Experiences Accelerating MATLAB Systems Biology Applications

Lukasz Szafaryn, Kevin Skadron, and Jeffrey J. Saucerman
University of Virginia
Outline

• MATLAB
• Optimizations to MATLAB
• GPU Acceleration with CUDA
• Applications
  (Heart Wall Tracking and Myocyte Simulation)
  – Problem
  – Algorithm
  – Optimization and performance
  – Lessons
• Conclusions
• Future Research
MATLAB

• Convenient but inefficient programming language of choice for scientists
  - Interpreted language
  - Most of the existing code and libraries are single-threaded
• MATLAB Parallel Toolbox - understanding of parallel programming
• Jacket and GPUMat - large parallelism to justify overhead
MATLAB contd.

- Interpreted language optimized by JIT compiler
  - 2x slower than C
- MATLAB Embedded Compiler has limited support
  - 1.2-1.4x slower than C
- MEX Interface to link C code
  - translating to C - many functions written from scratch
  - no support for convenient OpenMP standard, need to use thread libraries
Acceleration

1. Translation:
   - convert MATLAB to C

2. Parallelization:
   - C for multi-core CPU
   - CUDA for GPU

Experimental Setup

- CPU: 3.2 GHz quad-core Intel Core 2 Extreme
- GPU: NVIDIA GeForce GTX 280 (PCIe 2.0)
- MS Windows, MS C Compiler
Acceleration with GPU (CUDA)

C Program
- Allocate GPU memory
- Transfer inputs
- Launch kernel

CPU
- Return to CPU
- Transfer results
- Free GPU memory

GPU
- CUDA Kernel
Heart Wall Tracking

Application

• Speed and shape of contractions provides important information about body’s response to stimulus

• Measured by tracking inner and outer heart walls through multiple frames
Heart Wall Tracking Algorithm

- Processing 20 inner and 30 outer heart wall points, total 50 points (TLP)
- Processing of each point - sequence of operations on the surrounding area and template (DLP)
Heart Wall Tracking
Performance

- Times reported for processing of 300 frames (10s of ultrasound recording)
Heart Wall Tracking
Lessons

- Typical MATLAB code written by a scientist has room for optimization – 1.3x
- Conversion to C requires significant coding effort
- Selective offloading results in multiple CPU-GPU data transfer overheads
- Iterative codes require merging kernels and reusing variables to avoid overhead
- CUDA libraries cannot be used as a part of GPU code
- Good performance - significant changes to the structure of code, difficult for a scientist to understand
Myocyte Simulation
Application

- Models single cardiac myocyte and its electrical activity - determined to be a key aspect in the development of heart failure
- Modeled by 91 Ordinary Differential Equations (ODEs) and 250 supporting equations
Myocyte Simulation Algorithm

• Sequential nature of ODE solving does not allow processing of time steps in parallel
• Speed-up from parallelizing model evaluation is limited by Amdahl's law
• Mainly fine-grained TLP, no DLP, limited coarse-grained TLP by grouping equations
Myocyte Simulation Performance

- Time reported for 10,000-point simulation (10s of simulated time)
Myocyte Simulation
Lessons

- Typical MATLAB code written by a scientist has room for optimization – 2.0x
- Conversion of the model to C was straightforward
- More speedup possible by accelerating entire solver, not just the model evaluation
- GPU can still provide best acceleration if its overhead is eliminated (heterogeneous chip), but...
- Significant speedup is anticipated by offloading application to FPGA (well suited to fine-grained irregular parallelism)
Conclusions

- Limited availability of C libraries necessitates time consuming coding
- Many systems biology applications (even those with limited parallelism) benefit from GPU
- GPU overheads are significant (should be eliminated in new CPU-GPU architectures)
- Real-time processing feasible in near future
- Ultimately, acceleration of applications should be automated!
Future Research

• Automatic acceleration with the use of compiler
  - via use of architecture-specific libraries
  - via compiling for target architecture

• Merging of workloads
  - based on resource needs
  - based on dependency

• Acceleration with alternative architectures
  - well suited for fine-grained parallelism
  - esp. FPGA
Acknowledgements

• Funding provided by:
  – NSF grant IIS-0612049
  – SRC grant 1607.001

• Equipment donated by NVIDIA
Software

Source code will be soon available at:
http://lava.cs.virginia.edu/wiki/rodinia
Questions
Backup
Memory Transfer Overhead

Transfer Time (milliseconds)

Megabytes per Transfer

CPU to GPU
GPU to CPU
Memory Allocation Overhead

![Graph showing Memory Allocation Overhead](image)

- **malloc (CPU memory)**
- **cudaMalloc (GPU memory)**

---

**Legend:**
- Blue diamonds: malloc (CPU memory)
- Green triangles: cudaMalloc (GPU memory)

---

**Axes:**
- Y-axis: Time Per Call (microseconds)
- X-axis: Megabytes Allocated Per Call

---

**Graph Details:**
- The graph compares the time per call for memory allocation between CPU (malloc) and GPU (cudaMalloc).
- The x-axis represents the megabytes allocated per call, ranging from 1E-07 to 1000.
- The y-axis represents the time per call in microseconds, ranging from 0.0001 to 10000.
- The graph highlights the differences in allocation overhead between CPU and GPU memory allocations.
NVIDIA GPU Architecture
# NVIDIA GPU Specification

<table>
<thead>
<tr>
<th>Model</th>
<th>Year</th>
<th>Code name</th>
<th>Fab (nm)</th>
<th>Transistors (Billion)</th>
<th>Die Size (mm²)</th>
<th>Bus interface</th>
<th>Memory min (MB)</th>
<th>Config core</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeForce GTS 250</td>
<td>March 3, 2009</td>
<td>G92a/b</td>
<td>65/55</td>
<td>0.754</td>
<td>324/230</td>
<td>PCie x16 2.0</td>
<td>512 or 1024</td>
<td>123.64:10:10</td>
</tr>
<tr>
<td>GeForce GTX 260</td>
<td>June 26, 2008</td>
<td>G200</td>
<td>65</td>
<td>1.4</td>
<td>576</td>
<td>PCie x16 2.0</td>
<td>896</td>
<td>192:64:23:23</td>
</tr>
</tbody>
</table>

**Reference clock rate:** Core (MHz), Shader (MHz), Memory (MHz)

**Fillrate:** Pixel (Gflops), Texture (Gflops), Bandwidth (GiB/s), DRAM type

**Reference Memory Configuration**

**Graphics library support (version):** DirectX, OpenGL

**GFLOPS² (MAADD+MUL):**

**TDP (Watts):**
CUDA Programming Model
Myocyte Simulation

Molecule

Cell

Organ

ECG