BLUE WATERS

SUSTAINED PETASCALE COMPUTING

Petascale Software Challenges

William Gropp

www.cs.illinois.edu/~wgropp

GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

Petascale Software Challenges

- Why should you care?
- What are they?
- Which are different from non-petascale?
- What has changed since last year?

What Has Changed?

- Many things the same
  - No new parallel programming languages
- But there are changes
  - Greater realization of the importance of dynamic load balancing, even on a single multicore node
  - Maturing single node/GPU programming models
    - Mostly special cases, but important ones
  - Performance modeling providing better understanding and guidance
  - Programming models continue to evolve

Why Should You Care?

- Petascale is the canary in the mine
  - Clock speed scaling is over (ended 6 years ago)
  - Parallelism required to get more speed
    - GPUs are (just) another parallel architecture
  - Can only get so far by taking a serial code and making parts parallel
    - Amdahl's Law: Speedup limited to 1/(1-f), where f is the fraction of code (measured in time) that can be perfectly parallelized
    - Better solutions come from rethinking, not stretching existing solutions
- Necessary if you want to use Petascale systems soon

What are the Issues?

- Getting ready for Petascale means several things:
  - Performance on a node
    - Similar to issues on any system
    - Performance "correctness" requires having an analytic model
  - Parallelism
    - Finding and expressing concurrency
  - Scaling to 10k nodes (> 300K cores)
    - New algorithms and mathematical models
    - Adaptivity to "jitter"
    - Latency tolerance
  - Fault tolerance
    - Not required *yet*
    - But can help…

