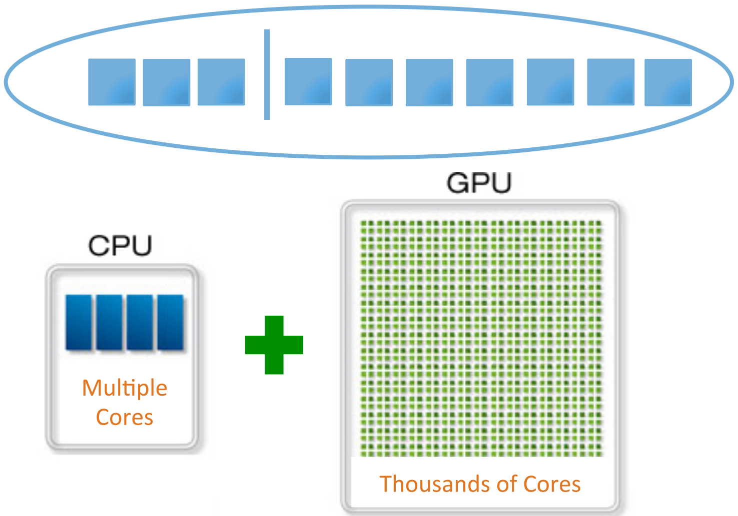
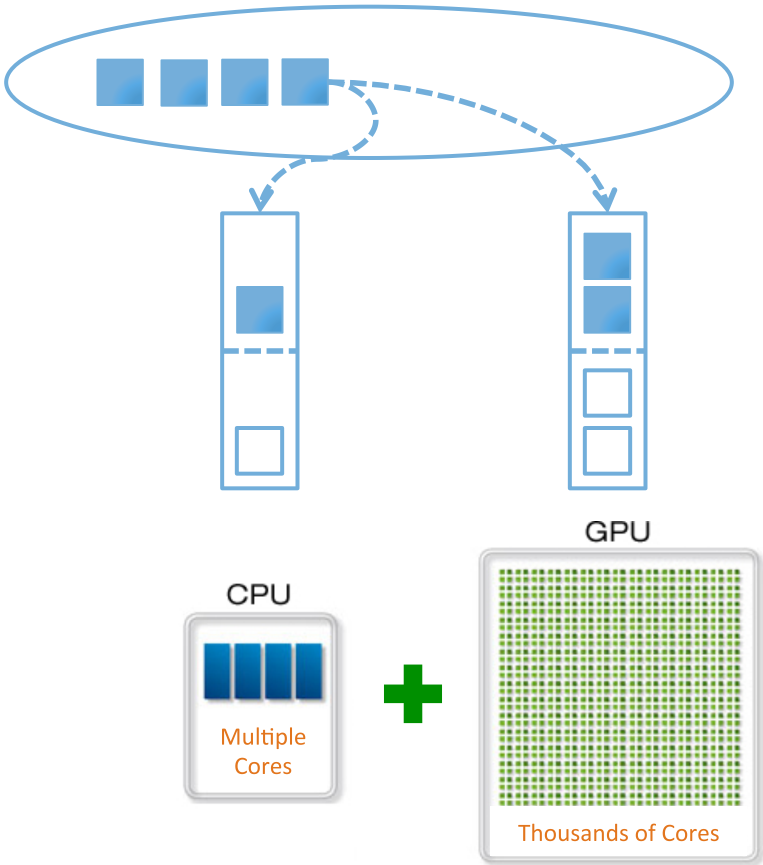
Determining the partition

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



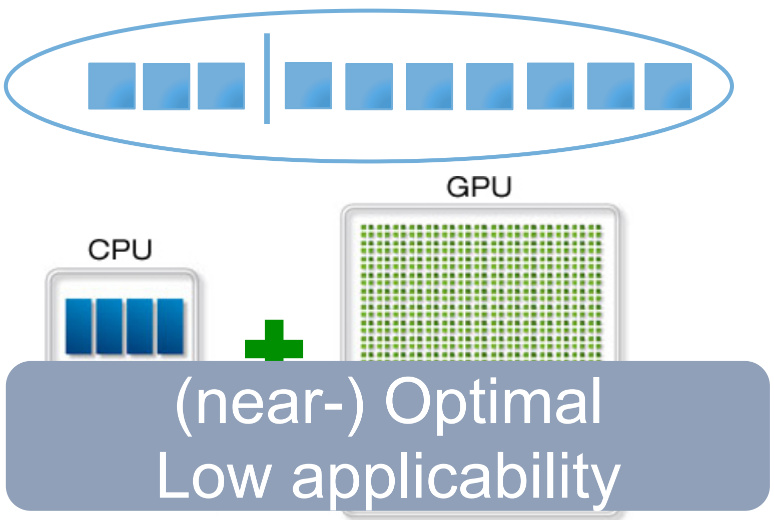


Static vs. dynamic

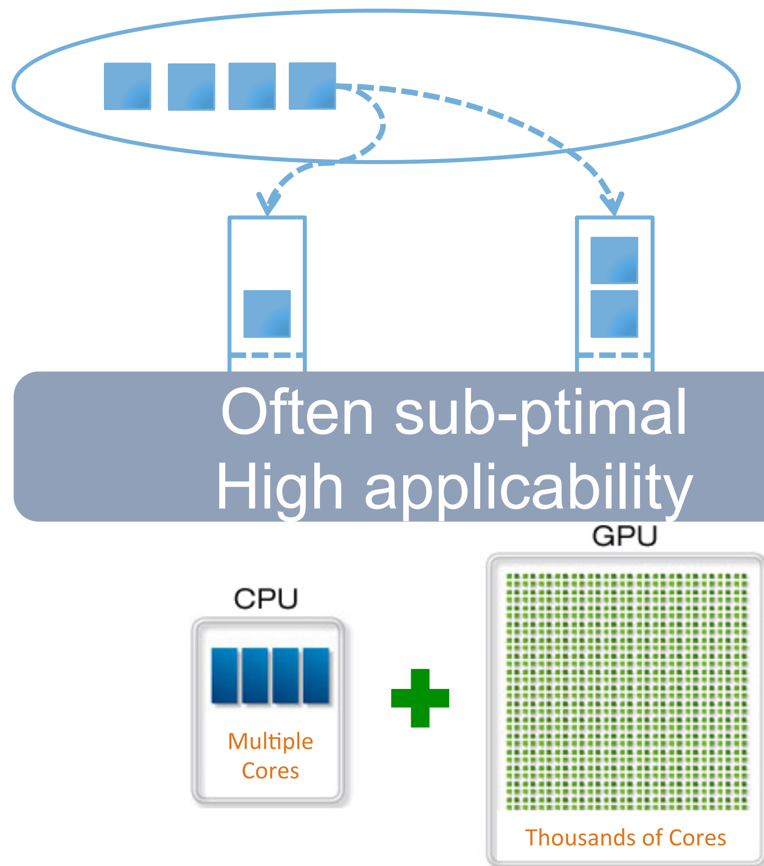
- Static partitioning
  - + can be computed before runtime => no overhead
  - + can detect GPU-only/CPU-only cases
  - + no unnecessary CPU-GPU data transfers
  - -- does not work for all applications
- Dynamic partitioning
  - + responds to runtime performance variability
  - + works for all applications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



(near-) Optimal
Low applicability



Often sub-ptimal
High applicability

Heterogeneous Computing today

Limited applicability.
Low overhead => high performance

Single kernel

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

Systems/frameworks: Qilin, Insieme, SKMD, Glinda, ...

Libraries: HPL, ...

Sporradic attempts and light runtime systems

Static

Dynamic

Glinda 2.0

Run-time based systems:
StarPU
OmpSS
...

Low overhead => high performance
Still limited in applicability.

