Sequoia

Mike Houston
Stanford University
July 9th, 2007 - CScADS Workshop

Today's outline

- Sequoia programming model
- Sequoia targets
- Tuning in Sequoia
- http://sequoia.stanford.edu
  - Supercomputing 2006 paper - Language
  - PPoPP 2007 paper - Compiler
  - Runtime and backend system papers under review
  - Auto-tuning research/thesis in progress

Emerging Themes

Writing high-performance code amounts to…

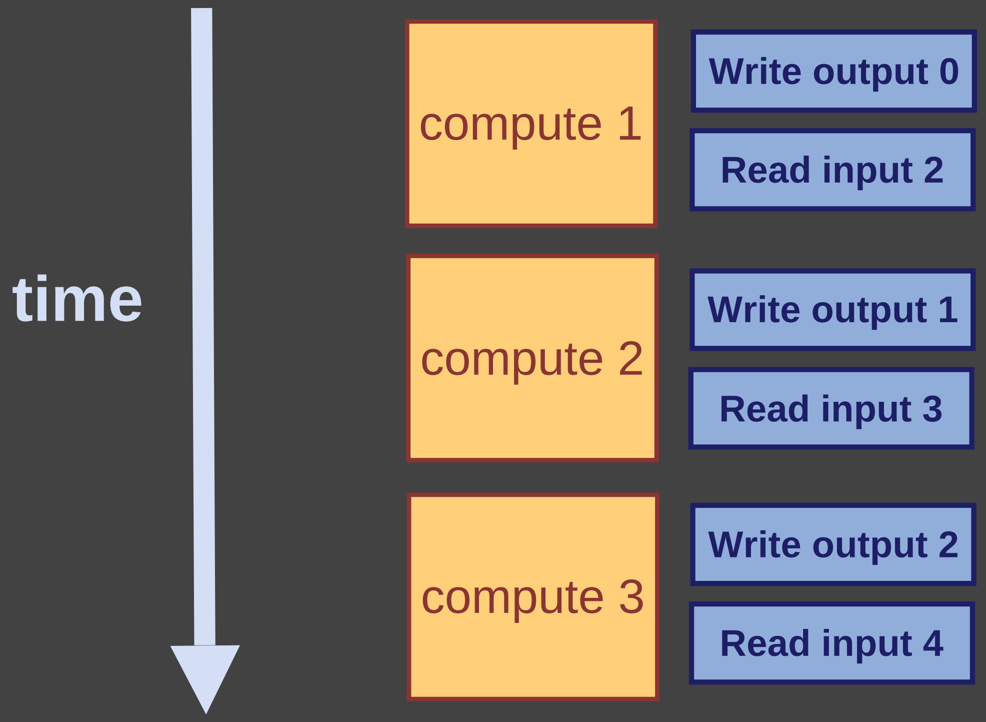
Intelligently structuring algorithms
[compiler help unlikely]

Efficiently using parallel resources
[compilers struggle without help]

Generating efficient inner loops (kernels)
[compilers coming around]

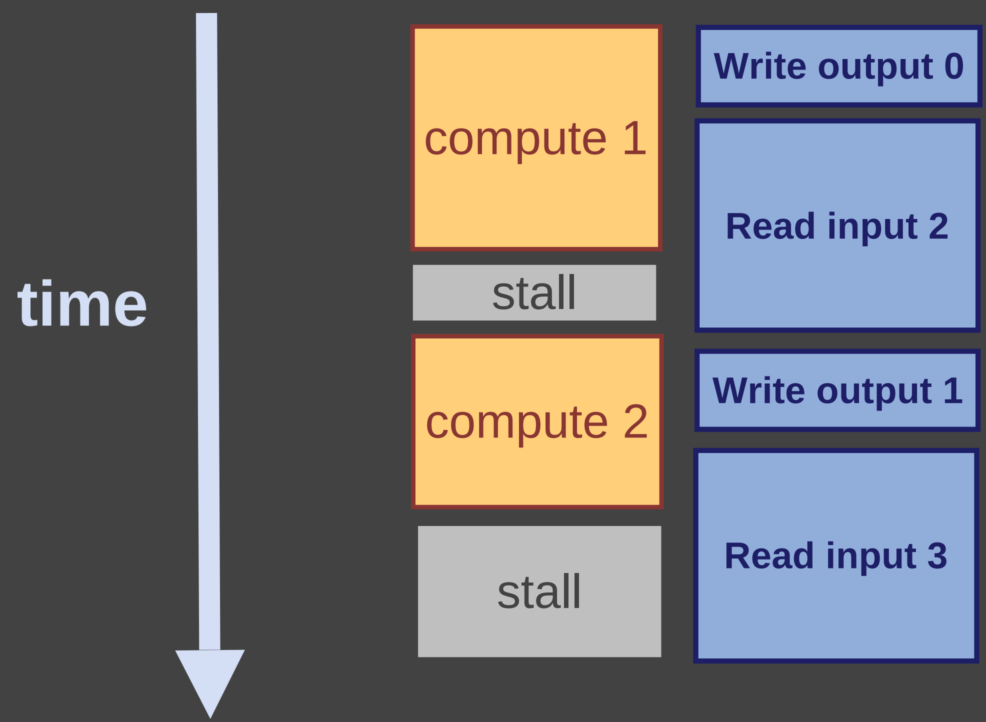
It's about program structure

- Preload batch of data
- Compute on data
- Initiate write of results (this data is done)
- 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: 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

