FAST FOURIER TRANSFORM FOR AMD GPUs

using OpenCL

Akira Nukada
Tokyo Institute of Technology, Japan
Researcher
FAST FOURIER TRANSFORM (FFT)

- FFT as well as BLAS
  - Important computations used in many HPC applications
  - Vendors (of processors, compilers, OSs) provide highly tuned library

- FFT
  - Conversion between frequency domain and space domain, time domain, etc.
  - Multimedia applications (Image, Audio, …)
  - HPC applications (Spectral method, Convolution, …)

- Performance of FFT
  - HPC applications use double-precision.
  - Memory access efficiency is also important. O(N) memory access for O(N log N) computation.
FAST FOURIER TRANSFORM USING GPU?

- **NVIDIA CUDA**
  - NukadaFFT library (Sep. 2010 release, core design in Apr. 2009)
  - Achieved much speed-up over multi-core CPUs
    - due to high memory bandwidth and many compute units of GPUs

- **How about AMD RADEON GPUs?**
  - CAL works only for RADEON GPUs
  - Now OpenCL SDK is available
  - Architecture of AMD GPU is different, but much similar to NVIDIA GPU
    - Many SIMD cores
    - Thread schedulers
    - Small cache memories
    - Small scratch pad memories
FFT is a fast algorithm to compute DFT (Discrete Fourier Transform).

\[ X'(k) = \sum_{j=0}^{N-1} X(j) e^{-2\pi i j k / N} \]

When the input size N can be factorized into M and L, N-point FFT is replaced by L x M-point FFTs, M x L-point FFT, and multiplications by twiddle factors.

\[ X'(k_0 + k_1 L) = \sum_{j_0=0}^{M-1} e^{-2\pi i j_0 k_1 / M} e^{-2\pi i j_0 k_0 / N} \sum_{j_1=0}^{L-1} X(j_0 + j_1 M) e^{-2\pi i j_1 k_0 / L} \]

- **Twiddle factors**
- **L-point FFT**
- **M-point FFT**
FAST FOURIER TRANSFORM ON GPU

Implementation with RADEON/OpenCL is similar to NVIDIA/CUDA

- Each thread computes a small FFT

- Data exchange between threads required
  - Via shared memory (CUDA)
  - Via local memory (OpenCL)

- Twiddle factor (cos&sin) table
  - On texture memory (CUDA)
  - On constant memory (OpenCL)

Example of 16-point FFT using 4 threads
Example of 256-point FFT using 64 threads

256 = 4 x 4 x 4 x 4

256-point FFT is divided into four stages of 4-point FFT
Example of 512-point FFT using 64 threads

512-point FFT is divided into 8-point FFTs.
MEMORY MAPPING FOR OPENCL

Kernel definition:

```c
__kernel void zfft256f(__global double2 *vin, __global double2 *vout, __constant double2 *wn);
```

vin: input buffer
vout: output buffer
wn: cos/sin table (4KB)

<table>
<thead>
<tr>
<th></th>
<th>vin</th>
<th>vout</th>
<th>wn</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA (NukadaFFT)</td>
<td>Global (cached)</td>
<td>Global (cached)</td>
<td>Texture (cached)</td>
</tr>
<tr>
<td>CUDA (NVIDIA CUFFT)</td>
<td>Global? (cached?)</td>
<td>Global (cached?)</td>
<td>? (cached?)</td>
</tr>
<tr>
<td>OpenCL(AMD)</td>
<td>Global</td>
<td>Global</td>
<td>Constant (cached)</td>
</tr>
</tbody>
</table>

In case of double-precision.
Performance in GFLOPS assumes $5N \log_2 N$ floating-point operations.

