Sequoia: Programming the Memory Hierarchy

Mike Houston & Kayvon Fatahalian

Oct. 26th, 2006

Today's outline

- Sequoia programming model

- Sequoia Cell backend

- http://sequoia.stanford.edu
  - Supercomputing 2006 paper
  - Compiler papers under review

Emerging Themes

Writing high-performance code amounts to…

Intelligently structuring algorithms

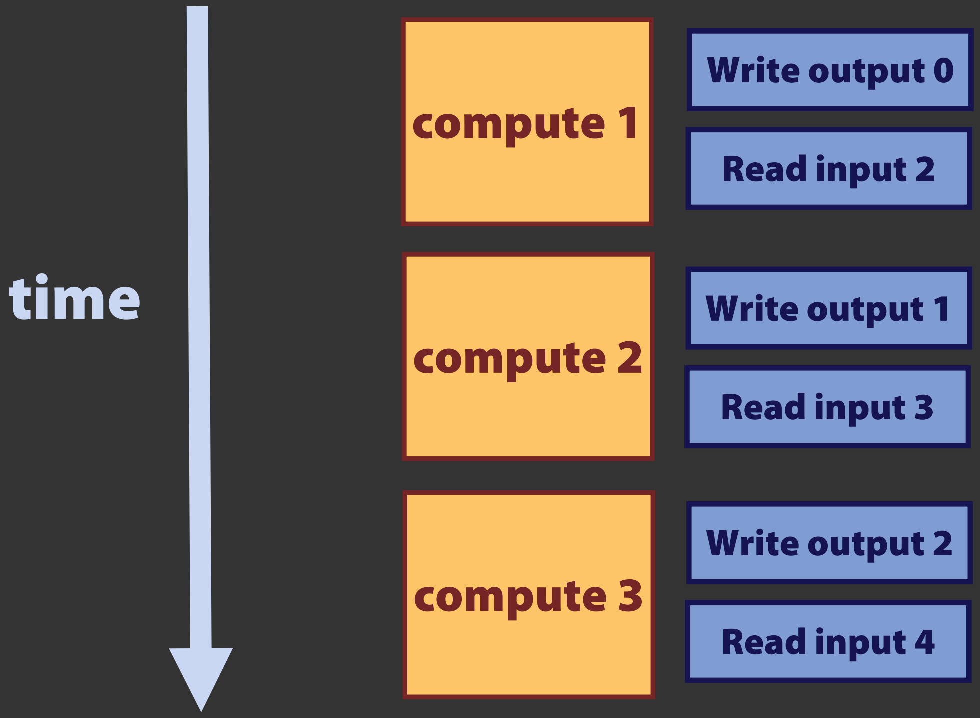
[compiler help unlikely]

Generating efficient inner loops (kernels)

[compilers might come around]

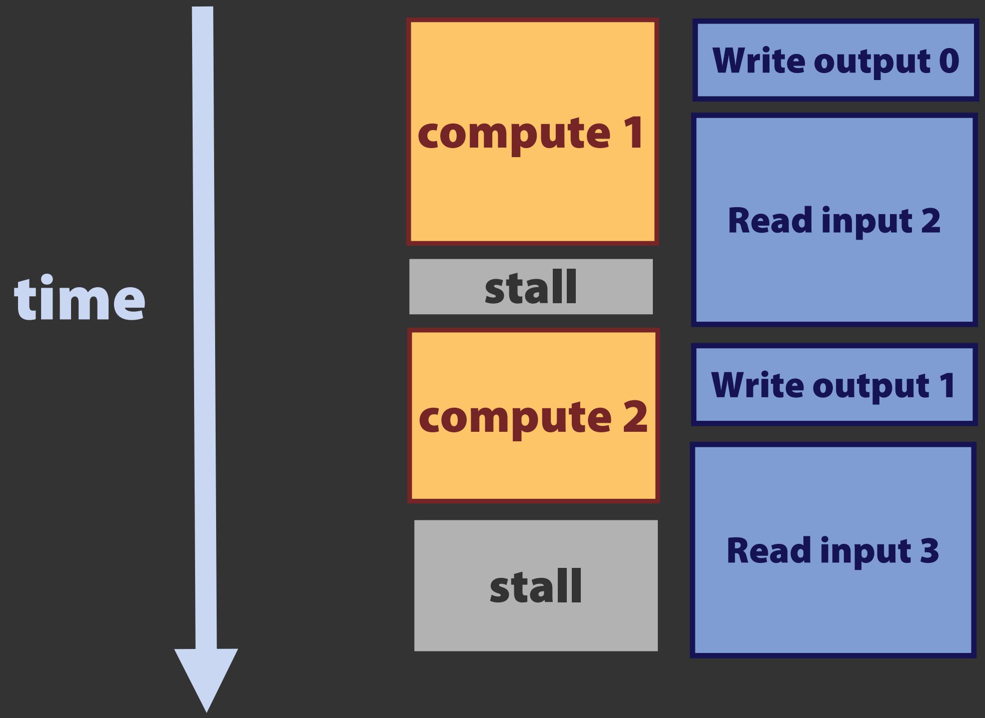
It’s about program structure

1. Preload batch of data
2. Compute on data
3. Initiate write of results (this data is done)
4. Compute on next batch (which should be loaded)



Need “arithmetic intensity”

- Using data faster than it can be loaded causes stalls



Roll of programming model

Encourage hardware-friendly structure

- Bulk operations

- Bandwidth matters most: structure code to maximize locality

- 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

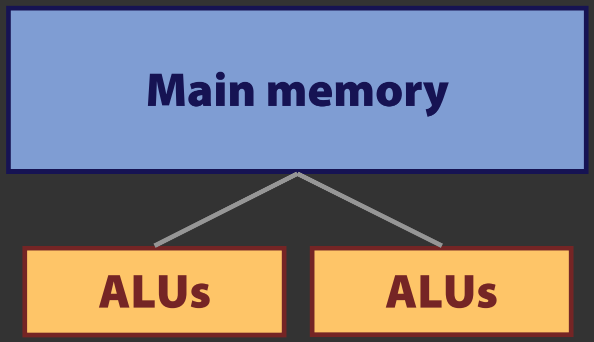
- Provide constructs that can be implemented efficiently without advanced compiler technology

