A PRACTICAL GUIDE TO HETEROGENEOUS COMPUTING

Olivier Aumage, STORM Team, Inria Bordeaux, FR
olivier.aumage@inria.fr
Jie Shen, Delft University of Technology, NL
j.shen@tudelft.nl
Ana Lucia Varbanescu, University of Amserdam, NL
a.l.varbanescu@uva.nl

UvA Universiteit van Amsterdam
informatics mathematics

Today's headlines

- Heterogeneous computing
  - And why do we care?
- Programming heterogeneous systems
  - Challenges
  - The Landscape
- Part 1: Single-kernel approaches
- Part 2: Multi-kernel approaches
- Take home message

Goals

- Get you interested in heterogeneous computing.
- Introduce methods for efficient heterogeneous computing
  - Single-node
  - Large-scale
- Practical examples
- Debate current challenges and open research questions.

- Fair to others, but advertise our research ☺

Terminology

- Multi-cores = CPUs
- Many-cores = GPUs
  - Used as accelerators
- Heterogeneous platform: mix of CPUs and GPUs
  - n x CPU + m x GPU
- Heterogeneous computing: compute using heterogeneous platforms (efficiently).

Assumption

- Multi-core and many-core processors are devices to be used for performance improvement of applications.
  - Corollary :
  - The performance gain due to the use of multi-/many-cores is highly dependent on the way applications make use of the hardware platform and **ALL** its components.

Why do we care ?

Multi-core & Many-core

IBM

intel

ARM

Do heterogeneous machines exist?

- Top 500 (June 2014)
  - State of the art in HPC (top500.org)

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National University of Defense Technology China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3120000	33862.7	54902.4	17808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560640	17590.0	27112.5	8209
3	DOE/NNSA/LLNL United States		1572864	17173.2	20132.7	7890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	Fujitsu	24	10510.0	11280.4	12660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786432	8586.6	10066.3	3945

195 cores/ node!

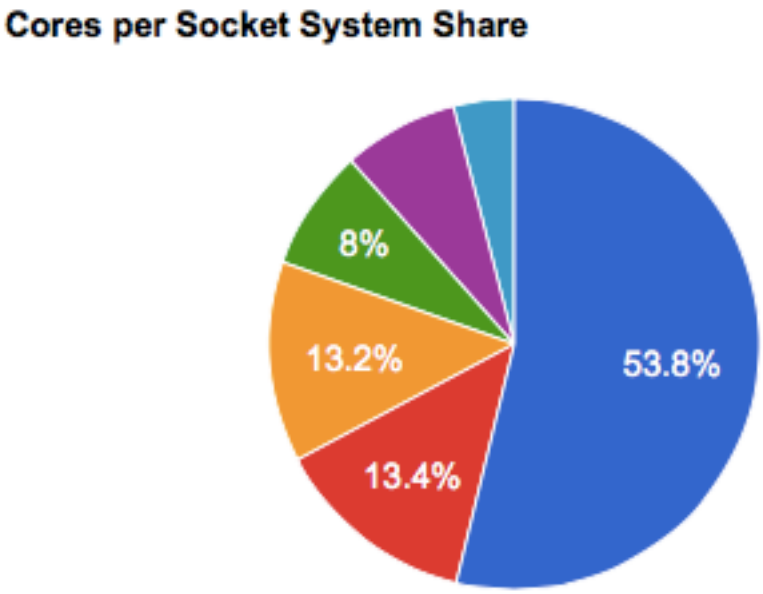
Heterogeneous!

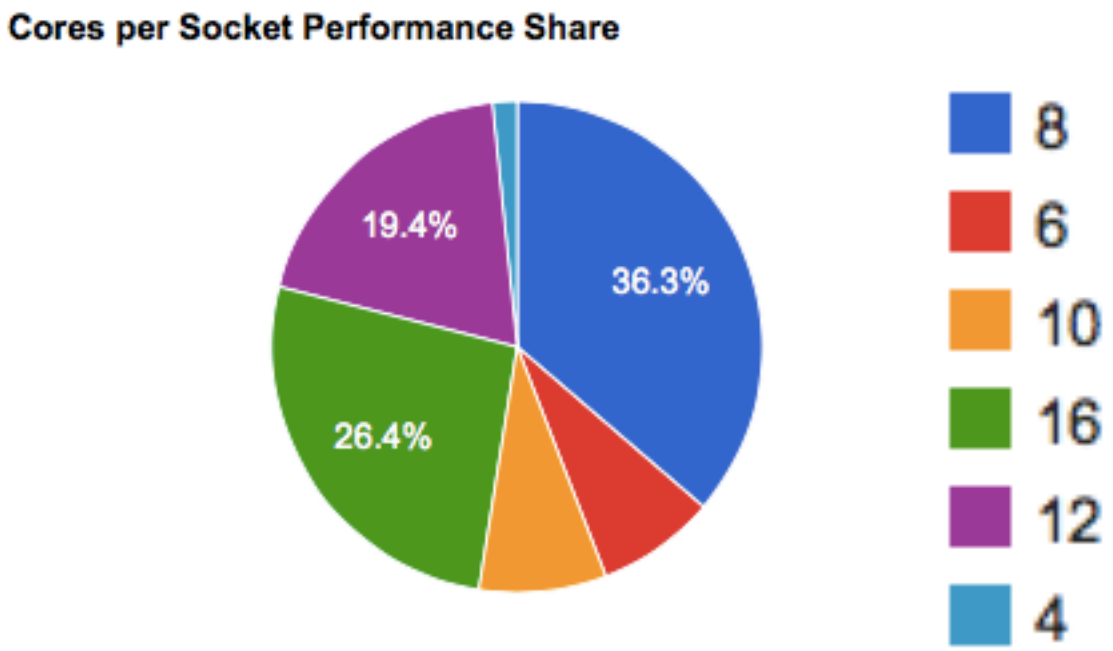
Heterogeneous!

64 heterogeneous machines in total.

Top500 – cores/socket

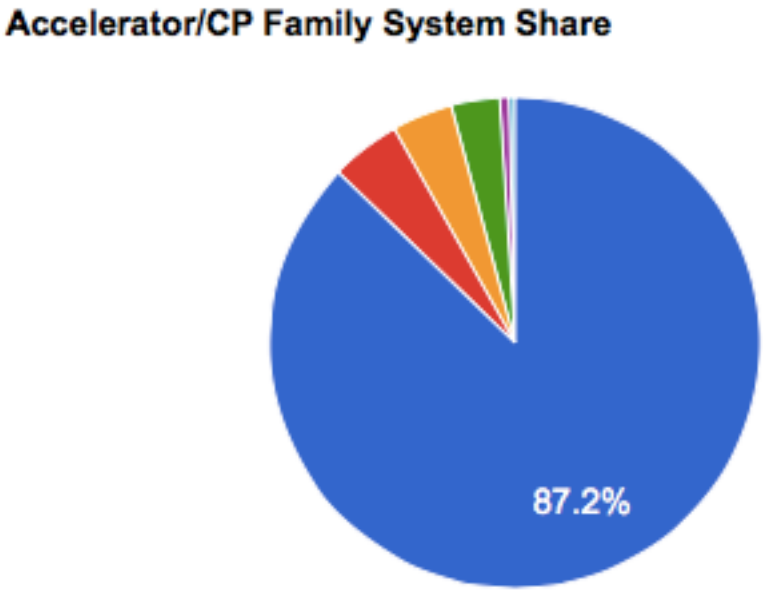
- Virtually all machines use multi-core CPUs

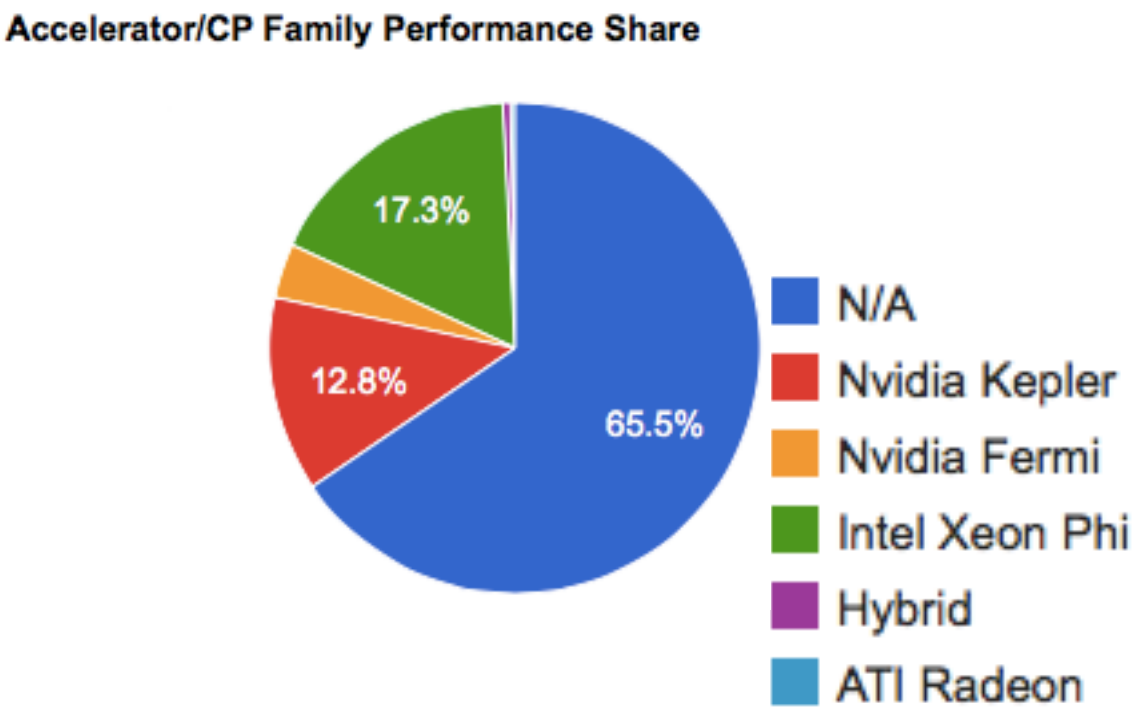




Top500 – accelerators

- Relatively low numbers
- High performance impact





NUDT's Tianhe-2 (China)

- #1 in Top500
  - 54.902 PFLOPs peak
  - 33.862 PFLOPs max
- 16.000 nodes
- = 16.000 x (2 x Xeon IvyBridge + 3 x Xeon Phi)
- = 3.120.000 cores (=> 195 cores/node)

Other such lists?

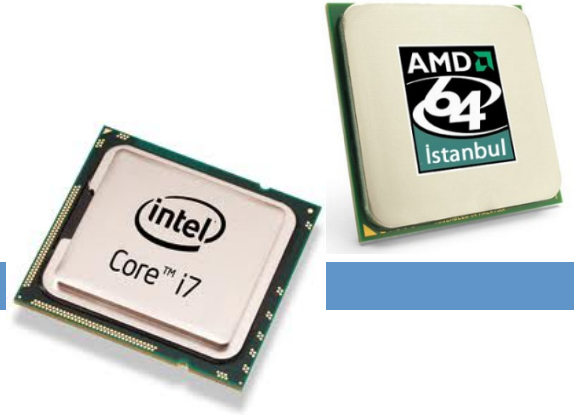
Research idea?

- Graph500
  - Not quite so many heterogeneous machines.
- Green500
  - Many heterogeneous machines.

CPUs vs GPUs

Competition, Coexistence, or Collaboration?

(Multi-core) CPUs



- Architecture
  - Few large cores (+ integrated GPUs)
  - Vector units
    - SSE, AVX
  - Stand-alone
- Memory
  - Shared, multi-layered
  - Per-core caches + shared caches
- Programming
  - Multi-threading
  - OS Scheduler

Parallelism

- Core-level parallelism ~ task/data parallelism (coarse)
  - 4-12 of powerful cores
    - Hardware hyperthreading (2x, 4x)
  - Local caches
  - Symmetrical or asymmetrical threading model
    - Implemented by the programmer / programming model
- SIMD parallelism = data parallelism (fine)
  - Sensitive to divergence
    - NOT the same instruction => performance loss
  - Implemented by programmer OR compiler

Programming models

- Pthreads + intrinsics
- TBB – Thread building blocks
  - Threading library
- OpenCL
  - To be discussed …
- OpenMP
  - Traditional parallel library
  - High-level, pragma-based
- Cilk
  - Simple divide-and-conquer model

(NVIDIA) GPUs

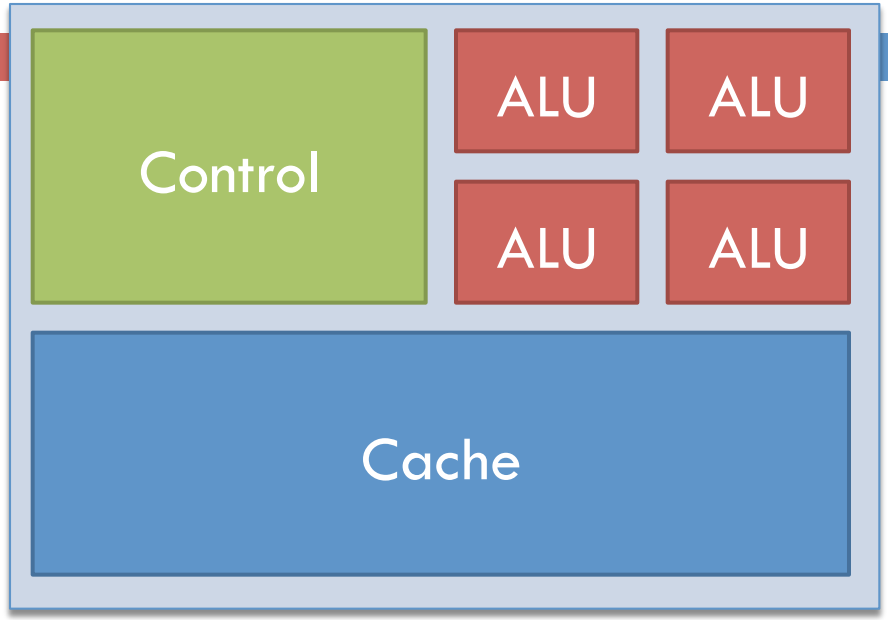
- Architecture
  - Many (100s) slim cores
  - Sets of cores grouped into “multiprocessors” with shared memory (SM(X) = stream multiprocessors)
  - Work as accelerators
    - A “host” is mandatory, with/without communication via PCI-e
- Memory
  - Shared L2 cache
  - Per-”core” caches + shared memory
  - Off-chip global memory
- Programming
  - Symmetric multi-threading
  - Hardware scheduler

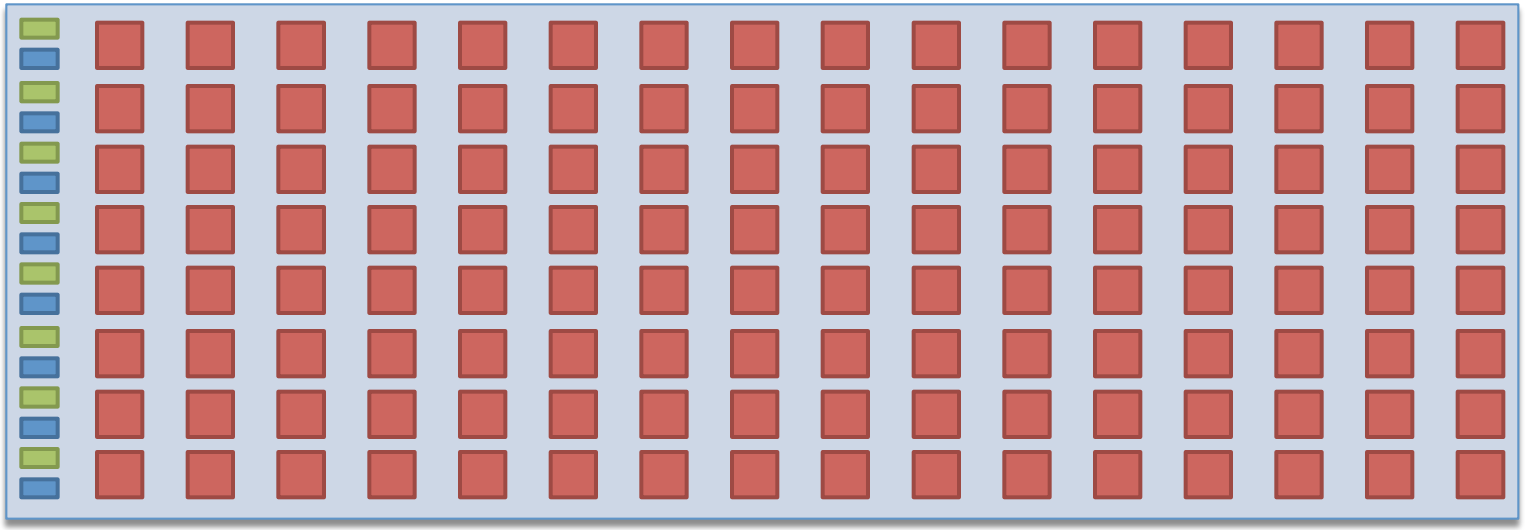
Parallelism

- Data parallelism (fine-grain)
- SIMT (Single Instruction Multiple Thread) execution
  - Many threads execute concurrently
    - Same instruction
    - Different data elements
    - HW automatically handles divergence
  - Not same as SIMD because of multiple register sets, addresses, and flow paths*
  - Hardware multithreading
    - HW resource allocation & thread scheduling
      - Excess of threads to hide latency
      - Context switching is (basically) free

*http://yosefk.com/blog/simd-simt-smt-parallelism-in-nvidia-gpus.html

CPU vs. GPU





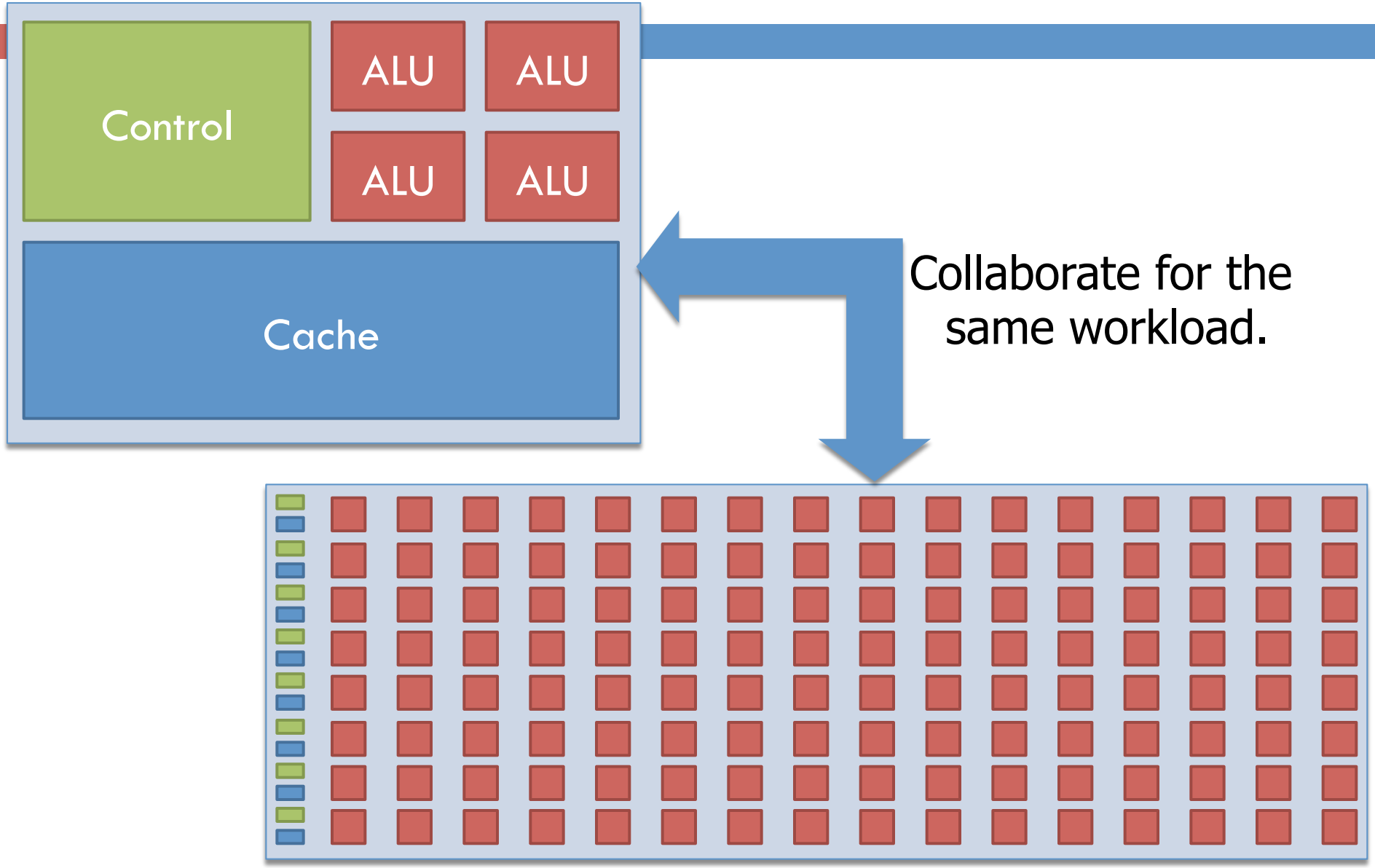
CPU vs. GPU

- Movie
- The Mythbusters
  - Jamie Hyneman & Adam Savage
  - Discovery Channel
- Appearance at NVIDIA's NVISION 2008

Why so different?

- Different goals produce different designs!
  - CPU must be good at everything
  - GPUs focus on massive parallelism
    - Less flexible, more specialized
- CPU: minimize latency experienced by 1 thread
  - big on-chip caches
  - sophisticated control logic
- GPU: maximize throughput of all threads
  - # threads in flight limited by resources => lots of resources (registers, etc.)
  - multithreading can hide latency => no big caches
  - share control logic across many threads

Reminder


Control
ALU
ALU
ALU
ALU
Cache
Collaborate for the same workload.

Collaborate?

- Why?
  - Gain performance
    - Different workload parts for different processors
    - Availability
    - Data communication bottleneck
- When?
  - Not all applications work best on a GPU.
- How?
  - Communication.
  - Partitioning.

Programming many-cores

Programming models

- Vendor-specific: e.g., NVIDIA CUDA
  - Portability concerns only legacy code
  - (Productivity is increased by hiding more and more details of the machine, assumed known)
- "Community" programming models
  - High-level: OpenMP, OpenACC, SAC, …
  - Not-so-high-level: OpenCL (may call it low, but is unfair)
  - (Very-)Low-level: assembly, C, intrinsics, …

OpenCL in a nutshell

- Open standard for portable multi-core programming
- Architecture independent
  - Explicit support for multi-/many-cores
- Low-level host API
    - High-level bindings (e.g., Java, Python)
- Separate kernel language
- Run-time compilation

Programming in OpenCL

- Kernels are the main functional units in OpenCL
  - Kernels are executed by work-items
  - Work-items are mapped transparently on the hardware platform
- Functional portability is guaranteed
  - Programs run correctly on different families of hardware
  - Explicit platform-specific optimizations are dangerous
- Performance portability is another discussion
  - Ask me about it!

