Autotuning for Petascale: An Architect's Perspective

Mattan Erez

The University of Texas at Austin

CScADS Autotuning Workshop

July 8, 2008

Snowbird, Utah

Outline

- What should we be tuning for?
  - Performance isn't everything
  - Tune anything that's important
- How should the programmer/user interact with the auto-tuner and software system?
  - Libraries aren't enough
    - Some programmers are always trying to be clever
  - Language should express what's important – including tuning
    - Too many choices and too many platforms
- Recent architecture research trend: fairness
  - Heterogeneous Multicore

Tune for Utility/Cost – not Performance

Building systems is all about the bottom line

Machine Cost Factors

- Acquisition ~$50M
  - Peak Processing
  - Peak Bandwidth
  - Peak Memory/Storage
  - **Reliability**
  - Usability
  - Facilities (power)
- Operation ~$5M/yr
  - **Power**
  - Maintenance/Administration
- **Optimize total work for total cost**
  - **Maximizing task performance doesn't always do that**

Fault Tolerance == Opportunity Cost

- Reliability is an increasing concern
  - Not just memory any more
  - Logic increasingly susceptible to soft errors
  - Smaller dimension more sensitive to radiation
  - Process variation is on the rise
- Reliability requires redundancy
- "Non-stop" hardware is too costly
  - We are using unreliable systems!
- What reliability options do we apply and when?
  - Algorithmic based fault tolerance
  - Assertions
  - Computation duplication
  - Hardware features occasionally
  - Checkpoint granularity and footprint

Power is the Dominant Architectural Problem

- Bad news: power scaling is slowing down
  - Can't scale Vt much in order to control leakage
    - New technology helps
  - → can't scale Vdd as much
  - → power doesn't go down as it used to
- Energy/device decreases slower than devices/chip
- Power goes up if performance scaling continues
  - For same processor architecture
- Roadrunner: 1PFLOP/2MW, BG/L 0.5PFLOP/2MW
  - How much for many PFLOPS?

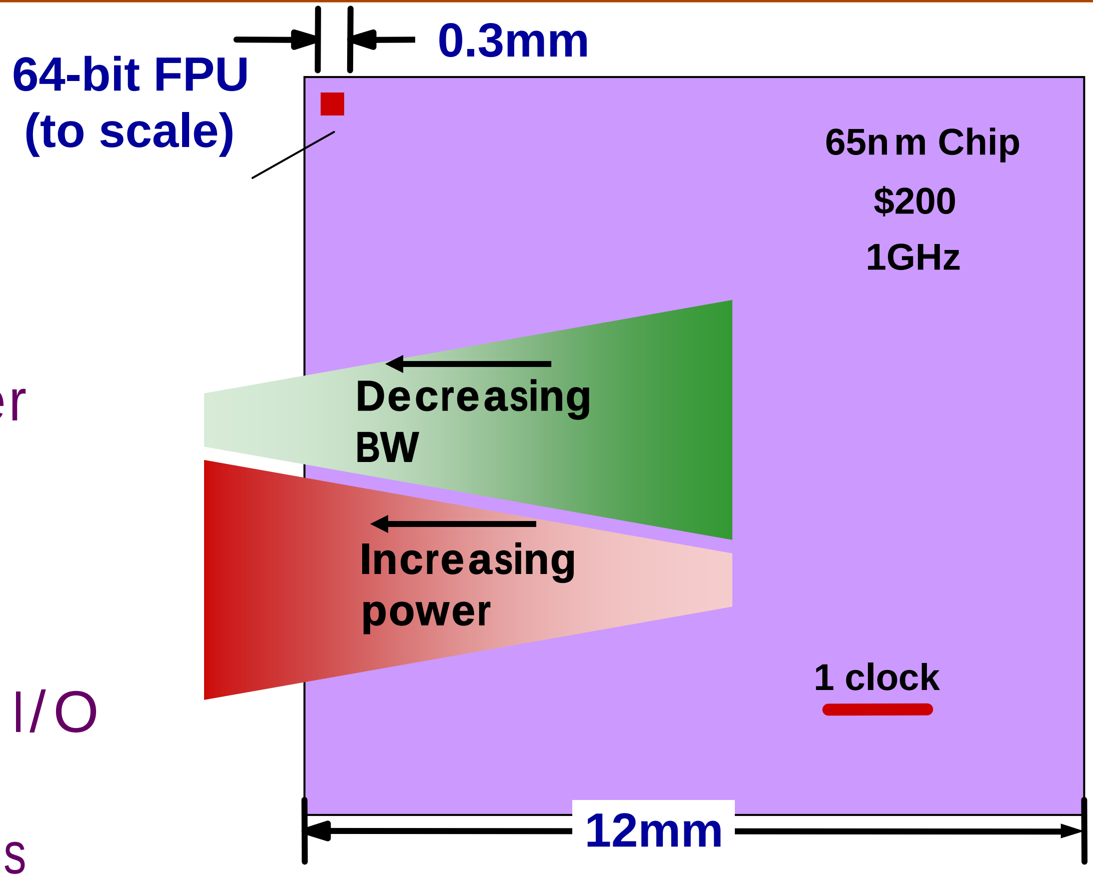
- More bad news: energy prices going up ☺

How Can We Reduce Power?

- Compute less
  - Use better algorithms
- Waste less
  - Don’t build/use unnecessary hardware
  - No unnecessary operations
  - **No unnecessary data movement**
  - Tuning can help – minimize power per acceptable performance goal
- Specialize more
  - Specialized circuits are more efficient
  - Tuning can help decide when

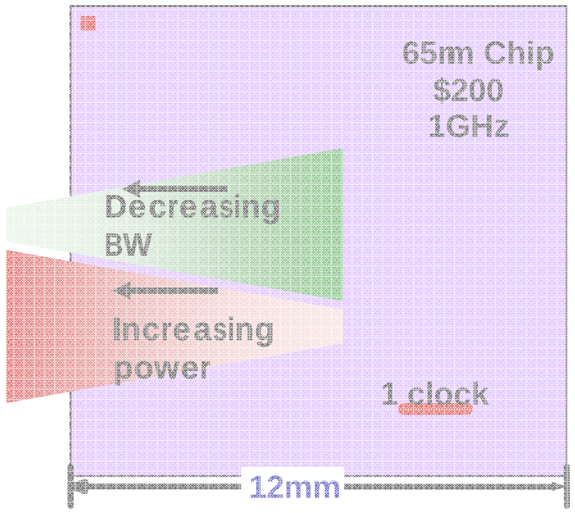
Wasting Less – Effective Performance in VLSI