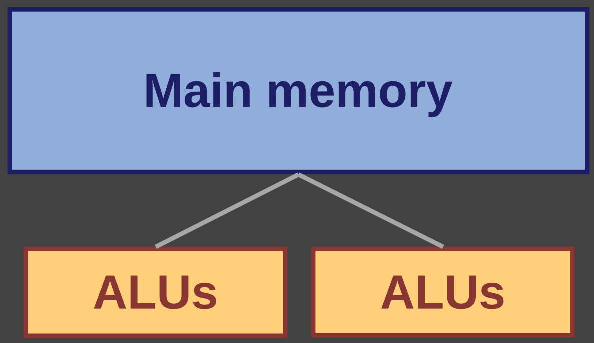
Sequoia's goals

- 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

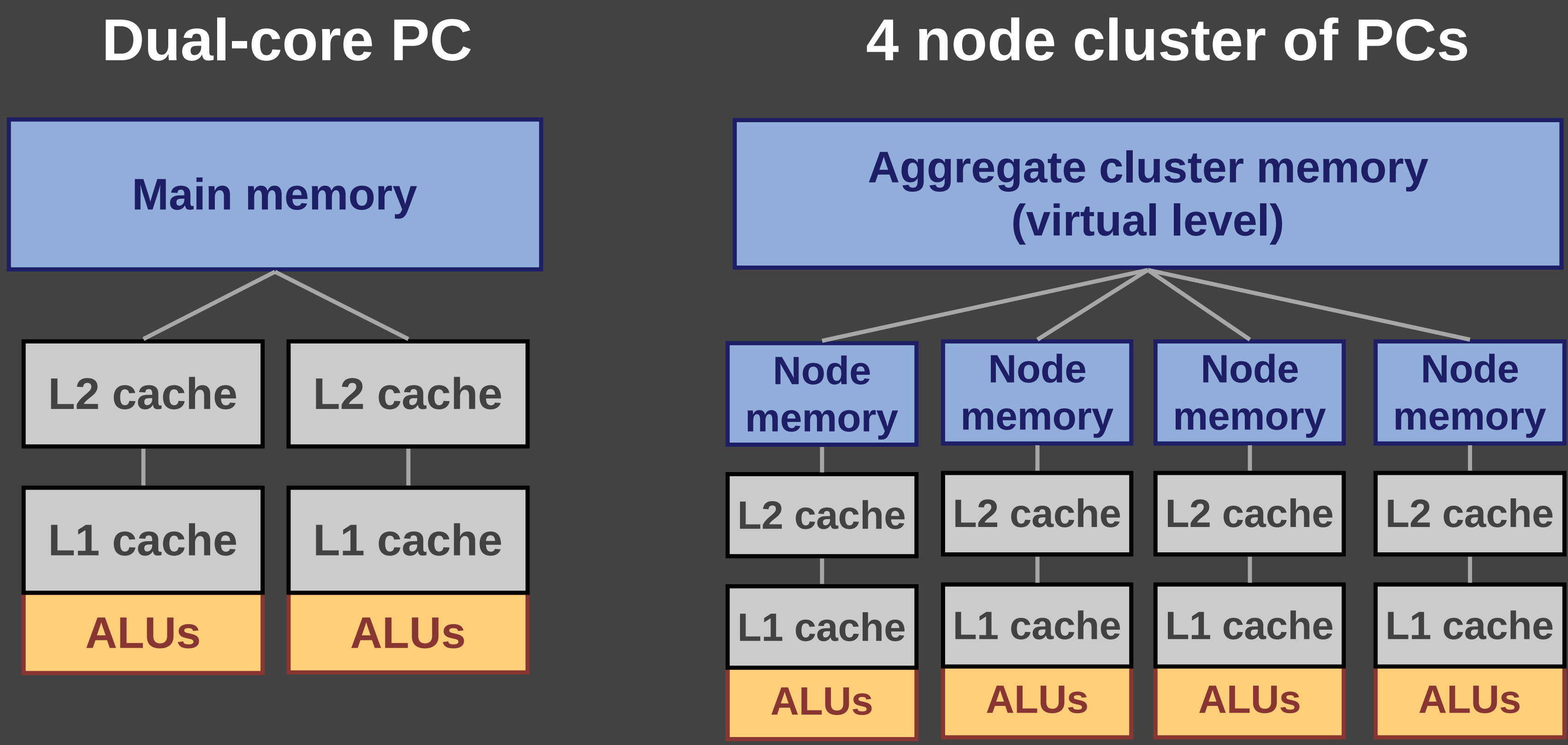


Similar to:

Parallel Memory Hierarchy Model (Alpern et al.)

