Fast Transforms using the Cell/B.E. Processor

David A. Bader
joint work with Seunghwa Kang and Virat Agarwal
Sony-Toshiba-IBM Center of Competence for the Cell/B.E. at Georgia Tech

- **Mission**: grow the community of Cell Broadband Engine users and developers
  - Fall 2006: Georgia Tech wins competition for hosting the STI Center
  - First publicly-available IBM QS20 Cluster
  - 200 attendees at 2007 STI Workshop
  - Multicore curriculum and training
  - Demonstrated performance on
    - Multimedia and gaming
    - Scientific computing
    - Medical applications
    - Financial services

David A. Bader, Director

http://sti.cc.gatech.edu

David A. Bader
Applications

• **CellBuzz**: Freely-available, open source libraries optimized for the Cell/B.E.
  http://sourceforge.net/projects/cellbuzz/
  - **ZLIB & GZIP**: data compression
  - **FFT**: fast Fourier transform
  - **RC5**: encryption
  - **MPEG-2**: video encoding and decoding
  - **JPEG2000**: digital content processing

• **Financial Modeling**
Cell/B.E. Libraries: FFT and JPEG2000

- **FFTC: Fastest Fourier Transform on the Cell/B.E.**
  - 1-Dimensional single precision DIF-FFT optimized for 1K-16K complex input samples
  - Parallelize & optimize computation of a single FFT computation
  - Design high performance synchronization barrier using inter-SPE communication
  - Demonstrated superior performance of 18.6 GFlop/s for 8K complex input samples.

- **JPEG2000 on the Cell/B.E.**
  - Optimize coding/decoding by data decomposition / data alignment / vectorization
  - Demonstrated average speedup of 3.1 over Intel 3.2 GHz Pentium-4

The source code is freely available from our CellBuzz project in SourceForge [http://sourceforge.net/projects/cellbuzz/](http://sourceforge.net/projects/cellbuzz/)
Cell/B.E. Libraries: ZLIB and MPEG-2

• ZLIB Data compression & decompression library
  – Vectorize compute intensive kernels and parallelize to run on multiple SPEs
  – Extend the gzip header format while maintaining compatibility with legacy gzip decompressors
  – Demonstrated speedup of 2.9 over high-end Intel Pentium-4 system

• MPEG-2 Video Decoding
  – First parallelization of a multimedia application on Cell/B.E.
  – Demonstrated a speedup of 3 over Intel 3.2GHz Xeon.

The source code is freely available from our CellBuzz project in SourceForge
http://sourceforge.net/projects/cellbuzz/

David A. Bader
"Retired Marine Lt. Gen. Bernard Trainor said the issue of aging aircraft is a constant complaint of all branches of service."

Atlanta Journal Constitution
April 27, 2002

• Fault Diagnosis
  – Estimate the crack length without disassembly based on vibration data collected from multiple sensors.

• Failure Prognosis
  – Estimate the expected time before crash
System-of-Systems Decomposition

- Actuator Leakage and Wear
- Power and Cooling Turbo Machine Life, Oil Condition, Oil Servicing and Filter Condition
- Generator Oil Level, Hydraulic Filters, Pump, and Hydraulic Fluid Level
- Engine
- Battery
- Oxygen Generator
- Nitrogen Generator and Filter
- Landing Gear and Arresting Hook Structure fatigue life
- Landing Gear Strut Pressure and Fluid Level
- Rotary Actuator Wear
Overview of the Diagnosis and Prognosis Process

Involves multiple computationally expensive modules!!!

David A. Bader
Fast Transforms on the Cell/B.E.

- Fast Fourier Transform
- Discrete Wavelet Transform
FFTCC: Fastest Fourier Transform for Cell/B.E.

- Focus on medium size FFT computations
  - Complex single-precision 1-Dimensional FFT
- Input samples and output results reside in main memory.
- Radix 2, 3 and 5.
- Optimized for 1K-16K input samples.
- Focus on achieving high performance for the computation of a single FFT, rather than increasing throughput.
Existing FFT Research on Cell/B.E.

• [Williams et al., 2006], analyzed peak performance.

• [Cico, Cooper and Greene, 2006] estimated 22.1 GFlops/s for an 8K complex 1D FFT that resides in the Local Store of one SPE.
  - 8 independent FFTs in local store of 8 SPEs gives 176.8 GFlops/s.

• [Chow, Fossum and Brokenshire, 2005] achieved 46.8 GFlops/s for 16M complex FFT.
  - Highly specialized for this particular input size.