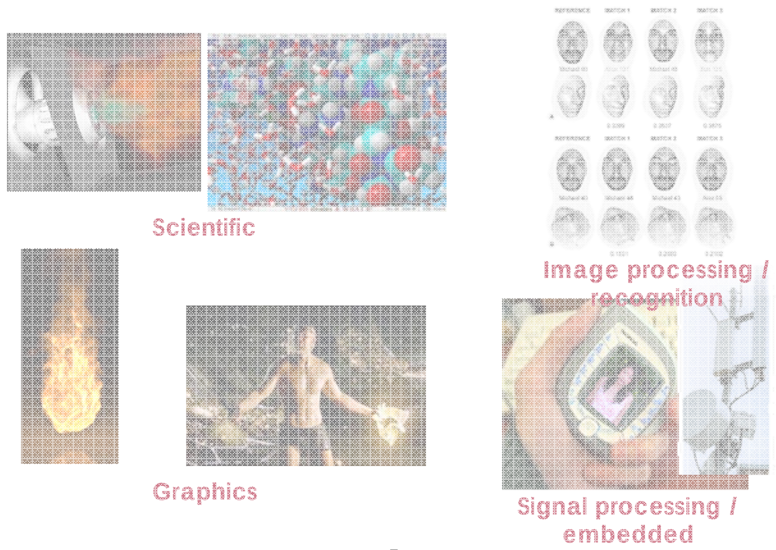
- Parallelism
  - 10s of FPUs per chip
  - Efficient control
- Locality
  - Locality lowers power
  - Reuse reduces global BW
- Throughput Design
  - Throughput oriented I/O
  - Tolerate Increasing on-/off-chip latencies
- Minimum control overhead



Parallelism, locality, latency tolerance, bandwidth, and efficient control

The Streaming Concept: Match Software with VLSI Strengths



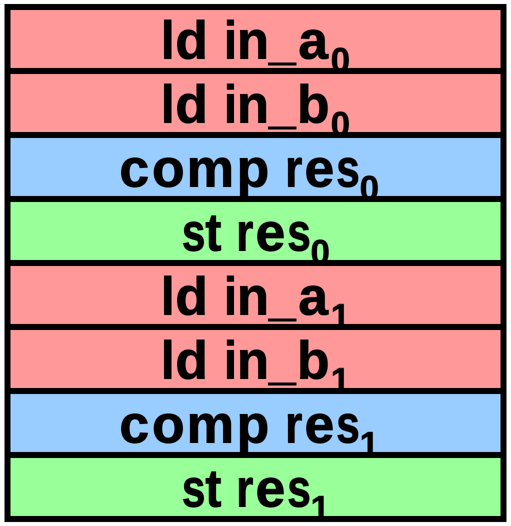


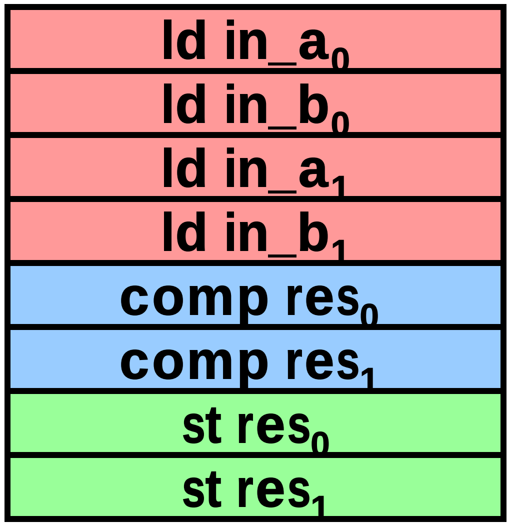
- Hardware matches VLSI strengths
  - Throughput-oriented design
  - Parallelism, locality, and partitioning
  - Hierarchical control
  - Minimalistic HW scheduling and allocation

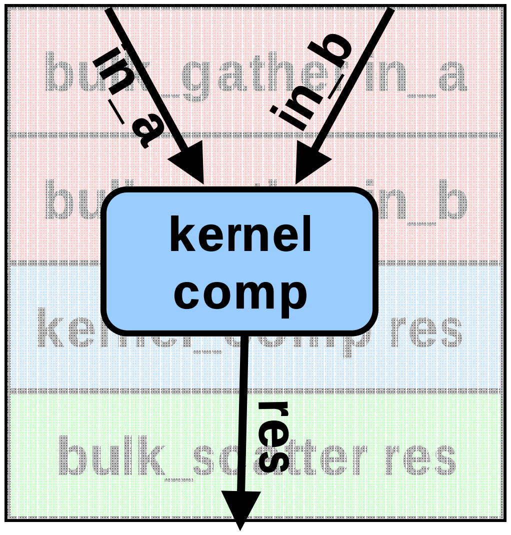
- Software **given** more explicit control
  - Explicit hierarchical scheduling and latency hiding
  - Explicit parallelism
  - Explicit locality management

Take Advantage of Software: Hierarchical Bulk Operations

- Data access determinable **well in advance** of data use
  - Latency hiding
  - Blocking
- Reformulate to ***gather – compute – scatter***
  - Block phases into ***bulk operations***