Multi-kernel (complex) DAG

High Applicability, high overhead

GLINDA

Glinda: our approach*

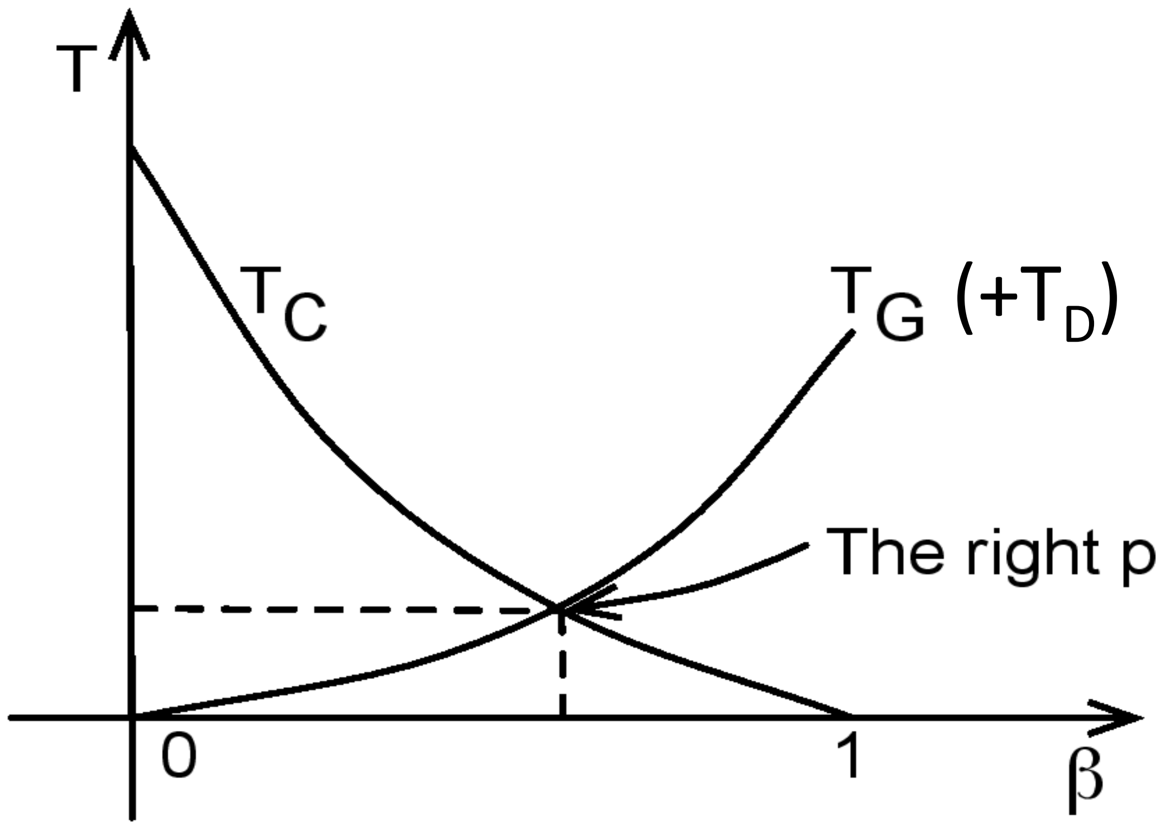
- Modeling the partitioning
  - The application workload
  - The hardware capabilities
  - The GPU-CPU data transfer
- Predict the optimal partitioning
- Making the decision in practice
  - Only-GPU
  - Only-CPU
  - CPU+GPU with the optimal partitioning

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

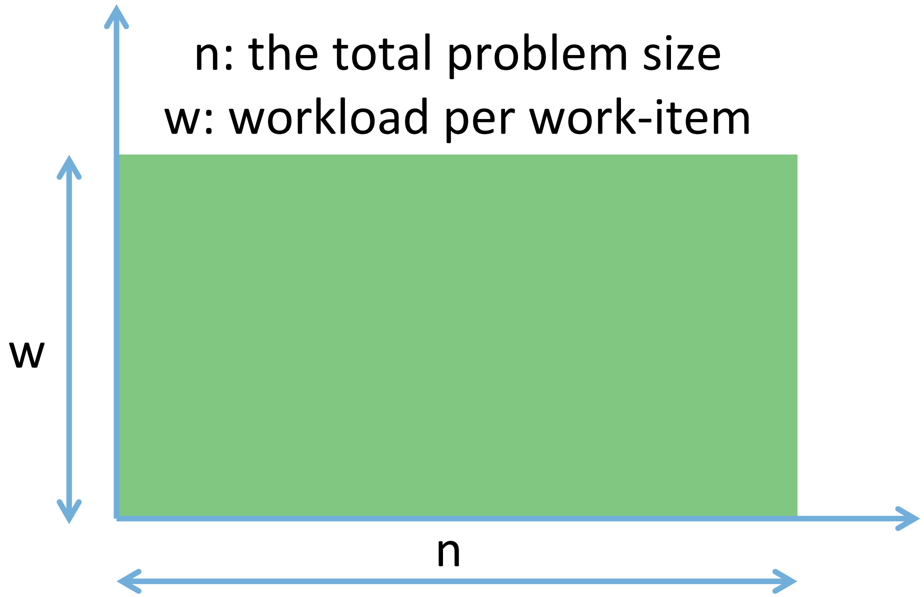
Modeling the partitioning

- Define the optimal (static) partitioning
  - β= the fraction of data points assigned to the GPU