Heterogeneous programming?

- Mixes different languages for different "kernels"
  - OpenMP + CUDA
  - TBB + CUDA
  - …
- Same language (+ pragma's)
  - OpenCL
  - OpenACC
  - HPL
  - …
- Specific APIs

End of Preliminaries

- Questions

2-mins break

- Networking is important!

- Stand-up.
- Shake hands and exchange name and affiliation with one of your neighbours.
- Tell your other neighbour where she/he is from.

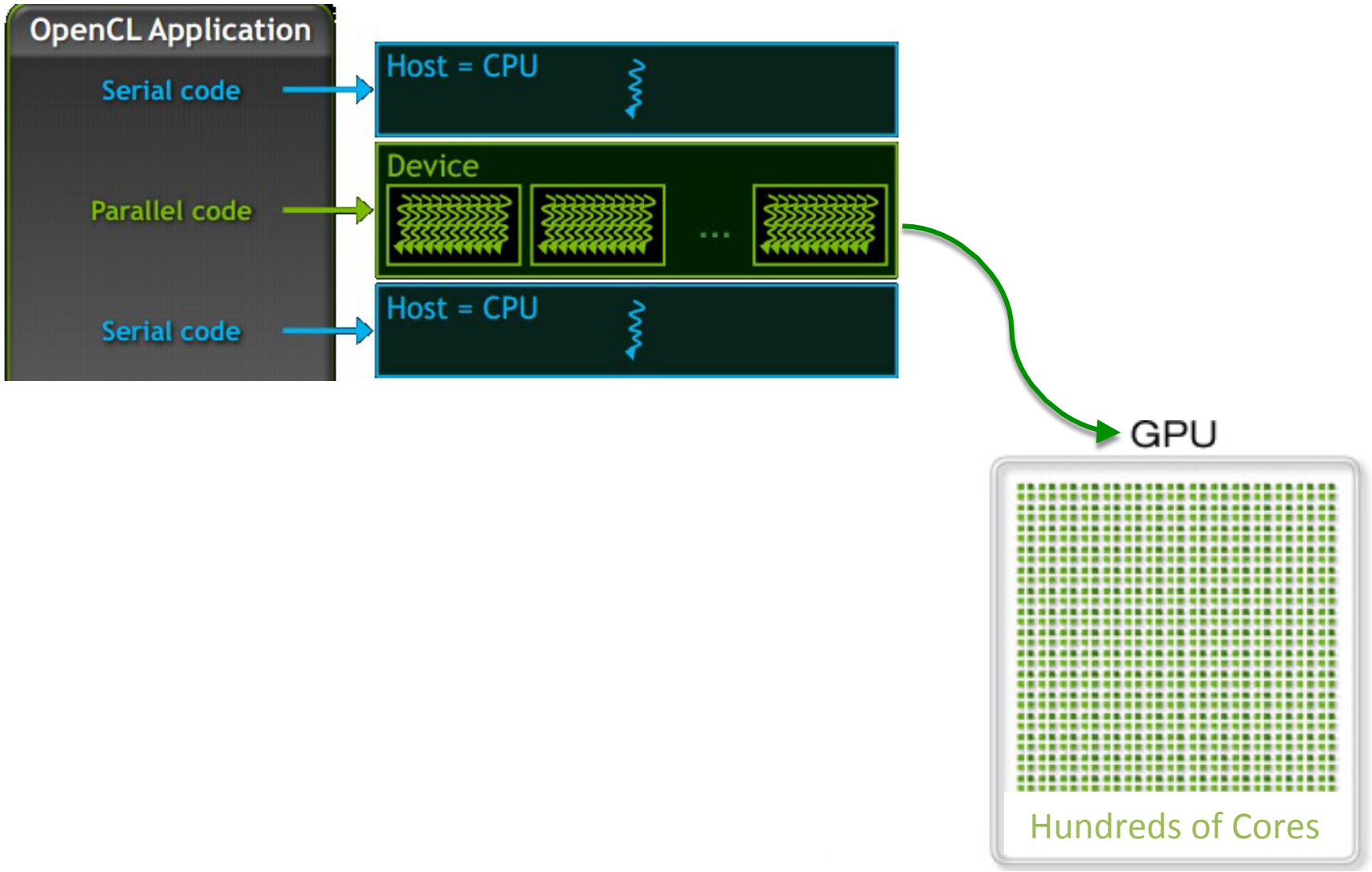
End of Preliminaries

- Questions

Heterogeneous computing

Main challenges

When using GPUs …



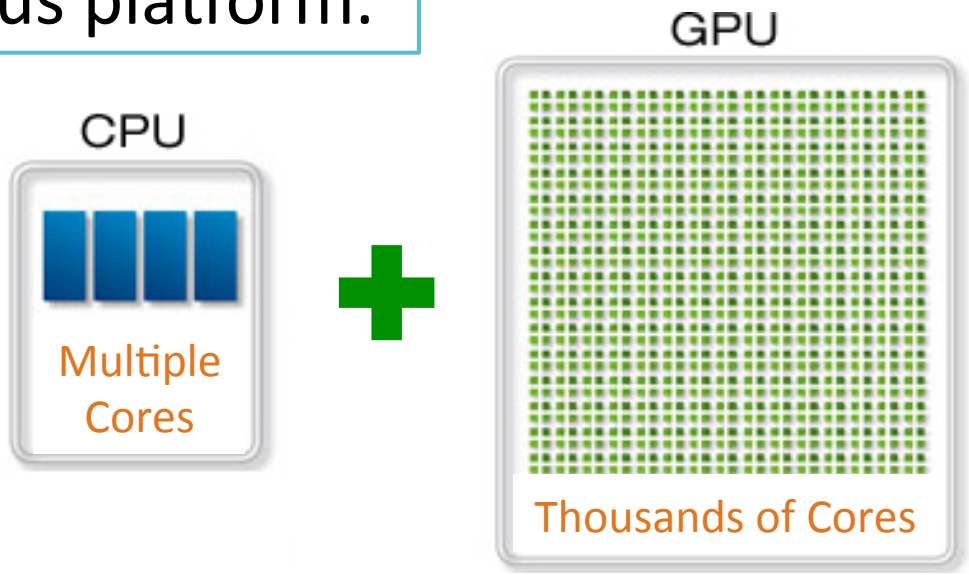
We must consider...

- Application parallelism
- CPU-GPU data transfers
- CPU is idle!

We must consider...

- Application parallelism
- CPU-GPU data transfers
- CPU is idle!

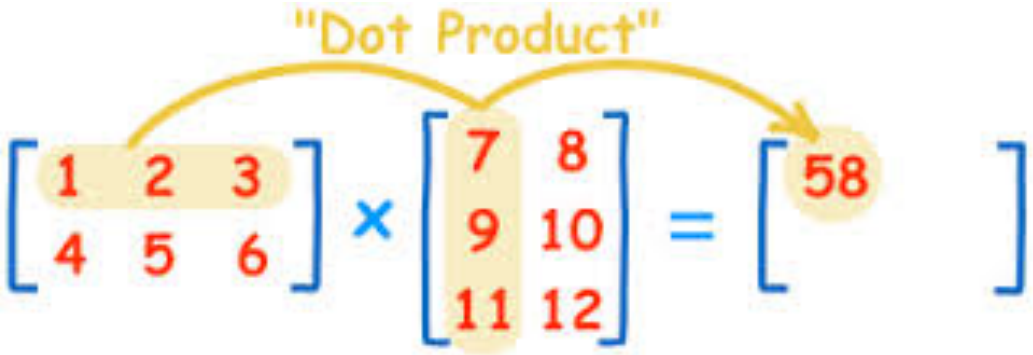
Make the best out of the existing heterogeneous platform.



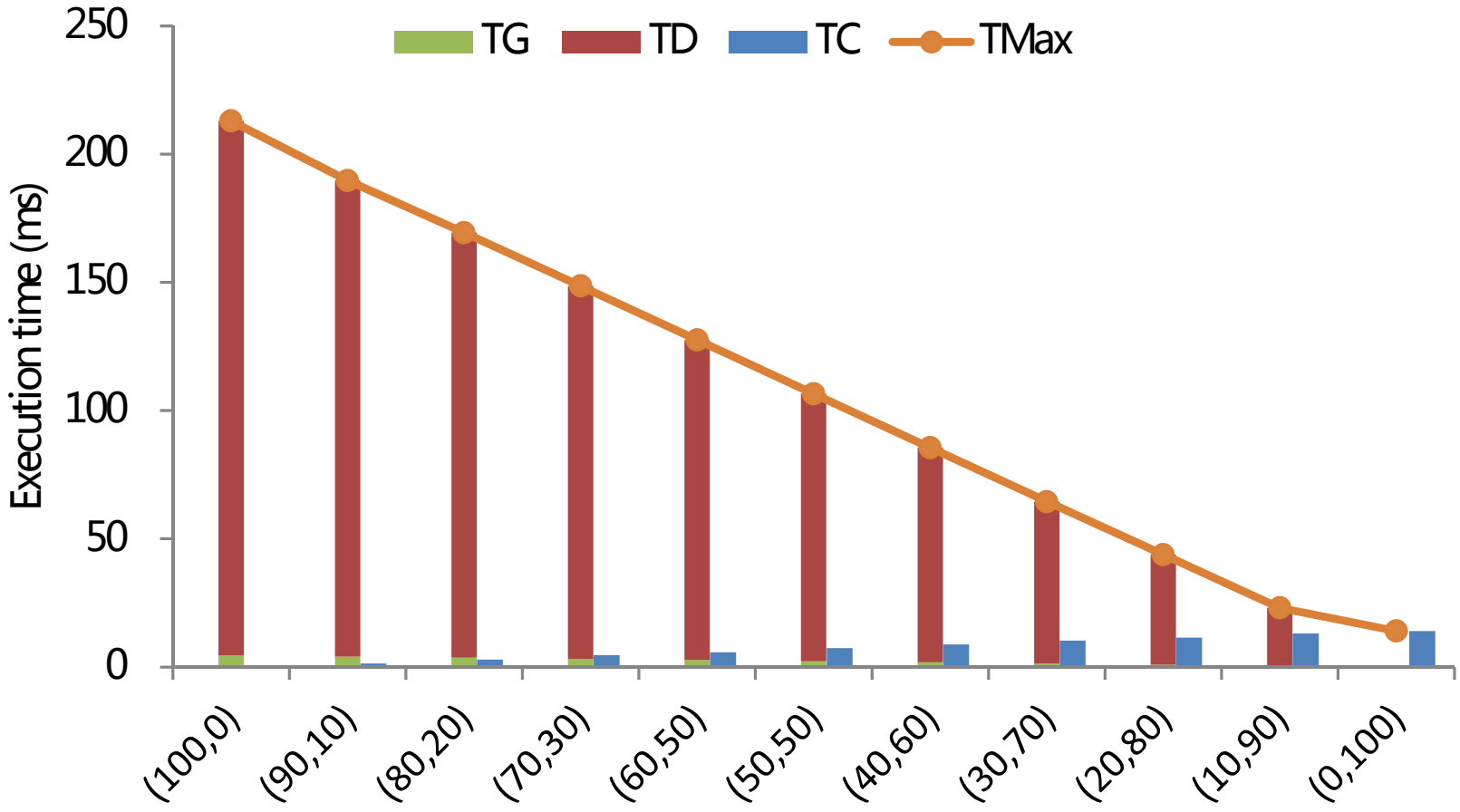
Example 1

- Dot product
  - Compute the dot product of 2 (1D) arrays

- GPU-friendly ?
- CPU-friendly?



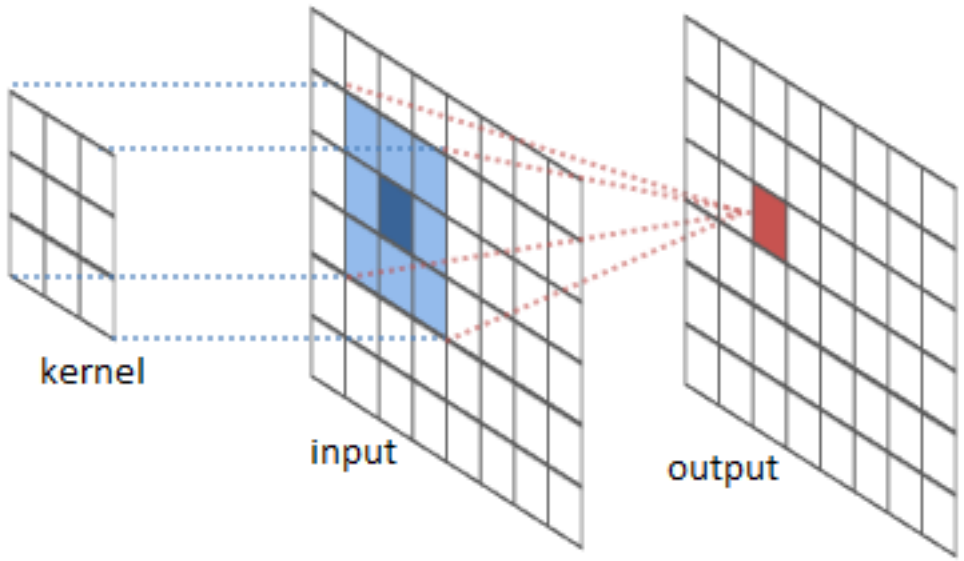
Example 1 : dot product



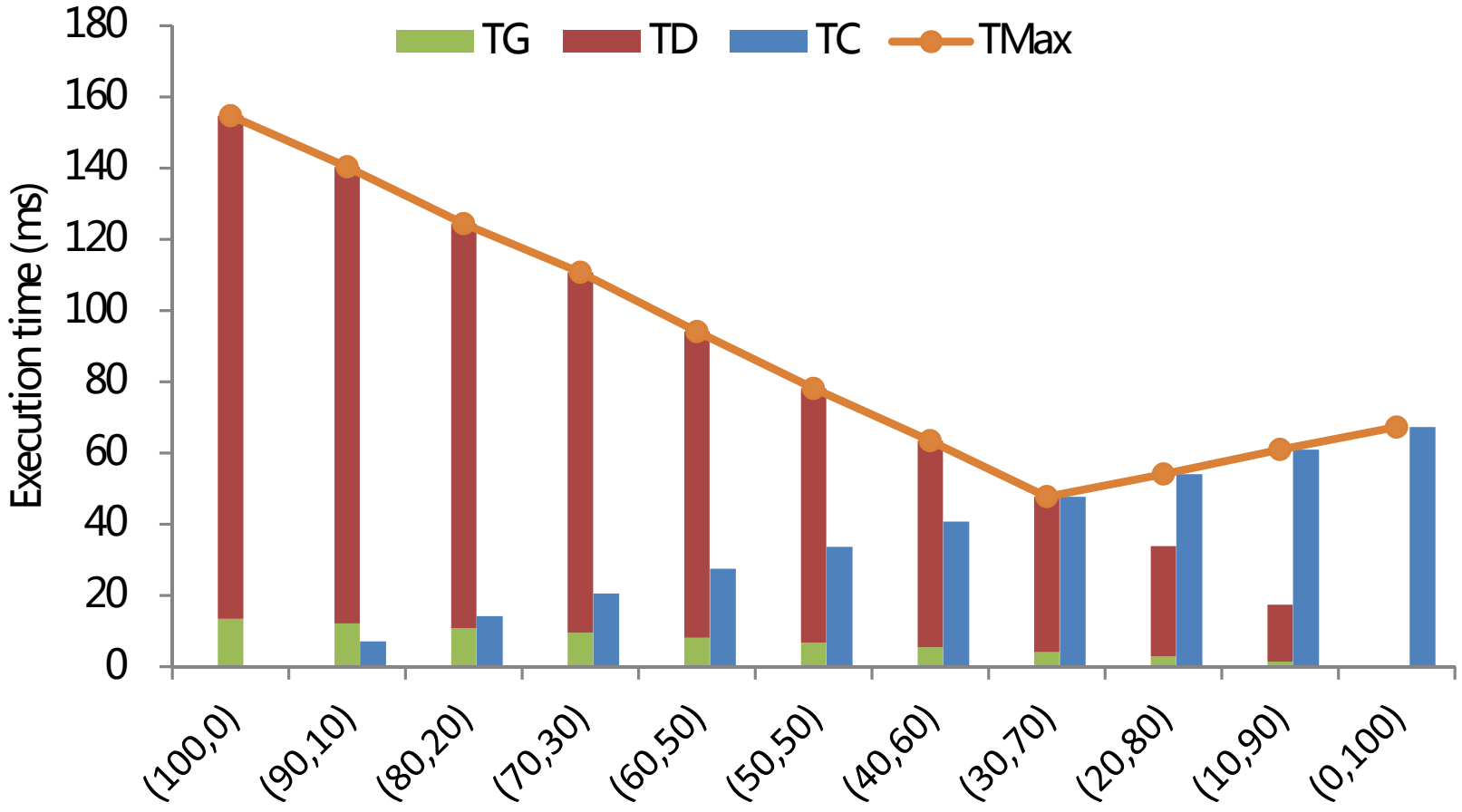
Example 2

- Separable convolution (CUDA SDK)
  - Apply a convolution filter (kernel) on a large image.
  - Separable kernel allows applying
    - Horizontal first
    - Vertical second

- GPU-friendly ?
- CPU-friendly?



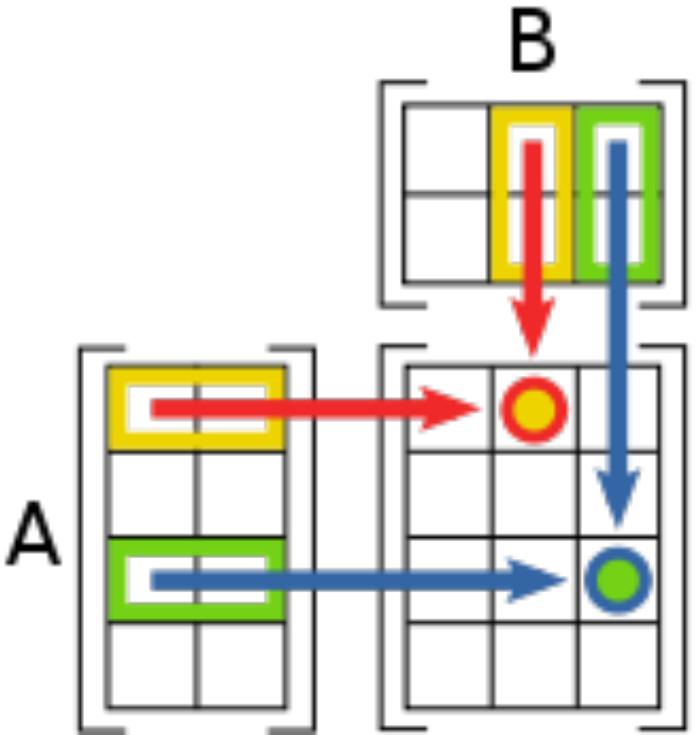
Example 2 : Separable convolution



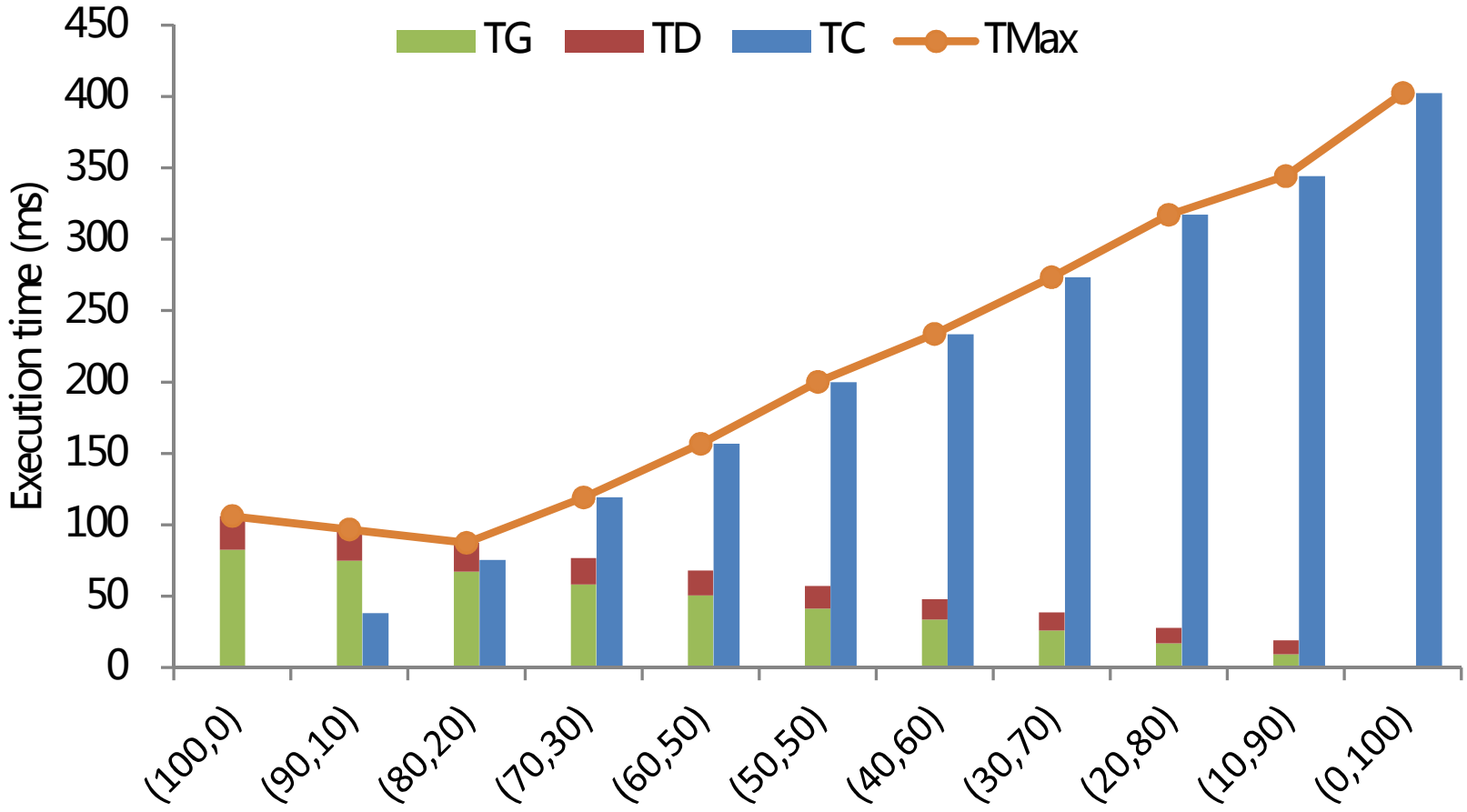
Example 3

- Matrix multiply
  - Multiply two dense matrices

- GPU-friendly ?
- CPU-friendly?