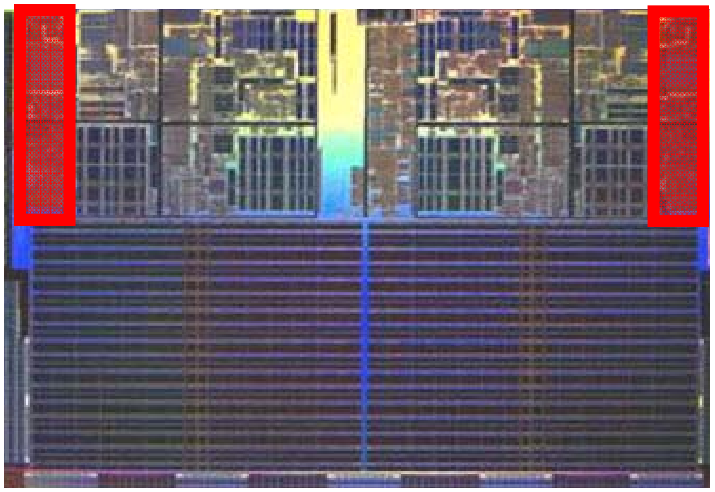


Bulk Operations Increase Performance

AMD dual-core Opteron

90nm | ~200 mm^2 | ~100 W

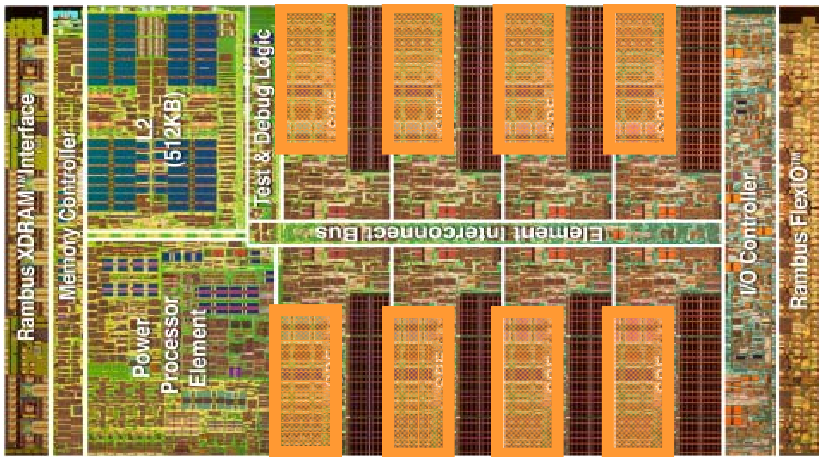
~20 GFLOPS

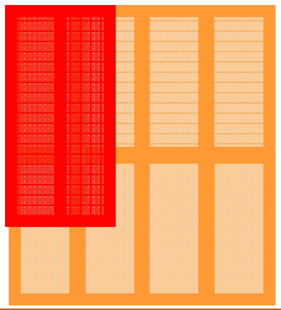


STI CELL processor

90nm | ~220 mm^2 | ~100 W

~200 GFLOPS

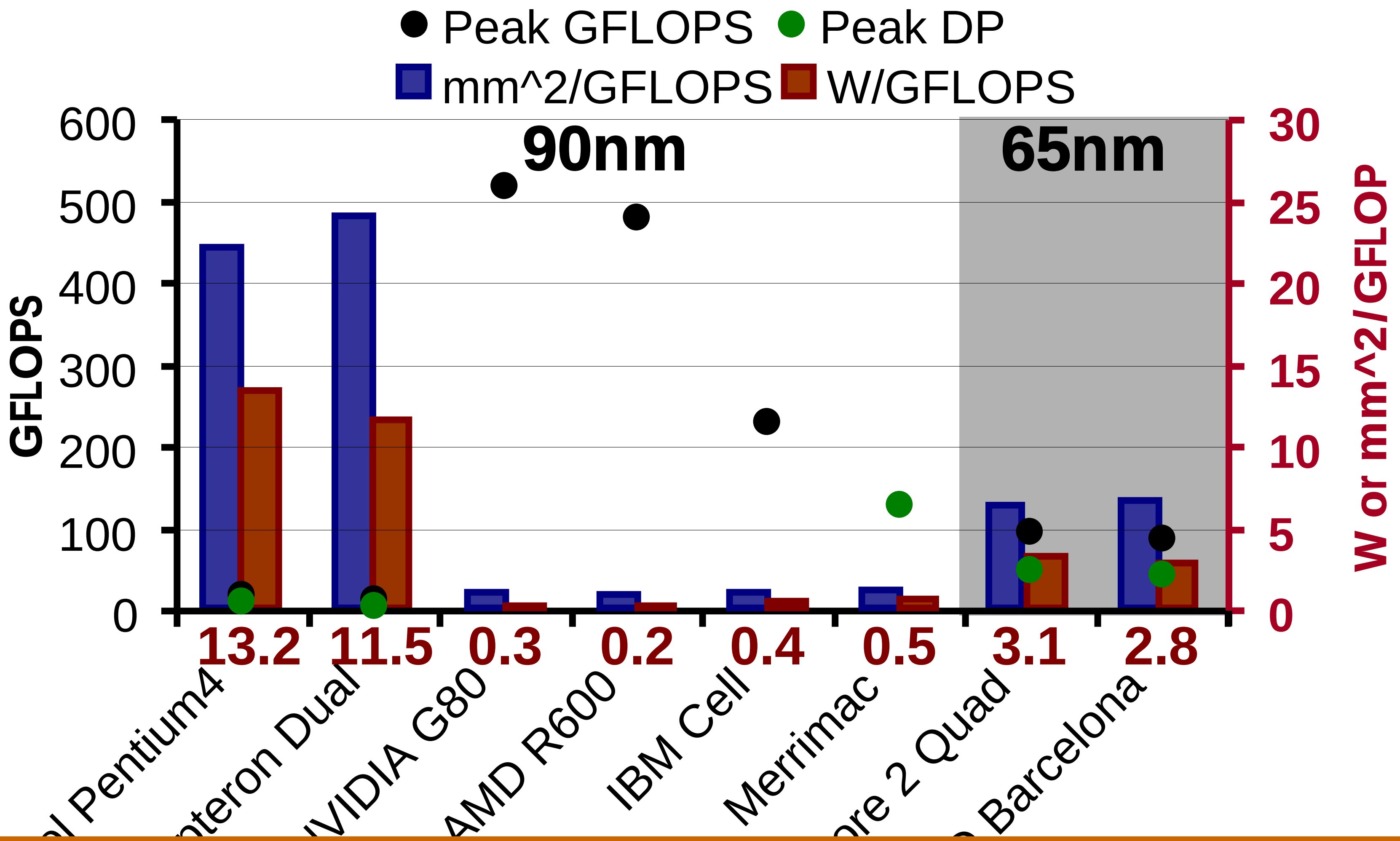




FPUs

Much more significant resources devoted to FPUs

Bulk Operations Achieve Efficiency and Performance



Even partial adoption of bulk operations has huge impact on performance and efficiency

Major Success but Not Enough

- Cell is ~1.5X BlueGene (based on Top500)
  - Merrimac estimates were ~6X better (in same tech node)
  - Still not enough for true Petascale
- Use better algorithms – often irregular
- Truly dynamic and irregular algorithms are challenging for bulk/streaming architectures
  - Beg for some degree of threading and caching
  - Hybrid bulk/thread architectures and models
- More work on memory systems
  - Granularity is a problem
- On-chip interconnection networks – no clear winner

