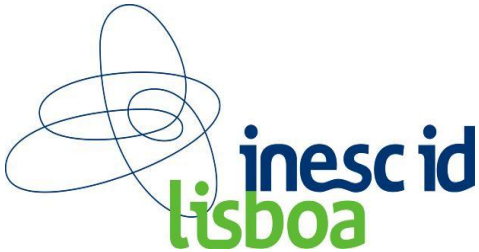
Hierarchical Divisible Load Scheduling on Heterogeneous Systems with Limited Memory

Leonel Sousa and Aleksandar Ilić



INESC-ID/
IST, TU Lisbon, Portugal

High Performance Computing, Grids and Clouds Workshop, Cetraro, 2011

Motivation

Commodity Computers = Heterogeneous Systems

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

Clusters of Commodity Computers = Highly Heterogeneous Systems

=> 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

- Discretely Divisible Load (DDL) Processing
- Heterogeneous Distributed Systems with Heterogeneous CPU+GPU Desktops as Computing Nodes
- Performance Modeling and 3-Step Hierarchical DDL Scheduling Algorithm
- Case Study: 2D FFT Batch Execution
- Conclusions and Future Work

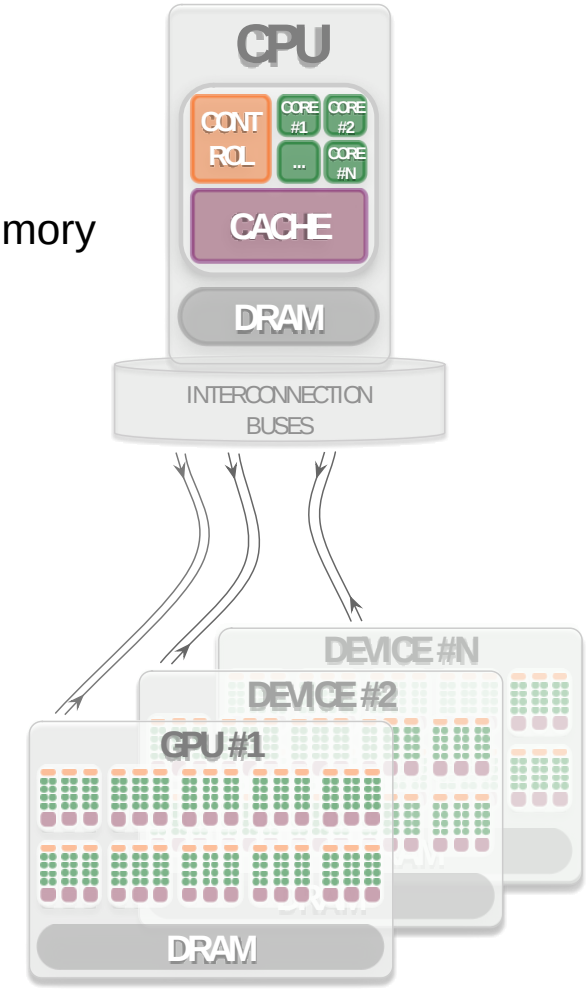
Discretely Divisible Load Processing

- 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
  - Very **recent**, but only for traditional distributed systems (no hierarchy, CPU-only)
  - Assume symmetric bandwidth and an one-port model for communication links
  - Limited memory: only the size of input load is considered, or 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 Desktop Systems

Heterogeneous star network (master-worker)

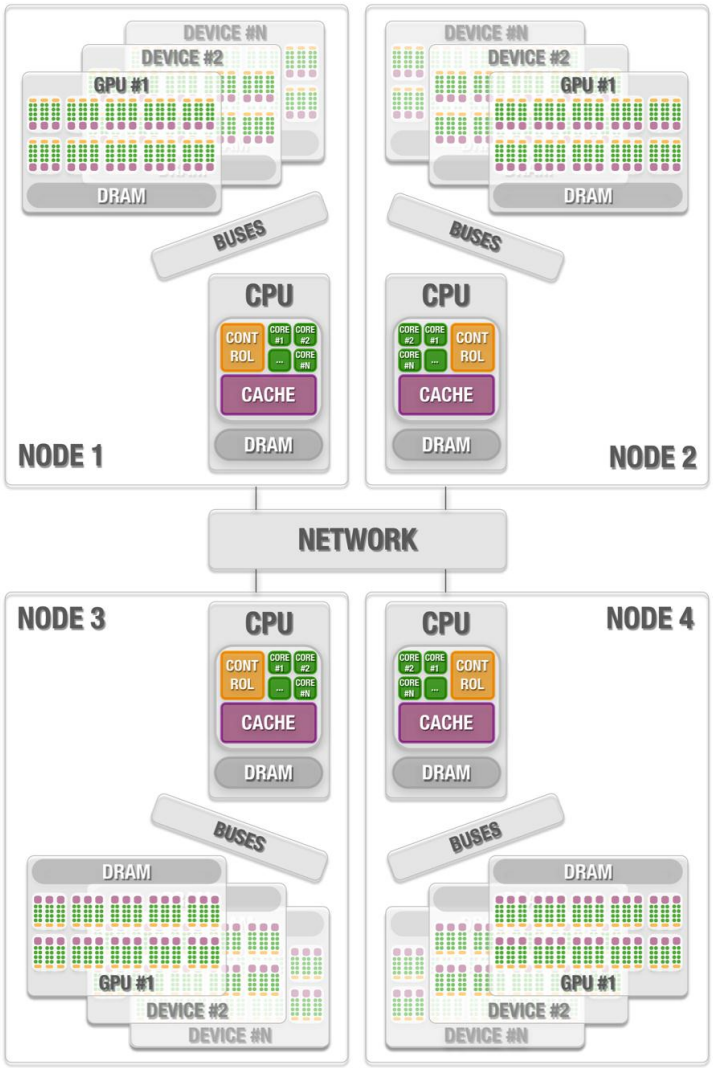
- Multi-core CPU (Master)
  - Global execution controller; access the whole global memory
  - All cores employed for execution
- Interconnection Buses
  - Bidirectional full-duplex asymmetric communication
  - Different concurrency levels
  - Potential execution bottleneck
- Devices (Distant workers)
  - Different architectures and programming models
  - Computation performed using local memories



Commodity Cluster Systems

Heterogeneous star network

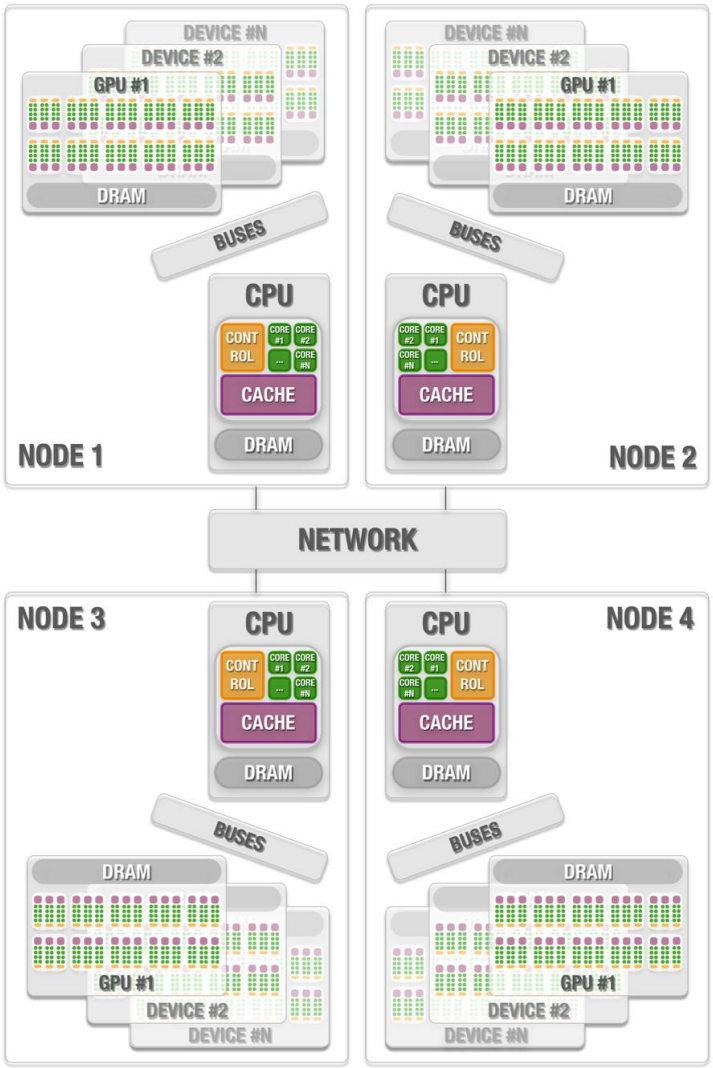
- Master Node
  - *Heterogeneous desktop system*
  - Positioned at the center; employed for execution as whole
- Interconnection Network
  - Limited asymmetric one-port communication
  - Potential execution bottleneck
- Worker Nodes
  - *Heterogeneous desktop systems*
  - All available devices employed in each



Proposed Algorithm Outline (1)

3-STEP HIERARCHICAL DDL SCHEDULING

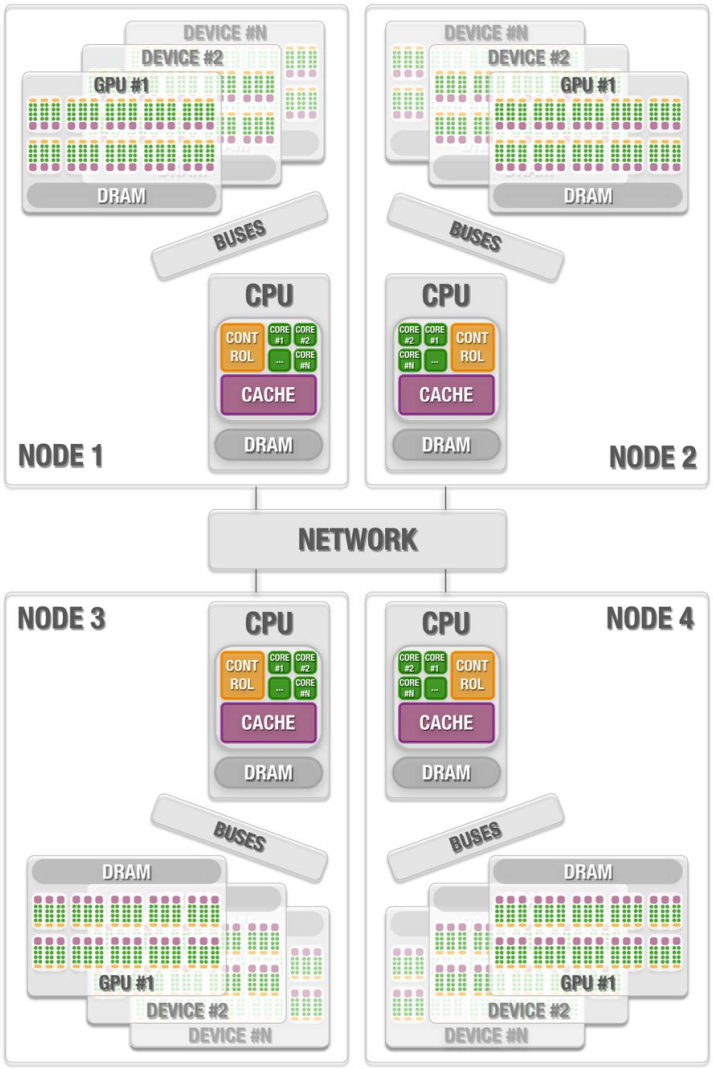
DDL



Proposed Algorithm Outline (2)

