9TH INTERNATIONAL CONFERENCE ON
PARALLEL PROCESSING AND APPLIED MATHEMATICS
September 11-14, 2011, Torun, Poland

Deconvolution of 3D Fluorescence Microscopy Images using Graphics Processing Units

Luisa D'Amore, University of Naples Federico II
Valeria Mele, University of Naples Federico II
Livia Marcellino, University of Naples Parthenope
Diego Romano, ICAR-CNR

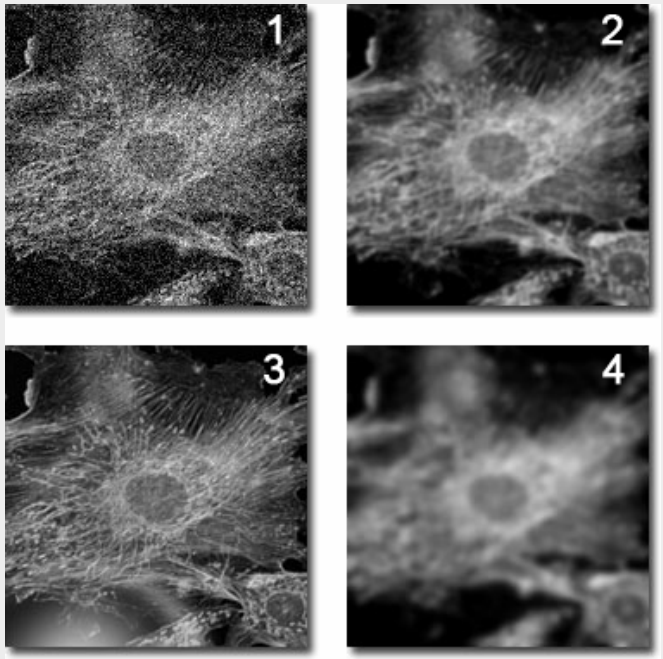
The present work

- we describe some performance analysis parameters for algorithms running on GPUs starting from the experience done with a

CUDA-based algorithm

for restoration of 3D images acquired by a fluorescence microscope based on the

Lucy-Richardson deconvolution algorithm



Images distorted by:
1.Noise
2.Scattering
3.Glare
4.Blur

Outline

- Application problem definition

Outline

- Application problem definition
- The algorithm and GPUs

Outline

- Application problem definition
- The algorithm and GPUs
- Algorithm effectiveness

Outline

- Application problem definition
- The algorithm and GPUs
- Algorithm effectiveness
- Performance analysis parameters

Outline

- Application problem definition
- The algorithm and GPUs
- Algorithm effectiveness
- Performance analysis parameters
- Conclusions and work in progress

Outline

- Application problem definition

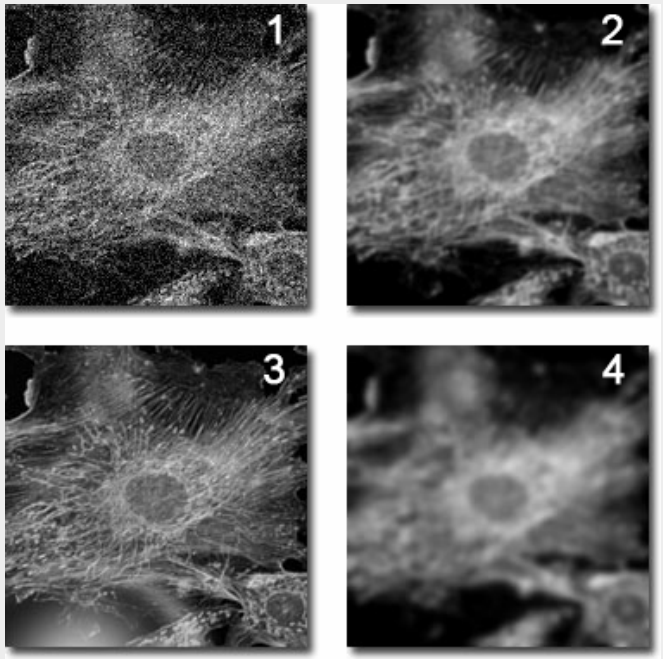
Fluorescence Microscopy

- A fluorescence microscope is a light microscope used to study properties of organic or inorganic substances, lighting them by UV rays
- In biomedical research, fluorescence microscopy is widely used to analyze 3-D structures of living biological cells and tissues.

- It uses the phenomenon of fluorescence and phosphorescence instead of - or in addition to - reflection and absorption.

Image Restoration by Deconvolution

- Fluorescence microscope imaging properties and measurement imperfections **distort** the original 3D image and reduce the maximal resolution obtainable by the imaging system, thereby restricting the quantitative analysis of the 3D specimen.

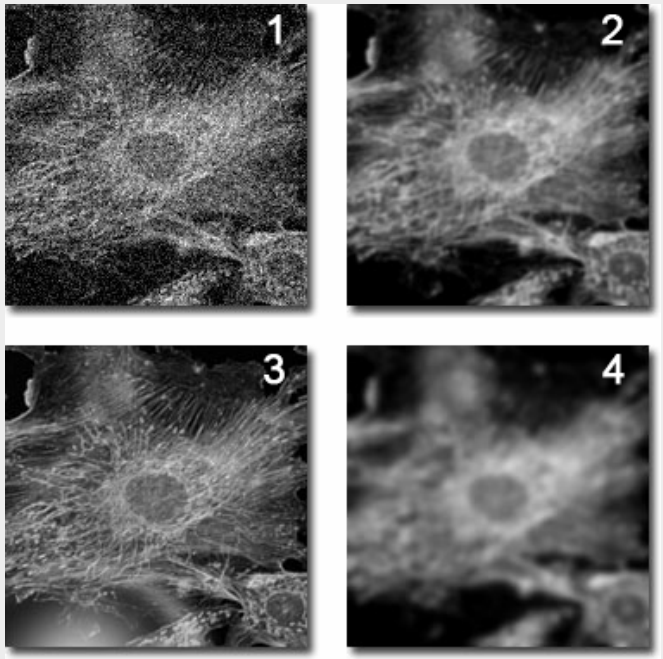


Images distorted by:
1.Noise
2.Scattering
3.Glare
4.Blur

Image Restoration by Deconvolution

- Fluorescence microscope imaging properties and measurement imperfections **distort** the original 3D image and reduce the maximal resolution obtainable by the imaging system, thereby restricting the quantitative analysis of the 3D specimen.

- **Deconvolution** is an operation that mitigates that distortion, that is by deconvolution we can restore the image



Images distorted by:
1.Noise
2.Scattering
3.Glare
4.Blur

Outline

- Application problem definition
- The algorithm and GPUs

The Lucy-Richardson algorithm

Mathematically, the deconvolution is an **inverse problem** written as:

$$H[f] = g$$

The Lucy-Richardson algorithm

Mathematically, the deconvolution is an **inverse problem** written as:

$$H[f] = g$$

•that we solve using the **Expectation Maximization-Richardson Lucy algorithm** (in matrix form)

$$f_{k+1} = f_k H^T \frac{g}{Y_k}, \qquad Y_k = H f_k$$

→ To be accellerated following [2]

[2] [D. S. C. Biggs and M. Andrews - Acceleration of iterative image restoration algorithms. Applied Optics. Vol 36(8), pp. 17661775, 1997]

The Lucy-Richardson algorithm

EM-RL (spatial domain)

$$f_{k+1} = f_k H^T \frac{g}{Y_k}, \qquad Y_k = H f_k$$

The Lucy-Richardson algorithm

EM-RL (spatial domain)

$$f_{k+1} = f_k H^T \frac{g}{Y_k}, \qquad Y_k = H f_k$$

Convolution Theorem

EM-RL (frequency domain)

$$f_{k+1} = f_k \cdot \mathscr{F}^{-1}\left(H^* \cdot \mathscr{F}\left(\frac{g}{\mathscr{F}^{-1}(H \cdot F_k)}\right)\right), \quad k > 0$$

Where:

