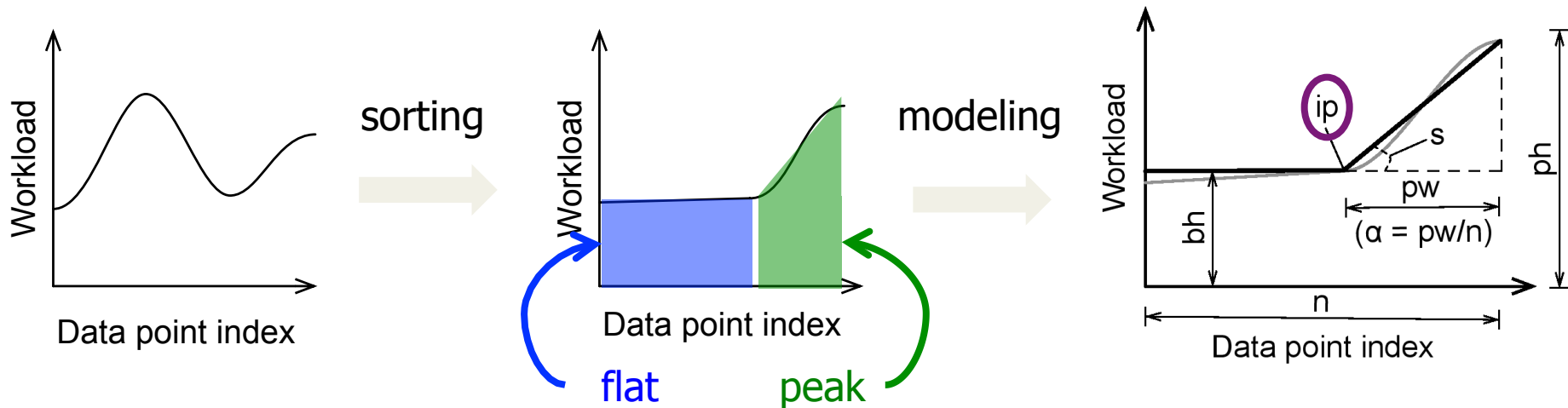


Peak
Processing iterations: ~ 7000

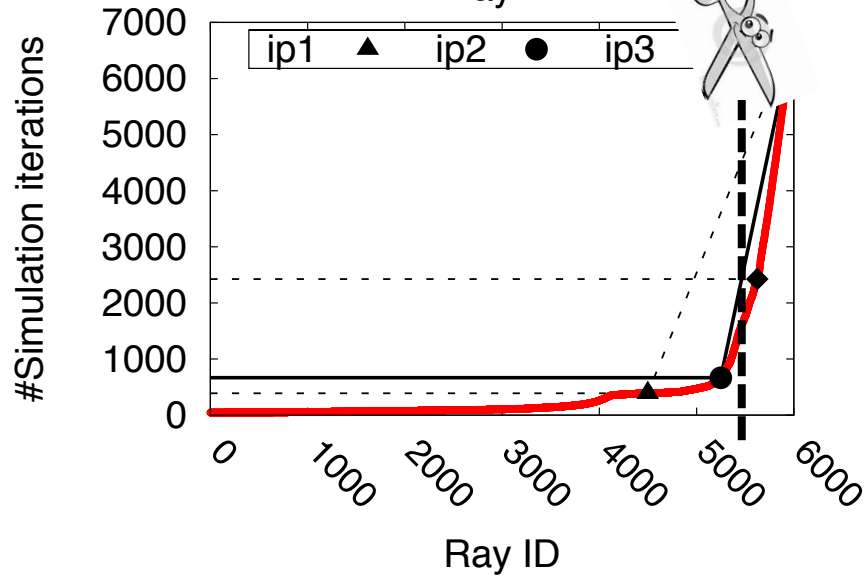
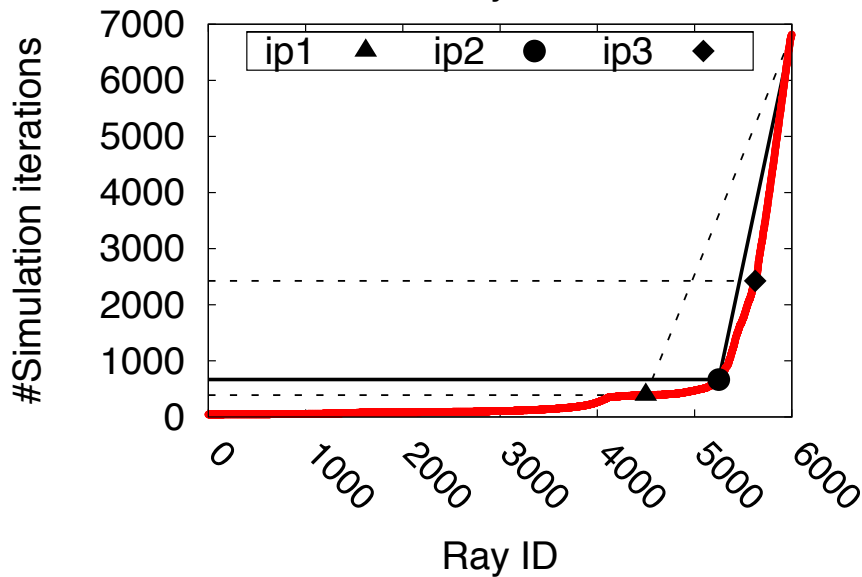
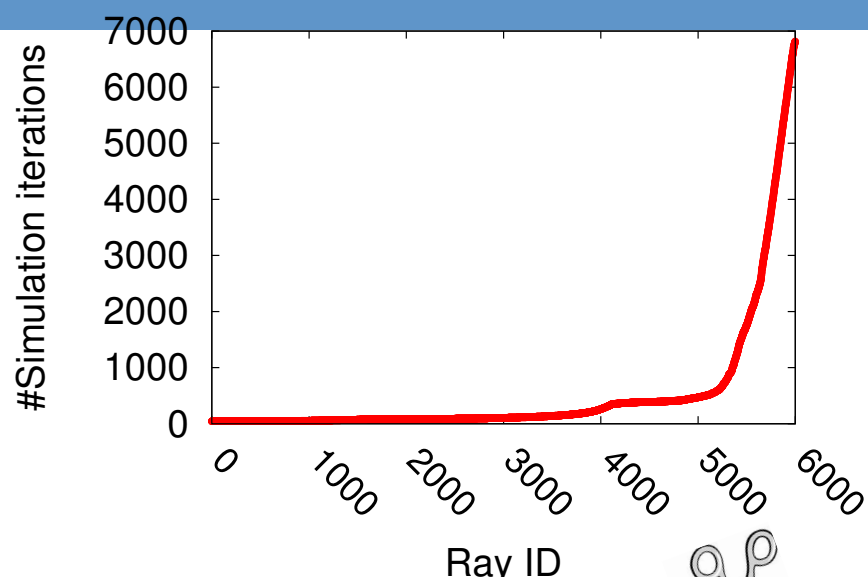
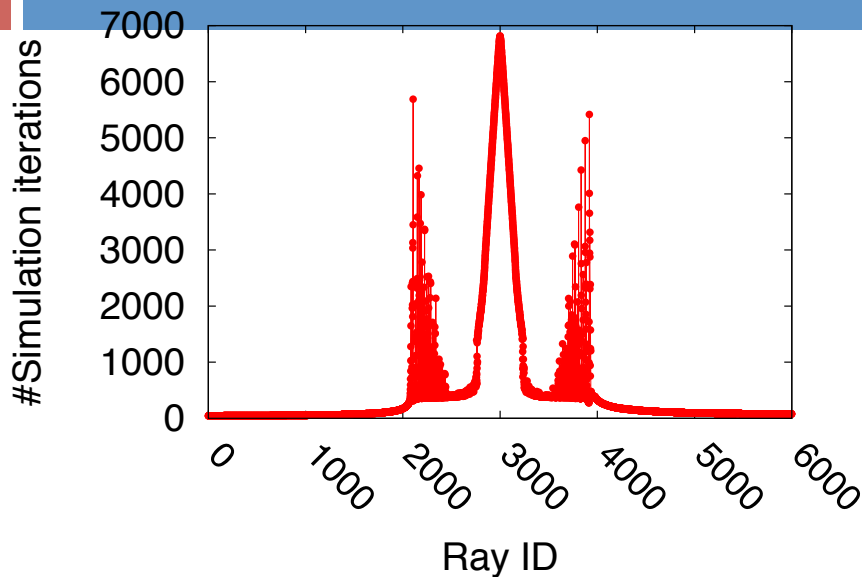


Bottom
Processing iterations: ~ 500

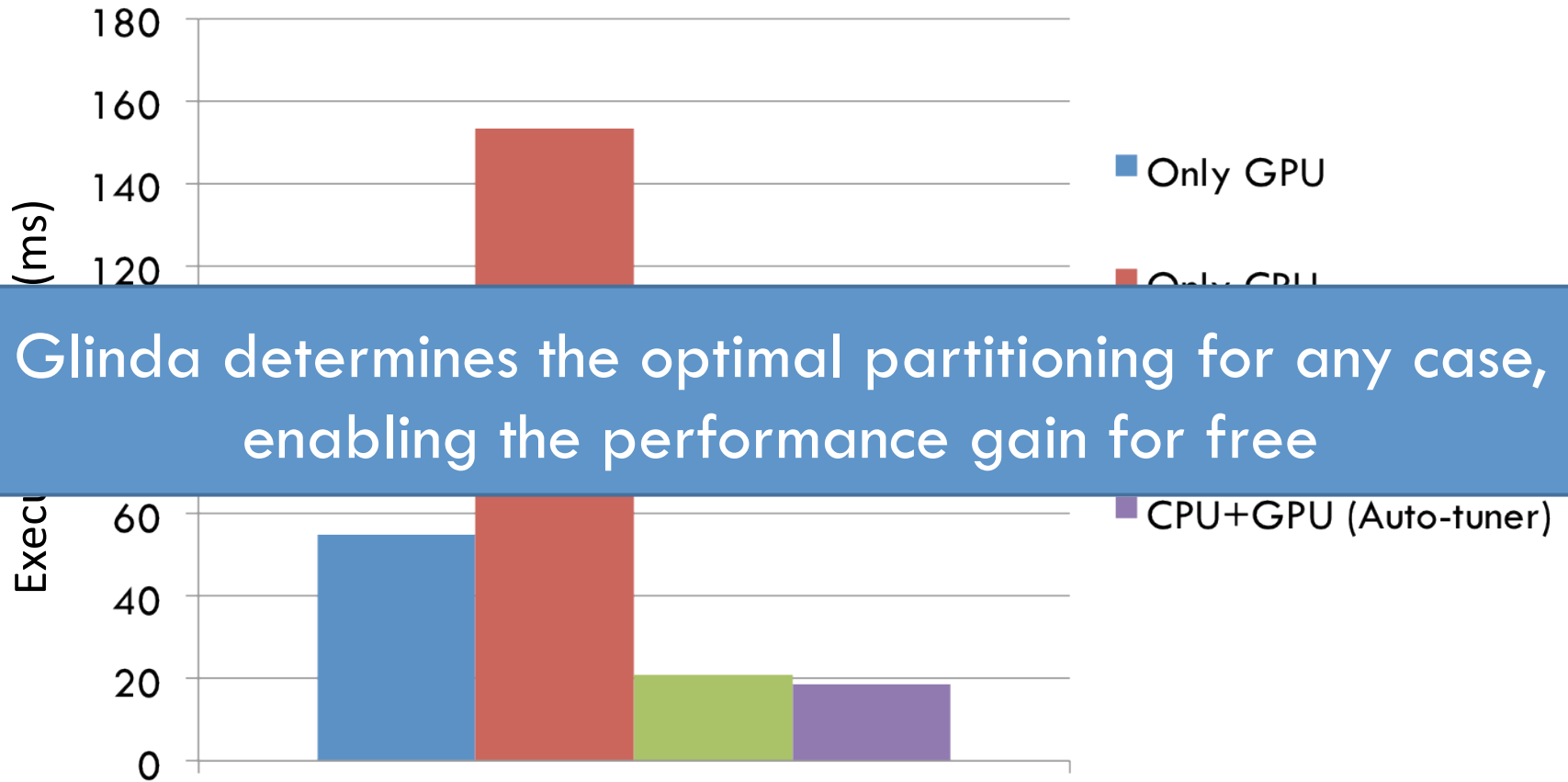
Modeling the imbalanced workload



Glinda for imbalanced workloads



Final results



Glinda determines the optimal partitioning for any case, enabling the performance gain for free

62% performance improvement compared to "Only-GPU"

More complex applications

- State-of-the-art: dynamic partitioning (OmpSs, StarPU)
 - ▣ Partition the kernels into chunks
 - ▣ Distribute chunks to *PUs
 - ▣ Keep data dependencies

Can we extend the use of static partitioning ?

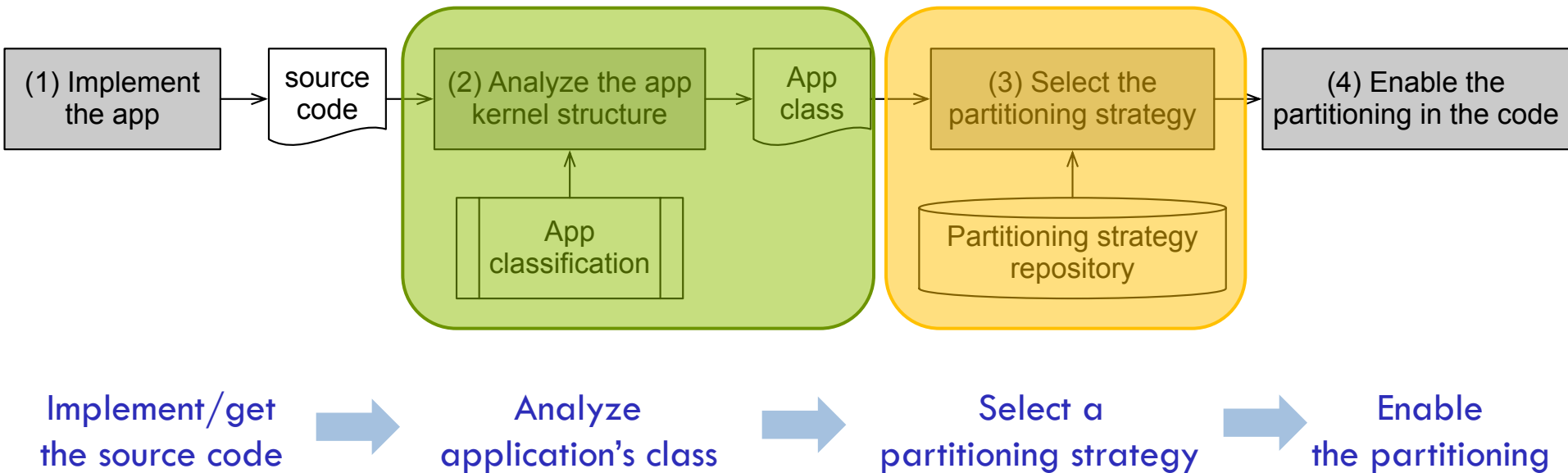
- ▣ (Often) leads to suboptimal performance
 - Scheduling policies and chunk size
 - Scheduling overhead (taking the decision, data transfer, etc.)

