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.
Cell Software Challenges

- **Separate code for PPE and SPEs**
  - Explicit synchronization

- **SPEs can only access memory through DMAs**
  - DMA is asynchronous, but prep instructions are part of SPE code
  - SW responsible for consistency and coherency
  - SW responsible for alignment, granularity, and bank conflicts

- **SPEs must be programmed with SIMD**
  - Alignment is up to SW
  - Lots of pipeline challenges left up to programmer/compiler
    - Deep pipeline with no branch predictor
    - 2-wide scalar pipeline needs static scheduling
    - LS shared by DMA, instruction fetch, and SIMD LD/ST
    - No memory protection on LS (Stack can “eat” data or code)
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.
Sequoia
Programming the Memory Hierarchy

Kayvon Fatahalian
Daniel Reiter Horn
Alex Aiken

Timothy J. Knight
Larkhoon Leem
William J. Dally

Mike Houston
Ji Young Park
Pat Hanrahan

Mattan Erez
Manman Ren

Stanford University
Sequoia

- Language: stream programming for machines with deep memory hierarchies

- Idea: Expose abstract memory hierarchy to programmer

- Implementation: benchmarks run well on Cell processor based systems and on cluster of PCs
Key challenge in high performance programming is:

communication (not parallelism)

Latency
Bandwidth
Avoiding latency stalls

1. Prefetch batch of data
2. Compute on data (avoiding stalls)
3. Initiate write of results

... Then compute on next batch (which should be loaded)

```
compute 1
  read input 2
  write output 0

compute 2
  read input 3
  write output 1

compute 3
  read input 4
  write output 2
```
Exploit locality

- Compute > bandwidth, else execution stalls
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.
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
  - Place computation and data in machine
  - Explicit parallelism and communication
  - Large bulk transfers
Hierarchical memory in Sequoia
Hierarchical memory

Single Cell blade

Main memory

Virtual aggregate LS

ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
void matmul_L1(int M, int N, int T,
    float* A,
    float* B,
    float* C)
{
    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];
}
void matmul_L2( int M, int N, int T,  
    float* A,  
    float* B,  
    float* C)  
{

    Perform series of L1 matrix multiplications.

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

C += A \times B
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

- 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
#define tunable int P, Q, R;

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

```c
#define 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
#define 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];
}
```

Callee task: matmul::leaf
Located at level Y

Calling task: matmul::inner
Located at level X
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;

    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
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
    - Compiler may not currently interchange loops

- What about interacting thread?
  - Not allowed!
  - Why?
Summary: Sequoia tasks

- Single abstraction for
  - Isolation / parallelism
    - With help from programmer
  - Explicit communication / working sets
  - Expressing locality

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

Sequoia task definitions (parameterized)

matmul::inner

matmul::leaf

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

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
}
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
}

PC task instances

- *matmul_node_inst*
  - variant: inner
  - P=256, Q=256, R=256
  - runs_at: main_memory

- *matmul_L2_inst*
  - variant: inner
  - P=32, Q=32, R=32
  - runs_at: L2_cache

- *matmul_L1_inst*
  - 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

... 64 total subtasks...

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

... 512 total subtasks...

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 task instances (not parameterized)

- matmul_node_inst
  - variant = inner
  - P=128, Q=64, R=128
- matmul_LS_inst
  - variant = leaf

Sequoia Compiler

Sequoia Compiler instance:

- name = matmul_node_inst
  - variant = inner
  - runs_at = main_memory
  - tunable P=128, Q=64, R=128

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

© Kayvon Fatahalian
Results
Early results

- We have a Sequoia compiler + runtime systems ported to Cell and a cluster of PCs

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

“Compilation for Explicitly Managed Memories”
Knight et al. PPOPP ’07
Early results

- **Scientific computing benchmarks**

<table>
<thead>
<tr>
<th>Linear Algebra</th>
<th>Blas Level 1 SAXPY, Level 2 SGEMV, and Level 3 SGEMM benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IterConv2D</td>
<td>Iterative 2D convolution with 9x9 support (non-periodic boundary constraints)</td>
</tr>
<tr>
<td>FFT3D</td>
<td>$256^3$ complex FFT</td>
</tr>
<tr>
<td>Gravity</td>
<td>100 time steps of N-body stellar dynamics simulation</td>
</tr>
<tr>
<td>HMMER</td>
<td>Fuzzy protein string matching using HMM evaluation</td>
</tr>
<tr>
<td></td>
<td>(ClawHMMer: Horn et al. SC2005)</td>
</tr>
</tbody>
</table>
Utilization

Execution on a Cell blade (left bars) and 16 node cluster (right bars)
Utilization

Execution on a Cell blade

- Idle waiting on memory/network
- Sequoia overhead
- Leaf task computation

Bandwidth bound apps achieve over 90% of peak DRAM bandwidth
Utilization

Execution on a Cell blade (left bars) and 16 node cluster (right bars)
Performance

