Efficient Execution on Heterogeneous Systems

Leonel Sousa[1],

Aleksandar Ilić[1], Frederico Pratas[1] and Pedro Trancoso[2]

work performed in the scope of HiPEAC FP7 project

[1] INESC-ID /
IST, TU Lisbon Portugal

[2] CASPER GROUP
UCY, Cyprus

High Performance Computing, Grids and Clouds Workshop, Cetraro, 2010

Motivation

COMMODITY COMPUTERS = HETEROGENEOUS SYSTEMS

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

+ SIGNIFICANT COMPUTING POWER

- Not yet explored for **COLLABORATIVE COMPUTING**
- Does it worth to use the available resources to improve real **Performance per Watt**?

- BUT HETEROGENEITY MAKES PROBLEMS MUCH MORE COMPLEX!

- **Performance modeling and load balancing**
- Different programming models, languages and implementations

Outline

technology
from seed

1. DEVELOPED COLLABORATIVE PROGRAMMING ENVIRONMENT FOR HETEROGENEOUS COMPUTERS (CPHC)

2. PERFORMANCE MODELING AND LOAD BALANCING
   – For heterogeneous systems, in particular for CPU+GPU

3. CASE STUDY: Decision-Support System benchmark TPC-H

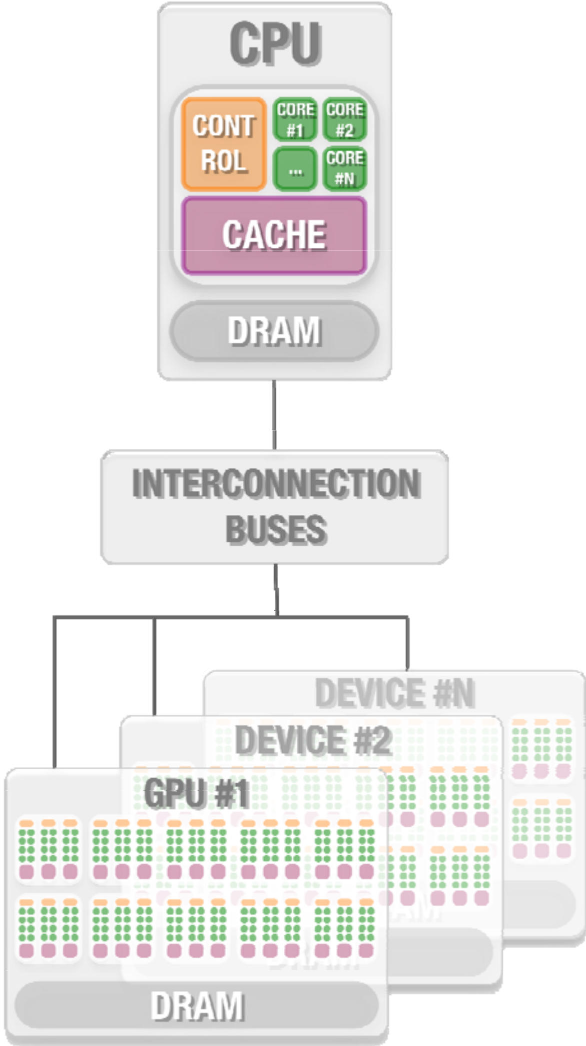
4. CONCLUSIONS AND FUTURE WORK

Desktop Heterogeneous Systems

technology
from seed

MASTER-SLAVE paradigm

- CPU (Master)
  - Global execution controller
  - Access the whole global memory
- INTERCONNECTION BUSSES
  - Limited and asymmetric communication bandwidth
  - Potential execution bottleneck
- UNDERLYING DEVICES (Slaves)
  - Different architectures and programming models
  - Computation performed using local memories



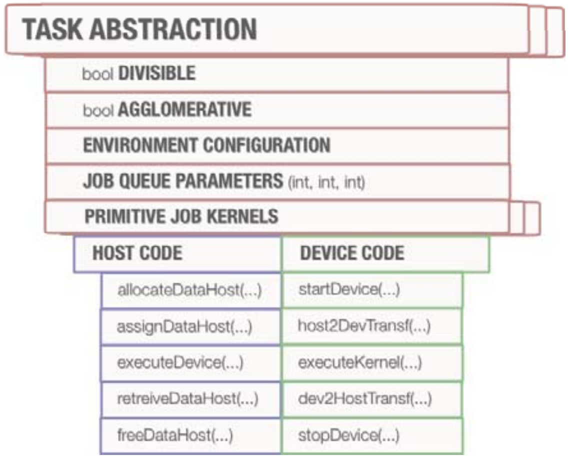
CPHC: Tasks and Primitive Jobs

technology
from seed

inesc id lisboa

TASK – basic programming unit (coarser-grained)

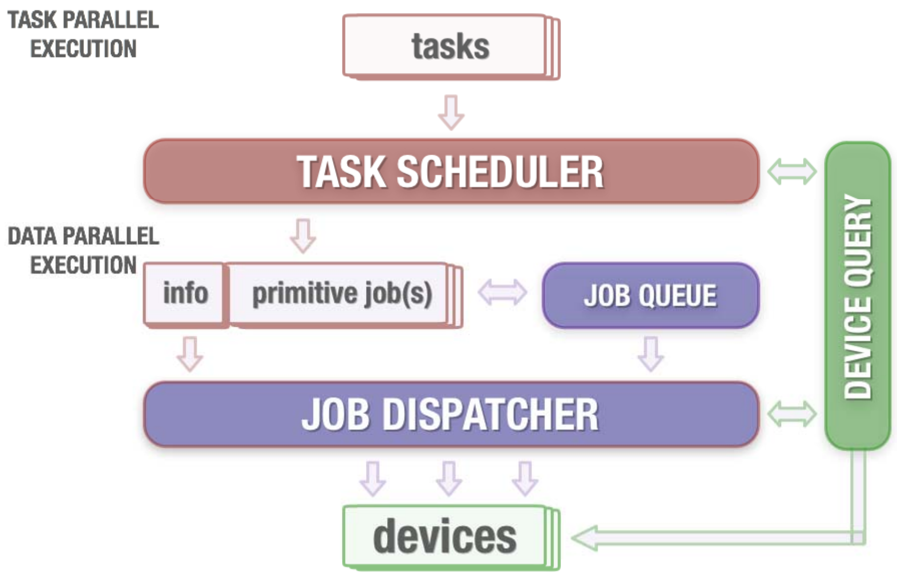
- CONFIGURATION PARAMETERS
  - Task: application, task dependency information
  - Environment: device type, number of devices…
- PRIMITIVE JOB WRAPPER
  - DIVISIBLE TASK – comprises several finer-grained Primitive Jobs
  - AGGLOMERATIVE TASK – allows grouping of Primitive Jobs



PRIMITIVE JOB – minimal program portion for parallel execution

- CONFIGURATION PARAMETERS
  - I/O and performance specifics, …
- CARRIES PER-DEVICE-TYPE IMPLEMENTATIONS
  - Vendor-specific programming models and tools
  - Specific optimization techniques

Primitive Job Granularity	Task Type	
	Divisible	Agglomerative
Coarser-grained	NO	--
Balanced	YES	NO
Finer/Balanced	YES	YES



Task Type	
Divisible	Agglomerative
NO	--
YES	NO
YES	YES

Task Level Parallelism

– TASK SCHEDULER forward independent tasks to

JOB DISPATCHER according to task and environment configuration and current platform (DEVICE QUERY)

Data Level Parallelism

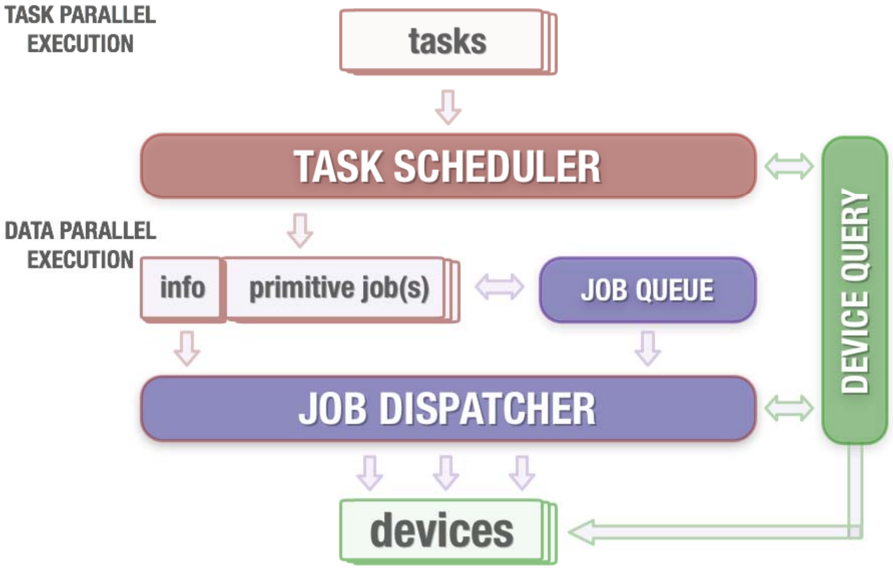
- PRIMITIVE JOBS arranged into JOB QUEUES (currently, 1D-3D grid organization) for DIVISIBLE (AGGLOMERATIVE) TASKS
- JOB DISPATCHER uses DEVICE QUERY and JOB QUEUE information to map (agglomerated) PRIMITIVE JOBS to the requested devices; then initiates and controls further execution

Nested Parallelism

- If provided, JOB DISPATCHER can be configured to perceive a certain number of cores of a multi-core as a single device

**Aleksandar Ilic and Leonel Sousa. "Collaborative Execution Environment for Heterogeneous Parallel Systems", In 12th Workshop on Advances in Parallel and Distributed Computational Models (APDCM/IPDPS 2010), April 2010.*

Collaborative Execution Environment for Heterogeneous Systems



Task Type	
Divisible	Agglomerative
NO	--
YES	NO
YES	YES

PROBLEM

How to make good DYNAMIC LOAD BALANCING decisions, using partial PERFORMANCE MODELS of the devices, while being aware of :
- application demands
- implementation specifics
- platform / device heterogeneity
- complex memory hierarchies
- limited asymmetric communication bandwidth
- …

CPHC should provide for CPU + GPU

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
  - Modeling of the overall CPU and GPU performance for different problem sizes

DYNAMIC LOAD BALANCING

- OPTIMAL DISTRIBUTION OF COMPUTATIONS (PRIMITIVE JOBS)
  - Partial estimations of the performance should be built and used to decide on optimal mapping
  - Returned solution should provide load balancing within a given accuracy

COMMUNICATION AWARENESS

- MODELING OF THE BANDWIDTH for interconnection busses DURING THE EXECUTION
  - To select problem sizes that maximize the interconnection bandwidth
  - The algorithm should be aware of asymmetric bandwidth for Host-To-Device and Device-To-Host transfers

CPU+GPU ARCHITECTURAL SPECIFICS

- Make use of ENVIRONMENT-SPECIFIC FUNCTIONS to ease performance modeling
  - Asynchronous transfers and CUDA streams to overlap communication with computation
  - Be aware of diverse capabilities of different devices, but also for devices of the same vendor (e.g. GT200 vs. Fermi)

Constant Performance Models (CPM)

- Device Performance (Speed) : constant positive number
  - Typically represents relative speed when executing a serial benchmark of a given size
- Computation Distribution : proportional to the speed of the device

Functional Performance Models (FPM)

- Device Performance (Speed) : continuous function of the problem size
  - Typically requires several benchmark runs and a significant amount of time to build it
- Computation Distribution : relies on the functional speed of the processor

FPM vs. CPM

- More Realistic : integrates important features of heterogeneous processor
  - Processor heterogeneity, the heterogeneity of memory structure, and other effects
- More Accurate Distribution of computation across heterogeneous devices
- Application-centric approach : different applications characterize speed by different functions

DATABASE APPLICATIONS : TPC-H BENCHMARK

- TRANSACTION PROCESSING PERFORMANCE COUNCIL (TPC)
  - Provides several representative queries of real database applications
  - TPC-H is a Decision Support System Benchmark used in industry
- QUERIES: specified in SQL and executed on top of DBMS
  - Database applications implement basic, well established operations, such as SCAN and JOIN