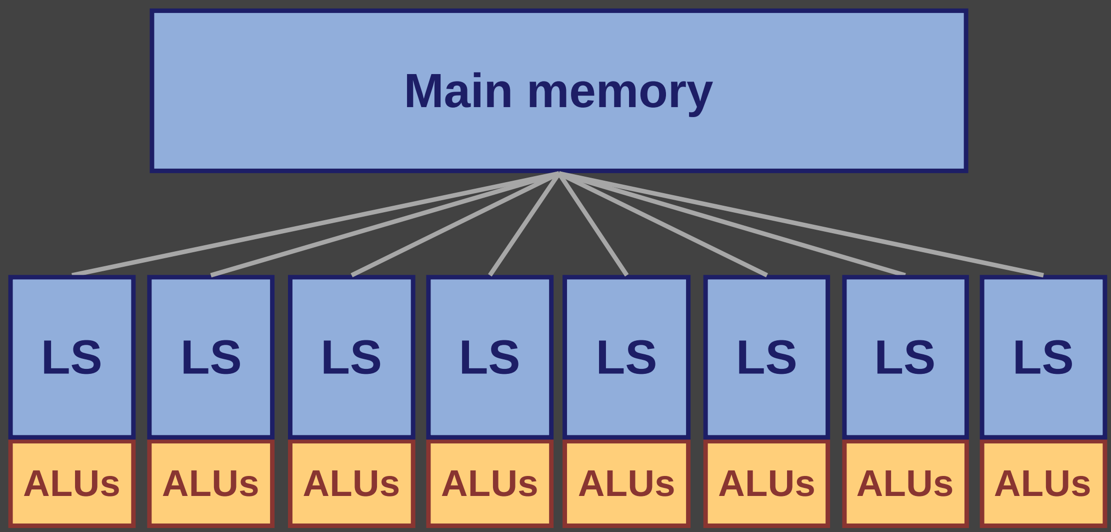
The idea

- Abstract machines a trees of memories

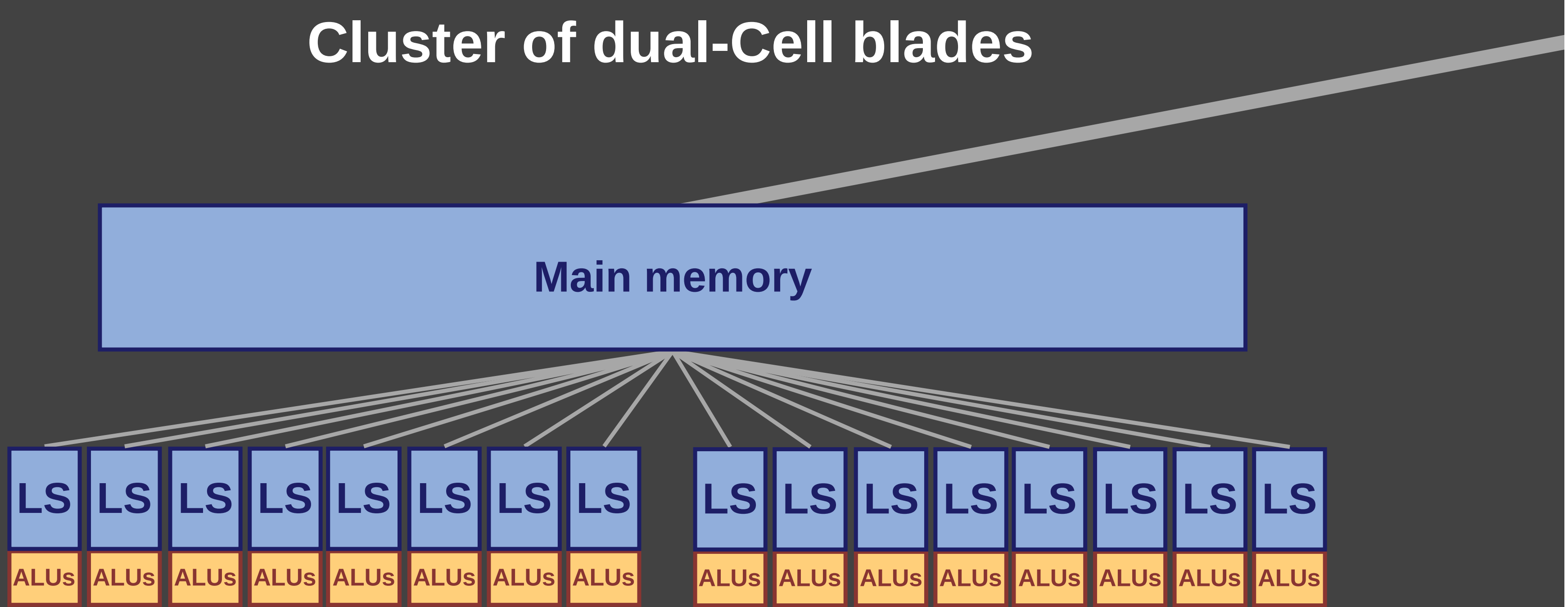


The idea

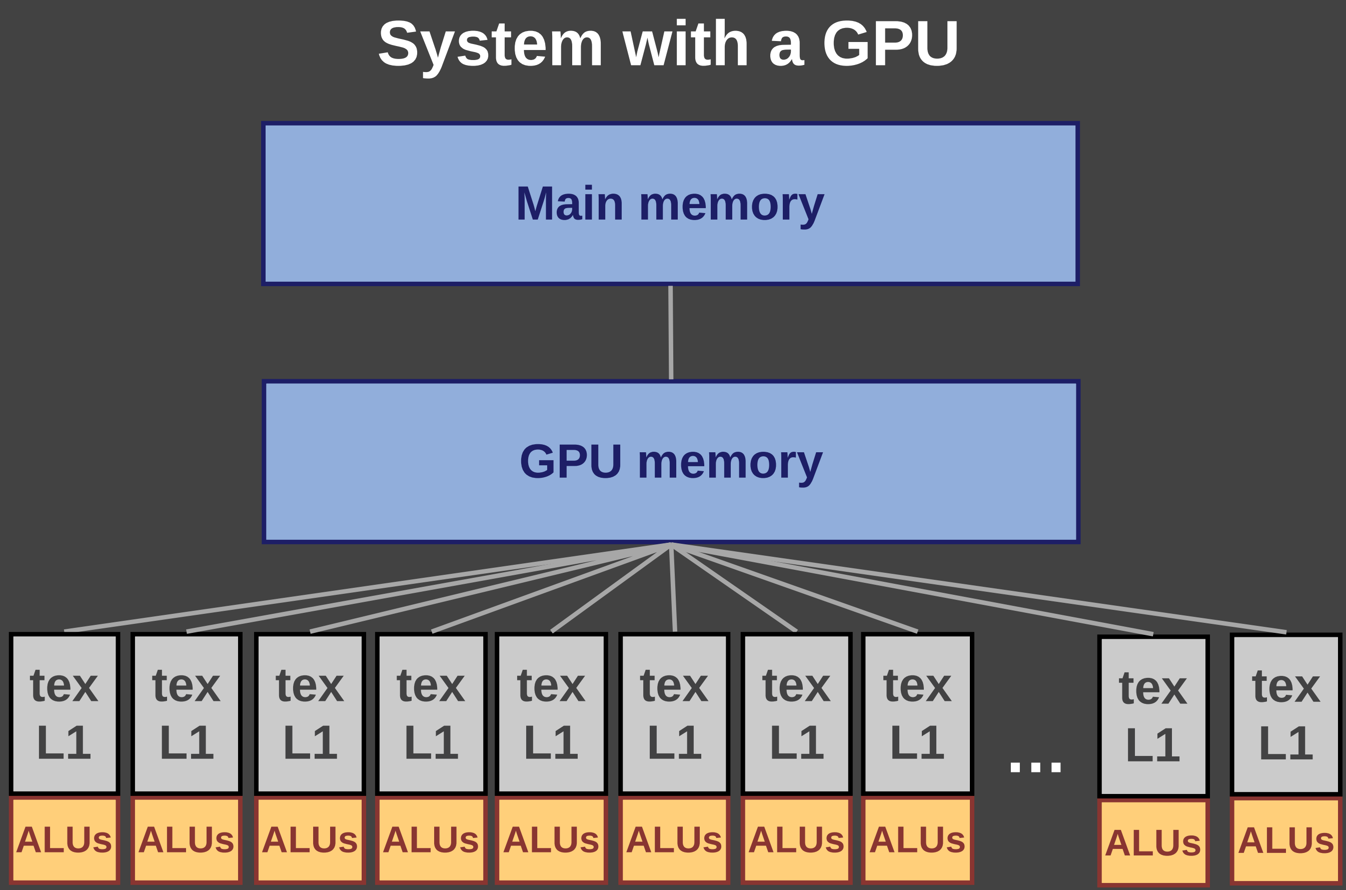
Cell Processor Blade



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



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 interpolate(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 and scalars**
- **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

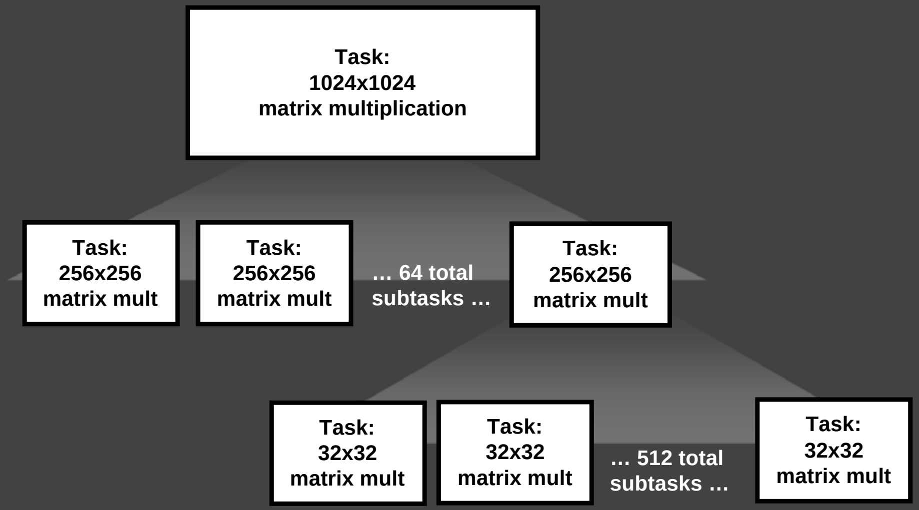
```
task foo(in float A[N], out float B[N])
{
  bar(A[0:N/2], B[0:N/2]);
  bar(A[N/2:N], B[N/2:N]);
}


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

Task isolation

Locality

- Tasks express decomposition



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 = 10;
  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

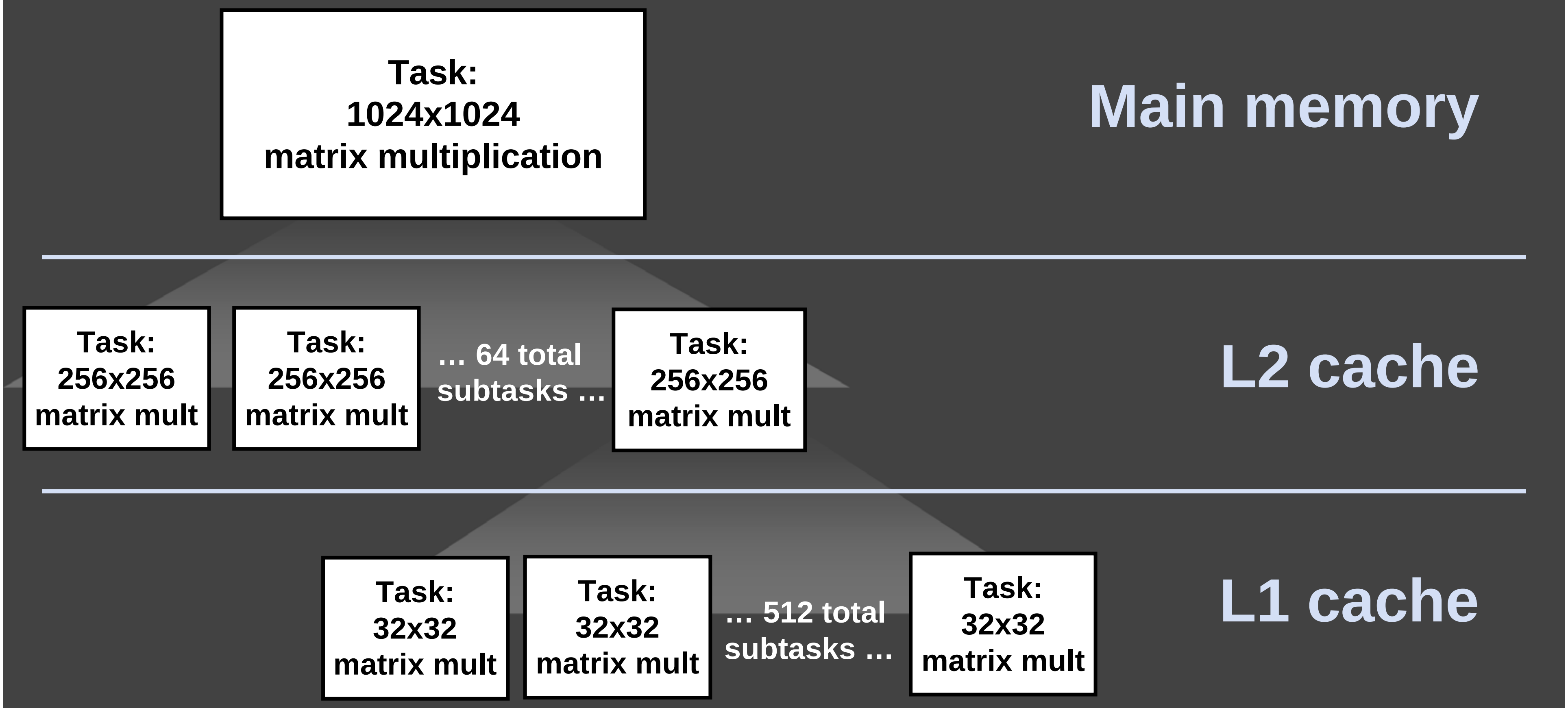
```
task bar(in float A[N], out float B[N])
{
  …
}
```

A and B in cache N = 10

Task parameterization

- Tasks are parameterized for adaptability
  - Allows tuning to each machine
    - “Auto” tuning possible
  - Task/algorithm description separate from machine mapping
- 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.

Example - 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]);

     }
  }
}
```