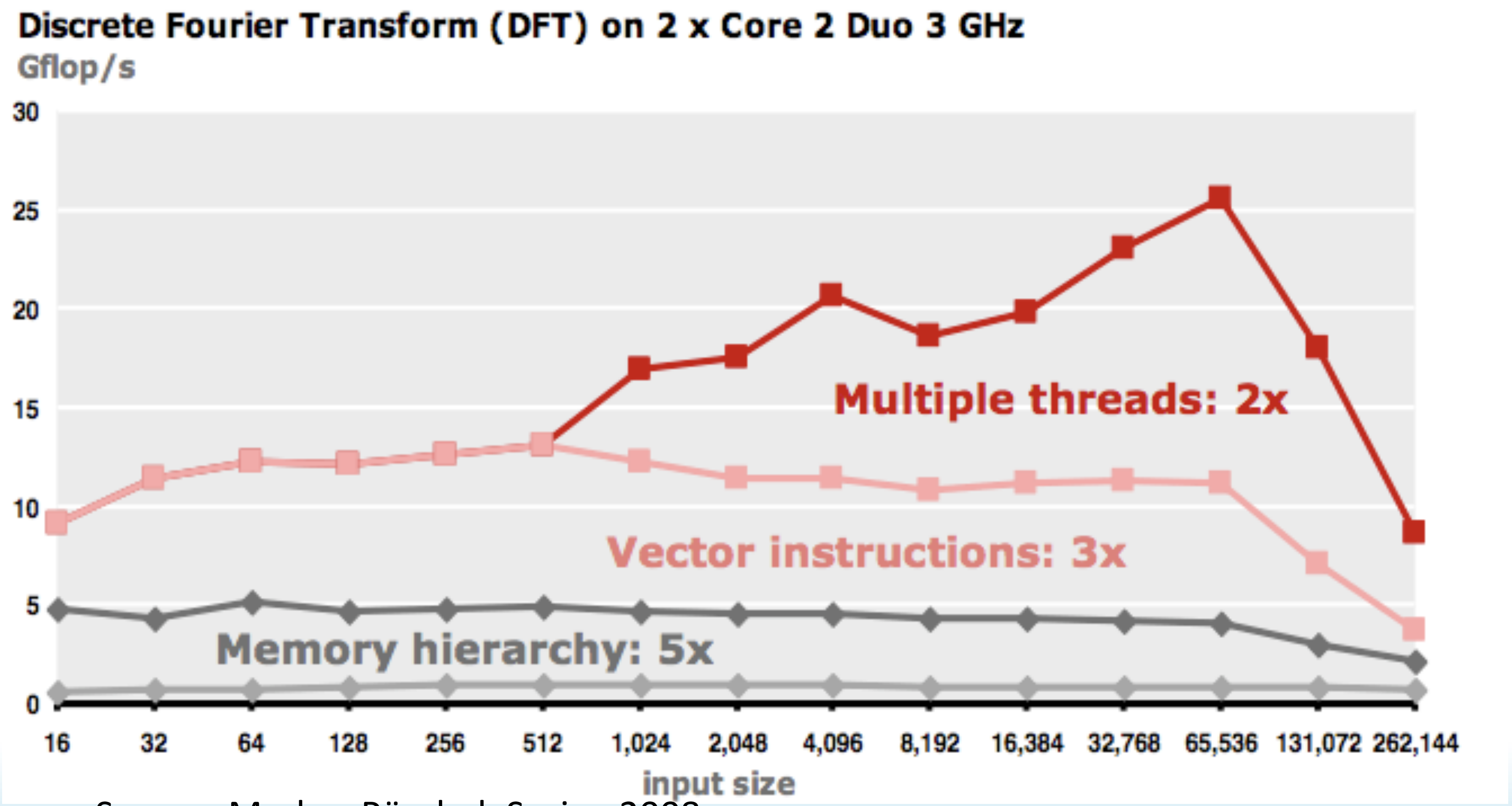
Performance on a Node

- Nodes are Symmetric Multiprocessors (SMP)
  - You have this problem on anything (even laptops)
- Tuning issues include the usual
  - Getting good performance out of the compiler (often means adapting to the memory hierarchy)
  - Also requires using all functional units (often means vectors)
- New (SMP) issues include
