A Couple Music Applications

Nils Peters and Eric Battenberg
Parallel Computing Laboratory (Par Lab)
The Center for New Music and Audio Technologies (CNMAT)
University of California, Berkeley
The 12 motifs

<table>
<thead>
<tr>
<th></th>
<th>Embed</th>
<th>SPEC</th>
<th>DB</th>
<th>Games</th>
<th>ML</th>
<th>CAD</th>
<th>HPC</th>
<th>Health</th>
<th>Image</th>
<th>Speech</th>
<th>Music</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Finite State Mach.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Circuits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Graph Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Structured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Dense Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Sparse Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Spectral (FFT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Dynamic Prog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Particle Methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Backtrack/ B&amp;B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Graphical Models</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Unstructured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Range of Apps
Hundreds of apps and plug-ins

Music Information Retrieval / Music Recommendation

New Musical Instruments / Controller / Music Creation

3D Loudspeakers / Microphone Arrays
Voices, Streams, Channels, Tracks, and Lines
Music’s Low Hanging Fruit Ripe for Parallelism

Usually a mix of task parallelism and data parallelism
Two music applications

1. Music Information Retrieval (MIR)
   Drum track extraction using Open MP and CUDA

2. Real-time audio processing
   Partitioned convolution using Pthreads
A few MIR tasks

- **Transcription** – Automatically generate a score or tablature from audio
- **Source separation** – Isolate certain instruments (including vocals)
- **Song similarity, playlist creation, music recommendation**
- **Artist, genre, mood classification**
  - Help organize a music archive
- **Score Following, lyrics sync, beat tracking**
  - Useful for DJs, karaoke, music education, and automated accompaniment.
- **Song Segmentation**
  - Partition song into discrete passages (verse, chorus, bridge) The hope is that someday you will be able to query for music like this:
Drum Track Extraction

Audio examples (listen for drums in original)

Input audio → Spectrogram → NMF → Component Feature Extraction

Spectral Feature Extraction → Percussive components → SVM Classifier

Audio Resynthesis → SVM Classifier

Percussive features

Time/frequency components

Drum track

Friday, August 17, 2012
Drum Track Extraction

- An example of source separation where the drum track is isolated.
  - Useful in drum transcription, beat tracking, and rhythm analysis.
- Audio spectrogram is factorized into components using Non-negative Matrix Factorization (NMF).
- Components are classified using a Support Vector Machine (SVM).
- “Percussive” components are used to synthesize an audio drum track.

**NMF step is most computationally intensive.**
- 80% of time in Matlab (18.5 sec of 23.1 sec total for 20 sec of audio)
- **We will parallelize NMF using OpenMP (for multi-core) and CUDA (for GPUs)**
Drum Track Extraction

- NMF source separation is accomplished using an iterative gradient-based solver
  - Our aim is to factorize a positive matrix $X$ (MxN) into 2 positive matrices $H$ (MxK) and $W$ (KxN), where $K$ is much smaller than $M$ and $N$

  - $M$: number of frequency components,
  - $N$: number of time frames
  - $K$: number of sources

Friday, August 17, 2012
Drum Track Extraction

- For $[512 \times 30 \times 3445]$ NMF,
  - 512 frequency components, 30 sources, 3445 time frames (~20 sec)
- For each iteration we have:
  - 423 Mflops of SGEMMs (Single-precision General Matrix Multiply)
  - 3.6 Mflops of element-divides (slow)
  - 0.1 Mflops element-multiplies
  - 0.1 Mflops sums (requires communication)

Also:
- Add a small constant to divisor matrices to prevent divide-by-zero. (Add EPS, 3.6 Mflops)
- Compute log-based cost function every 25 iterations to check for convergence.

$$D(X \| WX) = \sum_{ij} \left( X_{ij} \log \frac{X_{ij}}{(WH)_{ij}} - X_{ij} + (WH)_{ij} \right)$$
NMF in OpenMP