• FFTW is a highly portable FFT library of various types, precision and input size.
Our FFTC is based on Cooley Tukey

- **Input** is one dimensional vector of complex values.
- Algorithm is **iterative**, no recursion.
- **Out of Place** approach is used.
- Requires two arrays A&B for computation, one input and one output that are swapped at every stage.
- Out of place approach **prevents data reordering** after the last stage.
- Algorithm requires \( \log N \) stages. Each stage requires \( O(N) \) computation.
  - Complexity \( O(N \log N) \)
Algorithm 1: Sequential FFT algorithm

Input : array A[0] of size N

1. \( NP \leftarrow 1 \);
2. \( \text{problemSize} \leftarrow N \);
3. \( \text{dist} \leftarrow 1 \);
4. \( i1 \leftarrow 0 \);
5. \( i2 \leftarrow 1 \);
6. while \( \text{problemSize} > 1 \) do
7.     Begin Stage;
8.     \( a \leftarrow i1 \);
9.     \( b \leftarrow i2 \);
10.    \( k = 0, j\text{twiddle} = 0 \);
11. for \( j \leftarrow 0 \) to \( N - 1 \) step \( 2 \times NP \) do
12.     \( W \leftarrow w[j\text{twiddle}] \);
13.     for \( j\text{first} \leftarrow 0 \) to \( NP \) do
14.         \( b[j + j\text{first}] \leftarrow a[k + j\text{first}] + a[k + j\text{first} + N/2] \);
15.         \( b[j + j\text{first} + \text{dist}] \leftarrow -(a[k + j\text{first}] - a[k + j\text{first} + N/2]) \times W \);
16.      \( k \leftarrow k + NP \);
17.      \( j\text{twiddle} \leftarrow j\text{twiddle} + NP \);
18. swap(i1, i2);
19. \( NP \leftarrow NP \times 2 \);
20. \( \text{problemSize} \leftarrow \text{problemSize}/2 \);
21. \( \text{dist} \leftarrow \text{dist} \times 2 \);
22.     End Stage;

Output : array A[i1] of size N
Illustration of the Algorithm

- Illustration of the algorithm for n=16 complex values.

- Distance between pairs of output values double at every subsequent stage.

- Shows how output of one stage serves as the input to another.
FFT design on Cell/B.E.: Challenges

- Synchronize step after every stage leads to significant overhead.
- Reduce synchronization stages.
- Design efficient barrier synchronization routine.
- We will later describe an efficient tree-based synchronization algorithm based on inter-SPE communication.

```
Algorithm 1: Sequential FFT algorithm
Input : array A[0] of size N

1. NP ← 1;
2. problemSize ← N;
3. dist ← 1;
4. i1 ← 0;
5. i2 ← 1;
6. while problemSize > 1 do
7.   Begin Stage;
8.   a ← i1;
9.   b ← i2;
10.  k ← 0, jtwiddle ← 0;
11.  for j ← 0 to N − 1 step 2 * NP do
12.     W ← w[jtwiddle];
13.     for jfirst ← 0 to NP do
14.       b[j + jfirst ] ← a[k + jfirst ] + a[k + jfirst + N/2];
15.       b[j + jfirst + Dist ] ← (a[k + jfirst ] − a[k + jfirst + N/2]) * W;
16.     k ← k + NP;
17.     jtwiddle ← jtwiddle + NP;
18. swap(i1, i2);
19. NP ← NP * 2;
20. problemSize ← problemSize / 2;
21. dist ← dist * 2;
22. End Stage;

Output : array A[i1] of size N
```
FFTC design on Cell/B.E.: Challenges (contd.)

Load balancing to achieve better SPU utilization

- No SPE should wait at the synchronization barrier.
- Strategy should be scalable across multiple chips (large number of SPEs).

Vectorization difficult for every stage
- Stages 1 & 2, do not have regular data access pattern.
- Optimizing the first 2 stages become important for medium size inputs, as it may constitute 20-25% of the total running time.

First 2 stages.
- Require efficient parallelization technique to allocate data to SPEs.
- Require data reorganization to fully utilize the SPE computational power.
FFTC design on Cell/B.E.: Challenges (cont’d)

- **Limited local store**
  - require space for N/2 twiddle factors and input data.
  - loop unrolling and duplication increases size of the code.
  - Effectively manage code and data within 256KB.

- **Algorithm is branchy:**
  - Doubly nested for loop within the outer while loop
  - Lack of branch predictor compromises performance.
  - Provide branch hints and restructure the algorithm to eliminate branch.

