EE382V (17325): Principles in Computer Architecture
Parallelism and Locality
Fall 2007

Lecture 21 – Programming the Cell BE (II)

Mattan Erez

The University of Texas at Austin

Outline

- Cell programming challenges review
- Sequoia
  – Review + mapping
- Other Cell programming tools

- Sequoia part courtesy Kayvon Fatahalian, Stanford

- All Cell related images and figures © Sony and IBM
- Cell Broadband Engine ™ Sony Corp.

Cell Software Challenges

- Separate code for PPE and SPEs
  - Explicit synchronization
- SPEs can only access memory through DMAs
  - DMA is asynchronous, but prep instructions are part of SPE code
  - SW responsible for consistency and coherency
  - SW responsible for alignment, granularity, and bank conflicts
- SPEs must be programmed with SIMD
  - Alignment is up to SW
  - Lots of pipeline challenges left up to programmer / compiler
    - Deep pipeline with no branch predictor
    - 2-wide scalar pipeline needs static scheduling
    - LS shared by DMA, instruction fetch, and SIMD LD/ST
    - No memory protection on LS (Stack can "eat" data or code)

Outline

- Cell programming challenges review
- **Sequoia**
  - **Review + mapping**
- Other Cell programming tools

- Sequoia part courtesy Kayvon Fatahalian, Stanford

- All Cell related images and figures © Sony and IBM
- Cell Broadband Engine ™ Sony Corp.

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

Stanford University

Sequoia

- Language: stream programming for machines with deep memory hierarchies
- Idea: Expose abstract memory hierarchy to programmer
- Implementation: benchmarks run well on Cell processor based systems and on cluster of PCs

Key challenge in high performance programming is:

communication (not parallelism)

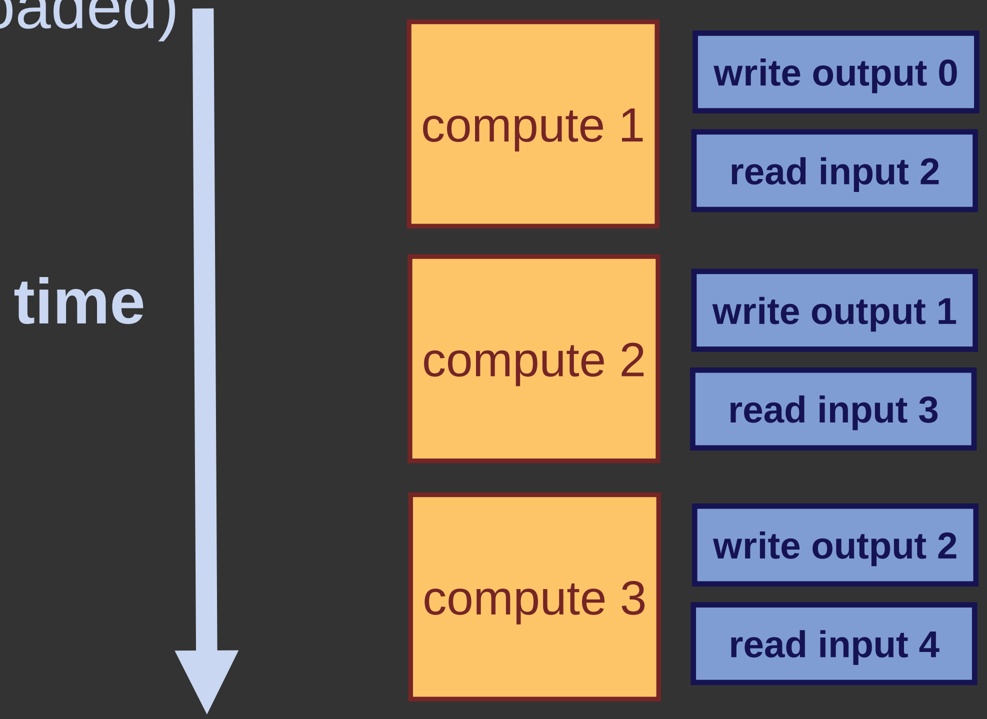
Latency

Bandwidth

Avoiding latency stalls

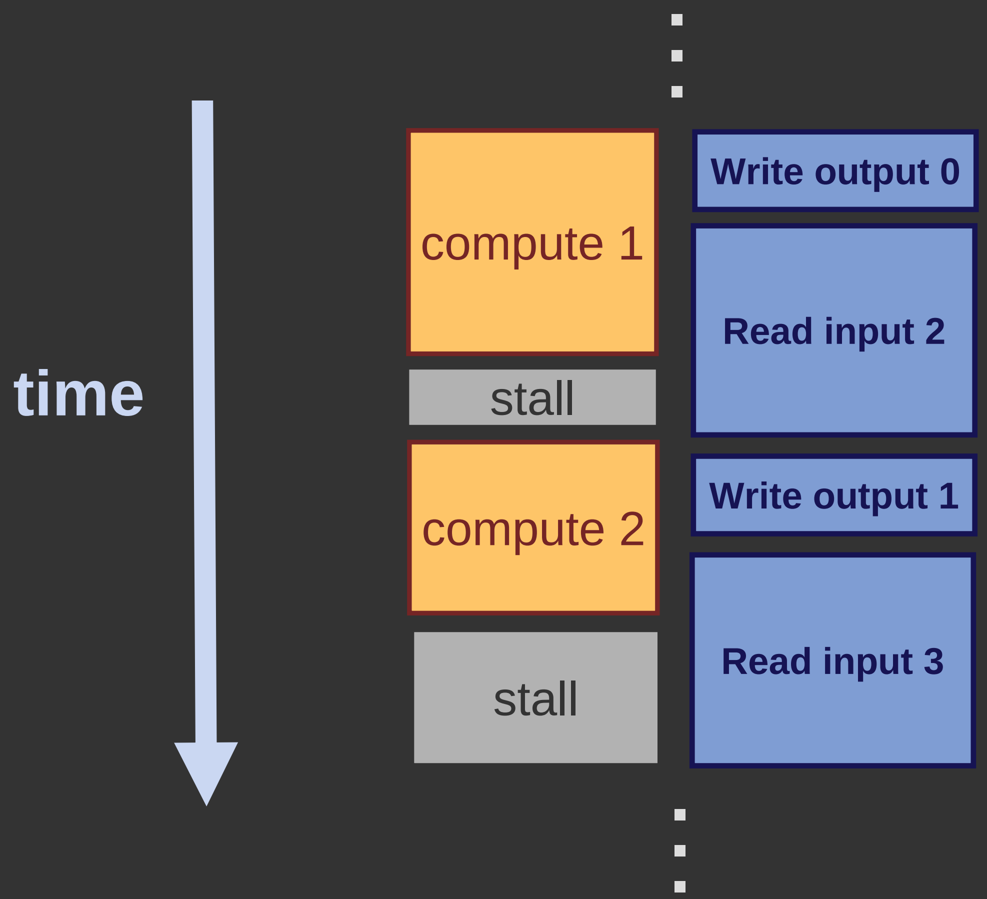
1. Prefetch batch of data
2. Compute on data (avoiding stalls)
3. Initiate write of results

… 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

Streaming

Streaming involves structuring algorithms as collections of independent [locality cognizant] computations with well-defined working sets.

