Autotuning:
Past, Present and Future

Kathy Yelick
Early Autotuners: Dense Matrices and FFTs

- The first autotuners were for dense linear algebra kernels and FFTs
- Few parameters to functions being tuned: size, machine
- **Functional portability** from C
- **Performance portability** from search at install time

![Performance Graph]

BLAS = Basic Linear Algebra Subroutine: matrix multiply, matrix-vector multiply,…
Recent Past Autotuners: Sparse Matrices

- **Sparse Matrix**
  - Significant index meta data
  - Irregular memory accesses
  - Memory bound

- **Autotuning**
  - Tune over data structures (add 0s)
  - Delayed tuning decisions until runtime
  - Still use significant install-time tuning (dense matrix in sparse format) with online specialization based on matrix structure

See theses from Im, Vuduc, Williams, and Jain
Recent Past Autotuners: Sparse Matrices

- OSKI: Optimized Sparse Kernel Interface
- Optimized for: size, machine, and matrix structure
- **Functional portability** from C (except for Cell/GPUs)
- **Performance portability** from install time search and model evaluation at runtime
- Later tuning, less opaque interface

See theses from Im, Vuduc, Williams, and Jain
Synchronization Avoiding Dense Linear Algebra

- LAPACK engineered around BLAS
- Hide parallelism in BLAS for productivity
- Creates serialization/synchronization point at interface
- More parallelism in higher level algorithms: LU, QR, Cholesky, Eigensolvers, etc.

Some edges omitted
Structured Mesh Autotuners

- **Stencil**
  - Regular memory accesses
  - Computational intensity depends on operators (# points, number of operations)

![3D Grid](image)

- [UltraSparc T2+ T5140 (Victoria Falls)](image)
- [Xeon X5550 (Nehalem)](image)

See PhD thesis by Datta and Williams, plus work by Kamil, Oliker, Shalf, Carter, and others
More Structured Grid Code Generators (LBMHD)

LBMHD is not always bandwidth limited: used SIMD, etc.

Joint work with Sam Williams, Lenny Oliker, John Shalf, and Jonathan Carter
Particle / Mesh Autotuning

- Iterate over particles; apply operator
- Adds synchronization tuning, plus density, distribution, size

Performance (GFlop/s)

Problem size B (150K grid points), 5 particles per cell.

C (Coarse), M (Medium), F (Fine), A (Atomics) indicate various synchronization strategies.

Kamesh Madduri, Sam Williams, Lenny Oliker, John Shalf
Tuning Over Aggregate Data Structures

- Need a better way to build tuners over common traversal patterns ("iterators") for arrays, trees, sets, lists
- Tuning depends on operator applied at each points, data structure to traverse, traversal order
- Generalize with SEJITS

No black box library for generated code
Trends in Autotuning: Higher Level Tuning

- Delay Tuning Decisions
  - Wait for information: matrix structure, system behavior,…

- Use Communication Avoiding Algorithms
  - Use larger kernels for better reuse of data

- Use Synchronization Avoiding Algorithms
  - Replace serial interfaces by event-driven schedulers

- Tune over more complex parameters
  - Data structure, heterogeneous architectures

- Additional optimization criteria
  - Performance, power, energy, reliability

- Tune over operators (higher order functions)
  - Stencils, graph/tree/particle traversals

- Tools to build and deploy: SEJITS and Synthesis
Approaches to Autotuning

How do we produce all of these (correct) versions?

- Using scripts (Python, perl, C,..)
- Transform high level representation (FFTW, Spiral)
- Compiling a domain-specific language (D-TEC)
- Compiling a general-purpose language (X-Tune)
- Dynamic compilation of a domain-specific (SEJITS)