- Data-parallel for loop
  - To be used for element-wise arithmetic
  - Create team of nt threads to do independent chunks of work

```c
#pragma omp parallel for num_threads(nt)
for(i=0;i<N;i++)
c[i] = a[i]*b[i];
```

- Reduction
  - For sums
  - Create team of nt threads to compute partial sums
  - Then add the partial sums to final variable s

```c
s = 0;
#pragma omp parallel num_threads(nt)
#pragma omp for reduction(+:s)
for(i=0;i<N;i++)
s += a[i];
```
NMF in OpenMP

- We use MKL for SGEMMs
- Use OpenMP for other routines
- Performance scaling on dual-socket Core i7 920:
  - SGEMMs show most significant speedup
    - Highest work to communication ratio
    - Non-linear speedup suggests this won’t scale well to more cores using this architecture and programming model.
  - However,
    - >7x speedup compared to Matlab
    - >4x speedup compared to sequential C
NMF in CUDA

- NMF Implementation in CUDA
  - SGEMMs – use CUBLAS 2.1, achieves 60% of peak (373 GFLOPS on GTX 280)
    - Padding matrices to multiples of 32 reduces SGEMM running time by 26%
  - Element-wise arithmetic – similar to example code
  - Reductions (sums) – a lot harder in CUDA than OpenMP
    - Use optimizations covered in CUDA SDK for shared memory reduction.
      - Reorganize binary tree traversal.
      - Loop unrolling, multiple reads per thread.
      - Run the 30 sums concurrently. An important optimization.

57x speedup overall
CUDA vs. OpenMP

- CUDA achieves much higher performance on current GPUs for highly data-parallel computations. (>30x speedup compared to Matlab, 4x faster than OpenMP+Nehalem)

- OpenMP can achieve multi-core speedup on data-parallel computations with very little programmer effort.

- If inter-thread communication is required, things become much more difficult.
  - OpenMP gets harder.
  - CUDA gets a lot harder.

- For music application developers, CUDA is only feasible for computational kernels that require very high performance.
Real-Time Audio Processing

- Very different requirements compared to offline batch MIR.
- Low-latency processing requires sub-10ms latency.
- If processing deadlines are missed, we hear a click in the audio stream
  - This must be avoided!
- Aim is to minimize the worst-case execution time of processing task.

User interfaces with pressure-sensitive multi-touch gestural interfaces
Convolution Reverb

\[ x(t) \ast h(t) = y(t) \]
Non-Uniform Partitioned Convolution

- Very important technique for:
  - Efficient, low-latency, real-time convolution reverb and FIR filtering.
- Shorter partitions $\rightarrow$ lower latency
- Longer partitions $\rightarrow$ higher computational efficiency
Scheduling Partitioned Convolution

- The processing of larger partitions takes longer but occurs less frequently.
- How do we schedule these long-running tasks?
(Better) Scheduling Options for Long-Running Tasks (LRTs)

3. **Preempt** LRTs with higher-priority tasks when necessary.

4. **Time-distribute** LRTs by breaking into sub-tasks.
Scheduling Performance Results

- Preemption vs. Time-Distributed (Single core)
- Preemptive version:
  - Better performance
  - Easier to program

Friday, August 17, 2012
Summary

• Music apps are valuable use cases for exploiting parallelism
• Data-parallelism and task-parallelism

• Two examples for parallel computing in music applications
• 1. Music Information Retrieval (MIR)
• 2. Real-Time Audio Processing

• Different ways of implementations based on requirements
A Couple Music Applications

Nils Peters and Eric Battenberg
Parallel Computing Laboratory (Par Lab)
The Center for New Music and Audio Technologies (CNMAT)
University of California, Berkeley

{nils, ericb}@eecs.berkeley.edu

4th Annual Short Course on Parallel Programming, August 15–17, 2012
References

• NMF on Open MP and CUDA:

• Real–Time Partitioned Convolution: