Center for Scalable Application Development Software (CScADS):

Automatic Performance Tuning Workshop

http://cscads.rice.edu/

Discussion and Feedback

Office of Science
U.S. DEPARTMENT OF ENERGY

Top Priority Questions for Discussion

- What are the tuning parameters? Are there a small fixed list of about 10, or does the list grow with each algorithm?
  - (Tiling, unrolling, prefetch, dma-list construction… are there more?)
- What should be the next set of challenge problems for compilers? Should we have library-oriented benchmarks for library-specific issues (as opposed to whole programs)?
- What kind of infrastructure could we share and how?
- What are other things to tune for besides performance?
- What architectures should we tune for? And how can this community feedback to architects?

Questions on Libraries

- What language / IR for code generators
  - Perl, Ruby (Cray), OCAML (FFTW), Lua (OSKI), GAP & SPL (Spiral), Python (Merrimac), Mathematica (Flame), C (Atlas), POET
- At what level should one autotune?
  - E.g., BLAS, LAPACK, applications
  - E.g., 1D Serial FFT, 3D FFT, applications
- Cache oblivious vs. aware
  - Does this depend on computational intensity?
- Can we eliminate empirical analysis through recursion or through model?

More questions

- Should we tune full applications (and if so how) or libraries/kernels?
- What are the performance (or productivity) difference that come from autotuning vs. tuning?
- Can autotuning help with non-obvious applications (i.e., when can it beat hand-tuning in performance or productivity)
- What are the tuning parameters? (Tiling, unrolling, prefetch, dma-list construction… are there more?)
- Heisenberg problem: how intrusive is measurement?
- What about applications where you don't have a fixed kernel or execution path? (e.g., sorting)

More Questions

- What about Petascale? Tuning for interconnect or MPI
- Do lessons learned at 8 apply to 100K cores?
- Are these very specialized tuning, or is it very general?
- Mutable and dynamic data structures vs. static?
- How could architecture make tuning easier or vice versa?
- Should the autotuning community use RAMP?
- Role of domain-specific languages in algorithm generation?
- How to do the things you can't get published? Boring but important and/or negative.

Questions

- What kind of infrastructure could we share and how?
  – Performance data
  – Performance counters
  – Reproducible results (Martin Vetterli from EPFL)
  – Test problems (e.g., matrices, kernels)
- What are other things to tune for besides performance?
  – Power (max heat)
  – Energy
  – Reliability
  – Multiple tuning parameters where one is held constant (e.g., sharing memory bus)

Questions on Compilers

- Compilers: what they can/should do?
  - Matteo: register allocation of straight-line code, but not scheduling
  - Keith: scheduling and more (NP-hard may be OK)
- If we only need roughly 8-10 transforms to get hand-tuned performance, why is this not already solved?
- Should we have library-oriented benchmarks for library-specific issues (as opposed to whole program).
  - In many common benchmarks (eg. spec), structures are declared statically, allowing compilers to fully analyze the code.
  - In libraries, we typically take unbounded arguments (i.e. matrices whose dimensions are only known at run time).
- Is there a role for compiler directed hardware counter information collection (via something like PAPI) to explore options for optimization during the compilation process?

