Making Good Enough...Better:

Addressing the Multiple Objectives of High-Performance Parallel Software with a Mixed Global-Local Worldview

John A. Gunnels
Research Staff Member/Manager
IBM T.J. Watson Research Center
Business Analytics & Mathematical Sciences

Outline

- Level of Ambition
- Tabloid Programming
- Performance Counters & Power Measurement
- Case Studies
  - Heavy- vs. Lightweight Synchronization: DGEMM
  - Trade-offs in Synchronization: HPL Benchmark
  - Shifting Operation Type: Stencil Computations
  - Lanczos Iteration Methodology: s-Step and Pipeline
  - Interacting Kernels: A Simple Tuning Framework
- Conclusions

Outline

- Level of Ambition
- Tabloid Programming
- Performance Counters & Power Measurement
- Case Studies
  - Heavy- vs. Lightweight Synchronization: DGEMM
  - Trade-offs in Synchronization: HPL Benchmark
  - Shifting Operation Type: Stencil Computations
  - Lanczos Iteration Methodology: s-Step and Pipeline
  - Interacting Kernels: A Simple Tuning Framework
- Conclusions

Level of Ambition

- Separation of concerns
  - “First, you get a million dollars …”
- Run-time agnostic
  - Task-based
    - GCD, PFunc, PLASMA, StarSs/OMPSs, Supermatrix, etc.
  - Traditional
    - MPI, OpenMP, Pthreads, SHMEM, SPI, etc …
  - PGAS
    - CAF, Chapel, Fortress, Titanium, UPC, X10 …
- Examples
  - Simple
  - Results can be applied somewhat more broadly

Outline

- Level of Ambition
- Tabloid Programming
- Performance Counters & Power Measurement
- Case Studies
  - Heavy- vs. Lightweight Synchronization: DGEMM
  - Trade-offs in Synchronization: HPL Benchmark
  - Shifting Operation Type: Stencil Computations
  - Lanczos Iteration Methodology: s-Step and Pipeline
  - Interacting Kernels: A Simple Tuning Framework
- Conclusions

Tabloid Programming

- Determine what is going on:
  - In my neighborhood & in my world
  - Where is the cut-off?
- Summarizing instrumentation data
  - Core(s)/Thread(s) devoted to it?
  - Descriptive, Predictive, and Prescriptive Analytics
- What would I like to do with the information
  - Annotate tasks/alter function pointers/re-time
  - Drive towards a profile (later)
  - Let others know my condition (Social Media Prog.?)
    - E.g. “doing error correction”

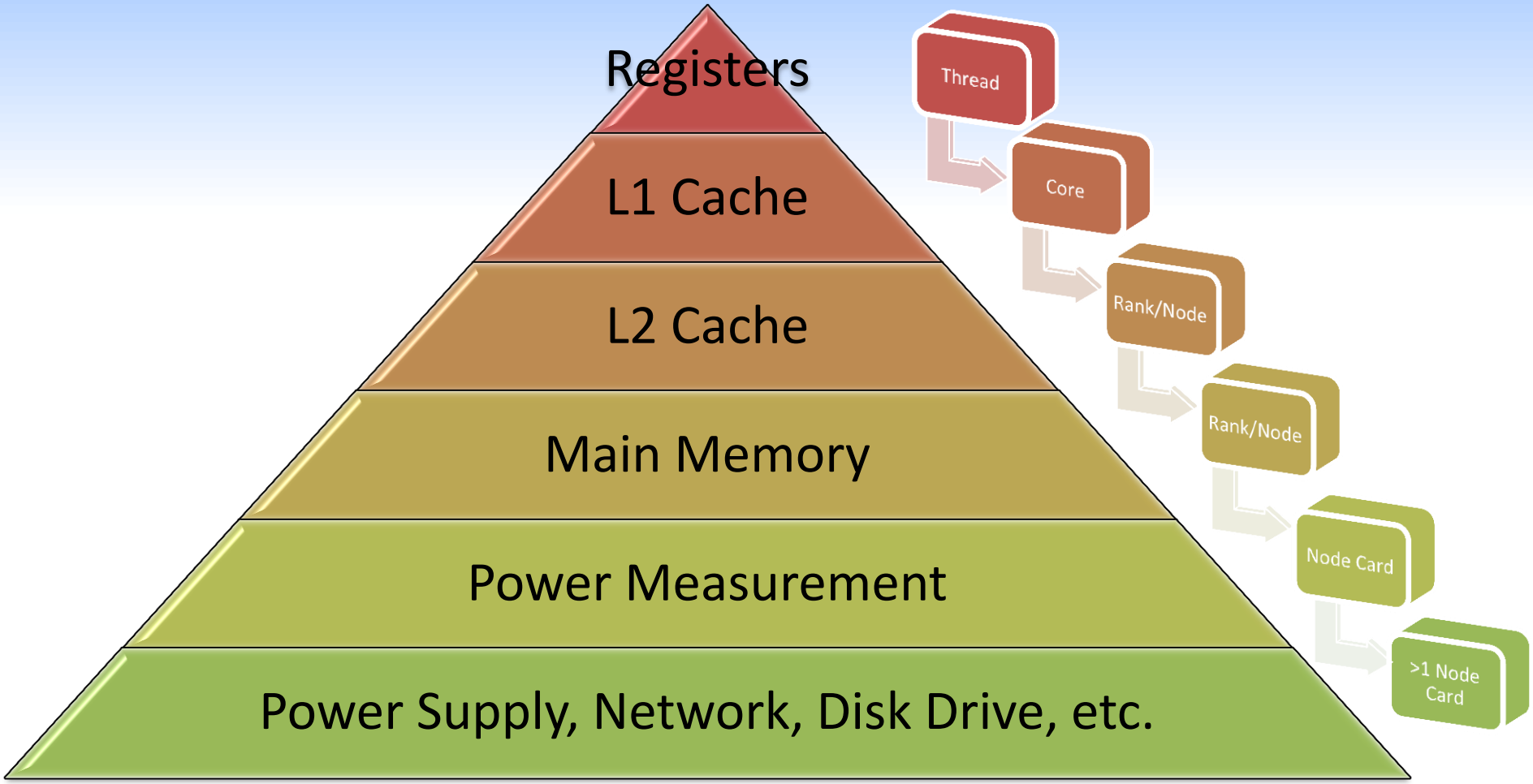
Outline

- Level of Ambition
- Tabloid Programming
- Performance Counters & Power Measurement
- Case Studies
  - Heavy- vs. Lightweight Synchronization: DGEMM
  - Trade-offs in Synchronization: HPL Benchmark
  - Shifting Operation Type: Stencil Computations
  - Lanczos Iteration Methodology: s-Step and Pipeline
  - Interacting Kernels: A Simple Tuning Framework
- Conclusions

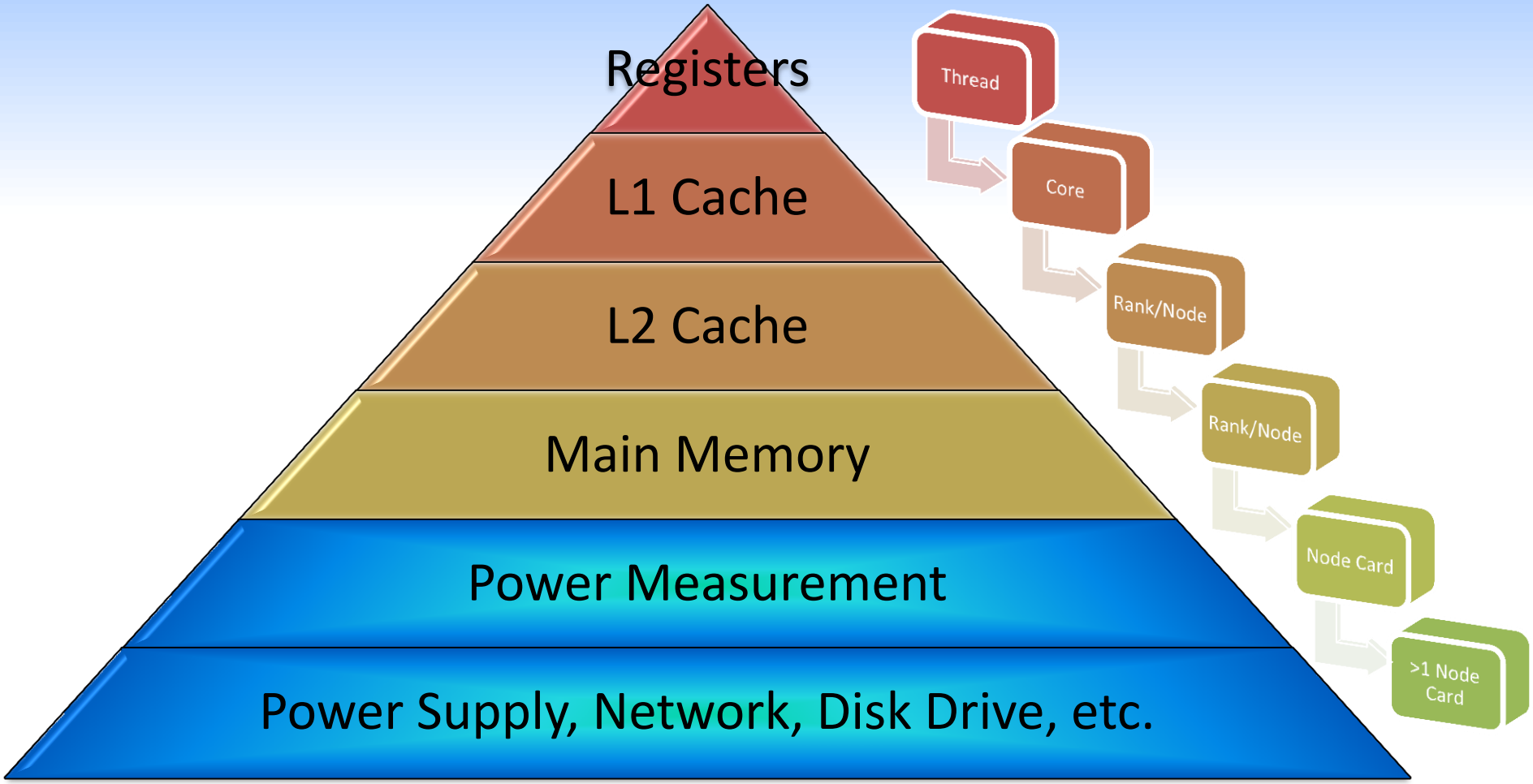
Performance Counters & Power Measurement

