Lecture 22 - Programming the Cell BE

Mattan Erez

The University of Texas at Austin
Outline

• Cell programming challenges review
• **Sequoia**
  - Review + mapping
• Other Cell programming tools

• Sequoia part courtesy Kayvon Fatahalian, Stanford

• All Cell related images and figures © Sony and IBM
• **Cell Broadband Engine™ Sony Corp.**
Emerging Themes

- Writing high-performance code amounts to...
  - Intelligently structuring algorithms
    [compiler help unlikely]
  - Efficiently using communication
  - Efficiently using parallel resources
    [compilers struggle without help]
  - Generating efficient inner loops (kernels)
    [compilers coming around]
Sequoia

- **Language**: stream programming for machines with deep memory hierarchies

- **Idea**: Expose abstract memory hierarchy to programmer

- **Implementation**: *language, compiler, tuner, and runtime*
  - benchmarks run well on Cell processor based systems, clusters of PCs, SMPs, out-of-core computation, and combinations of above
• Key challenge in high performance programming is:

• communication (not parallelism)

  • **Latency**
  • **Bandwidth**
Streaming

• Streaming involves structuring algorithms as collections of independent [locality cognizant] computations with well-defined working sets.

• **This structuring may be done at any scale.**

  Keep temporaries in registers
  Cache/scratchpad blocking
  Message passing on a cluster
  Out-of-core algorithms
Streaming

- Streaming involves structuring algorithms as collections of independent [locality cognizant] computations with well-defined working sets.

Efficient programs exhibit this structure at many scales.
Roll of programming model

- **Encourage hardware-friendly structure**
- Bulk operations
- Bandwidth matters: structure code to maximize locality
- Parallelism matters: make parallelism explicit
- Awareness of memory hierarchy applies everywhere
  - Keep temporaries in registers
  - Cache/scratchpad blocking
  - Message passing on a cluster
  - Out-of-core algorithms
Sequoia’s goals

• Facilitate development of hierarchy-aware stream programs...
  ... that remain portable across machines

• Provide constructs that can be implemented efficiently without requiring advanced compiler technology (but facilitate optimization)
  - Place computation and data in machine
  - Explicit parallelism and communication
  - Large bulk transfers

• Get out of the way when needed
Hierarchical memory in Sequoia
Hierarchical memory

- Abstract machines as trees of memories

Similar to: Parallel Memory Hierarchy Model (Alpem et al.)
Hierarchical memory

- Abstract machines as trees of memories
Hierarchical memory
Hierarchical memory

Main memory

LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS ALUs LS
Hierarchical memory

Aggregate cluster memory
(virtual level)

Main memory

Main memory
Hierarchical memory

- Disk
- Main memory
- ALUs
- LS
void matmul(int M, int N, int T, float* A, float* B, float* C)
{
    Perform series of L2 matrix multiplications.
}

\[ \mathbf{C} \leftarrow \mathbf{A} \times \mathbf{B} \]
Sequoia’s method

• Explicit communication between abstract memories

• Locality awareness

• Hierarchy portability
  – Across machines, within levels of a machine

• Programmer expresses combined computation and decomposition parameterized algorithm
  – System follows algorithm to map to a specific machine
Sequoia tasks
Sequoia tasks

- Special functions called **tasks** are the building blocks of Sequoia programs

```c
task matmul::leaf( in float A[M][T],
                  in float B[T][N],
                  inout float C[M][N] )
{
    for (int i=0; i<M; i++)
        for (int j=0; j<N; j++)
            for (int k=0; k<T; k++)
                C[i][j] += A[i][k] * B[k][j];
}
```

Read-only parameters M, N, T give sizes of multidimensional arrays when task is called.
Sequoia tasks

- Single abstraction for
  - Isolation / parallelism
  - Explicit communication / working sets
  - Expressing locality

