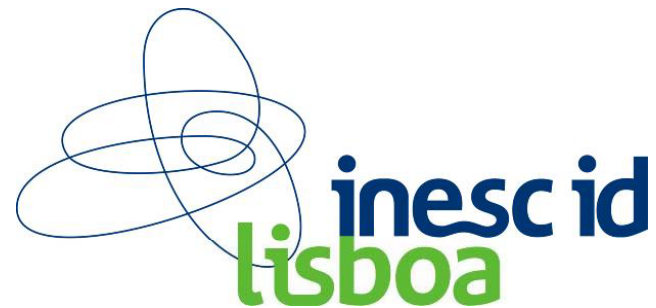


Extending the application of GPUs beyond processing accelerators

Leonel Sousa and Aleksandar Ilic and Frederico Pratas

CE Lab Colloquium
TUDelft

2009 August 26



Motivation: GPUs' evolution



technology
from seed

- Graphics Processing Units (GPUs) in integrated circuits since 80's
 - Alpha-Numeric Television Interface Circuit (ANTIC) LSI Atari μ C
- GPUs development mainly thrust by the gaming industry & market (e.g. Prince of Persia- 1st version 1989; 2nd version 2008)
 - Not only high computational power but growing at a high rate

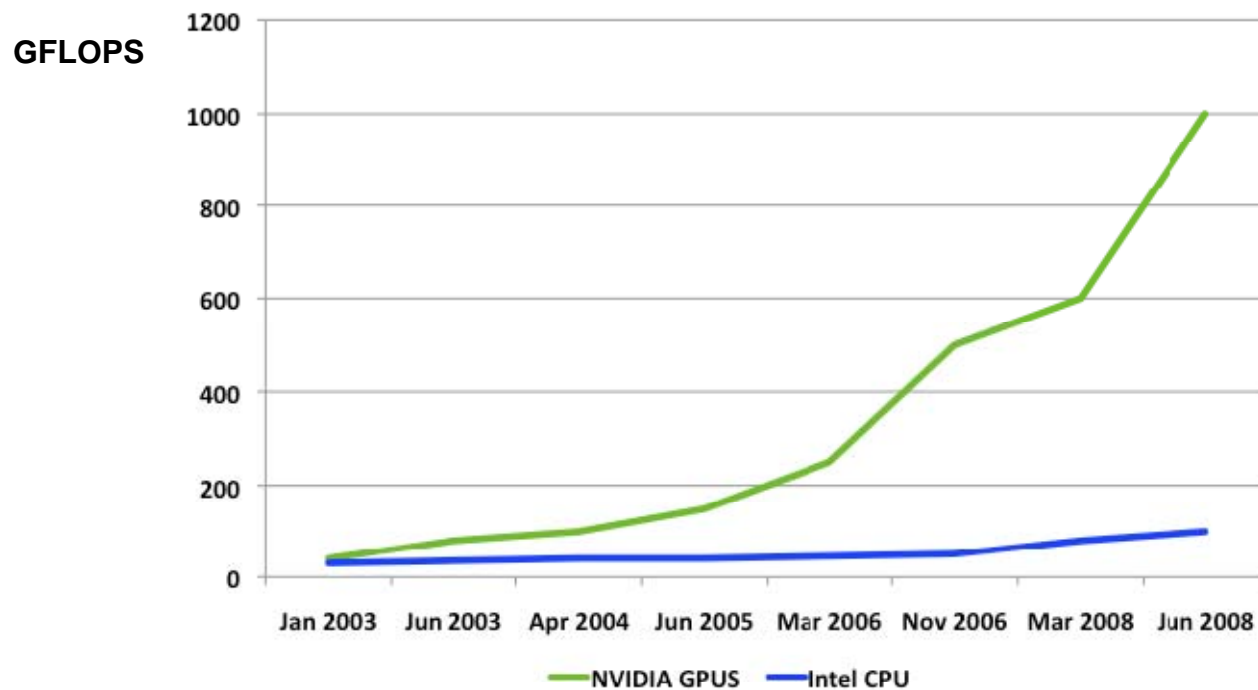


Motivation: CPUs vs GPUs performance



technology
from seed

- Originally GPUs only dedicated for graphics but the high computational capacities woke up interest for using them to other applications



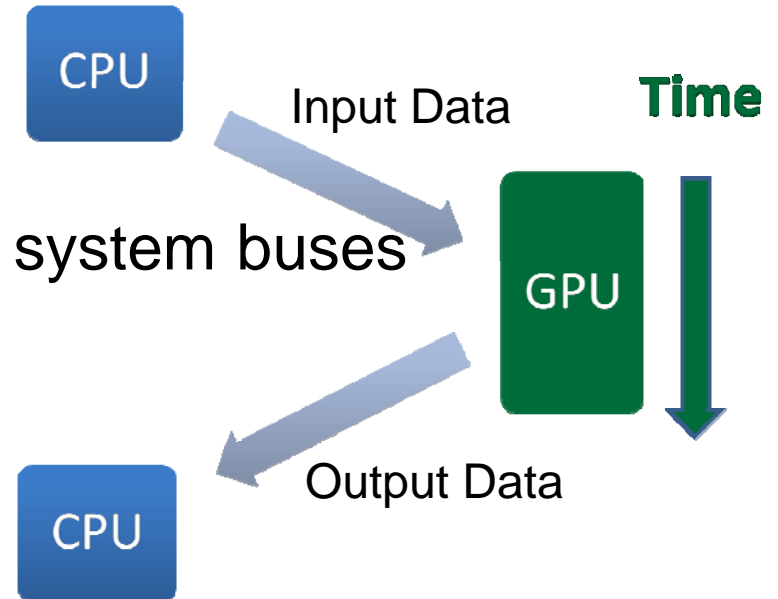
- General purpose Computation on Graphics Processing Units (GPGPU: www.gpgpu.org)
 - Acceleration of specific functions, mainly operating as a co-processor in host computers
 - Scientific computation (e.g., Fluid Dynamics)
 - Signal Processing (FFT, Filtering, Image and video processing,...)
 - High Performance libraries (BLAS,...)
 -

Motivation



technology
from seed

- GPUs hosted in computers through system buses (PCIe), no real speedup is achieved

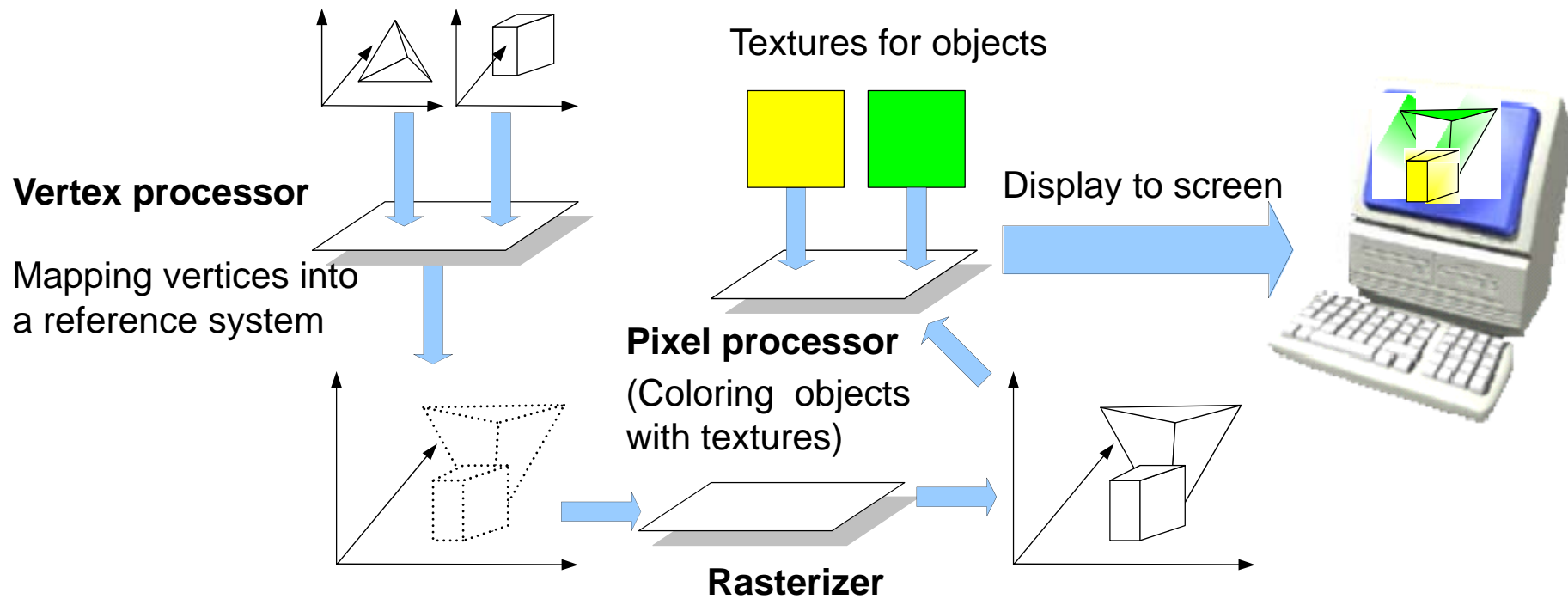


- To use computational power of GPUs in commodity computers:
 - CPU + GPU = heterogeneous system with slow communication
 - or as a computer component to design hardware processing structures