*Jie Shen et al., IEEE TPDS'16.

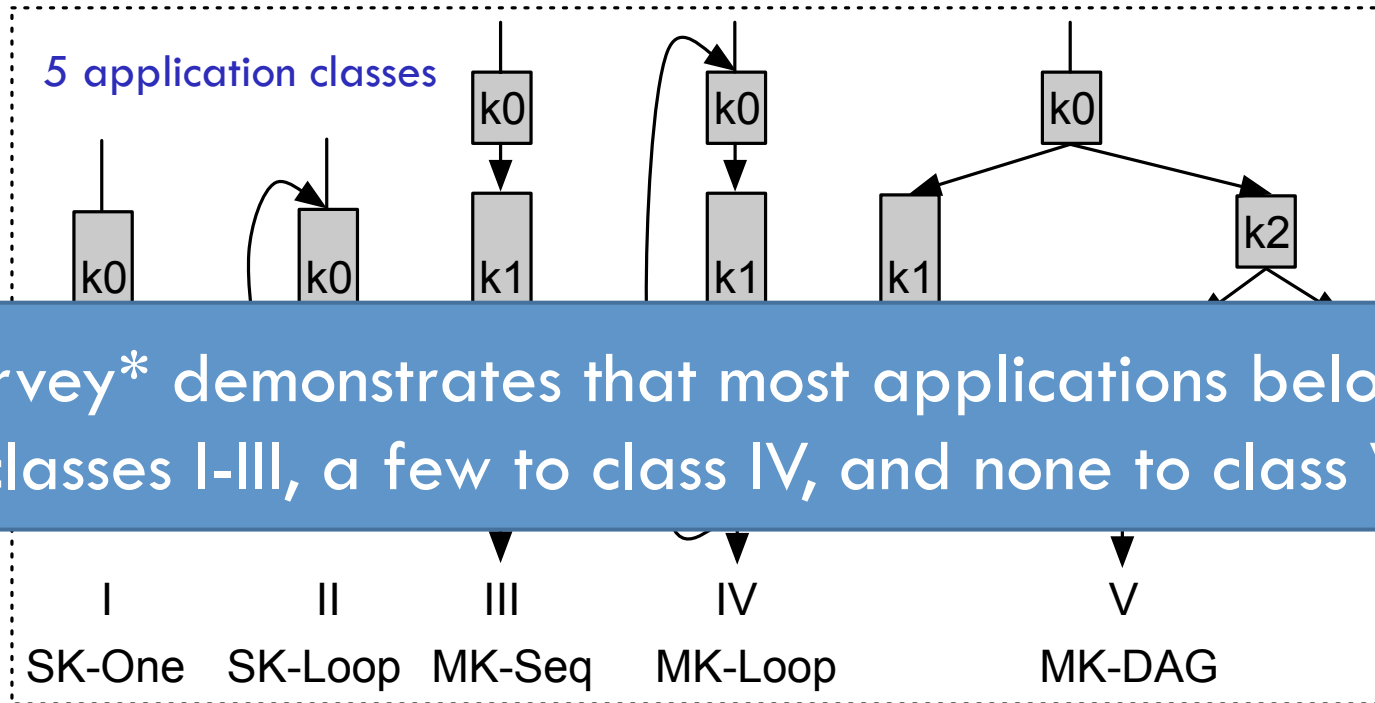
“Workload Partitioning for Accelerating Applications on Heterogeneous Platforms”

More complex applications

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



Application classification



A survey* demonstrates that most applications belong to classes I-III, a few to class IV, and none to class V.

Implement/get
the source code



Analyze
application's class



Select a
partitioning strategy

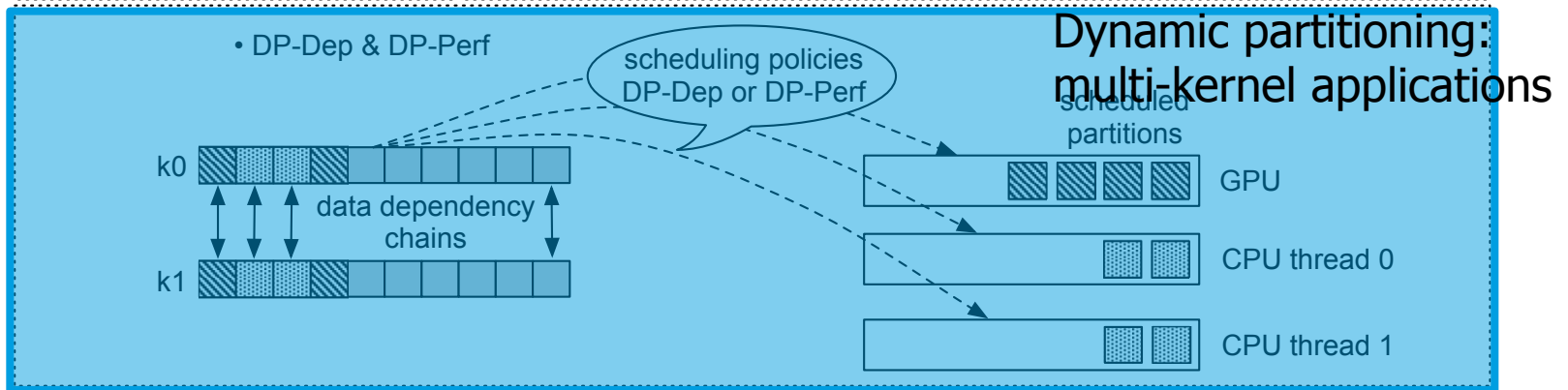
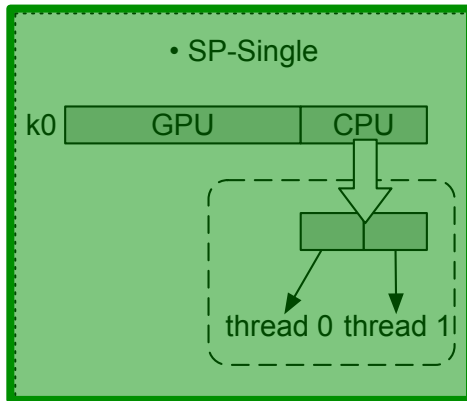


