Improving the performance of the Morphon framework using CUDA

GPGPU Computing
2010-06-01

Daniel Forsberg
Department of Biomedical Engineering
Linköping University, Sweden

SECTRA CMIV LiU expanding reality

Agenda

- Image registration
- The Morphon step by step
- Results & discussion
- Lessons learned
- Future work
- Q & A

Questions

- Mention two advantages for using phase-based methods in image registration.
- Mention one difference between devices with compute capability 1.0/1.1 and 1.2/1.3.
- Apart from using shared memory, what other memory types can be used to increase the performance of frequently used data?

Image registration

- Deform a source image to fit onto a target image, i.e. locate a deformation field so that the following equation holds:

$$I_T(\mathbf{x}) = I_S(\mathbf{x} + \mathbf{d}(\mathbf{x}))$$

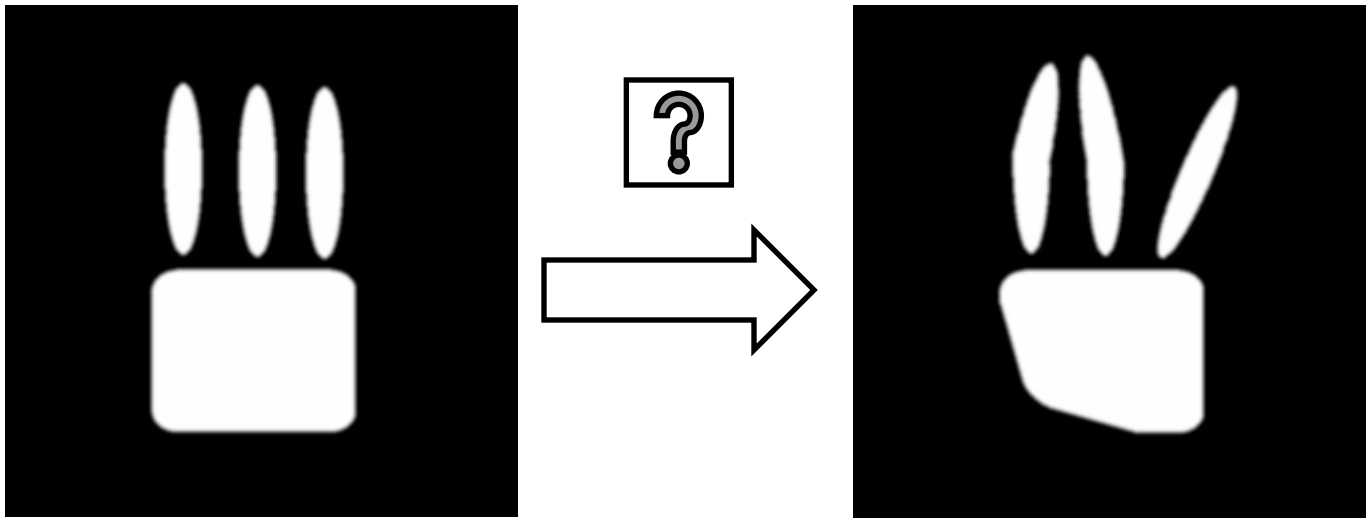


Image registration

- Basic image registration algorithm:
  - Locate a number of control points in both of the images.
  - Estimate the displacement for each set of control points.
  - Interpolate a global deformation field d(x) based upon the estimated displacement
  - Use the deformation field to deform the source image.
- Different algorithms are based on different similarity measures and different regularizers.

Image registration
Phase 1D

- In 1D the analytical signal can be used to estimate the phase of a signal.
- The phase describes how edge/line like the local neighborhood of the signal is.
- The phase possess several different useful characteristics:
  - Magnitude independent
  - Continuous
  - Stable
- The phase-difference between two signals can be used to measure disparity.