- capital letters denote the transformed functions
- $\mathscr{F}$ and $\mathscr{F}^{-1}$ are the Furier transform and the Inverse Fourier transform operators
- H is the Optical Transfer Function (OTF) and H^* is its complex conigate

ARL algorithm using a GPU

- This iterative algorithm is essentially made of some macro-operations that can be implemented in parallel:

$$f_{k+1} = f_k \cdot \mathscr{F}^{-1}\left(H^* \cdot \mathscr{F}\left(\frac{g}{\mathscr{F}^{-1}(H \cdot F_k)}\right)\right), \quad k > 0$$

ARL algorithm using a GPU

- This iterative algorithm is essentially made of some macro-operations that can be implemented in parallel:

$$f_{k+1} = f_k \cdot \mathscr{F}^{-1}\left(H^* \cdot \mathscr{F}\left(\frac{g}{\mathscr{F}^{-1}(H \cdot F_k)}\right)\right), \quad k > 0$$

  - Entry-wise matrix product
  - Entry-wise matrix ratio
  - Conjugate of complex matrix
  - Multiply-add combination of scalar-matrix and matrix-matrix (Entry-wise) operations
  - Calculation of acceleration parameter
  - DFT
  - Inverse DFT

ARL algorithm using a GPU

- This iterative algorithm is essentially made of some macro-operations that can be implemented in parallel:

$$f_{k+1} = f_k \cdot \mathscr{F}^{-1}\left(H^* \cdot \mathscr{F}\left(\frac{g}{\mathscr{F}^{-1}(H \cdot F_k)}\right)\right), \quad k > 0$$

  - Entry-wise matrix product
  - Entry-wise matrix ratio
  - Conjugate of complex matrix
  - Multiply-add combination of scalar-matrix and matrix-matrix (Entry-wise) operations
  - Calculation of acceleration parameter
  - DFT
  - Inverse DFT

- We just replaced the macro-operation sequential procedures with parallel CUDA kernels each followed by a synchronization barrier

ARL algorithm using a GPU

- Entry-wise calculations can be easily separated in different independent tasks on different cores utilizing the CUDA threading mechanism and there is almost no divergence cost during their execution on the GPU, as there are no conditional statements in the parallel version of such operations.

ARL algorithm using a GPU

- Entry-wise calculations can be easily separated in different independent tasks on different cores utilizing the CUDA threading mechanism and there is almost no divergence cost during their execution on the GPU, as there are no conditional statements in the parallel version of such operations.

- About the DFT calculation, we choose to utilize the CUDA optimized CUFFT library [6], that has been modeled after the widely used FFTW [7].

[6] [NVIDIA Corporation, Documentation for CUDA FFT (CUFFT) Library, 2008, http://developer:download:nvidia:com=compute=cuda=3¡2¡prod=toolkit=docs=CUBLASLibrary:pdf]

[7] [M. Frigo, S.G. Johnson, The design and implementation of fftw3, Proceedings of the IEEE, Volume 93, 2005]

ARL algorithm using a GPU

- Entry-wise calculations can be easily separated in different independent tasks on different cores utilizing the CUDA threading mechanism and there is almost no divergence cost during their execution on the GPU, as there are no conditional statements in the parallel version of such operations.

- About the DFT calculation, we choose to utilize the CUDA optimized CUFFT library [6], that has been modeled after the widely used FFTW [7].

- Each iteration is executed completely on the GPU device, without the need to move data between host and device memory (let suppose we have enough space on the GPU global memory).

[6] [NVIDIA Corporation, Documentation for CUDA FFT (CUFFT) Library, 2008, http://developer:download:nvidia:com=compute=cuda=3¡2¡prod=toolkit=docs=CUBLASLibrary:pdf]

[7] [M. Frigo, S.G. Johnson, The design and implementation of fftw3, Proceedings of the IEEE, Volume 93, 2005]

ARL algorithm using a GPU

- The CPU works as an high level macroscopic control unit which submit kernel executions, synchronizes the macro-operations, and executes the control flow directives.

ARL algorithm using a GPU

- The CPU works as an high level macroscopic control unit which submit kernel executions, synchronizes the macro-operations, and executes the control flow directives.

- We can consider each kernel as a parallel algorithm to analyze.

ARL algorithm using a GPU

- The CPU works as an high level macroscopic control unit which submit kernel executions, synchronizes the macro-operations, and executes the control flow directives.
- We can consider each kernel as a parallel algorithm to analyze.
- The iterations are stopped when the I-divergence reaches the minimum value, as usual for those iterative algorithms employed to solve ill posed problems, because they suffer from the so-called semi-convergence behavior.

Outline

- Application problem definition
- The algorithm and GPUs
- Algorithm effectiveness

ARL algorithm using a GPU

- Our algorithm has been implemented in C with CUDA extension to run on a system with 2 NVIDIA Tesla C1060

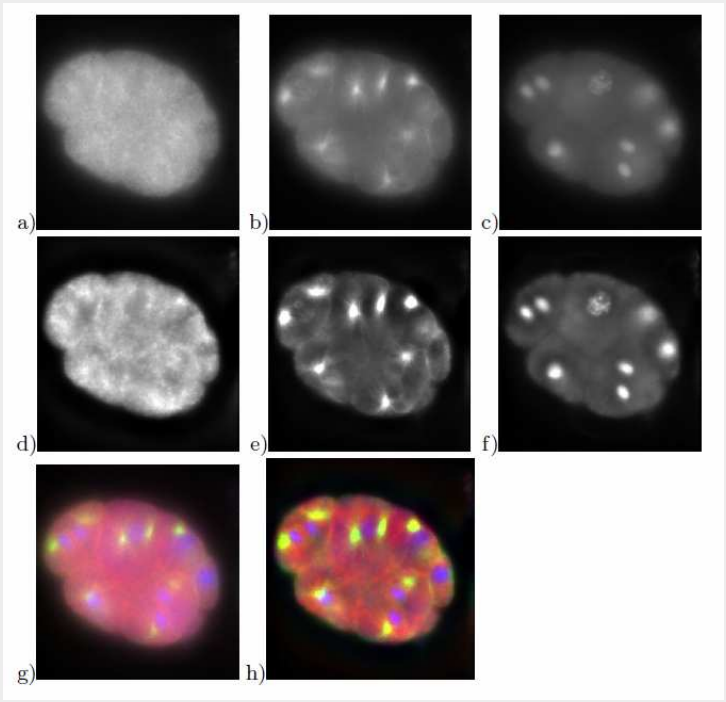
Tesla C1060 details

Codename	GT200
Interface	PCI Express 2.0 16x
Streaming Multiprocessors (SM)	30
Streaming Processor (SP) cores	240
SP core Frequency	1300 MHz
SDRAM	4 GB
Memory Bandwidth	102 GB/2
Peak GFLOPs Single precision	936
Peak GFLOPs Double precision	78

Experiments: TEST 1

- Images size is 256 × 271 × 103.
- a), b) e c) show three black and white bidimensional degraded slices at z = 49, related to the three fluorescence emissions (CY3, FITC, DAPI);
- d) e) f) show the corresponding restored images obtained after 9 iterations;
- g) is the RGB sum of a) b) and c);
- h) is the RGB sum of d), e) and f).

