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

Streaming

- Key challenge in high performance programming is:
  - communication (not parallelism)
    - Latency
    - Bandwidth

- Streaming involves structuring algorithms as collections of independent [locally 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
  - 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

• Abstract machines as trees of memories

Hierarchical memory in Sequoia

Similar to:
Parallel Memory Hierarchy Model (Alpert et al.)

Hierarchical memory

• Abstract machines as trees of memories

Hierarchical memory
Hierarchical memory

- Main memory
  - LS
  - ALUs

Single Cell blade

- Main memory
  - LS
  - ALUs

Dual Cell blade

- Main memory
  - LS
  - ALUs

Cluster of dual Cell blades

- Main memory
  - LS
  - ALUs

Aggregate cluster memory (virtual level)

- Main memory
  - LS
  - ALUs

Hierarchical memory

Blocked matrix multiplication

```c
void matmul(int M, int N, int T,
            float* A, float* B, float* C)
{
    Perform series of L2 matrix multiplications.
}
```

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

• Special functions called tasks are the building blocks of Sequoia programs

```c
void matmul::leaf( in float A[M][T],
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

Task hierarchies

```c
void matmul::inner( in float A[M][T],
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.
}
```

task matmul::leaf( in float A[M][T],
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
void matmul::inner( in float A[M][T],
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] );
        }
    }
}
```

Variant call graph
Task hierarchies

Calling task: matmul::inner
Located at level X

Calling task: matmul::leaf
Located at level Y

Task hierarchies

Calling task: matmul::inner
Located at level X

Calling task: matmul::leaf
Located at level Y

Leaf variants

• Be practical: Can use platform-specific kernels

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

- `matmul::inner`
- `matmul::leaf`

Mapping specification

Task instances (not parameterized)

- `matmul_node_inst`
  - variant = `inner`
  - `P=256, Q=256, R=256`
- `matmul_L2_inst`
  - variant = `inner`
  - `P=32, Q=32, R=32`
- `matmul_L1_inst`
  - variant = `leaf`
  - `P=32, Q=32, R=32`

Task mapping specification

- `instance`
  - `name = matmul_node_inst`
  - `variant = inner`
  - `P=256, Q=256, R=256`
  - `tunable`
  - `calls = matmul_L2_inst`
- `instance`
  - `name = matmul_L2_inst`
  - `variant = inner`
  - `P=32, Q=32, R=32`
  - `tunable`
  - `calls = matmul_L1_inst`
- `instance`
  - `name = matmul_L1_inst`
  - `variant = leaf`
  - `P=32, Q=32, R=32`

Mapping specification

Task instances: Cell

Sequoia task definitions

- `matmul::inner`
- `matmul::leaf`

Cell mapping specification

- `matmul_node_inst`
  - variant = `inner`
  - `P=32, Q=64, R=32`
- `matmul_L2_inst`
  - variant = `leaf`
  - `P=32, Q=64, R=32`

Task instances: Cell (not parameterized)

- `matmul_node_inst`
  - variant = `inner`
  - `P=32, Q=64, R=32`
  - `tunable`
  - `calls = matmul_L2_inst`
- `matmul_L2_inst`
  - variant = `leaf`
  - `P=32, Q=64, R=32`

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
  - 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\(^6\) 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 GHz 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>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>1.5</td>
</tr>
<tr>
<td>SGEMV</td>
<td>1.1</td>
<td>1.7</td>
<td>0.04</td>
<td>3.8</td>
<td>12</td>
</tr>
<tr>
<td>SGEMM</td>
<td>6.9</td>
<td>45</td>
<td>5.5</td>
<td>91</td>
<td>119</td>
</tr>
<tr>
<td>Conv2D</td>
<td>1.9</td>
<td>7.8</td>
<td>0.6</td>
<td>24</td>
<td>85</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>
</tr>
<tr>
<td>Gravity</td>
<td>4.8</td>
<td>40</td>
<td>3.7</td>
<td>68</td>
<td>97</td>
</tr>
<tr>
<td>HMMER</td>
<td>0.9</td>
<td>11</td>
<td>0.9</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

Results – Horizontal portability - GFlop/s

<table>
<thead>
<tr>
<th></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>1.5</td>
</tr>
<tr>
<td>SGEMV</td>
<td>1.1</td>
<td>1.7</td>
<td>0.04</td>
<td>3.8</td>
<td>12</td>
</tr>
<tr>
<td>SGEMM</td>
<td>6.9</td>
<td>45</td>
<td>5.5</td>
<td>91</td>
<td>119</td>
</tr>
<tr>
<td>Conv2D</td>
<td>1.9</td>
<td>7.8</td>
<td>0.6</td>
<td>24</td>
<td>85</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>
</tr>
<tr>
<td>Gravity</td>
<td>4.8</td>
<td>40</td>
<td>3.7</td>
<td>68</td>
<td>97</td>
</tr>
<tr>
<td>HMMER</td>
<td>0.9</td>
<td>11</td>
<td>0.9</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

