Scheduling Divisible Loads on Heterogeneous Desktop Systems with Limited Memory

Aleksandar Ilić and Leonel Sousa

INESC-ID/
IST, TU Lisbon, Portugal

9th Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms - HeteroPar, Bordeaux, 2011

Commodity Computers = Heterogeneous Systems

- Multi-core General-Purpose Processors (CPUs)
- Many-core Graphic Processing Units (GPUs)
- Special accelerators, co-processors, FPGAs,…

=> Significant computing power

- Not yet completely explored for **collaborative computing**
- to use the available resources and improve performance/watt

Heterogeneity makes problems much more complex!

- Scheduling, performance modeling and load balancing
- Different programming models, languages and implementations

Outline

technology from seed

inesc id lisboa

Discretely Divisible Load (DDL) Processing

Heterogeneous (CPU+GPU) Desktop Systems

Performance Modeling and DDL Scheduling Algorithm

Case Study: 2D FFT Batch Execution

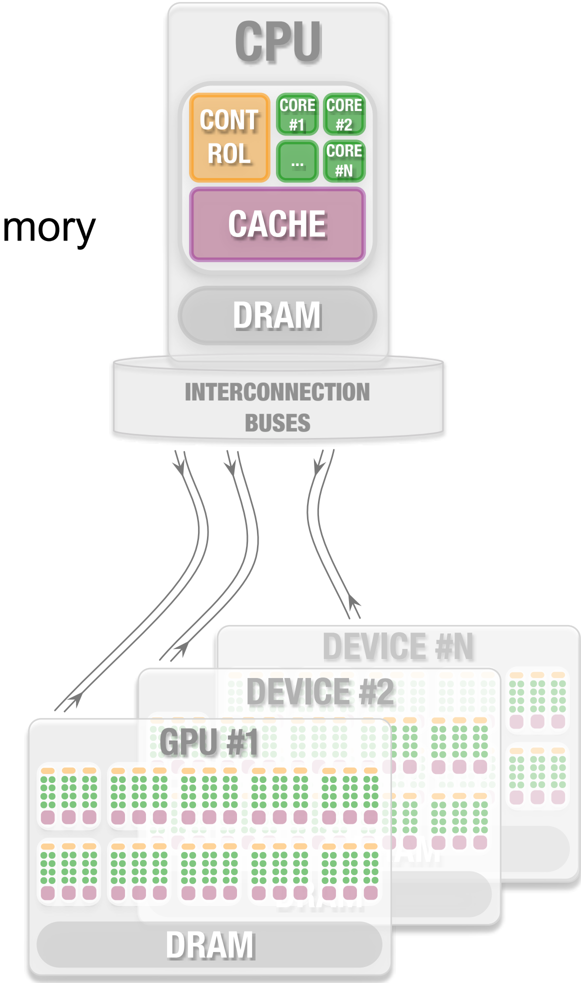
Conclusions and Future Work

Discretely Divisible Load Processing

- **Discretely Divisible Load (DDL) Applications**
  - Computations divisible into pieces of arbitrary sizes (integers)
  - Fractions independently processed in parallel with no precedence constraints
- **Applicable to a wide range of scientific problems**
  - Linear algebra, digital signal and image processing, database applications …
- **State of the art DDL approaches in Heterogeneous Distributed Computing**
  - Assume symmetric bandwidth and an one-port model for communication links
  - Limited memory: only the size of input load is considered; the exceeding load is simply redistributed among the nodes with available memory
  - Unrealistic: Computation/communication time is a linear/affine function of the #chunks

HETEROGENEOUS STAR NETWORK (MASTER-WORKER)

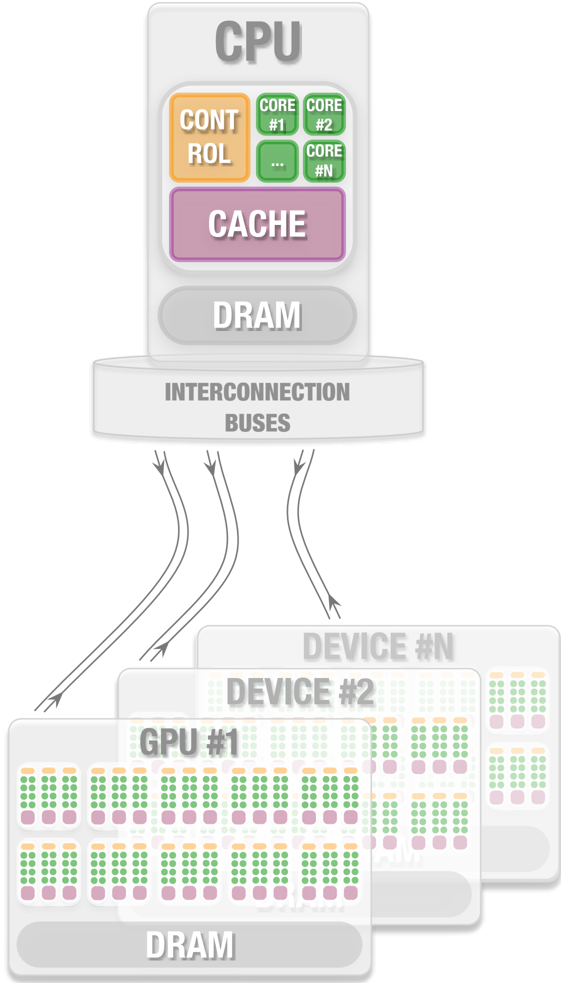
- **MULTI-CORE CPU** (Master)
  - Global execution controller; access the whole global memory
  - All cores employed for execution
- **INTERCONNECTION BUSES**
  - Bidirectional full-duplex asymmetric communication
  - Different concurrency levels
  - Potential execution bottleneck
- **DEVICES** (Distant workers)
  - Different architectures and programming models
  - Computation performed using local memories



Proposed Algorithm Outline (1)

2-STEP DIVISIBLE LOAD SCHEDULING

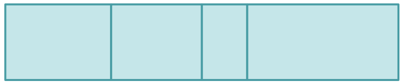
DDL

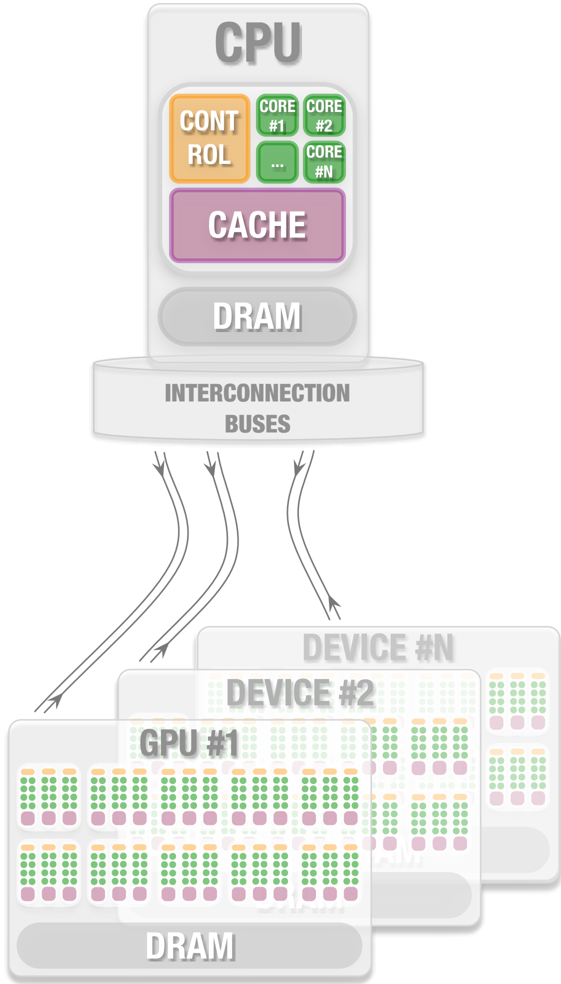


Proposed Algorithm Outline (2)

2-STEP DIVISIBLE LOAD SCHEDULING

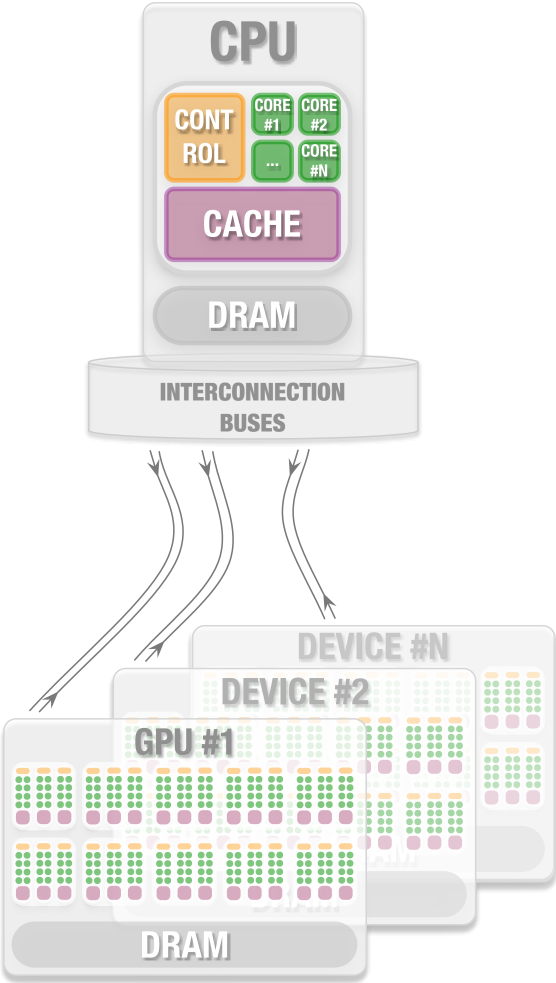
- **STEP 1 – SYSTEM-LEVEL LOAD BALANCING**
  - How many load units to send to each device?





2-STEP DIVISIBLE LOAD SCHEDULING

- **STEP 1 – SYSTEM-LEVEL LOAD BALANCING**
  - How many load units to send to each device?