Efficient programs exhibit this structure at many scales.

Sequoia's goals

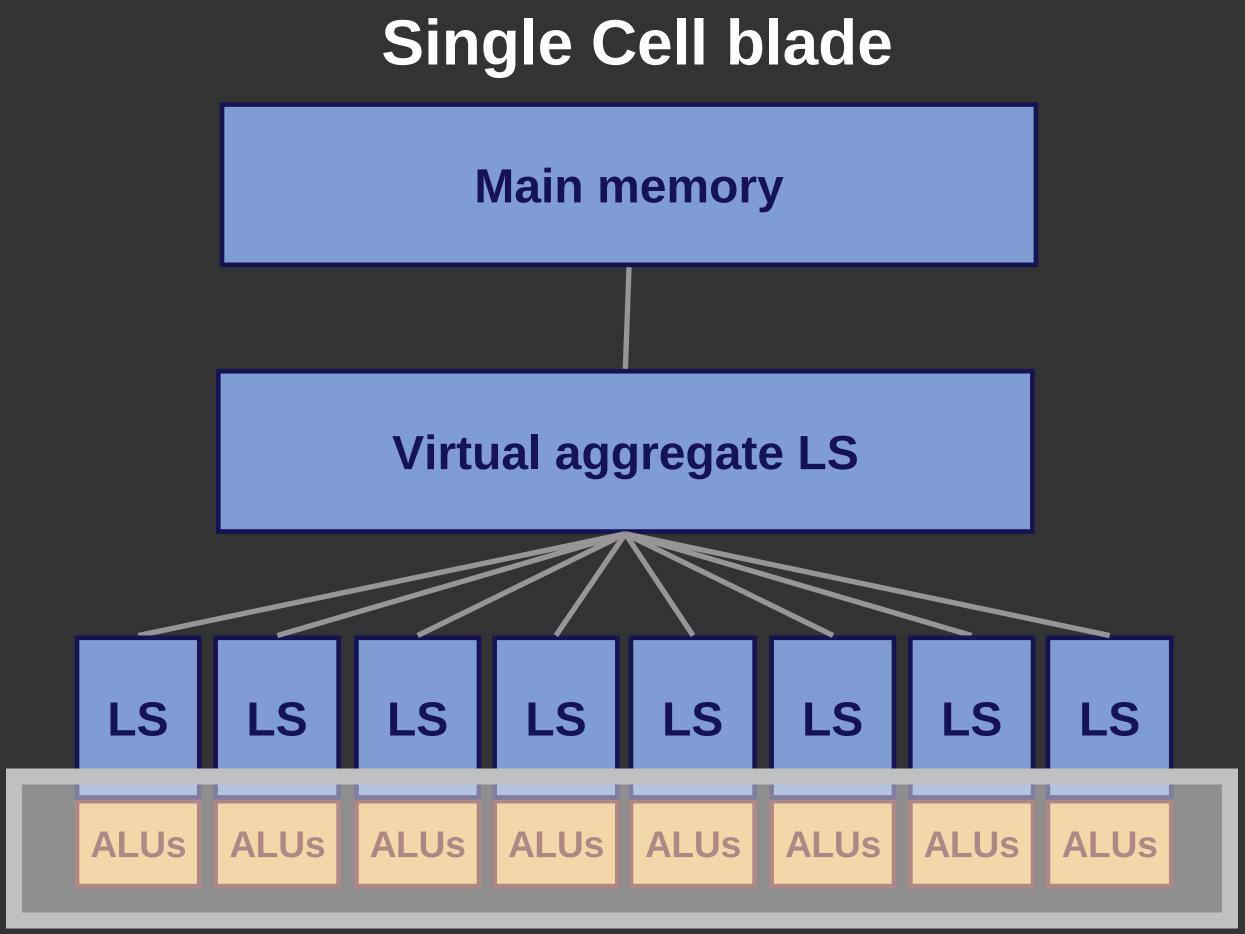
- Facilitate development of hierarchy-aware stream programs …

  … that remain portable across machines

- Provide constructs that can be implemented efficiently without requiring advanced compiler technology
  - Place computation and data in machine
  - Explicit parallelism and communication
  - Large bulk transfers

Hierarchical memory in Sequoia

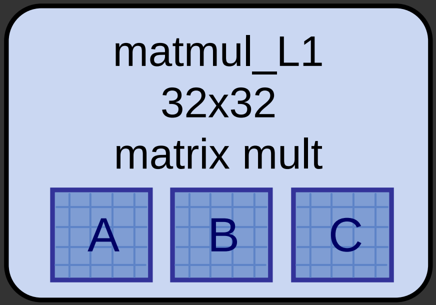
Hierarchical memory



Blocked matrix multiplication

C += A x B

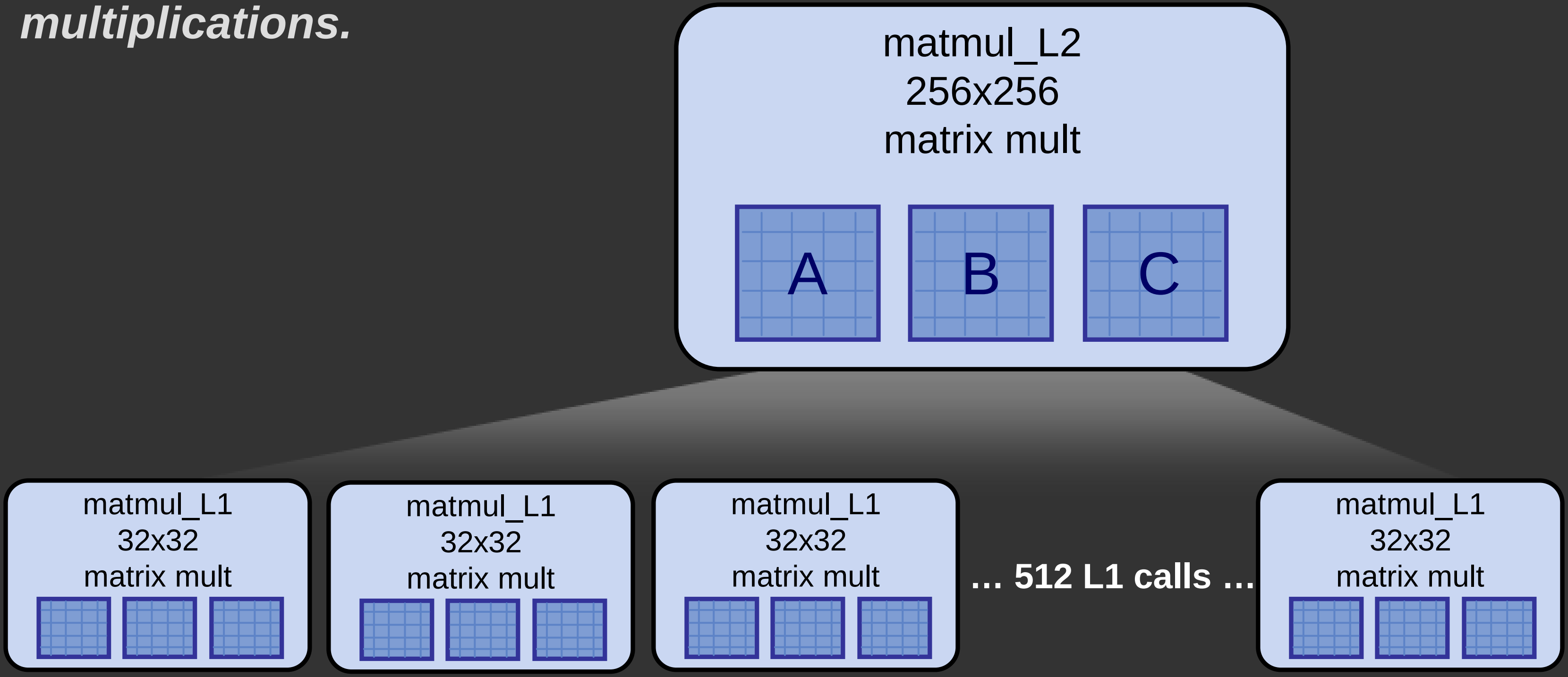
```
void matmul_L1( int M, int N, int T,
                float* A,
                float* B,
                float* C)
{
   for (int i=0; i<M; i++)
     for (int j=0; j<N; j++)
       for (int k=0; k<T; k++)
          C[i][j] += A[i][k] * B[k][j];
}
```



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.
}
```



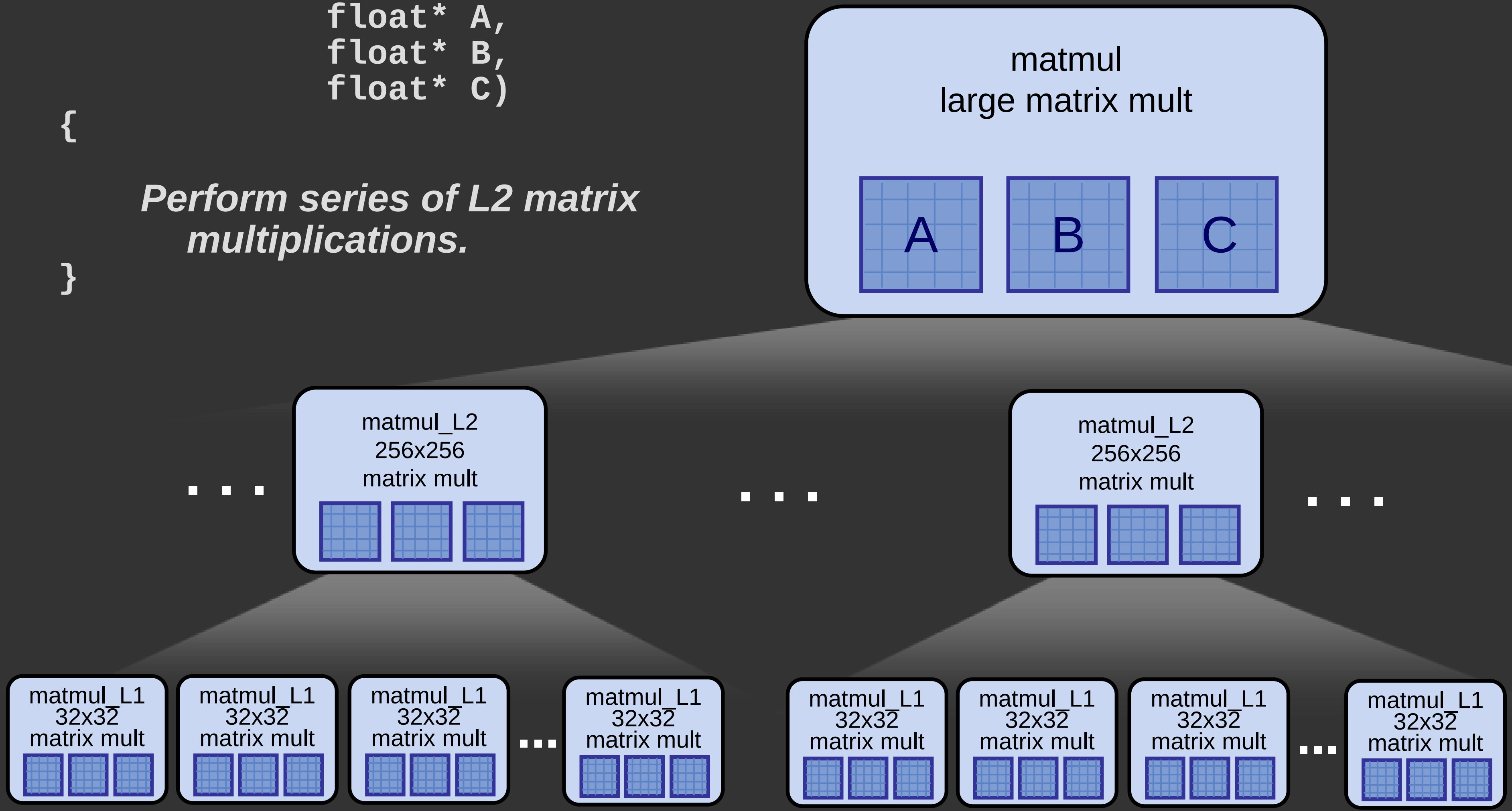
Blocked matrix multiplication

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

- 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.

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;
```

Recursively call **`matmul`** *task on submatrices of A, B, and C of size PxQ, QxR, and PxR.*

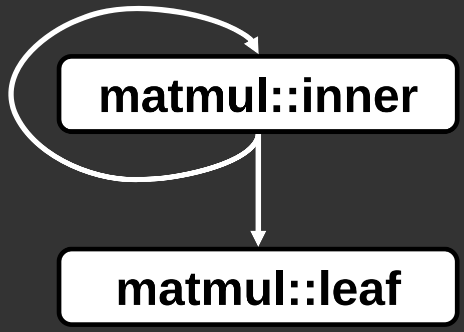
```
}
```

```
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];
}
```

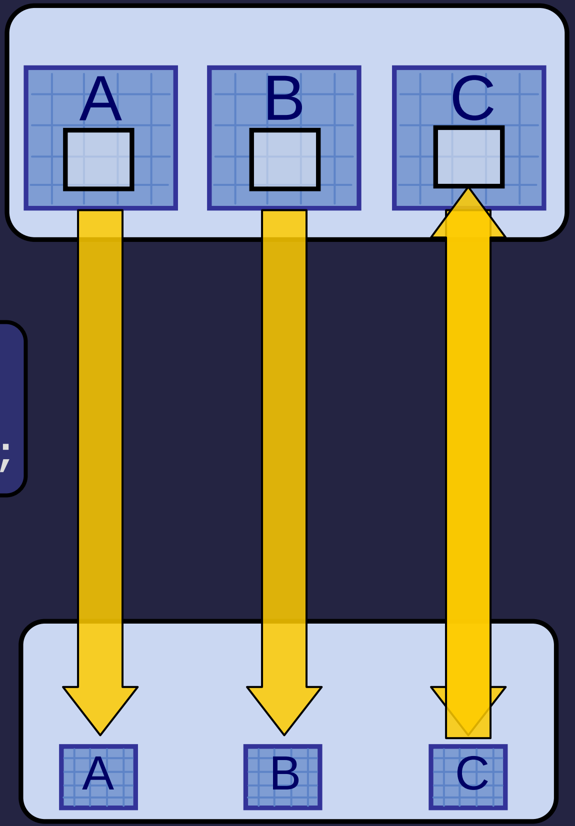
Variant call graph



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] );
     }
  }
}

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

Synchronization

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

- No explicit synchronization
  - Why?
- 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

Synchronization Impacts Parallelism

- Parallelism explicitly expressed using *mappar*
  - 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 mappar
    - Compiler may not currently interchange loops
- What about interacting thread?
  - **Not allowed!**
  - **Why?**

Summary: Sequoia tasks

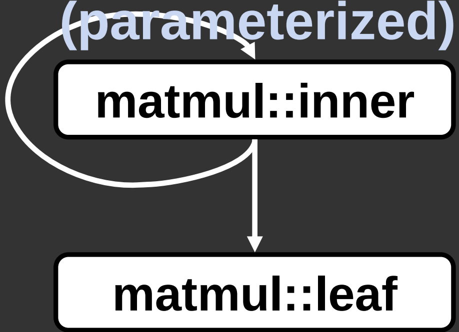
- Single abstraction for
  - **Isolation / parallelism**
    - With help from programmer
  - Explicit communication / working sets
  - Expressing locality

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

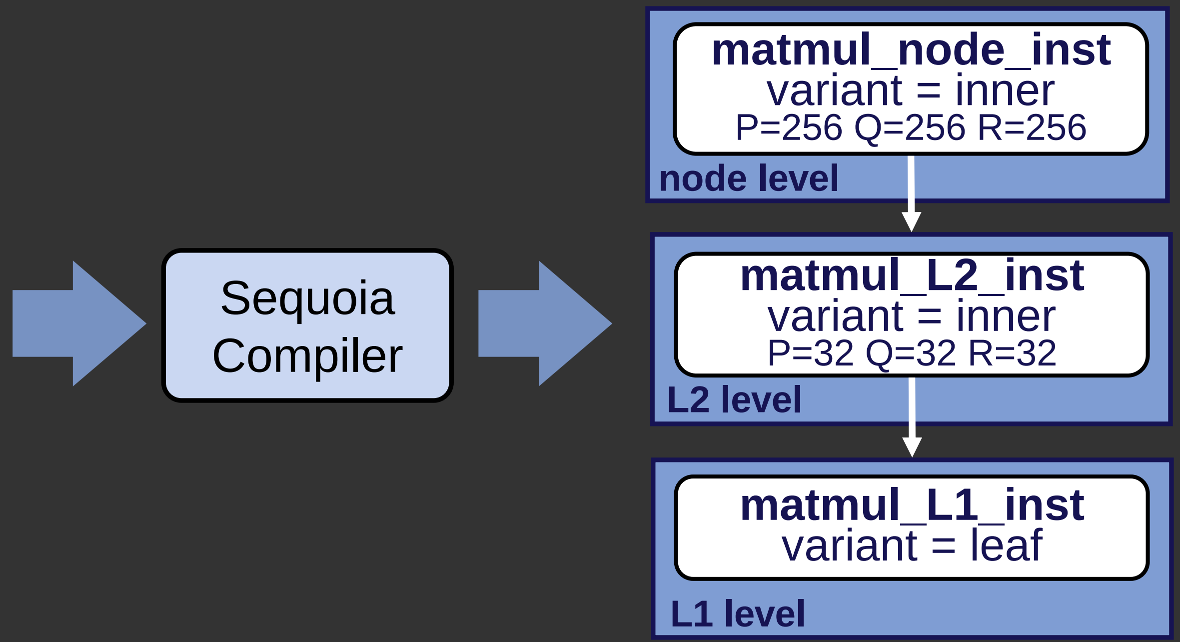
Mapping tasks to machines

How mapping works

Sequoia task definitions (parameterized)



Task instances (not parameterized)



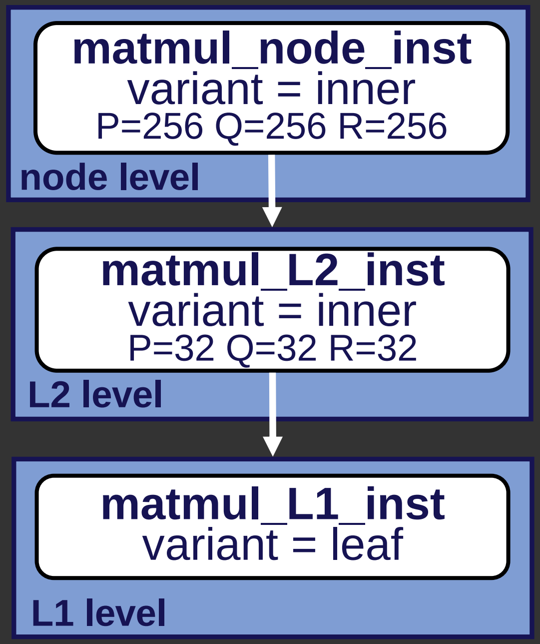
Mapping specification

```
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
}
```

Task mapping specification

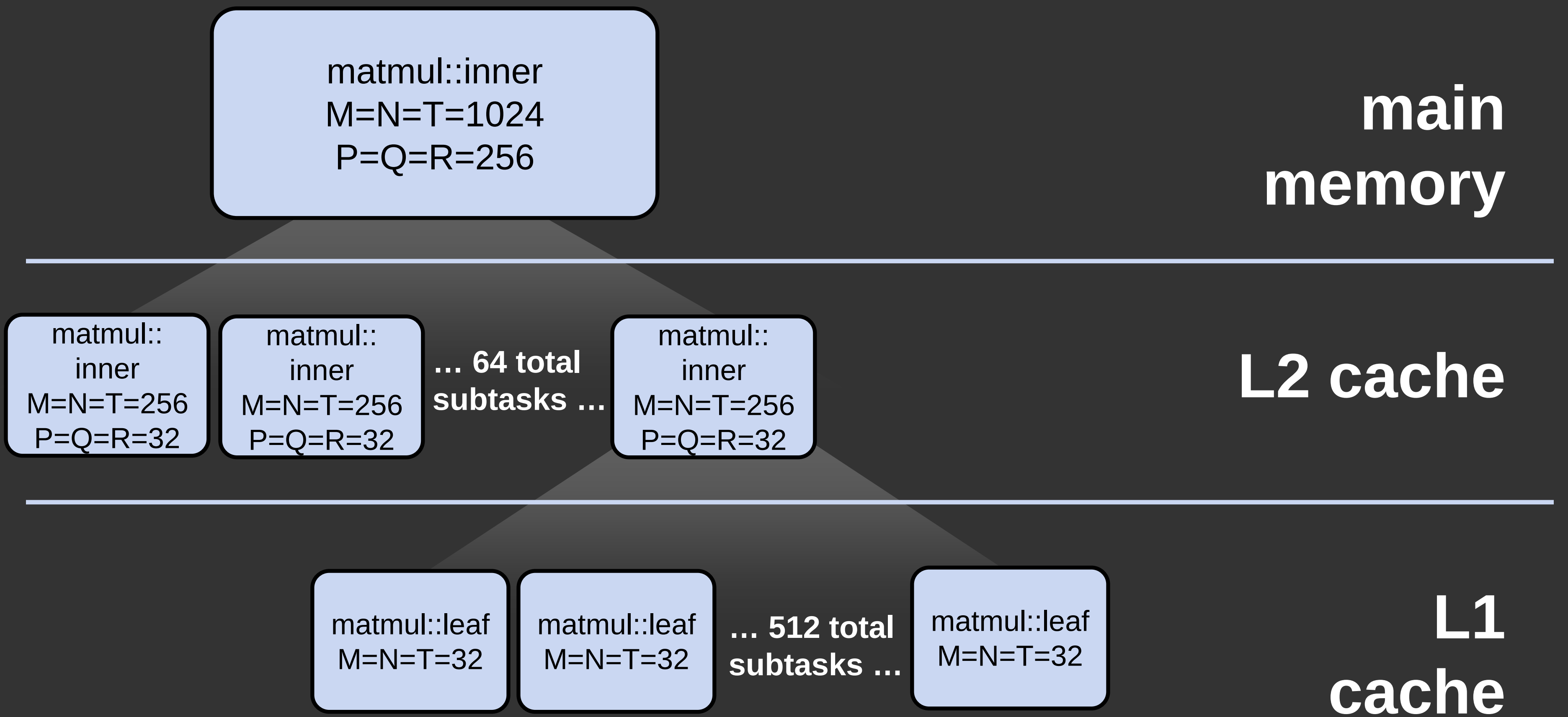
```
instance {
  name = matmul_node_inst
  task = matmul
  variant = inner
  runs_at = main_memory
  tunable P=256, Q=256, R=256
  calls = matmul_L2_inst
}

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



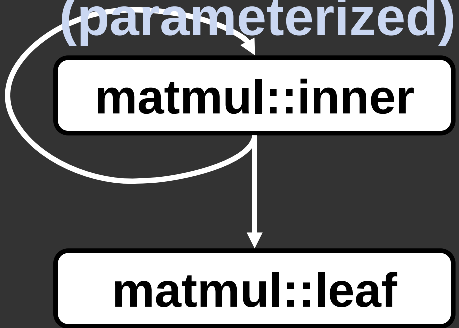
Specializing matmul

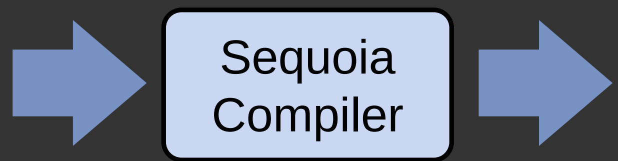
- Instances of tasks placed at each memory level



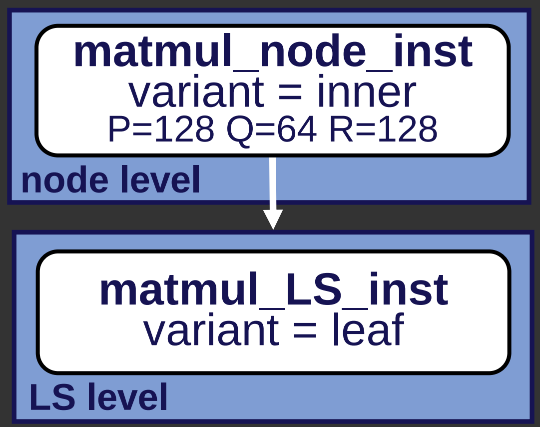
Task instances: Cell

Sequoia task definitions (parameterized)





Cell task instances (not parameterized)



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
}
```