$$T_G + T_D = T_C$$

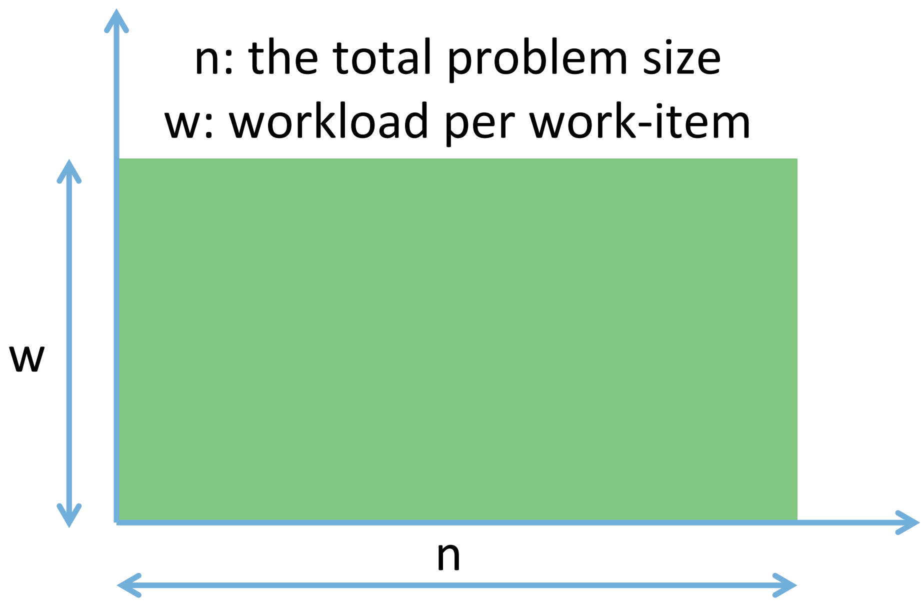


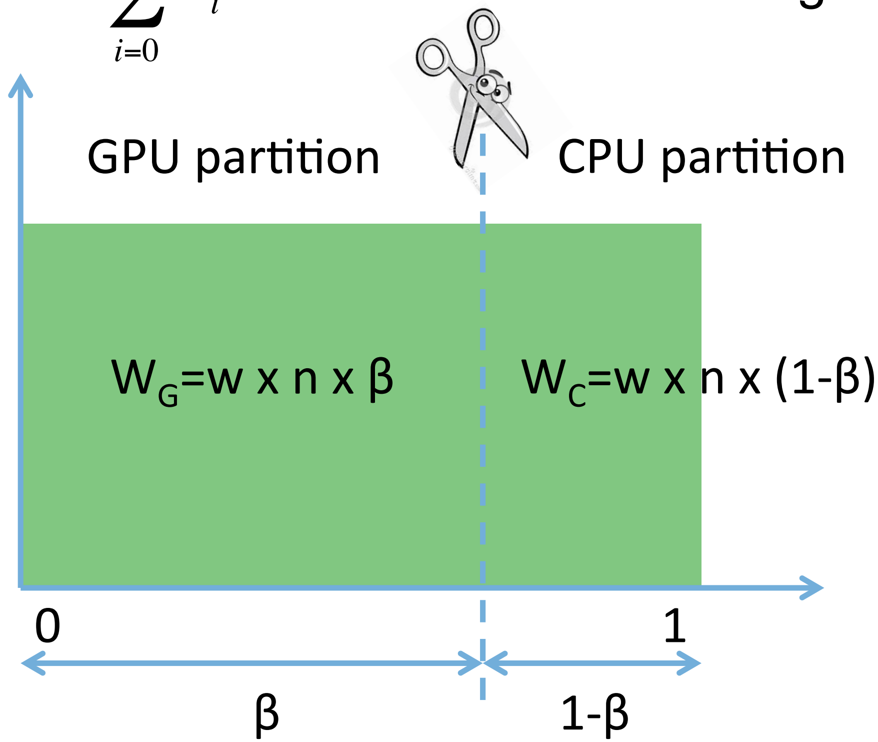
Model the app workload



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)**

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

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 …

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

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

… by solving this equation in β

Predict the partitioning

- Solve the equation

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

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

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

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

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

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

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

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

β predictor

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

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

No data transfer

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

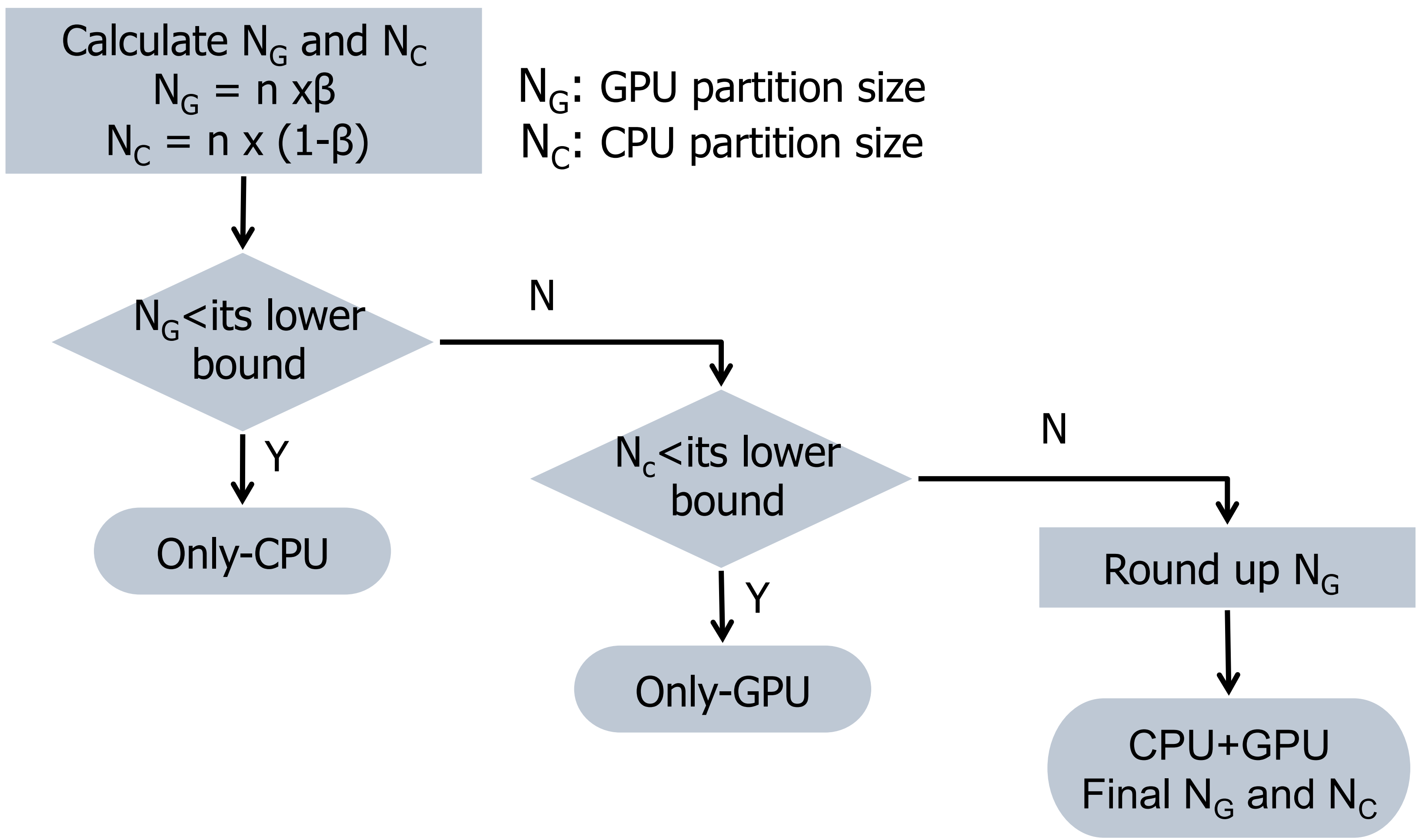
Partial data transfer

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

Full data transfer

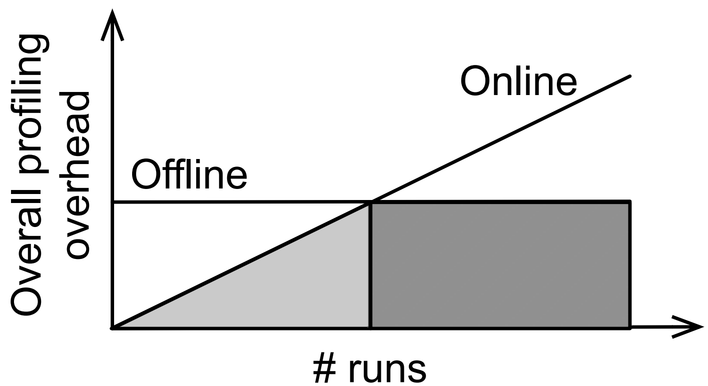
Making the decision in practice

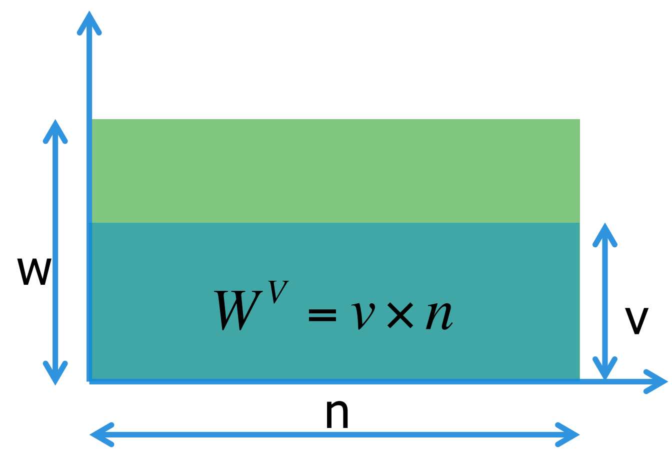
- From β to a practical HW configuration



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)

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?

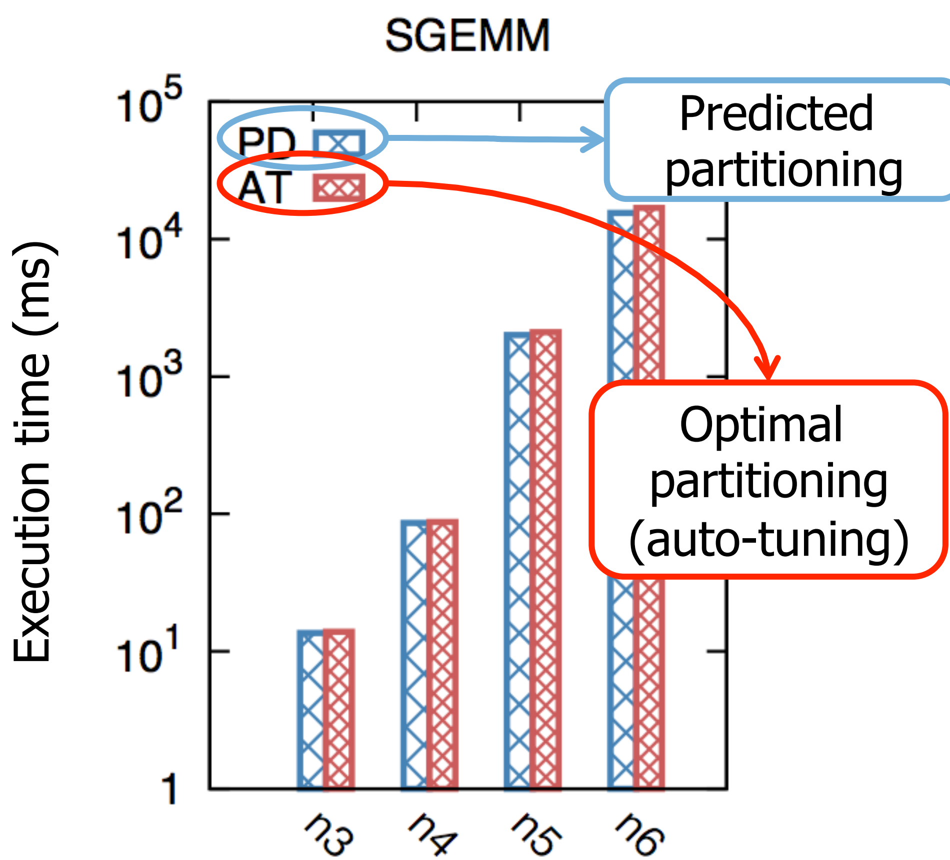
- Profile the platform: RGC, RGD
- Configure and use the solver: β
- Take the decision: Only-CPU, Only-GPU, CPU+GPU (and partitioning)
  - if needed, apply the partitioning

- 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

Results [1]

