Sequoia

CS315B
Lecture 10

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

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

Locality

Structure 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 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

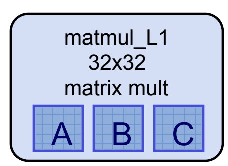
- Streams and kernels
  - Stream data off chip. Kernel data on chip.
  - StreamC/KernelC, Brook
  - GPU shading (Cg, HLSL)

Architecture specific

Only represent two levels

Blocked Matrix Multiplication

```
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];
}
```

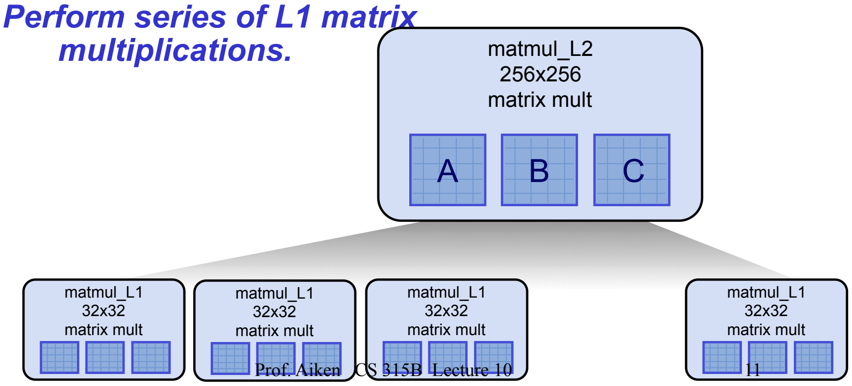


Blocked Matrix Multiplication

```
void matmul_L2( int M, int N, int T,
                float* A,
                float* B,
                float* C)
{
```

Perform series of L1 matrix multiplications.

```
}
```

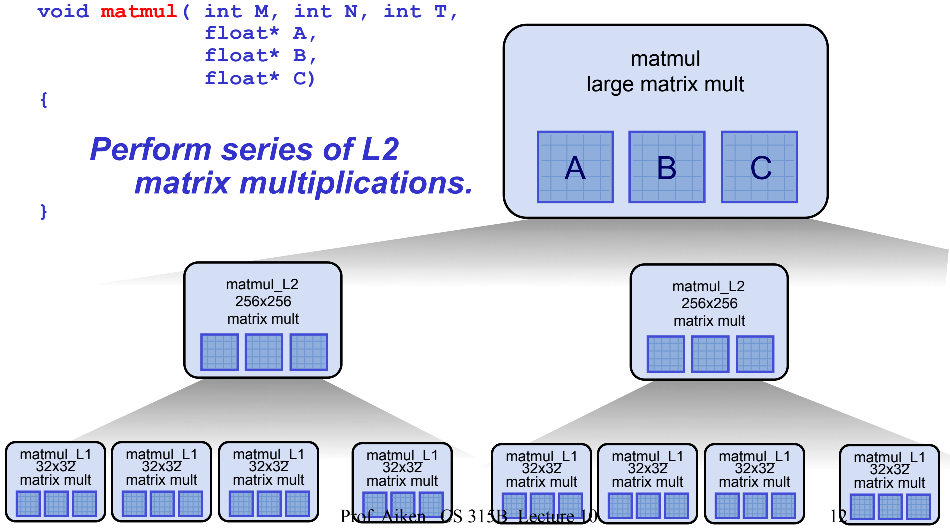


Blocked Matrix Multiplication

```
void matmul( int M, int N, int T,
             float* A,
             float* B,
             float* C)
{
```

Perform series of L2 matrix multiplications.

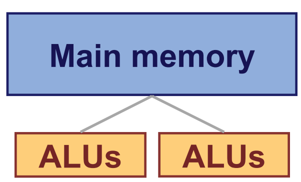
```
}
```



Hierarchical Memory

Hierarchical Memory

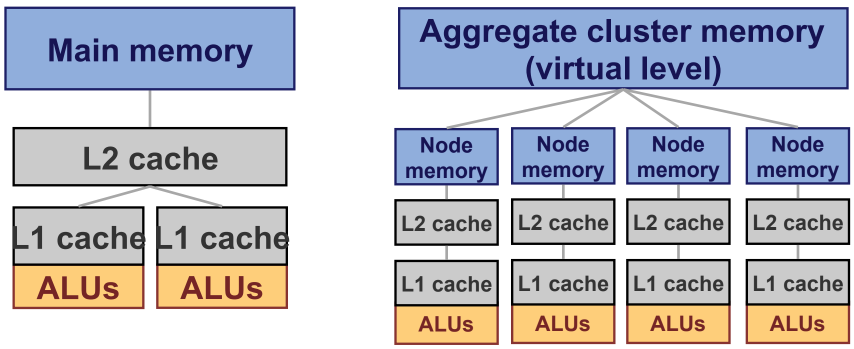
- Abstract machines as trees of memories