- Get out of the way when needed

The idea

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

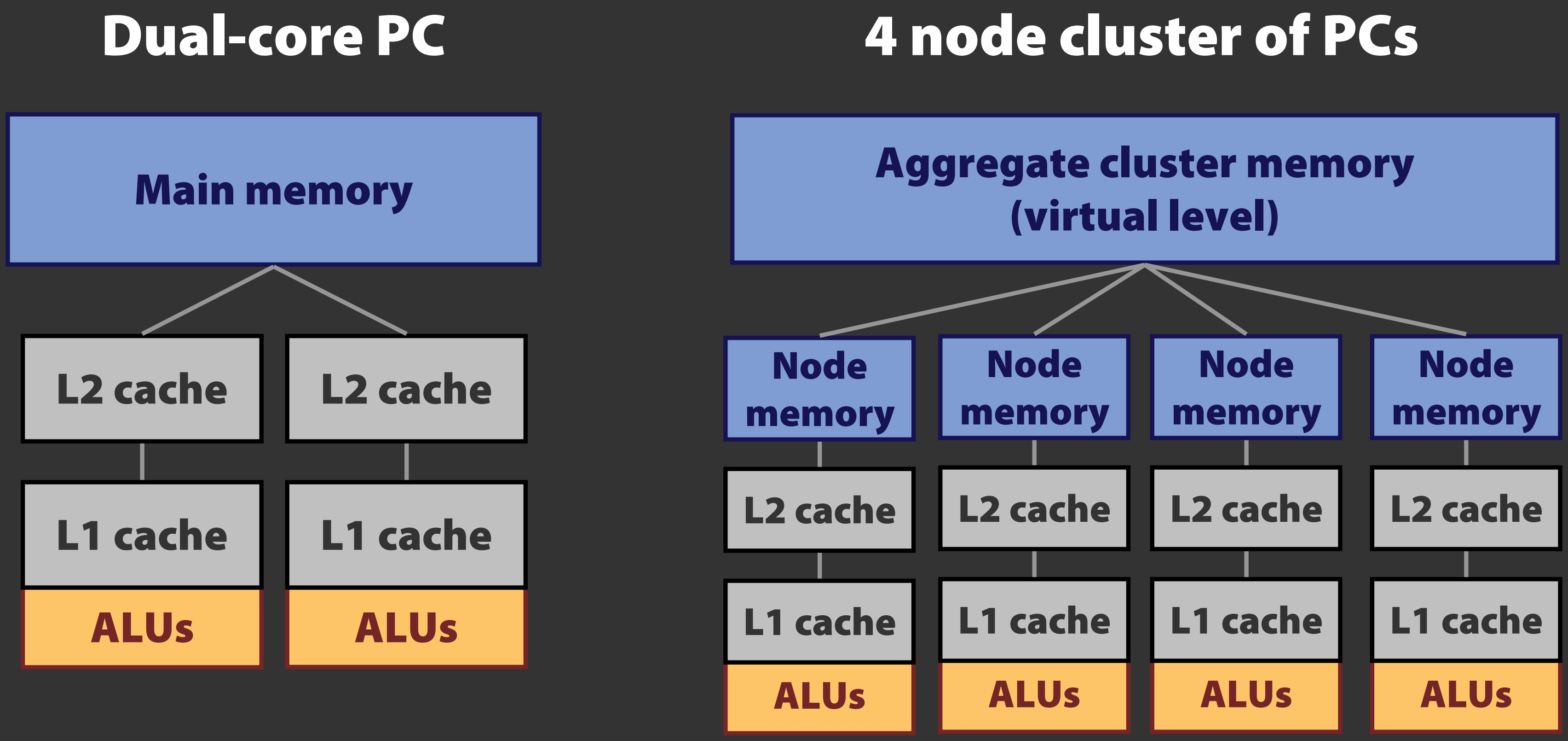
Dual-core PC



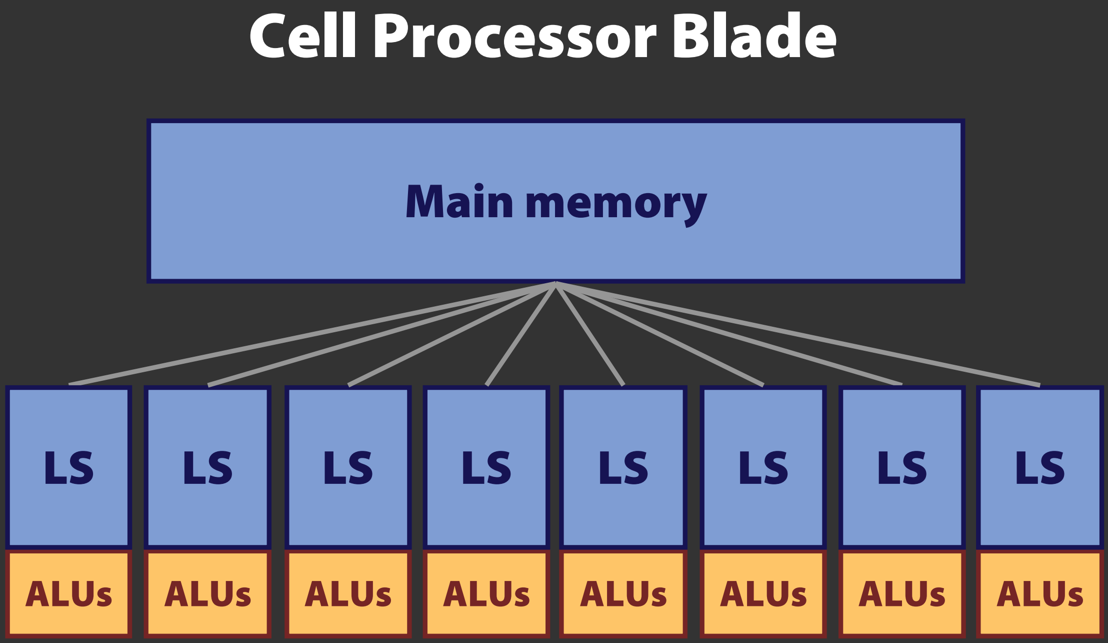
Similar to:
Parallel Memory Hierarchy Model
(Alpern et al.)