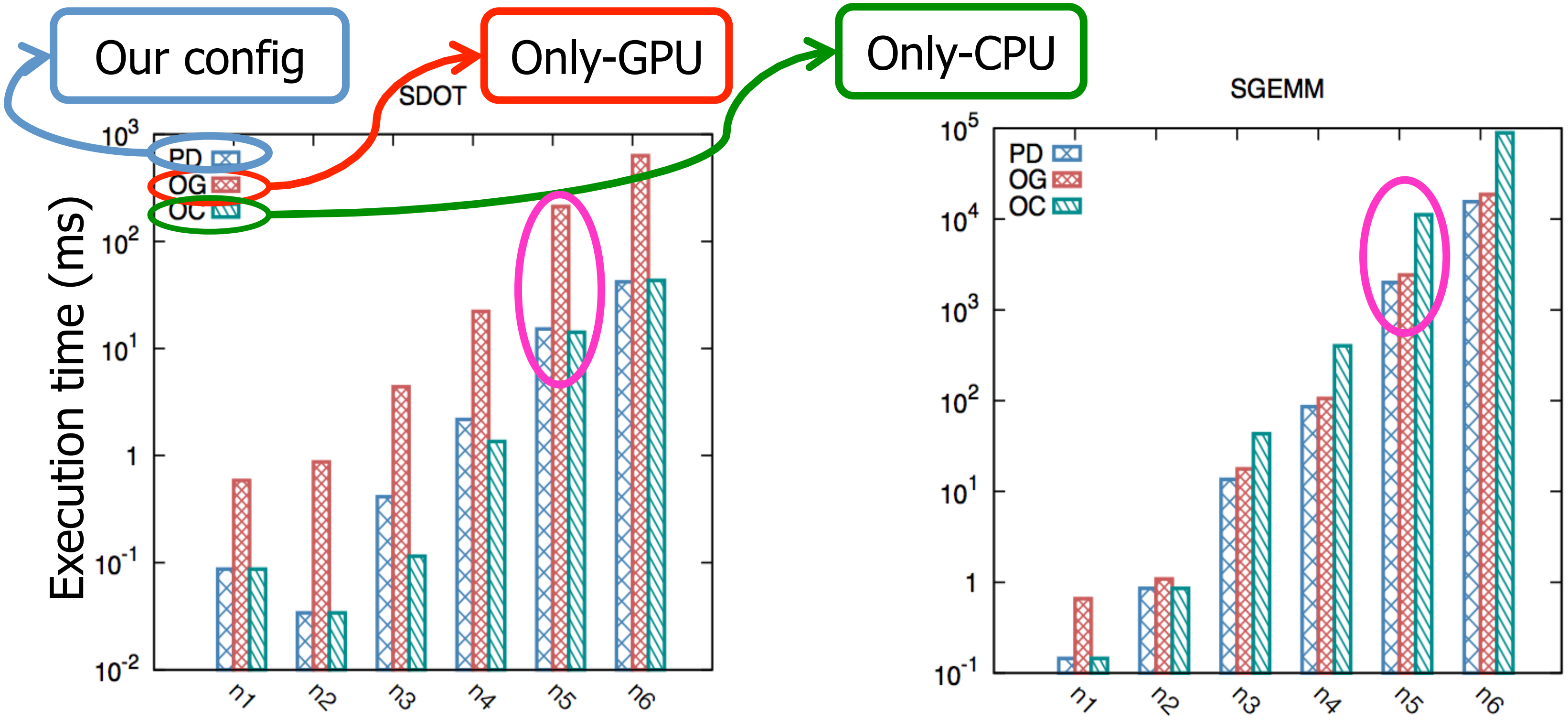
- Quality (compared to an oracle)
  - The right HW configuration in 62 out of 72 test cases
  - Optimal partitioning

  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



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, work from U. Edingburgh

Related work

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

		Machine learning [1,2]	Qilin [3]	Ours	
				Online	Offline
Cost (in relative comparison, +++ large, ++ medium, + small, ~ minor, 0 zero)	Collection	+++	++/+ (depending on *m*)	~	+
	Training	+++/++	+	0	+
	Deployment	~ (including code analysis)	0	0	0
	Adaption	+++	++/+ (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)

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

More about Glinda

- Simplistic application modeling >> Imbalanced applications
- Better than profiling? >> Performance modeling
- Single GPU only? >> NO, we support multiple GPUs/ accelerators on the same node
- Single node only? >> YES – for now .

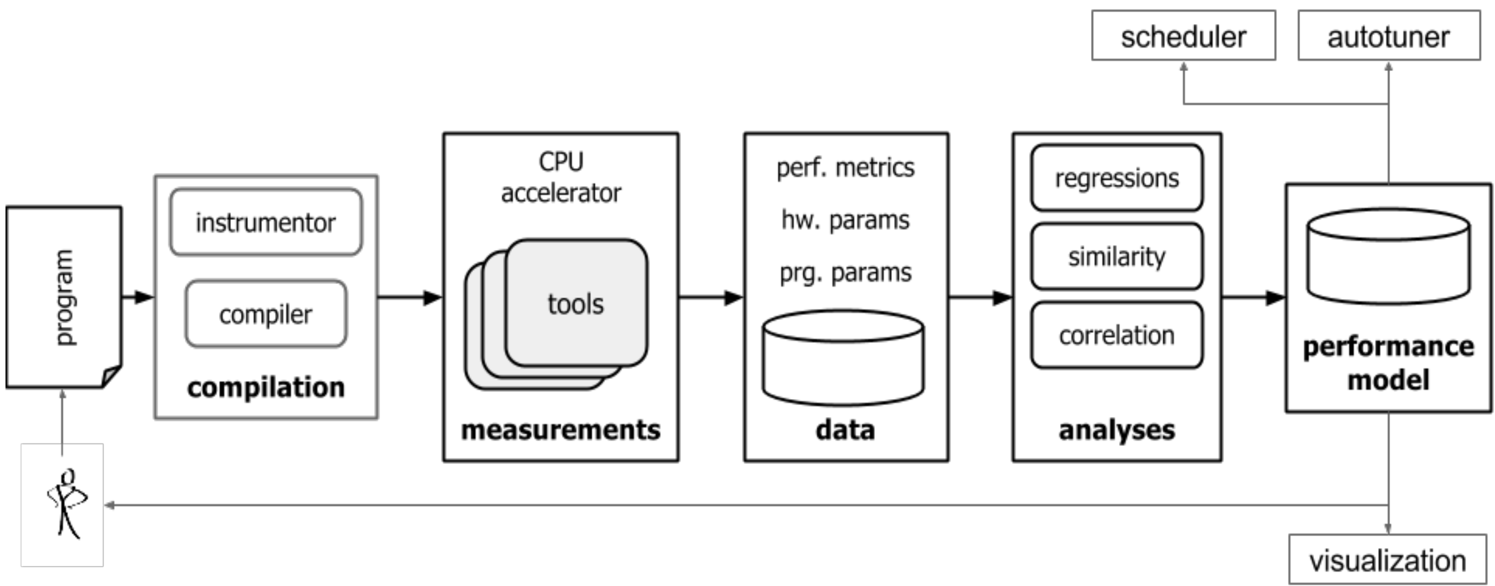
GPU Performance prediction

- State-of-the-art is disappointing

Model	Accuracy	Construction Effort	Evaluation Speed	Abstraction Level	Hardware Knowledge	Insight
PMAC	low	major	high	medium	limited	ETP
MWP-CWP	low	major	high	fine	high	ETP
GPUPerf	?	major	high	fine	high	ETP, PBA
GPU à la PRAM	V	moderate	high	coarse	none	ETP
Components	V	major	high	coarse	good	ETP, PBA
Eiger	low	moderate	low	coarse	good	ETP, OSE
STARGAZER	low	minor	low	fine	good	ETP, PBA, OSE
RandomForest	V	minor	high	coarse	none	ETP, PBA, OSE
QA	?	moderate	high	coarse	none	ETP, PBA
WFG	V	major	high	fine	good	ETP, PBA