Example 3 : matrix multiply



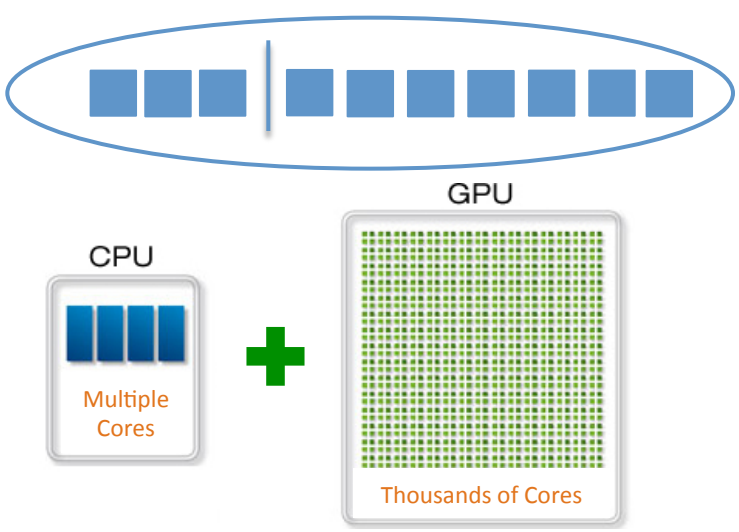
So …

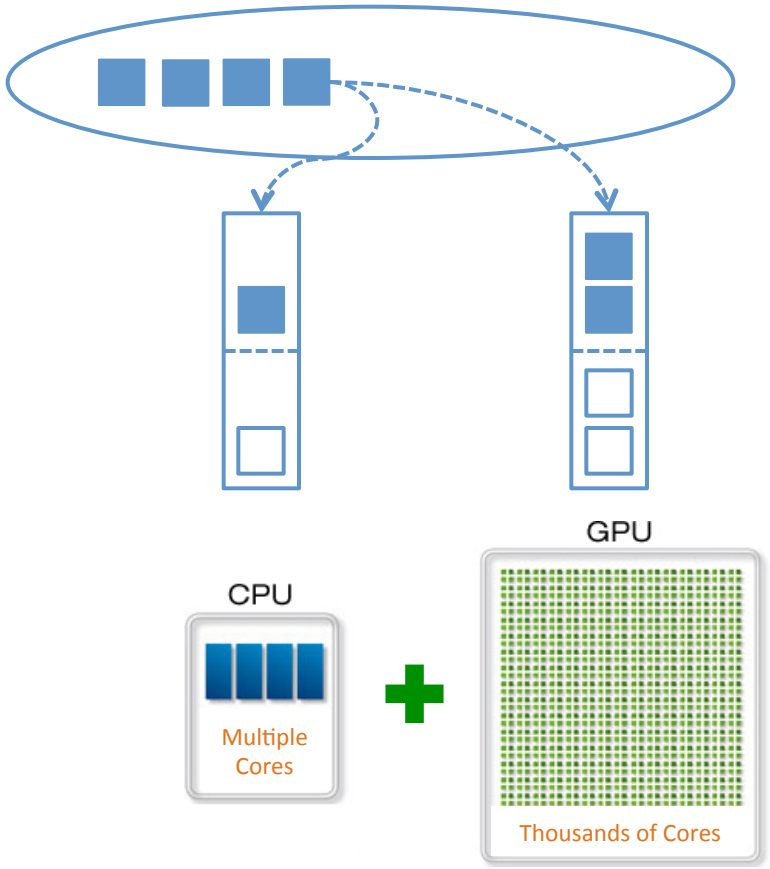
- There are **very** few GPU-only applications
  - The CPU – GPU communication bottleneck.
  - The increasing performance of CPUs
- A part of the computation can always be done by the CPU.
  - Which part?

Main challenge:
How to partition the application.

Determining the partition

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

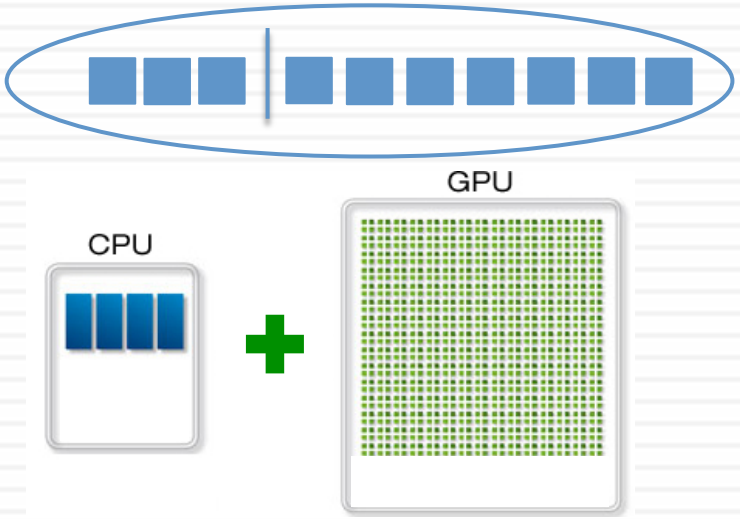




Static vs. dynamic

- 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

Static partitioning



Approaches to Static Partitioning

- Manual (or arbitrary) mapping (MM)
- Trial-and-error search (TE)
- Analytical modeling (AM)
- Learning-based methods (LM)

Comparison of different methods

	Accuracy (best performance)	Speed	Pre-cost	Adaptive to app changes	Adaptive to SW, HW changes
MM	++/--	++/--	n/a	N	N
TE	+++	---	n/a	Y	Y
AM	+/++	++/-	--/-	Y/N	Y/N
LM	+/++	+++	---	N	N

- Manual (or arbitrary) mapping (MM)
- Try-and-error search (TE)
- Analytical modeling (AM)
- Learning-based method (LM)

We first choose TE

	Accuracy (best performance)	Speed	Pre-cost	Adaptive to app changes	Adaptive to SW, HW changes
MM	++/--	++/--	n/a	N	N
TE	+++	---	n/a	Y	Y
AM	+/++	++/-	--/-	Y/N	Y/N
LM	+/++	+++	---	N	N

- Manual (or arbitrary) mapping (MM)
- Try-and-error search (TE)
- Analytical modeling (AM)
- Learning-based method (LM)

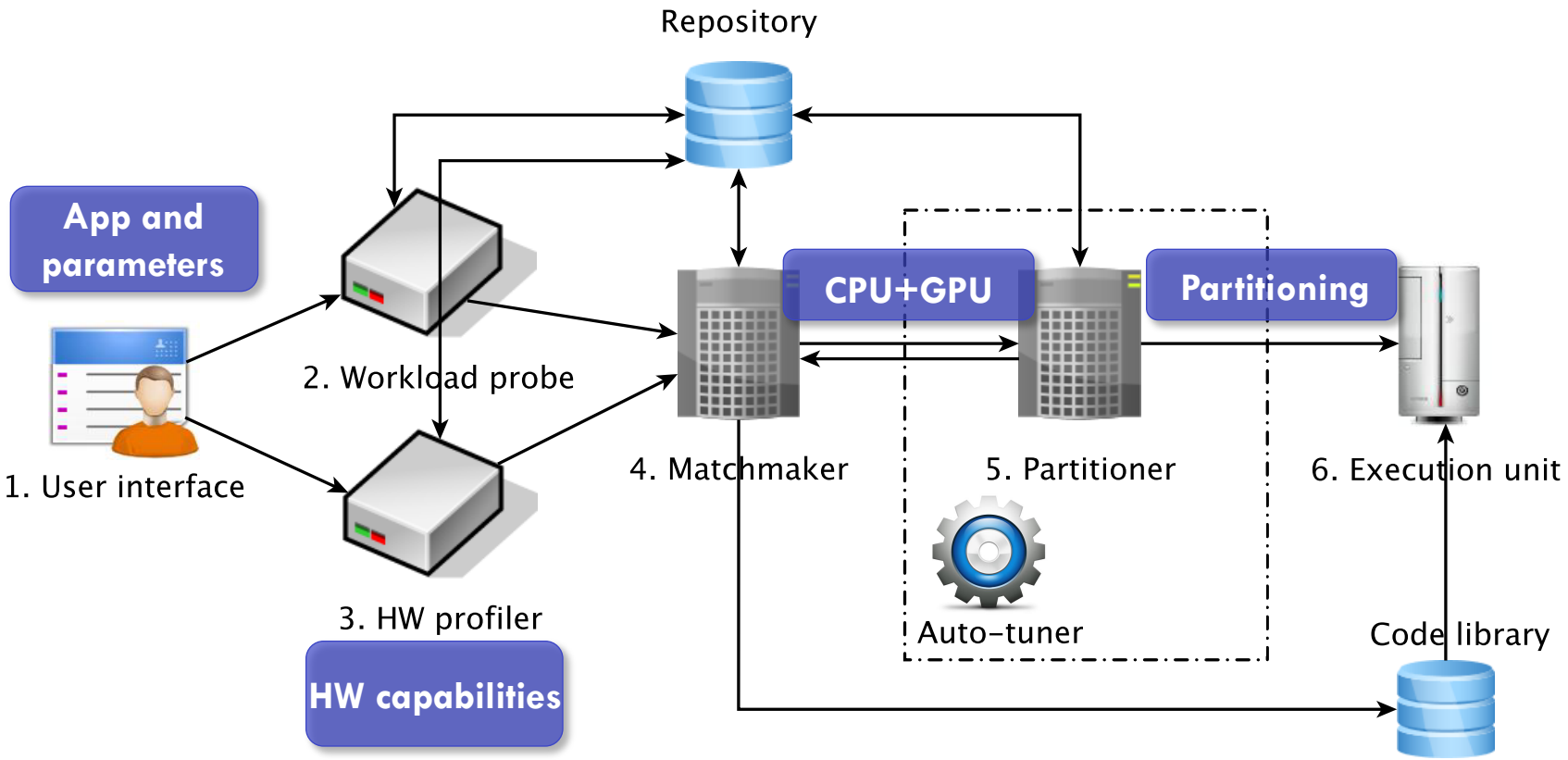
We build a workload partitioning framework based on TE (auto-tuning)

The Glinda framework

The Glinda framework

- A static-partitioning framework, based on auto-tuning
  - It targets massive data-parallel single-kernel applications
  - It aims to find the optimal partitioning, leading to the best performance

The Glinda Framework



What is wrong here?

Glinda's Pros and Cons

Pros	Cons
+ Finds the optimal partitioning, leading to the best performance	- Occupies the target platform to run the tuning
+ Utilizes the platform efficiently	- Time-consuming
+ Adapts to app and platform (SW, HW) changes	

Glinda's Pros and Cons

Pros	Cons
+ Finds the optimal partitioning, leading to the best performance	- Occupies the target platform to run the tuning
+ Utilizes the platform efficiently	- Time-consuming
+ Adapts to app and platform (SW, HW) changes	

Keep

Optimize

How can we do that?

Optimizing the partitioning process

We add AM into Glinda

	Accuracy (best performance)	Speed	Pre-cost	Adaptive to app changes	Adaptive to SW, HW changes
MM	++/--	++/--	n/a	N	N
TE	+++	---	n/a	Y	Y
AM	+/++	++/-	--/-	Y/N	Y/N
LM	+/++	+++	---	N	N

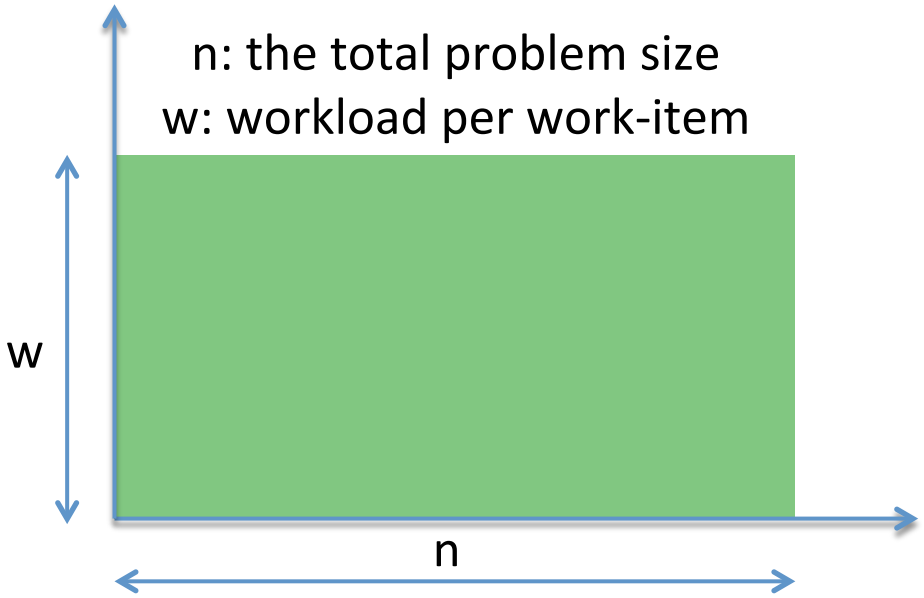
We want to optimize the partitioning process by predicting the partitioning

Our method*

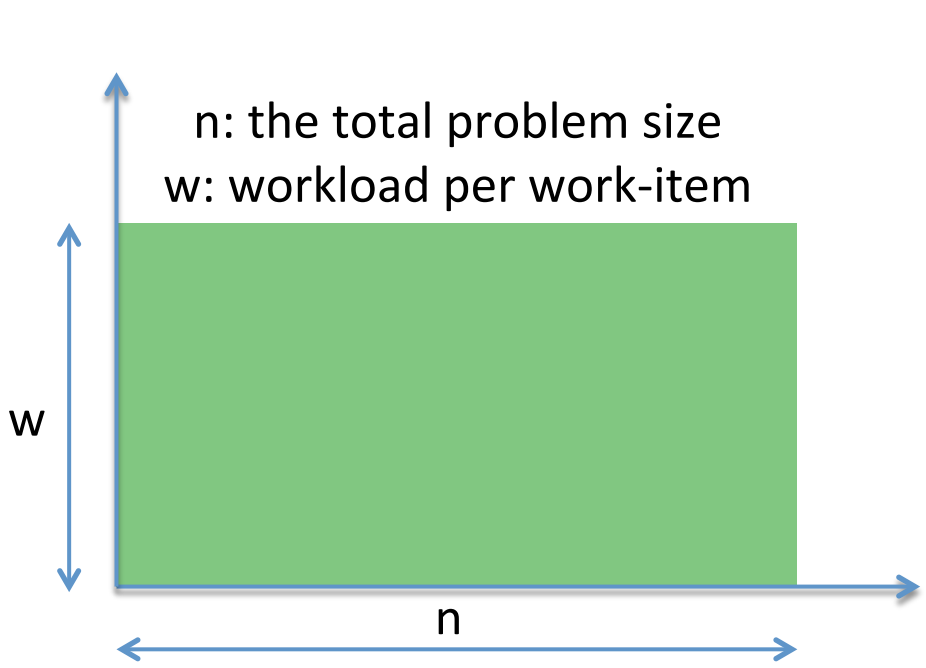
- Model (quantify)
  - the app workload,
  - the HW capabilities,
  - the GPU-CPU data-transfer.
- Build the partitioning model
  - An equation to solve.
- Predict the partitioning

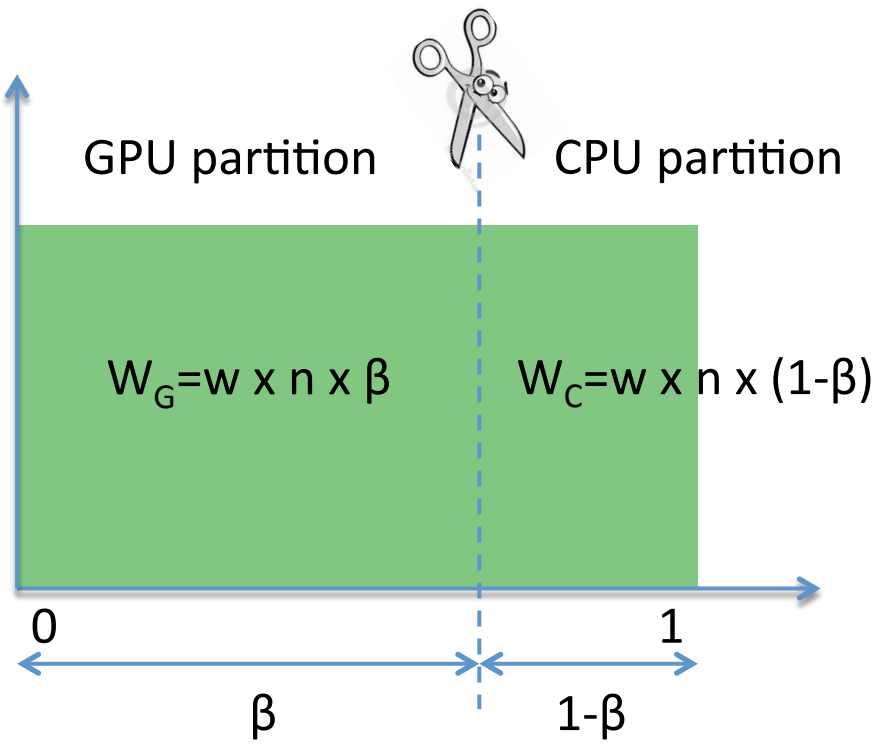
*Jie Shen et al., HPCC'14. "Look before you Leap: Using the Right Hardware. Resources to Accelerate Applications

Model the app workload



Model the app workload





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

Model the HW capabilities

- P (processing throughput)
  - Measured as workload processed per second
  - P evaluates the hardware capability of a processor

Model the HW capabilities

- 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 CPU-GPU data-transfer

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

Data-transfer time: $T_D = O/Q + (\text{Latency})$

Latency < 0.1 ms, negligible impact

Model the CPU-GPU data-transfer

- Three types of data transfers
  - ND: O=zero (no data transfer/data-transfer overhead can be ignored)
  - PD: O=β x Full data-transfer size (partial data transfer)
  - FD: O is independent of β (full data transfer)

The data-transfer type specifies whether to add T_D or not

Build the partitioning model

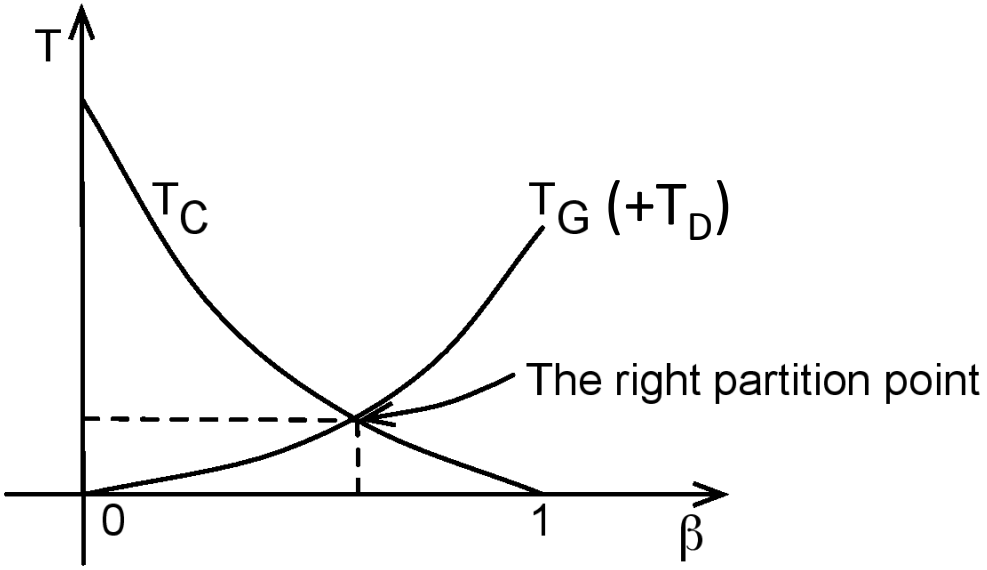
- Define the optimal partitioning

ND

$$T_G = T_C$$

PD, FD

$$T_G + T_D = T_C$$



Build the partitioning model

- Substitute the quantities

ND

$$T_G = T_C$$

$$\frac{W_G}{P_G} = \frac{W_C}{P_C}$$

$$\frac{W_G}{W_C} = \boxed{\frac{P_G}{P_C}}$$

The relative HW capability

PD, FD

$$T_G + T_D = T_C$$

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

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

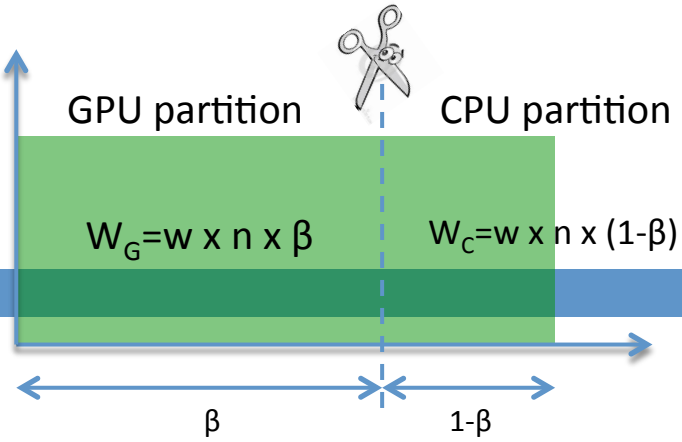
The impact of data transfer

Predict the partitioning

- Solve the equation

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

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

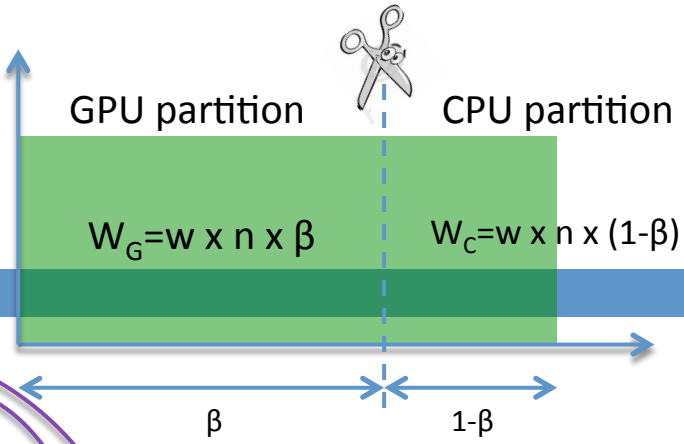
Expression

W_G=w x n x β

W_C=w x n x (1-β)