SELECTED QUERIES WITH DIFFERENT COMPLEXITIES

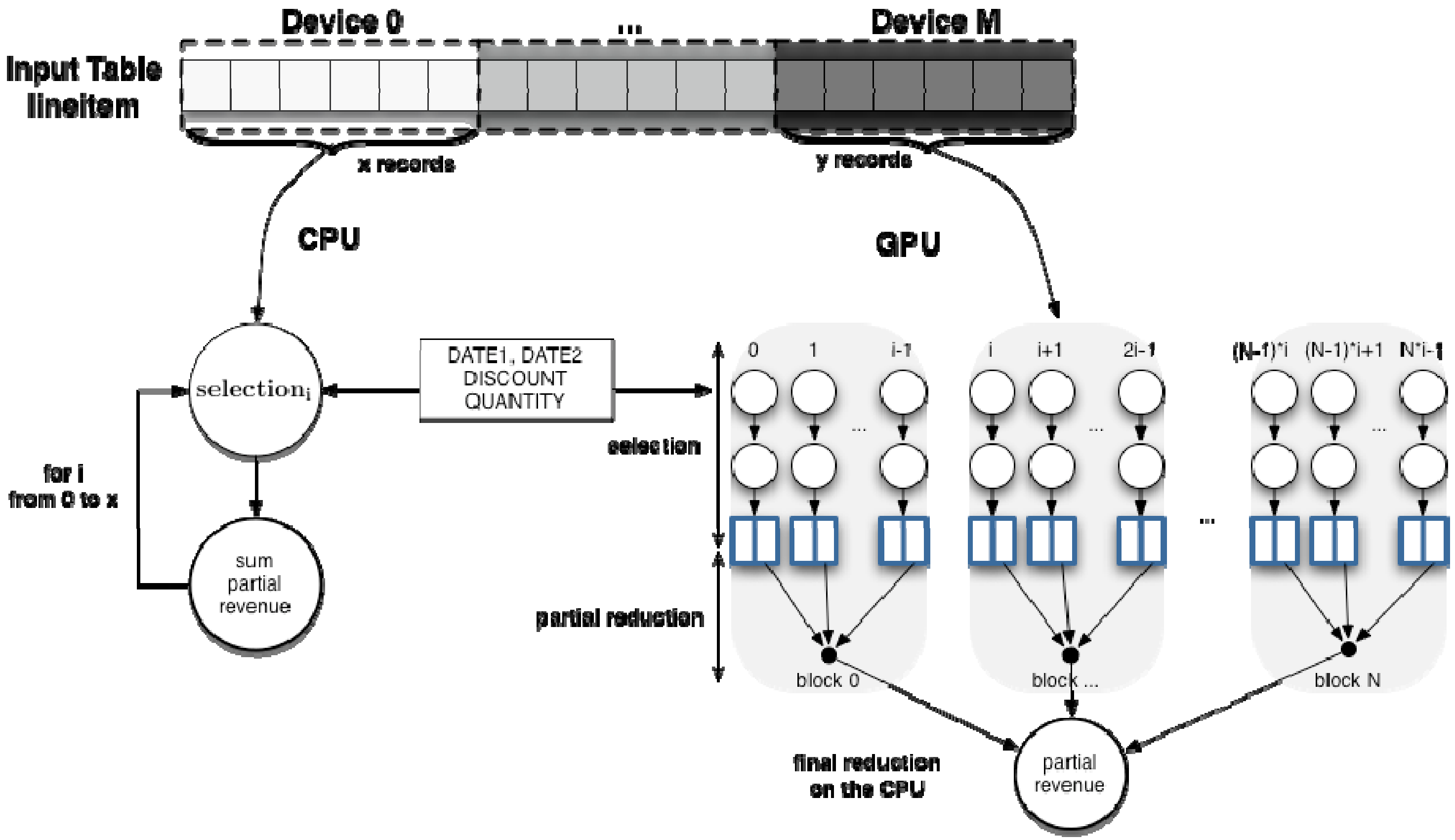
Q3: implements 2-NESTED HASHED JOINS

```
SELECT
  orderkey
FROM
  costumer, orders, lineitem
WHERE
  c_mktsegment = '[SEGMENT]' AND
  c_costumerkey = o_costumerkey AND
  l_orderkey = o_orderkey AND
  o_orderdate < date '[DATE]' AND
  l_shipdate > date '[DATE]';
```

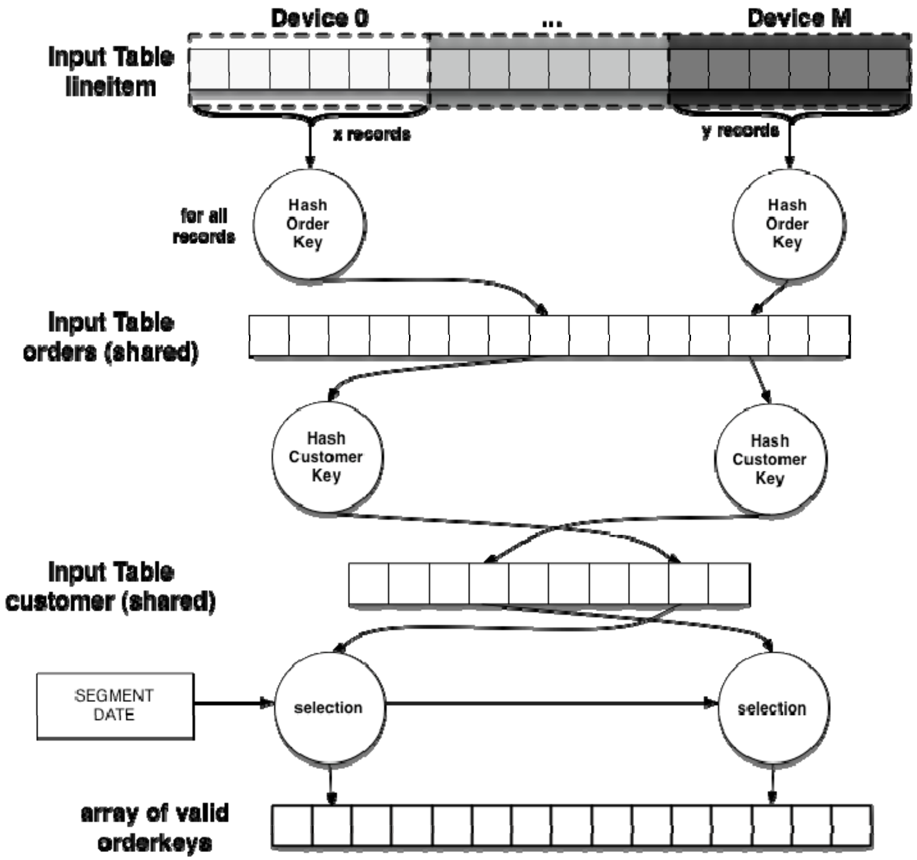
Q6: implements a SEQUENTIAL SCAN

```
SELECT
  sum(extendedprice*discount) AS revenue
FROM
  lineitem
WHERE
  shipdate >= date '[DATE1]' AND
  shipdate < date '[DATE2]' AND
  discount BETWEEN [DISCOUNT]-0.01 AND
                   [DISCOUNT]+0.01 AND
  quantity < [QUANTITY];
```

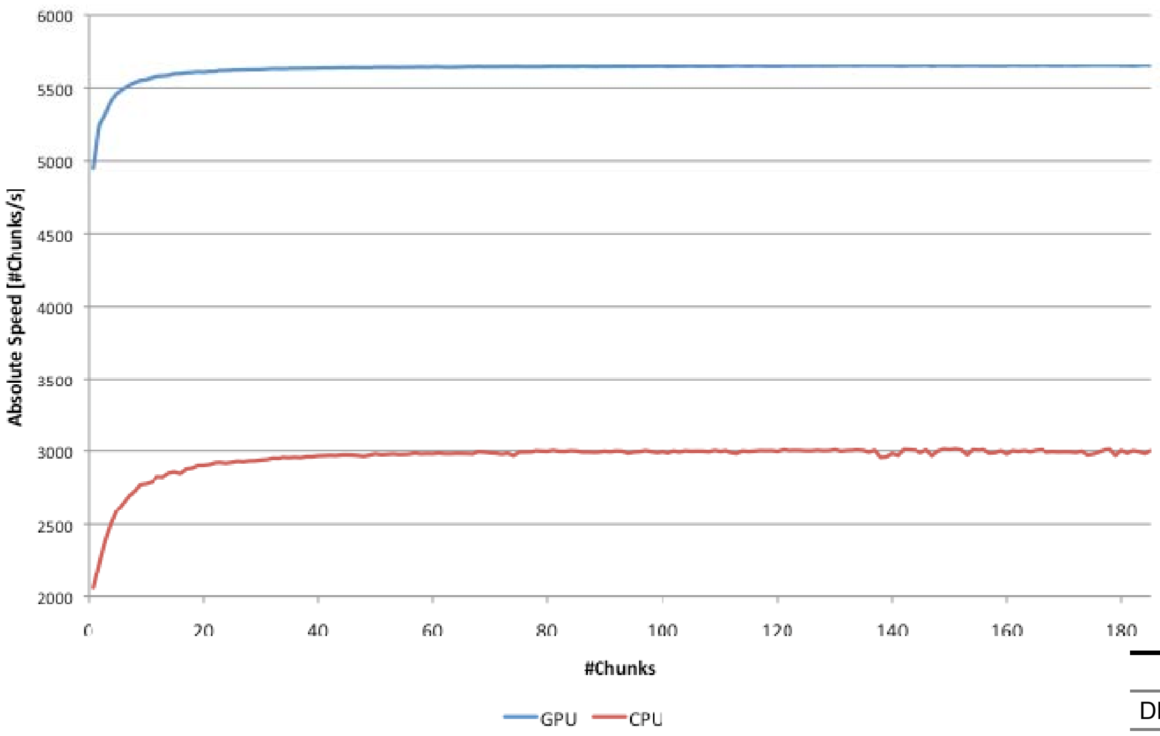
Case Study:
Q6 Query Parallelization



Case Study:
Q3 Query Parallelization



Case Study: Q3 Query
Building Full Performance Models



Full Performance Models: Per-Device Real Performance

- Experimentally obtained using CPU + GPU platform specifics
- Exhaustive search on the full range of problem sizes
- High cost of building in general!!!

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

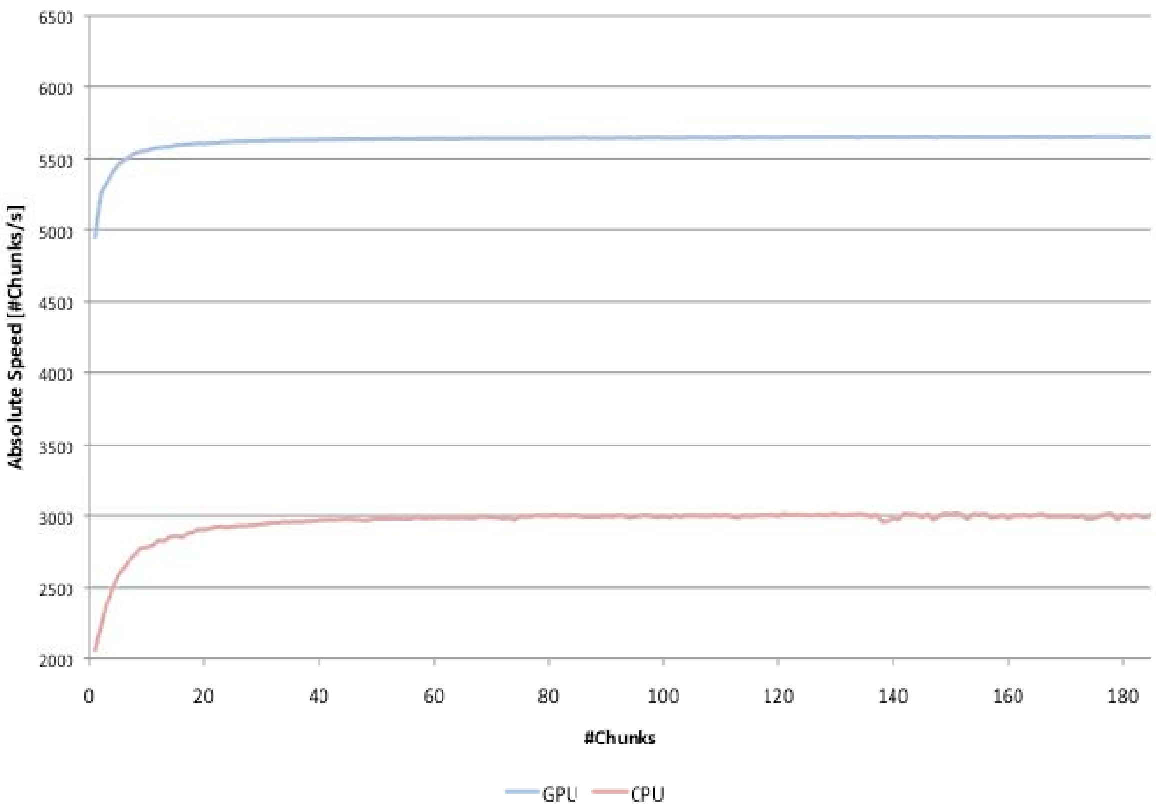
	Q3 Query
DBGEN Scale Factor	1
Input Data Size	57.8 MB (of 950 MB)
#lineitem records	6001215
#order records	1500000
#customer records	1500000
#records per chunk	32439
#chunks	185