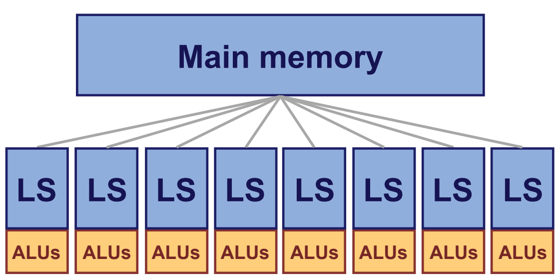
Similar to:
Parallel Memory Hierarchy Model
(Alpern et al.)

Hierarchical Memory

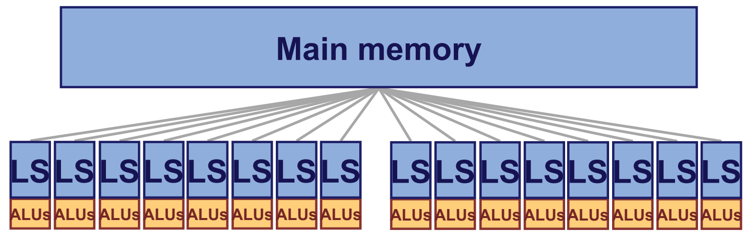
- Abstract machines as trees of memories



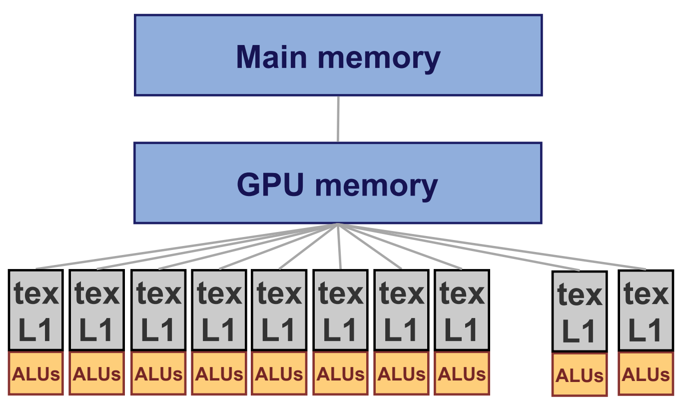
Hierarchical Memory



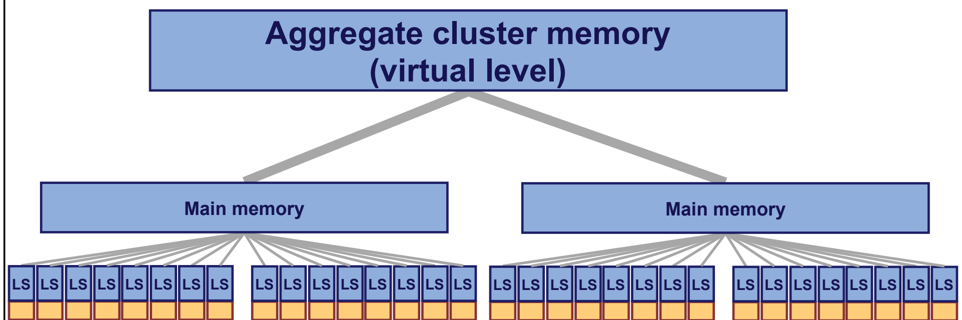
Hierarchical Memory



Hierarchical Memory



Hierarchical Memory



Tasks

Sequoia Tasks

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

```
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

- Task args & temporaries define working set
- Task working set resident at single location in abstract machine tree

```
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.)

- Sequoia 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 (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

```
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

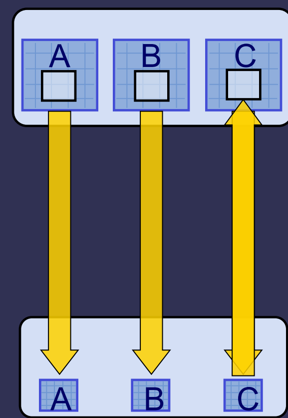
```
task matmul::inner( in    float A[M][T],
                    in    float B[T][N],
                    inout float C[M][N] )
{
  tunable int P, Q, R;

  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];
}
```

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;

  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: Permit platform-specific kernels