  - Sharing the SMP with other processes
  - Sharing the memory system
- Examples follow:
  - Vectorization (parallelism within a core)
  - Engineering for performance
  - Adapting to shared use

BLUE WATERS
SUSTAINED PETASCALE COMPUTING

GREAT LAKES CONSORTIUM
FOR PETASCALE COMPUTATION

Compilers versus Libraries in DFT



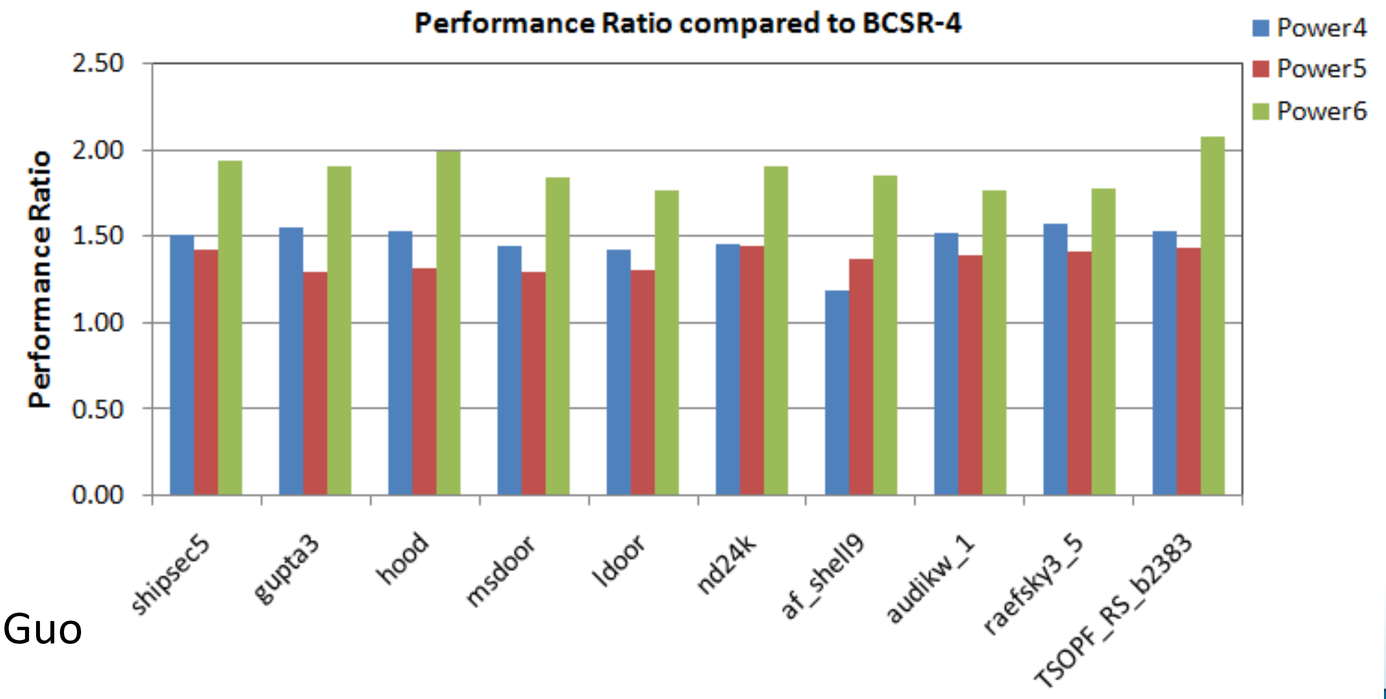
Source: Markus Püschel. Spring 2008.