O=Full data transfer or
Full data transfer x 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

β-independent terms

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

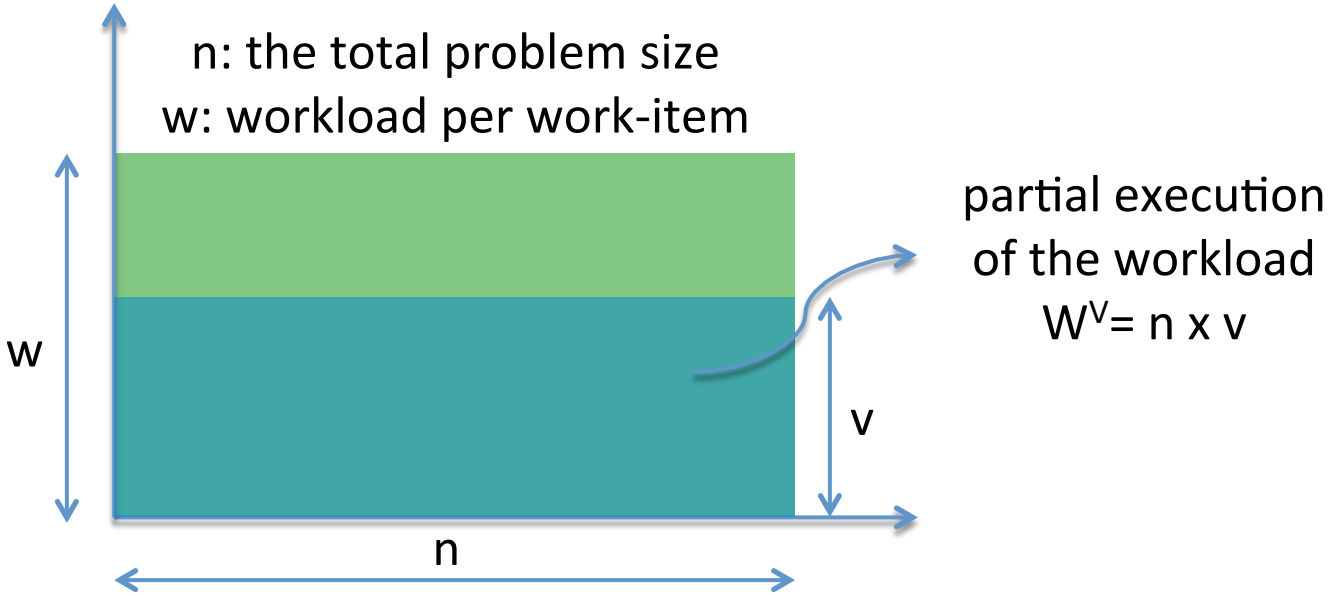
Predict the partitioning

- Estimate the β-independent terms $\frac{P_G}{P_C}$ $\frac{P_G}{Q}$
  - Profile HW in the (application, problem size) context

1. Provides a more realistic HW capabilities estimation
2. Respond quickly to application and platform (SW and HW) changes

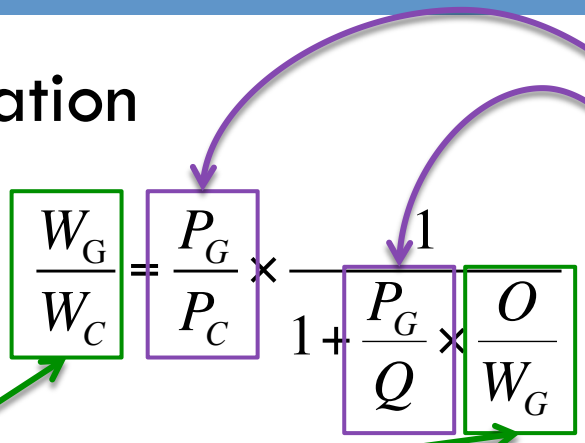
Predict the partitioning

- Estimate the β-independent terms $\frac{P_G}{P_C}$ $\frac{P_G}{Q}$
  - Profile HW in the (application, problem size) context
  - For iterative-based applications => partial profiling



Predict the partitioning

- Solve the equation



β-dependent terms

β-independent terms

Expression

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

Estimation

$$\frac{P_G}{P_C} = \frac{W^V / T_G^V}{W^V / T_C^V} = \frac{T_C^V}{T_G^V} = R_{GC}$$

$$\frac{P_G}{Q} = \frac{W^V / T_G^V}{O^V / T_D^V} = \frac{W^V}{O^V} \times \frac{T_D^V}{T_G^V} = \frac{W^V}{O^V} \times R_{GD}$$

Predict the partitioning

- Solve β from the equation

ND

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

$$O = O^V = 0$$

PD

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

$$O = O^V \times \beta$$

FD

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

$$O = O^V \neq 0$$

Glinda's Pros and Cons

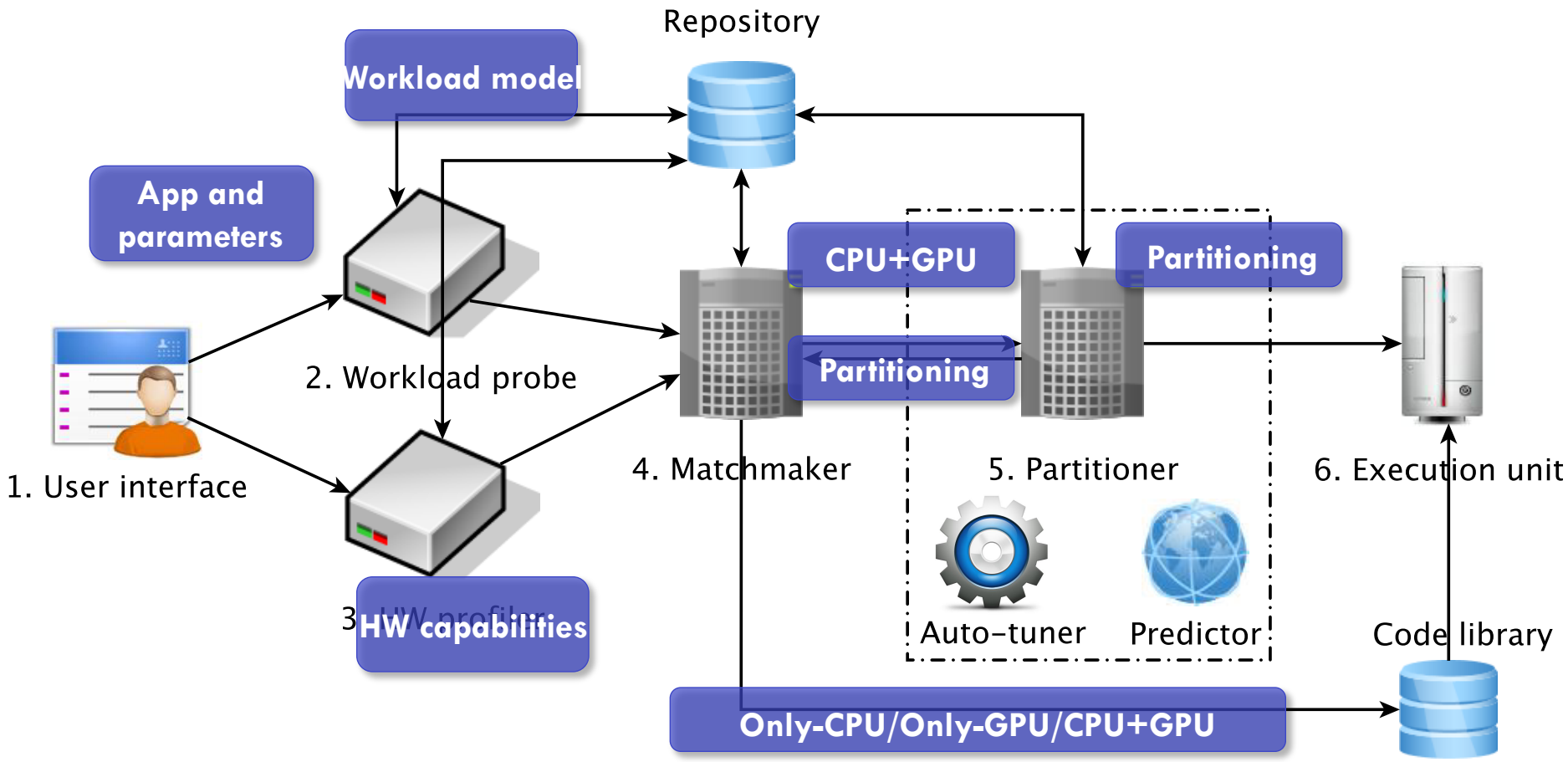
Pros	Cons
- Find the optimal partitioning, leading to the best perfo Case study	- Occupy the target platform to run the tuning ✓
- Utilize the platform efficiently ✓	- Time-consuming ✓
- Adapt to app and platform (SW, HW) changes ✓	

Keep

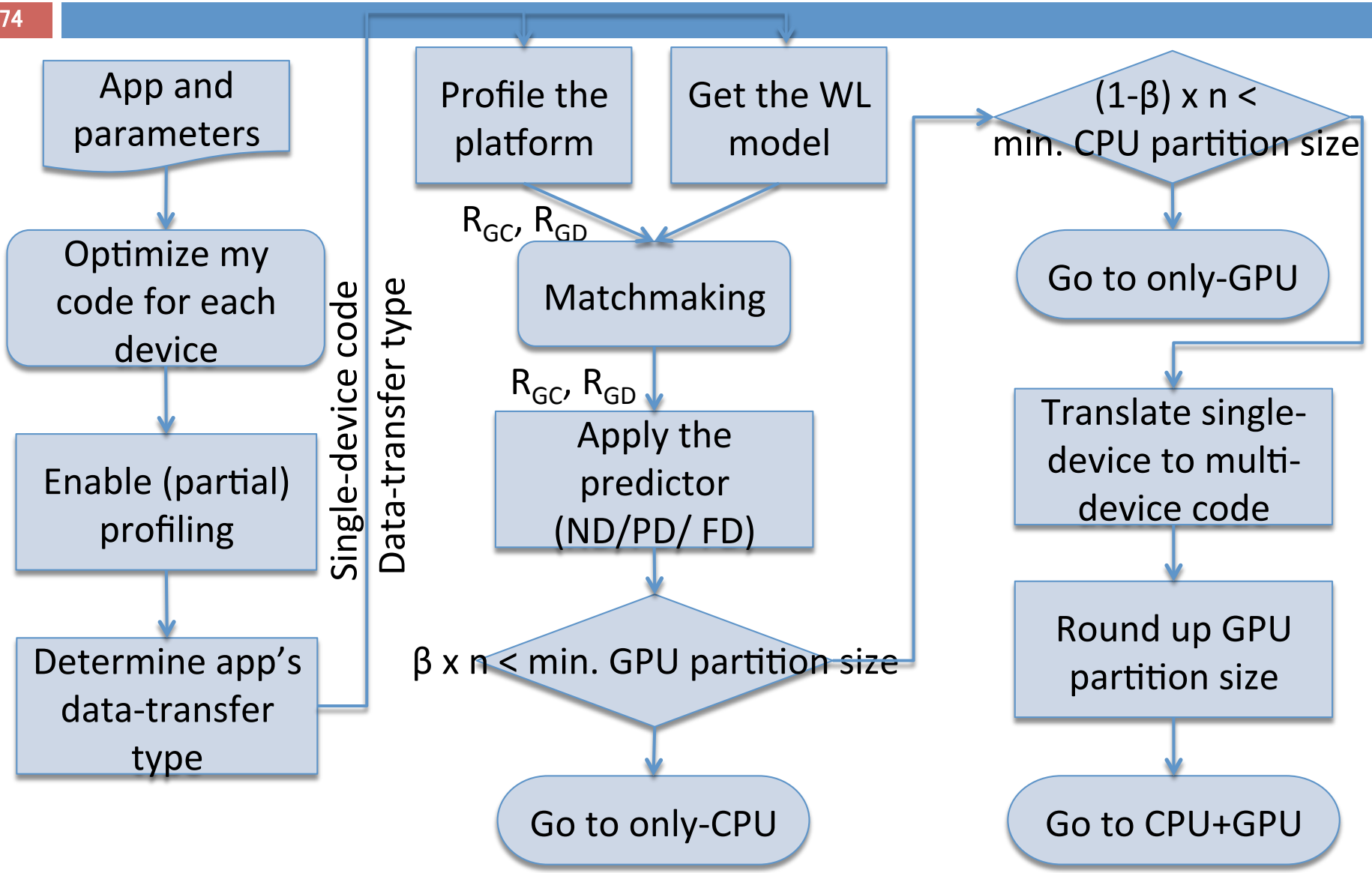
Optimized

What is still wrong here?

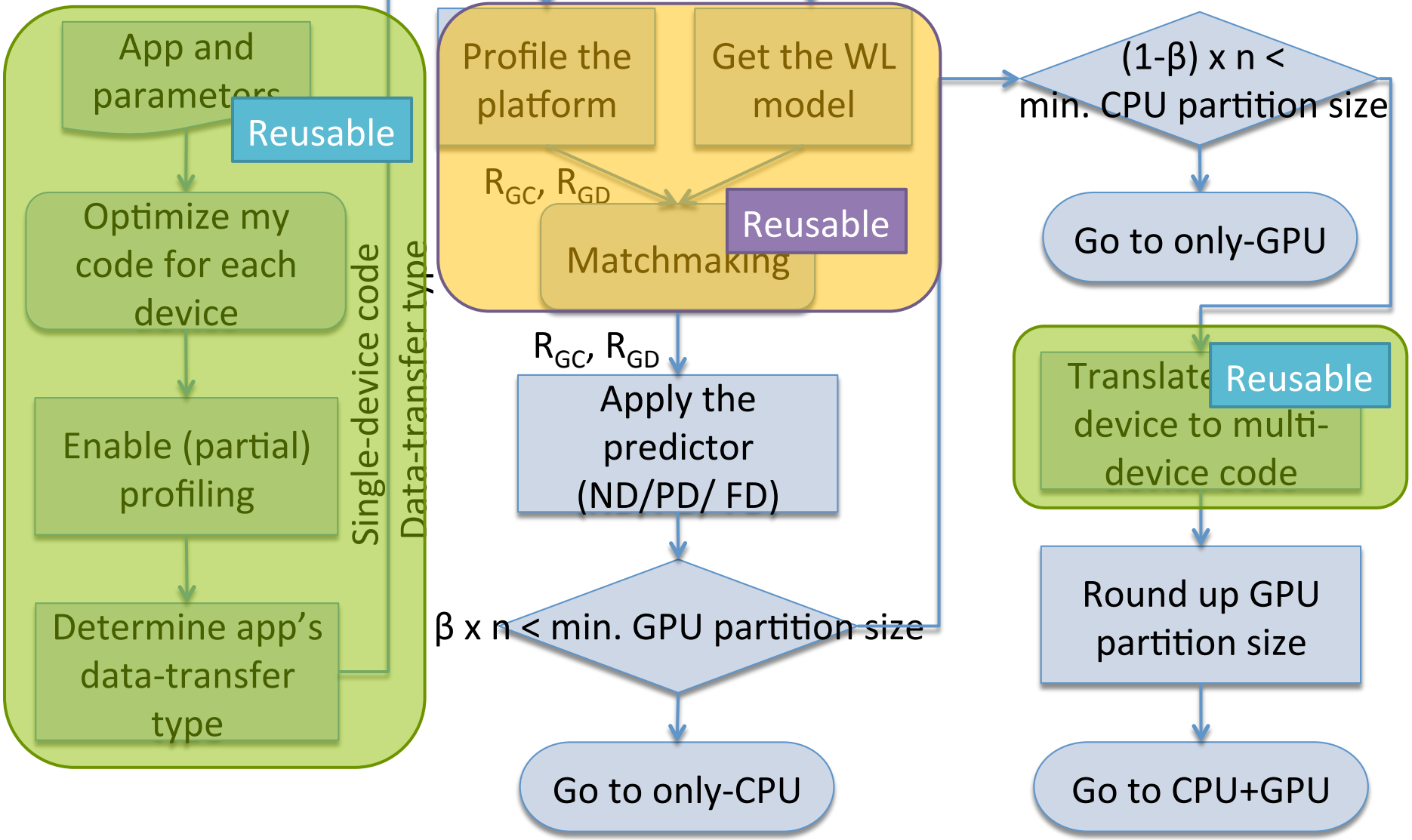
The Glinda Framework



Glinda step by step



Glinda step by step



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

Results

Applications and datasets

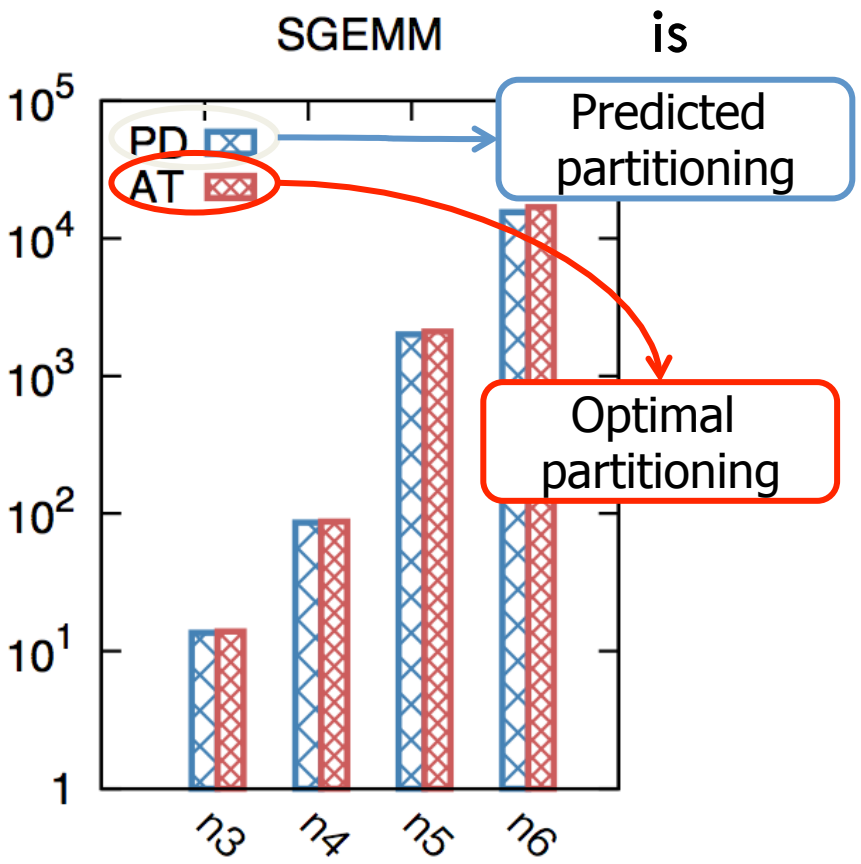
Application	Domain	Type
Vector Addition	Linear algebra	Memory-bound
Dot Product	Linear algebra	Memory-bound
Black-Scholes	Finance	Memory-bound
Mersenne Twister	Random number generator	Memory-bound
Separable Convolution	Image processing	Memory-bound
Matrix Multiplication	Linear algebra	Compute-intensive
NBody	Scientific simulation	Compute-intensive

Problem size: $n1 <$ CPU L1 & GPU L1, $n2 <$ CPU L2 & GPU L2,
$n3 <$ CPU L3, $n4 <$ 100 MB, $n5 <$ 1 GB, $n6 <$ GPU Memory.