- GPU architectures and programming models
- Extending the application of GPUs
 - **CHPS:** Collaborative-execution-environment for Heterogeneous Parallel Systems
 - **SDHA:** applying the Stream-based-computing-model to Design Hardware Accelerators
- Conclusions and future work

Graphics Operations: Rasterization-based Rendering

technology
from seed

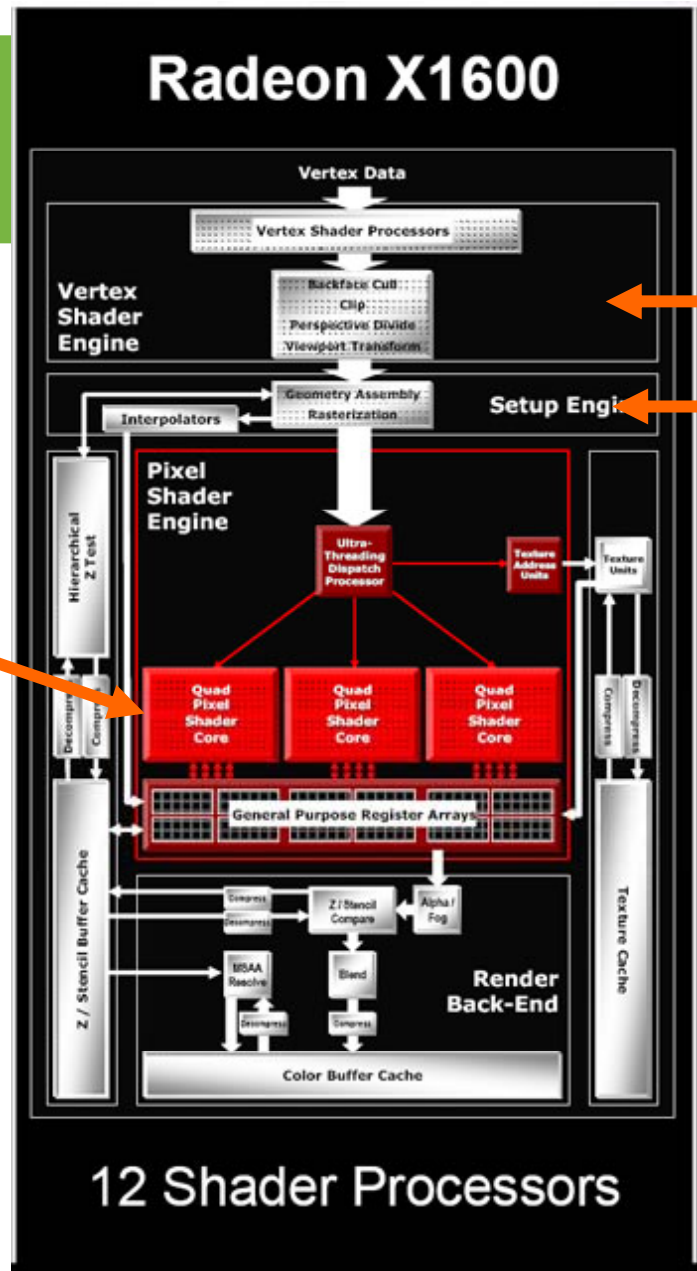


Architecture of Traditional GPU

Radeon X1600



technology
from seed



Vertex processor

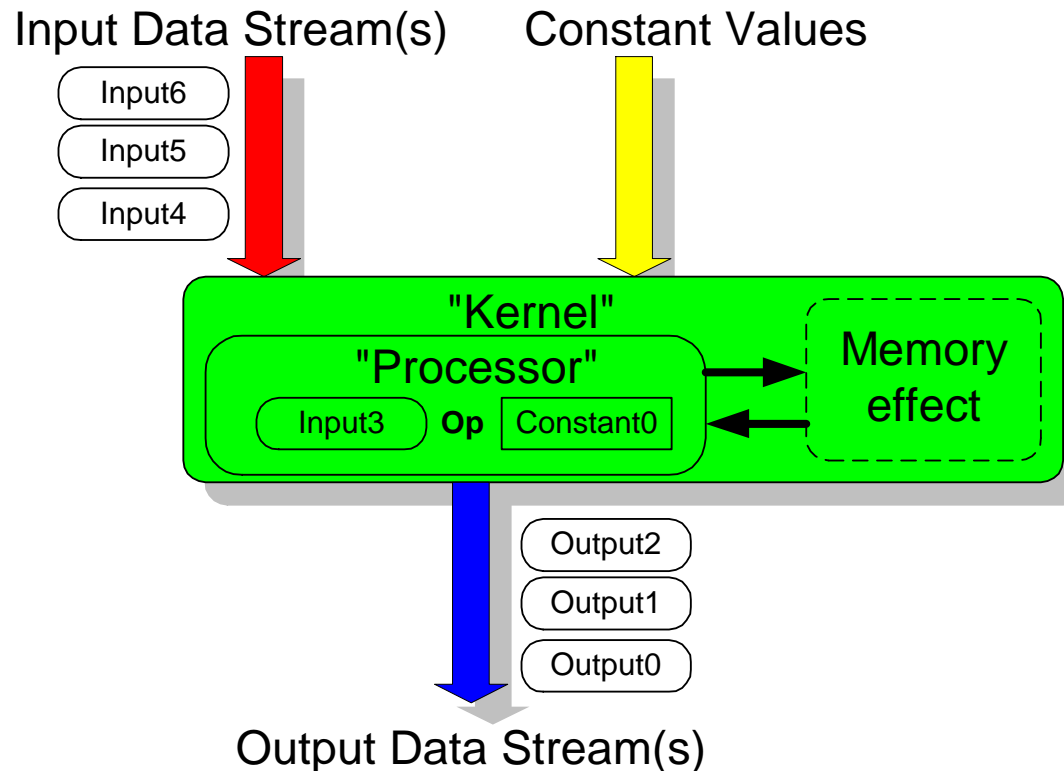
Rasterizer

12 x
pixel processors

Flow-model and Caravela Programming Tools for GPGPU



technology
from seed

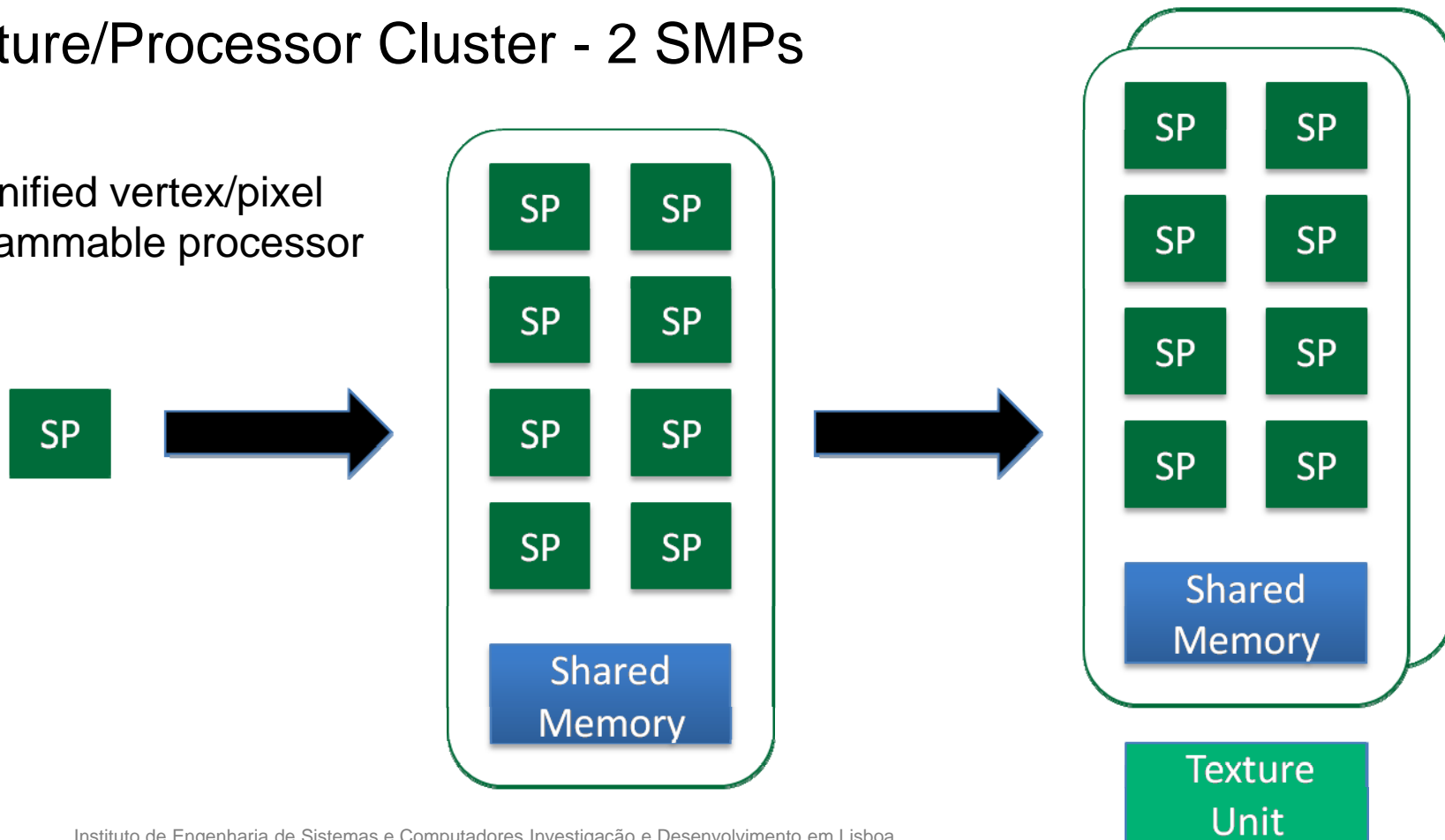


- Flow-model and Caravela programming tools (OpenGL, DirectX)
S. Yamagiwa and L. Sousa, "Caravela: A Novel Stream-Based Distributed Computing Environment", IEEE Computer, May 2007