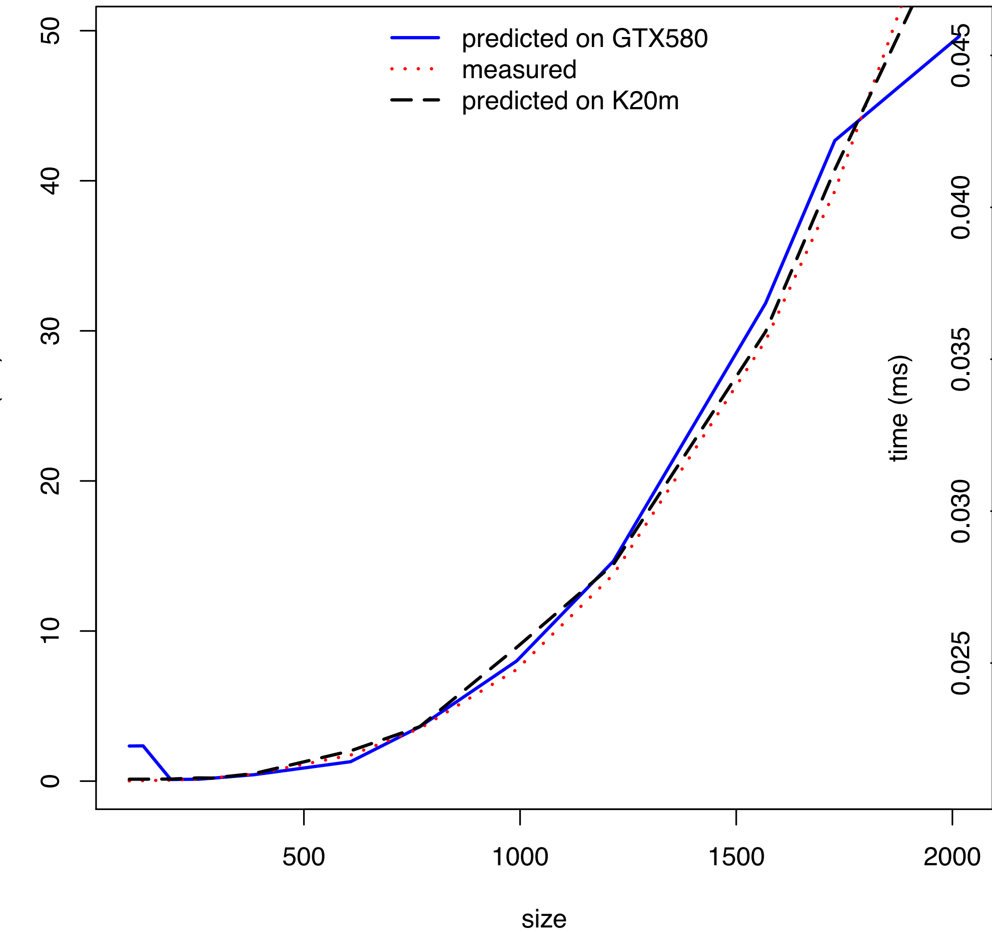
BlackForest Framework

- Compilation: optional, scope limitation by instrumentation
- Measurements: performance data collection via hardware performance counters
- Data: repository, file system, database
- Analyses: reveal correlation between counter behavior and performance