Comments

- Compiler provided limited performance
- Achieving full performance required attention to memory hierarchy and to use of special "vector" instructions (with their own constraints)
- Most important optimizations, in order, were memory, vectors, and threads (i.e., the hardest stuff was the most important, and the easiest was the least)
- Knowing when to put in more effort requires knowing when it may pay off. Simple models can help identify performance "deficiencies"
  - Detailed prediction requires much more work; often not necessary or relevant to the algorithm designer
- Another example: Because memory bandwidth often limits performance, hardware often Changing the data structures and code order can enable bandwidth enhancing features in hardware. The following slide shows what you can achieve by designing for the prefetch system in the POWER architecture

Comparison of S-BCSR-4 format to BCSR-4 format

- The matrices are chosen with large data size (> 32 MB) and the performance of BCSR format is close to or better than CSR format.
- Performance Improvement of S-BCSR-4 format compared to BCSR-4 format:
- P4: 20 -60%, P5: 30 -45%, P6: 75 -108%



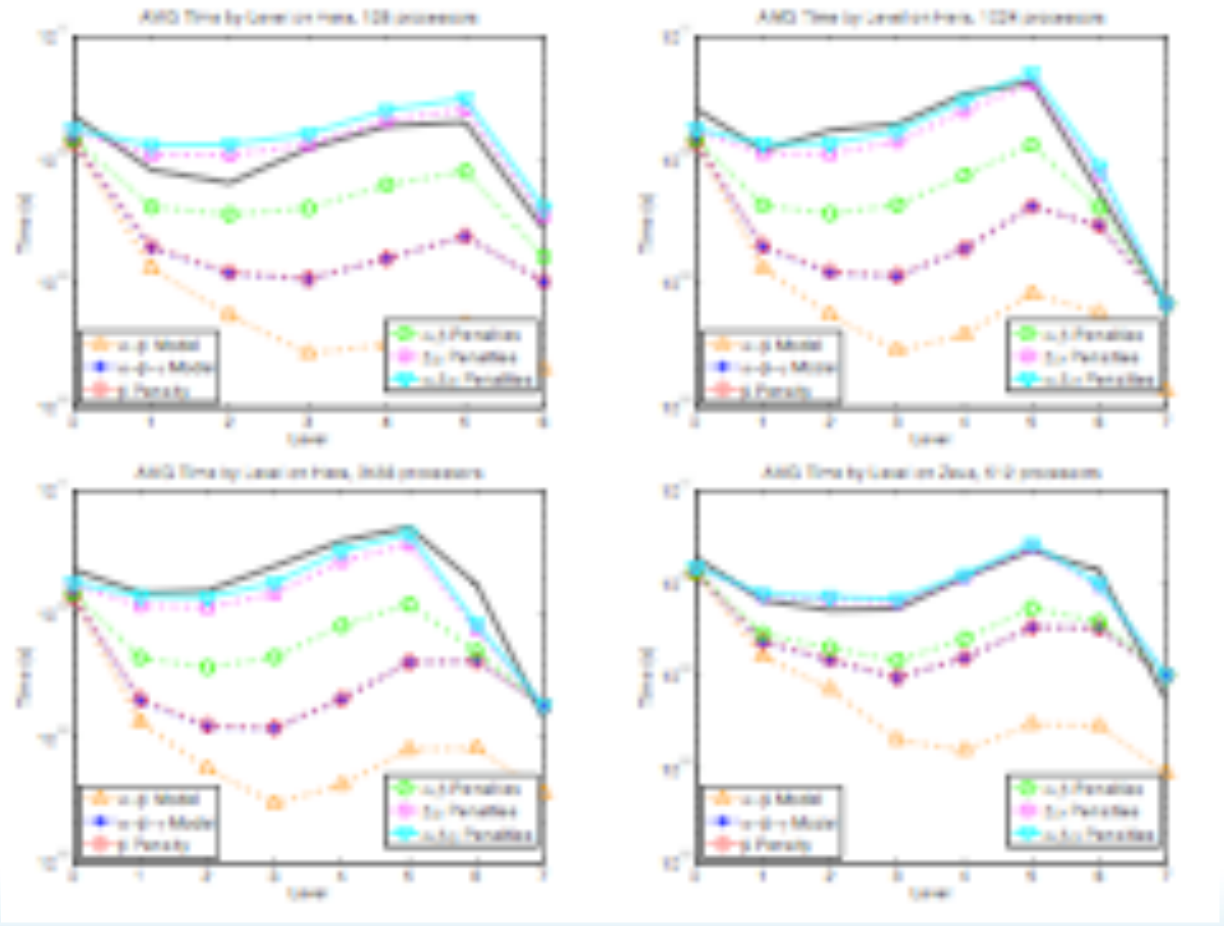
Thanks to Dahai Guo

Performance Modeling

- Even “back of the envelope” performance modeling can provide valuable insight
- Handle uncertainties by considering upper and lower bounds and/or considering different effects
- Performance modeling a key part of Blue Waters project, with efforts exploring different techniques
- Can be applied to complex codes….

AMG Performance Model

- Includes contention, bandwidth, multicore penalties
- 82% accuracy on Hera, 98% on Zeus
- Gahvari, Baker, Schulz, Yang, Jordan, Gropp



Implications of AMG Performance Model

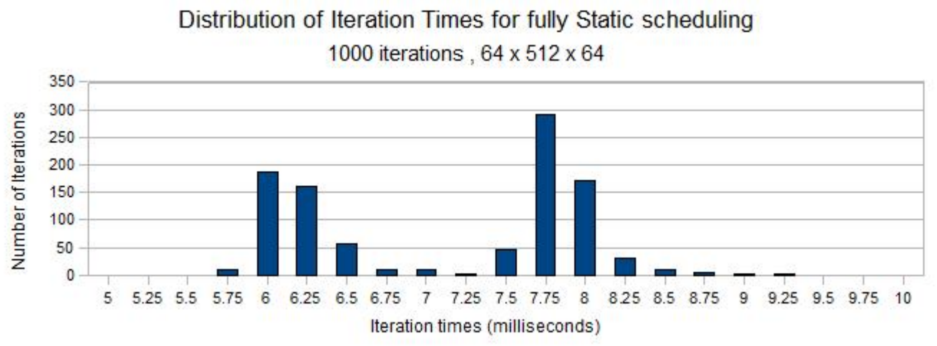
- Performance model identifies performance problems:
  - Bandwidth and contention in different systems; helps identify hardware features needed to get good performance on this application
- Suggests algorithmic changes to improve performance
  - Arithmetic rather than multiplicative versions (slightly poorer convergence rate but much more concurrency)
  - Communication time minimizing algorithms instead of communication volume minimizing