- Performance counters
  - Level of granularity (time, floorspace, etc.)
  - Post mortem analysis vs. in-flight steering
- Why power measurement
  - Synthesize info, can be fine-grained (Goal: Perf.)
  - Exascale (Goal: … well … power reduction)
    - To save power/minimize heat in aggregate or instantaneous
- Why both
  - Can disambiguate cases otherwise identical
  - Power is a shared resource (at a different level)

Shared Resource Hierarchy



Shared Resource Hierarchy



Outline

- Level of Ambition
- Tabloid Programming
- Performance Counters & Power Measurement
- Case Studies
  - Heavy- vs. Lightweight Synchronization: DGEMM
  - Trade-offs in Synchronization: HPL Benchmark
  - Shifting Operation Type: Stencil Computations
  - Lanczos Iteration Methodology: s-Step and Pipeline
  - Interacting Kernels: A Simple Tuning Framework
- Conclusions

Case Studies

- DGEMM
  - Synchronization strategies
  - Hierarchical, high-performance
- HPL Benchmark
  - Leveraging available data: a silver lining in synchronization
  - Utilizing additional hardware features
- Stencil Computations
  - Performance counters to guide bandwidth and instruction mix
  - Potential for linking/merging threads and “deep” synchronization
- Lanczos Iteration Methodology
  - s-Step and Pipeline: Reducing synchronization penalty, count, or both
- Auto-tuner
  - Utility of off-line system
  - A framework for the incorporation of new “operations” (atomics)

Outline

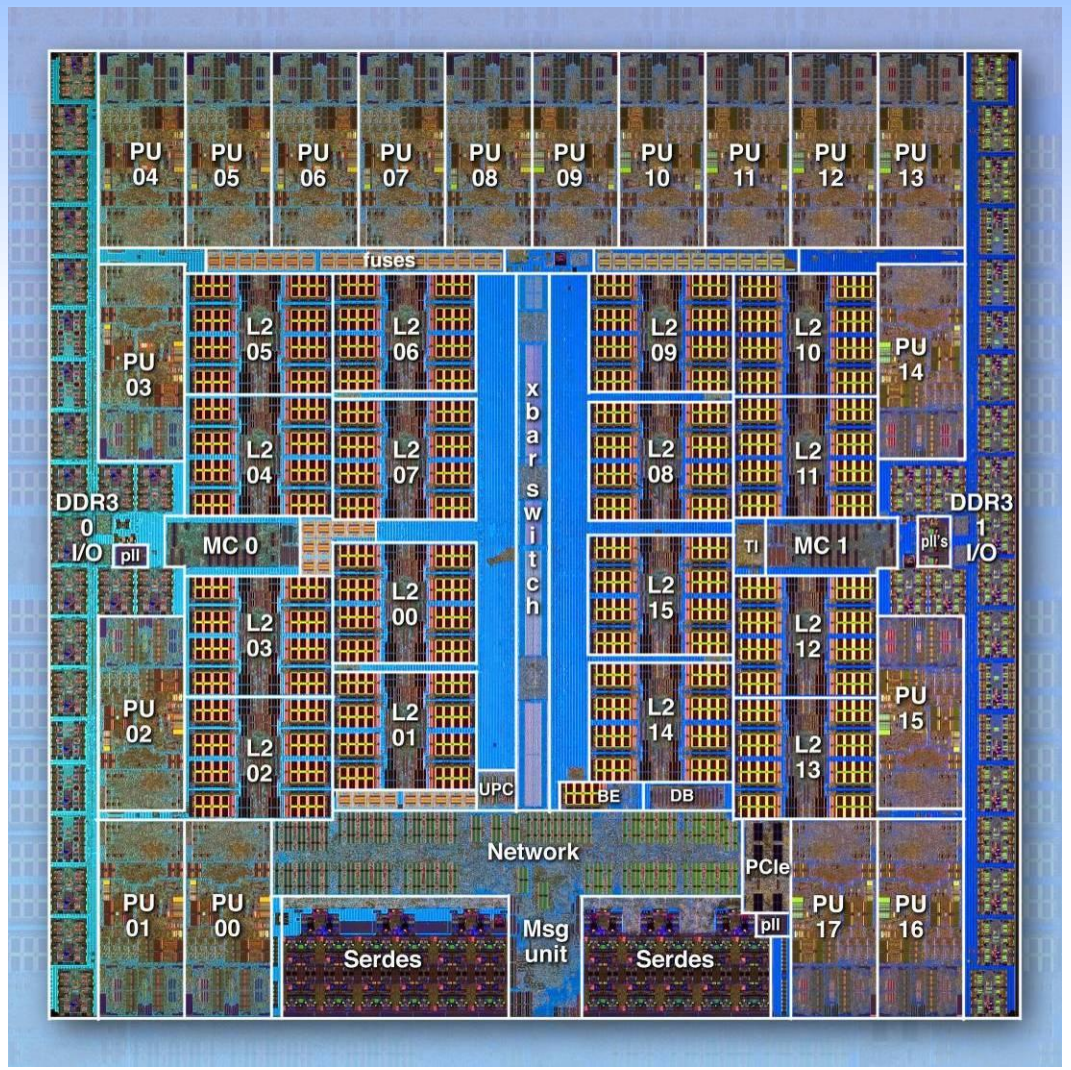
- Level of Ambition
- Tabloid Programming
- Performance Counters & Power Measurement
- Case Studies
  - Heavy- vs. Lightweight Synchronization: DGEMM
  - Trade-offs in Synchronization: HPL Benchmark
  - Shifting Operation Type: Stencil Computations
  - Lanczos Iteration Methodology: s-Step and Pipeline
  - Interacting Kernels: A Simple Tuning Framework
- Conclusions

Heavy- vs. Lightweight Synchronization: DGEMM

- Goal: Fewer **explicit** synchronization points
  - Explicit vs. implicit synchronization
  - Skew and anti synchronization
- Implicit synchronization through cooperation
  - Stitching threads and cores
    - At various levels of the cache hierarchy
  - Interleaving nodes lower on the pyramid
- What are the benefits
  - Realized
  - Potential

BlueGene/Q Compute chip

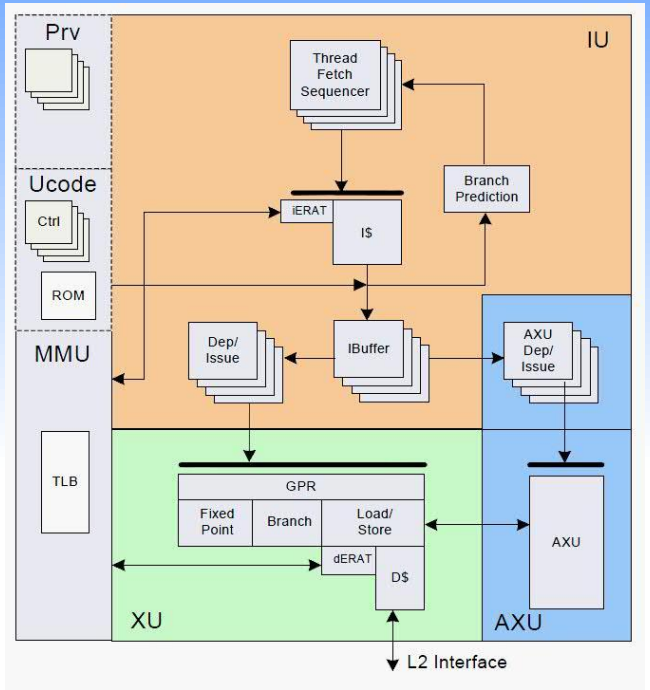
System-on-a-Chip design : integrates processors, memory and networking logic into a single chip



