Sequoia

Mattan Erez

The University of Texas at Austin
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’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
Sequoia

• Language: generalized 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
• Key challenge in high performance programming is:

• communication (not parallelism)

• Latency
• Bandwidth
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
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
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)
  - CUDA

Architecture specific
Only represent two levels
Hierarchy-aware models

- Cache obliviousness (recursion)

- Space-limited procedures (Alpern et al.)

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

- Abstract machines as trees of memories

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

- Abstract machines as trees of memories
Hierarchical memory
Hierarchical memory
Hierarchical memory

Aggregate cluster memory (virtual level)

Main memory

Main memory
Hierarchical memory

Disk

Main memory

ALUs

LS

ALUs

LS

ALUs

LS

ALUs

LS

ALUs

LS

ALUs

LS

ALUs

LS

ALUs

LS

ALUs

LS

ALUs

LS

ALUs

LS

ALUs
Hierarchical memory

Main memory

GPU memory

SM

SM

SM

SM

SM

SM

SM

ALUs

ALUs

ALUs

ALUs

ALUs

ALUs

ALUs

ALUs

ALUs
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];
}
```

\[
\text{C} += \text{A} \times \text{B}
\]
Blocked matrix multiplication

\[
C += A \times B
\]

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

© Mattan Erez
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’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
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

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

• Tasks operate on arrays, not array elements

• Tasks nest: they call subtasks
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 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 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 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 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] );
        }
    }
Leaf variants

- Be practical: Can use 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);
}
```

© Mattan Erez
Synchronization

• *mapseq* implies sync at end of every iteration
• *mappar* implies sync at end of iteration space

• No explicit synchronization

• 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

• Bulk synchronous / Sequoia more robust than interacting threads
Synchronization Impacts Parallelism

- Parallelism explicitly expressed using mappable
  - 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 mappable
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 (parameterized)

matmul::inner

matmul::leaf

Mapping specification

```python
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 (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
Task mapping specification

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

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

matmul::inner
M=N=T=256
P=Q=R=32

matmul::leaf
M=N=T=32

... 64 total subtasks ...

matmul::inner
M=N=T=256
P=Q=R=32

matmul::leaf
M=N=T=32

... 512 total subtasks ...

matmul::inner
M=N=T=256
P=Q=R=32

matmul::leaf
M=N=T=32

... 512 total subtasks ...
```
Task instances: Cell

Sequoia task definitions (parameterized)