3-STEP HIERARCHICAL DDL SCHEDULING:

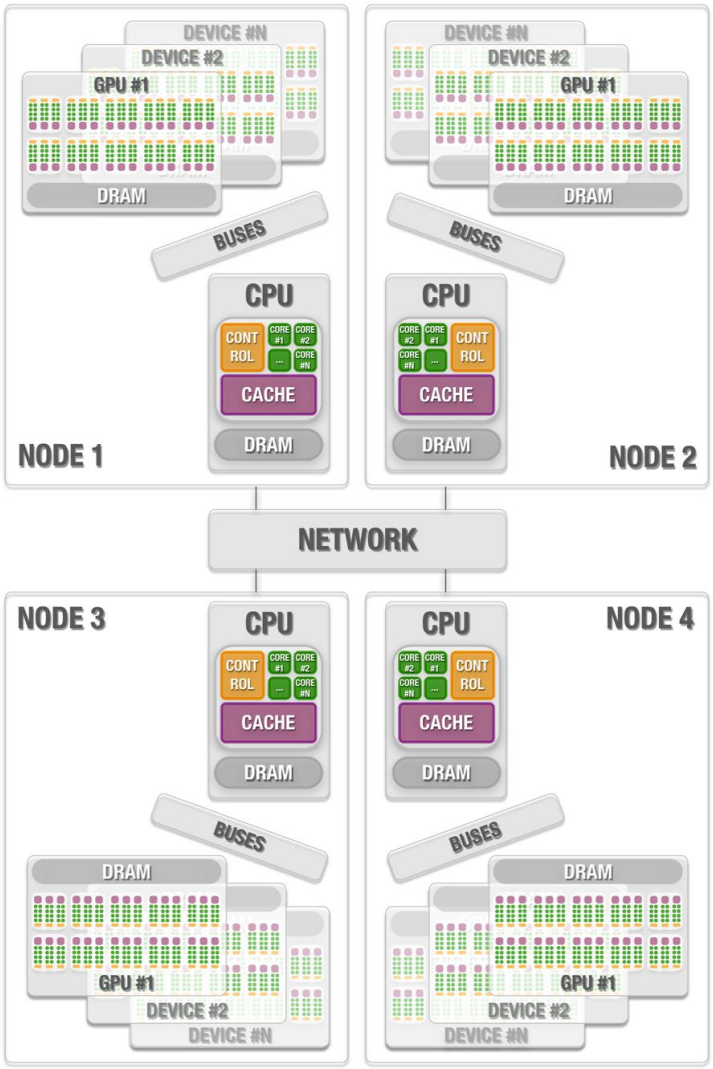
- STEP 1 – SYSTEM-LEVEL LOAD BALANCING
  - How many load units to send to each node?



Proposed Algorithm Outline (3)

3-STEP HIERARCHICAL DDL SCHEDULING:

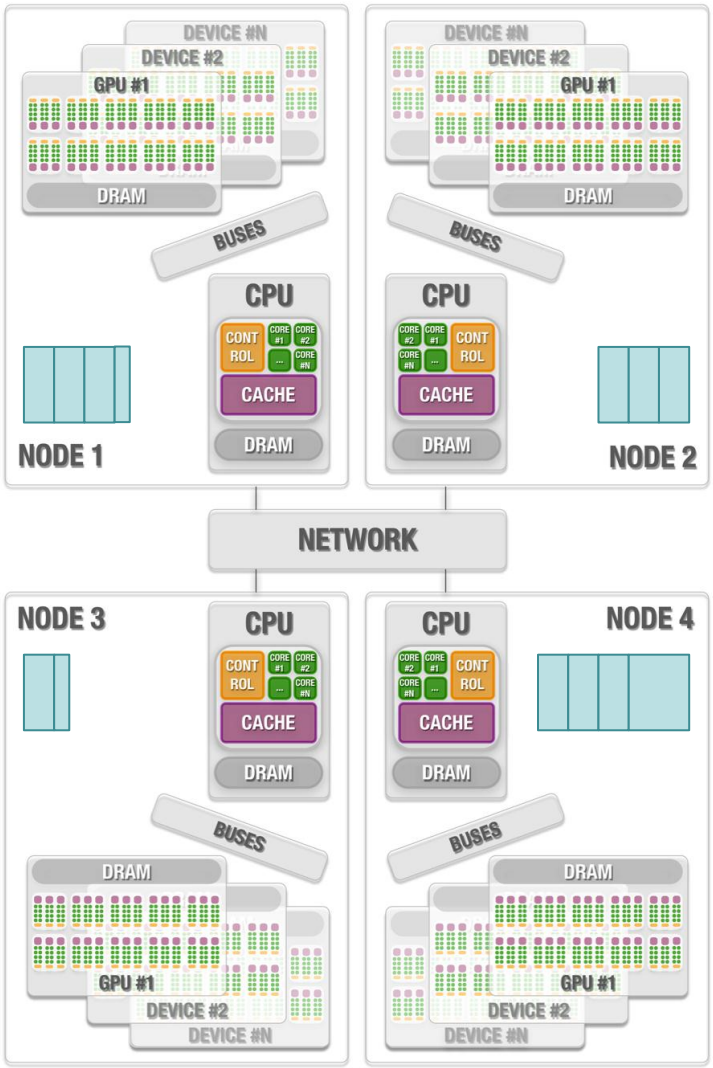
- STEP 1 – SYSTEM-LEVEL LOAD BALANCING
  - How many load units to send to each node?



Proposed Algorithm Outline (4)

3-STEP HIERARCHICAL DDL SCHEDULING:

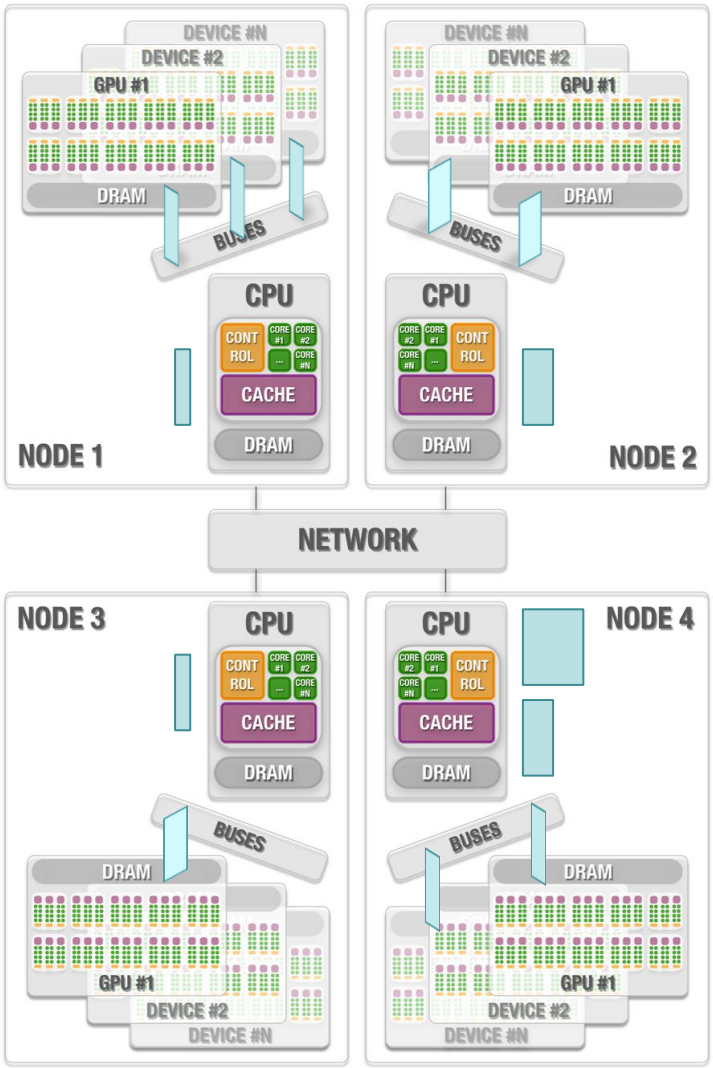
- STEP 1 – SYSTEM-LEVEL LOAD BALANCING
  - How many load units to send to each node?
- STEP 2 – NODE-LEVEL LOAD BALANCING
  - How many chunks to send to each device from a given node's load?



Proposed Algorithm Outline (5)

3-STEP HIERARCHICAL DDL SCHEDULING:

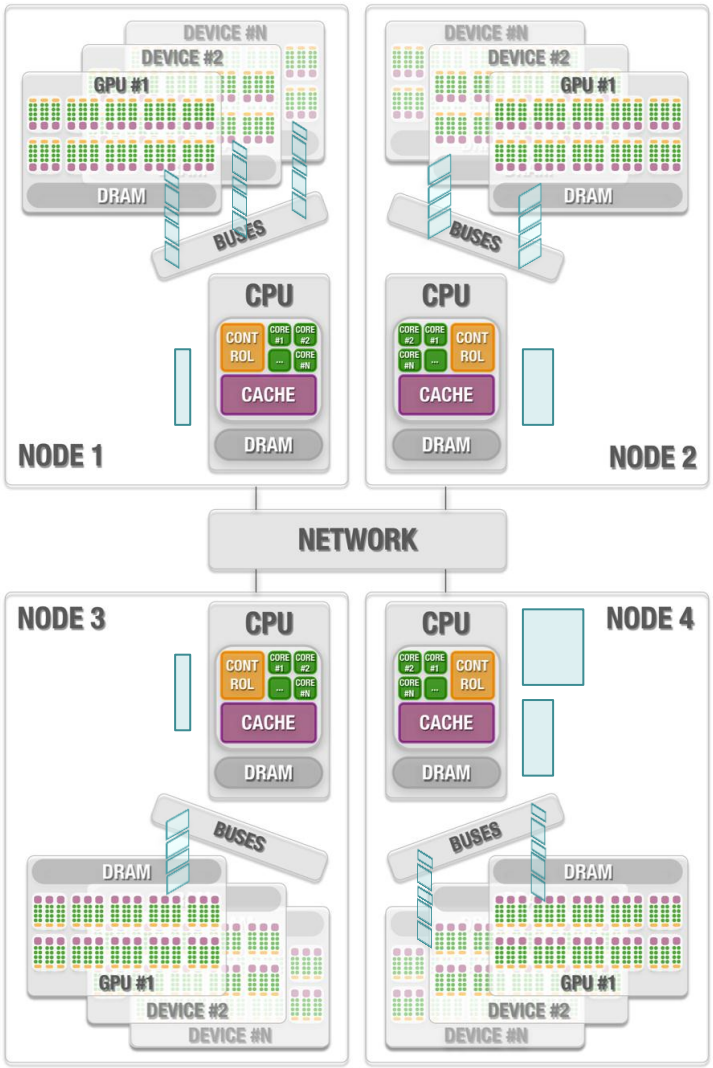
- STEP 1 – SYSTEM-LEVEL LOAD BALANCING
  - How many load units to send to each node?
- STEP 2 – NODE-LEVEL LOAD BALANCING
  - How many chunks to send to each device from a given node's load?



