Sequoia

Mike Houston
Stanford University
July 9th, 2007 - CScADS Workshop
Today’s outline

- Sequoia programming model
- Sequoia targets
- Tuning in Sequoia

http://sequoia.stanford.edu
  - Supercomputing 2006 paper - Language
  - PPoPP 2007 paper - Compiler
  - Runtime and backend system papers under review
  - Auto-tuning research/thesis in progress
Emerging Themes

Writing high-performance code amounts to…

Intelligently structuring algorithms
[compiler help unlikely]

Efficiently using parallel resources
[compilers struggle without help]

Generating efficient inner loops (kernels)
[compilers coming around]
It’s about program structure

- Preload batch of data
- Compute on data
- Initiate write of results (this data is done)
- Compute on next batch (which should be loaded)
Need “arithmetic intensity”

- Using data faster than it can be loaded causes stalls

<table>
<thead>
<tr>
<th>Time</th>
<th>Compute 1</th>
<th>Compute 2</th>
<th>Write output 0</th>
<th>Read input 2</th>
<th>Write output 1</th>
<th>Read input 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>stall</td>
<td>stall</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


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 bandwidth-efficient stream programs... that remain portable across a variety of machines

- Provide constructs that can be implemented efficiently without advanced compiler technology

- Get out of the way when needed
The idea

- Abstract machines a trees of memories (each memory is an address space)

**Dual-core PC**

![Diagram of Dual-core PC]

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

- Abstract machines a trees of memories

Dual-core PC

4 node cluster of PCs
The idea

Cell Processor Blade

Main memory

LS
ALUs
LS
ALUs
LS
ALUs
LS
ALUs
LS
ALUs
LS
ALUs
The idea

Cluster of dual-Cell blades

Main memory
The idea

System with a GPU

Main memory

GPU memory

ALUs

tex L1

ALUs

tex L1

ALUs

tex L1

ALUs

tex L1

ALUs

tex L1

ALUs

tex L1

ALUs

...
Memory model

- Explicit communication between abstract memories
- Locality awareness
- Hierarchy portability
  - Across machines, within levels of a machine
Sequoia tasks

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

```c
task interpolate(in float A[N],
                in float B[N],
                in float u,
                out float result[N])
{
    for (int i=0; i<N; i++)
        result[i] = u * A[i] + (1-u) * B[i];
}
```

- **Task arguments can be arrays and scalars**
- **Tasks arguments located within a single level of abstract memory hierarchy**
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 isolation

- Task args + temporaries define working set
- Task executes within private address space
- Subtask call induces change of address space

```
task foo(in float A[N], out float B[N])
{
    bar(A[0:N/2], B[0:N/2]);
    bar(A[N/2:N], B[N/2:N]);
}

task bar(in float A[N], out float B[N])
{
    ...
}
```
Task isolation

Locality
- Tasks express decomposition
Easy parallelism from isolation

- Task is granularity of parallelism
- Not cooperating threads
- Scheduling flexibility

```c
task parallel_foo(in float A[N], out float B[N])
{
    int X = 10;
    mappar(int i=0 to N/X) {
        bar( A[X*i : X*(i+1)], B[X*i : X*(i+1)] );
    }
}

task bar(in float A[N], out float B[N])
{
    ...
}
```
Communication

- Working set resident within single location in machine tree
- Data movement described by calling subtasks

```c
task parallel_foo(in float A[N], out float B[N])
{
    int X = 10;
    mappar(int i=0 to N/X) {
        bar( A[X*i : X*(i+1)], B[X*i : X*(i+1)] );
    }
}

task bar(in float A[N], out float B[N])
{
    ...
}
```

A and B in main memory N= unbounded
A and B in cache N = 10
Task parameterization

- Tasks are parameterized for adaptability
  - Allows tuning to each machine
    - “Auto” tuning possible
  - Task/algorithm description separate from machine mapping
- Allow multiple implementations (variants) of a single task
Example: dense matrix multiplication

- Task: 1024x1024 matrix multiplication
- Task: 256x256 matrix mult...
  - 64 total subtasks...
- Task: 256x256 matrix mult
- Task: 256x256 matrix mult...
- Task: 32x32 matrix mult...
  - 512 total subtasks...
- Task: 32x32 matrix mult
- Task: 32x32 matrix mult

Main memory

L2 cache

L1 cache
Example - task isolation

- Task arguments + local variables define working set
Example - parameterization

Tasks are written in parameterized form for portability

Different “variants” of the same task can be defined

Here is a “leaf version” of the matmul task. It doesn’t call subtasks.
Example - locality & communication

- Working set resident within single level of hierarchy
- Passing arguments to subtasks is only way to specify communication in Sequoia
Specializing matmul

- Instances of tasks placed at each memory level
  - Instances define a task variant and values for all parameters

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

```
... 64 total subtasks ...
```

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

```
... 512 total subtasks ...
```