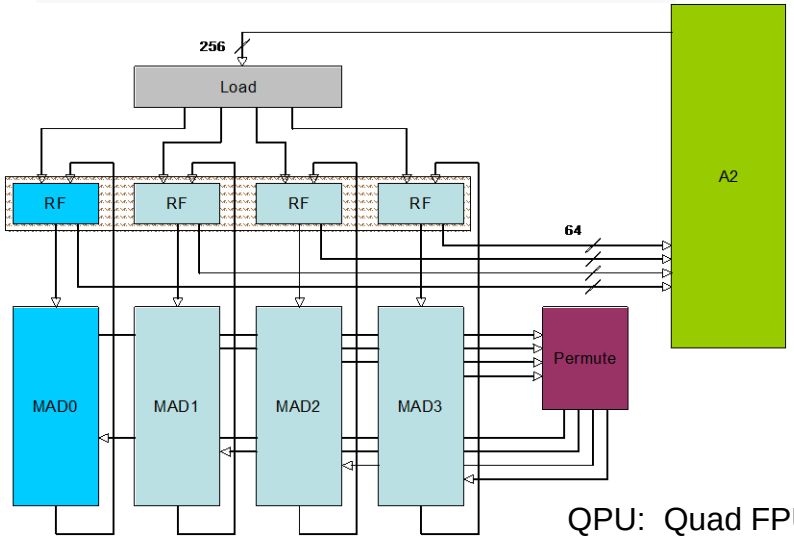
- **360 mm² Cu-45 technology (SOI)**
  - ~ 1.47 B transistors
- **16 user + 1 service processors**
  - plus 1 redundant processor
  - all processors are symmetric
  - **each 4-way multi-threaded**
  - 64 bits PowerISA™
  - 1.6 GHz
  - **L1 I/D cache = 16kB/16kB**
  - L1 prefetch engines
  - **each processor has Quad FPU (4-wide double precision, SIMD)**
  - peak performance 204.8 GFLOPS@55W
- **Central shared L2 cache: 32 MB**
  - eDRAM
  - multiversioned cache will support transactional memory, speculative execution.
  - supports atomic ops
- **Dual memory controller**
  - 16 GB external DDR3 memory
  - 1.33 Gb/s
  - 2 * 16 byte-wide interface (+ECC)
- **Chip-to-chip networking**
  - Router logic integrated into BQC chip.
- **External IO**
  - PCIe Gen2 interface

BG/Q Processor Unit

- **A2 processor core**
  - Mostly same design as in PowerEN™ chip
  - Implements 64-bit PowerISA™
  - Optimized for aggregate throughput:
    - **4-way simultaneously multi-threaded (SMT)**
    - **2-way concurrent issue 1 XU (br/int/l/s) + 1 FPU**
    - **in-order dispatch, execution, completion**
  - **L1 I/D cache = 16kB/16kB**
  - 32x4x64-bit GPR
  - Dynamic branch prediction
  - 1.6 GHz @ 0.8V

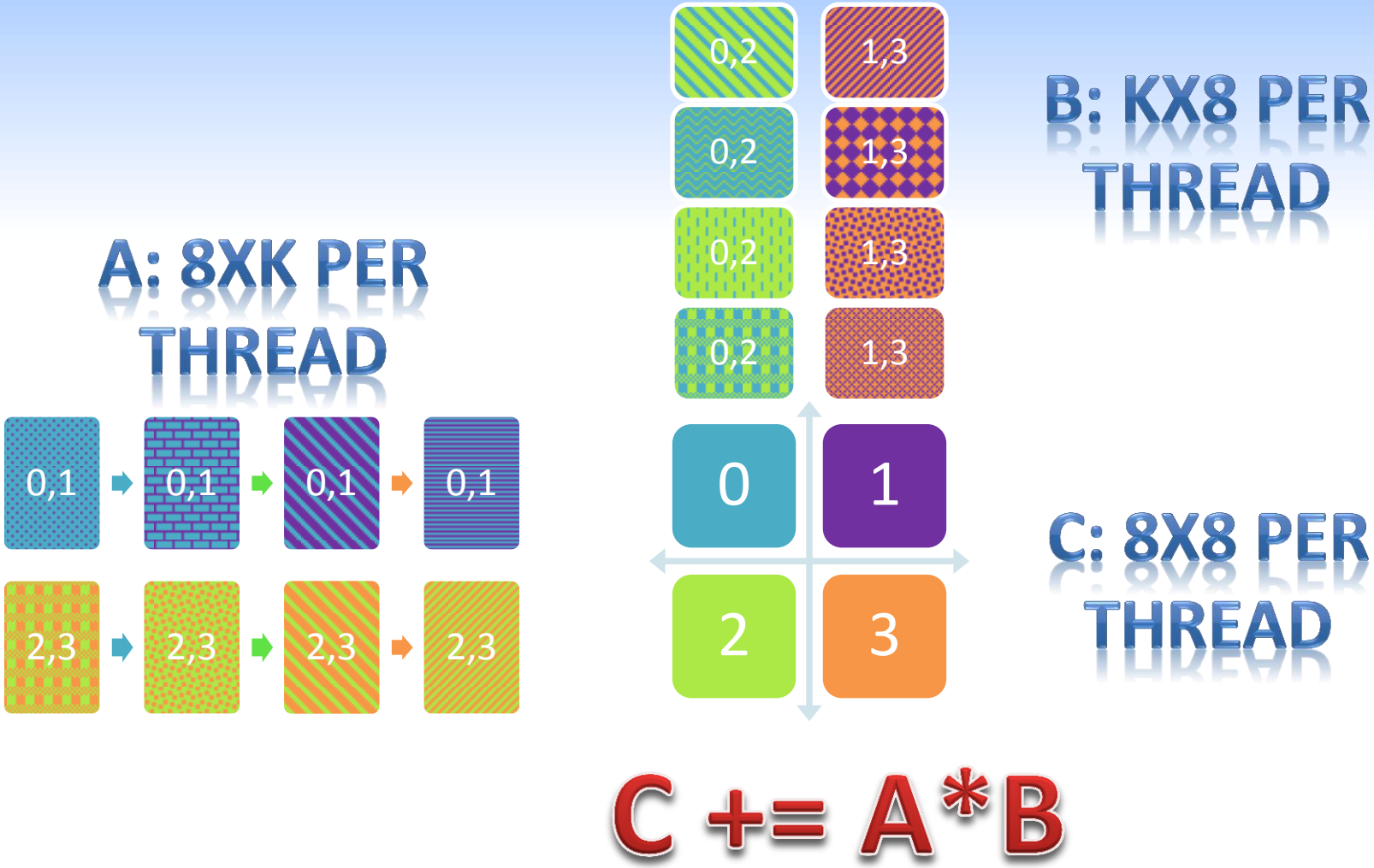


- **Quad FPU**
  - 4 double precision pipelines, usable as:
    - scalar FPU
    - 4-wide FPU SIMD
    - 2-wide complex arithmetic SIMD
  - Instruction extensions to PowerISA
  - 6 stage pipeline
  - 2W4R register file (2 * 2W2R) per pipe
  - 8 concurrent floating point ops (FMA) + load + store
  - Permute instructions to reorganize vector data
    - supports a multitude of data alignments



QPU: Quad FPU

Set of 8x8 Outer Products on BG/Q Basis of DGEMM



Streaming 16x16 Outer Products on BG/Q Basis of a **Better** DGEMM

COORDINATED PREFETCHES: USING THE L1 TO COORDINATE THROUGH L2 ACCESSES

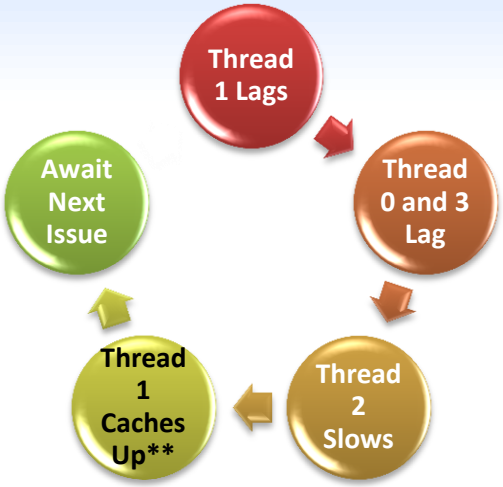
Streaming 16x16 Outer Products on BG/Q Basis of a **Better** DGEMM

- Of course, one can go further
  - Threads 0,1 prefetch A for 2 & 3
  - Threads 0,2 prefetch B for 1 & 3
  - Interleave the data (every thread prefetches every 4th expected request)
    - DGEMM specific

COORDINATED PREFETCHES: USING THE L1 TO COORDINATE THROUGH L2 ACCESSES

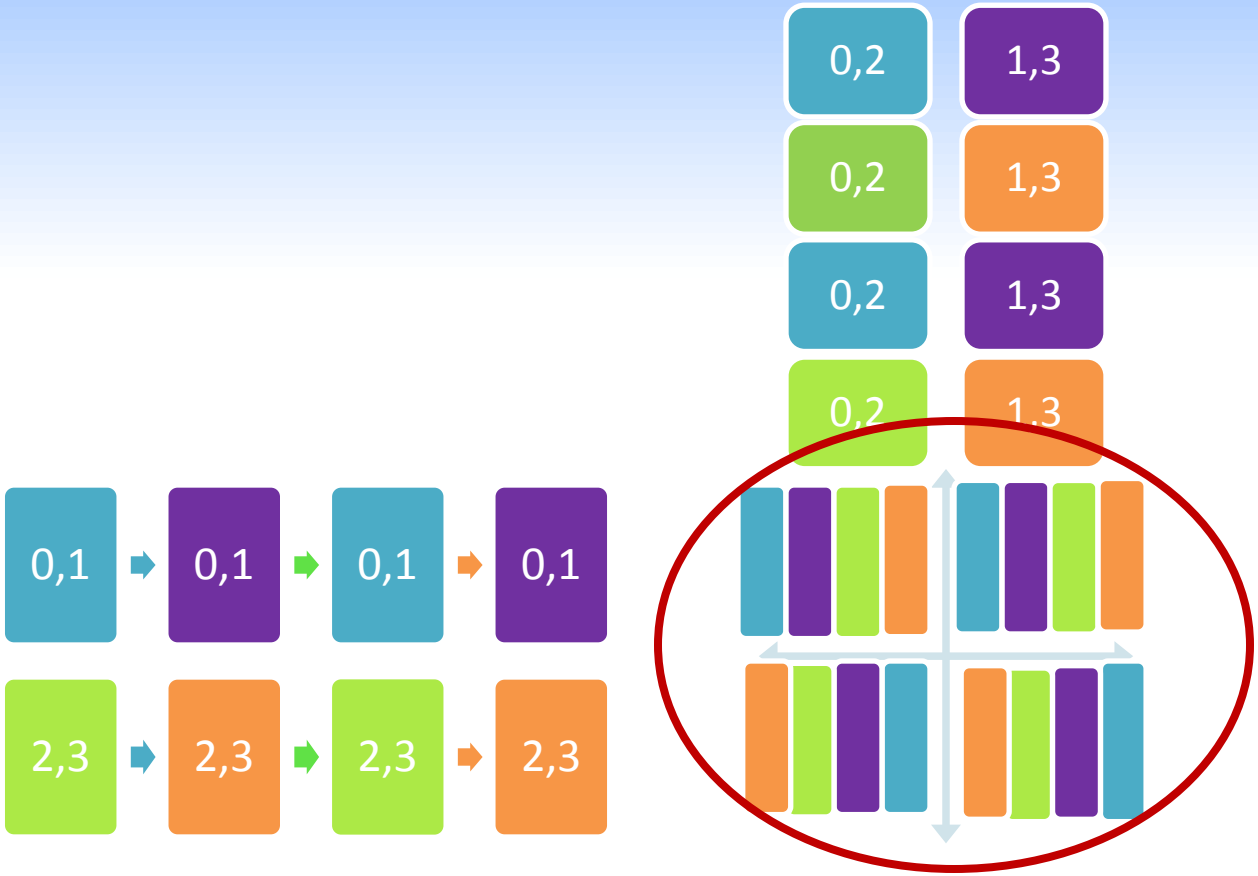
Streaming 16x16 Outer Products on BG/Q Basis of a **Self-Synchronizing** DGEMM

