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
John Clark

Stanford University
This Talk

- An brief overview of Sequoia

- What it is
  - Overview of Sequoia implementation

- Port of Sequoia to Roadrunner
  - Status of port and some initial benchmarks

- Plan
  - Future Sequoia work
Sequoia

- **Language**
  - Stream programming for deep memory hierarchies

- **Goals: Performance & Portability**
  - Expose abstract memory hierarchy to programmer

- **Implementation**
  - Benchmarks run well on many multi-level machines
  - Cell, PCs, clusters of PCs, cluster of PS3s, + disk
Key challenge in high performance programming is:

communication
(not parallelism)

Latency
Bandwidth
## Consider Roadrunner

<table>
<thead>
<tr>
<th>Computation</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster of 3264 nodes</td>
<td>Infiniband</td>
</tr>
<tr>
<td>... a node has 2 chips</td>
<td>Infiniband</td>
</tr>
<tr>
<td>... a chip has 2 Opterons</td>
<td>Shared memory</td>
</tr>
<tr>
<td>... an Opteron has a Cell</td>
<td>DACS</td>
</tr>
<tr>
<td>... a Cell has 8 SPEs</td>
<td>Cell API</td>
</tr>
</tbody>
</table>

How do you program a petaflop supercomputer?
Communication: Problem #1

- **Performance**
  - Roadrunner has plenty of compute power
  - The problem is getting the data to the compute units
  - Bandwidth is good, latency is terrible
  - (At least) 5 levels of memory hierarchy

- **Portability**
  - Moving data is done very differently at different levels
  - MPI, DACs, Cell API, ...
  - Port to a different machine => huge rewrite
    - Different protocols for communication
Sequoia’s goals

- Performance and Portability

- Program to an abstract memory hierarchy
  - Explicit parallelism
  - Explicit, but abstract, communication
    - “move this data from here to there”
  - Large bulk transfers

- Compiler/run-time system
  - Instantiate program to a particular memory hierarchy
  - Take care of details of communication protocols, memory sizes, etc.
The sequoia implementation

- Three pieces:
  - Compiler
  - Runtime system
  - Autotuner
Compiler

- Sequoia compilation works on hierarchical programs

- Many “standard” optimizations
  - But done at all levels of the hierarchy
  - Greatly increases leverage of optimization
  - E.g., copy elimination near the root removes not one instruction, but thousands-millions

- Input: Sequoia program
  - Sequoia source file
  - Mapping
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.
How mapping works

Sequoia task definitions (parameterized)