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 / padding**
  - **Scheduling (tasks and DMAs)**

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

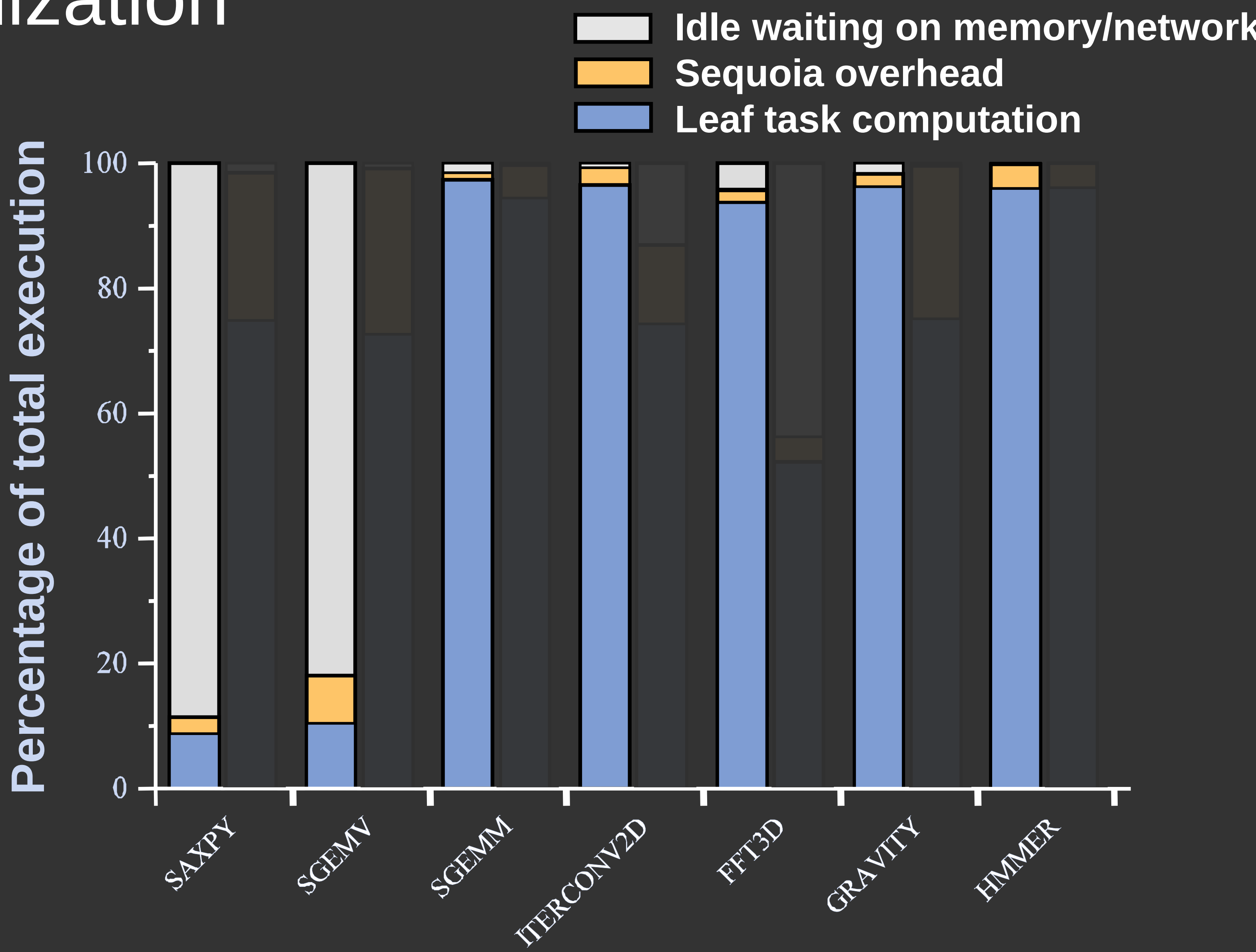
Early results

- 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

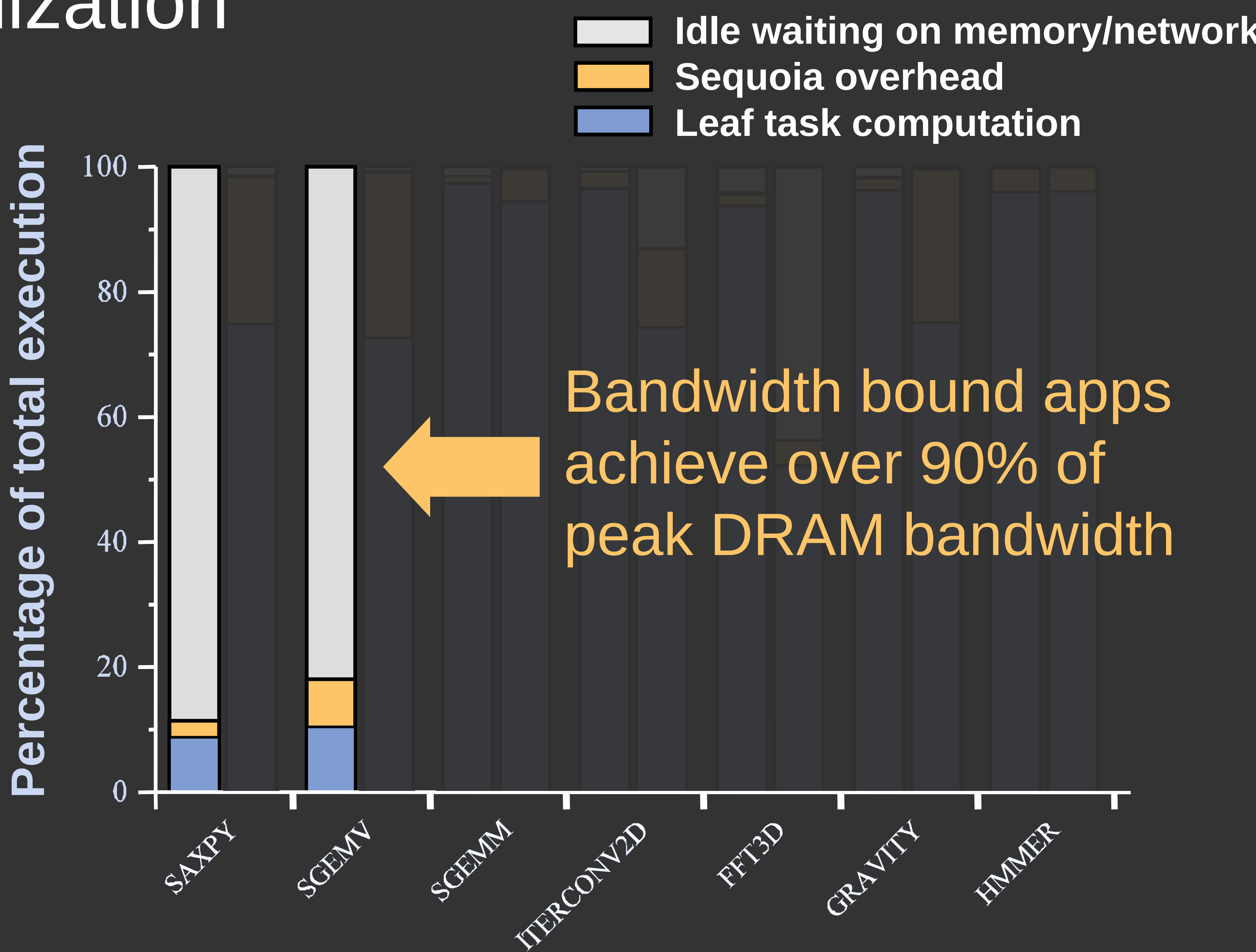
(ClawHMMer: Horn et al. SC2005)