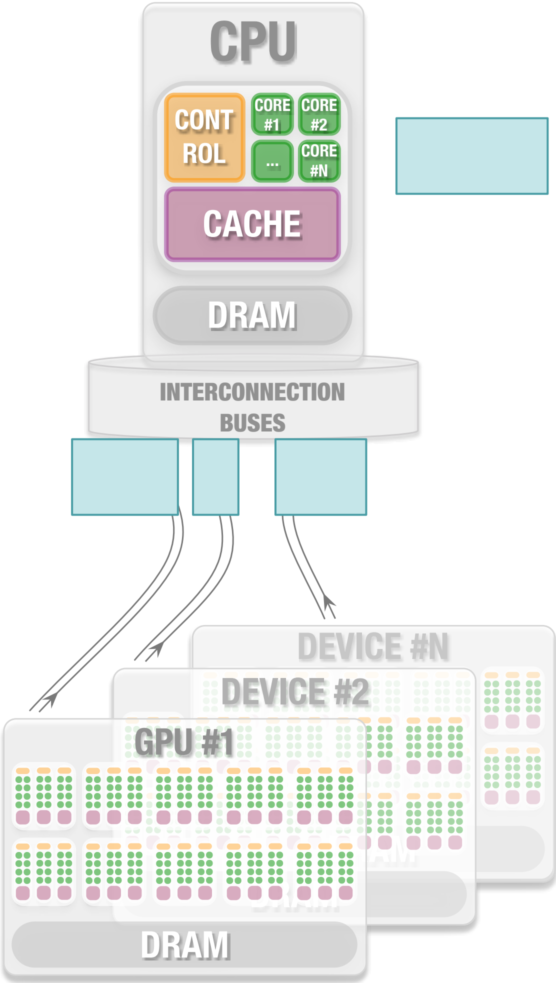


Proposed Algorithm Outline (4)

2-step Divisible Load Scheduling

- **Step 1 – System-Level Load Balancing**
  – How many load units to send to each device?

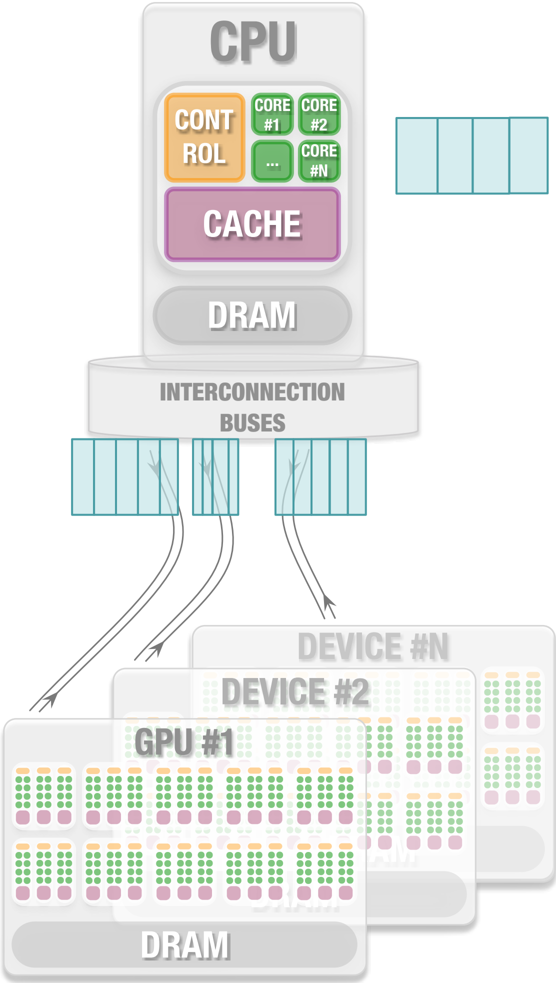
- **Step 2 – Device-level Load Scheduling**
  – How to sub-partition the device load to:
    - Reduce delays when distributing and retrieving
    - Overlap computation and communication
    - Efficiently use the bidirectional asymmetric bandwidth of buses
    - Respect the amount of supported concurrency
    - Fit into device limited memory



Proposed Algorithm Outline (5)

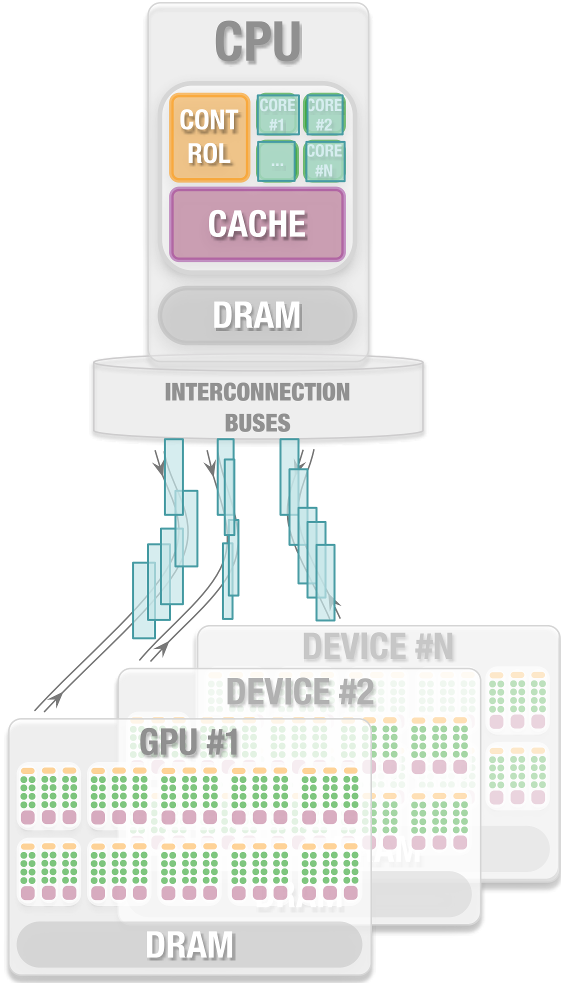
2-step Divisible Load Scheduling

- **Step 1 – System-Level Load Balancing**
  - How many load units to send to each device?
- **Step 2 – Device-level Load Scheduling**
  - How to sub-partition the device load to:
    - Reduce delays when distributing and retrieving
    - Overlap computation and communication
    - Efficiently use the bidirectional asymmetric bandwidth of buses
    - Respect the amount of supported concurrency
    - Fit into device limited memory



2-STEP DIVISIBLE LOAD SCHEDULING

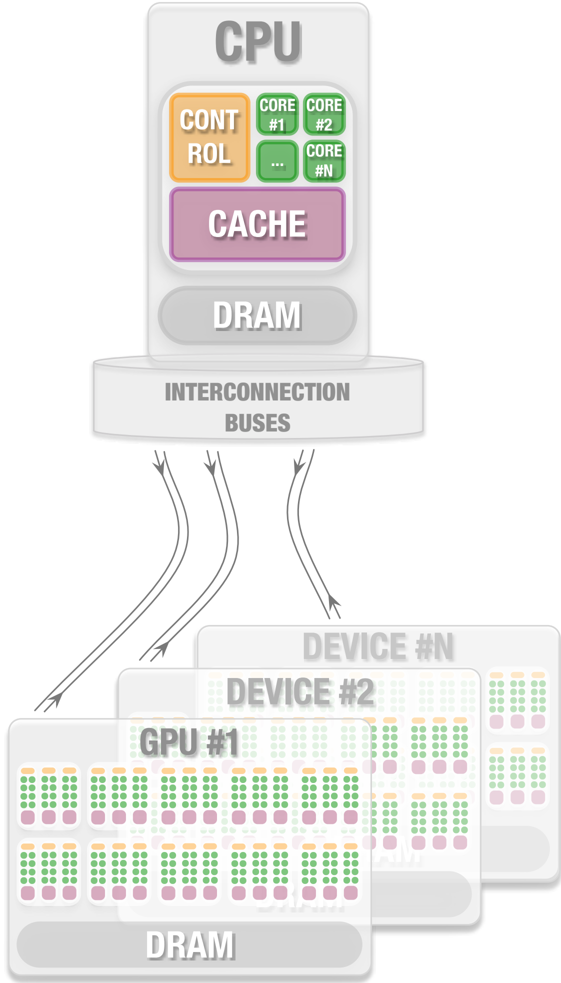
- **STEP 1 – SYSTEM-LEVEL LOAD BALANCING**
  - How many load units to send to each device?
- **STEP 2 – DEVICE-LEVEL LOAD SCHEDULING**
  - How to sub-partition the device load to:
    - Reduce delays when distributing and retrieving
    - Overlap computation and communication
    - Efficiently use the bidirectional asymmetric bandwidth of buses
    - Respect the amount of supported concurrency
    - Fit into device limited memory



Performance Modeling

FUNCTIONAL PERFORMANCE MODELS

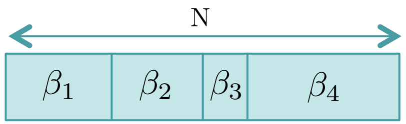
- Continuous performance functions (#chunks/time)
- Built from the real application execution
  - *No assumptions being made to ease modeling!*

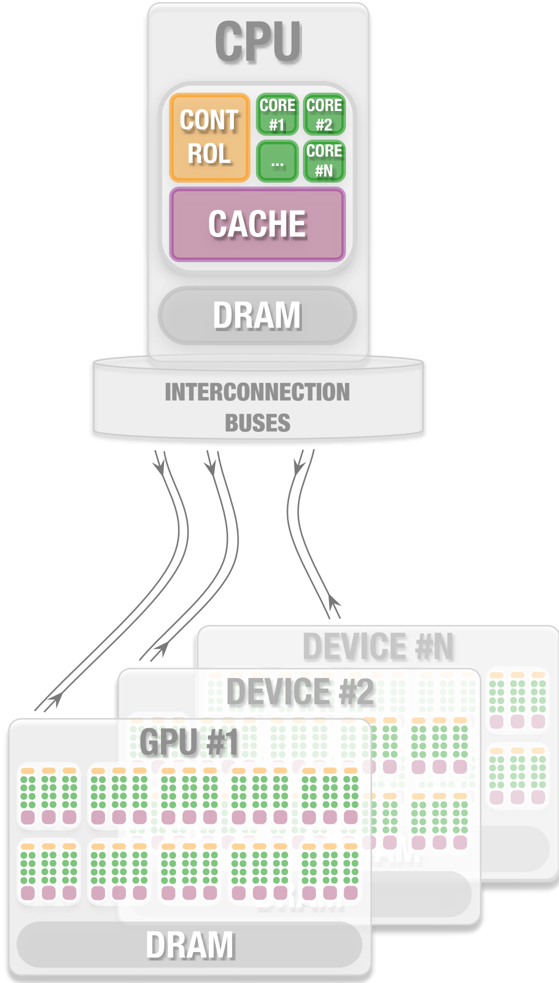


- **COMPUTATION PERFORMANCE MODELS - ψ_w**
  - For each master core and distant worker
- **FULL-DUPLEX COMMUNICATION BANDWIDTH - σ_ι, σ_o**
  - Bidirectional and asymmetric for each link
- **TOTAL PERFORMANCE ψ_τ OF EACH DEVICE**
  - Including computation and communication