Enable
the partitioning

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

Partitioning strategies: before

Static partitioning:
single-kernel applications



Implement/get
the source code



Analyze
application's class



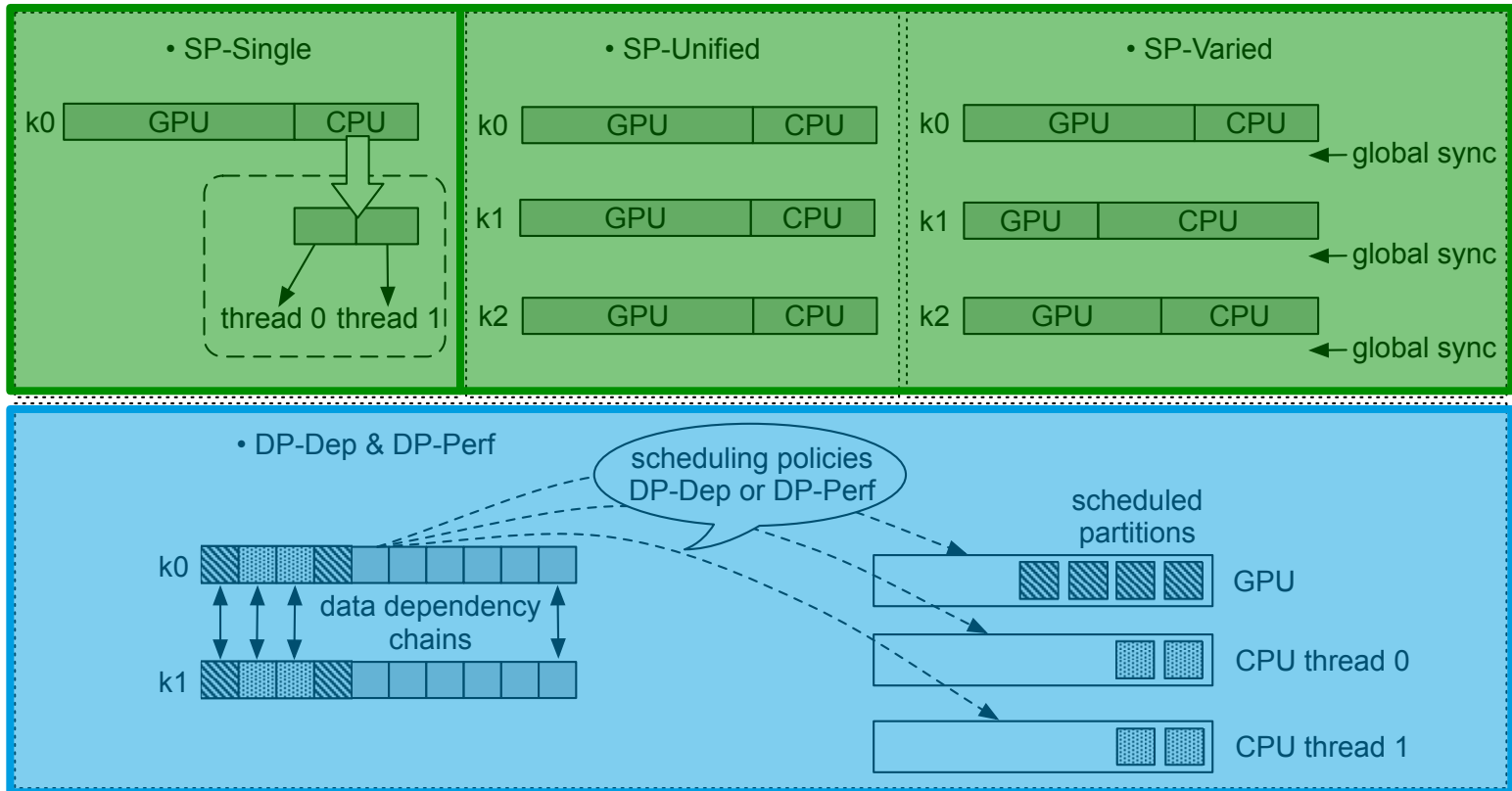
Select a
partitioning strategy



Enable
the partitioning

Partitioning strategies: now

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



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

Implemer
the source code

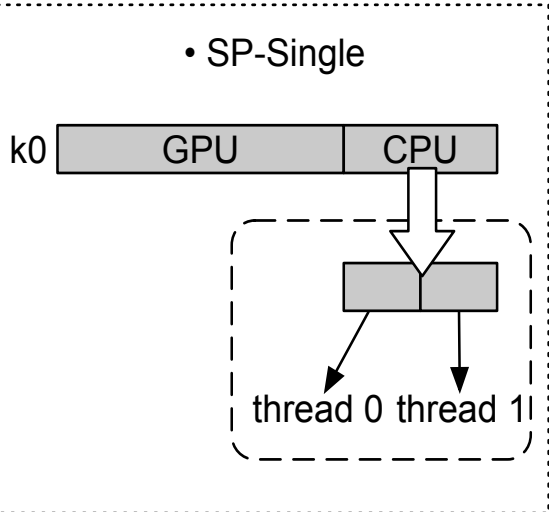
application's class

partitioning strategy

the partitioning

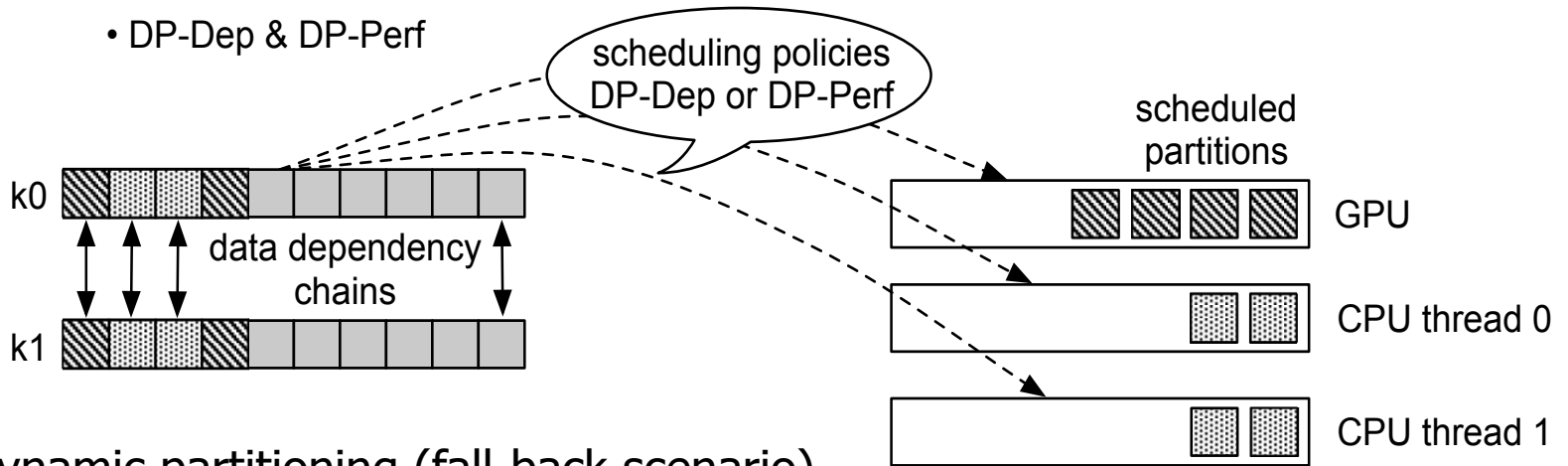