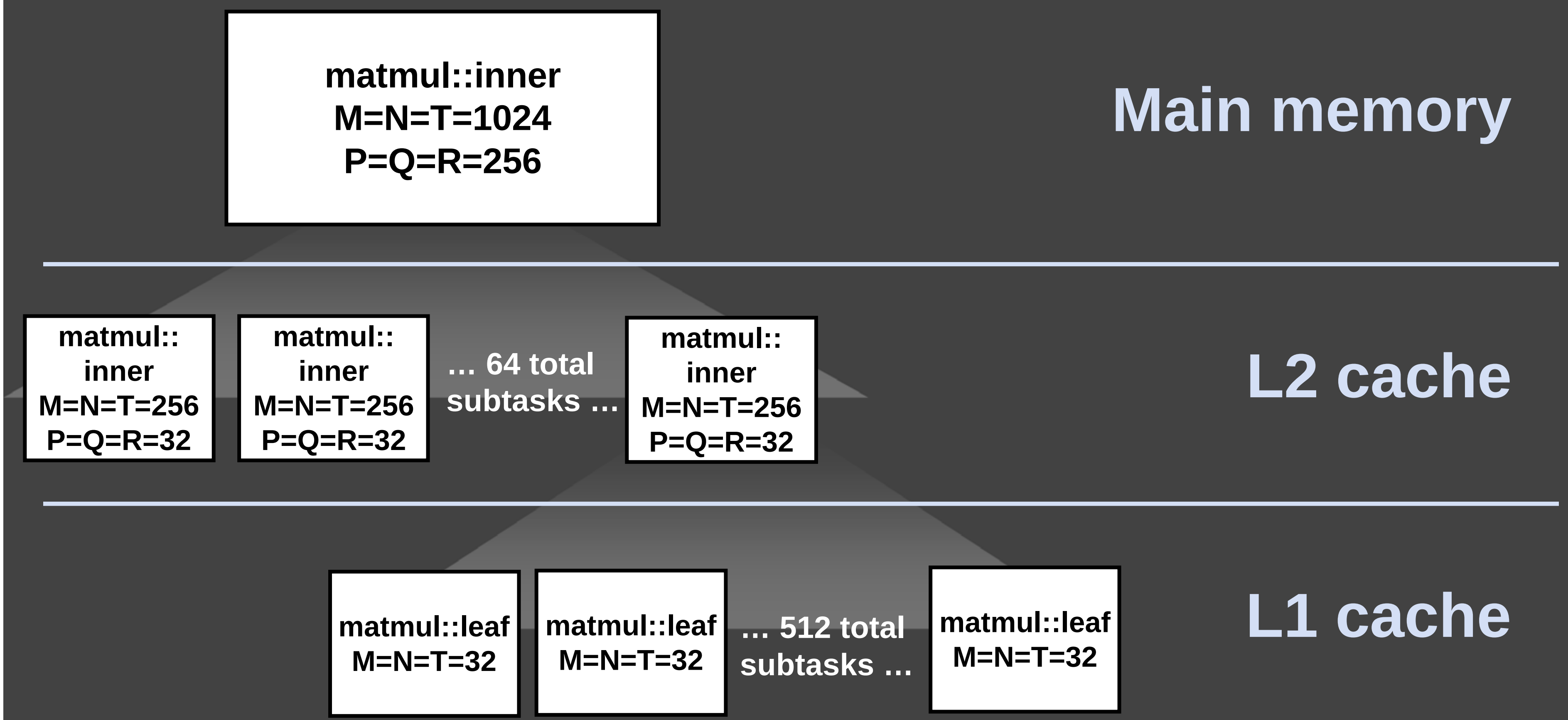
- Working set resident within single level of hierarchy

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

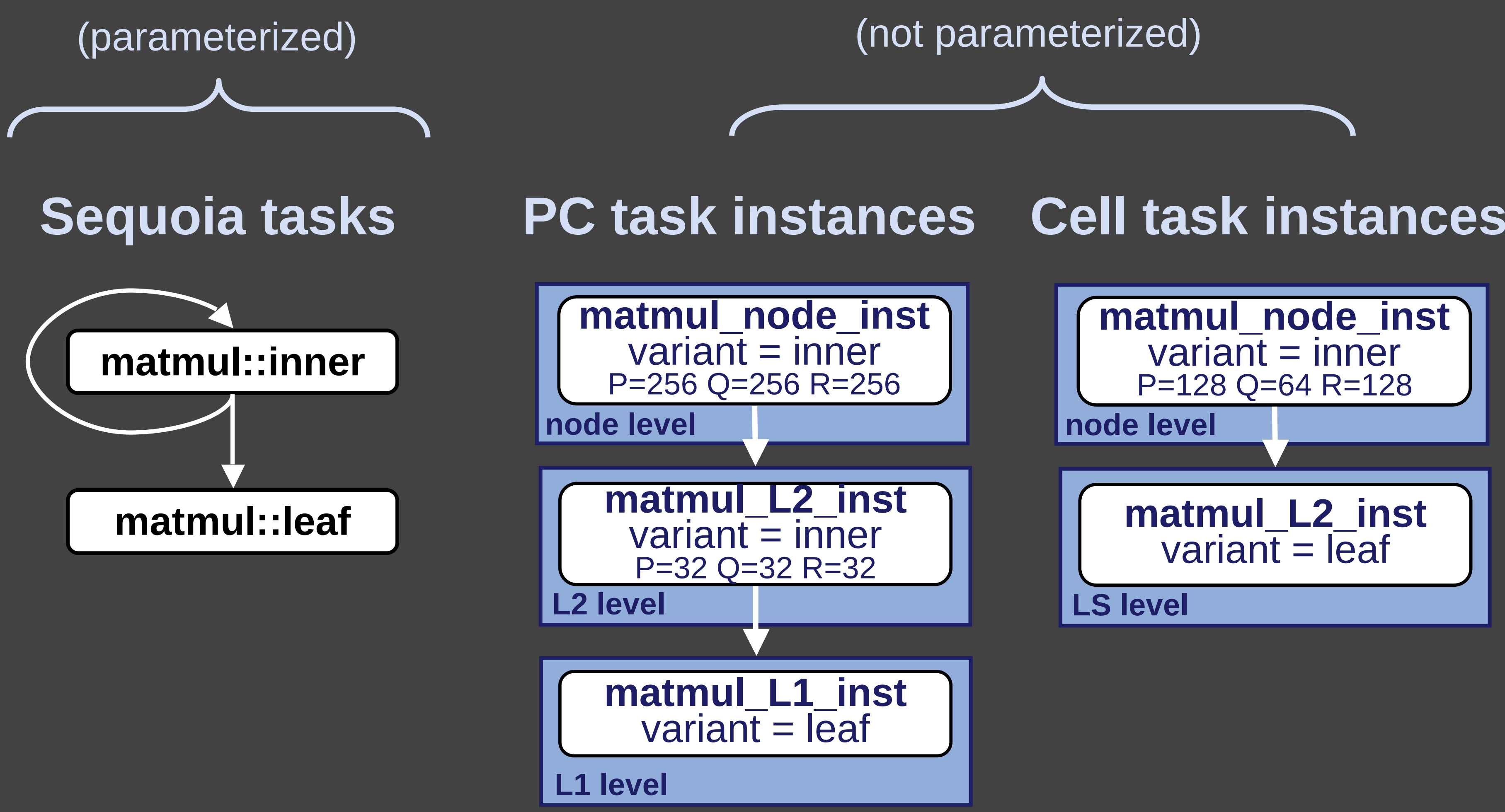
```
task matmul::leaf(in    float A[M][T],
                  in    float B[T][N],
                  inout float C[M][N])
{
   for (int i=0; i<M; i++)
     for (int j=0; j<N; j++)
       for (int k=0;k<T; k++)
          C[i][j] += A[i][k] * B[k][j];
}
```