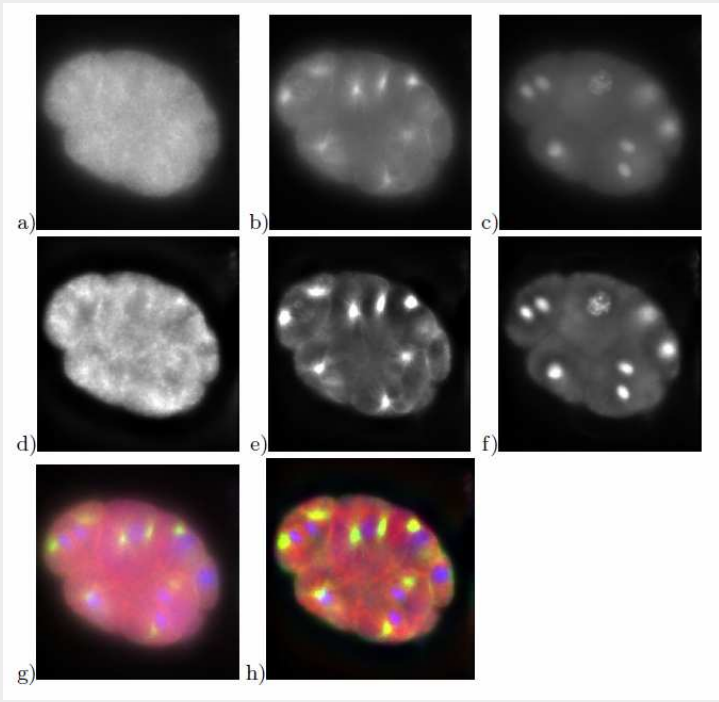
C. Elegans embrio

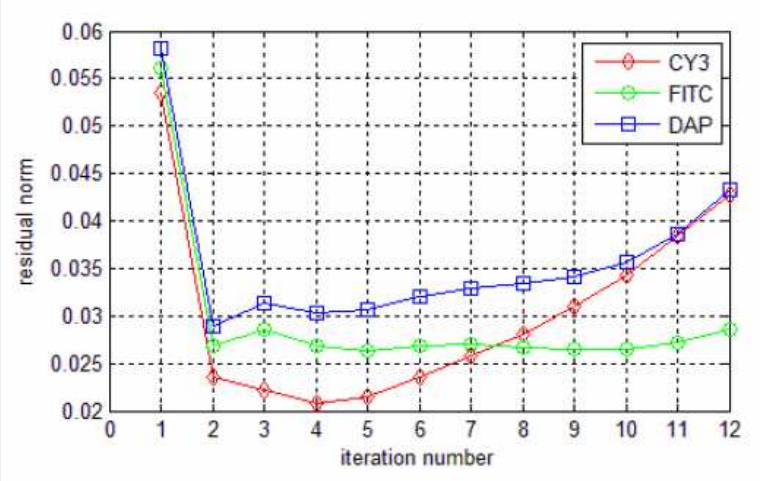


Experiments: TEST 1

- Images size is 256 × 271 × 103.
- a), b) e c) show three black and white bidimensional degraded slices at z = 49, related to the three fluorescence emissions (CY3, FITC, DAPI);
- d) e) f) show the correspondent restored images obtained after 9 iterations;
- g) is the RGB sum of a) b) and c);
- h) is the RGB sum of d), e) and f).

C. Elegans embrio



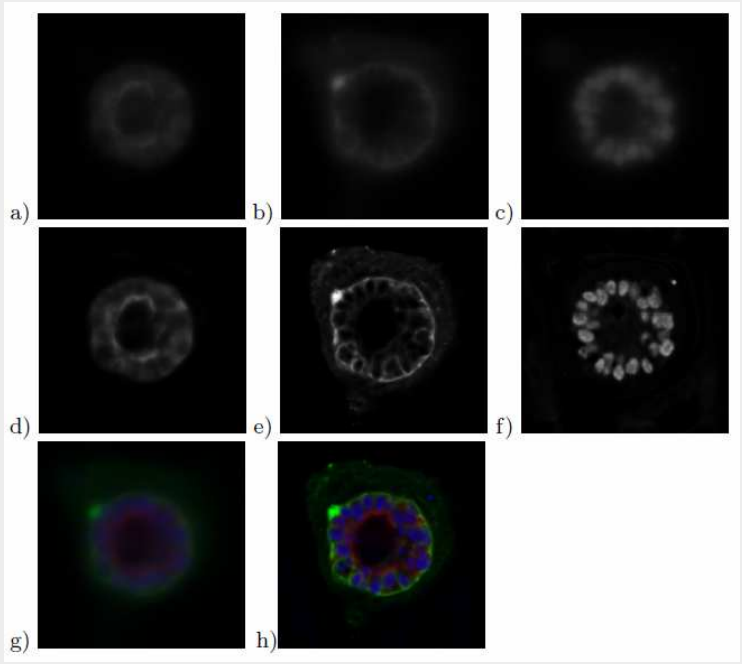


- ARL residuals behavior corresponding to CY3, FITC, DAPI, respectively.
- According to the semi-convergence of ARL algorithm., the minimum is reached at iterations 4, 5 and 4, respectively.

Experiments: TEST2

- Images size is 512 × 512 × 45. .
- a), b) e c) show three black and white bidimensional degraded slices at z = 29, related to the three fluorescence emissions (CY3, FITC, DAPI);
- d) e) f) show the correspondent restored images;
- g) is the RGB sum of a) b) and c);
- h) is the RGB sum of d), e) and f).

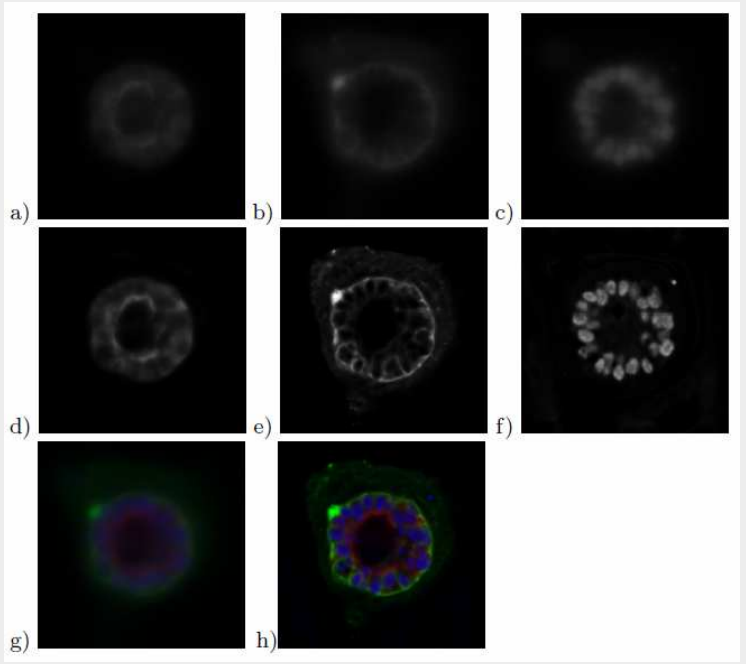
berry of a mouse mammary epithelial.

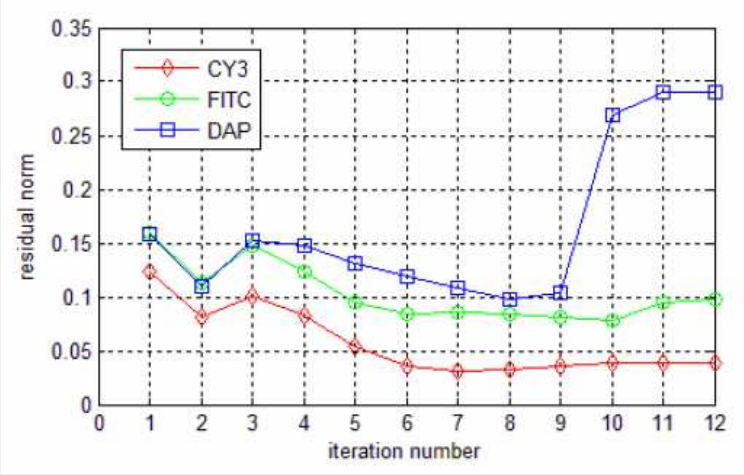


Experiments: TEST2

berry of a mouse mammary epithelial.

- Images size is 512 × 512 × 45. .
- a), b) e c) show three black and white bidimensional degraded slices at z = 29, related to the three fluorescence emissions (CY3, FITC, DAPI);
- d) e) f) show the correspondent restored images;
- g) is the RGB sum of a) b) and c);
- h) is the RGB sum of d), e) and f).