SPE scaling on 2.4Ghz Dual-Cell blade

Scaling on P4 cluster with Infiniband interconnect

![Graph showing SPE scaling and scaling on P4 cluster with Infiniband interconnect]
## Performance: GFLOP/sec

*(single precision floating point)*

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>3.2</td>
<td>4.0</td>
<td>3.6</td>
</tr>
<tr>
<td>SGEMV</td>
<td>9.8</td>
<td>11.0</td>
<td>11.1</td>
</tr>
<tr>
<td>SGEMM</td>
<td>96.3</td>
<td>174.0</td>
<td>97.9</td>
</tr>
<tr>
<td>IterConv2D</td>
<td>62.8</td>
<td>119.0</td>
<td>27.2</td>
</tr>
<tr>
<td>FFT3D</td>
<td>43.5</td>
<td>45.2</td>
<td>6.8</td>
</tr>
<tr>
<td>Gravity</td>
<td>83.3</td>
<td>142.0</td>
<td>50.6</td>
</tr>
<tr>
<td>HMMER</td>
<td>9.9</td>
<td>19.1</td>
<td>13.4</td>
</tr>
</tbody>
</table>

* 2.4 GHz Cell processor, DD2
** 2.4 GHz Pentium 4 per node
## Performance: GFLOP/sec

*(single precision floating point)*

<table>
<thead>
<tr>
<th></th>
<th>Single Cell (8 SPE)</th>
<th>Dual Cell (16 SPE)</th>
<th>Cluster (16 nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>3.2</td>
<td>4.0</td>
<td>3.6</td>
</tr>
<tr>
<td>SGEMV</td>
<td>9.8</td>
<td>11.0</td>
<td>11.1</td>
</tr>
<tr>
<td>SGEMM</td>
<td>96.3</td>
<td>174.0</td>
<td>97.9</td>
</tr>
<tr>
<td>IterConv2D</td>
<td>62.8</td>
<td>119.0</td>
<td>27.2</td>
</tr>
<tr>
<td>FFT3D</td>
<td>43.5</td>
<td>45.2</td>
<td>6.8</td>
</tr>
<tr>
<td>Gravity</td>
<td>83.3</td>
<td>142.0</td>
<td>50.6</td>
</tr>
<tr>
<td>HMMER</td>
<td>9.9</td>
<td>19.1</td>
<td>13.4</td>
</tr>
</tbody>
</table>

- Single Cell >= 16 node cluster of P4’s

* 2.4 GHz Cell processor, DD2
** 2.4 GHz Pentium 4 per node
### Performance: GFLOP/sec
(single precision floating point)

<table>
<thead>
<tr>
<th></th>
<th>Single Cell (8 SPE)</th>
<th>Dual Cell (16 SPE)</th>
<th>Cluster (16 nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>3.2</td>
<td>4.0</td>
<td>3.6</td>
</tr>
<tr>
<td>SGEMV</td>
<td>9.8</td>
<td>11.0</td>
<td>11.1</td>
</tr>
<tr>
<td>SGEMM</td>
<td>96.3</td>
<td>174.0</td>
<td>97.9</td>
</tr>
<tr>
<td>IterConv2D</td>
<td>62.8</td>
<td>119.0</td>
<td>27.2</td>
</tr>
<tr>
<td>FFT3D</td>
<td>43.5</td>
<td>45.2</td>
<td>6.8</td>
</tr>
<tr>
<td>Gravity</td>
<td>83.3</td>
<td>142.0</td>
<td>50.6</td>
</tr>
<tr>
<td>HMMER</td>
<td>9.9</td>
<td>19.1</td>
<td>13.4</td>
</tr>
</tbody>
</table>

- Results on Cell on-par or better than best-known implementations on any architecture

* 2.4 GHz Cell processor, DD2
** 2.4 GHz Pentium 4 per node
## Performance: GFLOP/sec

*(single precision floating point)*

<table>
<thead>
<tr>
<th></th>
<th>Single Cell <em>(8 SPE)</em></th>
<th>Dual Cell <em>(16 SPE)</em></th>
<th>Cluster <em>(16 nodes)</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>3.2</td>
<td>4.0</td>
<td>3.6</td>
</tr>
<tr>
<td>SGEMV</td>
<td>9.8</td>
<td>11.0</td>
<td>11.1</td>
</tr>
<tr>
<td>SGEMM</td>
<td>96.3</td>
<td>174.0</td>
<td>97.9</td>
</tr>
<tr>
<td>IterConv2D</td>
<td>62.8</td>
<td>119.0</td>
<td>27.2</td>
</tr>
<tr>
<td><strong>FFT3D</strong></td>
<td><strong>43.5</strong></td>
<td><strong>45.2</strong></td>
<td><strong>6.8</strong></td>
</tr>
<tr>
<td>Gravity</td>
<td>83.3</td>
<td>142.0</td>
<td>50.6</td>
</tr>
<tr>
<td>HMMER</td>
<td>9.9</td>
<td>19.1</td>
<td>13.4</td>
</tr>
</tbody>
</table>