Proposed Algorithm Outline (6)

3-STEP HIERARCHICAL DDL SCHEDULING:

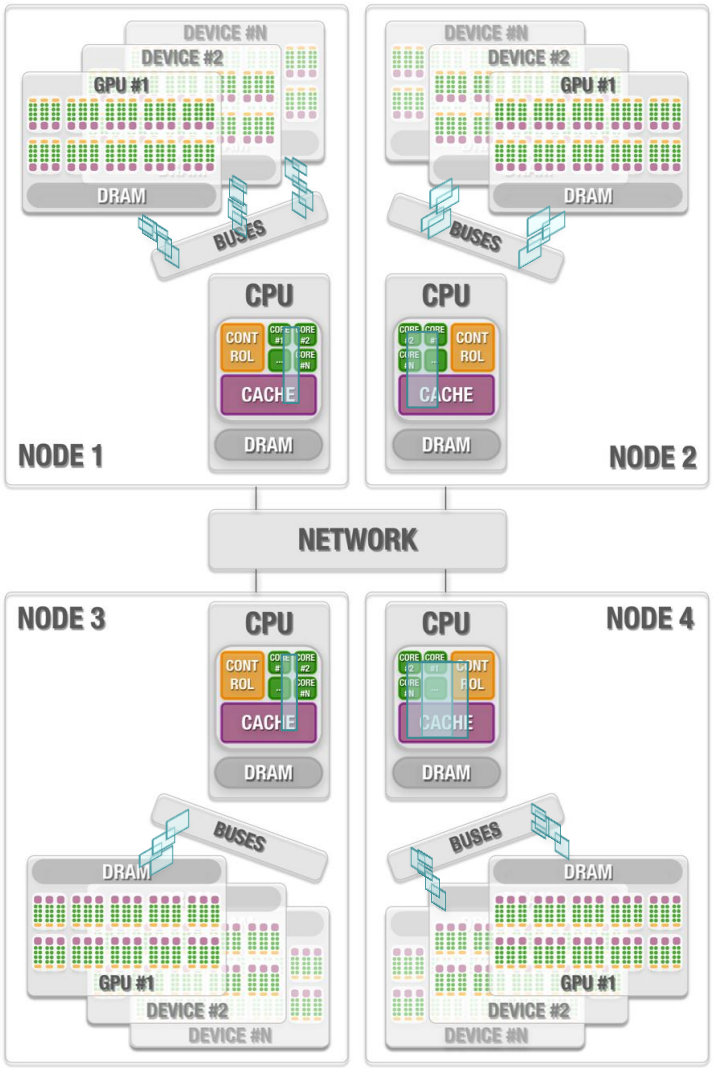
- STEP 1 – SYSTEM-LEVEL LOAD BALANCING
  - How many load units to send to each node?
- STEP 2 – NODE-LEVEL LOAD BALANCING
  - How many chunks to send to each device from a given node's load?
- STEP 3 – DEVICE-LEVEL SCHEDULING
  - How to sub-partition the device load to:
    - Reduce delays when distributing 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 (7)

3-STEP HIERARCHICAL DDL SCHEDULING:

- STEP 1 – SYSTEM-LEVEL LOAD BALANCING
  - How many load units to send to each node?
- STEP 2 – NODE-LEVEL LOAD BALANCING
  - How many chunks to send to each device from a given node's load?
- STEP 3 – DEVICE-LEVEL 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



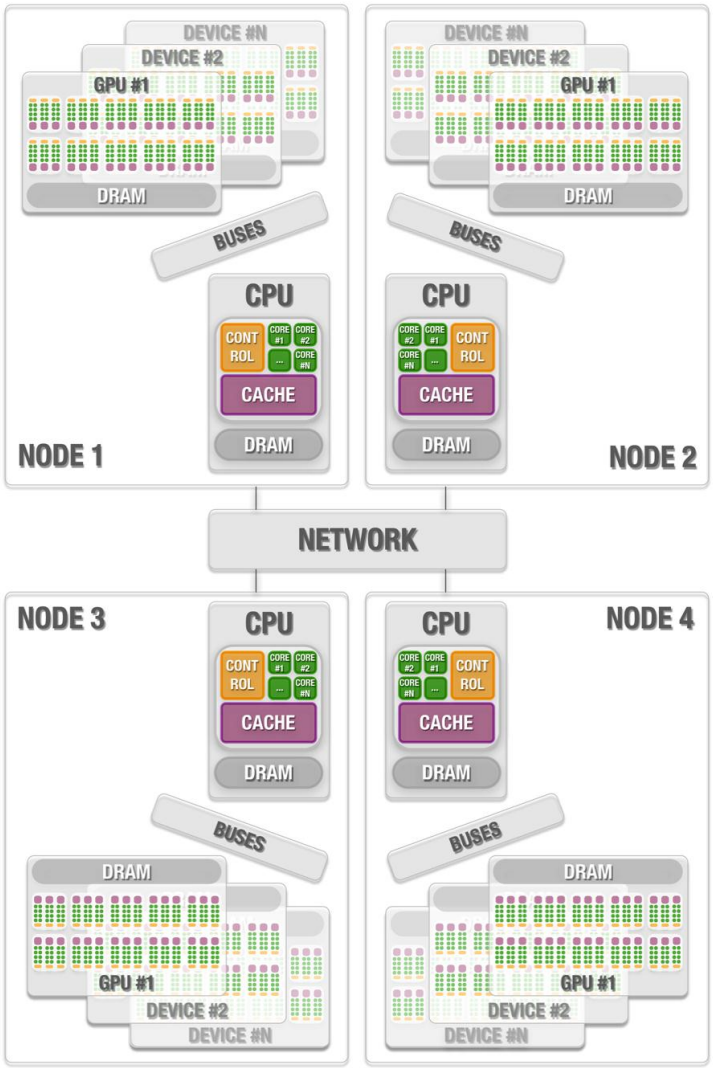
Performance Modeling

Functional Performance Models

- Built from the real application execution
  - *No assumptions being made to ease modeling!*

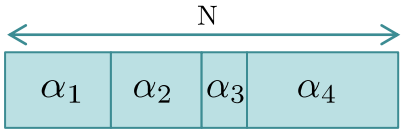
- Node-level Performance Models
  - Computation performance models (ψ_w)
    - For each master core and distant worker
  - Full-duplex communication bandwidth (σ_ι , σ_o)
    - Bidirectional and asymmetric for each link
  - Total performance (ψ_τ) of each device
    - Including computation and communication

- System-level Performance Models
  - Computation performance models (Ψ_w)
    - For each node (heterogeneous desktop system)
  - Communication bandwidth (Σ_ι , Σ_o)
    - Bidirectional, asymmetric for each network link
  - Total performance (Ψ_τ) of each node



Determination of Load Fractions (1)

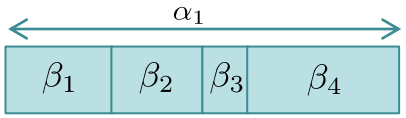
- STEP 1 – SYSTEM-LEVEL LOAD BALANCING



  - The optimal distribution between nodes lies on a straight line passing through the origin of coordinate system and intersecting total performance curves (Ψ_τ)*:

$$\frac{\alpha_1}{\Psi_{t_1}(\alpha_1)} = \cdots = \frac{\alpha_k}{\Psi_{t_k}(\alpha_k)} = \cdots = \frac{\alpha_n}{\Psi_{t_n}(\alpha_n)} \qquad \sum_{i=1}^{n} \alpha_i = \mathrm{N}$$

- STEP 2 – NODE-LEVEL LOAD BALANCING



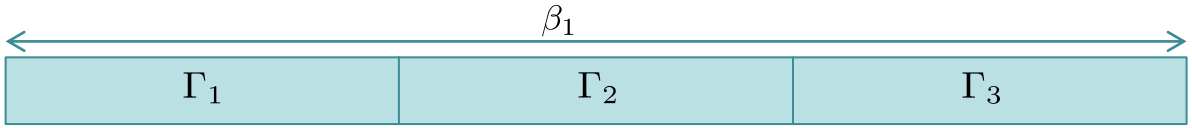
  - Each load fraction α_i is sub-partitioned between node's devices; the optimal distribution that lies on a straight line passing through the origin of coordinate system and intersecting communication-aware total performance curves (ψ_τ), such that*:

$$\frac{\beta_1}{\psi_{\tau_1}(\beta_1)} = \cdots = \frac{\beta_k}{\psi_{\tau_k}(\beta_k)} = \cdots = \frac{\beta_n}{\psi_{\tau_n}(\beta_n)} \qquad \sum_{j=1}^{n} \beta_j = \alpha_i$$

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.

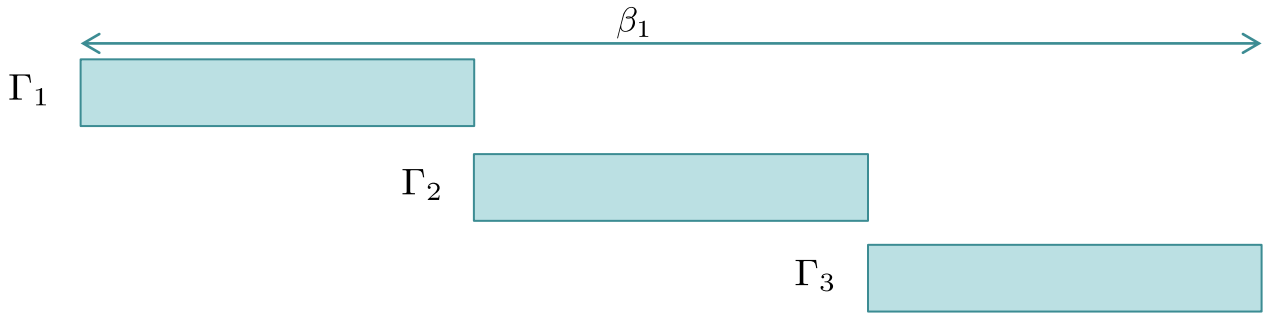
Determination of Load Fractions (2)

- STEP 3 – 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



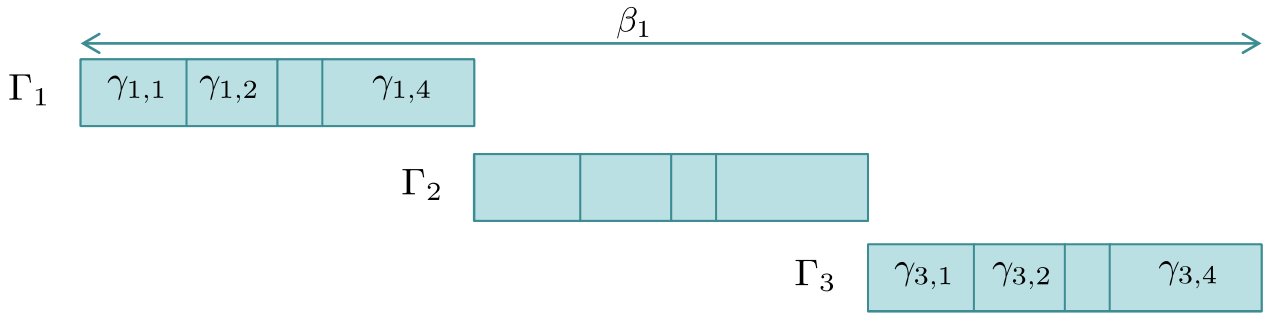
Determination of Load Fractions (3)

- STEP 3 – 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 3 – 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 -

- STEP 3 – 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!