- Tasks operate on arrays, not array elements

- Tasks nest: they call subtasks
Sequoia tasks

- Task arguments and temporaries define a working set
- **Task working set resident at single location in abstract machine tree**

```c
task matmul::leaf( in float A[M][T],
                   in float B[T][N],
                   inout float C[M][N] )
{
    for (int i=0; i<M; i++)
        for (int j=0; j<N; j++)
            for (int k=0; k<T; k++)
                C[i][j] += A[i][k] * B[k][j];
}
```
**Task hierarchies**

```c
task matmul::inner( in  float A[M][T],
                   in  float B[T][N],
                   inout float C[M][N] )
{
    tunable int P, Q, R;

    Recursively call matmul task on submatrices of A, B, and C of size PxQ, QxR, and PxR.
}
```

```c
task matmul::leaf( in  float A[M][T],
                  in  float B[T][N],
                  inout float C[M][N] )
{
    for (int i=0; i<M; i++)
        for (int j=0; j<N; j++)
            for (int k=0; k<T; k++)
                C[i][j] += A[i][k] * B[k][j];
}
```
Task hierarchies

task matmul::inner( in float A[M][T],
                   in float B[T][N],
                   inout float C[M][N] )
{
    tunable int P, Q, R;

    mappar( int i=0 to M/P, int j=0 to N/R ) {
        mapseq( int k=0 to T/Q ) {
            matmul( A[P*i:P*(i+1);P][Q*k:Q*(k+1);Q],
                    B[Q*k:Q*(k+1);Q][R*j:R*(j+1);R],
                    C[P*i:P*(i+1);P][R*j:R*(j+1);R] );
        }
    }
}

task matmul::leaf( in float A[M][T],
                   in float B[T][N],
                   inout float C[M][N] )
{
    for (int i=0; i<M; i++)
        for (int j=0; j<N; j++)
            for (int k=0; k<T; k++)
                C[i][j] += A[i][k] * B[k][j];
}
Task hierarchies

task matmul::inner( in float A[M][T],
in float B[T][N],
inout float C[M][N] )
{
  tunable int P, Q, R;

  mappar( int i=0 to M/P,
           int j=0 to N/R ) {
    mapseq( int k=0 to T/Q ) {
      matmul( A[P*i:P*(i+1);P][Q*k:Q*(k+1);Q],
              B[Q*k:Q*(k+1);Q][R*j:R*(j+1);R],
              C[P*i:P*(i+1);P][R*j:R*(j+1);R] );
    }
  }
}

task matmul::leaf( in float A[M][T],
in float B[T][N],
inout float C[M][N] )
{
  for (int i=0; i<M; i++)
    for (int j=0; j<N; j++)
      for (int k=0; k<T; k++)
        C[i][j] += A[i][k] * B[k][j];
}
Task hierarchies

```cpp
task matmul::inner( in float A[M][T],
                    in float B[T][N],
                    inout float C[M][N] )
{
    tunable int P, Q, R;

    mappar( int i=0 to M/P,
            int j=0 to N/R ) {
        mapseq( int k=0 to T/Q ) {
            matmul( A[P*i:P*(i+1);P][Q*k:Q*(k+1);Q],
                    B[Q*k:Q*(k+1);Q][R*j:R*(j+1);R],
                    C[P*i:P*(i+1);P][R*j:R*(j+1);R] );
        }
    }
}
```

- Tasks express multiple levels of parallelism
Leaf variants

- Be practical: Can use platform-specific kernels

```c
task matmul::leaf(in float A[M][T],
                  in float B[T][N],
                  inout float C[M][N])
{
    for (int i=0; i<M; i++)
        for (int j=0; j<N; j++)
            for (int k=0; k<T; k++)
                C[i][j] += A[i][k] * B[k][j];
}

task matmul::leaf_cblas(in float A[M][T],
                        in float B[T][N],
                        inout float C[M][N])
{
    cblas_sgemm(A, M, T, B, T, N, C, M, N);
}
```
Synchronization

- `mapseq` implies sync at end of every iteration
- `mappar` implies sync at end of iteration space