- **FFT3D on par with best-known Cell implementation**

* 2.4 GHz Cell processor, DD2
** 2.4 GHz Pentium 4 per node
Performance: GFLOP/sec
(single precision floating point)

<table>
<thead>
<tr>
<th></th>
<th>Single Cell</th>
<th>Dual Cell</th>
<th>Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(8 SPE)</td>
<td>(16 SPE)</td>
<td>(16 nodes)</td>
</tr>
<tr>
<td>SAXPY</td>
<td>3.2</td>
<td>4.0</td>
<td>3.6</td>
</tr>
<tr>
<td>SGEMV</td>
<td>9.8</td>
<td>11.0</td>
<td>11.1</td>
</tr>
<tr>
<td>SGEMM</td>
<td>96.3</td>
<td>174.0</td>
<td>97.9</td>
</tr>
<tr>
<td>IterConv2D</td>
<td>62.8</td>
<td>119.0</td>
<td>27.2</td>
</tr>
<tr>
<td>FFT3D</td>
<td>43.5</td>
<td>45.2</td>
<td>6.8</td>
</tr>
<tr>
<td>Gravity</td>
<td><strong>83.3</strong></td>
<td><strong>142.0</strong></td>
<td><strong>50.6</strong></td>
</tr>
<tr>
<td>HMMER</td>
<td>9.9</td>
<td>19.1</td>
<td>13.4</td>
</tr>
</tbody>
</table>

- Gravity outperforms custom ASICs

* 2.4 GHz Cell processor, DD2
** 2.4 GHz Pentium 4 per node
### Performance: GFLOP/sec

*(single precision floating point)*

<table>
<thead>
<tr>
<th></th>
<th>Single Cell <em>(8 SPE)</em></th>
<th>Dual Cell <em>(16 SPE)</em></th>
<th>Cluster <em>(16 nodes)</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>3.2</td>
<td>4.0</td>
<td>3.6</td>
</tr>
<tr>
<td>SGEMV</td>
<td>9.8</td>
<td>11.0</td>
<td>11.1</td>
</tr>
<tr>
<td>SGEMM</td>
<td>96.3</td>
<td>174.0</td>
<td>97.9</td>
</tr>
<tr>
<td>IterConv2D</td>
<td>62.8</td>
<td>119.0</td>
<td>27.2</td>
</tr>
<tr>
<td>FFT3D</td>
<td>43.5</td>
<td>45.2</td>
<td>6.8</td>
</tr>
<tr>
<td>Gravity</td>
<td>83.3</td>
<td>142.0</td>
<td>50.6</td>
</tr>
<tr>
<td><strong>HMMER</strong></td>
<td><strong>9.9</strong></td>
<td><strong>19.1</strong></td>
<td><strong>13.4</strong></td>
</tr>
</tbody>
</table>

- **HMMER outperforms Horn et al.’s GPU implementation from SC05**

---

* 2.4 GHz Cell processor, DD2
** 2.4 GHz Pentium 4 per node
Sequoia portability

- No Sequoia source level modifications except for FFT3D*
  - Changed task parameters
  - Ported leaf task implementations

- Cluster → Cell port (or vice-versa) took 1-2 days

* FFT3D used a different variant on Cell
Sequoia limitations

- Require explicit declaration of working sets
  - Programmer must know what to transfer
  - Some irregular applications present problems

- Manual task mapping
  - Understand which parts can be automated
  - Some progress in automated search for parameters (auto-tuning style)
Sequoia summary

- Enforce structuring already required for performance as integral part of programming model

- Make these hand optimizations portable and easier to perform
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 map parameters 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
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.
Tools From IBM

- **Cell SDK 3.0**
  - API calls for handling communication, synchronization, and DMA
  - LIBSPU and SPUFS for getting the SPEs to do something and setting up threads and memory
  - Intrinsics for programming the SPE pipeline directly
  - GCC port for PPE and SPE part (separate compilers)
    - Only handles non-SPE specific optimizations + intrinsics
  - XLC port for PPE and SPE part (separate compilers)
    - XLC supposed to optimize for SPE pipeline with branch hints, scheduling, instruction prefetch, ...
    - Automatic SIMD-ization?

- **Accelerated Library Framework (ALF)**
  - APIs for work queue based model to program control-plane

- **“Octopiler” – single-source XLC for Cell**
  - OpenMP directives
  - Relies on SW cache to get the OpenMP working
  - Automatic SIMD-ization
Tools from Industry

- **Mercury Systems**
  - Array based language
  - Highly-tuned BLAS and FFT

- **RapidMind**
  - Dynamically compiled program
  - Relies on array data types
  - Builds up kernels and DMAs
Tools From Academia

- **Sequoia**

- **Cell Superscalar (CellSs)**
  - Program with OpenMP like directives to identify kernels
  - Uses SW cache intensively
  - Runtime applies superscalar style optimization and scheduling to coarse-grained kernels (identified above)

- **Charm++**
  - Runtime based approach
  - Objects with explicit communication and “entry points” for synchronization
  - Uses a work queue and peaks into it to do the DMAs