What happens if Thread 1 falls behind?



COORDINATED PREFETCHES: USING THE L1 TO COORDINATE THROUGH L2 ACCESSES

Streaming 16x16 Outer Products on BG/Q
A More **Performance-Robust** DGEMM



COORDINATED PREFETCHES: USING THE L2 TO COORDINATE THROUGH DDR ACCESSES

Benefits of Layered Implicit Synchronization

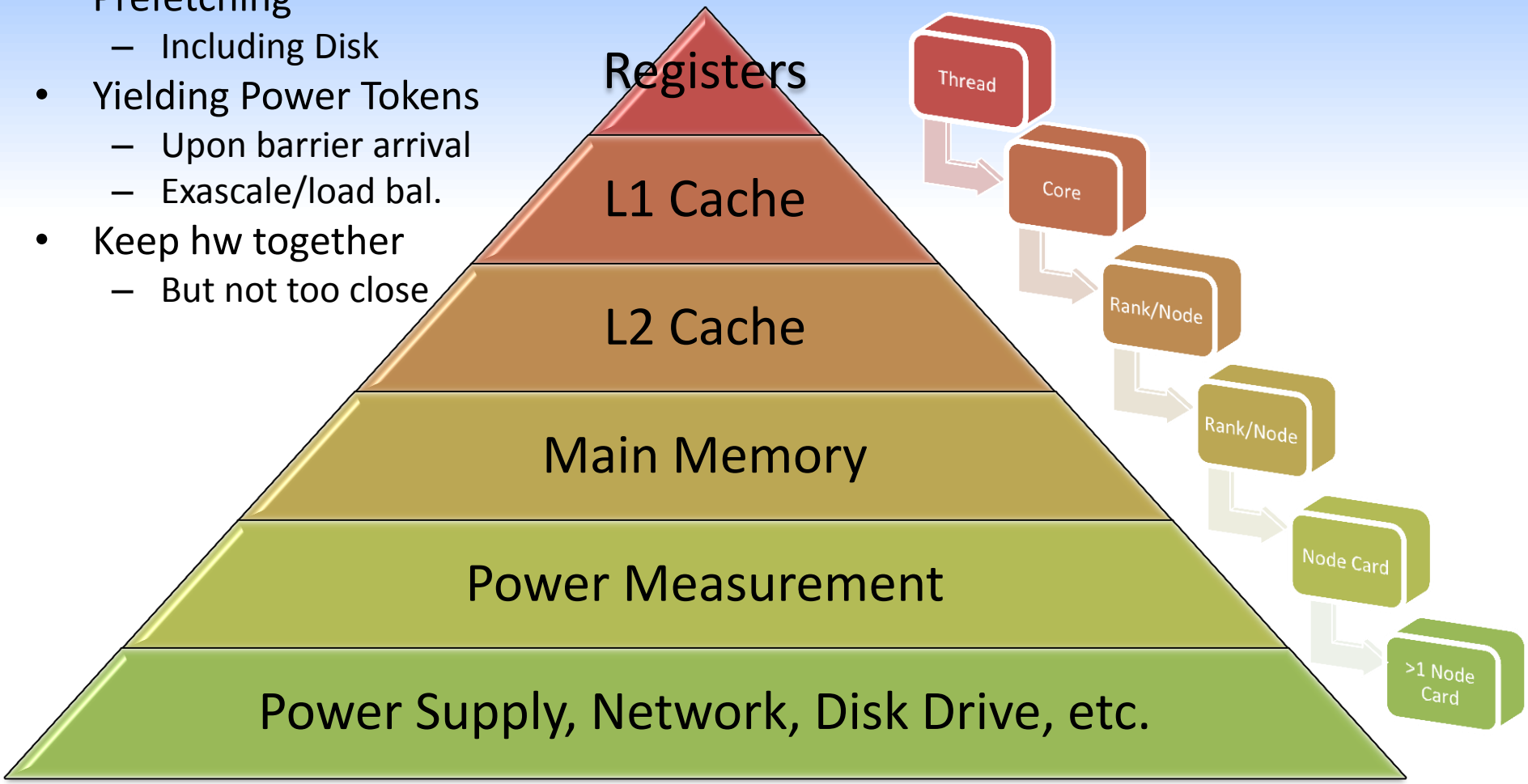
- Extremely infrequent explicit barriers
- Fewer instructions executed
  - No “expected false” prefetches
- 4 bytes/cycle/core L2 bandwidth
  - More reliably
- Similar approach
  - Quadruple SIMD length/double bandwidth
    - |loads| <= |FMAs| ((1x4)x(32x10) kernels)
    - Could be fed by an 8 byte/cycle L2
    - Instruction mix continues to allow explicit prefetch
- But is it only good for DGEMM?
  - Cooperative prefetching is more generally applicable
  - Works with hand-tuned ASM (need a lot of details to work well)
  - Some parts better-suited for compilers (detail management)

Skew and Anti Synchronization

- Skew synchronization
  - Goal: smoothing burst requests on a shared resource
  - Implement: differential blocking, kernel/method used
  - Result: staggering of task initialization/completion
- Anti synchronization
  - Akin to hands-on, even cycle-by-cycle, skewing
  - Enforce staggering, usually on a finer grain
    - Through implicit or explicit means (simple example …)
      - Thread 0 prefetches 100 cycles ahead of thread 1
      - Thread 1 prefetches 8 cycles ahead of thread 0

Shared Resource Hierarchy

- Cooperative Prefetching
  - Including Disk
- Yielding Power Tokens
  - Upon barrier arrival
  - Exascale/load bal.
- Keep hw together
  - But not too close



Outline

- Level of Ambition
- Tabloid Programming
- Performance Counters & Power Measurement
- Case Studies
  - Heavy- vs. Lightweight Synchronization: DGEMM
  - Trade-offs in Synchronization: HPL Benchmark
  - Shifting Operation Type: Stencil Computations
  - Lanczos Iteration Methodology: s-Step and Pipeline
  - Interacting Kernels: A Simple Tuning Framework
- Conclusions

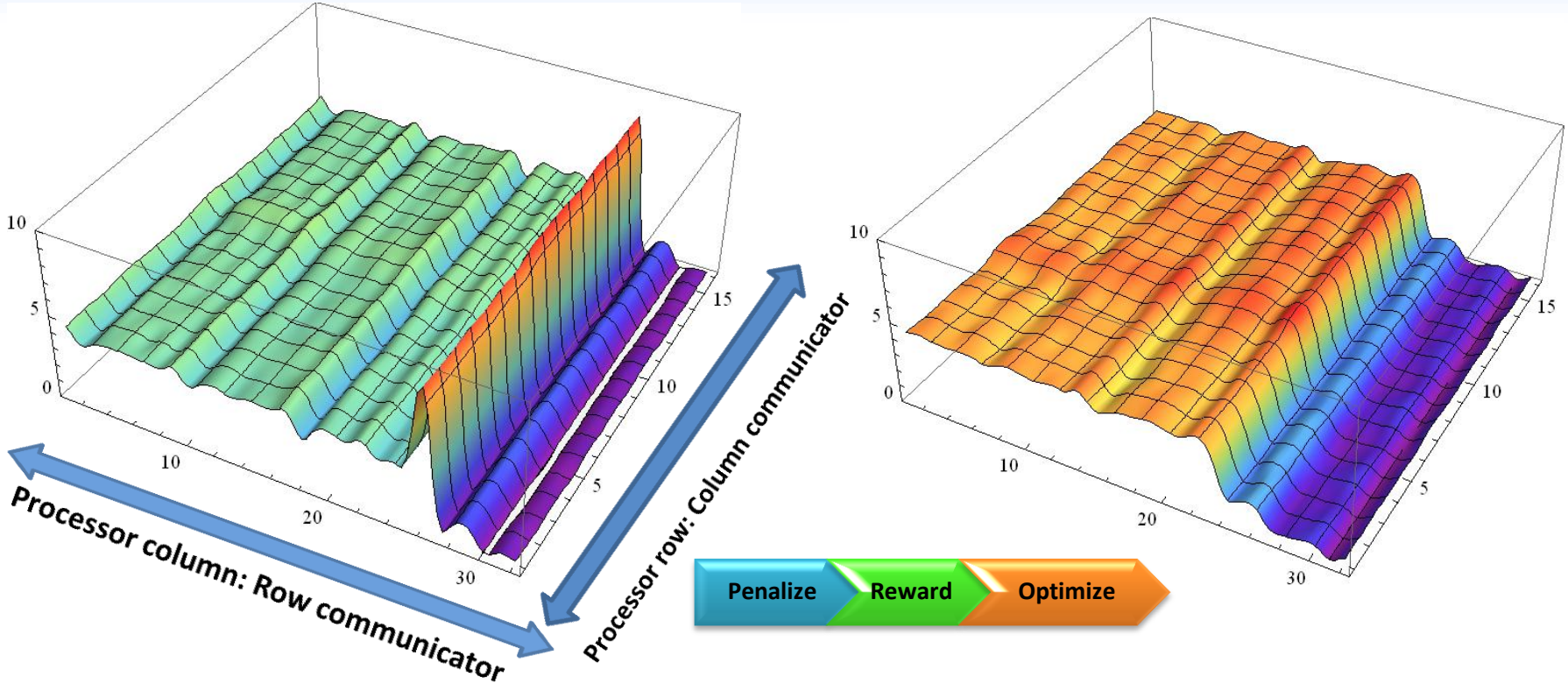
Trade-offs in Synchronization: HPL Benchmark