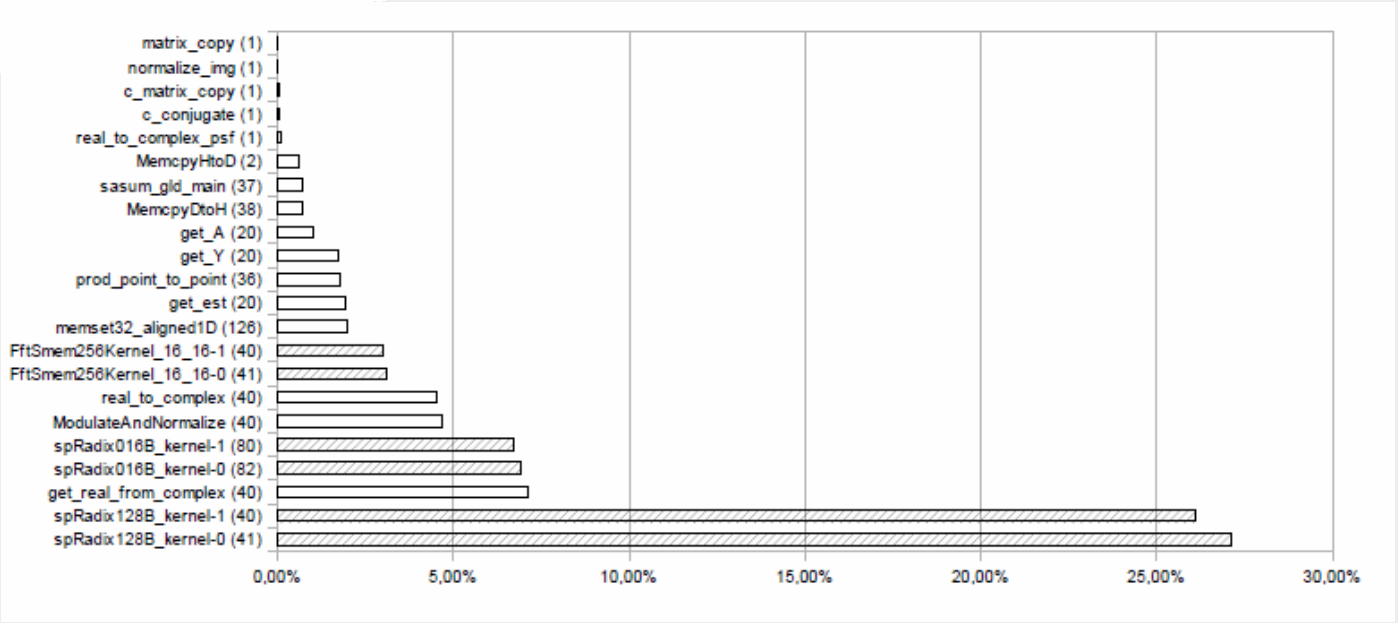




- ARL residuals behavior corresponding to CY3, FITC, DAPI, respectively.
- According to the semi-convergence of ARL algorithm, the minimum is reached at iterations 7, 10 and 8, respectively.

ARL algorithm using a GPU

- Most of the total execution time is devoted to DFT and IDFT calculation.



Percentage of kernel execution time over total time, per kernel.
Kernels using CUFFT are marked with stripes. In brackets the number of executions for a deconvolution with a 20 iterations loop.

ARL algorithm using a GPU

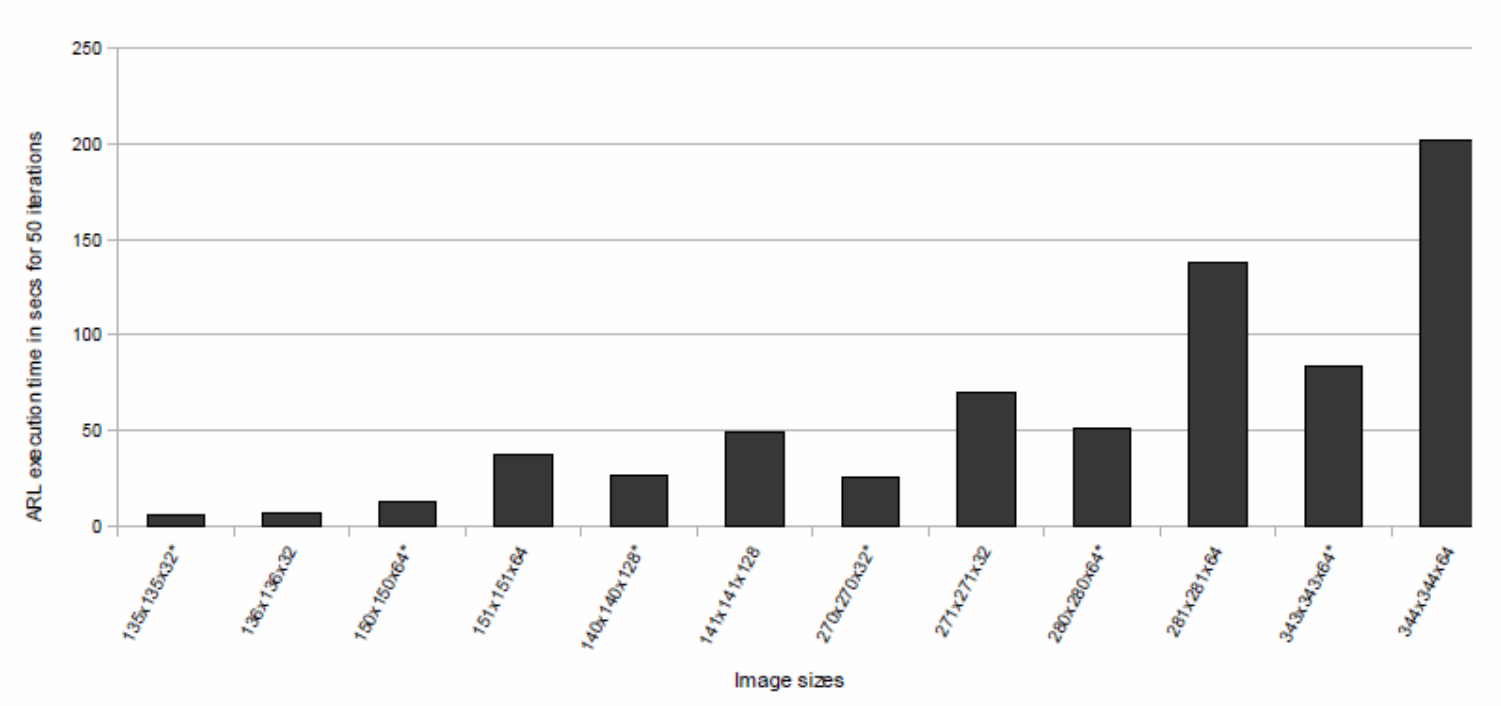
- As those calculations are performed using the CUFFT package:
  - we rely on developers directions to gain the best performance

ARL algorithm using a GPU

- As those calculations are performed using the CUFFT package:
  - we rely on developers directions to gain the best performance
  - The FFT algorithms implemented in the package work with the best accuracy and performance if the transform sizes are (in descending order):
    1. power of a single factor, if the transform fits in CUDA's shared memory,
    2. power of two, if the transform doesn't fit in CUDA's shared memory,
    3. power of four or other small primes (such as three, five, or seven).