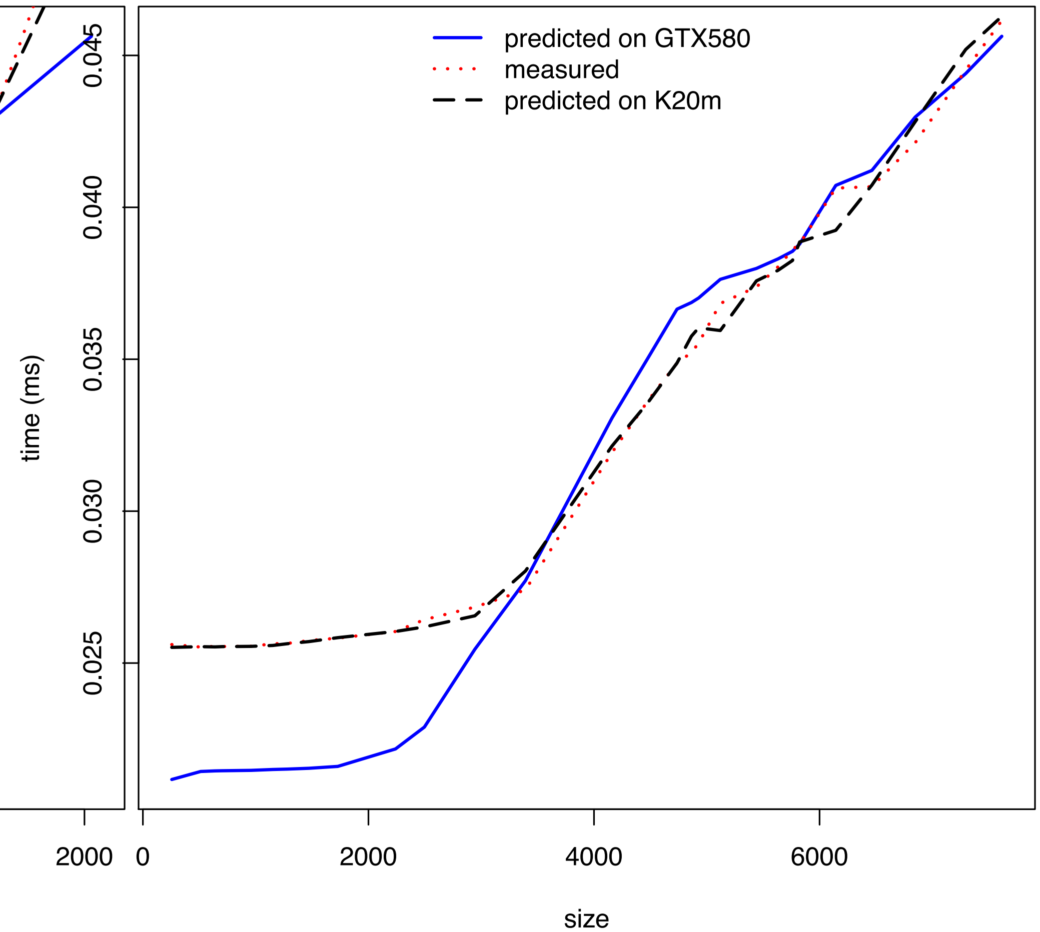


Results*

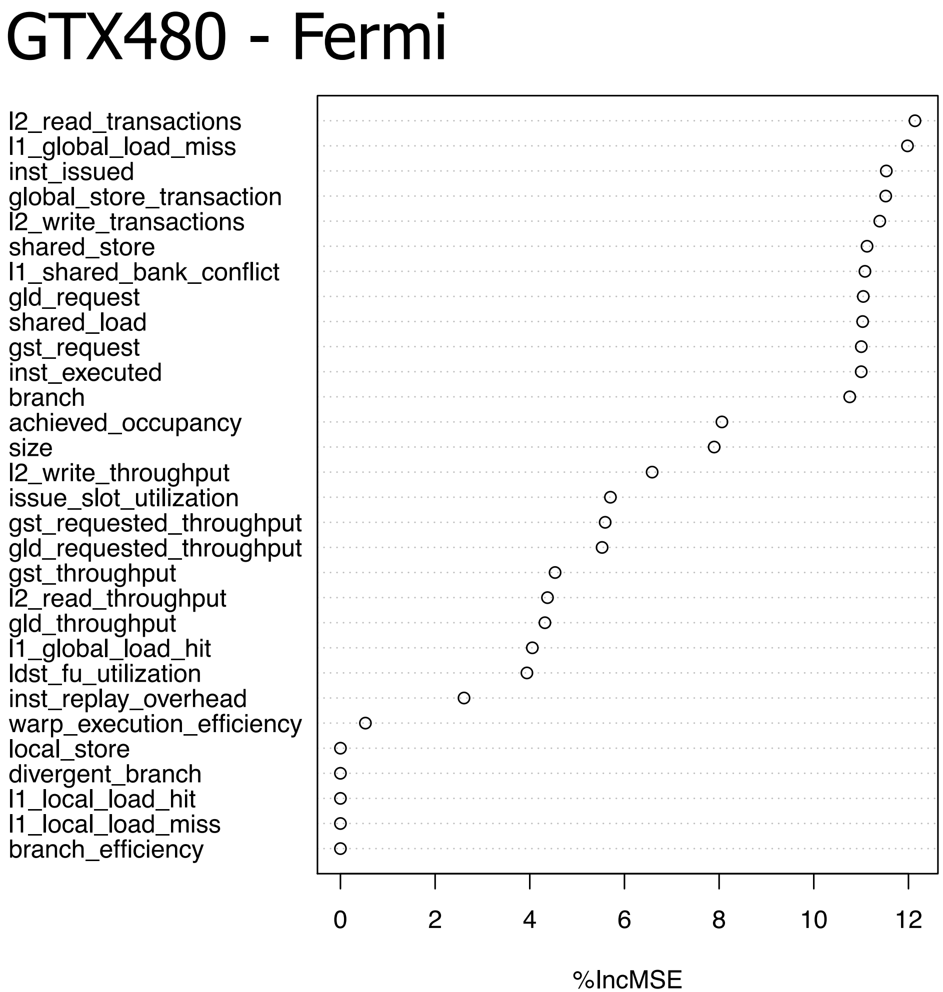
Matrix multiply

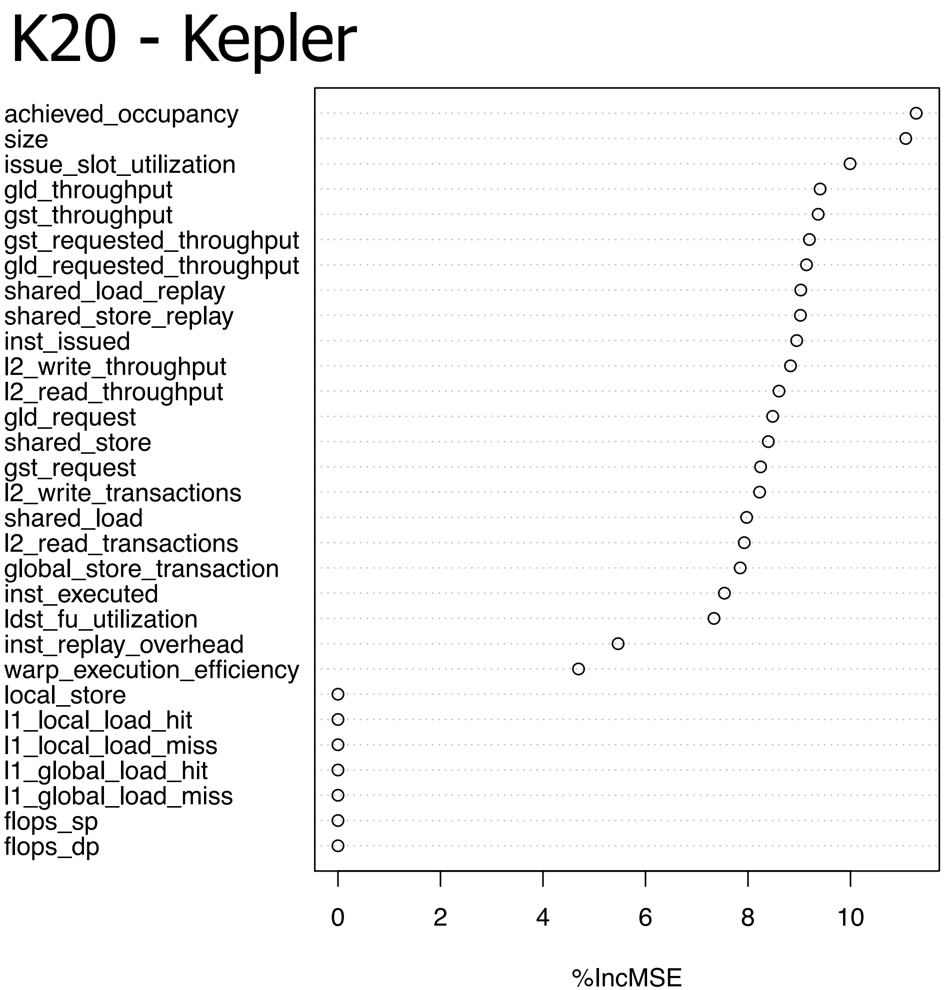


Needleman-Wunsch



Challenges: GPU generations change





Challenges

- Statistical approaches
  - Training data
  - Performance counters
- Analytical/modeling-based approaches
  - Hardware modeling
  - Application modeling
  - Calibration
- Putting these together … ?

Multi-kernel applications?

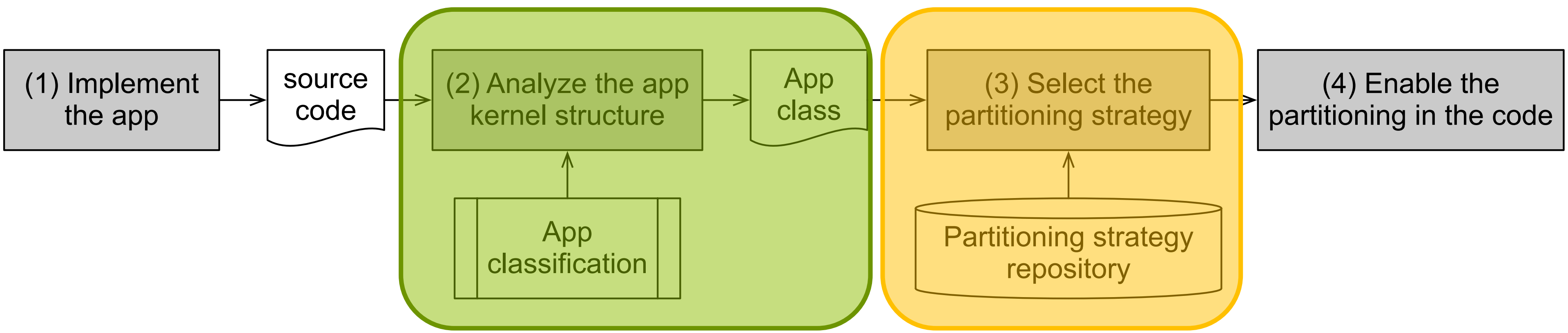
- Use dynamic partitioning: OmpSs, StarPU
  - Partition the kernels into chunks
  - Distribute chunks to *PUs
  - Keep data dependencies
  - (Often) lead to suboptimal performance
    - Scheduling policies and chunk size

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

Can we get the best of both worlds?

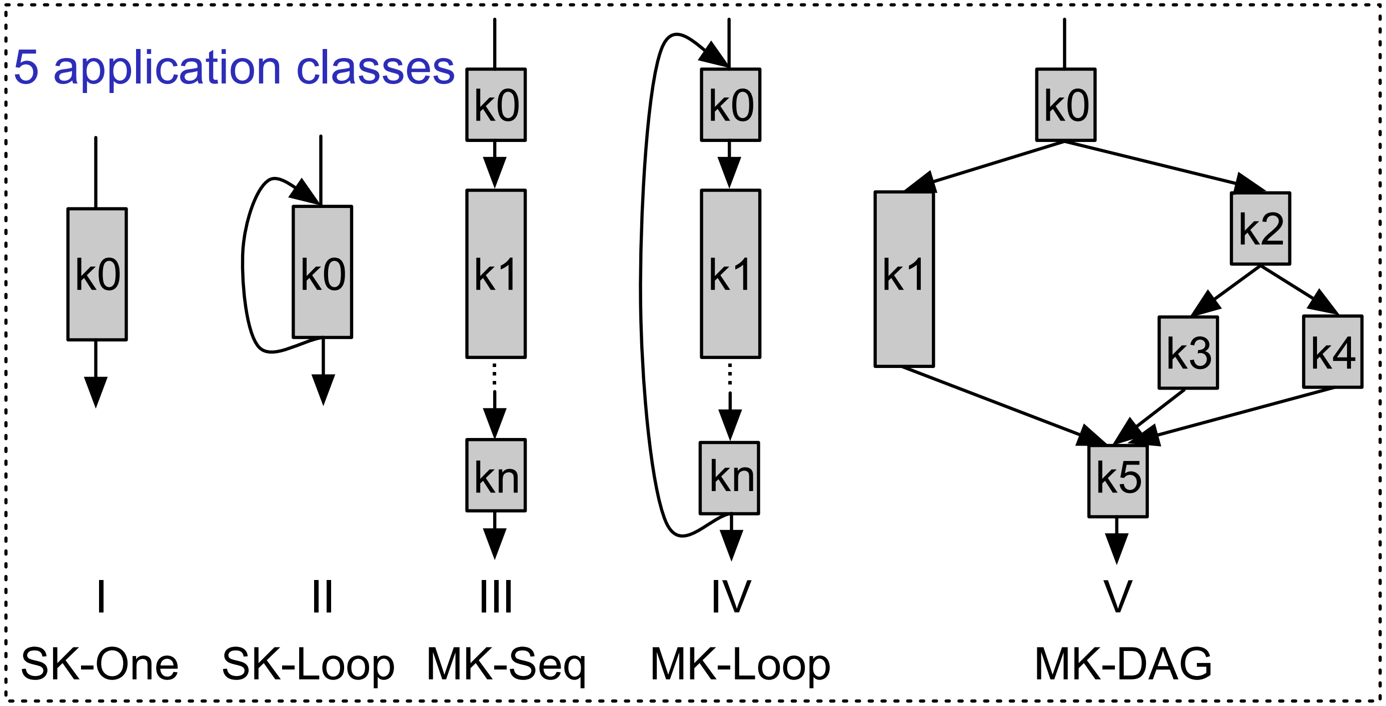
How to satisfy both requirements?