Utilization



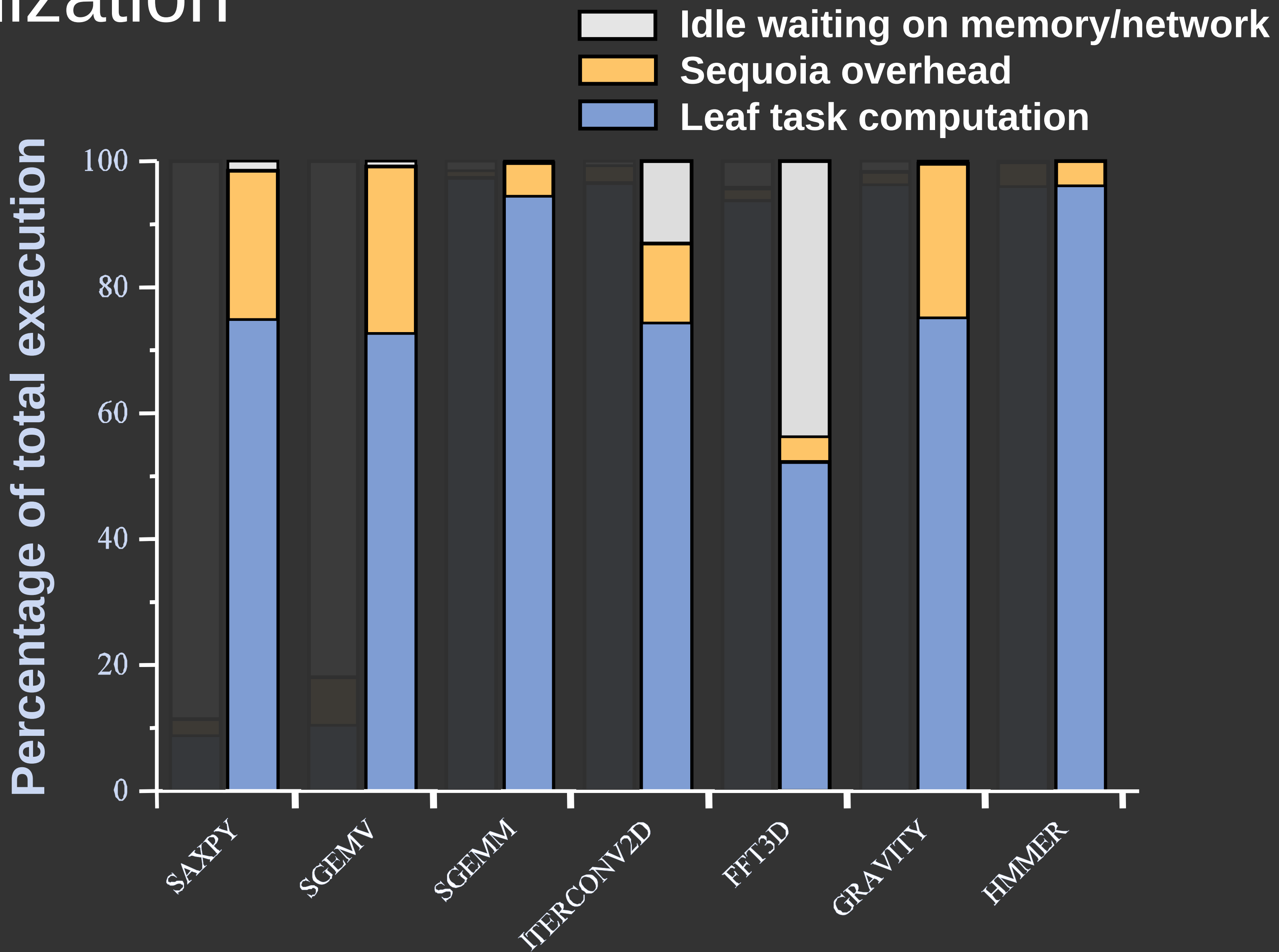
Execution on a Cell blade (left bars) and 16 node cluster (right bars)