The idea

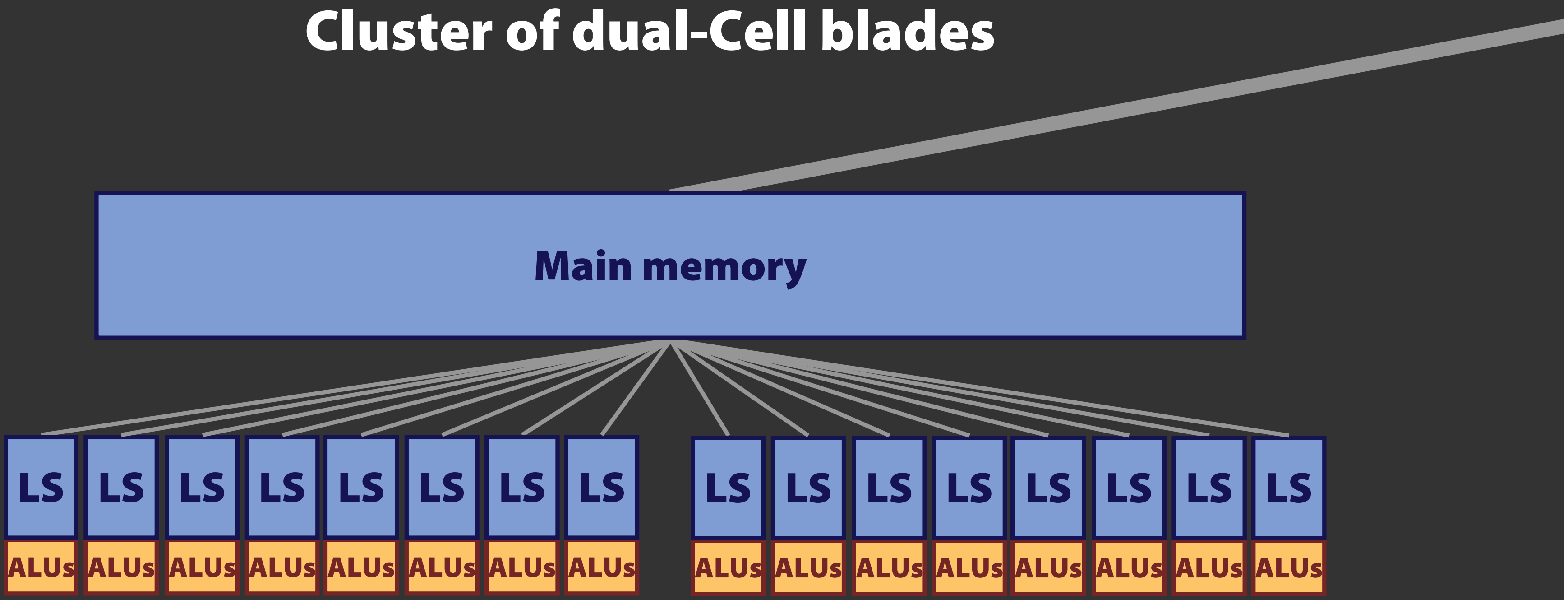
- Abstract machines a trees of memories



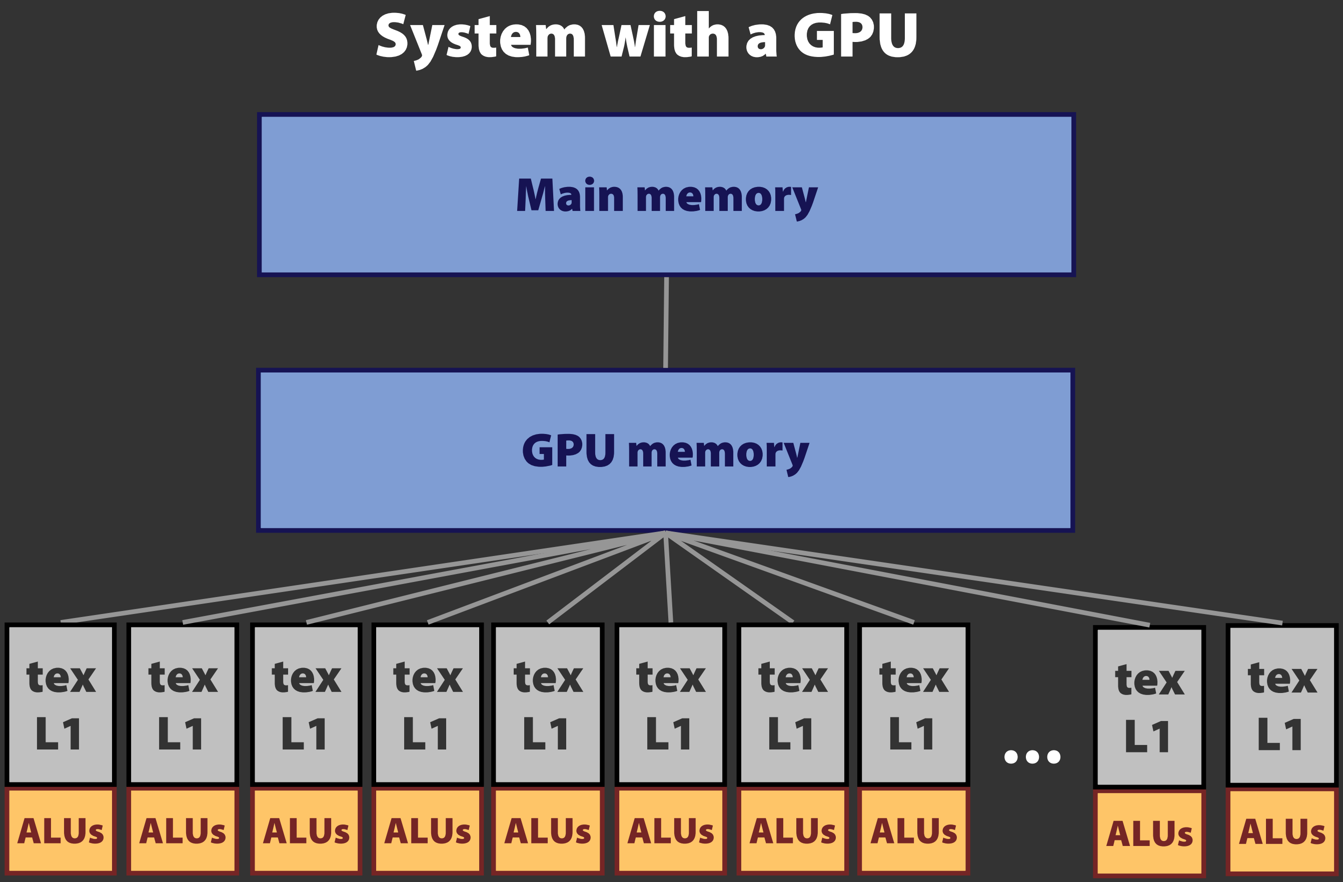
The idea


Cell Processor Blade
Main memory
LS
LS
LS
LS
LS
LS
LS
LS
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs

The idea


Cluster of dual-Cell blades
Main memory
LS
LS
LS
LS
LS
LS
LS
LS
LS
LS
LS
LS
LS
LS
LS
LS
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs

The idea


System with a GPU
Main memory
GPU memory
tex L1
tex L1
tex L1
tex L1
tex L1
tex L1
tex L1
tex L1
...
tex L1
tex L1
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
ALUs
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**

```
task lerp(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**
- **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

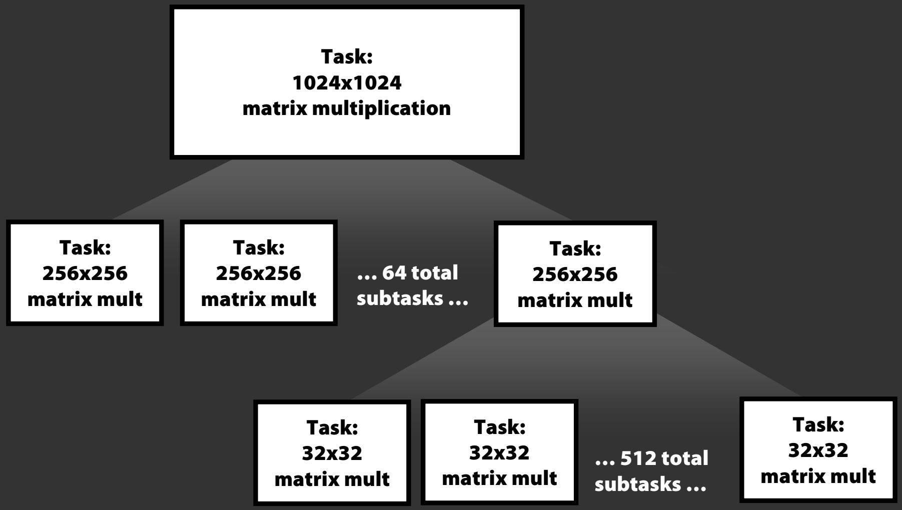
```
task foo(in float A[N], out float B[N])
{
  bar(A[0:N/2], B[0:N/2]);
  bar(A[N/2:N], B[N/2:N]);
}


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

Task isolation

Locality

- Tasks express decomposition



Task:
1024x1024
matrix multiplication

Task:
256x256
matrix mult

Task:
256x256
matrix mult

... 64 total
subtasks ...

Task:
256x256
matrix mult

Task:
32x32
matrix mult

Task:
32x32
matrix mult

... 512 total
subtasks ...

Task:
32x32
matrix mult

Easy parallelism from isolation

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

```
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

```
task parallel_foo(in float A[N], out float B[N])
{
  int x = 128;
  mappar(int i=0 to N/X) {
       bar( A[X*i : X*(i+1)], B[X*i : X*(i+1)] );
  }
}
```

A and B in
main memory
N= unbounded

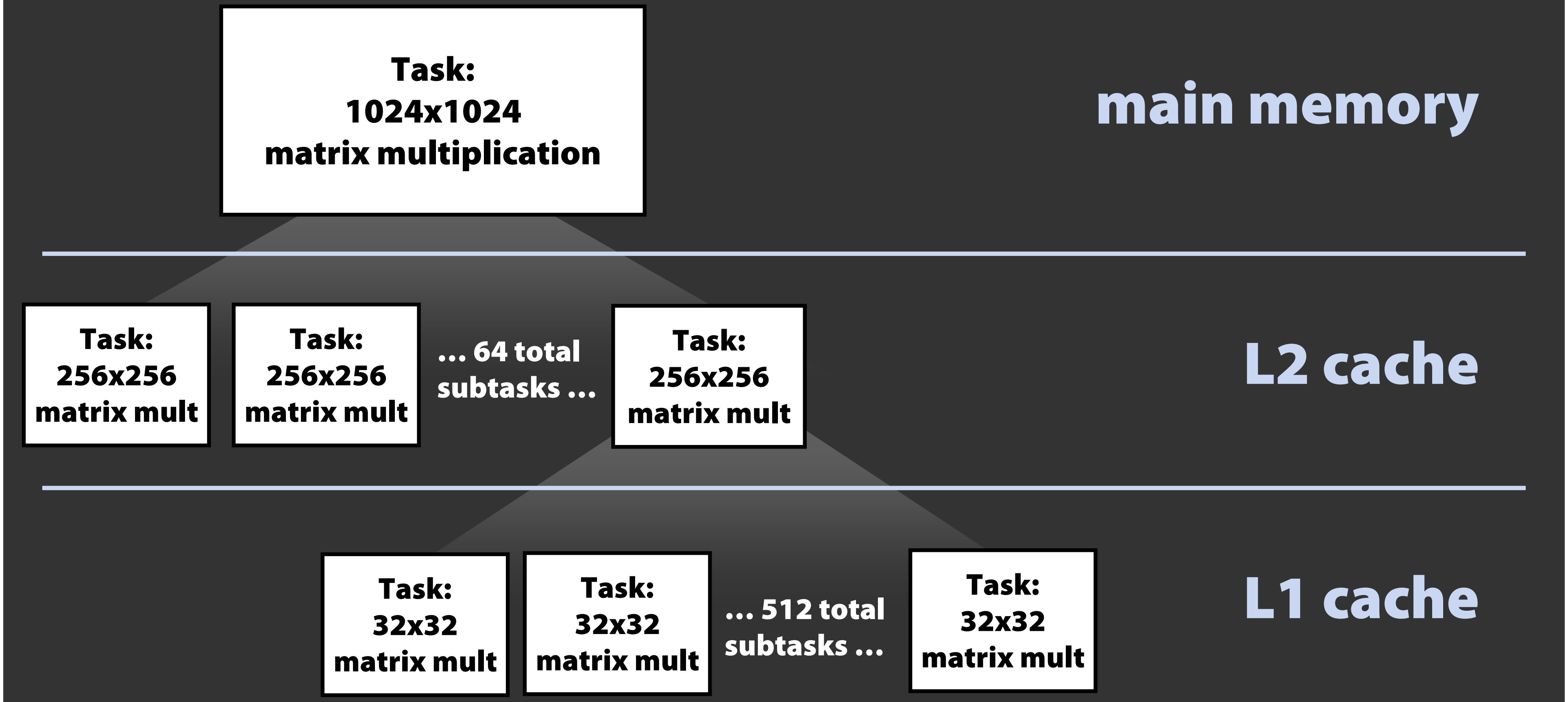
```
task bar(in float A[N], out float B[N])
{
  …
}
```

A and B in cache
N = 10

Task parameterization

- Tasks are parameterized for adaptability
- Allow multiple implementations (variants) of a single task

Example: dense matrix multiplication



Example: Task isolation

```
task matmul::inner(in    float A[M][T],
                   in    float B[T][N],
                   inout float C[M][N])
{


}
```

- Task arguments + local variables define working set

Example: parameterization

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







}
```

- Tasks are written in parameterized form for portability

- Different "variants" of the same task can be defined

```
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];
}
```

Here is a "leaf version" of the matmul task. It doesn't call subtasks.