- 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

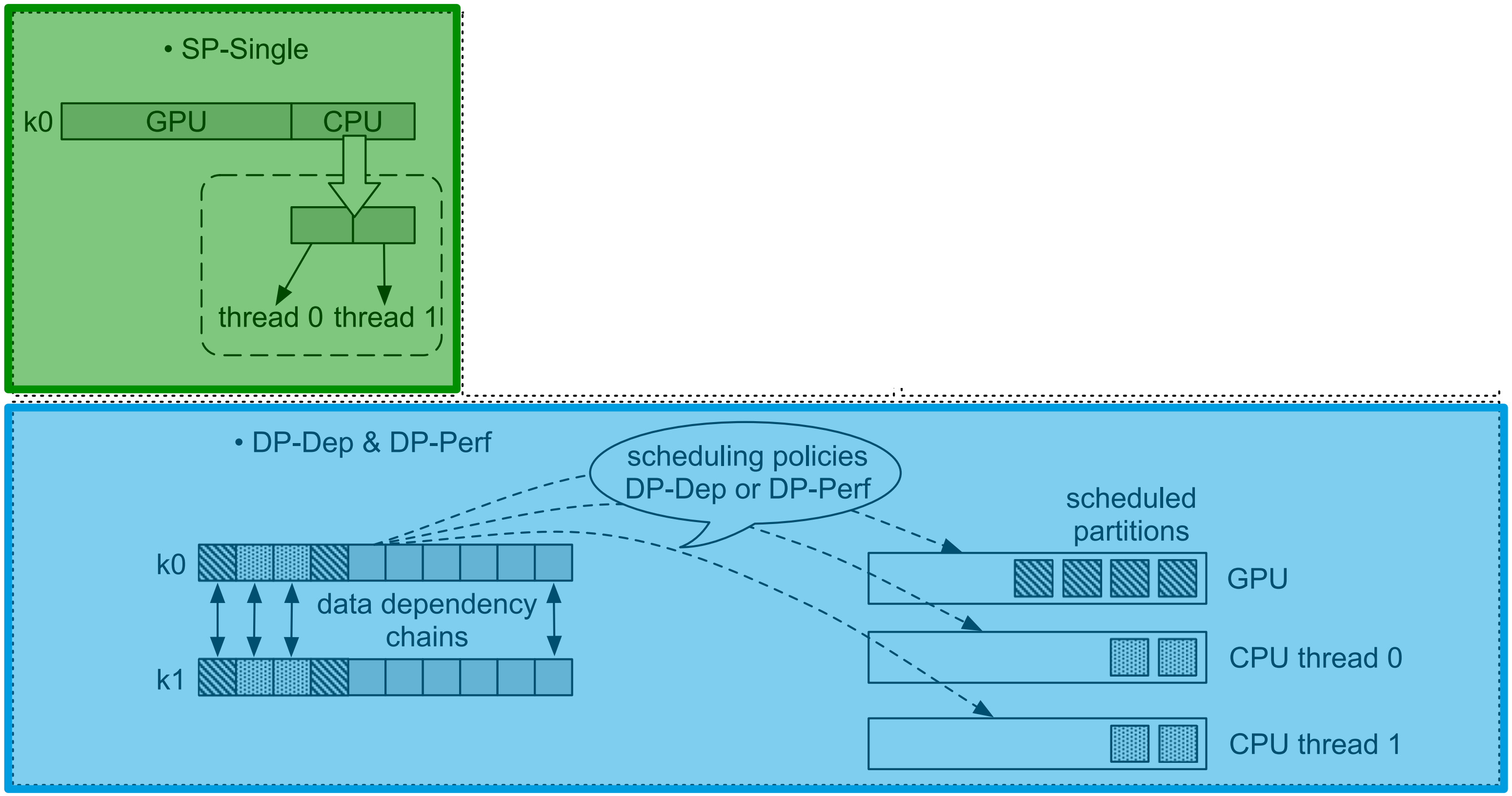


Implement/get the source code → Analyze application's class → Select a partitioning strategy → Enable the partitioning

Partitioning strategies

Static partitioning:
single-kernel applications

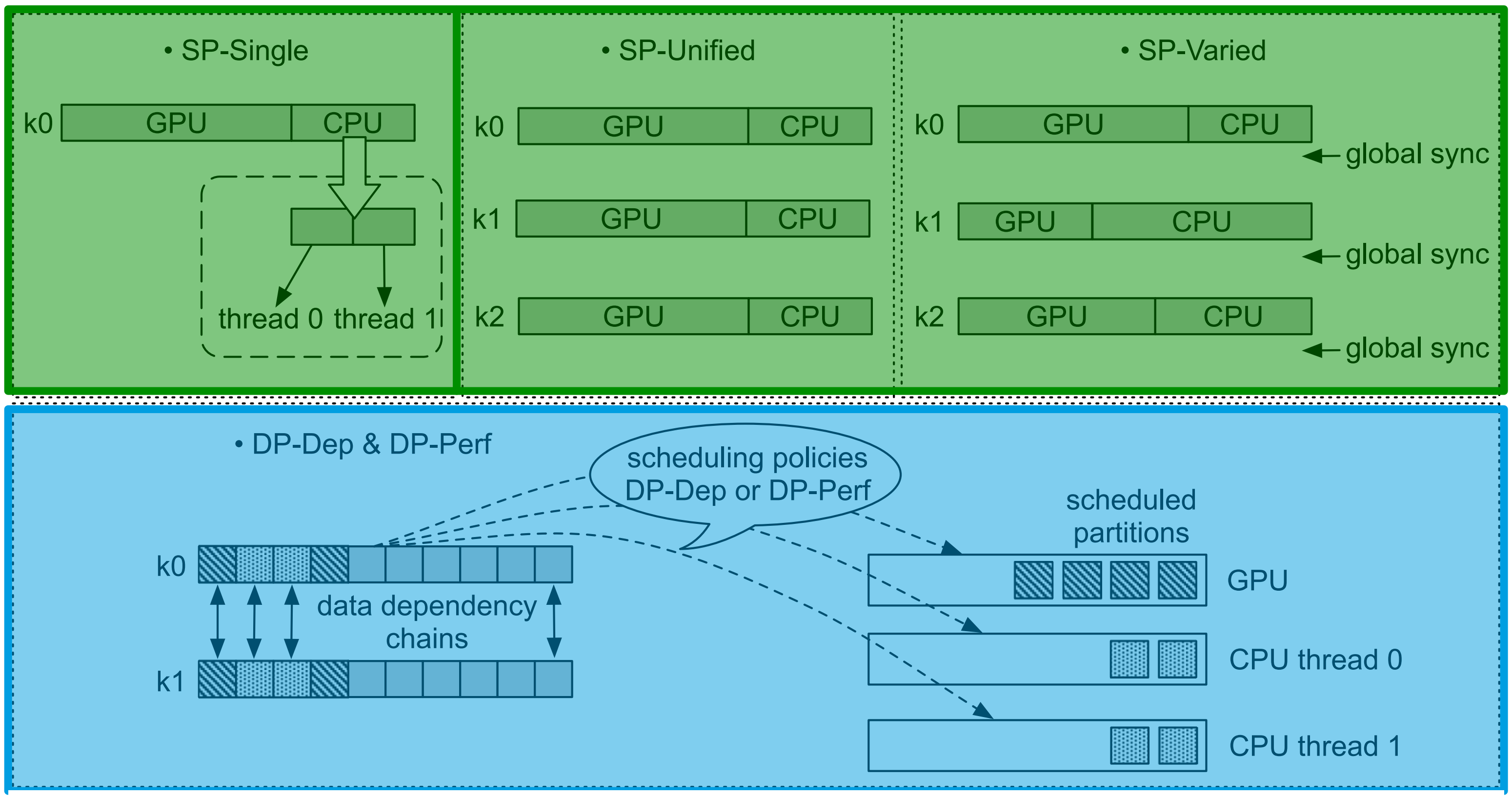
Dynamic partitioning:
multi-kernel applications