- No explicit synchronization
  - Why?
- Synchronization is the trickiest part of parallel programming and one of the least portable
  - Help the user by structuring sync and allowing compiler to optimize the mechanism
Synchronization Impacts Parallelism

- Parallelism explicitly expressed using mappar
  - DLP
- What about ILP?
  - Parallelism can exist within a leaf
    - Ignored by Sequoia but potential for ILP and SIMD
- What about TLP?
  - Implicit in dependence of operations
  - Allows pipeline parallelism within a mappar
- What about interacting thread?
  - Not allowed!
  - Why?
Summary: Sequoia tasks

• Single abstraction for
  - Isolation / parallelism
  - Explicit communication / working sets
  - Expressing locality

• Sequoia programs describe hierarchies of tasks
  - Mapped onto memory hierarchy
  - Parameterized for portability
  - Algorithm for decomposition
Mapping tasks to machines
How mapping works

Sequoia task definitions (parameterized)

matmul::inner

matmul::leaf

Mapping specification

instance {
    name = matmul_node_inst
    variant = inner
    runs_at = main_memory
    tunable P=256, Q=256, R=256
}

instance {
    name = matmul_L2_inst
    variant = inner
    runs_at = L2_cache
    tunable P=32, Q=32, R=32
}

instance {
    name = matmul_L1_inst
    variant = leaf
    runs_at = L1_cache
}

Task instances (not parameterized)

matmul_node_inst
variant = inner
P=256 Q=256 R=256
node level

matmul_L2_inst
variant = inner
P=32 Q=32 R=32
L2 level

matmul_L1_inst
variant = leaf
L1 level
Task mapping specification

instance {
    name = matmul_node_inst
    task = matmul
    variant = inner
    runs_at = main_memory
    tunable P=256, Q=256, R=256
    calls = matmul_L2_inst
}

instance {
    name = matmul_L2_inst
    task = matmul
    variant = inner
    runs_at = L2_cache
    tunable P=32, Q=32, R=32
    calls = matmul_L1_inst
}

instance {
    name = matmul_L1_inst
    task = matmul
    variant = leaf
    runs_at = L1_cache
    }
Specializing matmul

- Instances of tasks placed at each memory level

```
matmul::inner
M=N=T=1024
P=Q=R=256

matmul::inner
M=N=T=256
P=Q=R=32

matmul::inner
M=N=T=256
P=Q=R=32

matmul::inner
M=N=T=256
P=Q=R=32

matmul::leaf
M=N=T=32

matmul::leaf
M=N=T=32

matmul::leaf
M=N=T=32
```

- Main memory
- L2 cache
- L1 cache
Task instances: Cell

Sequoia task definitions (parameterized)

matmul::inner

matmul::leaf

Cell mapping specification

instance {
    name = matmul_node_inst
    variant = inner
    runs_at = main_memory
    tunable P=32, Q=64, R=32
calls = matmul_LS_inst
}

instance {
    name = matmul_LS_inst
    variant = leaf
    runs_at = LS_cache
}

Cell task instances (not parameterized)

matmul_node_inst
variant = inner
P=32 Q=64 R=32
node level

matmul_LS_inst
variant = leaf
LS level
Preview of results

- Performance competitive with native code
- Portable: no source-code changes for different configurations
- Maximizes resources (compute or communication)
- Low overhead
Results

• We have a Sequoia compiler + runtime systems for multiple platforms
  - Cell/PS3
  - Cluster
  - Disk
  - SMP

• Static compiler optimizations (bulk operation IR)
  - Copy elimination
  - DMA transfer coalescing
  - Operation hoisting
  - Array allocation / packing
  - Scheduling (tasks and DMAs)

• Runtimes can be composed
  - Cluster of PS3s
  - Disk + Cell
  - Cluster of SMPs
Scientific computing benchmarks

Linear Algebra  Blas Level 1 SAXPY, Level 2 SGEMV, and Level 3 SGEMM benchmarks

Conv2D  2D convolution with 9x9 support (non-periodic boundary constraints)