Locality, Parallelism, and Hierarchy throughout the system

Tuning for Power

- Need to co-search for power and performance
  - Optimize cost, not performance
  - Opportunity cost too (fault tolerance)
- Maximize locality / minimize data movement
  - Power impacted significantly by interconnect and memory
- Try to specialize
  - Utilize control hierarchy
  - Utilize specialized hardware
- Minimize waste
  - Strong interactions with load balancing
  - Processor/memory dynamic power management is key

Languages Need to Abstractly Expose Important Factors and Tuning

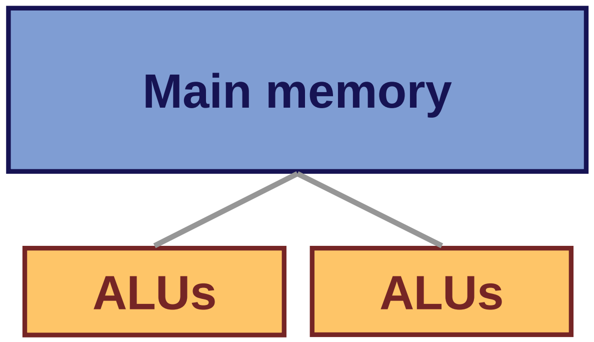
- How should the programmer/user interact with the auto-tuner and software system?
  - Libraries aren’t enough
    - Some programmers are always trying to be clever
  - Language should express what’s important – including tuning
    - Too many choices and too many platforms

Sequoia: Abstract Streaming/Bulk Programming

- 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
- Facilitate tuning
  - Decouple algorithm and decomposition from setting parameters
  - Sequoia language only expresses strategy

Hierarchical memory

- Abstract machines as trees of memories

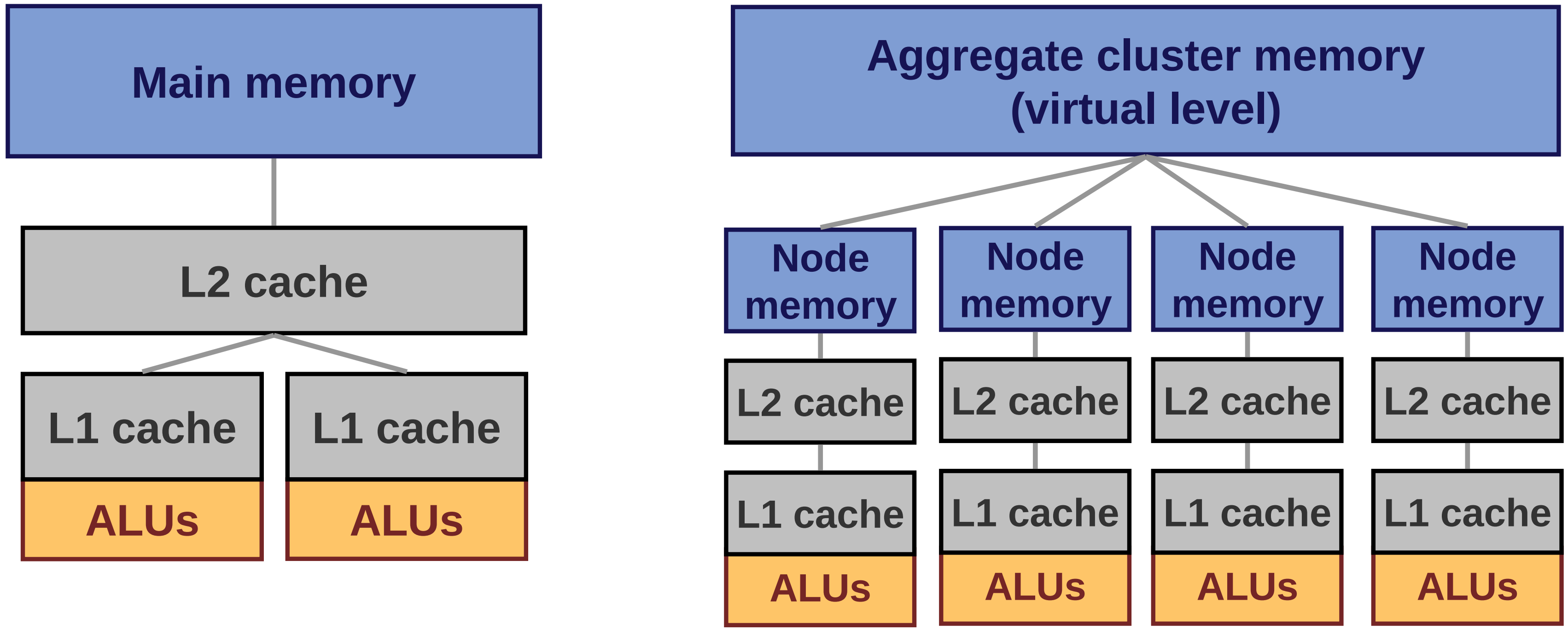


Similar to:

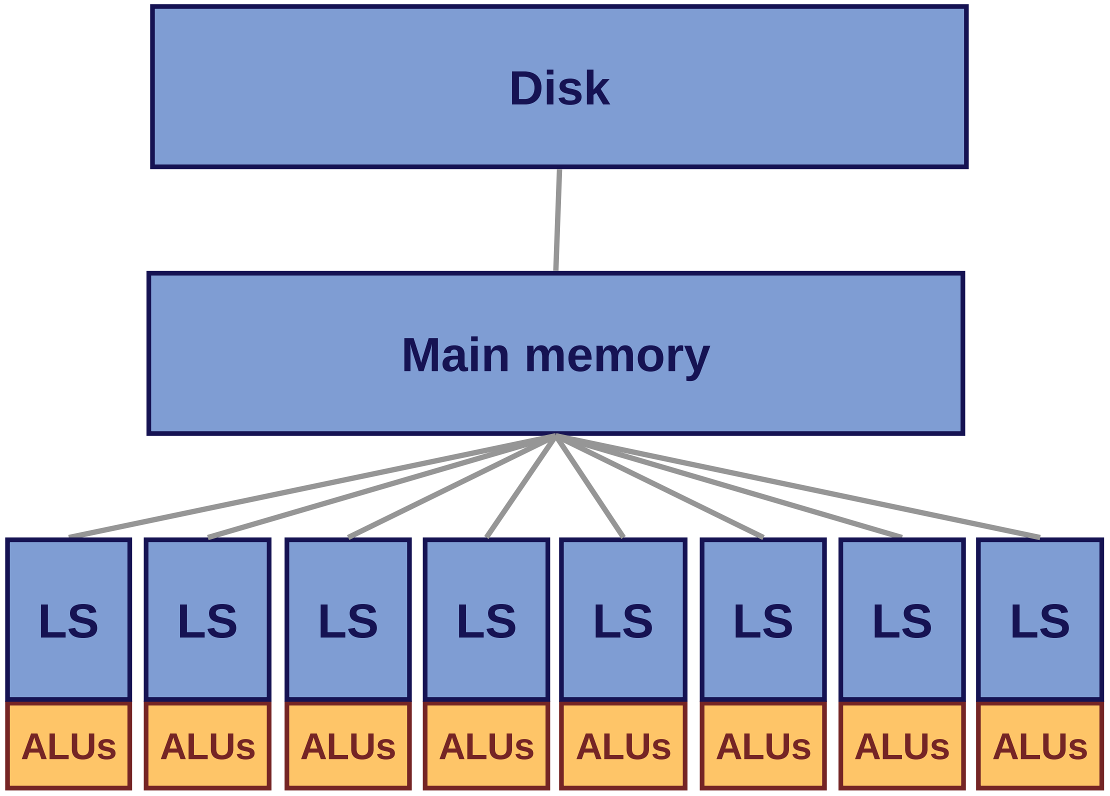
Parallel Memory Hierarchy Model (Alpern et al.)

Hierarchical memory

- Abstract machines as trees of memories



Hierarchical memory



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**

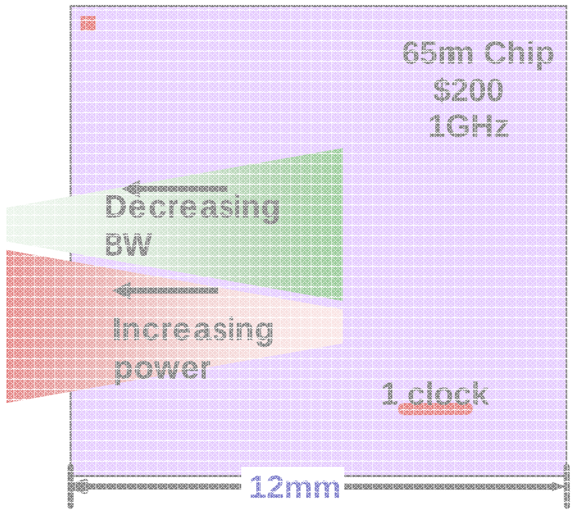
Sequoia tasks

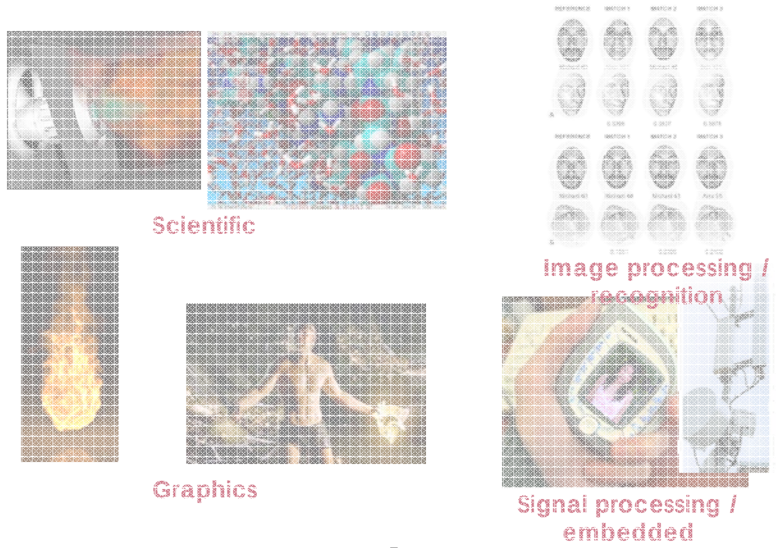
- Single abstraction for
  - Isolation / parallelism
  - Explicit communication / working sets
  - Expressing locality

- Tasks operate on arrays, not array elements

- Tasks nest: they call subtasks

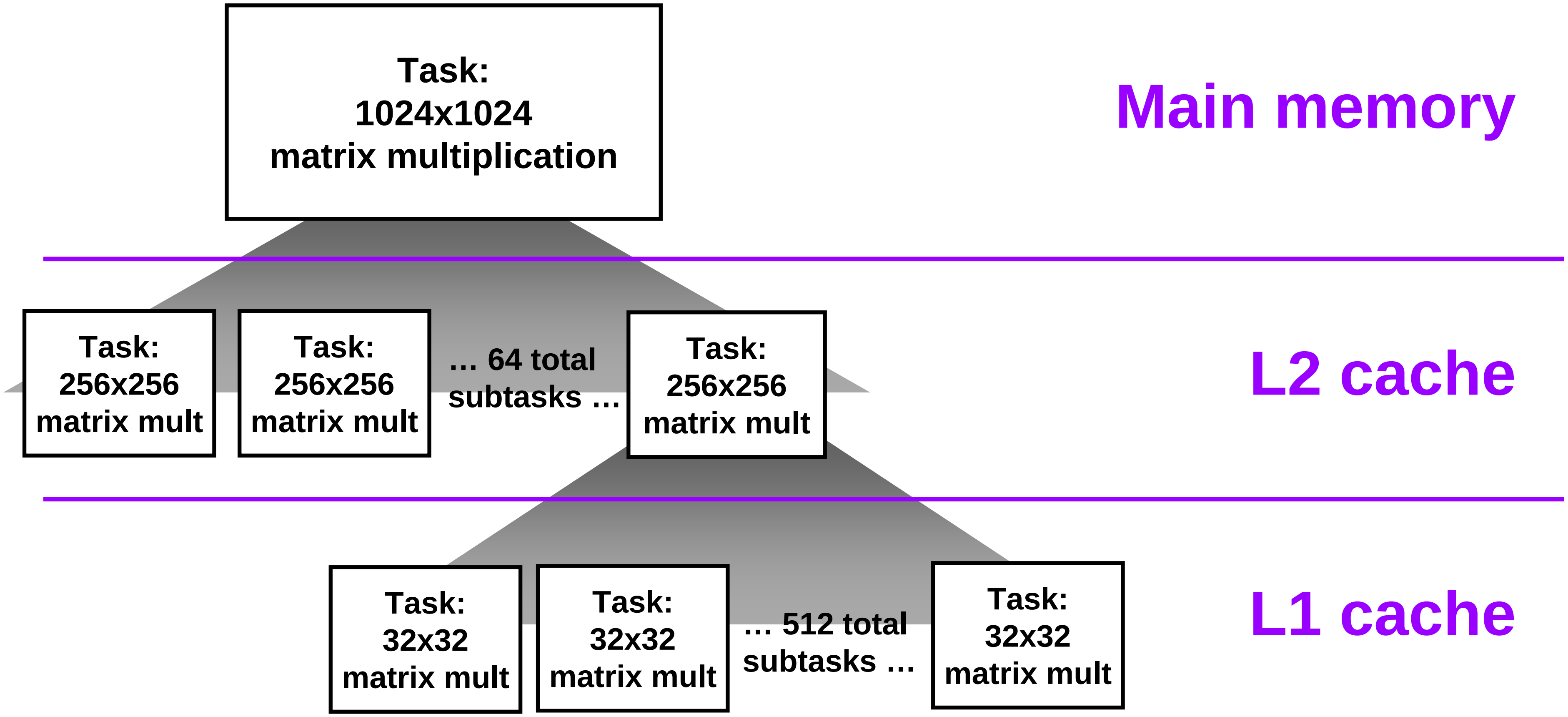
The Streaming Concept: Match Software with VLSI Strengths