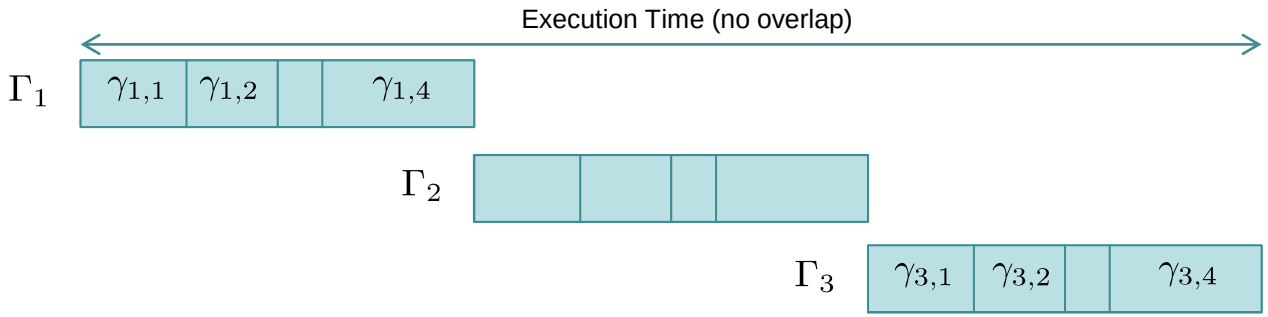
$\Rightarrow$ In each Γ_k *sub-distribution* the whole amount of memory may be consumed, such that:

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

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

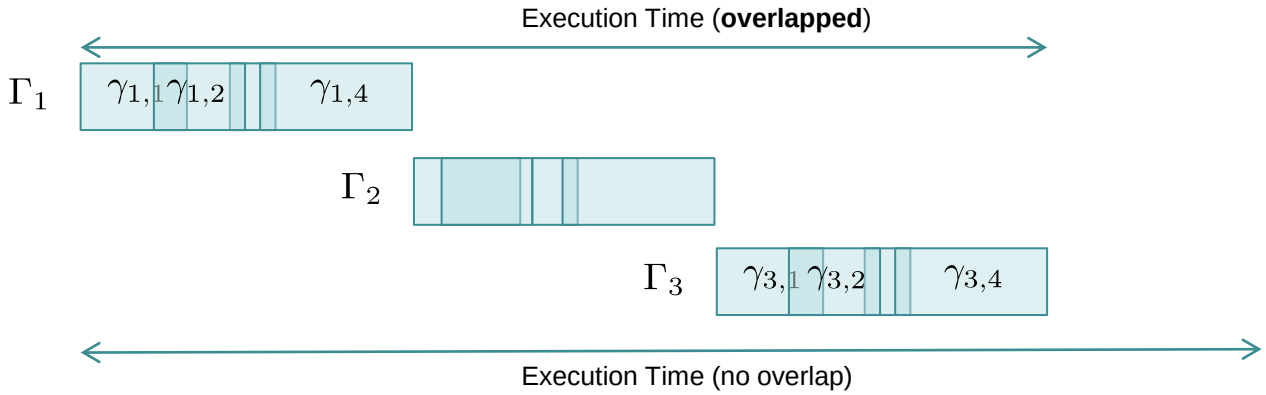
Determination of Load Fractions (6)
- Computation/Communication Overlapping -

- STEP 3 – 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



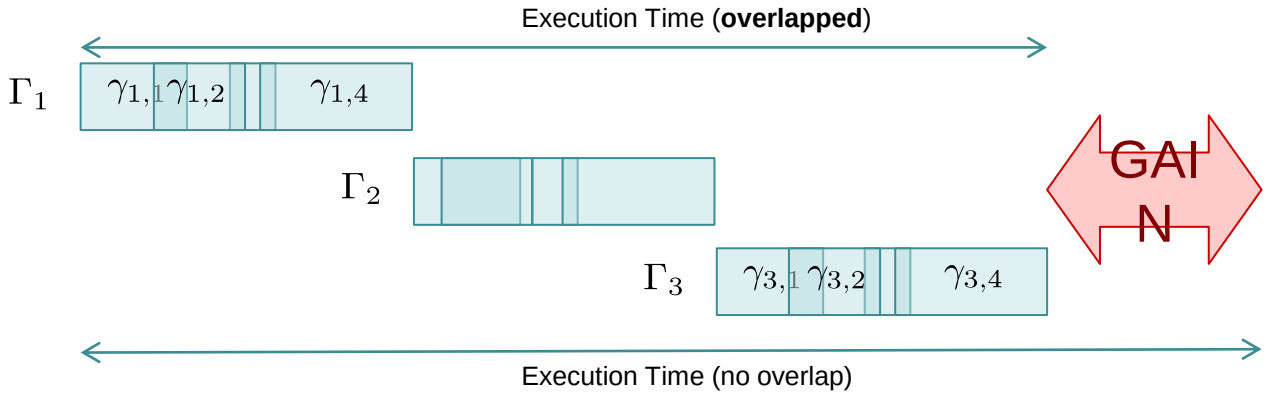
Determination of Load Fractions (7)
- Computation/Communication Overlapping -

- STEP 3 – 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



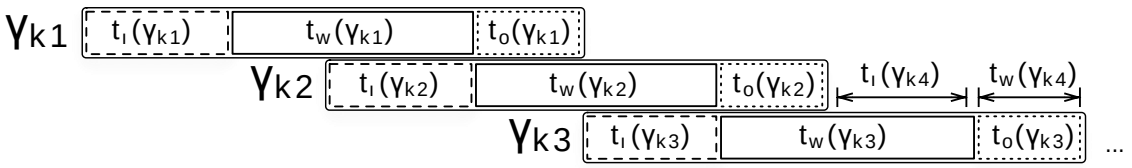
Determination of Load Fractions (8)
- Computation/Communication Overlapping -

- STEP 3 – 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

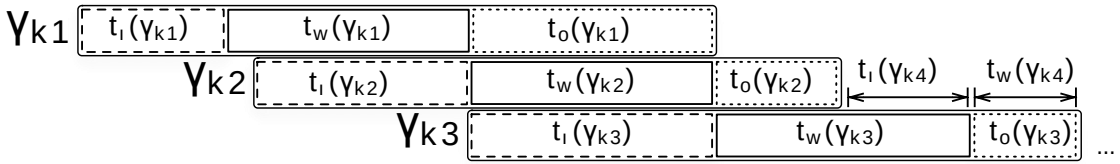


Determination of Load Fractions (9)
- Computation/Communication Overlapping -

- STEP 3 – 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:



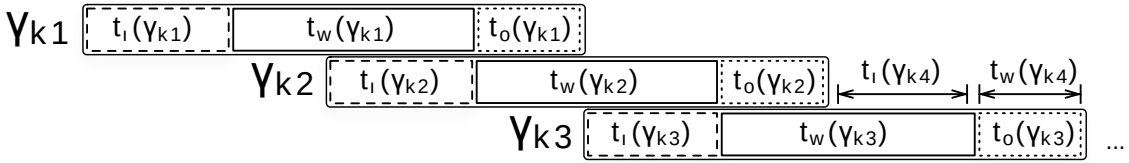
(a) Overlap of a single communication with computation at the time



(b) Complete concurrency between communication and computation

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

- Step 3 – Device-level Scheduling Algorithm*/**



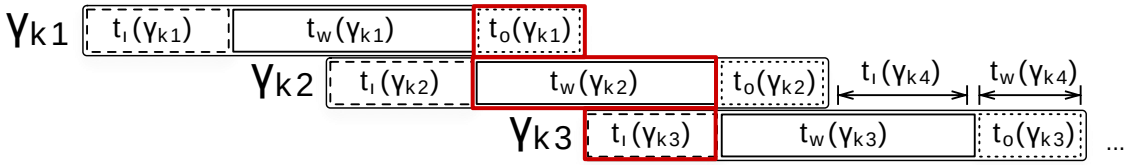
  - According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)

** Ilić, A., and Sousa, L., "Scheduling Divisible Loads on Heterogeneous Desktop Systems with Limited Memory" (submitted to HeteroPar 2011)*

*** 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 3 – Device-level Scheduling Algorithm*/**



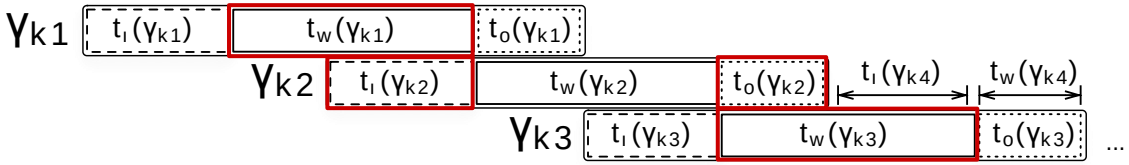
  - According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)
    - Step 3-I. Determination of the initial optimal distribution with three load fractions.

** Ilić, A., and Sousa, L., "Scheduling Divisible Loads on Heterogeneous Desktop Systems with Limited Memory" (submitted to HeteroPar 2011)*

*** 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 3 – Device-level Scheduling Algorithm*/**



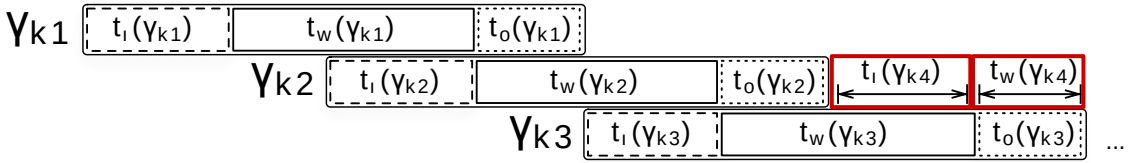
  - According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)
    - Step 3-I. Determination of the initial optimal distribution with three load fractions.
    - Step 3-II. Generate additional three-fraction distributions.

** Ilić, A., and Sousa, L., "Scheduling Divisible Loads on Heterogeneous Desktop Systems with Limited Memory" (submitted to HeteroPar 2011)*

*** 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 -

- Step 3 – Device-level Scheduling Algorithm*/**



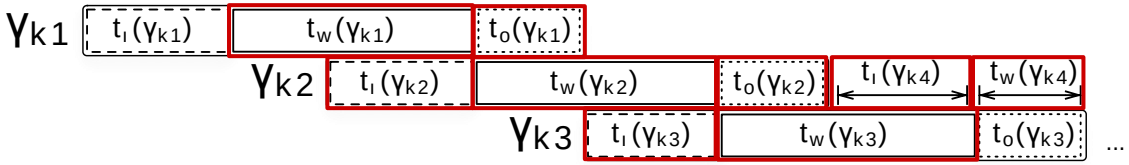
  - According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)
    - Step 3-I. Determination of the initial optimal distribution with three load fractions.
    - Step 3-II. Generate additional three-fraction distributions.
    - Step 3-III. Insert additional load fractions into existing sub-distributions (iterative).