Case Study: Q3 Query
CPU + GPU Performance Modeling (1)

technology from seed

inesc id lisboa

Performance Metric → Initialization → Approximation → Iteration



Problem Definition: Primitive Job

- n^{in} – input data requirements
- n^{out} – output data requirements
- n – number of assigned chunks

Metric : Absolute Speed*

- p devices: $P_1, P_2, \ldots, P_P$
- N total #Primitive Jobs (chunks)
- Device Load [chunks]: $n_1, n_2, \ldots, n_P$
- Absolute speed:

$$s_i(n_i) = n_i/t_i(n_i),\ 1 \le i \le p$$

Solution: Optimal Load Balancing*

Lies on the straight line that passes through the origin of coordinate system, such that:

$$x_1/s_1(x_1) = x_2/s_2(x_2) = \ldots = x_p/s_p(x_p)$$
$$x_1 + x_2 + \ldots + x_p = N$$

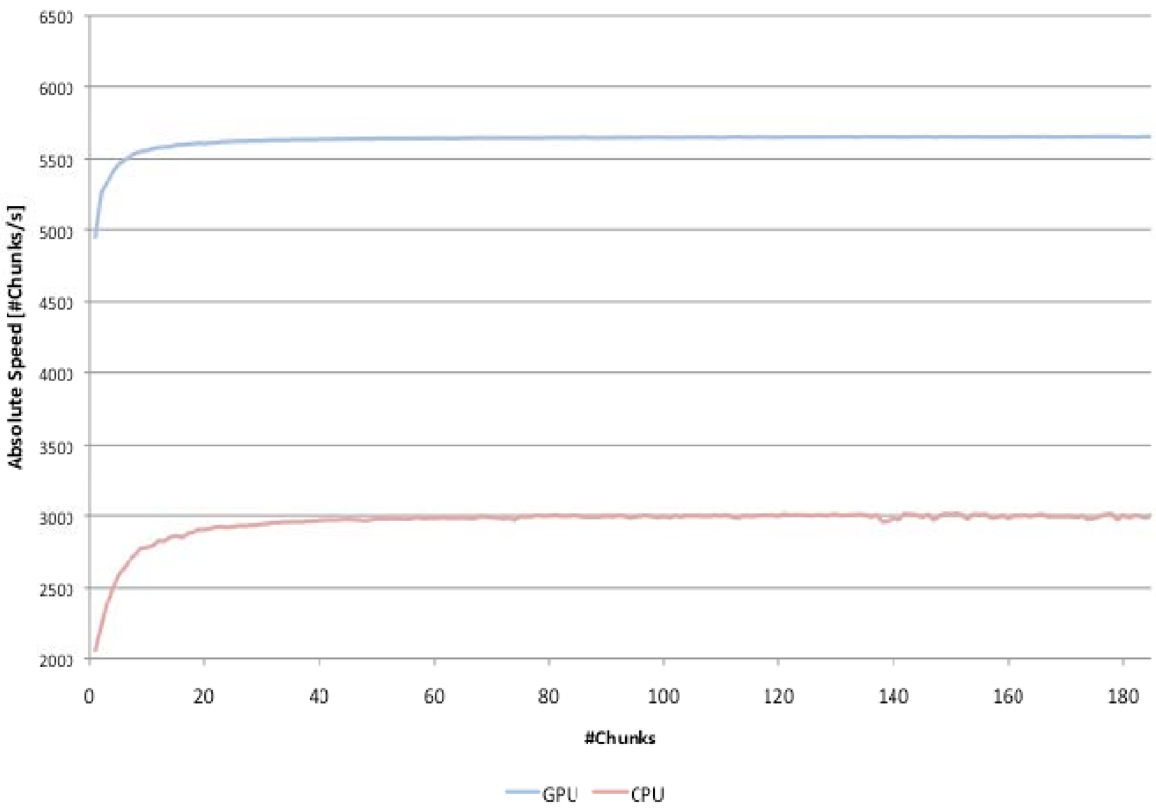
*Lastovetsky, A., and R. Reddy, "Distributed Data Partitioning for Heterogeneous Processors Based on Partial Estimation of their Functional Performance Models", HeteroPar 2009, Netherlands, Lecture Notes in Computer Science, vol. 6043, Springer, pp. 91-101, 25/9/2009, 2010.

Case Study: Q3 Query
CPU + GPU Performance Modeling (2)

technology
from seed

inesc id lisboa

Performance Metric → **Initialization** → Approximation → Iteration



① All the `P` computational units execute `N/p` chunks **in parallel**

$$n_i = N/p, \quad 1 \le i \le p$$

② IF `(device is GPU) AND (task is Divisible and Agglomerative)`
THEN go to 3
ELSE go to 4

③ Split the given computational load into streams and use asynchronous transfers to overlap communication with computation

Case Study: Q3 Query
CPU + GPU Performance Modeling (3)

technology
from seed

inesc id lisboa

Performance Metric → Initialization → Approximation → Iteration

Streaming

– SUBDIVIDE n_i computational chunks using DIV2 STRATEGY
  – No prior knowledge of the performance of an application!
  – Each stream has half the load of the previous stream
  – The algorithm may continue to split the workload until assigning a stream with load equal to 1

– BANDWIDTH-AWARE DIV2 STRATEGY
  – Interconnection bandwidth depends on the amount of data that should be transferred and not on the computational demands
  – Run small pre-calibration tests for HOST-TO-DEVICE AND DEVICE-TO-HOST transfers
  – Tests can be stopped when saturation points are detected, or when transfers reach the desired value (e.g. 60% of the theoretical maximum)
  – CASE STUDY: $n^{min\_size} = 1$
    – Primary agglomeration into the chunks is performed in order to be bandwidth aware

Stream 0
Stream 1
Stream 2

1 All the P computational units execute N/p chunks **in parallel**

$$n_i = N/p, \quad 1 \le i \le p$$

2 IF (device is GPU) AND (task is Divisible and Agglomerative)
THEN go to 3
ELSE go to 4

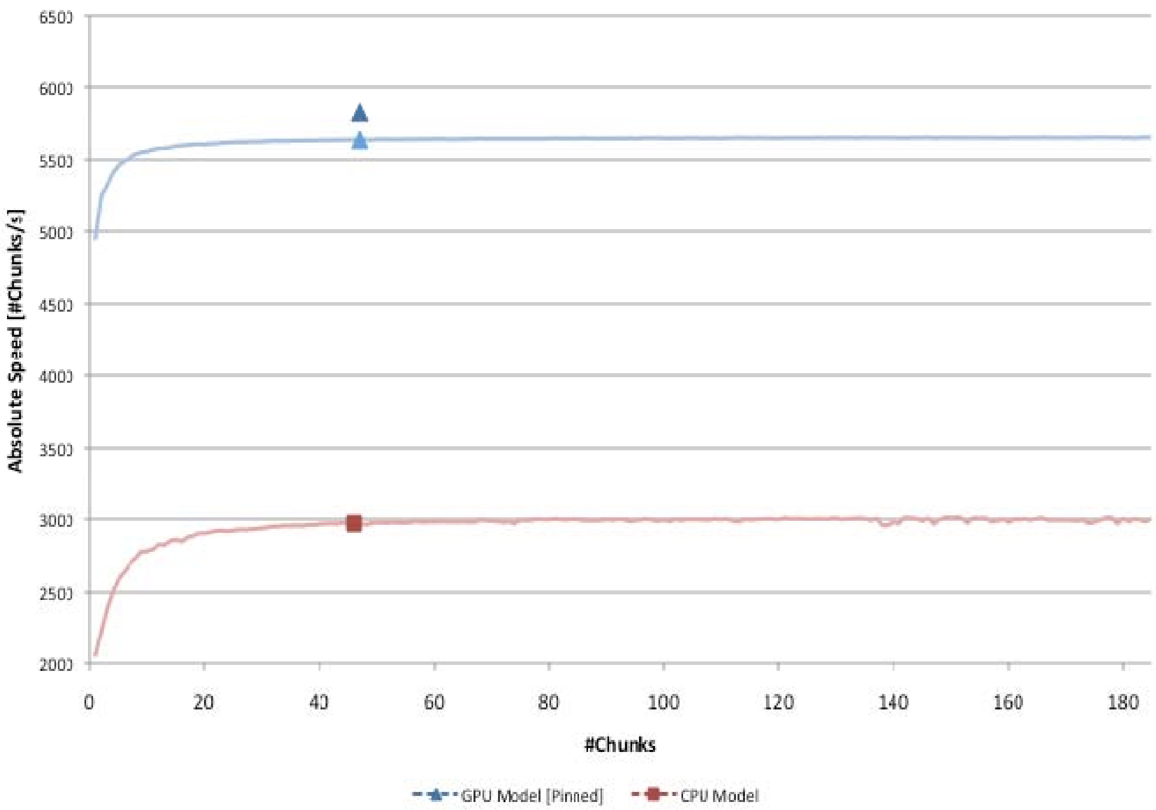
3 Split the given computational load into streams and use asynchronous transfers to overlap communication with computation

Case Study: Q3 Query
CPU + GPU Performance Modeling (4)

technology
from seed

inesc id lisboa

Performance Metric → **Initialization** → Approximation → Iteration



1. All the P computational units execute N/p chunks **in parallel**

   $$n_i = N/p, \quad 1 \le i \le p$$

2. IF (device is GPU) AND (task is Divisible and Agglomerative)
   THEN go to 3
   ELSE go to 4

3. Split the given computational load into streams and use asynchronous transfers to overlap communication with computation

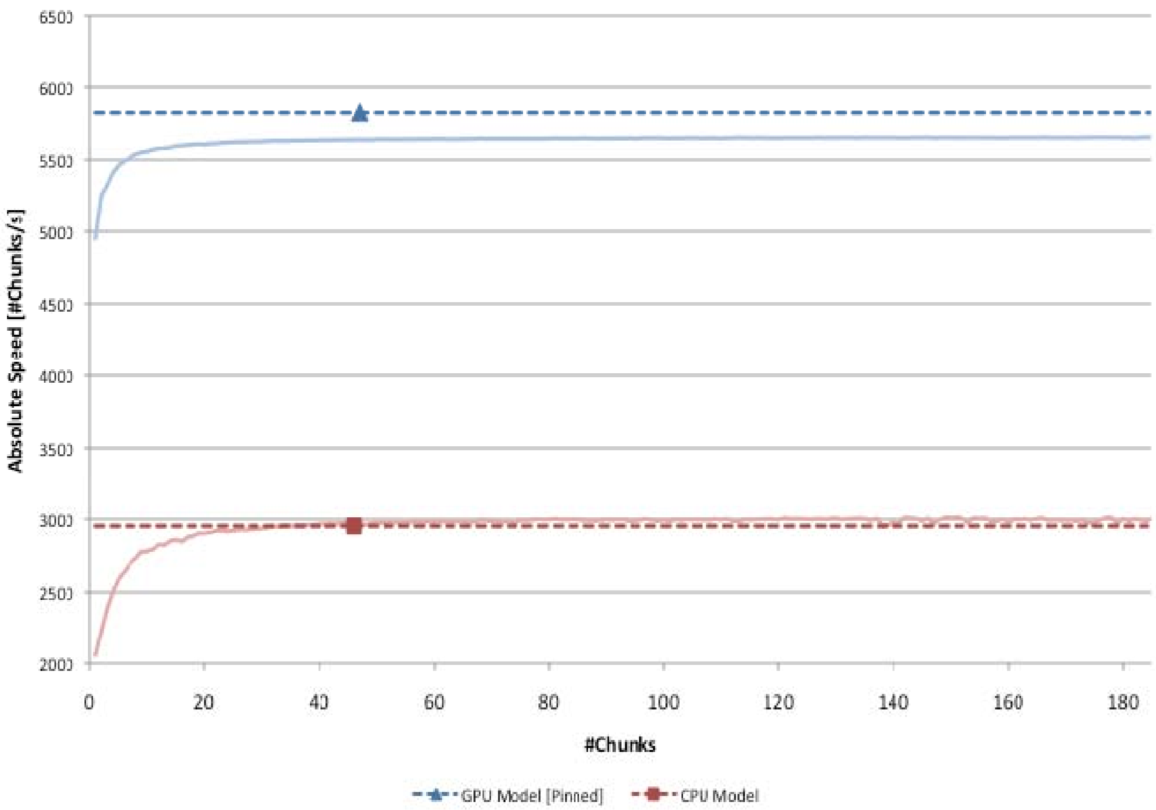
④ Execute & record execution times: $t_i(N/p)$