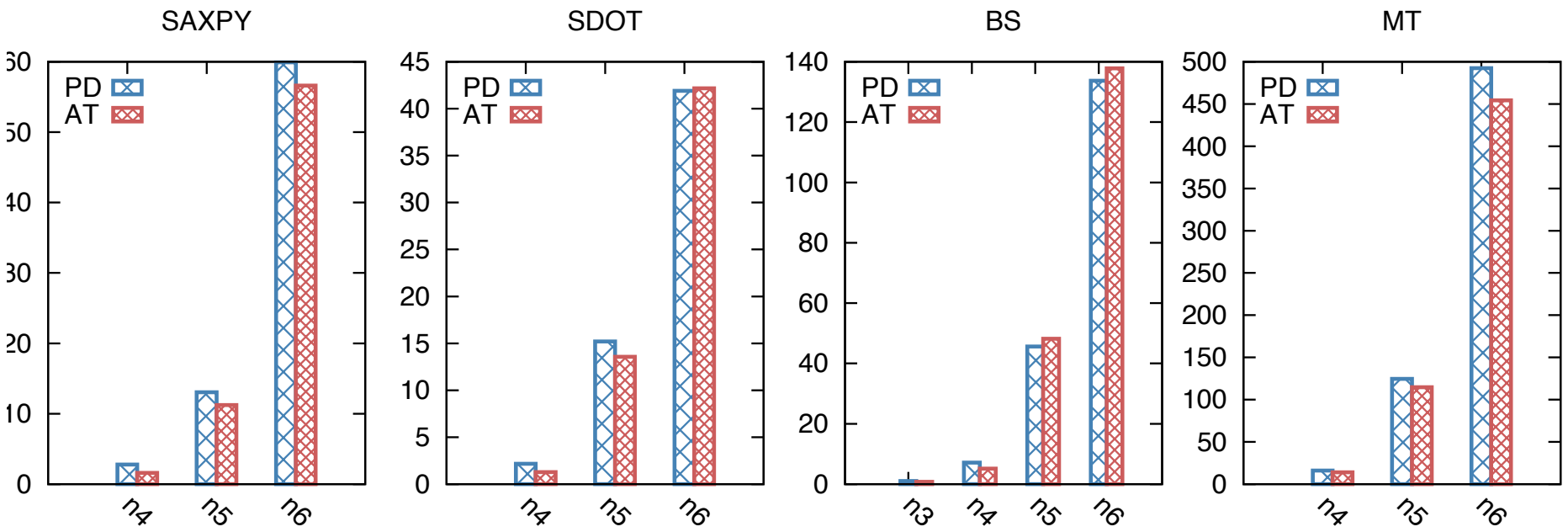
Experimental evaluation

- Quality (compared to an oracle)
  - The right HW configuration in 38 out of 42 test cases

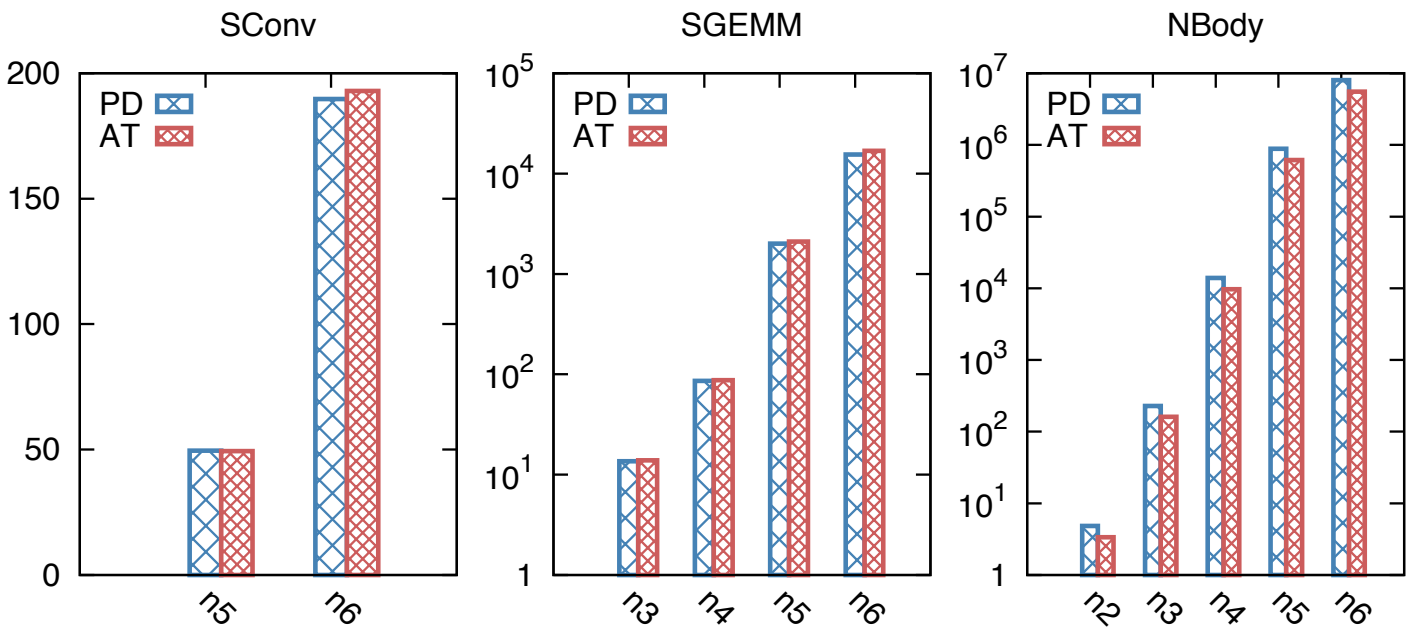
	Application	SGEMM
$n1$	R_{GC}, R_{GD}	6.04, 26.54
	β	0.1799
	Decision	Only-CPU
$n2$	R_{GC}, R_{GD}	4.78, 5.07
	β	0.4406
	Decision	Only-CPU
$n3$	R_{GC}, R_{GD}	4.15, 0.70
	β	0.7090
	Decision	CPU+GPU
$n4$	R_{GC}, R_{GD}	4.88, 0.29
	β	0.7916
	Decision	CPU+GPU
$n5$	R_{GC}, R_{GD}	5.02, 0.09
	β	0.8220
	Decision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



Results: accuracy

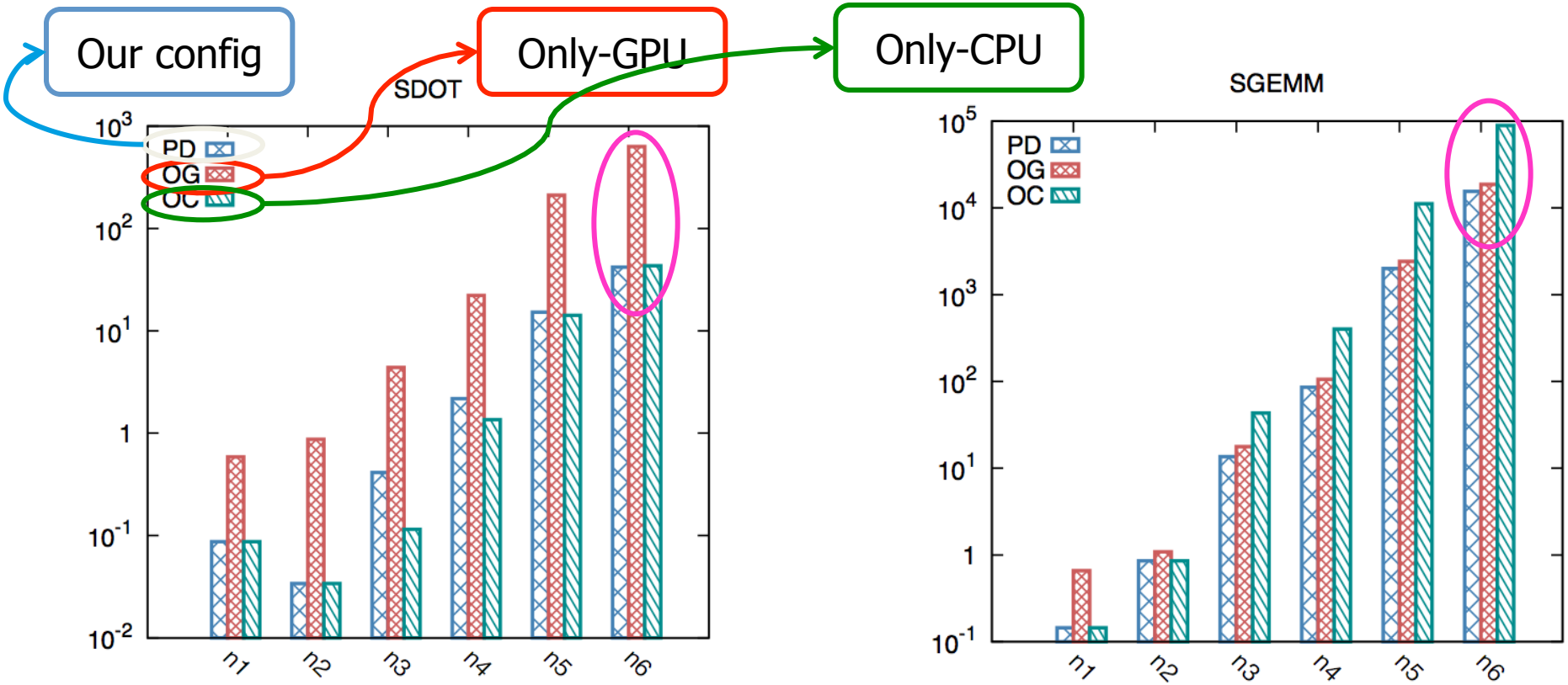


Results: accuracy

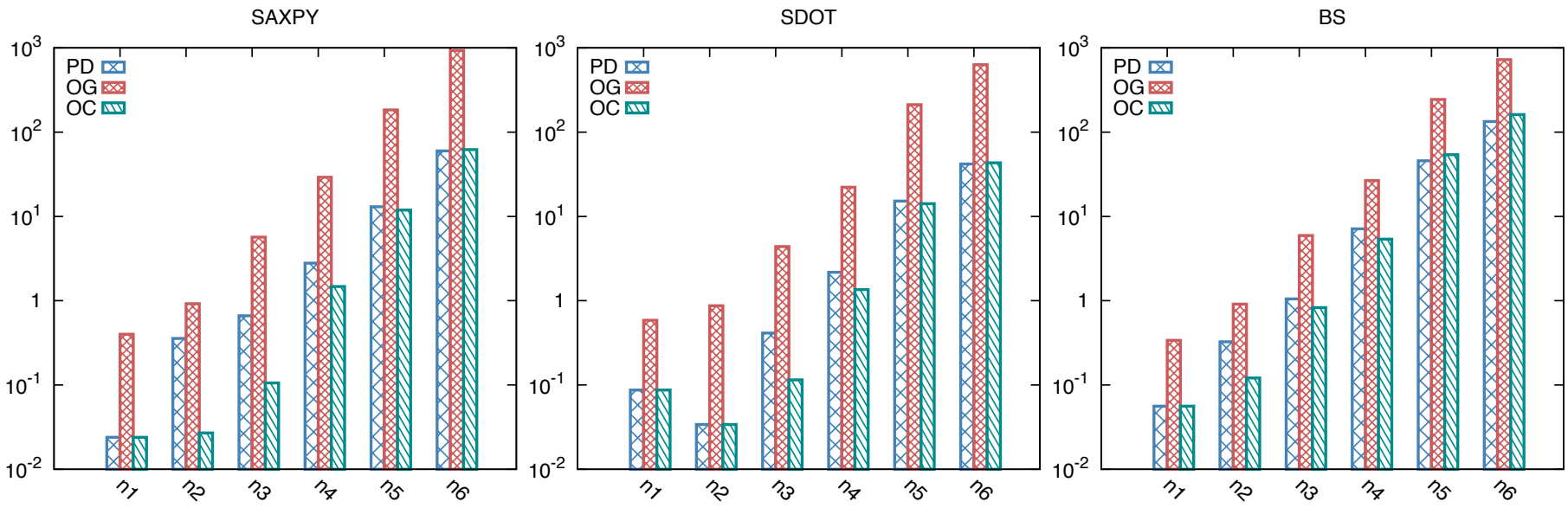


Experimental evaluation

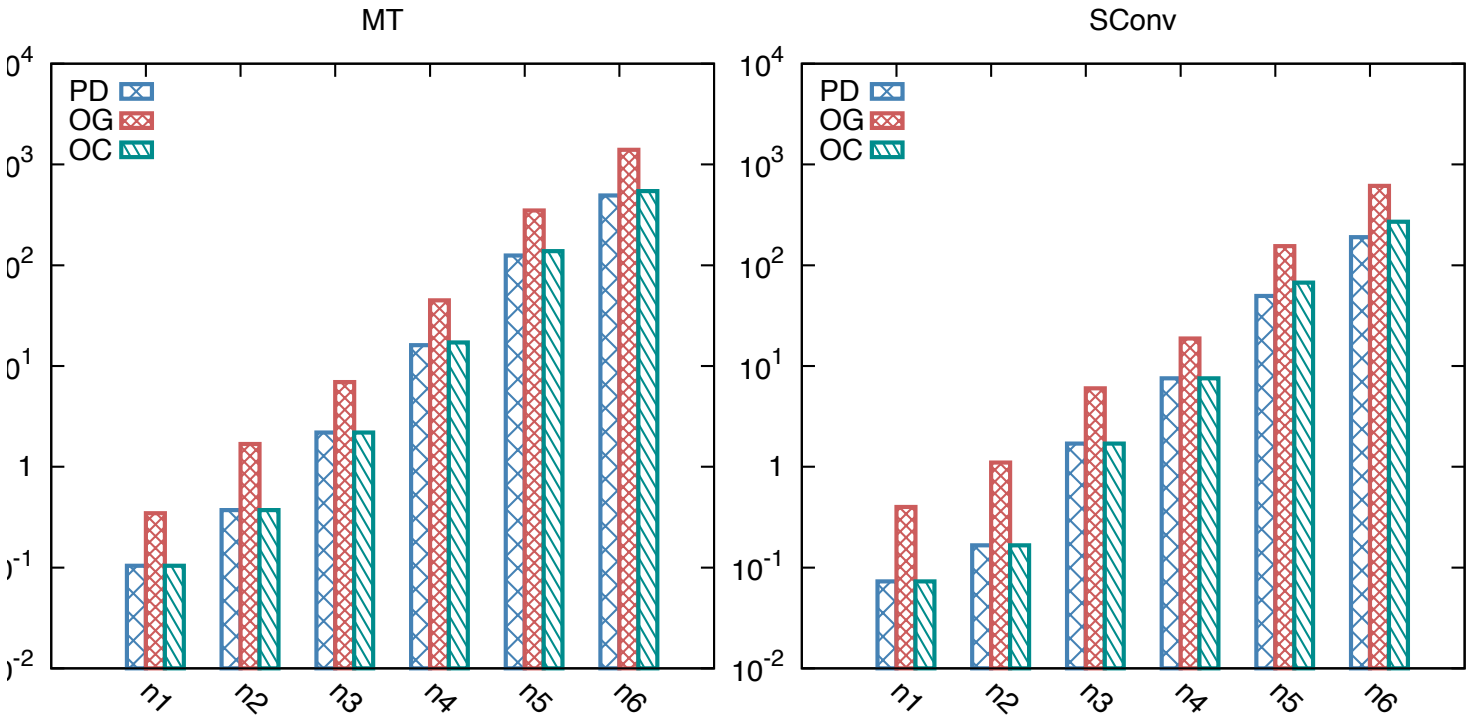
- Effectiveness (compared to Only-CPU/Only-GPU)
  - Up to 12.6x/6.6x speedup
  - Only GPU: up to 96% performance will be lost



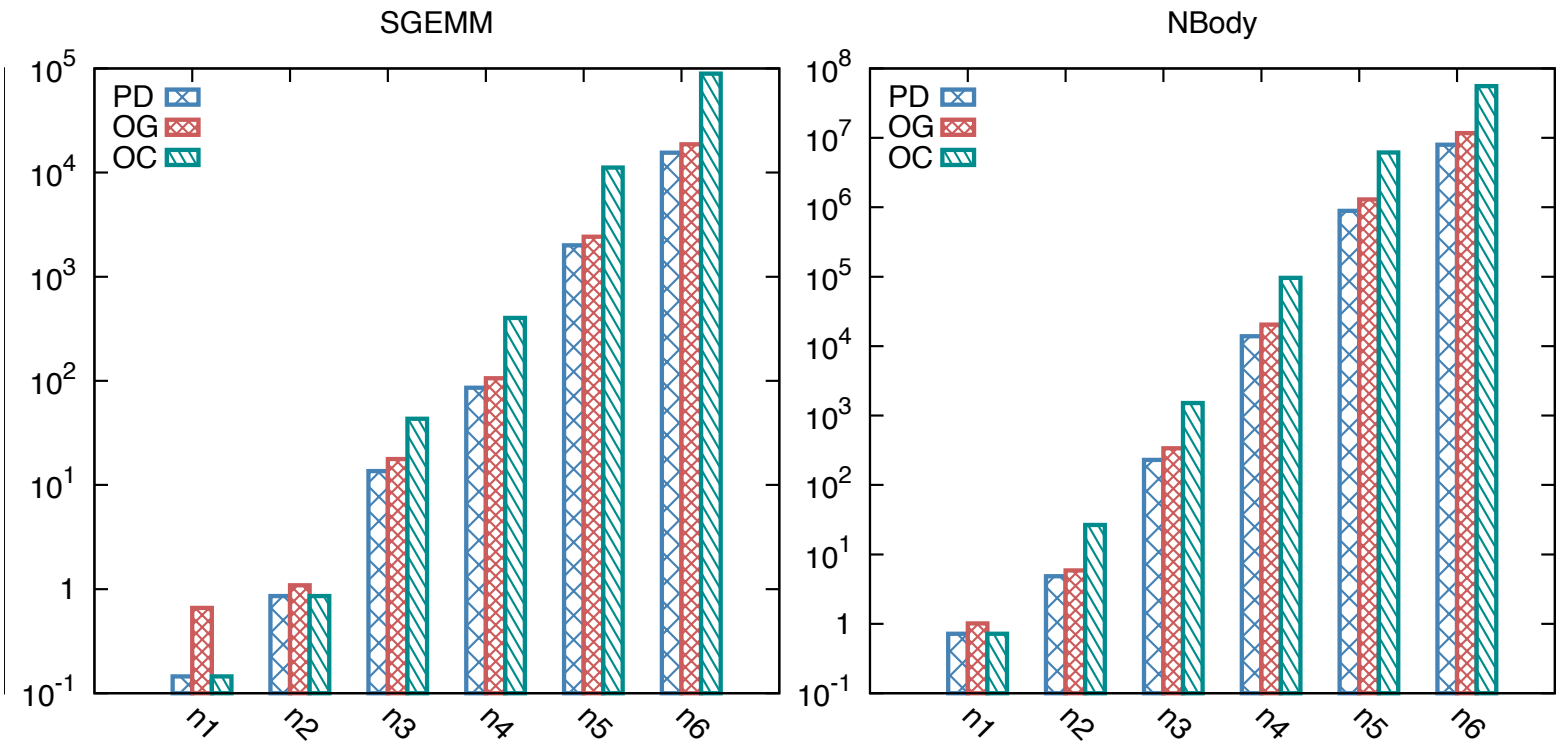
Results: performance



Results: performance



Results: performance



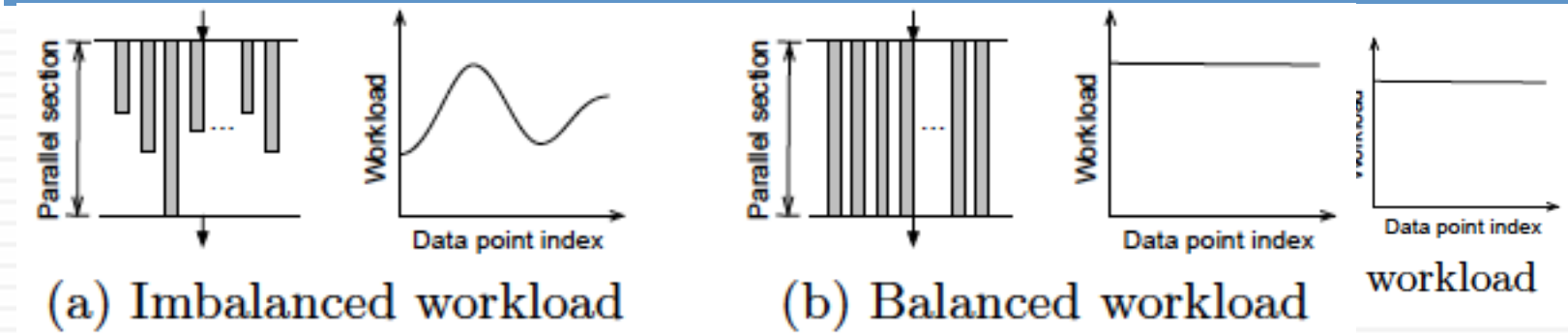
Statistics break!

- Please raise your hand if you are affiliated with (== work at) an institution in:
  - Netherlands
  - Romania
  - Belgium
  - Spain
  - France
  - Germany
  - Italy
  - UK
  - USA
  - China
  - Another country ?

Any more …

- Questions

BONUS: Imbalanced applications



(a) Imbalanced workload

(b) Balanced workload

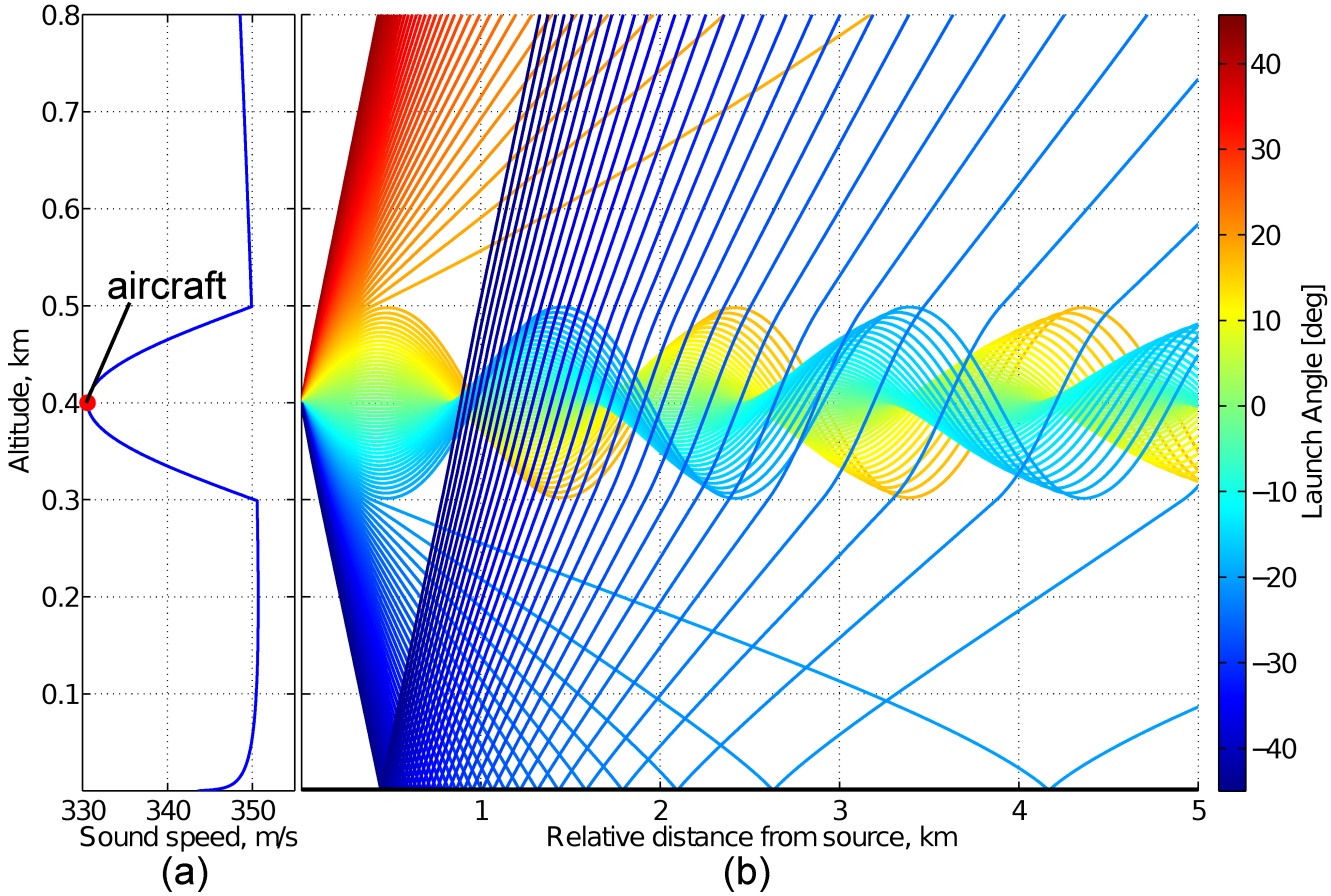
workload

Sound ray tracing

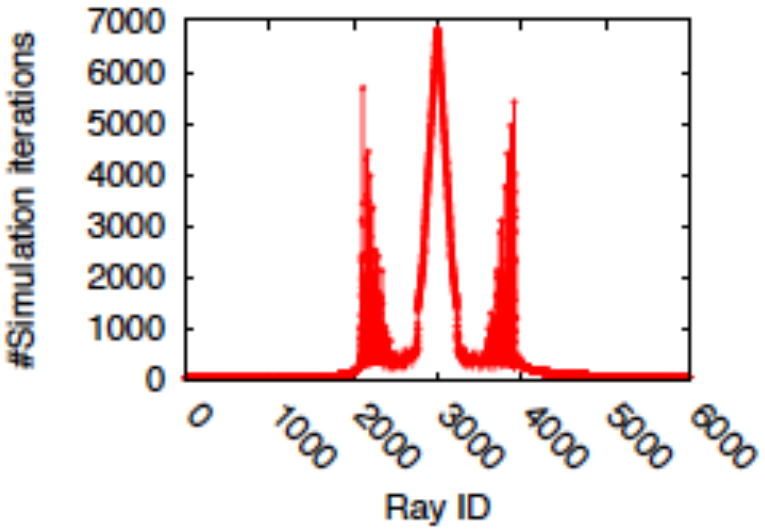
- A collaboration with Dutch NLR
- Simulate the sound propagation
  - from an aircraft to receivers
- Assess aircraft flyover noise during the aircraft take-off and approach procedures