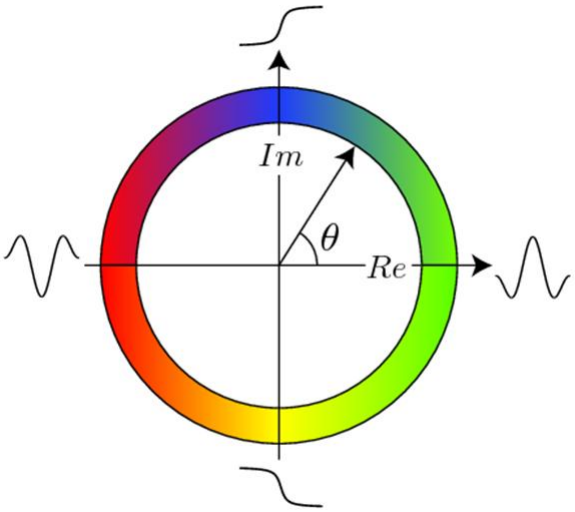


Image registration
Phase nD

- The analytic signal is well defined in 1D.
- A generalization to nD is achieved by using quadrature filters, i.e. bandpass filters in one direction and only for positive frequencies (line/edge filter in the spatial domain).

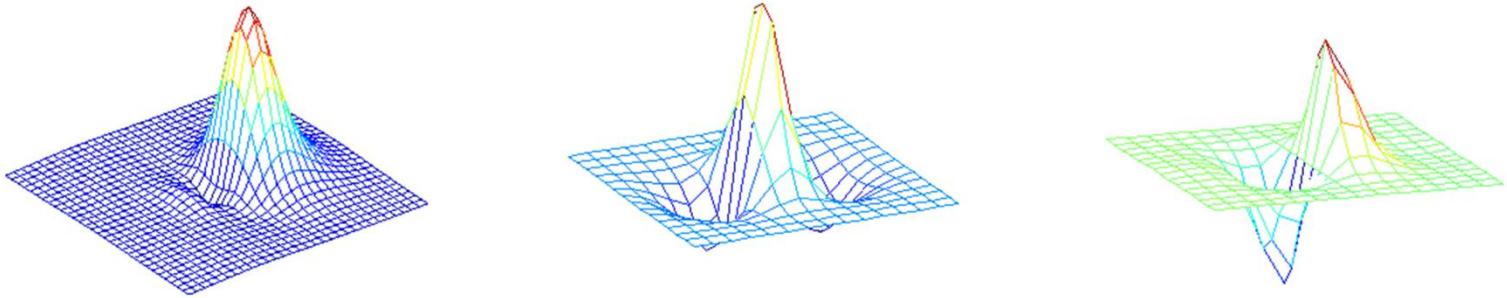


Image registration
The Morphon

- Basic algorithm for each iteration:
  - Estimate an incremental displacement field (based upon quadrature phase-difference).
  - Update the accumulated deformation field.
  - Deform the source image and use as input in the next iteration.
- Iterate the algorithm over multiple scales (coarse to fine).

The Morphon step by step
Background

- Today the Morphon is implemented using Matlab.
- A faster implementation is needed to:
  - Allow testing on larger data sets.
  - Evaluate the feasibility for usage in a clinical setting.
- The Matlab implementation will be used for benchmarking.
- Tested on two different systems where the some kernels had to be split due to insufficient number of registers.
  - Intel Xeon Quad 2.67 GHz, 8GB, Fedora 12 x64, **GTX 285 2GB**
  - Intel Core 2 Duo T9600 2.8 GHz, 4 GB, Windows 7 x64, **Quadro FX 770M 512 MB**

The Morphon step by step

Subsampling

$$
\begin{aligned}
I_S(x,y) &= I_S(2x,2y) * g_{subsampling} \\
I_T(x,y) &= I_T(2x,2y) * g_{subsampling}
\end{aligned}
$$

- Issues:
  - Large number of texture cache misses
- Kernels:
  - DownSampleFactor2 (either one or two calls)

The Morphon step by step

Estimate dk, ck and T

$$
\begin{aligned}
k &= 1 \ldots N \\
q_{S_k} &= I_S * f_k \\
q_{T_k} &= I_T * f_k \\
Q_k &= q_{S_k} q_{T_k}^* \\
d_k &= \arg\left(Q_k\right) \\
c_k &= \left|Q_k\right|^{1/2} \cos^2\left(\frac{\arg(Q_k)}{2}\right) \\
\mathbf{T} &= \sum_k |q_{S_k}| \mathbf{M_k}
\end{aligned}
$$

The Morphon step by step

Estimate dk, ck and T cont.