Partitioning strategies

Glinda: single-kernel



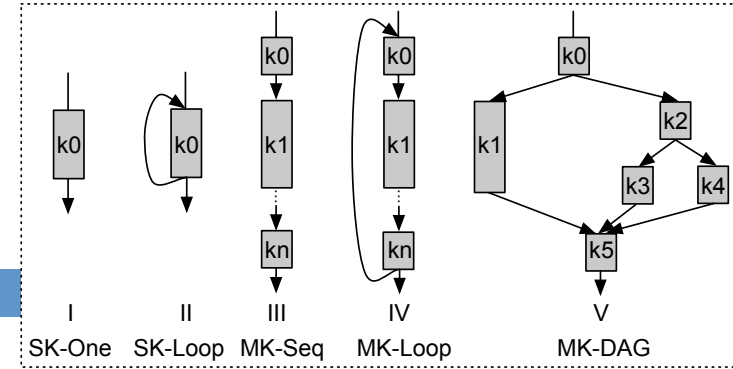
Glinda 2.0: multi-kernel

- DP-Dep & DP-Perf



OmpSs: dynamic partitioning (fall-back scenario)

Success story #3



- 6 applications
 - ▣ 2 type I, 2 type II, 1 type III, and 1 type IV
- Glinda detects and uses the best partitioning strategy
 - ▣ SP-Single for type I, II
 - ▣ SP-Unified or SP-Varied for types III and IV
 - Depends on the synchronization model
- In all cases, Glinda's static partitioning **outperforms** OmpSS' dynamic partitioning

Glinda is the only static partitioner to support multi-kernel applications.