- Background: How is HPL asynchronous?
- What is the downside to synchronization
  - Performance
- What are the potential benefits
  - Multiple link usage/5D torus
  - Consistent numerical results
- Steps to reduce the disadvantages
  - What do timers tell us
  - Performance counters
  - Power measurements

What do we know and when do we know it?

- And how do we know it?
- A single step:
  - That panel factorization is a bottleneck (timers)
- Successive iterations:
  - Panel factorization is getting worse (timers)
  - What resource allocations help (perf. ctrs + timers)
- Successive rounds:
  - Which strategies were successful (pc + timers)
  - Predict success of overall plan (both + analytics)

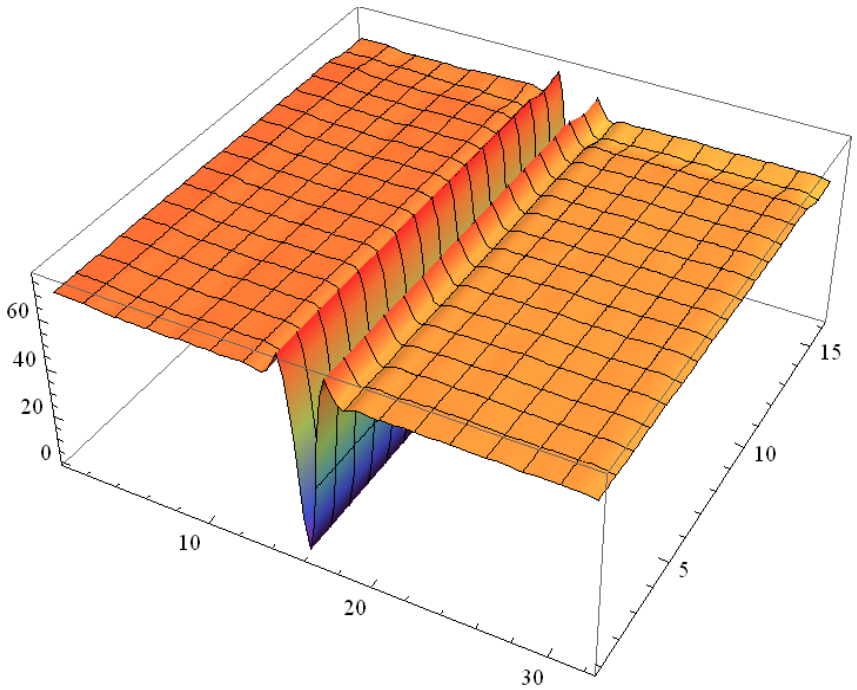
Driving Towards a Desired Profile



Dependent variable: Z-axis: Time in barrier (measured in terms of DGEMM register panels)

Prioritizing Resources

Panel Factorization Dominates



How To Accelerate Critical Path

- Performance counters information
  - High priority task is lagging
  - Lower priority tasks use conflicting resources
- Synthesize performance counter information at correct (perhaps dynamic) granularity (task)
- Throttle down the algorithm or priority of the lower priority tasks
- Increase the expected performance of the higher priority task
  - Always critical path
  - But it's **resource priority** was previously low
  - Larger "gang" for scheduling

Outline

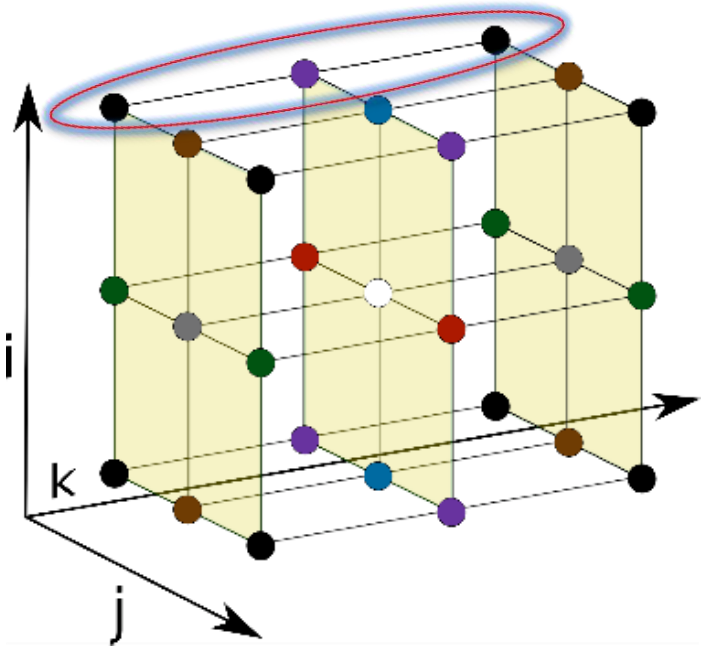
- Level of Ambition
- Tabloid Programming
- Performance Counters & Power Measurement
- Case Studies
  - Heavy- vs. Lightweight Synchronization: DGEMM
  - Trade-offs in Synchronization: HPL Benchmark
  - Shifting Operation Type: Stencil Computations
  - Lanczos Iteration Methodology: s-Step and Pipeline
  - Interacting Kernels: A Simple Tuning Framework
- Conclusions

Shifting Operation Type: Stencil Computations

- Simple stencil computations
- Tuning: unroll-and-jam + asm code scheduler
  - How far can you take this
    - How symmetric is your stencil
    - How many registers can you use/control
  - How far do you need to take it
- Instruction mix on Blue Gene/P
- Threading, synchronization, and instruction mix on Blue Gene/Q

Engineering tactics

- Building block: 3-point stencil computation
  - Optimize then replicate into larger stencils



Why is tuning this computation on the BG/P PowerPC 450d difficult?

- Utilizes features to improve efficiency
  - SIMDized fused floating point units
  - Multiple loads or fewer loads + shifts

For (i=0; i<N; i++)

A[i] = B[i] + B[i+1]

Not Aligned

B	0	1	2	3	4	5	.	.	N

A	0	1	2	3	4	5	.	.	N

Example

Python code

```
for i in self.block_ind: istream += self.fma_block(com.w, com.streams, com.results, i, self.K0)
istream += [isa.lfsdux(com.streams[i],com.a_ptr,com.a_indexing[i]) for i in range(self.FRAME_SIZE)]
```

Generated code

Without interleaving – 19 cycles

```
asm volatile("fxpmul 16, 31, 0");
asm volatile("fxpmul 17, 31, 1");
asm volatile("fxpmul 18, 31, 4");
asm volatile("fxpmul 19, 31, 5");
asm volatile("fxcpmadd 16, 30, 1, 16");
asm volatile("fxcpmadd 17, 30, 2, 17");
asm volatile("fxcpmadd 18, 30, 5, 18");
asm volatile("fxcpmadd 19, 30, 6, 19");
asm volatile("fxcpmadd 16, 31, 2, 16");
asm volatile("fxcpmadd 17, 31, 3, 17");
asm volatile("fxcpmadd 18, 31, 6, 18");
asm volatile("fxcpmadd 19, 31, 7, 19");
asm volatile("lfsdux 0, %0, %1":"+b" (a_ptr):"b" (next_frame));
asm volatile("lfsdux 1, %0, %1":"+b" (a_ptr):"b" (next_j_jam));
asm volatile("lfsdux 2, %0, %1":"+b" (a_ptr):"b" (next_j_jam));
asm volatile("lfsdux 3, %0, %1":"+b" (a_ptr):"b" (next_j_jam));
```

[0] fxpmul(rt=16, ra=31, rc=0)

[0] -- Instruction unit in use: floating point

[1] fxpmul(rt=17, ra=31, rc=1)

[1] -- Instruction unit in use: floating point

[2] fxpmul(rt=18, ra=31, rc=4)

[2] -- Instruction unit in use: floating point

.

.

