Memory Saving Discrete Fourier Transformation on CUDA

Daniel Kauker Steffen Frey Harald Sanftmann
kaukerdl@studi.informatik.uni-stuttgart.de
steffen.frey@vis.uni-stuttgart.de
harald.sanftmann@vis.uni-stuttgart.de
Visualization and Interactive Systems Group

October 1, 2009
Motivation

- Fourier transformation is often used in image and video processing software (filtering, etc).
- Fast implementations available for graphics hardware, e.g. NVidia's CUFFT library.
- Bring this technique to users, who might not have the latest high end graphics cards
- Problem: the created CUFFT-„Plans“ consume too much memory. Large images cannot be processed, as the operating system and other applications also consume video memory. Example for 8 Megapixel image:
  32 Megabytes data + ca. 121 Megabytes plan
  but only ca. 120 Megabytes of memory available on a 256 Megabytes device. (4-byte-float, one channel)
Table of Contents

1. Fourier Transformation
2. Flexible Fourier Transformation
3. Results
Fourier Transformation

Standard two-dimensional Fourier transformations:

- Discrete Fourier Transformation: $O(N^4)$
- Fast Fourier Transformation: $O(N^2 \cdot \log(N^2))$

for a $N \times N$ image.

Use separability to calculate the Fourier transformation for 2d images in $2 \cdot N$ 1d transformations.
Existing Libraries

- **CPU**
  - OpenCV: Open source computer vision framework
  - Intel Performance Primitives: Commercial, enhance Intel CPU architecture
  - libfftw: Open source library implementing sophisticated Fourier transformation algorithms
  - And several others

- **Graphic Cards**
  - NVidias CUFFT: Fast, but very memory consuming
  - GPUTFFTW: Power-of-Two FFT Library from UNC
Previous Work on GPU

- Basic Shader Model FFT implementations
  [Kenneth Moreland and Edward Angel, 2003]
- FFT algorithms for CUDA aiming at efficiently exploiting shared memory
  [Naga K. Govindaraju et al., 2008]
- ...

In this presentation:
Use CUFFT but exploit separability of the Fourier transformation.
Flexible Fourier Transformation - Idea

- Separate Fourier transformation using one dimensional transforms
- This requires to transpose the intermediate results twice
- If the memory is not sufficient, transform the data in chunks
Flexible Fourier Transformation

Input Image

\[
\frac{M}{c} \cdot \text{cufft1d}
\]

1st dimension transformed

\[
\text{transpose}
\]

\[
\text{cufft2d}
\]

\[
\frac{N}{c} \cdot \text{cufft1d}
\]

fftshift

Fourier transformed

\[
\text{transpose}
\]

both dimensions transformed

\[c: \text{chunk size, } N: \text{image width, } M: \text{image height}\]
Flexible Fourier Transformation – Implementation

\[ F = FFT(f) \]
\[ = cufft2d(f, width, height) \]
\[ = cufft1d(cufft1d(f_x, width, height \text{ times})^T, height, width \text{ times})^T \]
\[ 1^{st} \text{ dimension transformed} \]
\[ = cufft1d(F_x, height, width \text{ times})^T \]
\[ \text{fully transformed} \]
Memory Saving one dimensional Transformations

one chunk, $c$ lines each

$c$: chunk size

Only one chunk is processed at a time. The chunk size should be as large as possible to exploit the memory on the card and avoid memory copy overhead.
Hardware

- Intel Core Quad Q9550 @ 2.83 GHz (March 2008)
- 8 GB RAM
- Graphic Cards:
  - NVidia GeForce 8600 GTS, 256MB (November 2006)
  - NVidia GeForce GTX 280, 1024MB (June 2008)
- Windows 7 RC1
## Evaluated Image Sizes

<table>
<thead>
<tr>
<th>Width</th>
<th>Height</th>
<th>Megapixel</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>6048</td>
<td>4032</td>
<td>24.38</td>
<td>Large DSLR Camera Format</td>
</tr>
<tr>
<td>4096</td>
<td>4096</td>
<td>16.78</td>
<td>Large Power of 2 Example</td>
</tr>
<tr>
<td>4752</td>
<td>3168</td>
<td>15.05</td>
<td>Standard Digital Camera Format</td>
</tr>
<tr>
<td>3456</td>
<td>2304</td>
<td>7.96</td>
<td>Standard Digital Camera Format</td>
</tr>
<tr>
<td>2560</td>
<td>1920</td>
<td>4.92</td>
<td>Photo Mobile Phone Camera</td>
</tr>
<tr>
<td>2048</td>
<td>2048</td>
<td>4.19</td>
<td>Power of 2 Example</td>
</tr>
<tr>
<td>2353</td>
<td>1568</td>
<td>3.69</td>
<td>Standard Digital Camera Format</td>
</tr>
<tr>
<td>1920</td>
<td>1080</td>
<td>2.07</td>
<td>Full HD Video Format &amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Desktop Resolution (16 : 9)</td>
</tr>
<tr>
<td>1280</td>
<td>1024</td>
<td>1.31</td>
<td>Typical Desktop Resolution (4 : 3)</td>
</tr>
<tr>
<td>720</td>
<td>480</td>
<td>0.35</td>
<td>High Definition Video Format</td>
</tr>
<tr>
<td>512</td>
<td>512</td>
<td>0.26</td>
<td>Small Power of 2 Example</td>
</tr>
</tbody>
</table>
Performance - GTS 8600 256MB

Transforming one channel

X marks power-of-2 sizes which perform best on FFT

Measuring

plan CUFFT
sync threads
start timer
for $i = 1:k$
  until all rows done
  copy to GPU
  1d transform
  copy to Host
  transpose
  until all rows done
  copy to GPU
  1d transform
  copy to Host
  transpose
sync threads
t = stop timer / k
Performance - GTS 8600 256MB

Interpretation

- CUFFT and FlexFT achieve similar performance.
- FFTW always faster, especially $\geq 5$ MP due to memory copies in FlexFT and better plans (FFTW\_PATIENT)

Transforming one channel

X marks power-of-2 sizes which perform best on FFT
Introduction

Fourier Transformation

Flexible Fourier Transformation

Results 14 / 18

Performance - GTX 280 1024MB

Interpretation

- CUFFT and FlexFT exhibit similar performance
- CUDA overtakes CPU due to heavy multi processing
- CUFFT requires too much memory around 38 Megapixels

Transforming one channel

X marks power-of-2 sizes which perform best on FFT
Memory Usage CUFFT vs. FlexDFT

- CUFFT
- 8600 GTS
- GTX 280

Image Size in Megapixel vs. Memory Used
Memory Usage CUFFT vs. FlexDFT

**GeForce 8600 GTS**
Uses as much memory as possible, maximum of 90% of free memory.
Chunkifies the input to according to fitting partition sizes.

**GeForce GTX 280**
Allocates buffer for input and output on the device.
Transforms the data in chunks then, as there is not enough memory left for a single transform.
This approach has no internal CPU ↔ GPU memory copies and thus is much faster.
Conclusions

- Use separability saves memory on device without timing loss.
- Performance equal to 2D CUFFT but much larger transformations with little memory possible.
- Allows trading memory consumption for speed.
- Extensible for higher dimensions (e.g. for geological or medical 3d scans).
Thank you for your attention!

Questions?