- Issues:
  - Use functions with higher precision if needed instead of + and *.

    “Addition and multiplication are IEEE-compliant, so have a maximum error of 0.5 ulp. However, on the device, the compiler often combines them into a single multiply-add instruction (FMAD) and for devices of compute capability 1.x, FMAD truncates the intermediate result of the multiplication. This combination can be avoided by using the __fadd_rn() and __fmul_rn() intrinsic functions.“
- Kernels:
  - EstimateQa, EstimateQb, EstimateDkCkSingleScale and EstimateTSingleScale or EstimateDkCkAndT

The Morphon step by step

Normalize and average T

$$\mathbf{T}_{\mathbf{cert}} = \sqrt{t_{11}^2 + 2t_{12}^2 + t_{22}^2}$$

$$\mathbf{T} = \frac{\mathbf{T}_{\mathbf{cert}}\,\mathbf{T} * g}{\mathbf{T}_{\mathbf{cert}} * g}$$

$$\mathbf{T} = \frac{\mathbf{T}}{\sqrt{t_{11}^2 + 2t_{12}^2 + t_{22}^2}}$$

- Kernels:
  - SumAndSquare, MultiplyElements, Convolve2DSeparableRowsSame, Convolve2DSeparableColsSame, DivideElements, NormalizeT

The Morphon step by step

Estimate A and b

$$\min_{\mathbf{d_i}} \sum_{k=1}^{N} \left[c_k \mathbf{T} \left(d_k \hat{\mathbf{n}}_\mathbf{k} - \mathbf{d_i}\right)\right]^2 \Leftrightarrow A\mathbf{d_i} = \mathbf{b}$$

$$\begin{aligned}
a_{11} &= \sum_{k=1}^{N} c_k t_{11} \\
a_{12} &= \sum_{k=1}^{N} c_k t_{12} \\
a_{21} &= a_{21} \\
a_{22} &= \sum_{k=1}^{N} c_k t_{22} \\
b_1 &= \sum_{k=1}^{N} c_k d_k (\hat{\mathbf{n}}_{\mathbf{1_k}} t_{11} + \hat{\mathbf{n}}_{\mathbf{2_k}} t_{12}) \\
b_2 &= \sum_{k=1}^{N} c_k d_k (\hat{\mathbf{n}}_{\mathbf{1_k}} t_{12} + \hat{\mathbf{n}}_{\mathbf{2_k}} t_{22})
\end{aligned}$$

The Morphon step by step

Estimate A and b cont.

- Implementation:
  - Used constant memory for filter direction to avoid unnecessary reads from global memory.
- Kernels:
  - EstimateA and EstimateB or EstimateAAndB

The Morphon step by step

Regularize A and b

$$
\begin{aligned}
a_{11} &= a_{11} * g \\
a_{12} &= a_{12} * g \\
a_{21} &= a_{21} * g \\
a_{22} &= a_{22} * g \\
b_1 &= b_1 * g \\
b_2 &= b_2 * g
\end{aligned}
$$

- Kernels:
  - Convolve2DSeparableRowsSame, Convolve2DSeparableColsSame

The Morphon step by step

Estimate di and ci

$$A\mathbf{d_i} = \mathbf{b} \quad \Leftrightarrow \quad \mathbf{d_i} = A^{-1}\mathbf{b}$$

$$c_i = \left| \left(a_{11} + a_{22} - \sqrt{(a_{11} - a_{22})^2 + 4a_{12}^2} \right) /2 \right|$$

$$d_{1_i} = \frac{a_{22}b_1 - a_{12}b_2}{a_{11}a_{22} - 2a_{12}}$$

$$d_{2_i} = \frac{-a_{21}b_1 + a_{11}b_2}{a_{11}a_{22} - 2a_{12}}$$

- Kernels:
  - EstimateDiAndCi

The Morphon step by step

Scale compensation

$$
\begin{aligned}
c_i &= 2^{scale} c_i \\
\mathbf{d_i} &= scaleFactor(scale)\mathbf{d_i}
\end{aligned}
$$