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

Mapping specification

```plaintext
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

- `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
Runtime system

- A runtime implements one memory level
  - Simple, portable API interface
  - Handles naming, synchronization, communication
  - For example Cell runtime abstracts DMA

- A number of existing implementations
  - Cell, disk, PC, clusters of PCs, disk, DACS, ...

- Runtimes are composable
  - Build runtimes for complex machines from runtimes for each memory level

- Compiler target
Graphical runtime representation

Memory Level $i+1$

CPU Level $i+1$

Memory Level $i$

Child $N$

Memory Level $i$

Child 1

CPU Level $i$

Child 1

CPU Level $i$

...

CPU Level $i$

...

CPU Level $i$

Child $N$
Autotuner

- Many parameters to tune
  - Sequoia codes parameterized by tunables
  - Abstract away from machine particulars
    - E.g., memory sizes

- The tuning framework sets these parameters
  - Search-based
  - Programmer defines the search space
  - Bottom line: The Autotuner is a big win
    - Never worse than hand tuning (and much easier)
    - Often better (up to 15% in experiments)
Target machines

- **Scalar**
  - 2.4 GHz Intel Pentium4 Xeon, 1GB
- **8-way SMP**
  - 4 dual-core 2.66GHz Intel P4 Xeons, 8GB
- **Disk**
  - 2.4 GHz Intel P4, 160GB disk, ~50MB/s from disk
- **Cluster**
  - 16, Intel 2.4GHz P4 Xeons, 1GB/node, Infiniband interconnect (780MB/s)
- **Cell**
  - 3.2 GHz IBM Cell blade (1 Cell - 8 SPE), 1GB
- **PS3**
  - 3.2 GHz Cell in Sony Playstation 3 (6 SPE), 256MB (160MB usable)

- **Cluster of SMPs**
  - Four 2-way, 3.16GHz Intel Pentium 4 Xeons connected via GigE (80MB/s peak)
- **Disk + PS3**
  - Sony Playstation 3 bringing data from disk (~30MB/s)
- **Cluster of PS3s**
  - Two Sony Playstation 3’s connected via GigE (60MB/s peak)
Port of Sequoia to Roadrunner

- Ported existing Sequoia runtimes: cluster and Cell
- Built new DaCS runtime
- Composition DaCS-Cell runtime

Current status of port:
- DaCS runtime works
- Currently adding composition: cluster-DaCS
- Developing benchmarks for Roadrunner runtime
Some initial benchmarks

- **Matrixmult**
  - 4K x 4K matrices
  - AB = C

- **Gravity**
  - 8192 particles
  - Particle-Particle stellar N-body simulation for 100 time steps

- **Conv2D**
  - 4096 x 8192 input signal
  - Convolution of 5x5 filter
Some initial benchmarks

- **Cell runtime timings**
  - Matrixmult: 112 Gflop/s
  - Gravity: 97.9 Gflop/s
  - Conv2D: 71.6 Gflop/s

- **Opteron reference timings**
  - Matrixmult: .019 Gflop/s
  - Gravity: .68 Gflop/s
  - Conv2D: .4 Gflop/s
DaCS-Cell runtime latency

- DaCS-Cell runtime performance of matrixmult
  - Opteron-Cell transfer latency
  - ~63 Gflop/s
  - ~40% of time spent in transfer from Opteron to PPU

- Cell runtime performance of matrixmult
  - No Opteron-Cell latency
  - 112 Gflop/s
  - Negligible time spent in transfer

- Computation / Communication ratio
  - Effected by the size of the matrices
  - As matrix size increases ratio improves
Plans: Roadrunner port

- Extend Sequoia support to full machine
- Develop solid benchmarks
- Collaborate with interested applications groups with time on full machine
Plans: Sequoia in general

- **Goal:** run on everything
- Currently starting Nvidia GPU port
- Language extensions to support dynamic, irregular computations
Questions?

http://sequoia.stanford.edu
Hierarchical memory

- Abstract machines as trees of memories

Dual-core PC

Main memory

ALUs

ALUs

Similar to:
Parallel Memory Hierarchy Model
(Alpern et al.)
## Sequoia Benchmarks

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

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

### FFT3D
- Complex single precision FFT

### Gravity
- 100 time steps of N-body stellar dynamics simulation ($N^2$) single precision

### HMMER
- Fuzzy protein string matching using HMM evaluation (Horn et al. SC2005 paper)

### SUmb
- Stanford University multi-block

Best available implementations used as leaf task
Best Known Implementations

- **HMMer**
  - ATI X1900XT: 9.4 GFlop/s
    (Horn et al. 2005)
  - Sequoia Cell: 12 GFlop/s
  - Sequoia SMP: 11 GFlop/s

- **Gravity**
  - Grape-6A: 2 billion interactions/s
    (Fukushige et al. 2005)
  - Sequoia Cell: 4 billion interactions/s
  - Sequoia PS3: 3 billion interactions/s
## Out-of-core Processing

<table>
<thead>
<tr>
<th></th>