Locality & communication

```
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]);
     }
  }
}
```

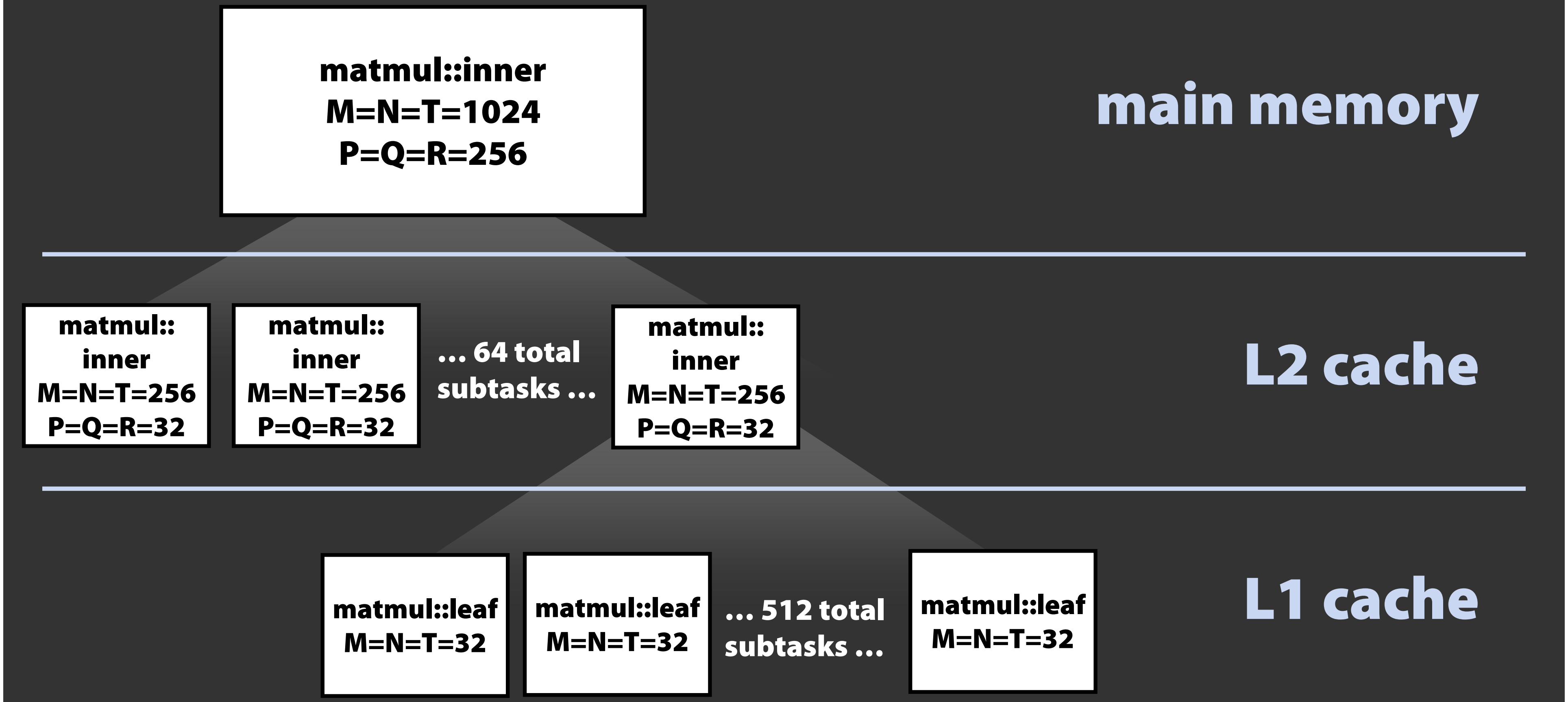
- **Working set resident within single level of hierarchy**

```
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];
}
```

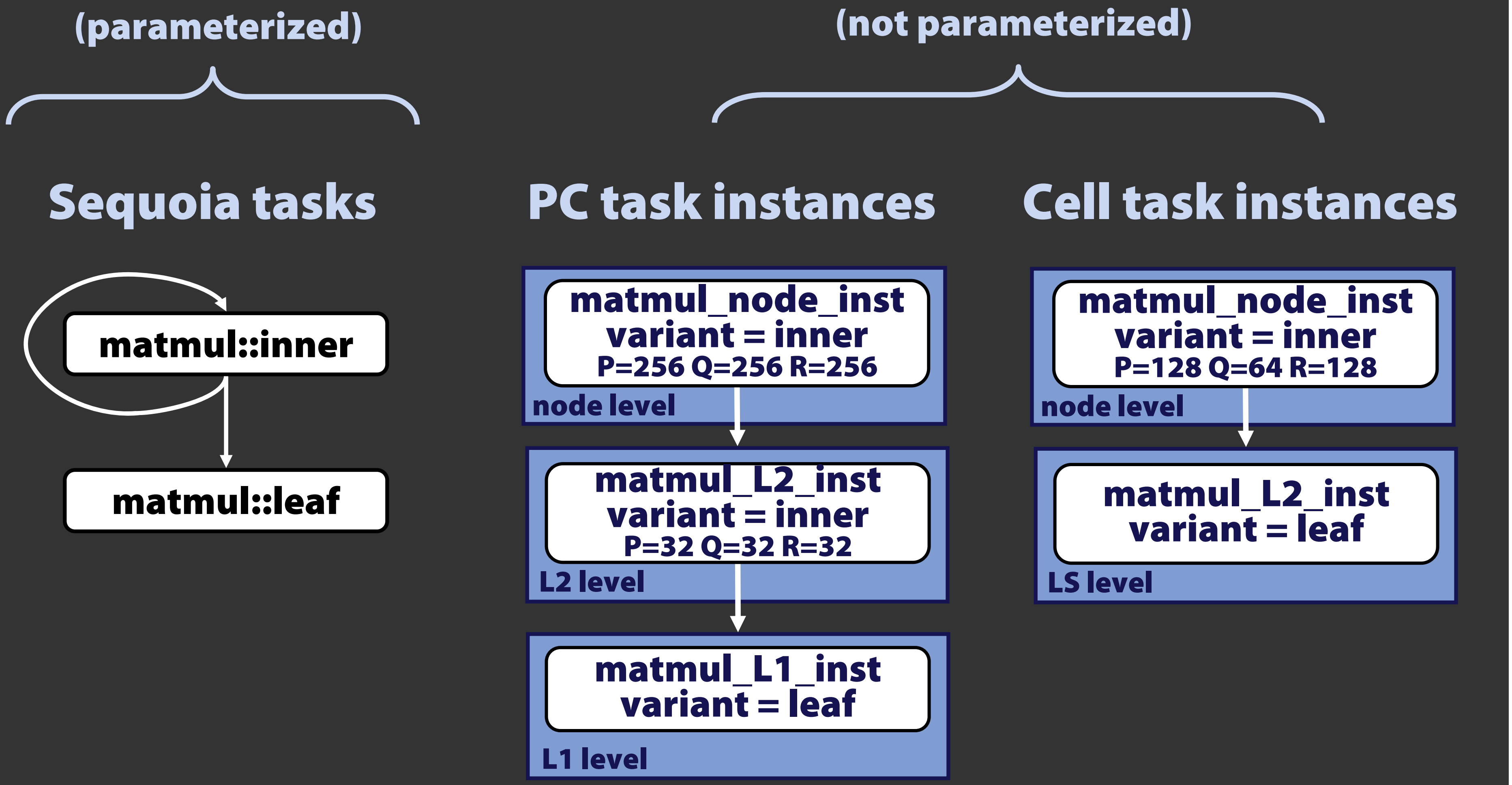
- **Passing arguments to subtasks is only way to specify communication in Sequoia**