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

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

Main memory

L2 cache

L1 cache
Task instances

Sequoia tasks

(mat parameterized)

matmul::inner

\( \text{matmul::leaf} \)

PC task instances

(mat 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)

Cell task instances

(mat parameterized)

(matmul_node_inst
  variant = inner
  P=128 Q=64 R=128
  node level)

(matmul_L2_inst
  variant = leaf
  LS level)
Sequoia methodology

- Express algorithms as machine independent parameterized tasks
  - structure provided explicitly from programmer

- Map tasks to hierarchical representation of a target machine

- Practical: use platform-specific kernel implementations
Leaf variants

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);
}
Results

- We have a Sequoia compiler + runtime systems for multiple platforms (runtimes actively being researched)
  - 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

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linear Algebra</strong></td>
<td>Blas Level 1 SAXPY, Level 2 SGEMV, and Level 3 SGEMM benchmarks</td>
</tr>
<tr>
<td><strong>Conv2D</strong></td>
<td>2D single precision convolution with 9x9 support (non-periodic boundary constraints)</td>
</tr>
<tr>
<td><strong>FFT3D</strong></td>
<td>Complex single precision FFT</td>
</tr>
<tr>
<td><strong>Gravity</strong></td>
<td>100 time steps of N-body stellar dynamics simulation (N^2) single precision</td>
</tr>
<tr>
<td><strong>HMMER</strong></td>
<td>Fuzzy protein string matching using HMM evaluation (Horn et al. SC2005 paper)</td>
</tr>
</tbody>
</table>
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 (1 Cell - 8SPE), 1GB

- **PS3**
  - 3.2 GHz Cell in Sony Playstation 3 (6 SPE), 256MB (160MB usable)
2 Level Utilization

SMP, Disk, Cluster, Cell, PS3 left to right for each application
Applications: saxpy, sgemv, sgemm, conv2d, fft3d, gravity, HMMer
## 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>SGEMV</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>SGEMM</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 $128^3$ 3D FFT, random 160MB subset of NCBI
## 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>SGEMV</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>SGEMM</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 $128^3$ 3D FFT, random 160MB subset of NCBI

**Bandwidth bound**
Composed systems utilization

Cluster of SMPs, Disk and PS3, cluster of PS3s left to right for each application
Applications: 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>Function</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>

Bandwidth bound
Cell utilization

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

<table>
<thead>
<tr>
<th>Resource</th>
<th>DRAM Utilization (%)</th>
<th>SPE Utilization (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>95%</td>
<td>10%</td>
</tr>
<tr>
<td>SGEMV</td>
<td>98%</td>
<td>20%</td>
</tr>
<tr>
<td>FFT3D</td>
<td>90%</td>
<td>50%</td>
</tr>
<tr>
<td>SGEMM</td>
<td>85%</td>
<td>80%</td>
</tr>
<tr>
<td>CONV2D</td>
<td>70%</td>
<td>70%</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>100%</td>
<td>90%</td>
</tr>
</tbody>
</table>
Performance scaling

SPE scaling on 2.4GHz Dual-Cell blade

Scaling on P4 cluster with Infiniband interconnect

Graphs showing speedup for different operations as a function of the number of SPEs and nodes.
Key ideas

- Incorporate hierarchal memory tightly into programming model
  - Programming memory hierarchy

- Abstract [horizontal + vertical] communication and locality
  - Vertical portability

- Leverage task abstraction for critical properties of application
Auto-tuning

- Sequoia provides a framework for tuning
  - Algorithm definition separate from mapping!
- Programmer defines search space
  - Tunables
  - Variants
- Machine model defined by system architect
- Map algorithm to machine via searching the tunable and variant space
Current status

- Profiling driven optimization
  - Set tunables, run, retune, rerun, ...
- Greedy search for kernel fusion
- Greedy search for tunables
  - Bottom up - Set L0, tune, move to L1 tuning
  - Obviously non-optimal
- Brute force variant search
  - Try all variants at a level if mappable to that level
Sequoia tuning overview

- AST
  - Choose Variant
  - Set Memory/Control Level
  - MITT
    - Loop Fusion
    - Loop Interchange/Split/Align
    - Array SSA
    - Circular Buffer
  - MITT
    - Set Tunables
  - Machine Description

- Search Engine

- Compiler Passes
- Code Generation
- Code profiling
- GSOP
  - IR Lowering
  - Scheduling
Conv2D tuning on Cell (5x5 window)
SGEMM tuning Cell

![Graph showing Gflop/s vs Search iteration](image-url)
Future work

- **Runtime system release**
  - SMP, Cluster, Disk, Mercury CAB
  - Cray XT3/4
  - Roadrunner
  - BG/L, BG/P (?)

- **More complex algorithms**
  - Sweep3D
  - SPaSM/GROMACS
  - SUmb/FEM
  - MG

- **Auto-tuning**
  - Researching variant space search
  - Better search techniques
  - Kernel fusion tricky
Questions?