ARL algorithm using a GPU

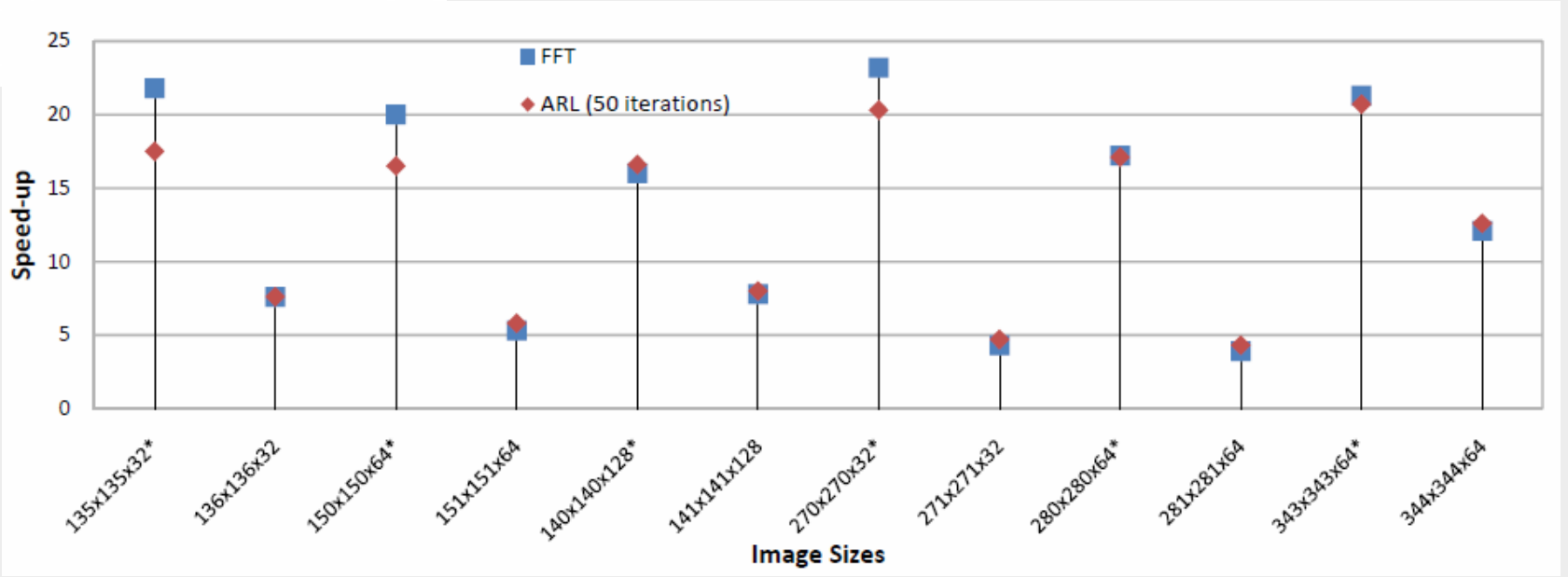
- The execution time is shorter if image sizes are factorizable as of $2^a \cdot 3^b \cdot 4^c \cdot 5^d$, longer if they are not.



Execution time of ARL algorithm with different image sizes. Those marked with "*" are factorizable as $2^a \cdot 3^b \cdot 4^c \cdot 5^d$

ARL algorithm using a GPU

- Slow FFTs prolong the entire algorithm execution
- So the parallel ARL on GPU leads to smaller speed-ups if the image sizes are not factorizable with primes.



Speed-up of ARL algorithm.
Sequential FFT has been implemented utilizing FFTW. Image sizes marked with "*" are factorizable as $2^a \cdot 3^b \cdot 4^c \cdot 5^d$

Outline

- Application problem definition
- The algorithm and GPUs
- Algorithm effectiveness
- Performance analysis parameters

How to evaluate the performance

- The CUDA-version algorithm leads clearly to a significant gain in comparison with the sequential one

How to evaluate the performance

- The CUDA-version algorithm leads clearly to a significant gain in comparison with the sequential one
- But this gain is not easily to explain using the classical parameters for the evaluation of parallel algorithms

How to evaluate the performance

- The CUDA-version algorithm leads clearly to a significant gain in comparison with the sequential one
- But this gain is not easily to explain using the classical parameters for the evaluation of parallel algorithms
- So, we notice the need to model the GPUs architectures and their characteristics to describe the behavior of GPU-algorithms and what we can expect of them

How to evaluate the performance

- The CUDA-version algorithm leads clearly to a significant gain in comparison with the sequential one
- But this gain is not easily to explain using the classical parameters for the evaluation of parallel algorithms
- So, we notice the need to model the GPUs architectures and their characteristics to describe the behavior of GPU-algorithms and what we can expect of them
- Let's introduce some results about this focus, following from the application we described and some others

Parallel programming- Preliminaries

Definition 1:
Given any algorithm A, it may be decomposed into two parts (two sets of instructions):

- Seq_A , refers to the *sequential part* of A, made of operations to execute sequentially,
- Par_A refers to the *parallel part* of A, including operations that can be executed concurrently.

Parallel programming- Preliminaries

Definition 2:
Given any algorithm A, and its parallelized version *A'*
– T_{seq} *(A')* is the time to execute *A'* instructions sequentially,
– T_{conc} *(A', N)* is the time to execute *A'* instructions if the parallel part is executed by N concurrent streams of execution.

So

$$T_{seq}(A') = T_{seq}(Seq_{A'}) + T_{seq}(Par_{A'})$$

and

$$T_{conc}(A', N) = T_{seq}(Seq_{A'}) + T_{conc}(Par_{A'}, N)$$

Parallel programming- Preliminaries

Definition 3:
Let T_{seq} (A) be the sequential execution time of a given algorithm A.
It is always decomposable in:
– $T_{seq[f\ lop]}$(A), that measures the time spent in floating point operations by A,
– $T_{seq[mem]}$ (A), that measures the time spent in memory accesses by A.
Thus

$$T_{seq}(A) = T_{seq[f\ lop]}(A) + T_{seq[mem]}(A)$$

Parallel programming- Preliminaries

Definition 3:
Let T_{seq} (A) be the sequential execution time of a given algorithm A.
It is always decomposable in:
– $T_{seq[f\ lop]}$(A), that measures the time spent in floating point operations by A,
– $T_{seq[mem]}$ (A), that measures the time spent in memory accesses by A.
Thus

$$T_{seq}(A) = T_{seq[f\ lop]}(A) + T_{seq[mem]}(A)$$

We can give an analogous definition for T_{conc} (A', N), so it's also

$$T_{conc}(A', N) = T_{conc[f\ lop]}(A', N) + T_{conc[mem]}(A', N)$$

Architecture – functional scheme

Let's now suppose that our GPU-based computing architecture is like the one described in [10], made of

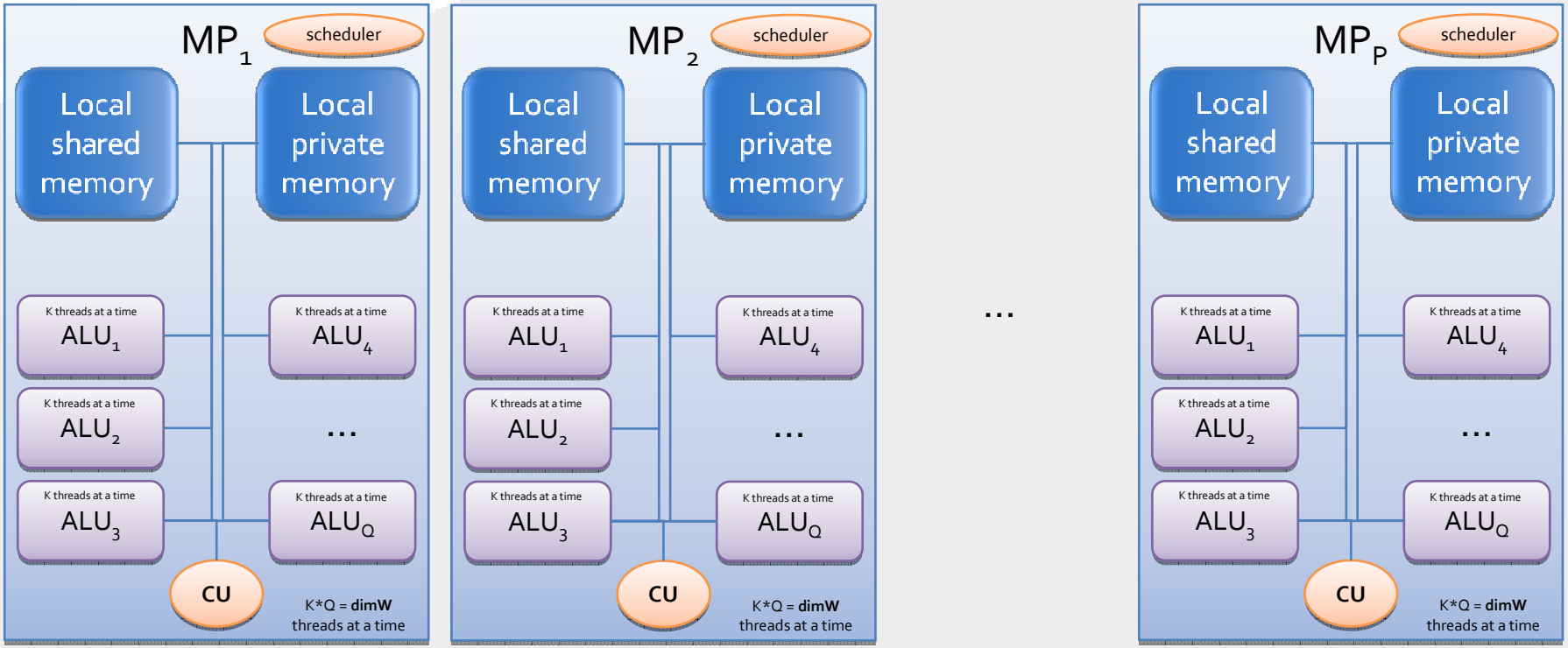
P Multiprocessors (MP), and ***Q*** ALU per MP.

