Domain specific translation:
reducing the semantic gap between the user and the GPU hardware

Scott B. Baden
Dept. of Computer Science and Engineering
University of California, San Diego
Technological disruption

• Transformational
  ‣ New science, benefits to society

• Challenges
  ‣ New wisdom for delivering a solution
  ‣ Software development costs

Cray-1, 1976, 240 Megaflops
Connection Machine CM-2, 1987
Beowulf cluster, late 1990s
Nvidia Tesla, 4.14 Tflops, 2009

ASCII Red, 1997, 1Tflop
Sony Playstation 3, 150 Gflops, 2006
Intel 48 core processor, 2009
Tilera 100 core processor, 2009
Latest disruption: the NVIDIA GPU family

- Specialized many many core processor
- SIMT execution: piecewise SIMD on long vectors
- Massive virtual multithreading, fine grained
- Exposed memory hierarchy
- Hybrid designs coming

Scott B. Baden / Domain specific translation - IWLS 2011
A prescription for high performance

- Put frequently accessed data on-chip to avoid costly global memory accesses
- Spawn enough threads to hide latency without using too much fast memory
- Induce memory accesses to coalesce
- Treat irregularity carefully to embrace SIMT execution
Domain specific, source-source translation

• Hide idiosyncratic device behavior from the user
  ‣ Broaden the user community
  ‣ Focus on domain science
  ‣ Enables user to work within a conventional programming model

• Cardiac Electrophysiology (Europar ’10)
  ‣ Python, model-level description
  ‣ Fred Lionetti (MS, 2010)

• Mint: Stencil methods (ICS ’11)
  ‣ Annotated C
  ‣ Didem Unat (PhD, 2012)
Motivating application: heart modeling

Cell Models

<table>
<thead>
<tr>
<th>Model</th>
<th>Inception</th>
<th>Vars</th>
</tr>
</thead>
<tbody>
<tr>
<td>FitzHugh Nagumo</td>
<td>1961</td>
<td>2</td>
</tr>
<tr>
<td>Beeler-Reuter</td>
<td>1977</td>
<td>8</td>
</tr>
<tr>
<td>Puglisi-Bers</td>
<td>2001</td>
<td>18</td>
</tr>
<tr>
<td>Flaim</td>
<td>2006</td>
<td>87</td>
</tr>
<tr>
<td>Grandi-Bers</td>
<td>2009</td>
<td>42</td>
</tr>
</tbody>
</table>

Whole cell ionic model (ODEs)

3-D mono- or bidomain finite element model (PDEs)

Scott B. Baden / Domain specific translation - IWLS 2011
Source-to-source optimization

- Produce optimized GPU kernels from high-level description
  - No knowledge of CUDA required
  - Abstract Syntax Tree (AST)
- Retargetable approach
  - OpenMP
  - ATI Firestream
  - OpenCL
Software and hardware test beds

- Left ventricular electrophysiology simulation
  - Continuity-6 framework  [www.continuity.ucsd.edu](http://www.continuity.ucsd.edu)
  - Andrew McCulloch (UCSD)
  - Cell model: Flaim (87 state variables)
  - 5280 finite elements, 42240 collocation points
  - Model input: 500 lines
  - Optimized CUDA code: 2500 lines

- Hardware test bed
  - Intel Quad Core i7 CPU 940 @ 2.93GHz 12GB RAM
  - Integrated nVidia GTX-295: 1.242 GHz, 1.3 capable

- CUDA occupancy: 4 blocks × 64 threads
Optimizations

• Remove redundant expression evaluations
  assembly ops: 19,747 → 6,685

• Keep frequently accessed data in fast mem.
  ‣ 87 state vars (global, 87kb) + temporaries
  ‣ LCOPY: copy global state variables to thread locals, compiler manages registers
  ‣ SCACHE
    ‣ Software managed cache (offline)
    ‣ Belady’s MIN, evict “furthest in future”
  ‣ SPLIT: kernel partitioning, reduces reg. spills