** Ilić, A., and Sousa, L., "Scheduling Divisible Loads on Heterogeneous Desktop Systems with Limited Memory" (submitted to HeteroPar 2011)*

*** 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 3 – Device-level Scheduling Algorithm*/**



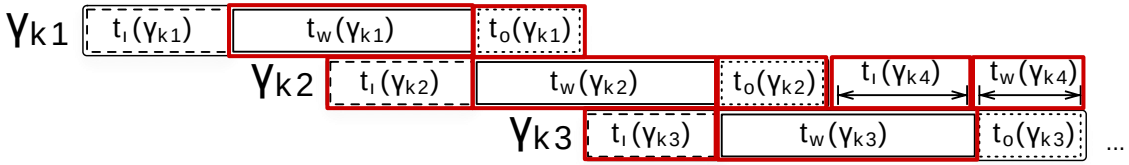
  - According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)
    - Step 3-I. Determination of the initial optimal distribution with three load fractions.
    - Step 3-II. Generate additional three-fraction distributions.
    - Step 3-III. Insert additional load fractions into existing sub-distributions (iterative).
    - Step 3-IV. Generate new sub-distributions by restarting.

** Ilić, A., and Sousa, L., "Scheduling Divisible Loads on Heterogeneous Desktop Systems with Limited Memory" (submitted to HeteroPar 2011)*

*** 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 -

- Step 3 – Device-level Scheduling Algorithm*/**



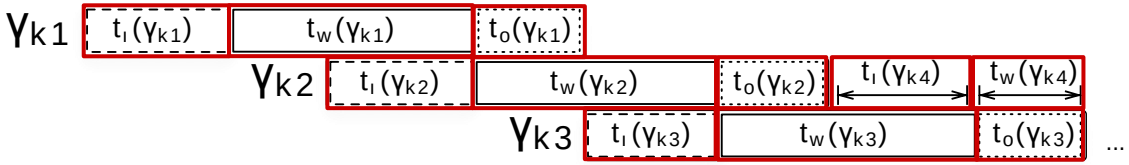
  - According to the performance models for device computation (ψ_w) and bidirectional asymmetric full-duplex communication links (σ_ι, σ_o)
    - Step 3-I. Determination of the initial optimal distribution with three load fractions.
    - Step 3-II. Generate additional three-fraction distributions.
    - Step 3-III. Insert additional load fractions into existing sub-distributions (iterative).
    - Step 3-IV. Generate new sub-distributions by restarting.
    - Step 3-V. Expand all sub-distributions.

** Ilić, A., and Sousa, L., "Scheduling Divisible Loads on Heterogeneous Desktop Systems with Limited Memory" (submitted to HeteroPar 2011)*

*** 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 -

- Step 3 – 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 3-I. Determination of the initial optimal distribution with three load fractions.
    - Step 3-II. Generate additional three-fraction distributions.
    - Step 3-III. Insert additional load fractions into existing sub-distributions (iterative).
    - Step 3-IV. Generate new sub-distributions by restarting.
    - Step 3-V. Expand all sub-distributions.
    - Step 3-VI. Select the distribution with maximum relative performance.

** Ilić, A., and Sousa, L., "Scheduling Divisible Loads on Heterogeneous Desktop Systems with Limited Memory" (submitted to HeteroPar 2011)*

*** 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

Double Floating Point Complex 2D FFT Batch Execution

- Size: 1024 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

Distributed Heterogeneous System with 4 CPU+GPU Nodes

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

	#CPU Cores	#GPUs
NODE 1	3	1
NODE 2	2	
NODE 3	1	
NODE 4	-	

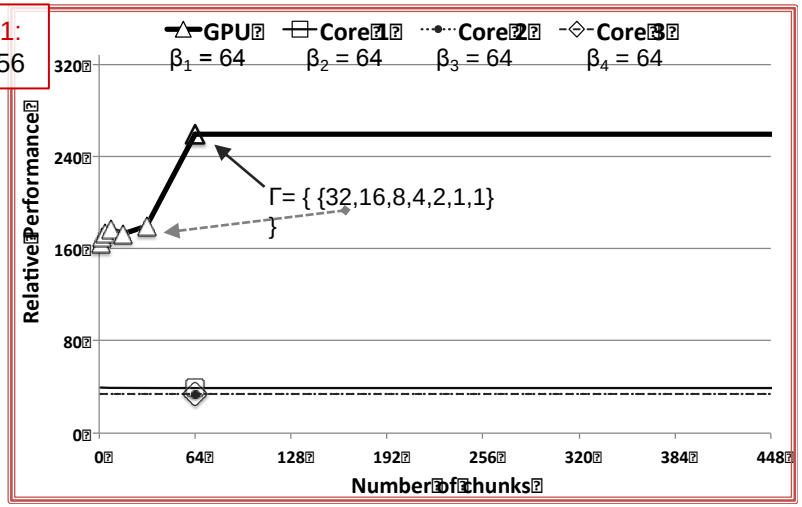
Iterative Procedure for Online Performance Modeling

- 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 nodes/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

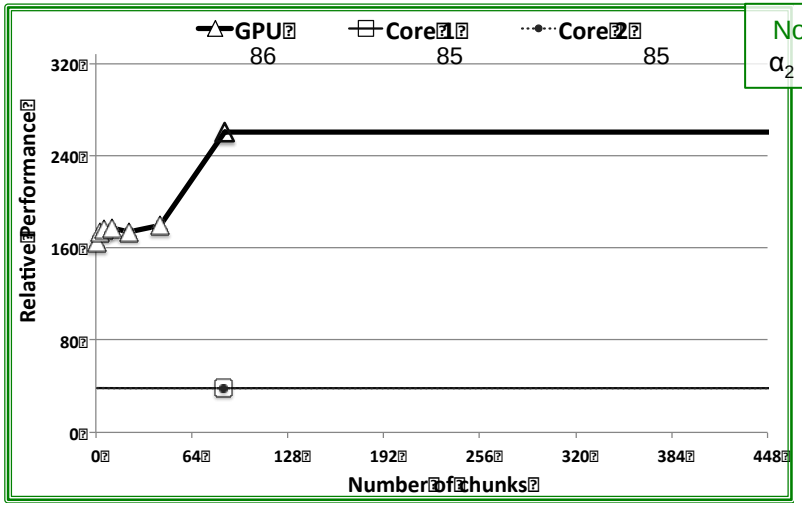
** Ilić, A., and Sousa, L., "Algorithm For Divisible Load Scheduling on Heterogeneous Systems with Realistic Performance Models", Tech. rep., INESC-ID (May 2011)*