Determination of Load Fractions (1)

- **STEP 1 – SYSTEM-LEVEL LOAD BALANCING**





  - The total load N is partitioned between devices
  - The optimal distribution lies on a straight line passing through the origin of coordinate system and intersecting communication-aware total performance curves (ψ_{τ}), such that*:

$$\frac{\beta_1}{\psi_{\tau_1}(\beta_1)} = \cdots = \frac{\beta_k}{\psi_{\tau_k}(\beta_k)} = \cdots = \frac{\beta_n}{\psi_{\tau_n}(\beta_n)}$$

$$\sum_{j=1}^{n} \beta_j = \mathrm{N}$$

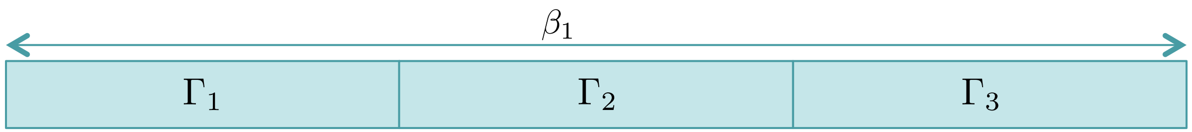
Lastovetsky, A., and R. Reddy, "Distributed Data Partitioning for Heterogeneous Processors Based on Partial Estimation of their Functional Performance Models", HeteroPar 2009, LNCS, vol. 6043, Springer, pp. 91-101, 2010.

Determination of Load Fractions (2)

technology from seed

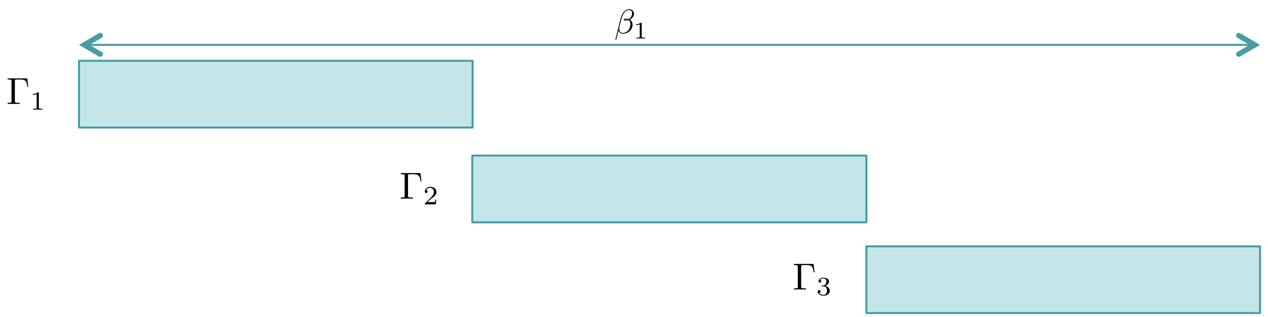
inesc id lisboa

- **STEP 2 – DEVICE-LEVEL SCHEDULING**
  - Per-device distributions β_j are allowed to exceed the device memory limits, b_j
    - Device-level **multi-installment** processing with **multi-distributions**
      - Γ_k sub-distributions with $\gamma_{k,l}$ sub-load fractions



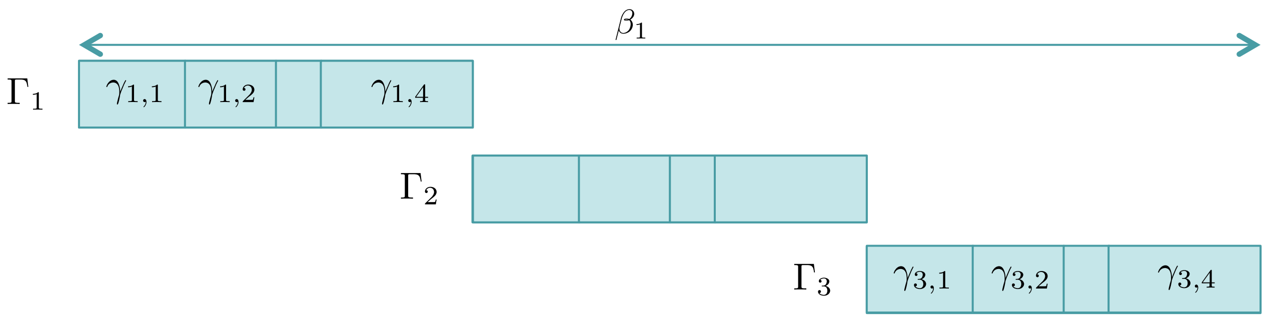
Determination of Load Fractions (3)

- **STEP 2 – DEVICE-LEVEL SCHEDULING**
  - Per-device distributions β_j are allowed to exceed the device memory limits, b_j
    - Device-level **multi-installment** processing with **multi-distributions**
      - Γ_k sub-distributions with $\gamma_{k,l}$ sub-load fractions



Determination of Load Fractions (4)

- **STEP 2 – DEVICE-LEVEL SCHEDULING**
  - Per-device distributions β_j are allowed to exceed the device memory limits, b_j
    - Device-level **multi-installment** processing with **multi-distributions**
      - Γ_k sub-distributions with $\gamma_{k,l}$ sub-load fractions



Determination of Load Fractions (5)
- Limited Memory -

technology
from seed

inesc id lisboa

- **STEP 2 – DEVICE-LEVEL SCHEDULING**
  - Per-device distributions β_j are allowed to exceed the device memory limits, b_j
    - Device-level **multi-installment** processing with **multi-distributions**
      - Γ_k sub-distributions with $\gamma_{k,l}$ sub-load fractions
  - Application **memory requirements** are modeled with three functions of load size:
    - *Input memory requirements*, $\mu_\iota(x)$
    - *Output memory requirements*, $\mu_o(x)$
    - **Execution memory requirements**, $\mu_w(x, P)$
      - Different implementations of the same problem might have different memory requirements!

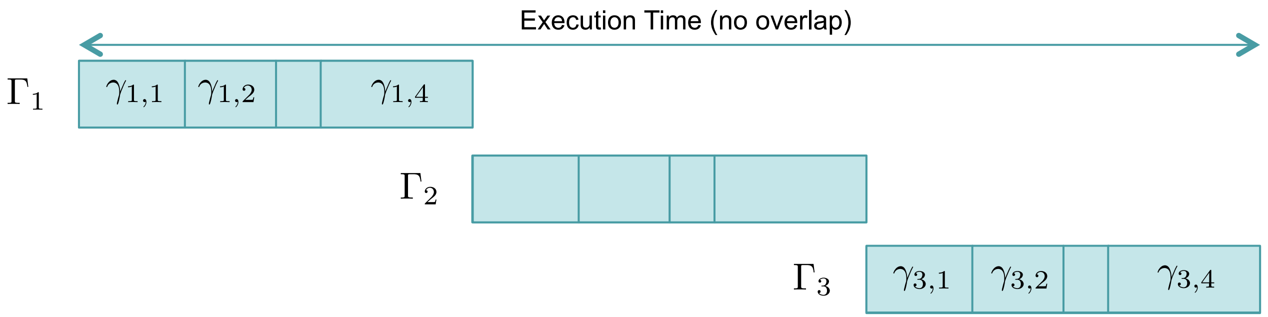
⇒ Each Γ_k *sub-distribution* may request *the* whole amount of memory, such that:

$$\sum_{l=1}^{|\Gamma_k|} (\mu_\iota(\gamma_{k,l}) + \mu_w(\gamma_{k,l}, p_j) + \mu_o(\gamma_{k,l})) \leq b_j$$

$$\sum_{k=1}^{|\Gamma|} \sum_{l=1}^{|\Gamma_k|} \gamma_{k,l} = \beta_j$$

Determination of Load Fractions (6)
- Computation/Communication Overlapping -

- **STEP 2 – DEVICE-LEVEL SCHEDULING**
  - For each Γ_k sub-distribution, $\gamma_{k,l}$ sizes are carefully chosen to allow as best as possible ***overlapping of computation and communication*** between subsequent sub-fractions

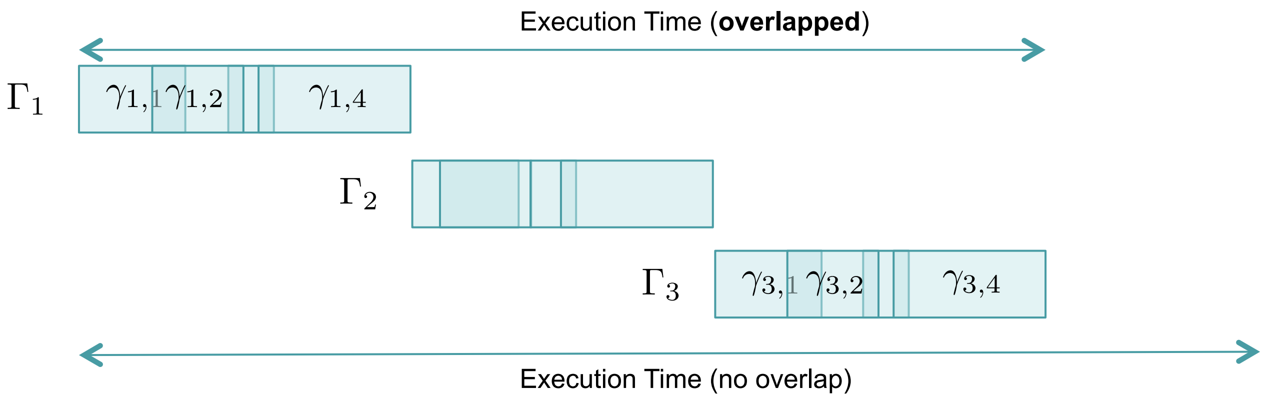


Determination of Load Fractions (7)
- Computation/Communication Overlapping -

technology
from seed

inesc id lisboa

- **STEP 2 – DEVICE-LEVEL SCHEDULING**
  - For each Γ_k sub-distribution, $\gamma_{k,l}$ sizes are carefully chosen to allow as best as possible ***overlapping of computation and communication*** between subsequent sub-fractions