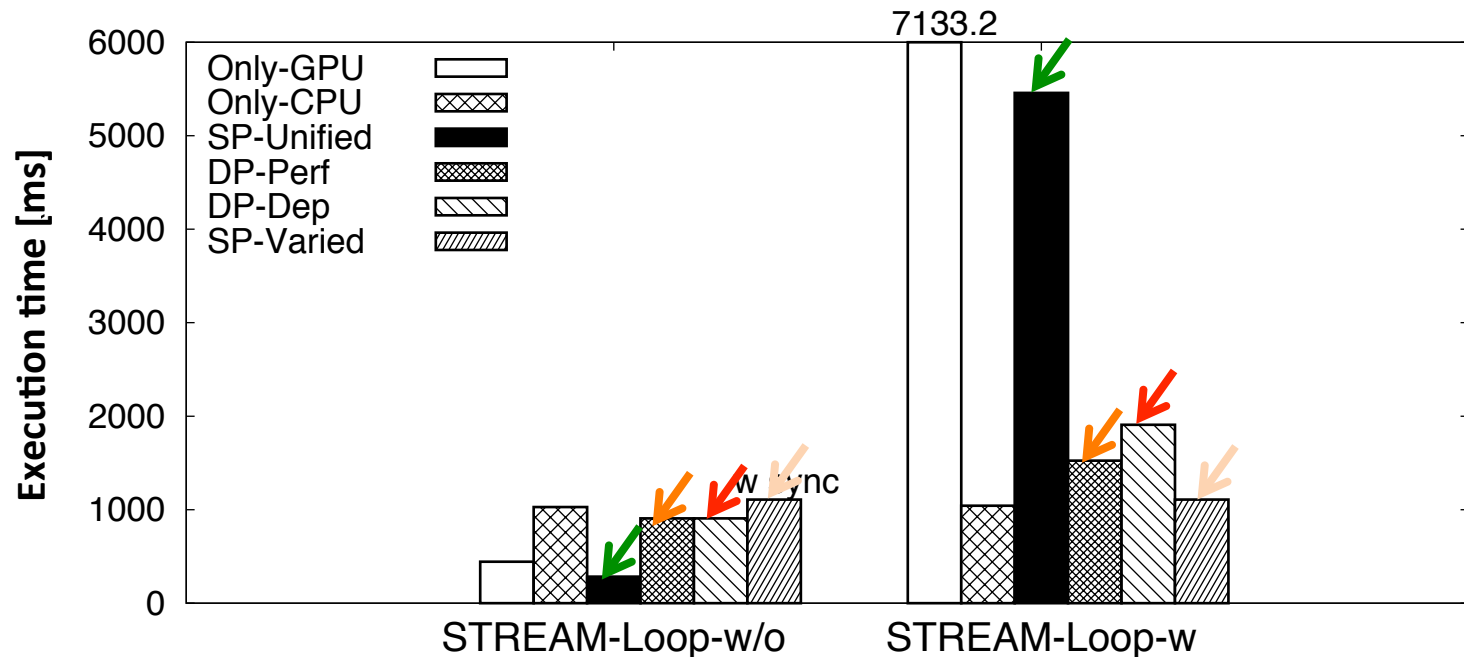
Results*

Similar results for all 6 applications with multiple kernels.

□ 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**



*Jie Shen et al., ICPP'15.

"Matchmaking Applications and Partitioning Strategies for Efficient Execution on Heterogeneous Platforms"

Heterogeneous Computing today*

Limited applicability.
Low overhead => high performance

Not interesting,
given that static &
run-time based
systems exist.

**Qilin, Insieme, SKMD,
Glinda, ...**

Sporadic attempts
and light runtime
systems

Static

Dynamic

**Only Glinda, for restricted
types of DAGs.**

**Run-time based systems:
StarPU
OmpSS
...**

Improved applicability, but
remains limited.
Low overhead => high
performance

High Applicability,
potentially high
overhead

Multi-kernel
(complex) DAG

*A.L.Varbanescu et al., FDL'16.

"Heterogeneous Computing with Accelerators: an Overview with Examples."



