Processing biomedical images on the GPU: Implementation of an optimized CUDA library

Antonio Ruiz, Manuel Ujaldón, Timothy Hartley, Umit Catalyurek, Francisco Igual, Rafael Mayo

Universidad de Málaga (Málaga, Spain), Ohio State University (Coumbus, USA) and Universitat Jaume I (Castellón, Spain)

Summary and Motivation

- Cancer prognosis: early detection of cancer
- Based on the evaluation of tissue samples ⇒ **large scale images**
- **Main goal**: Optimize the execution of biomedical image analysis procedures exclusively on the GPU
- Why do we need HPC here?
  1. Due to the large size of the images
     – A typical $120K \times 120K$ image occupies more than **40GB**
  2. Due to the large processing time on CPU

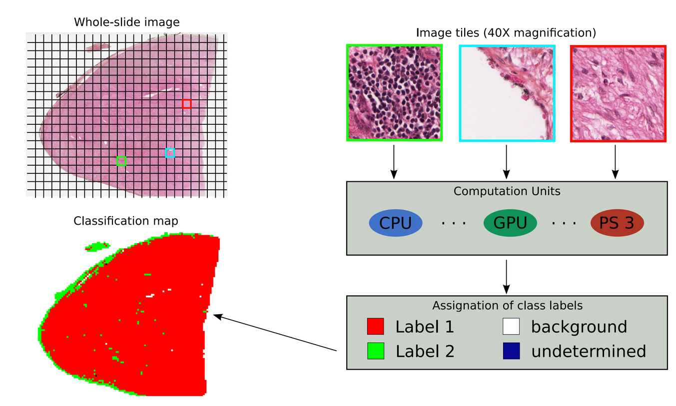
Image size	Matlab	C++
SMALL	2h 57' 29"	43' 40"
MEDIUM	6h 25' 45"	1h 34' 51"
LARGE	11h 39' 28"	2h 51' 23"

  3. Due to the large number of medical samples per patient
     – **Months or even years of computation**
- **Our result**: optimized library of biomedical image analysis and classification kernels, including:
  – Color conversion
  – Feature extraction routines
  – Classifiers

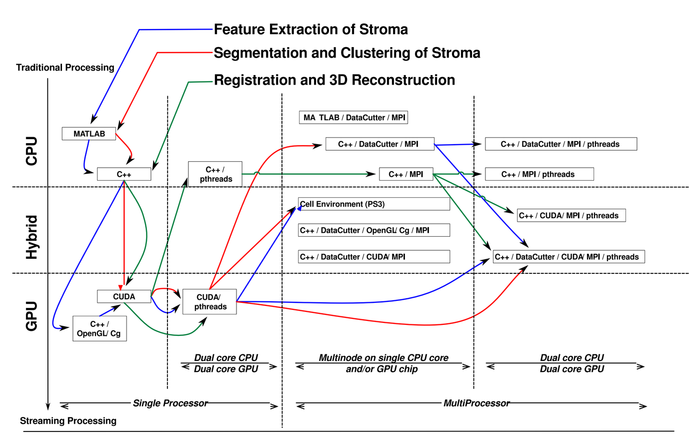
1

General Framework and Methodology

Process overview

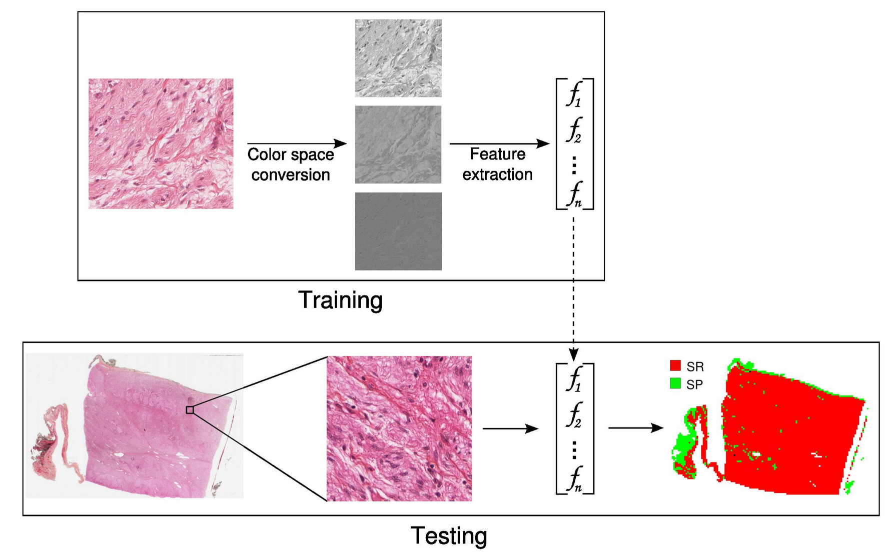


Explored implementations



2

Tile Processing. Color Conversion



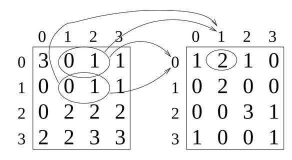
- Color conversion procedures are typically **GPU-like**
- Results attained on GPU are very promising ⇒ Stream-oriented procedures
- Some results:

Format conversion	CPU time	GPU time	GPU Speedup
RGB to XYZ	140.01 ms	1.27 ms	**109.47x**
RGB to Luv	273.83 ms	1.42 ms	**191.62x**
RGB to L*A*B*	267.92 ms	2.23 ms	**119.66x**

3

Feature Extraction: Co-occurrence matrices (I)

- Introduced by Haralick in 1973
- Joint histogram of intensity levels of a pair of pixels with a given spatial relationship $[d_x, d_y]$
- Intermediate data structure to extract features: contrast, correlation,...
- Simple example: for a 4×4 window, and four intensity levels:

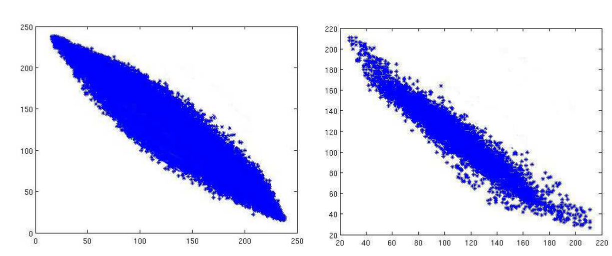


- Co-occurrence matrix calculation is a **CPU-like** operation
- **Goal**: optimize it for GPU calculation
- Main optimization strategies:
  1. Discretized co-occurrence matrices ⇒ Smaller ⇒ Fit in **shared memory**
  2. Non discretized co-occurrence matrices ⇒ Use sparse representations ⇒ Fit in **shared memory**
  3. Per-pixel calculation of the co-occurrence matrix ⇒ **Argenti's method** (neighbour co-oc. matrices are related)

4

Feature Extraction: Co-occurrence matrices (II)

- Potential optimizations according to the shape of the matrix:



  – Diagonal dense storage
  – Improvement of insertions on sparse formats
  – Blocked computation of the diagonal values of the co-occurrence matrix (in progress)
- Results:

Impact of discretization

Co. size	CPU	Dense	S.up
16x16	2.82	0.23	12.26x
32x32	2.82	0.31	9.09x
64x64	2.82	0.67	4.20x
128x128	2.82	2.09	1.34x
256x256	2.82	7.58	0.37x

Impact of window size

Window	CPU	Sparse	S.up
16x16	2.82	0.39	7.23x
32x32	3.04	0.74	4.10x
64x64	3.08	1.74	1.77x
128x128	2.94	7.70	0.38x
256x256	2.96	46.49	0.06x

- Each optimization focuses a given scenario

5

Feature Extraction: Zernike Moments

- Spatial domain filter ⇒ direct way to capture texture properties
- Legendre and Zernike polynomials represent an image by a set of mutually independent descriptors
- The moments within a window centered at a given pixel can be interpreted as a convolution of the image with a mask
- The more moments ⇒ The better reconstructed image
- **Problem**: computational cost (up to order M for an $N \times N$ image requires $O(M^2N^2)$ adds and mults)
- **Experimental results**:

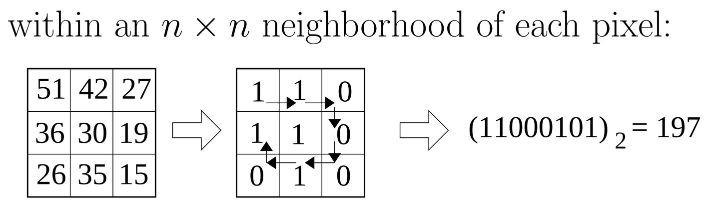
All moments of an order	Execution times on a 1024x1024 image				Speed-up on GPU versus:		
	Mukundan (1995)	Hwang (2006)	Al-Rawi (2008)	Direct on GPU	Mukundan (1995)	Hwang (2006)	Al-Rawi (2008)
$A_{4,*}$ (3)	1 391.0	258.0	62.5	19.0	73.20x	13.57x	3.28x
$A_{8,*}$ (5)	3 820.5	859.0	54.5	36.6	104.38x	23.47x	1.48x
$A_{12,*}$ (7)	7 703	1 969.0	62.5	50.5	152.53x	38.99x	1.23x
$A_{16,*}$ (9)	13 187.5	3 836.0	78.0	68.2	193.36x	56.24x	1.14x
$A_{20,*}$ (11)	20 109.5	6 586.0	93.5	90.0	223.43x	73.17x	1.03x
$A_{24,*}$ (13)	28 719	10 617.0	117.5	111.5	257.56x	95.21x	1.05x

- More potential optimizations to be implemented

6

Feature Extraction: LBP Operator

- LBP: functional and easy-to-implement texture feature
- Widely used in facial expression recognition, content based image retrieval,...
- Defined within an $n \times n$ neighborhood of each pixel:



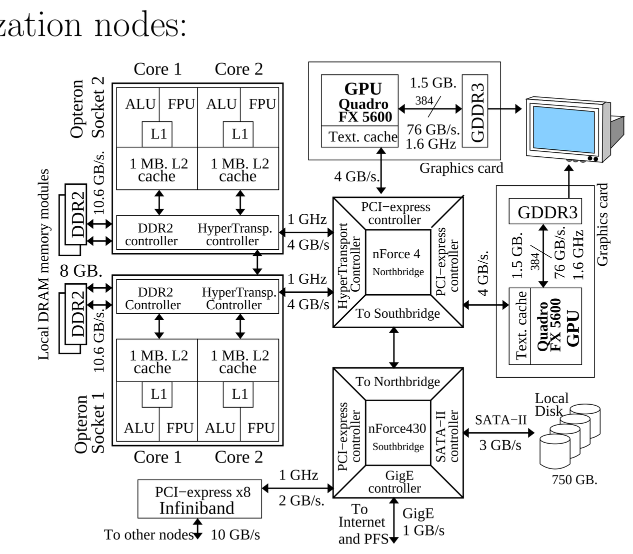
- The LBP feature is invariant to rotation and local or global intensity variations
- Some results (including Cg-CUDA comparison):

Image size	CPU C++	GPU (Cg)	GPU (CUDA)	GPU/CPU speed up
128x128	3.95	1.01	0.072	54.86x
256x256	17.83	1.09	0.140	127.35x
512x512	76.70	1.92	0.415	184.81x
1024x1024	310.65	6.88	1.564	198.62x
2048x2048	1234.96	23.91	6.114	201.98x

7

CPU/GPU Cluster Implementation

- Tested on a GPU/CPU cluster (BALE cluster, Ohio Supercomputer Center)
- **16** visualization nodes:



- Using *Datacutter* middleware for the parallelization
- Attained **very good scalability** results

8

Conclusions and Further Work

- We have developed a set of image processing routines oriented to the biomedical image analysis
- Attained performance results on GPU depends on the nature of the operation:

	Color conversion	**LBP** feature	**Zernike** moments	**Co-occurrence** matrices
Input	Pixel	3x3 window	Image tile	Var. size window
Output	Pixel	Single value	Set of values	Var. size matrix
Color channels	Three	One	One	Three
Computat. range	Per-pixel	Per-pixel	Per-tile	Per-pixel
Computat. weight	Very light	Light	Strong	Heavy
Operator type	Streaming	Streaming	Recursive	Recurrence
Data reuse	None	Little	Heavy	Strong
Locality access	None	Little	Strong	Heavy
Arithm. intensity	Heavy	Average	Strong	Low
ALU or memory intensive	Arithmetic	Arithm. and m.a.	Arithmetic	Memory access
Memory access	Low	Average	Strong	Heavy
GPU speed up	**25-250x**	**50-200x**	**1-2x**	**0.7-1x**

- Most of the computations are performed on the GPU

Further work

- Advanced architectures: Tesla, SLI-based multiGPU systems...
- Further optimizations of co-occ. matrices and Zernike moments
- Evaluate the impact of double precision support on modern GPUs

9