[17] lfsdux(frt=2, ra=3, rb=5)

[17] -- Instruction unit in use: load/store

[18] -- Instruction unit in use: load/store

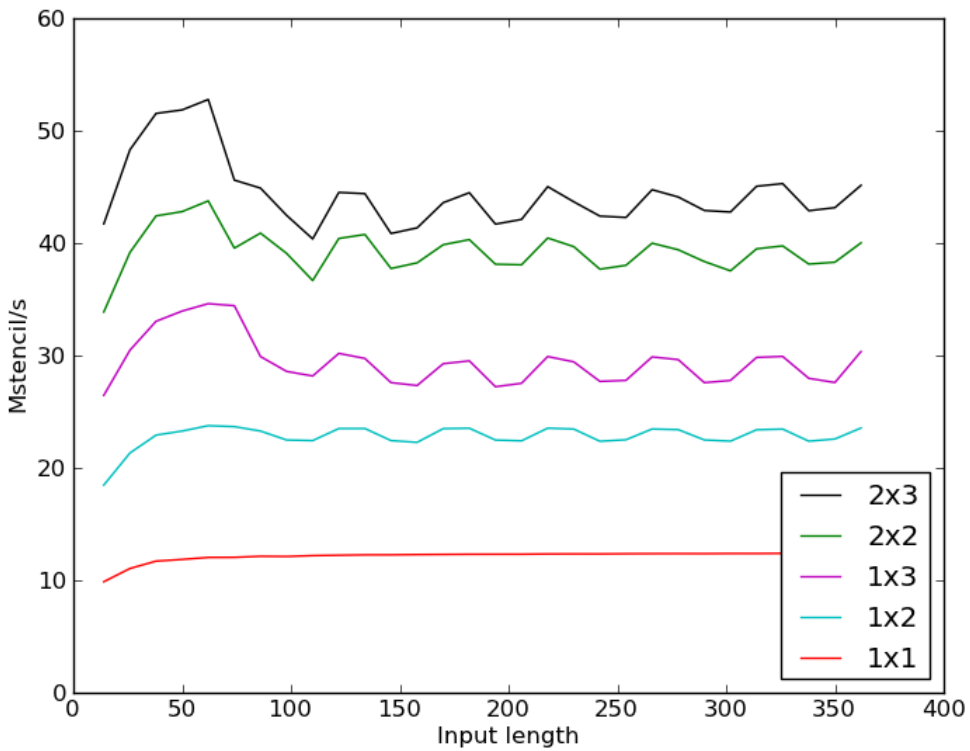
[19] lfsdux(frt=3, ra=3, rb=5)

With interleaving – 13 cycles

```
asm volatile("fxpmul 16, 31, 0");
asm volatile("lfsdux 0, %0, %1":"+b" (a_ptr):"b" (next_frame));
asm volatile("fxpmul 17, 31, 1");
asm volatile("fxpmul 18, 31, 4");
asm volatile("fxpmul 19, 31, 5");
asm volatile("fxcpmadd 16, 30, 1, 16");
asm volatile("lfsdux 1, %0, %1":"+b" (a_ptr):"b" (next_j_jam));
asm volatile("fxcpmadd 17, 30, 2, 17");
asm volatile("fxcpmadd 18, 30, 5, 18");
asm volatile("fxcpmadd 19, 30, 6, 19");
asm volatile("fxcpmadd 16, 31, 2, 16");
asm volatile("lfsdux 2, %0, %1":"+b" (a_ptr):"b" (next_j_jam));
asm volatile("fxcpmadd 17, 31, 3, 17");
asm volatile("fxcpmadd 18, 31, 6, 18");
asm volatile("lfsdux 3, %0, %1":"+b" (a_ptr):"b" (next_j_jam));
asm volatile("fxcpmadd 19, 31, 7, 19");
```

27-Point Stencil Results

- **Increasing arithmetic intensity (+)**
- **Right mix of instructions (+)**
- **Improving perform. model (+)**
- **Uneven performance due to co-alignment effects (-)**
- **"Optimizing the Performance of Streaming Numerical Kernels on the IBM Blue Gene/P PowerPC 450“ (M.S. Thesis)**



Architectural/Implementation Evolution

Blue Gene/P

- 2-way SIMD Operations
- Dual-issue per thread
  - One thread per core
- Rich Load/Store ISA
- High main memory bw
  - Streaming important
- 5 prefetch streams/core
- 3 outstanding loads/core
- 9 loads/8 shifts vs. 16 loads

Blue Gene/Q

- 4-way SIMD Operations
- Single-Issue per thread
  - Dual-Issue per core
- Rich Permute ISA
- BW/FLOPS reduced
  - Blocking more important
- 16 prefetch streams/core
- 9 outstanding loads/core
- 5 loads/4 perms vs. 16 loads

- Manage cache line/bank accesses:
  - Synchronize: layout was extremely careful, stencil driving, or skew
  - Async: between cores, drift may get multiple bank accesses (other within core)
- Manage cache occupancy, stream count
  - Synchronized: Explicit, “forced” implicit
  - Asynchronous: Merge kernels?, L1 blocking for worst case behavior

Outline

- Level of Ambition
- Tabloid Programming
- Performance Counters & Power Measurement
- Case Studies
  - Heavy- vs. Lightweight Synchronization: DGEMM
  - Trade-offs in Synchronization: HPL Benchmark
  - Shifting Operation Type: Stencil Computations
  - Lanczos Iteration Methodology: s-Step and Pipeline
  - Interacting Kernels: A Simple Tuning Framework
- Conclusions

Lanczos Iteration

- Recursion relation

$$
\begin{aligned}
\beta_{i+1} q_{i+1} &= A q_i - \alpha_i q_i - \beta_i q_{i-1} \\
\alpha_i &= (q_i, A q_i) \\
\beta_{i+1} &= |A q_i - \alpha_i q_i - \beta_i q_{i-1}|
\end{aligned}
$$

- Global synch evaluating inner product
- Latency must be paid at every iteration

Hiding the latency

- The idea
  - Overlapping M-v multiplication and inner product

	Ready	Deduce	Start
	q_0		Aq_0
$i=0$	$Aq_0, (q_0, q_0) = 1$		$A^2q_0, (q_0, Aq_0), (q_0A, Aq_0)$
$i=1$	$A^2q_0, (q_0, Aq_0), (q_0A, Aq_0)$	α_0, β_1	
		Aq_1, q_1	$A^2q_1, (q_1, Aq_1), (q_1A, Aq_1)$
$i=2$	$A^2q_1, (q_1, Aq_1), (q_1A, Aq_1)$	α_1, β_2	
		Aq_2, q_2	$A^2q_2, (q_2, Aq_2), (q_2A, Aq_2)$
$i=3$	$A^2q_2, (q_2, Aq_2), (q_2A, Aq_2)$	α_2, β_3	
		Aq_3, q_3	$A^2q_3, (q_3, Aq_3), (q_3A, Aq_3)$
	$\vdots$	$\vdots$	$\vdots$

Hiding the latency

- If the latency is dominating
  - e.g. inner product takes twice as long as M-v

	Ready	Deduce	Start
	q_0		Aq_0
	Aq_0		$A^2q_0, (q_0, Aq_0), (q_0A, Aq_0)$
$i = 0$	A^2q_0		$A^3q_0, (q_0A^2, A^2q_0), (q_0A^2, Aq_0)$
$i = 1$	$A^3q_0, (q_0, Aq_0), (q_0A, Aq_0)$	α_0, β_1 A^2q_1, Aq_1, q_1	$A^3q_1, (q_1A, A^2q_1), (q_1A^2, A^2q_1)$
$i = 2$	$A^3q_1, (q_0A, A^2q_0), (q_0A^2, A^2q_0)$	α_1, β_2 A^2q_2, Aq_2, q_2	$A^3q_2, (q_2A, A^2q_2), (q_2A^2, A^2q_2)$
$i = 3$	$A^3q_2, (q_1A, A^2q_1), (q_1A^2, A^2q_1)$	α_2, β_3 A^2q_3, Aq_3, q_3	$A^3q_3, (q_3A, A^2q_3), (q_3A^2, A^2q_3)$
$i = 4$	$A^3q_3, (q_2A, A^2q_2), (q_2A^2, A^2q_2)$	α_3, β_4 A^2q_4, Aq_4, q_3	$A^3q_4, (q_4A, A^2q_4), (q_4A^2, A^2q_4)$
	$\vdots$	$\vdots$	$\vdots$

Hiding the latency

- The latency is paid only once
- Deducing step is completely local
  - Only vector addition. Daxpy.
  - Small overhead
- The algorithm depends on indirect evaluation of vector norm (e.g. β)
  - Numerical stability issue
- Similar technique might be applied to CG
  - Numerical stability might be improved by clever method
- Analogous "plumbing" was applied in the context of an optimization problem on Blue Gene/P
  - "Efficient high-precision matrix algebra on parallel architectures for nonlinear combinatorial optimization"
  - Currently using MPI/SPI approach
  - Exploring task-based libraries, including PFunc

Outline

- Level of Ambition
- Tabloid Programming
- Performance Counters & Power Measurement
- Case Studies
  - Heavy- vs. Lightweight Synchronization: DGEMM
  - Trade-offs in Synchronization: HPL Benchmark
  - Shifting Operation Type: Stencil Computations
  - Lanczos Iteration Methodology: s-Step and Pipeline
  - Interacting Kernels: A Simple Tuning Framework
- Conclusions

Interacting Kernels: A Simple Tuning Framework

- A symbolic execution framework
  - Target: Blue Gene/Q
  - With “hooks” for generic architecture
- Some advantages of symbolic execution
- Detailed knowledge of architecture
  - Straightforward (slow) architecture simulation
  - Time-stepping techniques help
- Feedback to user (library writer, others)
  - Timings, color-coded accesses