Specializing matmul

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



Task instances



(parameterized)

(not parameterized)

Sequoia tasks

matmul::inner

matmul::leaf

PC task instances

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

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);
}
```

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)

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

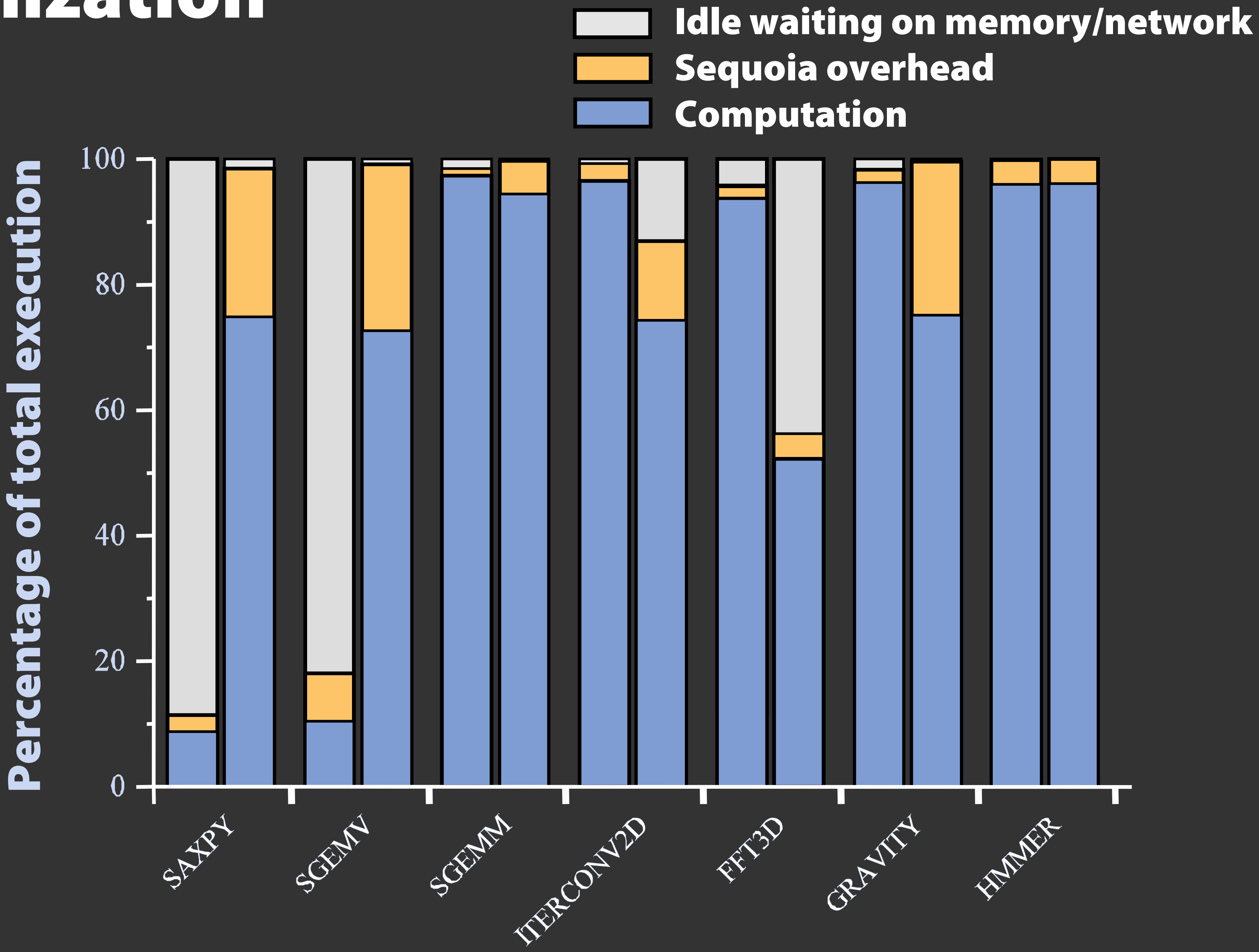
HMMER Fuzzy protein string matching using HMM evaluation

(Daniel Horn’s SC2005 paper)

Performance: 2.4 GHz Cell DD2 (in GFlops)

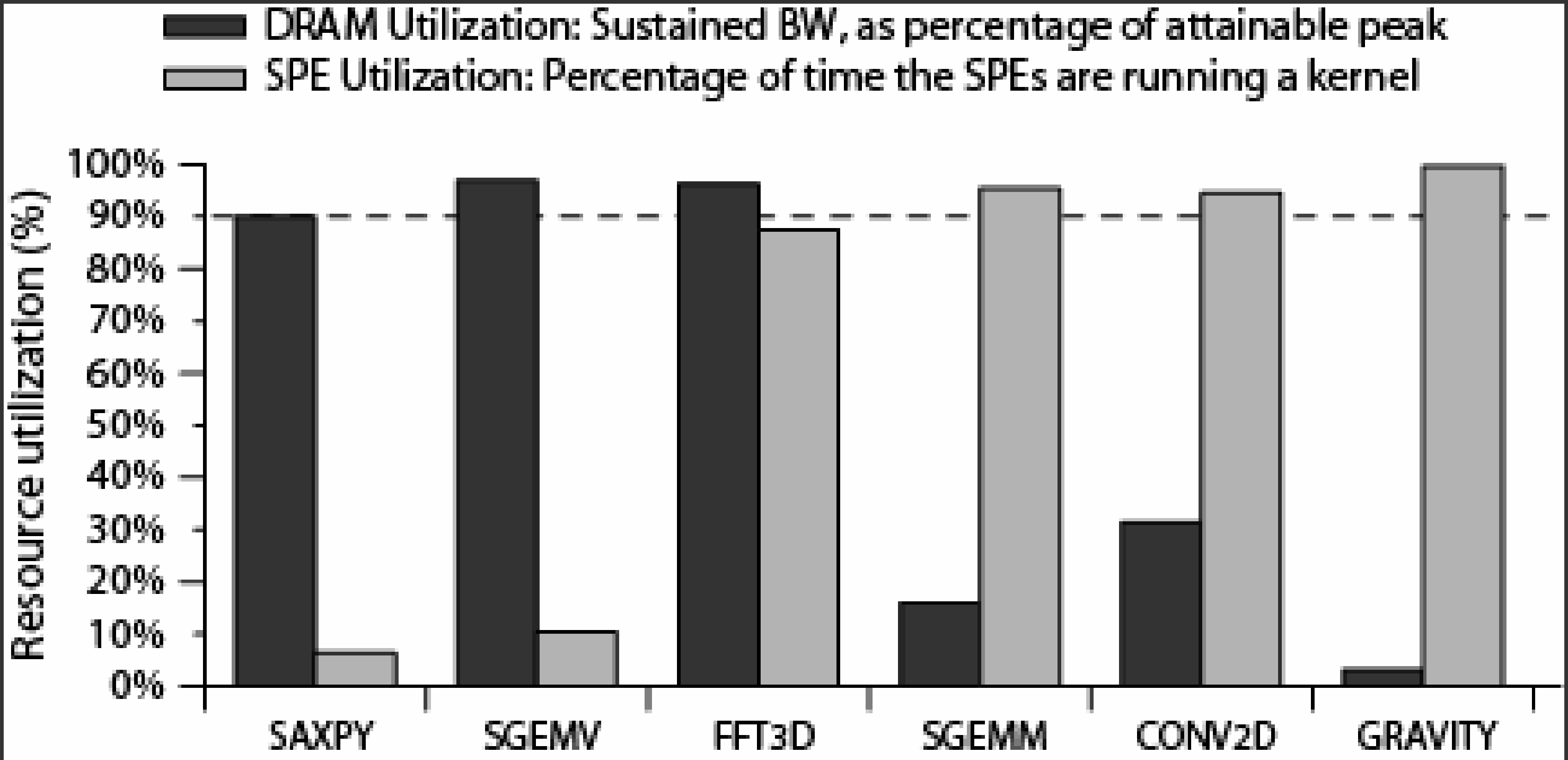
	Cell 8 SPE	Cell 16 SPE	Cluster	
			Pre-distrib	Overall
SAXPY	3.2 (22GB/s)	4.0	3.6	0.1
SGEMV	9.8 (18GB/s)	11.0	11.1	0.2
SGEMM	96.3	174	97.9	72.5
IterConv2D	62.8	119	27.2	19.9