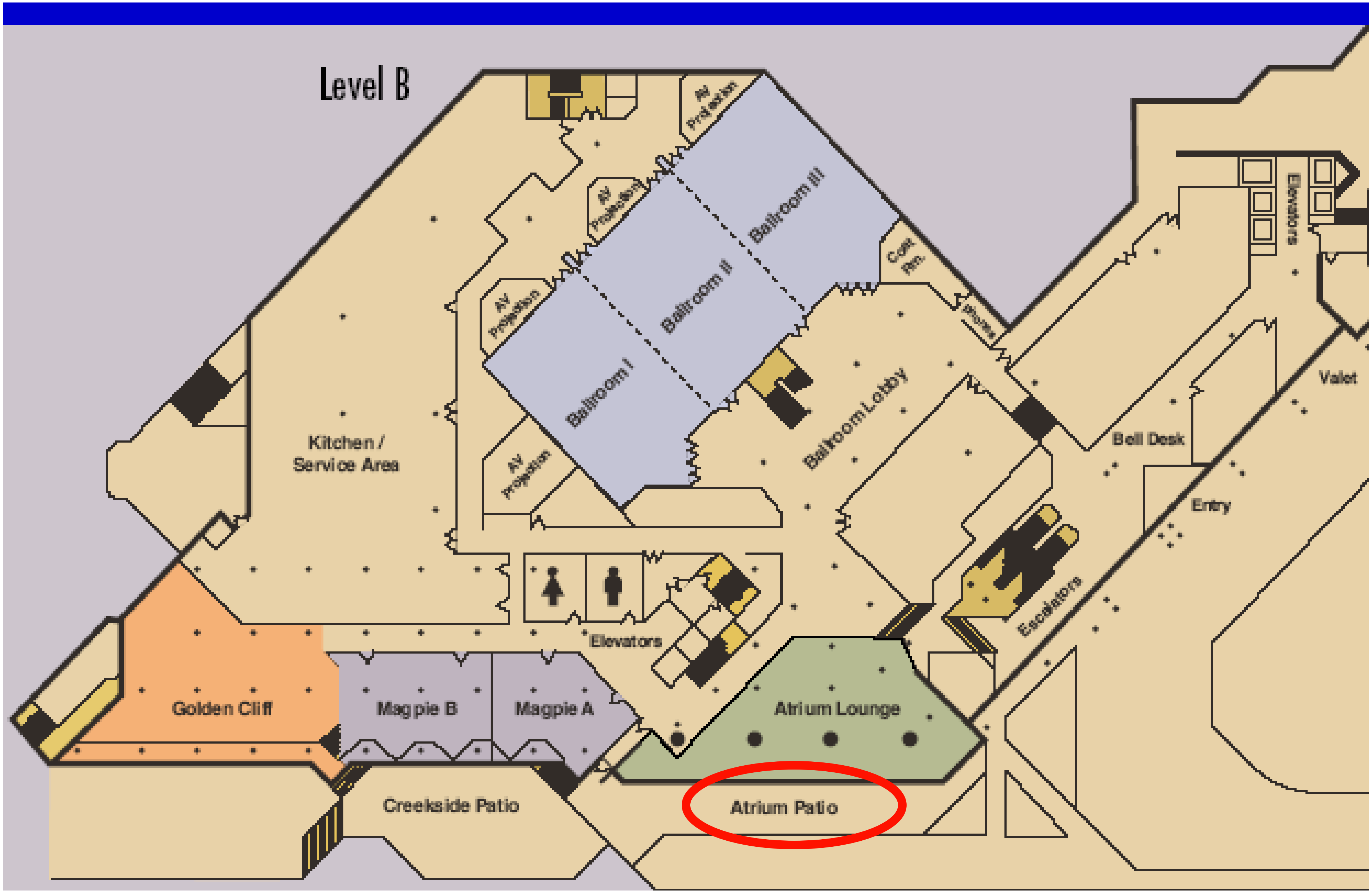
Questions for Tuning and Parallelism

- Can autotuners help get us over the multicore hurdle?
- How much parallelism can autotuners handle? Does search space get too large?
- If cores get simpler, will autotuning be less important
- Performance models: are they predictive enough?
- Are multicore chips easier/harder to get performance from than multi-socket shared memory multiprocessors?
- What architectures should we tune for? And how can this community feedback to architects?

