Sequoia

Programming the Memory Hierarchy

Kayvon Fatahalian, Timothy J. Knight, Mike Houston, Mattan Erez
Daniel Reiter Horn, Larkhoon Leem, Ji Young Park, Manman Ren
Alex Aiken, William J. Dally, Pat Hanrahan, 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

Computation	Communication
▪ Cluster of 3264 nodes	Infiniband
▪ … a node has 2 chips	Infiniband
▪ … a chip has 2 Opterons	Shared memory
▪ … an Opteron has a Cell	DACS
▪ … a Cell has 8 SPEs	Cell API

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

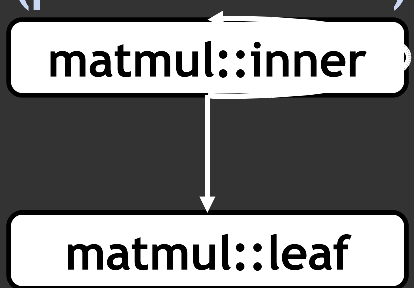
- Special functions called **tasks** are the building blocks of Sequoia programs

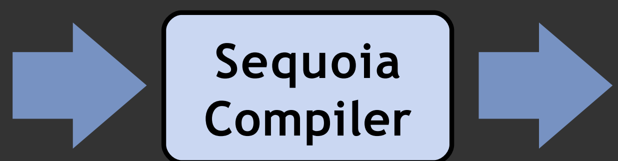
```
task matmul::leaf( in    float A[M][T],
                   in    float B[T][N],
                   inout float C[M][N]  )
{
   for (int i=0; i<M; i++)
     for (int j=0; j<N; j++)
       for (int k=0; k<T; k++)
          C[i][j] += A[i][k] * B[k][j];
}
```

Read-only parameters M, N, T give sizes of multidimensional arrays when task is called.

How mapping works

Sequoia task definitions (parameterized)





Task instances

Mapping specification

```
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
}
```

Runtime system

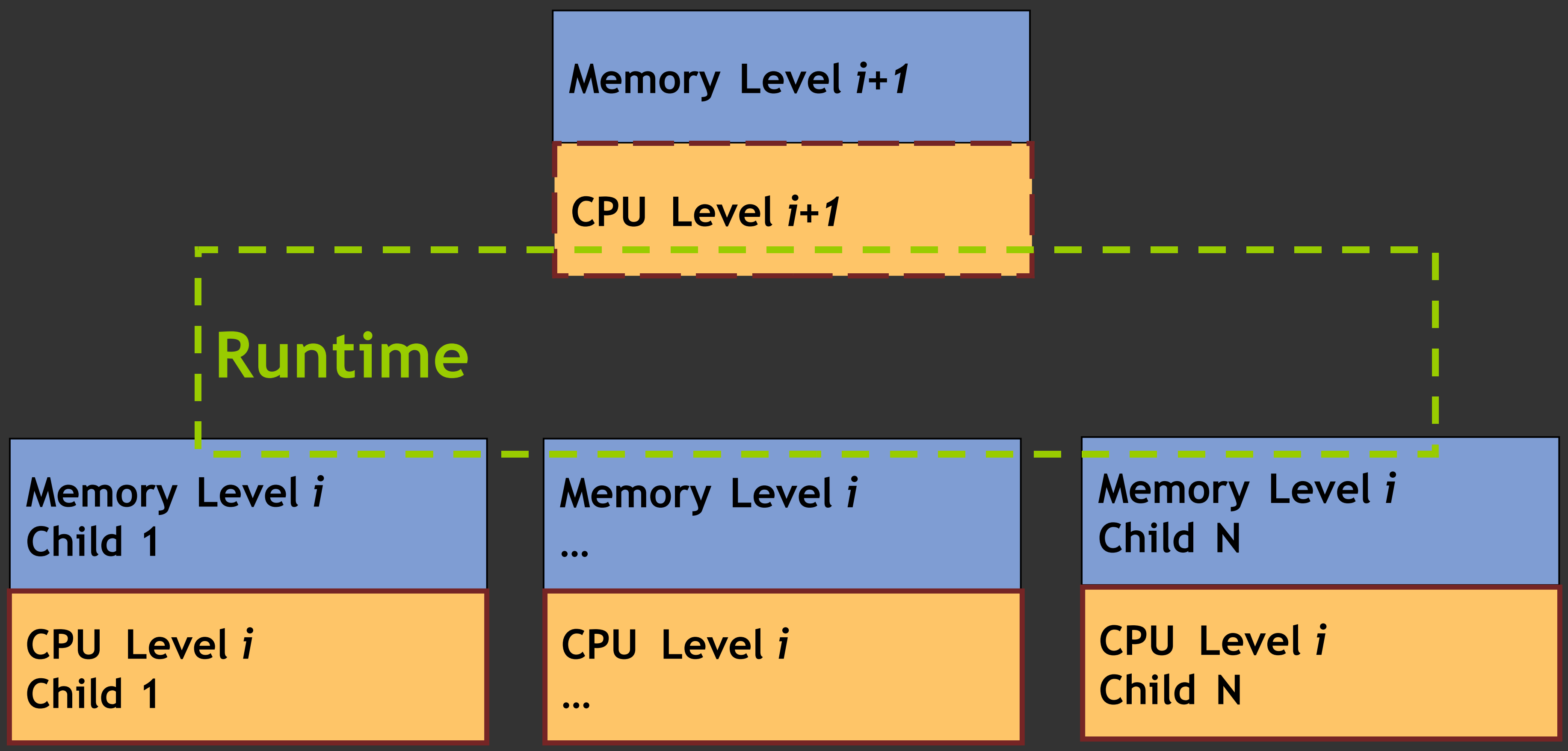
- A runtime implements one memory level
  - Simple, portable API interface
  - Handles naming, synchronization, communication
  - For example Cell runtime abstracts DMA

- 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



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 compostion: 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

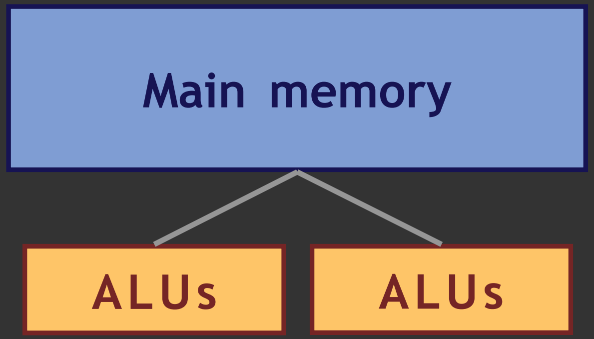
Questions?

http://sequoia.stanford.edu

Hierarchical memory

- Abstract machines as trees of memories

Dual-core PC



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

	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

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

	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

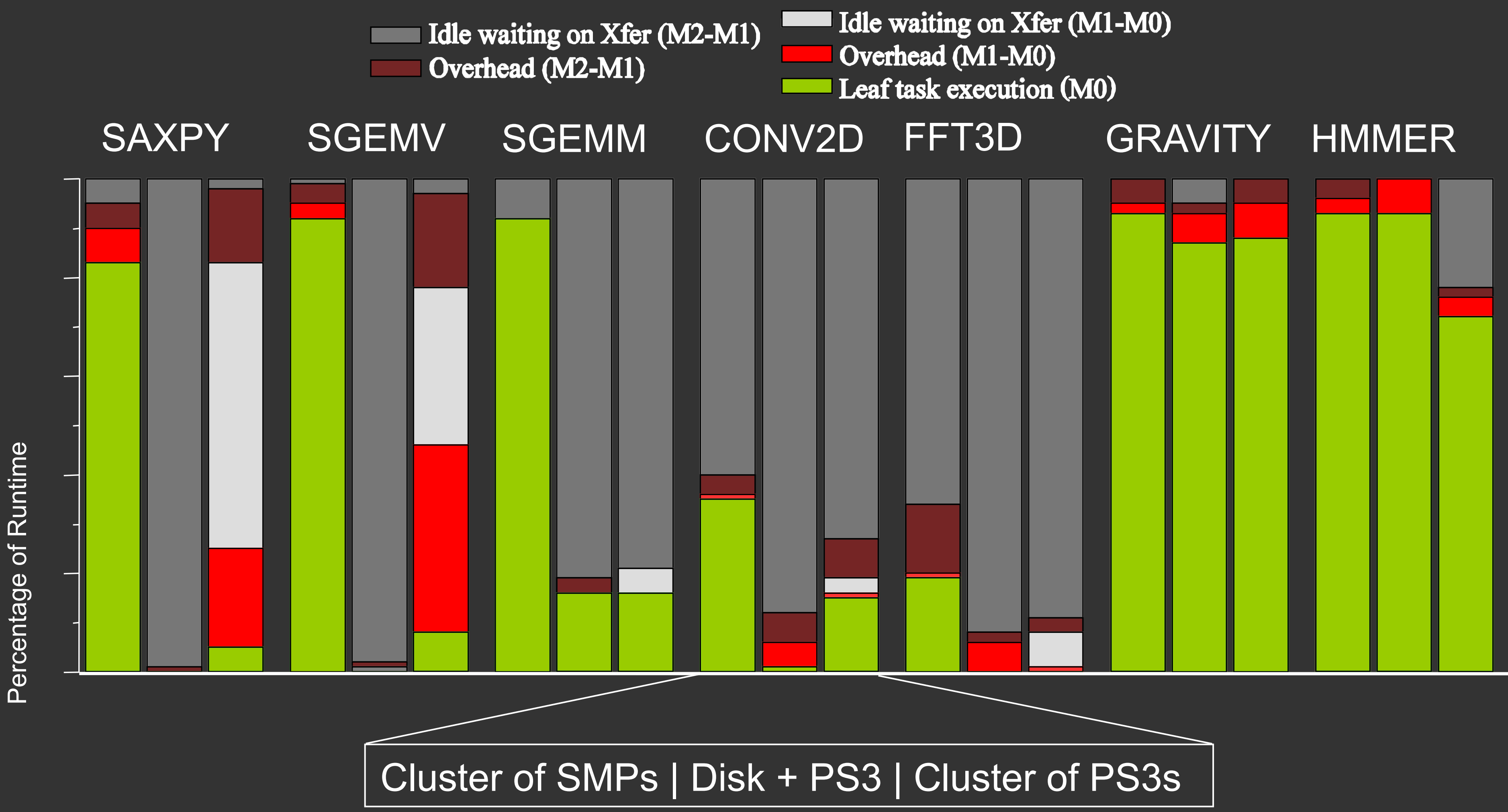
	Cluster	PS3
SAXPY	4.9	3.1
SGEMV	12	10
SGEMM	91	94
CONV2D	24	62
FFT3D	5.5	31
GRAVITY	68	71
HMMER	12	7.1