FFT3D	43.5	45.2	6.8	1.98
Gravity	83.3	142	50.6	50.5
HMMER	9.9	19.1	13.4	12.7

Utilization



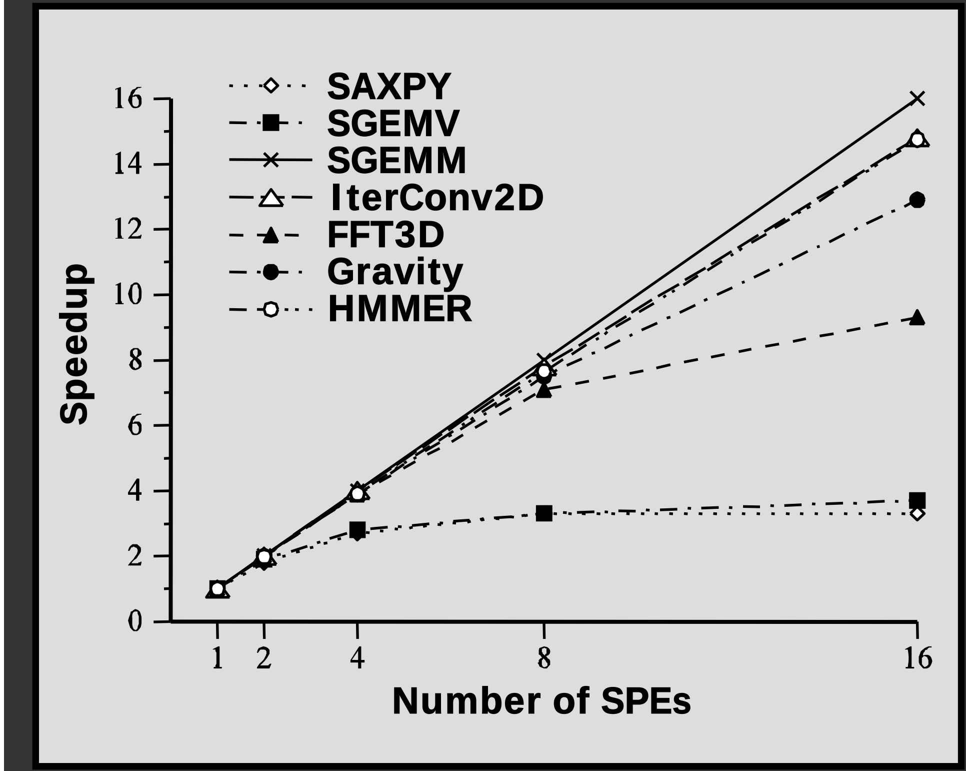
Execution on a Cell blade (left bars) and 16 node cluster (right bars)

Cell utilization

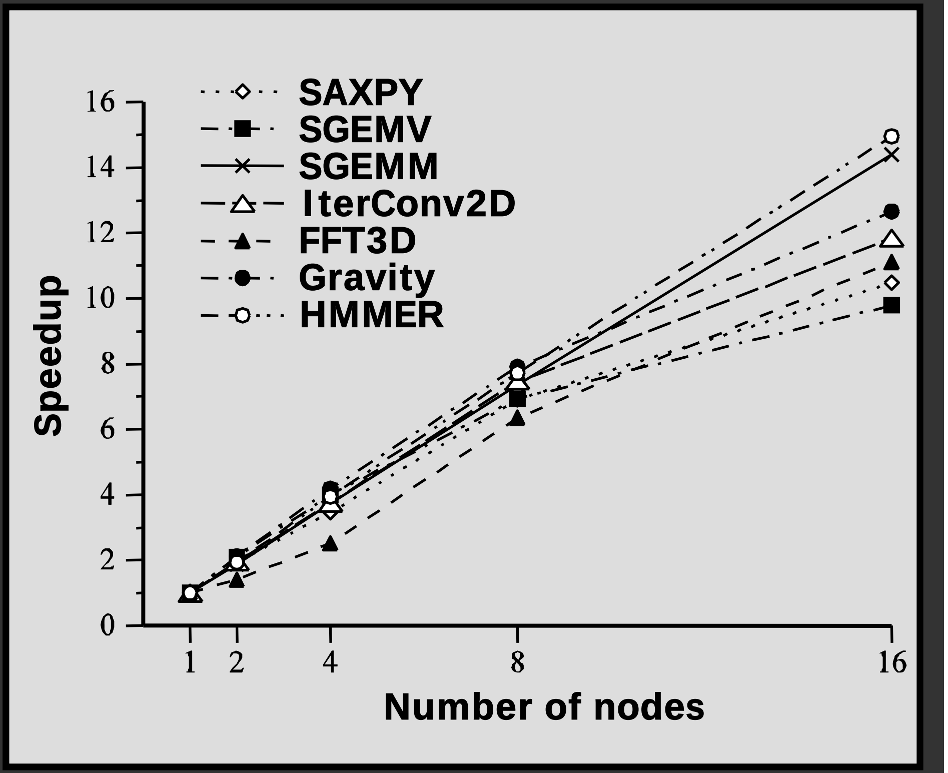


Performance scaling

SPE scaling on 2.4Ghz Dual-Cell blade



Scaling on P4 cluster with Infiniband interconnect



Key ideas

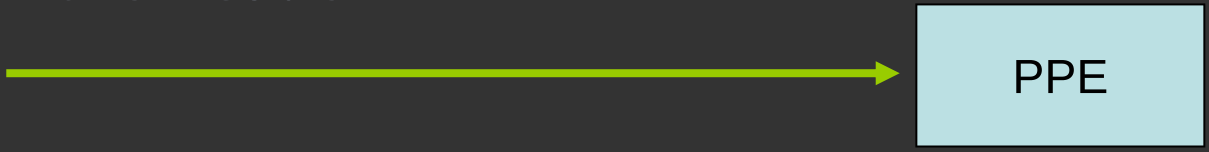
- Incorporate hierarchal memory tightly into programming model
  - Programming memory hierarchy

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

- Leverage task abstraction for critical properties of application

How this all works

- Back to SGEMM example:
  - User initializes Sequoia and allocates data from their code

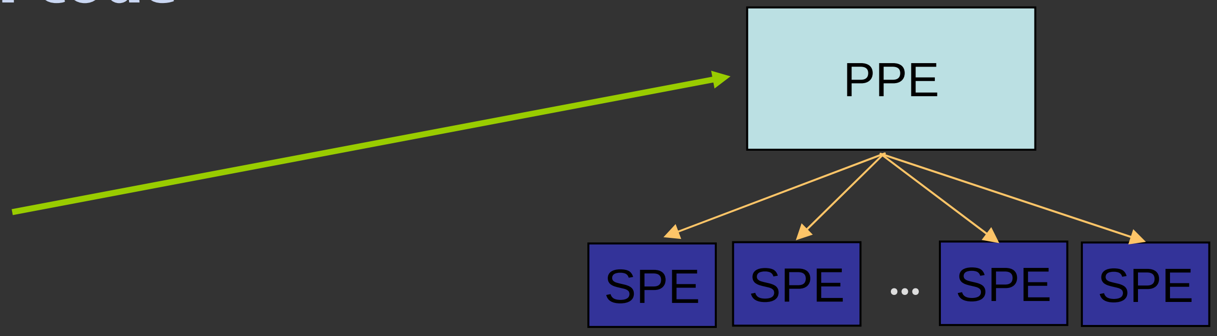