- Kernels:
  - StepSizeCompensation, MultiplyElements

The Morphon step by step

Upsampling

$$
\begin{aligned}
c_i &= \text{interp}(c_i, scale) \\
\mathbf{d_i} &= \text{interp}(\mathbf{d_i}, scale)
\end{aligned}
$$

- Implementation:
  - Stored data to be interpolated in textures and used built-in linear interpolation.
- Kernels:
  - Interpolation2D

The Morphon step by step

Accumulate di/ci to da/ca

$$\mathbf{d_a} = \frac{c_a \mathbf{d_a} + c_i(\mathbf{d_a} + \mathbf{d_i})}{c_a + c_i}$$

$$c_a = \frac{c_a^2 + c_i^2}{c_a + c_i}$$

- Kernels:
  - AccumulateDiAndCiToDaAndCa

The Morphon step by step

Regularize da

$$\mathbf{d_a} = \frac{(c_a \mathbf{d_a}) * g}{c_a * g}$$

- Kernels:
  - MultiplyElements, Convolve2DSeparableRowsSame, Convolve2DSeparableColsSame, DivideElements

The Morphon step by step

Morph

$$I_D(\mathbf{x}) = I_S(\mathbf{x} + \mathbf{d_a}(\mathbf{x}))$$

- Implementation:
  - Stored data to be interpolated in textures and used built-in linear interpolation.
- Issues:
  - Not possible to choose an arbitrarily located position between the texture elements, if needed make your own linear interpolation.

    “Alpha and beta are stored in 9-bit fixed point format with 8 bits of fractional value (so 1.0 is exactly represented).”
- Kernels:
  - Morph2D

The Morphon step by step

General implementation issues

- 2D convolution has been implemented with filters in shared memory and data to be filtered in textures.
- 2D separable convolution has been implemented with filters and data to be filtered in shared memory (low occupancy).

Results & discussion

- Tested implementation on 1024x1024 images with a maximum subsampling factor of 4.
- Used a constant block size of 16x16.
- Used CUDA Visual Profiler to improve the results.
- No uncoalesced memory reads/writes.
- Minimized number of divergent branches.
- No serialized warps.

Results & discussion

System 1

[ms]	GPU	% of total	CPU	% of total	CPU/GPU
Total registration time	898,2	100,0	54610,0	100,0	60,8
Subsampling	99,2	11,0	728,3	1,3	7,3
Estimate dk, ck and T	448,5	49,9	37014,0	67,8	82,5
Normalize and average T	75,7	8,4	4208,9	7,7	55,6
Estimate A and b	14,7	1,6	1841,6	3,4	125,0
Regularize A and b	86,1	9,6	1752,3	3,2	20,3
Estimate di and ci	9,7	1,1	322,7	0,6	33,1
Scale compensation	5,1	0,6	48,9	0,1	9,6
Upsampling	31,5	3,5	2284,3	4,2	72,6
Accumulate di/ci to da/ca	6,3	0,7	480,1	0,9	75,9
Regularize da	114,6	12,8	1984,1	3,6	17,3
Morph	5,6	0,6	3370,4	6,2	604,2

Results & discussion

System 2

[ms]	GPU	% of total	CPU	% of total	CPU/GPU
Total registration time	10118,0	100,0	57170,0	100,0	5,7
Subsampling	568,4	5,6	997,8	1,7	1,8
Estimate dk, ck and T	5842,1	57,7	23243,0	40,7	4,0
Normalize and average T	814,3	8,0	7979,0	14,0	9,8
Estimate A and b	261,4	2,6	4350,1	7,6	16,6
Regularize A and b	755,0	7,5	2973,2	5,2	3,9
Estimate di and ci	123,6	1,2	721,0	1,3	5,8
Scale compensation	70,3	0,7	156,8	0,3	2,2
Upsampling	409,6	4,0	2854,6	5,0	7,0
Accumulate di/ci to da/ca	62,8	0,6	1248,7	2,2	19,9
Regularize da	1115,1	11,0	3941,6	6,9	3,5
Morph	76,8	0,8	7380,4	12,9	96,1