<th>Scalar</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>0.3</td>
<td>0.007</td>
</tr>
<tr>
<td>SGEMV</td>
<td>1.1</td>
<td>0.04</td>
</tr>
<tr>
<td>SGEMM</td>
<td>6.9</td>
<td>5.5</td>
</tr>
<tr>
<td>CONV2D</td>
<td>1.9</td>
<td>0.6</td>
</tr>
<tr>
<td>FFT3D</td>
<td>0.7</td>
<td>0.05</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>4.8</td>
<td>3.7</td>
</tr>
<tr>
<td>HMMER</td>
<td>0.9</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Sequoia’s goals

- Portable, memory hierarchy aware programs

- Program to an abstract memory hierarchy
  - Explicit parallelism
  - Explicit, but abstract, communication
    - “move this data from here to there”
  - Large bulk transfers

- Compiler/run-time system
  - Instantiate program to a particular memory hierarchy
  - Take care of details of communication protocols, memory sizes, etc.
## Out-of-core Processing

<table>
<thead>
<tr>
<th></th>
<th>Scalar</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>0.3</td>
<td>0.007</td>
</tr>
<tr>
<td>SGEMV</td>
<td>1.1</td>
<td>0.04</td>
</tr>
<tr>
<td>SGEMM</td>
<td>6.9</td>
<td>5.5</td>
</tr>
<tr>
<td>CONV2D</td>
<td>1.9</td>
<td>0.6</td>
</tr>
<tr>
<td>FFT3D</td>
<td>0.7</td>
<td>0.05</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>4.8</td>
<td>3.7</td>
</tr>
<tr>
<td>HMMER</td>
<td>0.9</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Some applications have enough computational intensity to run from disk with little slowdown.
## Cluster vs. PS3

<table>
<thead>
<tr>
<th></th>
<th>Cluster</th>
<th>PS3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>4.9</td>
<td>3.1</td>
</tr>
<tr>
<td>SGEMV</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>SGEMM</td>
<td>91</td>
<td>94</td>
</tr>
<tr>
<td>CONV2D</td>
<td>24</td>
<td>62</td>
</tr>
<tr>
<td>FFT3D</td>
<td>5.5</td>
<td>31</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>68</td>
<td>71</td>
</tr>
<tr>
<td>HMMER</td>
<td>12</td>
<td>7.1</td>
</tr>
</tbody>
</table>

**Cost**
- Cluster: $150,000
- PS3: **$499**
Multi-Runtime Utilization

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

Cluster of SMPs | Disk + PS3 | Cluster of PS3s
Cluster of PS3 Issues

Cluster of SMPs | Disk + PS3 | Cluster of PS3s
Resource Utilization - IBM Cell

![Bar chart showing resource utilization for different tasks]

- **SAXPY**
- **SGEMV**
- **FFT3D**
- **SGEMM**
- **CONV2D**
- **GRAVITY**

**Y-axis:** Resource Utilization (%)

**Legend:**
- Bandwidth utilization
- Compute utilization
## Single Runtime Configurations - 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>4.9</td>
<td>3.5</td>
<td>3.1</td>
</tr>
<tr>
<td>SGEMV</td>
<td>1.1</td>
<td>1.7</td>
<td>0.04</td>
<td>12</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>SGEMM</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>0.7</td>
<td>3.9</td>
<td>0.05</td>
<td>5.5</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>
Cluster of PS3 Issues

Cluster of PS3s | PS3

Idle waiting on Xfer (M2-M1)  Idle waiting on Xfer (M1-M0)
Overhead (M2-M1)  Overhead (M1-M0)

SAXPY  SGEMV

Percentage of Runtime

Cluster of PS3s | PS3
## Multi-Runtime Configurations - 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>1.9</td>
<td>0.004</td>
<td>5.3</td>
</tr>
<tr>
<td>SGEMV</td>
<td>4.4</td>
<td>0.014</td>
<td>15</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>19</td>
</tr>
<tr>
<td>FFT3D</td>
<td>1.1</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>
## SMP vs. Cluster of SMP

<table>
<thead>
<tr>
<th></th>
<th>Cluster of SMPs</th>
<th>SMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>1.9</td>
<td>0.7</td>
</tr>
<tr>
<td>SGEMV</td>
<td>4.4</td>
<td>1.7</td>
</tr>
<tr>
<td>SGEMM</td>
<td>48</td>
<td>45</td>
</tr>
<tr>
<td>CONV2D</td>
<td>4.8</td>
<td>7.8</td>
</tr>
<tr>
<td>FFT3D</td>
<td>1.1</td>
<td>3.9</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>HMMER</td>
<td>14</td>
<td>11</td>
</tr>
</tbody>
</table>
## SMP vs. Cluster of SMP

<table>
<thead>
<tr>
<th></th>
<th>Cluster of SMPs</th>
<th>SMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>1.9</td>
<td>0.7</td>
</tr>
<tr>
<td>SGEMV</td>
<td>4.4</td>
<td>1.7</td>
</tr>
<tr>
<td>SGEMM</td>
<td>48</td>
<td>45</td>
</tr>
<tr>
<td>CONV2D</td>
<td>4.8</td>
<td>7.8</td>
</tr>
<tr>
<td>FFT3D</td>
<td>1.1</td>
<td>3.9</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>HMMER</td>
<td>14</td>
<td>11</td>
</tr>
</tbody>
</table>

Same number of total processors

Compute limited applications agnostic to interconnect