• Both GPUs
Full simulation results

• One full heartbeat (300 ms)
  ‣ SCACHE + SPLIT + 2GPUs
  ‣ 11 word software cache (64 → ∞)

• CPU (quad core i7): 16665 sec (4.6 hours)
  ‣ ODEs (OpenMP): 16015 seconds (96%)
  ‣ PDEs (SuperLU w/ MPI): 650 seconds (3.9%)

• Nvidia GTX-295: 759 seconds (12.6 min)
  ‣ Overall: ×21 faster, ODE: ×134
  ‣ ODEs: 119 seconds (16%)
  ‣ Same PDE solver: 650 seconds (84%)
Discussion

• How did domain specific knowledge help?
  ‣ No significant branching in the numerical inner loops
  ‣ Use an optimal off-line caching algorithm

• Performance baseline on the multicore is in need of improvement

• We couldn’t apply kernel splitting naively
Related Work

• Caching
  ‣ NVIDIA [’09]
  ‣ Silberstein et al. [’08] – on-line software cache
  ‣ Eichenberger et al. [’06] – CELL software cache

• Translation
  • Eichenberger et al. [’06] – function partitioning
  • Lee et al. [’09] – OpenMP to CUDA
  • Markall et al. [’10] – optimized GPU code from high level specification

• Domain specific modeling languages
  ‣ FE solvers [FEniCS Form compiler, Kirby&Logg ’06]
  ‣ FreeFEM [www.freefem.org]
Domain specific, source-source translation

- Cardiac Electrophysiology (Europar ’10)
  - Python, model level description
- Mint translator for stencil methods (ICS ‘11)
  - Annotated C
  - Doctoral Thesis topic of Didem Unat
Mint

• Translates annotated C source to optimized CUDA C
• Domain specific: 3D stencil methods
  ‣ An important class of scientific applications
  ‣ Optimizations benefit from the domain-specific knowledge

for (i,j,k) in 1:N x 1:N x 1:N
u'[i,j,k] = (u[i-1,j,k] + u[i+1,j,k] +
           u[i,j-1,k] + u[i,j+1,k] +
           u[i,j,k+1] + u[i,j,k-1]) / 6
- Tesla C1060: Mint achieved 79% of hand-optimized CUDA
- OpenMP ran on Intel Nehalem with 4 threads
- Volkov and Demmel’s hand optimized CUDA [SC’08]
Mint Execution Model

- Fork-Join – like OpenMP
- Parallel regions run on the device
- Thread teams process for loops
- Move data between host and device memories
Mint Program for the 3D Heat Eqn

1. `#pragma mint copy(U, toDevice, (n+2), (m+2), (k+2))`
2. `#pragma mint copy(Unew, toDevice, (n+2), (m+2), (k+2))`
3. 
4. `#pragma mint parallel default(shared)`
5. {
6.   int t=0;
7.   while( t++ < T ){
8.     `#pragma mint for nest(all) tile(16,16,64) chunksize(1,1,64)`
9.     for (int z=1; z<= k; z++)
10.    for (int y=1; y<= m; y++)
11.     for (int x=1; x<= n; x++)
12.        Unew[z][y][x] = c0 * U[z][y][x] +
13.            c1 * (U[z][y][x-1] + U[z][y][x+1] +
14.                U[z][y-1][x] + U[z][y+1][x] +
15.                U[z-1][y][x] + U[z+1][y][x]);
16.    `#pragma mint single`
17.     double*** tmp;
18.     tmp = U; U = Unew; Unew = tmp;
19. } // end of single
20. }
21. } // end of while
22. } // end of parallel region
23. 
24. `#pragma mint copy(U, fromDevice, (n+2), (m+2), (k+2))`
#pragma mint for clauses