Cost

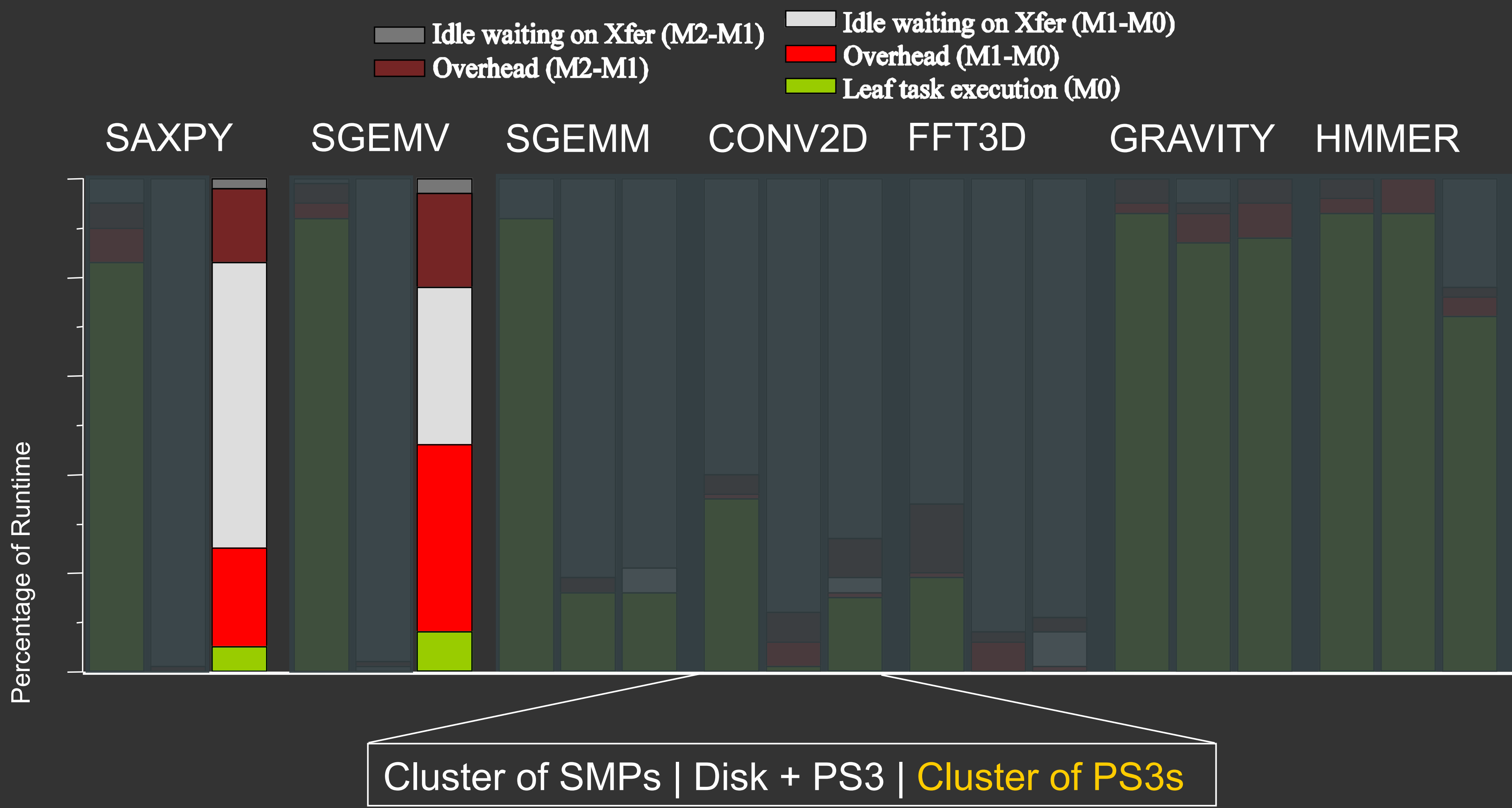
Cluster: $150,000

PS3: $499

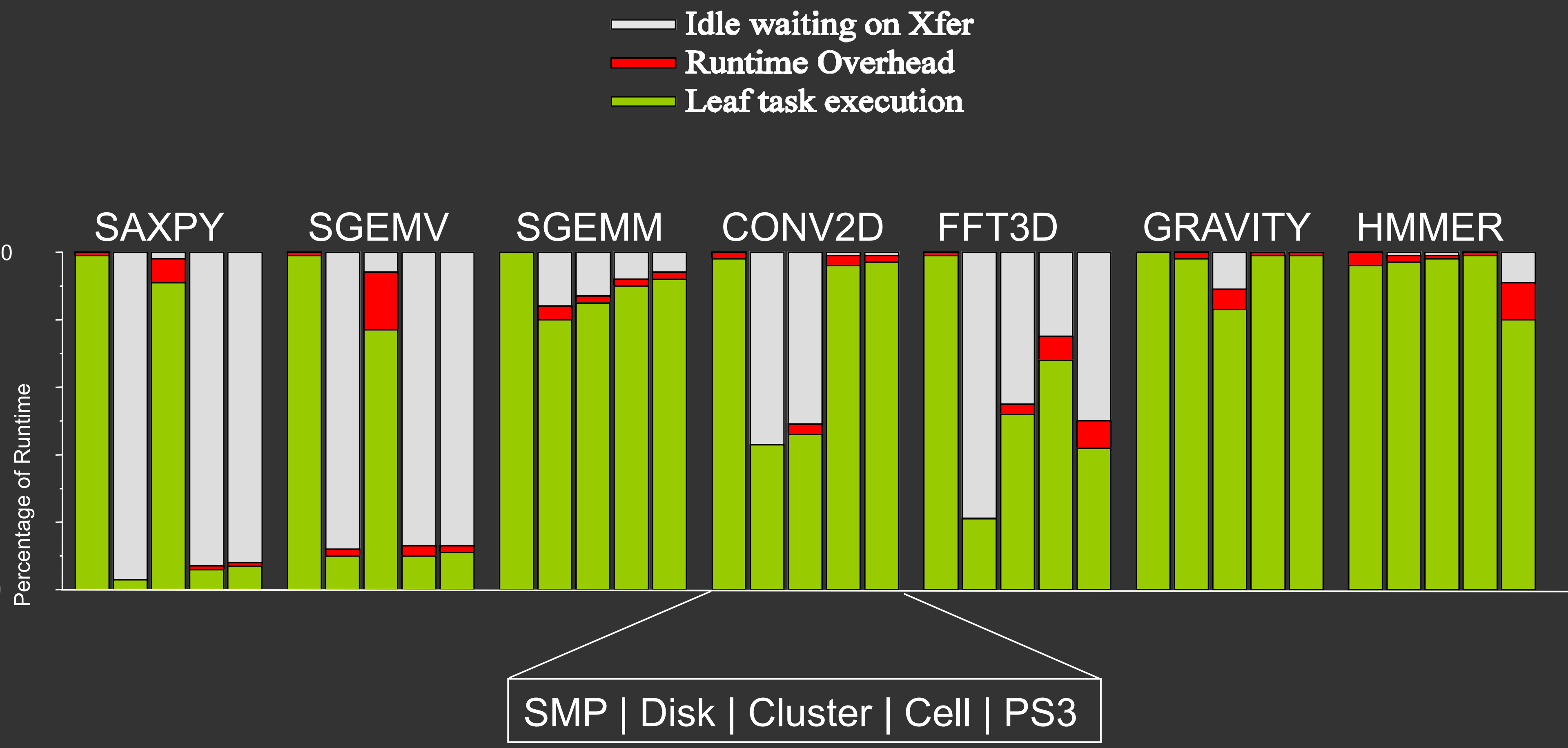
Multi-Runtime Utilization



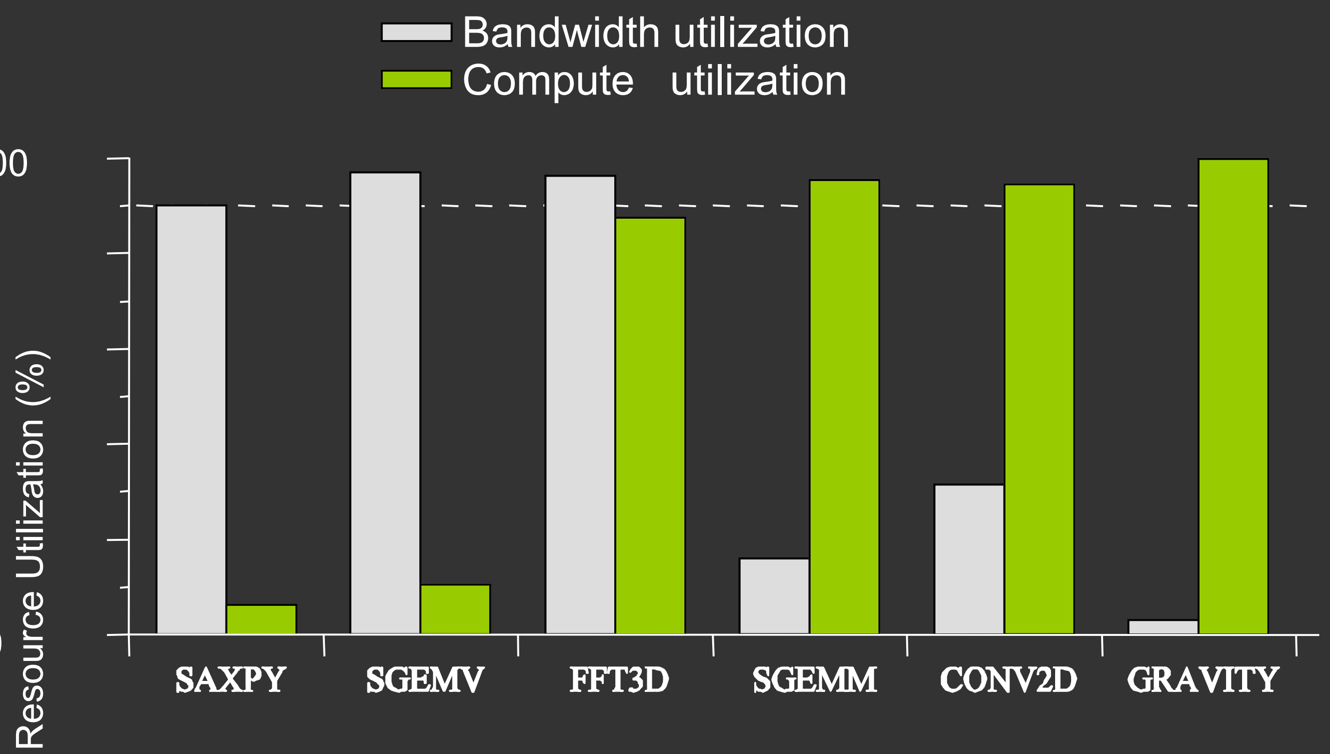
Cluster of PS3 Issues



System Utilization



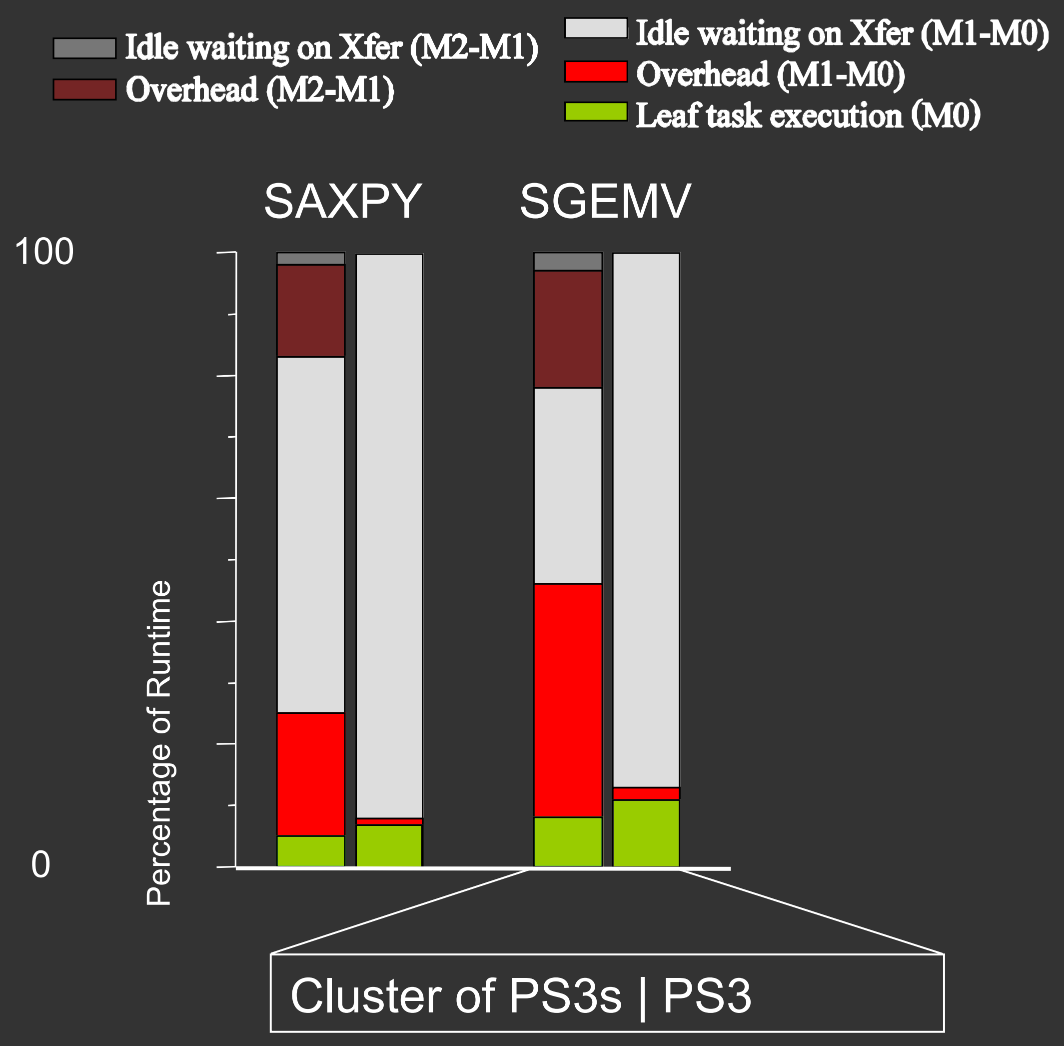
Resource Utilization – IBM Cell



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

Cluster of PS3 Issues



Multi-Runtime Configurations - GFlop/s

	Cluster-SMP	Disk+PS3	PS3 Cluster
SAXPY	1.9	0.004	5.3
SGEMV	4.4	0.014	15
SGEMM	48	3.7	30
CONV2D	4.8	0.48	19
FFT3D	1.1	0.05	0.36
GRAVITY	50	66	119
HMMER	14	8.3	13

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

Disk+PS3 Comparison

	Disk+PS3	PS3
SAXPY	0.004	3.1
SGEMV	0.014	10