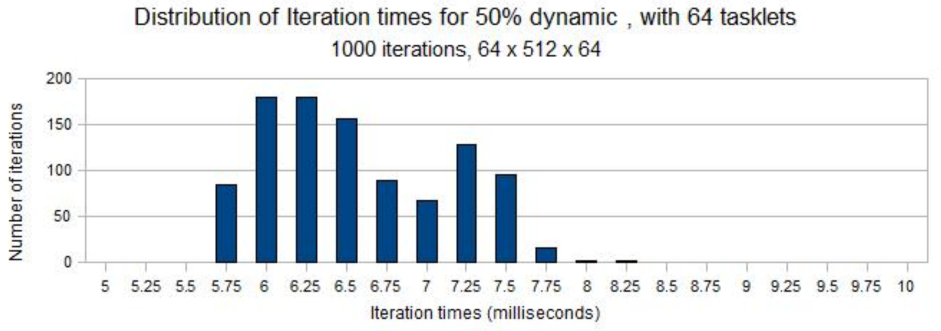
New Wrinkle – Avoiding Jitter

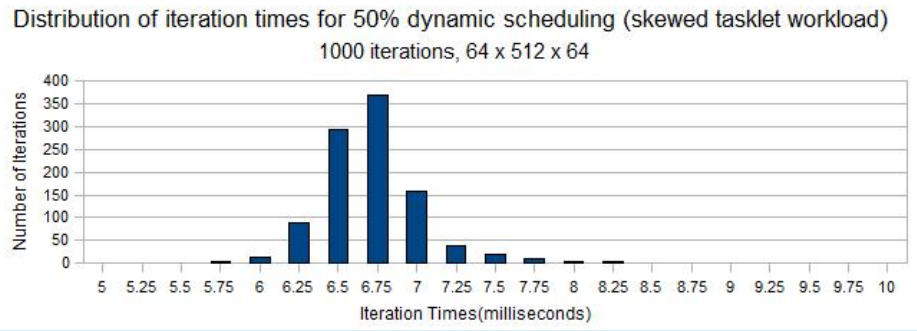
- Jitter here means the variation in time measured when running identical computations
  - Caused by other computations, e.g., an OS interrupt to handle a network event or runtime library servicing a communication or I/O request
- This problem is in some ways *less* serious on HPC platform, as the OS and runtime services are tuned to minimize impact
  - However, cannot be eliminated entirely

Sharing an SMP

- Having many cores available makes everyone think that they can use them to solve other problems ("no one would use all of them all of the time")
- However, compute-bound scientific calculations are often *written* as if all compute resources are owned by the application
- Such *static* scheduling leads to performance loss
- Pure dynamic scheduling adds overhead, but is better
- Careful mixed strategies are even better – OpenMP could do this for you (but doesn't yet)
- Thanks to Vivek Kale







Expressing Parallelism

- Programming Model Libraries
  - OpenMP; threads
  - MPI
  - (Open)SHMEM, GA
- Parallel Programming Languages
  - UPC, CAF in Fortran 2008
  - HPCS (Chapel, X10, Fortress) and successors
- Hybrid Models
  - MPI + Threads; MPI + OpenMP; MPI + UPC; …
- Libraries/Frameworks
  - Math libraries
  - I/O libraries
  - Parallel programming frameworks (e.g., Charm++, PETSc)
- Solving the node performance problem
  - Using local extensions/annotations and source-to-source transformations
  - Even simple tools can help; currently integrating into Eclipse framework
  - Proposed in DOE SciDAC program

OpenMP

- OpenMP is a set of compiler directives (in comments, like HPF) and library calls
- The comments direct the execution of loops in parallel in a convenient way.
- Data placement is not controlled, so performance can be hard to get
- Essentially a high-level model for using threads
  - Most common threads model for HPC
  - Has limitations (hard to express some operations; performance features deliberately hidden in misguided effort to be more productive for the programmer)
- IWOMP'11 www.ncsa.illinois.edu/Conferences/IWOMP11/

MPI

- MPI (Message Passing Interface) is the dominant model for HPC
  - An ad hoc standard, defined by the MPI Forum
  - MPI-1 defined two-sided message passing, along with a rich set of collective communication and computation
  - MPI-2 added one-sided , parallel I/O, dynamic process creation. MPI-2.2 is the current version
  - MPI-3 currently being written. To date includes nonblocking collectives; active work on improved one-sided, support for hybrid programming; tools interfaces
- Successful because complete, performance-focused, permits use of special hardware but can be efficiently implemented on everything

New Features in MPI-3

- Note: All these are still under discussion in the Forum and not final.
- Support for hybrid programming
  - Extend MPI to allow multiple communication endpoints per process
  - Helper threads: application sharing threads with the implementation
  - In general, support for "MPI + X" from the MPI side without having to specify what "X" is. This has worked well with OpenMP.
- Improved Remote Memory Access (one-sided) operations
  - Fix the limitations of MPI-2 RMA
  - New compare-and-swap, fetch-and-operation functions
  - Collective window memory allocation; dynamically attach memory to a window
  - Query function to determine whether system is cache coherent (for reduced synchronization requirement)
  - Others…

New Features in MPI-3 (cont.)

- New collective operations
  - Nonblocking collectives (MPI_Ibcast, MPI_Ireduce, etc)
  - Sparse, neighborhood collectives being defined as alternatives to irregular collectives that take vector arguments
- Fault tolerance
  - Detecting when a process has failed; agreeing that a process has failed
  - Rebuilding communicator when a process fails or allowing it to continue in a degraded state
  - Timeouts for dynamic processes (connect-accept)
  - Piggybacking messages to enable application-level fault tolerance
  - Others

New Features in MPI-3 (cont.)

- Fortran 2008 bindings
  - Full and better quality argument checking with individual handles
  - Support for choice arguments, similar to (void *) in C
  - Passing array subsections to nonblocking functions
  - Many other issues addressed
- Better support for Tools
  - MPIT performance interface to query performance information internal to an implementation
  - Standardizing an interface for parallel debuggers
- Topology functions
  - API changed to permit more scalable implementation

The PGAS Languages

- PGAS (Partitioned Global Address Space) languages attempt to combine the convenience of the global view of data with awareness of data locality, for performance
  - Co-Array Fortran (CAF), an extension to Fortran 90
  - UPC (Unified Parallel C), an extension to C
  - Chapel, one of the HPCS languages, extends the PGAS model

Hybrid Programming Models

- No one programming model is best for all parts of most applications
- Combining programming models provides a powerful set of tools
  - Can give very good results
  - But relies on a clean and efficient interface between programming models – this is often missing
- On Blue Waters, MPI, UPC, CAF, and others will be interoperable
  - Can build library routines/components in most appropriate model
  - Link application together
  - Work still needs to be done to understand how best to coordinate the models
    - On BW, all models make use of a single lower level, simplifying that coordination. However, threads and internode support not unified
  - Algorithms and data structures must also be changed to fully exploit hybrid programming models

Improving Achieved Node Performance

- It remains the case that most compilers cannot compete with hand-tuned or autotuned code on simple code
  - Just look at dense matrix-matrix multiplication or matrix transpose
  - Try it yourself!
    - Matrix multiply on my laptop:
    - N=100 (in cache): 1818 MF (1.1ms)
    - N=1000 (not): 335 MF (6s)

How Do We Change This?

