Questions for Discussion (1)

- (3,3) Scalability (Ex: MHD)
  - Target platforms (petascale, multicore, clusters, …) (beyond 1000, 10K procs, plan for more)
  - Memory scalability, not just performance
    - Important on Petascale
  - Fraction of peak
  - Memory hierarchies / Out-of-core
  - Hierarchical machines -> hierarchical algs & SW
    (PETSC scaling on Jaguar, multicore)
- Standards to simplify…
  - Interfaces
  - Mixed shared / distributed memory

Questions for Discussion (2)

- What do users want from libraries that they don't have now? *(EX: MHD, fusion transport)*
  - Functionality
    - Operations (many small matrices)
    - Types/precisions/data layouts/ (quad for fusion transport)
    - New algorithms / helping users with algorithm choice
      - Automatic choice vs consulting vs education
    - New preconditioners, substitute user's own
    - Support for verification (condition estimates, error bounds, whenever possible, how to express error bars (on error bars…))

Questions for Discussion (3)

- What do users want from libraries that they don't have now? *(EX: MHD)*
  - Ease of use
    - (1,1) Portability, ubiquity (from development to production platforms)
    - Interoperability
      - Mixing MPI / Shared memory
      - Need to mix libraries, legacy code, new code in new languages
    - Reproducibility
    - Maintainability
    - (2,?) Instalability (easy build system, get right versions, impact on tuning)
    - Languages (native vs wrappers, F77/F90, C, Python)
    - Fault tolerance (~1 day enough?) (need user survey from NERSC, ORNL,…)
    - Memory models (Distributed, shared, PGAS)
    - Productivity:
      - Easy to use, if slower version, for development, plus easy path to substitute high performance versions
      - Python to prototype
    - Debug support
      - Automatically capturing test cases that fail (correctness or performance)

Questions for Discussion (4)

- Role of Automatic code generation and tuning?
  - When is it worth starting over to write a library generator rather than a library?
    - Dealing with hierarchical machines
  - What notation/language/annotations do we use to express and explore the tuning space?
    - What do we do, what do we leave to the compiler community? To the vendor?
  - Maintainability
    - Invest now for longer term reduction in costs/effort
  - Debuggability
    - How to debug if generated code is unreadable?
    - What is right level of abstraction, below which readability doesn't matter
      - The higher the better
    - Debuggers need to deal with mixed languages (they do now)
    - Role of assertions
  - Adapting to new architectures
    - Multicore, GPU (new memory bottlenecks), FPGA
    - What are tech trends that we have to live with?
    - New algorithms (bisection, any others?)
  - How much are users willing to accommodate runtime tuning in their applications?
    - Good idea if many simillar problems solved
    - User annotates to help collection of workload information (including phases of workload)
  - Integrate use of performance monitors to identify bottlenecks, help tuning
    - Capture test cases automatically
  - Sketching to generate correct optimized code from simple standard implementation
    - Need to be confident of correctness
    - "Torture test" code still needed
  - Division of labor between compiler / library team

Questions for Discussion (5)

- Role of vendors / SW companies
  - What do they build, what do we build?
  - What do they support us to build?
  - Multicore as opportunity to fund building some kernels
  - Open source and/or proprietary
    - Licensing (LGPL vs mBSD)
- Tools for future
  - Scalability testbed (eg RAMP)
  - Reproducibility (need vendor/OS support!)

Conclusions (for DOE)

- You should invest in…
  - Meet user goals
    - Scalability, even if code mods necessary
    - Incremental approach, with feedback, preserve ubiquity
  - Do this by
    - Automation…
      - Kernels, based on past success
        - » Workshop with hands-on user code to tune
      - Whole scale generation
      - Will ultimately lower maintenance costs
    - Tools to simplify rough performance modeling (2x good enough)
      - Preparation for Petascale
      - Libraries should come with performance models
      - Integrate into tools like TAU, IMP, …
    - Success metric: size of code base for multiple platforms, fraction of peak, other performance metrics vs older hand-written code
  - Code maintenance is expense, meanwhile keep funded
    - Automation of
      - Configuration, testing across environments, coverage
      - Reusable for application codes
    - Success Metric: fewer FTEs, fewer bug reports
  - Collect test cases (a la sparse matrix collections) for performance tuning, kernels and full apps
    - Need computer resources for testing, not just science (pre – INCITE)
  - Workshops with application / library teams
    - Tuning, performance modeling