SGEMM	3.7	94
CONV2D	0.48	62
FFT3D	0.05	31
GRAVITY	66	71
HMMER	8.3	7.1

Disk+PS3 Comparison

	Disk+PS3	PS3
SAXPY	0.004	3.1
SGEMV	0.014	10
SGEMM	3.7	94
CONV2D	0.48	62
FFT3D	0.05	31
GRAVITY	66	71
HMMER	8.3	7.1

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

Disk+PS3 Comparison

	Disk+PS3	PS3
SAXPY	0.004	3.1
SGEMV	0.014	10
SGEMM	3.7	94
CONV2D	0.48	62
FFT3D	0.05	31
GRAVITY	66	71
HMMER	8.3	7.1

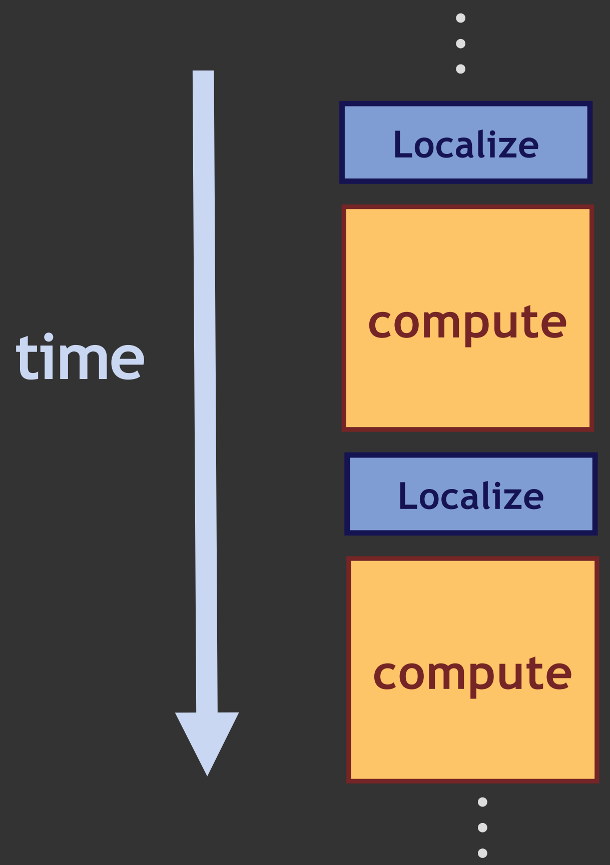
We can't use large enough blocks in memory to hide latency

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

Avoiding latency stalls

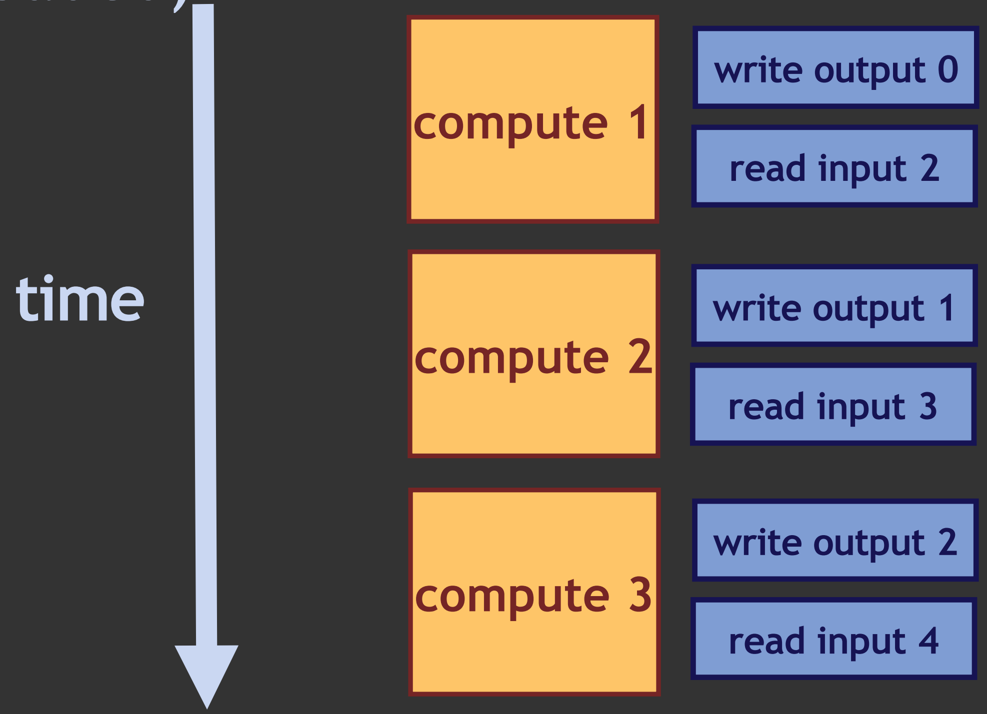
- Exploit locality to minimize number of stalls
  - Example: Blocking / tiling