---

David Bader

```c
Algorithm 1: Sequential FFT algorithm
Input : array A[0] of size N
1. NP <- 1;
2. problemSize <- N;
3. dist <= 1;
4. i1 <- 0;
5. i2 <- 0;
6. while problemSize > 1 do
   Begin Stage;
   a <- i1;
   b <- i2;
   k = 0; jtwiddle = 0;
   for i = 0 to N - 1 step 2 * NP do
      W <- w[jtwiddle];
      for jfirst <- 0 to NP do
         b[j + jfirst] = a[k + jfirst] + a[k + jfirst + N/2];
         b[j + jfirst + Dist] = a[k + jfirst] - a[k + jfirst + N/2] * W;
      k <- k + NP;
      jtwiddle = jtwiddle + NP;
   End Stage;
7. swap(i1, i2);
8. NP <- NP * 2;
9. problemSize = problemSize / 2;
10. dist = dist * 2;
Output : array A[i1] of size N
```
Paralleling FFTC on the Cell/B.E.

- Input size N (complex samples)
- Divide the input array in 2*p chunks where \( p \): number of available SPEs.
- PPE allocates chunk \( i \) and \( i+p \) SPE \( i \), spawns threads and waits for completion.
  - Data allocation technique is same at every stage.
  - Efficient technique as it prevents intervention from PPE during the computation.
- Achieves load balancing, each SPE receives equal amount of work.
Optimization for SPE

- The input data at every stage is fetched using DMA in a multi-buffered way.
  - The block size is limited by a global parameter `buffer_size`.

- **While loop duplication for Stages 1 & 2**
  - For vectorization of these stages we need to use spe shuffle intrinsic on the output vector.
  - The figure above gives the shuffle pattern for Stage 1, and the figure below for Stage 2.

- **Loop duplication increases code size** in the already limited local store.
Optimization for SPE (cont’d)

- Duplicate while loop when loop counter is < buffersize and otherwise.
  - Need to stall for DMA get at different places within the inner for loop.
  - The second case allows for efficient loop unrolling in the inner-most for loop.

- While loop duplication for these 2 cases further increase code size, that limits the size of FFT that can be computed using this methodology.

```c
while problemSize > 1 do
   Begin Stage;
   Initiate all DMA transfers to get data;
   Initialize variables;
   for k <- 0 to chunksize do
      for jfirst <- 0 to min(NP, chunksize - k) step buffersize do
         Stall for DMA buffer;
         for i <- 0 to buffersize do
            SIMDize computation as buffersize > 4;
            Initiate DMA put for the computed results;
            Update j, k, jtwiddle;
      swap(fetaddr, putaddr);
      NP <- NP * 2;
      problemSize <- problemSize/2;
      dist <- dist * 2;
   End Stage;
   Synchronize using Inter SPE communication;
```

David Bader
Experimental Setup

- Manual Loop unrolling, multi-buffering, inter SPE communication, odd-even pipelining, vectorization.

- Instruction level profiling and performance analysis using Cell SDK 3.0, used xlc compiler at level 3 optimization.

- FLOP analysis
  - Operation Count : \((5*N \log N)\) floating point operations
  - For 2 complex value computations we require
    - One complex subtraction (2 FLOP), One complex addition (2 FLOP) and one complex multiplication (6 FLOP).
Performance analysis: Scaling across SPEs

- Near linear scaling from 1 to 8 SPEs. Thus it should scale well across multiple chips as well.
- Speedup increases with larger input size.
Performance Comparison of FFTs

* Performance numbers from BenchFFT.

David Bader
FFTCC Summary

- Use various techniques such as Manual Loop unrolling, multi-buffering, inter SPE communication, odd-even pipelining, vectorization to achieve performance on a SPE.
- Loop duplication increases the code size but helps in further optimizations on an SPE.
- We demonstrate superior performance of 18.6 GigaFlop/s for an FFT of size 8k-16K, and believe we have the fastest FFT implementation on the Cell/B.E.
- Code available at http://sourceforge.net/projects/cellbuzz/
Discrete Wavelet Transform on Cell/B.E.