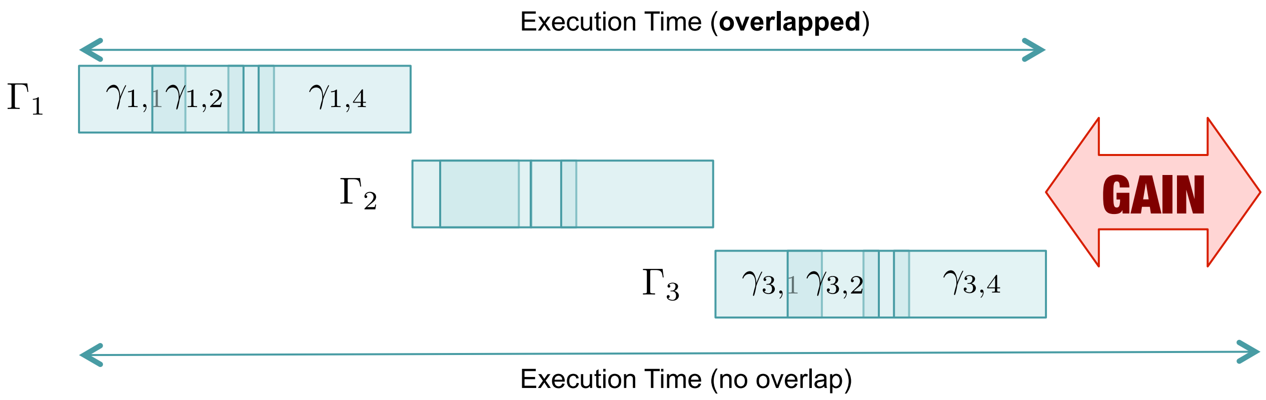


Determination of Load Fractions (8)
- Computation/Communication Overlapping -

technology from seed

inesc id lisboa

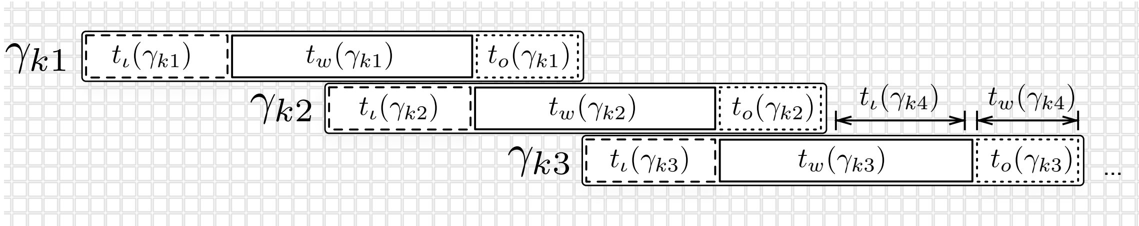
- **STEP 2 – DEVICE-LEVEL SCHEDULING**
  - For each Γ_k sub-distribution, $\gamma_{k,l}$ sizes are carefully chosen to allow as best as possible ***overlapping of computation and communication*** between subsequent sub-fractions



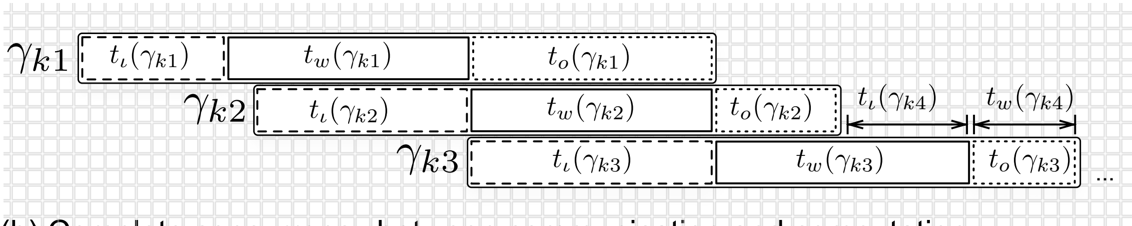
  - Gain size depends on how well the chunks are overlapped!

Determination of Load Fractions (9)
- Computation/Communication Overlapping -

- **STEP 2 – DEVICE-LEVEL SCHEDULING**
  - For each Γ_k sub-distribution, $\gamma_{k,l}$ sizes are carefully chosen to allow as best as possible ***overlapping of computation and communication*** between subsequent sub-fractions
  - The decisions are made according to the **amount of overlapping concurrency** supported by the device:



(a) Overlap of a single communication with computation at the time



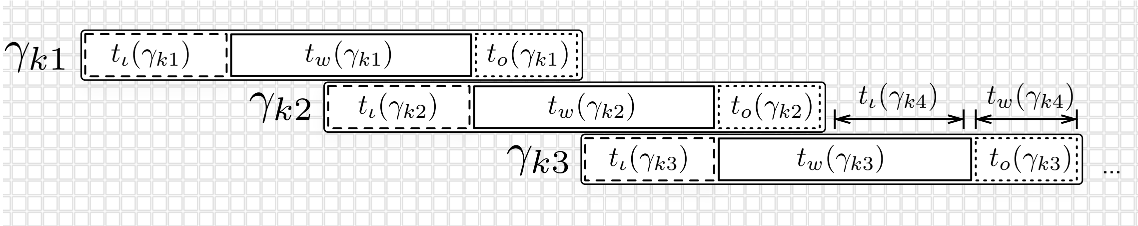
(b) Complete concurrency between communication and computation

Determination of Load Fractions (10)
- Device-level Scheduling Algorithm -

technology from seed

inesc id lisboa

- **Step 2 – Device-Level Scheduling Algorithm***

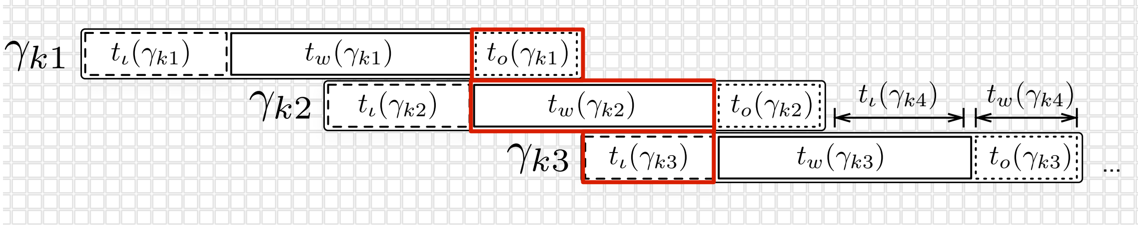


  – According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)

** Ilić, A., and Sousa, L., "Algorithm For Divisible Load Scheduling on Heterogeneous Systems with Realistic Performance Models", Tech. rep., INESC-ID (May 2011)*

Determination of Load Fractions (11)
- Device-level Scheduling Algorithm -

- **Step 2 – Device-Level Scheduling Algorithm***

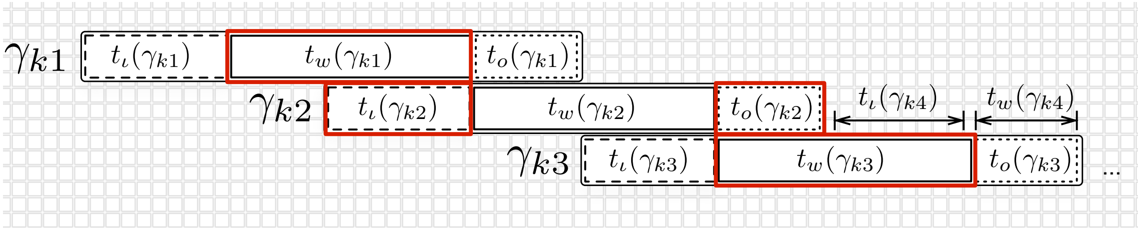


– According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)
  - **Step 2-I.** Determination of the initial optimal distribution with three load fractions.

* Ilić, A., and Sousa, L., "Algorithm For Divisible Load Scheduling on Heterogeneous Systems with Realistic Performance Models", Tech. rep., INESC-ID (May 2011)

Determination of Load Fractions (12)
- Device-level Scheduling Algorithm -

- **Step 2 – Device-Level Scheduling Algorithm***



  – According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)
    - **Step 2-I.** Determination of the initial optimal distribution with three load fractions.
    - **Step 2-II.** Generate additional three-fraction distributions.

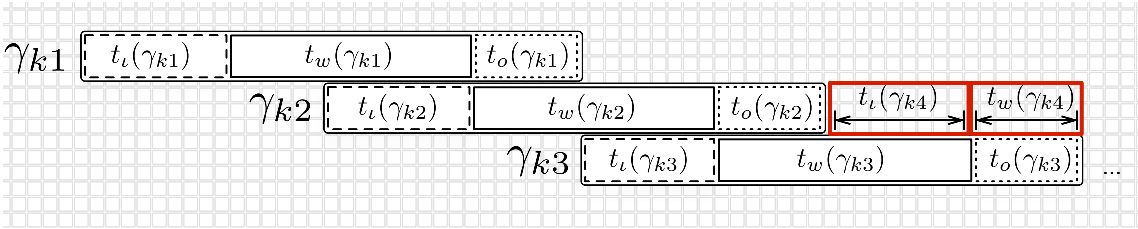
* *Ilić, A., and Sousa, L., "Algorithm For Divisible Load Scheduling on Heterogeneous Systems with Realistic Performance Models", Tech. rep., INESC-ID (May 2011)*

Determination of Load Fractions (13)
- Device-level Scheduling Algorithm -

technology
from seed

inesc id lisboa

- **Step 2 – Device-Level Scheduling Algorithm***



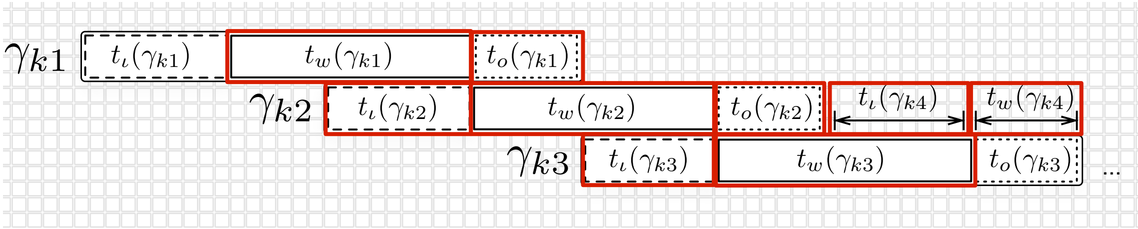
  – According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)

    - **Step 2-I.** Determination of the initial optimal distribution with three load fractions.
    - **Step 2-II.** Generate additional three-fraction distributions.
    - **Step 2-III.** Insert additional load fractions into existing sub-distributions (iterative).