Avoiding latency stalls

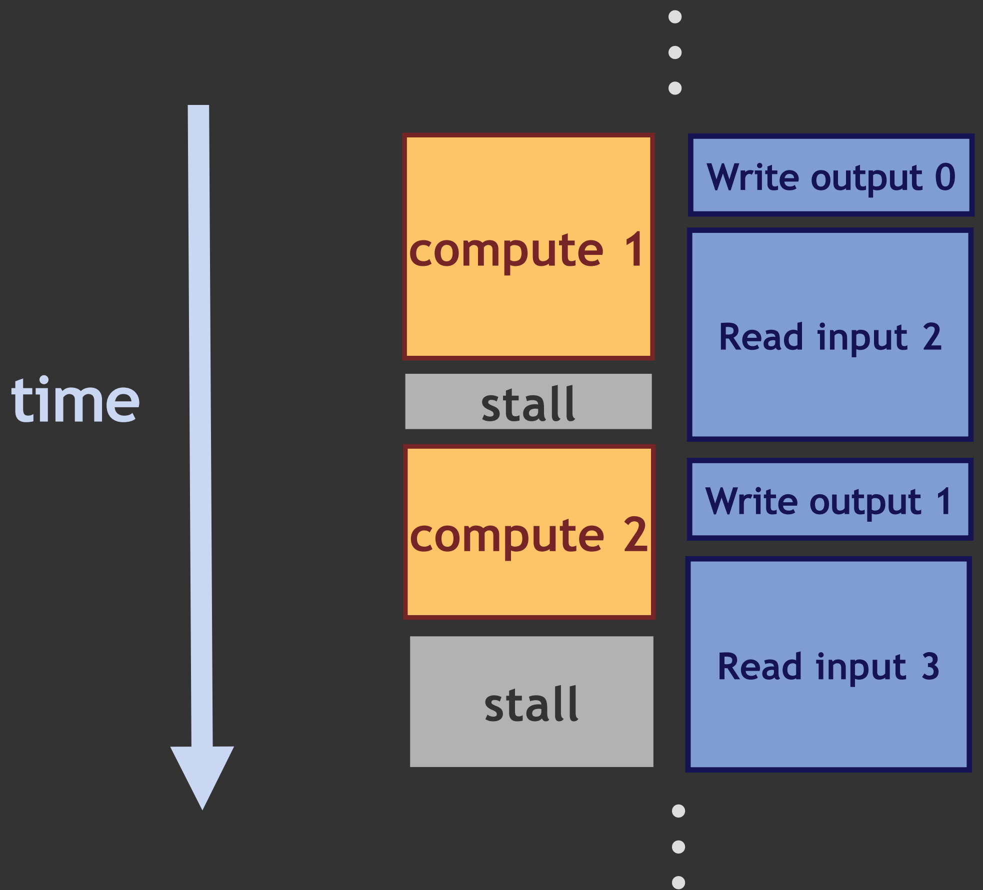
1. Prefetch batch of data
2. Compute on data (avoiding stalls)
3. Initiate write of results

… 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

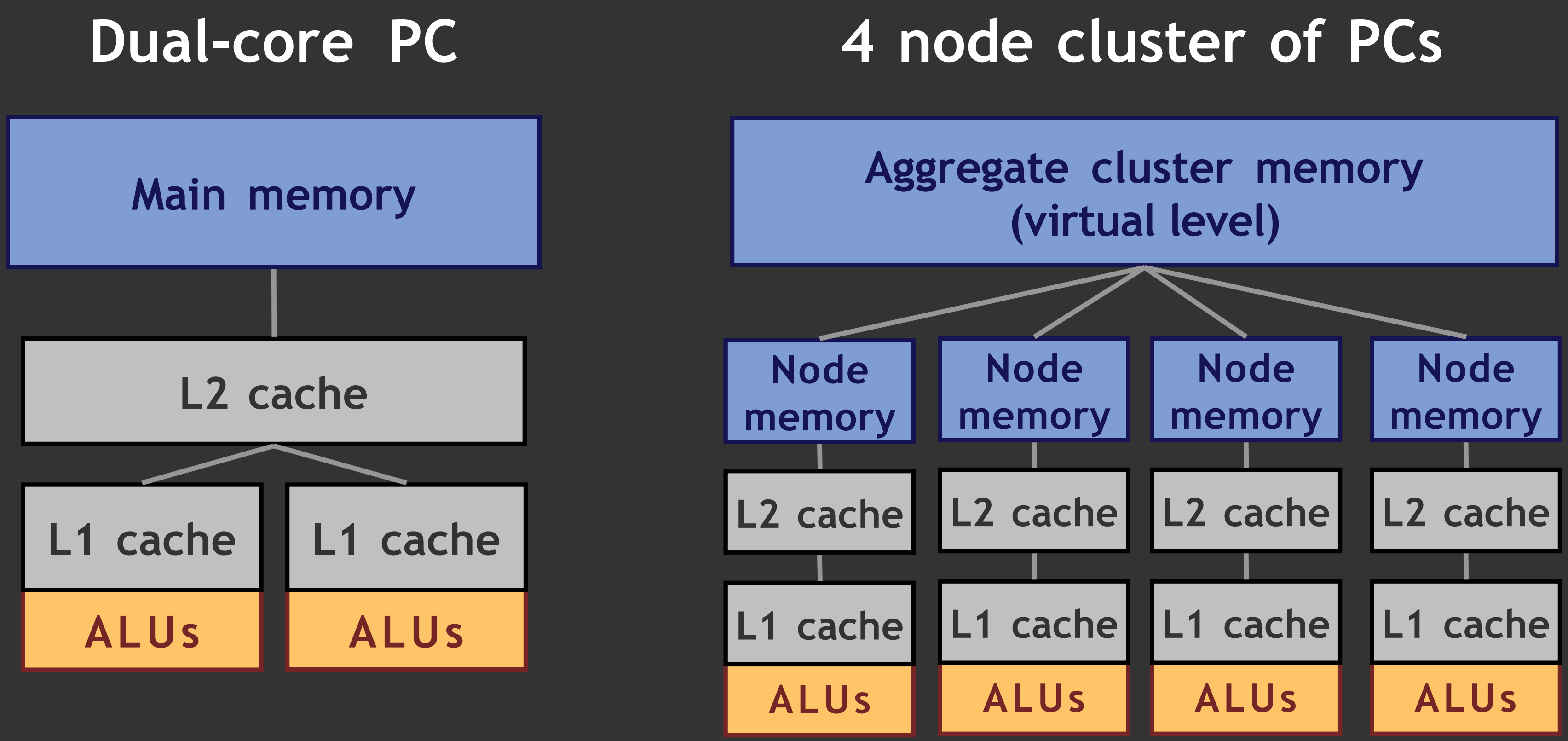
- Cache obliviousness (recursion)
- Space-limited procedures (Alpern et al.)