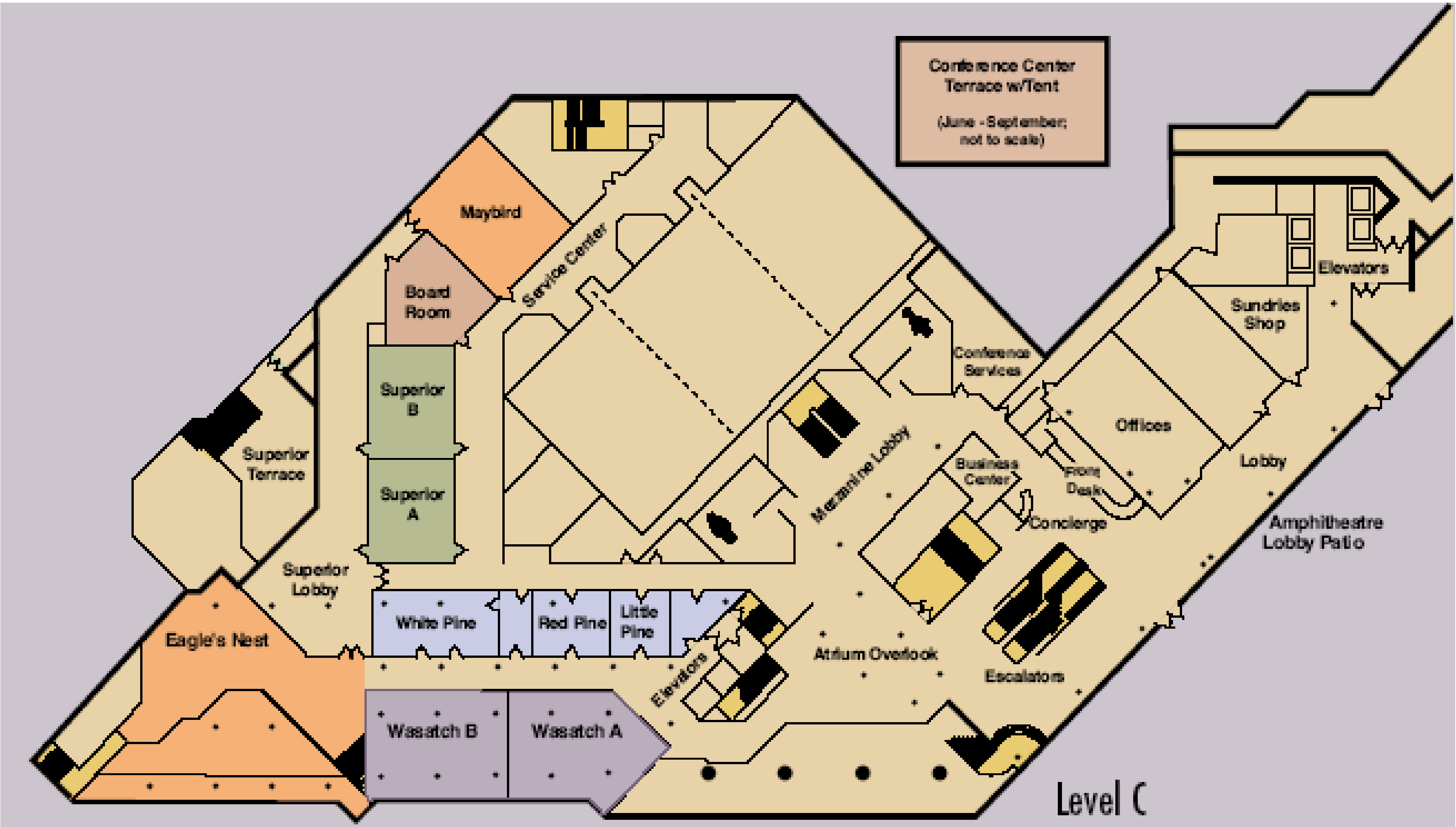
Infrastructure to Share

- Cache flush
  - Atlas BlAS1 self-flush: allocate multiple vectors
  - Context-sensitive timing; GEMV in or out of cache

Dinner/Reception Level (B)



Cliff House Registration Level (C)



Plans for Next Year

- Role of runtime adaptation
- Petascale vendors who tune application
- A lot of tuning that wasn't autotuning
- Groups to include next time
  - Advance Execution Systems crowd (NSF)
  - Embedded systems folks
- Tuning for other performance parameters:
  - Power
  - Bandwidth
  - Energy
  - Reliability
  - Memory size
  - Quality of Service and/or Worst case execution time
  - Accuracy

Benchmarks

- Beyond dense matrix multiply
- Beyond SPEC: testing what we used to run
- Formulation of problem statements, rather than code
- What autotuners are the highest priority
  - Sorting, graph algorithms, graphical models
  - Complex mutable data structures: mesh generation, K-d tree build, sparse LU,

Success Metrics

- Metrics
  - Percent of memory bandwidth (or throughput)
  - Utilization of a particular part of the system
  - Productivity (use vs. time to write)
- Reproducible results
  - Including version numbers
  - Processor (CPU) ID; chipset and its setting; BIOS revision, DRAM parameters; clock speed,
  - Microbenchmarks for load/store bandwidth, latency, e.g., XRay, Atlas (very general and portable), GPU bench
  - FutureMark

What Hardware?

- Relying on shared memory with coherence will run out
  - E.g., local stores or caches without coherence
  - GPU will converge with CPU
  - We had: 1) superscalar multicore 2) streaming multicore; 3) multithreading multicore (don't count on #1)
  - 1-2 big cores + smaller is hard to schedule; multiprogramming will lead to homogeneity
- General issues:
  - Minimizing the number of messages
  - Hiding latency
  - Hierarchical memory systems
  - Optimizing DRAM (large transfers) cost of opening a page his high

More on hardware

- Machines optimized for throughput will have worse latency
- Local bandwidth will scale maybe with optical global on-chip will scale
- Off-chip bandwidth
  - Stacking (L4, L5,…L8 caches)
  - Can use as aggregation buffer for slower memory
- Non-uniform caches will come soon:
  - Currently have stacked hierarchy
  - Each core will have L2, but will act as coherent L2
  - Single large with different costs to hit (partition cache by addresses)

Driving Applications

- What are the driving applications
  - Single threaded applications (“no” controversial)
  - What is good enough? Not in single thread performance, it’s in multidata (video, media) → many small cores
- Exercise: Assume
  - Single thread (no parallelism)
  - 95% cache hit rate
  - How do you make it scale?
- Games
  - Speech recognition, AI, …
  - Willing to give up order of magnitude in performance for ease of programming
  - Game is a 4-5 year commitment (get Tim Sweeney from EPIC Games or Gabe Newell from Valve)

Algorithms in Games

- Object collision (on variable tree structures)
- Particle systems (trees, n^2 with cutoff)
- Transport algorithms
- Geometric deformation (unstructured grids, AMR)
- Stencil computations (convolutions, etc.)

Short vs. Long-term Focus

- Mostly < 5-year picture
- Need longer term vision
- How to influence *future* hardware
- Bring numbers

Theory Question

- Given a computation DAG, with weighted nodes (computation) and edges (communication)
- Parallelism is partitioning
- Sequential case on memory hierarchy is scheduling
- Replication is a standard way to transform the DAG to avoid communication in parallelism
- These are all discrete problems; can you transform to something continuous?

What Data Structures?

- What Data Structures can compilers analyzing?
- Should it be data-structure focused rather than algorithm focused
- Parallel data structure problem: too much locking
- Using prefix instead of locking and things in between