⑤ IF $\max_{1\le i,j\le p}\{((t_i(N/p)-t_j(N/p))/t_i(N/p)\}\le\varepsilon$
THEN even distribution solves the problem and the algorithm stops;
ELSE performance of devices is calculated, such that:

$$s_i(N/p) = (N/p)/t_i(N/p), \quad 1 \le i \le p$$

Case Study: Q3 Query
CPU + GPU Performance Modeling (5)

Performance Metric → Initialization → **Approximation** → Iteration



① Traditional approach*: **Performance** of each device is **modeled** as a **constant**

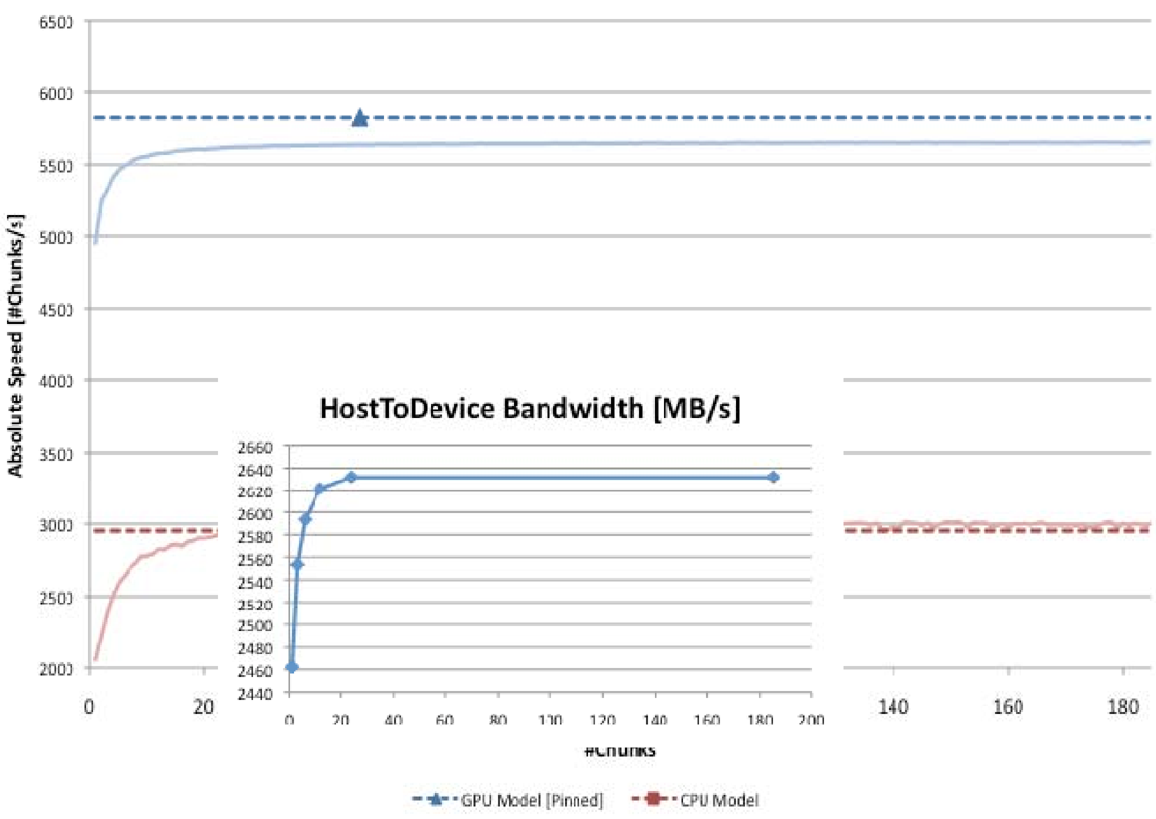
$$s_i(x) = s_i(N/p), \quad 1 \le i \le p$$

Lastovetsky, A., and R. Reddy, "Distributed Data Partitioning for Heterogeneous Processors Based on Partial Estimation of their Functional Performance Models", HeteroPar 2009, Netherlands, Lecture Notes in Computer Science, vol. 6043, Springer, pp. 91-101, 25/9/2009, 2010.

Case Study: Q3 Query
CPU + GPU Performance Modeling (6)

technology
from seed

Performance Metric → Initialization → **Approximation** → Iteration



1 Traditional approach: **Performance** of each device is **modeled** as a **constant**

$$s_i(x) = s_i(N/p), \ 1 \le i \le p$$

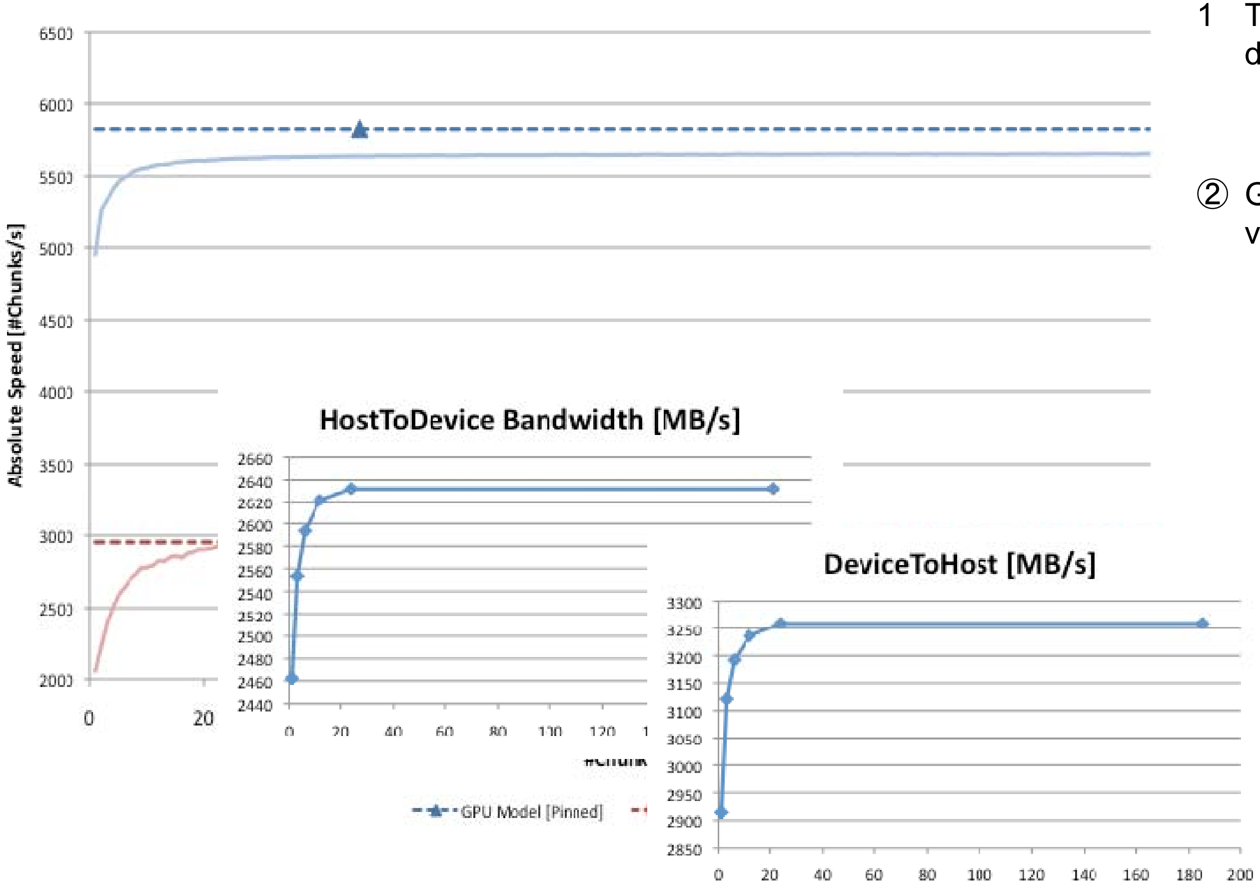
② GPU-specific modeling: Using the obtained values from streaming execution

– *HostToDevice* Bandwidth

Case Study: Q3 Query
CPU + GPU Performance Modeling (7)

technology from seed

Performance Metric → Initialization → **Approximation** → Iteration



1 Traditional approach: **Performance** of each device is **modeled** as a **constant**

$$s_i(x) = s_i(N/p), \quad 1 \le i \le p$$

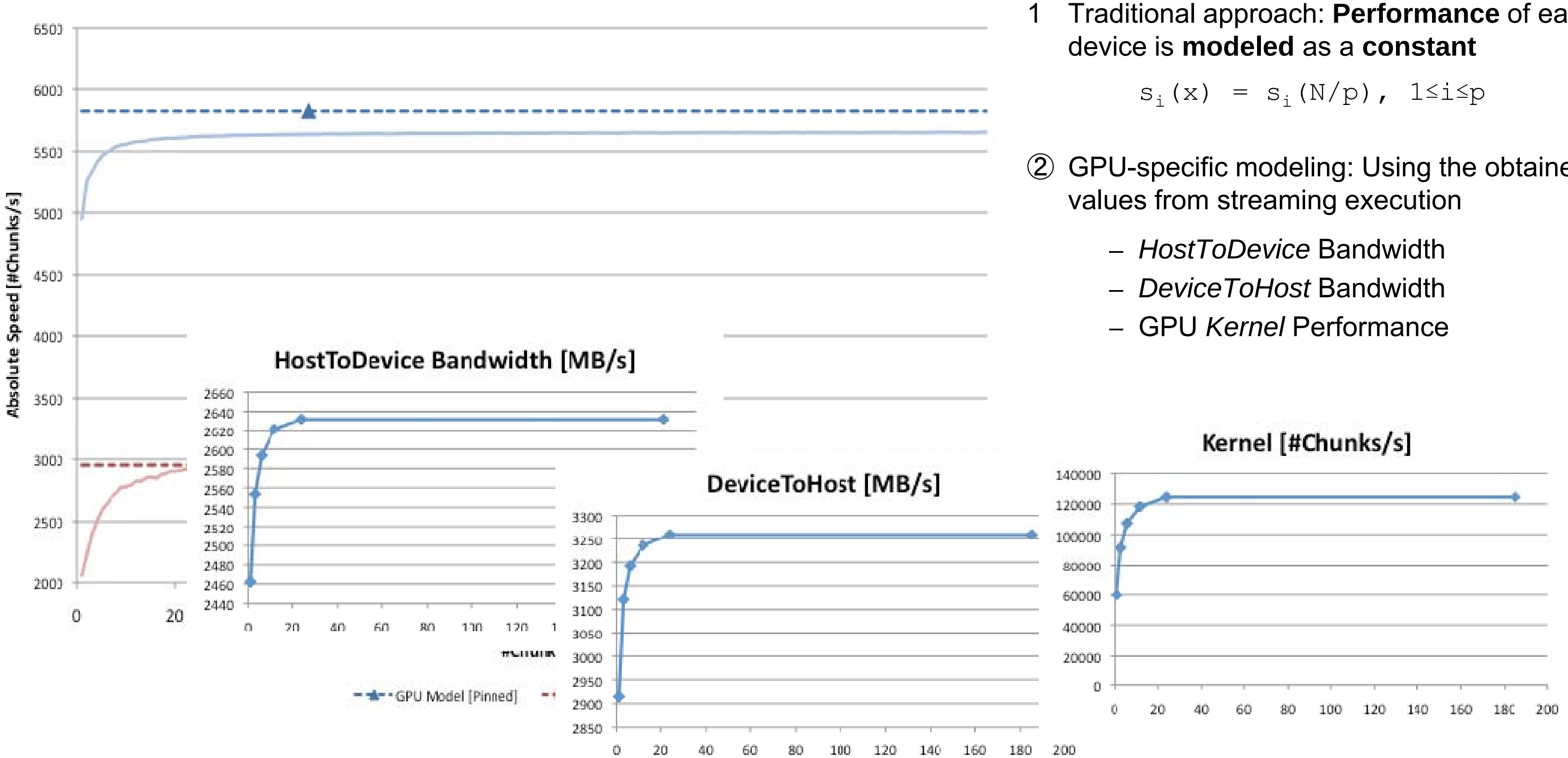
② GPU-specific modeling: Using the obtained values from streaming execution