## Disk+PS3 Comparison

<table>
<thead>
<tr>
<th></th>
<th>Disk+PS3</th>
<th>PS3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SAXPY</strong></td>
<td>0.004</td>
<td>3.1</td>
</tr>
<tr>
<td><strong>SGEMV</strong></td>
<td>0.014</td>
<td>10</td>
</tr>
<tr>
<td><strong>SGEMM</strong></td>
<td>3.7</td>
<td>94</td>
</tr>
<tr>
<td><strong>CONV2D</strong></td>
<td>0.48</td>
<td>62</td>
</tr>
<tr>
<td><strong>FFT3D</strong></td>
<td>0.05</td>
<td>31</td>
</tr>
<tr>
<td><strong>GRAVITY</strong></td>
<td>66</td>
<td>71</td>
</tr>
<tr>
<td><strong>HMMER</strong></td>
<td>8.3</td>
<td>7.1</td>
</tr>
</tbody>
</table>
### Disk+PS3 Comparison

<table>
<thead>
<tr>
<th>Application</th>
<th>Disk+PS3</th>
<th>PS3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>0.004</td>
<td>3.1</td>
</tr>
<tr>
<td>SGEMV</td>
<td>0.014</td>
<td>10</td>
</tr>
<tr>
<td>SGEMM</td>
<td>3.7</td>
<td>94</td>
</tr>
<tr>
<td>CONV2D</td>
<td>0.48</td>
<td>62</td>
</tr>
<tr>
<td>FFT3D</td>
<td>0.05</td>
<td>31</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>66</td>
<td>71</td>
</tr>
<tr>
<td>HMMER</td>
<td>8.3</td>
<td>7.1</td>
</tr>
</tbody>
</table>

Some applications have enough computational intensity to run from disk with little slowdown.
## Disk+PS3 Comparison

<table>
<thead>
<tr>
<th></th>
<th>Disk+PS3</th>
<th>PS3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>0.004</td>
<td>3.1</td>
</tr>
<tr>
<td>SGEMV</td>
<td>0.014</td>
<td>10</td>
</tr>
<tr>
<td>SGEMM</td>
<td><strong>3.7</strong></td>
<td><strong>94</strong></td>
</tr>
<tr>
<td>CONV2D</td>
<td>0.48</td>
<td>62</td>
</tr>
<tr>
<td>FFT3D</td>
<td>0.05</td>
<td>31</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>66</td>
<td>71</td>
</tr>
<tr>
<td>HMMER</td>
<td>8.3</td>
<td>7.1</td>
</tr>
</tbody>
</table>

We can’t use large enough blocks in memory to hide latency.
## PS3 Cluster as a compute platform?

<table>
<thead>
<tr>
<th></th>
<th>PS3 Cluster</th>
<th>PS3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>5.3</td>
<td>3.1</td>
</tr>
<tr>
<td>SGEMV</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>SGEMM</td>
<td>30</td>
<td>94</td>
</tr>
<tr>
<td>CONV2D</td>
<td>19</td>
<td>62</td>
</tr>
<tr>
<td>FFT3D</td>
<td>0.36</td>
<td>31</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>119</td>
<td>71</td>
</tr>
<tr>
<td>HMMER</td>
<td>13</td>
<td>7.1</td>
</tr>
</tbody>
</table>
Avoiding latency stalls

- Exploit locality to minimize number of stalls
  - Example: Blocking / tiling
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)
Exploit locality

- Compute > bandwidth, else execution stalls
Locality in programming languages

- Local (private) vs. global (remote) addresses
  - UPC, Titanium

- Domain distributions (map array elements to location)
  - HPF, UPC, ZPL
  - Adopted by DARPA HPCS: X10, Fortress, Chapel

Focus on communication between nodes
Ignore hierarchy within a node
Locality in programming languages

- Streams and kernels
  - Stream data off chip. Kernel data on chip.
  - StreamC/KernelC, Brook
  - GPU shading (Cg, HLSL)

Architecture specific
Only represent two levels
Hierarchy-aware models

- Cache obliviousness (recursion)
- Space-limited procedures (Alpern et al.)

Programming methodologies, not programming environments
Hierarchical memory in Sequoia
Hierarchical memory

- Abstract machines as trees of memories

Dual-core PC

Main memory

L2 cache

L1 cache

ALUs

L1 cache

ALUs

4 node cluster of PCs

Aggregate cluster memory (virtual level)

Node memory

L2 cache

L1 cache

ALUs

Node memory

L2 cache

L1 cache

ALUs

Node memory

L2 cache

L1 cache

ALUs

Node memory

L2 cache

L1 cache

ALUs
Hierarchical memory

Single Cell blade

Main memory

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

Dual Cell blade

Main memory

(No memory affinity modeled)
Hierarchical memory

System with a GPU

Main memory

GPU memory

- tex
- L1
- 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.
}

C += A \times B
void matmul(int M, int N, int T, float* A, float* B, float* C) {

    Perform series of L2 matrix multiplications.
}

C += A x B
Sequoia tasks
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

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];
}

Calling task: matmul::inner
Located at level X

Callee task:
matmul::leaf
Located at level Y
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;

    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],
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