Case Study: 2D FFT Batch
Iteration 1: Factoring-by-Two Strategy

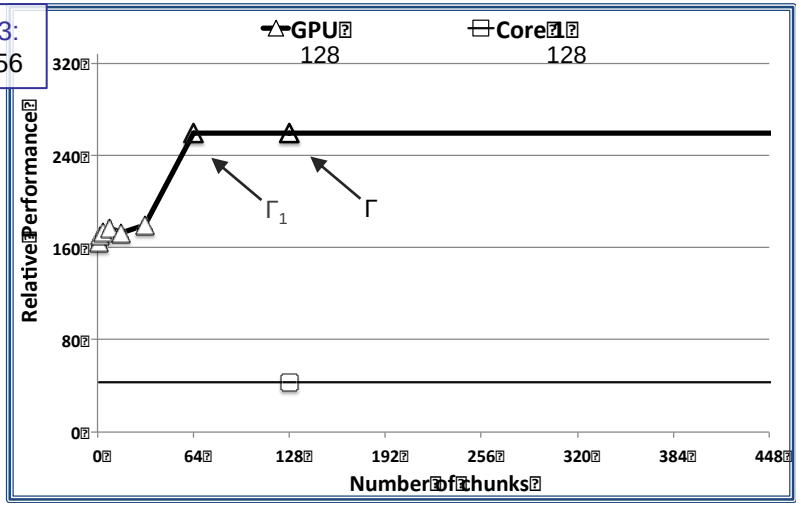
Node 1:
$\alpha_1 = 256$



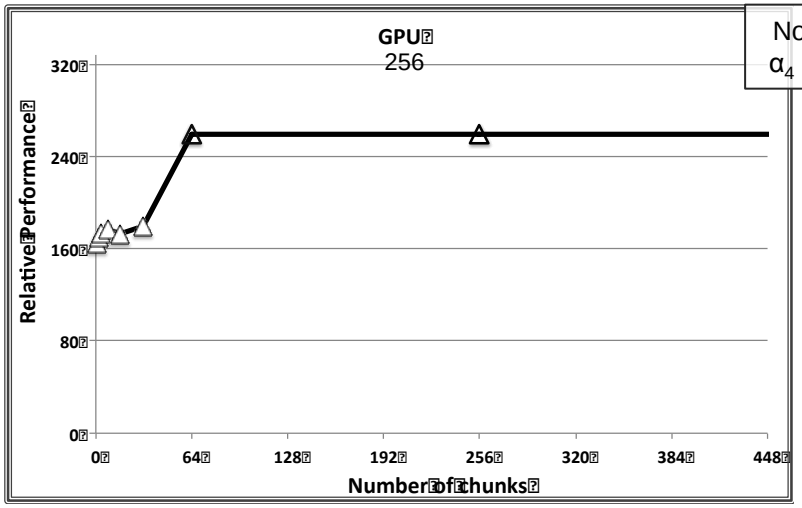
Node 2:
$\alpha_2 = 256$



Node 3:
$\alpha_3 = 256$



Node 4:
$\alpha_4 = 256$

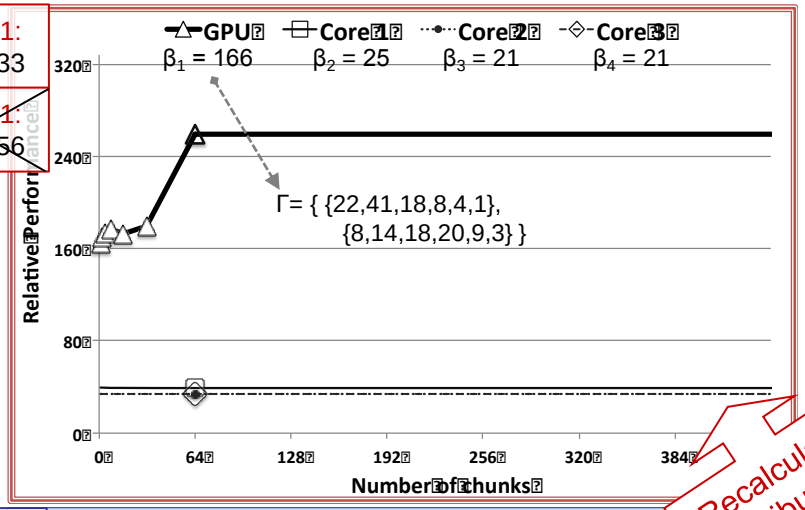


Case Study: 2D FFT Batch
Iteration 1: Proposed Algorithm

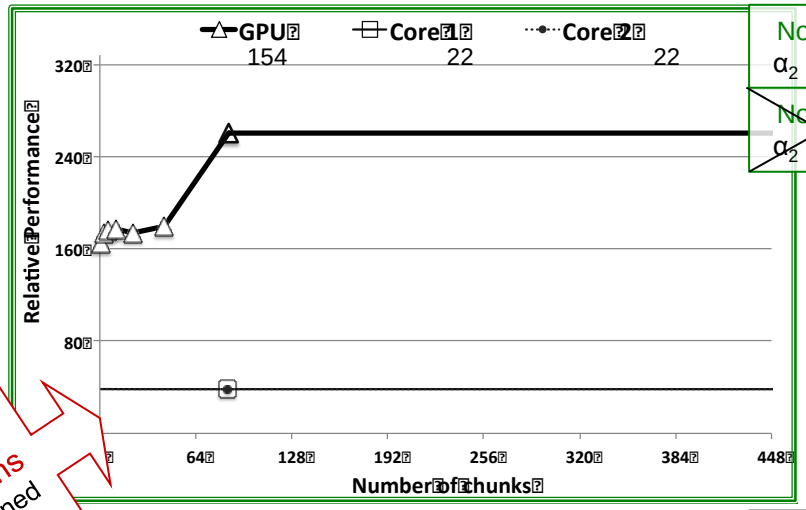
technology
from seed

inesc id lisboa

Node 1:
$\alpha_1 = 233$
~~Node 1:~~
~~$\alpha_1 = 256$~~

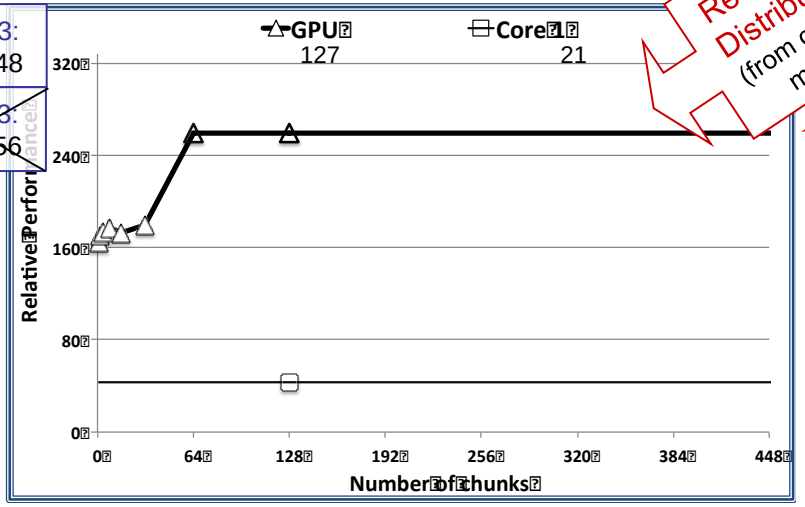


Node 2:
$\alpha_2 = 198$
~~Node 2:~~
~~$\alpha_2 = 256$~~

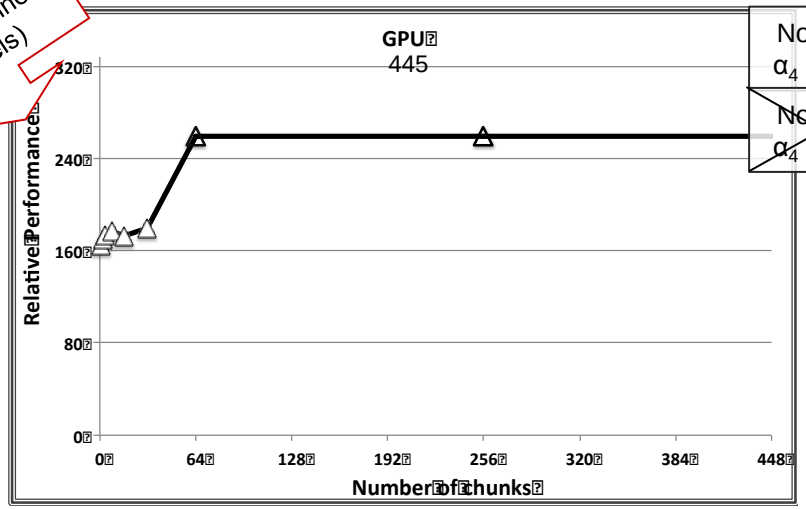


Recalculate Distributions (from obtained models)

Node 3:
$\alpha_3 = 148$
~~Node 3:~~
~~$\alpha_3 = 256$~~



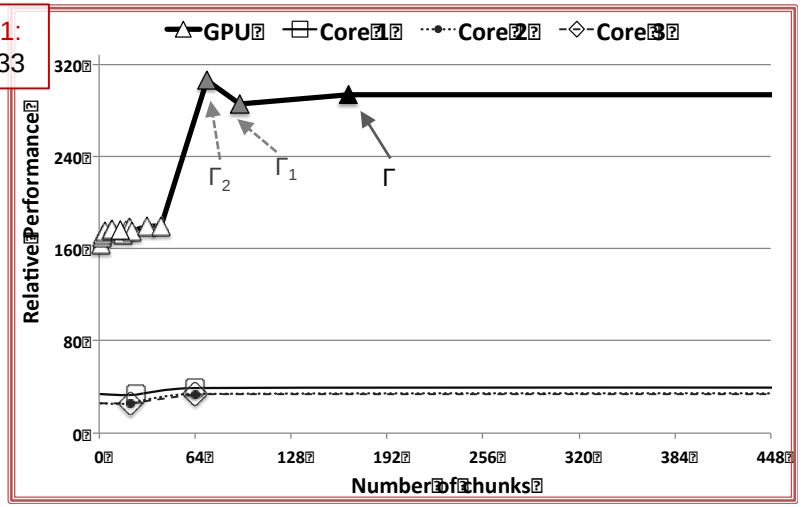
Node 4:
$\alpha_4 = 445$
~~Node 4:~~
~~$\alpha_4 = 256$~~



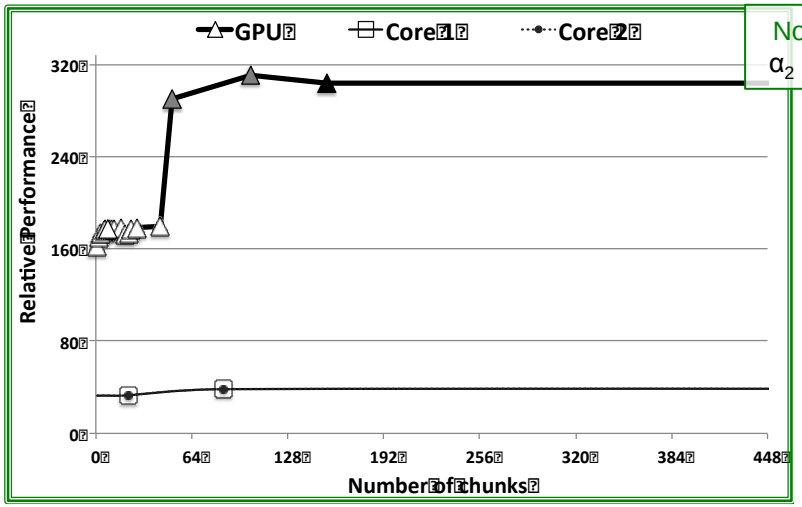
Case Study: 2D FFT Batch
ITERATION 2: PERFORMANCE MODELS