Specializing matmul

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



Task instances



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);
}
```

Results

- We have a Sequoia compiler + runtime systems for multiple platforms (runtimes actively being researched)
  - 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**

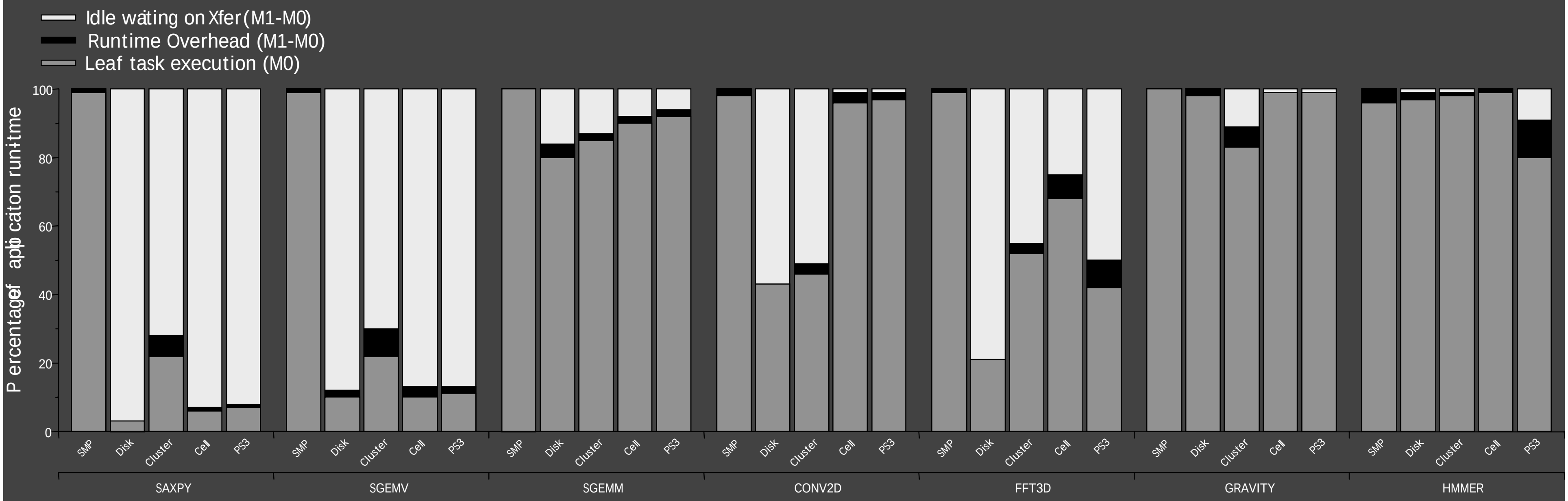
Scientific computing benchmarks

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)

System configurations

- Disk
  - 2.4 GHz Intel P4, 160GB disk, ~50MB/s from disk
- 8-way SMP
  - 4 dual-core 2.66 Intel P4 Xeons, 8GB
- Cluster
  - 16, 2-way Intel 2.4GHz P4 Xeons, 1GB/node, Infiniband
- Cell
  - 3.2 GHz IBM Cell blade (1 Cell - 8SPE), 1GB
- PS3
  - 3.2 GHz Cell in Sony Playstation 3 (6 SPE), 256MB (160MB usable)

2 Level Utilization



SMP, Disk, Cluster, Cell, PS3 left to right for each application

Applications: saxpy, sgemv, sgemm, conv2d, fft3d, gravity, HMMer