[10][V. Mele, A. Murli, D. Romano, Some remarks on performance evaluation in parallel GPU computing, Preprint del Dipartimento di Matematica e applicazioni, Univerity of Naples Federico II, 2011]

Architecture – functional scheme

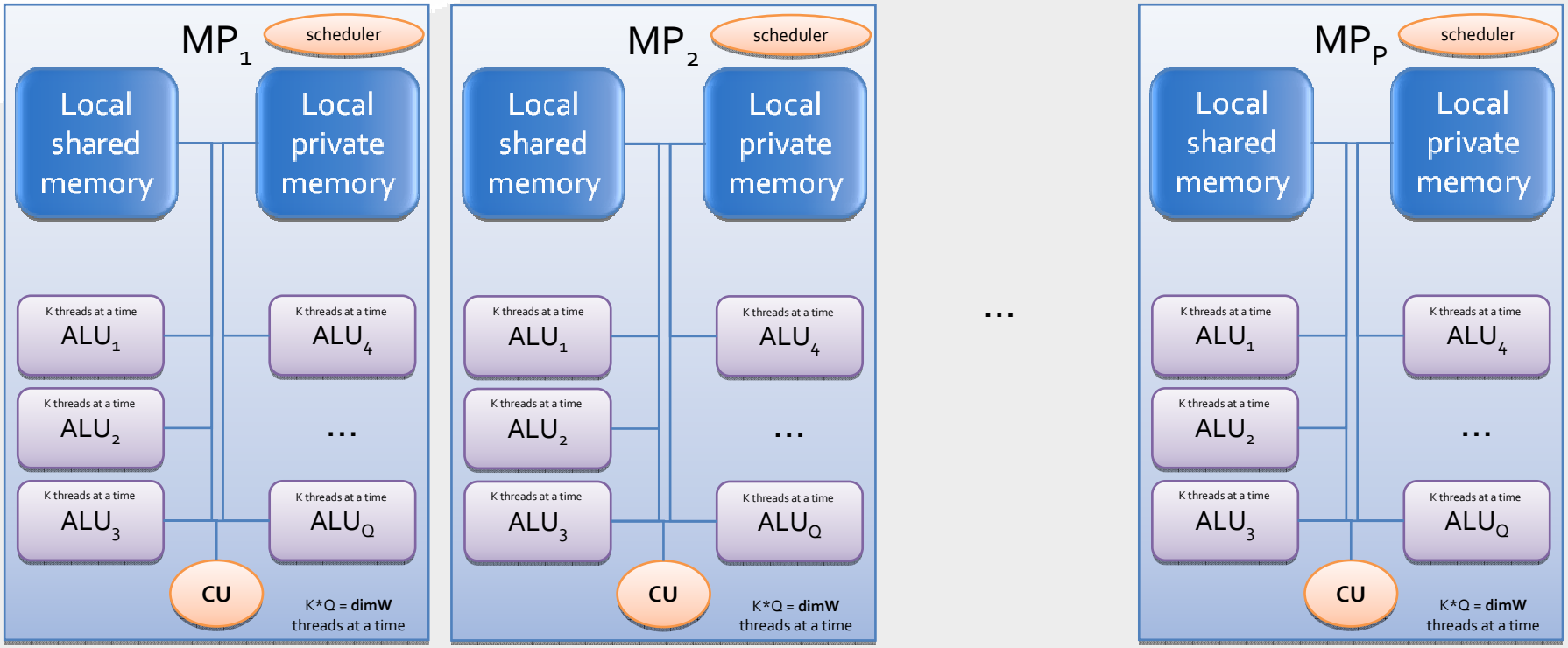
Let's now suppose that our GPU-based computing architecture is like the one described in [10], made of

P Multiprocessors (MP), and ***Q*** ALU per MP.



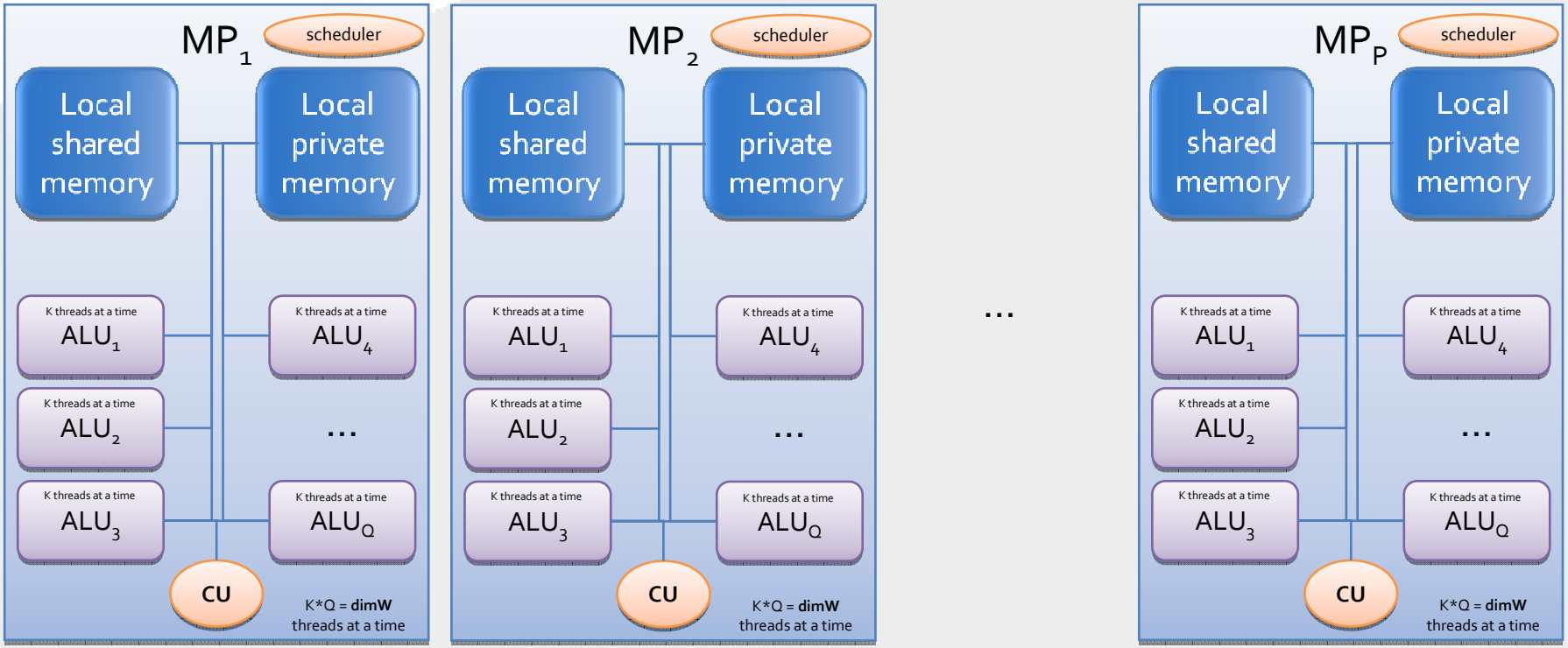
Architecture – functional scheme

Let's call ***warp*** a group of threads that is the execution unit on that machine, that is the fixed number of threads running simultaneously on the ALUs of each MP at the same time.