**GeForce and Tesla**
- Intel Core i7 CPU, X58 Express Chipset
- CUDA 4.0
- NVIDIA CUFFT library 4.0, or NukadaFFT

**RADEON**
- AMD Phenom 9500 CPU
- AMD APP SDK 2.4
- Our custom FFT code in OpenCL
PERFORMANCE OF CUDA FFTS

GFLOPS

CUFFT is faster esp. for powers of two sizes.
BOTTLE-NECK & EFFICIENCY / ratio to theoretical peak

- **DP performance**
  - Bottle-neck on GeForce

- **Memory access efficiency**
  - Double-complex (double2) data
  - Good for RADEON, Bad for GeForce

<table>
<thead>
<tr>
<th></th>
<th>AMD RADEON HD 6970</th>
<th>NVIDIA GeForce GTX 580</th>
<th>NVIDIA Tesla C2050</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak DP performance</td>
<td>675 GFLOPS</td>
<td>197GFLOPS</td>
<td>515GFLOPS</td>
</tr>
<tr>
<td>Achieved performance</td>
<td>171GFLOPS (25.3%)</td>
<td>144GFLOPS (73.1%)</td>
<td>114GFLOPS (22.1%)</td>
</tr>
<tr>
<td>Peak Memory B/W</td>
<td>176GB/s</td>
<td>192GB/s</td>
<td>144GB/s (128GB/s*9/8)</td>
</tr>
<tr>
<td>Achieved B/W</td>
<td>137GB/s (77.8%)</td>
<td>115GB/s (59.9%)</td>
<td>91GB/s (71.1%)</td>
</tr>
<tr>
<td>Bottle-neck</td>
<td>Memory</td>
<td>DP perf.</td>
<td>Memory</td>
</tr>
</tbody>
</table>
NUMBER OF FLOATING POINT OPERATIONS

- $5N \log_2 N$ is pseudo number of FP operations

- Powers-of-two FFT
  - Large number of FPADD/FPSUB ops.
  - Low ratio of FPMAD combination

- RADEON GPU architecture can execute one of the following instructions in a cycle
  - 2 FPADD/FPSUB
  - 1 FPMUL
  - 1 FPMAD

Real number of FP ops. and FP instructions

<table>
<thead>
<tr>
<th></th>
<th>256-point</th>
<th>512-point</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5N \log_2 N$</td>
<td>10,240</td>
<td>23,040</td>
</tr>
<tr>
<td>ADD/SUB ops.</td>
<td>4,672</td>
<td>11,776</td>
</tr>
<tr>
<td>MUL ops.</td>
<td>2,304</td>
<td>4,352</td>
</tr>
<tr>
<td>FPADD/FPSUB</td>
<td>3,520</td>
<td>9,984</td>
</tr>
<tr>
<td>FPMUL</td>
<td>1,152</td>
<td>2,560</td>
</tr>
<tr>
<td>FPMAD</td>
<td>1,152</td>
<td>1,792</td>
</tr>
<tr>
<td>Min. FP cycle (AMD)</td>
<td>4,064</td>
<td>9,344</td>
</tr>
<tr>
<td>Min. FP cycle (NVIDIA)</td>
<td>5,824</td>
<td>14,336</td>
</tr>
</tbody>
</table>
### REAL FP EFFICIENCY | ratio to theoretical peak

- **Real cycles for DP operations**
  - [for AMD] \((\text{FPADD} + \text{FPSUB}) / 2 + \text{FPMUL} + \text{FPMAD}\)
  - [for NVIDIA] \(\text{FPADD} + \text{FPSUB} + \text{FPMUL} + \text{FPMAD}\)