FFT3D  $256^3$ complex FFT

Gravity  100 time steps of N-body stellar dynamics simulation

HMMER  Fuzzy protein string matching using HMM evaluation (Daniel Horn’s SC2005 paper)
System configurations

- **Disk**
  - 2.4 GHz Intel P4, 160GB disk, ~50MB/s from disk

- **8-way SMP**
  - 4 dual-core 2.66 Intel P4 Xeons, 8GB

- **Cluster**
  - 16, 2-way Intel 2.4GHz P4 Xeons, 1GB/node, Infiniband

- **Cell**
  - 3.2 GHz IBM Cell blade (8SPE), 1GB

- **PS3**
  - 3.2 GHz Cell in Sony Playstation 3 (6 SPE), 256MB (160MB usable)
## Results - Horizontal portability - GFlop/s

<table>
<thead>
<tr>
<th></th>
<th>Scalar</th>
<th>SMP</th>
<th>Disk</th>
<th>Cluster</th>
<th>Cell</th>
<th>PS3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>0.3</td>
<td>0.7</td>
<td>0.007</td>
<td>1.4</td>
<td>3.5</td>
<td>3.1</td>
</tr>
<tr>
<td>SG EMV</td>
<td>1.1</td>
<td>1.7</td>
<td>0.04</td>
<td>3.8</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>SG EMM</td>
<td>6.9</td>
<td>45</td>
<td>5.5</td>
<td>91</td>
<td>119</td>
<td>94</td>
</tr>
<tr>
<td>CONV2D</td>
<td>1.9</td>
<td>7.8</td>
<td>0.6</td>
<td>24</td>
<td>85</td>
<td>62</td>
</tr>
<tr>
<td>FFT3D</td>
<td>1.5</td>
<td>7.8</td>
<td>0.1</td>
<td>7.5</td>
<td>54</td>
<td>31*</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>4.8</td>
<td>40</td>
<td>3.7</td>
<td>68</td>
<td>97</td>
<td>71</td>
</tr>
<tr>
<td>HMMER</td>
<td>0.9</td>
<td>11</td>
<td>0.9</td>
<td>12</td>
<td>12</td>
<td>7.1*</td>
</tr>
</tbody>
</table>
### Results - Horizontal portability - GFlop/s

<table>
<thead>
<tr>
<th></th>
<th>Scalar</th>
<th>SMP</th>
<th>Disk</th>
<th>Cluster</th>
<th>Cell</th>
<th>PS3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SAXPY</strong></td>
<td>0.3</td>
<td>0.7</td>
<td>0.007</td>
<td>1.4</td>
<td>3.5</td>
<td>3.1</td>
</tr>
<tr>
<td><strong>SG EMV</strong></td>
<td>1.1</td>
<td>1.7</td>
<td>0.04</td>
<td>3.8</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td><strong>SG EMM</strong></td>
<td>6.9</td>
<td>45</td>
<td>5.5</td>
<td>91</td>
<td>119</td>
<td>94</td>
</tr>
<tr>
<td><strong>CONV2D</strong></td>
<td>1.9</td>
<td>7.8</td>
<td>0.6</td>
<td>24</td>
<td>85</td>
<td>62</td>
</tr>
<tr>
<td><strong>FFT3D</strong></td>
<td>1.5</td>
<td>7.8</td>
<td>0.1</td>
<td>7.5</td>
<td>54</td>
<td>31*</td>
</tr>
<tr>
<td><strong>GRAVITY</strong></td>
<td>4.8</td>
<td>40</td>
<td>3.7</td>
<td>68</td>
<td>97</td>
<td>71</td>
</tr>
<tr>
<td><strong>HMMER</strong></td>
<td>0.9</td>
<td>11</td>
<td>0.9</td>
<td>12</td>
<td>12</td>
<td>7.1*</td>
</tr>
</tbody>
</table>