- *HostToDevice* Bandwidth
- *DeviceToHost* Bandwidth

Case Study: Q3 Query
CPU + GPU Performance Modeling (8)

technology
from seed

Performance Metric → Initialization → **Approximation** → Iteration



1 Traditional approach: **Performance** of each device is **modeled** as a **constant**

$$s_i(x) = s_i(N/p), \quad 1 \le i \le p$$

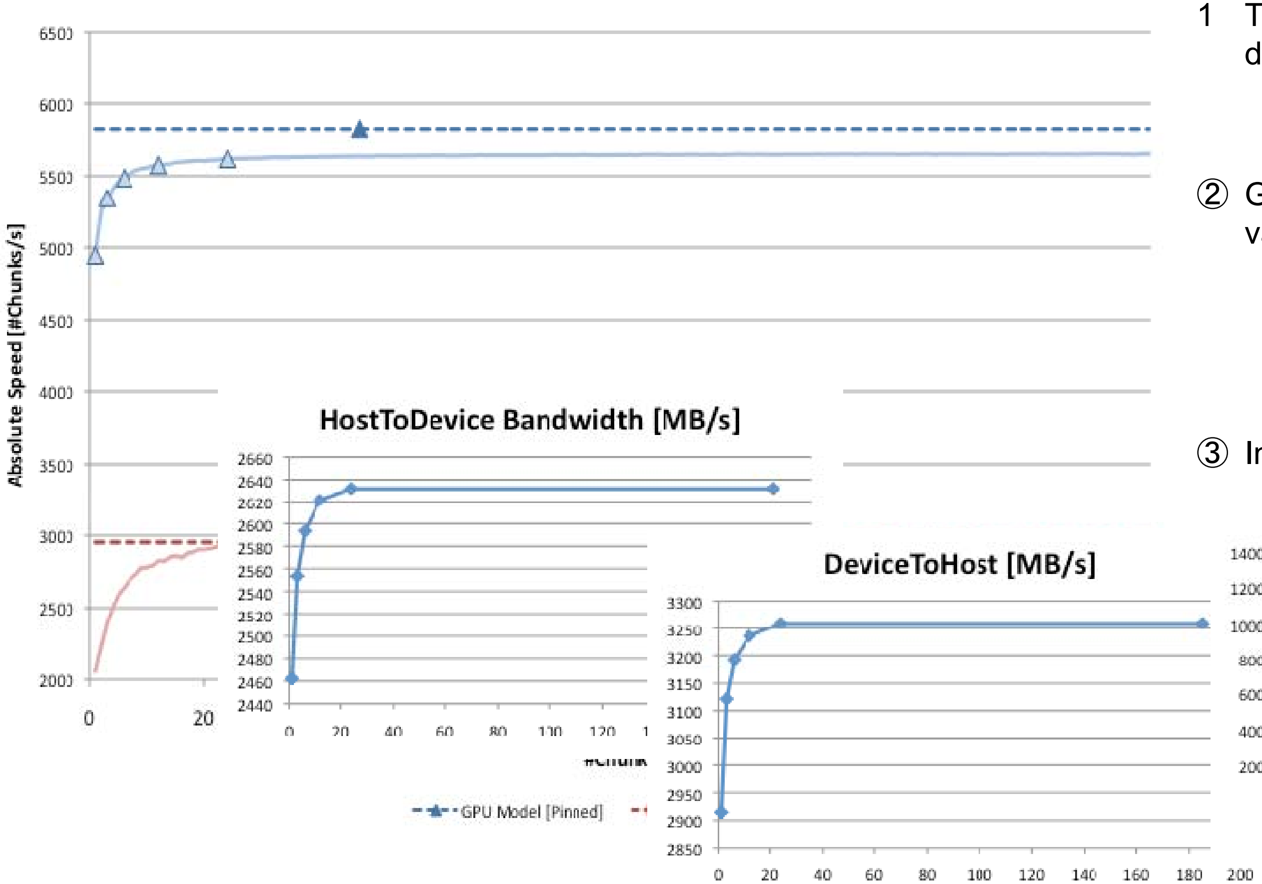
② GPU-specific modeling: Using the obtained values from streaming execution

- *HostToDevice* Bandwidth
- *DeviceToHost* Bandwidth
- GPU *Kernel* Performance

Case Study: Q3 Query
CPU + GPU Performance Modeling (9)

technology
from seed

Performance Metric → Initialization → **Approximation** → Iteration



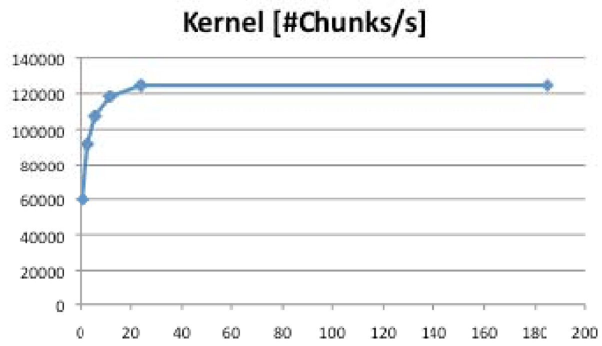
1 Traditional approach: **Performance** of each device is **modeled** as a **constant**

$$s_i(x) = s_i(N/p),\ 1 \le i \le p$$

② GPU-specific modeling: Using the obtained values from streaming execution

– *HostToDevice* Bandwidth
– *DeviceToHost* Bandwidth
– GPU *Kernel* Performance

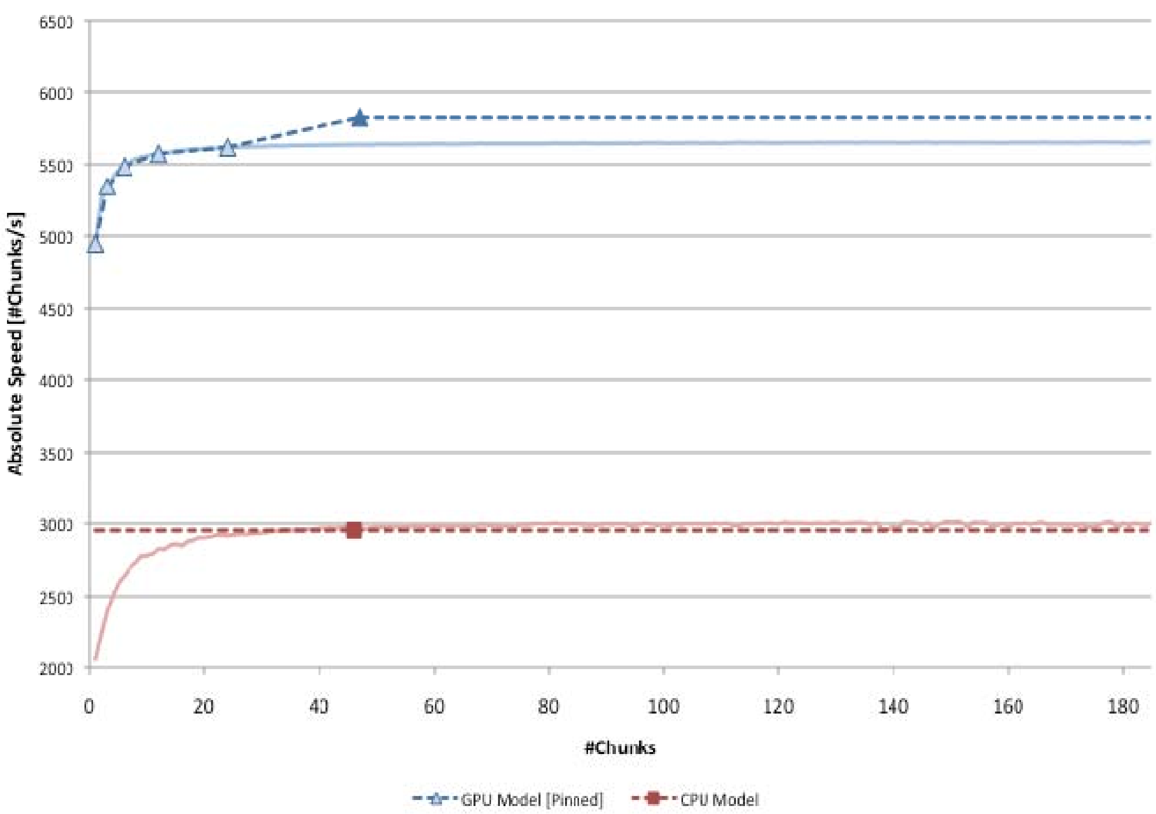
③ Incorporate streaming results



Case Study: Q3 Query
CPU + GPU Performance Modeling(10)

technology from seed

Performance Metric → Initialization → **Approximation** → Iteration



1 Traditional approach: **Performance** of each device is **modeled** as a **constant**

$$s_i(x) = s_i(N/p),\ 1 \le i \le p$$

2 GPU-specific modeling: Using the obtained values from streaming execution

- *HostToDevice* Bandwidth
- *DeviceToHost* Bandwidth
- GPU *Kernel* Performance

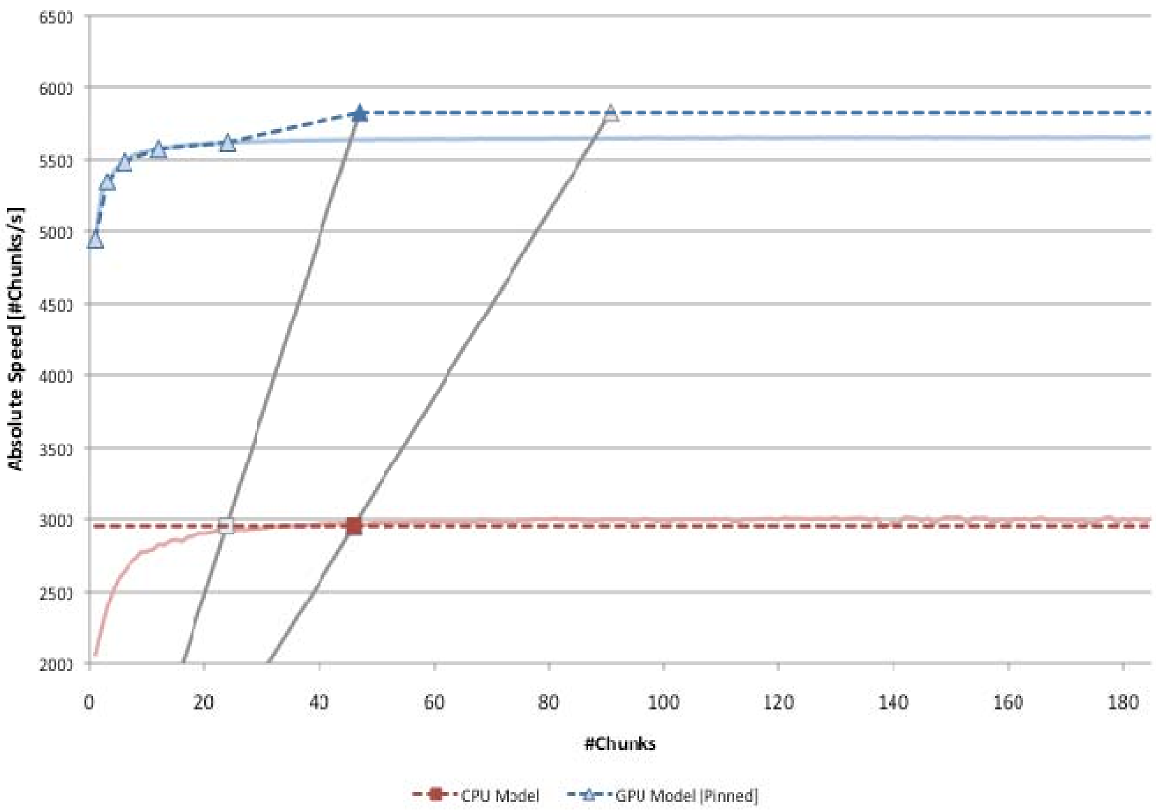
③ Incorporate streaming results

Case Study: Q3 Query
CPU + GPU Performance Modeling(11)

technology
from seed

inesc id lisboa

Performance Metric → Initialization → Approximation → **Iteration**



① Draw Upper U and Lower L lines through the following points:

$(0,0),\ (N/p,\ \max_i\{s_i(N/p)\})$

$(0,0),\ (N/p,\ \min_i\{s_i(N/p)\})$

② Let $x_i^{(U)}$ and $x_i^{(L)}$ be the intersections with $s_i(x)$

IF exists $x_i^{(L)} - x_i^{(U)} \geq 1$

THEN go to 3

ELSE go to 5

③ Bisect the angle between U and L by the line M, and calculate intersections $x_i^{(M)}$