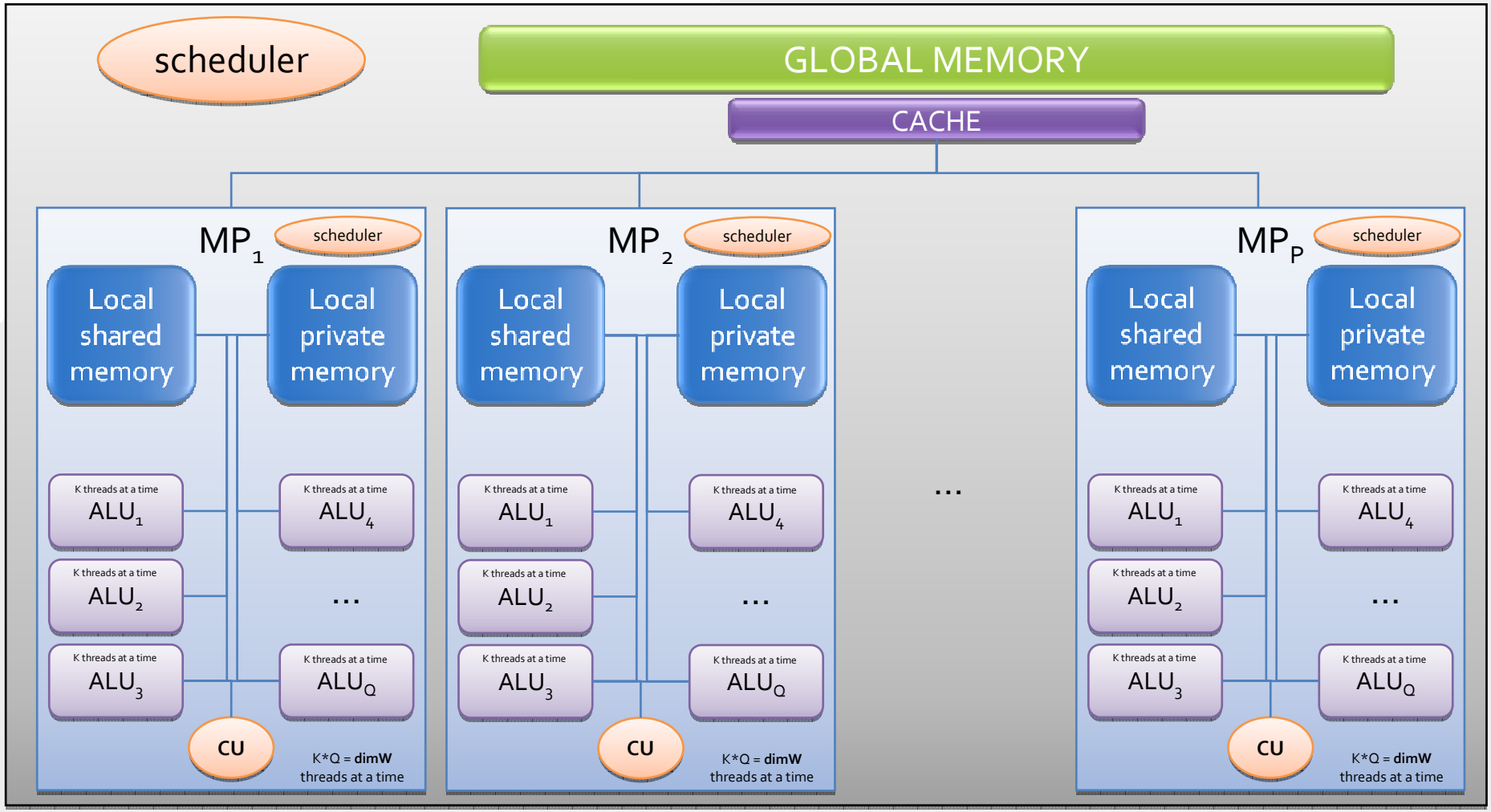


Architecture – functional scheme

Let be ***dimW*** the dimension of the warp



Architecture – functional scheme



Architecture – Occupancy

Suppose that the parallel part of A', $Par_{A'}$, is executed by ***p*** sets of ***q*** threads and let q be a multiple of dimW.

Definition 4:
At a given instant, the ***occupancy*** of each MP is a function of the number of threads running concurrently on that MP, say p^1 *q*, with $p^1 \leq p$, and is defined as

$$\vartheta(p^1 q) = \frac{\#\text{threads\_per\_MP}(p^1 q)}{\text{dimW}} \cdot \frac{1}{\#\text{max\_warps\_per\_MP}} = \frac{\#\text{active\_warps\_MP}}{\#\text{max\_warps\_per\_MP}}$$

where
– *#threads_per_MP* ≤ *#max_threads_per_MP* is both hardware and p^1 *q* dependent
– *#max_warps_per_MP* is hardware dependent
– *#threads_per_MP*(p^1 *q*)*/dimW* ≤ *#max_warps_per_MP*

Architecture – Occupancy

Suppose that the parallel part of A', $Par_{A'}$, is executed by ***p*** sets of ***q*** threads and let q be a multiple of dimW.

Definition 4:
At a given instant, the ***occupancy*** of each MP is a function of the number of threads running concurrently on that MP, say $p^1\ q$, with $p^1 \le p$, and is defined as

$$\vartheta(p^1 q) = \frac{\#\text{threads\_per\_MP}(p^1 q)}{\text{dimW}} \cdot \frac{1}{\#\text{max\_warps\_per\_MP}} = \frac{\#\text{active\_warps\_MP}}{\#\text{max\_warps\_per\_MP}}$$

where
– *#threads_per_MP* ≤ *#max_threads_per_MP* is both hardware and $p^1\ q$ dependent
– *#max_warps_per_MP* is hardware dependent
– $\#threads\_per\_MP(p^1\ q)/dimW \le \#max\_warps\_per\_MP$

- The occupancy describes how much are exploited the capabilities of the MPs.

Performance Analysis

Proposition

The expected total execution time of a parallel algorithm *A′* designed to run on a single MP of the described architecture by p sets of q threads, could be written as follows:

$$T_{conc}(A', pq) = T_{conc}(A', p, q) =$$
$$= T_{conc[\,flop]}(A', p, q, \vartheta) + T_{conc[mem]}(A', p, q, \vartheta) + T_{Oh}(A', p, q)$$

Performance Analysis

Proposition

The expected total execution time of a parallel algorithm *A'* designed to run on a single MP of the described architecture by p sets of q threads, could be written as follows:

$$T_{conc}(A', pq) = T_{conc}(A', p, q) =$$
$$= T_{conc[\,flop]}(A', p, q, \vartheta) + T_{conc[mem]}(A', p, q, \vartheta) + T_{Oh}(A', p, q)$$

Where, depending on occupancy

if $T_{conc[flop]}(warp_i, dimW)$ is the execution time spent in floating point operations by the i^{th} warp, on a single MP,

$$T_{conc[\,flop]}(A', p, q, \vartheta) = \frac{\sum_{i=0}^{q/\dim W - 1} T_{conc[\,flop]}(warp_i, \dim W)}{k_1 \cdot \vartheta(pq)} \qquad k_1 > 1$$

Performance Analysis

Proposition

The expected total execution time of a parallel algorithm *A'* designed to run on a single MP of the described architecture by p sets of q threads, could be written as follows:

$$T_{conc}(A', pq) = T_{conc}(A', p, q) =$$
$$= T_{conc[\,flop]}(A', p, q, \vartheta) + T_{conc[mem]}(A', p, q, \vartheta) + T_{Oh}(A', p, q)$$