Results – Horizontal portability - GFlop/s

	Scalar	SMP	Disk	Cluster	Cell	PS3
SAXPY	0.3	0.7	0.007	1.4	3.5	3.1
SGEMV	1.1	1.7	0.04	3.8	12	10
SGEMM	6.9	45	5.5	91	119	94
CONV2D	1.9	7.8	0.6	24	85	62
FFT3D	1.5	7.8	0.1	7.5	54	31*
GRAVITY	4.8	40	3.7	68	97	71
HMMER	0.9	11	0.9	12	12	7.1*

* Reduced dataset size to fit in memory 128^3 3D FFT, random 160MB subset of NCBI

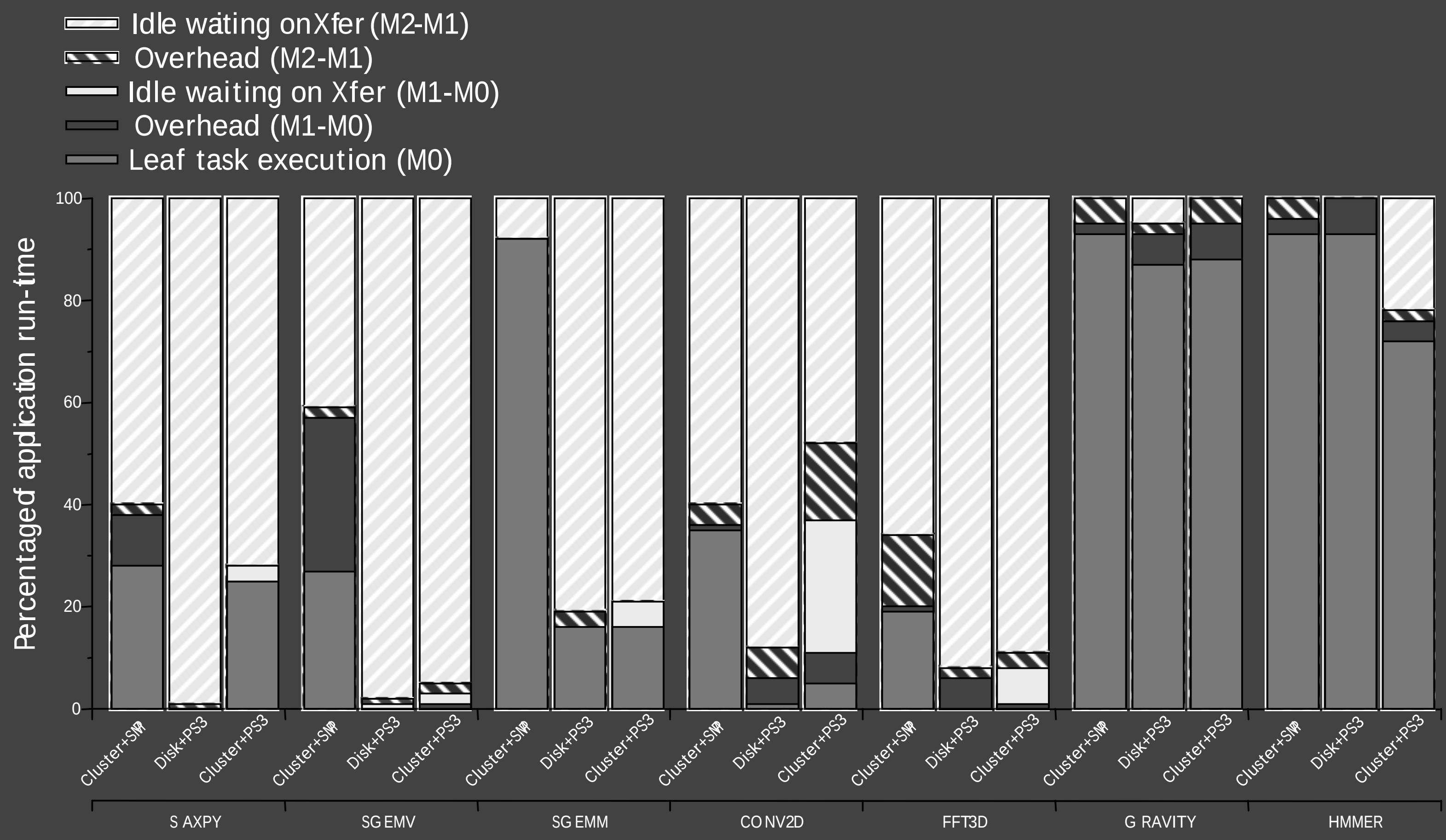
Results – Horizontal portability - GFlop/s

	Scalar	SMP	Disk	Cluster	Cell	PS3
SAXPY	0.3	0.7	0.007	1.4	3.5	3.1
SGEMV	1.1	1.7	0.04	3.8	12	10
SGEMM	6.9	45	5.5	91	119	94
CONV2D	1.9	7.8	0.6	24	85	62
FFT3D	1.5	7.8	0.1	7.5	54	31*
GRAVITY	4.8	40	3.7	68	97	71
HMMER	0.9	11	0.9	12	12	7.1*

Bandwidth bound

* Reduced dataset size to fit in memory 128^3 3D FFT, random 160MB subset of NCBI

Composed systems utilization



Cluster of SMPs, Disk and PS3, cluster of PS3s left to right for each application