Interacting Kernels: A Simple Tuning Framework

- How does this relate to synchronization?
  - Engage multiple threads
  - Utilize multiple cores, introduce noise
  - Are they coordinated? Should they be? In what way?
- Moderate success thus far
  - Scheduled new DGEMM **kernels**
    - Reflects potential for cooperative prefetch, does not automate it
  - "Re-"scheduled ddcMD kernel (BG/P matching BG/L perf.)
  - Co-mingle two thread kernels under certain assumptions
    - Sometimes split, sometimes combine

High-Level Improvements Needed

- Discovering patterns:
  - Shared L2 prefetch
    - Easy to see, does not happen every time, difficult to auto-discover
  - Similar schedules
    - By default, the system constructs 64 scheduled instruction streams
      - Sometimes this makes sense, but usually it does not
  - More intelligent use of “macro operators”
    - First, wrt data layouts (currently: “greedy-not-quite-stupid”)
  - The instruction streams only self schedule per thread
    - Information that a particular prefetch was wasted present, not used
  - Suggest “code fusion”
    - Register re-coloring
    - Barring that … summarize which threads could be fused

Practical Concerns: Runtime

- The Timing is linear in the size of the array, but not practical for some goals
  - In[5]:= Timing[For[i=0,i<= 1000,i++,Rest[L2]];]
    - Out[5]= {3.775,Null} (* 3.8 seconds for 1000 steps!!! *)
  - In[6]:= Timing[For[i=0,i<= 1000000,i++,Rest[L1]];]
    - Out[6]= {1.919,Null}
  - In[7]:= Length[L2]/Length[L1]
    - Out[7]= 2048
- Some fixes are simple
  - Associativity, homogenous core action/sharing, etc., but sometimes at odds with reality

Outline

- Level of Ambition
- Tabloid Programming
- Performance Counters & Power Measurement
- Case Studies
  - Heavy- vs. Lightweight Synchronization: DGEMM
  - Trade-offs in Synchronization: HPL Benchmark
  - Shifting Operation Type: Stencil Computations
  - Lanczos Iteration Methodology: s-Step and Pipeline
  - Interacting Kernels: A Simple Tuning Framework
- Conclusions

Conclusions

- Synchronization opportunities and trade-offs
  - Exchange information (+)
  - Provide a timing heartbeat (+ … for some cases)
  - Often things settle to a reasonable level (-)
- Task characterization and accumulation
  - Benefit to co-scheduling complementary tasks
    - And task characterization (chokepoints)
  - Benefit to co-scheduling identical tasks
    - Thread recruitment, dynamic ranks-per-node, etc.
  - Like to be able to break task encapsulation
    - Simple example: pull off a task “blob” …
      - Need to be able to gang schedule or push it back for better time

Conclusions

- Descriptive, Predictive, Prescriptive Analytics might have a place in exascale HPC
  - You say those flops are free? Intops?
- Power might need to be considered as a parameter in lower-level codes (libraries)
- Ideally, would like to control how far apart operations are without incurring crosstalk
  - Sometimes want them close, other times … no

Current Work

- Code generator/tuner
  - Present focus, incorporating power estimation
    - GreenBLAS
  - Adding more instructions to repertoire
    - ASM, intrinsics, C-like (building blocks)
    - Cross-thread/core
  - Compressing information
    - Multiple time steps in generator
    - Useful patterns from performance counters+power
- Exascale solvers
  - Range of applicability, stability and iteration issues
  - How to implement the underlying communication
  - Kernel coding and fusion

Acknowledgments

- Argonne National Laboratory
  - **Jed Brown***
- KAUST
  - Aron Ahamdia
  - **David Keyes***
  - Tareq Malas
- Lawrence Livermore National Laboratory
  - Bor Chan
  - Erik Draeger
  - James Glosli
  - David Richards
- London School of Economics
  - Gregory Sorkin
- Penn State University
  - Susan Margulies
- University of Michigan
  - Jon Lee
- IBM Research
  - Vernon Austel
  - Haim Avron
  - Fabio Checconi
  - Alexandre Eichenberger
  - Anshul Gupta
  - Prabhanjan Kambadur
  - Changhoan Kim
  - Fabrizio Petrini
  - **James Sexton***
  - Robert Walkup
- *The errors, oversights, and gaffes introduced are, of course, solely owned by the speaker*

***Workshop attendee**

Acknowledgements

- The Blue Gene/Q project has been supported and partially funded by Argonne National Laboratory and the Lawrence Livermore National Laboratory on behalf of the United States Department of Energy, under Lawrence Livermore National Laboratory Subcontract No. B554331
- Investigation into Blue Gene/P architectural simulation, nonlinear optimization, stencil computations, and asynchronous solvers was funded by The King Abdullah University of Science and Technology (KAUST)

Backup

Lasciate ogne speranza,
voi ch'entrate

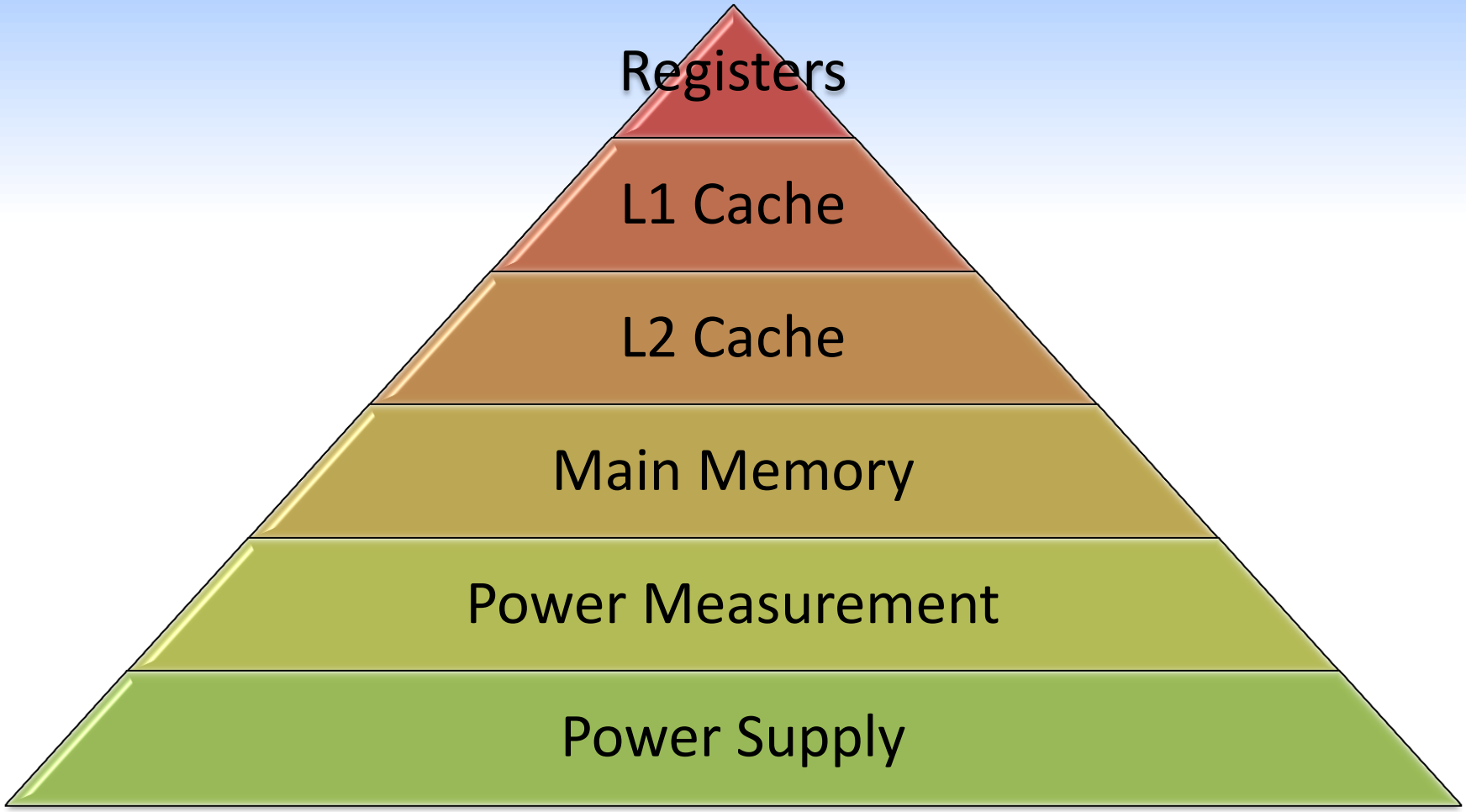
Making Good Enough...Better:

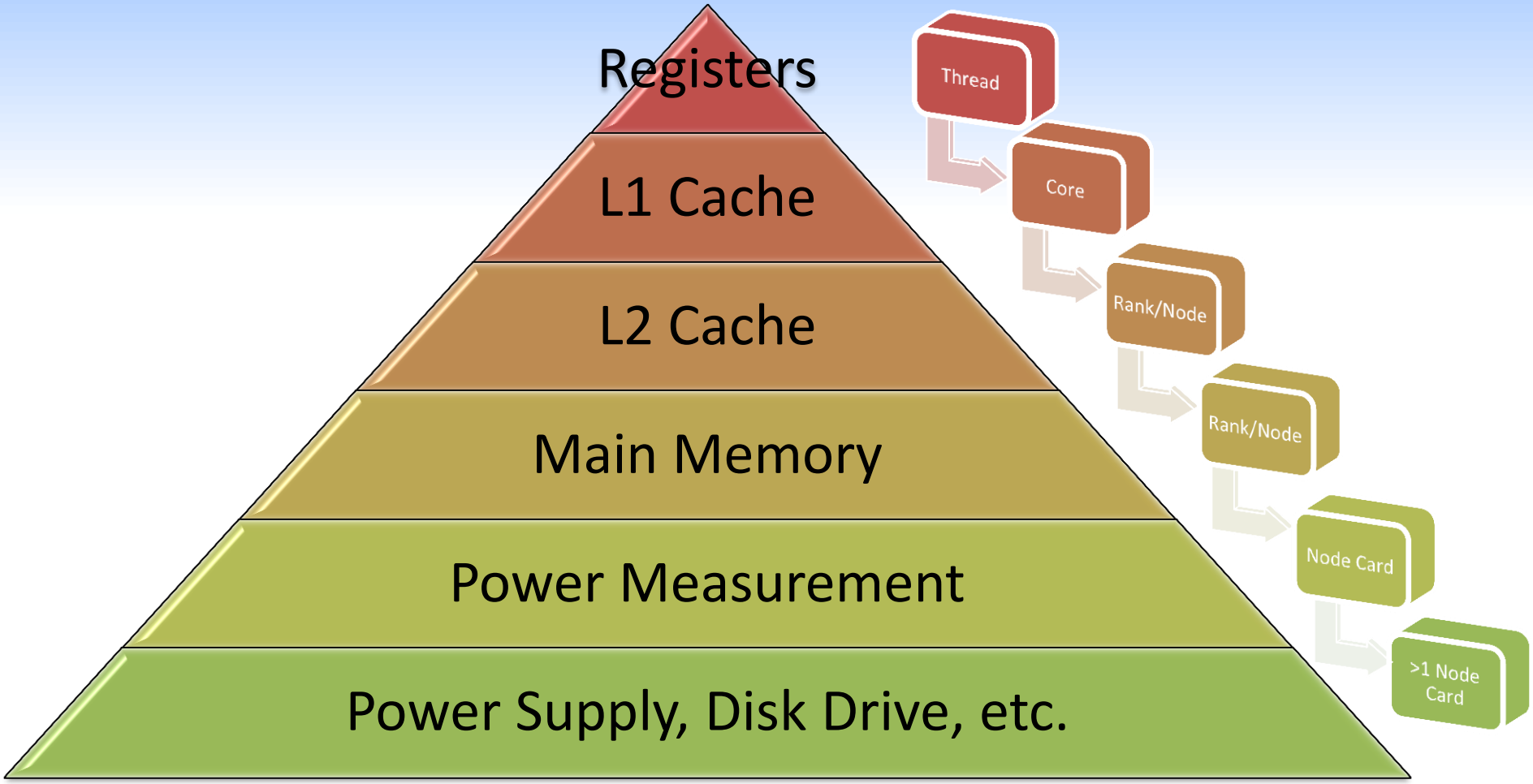
Addressing the Multiple Objectives of High-Performance Parallel Software with a Mixed Global-Local Worldview

John A. Gunnels
Research Staff Member/Manager
IBM T.J. Watson Research Center
Business Analytics & Mathematical Sciences

PFunc

- Highly portable open-source shared-memory task parallel library for C/C++
- Some differentiating features from Cilk, TBB, and other Cilk-derivatives
  - Customizable task scheduling, task stealing, and task priorities
  - Cilk-style, FIFO, LIFO, Priority-based pre-included
  - Support for SPMD-style parallelization through task groups
  - Spawn tasks on specific queues, bind threads to processors
  - Move seamlessly from work-stealing to work-sharing
  - Tasks can have multiple parents; native support for DAG executions
  - Zero abstraction penalty ensured by using template programming
- PFunc can execute DAGs similar to PLASMA and SuperMatrix
  - See ”Demand-driven execution of Static Directed Acyclic Graphs Using Task Parallelism” in HiPC 2009 --- demonstrates methodology for parallelizing a unsymmetric-pattern multifrontal algorithm for LU factorization with partial pivoting


Registers
L1 Cache
L2 Cache
Main Memory
Power Measurement
Power Supply


Registers
L1 Cache
L2 Cache
Main Memory
Power Measurement
Power Supply, Disk Drive, etc.
Thread
Core
Rank/Node
Rank/Node
Node Card
>1 Node Card

Latency Hiding Conjugate Gradient

Krylov Space

- Spanned by vectors generated by successive applications of matrix A
- Generation of those vectors requires only local communication
- Orthonormalization requires inner products of those vectors which requires *global communication*
- Here, the time unit is the time a single matrix vector multiplication takes. We denote the global communication latency as L.

Lanczos Iteration

- Core part of CG
- Method of orthonormalizing Krylov space for symmetric matrix
- Simpler than CG
  - Recursion relation

$$\beta_{i+1} p_{i+1} = A p_i - \alpha_i p_i - \beta_i p_{i-1}$$

$$\alpha_i = p_i \cdot A p_i$$

$$\beta_{i+1} = |A p_i - \alpha_i p_i - \beta_i p_{i-1}|$$

Lanczos iteration II

- The idea of hiding latency of inner product is to pre-calculate the inner product.
- We define
  - Because of symmetry of the matrix

$$h(i,j,k) = p_i^T A^j p_k$$

$$h(i,j,k) = h(k,j,i)$$

  - Using Lanczos recursion

$$h(i,j,k) = h(i,j+1,k-1) - \alpha_{k-1} h(i,j,k-1) - \beta_{k-1} h(i,j,k-2)$$

Lanczos iteration III

- The strategy is to pre-calculate
  - $h(i-L, 2L+1, i-L)$ for $\alpha_i = h(i, 1, i)$
  - $h(i-L, 2L+2, i-L)$ for β_{i+1}
- Hence, iterate with

$$\beta_{i+1} A^L p_{i+1} = A A^L p_i - \alpha_i A^L p_i - \beta_i A^L p_{i-1}$$

- calculating

$$p_i A^{2L+1} p_i, \qquad p_i A^{2L+2} p_i$$

  for α_{i+L} and β_{i+L+1}

Lanczos iteration III

- how to deduce $h(i,1,i)$ from $h(i-L,2L+1,i-L)$:

$$\begin{aligned}
h(i,1,i) &\rightarrow h(i,2,i-1), h(i,1,i-1), h(i,1,i-2) \\
h(i,2,i-1) &\rightarrow h(i-1,3,i-1), h(i-1,2,i-1), h(i-2,2,i-1) \\
h(i,1,i-1) &\rightarrow h(i-1,2,i-1), h(i-1,1,i-1), h(i-2,1,i-1) \\
h(i,1,i-2) &\rightarrow h(i-1,2,i-2), h(i-1,1,i-2), h(i-2,1,i-2)
\end{aligned}$$

- how to deduce $h(i,2,i)$ from $h(i-L,2L+2,i-L)$:

$$\begin{aligned}
h(i,2,i) &\rightarrow h(i,3,i-1), h(i,2,i-1), h(i,2,i-2) \\
h(i,3,i-1) &\rightarrow h(i-1,4,i-1), h(i-1,3,i-1), h(i-2,3,i-1) \\
h(i,2,i-1) &\rightarrow h(i-1,3,i-1), h(i-1,2,i-1), h(i-2,2,i-1) \\
h(i,2,i-2) &\rightarrow h(i-1,3,i-2), h(i-1,2,i-2), h(i-2,2,i-2)
\end{aligned}$$

- all the terms except $h(i-1,4,i-1)$ and $h(i-1,3,i-1)$ are already calculated from the last iteration.

Numerical instability

- During the testing the new recursion, numerical instability has been detected.
- Evaluation of a norm becomes negative

CG Iteration

$$
\begin{aligned}
d_0 &= r_0 = b - Ax_0 \\
\gamma_i &= \frac{r_i \cdot r_i}{d_i A d_i} \\
x_{i+1} &= x_i + \gamma_i d_i \\
r_{i+1} &= r_i - \gamma_i A d_i \\
\delta_{i+1} &= \frac{r_{i+1} \cdot r_{i+1}}{r_i r_i} \\
d_{i+1} &= r_{i+1} + \delta_{i+1} d_i
\end{aligned}
$$

CG iteration II

- Residuals are orthogonal to each other.
  - Analogous to Lanczos iteraiton
  - r_i recursion :

$$
\begin{aligned}
d_{i+1} &= r_{i+1} + \delta_{i+1} d_i \\
\gamma_{i-1} r_{i+1} &= \gamma_{i-1} r_i - \gamma_{i-1}\gamma_i A d_i \\
\delta_i \gamma_i r_i &= \delta_i \gamma_i r_{i-1} - \delta_i \gamma_i \gamma_{i-1} A d_{i-1} \\
\gamma_{i-1} r_{i+1} - \delta_i \gamma_i r_i &= \gamma_{i-1} r_i - \gamma_{i-1}\gamma_i A r_i - \delta_i \gamma_i r_{i-1}
\end{aligned}
$$

CG iteration III

- Deducing

$$
\begin{aligned}
d_i A d_i &= (r_i + \delta_i d_{i-1}) A (r_i + \delta_i d_{i-1}) \\
\gamma_{i-1} A d_{i-1} &= r_i - r_{i-1} \\
\gamma_{i-1} d_{i-1} A r_i &= r_i \cdot r_i \\
d_i A d_i &= r_i A r_i + \delta_i^2 d_{i-1} A d_{i-1} + \frac{\delta_i}{2\gamma_{i-1}} r_i \cdot r_i
\end{aligned}
$$

- $d_{i-1}Ad_{i-1}$ is known from previous iteration and everything is on r_i which is analogous to Lanczos vectors

CG Iteration IV

- This iteration shows better numerical precision yet still worse than the standard CG iteration.
- Maybe use restarting method more often.
- More study is needed