```
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);
}
```

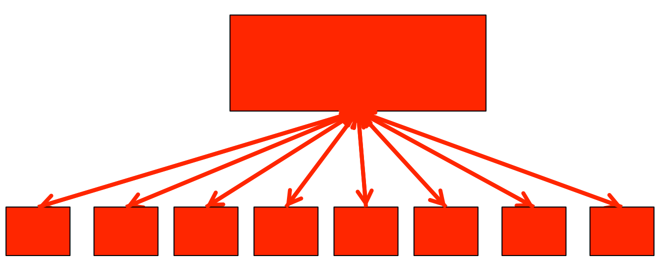

Summary: Sequoia Tasks

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

- Sequoia programs describe hierarchies of tasks
  - Parameterized for portability

Generalizing Tasks

A Task Call



Mapping

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

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

```
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

Performance Results

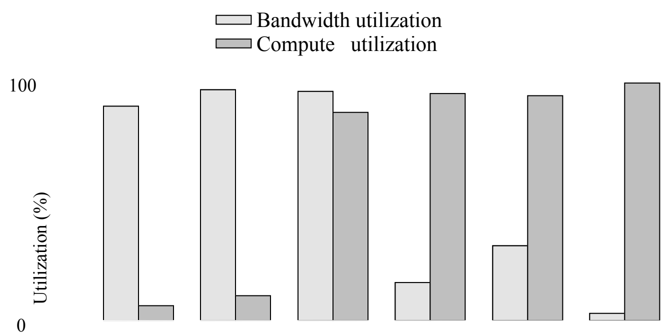
Sequoia Benchmarks

- 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
  - 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)

Resource Utilization – IBM Cell



Single Runtime Configurations - GFlop/s

	Scalar	SMP	Disk	Cluster	Cell	PS3
SAXPY	0.3	0.7	0.007	4.9	3.5	3.1
SGEMV	1.1	1.7	0.04	12	12	10
SGEMM	6.9	45	5.5	91	119	94
CONV2D	1.9	7.8	0.6	24	85	62
FFT3D	0.7	3.9	0.05	5.5	54	31
GRAVITY	4.8	40	3.7	68	97	71
HMMER	0.9	11	0.9	12	12	7.1

SGEMM Performance

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

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

	Scalar	Disk
SAXPY	0.3	0.007
SGEMV	1.1	0.04
SGEMM	6.9	5.5
CONV2D	1.9	0.6
FFT3D	0.7	0.05
GRAVITY	4.8	3.7
HMMER	0.9	0.9

Out-of-core Processing

	Scalar	Disk
SAXPY	0.3	0.007
SGEMV	1.1	0.04
SGEMM	6.9	5.5
CONV2D	1.9	0.6
FFT3D	0.7	0.05
GRAVITY	4.8	3.7
HMMER	0.9	0.9

Some applications have enough computational intensity to run from disk with little slowdown

Cluster vs. PS3

	Cluster	PS3
SAXPY	4.9	3.1
SGEMV	12	10
SGEMM	91	94
CONV2D	24	62
FFT3D	5.5	31
GRAVITY	68	71
HMMER	12	7.1

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

	Cluster of SMPs	SMP
SAXPY	1.9	0.7
SGEMV	4.4	1.7
SGEMM	48	45
CONV2D	4.8	7.8
FFT3D	1.1	3.9
GRAVITY	50	40
HMMER	14	11

SMP vs. Cluster of SMP

	Cluster of SMPs	SMP
SAXPY	1.9	0.7
SGEMV	4.4	1.7
SGEMM	48	45
CONV2D	4.8	7.8
FFT3D	1.1	3.9
GRAVITY	50	40
HMMER	14	11

Same number of total processors

Compute limited applications agnostic to interconnect

PS3 Cluster as a Compute Platform?

	PS3 Cluster	PS3
SAXPY	5.3	3.1
SGEMV	15	10
SGEMM	30	94
CONV2D	19	62
FFT3D	0.36	31
GRAVITY	119	71
HMMER	13	7.1

Autotuner Results

		conv2d	sgemm	fft3d
Cell	**auto**	99.6	137	57
	hand	85	119	54
	other			39(FFTW)
				46.8(IBM)
Cluster of PCs	**auto**	26.7	92.4	5.5
	hand	24	90	5.5
	other		101(MKL)	5.3(FFTW)
Cluster of PS3s	**auto**	20.7	33.4	0.57
	hand	19	30	0.36

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
- Structured data only
- One hierarchy for tasks & data
- Machine model is a memory hierarchy

- Regent is dynamic
- Structure & unstructured data
- Separate task/data hierarchies
  - And multiple hierarchies for data!
  - Incurs additional complexity
  - E.g., privileges
- Machine is a graph of processors & memories