** Ilić, A., and Sousa, L., "Algorithm For Divisible Load Scheduling on Heterogeneous Systems with Realistic Performance Models", Tech. rep., INESC-ID (May 2011)*

Determination of Load Fractions (14)
- Device-level Scheduling Algorithm -

- **Step 2 – Device-Level Scheduling Algorithm***



– According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)

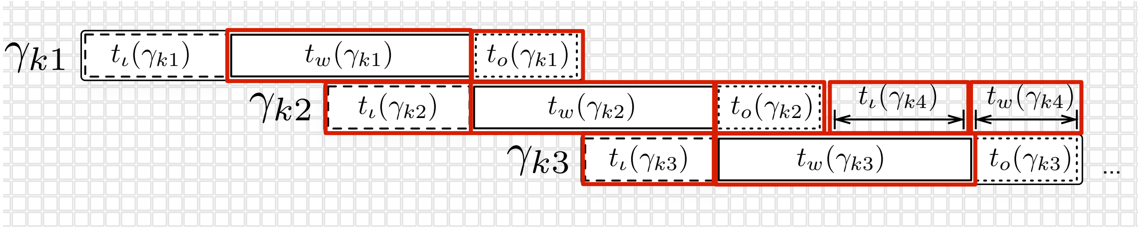
- **Step 2-I.** Determination of the initial optimal distribution with three load fractions.
- **Step 2-II.** Generate additional three-fraction distributions.
- **Step 2-III.** Insert additional load fractions into existing sub-distributions (iterative).
- **Step 2-IV.** Generate new sub-distributions by restarting.

** Ilić, A., and Sousa, L., "Algorithm For Divisible Load Scheduling on Heterogeneous Systems with Realistic Performance Models", Tech. rep., INESC-ID (May 2011)*

Determination of Load Fractions (15)
- DEVICE-LEVEL SCHEDULING ALGORITHM -

technology
from seed

- **STEP 2 – DEVICE-LEVEL SCHEDULING ALGORITHM***



  – According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)
    - **Step 2-I.** Determination of the initial optimal distribution with three load fractions.
    - **Step 2-II.** Generate additional three-fraction distributions.
    - **Step 2-III.** Insert additional load fractions into existing sub-distributions (iterative).
    - **Step 2-IV.** Generate new sub-distributions by restarting.
    - **Step 2-V.** Expand all sub-distributions.

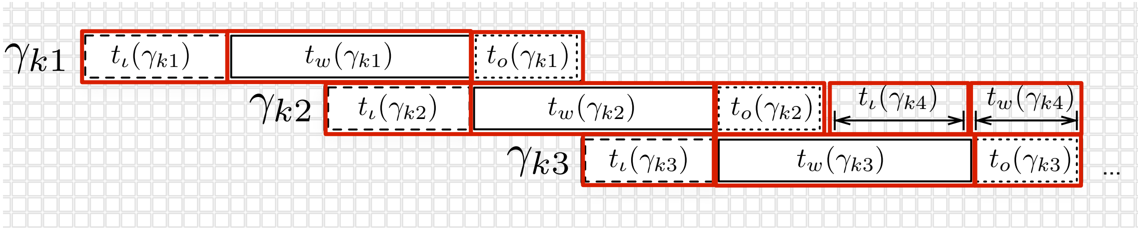
** Ilić, A., and Sousa, L., "Algorithm For Divisible Load Scheduling on Heterogeneous Systems with Realistic Performance Models", Tech. rep., INESC-ID (May 2011)*

Determination of Load Fractions (16)
- Device-level Scheduling Algorithm -

technology
from seed

inesc id lisboa

- **Step 2 – Device-Level Scheduling Algorithm***



– According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)
  - **Step 2-I.** Determination of the initial optimal distribution with three load fractions.
  - **Step 2-II.** Generate additional three-fraction distributions.
  - **Step 2-III.** Insert additional load fractions into existing sub-distributions (iterative).
  - **Step 2-IV.** Generate new sub-distributions by restarting.
  - **Step 2-V.** Expand all sub-distributions.
  - **Step 2-VI.** Select the distribution with maximum relative performance.

** Ilić, A., and Sousa, L., "Algorithm For Divisible Load Scheduling on Heterogeneous Systems with Realistic Performance Models", Tech. rep., INESC-ID (May 2011)*

Case Study: 2D FFT Batch

technology
from seed

inesc id lisboa

Double Floating Point Complex 2D FFT Batch Execution

- Size: 256 times 512 × 512; divisible in the first dimension
- The optimal vendor-provided FFT implementations are used
  - NVIDIA's CUFFT 3.2 for the GPU and Intel MKL 10.3 for the CPU

Heterogeneous CPU+GPU Desktop System

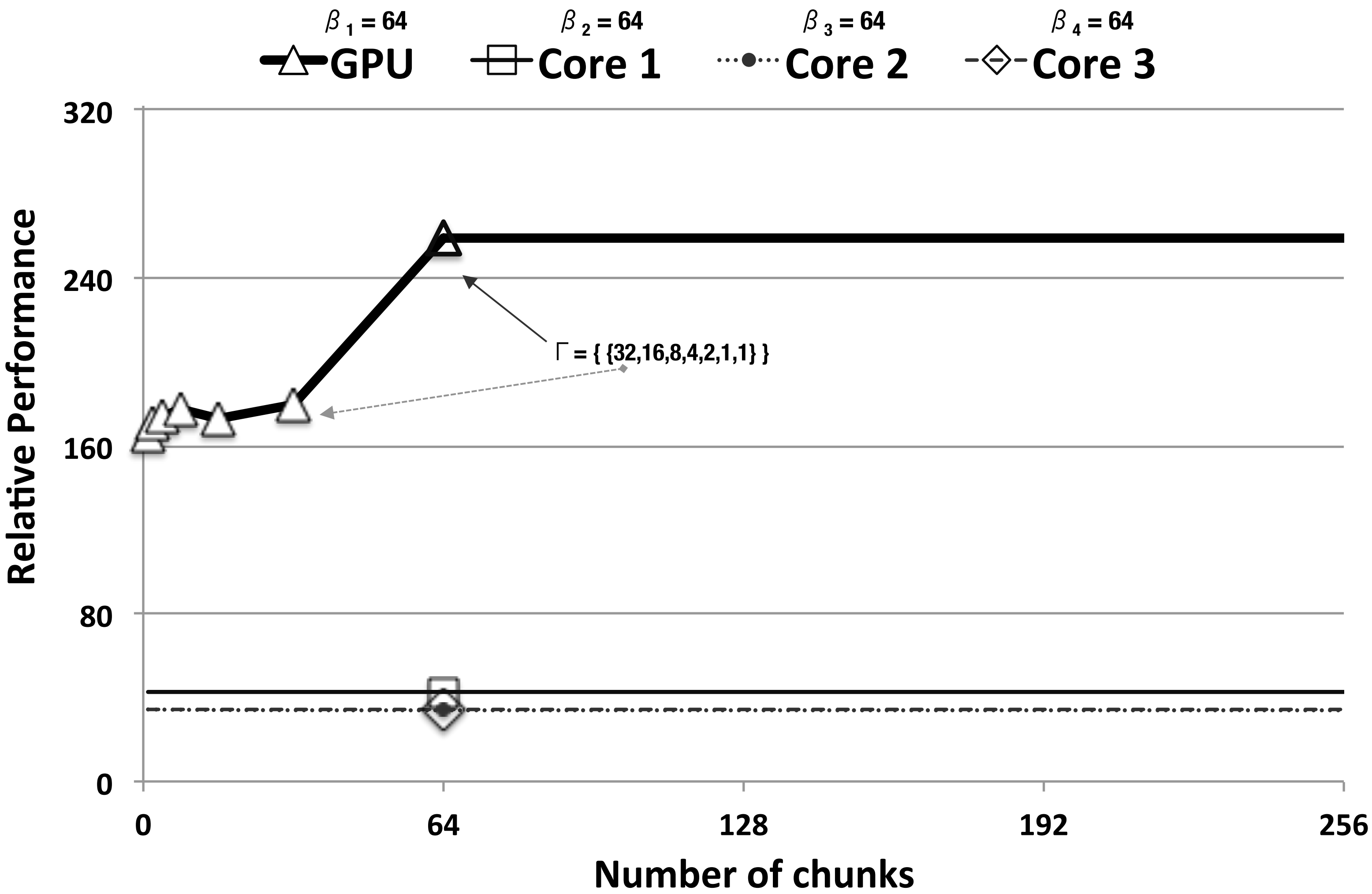
Experimental Setup	CPU	GPU
	Intel Core 2 Quad	nVIDIA GeForce 285GTX
Speed/Core (GHz)	2.83	1.476
Global Memory (MB)	4096	1024

Iterative Procedure for Online Performance Modeling

- Performance estimation of all heterogeneous devices during the execution
  - No prior knowledge on the performance of an application is available on any of the devices
- **Initially,** the load is distributed among devices using factoring-by-two strategy
  - Limited Memory: Factoring-by-two partitioning of the largest loads into new sub-distributions until satisfying the memory limitations
- **In each following iteration,** the load is distributed using the presented approach