Applications: saxpy, sgemv, sgemm, conv2d, fft3d, gravity, HMMer

Results – Vertical Portability - GFlop/s

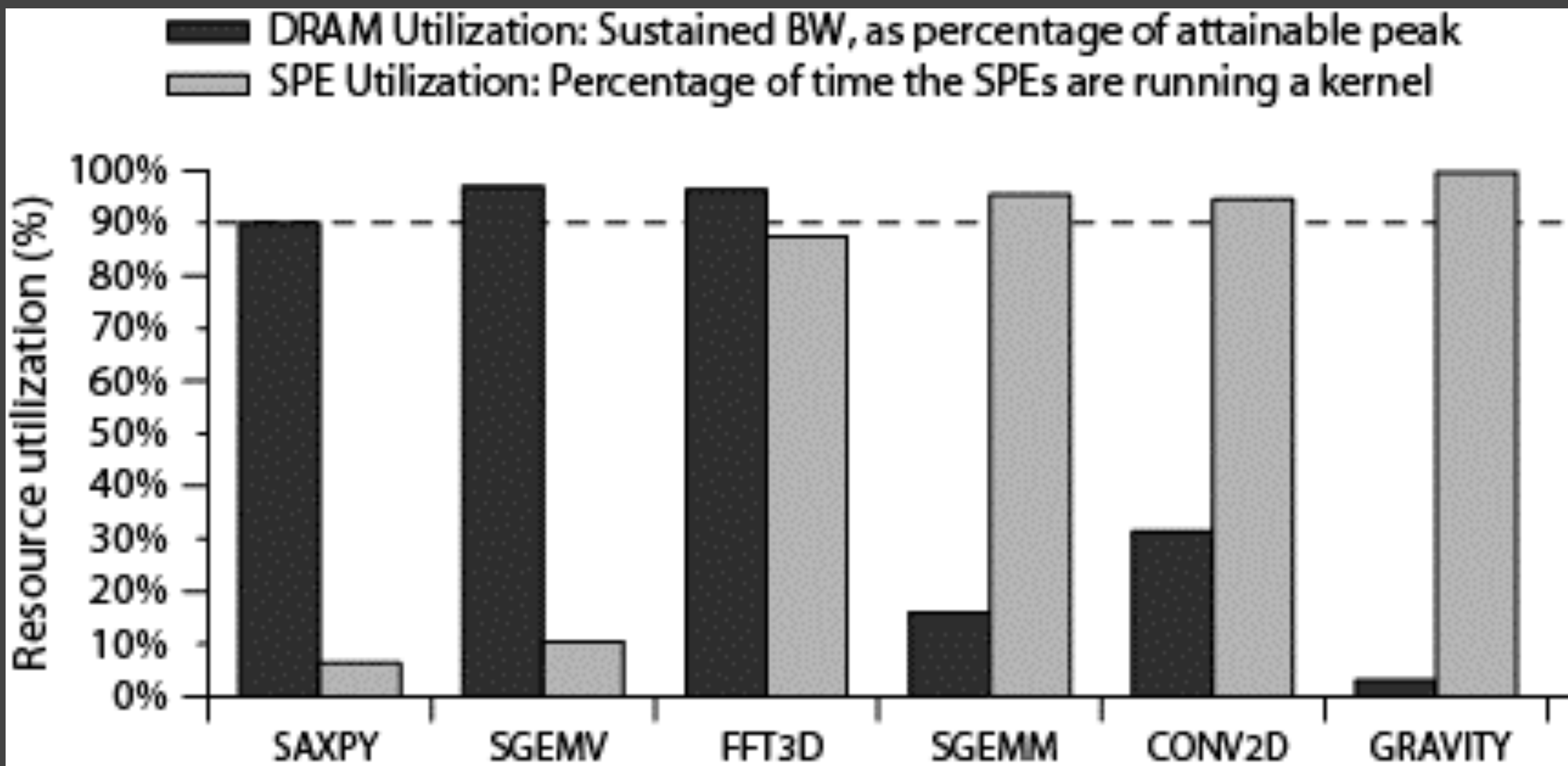
	Cluster-SMP	Disk+PS3	PS3 Cluster
SAXPY	0.5	0.004	0.23
SGEMV	1.4	0.014	1.3
SGEMM	48	3.7	30
CONV2D	4.8	0.48	3.24
FFT3D	2.1	0.05	0.36
GRAVITY	50	66	119
HMMER	14	8.3	13

Results – Vertical Portability - GFlop/s

	Cluster-SMP	Disk+PS3	PS3 Cluster
SAXPY	0.5	0.004	0.23
SGEMV	1.4	0.014	1.3
SGEMM	48	3.7	30
CONV2D	4.8	0.48	3.24
FFT3D	2.1	0.05	0.36
GRAVITY	50	66	119
HMMER	14	8.3	13

Bandwidth bound

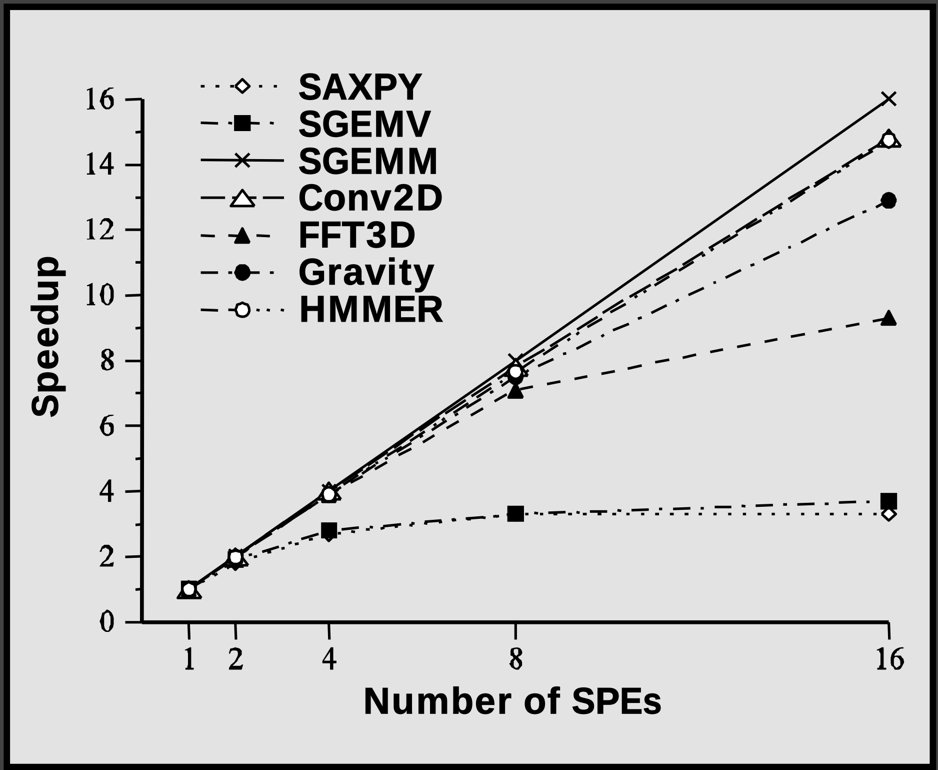
Cell utilization

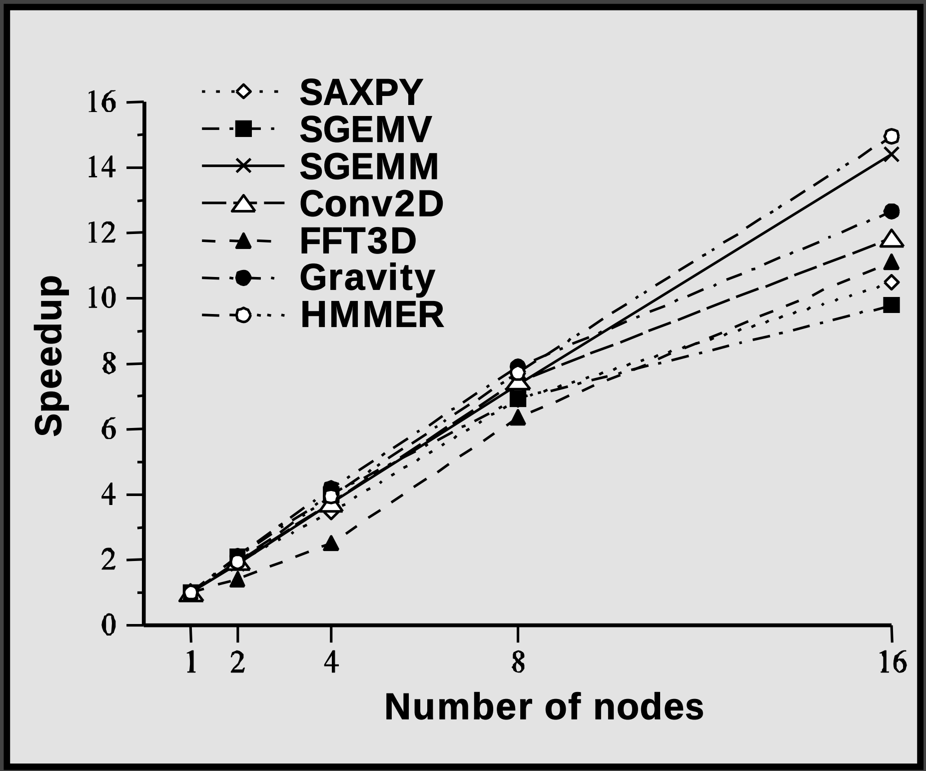


Performance scaling

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

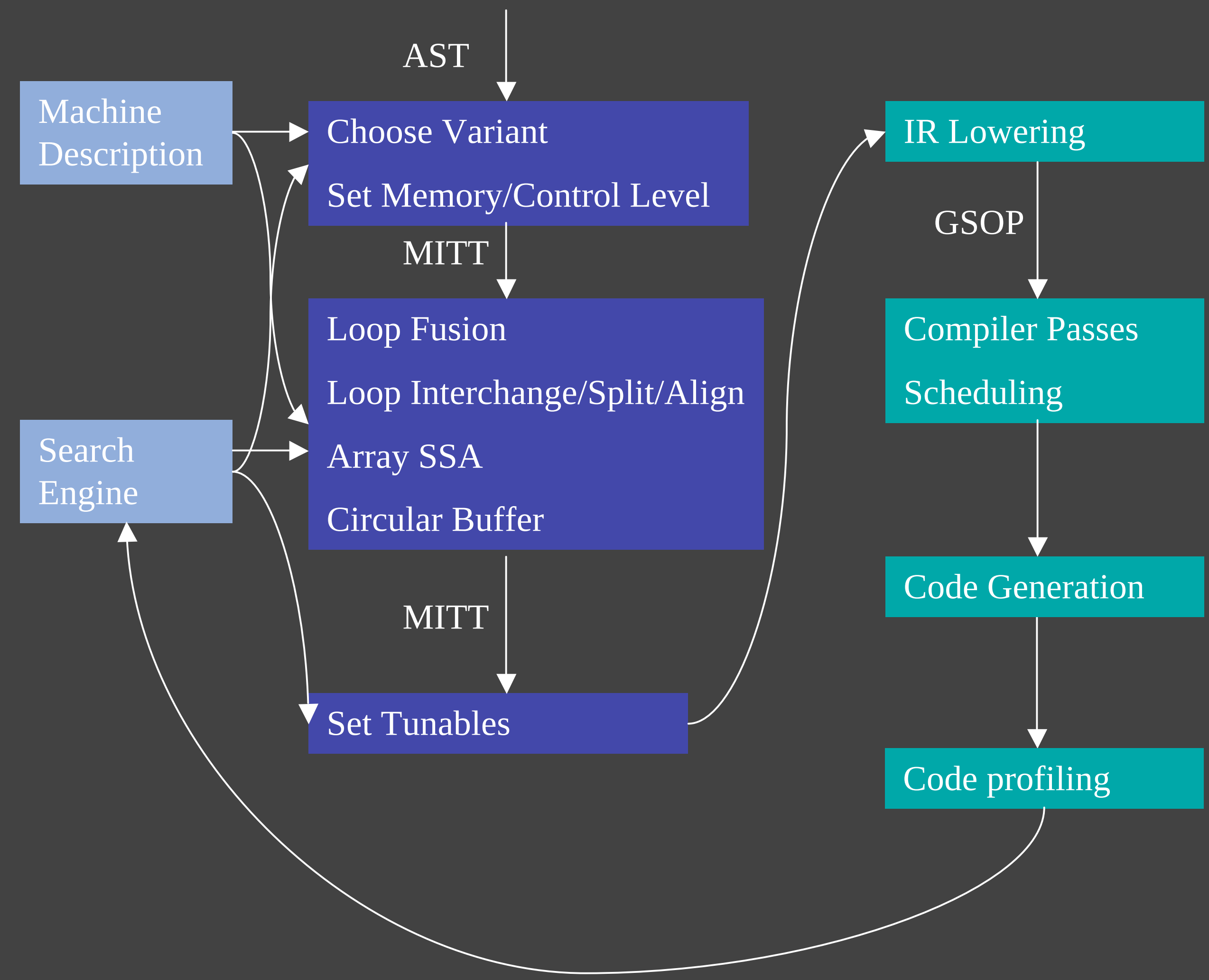
Auto-tuning

- Sequoia provides a framework for tuning
  - Algorithm definition separate from mapping!
- Programmer defines search space
  - Tunables