<table>
<thead>
<tr>
<th></th>
<th>AMD RADEON HD 6970</th>
<th>NVIDIA GeForce GTX 580</th>
<th>NVIDIA Tesla C2050</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Peak DP performance</strong></td>
<td>675 GFLOPS</td>
<td>197GFLOPS</td>
<td>515GFLOPS</td>
</tr>
<tr>
<td><strong>FP Usage</strong></td>
<td>20.1%</td>
<td>83.1%</td>
<td>25.2%</td>
</tr>
<tr>
<td><strong>Peak Memory B/W</strong></td>
<td>176GB/s</td>
<td>192GB/s</td>
<td>144GB/s (128GB/s*9/8)</td>
</tr>
<tr>
<td><strong>Achieved B/W</strong></td>
<td>137GB/s (77.8%)</td>
<td>115GB/s (59.9%)</td>
<td>91GB/s (71.1%)</td>
</tr>
<tr>
<td><strong>Bottle-neck</strong></td>
<td>Memory</td>
<td>DP perf.</td>
<td>Memory</td>
</tr>
</tbody>
</table>
MEMORY BANDWIDTH | 256-point FFTs

- **Simple Copy**
  - Same memory access pattern
  - No FP operations

- **FFT without barriers achieves 97.3%**
  - Same memory access pattern
  - Full FP operations
  - No barriers, only works on AMD GPUs

Work-items per group = 64 (wavefront size)

- **FFT with barriers**
  - Only for comparison

---

**Achieved Bandwidth on RADEON HD 6970**

- **Simple Copy**
- **FFT without barriers**
- **FFT with barriers**

![Bar chart showing achieved bandwidth](image-url)
PORTABILITY

- OpenCL code works on both AMD GPUs and NVIDIA GPUs

  - To enable double precision
    ```
    #ifdef cl_khr_fp64 // For NVIDIA GPUs and others.
    #pragma OPENCL EXTENSION cl_khr_fp64 : enable
    #endif
    
    #ifdef cl_amd_fp64 // For AMD GPUs
    #pragma OPENCL EXTENSION cl_amd_fp64 : enable
    #endif
    ```

  - Synchronization to ensure the data exchange via local memory
    ```
    (AMD) mem_fence(CLK_LOCAL_MEM_FENCE)
    (NVIDIA) barrier(CLK_LOCAL_MEM_FENCE)
    ```
PERFORMANCE PORTABILITY

- OpenCL does not guarantee any performance portability
  - Our OpenCL code achieves only 80GFLOPS on GeForce GTX 580
    - About 50% of CUFFT library

- Everything is unknown in terms of performance
  - Can we achieve same performance using OpenCL?
    - Can we use all CUDA resources in OpenCL?
    - Can we use all CAL resources in OpenCL?
    - Same optimization level available?
SUMMARY

- OpenCL makes it easy to write programs for AMD GPUs.
  - Written in OpenCL, however optimized for AMD GPUs
  - FFT code achieved competitive performance to NVIDIA GPUs.

- Why was the optimization easy?
  - Experience of NVIDIA CUDA
    - Helpful for startup OpenCL due to similar parallel programming model
    - May be confused in hardware differences.
  - Experience of each application code
    - Reduces time to write and tune the code.
  - Little knowledge about AMD GPUs
    - Double complex (128bit) : Best memory access granularity
    - Double precision: No need to consider VLIW.
CONCLUSIONS AND FUTURE WORK

- Achieved high performance using OpenCL on AMD GPUs
  - OpenCL is READY as primary programming language for RADEONs.
  - As a unified programming language for many-core processors?

- Better than NVIDIA? I don’t think so.
  - At least competitive for some cases
  - It depends on the application, and parameters….

- Future work: An FFT library with auto-tuning mechanism in OpenCL
  - NukadaFFT library: in CUDA, auto-tuning for many NVIDIA GPUs, for many transform sizes.
  - In OpenCL, auto-tuning for several AMD GPUs, for many transform sizes: Maybe this is easy.
  - In OpenCL, auto-tuning for many GPUs (and CPUs?), for many transform sizes: This is difficult.
Disclaimer & Attribution
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

The contents of this presentation were provided by individual(s) and/or company listed on the title page. The information and opinions presented in this presentation may not represent AMD’s positions, strategies or opinions. Unless explicitly stated, AMD is not responsible for the content herein and no endorsements are implied.