- **mint for**
  - Marks the succeeding nested loops for acceleration
  - Transformed into a multi-dim CUDA kernel
  - Manages data decomposition and thread work-assignment.
- **nest(#,all)**
  - Indicates the depth of for-loop parallelization.
  - Supports multi-dim thread geometries.
- **tile(t_x,t_y,t_z)**
  - Divides the iteration space into tiles
  - Data points computed by a CUDA thread block
- **chunksize(c_x,c_y,c_z)**
  - Determines the workload of a thread
  - Similar to OpenMP schedule clause
CUDA thread blocks

- Split the grid into 3D tiles based on `tile` clause
- Together with the `chunksize` clause, determines mapping of tile elements to CUDA thread blocks

```cpp
#pragma mint for nest(#,all) tile(t_x,t_y,t_z) chunksize(c_x,c_y,c_z)
```

![Diagram showing grid splitting and thread mapping]
Mint Translator

- Fully automated translation and optimization system: C to CUDA

- Translator is built on top of the ROSE compiler framework
  - ROSE provides an API for generating and manipulating ASTs

- Mint performs transformations on the AST

- The Mint translator generates both host and device code
```c
__global__ void mint_1_1517(
    cudaMemcpyPtr ptr_dU ...)
{
    double* U = (double *) (ptr_dU.ptr);
    int widthU = ptr_dU.pitch / sizeof(double);
    int sliceU = ptr_dU.ysize * widthU;
    ...

    int _idx = threadIdx.x + 1;
    int _gidx = _idx + blockDim.x * blockIdx.x;
    ...

    if (_gidz >= 1 && _gidz <= k)
        if (_gidy >= 1 && _gidy <= m)
            if (_gidx >= 1 && _gidx <= n)
}
```

**Unpack CUDA pitched ptrs**

**Compute local and global indices using Thread & block IDs**

**If-statements derived from for-statements**
• The translated (un-optimized) code performs all the memory references through global memory
• Optimizations focus on using fast on-chip memory
• **Analyzer determines pattern of array accesses involving central point and nearest neighbors**
• **How much shared memory do we need?**
• **Which ghost cells to load?**
## Other annotation based translators

<table>
<thead>
<tr>
<th>Gflops</th>
<th>Mint</th>
<th>PGI</th>
<th>OpenMPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-pt Heat Eqn (2D)</td>
<td>14.4</td>
<td>-</td>
<td>7.3</td>
</tr>
<tr>
<td>7pt Heat Eqn (3D)</td>
<td>22.2</td>
<td>9.0</td>
<td>1.06</td>
</tr>
<tr>
<td>19-pt Heat Eqn (3D)</td>
<td>15.8</td>
<td>11.3</td>
<td>-</td>
</tr>
</tbody>
</table>

PGI results (V11.1): Lincoln @ NCSA (Tesla C1060)  
Mint and OpenMPC: Tesla C1060 @ UCSD
Mint achieves on average 76% of hand-code CUDA on the 400-series and 79% on the 200-series.
Whole applications

- Greater optimization challenges
  - Many input grids are involved
  - Split kernels because of limited on-chip memory
- Harris Corner Detection algorithm
  - With Han Suk Kim
  - ×26 speedup compared with 4 OpenMP threads
- Earthquake simulation
  (Gordon Bell Finalist SC’10)
  - Yifeng Cui - San Diego Supercomputer Center
Discussion and Conclusions

- Domain-specific translation encapsulates expert knowledge, insulates SW against disruption
- 2 approaches that enable the domain scientist to work with a familiar notation
  - Heart modeling translator
  - Mint: stencil methods
    - CUDA-free programming based on just 5 pragmas
    - Performance competitive w/ hand coded CUDA
- Future work: new applications, more comprehensive optimizations
- Family of compilers responding to the needs of different application motifs (Colella’s 7 dwarfs)
Acknowledgements and Support

• Xing Cai (Simula), Fred Lionetti (UCSD), Andrew McCulloch (UCSD), Han Suk Kim (UCSD), Ross Walker (SDSC)

• Mint will be available in the summer
  http://sites.google.com/site/mintmodel
  http://www-cse.ucsd.edu/groups/hpcl/scg