④ IF $\Sigma_i\ x_i^{(M)} \leq N$

THEN $U=M$

ELSE $L=M$

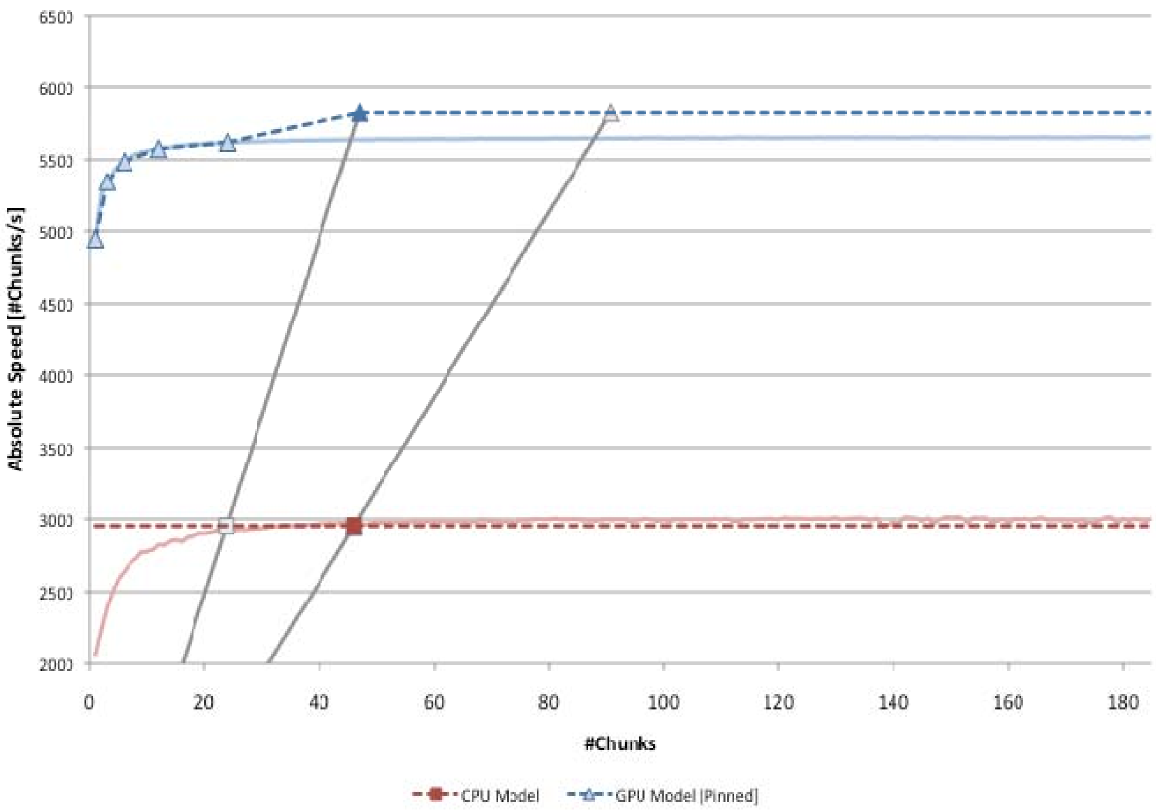
REPEAT 2

**Lastovetsky, A., and R. Reddy, "Data Partitioning with a Functional Performance Model of Heterogeneous Processors", International Journal of High Performance Computing Applications, vol. 21, issue 1: Sage, pp. 76-90, 2007*

Case Study: Q3 Query
CPU + GPU Performance Modeling(12)

technology
from seed

Performance Metric → Initialization → Approximation → **Iteration**



1 Draw Upper U and Lower L lines through the following points:

$(0,0),\ (N/p,\ \max_i\{s_i(N/p)\})$

$(0,0),\ (N/p,\ \min_i\{s_i(N/p)\})$

2 Let $x_i^{(U)}$ and $x_i^{(L)}$ be the intersections with $s_i(x)$
IF exists $x_i^{(L)} - x_i^{(U)} \geq 1$
THEN go to 3
ELSE go to 5

3 Bisect the angle between U and L by the line M, and calculate intersections $x_i^{(M)}$

4 IF $\Sigma_i\ x_i^{(M)} \leq N$
THEN $U=M$
ELSE $L=M$
REPEAT 2

⑤ Employ **streaming strategy** on the calculated workload value

Case Study: Q3 Query
CPU + GPU Performance Modeling(13)

Performance Metric → Initialization → Approximation → Iteration

Streaming

- STREAMING STRATEGY
  - Results obtained using DIV2 STRATEGY give the possibility to characterize the application demands (e.g. communication-to-computation ratio)
  - Workload size for the next stream should be chosen in order to OVERLAP TRANSFERS WITH COMPUTATION in the previous stream
- BANDWIDTH-AWARE STREAMING STRATEGY
  - Reuses the MINIMAL WORKLOAD SIZE FROM DIV2 STRATEGY (obtained via HOST-TO-DEVICE and DEVICE-TO-HOST tests)

    IF $(n^{curr} \geq n^{min\_size})$
    THEN use strategy (cont. overlapping)
    ELSE restart strategy on n^{curr} load
  - If the load drops below n^{min_size}, strategy is restarted on the remaining load

1 Draw Upper U and Lower L lines through the following points:

$(0,0)$, $(N/p, \max_i\{s_i(N/p)\})$

$(0,0)$, $(N/p, \min_i\{s_i(N/p)\})$

2 Let $x_i^{(U)}$ and $x_i^{(L)}$ be the intersections with $s_i(x)$

IF exists $x_i^{(L)} - x_i^{(U)} \geq 1$

THEN go to 3

ELSE go to 5

3 Bisect the angle between U and L by the line M, and calculate intersections $x_i^{(M)}$

4 IF $\Sigma_i\ x_i^{(M)} \leq N$

THEN U=M

ELSE L=M

REPEAT 2

5 Employ **streaming strategy** on the calculated workload value

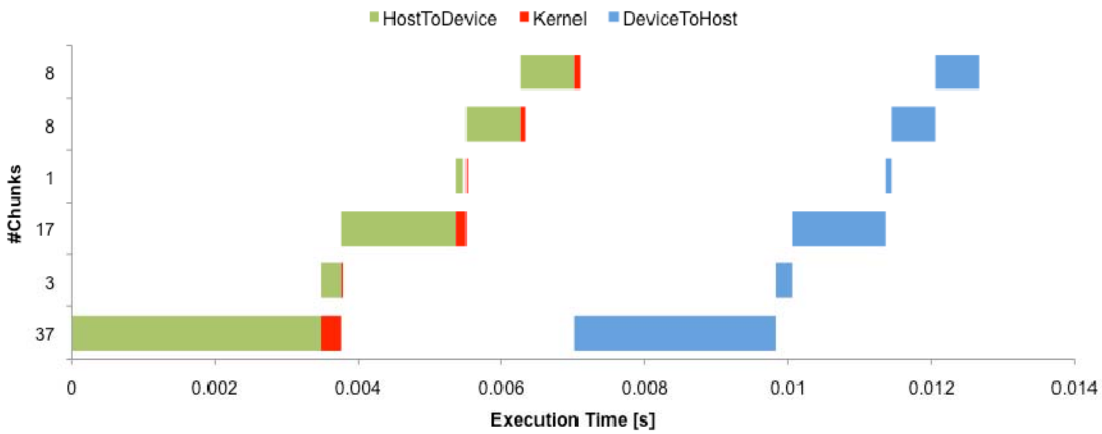
Case Study: Q3 Query
CPU + GPU Performance Modeling(14)

technology
from seed

inesc id lisboa

Performance Metric → Initialization → Approximation → **Iteration**
Streaming

- CASE STUDY: **Q3 QUERY**
  - About 96% of total execution time goes on data transfers
    - HOSTTODEVICE Transfers – 53%
    - KERNEL Execution – 4%
    - DEVICETOHOST Transfers – 43%
- BANDWIDTH-AWARE STREAMING STRATEGY
  - $n^{min\_size} = 1$, overlap HOSTTODEVICE transfers and KERNEL execution between two streams



1 Draw Upper U and Lower L lines through the following points:

$(0,0),\ (N/p,\ \max_i\{s_i(N/p)\})$

$(0,0),\ (N/p,\ \min_i\{s_i(N/p)\})$

2 Let $x_i^{(U)}$ and $x_i^{(L)}$ be the intersections with $s_i(x)$
IF exists $x_i^{(L)} - x_i^{(U)} \geq 1$
THEN go to 3
ELSE go to 5

3 Bisect the angle between U and L by the line M, and calculate intersections $x_i^{(M)}$

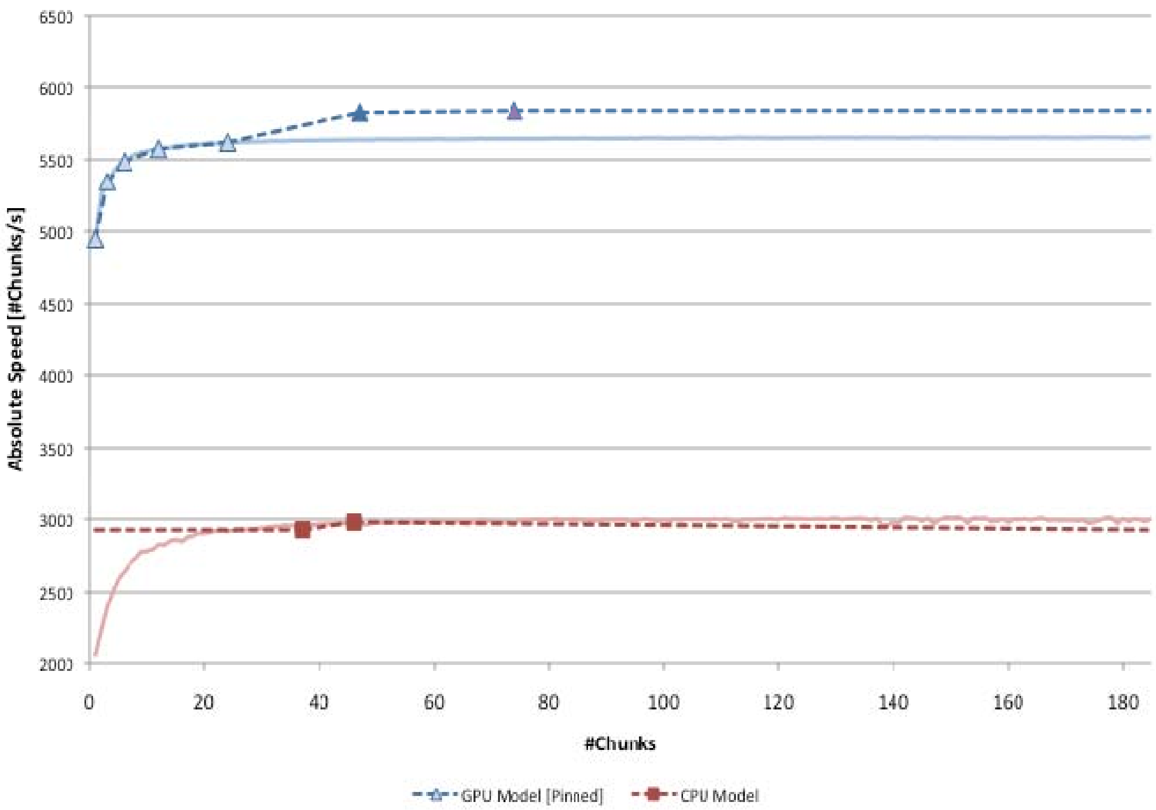
4 IF $\Sigma_i\ x_i^{(M)} \leq N$
THEN $U{=}M$
ELSE $L{=}M$
REPEAT 2