Programming methodologies, not programming environments

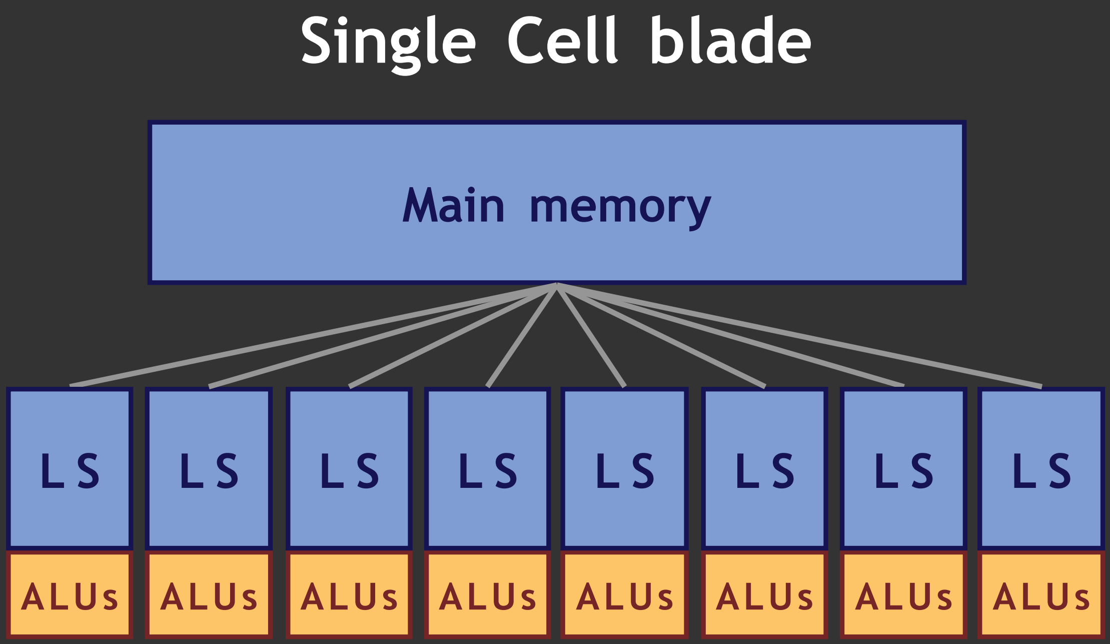
Hierarchical memory in Sequoia

Hierarchical memory

- Abstract machines as trees of memories

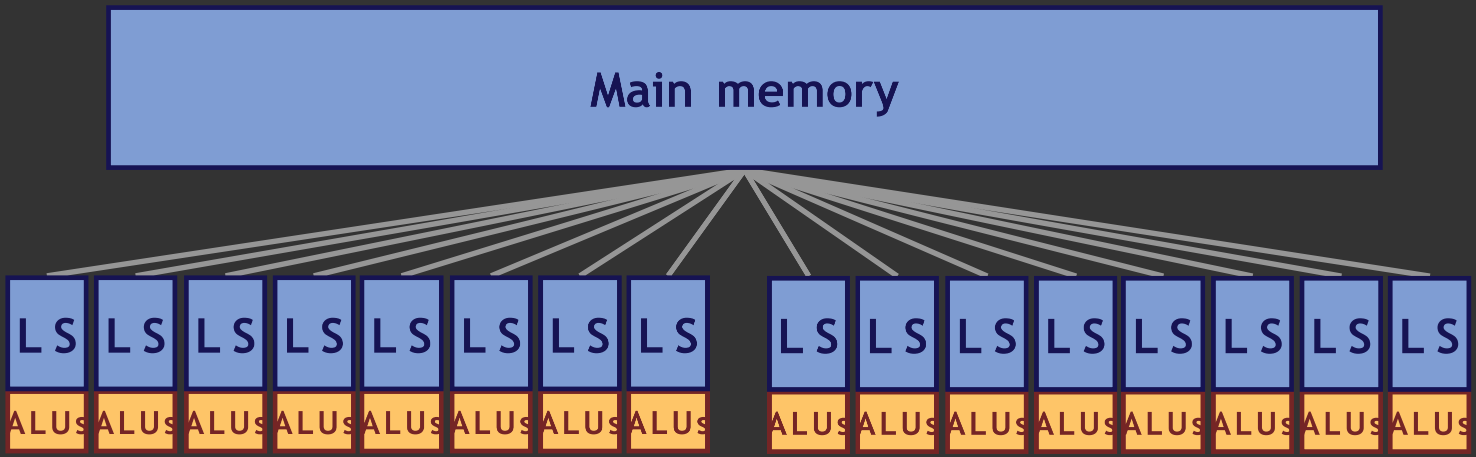


Hierarchical memory



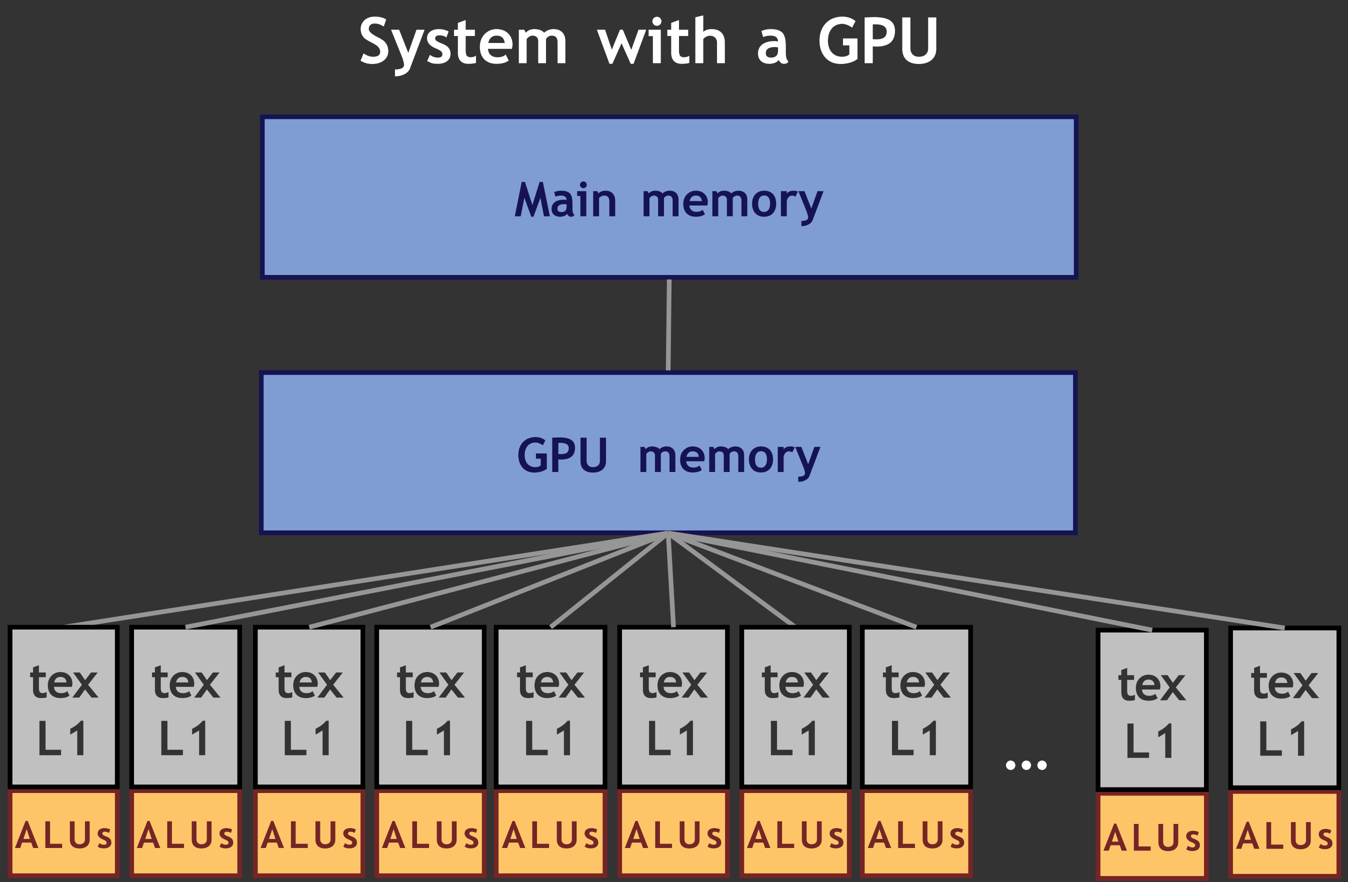
Hierarchical memory

Dual Cell blade



(No memory affinity modeled)