- Test compiler against “equivalent” code (e.g., best hand-tuned or autotuned code that performs the same computation, under some interpretation or “same”)
  - In a perfect world, the compiler would provide the same, excellent performance for all equivalent versions
- As part of the Blue Waters project, Padua, Garzaran, Maleki are developing a test suite that evaluates how the compiler does with such equivalent code
  - Working with vendors to improve the compiler
  - Identify necessary transformations
  - Identify opportunities for better interaction with the programmer to facilitate manual intervention.
  - Main focus has been on code generation for vector extensions
  - Result is a compiler whose realized performance is less sensitive to different expression of code and therefore closer to that of the best hand-tuned code.
  - Just by improving automatic vectorization, loop speedups of more than 5 have been observed on the Power 7.
- But this is a long-term project
  - What can we do in the meantime?

Give "Better" Code to the Compiler

- Augmenting current programming models and languages to exploit advanced techniques for performance optimization (i.e., *autotuning*)
- Not a new idea, and some tools already do this.
- But how can these approaches become part of the mainstream development?

How Can Autotuning Tools Fit Into Application Development?

- In the short run, just need effective mechanisms to replace user code with tuned code
  - Manual extraction of code, specification of specific collections of code transformations
- But this produces at least two versions of the code (tuned (for a particular architecture and problem choice) and untuned). And there are other issues.
- What does an application want (what is the Dream)?

Application Needs Include

- Code must be portable
- Code must be persistent
- Code must permit (and encourage) experimentation
- Code must be maintainable
- Code must be correct
- Code must be faster

Implications of These Requirements

- Portable - augment existing language. Either use pragmas/comments or extremely portable precompiler
  - Best if the tool that performs all of these steps looks like just like the compiler, for integration with build process
- Persistent
  - Keep original and transformed code around
- Maintainable
  - Let use work with original code *and* ensure changes automatically update tuned code
- Correct
  - Do whatever the app developer needs to believe that the tuned code is correct
    - In the end, this will require running some comparison tests
- Faster
  - Must be able to interchange tuning tools - pick the best tool for *each* part of the code
  - No captive interfaces
  - Extensibility - a clean way to add new tools, transformations, properties, …

Application-Relevant Abstractions

- Language for interfacing with autotuning must convey concepts that are meaningful to the application programmer
- Wrong: unroll by 5
    - Though could be ok for performance expert, and some compilers already provide pragmas for specific transformations
- Right (maybe): Performance precious, typical loop count between 100 and 10000, even, not power of 2
- We need work at developing higher-level, performance-oriented languages or language extensions
- This work has been proposed in the recent DOE SciDAC call

What's Different at Petascale

- Performance Focus
  - Only a little – basically, the resource is expensive, so a premium placed on making good use of resource
  - Quite a bit – node is more complex, has more features that must be exploited
- Scalability
  - Solutions that work at 100-1000 way often inefficient at 100,000-way
  - Some algorithms scale well
    - Explicit time marching in 3D
  - Some don't
    - Direct implicit methods
  - Some scale well for a while
    - FFTs (communication volume in Alltoall)
  - Load balance, latency are critical issues
- Fault Tolerance becoming important
  - Now: reduce time spent in checkpoints
  - Soon: Lightweight recovery from transient errors

Recommendations

- Don’t do it yourself
  - Use frameworks and libraries where possible
  - Exploit principles used in those libraries if you need to write your own
- Know your application
  - Have a (even very simple) model of application performance
- Upgrade existing programs
  - Much can be done by updating/replacing core parts of the application
  - But must be guided by performance understanding – don’t “upgrade” the wrong parts!
- Embrace multicore
  - “MPI everywhere” not a solution
- Start over (at least for parts)
  - Real Petascale may require new algorithms and even mathematical models

Summary

- Many things are the same
  - Programming models, performance issues
- But balance is different
  - Small effects can dominate at scale
  - Greater attention must be paid to scaling, overheads, jitter
- Tools available to help
  - Scalable libraries and frameworks; performance analysis