Results & discussion

System 1 vs System 2

[ms]	GPU 1	% of total	GPU 2	% of total	CPU 2/GPU 1
Total registration time	898,2	100,0	10058,0	100,0	11,2
Subsampling	99,2	11,0	561,0	5,6	5,7
Estimate dk, ck and T	448,5	49,9	5822,6	57,9	13,0
Normalize and average T	75,7	8,4	806,3	8,0	10,7
Estimate A and b	14,7	1,6	262,7	2,6	17,8
Regularize A and b	86,1	9,6	751,9	7,5	8,7
Estimate di and ci	9,7	1,1	122,0	1,2	12,5
Scale compensation	5,1	0,6	70,7	0,7	13,9
Upsampling	31,5	3,5	404,0	4,0	12,8
Accumulate di/ci to da/ca	6,3	0,7	61,6	0,6	9,7
Regularize da	114,6	12,8	1104,7	11,0	9,6
Morph	5,6	0,6	71,9	0,7	12,9

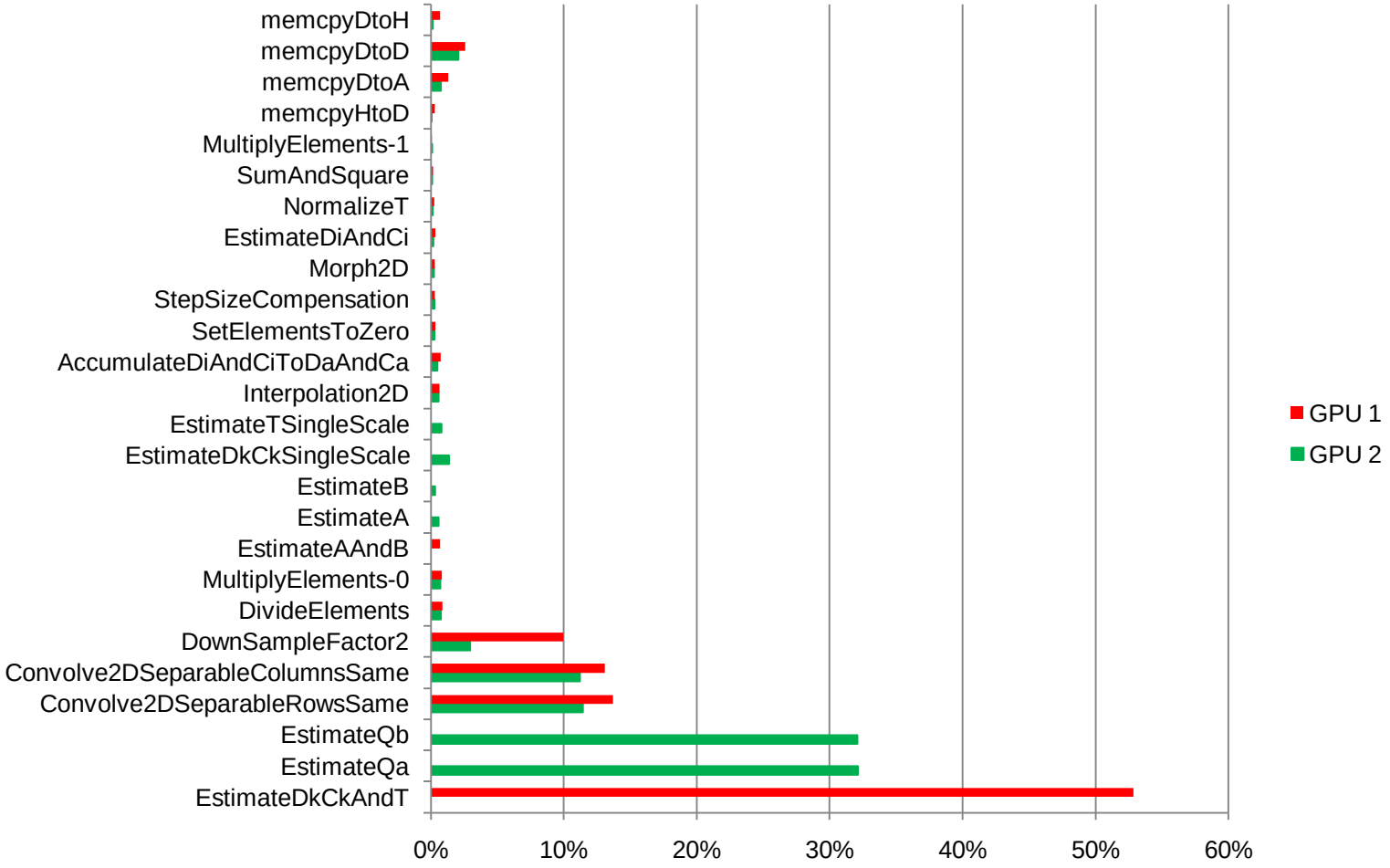
Results & discussion

Including data transfer

[ms]	GPU 1	% of total	GPU 2	% of total	CPU 2/GPU 1
Total time	1069,1	100	10277,8	100	9,6
Transfer image daga	91,4	8,6	109	1,1	1,2