matmul::inner

matmul::leaf

Cell mapping specification

```
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
Autotuning Sequoia Programs

- Autotuner helps user with mapping (user can always override)
Specialization with Autotuning

• Work by Manman Ren (Stanford), PACT 2008
• Use Sequoia to identify what needs tuning
  – Explicit tunables and parameters in the language
• Tuning framework for SW-managed hierarchies
• Automatic profile guided search across tunables
  – Aggressive pruning
  – Illegal parameters (don’t fit in memory level)
  – Tunable groups
  – Programmer input on ranges
  – Coarse → fine search
• Loop fusion across multiple loop levels
  – Measure profitability from tunable search
  – Adjust for “tunable mismatch”
  – Realign reuse to reduce communication
Overview: mapping the program

- Mapped versions are generated
  - Matching the decomposition hierarchy with the machine hierarchy
  - Choosing a variant for each call site
  - Set level of data objects and control statements
Explicit SW Management Simplifies Tuning

Conv2d on Cell

- Smooth search space
- Performance models can also work
  - For Cell, not cluster

© Mattan Erez
### Autotuning OutPerforms Programmer

<table>
<thead>
<tr>
<th></th>
<th>CONV2D</th>
<th>SGEMM</th>
<th>FFT3D</th>
<th>SUmb</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cell</strong></td>
<td>auto</td>
<td>99.6</td>
<td>137</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>hand</td>
<td>85</td>
<td>119</td>
<td>54</td>
</tr>
<tr>
<td><strong>Cluster</strong></td>
<td>auto</td>
<td>26.7</td>
<td>92.4</td>
<td>5.5</td>
</tr>
<tr>
<td>of PCs</td>
<td>hand</td>
<td>24</td>
<td>90</td>
<td>5.5</td>
</tr>
<tr>
<td><strong>Cluster</strong></td>
<td>auto</td>
<td>20.7</td>
<td>33.4</td>
<td>0.57</td>
</tr>
<tr>
<td>of PS3s</td>
<td>hand</td>
<td>19</td>
<td>30</td>
<td>0.36</td>
</tr>
</tbody>
</table>

- CONV2D
- SGEMM
- FFT3D
- SUmb
Summary of results

• Performance competitive with native code
  – Generic optimization for hierarchical bulk programs
    • Copy elimination
    • DMA transfer coalescing
    • Operation hoisting
    • Array allocation / packing
    • Scheduling (tasks and DMAs)
  – Automatic tuning for performance

• Portable: no source-code changes for different configurations
  – Cell, SMP, Cluster, Disk
  – Compositions of above
  – Automatic tuning

• Maximizes resources (compute or communication)
• Low overhead
Results

• We have a Sequoia compiler + runtime systems for multiple platforms
  – 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 convolution with 9x9 support (non-periodic boundary constraints)</td>
</tr>
<tr>
<td><strong>FFT3D</strong></td>
<td>$256^3$ complex FFT</td>
</tr>
<tr>
<td><strong>Gravity</strong></td>
<td>100 time steps of N-body stellar dynamics simulation</td>
</tr>
<tr>
<td><strong>HMMER</strong></td>
<td>Fuzzy protein string matching using HMM evaluation (Daniel Horn’s 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 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>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>1.4</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>3.8</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>1.5</td>
<td>7.8</td>
<td>0.1</td>
<td>7.5</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>

* Reduced dataset size to fit in memory
## 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>SAXPY</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>SGEMV</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>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>1.5</td>
<td>7.8</td>
<td>0.1</td>
<td>7.5</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>

* Reduced dataset size to fit in memory

**Bandwidth bound**
2 Level Utilization

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

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></th>
<th>Cluster-SMP</th>
<th>Disk+PS3</th>
<th>PS3 Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SAXPY</strong></td>
<td>0.5</td>
<td>0.004</td>
<td>0.23</td>
</tr>
<tr>
<td><strong>SGEMV</strong></td>
<td>1.4</td>
<td>0.014</td>
<td>1.3</td>
</tr>
<tr>
<td><strong>SGEMM</strong></td>
<td>48</td>
<td>3.7</td>
<td>30</td>
</tr>
<tr>
<td><strong>CONV2D</strong></td>
<td>4.8</td>
<td>0.48</td>
<td>3.24</td>
</tr>
<tr>
<td><strong>FFT3D</strong></td>
<td>2.1</td>
<td>0.05</td>
<td>0.36</td>
</tr>
<tr>
<td><strong>GRAVITY</strong></td>
<td>50</td>
<td>66</td>
<td>119</td>
</tr>
<tr>
<td><strong>HMMER</strong></td>
<td>14</td>
<td>8.3</td>
<td>13</td>
</tr>
</tbody>
</table>

*Bandwidth bound*
Composed systems utilization

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

Percentage application run-time

Applications: AXPY, SEMV, SEMM, CPU2, FFT3D, RAQITY, HMMER

© Mattan Erez
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></th>
<th>Resource utilization (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPY</td>
<td>100%</td>
</tr>
<tr>
<td>SGEMV</td>
<td>90%</td>
</tr>
<tr>
<td>FFT3D</td>
<td>80%</td>
</tr>
<tr>
<td>SGEMM</td>
<td>70%</td>
</tr>
<tr>
<td>CONV2D</td>
<td>60%</td>
</tr>
<tr>
<td>GRAVITY</td>
<td>50%</td>
</tr>
</tbody>
</table>
Performance scaling

SPE scaling on 2.4GHz Dual-Cell blade

Scaling on P4 cluster with Infiniband interconnect

[Graphs showing speedup vs. number of SPEs and number of nodes for various operations such as SAXPY, SGEMV, SGEMM, Conv2D, FFT3D, Gravity, and HMMER]
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

• Mapping to specific machine is separate from algorithm description
Sequoia limitations

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

- Task mapping somewhat laborious
  - Autotuning helps
  - Understand which parts can be automated better

- Leaf-tasks (kernels) are external
  - Limits potential optimizations
  - Cumbersome