GPU based on Tesla Architecture

- Streaming Multiprocessor (SM): 8 cores (SP) + shared memory
- Texture/Processor Cluster - 2 SMPs

SP: unified vertex/pixel programmable processor

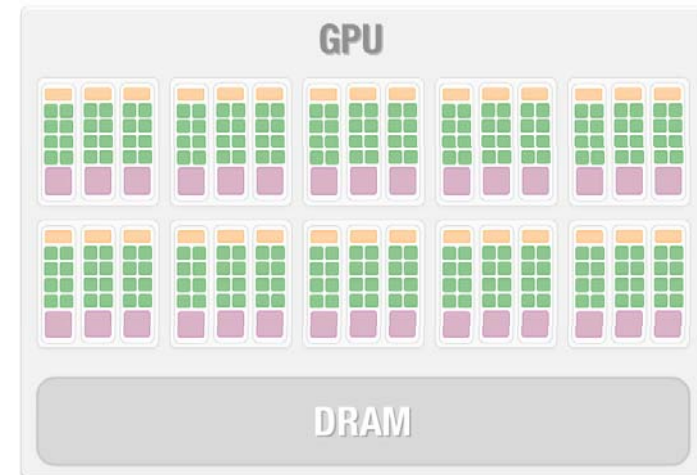


GPU based on Tesla Architecture



technology
from seed

- Warp – groups of 32 parallel threads
 - Each SM manages a pool of 24 warps
 - SIMT controls execution and branching behavior of one thread
 - Large number of threads scheduled by hardware masks memory latency
- NVIDIA GT200 GPUs
 - A collection of SIMD SMs- up to 30
 - High memory bandwidth-up to 150GB/s
 - Connected via PCIe bus, with 2GB/s bandwidth



- Compute Unified Device Architecture (CUDA)
 - Divide a kernel into thousands of parallel threads
 - Organize these threads into blocks, which run independently and in parallel in different SMs and share 16KB of memory each
 - Group blocks into grids
 - All threads have access to global memory
- Programming interface
 - Standard C programming languages with minimal extensions for parallel programming

Work



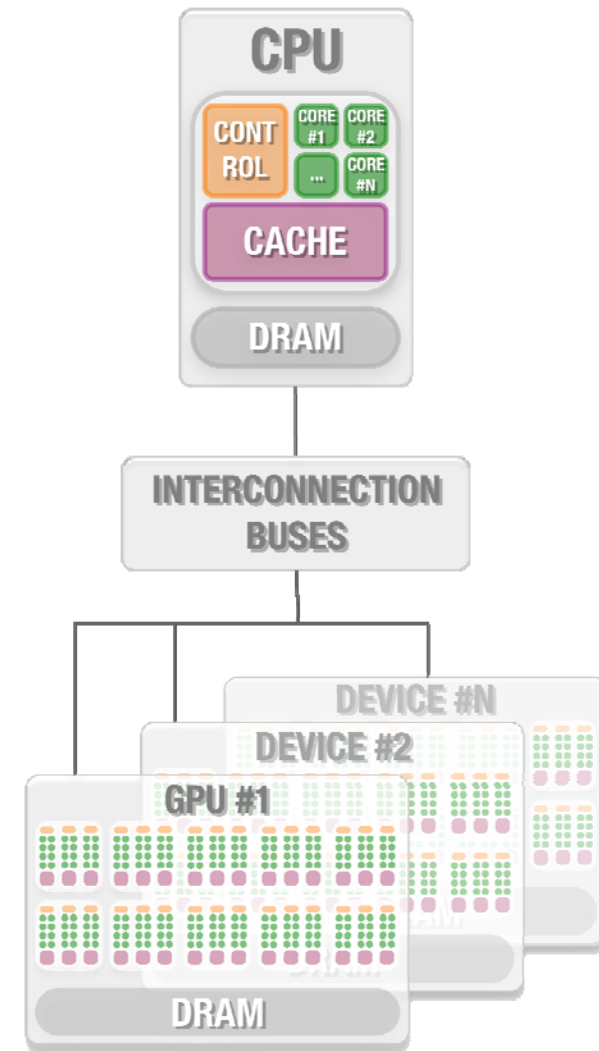
technology
from seed

- **CHPS** - Collaborative Execution Environment for Heterogeneous Parallel Systems
- **SDHA** - Applying the Stream-Based Computing Model to Design Hardware Accelerators

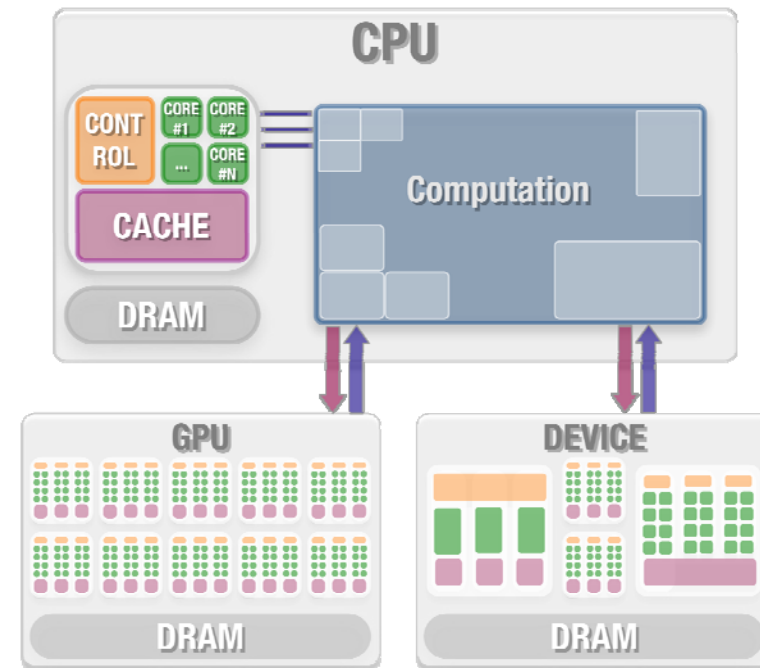
- Commodity computers = **Heterogeneous systems**
 - Multi-core general-purpose processors (CPUs)
 - Many-core graphic processing units (GPUs)
 - Special accelerators, co-processors, FPGAs, DSPs
- ⇒ Huge **collaborative** computing power
 - Not yet explored
 - In most research done target the usage of one these devices at time, mainly domain-specific computations
- Heterogeneity makes problems much more complex
 - many programming **challenges**

Master-slave execution paradigm

- Distributed-memory programming techniques
- **CPU (Master)**
 - Global execution controller
 - Access the whole global memory
- **Interconnection Busses**
 - Reduced communication bandwidth comparing to distributed-memory systems
- **Underlying Devices (Slaves)**
 - Different architectures and programming models
 - Computation performed using local memories



- **Computation Partitioning**
 - To fulfill device capabilities/limitations and achieve optimal load-balancing
- **Data Migration**
 - Significant and usually asymmetric
 - Require detailed modeling
 - Potential execution bottleneck
- **Synchronization**
- **Different programming models**
 - *Per* device type and vendor-specific
 - Vendors provide high performance libraries and software

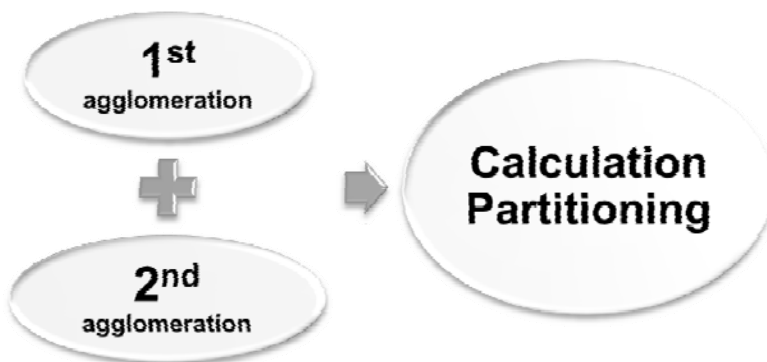


- **Application Optimization**
 - Very large set of parameters and solutions affects performance

primitive jobs

JOB QUEUE

SCHEDULER



Primitive jobs

- Minimal problem portions for parallel execution (1st agglomeration)
- Balanced granularity: fine enough partition to sustain execution on every device in the system, but coarse enough to reduce data movement problems/overheads

Job Queue

- Classifies the primitive jobs to support specific parallel execution scheme
- Accommodates primitive jobs' batching into a single job according to the individual device demands (2nd agglomeration)

primitive jobs

JOB QUEUE

SCHEDULER

devices

DEVICE QUERY

Device Query

- Identifies all underlying devices
- Holds per-device information (device type, current status, requested workload and performance history)

Scheduler

- Adopts a scheduling scheme according to devices' capabilities, availability and restrictions (currently, FCFS)
- Forms execution pair:
<device, computational_portion>
- Upon the execution completion, the data is transferred back to the host, and the device is free to support another request from the Scheduler