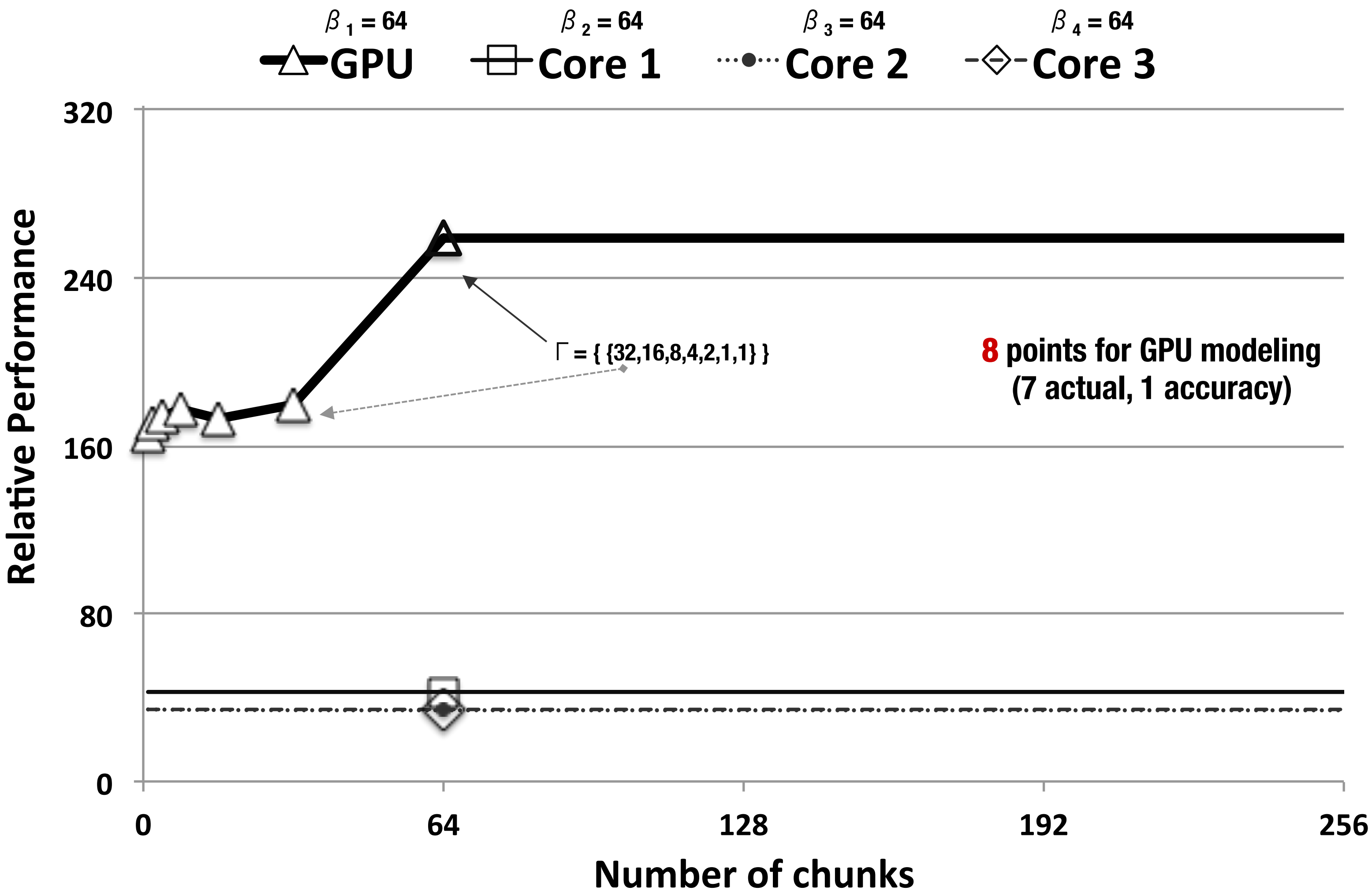
Case Study: 2D FFT Batch

Iteration 1: Factoring-by-Two Strategy



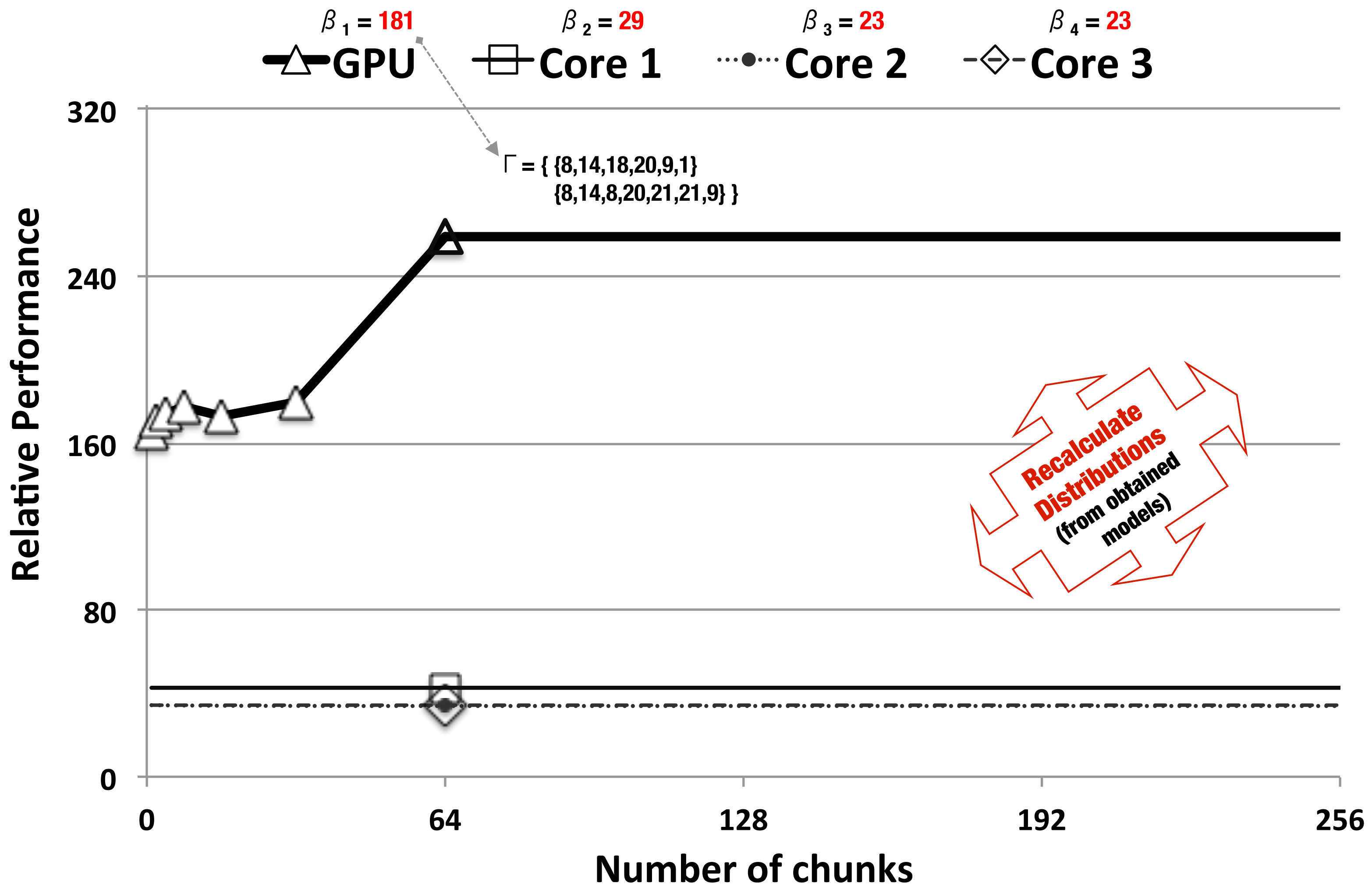
Case Study: 2D FFT Batch

Iteration 1: Factoring-by-Two Strategy