Sound ray tracing

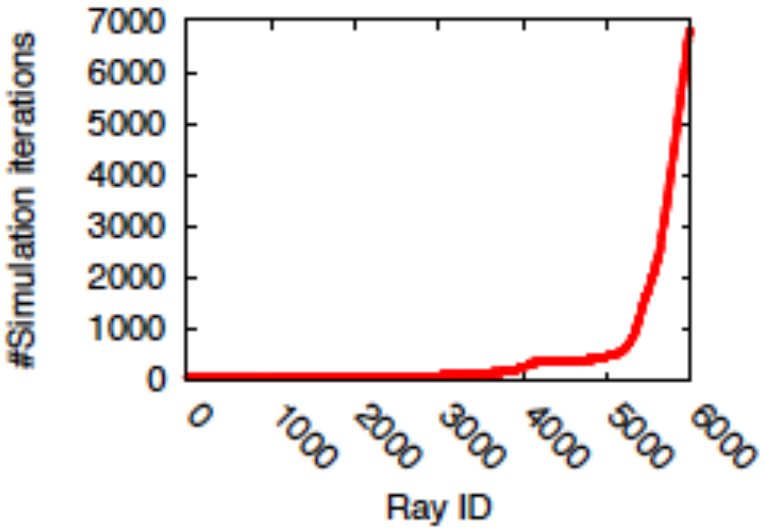
Sound ray tracing



Workload profile

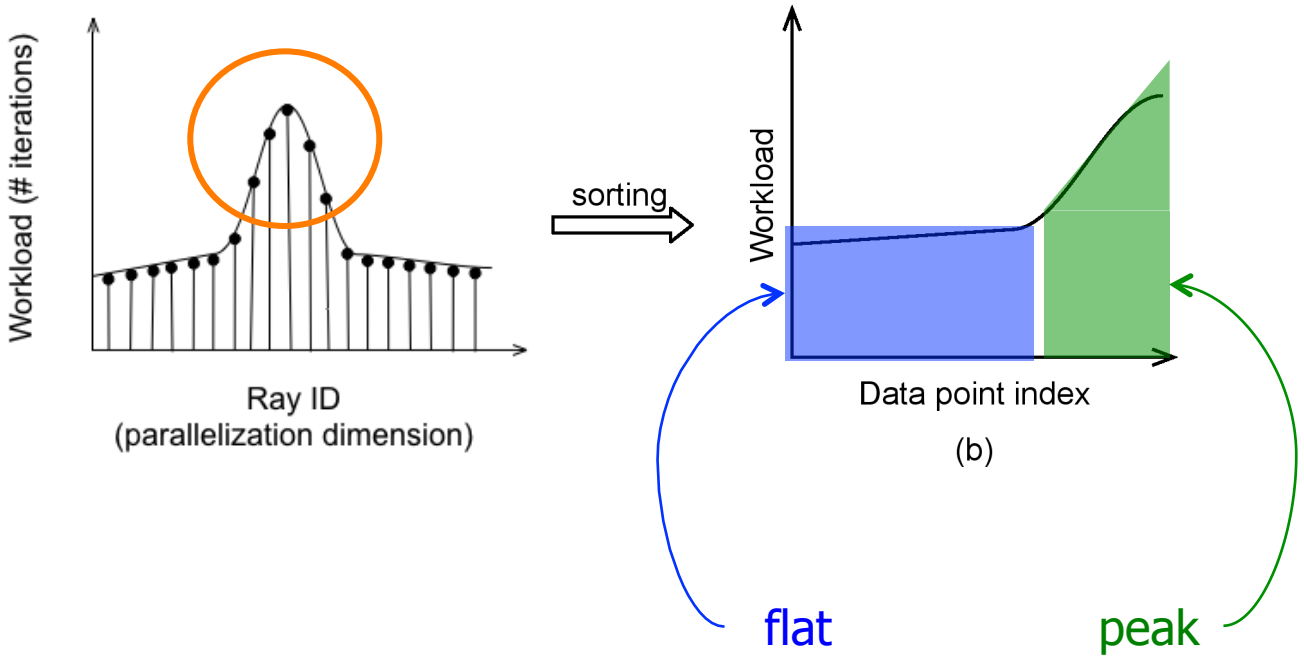


(a) the original workload

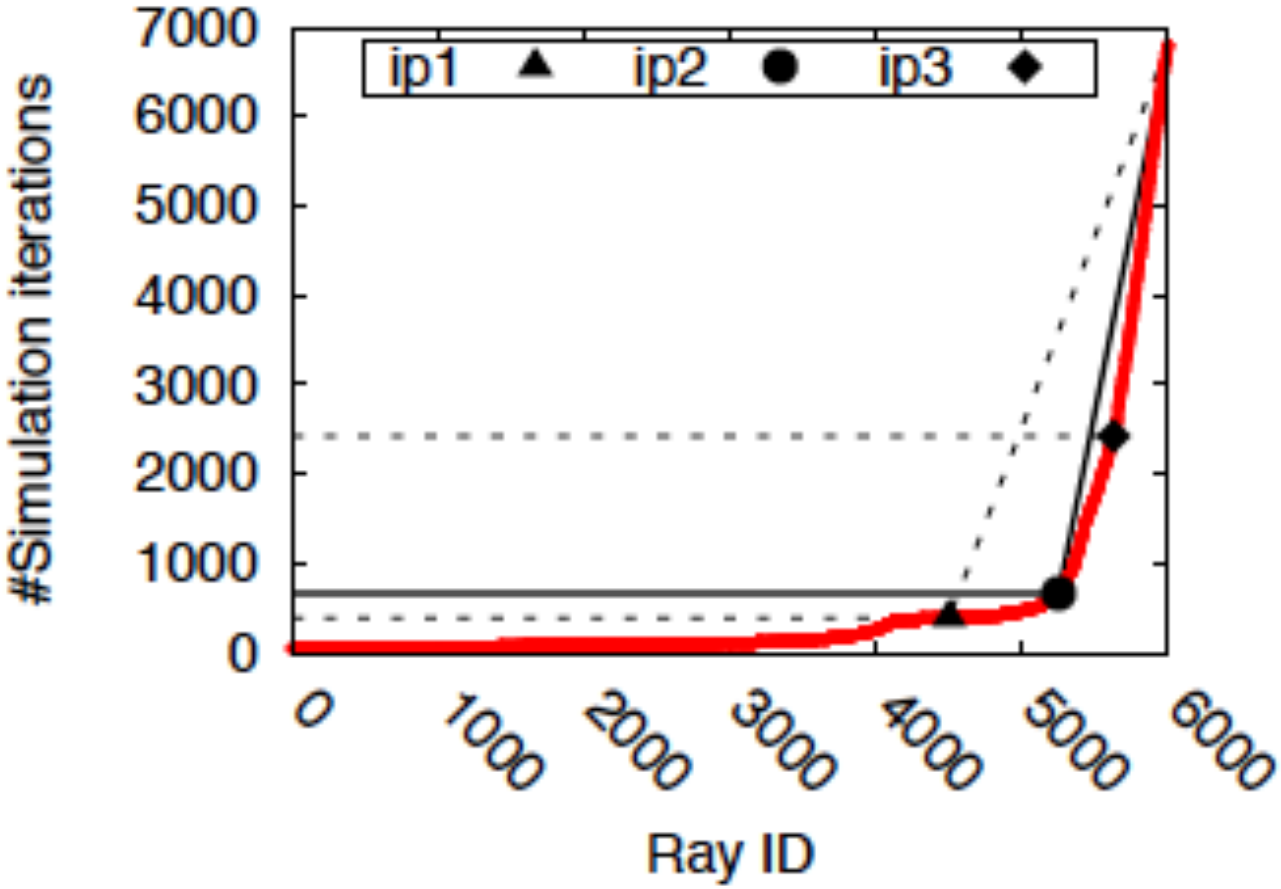


(b) after sorting

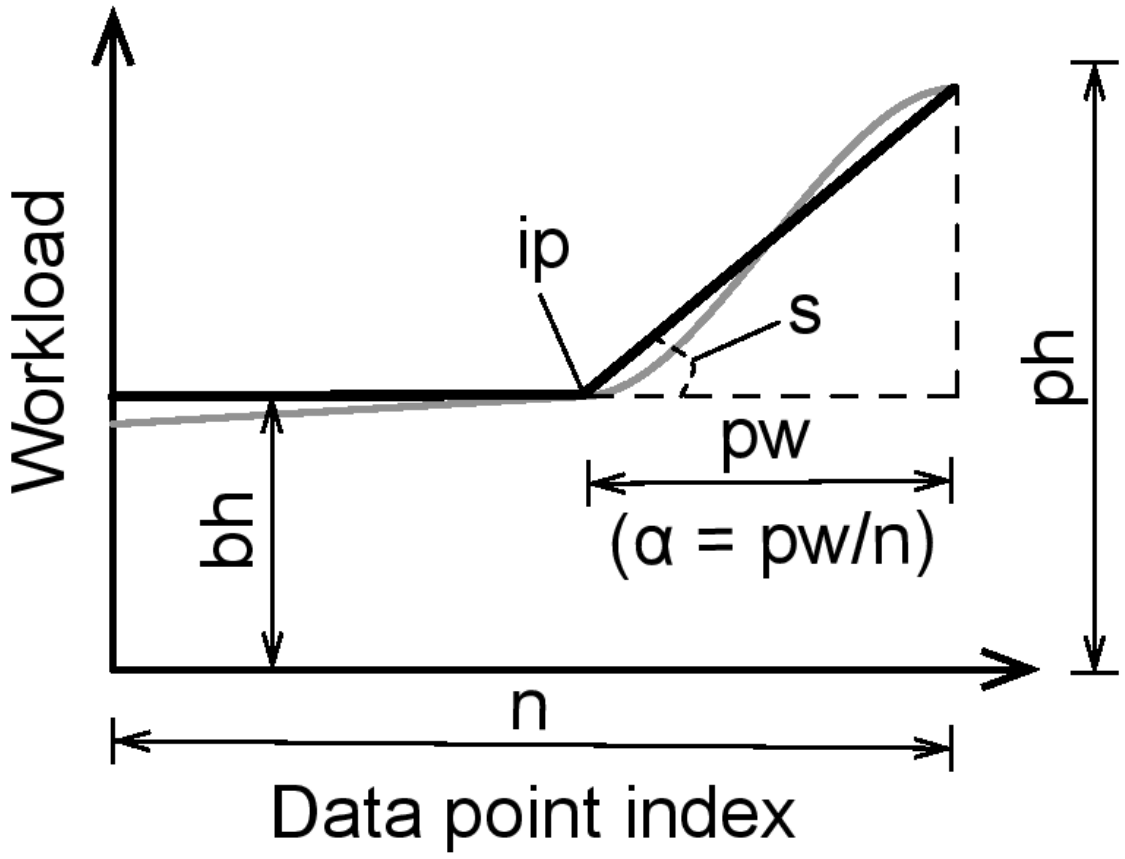
Model


Workload (# iterations)
Ray ID
(parallelization dimension)
sorting
Workload
Data point index
(b)
flat
peak

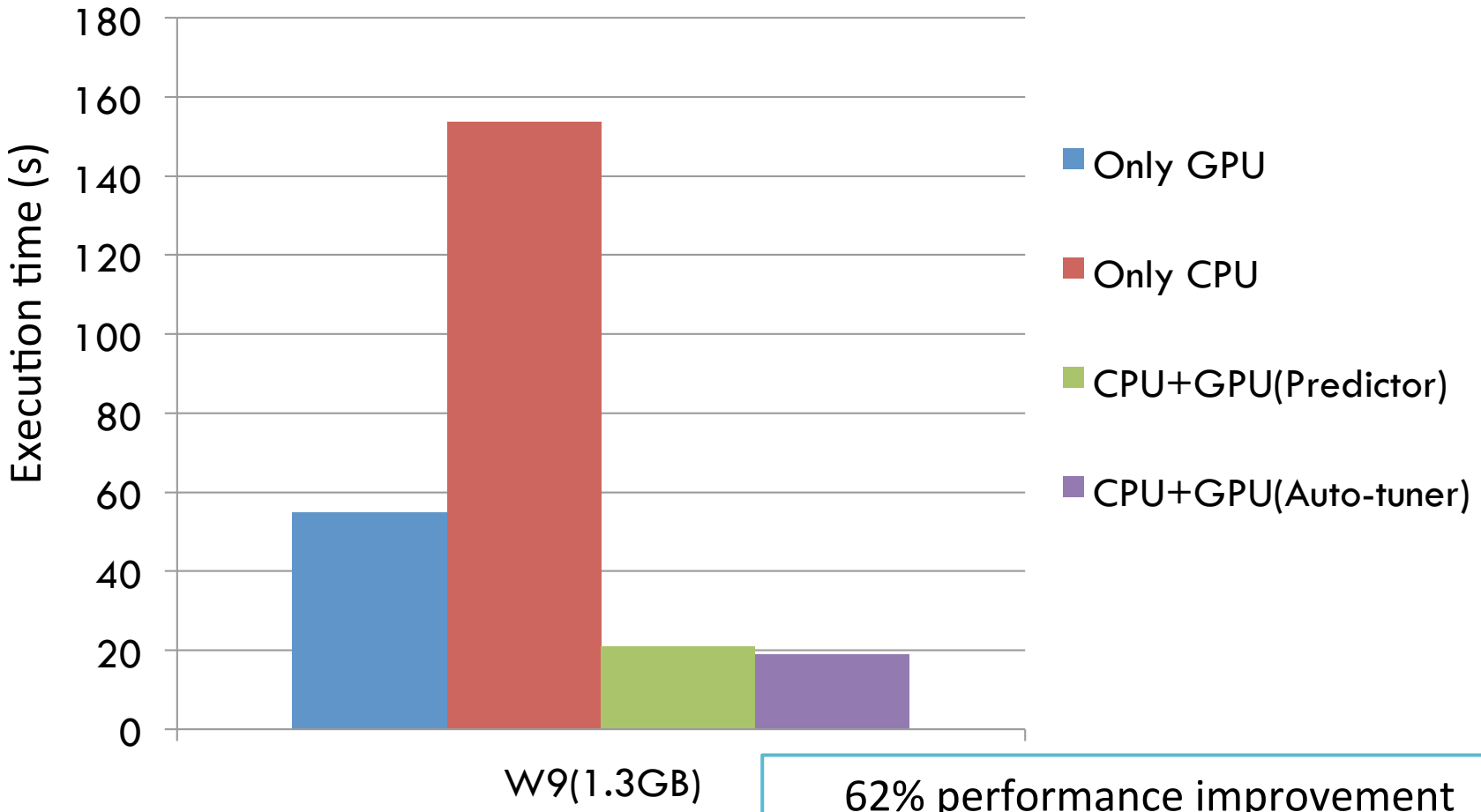
Modeling


#Simulation iterations
7000
6000
5000
4000
3000
2000
1000
0
ip1
ip2
ip3
0
1000
2000
3000
4000
5000
6000
Ray ID

Model



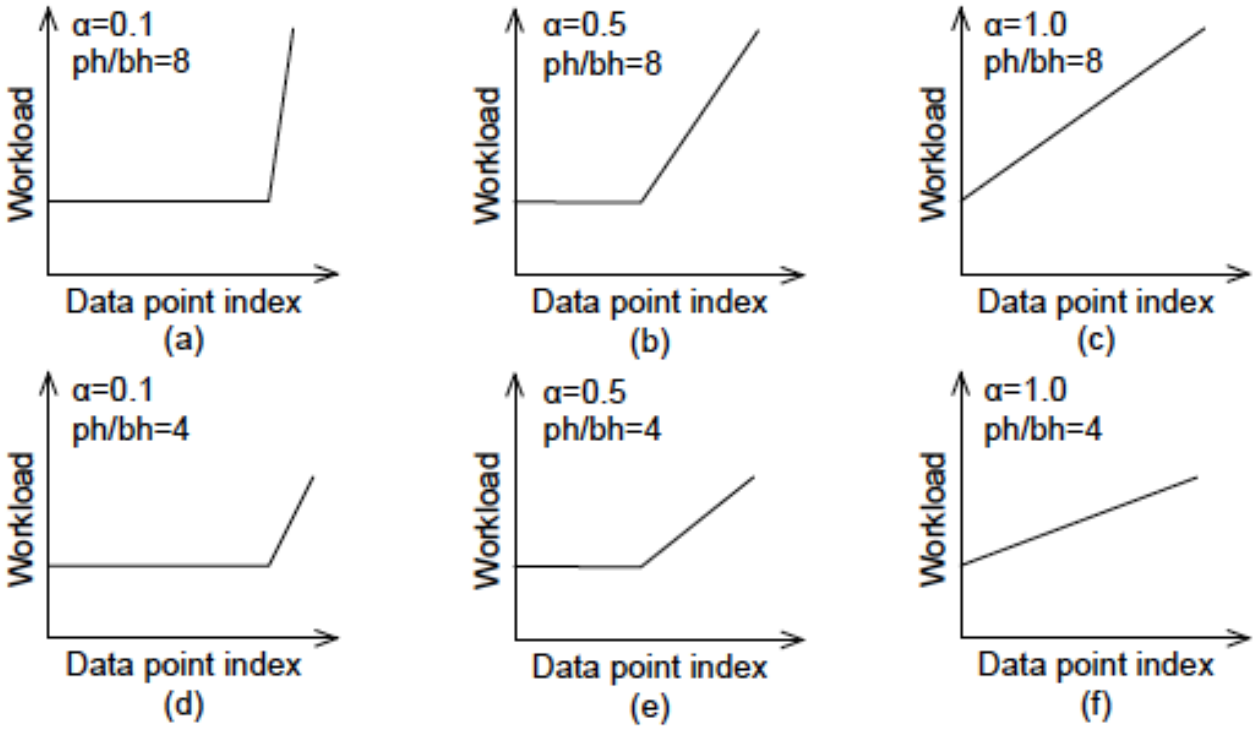
Results [1]



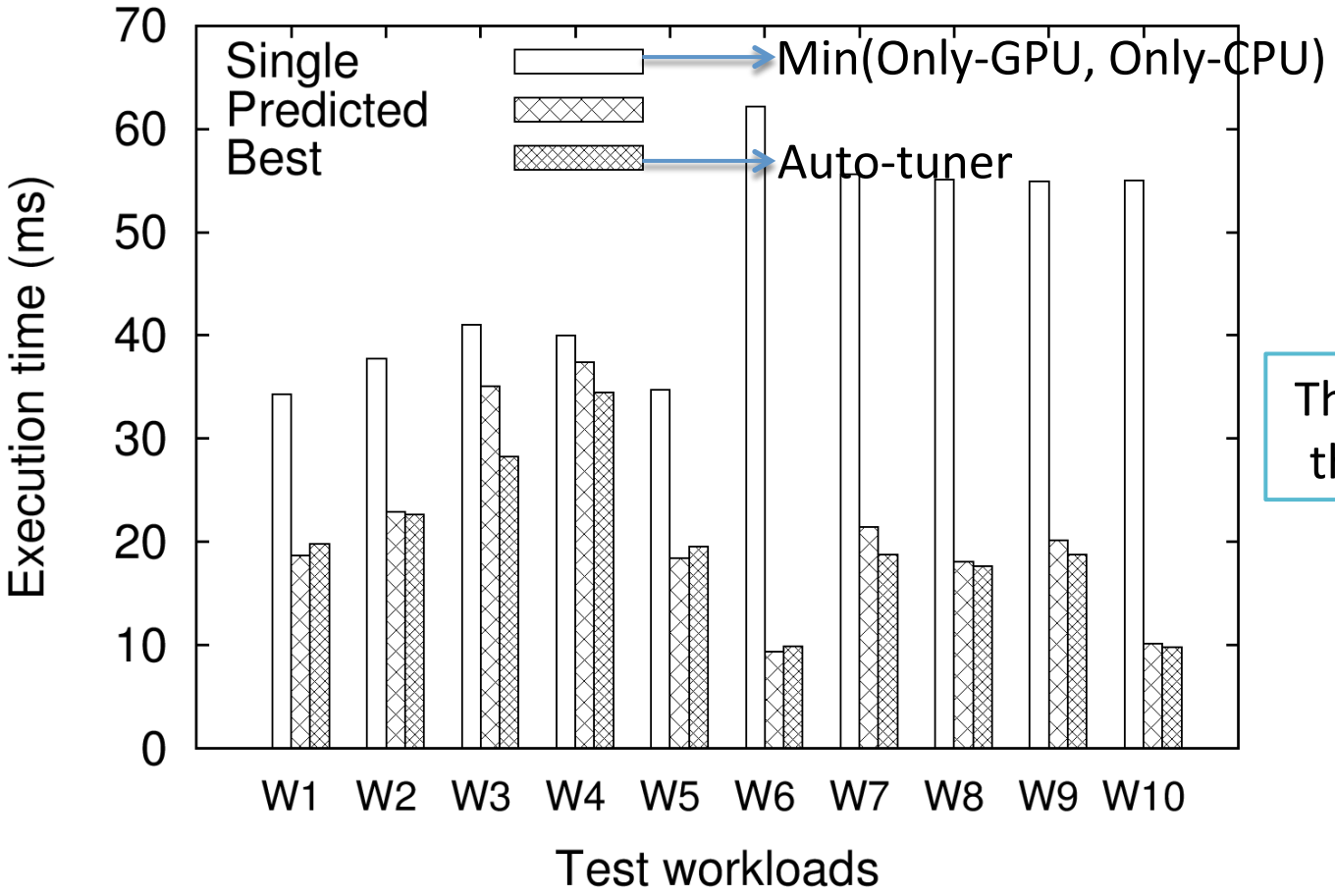
62% performance improvement compared to “Only-GPU”

Results on 10 workloads

- Changing the flight parameters we obtain 10 different wokloads.



Results [2]



The more imbalance, the larger speedup!

Summary Glinda

Why use it?

- Glinda enables the use of heterogeneous hardware for OpenCL workloads
  - Static partitioning is computed and implemented
  - Works for multi-GPUs
  - Works for data parallel kernels
- Resulting partitioned code can be re-used with minimum to moderate effort:
  - Different datasets
  - Different scenarios
  - Different platforms

How to use it?