CHPS: Case study

Dense Matrix Multiplication



technology
from seed

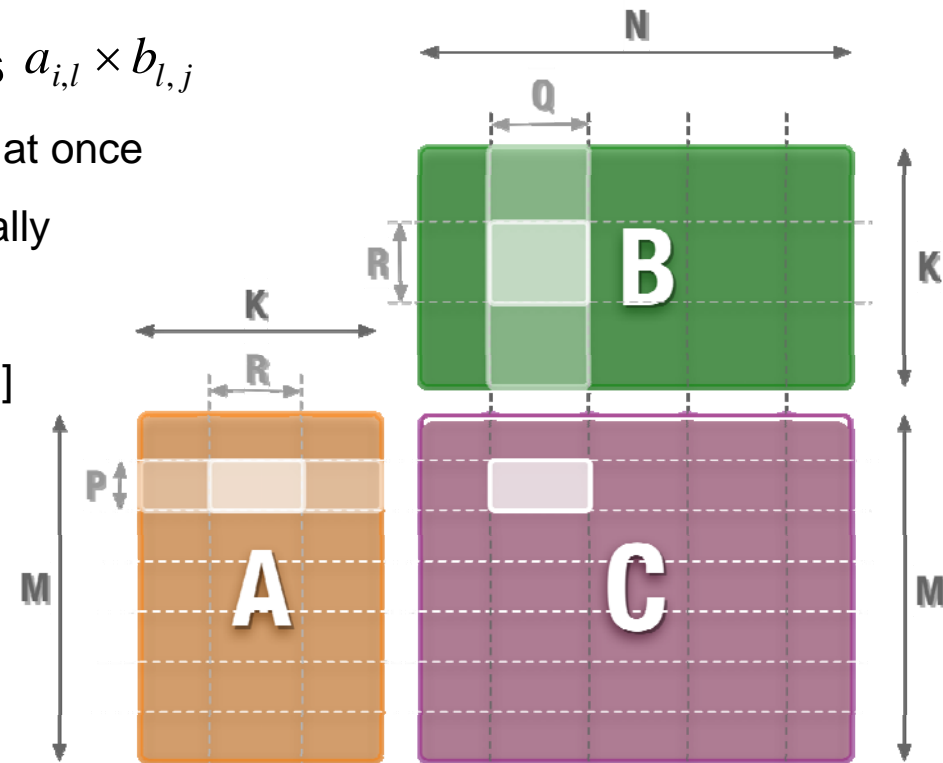
Matrix multiplication $C_{M \times N} = A_{M \times K} \times B_{K \times N}$ is based on **sub-matrix partitioning**

$$c_{i,j} = \sum_{l=0}^{K/R} a_{i,l} \times b_{l,j}, \quad 0 \leq i \leq \frac{M}{P}; 0 \leq j \leq \frac{N}{Q}$$

by creating a set of **primitive jobs** $a_{i,l} \times b_{l,j}$

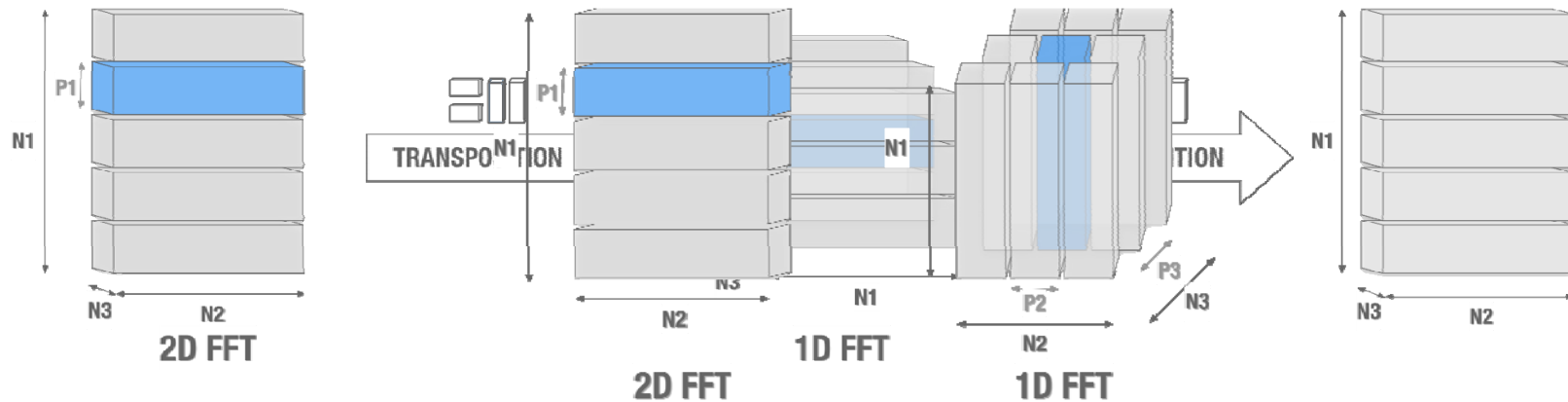
- Devices can request several primitive jobs at once
- Single $C_{i,j}$ updates are performed atomically
- High performance **libraries**:
 - *ATLAS CBLAS* [CPU], *CUBLAS* [GPU]

Problem size	M	N	K
Primitive Job parameters	P	Q	R
Job Queue size	$\frac{M}{P} \times \frac{N}{Q} \times \frac{K}{R}$		



CHPS: Case study 3D Fast Fourier Transform

technology
from seed



$$H = FFT_{1D}(FFT_{2D}(h))$$

Traditional parallel implementation requires transpositions

- between FFTs applied on different dimensions and after executing the final FFT

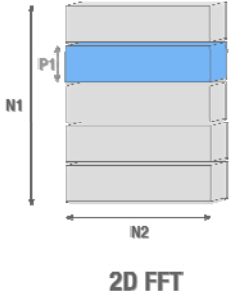
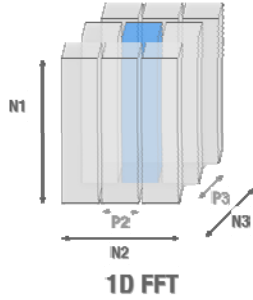
Our implementation

- Only portions of data which are assigned to the device are transposed, and the results are retrieved on adequate positions after the final FFT
- High performance libraries: *FFTW* [CPU], *CUFFT* [GPU]

CHPS: Case study 3D Fast Fourier Transform

technology
from seed



Problem size	N_1	N_2	N_3	
Primitive Job parameters	P_1	P_2	P_3	
 <p>2D FFT</p>	Total Size	N_1 2D FFTs of a size $N_2 \times N_3$		
	Primitive Job	✓	✗	✗
	Job Queue Size	N_1 / P_1		
 <p>1D FFT</p>	Total Size	$N_2 \times N_3$ 1D FFTs of a size N_1		
	Primitive Job	✗	✓	✓
	Job Queue Size	$(N_2 / P_2) \times (N_3 / P_3)$		

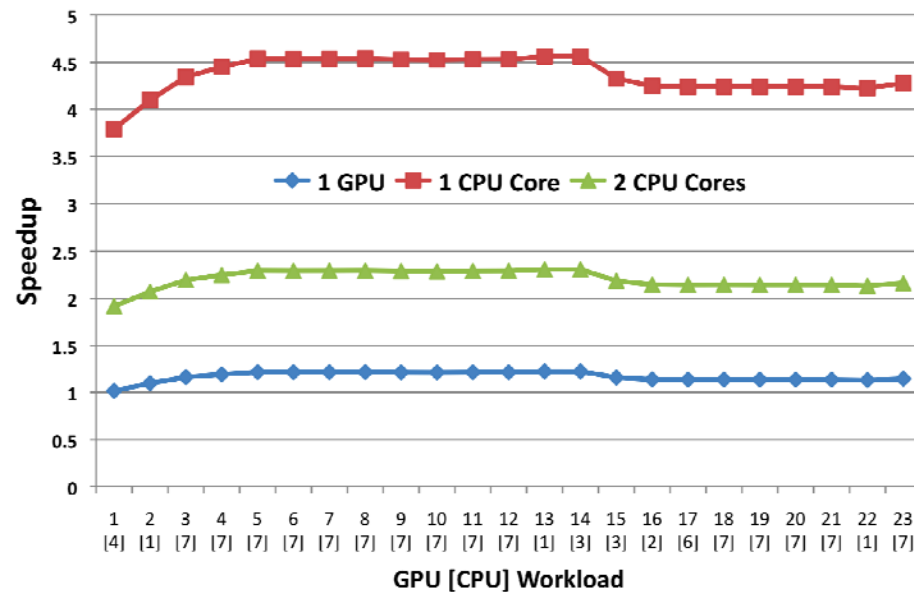
CHPS: Experimental Results

Optimized Dense Matrix Multiplication

technology
from seed



RESULTS		
Exhaustive search	32x32 tests	
Optimal load-balancing	GPU	CPU
	13	1
Obtained speedup (comparing to):		
1 CPU core	4.5	
2 CPU cores	2.3	
1 GPU	1.2	



Single precision floating-point arithmetic (Largest problem executable by the GPU)			
Problem size	M	N	K
	4096	4096	4096
Primitive Job parameters	P	Q	R
	4096	16	4096
Job Queue size	32		

Experimental Setup	CPU	GPU
		Intel Core 2
Speed/Core (GHz)	2.33	0.54
Global Memory (MB)	2048	256
High Performance Software		
Matrix Multiplication	ATLAS 3.8.3	CUBLAS 2.1
3D FFT	FFTW 3.2.1	CUFFT 2.1

