Halfway!

- First half of the course is over
  - Overview/Philosophy of Regent

- Now start the second half
  - Lectures on other programming models
  - Comparing/contrasting with Regent

- Start with an easy one today: Sequoia

Sequoia

CS315B
Lecture 9

A Point of View

- Parallelism is relatively easy
  - Not hard to find lots of parallelism in many apps

- The hard part is communication
  - Compute is easy
  - More difficult to ensure data is where it is needed

Sequoia

- Language: stream programming for machines with deep memory hierarchies

- Idea: Expose abstract memory hierarchy to programmer

- Implementation: benchmarks run well on many multi-level machines
  - Cell, PCs, clusters of PCs, cluster of PS3s, also + disk, GPUs
Locality

Structure algorithms as collections of independent and 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

Efficient programs exhibit this structure at many scales.

Sequoia's Goals

- Facilitate development of locality-aware programs ...
  ... that remain portable across machines

- Provide constructs that can be implemented efficiently
  - Place computation and data in machine
  - Explicit parallelism and communication
  - Large bulk transfers

Locality in Programming Languages

- Local (private) vs. global (remote) addresses
  - MPI (via message passing) UPC, Titanium

- Domain distributions
  - map array elements to locations
  - HPF, UPC, Titanium, ZPL
  - 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

**Blocked Matrix Multiplication**

```c
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];
}
```

`matmul_L1` 32x32 matrix mult

**Blocked Matrix Multiplication**

```c
void matmul_L2(int M, int N, int T,
                float* A,
                float* B,
                float* C)
{
    Perform series of L1 matrix multiplications.
}
```

`matmul_L2` 256x256 matrix mult

**Blocked Matrix Multiplication**

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

`matmul` large matrix mult
Hierarchical Memory

- Abstract machines as trees of memories

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

Dual Cell blade

Main memory

Hierarchical Memory

Cluster of dual Cell blades

Aggregate cluster memory (virtual level)

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];
}
```

Sequoia Tasks (Cont.)

- Sequence parameter passing semantics are not
  - Call by value
  - Call by name

- Rather
  - Copy-in, copy-out
  - Or Call-by-value-result

- Expresses the communication of arguments and results

Sequoia Tasks

- Task args & temporaries define 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];
}
```

Sequoia Tasks (Cont.)

- A task says *what* is copied

- Not *how* it is copied

- The latter is machine dependent
  - File operations for a disk
  - MPI operations for a cluster
  - DMAs for Cell processor
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

Tasks express multiple levels of parallelism
Leaf Variants

- Be practical: Permit 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);
}
```

Summary: Sequoia Tasks

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

- Sequoia programs describe hierarchies of tasks
  - Parameterized for portability

A Task Call

Generalizing Tasks
Abstraction vs. Reality

- The task hierarchy is abstract
- A task may have an unspecified number of sub-tasks
- The number of levels of sub-tasks may be unspecified
- Actual machines have limits in both dimensions

Mapping

Machine Descriptions

- A separate file describes each machine
  - The number of levels of memory hierarchy
  - The amount of memory at each level
  - The number of processors at each level
- This file is written once per machine
  - Use for each program compiled for that machine

Mappings

- A mapping file says how a particular program is mapped on to a specific machine
  - Settings for tunables
  - Degree of parallelism for each level
  - Whether to software pipeline compute/communication

```c
control(level 0)
{
  loop k[0]
  }
spmd {fullrange = 0.6; ways = 6; iterblk = 1;}
}
```
Compilation Overview

• The Sequoia compiler takes
  - A Sequoia program
  - A mapping file
  - A machine description

• Generates code for
  - All levels of the memory hierarchy
  - Glue to pass/return task arguments using appropriate communication primitives

Mapping Summary

• The abstract program must be made concrete for a particular machine

• Separate machine-specific parameters into:
  - Information that is common across programs
  - Machine descriptions
  - Information specific to a machine-program pair
  - Mapping files

• Mapping files can be (partially) automated

Sequoia Benchmarks

Blas Level 1 SAXPY, Level 2 SGEMV, and Level 3 SGEMM benchmarks
2D single precision convolution with 9x9 support (non-periodic boundary constraints)
Complex single precision FFT
100 time steps of N-body stellar dynamics simulation ($N^2$ single precision)
Fuzzy protein string matching using HMM evaluation (Horn et al. SC2005 paper)
Single Runtime System Configurations

- Scalar
  - 2.4 GHz Intel Pentium4 Xeon, 1GB
- 8-way SMP
  - 2.66GHz Intel P4 Xeon, 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)

Resource Utilization - IBM Cell

<table>
<thead>
<tr>
<th>Resource Utilization (%)</th>
<th>Bandwidth utilization</th>
<th>Compute utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SGEMM Performance

- Cluster
  - Intel Cluster MKL: 101 GFlop/s
  - Sequoia: 91 GFlop/s
- SMP
  - Intel MKL: 44 GFlop/s
  - Sequoia: 45 GFlop/s

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>SGEVM</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>8.9</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>58</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>

Prof. Aiken   CS 315B Lecture 9
FFT3D Performance

- **Cell**
  - Mercury Computer: 58 GFlop/s
  - FFTW 3.2 alpha 2: 35 GFlop/s
  - Sequoia: 54 GFlop/s
- **Cluster**
  - FFTW 3.2 alpha 2: 5.3 GFlop/s
  - Sequoia: 5.5 GFlop/s
- **SMP**
  - FFTW 3.2 alpha 2: 4.2 GFlop/s
  - Sequoia: 3.9 GFlop/s

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>

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

<table>
<thead>
<tr>
<th>Task</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>84</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 System Configurations

- **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)

### SMP vs. Cluster of SMP

<table>
<thead>
<tr>
<th>Task</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>Task</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
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>

Autotuner Results

<table>
<thead>
<tr>
<th></th>
<th>conv2d</th>
<th>sgemm</th>
<th>fft3d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell</td>
<td>auto</td>
<td>99.6</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td>hand</td>
<td>85</td>
<td>119</td>
</tr>
<tr>
<td></td>
<td>other</td>
<td></td>
<td>39(FFTW)</td>
</tr>
<tr>
<td>Cluster of PCs</td>
<td>auto</td>
<td>26.7</td>
<td>92.4</td>
</tr>
<tr>
<td></td>
<td>hand</td>
<td>24</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>other</td>
<td></td>
<td>101(MKL)</td>
</tr>
<tr>
<td>Cluster of PS3s</td>
<td>auto</td>
<td>20.7</td>
<td>33.4</td>
</tr>
<tr>
<td></td>
<td>hand</td>
<td>19</td>
<td>30</td>
</tr>
</tbody>
</table>

Sequoia Summary

- **Problem:**
  - Deep memory hierarchies pose perf. programming challenge
  - Memory hierarchy different for different machines

- **Solution:**
  - Hierarchical memory in the programming model
  - Program the memory hierarchy explicitly
  - Expose properties that affect performance

- **Approach: Express hierarchies of tasks**
  - Execute in local address space
  - Call-by-value-result semantics exposes communication
  - Parameterized for portability

Sequoia vs. Regent

- **Sequoia is static**
  - Mapping, task & data hierarchy
- **Regent is dynamic**
  - Structure & unstructured data

- **Structured data only**
- **One hierarchy for tasks & data**
- **Machine model is a memory hierarchy**
  - Separate task/data hierarchies
  - And multiple hierarchies for data!
  - Incurs additional complexity
  - E.g., privileges
  - Machine is a graph of processors & memories