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

Partitioning strategies

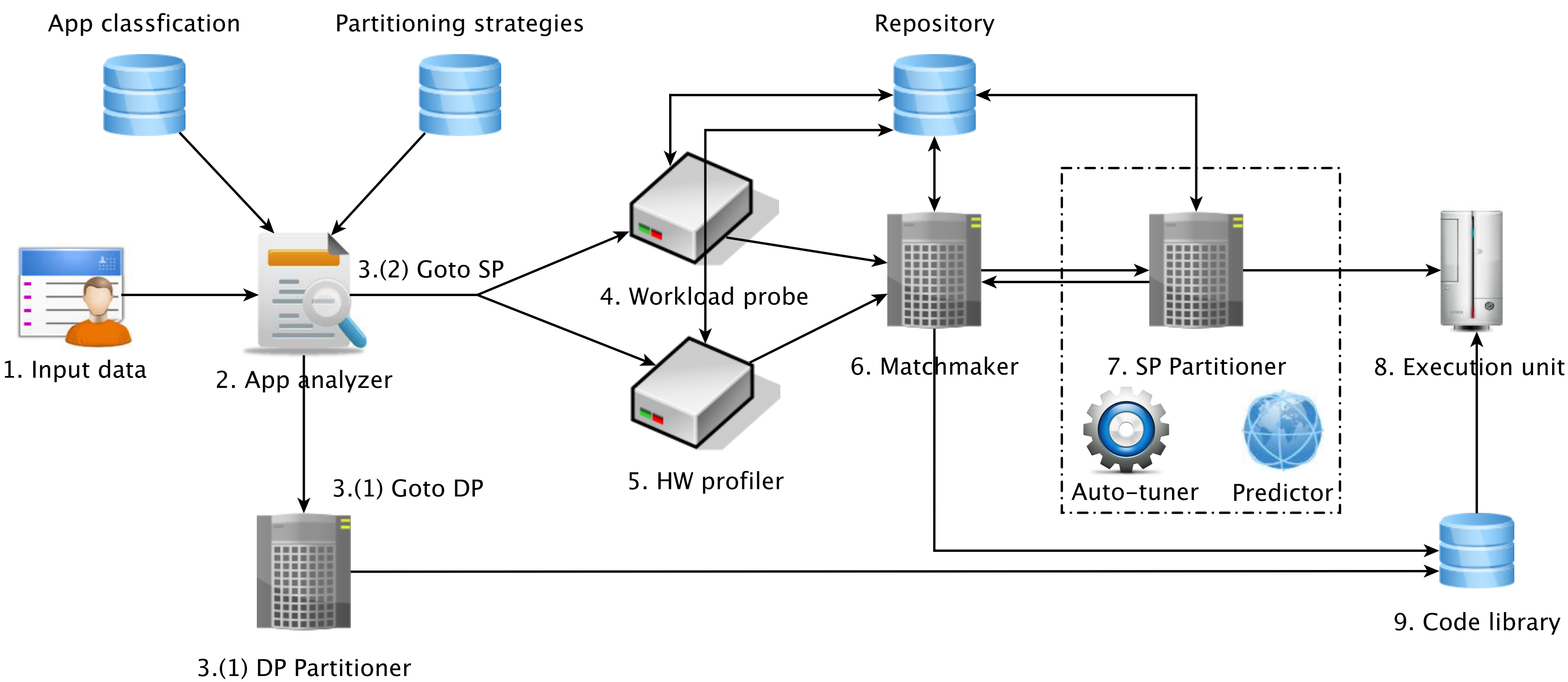
Static partitioning: in Glinda
single-kernel + multi-kernel applications



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

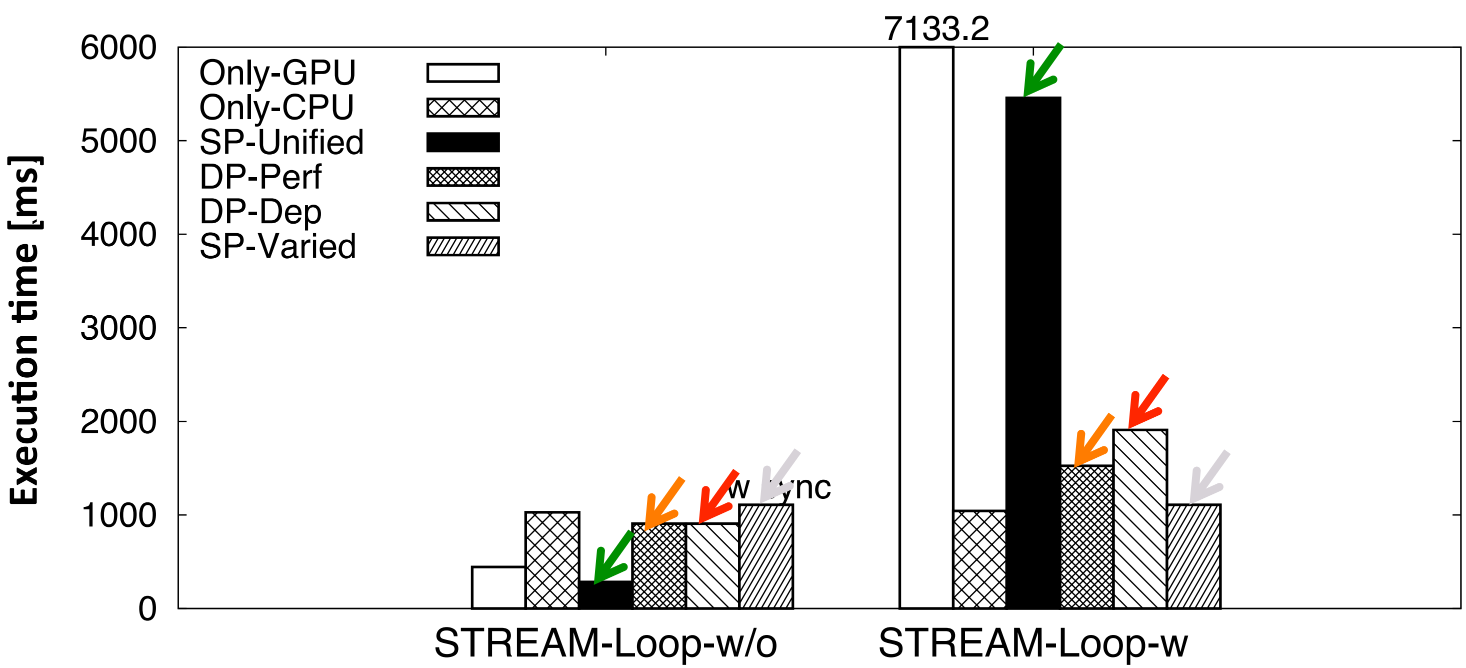
Implem
the source code
application's class
partitioning strategy
ble
the partitioning

Putting it all together: Glinda 2.0



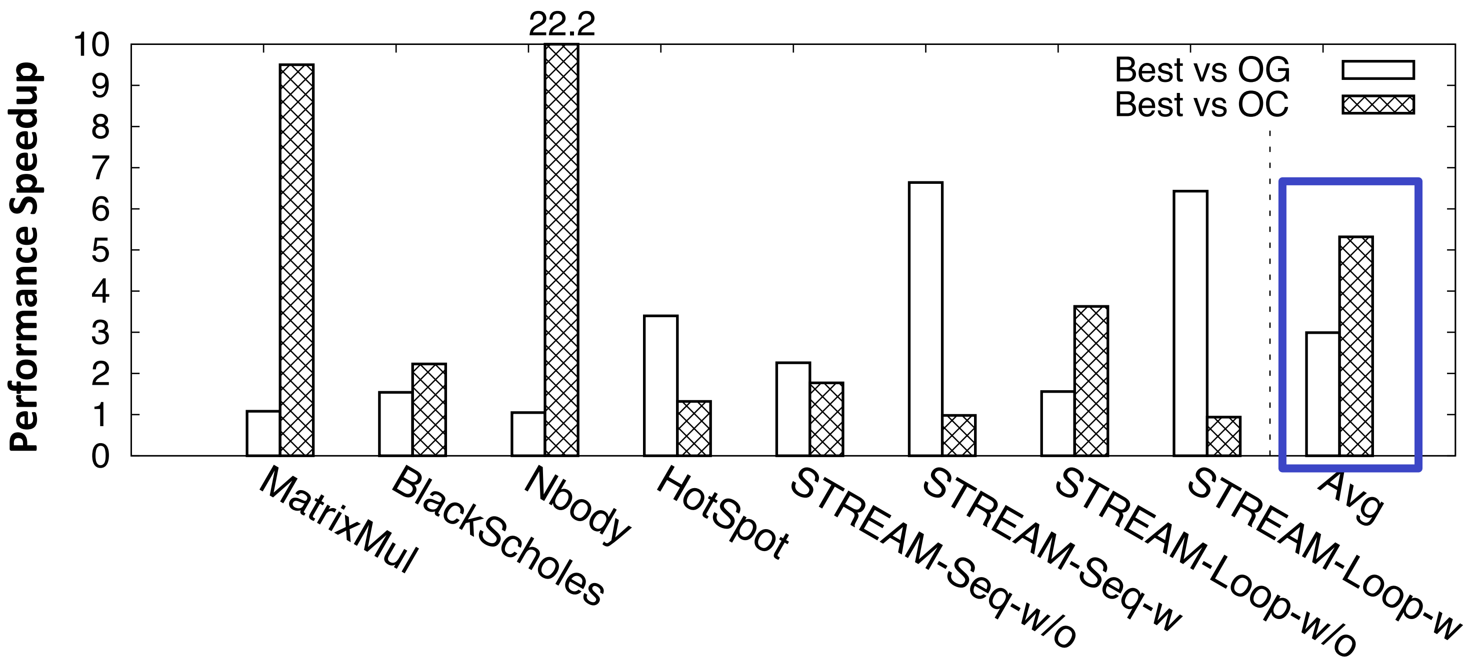
Results [4]

- 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



Results: performance gain

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



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

Dynamic partitioning: StarPU, OmpSS