CHPS: Experimental Results

Complex 3D FFT [2D FFT Batch]

technology
from seed



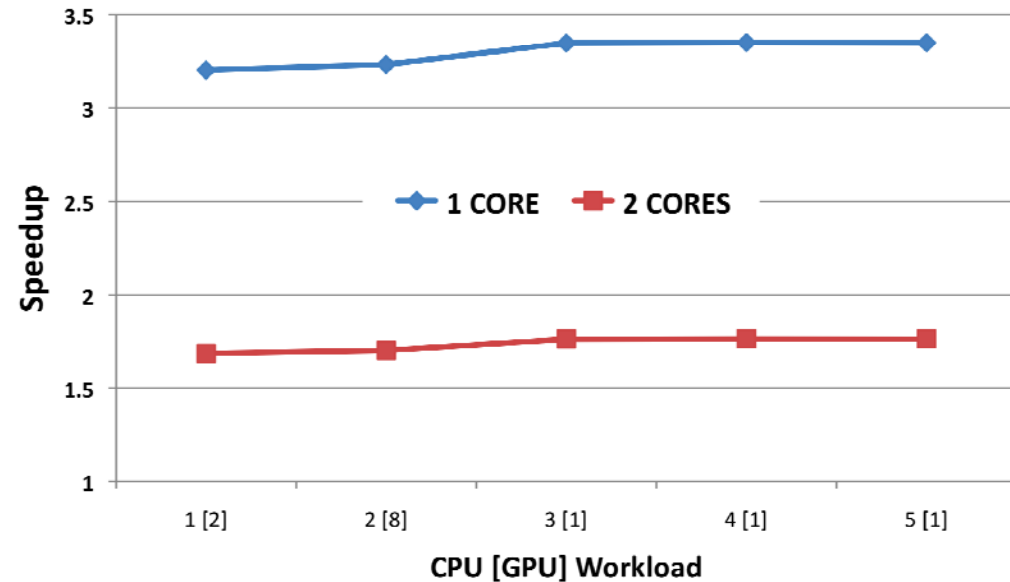
RESULTS

Exhaustive search	160 tests	
Optimal load-balancing	GPU	CPU
	4	1

Obtained speedup (comparing to):

1 CPU core	3.2
2 CPU cores	1.6
1 GPU	x

- GPU maximal workload is 5x2D FFTs (512x512)



Complex 3D fast Fourier transform
(Problem not possible to execute on the GPU)

Problem size	N ₁	N ₂	N ₃
	512	512	512
Primitive Job parameters	P ₁	P ₂	P ₃
	16	x	x
Job Queue size	32 x 2D FFT (512x512)		

CHPS: Experimental Results

Complex 3D FFT [1D FFT Batch]



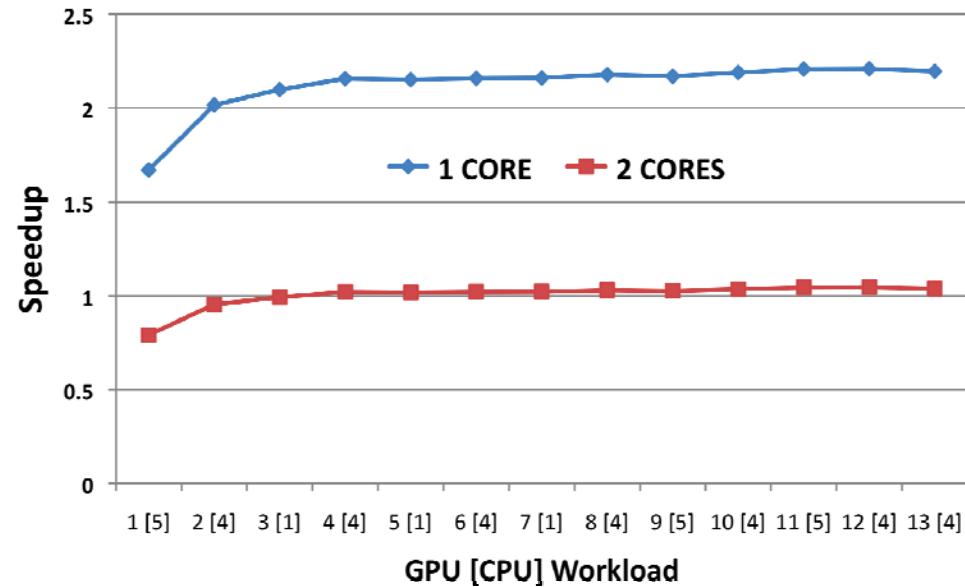
technology
from seed

RESULTS

Exhaustive search	416 tests	
Optimal load-balancing	GPU	CPU
	12	4

Obtained speedup (comparing to):

1 CPU core	2.2
2 CPU cores	~1!
1 GPU	x



Complex 3D fast Fourier transform
(Problem not possible to execute on the GPU)

Problem size	N_1	N_2	N_3
	512	512	512
Primitive Job parameters	P_1	P_2	P_3
	x	16	16
Job Queue size	32x32 1D FFTs (512)		

- GPU maximal workload is 13 x 1D FFTs (512)
- **Results include transpose time!**



CHPS: Experimental Results

Complex 3D FFT [1D FFT Batch]



technology
from seed

3D FFT RESULTS (TOTAL)

Obtained speedup (comparing to):

1 CPU core	2.8
2 CPU cores	1.3
1 GPU	x

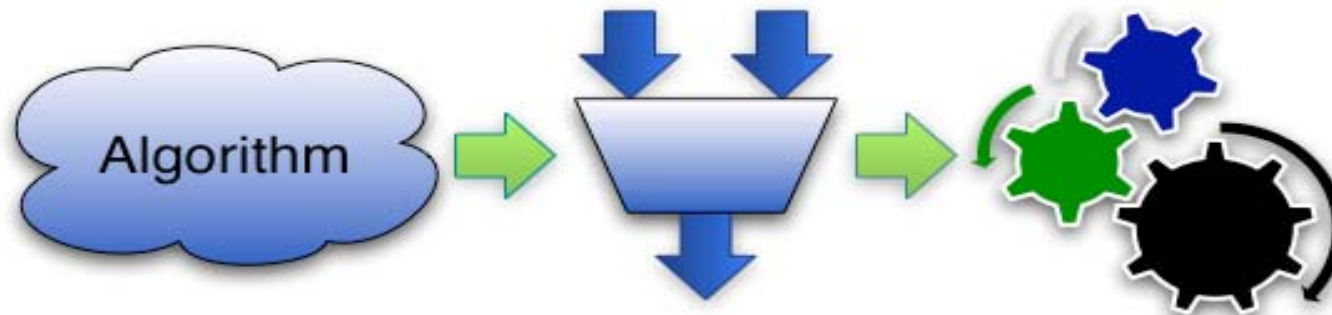
- **Transposition time** dominates the execution for higher workloads!
- **Memory transfers** are significant!

- **CHPS** - Collaborative Execution Environment for Heterogeneous Parallel Systems
- **SDHA** - Applying the Stream-Based Computing Model to Design Hardware Accelerators: A Case Study

- Co-processing hardware can be a very efficient solution to accelerate certain types of applications
- However, mapping algorithms into hardware is not straightforward
- Relatively easier to program according to the stream-based computing model (many algorithms have been proposed for different applications, e.g. for running on GPUs)



- The stream-based description seems to be suitable as an approximation to hardware description:
 - decouples computation from memory accesses
 - exposes maximum parallelism available in the application
- Stream-based description can be used as an intermediate step:
 - besides being easier and faster to implement it also provides the HW designer with early-stage programmable prototyping platforms



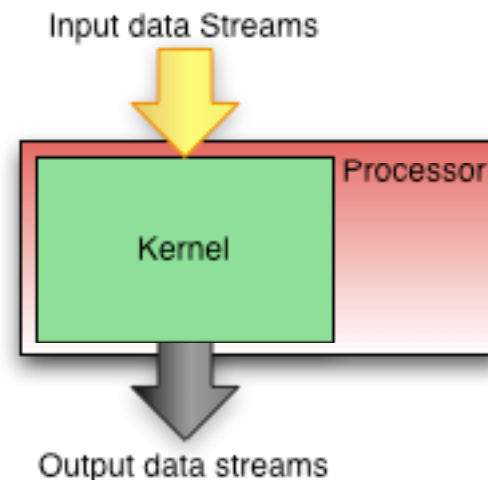
SDHA: Overview

Stream-based Computation



technology
from seed

- A data stream is a **sequence** of data items.
- In stream computing a **kernel**, comprising several operations in sequence, is applied to a single or multiple **input** data streams to **produce** an **output** data stream.