Transfer quad filters	0,2	0	2,3	0	11,5
Transfer subsamp filter	0	0	0,1	0	N/A
Image registration	898,2	84	10057,7	97,9	11,2
Transfer results	51,1	4,8	73	0,7	1,4
Total performance improvement	51,1		5,1		

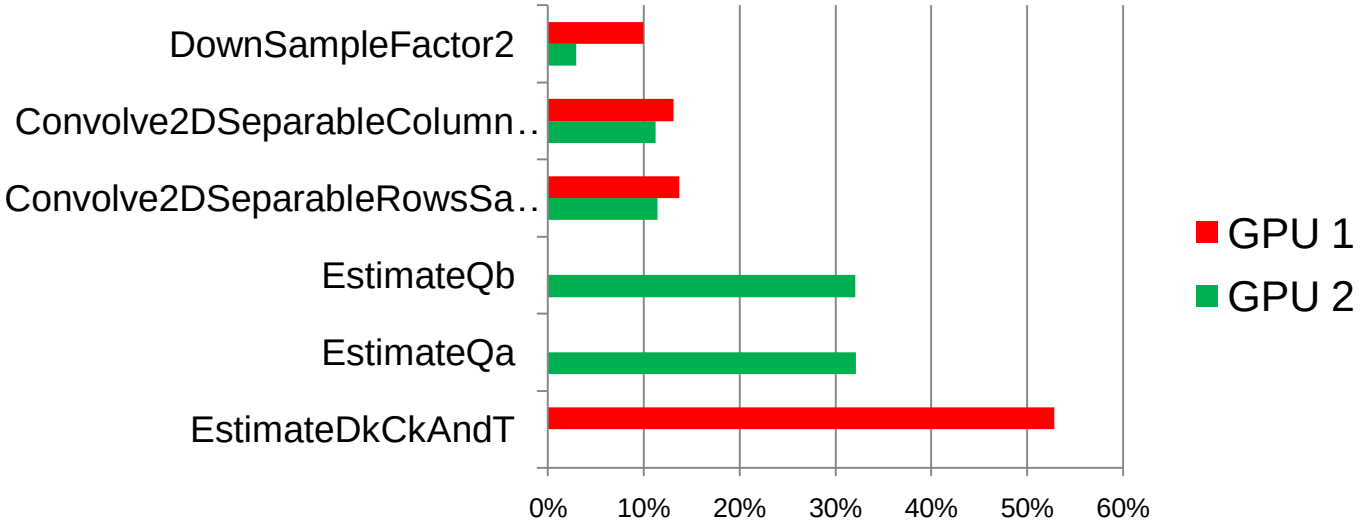
Results & discussion

Kernel summary



Results & discussion

Kernel summary



Lessons learned

- Single precision is not always single precision.
- Avoid using powf to minimize branching.
- Make us of (minimally) existent error handling.
- Be aware of the difference between dynamic and static shared memory.
- CUDA Visual Profiler is a useful tool for:
  - Detecting uncoalesced reads/writes.
  - Improving occupancy.
  - Detecting texture caches misses.
  - Detecting unnecessary divergent branches.

Future work

- Improve convolution
- Extend to 3D

Questions or comments?


Thank you for your attention!