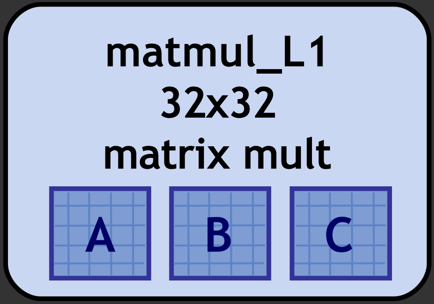
Hierarchical memory



Blocked matrix multiplication

C += A x B

```
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];
}
```



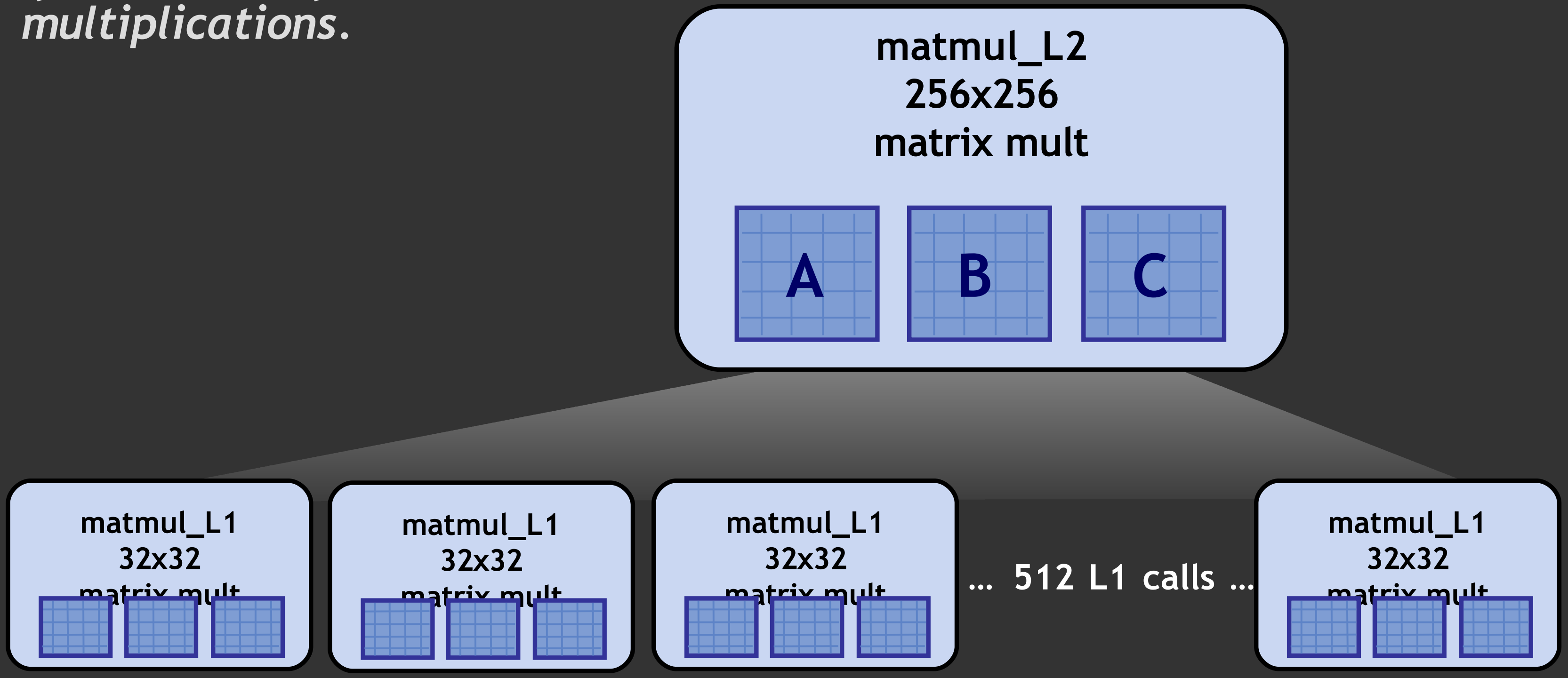
Blocked matrix multiplication

C += A x B

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

Perform series of L1 matrix multiplications.

```
}
```



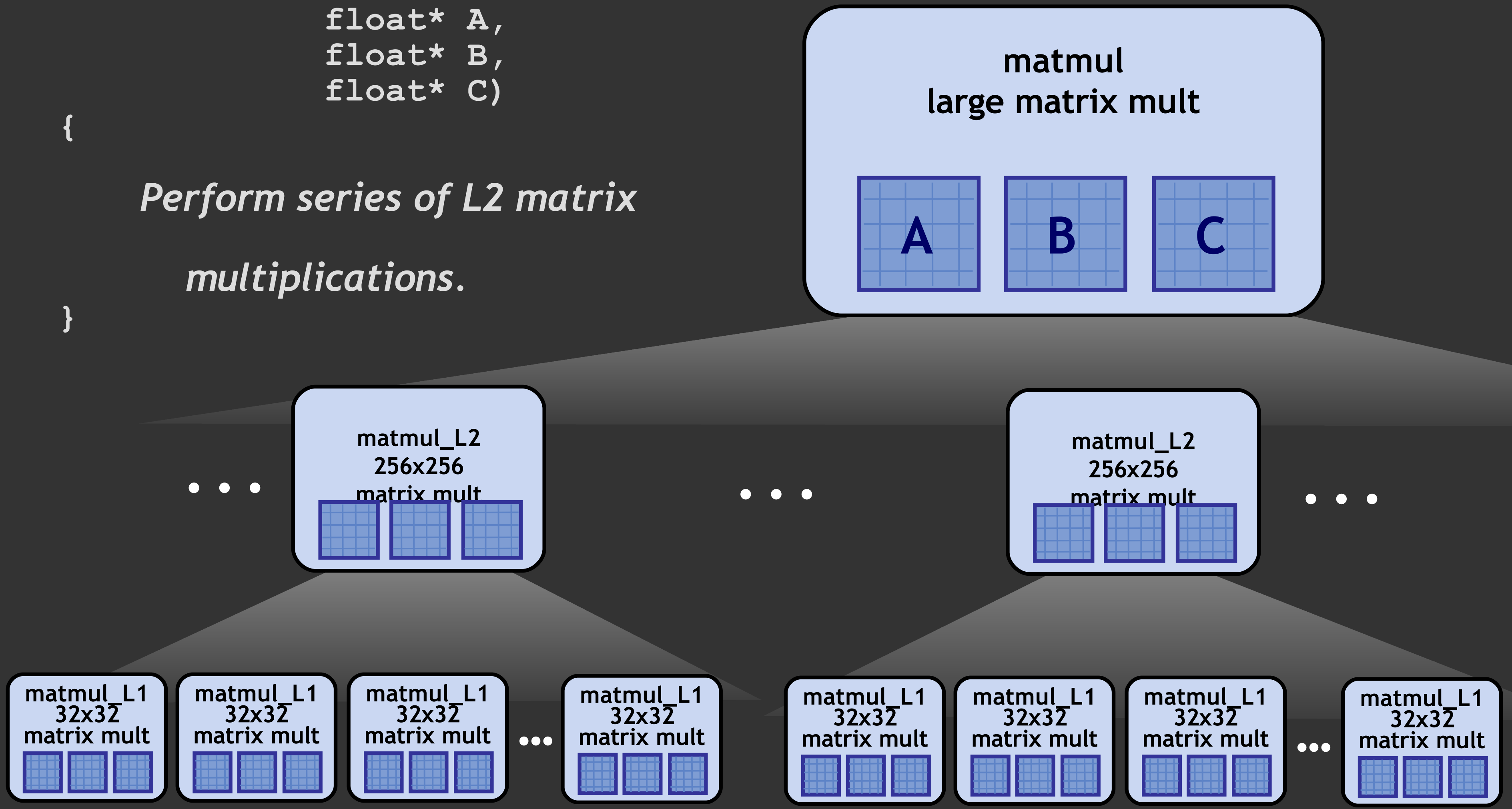
Blocked matrix multiplication

C += A x B

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

Perform series of L2 matrix multiplications.

```
}
```



Sequoia tasks

Sequoia tasks

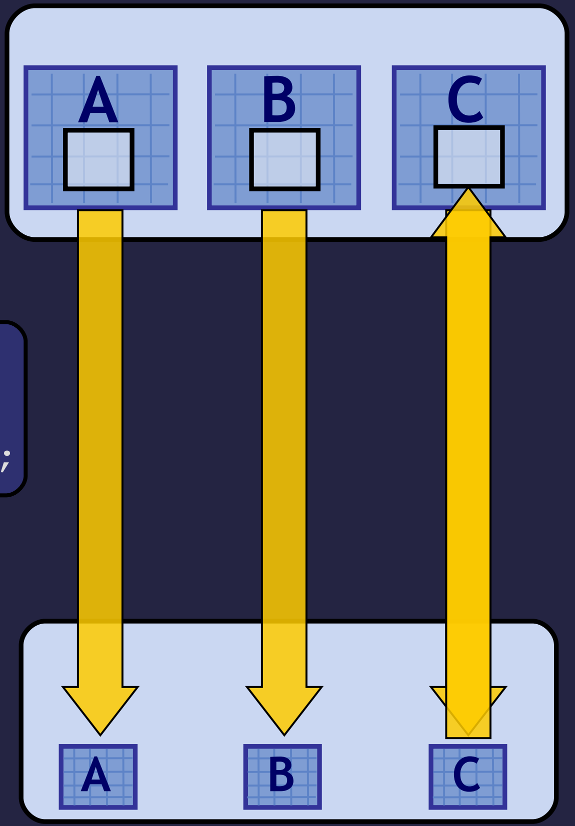
- Task arguments and temporaries define a working set
- Task working set resident at single location in abstract machine tree

```
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];
}
```

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.

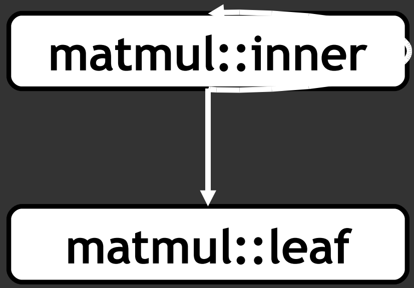
```
}
```

```
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];
}
```

Variant call graph



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: Can use 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

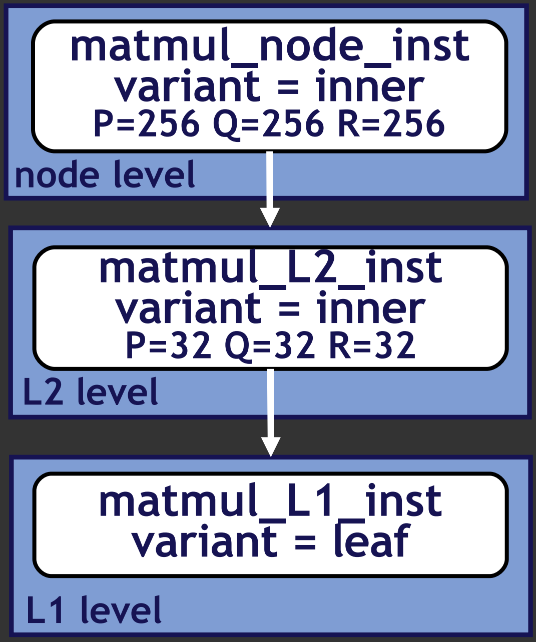
- Sequoia programs describe hierarchies of tasks
  - Mapped onto memory hierarchy
  - Parameterized for portability

Mapping tasks to machines

Task mapping specification

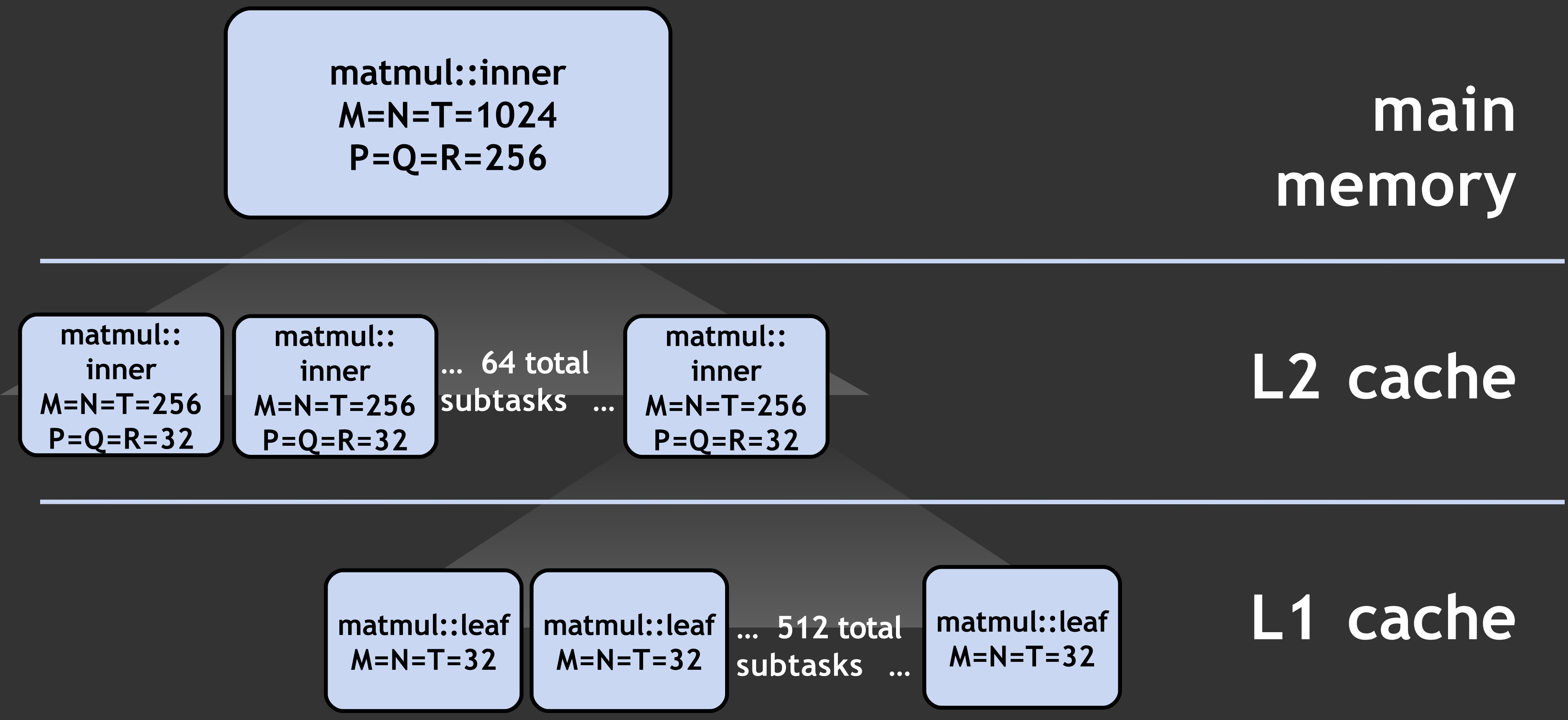
```
instance {
  name = matmul_node_inst
  task = matmul
  variant = inner
  runs_at = main_memory
  tunable P=256, Q=256, R=256
  calls = matmul_L2_inst
}

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
}
```

PC task instances



Specializing matmul

- Instances of tasks placed at each memory level



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
}
```

Sequoia Compiler

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)

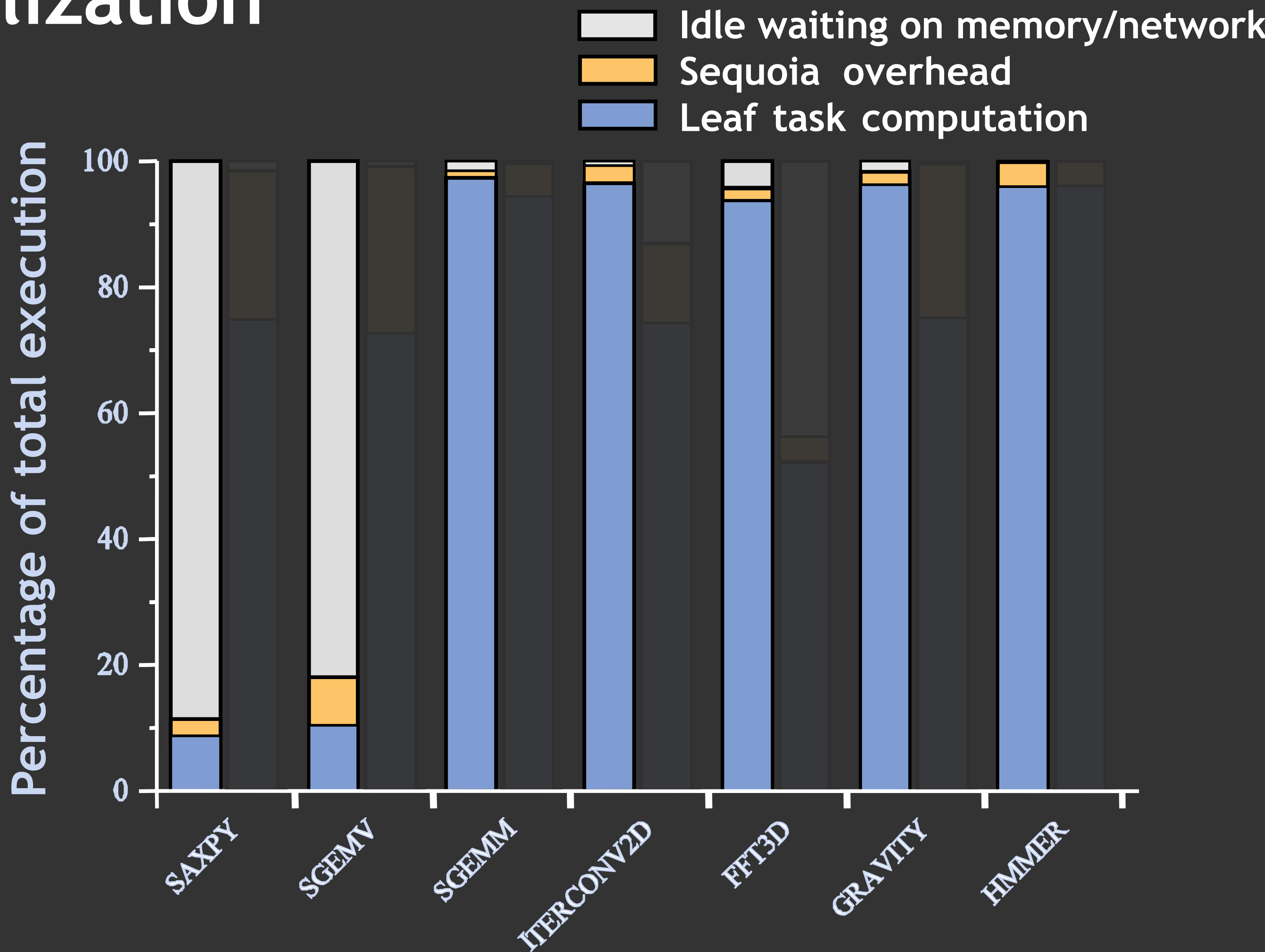
"Compilation for Explicitly Managed Memories"
Knight et al. To appear in PPOPP '07

Early results

- Scientific computing benchmarks

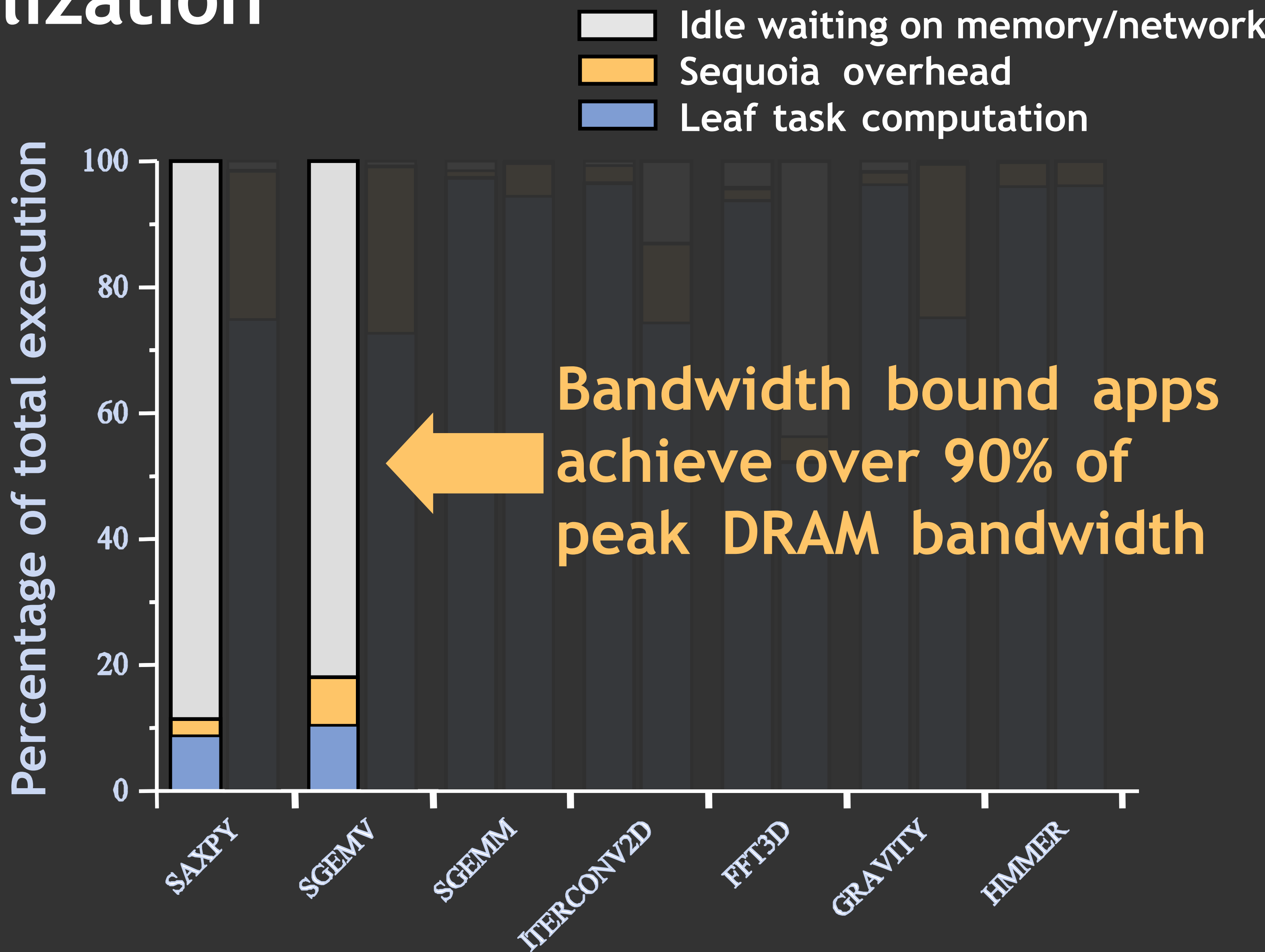
Linear Algebra	Blas Level 1 SAXPY, Level 2 SGEMV, and Level 3 SGEMM benchmarks
IterConv2D	Iterative 2D convolution with 9x9 support (non-periodic boundary constraints)
FFT3D	256^3 complex FFT
Gravity	100 time steps of N-body stellar dynamics simulation
HMMER	Fuzzy protein string matching using HMM evaluation (ClawHMMer: Horn et al. SC2005)

Utilization



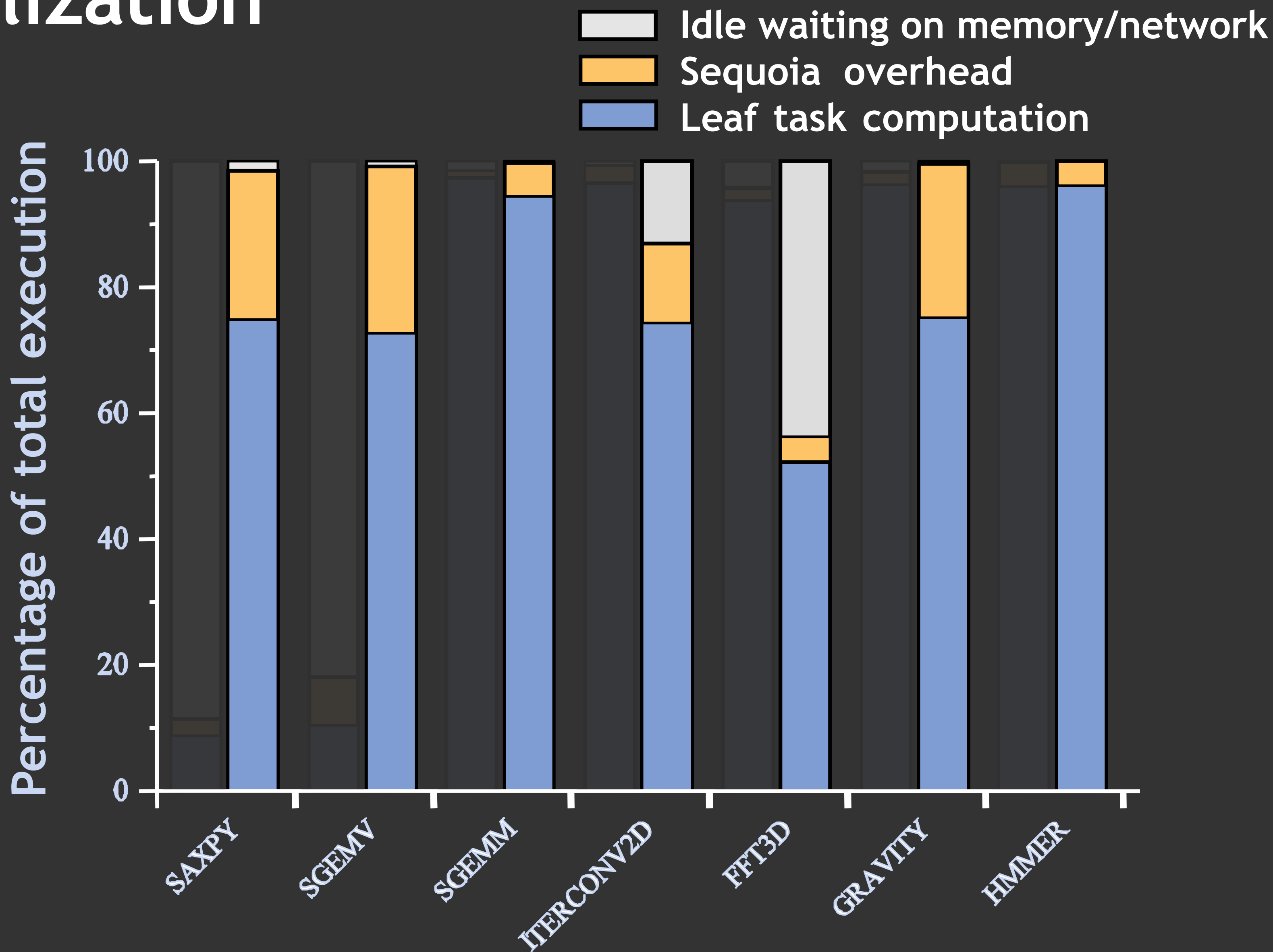
Execution on a Cell blade (left bars) and 16 node cluster (right bars)

Utilization



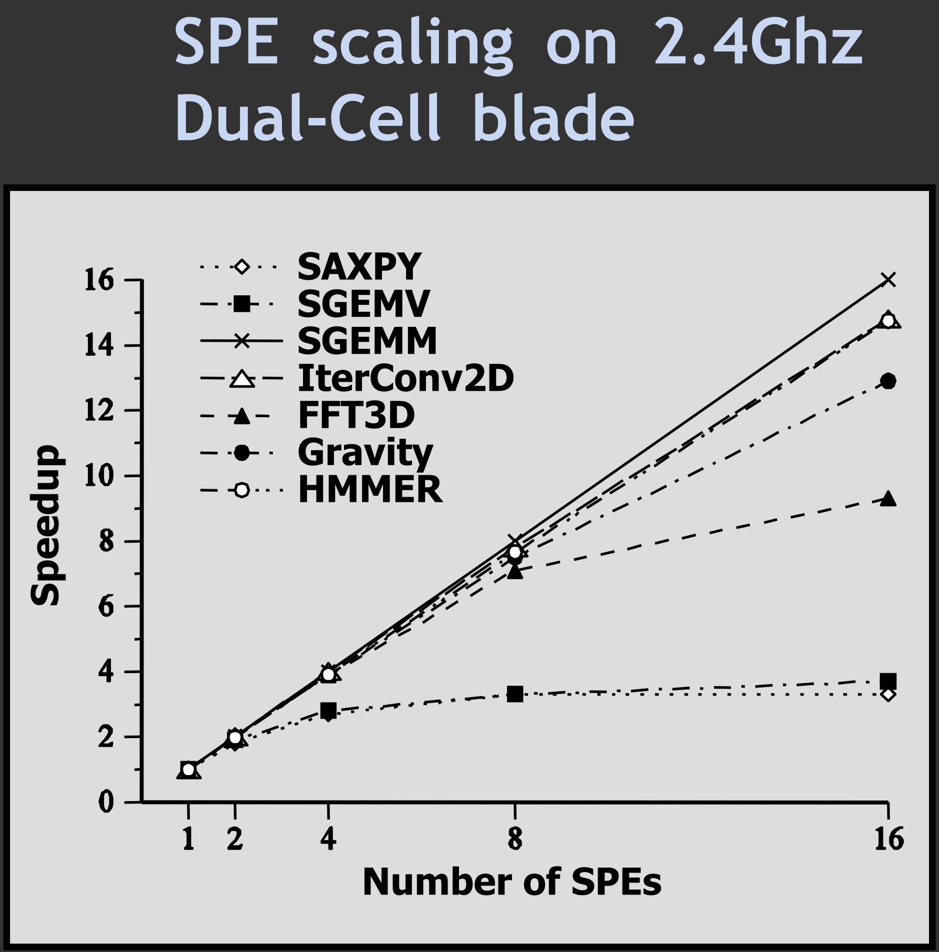
Execution on a Cell blade

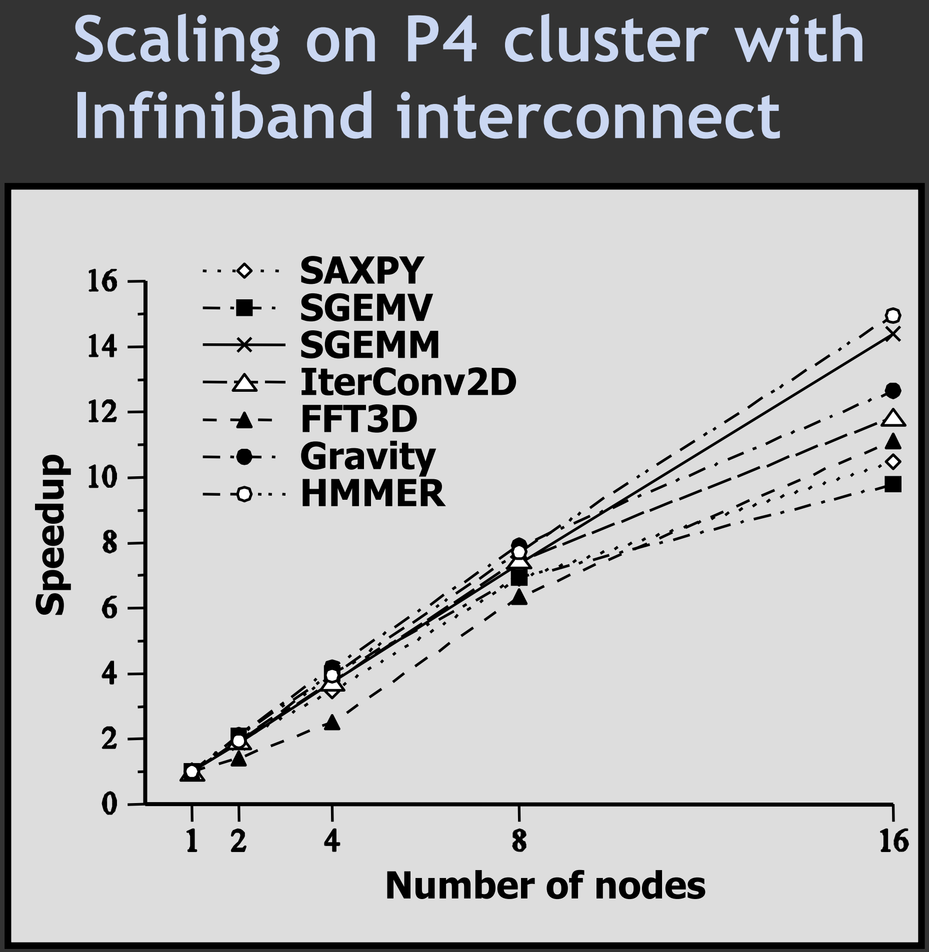
Utilization



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

Performance





Performance: GFLOP/sec

(single precision floating point)

	Single Cell * (8 SPE)	Dual Cell * (16 SPE)	Cluster ** (16 nodes)
SAXPY	3.2	4.0	3.6
SGEMV	9.8	11.0	11.1
SGEMM	96.3	174.0	97.9
IterConv2D	62.8	119.0	27.2
FFT3D	43.5	45.2	6.8
Gravity	83.3	142.0	50.6
HMMER	9.9	19.1	13.4

* 2.4 GHz Cell processor, DD2

** 2.4 GHz Pentium 4 per

Performance: GFLOP/sec

(single precision floating point)

	Single Cell * (8 SPE)	Dual Cell * (16 SPE)	Cluster ** (16 nodes)
SAXPY	3.2	4.0	3.6
SGEMV	9.8	11.0	11.1
SGEMM	96.3	174.0	97.9
IterConv2D	62.8	119.0	27.2
FFT3D	43.5	45.2	6.8
Gravity	83.3	142.0	50.6
HMMER	9.9	19.1	13.4

- Single Cell >= 16 node cluster of P4's