- **Efficient solutions but complex design**

Customizable

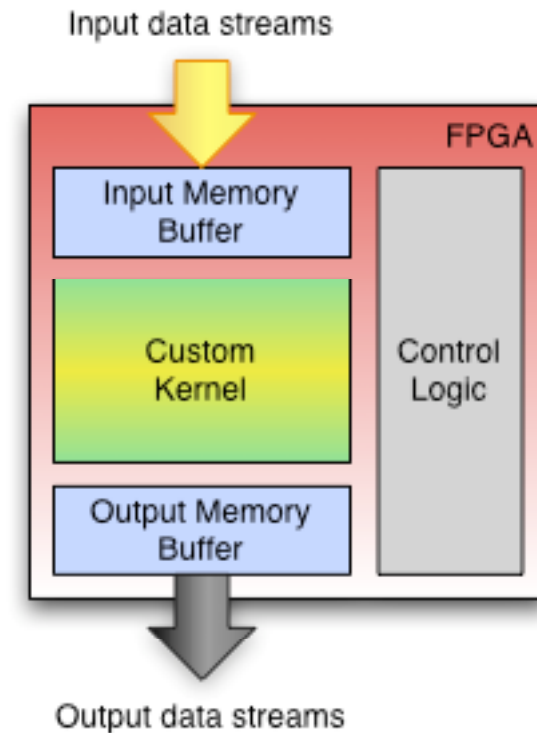
- Differently optimized for each application

Adaptable/Reconfigurable

- Different applications can be dynamically loaded

Hard to:

- We try to overcome these difficulties:
stream-based computing (GPUs)
- hardware accelerators



SDHA: Case Study MrBayes



technology
from seed

- MrBayes is a bioinformatics application that performs Bayesian inference of evolutionary (phylogenetic) trees.

Phylogenetic trees are used for instance in

“Origins and evolutionary genomics of the 2009 swine-origin H1N1 influenza A epidemic”

<http://www.nature.com/nature/journal/v459/n7250/full/nature08182.html>

- A current real-world phylogenomic data set study contains 1,500 genes and requires 2,000,000 CPU hours on a BlueGene/L system¹.

¹ M. Ott, et. al, “*Large-scale Maximum Likelihood-based Phylogenetic Analysis on the IBM BlueGene/L*”, In SC’07, NY, USA.

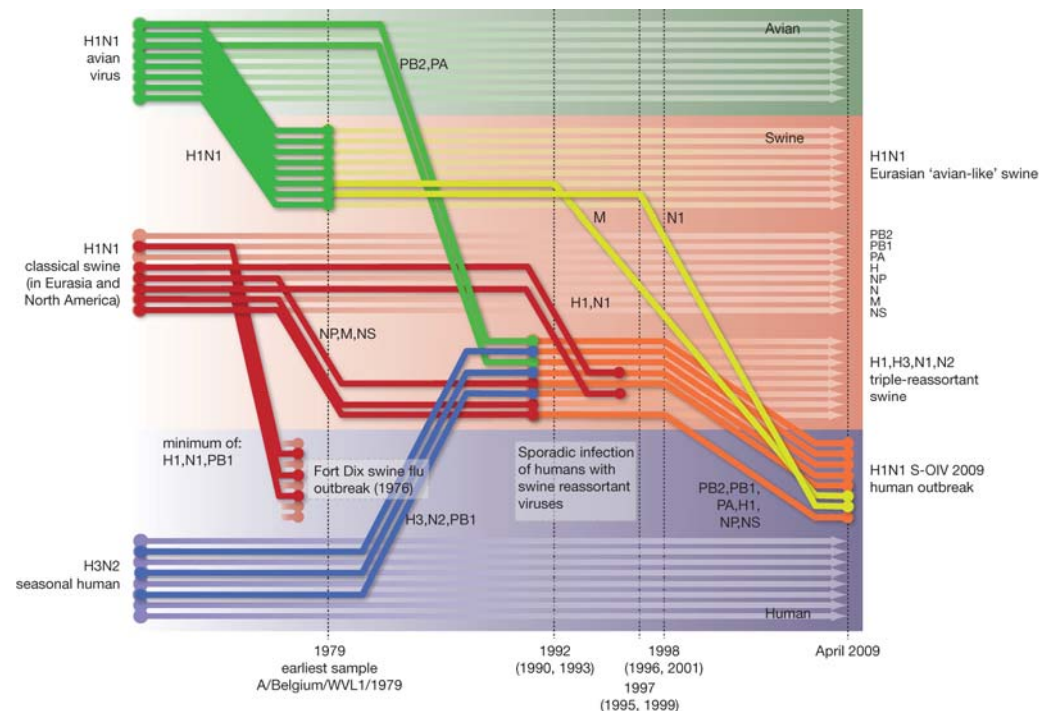
SDHA: Case Study MrBayes

technology
from seed



- Phylogenetic Tree: Origin and evolution of the H1N1 (simplified)

Reconstruction of the sequence of reassortment events leading up to the emergence of S-OIV.



GJD Smith *et al.* *Nature* **459**, 1122-1125 (2009) doi:10.1038/nature08182

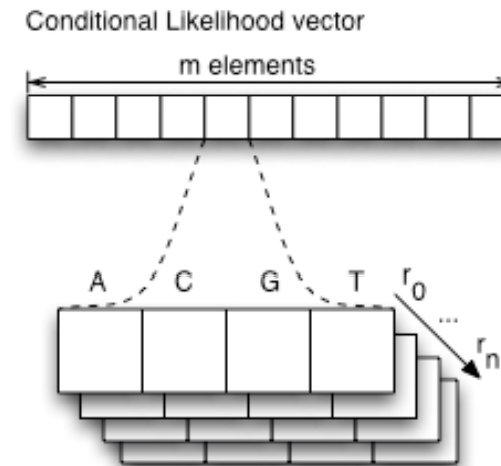
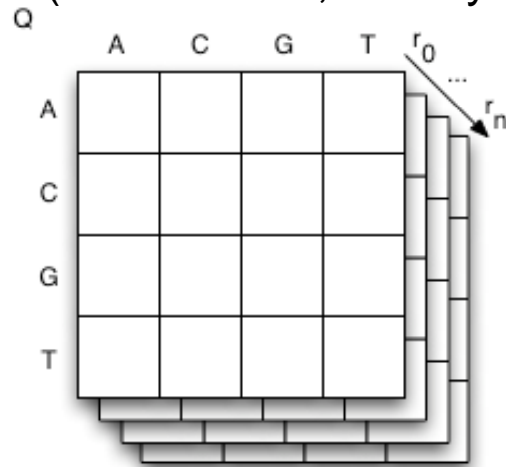
SDHA: Case Study MrBayes

technology
from seed



- Profiling showed that >85% of the program execution is spent in the Phylogenetic Likelihood Functions (PLF).
 - 2 PLF, *CondLikeDown* and *CondLikeRoot*
- Data Structures
 - Nucleotide substitution matrix

(A – Adenine, C – Cytosine, G – Guanine, T- Thymine)



SDHA: MrBayes Stream-based Parallelization

technology
from seed



- Primitive kernel

Input: cl Arrays (Left/Right)
Input: Substitution Matrices Q
(Left/Right)

Output: Result clP

foreach cl element i **do**

foreach Discrete rate r **do**

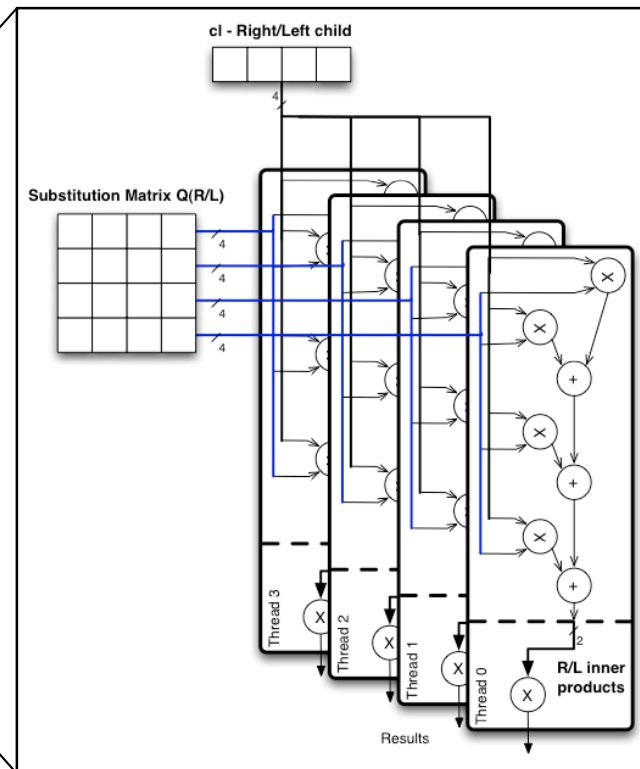
foreach Q row j **do**
 Calculate Inner Products
 ($Q_{Left,j}, cl_{Left}$)
 ($Q_{Right,j}, cl_{Right}$)
 end

 Multiply the final arrays.