  - Variants
- Machine model defined by system architect
- Map algorithm to machine via searching the tunable and variant space

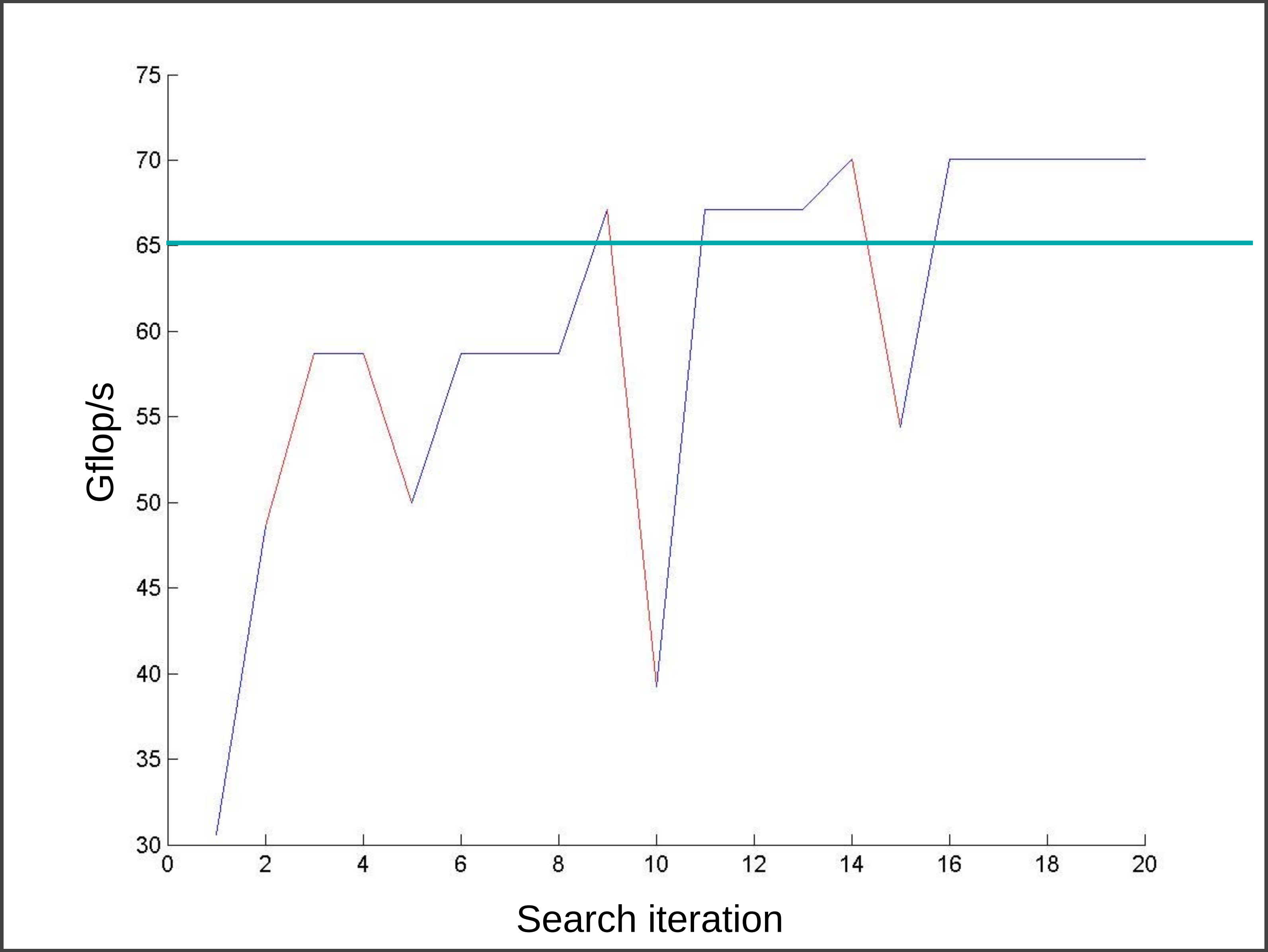
Current status

- Profiling driven optimization
  - Set tunables, run, retune, rerun, …
- Greedy search for kernel fusion
- Greedy search for tunables
  - Bottom up - Set L0, tune, move to L1 tuning
  - Obviously non-optimal
- Brute force variant search
  - Try all variants at a level if mappable to that level

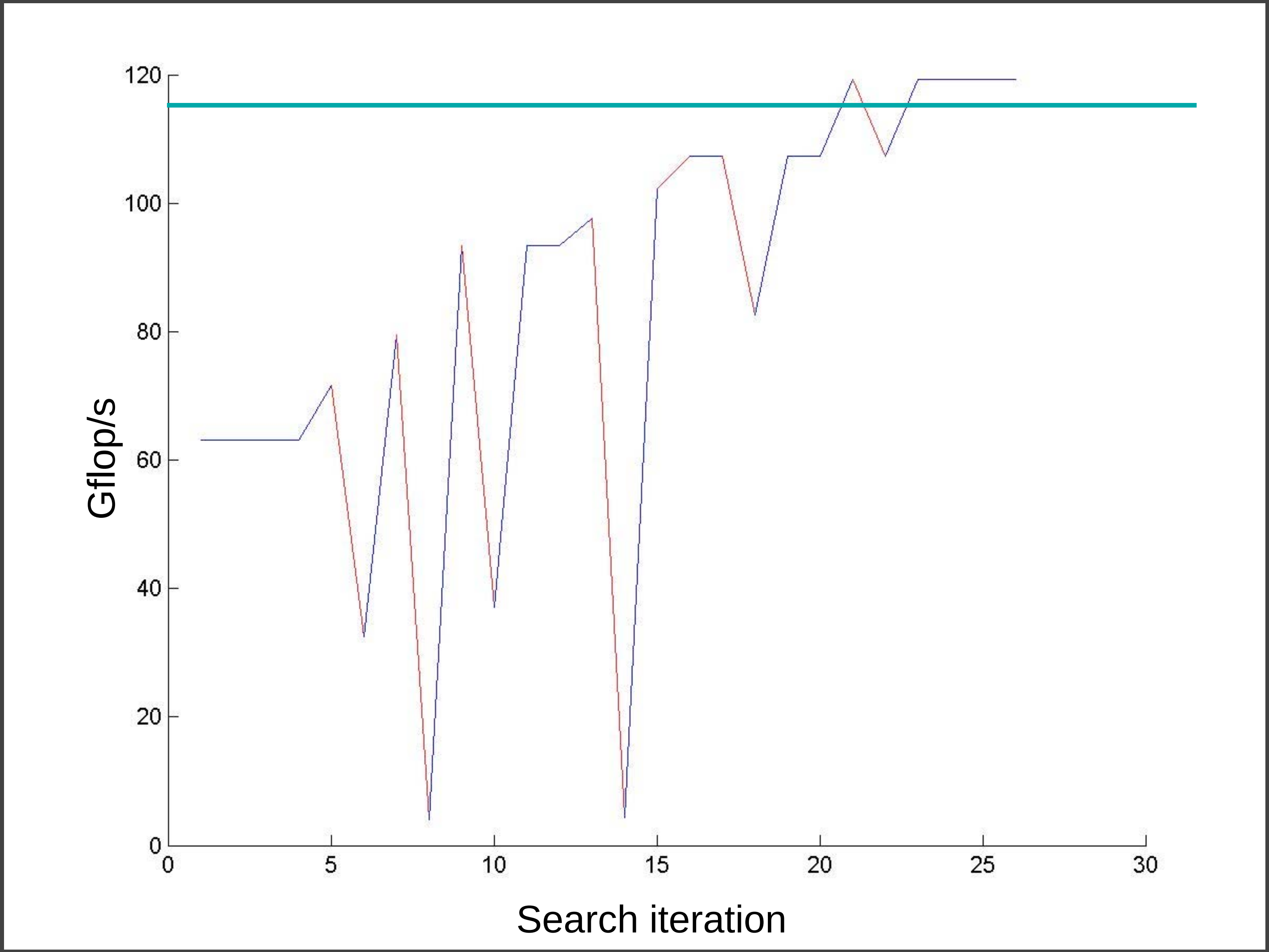
Sequoia tuning overview



Conv2D tuning on Cell (5x5 window)



SGEMM tuning Cell



Future work

- Runtime system release
  - SMP, Cluster, Disk, Mecury CAB
  - Cray XT3/4
  - Roadrunner
  - BG/L, BG/P (?)
- More complex algorithms
  - Sweep3D
  - SPaSM/GROMACS
  - SUmb/FEM
  - MG
- Auto-tuning
  - Researching variant space search
  - Better search techniques
  - Kernel fusion tricky

Questions?