```cpp
void 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];
}

void 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);
}
```
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
Mapping tasks to machines
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

- **Main memory**
  - matmul::inner
    - M=N=T=1024
    - P=Q=R=256
  - 64 total subtasks

- **L2 cache**
  - matmul::inner
    - M=N=T=256
    - P=Q=R=32
  - 64 total subtasks

- **L1 cache**
  - matmul::leaf
    - M=N=T=32
  - 512 total subtasks
Task instances: Cell

Sequoia task definitions (parameterized)
- `matmul::inner`
- `matmul::leaf`

Cell mapping specification

```cpp
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
}
```

Cell task instances (not parameterized)
- `matmul_node_inst`
  - variant = inner
  - P=128, Q=64, R=128 (node level)
- `matmul_LS_inst`
  - variant = leaf
  - (LS level)
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
  - Scheduling (tasks and DMAs)

“Compilation for Explicitly Managed Memories”
Knight et al. To appear in 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

<table>
<thead>
<tr>
<th>Percentage of total execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle waiting on memory/network</td>
</tr>
<tr>
<td>Sequoia overhead</td>
</tr>
<tr>
<td>Leaf task computation</td>
</tr>
</tbody>
</table>

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

Percentage of total execution

Execution on a Cell blade

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

- 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 *</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>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 $\rightarrow$ 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
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