⑤ Employ **streaming strategy** on the calculated workload value

Case Study: Q3 Query
CPU + GPU Performance Modeling(15)

technology from seed

Performance Metric → Initialization → Approximation → **Iteration**



1. Draw Upper U and Lower L lines through the following points:

   $(0,0),\ (N/p,\ \max_i\{s_i(N/p)\})$

   $(0,0),\ (N/p,\ \min_i\{s_i(N/p)\})$

2. Let $x_i^{(U)}$ and $x_i^{(L)}$ be the intersections with $s_i(x)$

   IF exists $x_i^{(L)} - x_i^{(U)} \geq 1$

   THEN go to 3

   ELSE go to 5

3. Bisect the angle between U and L by the line M, and calculate intersections $x_i^{(M)}$

4. IF $\Sigma_i\ x_i^{(M)} \leq N$

   THEN $U=M$

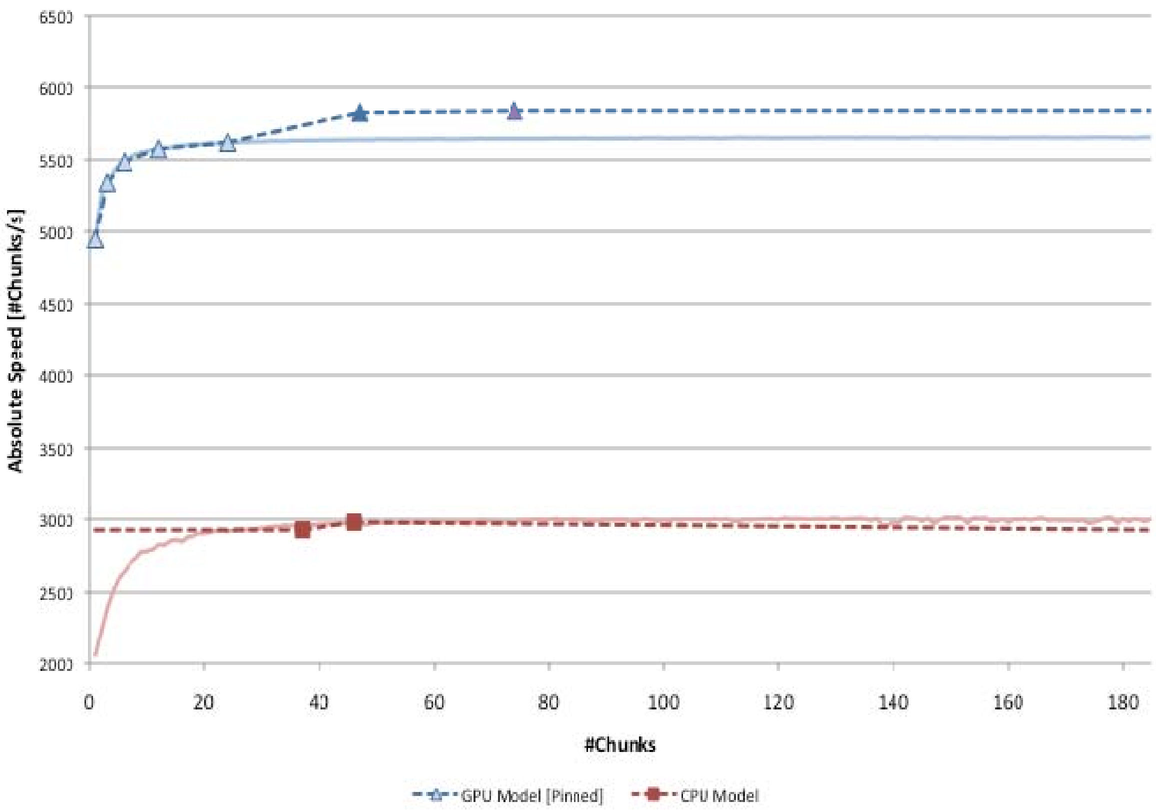
   ELSE $L=M$

   REPEAT 2

5. Employ **streaming strategy** on the calculated workload value

Case Study: Q3 Query
CPU + GPU Performance Modeling(16)

Performance Metric → Initialization → **Approximation** → Iteration



① Refine performance models with the newly obtained results

2 GPU-specific modeling: Using the obtained values from streaming execution

- *HostToDevice* Bandwidth
- *DeviceToHost* Bandwidth
- GPU *Kernel* Performance

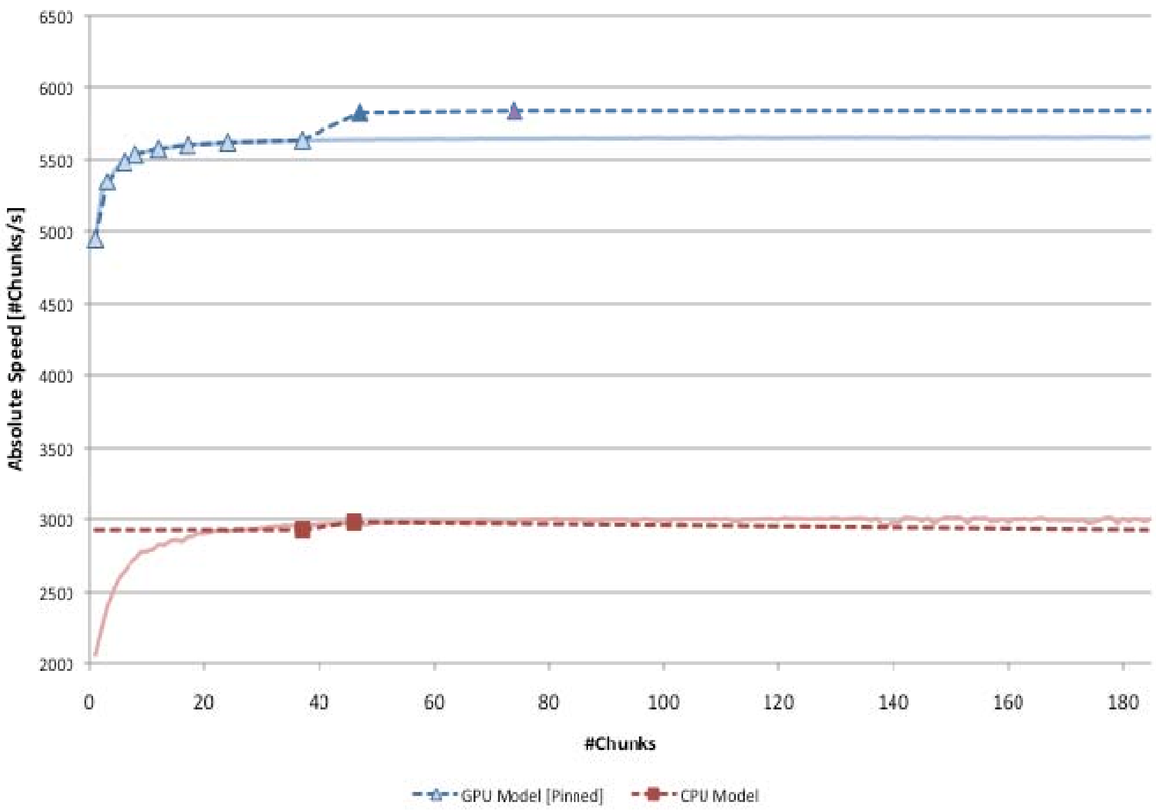
3 Incorporate streaming results

Case Study: Q3 Query
CPU + GPU Performance Modeling(17)

technology
from seed

inesc id lisboa

Performance Metric → Initialization → **Approximation** → Iteration



1 Refine performance models with the newly obtained results

② GPU-specific modeling: Using the obtained values from streaming execution

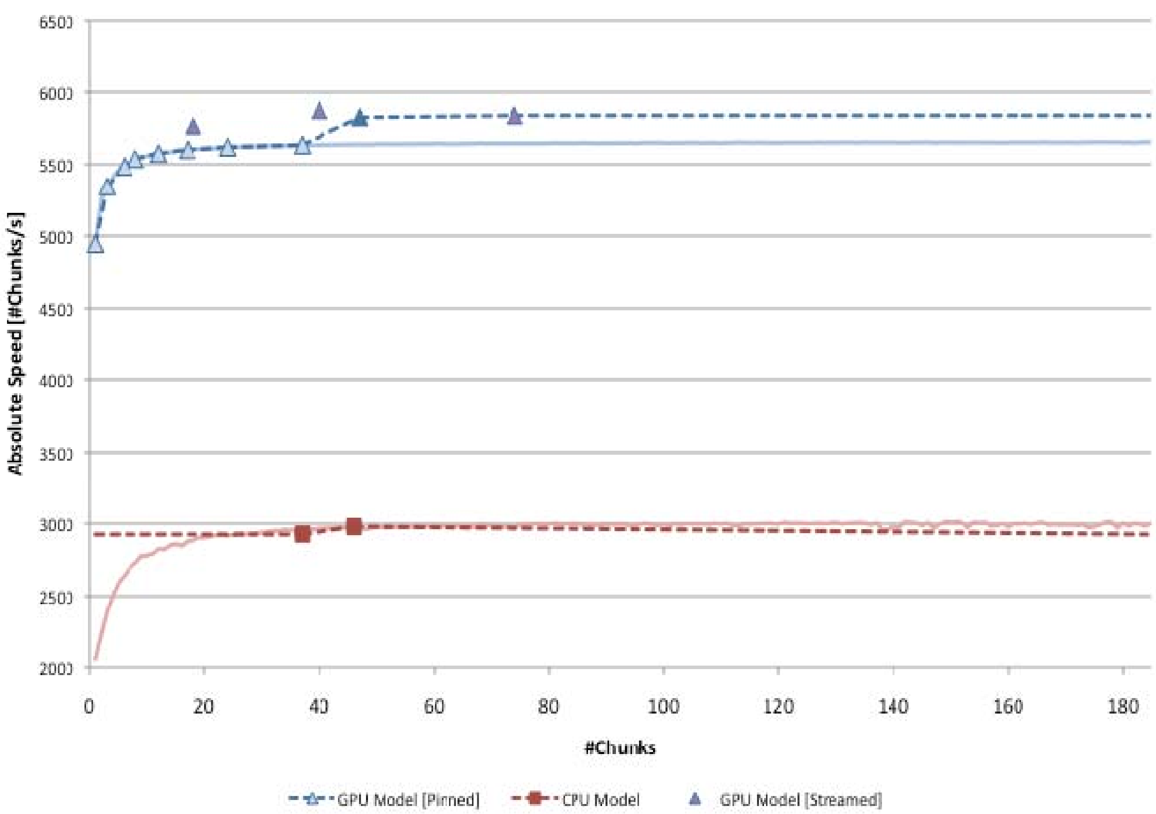
– *HostToDevice* Bandwidth
– *DeviceToHost* Bandwidth
– GPU *Kernel* Performance

③ Incorporate streaming results

– for each stream

Case Study: Q3 Query
CPU + GPU Performance Modeling(18)

Performance Metric → Initialization → **Approximation** → Iteration



1 Refine performance models with the newly obtained results

② GPU-specific modeling: Using the obtained values from streaming execution

- *HostToDevice* Bandwidth
- *DeviceToHost* Bandwidth
- GPU *Kernel* Performance

③ Incorporate streaming results

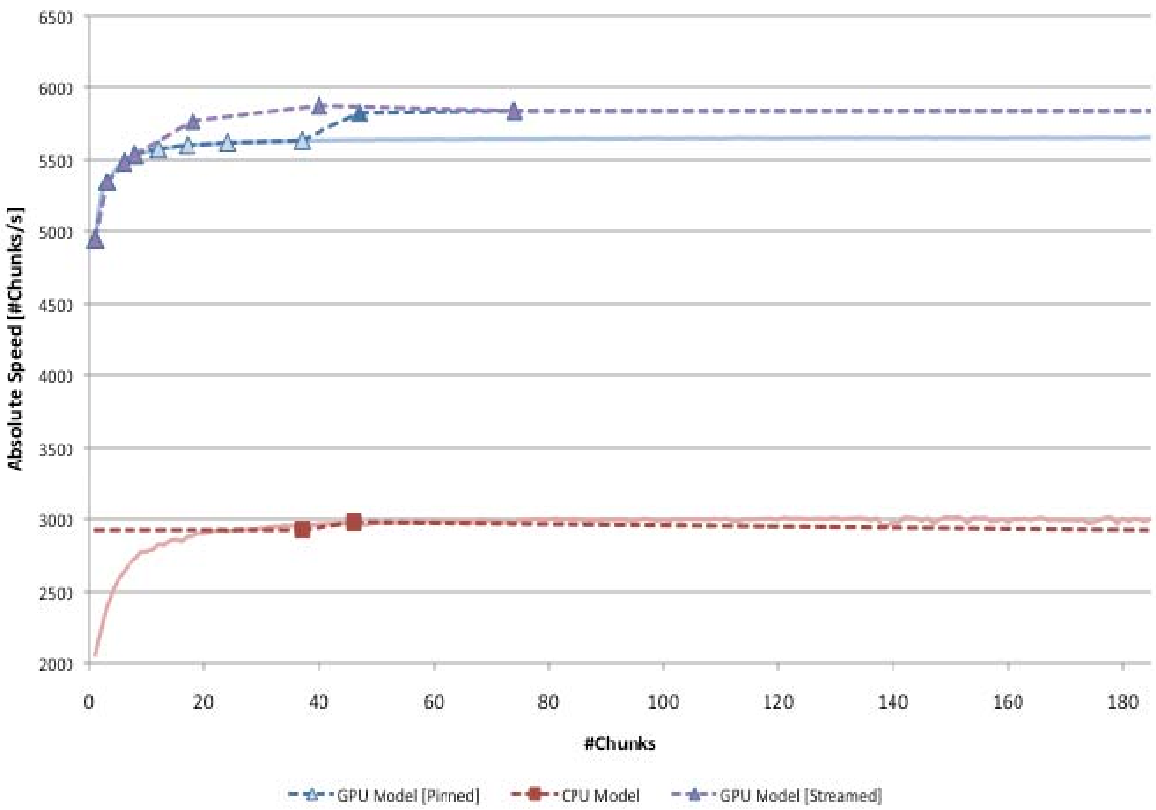
- for each stream
- for each stream restart