* Reduced dataset size to fit in memory
2 Level Utilization

- Idle waiting on Xfer (M1-M0)
- Runtime Overhead (M1-M0)
- Leaf task execution (M0)

Percentage of application run-time

SAXPY  SGEMV  SGEMM  CONV2D  FFT3D  GRAVITY  HMMER
## Results - Vertical Portability - GFlop/s

<table>
<thead>
<tr>
<th></th>
<th>Cluster-SMP</th>
<th>Disk+PS3</th>
<th>PS3 Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>0.5</td>
<td>0.004</td>
<td>0.23</td>
</tr>
<tr>
<td>SGEMV</td>
<td>1.4</td>
<td>0.014</td>
<td>1.3</td>
</tr>
<tr>
<td>SGEMM</td>
<td>48</td>
<td>3.7</td>
<td>30</td>
</tr>
<tr>
<td>CONV2D</td>
<td>4.8</td>
<td>0.48</td>
<td>3.24</td>
</tr>
<tr>
<td>FFT3D</td>
<td>2.1</td>
<td>0.05</td>
<td>0.36</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>50</td>
<td>66</td>
<td>119</td>
</tr>
<tr>
<td>HMMER</td>
<td>14</td>
<td>8.3</td>
<td>13</td>
</tr>
</tbody>
</table>
## Results - Vertical Portability - GFlop/s

<table>
<thead>
<tr>
<th></th>
<th>Cluster-SMP</th>
<th>Disk+PS3</th>
<th>PS3 Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>0.5</td>
<td>0.004</td>
<td>0.23</td>
</tr>
<tr>
<td>SG EMV</td>
<td>1.4</td>
<td>0.014</td>
<td>1.3</td>
</tr>
<tr>
<td>SG EMM</td>
<td>48</td>
<td>3.7</td>
<td>30</td>
</tr>
<tr>
<td>CONV2D</td>
<td>4.8</td>
<td>0.48</td>
<td>3.24</td>
</tr>
<tr>
<td>FFT3D</td>
<td>2.1</td>
<td>0.05</td>
<td>0.36</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>50</td>
<td>66</td>
<td>119</td>
</tr>
<tr>
<td>HMMER</td>
<td>14</td>
<td>8.3</td>
<td>13</td>
</tr>
</tbody>
</table>

*Bandwidth bound*
Composed systems utilization

- Idle waiting on Xfer (M2-M1)
- Overhead (M2-M1)
- Idle waiting on Xfer (M1-M0)
- Overhead (M1-M0)
- Leaf task execution (M0)

Percentage of application run-time

Applications: SAXPY, SG, EM, SG, EMV, SG, EMM, CONV2DF, F, 3D, GRAVITY, HMMER
Cell utilization

- DRAM Utilization: Sustained BW, as percentage of attainable peak
- SPE Utilization: Percentage of time the SPEs are running a kernel
Performance scaling

SPE scaling on 2.4GHz Dual-Cell blade

Scaling on P4 cluster with Infiniband interconnect

- SAXPY
- SGEMV
- SGEMM
- Conv2D
- FFT3D
- Gravity
- HMMER
Sequoia summary

• Problem:
  – Deep memory hierarchies pose perf. programming challenge
  – Memory hierarchy different for different machines

• Solution: Abstract hierarchical memory in programming model
  – Program the memory hierarchy explicitly
  – Expose properties that effect performance

• Approach: Express hierarchies of tasks
  – Execute in local address space
  – Call-by-value-result semantics exposes communication
  – Parameterized for portability
Sequoia and Cell Programming Challenges

- Sequoia manages threading and synchronization
- Sequoia manages communication and all DMAs
  - Including padding and performance, but not alignment
- Sequoia manages LS
  - Allocation and packing
- Sequoia manages scheduling
  - SWP of mapparto to hide communication latency

- Sequoia doesn’t help with SPE code
  - Use low-level compiler tools such as XLC
- Sequoia doesn’t currently help with some memory restrictions
  - Alignment
  - Banks