- Hardware matches VLSI strengths
  - Throughput-oriented design
  - Parallelism, locality, and partitioning
  - Hierarchical control
  - Minimalistic HW scheduling and allocation
- Software **given** more explicit control
  - Explicit hierarchical scheduling and latency hiding
  - Explicit parallelism
  - Explicit locality management

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;







}
```

```
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];
}
```

- Tasks are written in parameterized form for portability
- Different “variants” of the same task can be defined

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]);

     }
  }
}
```

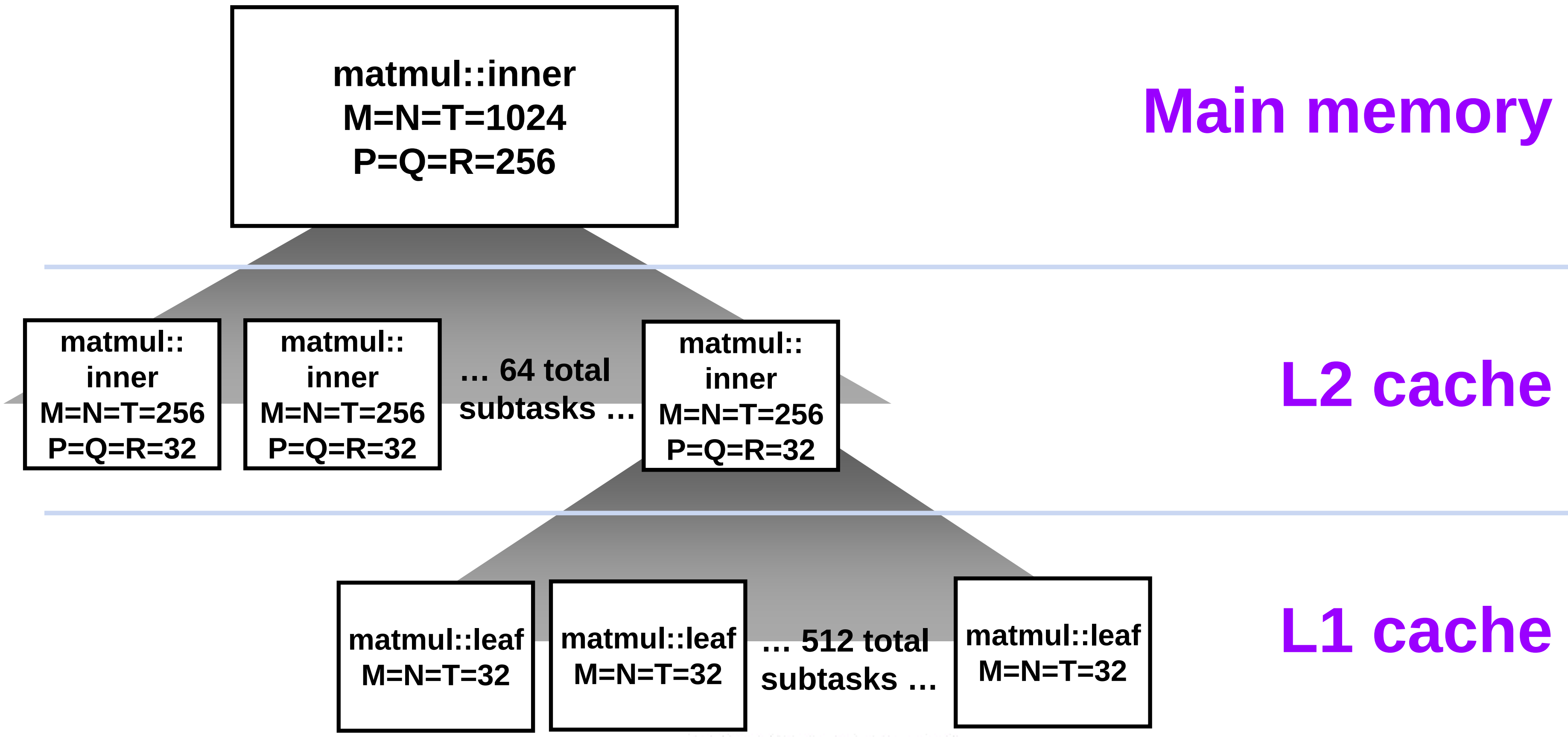
- Working set resident within single level of hierarchy