end

end

- Stream-Based Parallelization

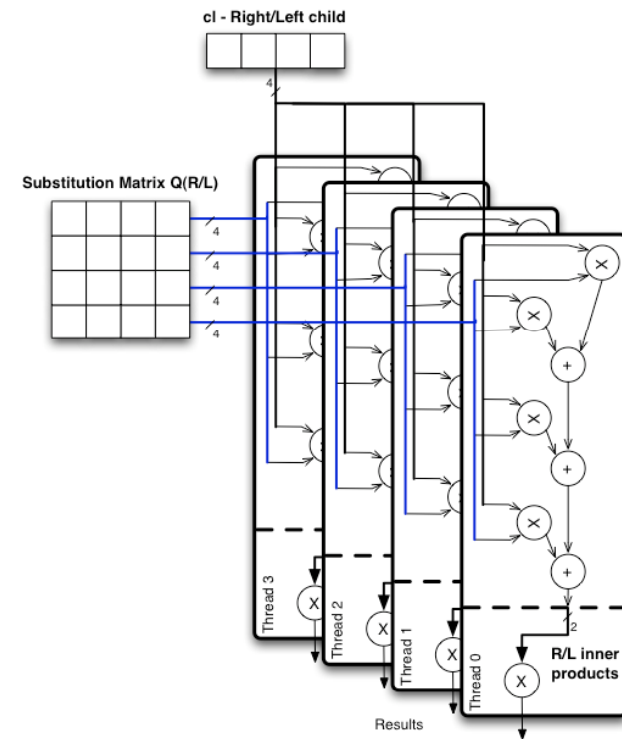


SDHA: MrBayes Stream-based Parallelization

technology
from seed



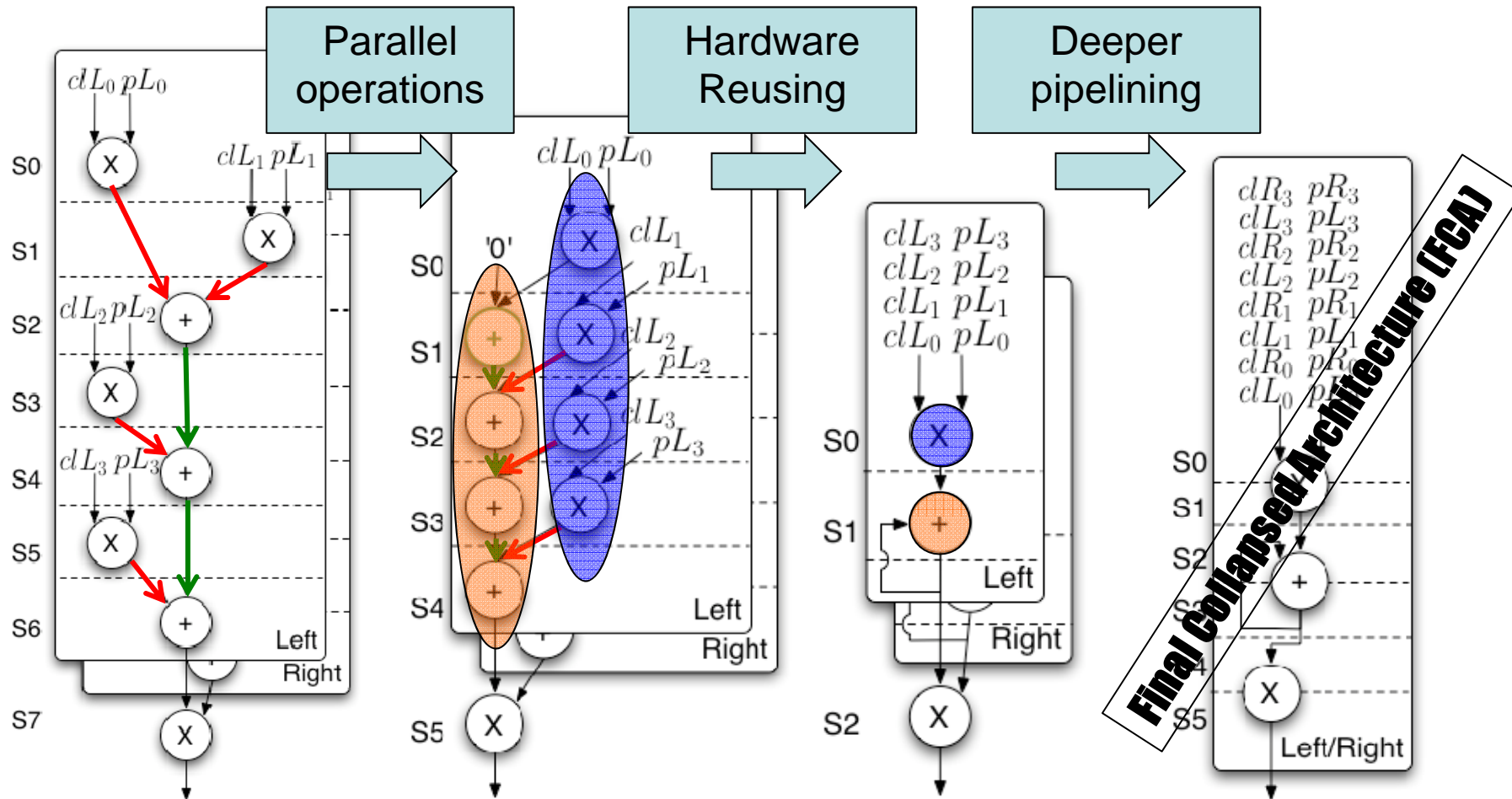
- The stream-based version was implemented with CUDA for GPGPU
 - The PLF are executed on the GPU, while the remaining part of the program is computed on the CPU
 - Each thread computes one **inner product**, which can be seen as a reduction
 - Number of threads used: 10240
 - The data streams are split between blocks
 - Number of blocks used: 40



More details about the CUDA implementation and the algorithm parallelization can be found at F. Pratas, et. al, “*Fine-grain Parallelism using Multi-core, Cell/BE, and GPU Systems: Accelerating the Phylogenetic Likelihood Function*”, In ICPP’09, Vienna, Austria.

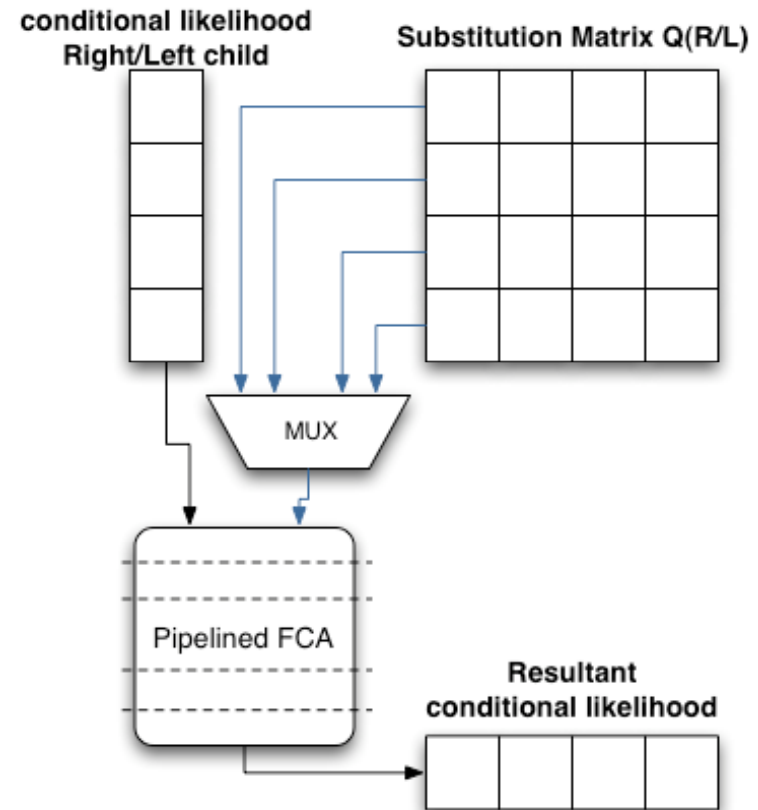
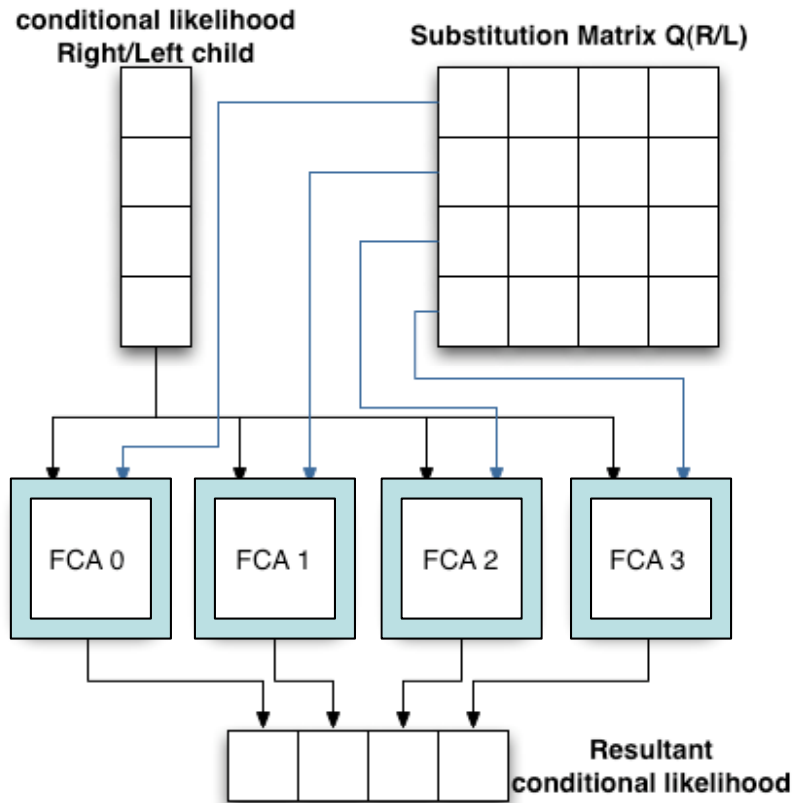
SDHA: MrBayes

