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. NVidias 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
  - GPUFFTW:
    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

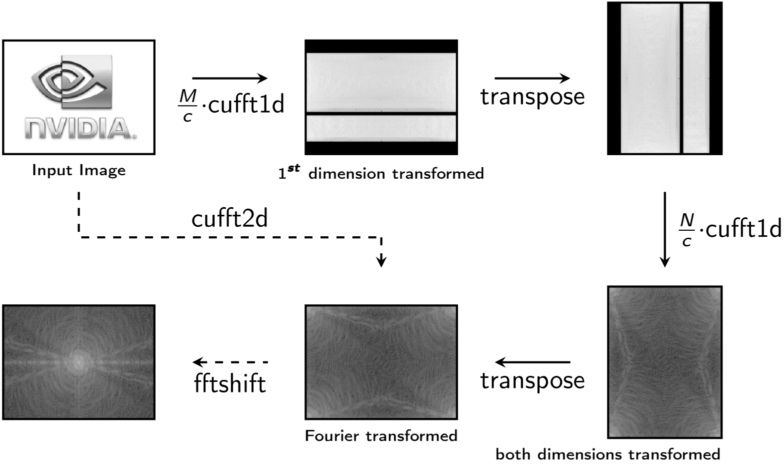
  [Naga K. Govindaraju et al., 2008]
- . . .

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

$\frac{M}{c}\cdot$cufft1d

1^{st} dimension transformed

transpose

$\frac{N}{c}\cdot$cufft1d

cufft2d

both dimensions transformed

transpose

Fourier transformed

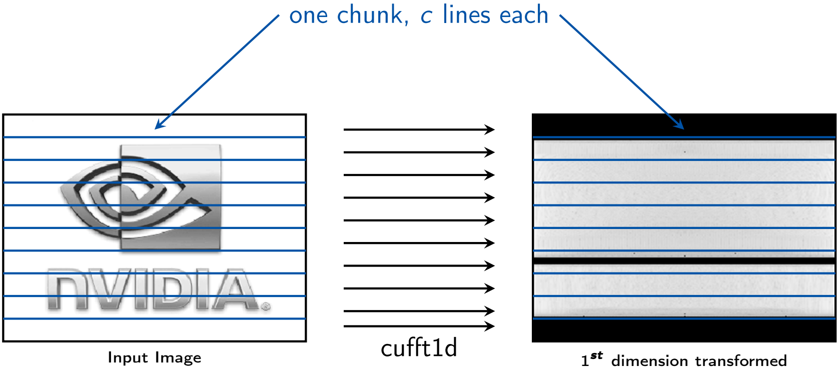
fftshift

c: chunk size, N: image width, M: image height

Flexible Fourier Transformation – Implementation

$$\begin{aligned}
F &= FFT(f) \\
&= cufft2d(f, width, height) \\
&= cufft1d(\underbrace{cufft1d(f_x, width, height\ \ times)}_{1^{st}\ \ dimension\ \ transformed}{}^T, height, width\ \ times)^T \\
&= \underbrace{cufft1d(F_x, height, width\ \ times)^T}_{fully\ \ transformed}
\end{aligned}$$

Memory Saving one dimensional Transformations



one chunk, c lines each

Input Image

cufft1d

1^{st} dimension transformed

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.