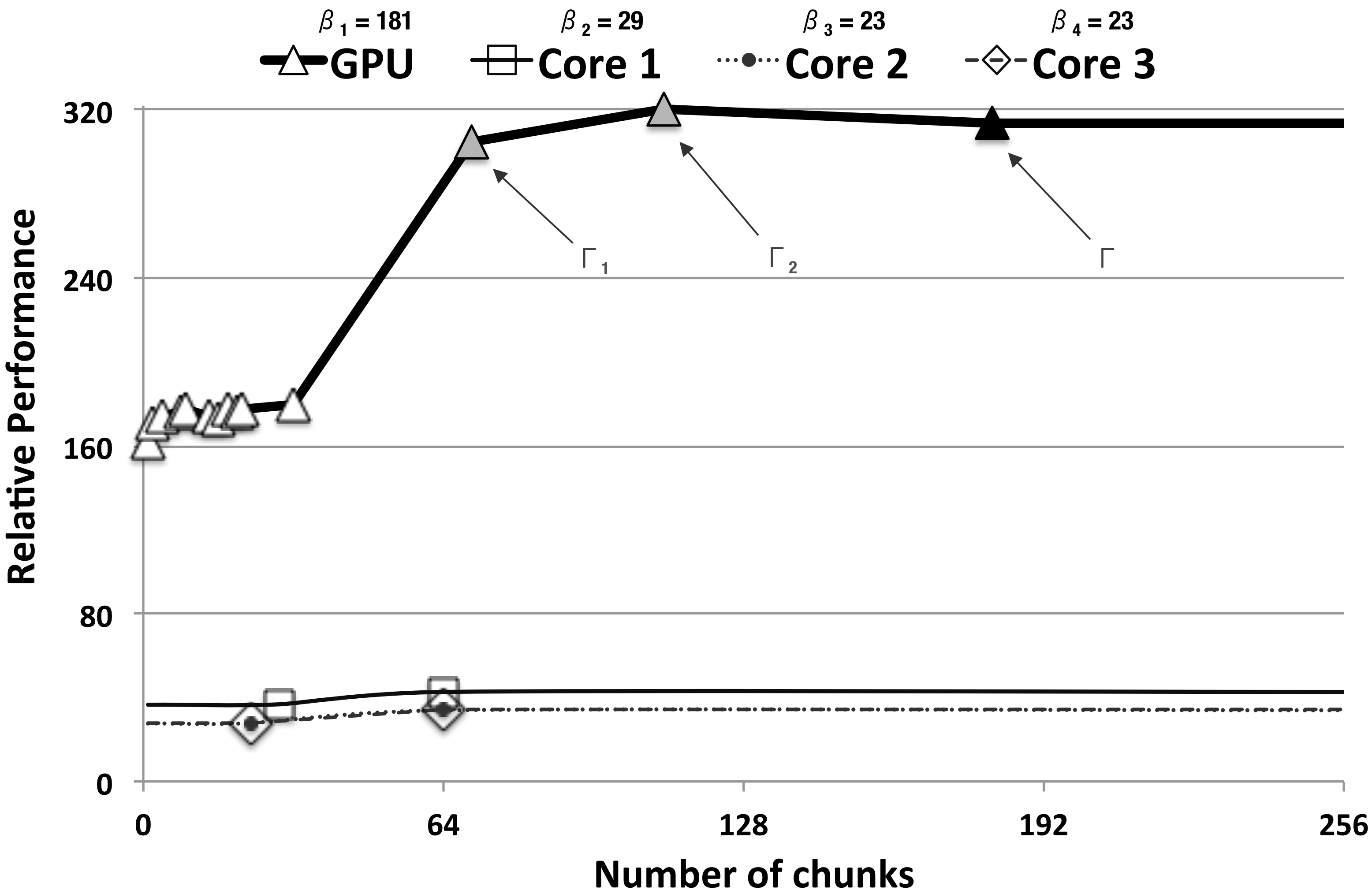


Case Study: 2D FFT Batch

ITERATION 1: PROPOSED ALGORITHM



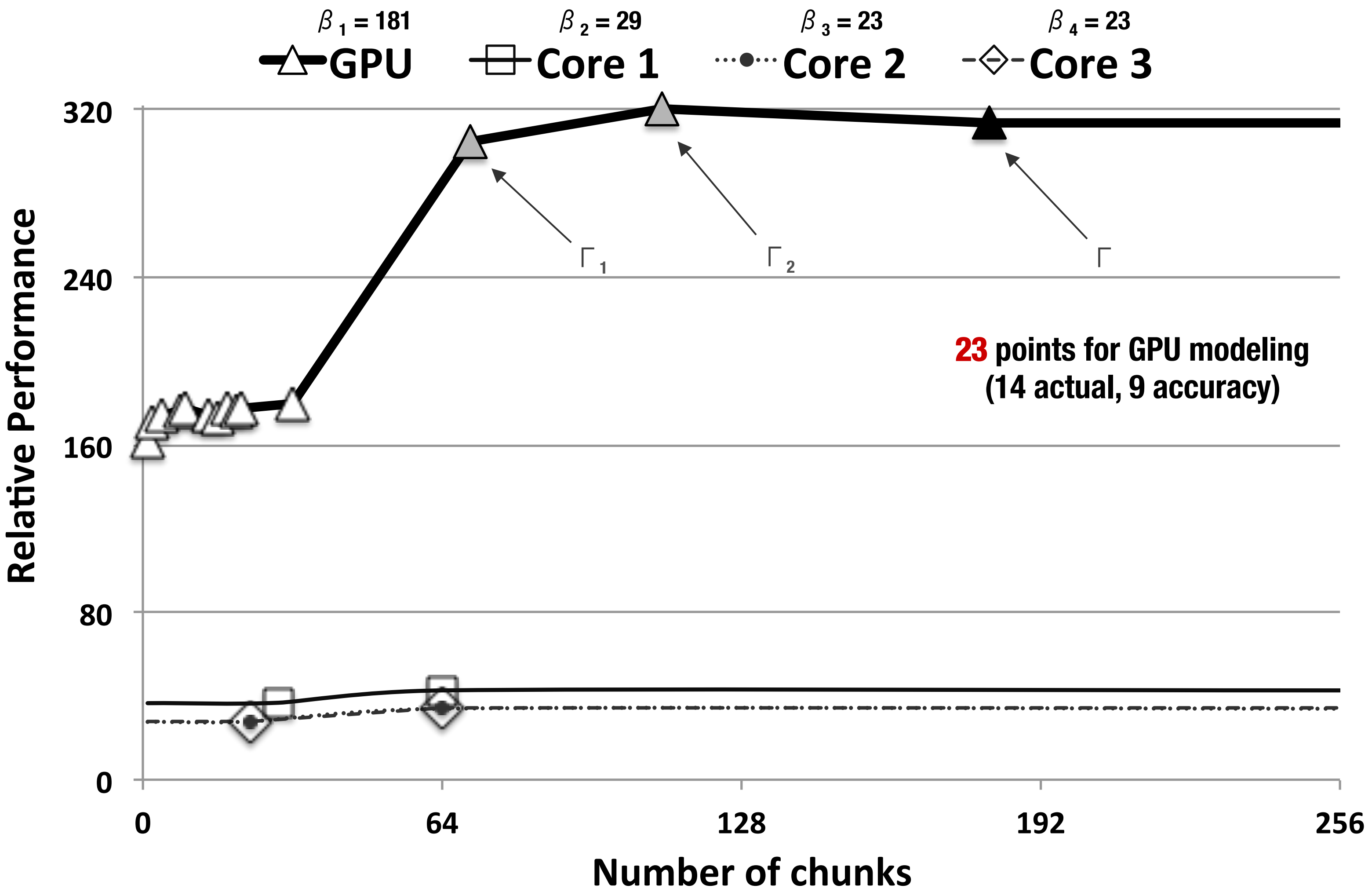
Case Study: 2D FFT Batch
Iteration 2: Performance models