technology
from seed

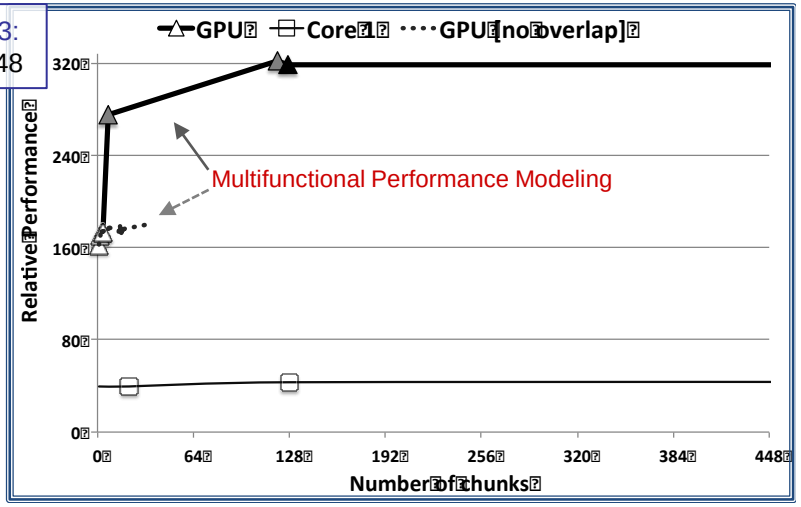
Node 1:
$\alpha_1 = 233$



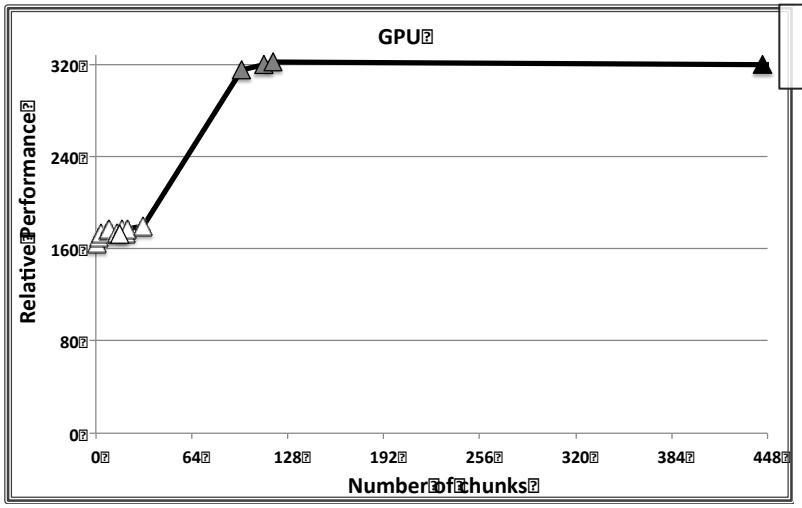
Node 2:
$\alpha_2 = 198$



Node 3:
$\alpha_3 = 148$



Node 4:
$\alpha_4 = 445$

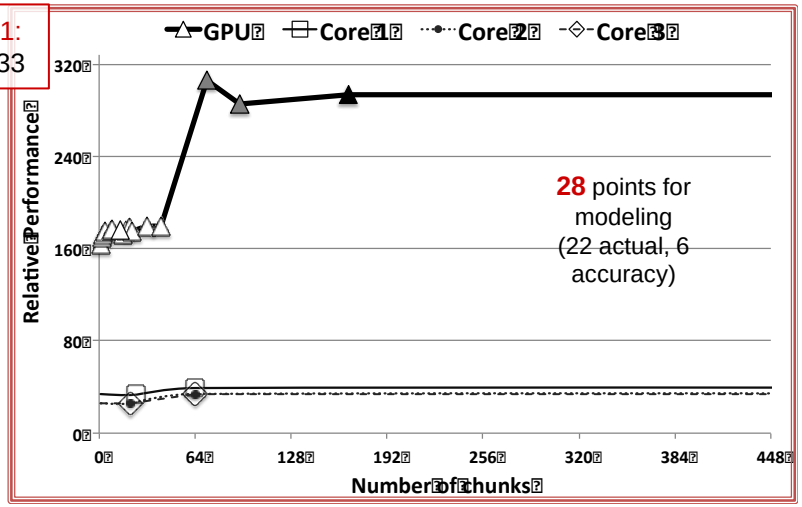


Case Study: 2D FFT Batch
ITERATION 2: MODELING EFFCIENCY

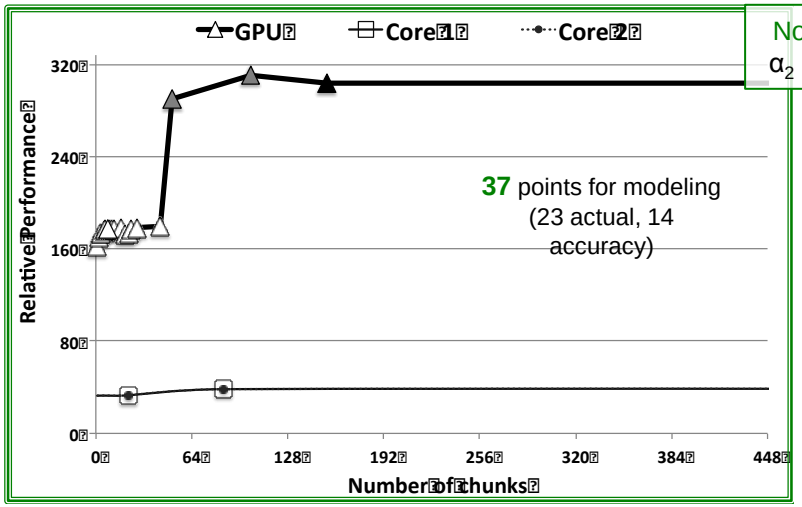
technology
from seed

inesc id lisboa

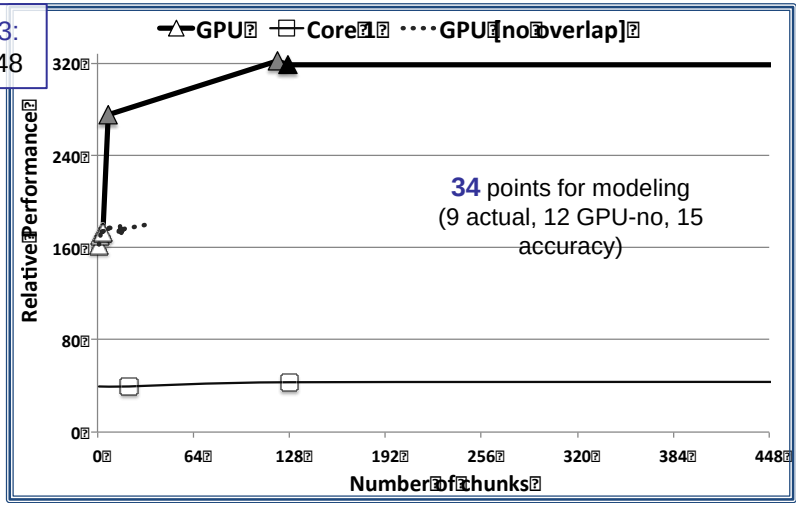
Node 1:
$\alpha_1 = 233$



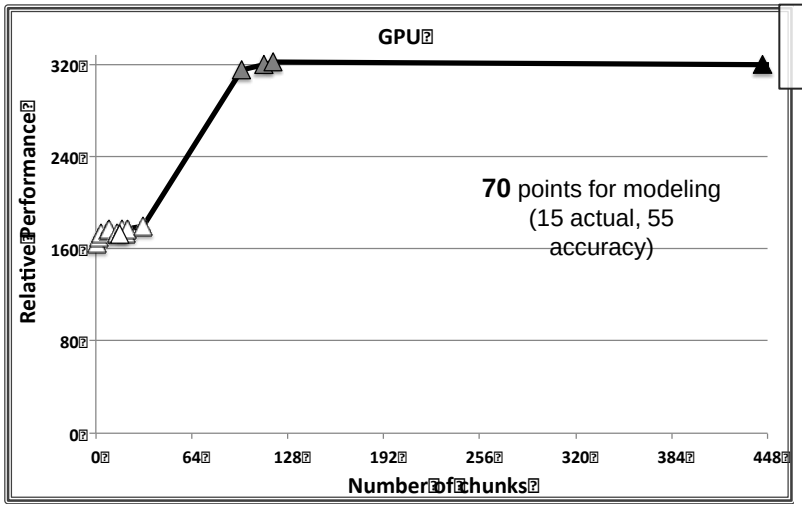
Node 2:
$\alpha_2 = 198$



Node 3:
$\alpha_3 = 148$



Node 4:
$\alpha_4 = 445$

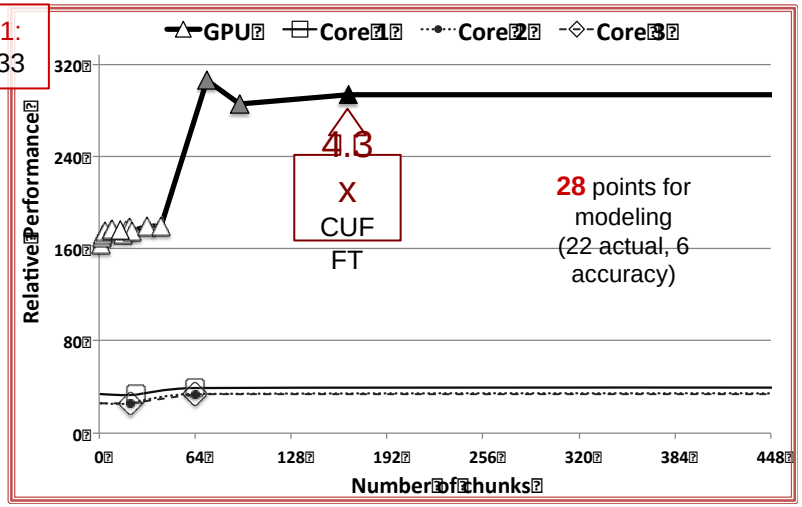


Case Study: 2D FFT Batch
ITERATION 2: PERFORMANCE

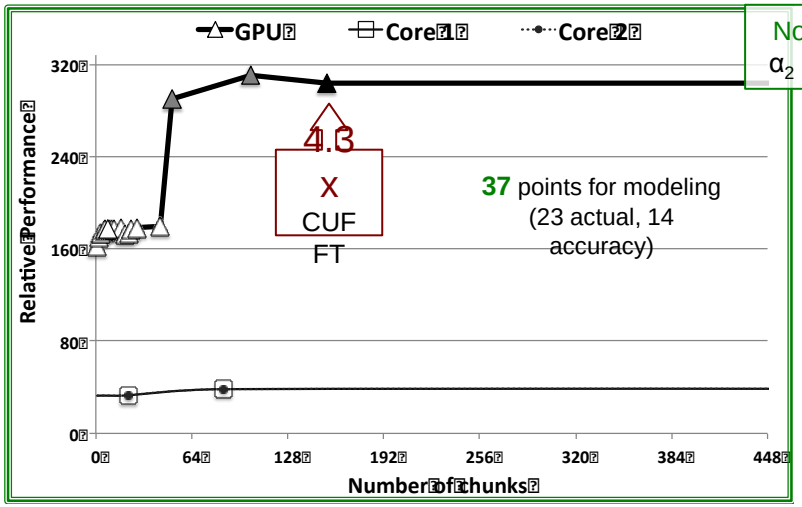
technology
from seed

inesc id lisboa

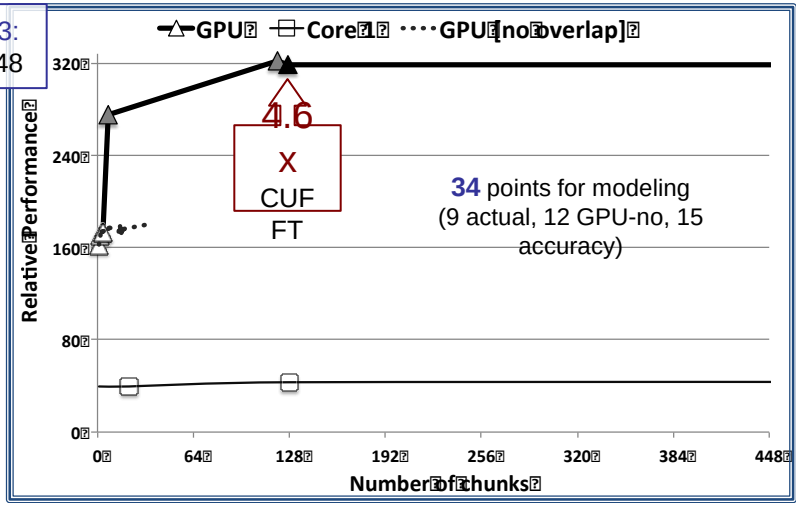
Node 1:
$\alpha_1 = 233$



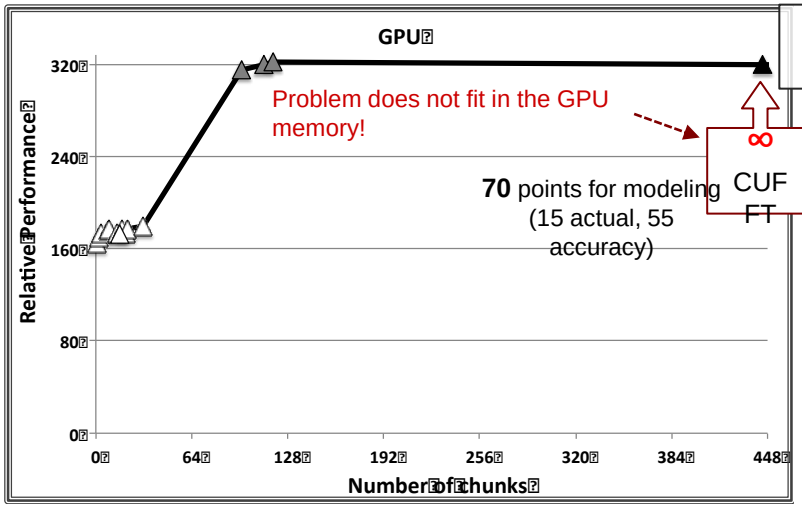
Node 2:
$\alpha_2 = 198$



Node 3:
$\alpha_3 = 148$



Node 4:
$\alpha_4 = 445$

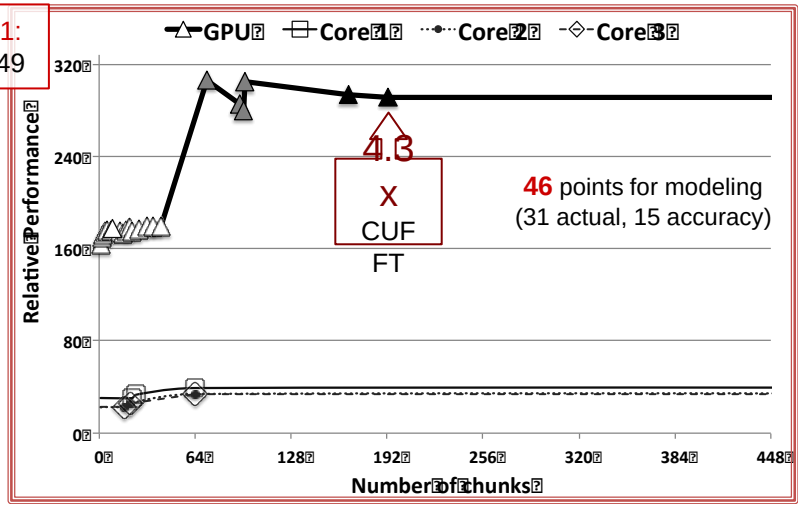


Case Study: 2D FFT Batch
ITERATION 3

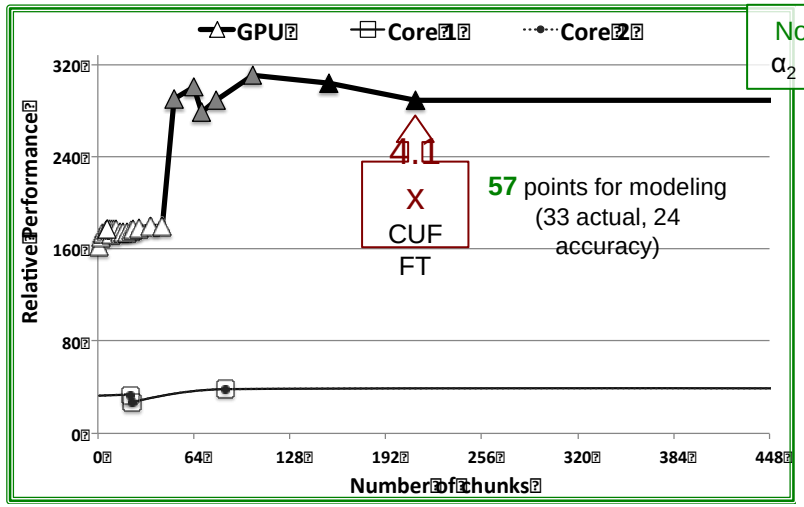
technology
from seed

inesc id lisboa

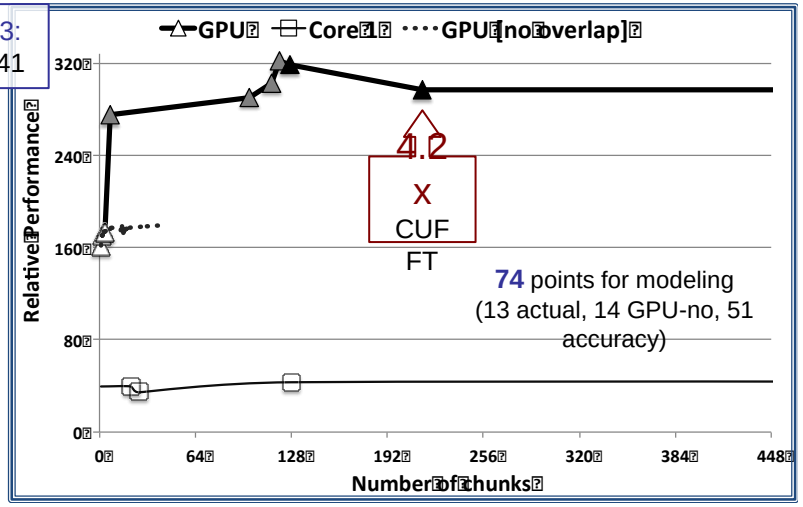