Instead of conclusion ...

to the office Take home message [1]



52

- Heterogeneous computing works!
 - ▣ *More resources.*
 - ▣ *Specialized resources.*
- Performance gain **for free**
 - ▣ Or at the price of some minor code changes
- Plenty of systems to support you
 - ▣ Different programming models
 - ▣ Generic systems for static/dynamic partitioning
 - ▣ Domain-specific/Application-specific models
 - Totem, HyGraph – graph processing
 - GlassWing – MapReduce
 - Cashmere – Divide&Conquer

to the office Take home message [2]

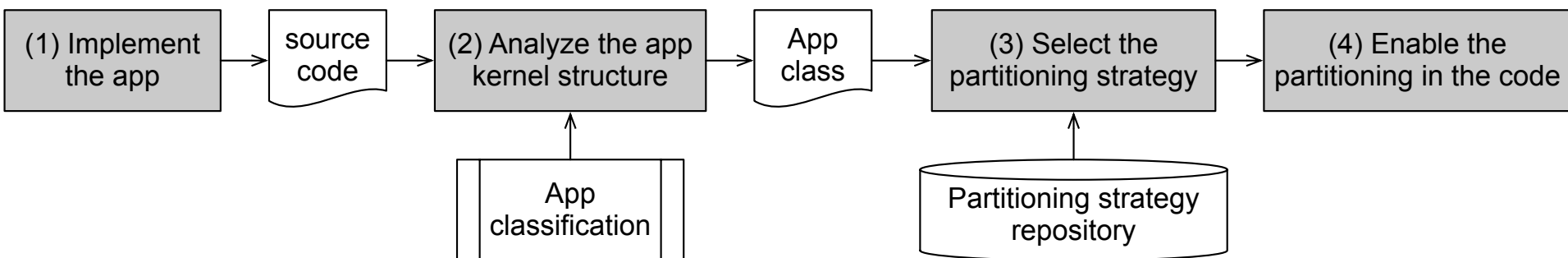
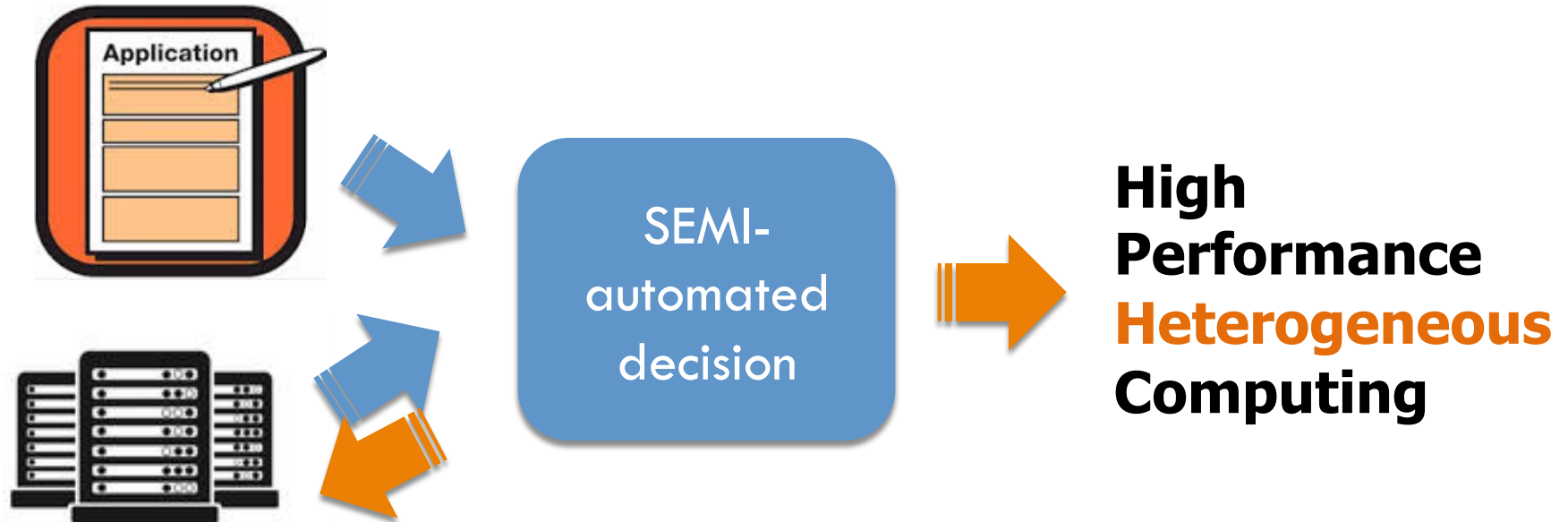


53

You have an application to run?

- **Now:** Choose one solution based on your application scenario:
 - ▣ Single-kernel vs. multi-kernel
 - ▣ Massive parallel vs. Data-dependent
 - ▣ Single run vs. Multiple run
 - ▣ Programming model of choice
- **WiP:** Framework to combine them all
 - ▣ Start from: Glinda + OmpSS
 - We are still working on it 😊

Ultimate goal (WiP)



Open questions

- Analytical modeling instead of profiling
- Unified programming models
 - ▣ Performance portability
- Extending to more specific types of workloads
- Performance modeling and prediction
 - ▣ What is the right hardware for my software?
- Understand the links with other fields where heterogeneous computing is already heavily used:
 - ▣ Embedded systems, cyber-physical systems, etc.

Heterogeneous computing works!