Case Study: 2D FFT Batch

Iteration 2: Modeling efficiency

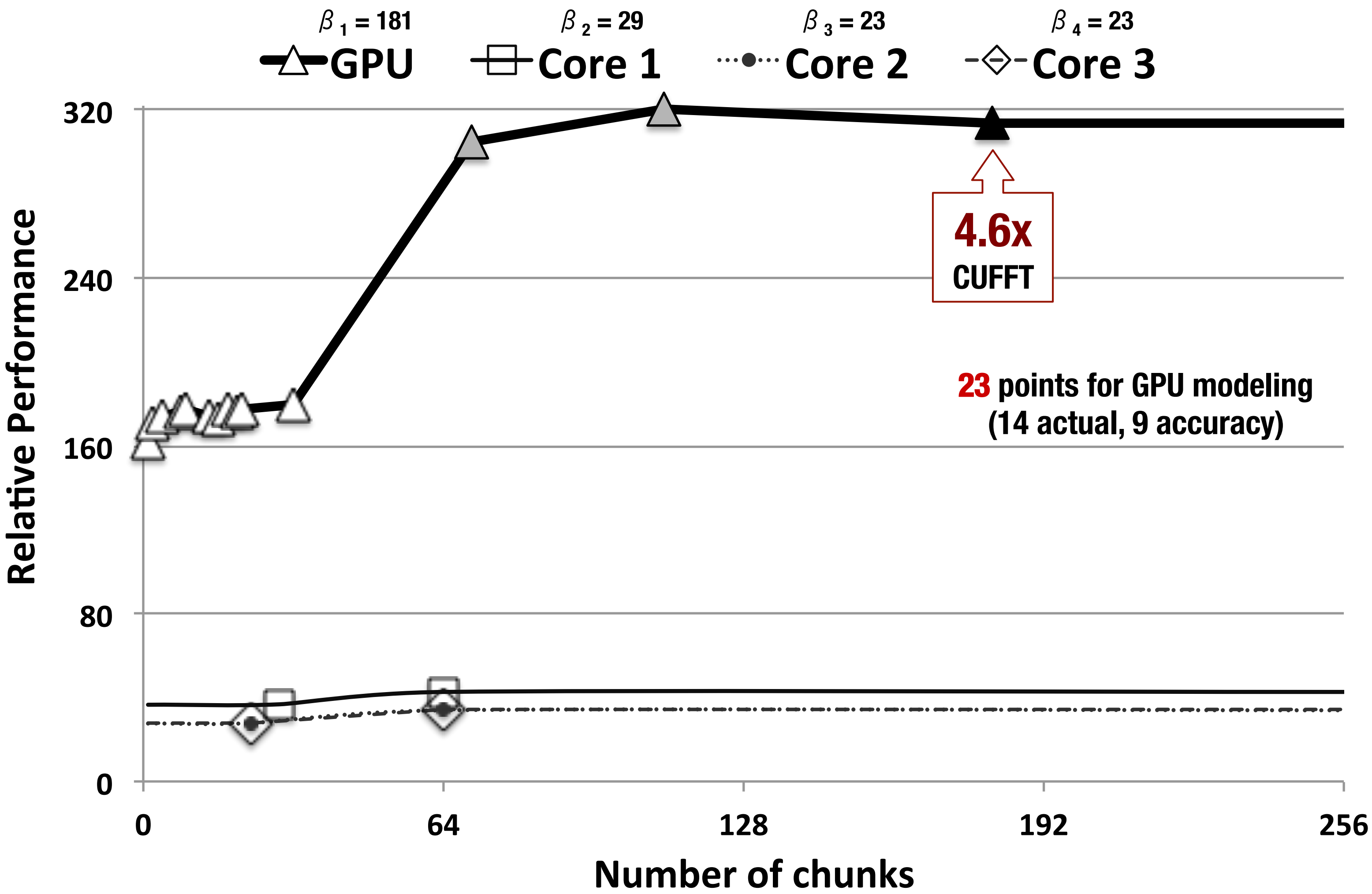
technology
from seed



Case Study: 2D FFT Batch

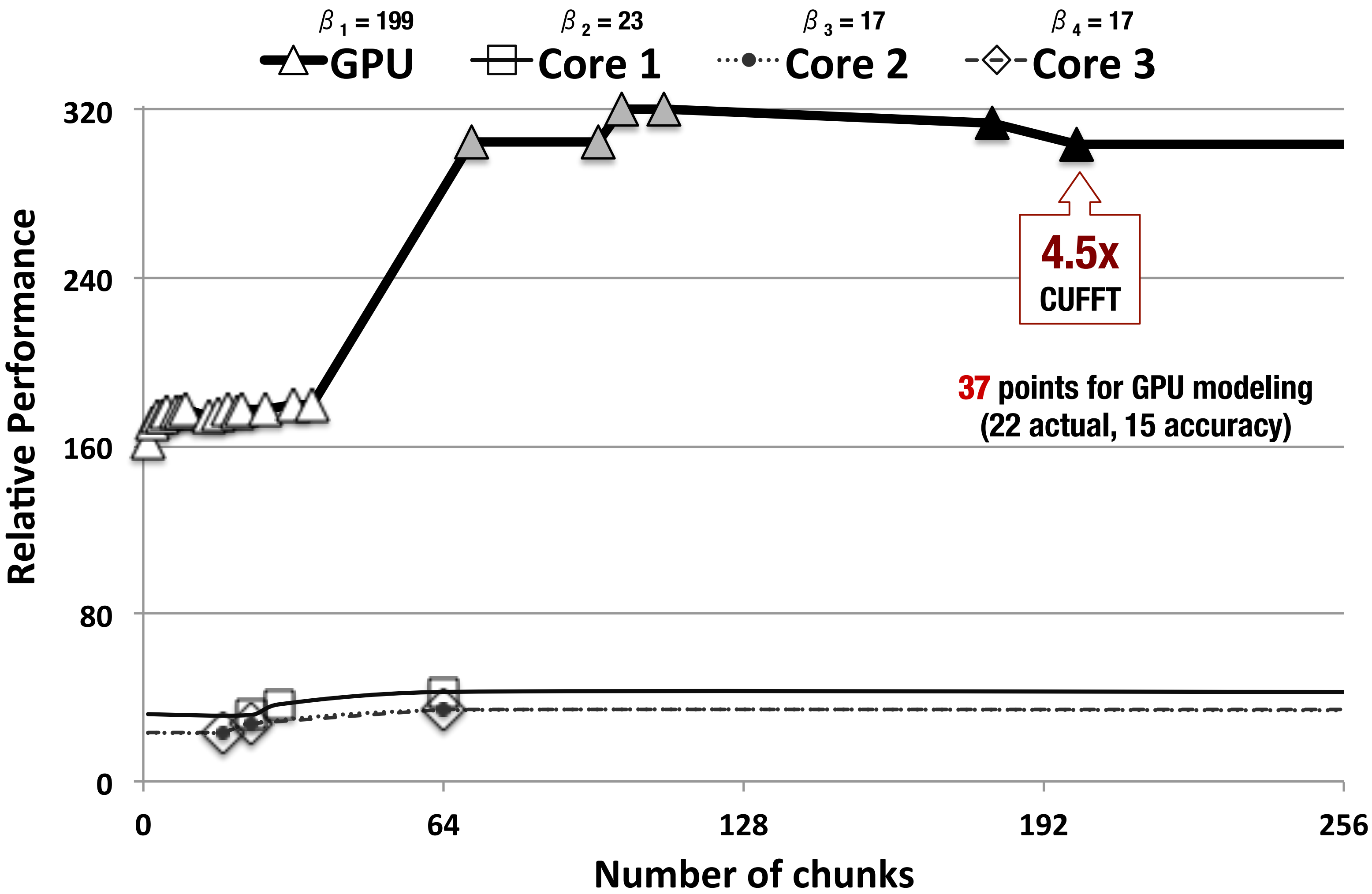
Iteration 2: Algorithm efficiency

technology
from seed



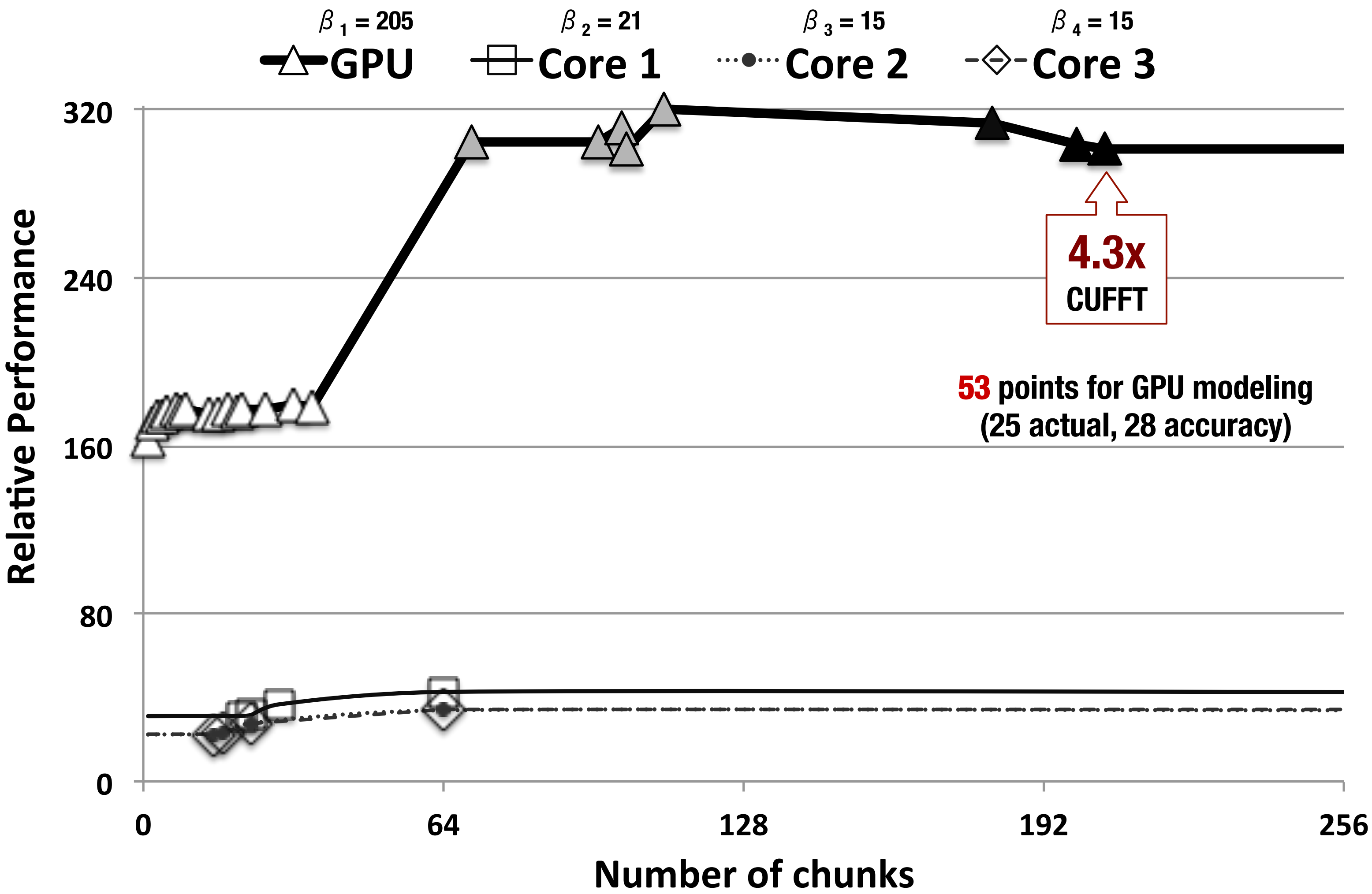
Case Study: 2D FFT Batch

ITERATION 3



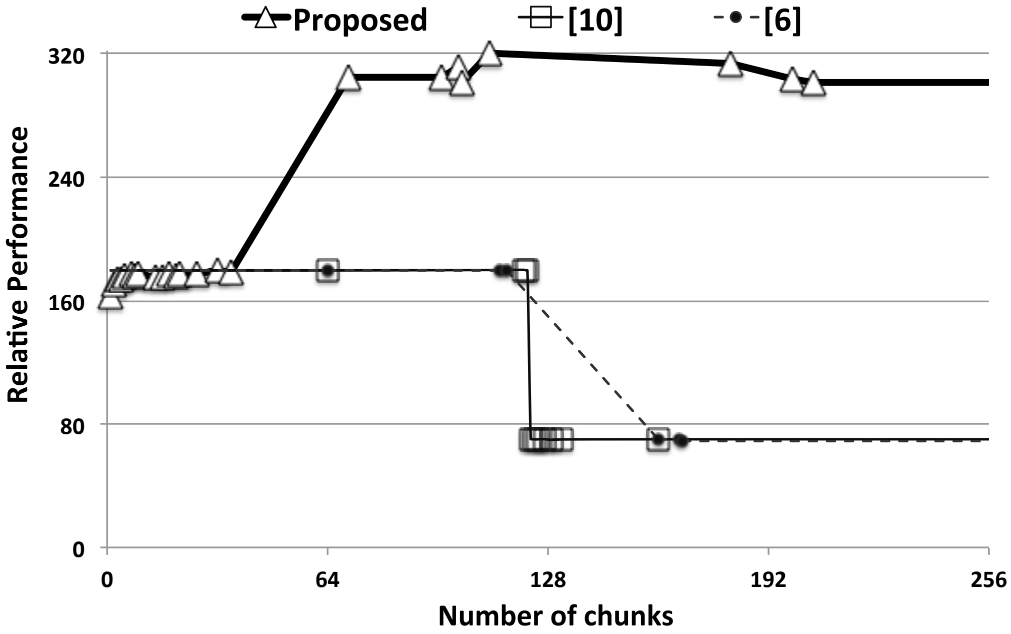
Case Study: 2D FFT Batch

Iteration 4



Case Study: 2D FFT Batch

Comparison with State of the Art Approaches



	Proposed	[10]	[6]
#Iterations to converge	**4**	10	8
Load Balancing	**YES**	**NO**	**NO**
#Points for GPU Modeling	**53** (4 iters)	10 (10 iters)	8 (8 iters)
#Points per iteration	**13.25**	1	1

- **4.3x faster** in determining the steady-state distribution (complete algorithm time) comparing to the approach from [10], and **3.2x faster** comparing to [6] (ping-pong state)
- Load balanced distribution achieves **2x better performance** comparing to the steady-state distribution from [10], and **2.2x better performance** than [6]

[6] Galindo, I., Almeida, F., Badıa-Contelles, J.M.: Dynamic load balancing on dedicated heterogeneous systems. In: PVM/ MPI. pp. 64–74 (2008)
[10] Lastovetsky, A., Reddy, R.: Data partitioning with a functional performance model of heterogeneous processors. Int. J. High Perform. Comput. Appl. 21, 76–90 (2007)

Conclusions

technology from seed

Dynamic Load Balancing

– Obtained in **4 iterations** and **4.1 seconds** (in total)

Traditional Approaches for Performance Modeling

– Approximate the performance using number of points equal to the number of iterations
– In this case, 4 **points** in total for GPU performance modeling (4 iterations)

Presented DDL Scheduling Approach

– Models the GPU performance using **53 points**, in this case ~13x more than with traditional modeling
– Load balancing solution is 2x faster than the current state of the art approach
– **In this case, obtained GPU performance is at least 4.3x better than the "optimal" CUFFT execution**


Questions?
Thank you
inesc id
lisboa