Where, depending on occupancy

Performance Analysis

Proposition

The expected total execution time of a parallel algorithm *A'* designed to run on a single MP of the described architecture by p sets of q threads, could be written as follows:

$$T_{conc}(A', pq) = T_{conc}(A', p, q) =$$
$$= T_{conc[\,flop]}(A', p, q, \vartheta) + T_{conc[mem]}(A', p, q, \vartheta) + T_{Oh}(A', p, q)$$

Where, depending on occupancy

if $T_{conc[mem]}(warp_i, dimW)$ is the execution time spent in memory accesses by the i^{th} warp, on a single MP,

$$T_{conc[mem]}(A', p, q, \vartheta) = \frac{\sum_{i=0}^{q/\dim W - 1} T_{conc[mem]}(warp_i, \dim W)}{\varphi(\vartheta(pq))} \qquad 0 < \varphi(\vartheta(pq)) \le k_2 > 1$$

Performance Analysis

Proposition

The expected total execution time of a parallel algorithm *A'* designed to run on a single MP of the described architecture by p sets of q threads, could be written as follows:

$$T_{conc}(A', pq) = T_{conc}(A', p, q) =$$
$$= T_{conc[\,flop]}(A', p, q, \vartheta) + T_{conc[mem]}(A', p, q, \vartheta) + T_{Oh}(A', p, q)$$

and where

$T_{Oh}(p, q)$ is the overhead that includes cost of kernel launch, host/device data transfers, synchronization, divergence and data non-coalescence.

Outline

- Application problem definition
- The algorithm and GPUs
- Algorithm effectiveness
- Performance analysis parameters
- Conclusions and work in progress

Conclusions

- We described the benefits arising from facing medical imaging problems on GPUs, that are non-expensive parallel processing devices available on many up-to-date personal computers.

Conclusions

- We described the benefits arising from facing medical imaging problems on GPUs, that are non-expensive parallel processing devices available on many up-to-date personal computers.

- We consider the **deconvolution of 3D Fluorescence Microscopy images**:
  - The algorithm reaches a high performance on GPUs because many of the steps in the sequential algorithm consist in entry-wise matrix operations, that means they are embarrassingly parallel tasks efficiently executable on many-core GPUs.
  - such operations on big images can keep the GPU well occupied making the most of the Streaming Multiprocessor (SM) compute capabilities.

Conclusions

- We described the benefits arising from facing medical imaging problems on GPUs, that are non-expensive parallel processing devices available on many up-to-date personal computers.

- We consider the **deconvolution of 3D Fluorescence Microscopy images**:
  - The algorithm reaches a high performance on GPUs because many of the steps in the sequential algorithm consist in entry-wise matrix operations, that means they are embarrassingly parallel tasks efficiently executable on many-core GPUs.
  - such operations on big images can keep the GPU well occupied making the most of the Streaming Multiprocessor (SM) compute capabilities.

- We built an efficient implementation with significant speed ups on the real case.

Conclusions

- Obtained results open some considerations about the applicability of classical **evaluation parameters** and leads to the aim of modeling the performance of algorithms on the modern GPU-enhanced computing environments

Conclusions

- Obtained results open some considerations about the applicability of classical **evaluation parameters** and leads to the aim of modeling the performance of algorithms on the modern GPU-enhanced computing environments

- We first expressed the execution time of the algorithm in terms of the widely used optimization parameter, that is the **occupancy**.

Conclusions

- Obtained results open some considerations about the applicability of classical **evaluation parameters** and leads to the aim of modeling the performance of algorithms on the modern GPU-enhanced computing environments
- We first expressed the execution time of the algorithm in terms of the widely used optimization parameter, that is the **occupancy**.
- But it's not the end of the way…

Work in progress

- We should identify but also well-define some parameters that influence the expected speed up and the actual performance

Work in progress

- We should identify but also well-define some parameters that influence the expected speed up and the actual performance
- First of all, the $\varphi(\vartheta(pq))$ function has to be characterized

Work in progress

- We should identify but also well-define some parameters that influence the expected speed up and the actual performance
- First of all, the $\varphi(\vartheta(pq))$ function has to be characterized
- It is useful to define a kind of *ideal Speed up* to know what gain to expect when programming in a GPU-enabled environment.

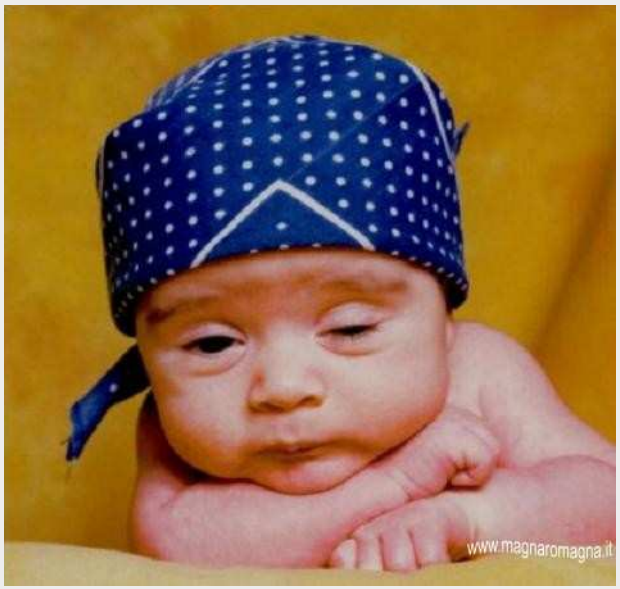
Work in progress

- We should identify but also well-define some parameters that influence the expected speed up and the actual performance
- First of all, the $\varphi(\vartheta(pq))$ function has to be characterized
- It is useful to define a kind of *ideal Speed up* to know what gain to expect when programming in a GPU-enabled environment.
- Each found parameter has to be studied with a variety of known algorithm and applications field to validate the performance model

Work in progress

- We should identify but also well-define some parameters that influence the expected speed up and the actual performance
- First of all, the $\varphi(\vartheta(pq))$ function has to be characterized
- It is useful to define a kind of *ideal Speed up* to know what gain to expect when programming in a GPU-enabled environment.
- Each found parameter has to be studied with a variety of known algorithm and applications field to validate the performance model
- For now…

Thank you
For Attention!



References

1. Agard D. A., Hiraoki Y. and Sedat J.W. - *Three-dimensional microscopy: image processing for high-resolution subcellular imaging,* Proc. SPIE 1161, 1989, pp. 24-30.

2. D. S. C. Biggs and M. Andrews - *Acceleration of iterative image restoration algorithms. Applied Optics*. Vol 36(8), pp. 17661775, 1997.

3. Csiszar I. - *Why least squares and maximum entropy? An axiomatic approach to inference for linear inverse problems*. The Annals of Statistics, 1991, Vol. 19, n.4, pp. 2031-2066.

4. L. B. Lucy - *An iterative technique for the rectification of observed images*. The Astronomical Journal. Vol 79(6), pp. 745-754, 1974.

5. W. H. Richardson - *Bayesian-based iterative method of image restoration*. Journal of the Optical Society of America. Vol 62(1), pp. 55-59, 1972.

6. A. N. Tikhonov, and V. Y. Arsenin - *Solutions of ill-posed problems*, (1977), New York, Wiley

References

6. NVIDIA Corporation, *Documentation for CUDA FFT (CUFFT) Library*, 2008, http ://developer:download:nvidia:com=compute=cuda=3¡2¡prod=toolkit=docs=CUBLASLibrary: pdf

9. M. Frigo, S.G. Johnson, *The design and implementation of fftw3*, Proceedings of the IEEE, Volume 93, 2005

10. V. Mele, A. Murli, D. Romano, *Some remarks on performance evaluation in parallel GPU computing*, Preprint del Dipartimento di Matematica e applicazioni, Univerity of Naples Federico II, 2011

11. Volkov, V., and Demmel, J. W. 2008. *Benchmarking GPUs to tune dense linear algebra*, Proceedings of the ACM/IEEE Conference on Supercomputing (SC08), 2008

12. Volkov, V. *Better performance at lower occupancy*, Presentations at GPU Technology Conference 2010 (GTC 2010)