- Profile the platform: R_{GC}, R_{GD}
- 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

Glinda in HPL

- HPL = Heterogeneous Programming Library*
- Two usage scenarios:
  - on top of existing OpenCL kernels
  - developing the heterogeneous kernels in the embedded language it provides.
- Implementation of Glinda in HPL
  - Automatically profiles the application
  - Computes the partitioning
  - It applies it on the array
- Minimizes programmer effort.

* HPL: http://hpl.des.udc.es

Almost done with static partitioning

- Questions

Other systems

Qilin

Insieme

SKMD

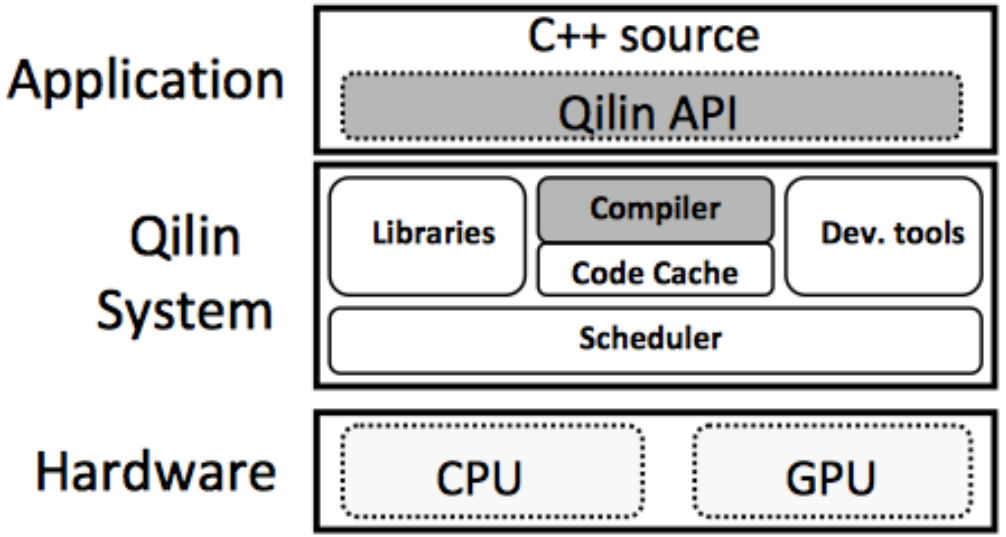
Qilin: API + partitioning

C.-K. Luk et al., PLDI'09 (MICRO'09)

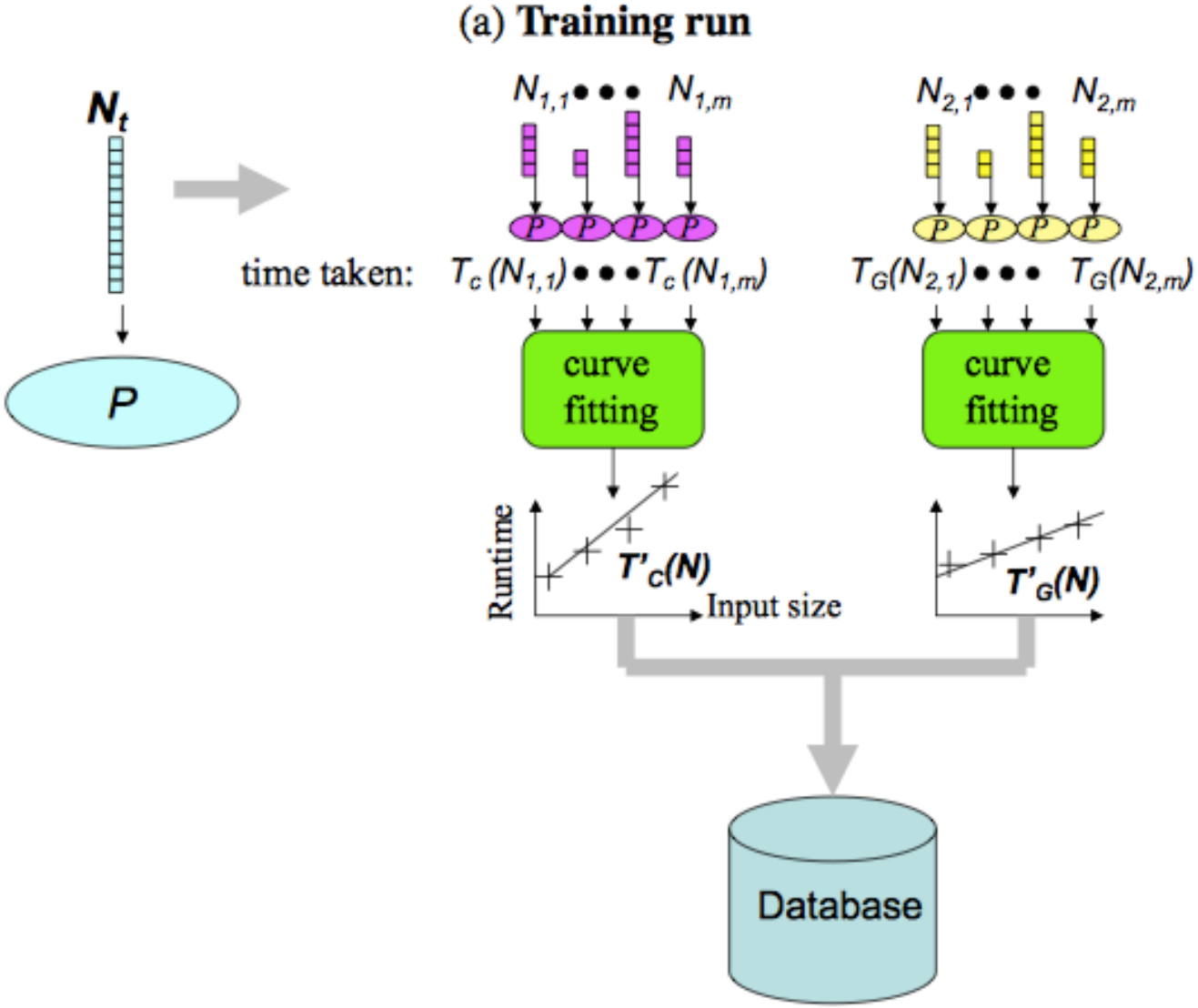
Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping

Qilin

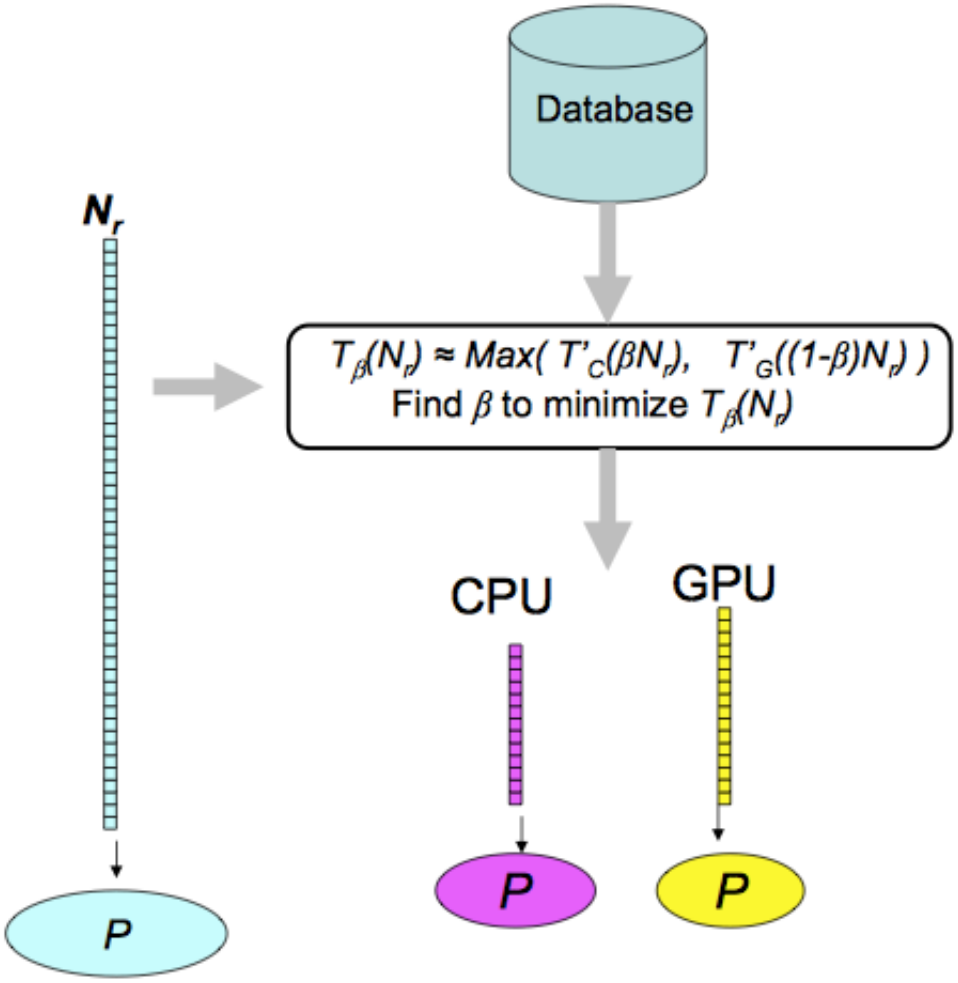
- *Fully automatic* approach to map computations to processing elements using *run-time adaptation*.
- *Qilin* = programming heterogeneous multiprocessors.
  - API for parallelizable operations
  - Generates partitioned code for the CPU and the GPU



Training phase



Deployment phase



Summary Qilin

- Adaptive mapping
- Model based on curve-fitting with linear equations.
- Relatively little training
- Own API and compiler
- Uses TBB and CUDA

What is different from Glinda?

Insieme

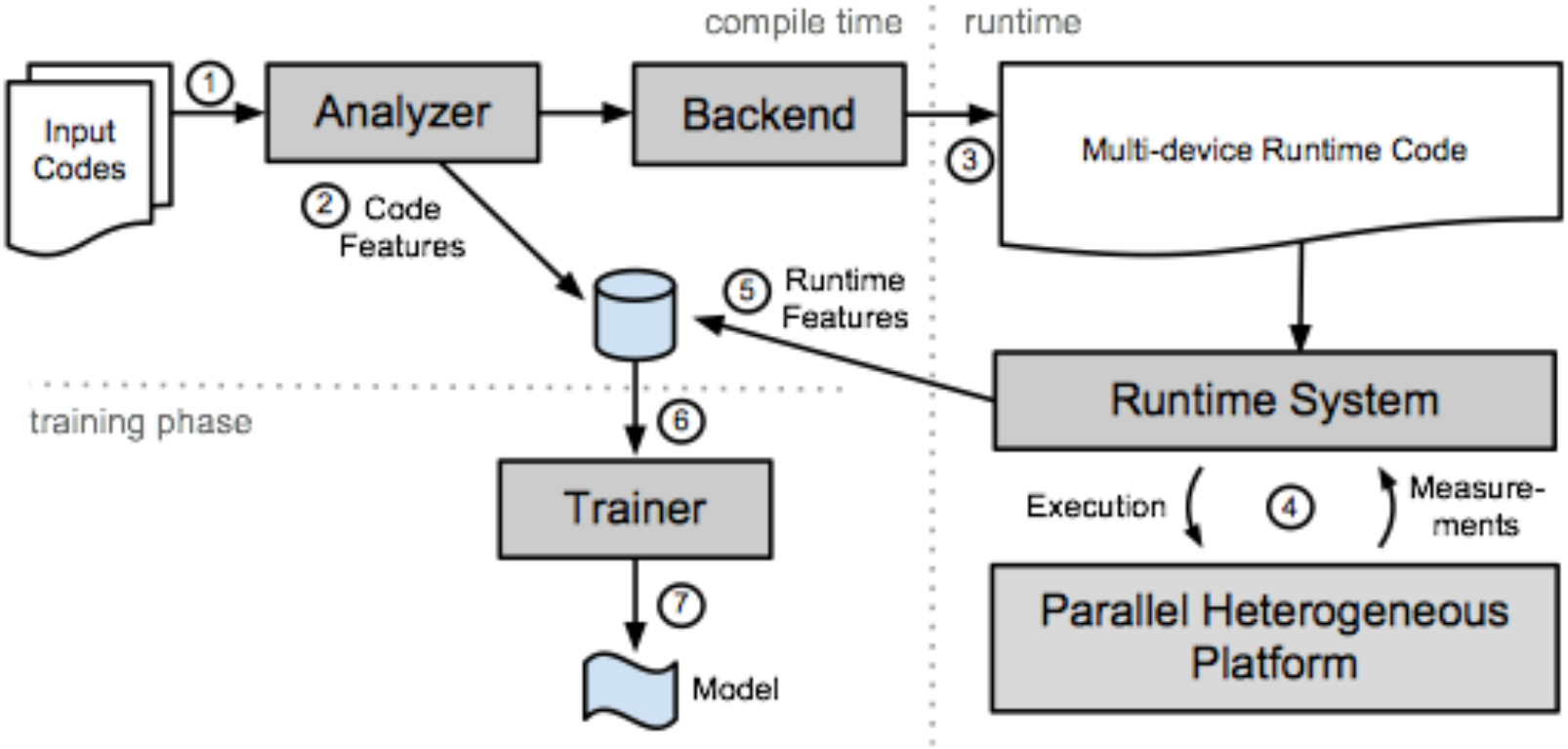
K.Kofler et. al, ICS'13

An Automatic Input-Sensitive Approach for Heterogeneous Task Partitioning

Goal

- Compiler + runtime method for task partitioning of OpenCL programs on heterogeneous systems.
  - Machine learning for combining *static analysis* with *runtime feature evaluation* => *machine model*
- Predicts optimal task partitioning for every combination of (program, problem size, and hardware configuration).
- Uses the Insieme Compiler and Runtime framework

Training


compile time
runtime
Input Codes
1
Analyzer
Backend
3
Multi-device Runtime Code
2 Code Features
5 Runtime Features
Runtime System
training phase
6
Trainer
Execution
4
Measure-ments
7
Model
Parallel Heterogeneous Platform

Training data

- A set of 23 codes applications/"codes".
- All training codes are compiled with the Insieme source-to-source compiler
- Static program features are collected in a database.
- Programs are executed with various problem sizes (9 – 18) and task partitionings
  - information about runtime features and execution times is added to the database.
- Two machines, 23 codes, 9-18 problem sizes, 21 partitionings: $355 \times 21 = 7455$
  - 355 problem size/program combinations
  - 21 task partitionings.

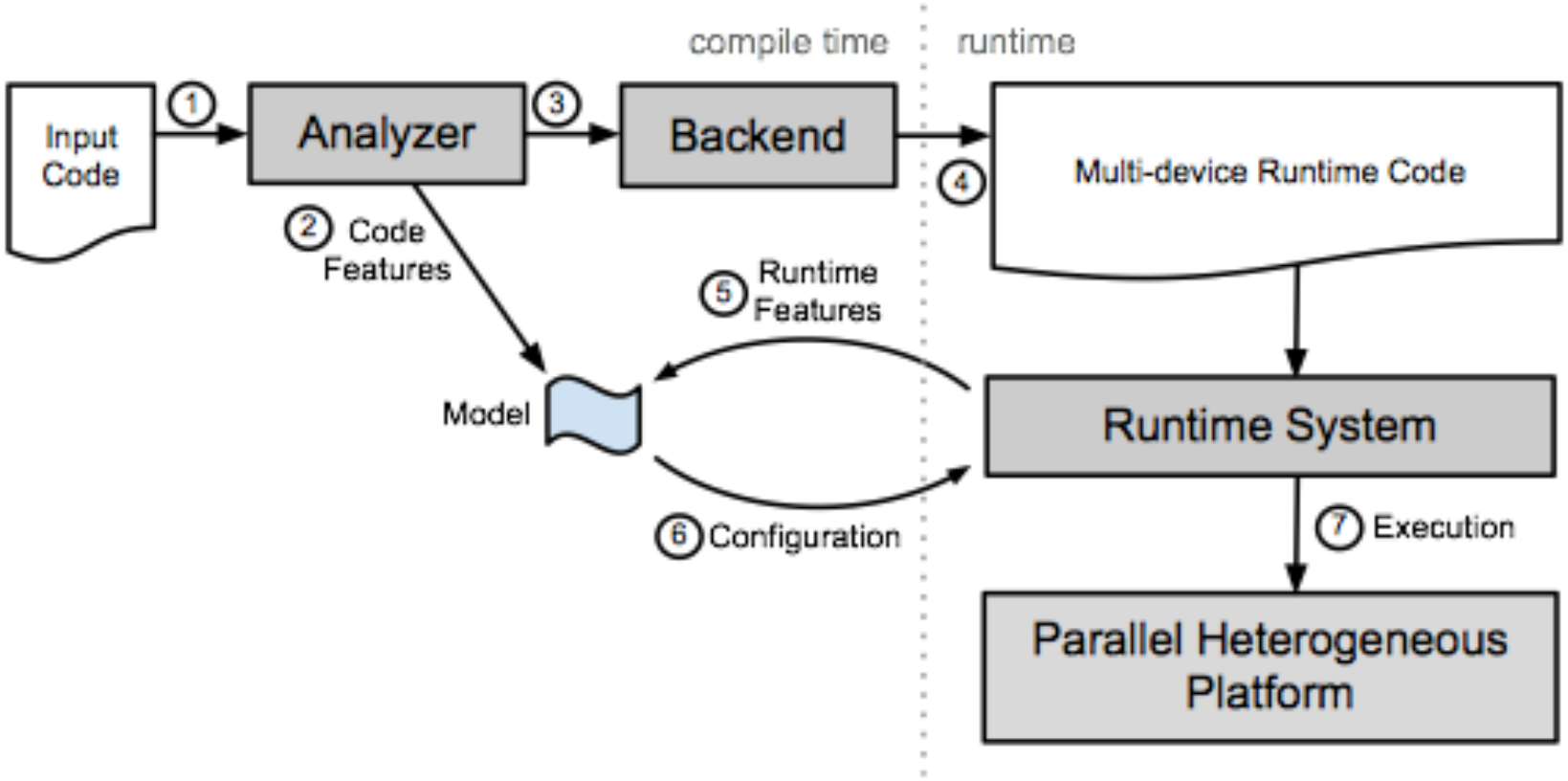
Features

- Examples of features
  - Specific for one hardware configuration
  - Used for partitioning on that configuration

Rk.	static program features	MSE
2	OpenCL built-in functions	76.3
3	Number of branches / number of statements	64.4
4	Scalar float operations /number of statements	61.1

Rk.	Runtime features	MSE
1	Data transfer size for *splittable buffer* (device to host)	99.7
5	Number of global work items	60.0
6	Data transfer size for *splittable buffer* (host to device)	47.6
7	Runtime feature Nr. 3 / total number of arith. operations	47.5

Deployment


compile time
runtime
Input Code
1
Analyzer
3
Backend
4
Multi-device Runtime Code
2 Code Features
5 Runtime Features
Model
Runtime System
6 Configuration
7 Execution
Parallel Heterogeneous Platform

Summary: Insieme

- Automated model based on machine learning
- Extensive training with predefined set of applications
  - No selection needed
- Supports multiple CPUs and GPUs in a single node
- Uses unaltered OpenCL code
  - Decides on splittable kernels and uses them.

What is different from Glinda?

Summary: static approaches

- Main goal: statically partition the workload *once*
- Main challenges:
  - Determine platform capabilities
  - Determine application performance characteristics
    - Profiling
      - On-line training, not feasible for one-time execution
    - Modeling
      - Off-line training + machine learning
      - On-line training + curve fitting
- Programming models: CUDA, OpenMP, TBB, OpenCL
  - Extra APIs provided

Summary

Feature	Glinda	Qilin	Insieme	SKMD
Training	Online: Profiling per datasize	Online: Profiling per datasize + sampling	Special training set	On-line profiling
Partitioning	Off-line	On-line	On-line	On-line
Granularity	Data-driven	Data-driven	Data-driven	Workgroup
Model	Analytical	Curve-fitting	Machine learning	Decision Tree Heuristics
GPUs	Multi, asymmetrical	Single	Single [?]	Multi, asymmetrical
Programming model	OpenCL (OpenMP+CUDA)	TBB+CUDA	OpenCL	OpenCL
Manual effort	Enable multi-device code	Little	None	Little

Done with static partitioning

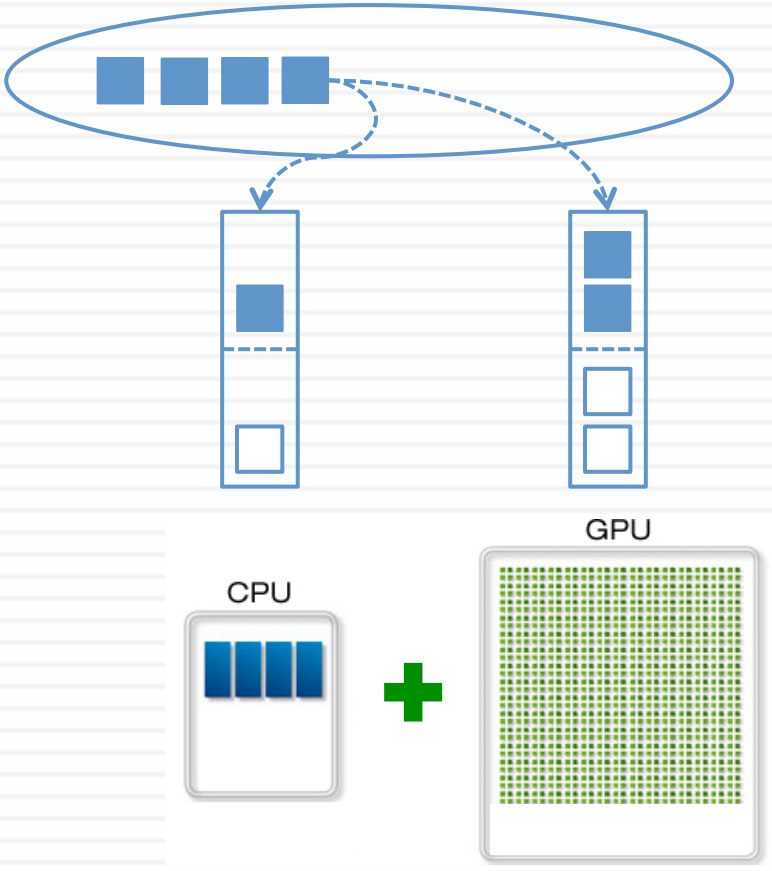
- Questions

Dialogue is good

- Talk to your neighbour on the right
- Exchange 2 keywords from your research

- Now say out loud one of them.