- 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

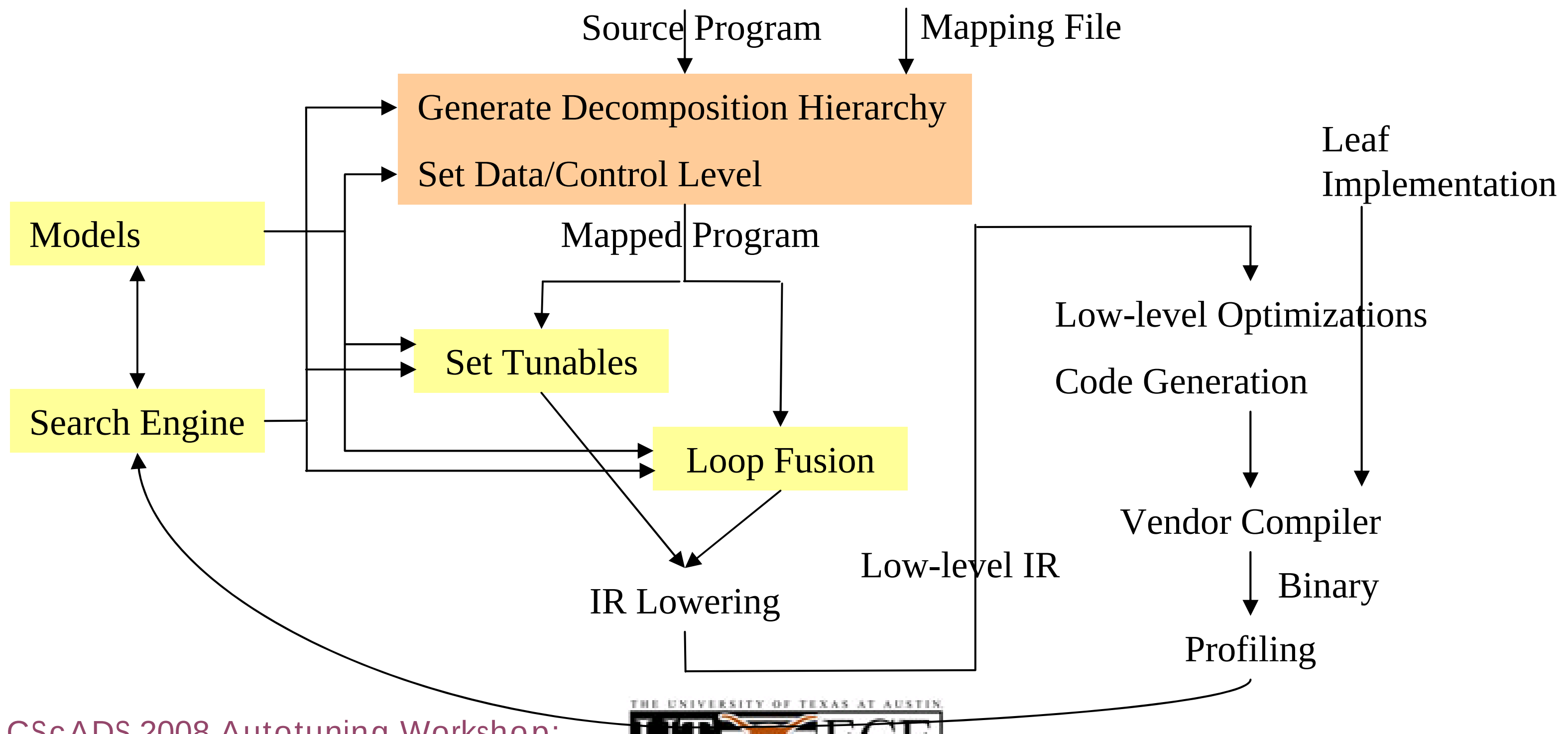


Specialization with Autotuning

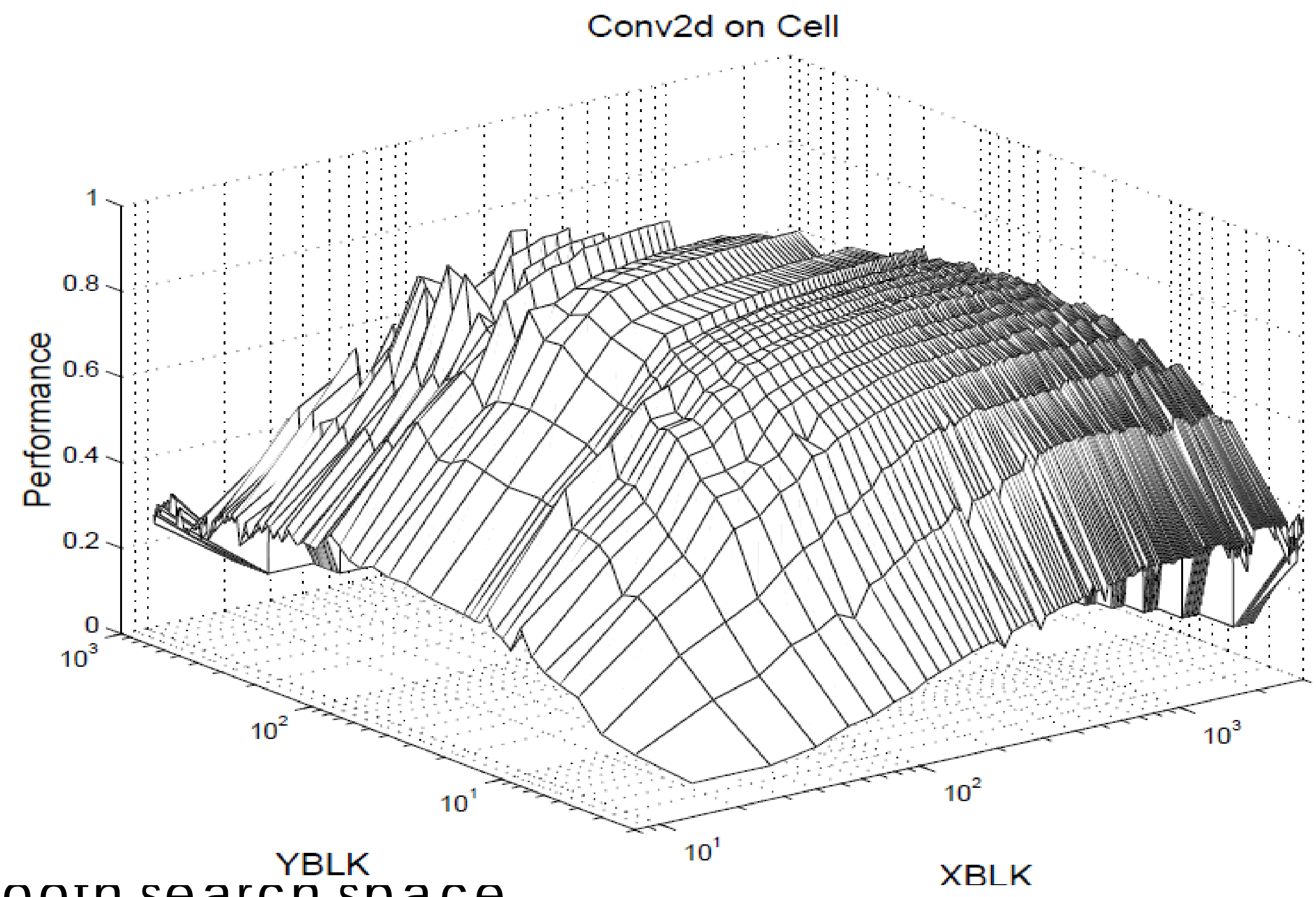
- Work by Manman Ren (Stanford), PACT 2008
- Use Sequoia to identify what needs tuning
  - Explicit tunables and parameters in the language
- Tuning framework for SW-managed hierarchies
- Automatic profile guided search across tunables
  - Aggressive pruning
  - Illegal parameters (don’t fit in memory level)
  - Tunable groups
  - Programmer input on ranges
  - Coarse $\rightarrow$ fine search
- Loop fusion across multiple loop levels
  - Measure profitability from tunable search
  - Adjust for “tunable mismatch”
  - Realign reuse to reduce communication