- We design an efficient data decomposition scheme to achieve high performance with affordable programming complexity.
- We introduce multiple Cell/B.E. and DWT specific optimization issues and solutions.
- Our implementation achieves 34 and 56 times speedup over one PPE performance, and 4.7 and 3.7 times speedup over the cutting edge multicore processor (AMD Barcelona), for lossless and lossy DWT, respectively.
Discrete Wavelet Transform (in JPEG2000)

- Decompose an image in both vertical and horizontal direction to the sub-bands representing the coarse and detail part while preserving space information.
Discrete Wavelet Transform (in JPEG2000)

- Vertical filtering followed by horizontal filtering
- Highly parallel but bandwidth intensive
- Distinct memory access pattern becomes a problem
- Adopt Jasper [Adams2005] as a baseline code
Previous work

- Column grouping [Chaver2002] to enhance cache behavior in vertical filtering
- Muta et al. [Muta2007] optimized convolution based (require up to 2 times more operations than lifting based approach) DWT for Cell/B.E.
  - High single SPE performance
  - Does not scale above 1 SPE
Data Decomposition Scheme

Cache line aligned

A multiple of the cache line size

2-D array width

Row padding

2-D array height

A unit of data transfer and computation

A unit of data distribution to the processing elements

A multiple of the cache line size

Remainder

Distributed to the SPEs

Processed by the PPE

David Bader
Data Decomposition Scheme

- Satisfies the alignment and size requirements for efficient DMA data transfer and vectorization.
- Fixed LS space requirements regardless of an input image size.
- Constant loop count.
Vectorization – Real number representation

• Jasper adopts fixed point representation
  – Replace floating point arithmetic with fixed point arithmetic
  – Not a good choice for Cell/B.E.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Latency (SPE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpyh</td>
<td>7 cycles</td>
</tr>
<tr>
<td>mpyu</td>
<td>7 cycles</td>
</tr>
<tr>
<td>a</td>
<td>2 cycles</td>
</tr>
<tr>
<td>fm</td>
<td>6 cycles</td>
</tr>
</tbody>
</table>

David Bader
Loop Interleaving

- In a naïve approach, a single vertical filtering involves 3 or 6 times data transfer.
- Bandwidth becomes a bottleneck.
- Interleave splitting, lifting, and optional scaling steps do not fit into the LS.
Loop Interleaving

- First interleave multiple lifting steps
- Then, merge splitting step with the interleaved lifting step

- Use temporary main memory buffer for the upper half
Fine-grain Data Transfer Control

- Initially, we copy data from the buffer after the interleaved loop is finished
- Yet, we can start it just after low\textsubscript{2} and high\textsubscript{2} are read
- Cell/B.E.’s software controlled DMA data transfer enables this

\begin{align*}
\text{low0} & \rightarrow \text{low1} & \rightarrow \text{low2} & \rightarrow \text{low3} & \rightarrow \\
\text{high0} & \rightarrow \text{low1} & \rightarrow \text{high1} & \rightarrow \text{low2} & \rightarrow \\
\text{low1} & \rightarrow \text{low2} & \rightarrow \text{low3} & \rightarrow \text{high0} & \\
\text{high1} & \rightarrow \text{low2} & \rightarrow \text{high0} & \rightarrow \text{high1} & \\
\text{low2} & \rightarrow \text{high1} & \rightarrow \text{high0} & \rightarrow \text{high1} & \\
\text{high2} & \rightarrow \text{high1} & \rightarrow \text{high0} & \rightarrow \text{high1} & \\
\text{low3} & \rightarrow \text{high2} & \rightarrow \text{high3} & \\
\text{high3} & \rightarrow \text{high3} & & & \\
\end{align*}
Performance Evaluation

- 3800 x 2600 color image, 5 resolution levels
- Execution time and scalability up to 2 Cell/B.E. chips (IBM QS20)

David Bader
Performance Evaluation
Comparison with x86 Architecture

- One 3.2 GHz Cell/B.E. chip (IBM QS20)
- One 2.0 GHz AMD Barcelona chip (AMD Quad-core Opteron 8350)

<table>
<thead>
<tr>
<th>Parallelization</th>
<th>OpenMP based parallelization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectorization</td>
<td>Auto-vectorization with compiler directives</td>
</tr>
<tr>
<td>Real Number Representation</td>
<td>Identical to the Cell/B.E. case</td>
</tr>
<tr>
<td>Loop Interleaving</td>
<td>Identical to the Cell/B.E. case</td>
</tr>
<tr>
<td>Run-time profile feedback</td>
<td>Compile with run-time profile feedback</td>
</tr>
</tbody>
</table>