Dynamic approaches



OpenCL single-node approach

Boyer et al, CF'13

Load Balancing in a Changing World: Dealing with Heterogeneity and Performance Variability

Goal

- Dynamic OpenCL-based approach to work-partitioning
  - no offline training
  - responds automatically to runtime performance variability
- Single kernel
  - partition the kernel efficiently into chunks of contiguous work groups
  - Schedule chunks for execution across multiple devices

Approach

- "Sample-and-model"
  - send a small portion of the available work to each device and run in parallel
  - Use the execution time of initial work to build a model to partition remaining work.
  - Partition at workgroup level

Results show improvement over non-heterogeneous execution. Note different behavior for different applications, as some applications are more predictable than others.

An OpenMP idea

Thomas R. W. Scogland et. al, IPDPS'12

Heterogeneous Task Scheduling for Accelerated OpenMP

Goal and approach

- Runtime system that can divide an accelerated OpenMP region across all available resources automatically.
  - Dynamic task scheduling across CPUs and GPUs.
  - Must
    - (1) split regular OpenMP accelerator loop regions across compute devices
    - (2) manage the distribution of inputs and outputs while preserving the semantics of the original region transparently.
- Specify the relationship between the compute units, rather than trying to find a single sensible unit of work to assign to both.

Different splitting policies

Divide tasks at the beginning of a parallel region, using a CPU:GPU ratio

Static

- Ratio based on theoretical compute capabilities

Dynamic

- Measure one run, change the ratio accordingly

Split

- Adjust every iteration

Quick

- Dynamic + split

Results: Performance is better than non-heterogeneous, and the variety of policies provides an overhead tuning knob.

Summary dynamic partitioning

- Scenario:
  - one-time run of a (large/complex) application
  - Single-kernel / independent kernel[s]
- Principle
  - Chunk work in sample intervals
  - Tune granularity
  - Trade-off between on-line trial runs and “good” runs
- Efficiency
  - Balance between sampling granularity and application variability

Done with dynamic partitioning

- Questions

Quiz

- What do all these systems have in common ?

“Domain specific” attempts

Graph processing: Totem

MapReduce: GlassWing

Heterogeneous graph processing

TOTEM

A. Gharaibeh et al: *A Yoke of Oxen and a Thousand Chickens for Heavy Lifting Graph Processing, PACT'12*

GlassWing

Graph processing challenges

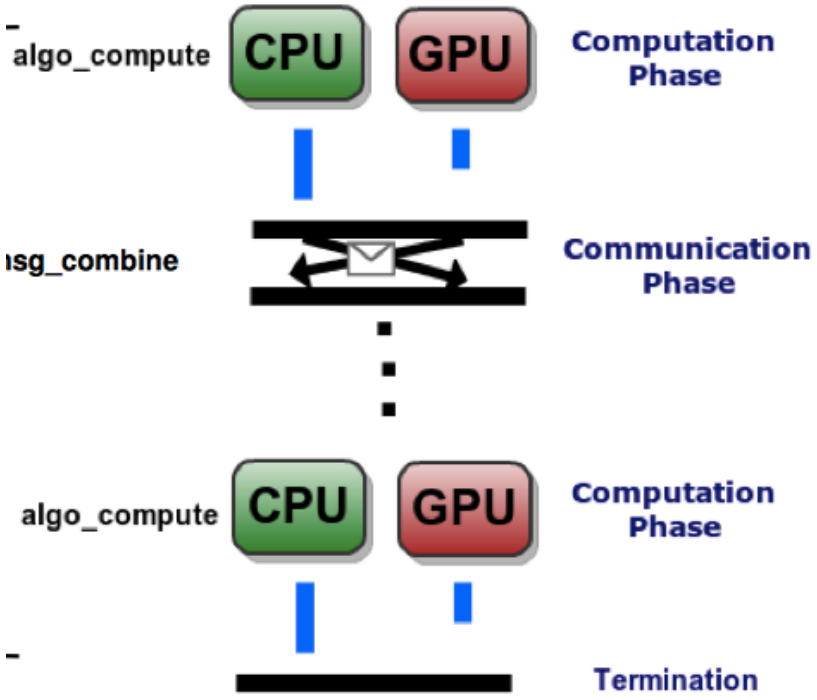
- Poor locality
- Data-dependent memory access patterns
- Low compute-to-memory access ratio
- Large memory footprint (>256GB)
- Varying degrees of parallelism (both intra- and inter- stage)

Can we efficiently use hybrid systems for large-scale graph processing?

Totem

A graph processing engine for hybrid (read: heterogeneous) systems

- Bulk Synchronous Parallel
  - Rounds of computation and communication phases
  - Updates to remote vertices are delivered in the next round
  - Partitions vote to terminate execution



Totem

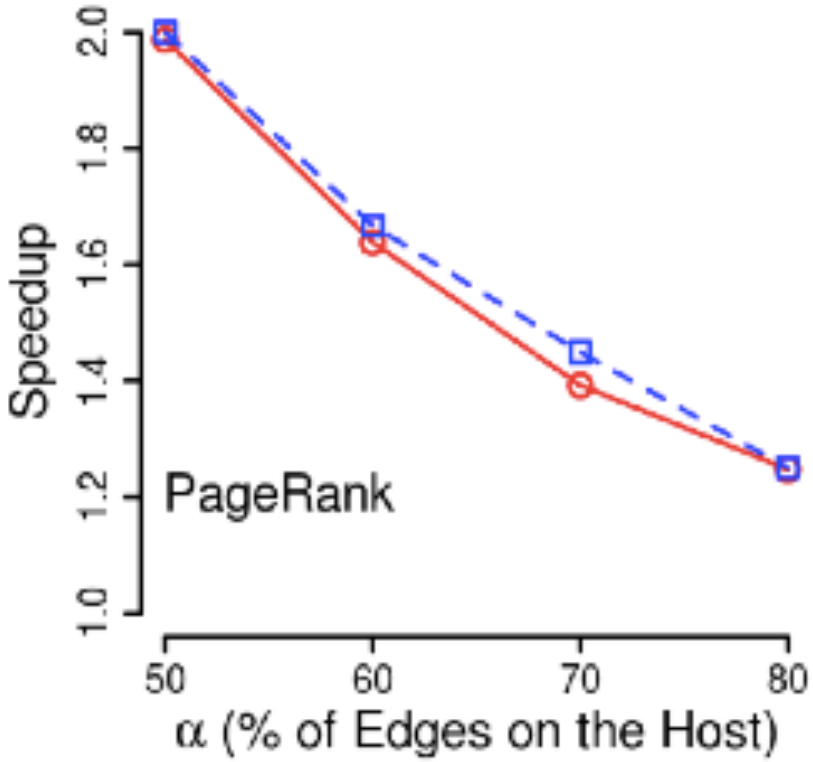
- Partition the graph
- A user-defined kernel runs simultaneously on each partition
  - Run same computation on each partition
  - Vertices are processed in parallel within each partition
- Messages to remote vertices are combined
  - Applies algorithm-agnostic optimizations
- Enables studies on partitioning Strategies
  - Can one do better than random?

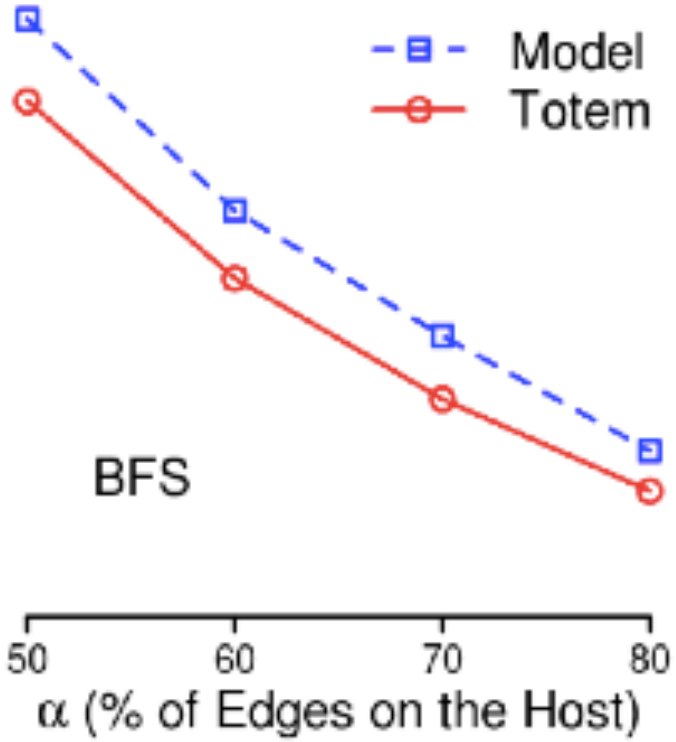
Performance model

- Predicts speedup
- Partition per processor p : Gp = (Vp, Ep)
- Define makespan: m(G) = max { t(Gp) }
- Parameters per partition:
  - α = *share of* edges
  - β = percentage of *boundary* edges (i.e., the edges that cross the partition).
- Predicted speed-up:

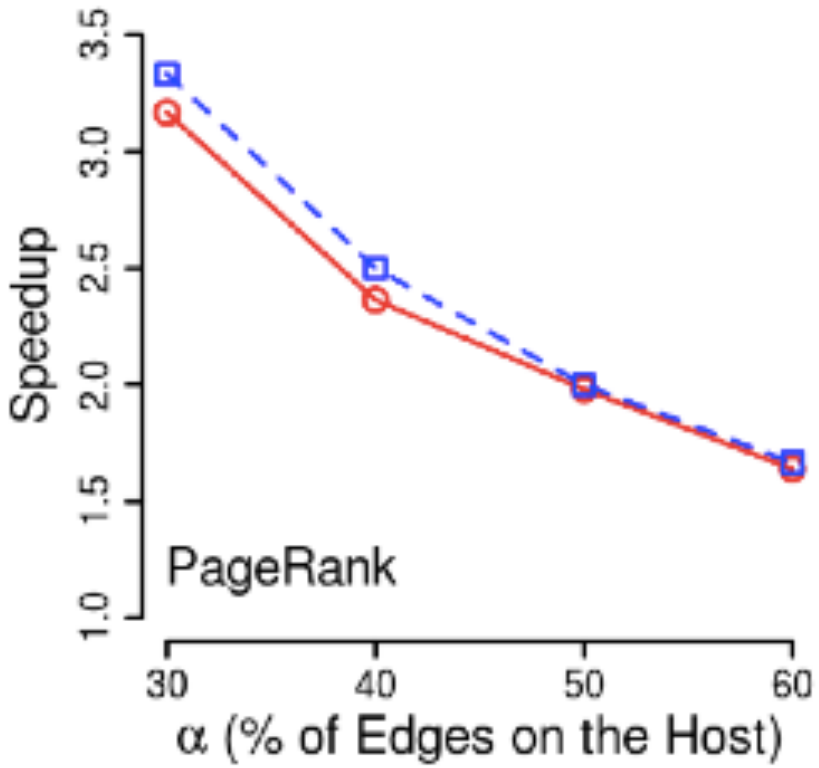
$$s_P(G) = \frac{|E|/r_{cpu}}{\beta|E|/c + \alpha|E|/r_{cpu}} = \frac{c}{\beta r_{cpu} + \alpha c} = \frac{1}{\frac{\beta \cdot r_{cpu}}{c} + \alpha} \quad (4)$$

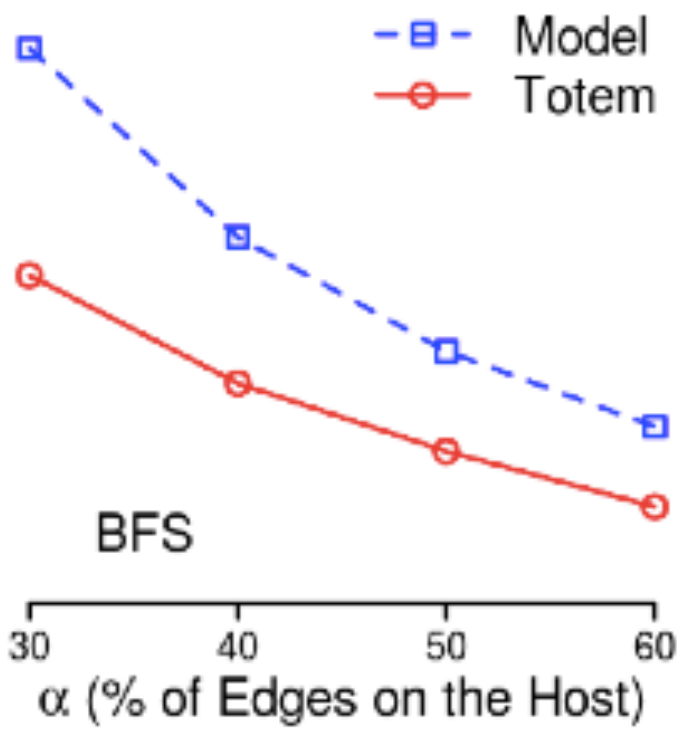
Results: 1GPU





Results: 2 GPUs





Summary: Totem

- Analytical model for performance prediction of graph processing on heterogeneous systems
- Understand the effectiveness of GPU processing for graphs
- Single-node, multi-GPU solution

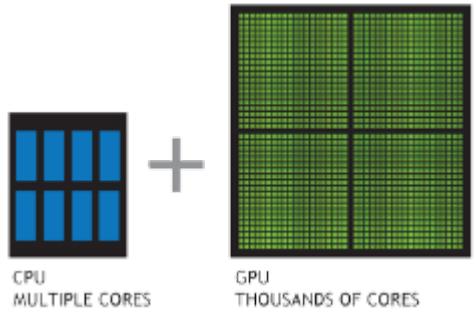
Heterogeneous MapReduce

Glasswing

Ismail El Helw, Rutger Hofman, Henri Bal – in HPDC'2014, SC'2014

Glasswing: Rethinking MapReduce

- Use accelerators (OpenCL) as mainstream feature
- Massive out-of-core data sets
- Scale vertically & horizontally
- Maintain MapReduce abstraction



GPU optimizations

- Overlaps computation, communication & disk access
- Supports multiple buffering levels
- GPU optimizations:
  - Memory management
  - Some shared memory optimizations
  - Data movement, data staging
- Programmer:
  - Focusses on the map and reduce kernels (using OpenCL)
  - Can do kernel optimizations if needed
    - Coalescing, memory banks, etc.

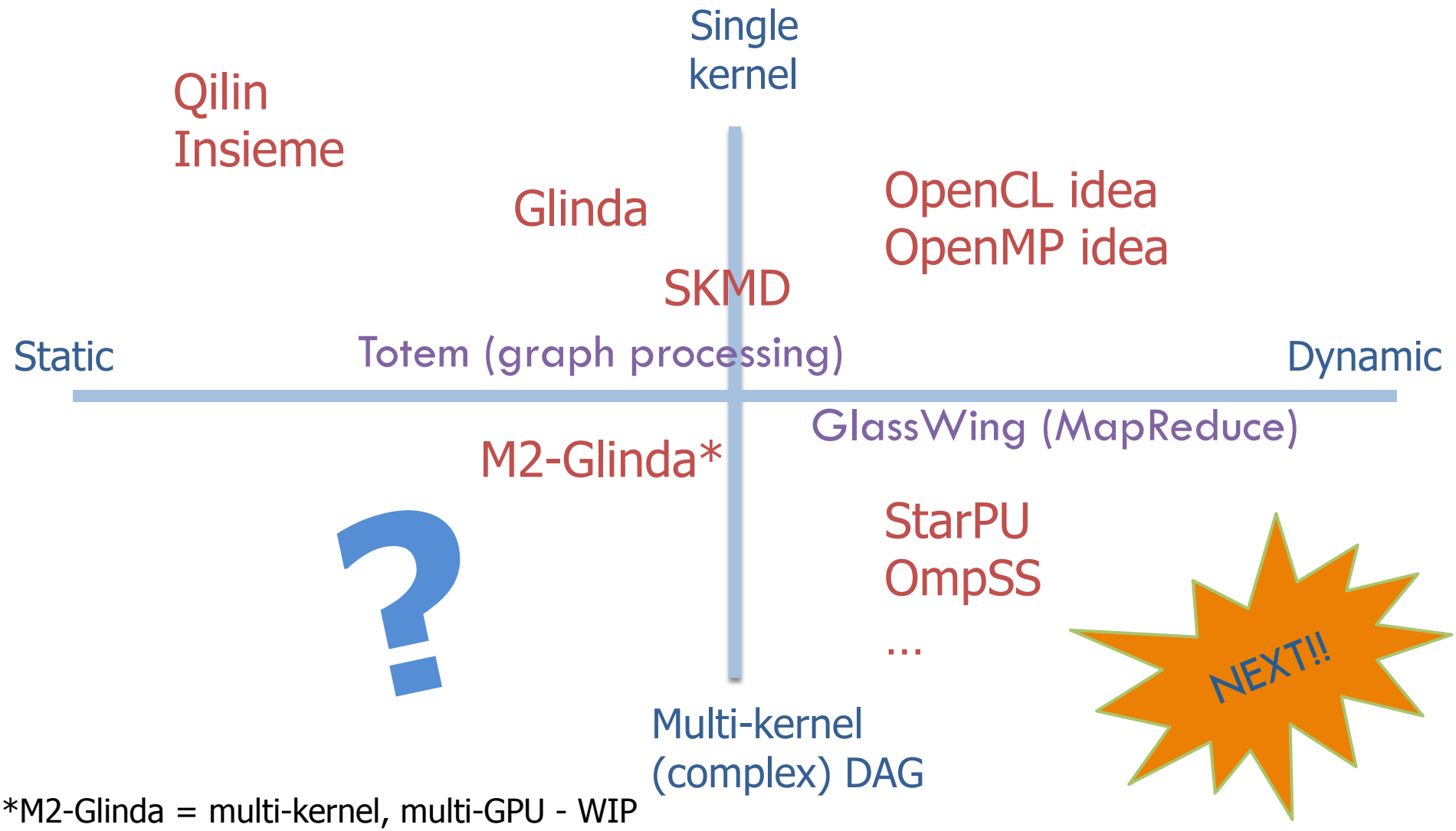
Lessons learned

- Scalable MapReduce framework combining coarse-grained and fine-grained parallelism
- Handles out-of-core data, sticks with MapReduce model
- Overlaps kernel executions with memory transfers, network communication and disk access
- Outperforms Hadoop by 1.2 – 4x on CPUs and 20 – 30x on GPUs

144

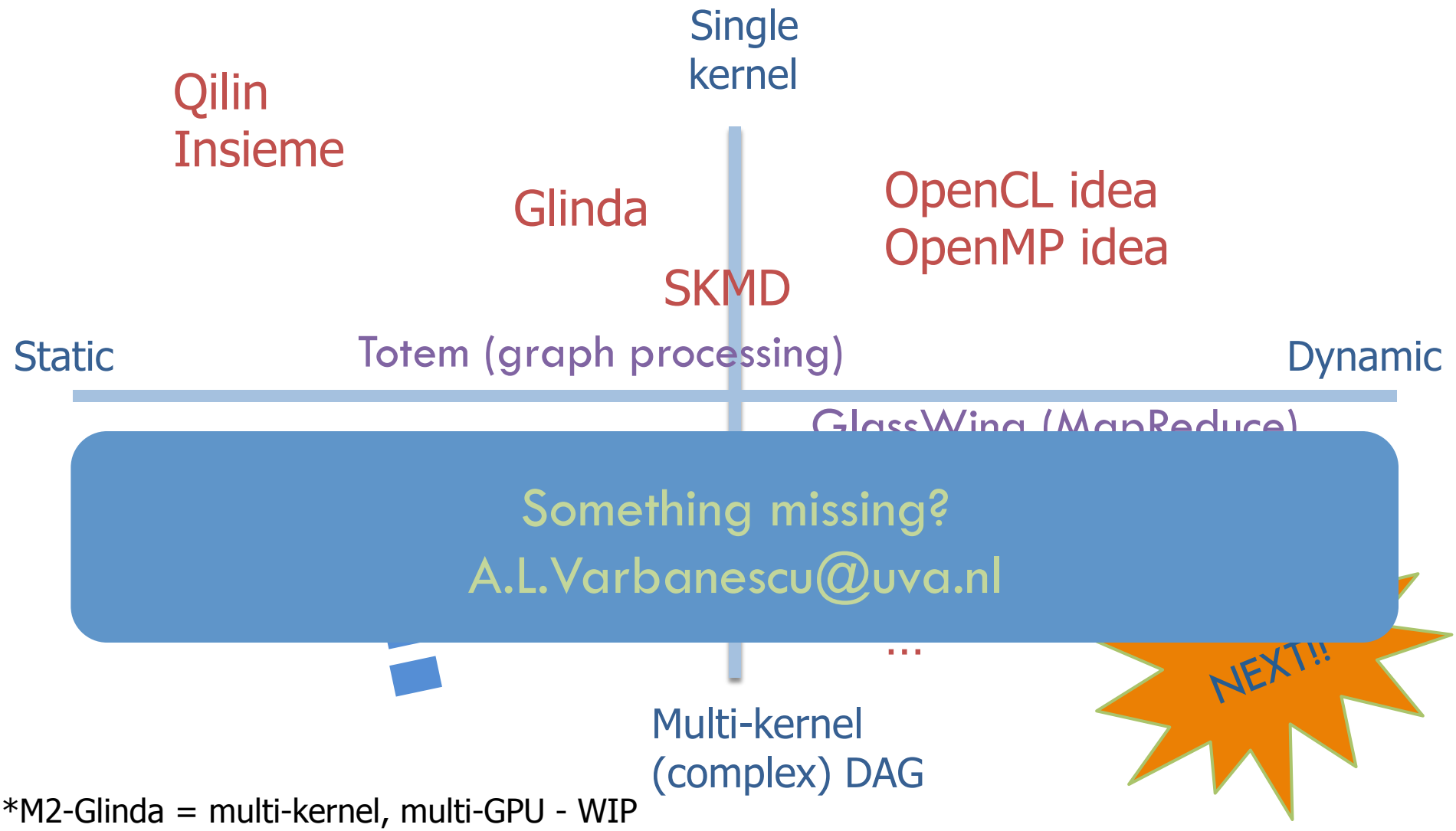
Landscaping attempt

Heterogeneous Computing



*M2-Glinda = multi-kernel, multi-GPU - WIP

Heterogeneous Computing



*M2-Glinda = multi-kernel, multi-GPU - WIP

Open questions

- Connection with embedded systems?
- Connection with distributed systems?

How much are we actually reinventing the wheel ?

Done with the simple part

- Single-kernel applications
  - Or collections of independent kernels
- Static and dynamic partitioning
- Small set of devices

- Part 2:

Multi-kernel applications and runtime-based solutions for heterogeneous computing.

Take home message [1]

- Heterogeneous computing works!
  - *More resources.*
  - *Specialized resources.*
- We mostly discussed Glinda and StarPU
  - Glinda = static partitioning for single-kernel, data parallel applications
  - StarPU = run-time based dynamic partitioning for multi-kernel, complex DAG applications

Take home message [2]

- 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?

We just made your life harder:
yet another choice to make!