```
main()
{
    sqInit();
    ...
    A = sqAlloc2D(...);
    B = sqAlloc2d(...);
    C = sqAlloc2d(...);
    ...
    matmul(A,B,C);
    ...
    sqShutdown();
}
```

How this all works

- **Back to SGEMM example:**
  - **User initializes Sequoia and allocates data from their code**

```
main()
{
    sqInit();
    ...
    A = sqAlloc2D(...);
    B = sqAlloc2d(...);
    C = sqAlloc2d(...);
    ...
    matmul(A,B,C);
    ...
    sqShutdown();
}
```



PPE launches bootstrap threads on SPEs

How this all works

- **Back to SGEMM example:**
    - **User initializes Sequoia and allocates data from their code**

```
main()
{
    sqInit();
    ...
    A = sqAlloc2D(...);
    B = sqAlloc2d(...);
    C = sqAlloc2d(...);
    ...
    matmul(A,B,C);
    ...
    sqShutdown();
}
```

Allocate data

How this all works

- **Back to SGEMM example:**
    - **User initializes Sequoia and allocates data from their code**

```
main()
{
    sqInit();
    ...
    A = sqAlloc2D(...);
    B = sqAlloc2d(...);
    C = sqAlloc2d(...);
    ...
    matmul(A,B,C);
    ...
    sqShutdown();
}
```

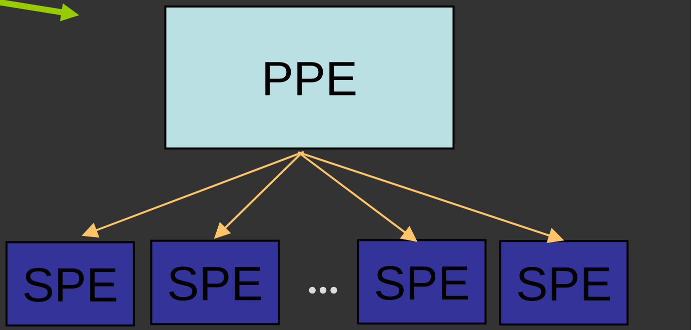
Call task

Top level task call

```
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]);
     }
  }
}
```



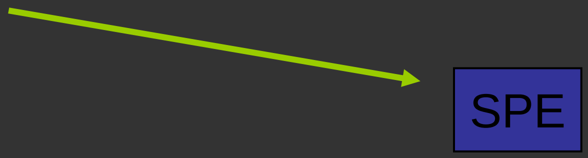
PPE mails SPE leaf task to instruct olay load and execution

Leaf task call

```
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]);
     }
  }
}
```



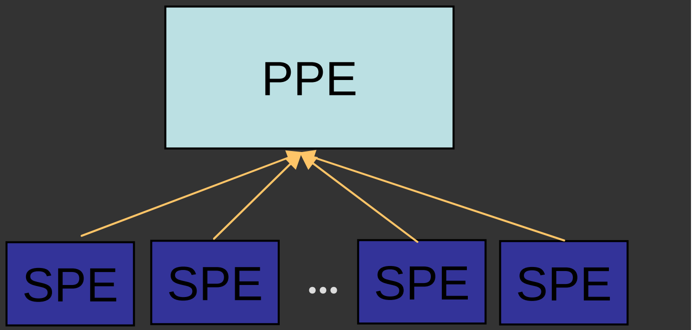
SPE id controls assignment of iteration space and DMA list offsets

Leaf task return

```
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]);
     }
  }
}
```



SPE mails PPE and waits for command

Control return to user code

- **Back to SGEMM example:**
  - **User initializes Sequoia and allocates data from their code**

```
main()
{
    sqInit();
    ...
    A = sqAlloc2D(...);
    B = sqAlloc2d(...);
    C = sqAlloc2d(...);
    ...
    matmul(A,B,C);
    ...
    sqShutdown();
}
```

Kill off threads and cleanup

Questions?