* Optimization for the Barcelona

David Bader
DWT Summary

- Cell/B.E. has a great potential to speed-up parallel transforms but requires careful implementation.
- We design an efficient data decomposition scheme to achieve high performance with affordable programming complexity.
- Our implementation demonstrates 34 and 56 times speedup over one PPE, and 4.7 and 3.7 times speedup over the AMD Barcelona processor with one Cell/B.E. chip.
- Cell/B.E. can also be used as an accelerator in combination with the traditional microprocessor.
IBM QS22

- Delivers 204.8 Gflop/s (double-precision) peak performance with FMA (fused-multiply-and-add) in comparison with 29.2 Gflop/s in QS20 or QS21
- Supports 4 to 32 GB (DDR2) main memory in comparison with 1 GB (XDR) in QS20 or QS21
R Optimizations for the Cell

- Optimize R statistics package for the Cell/B.E. processor
  - BLAS
  - LAPACK
  - random number generator, and
  - variance/covariance/correlation
- IBM & Georgia Tech collaboration; freely-available, open source (GPL) code will be released on SourceForge, based on R-2.7.0
- Demonstration of native double-precision performance using the IBM QS22 Blade with dual Power XCell 8i processors
R Performance on the QS22: Covariance w/ Pearson’s method

- Covariance computation with Pearson’s method (without cache blocking)
- QS20: was compute bounded
- QS22: is now bandwidth bounded

1024 items
* 8192 samples/item

16 18
10 12
8 10
6 8
4 6
2 4
0

QS22
QS20

Gflop/s

David Bader
R Performance on the QS22: Covariance w/ Kendall’s method

• Covariance computation with Kendall’s method
• Compute bounded in both systems
• Invokes sign() function in the loop body
  – Needed for correctness checking

128 items
* 4096 samples/item

![Graph showing Gflop/s for QS22 and QS20]
R Performance on the QS22: Covariance, further optimizations

- Test kernel created by removing sign() function call in Kendall’s method
  - Does not affect correctness
- Compute bounded in both systems

128 items

4096 samples/item
Financial Services using R on Cell

- CreditMetrics (R extension package) on the QS22 with optimized BLAS, LAPACK, and RNG libraries
- Additional optimizations (e.g. modifying script for smaller memory footprint, not specific to the Cell/B.E.)
Acknowledgment of Support

[Logos of various organizations]
Cell/B.E. Apps: Financial Modeling

- **Objective:** Demonstrate a competitive edge of the Cell/B.E. for Financial Services.

- **European Option Pricing.** Black - Scholes equation:
  \[ dS(t) = \mu S(t) dt + \sigma S(t) dW(t) \]

- **Collateralized Debt Obligation (CDO) pricing**
  - Gaussian Copula, Monte Carlo simulation

- **Optimize various**
  - **random number generators:** Mersenne Twister, Hammersley sequence, LCG.
  - **normalization techniques:** Box Mueller Polar/Cartesian, Low Distortion Map.

David A. Bader
Cell/B.E. Apps: Financial Modeling

Performance Analysis: Random Number Generation

Performance comparison of RNG (Mersene Twister) on various architectures

Over 3 Billion random numbers per second from a single Cell/B.E.

## Cell/B.E. Apps: Financial Modeling

### Performance Analysis: European Option pricing

<table>
<thead>
<tr>
<th>Version</th>
<th>Platform</th>
<th>Performance</th>
<th>Transformation</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOP-BMP (our result)</td>
<td>Cell/B.E.</td>
<td>1040M/s</td>
<td>Box Mueller /Polar form</td>
<td>Cell SDK 2.1</td>
</tr>
<tr>
<td>IBM SDK Sample [2]</td>
<td>Cell/B.E.</td>
<td>190M/s</td>
<td>Box Mueller /Polar form</td>
<td>Cell SDK 2.1</td>
</tr>
<tr>
<td>RMCell-BMP</td>
<td>Cell/B.E.</td>
<td>605M/s</td>
<td>Box Mueller /Polar form</td>
<td>RapidMind SDK 2.1</td>
</tr>
<tr>
<td>EOP-BMC (our result)</td>
<td>Cell/B.E.</td>
<td>1824M/s</td>
<td>Box Mueller /Cartesian form</td>
<td>Cell SDK 2.1</td>
</tr>
<tr>
<td>CUDA-BMC [1]</td>
<td>NVIDIA G80(GPU)</td>
<td>1209M/s</td>
<td>Box Mueller /Cartesian form</td>
<td>CUDA SDK 1.0</td>
</tr>
</tbody>
</table>


**1.5x over optimized CUDA implementation for NVIDIA G80.**

**2x over optimized implementation for RapidMind on Cell.**

*Double precision will be essential*