* 2.4 GHz Cell processor, DD2

** 2.4 GHz Pentium 4 per

Performance: GFLOP/sec

(single precision floating point)

	Single Cell * (8 SPE)	Dual Cell * (16 SPE)	Cluster ** (16 nodes)
SAXPY	3.2	4.0	3.6
SGEMV	9.8	11.0	11.1
SGEMM	96.3	174.0	97.9
IterConv2D	62.8	119.0	27.2
FFT3D	43.5	45.2	6.8
Gravity	83.3	142.0	50.6
HMMER	9.9	19.1	13.4

- 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

Performance: GFLOP/sec

(single precision floating point)

	Single Cell * (8 SPE)	Dual Cell * (16 SPE)	Cluster ** (16 nodes)
SAXPY	3.2	4.0	3.6
SGEMV	9.8	11.0	11.1
SGEMM	96.3	174.0	97.9
IterConv2D	62.8	119.0	27.2
FFT3D	43.5	45.2	6.8
Gravity	83.3	142.0	50.6
HMMER	9.9	19.1	13.4

- FFT3D on par with best-known Cell implementation

* 2.4 GHz Cell processor, DD2

** 2.4 GHz Pentium 4 per

Performance: GFLOP/sec

(single precision floating point)

	Single Cell * (8 SPE)	Dual Cell * (16 SPE)	Cluster ** (16 nodes)
SAXPY	3.2	4.0	3.6
SGEMV	9.8	11.0	11.1
SGEMM	96.3	174.0	97.9
IterConv2D	62.8	119.0	27.2
FFT3D	43.5	45.2	6.8
Gravity	83.3	142.0	50.6
HMMER	9.9	19.1	13.4

- Gravity outperforms custom ASICs

* 2.4 GHz Cell processor, DD2

** 2.4 GHz Pentium 4 per

Performance: GFLOP/sec

(single precision floating point)

	Single Cell * (8 SPE)	Dual Cell * (16 SPE)	Cluster ** (16 nodes)
SAXPY	3.2	4.0	3.6
SGEMV	9.8	11.0	11.1
SGEMM	96.3	174.0	97.9
IterConv2D	62.8	119.0	27.2
FFT3D	43.5	45.2	6.8
Gravity	83.3	142.0	50.6
HMMER	9.9	19.1	13.4

- HMMER outperforms Horn et al.'s GPU implementation from SC05

* 2.4 GHz Cell processor, DD2

** 2.4 GHz Pentium 4 per

Sequoia portability

- No Sequoia source level modifications except for FFT3D*
  - Changed task parameters
  - Ported leaf task implementations
- Cluster → 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