Utilization



Execution on a Cell blade

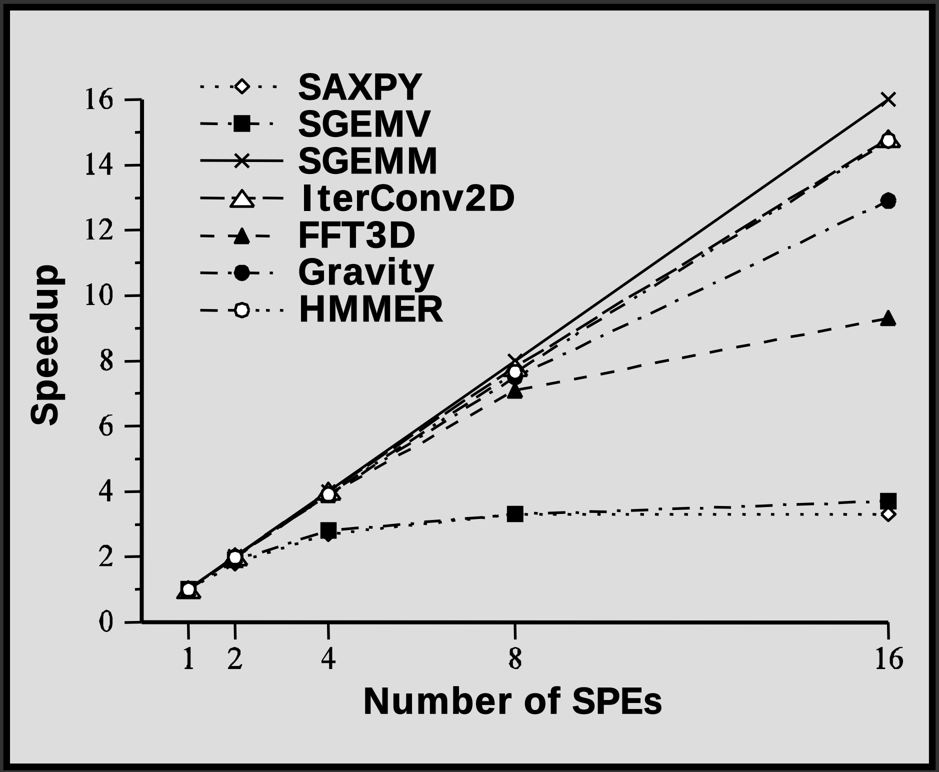
Utilization



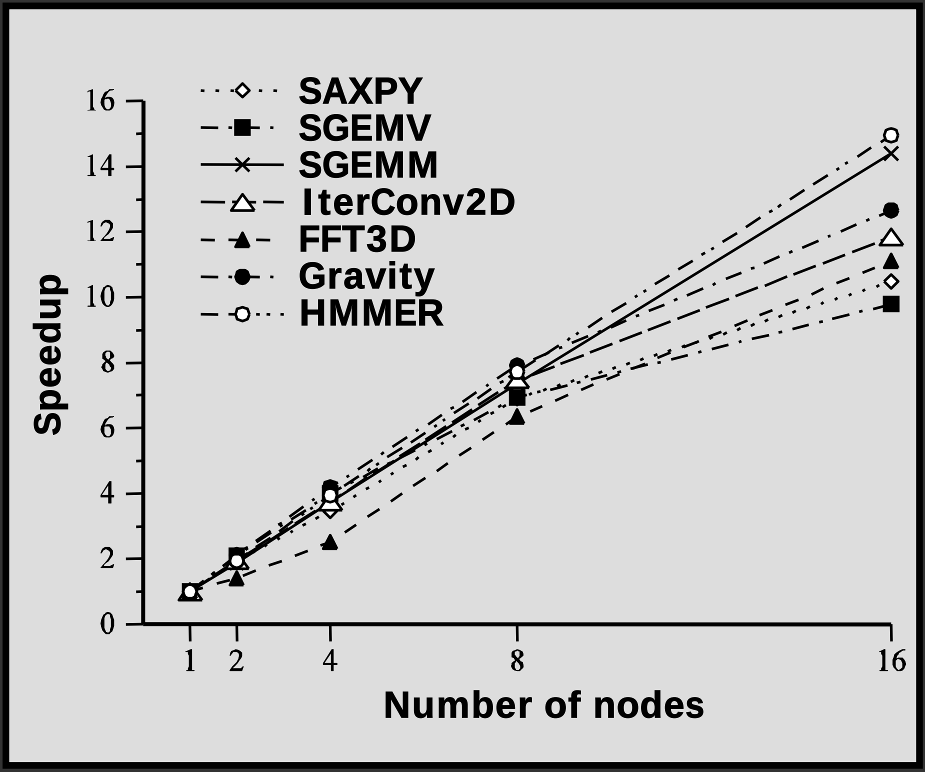
Execution on a Cell blade (left bars) and 16 node cluster (right bars)

Performance

SPE scaling on 2.4Ghz Dual-Cell blade



Scaling on P4 cluster with Infiniband interconnect



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 node

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 node

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 node

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 node

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 node

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 node

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**
  - **Some progress in automated search for parameters (auto-tuning style)**

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

Sequoia and Cell Programming Challenges

- Sequoia manages threading and synchronization
- Sequoia manages communication and all DMAs
  - Including padding and performance, but not alignment
- Sequoia manages LS
  - Allocation and packing
- Sequoia manages scheduling
  - SWP of mappar to hide communication latency

- Sequoia doesn't help with SPE code
  - Use low-level compiler tools such as XLC
- Sequoia doesn't currently help with some memory restrictions
  - Alignment
  - Banks

Outline

- Cell programming challenges review
- Sequoia
  - Review + mapping
- **Other Cell programming tools**

- Sequoia part courtesy Kayvon Fatahalian, Stanford

- All Cell related images and figures © Sony and IBM
- Cell Broadband Engine ™ Sony Corp.

Tools From IBM

- Cell SDK 3.0
  - API calls for handling communication, synchronization, and DMA
  - LIBSPU and SPUFS for getting the SPEs to do something and setting up threads and memory
  - Intrinsics for programming the SPE pipeline directly
  - GCC port for PPE and SPE part (separate compilers)
    - Only handles non-SPE specific optimizations + intrinsics
  - XLC port for PPE and SPE part (separate compilers)
    - XLC supposed to optimize for SPE pipeline with branch hints, scheduling, instruction prefetch, …
    - Automatic SIMD-ization?
- Accelerated Library Framework (ALF)
  - APIs for work queue based model to program control-plane
- "Octopiler" – single-source XLC for Cell
  - OpenMP directives
  - Relies on SW cache to get the OpenMP working
  - Automatic SIMD-ization

Tools from Industry

- Mercury Systems
  - Array based language
  - Highly-tuned BLAS and FFT

- RapidMind
  - Dynamically compiled program
  - Relies on array data types
  - Builds up kernels and DMAs

Tools From Academia

- Sequoia

- Cell Superscalar (CellSs)
  - Program with OpenMP like directives to identify kernels
  - Uses SW cache intensively
  - Runtime applies superscalar style optimization and scheduling to coarse-grained kernels (identified above)

- Charm++
  - Runtime based approach
  - Objects with explicit communication and “entry points” for synchronization
  - Uses a work queue and peaks into it to do the DMAs