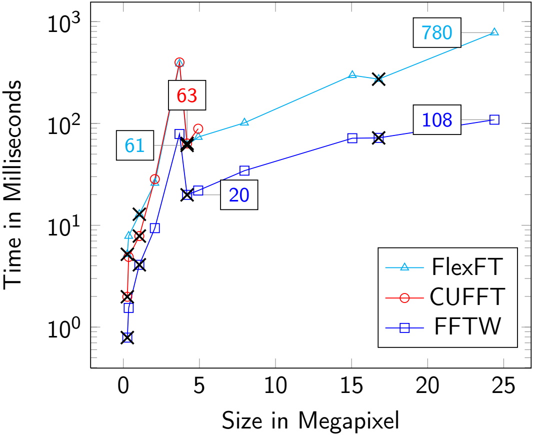
Hardware

- 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

Width	Height	Megapixel	Comment
6048	4032	24.38	Large DSLR Camera Format
4096	4096	16.78	Large Power of 2 Example
4752	3168	15.05	Standard Digital Camera Format
3456	2304	7.96	Standard Digital Camera Format
2560	1920	4.92	Photo Mobile Phone Camera
2048	2048	4.19	Power of 2 Example
2353	1568	3.69	Standard Digital Camera Format
1920	1080	2.07	Full HD Video Format & Desktop Resolution (16 : 9)
1280	1024	1.31	Typical Desktop Resolution (4 : 3)
720	480	0.35	High Definition Video Format
512	512	0.26	Small Power of 2 Example

Performance - GTS 8600 256MB



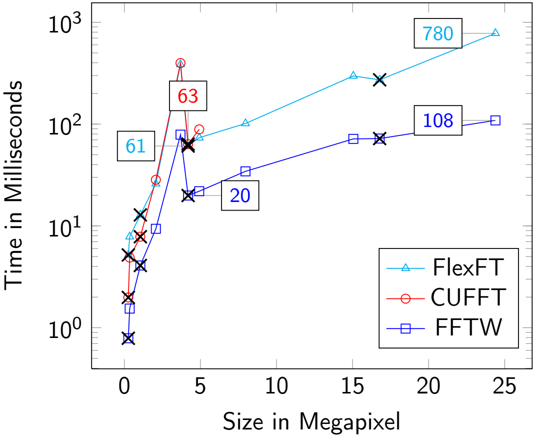
Transforming one channel

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



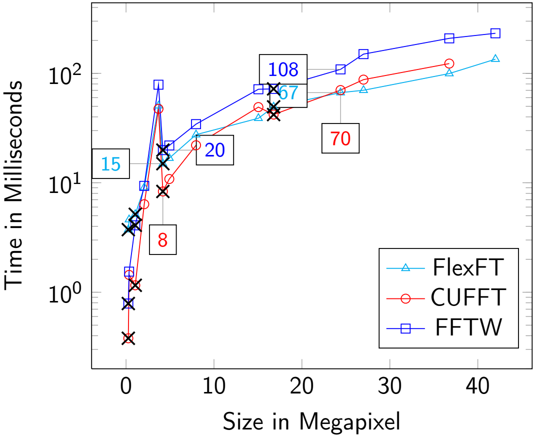
Transforming one channel

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

Interpretation

- CUFFT and FlexFT achieve similar performance
- FFTW always faster, especially ≥ 5 MP due to memory copies in FlexFT and better plans (FFTW_PATIENT)

Performance - GTX 280 1024MB



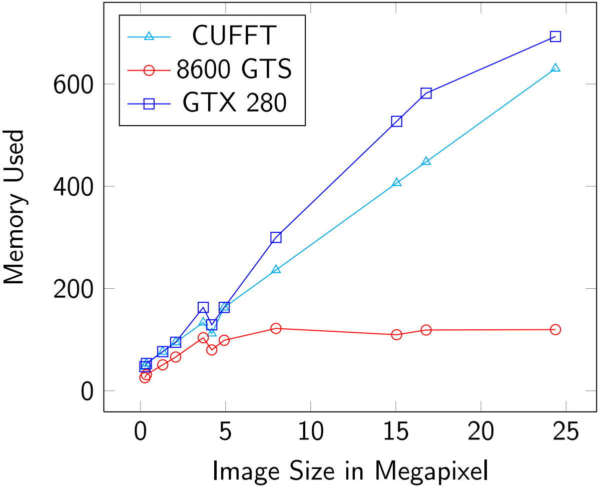
Transforming one channel

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

Interpretation

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

Memory Usage CUFFT vs. FlexDFT



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?