Overview: mapping the program

- Mapped versions are generated
  - Matching the decomposition hierarchy with the machine hierarchy
  - Choosing a variant for each call site
  - Set level of data objects and control statements



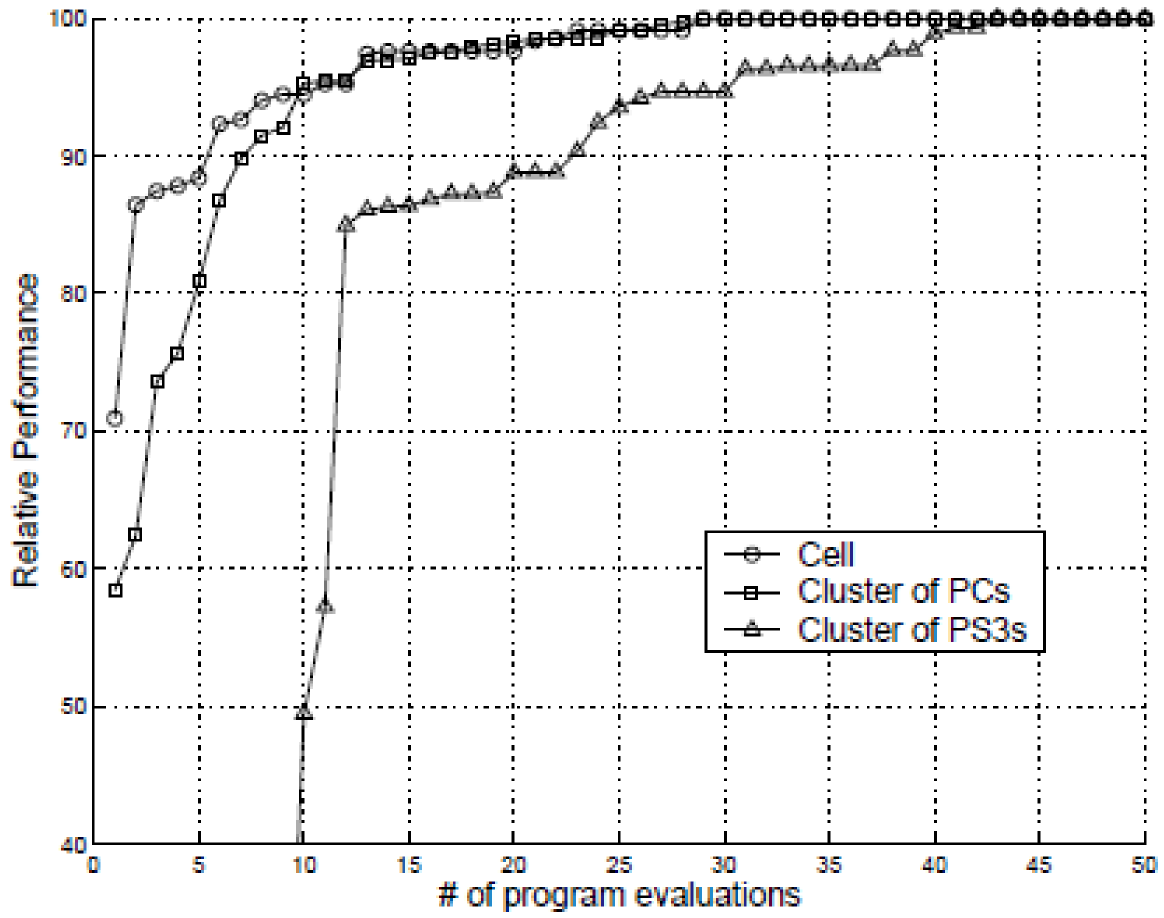
Explicit SW Management Simplifies Tuning



- Smooth search space
- Performance models can also work
  - For Cell, not cluster

Guided Search Converges Quickly

- Smoothness leads to quick convergence



Autotuning Out Performs Programmer

		CONV2D	SGEMM	FFT3D	SUmb
Cell	**auto** hand	**99.6** 85	**137** 119	**57** 54	**12.1**
Cluster of PCs	**auto** hand	**26.7** 24	**92.4** 90	**5.5** 5.5	**2.2**
Cluster of PS3s	**auto** hand	**20.7** 19	**33.4** 30	**0.57** 0.36	**0.63**

Architecture Trend: Fairness in Multicore/Multi-threaded Processors

Hardware balances shared resources

Maintain Overall Performance through Fair Partitioning of Shared Resources

- Motivating applications: multiprogramming
- Shared cache
  - Allocate partitions of ways in a set-associative cache to threads
  - Prevent low-locality thread from evicting useful data
- Shared memory bandwidth
  - Schedule memory operations from different threads fairly
- Definition of fairness?
  - All threads suffer performance degradation relative to running in isolation

Conclusions

- Autotuning should match architecture optimizations – **maximum utility/cost**
  - Maximize locality / minimize communication
  - Take advantage of control hierarchy
  - Specialized hardware units
  - Reliability is another opportunity
- Languages should expose what's important (in an abstract portable way)
  - Expose tuning – it's an essential part of the software system
  - Sequoia is one early attempt