2 Level Utilization

<table>
<thead>
<tr>
<th></th>
<th>SMP</th>
<th>Disk</th>
<th>Cluster</th>
<th>Cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>0.3</td>
<td>0.7</td>
<td>0.007</td>
<td>1.4</td>
</tr>
<tr>
<td>SGEMV</td>
<td>1.1</td>
<td>1.7</td>
<td>0.04</td>
<td>3.8</td>
</tr>
<tr>
<td>SGEMM</td>
<td>6.9</td>
<td>45</td>
<td>5.5</td>
<td>91</td>
</tr>
<tr>
<td>Conv2D</td>
<td>1.9</td>
<td>7.8</td>
<td>0.6</td>
<td>24</td>
</tr>
<tr>
<td>FFT3D</td>
<td>1.5</td>
<td>7.8</td>
<td>0.1</td>
<td>7.5</td>
</tr>
<tr>
<td>Gravity</td>
<td>4.8</td>
<td>40</td>
<td>3.7</td>
<td>68</td>
</tr>
<tr>
<td>HMMER</td>
<td>0.9</td>
<td>11</td>
<td>0.9</td>
<td>12</td>
</tr>
</tbody>
</table>

Bandwidth bound

<table>
<thead>
<tr>
<th></th>
<th>SMP</th>
<th>Disk</th>
<th>Cluster</th>
<th>Cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>0.3</td>
<td>0.7</td>
<td>0.007</td>
<td>1.4</td>
</tr>
<tr>
<td>SGEMV</td>
<td>1.1</td>
<td>1.7</td>
<td>0.04</td>
<td>3.8</td>
</tr>
<tr>
<td>SGEMM</td>
<td>6.9</td>
<td>45</td>
<td>5.5</td>
<td>91</td>
</tr>
<tr>
<td>Conv2D</td>
<td>1.9</td>
<td>7.8</td>
<td>0.6</td>
<td>24</td>
</tr>
<tr>
<td>FFT3D</td>
<td>1.5</td>
<td>7.8</td>
<td>0.1</td>
<td>7.5</td>
</tr>
<tr>
<td>Gravity</td>
<td>4.8</td>
<td>40</td>
<td>3.7</td>
<td>68</td>
</tr>
<tr>
<td>HMMER</td>
<td>0.9</td>
<td>11</td>
<td>0.9</td>
<td>12</td>
</tr>
</tbody>
</table>

* Reduced dataset size to fit in memory
Results – Vertical Portability - GFlop/s

<table>
<thead>
<tr>
<th>Application</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>HAMMER</td>
<td>14</td>
<td>8.3</td>
<td>13</td>
</tr>
</tbody>
</table>

Composed systems utilization

<table>
<thead>
<tr>
<th>Component</th>
<th>Bandwidth bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td></td>
</tr>
<tr>
<td>SGEMV</td>
<td></td>
</tr>
<tr>
<td>SGEMM</td>
<td></td>
</tr>
<tr>
<td>CONV2D</td>
<td></td>
</tr>
<tr>
<td>FFT3D</td>
<td></td>
</tr>
<tr>
<td>GRAVITY</td>
<td></td>
</tr>
<tr>
<td>HAMMER</td>
<td></td>
</tr>
</tbody>
</table>

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

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
<table>
<thead>
<tr>
<th>Sequoia and Cell Programming Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Sequoia manages threading and synchronization</td>
</tr>
<tr>
<td>• Sequoia manages communication and all DMAs</td>
</tr>
<tr>
<td>- Including padding and performance, but not alignment</td>
</tr>
<tr>
<td>• Sequoia manages LS</td>
</tr>
<tr>
<td>- Allocation and packing</td>
</tr>
<tr>
<td>• Sequoia manages scheduling</td>
</tr>
<tr>
<td>- SWP of mappar to hide communication latency</td>
</tr>
<tr>
<td>• Sequoia doesn’t help with SPE code</td>
</tr>
<tr>
<td>- Use low-level compiler tools such as XLC</td>
</tr>
<tr>
<td>• Sequoia doesn’t currently help with some memory restrictions</td>
</tr>
<tr>
<td>- Alignment</td>
</tr>
<tr>
<td>- Banks</td>
</tr>
</tbody>
</table>