From Stream-based to Reconfigurable Hardware

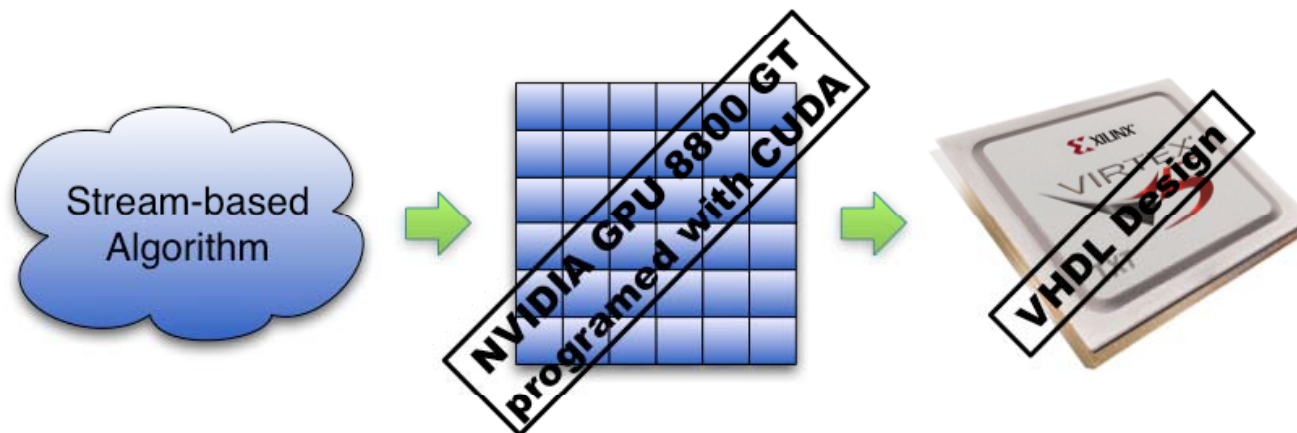


SDHA: MrBayes From Stream-based to Reconfigurable Hardware

technology
from seed



- Prototyping platform
 - The stream-based algorithm is programmed in CUDA
 - The GPU is used for optimization and debugging
 - The algorithm is ported to VHDL with a minimum effort
 - A simple control unit can be used for several units following a SIMD approach



SDHA: Experimental Setup

technology
from seed

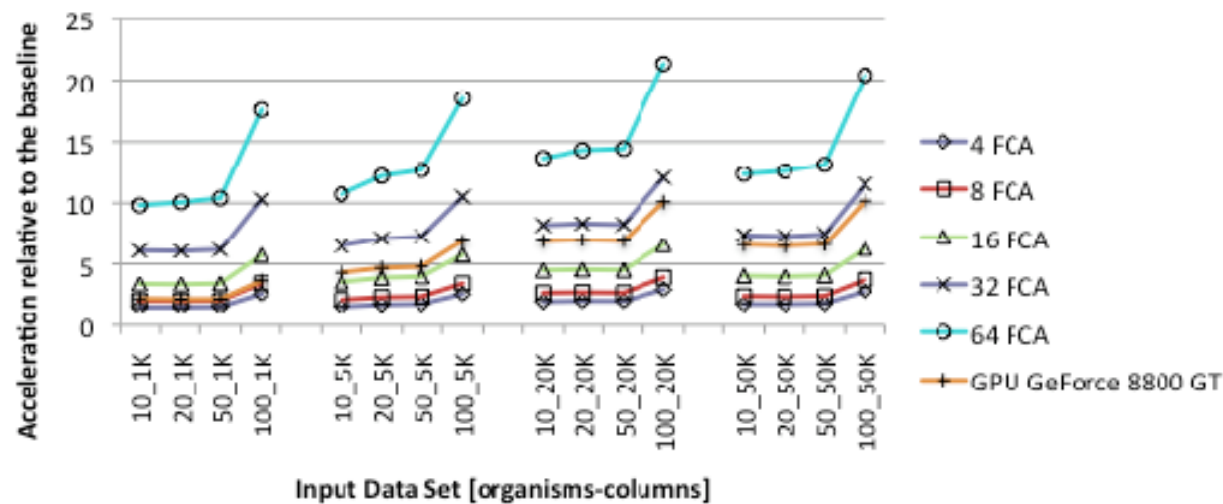


Characteristics	Baseline Intel x86	GPU GeForce 8800 GT	Virtex 4 LX100	Virtex 5 LX110	Virtex 5 FX200T
Frequency [GHz]	3.000	0.575	0.450	0.667	0.450
Power [Watts]	65	105	<10	<10	<10
# Cores	1	112	–	–	–
# Slices ⁽¹⁾ / # DSP ⁽²⁾	–	–	49152 / 96	17280 / 64	122880 / 384
Technology	45-nm	65-nm	90-nm	65-nm	65-nm

- The Stream-based implementation was programmed with CUDA API v2.1
- The hardware was described in VHDL and the results were obtained with Xilinx ISE 10.1 after Place-and-Route.
- Baseline is a general-purpose architecture and is used as reference system.
- MrBayes version 3.1.2 with input datasets obtained from Seq-Gen.
- Datasets are named according to X_Y, where X is related to the number of PLF calls and Y is related to the size of the data set.

SDHA: Experimental Results

- Scalability of the accelerated kernel for different number of FCAs on the Virtex5 FX200T



- We implemented Super-pipelined FCAs, with 8 stages floating-point units.
- Multiples of 4 FCAs were used to simplify the control due to the number of discrete rates.

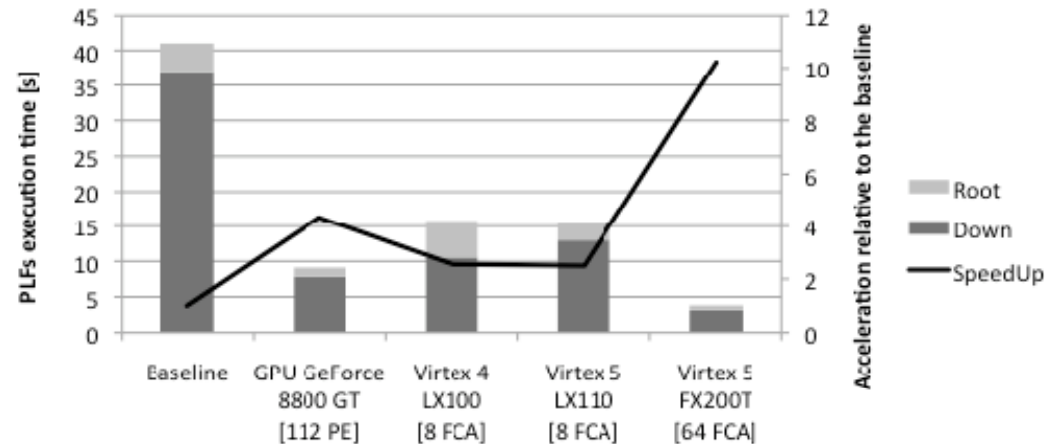
SDHA: Experimental Results

PLF function	Characteristics	Virtex 4 LX100	Virtex 5 LX110	Virtex 5 FX200T
CondLikeDown	Occupation (Slices / DSPs)	15% / 100%	17% / 75%	63% / 100%
	Frequency [MHz]	128	104	54
	# of cycles	40	40	40
	Maximum # FCAs	8	8	64
CondLikeRoot	Occupation (Slices / DSPs)	15% / 54%	31% / 88%	84% / 95%
	Frequency [MHz]	98	105	56
	# of cycles	72	72	72
	Maximum # FCAs	4	8	52

- The CondLikeRoot function has one extra inner product per conditional likelihood element → extra floating-point units.
- The obtained frequency is similar for the two FPGAs when using approximately the same number of FCAs.
- The frequency decreases with the increase of the number of FCAs due to the FPGA occupancy and associated routing overhead.
- The latency is constant because the parallelism is the only factor being modified across different topologies.

SDHA: Experimental Results

technology
from seed



- Results are relative to the kernel functions. FPGA can be reconfigured two implement one of the PLFs at a time
- It is assumed that the input data fits in the main memory.
- The fastest architecture is the Virtex5 FX200T with 64 FCAs, showing a speedup of 10.2x. The GPU achieves a speedup of 4.3x.
- Although the difference between the # FCAs in the two Virtex5 is 8x, the speedup obtained is only 4x (different operating frequencies).

- **CHPS - Collaborative Execution Environment for Heterogeneous Parallel Systems**

Aleksandar Ilic and Leonel Sousa, “*Collaborative Execution Environment for Heterogeneous Parallel Systems*”, submitted to 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP) 2010.

- **SDHA - Applying the Stream-Based Computing Model to Design Hardware Accelerators**

Frederico Pratas and Leonel Sousa, “*Applying the Stream-Based Computing Model to Design Hardware Accelerators: A Case Study*”, In International Workshop on Systems, Architectures, MOdeling, and Simulation (SAMOS), Springer, July 2009.

- The proposed unified execution environment achieves significant **speedups** relatively to the single CPU core execution:
 - **4.5** for dense matrix multiplication
 - **2.8** for complex 3D fast Fourier transform
- **Future work:**
 - Implementation in more heterogeneous systems (more GPUs, more CPU cores, or special-purpose accelerators)
 - **Asynchronous** memory transfers
 - Adoption of advanced **scheduling policies**
 - Performance prediction and application **self-tuning**
 - To identify limits in performance to choose the processor to use (e.g GPU versus CPU)

- We were able to obtain a hardware implementation of the target algorithm using the stream-based
- Due to the stream-based parallel characteristics it naturally leads to scalable hardware architectures
- Typical folding procedures were used systematically to optimize the operations in hardware
- The results show that the solution is efficient, even not being a completely custom solution
- **Future work:**
- GPUs can be used as prototyping platforms to design hardware
- Development of software tools to assist hardware design.

Questions?

Thank you