Node 1:
$\alpha_1 = 249$



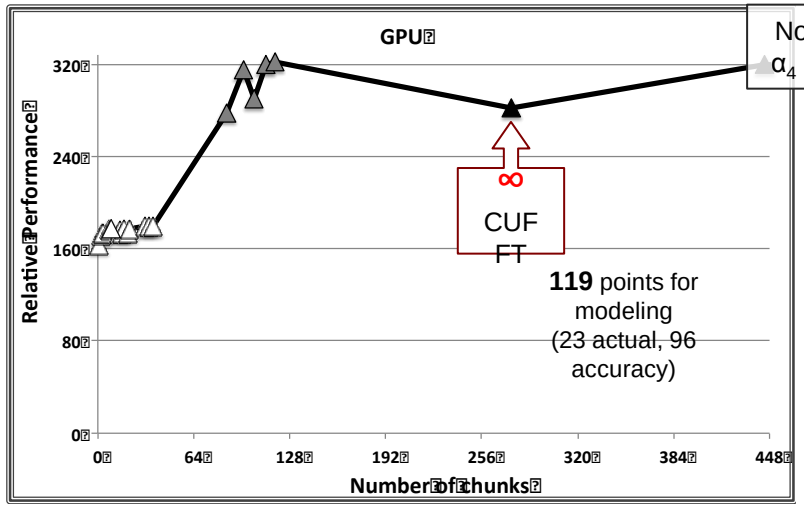
Node 2:
$\alpha_2 = 258$



Node 3:
$\alpha_3 = 241$



Node 4:
$\alpha_4 = 276$

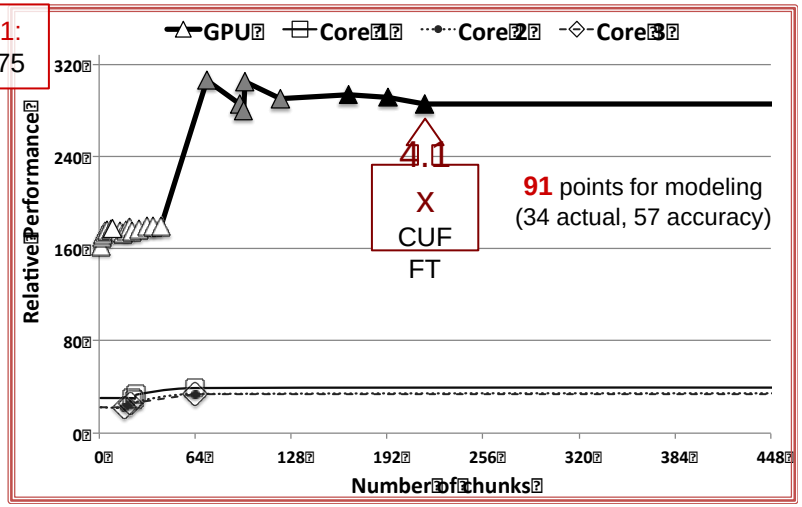


Case Study: 2D FFT Batch
ITERATION 4

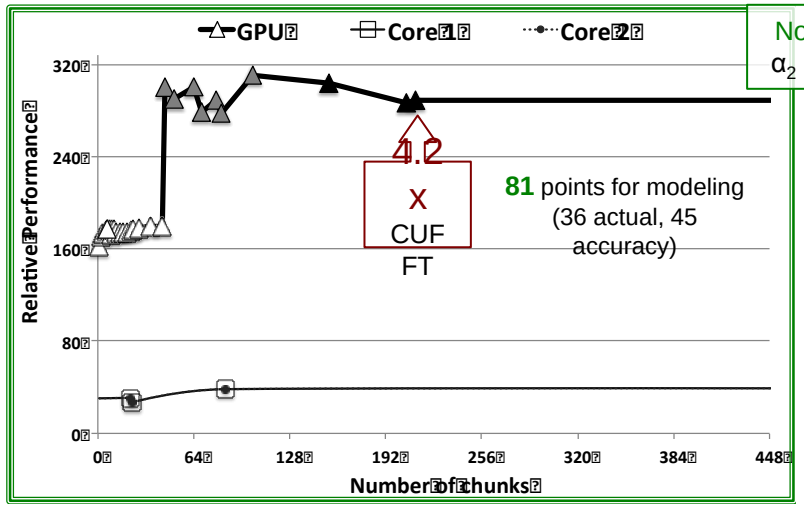
technology
from seed

inesc id lisboa

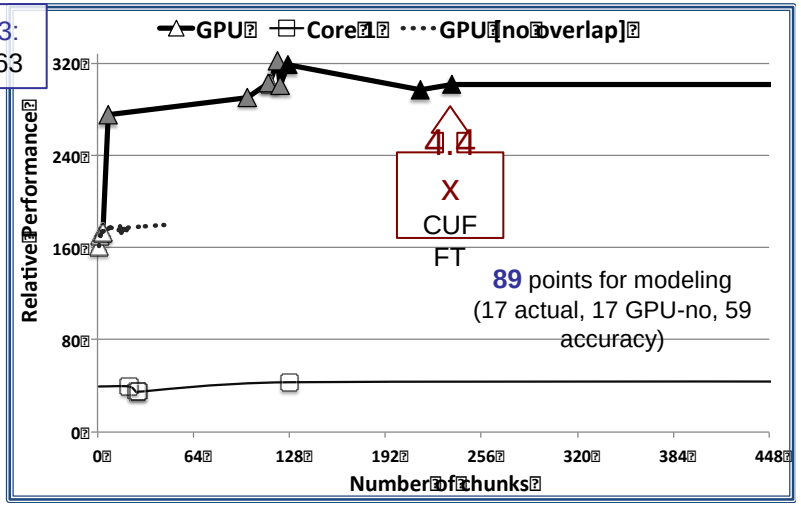
Node 1:
$\alpha_1 = 275$



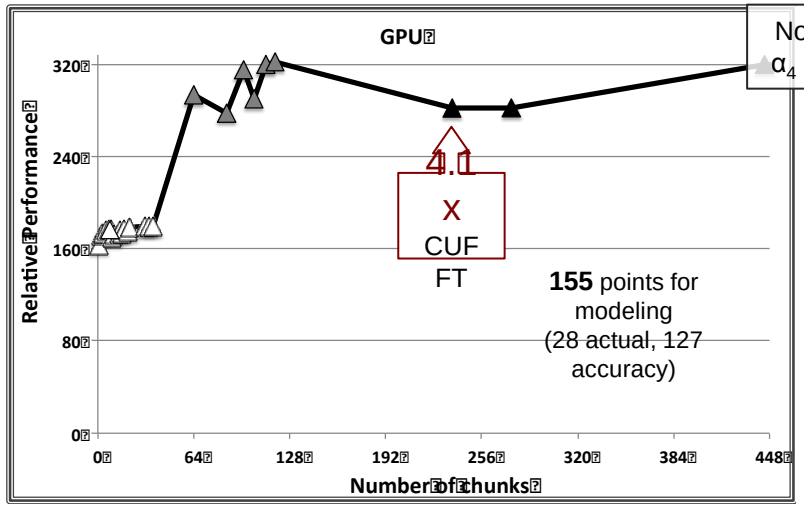
Node 2:
$\alpha_2 = 250$



Node 3:
$\alpha_3 = 263$



Node 4:
$\alpha_4 = 236$



Conclusions

Dynamic Load Balancing

- Obtained in **4 iterations** and **6.1 seconds**

Traditional Approaches for Performance Modeling

- Approximate the performance using number of points equal to the number of iterations
- In this case, **40 points** in total

Presented DDL Scheduling Approach

- Models the performance using **416 points**, in this case ~10x more than with traditional modeling
- Load balancing solution is 3x faster than the current state of the art approaches
- **In this case, obtained GPU performance is at least 4.1x better than the "optimal" CUFFT execution**

Questions?

Thank you