Case Study: Q3 Query
CPU + GPU Performance Modeling(19)

technology
from seed

inesc id lisboa

Performance Metric → Initialization → **Approximation** → Iteration



1 Refine performance models with the newly obtained results

② GPU-specific modeling: Using the obtained values from streaming execution

- *HostToDevice* Bandwidth
- *DeviceToHost* Bandwidth
- GPU *Kernel* Performance

③ Incorporate streaming results

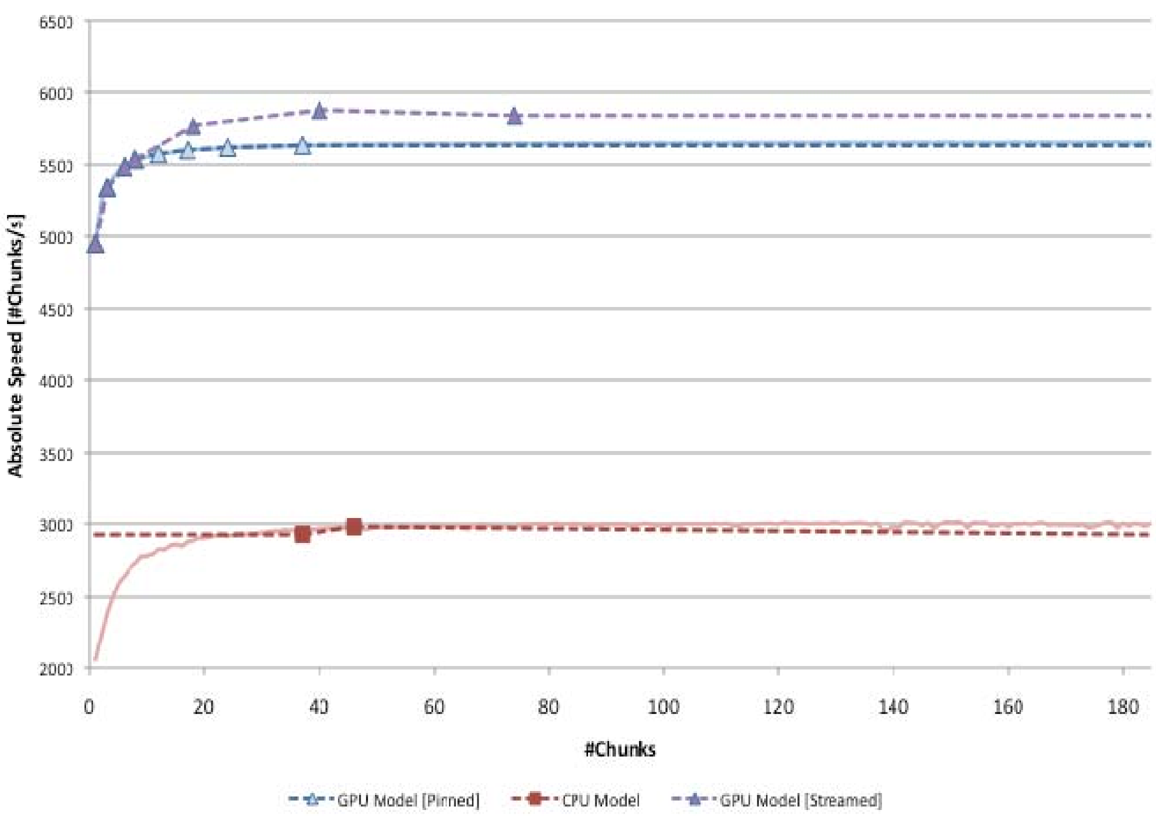
- for each stream
- for each stream restart
- for every stream combination

Case Study: Q3 Query
CPU + GPU Performance Modeling(20)

technology
from seed

inesc id lisboa

Performance Metric → Initialization → **Approximation** → Iteration



1. Refine performance models with the newly obtained results

② GPU-specific modeling: Using the obtained values from streaming execution
   - *HostToDevice* Bandwidth
   - *DeviceToHost* Bandwidth
   - GPU *Kernel* Performance

③ Incorporate streaming results
   - for each stream
   - for each stream restart
   - for every stream combination

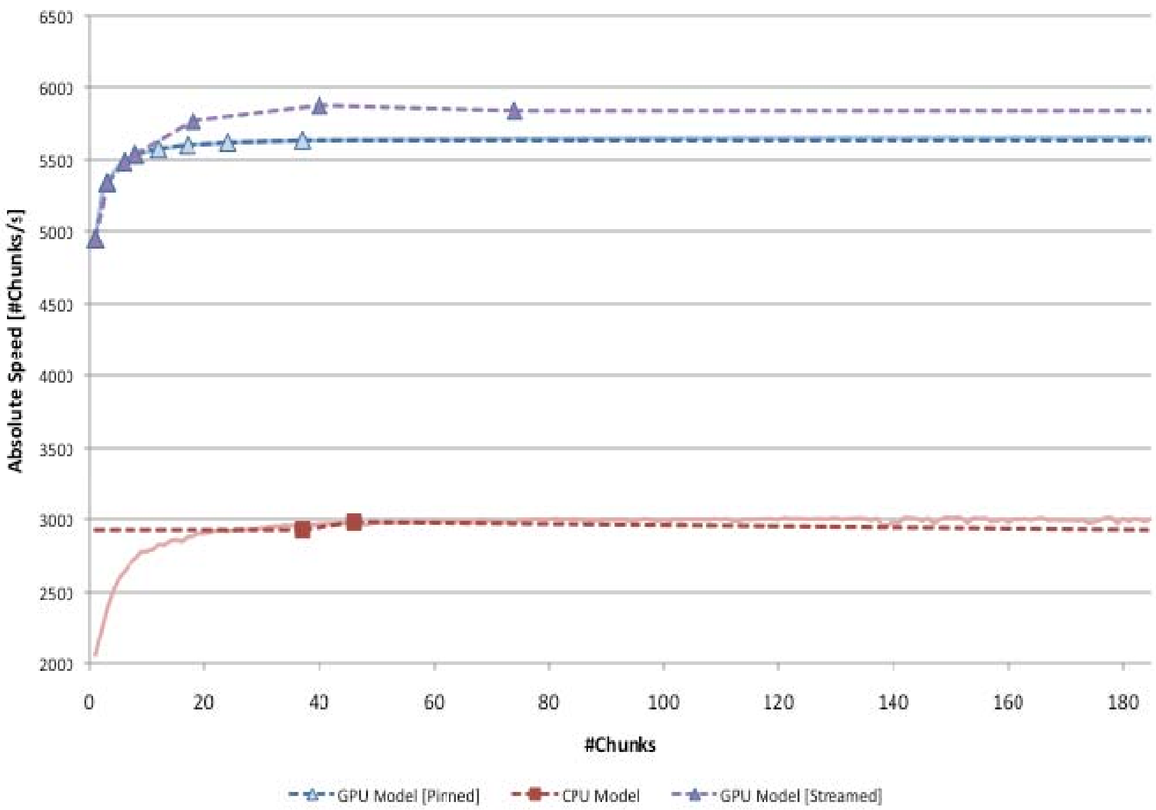
④ Remove the streaming point obtained using DIV2 STRATEGY and approximate both models

Case Study: Q3 Query
CPU + GPU Performance Modeling(20)

technology
from seed

inesc id lisboa

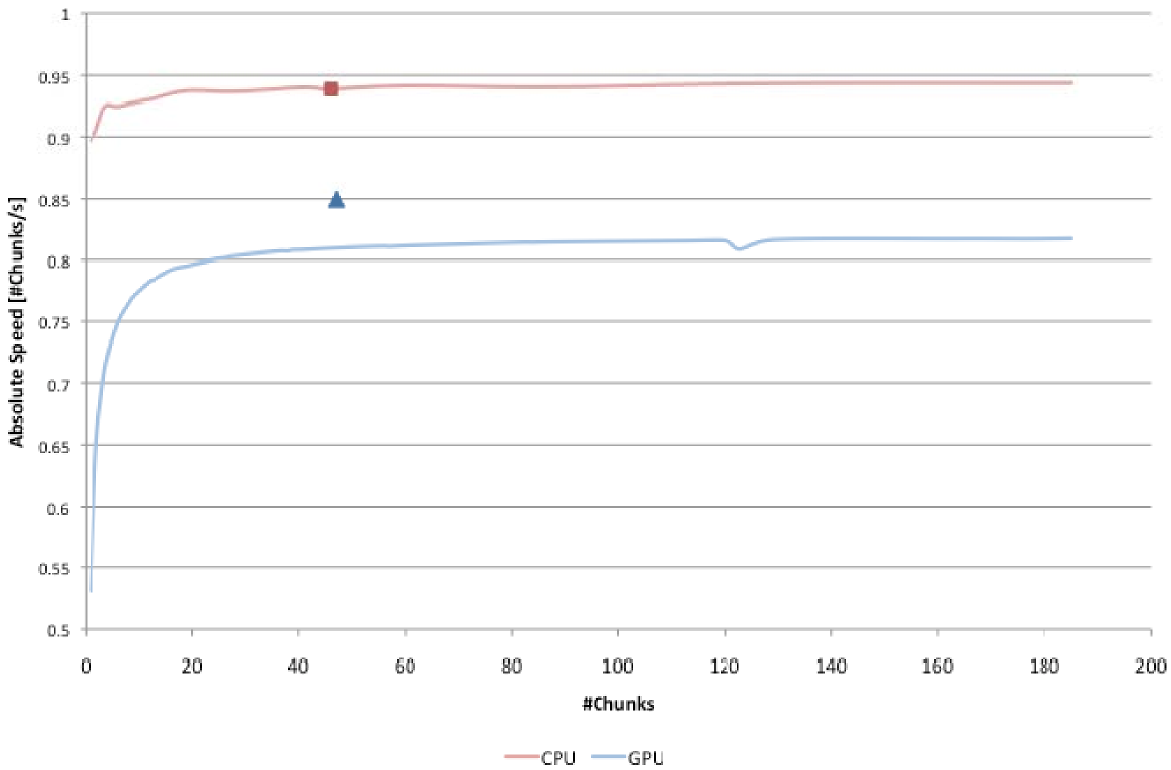
Performance Metric → Initialization → **Approximation** → Iteration



	Time [ms]	Speedup	Improvement [%]
1 Core	61.3	1	0
2 Cores	33.6	1.82	45.19
4 Cores	21.8	2.82	64.44
3 Cores + GPU [dummy]	15.4	3.98	74.88
3 Cores + GPU [our approach]	12.6	4.86	79.45

Case Study: Q6 Query

– APPLICATION OF THE PRESENTED APPROACH TO THE Q6 QUERY
  – The presented approach was tested for certain corner cases, e.g. Q6 QUERY, where it was spotted that the GPU performance was lower than the performance obtained using a single CPU core
  – Nevertheless, our algorithm CORRECTLY PREDICTED IN THE FIRST RUN that it is NOT WORTHWHILE TO SUBSTITUTE one core with the execution in the GPU, thus deciding that the best execution in this case is achieved by ONLY using the CPU



Conclusions

Dynamic Load Balancing

- Obtained in only **2 iterations** and **0.03 seconds**
  - 200 times faster than using exhaustive search

Traditional Approaches for Performance Modeling

- Approximate the performance using number of points equal to the number of iterations
- In this case, **2 points** per each device

Presented Approach for Performance Modeling

- Models the performance using **more than 14 points**, in this case
- **Communication-aware** – schedules in respect to limited and asymmetric interconnection bandwidth
- Employs **Streaming Strategies** to overlap communication with computation
- **Builds several** per-device **models** at the same time
  - Overall Performance for each device + Streaming GPU Performance
  - HostToDevice Bandwidth Modeling
  - DeviceToHost Bandwidth Modeling
  - GPU Kernel Performance Modeling

Future work

technology
from seed

To Experiment Different Types of Applications

- Programmed in OpenCL can run on both devices (CPU and GPU)
- In particular, we have in mind a bio-informatic application

To Derive Automatic Strategies for Defining the Streaming Process

- We have ideas and we already have done some work in this direction

To Apply CPHC in More Heterogeneous multicore Systems

- e.g. with reconfigurable processors

Acknowledgments

THE AUTHORS WOULD LIKE TO THANK TO

- **ALEXEY LASTOVETSKY** and **ROBERT HIGGINS** from the University College Dublin, Ireland
- The HETEROGENEOUS COMPUTING LABORATORY (HCL), School of Computer Science and Informatics, UCD, Ireland


Questions?
Thank you
inesc id
lisboa