Questions for Discussion (1)

- What do users want from libraries that they don't have now?
  - Functionality
    - Operations
    - Types/precisions/data layouts/
    - New algorithms / helping users with algorithm choice
      - Automatic choice vs consulting vs education
  - Ease of use
    - Portability
    - Interoperability
      - Mixing MPI / Shared memory
    - Reproducibility
    - Maintainability
      - Spend 50% time helping users. Automation will not help.
    - Installability
    - Languages (native vs wrappers)
    - Fault tolerance
    - Memory models (Distributed, shared, PGAS)
  - Scalability
    - Target platforms (petascale, multicore, clusters, …)
    - Fraction of peak
    - Memory hierarchies / Out-of-core
    - Hierarchical machines -> hierarchical algs & SW
  - Standards to simplify…
    - Interfaces
    - Mixed shared / distributed memory

Questions for Discussion (2)

- Role of Automatic code generation and tuning?
  - When is it worth starting over to write a library generator rather than a library?
    - Dealing with hiearchical machines
  - Maintainability
    - Invest now for longer term reduction in costs/effort
  - Adapting to new architectures
  - How much are users willing to accommodate runtime tuning in their applications?

Questions for Discussion (3)

- 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

Maintainability

- Hong:
  - 50% time helping users.
  - Automation will not help.
  - 3 people continuous for PETSc.
  - Mike H: 3 people for Trilinos.
  - Documentation alone does not eliminate.
  - One-to-one is very important.
  - Users are testers. Provide ideas for new development.

Maintainability, 2

- Marc:
  - Tutorial for users starting from the class of problem they want to solve.
  - Database of what is available to solve my problem.
- Jack: Coordination of the libraries: DOE, Vendors.
  - How the libraries install, work together.
  - Common look & feel, common accessibility.

Maintainability,3

- List of libraries minimally needed on a CSE system.
  - Include public libraries and vendor libraries.
  - Guidance on the choice and use.
- Coordination of communication:
  - Release announcements.
  - Netlib forum for announcements.
  - Single meta-site for users of CSE libraries.
    - BLOG, Wiki, interactive environment, RSS feed for announcements.
    - Archive of discussions.

Maintenance, 4

- Model of support is broken.
  - Mature, used but not actively developed, software is not well supported.
  - DOE has large collection of very valuable software.
  - Stewardship: little is done.
  - Should be an incentive to continue development of successful SW.
  - Currently penalized, since new development is given priority.

Coordination

- Coordination of communication:
  - Is already good, and improving, can do more.
  - Release announcements.
  - Netlib forum for announcements.
  - Single meta-site for users of CSE libraries.
    - BLOG, Wiki, interactive environment, RSS feed for announcements.
    - Archive of discussions.
- Workshops, events.
  - ACTS Toolkit workshop:
    - but more accessible.
    - Bigger event.
  - Coordinated slide show at SCXY.
- Ron:
  - Coordinated distribution of CSE libraries:
    - Single distribution. Reduce incompatibility problems.
    - E.g, Linux distribution approach.

Jack's 4 challenges

- Manycore: no contention.
- Autotuning: no contention.
  - Addressing several axes of performance:
    - Speed, memory use, accuracy, etc.
    - Saving power, reduce clock speed dynamically.
- Fault-tolerance (at algorithm level).
- Use of mixed precision:
  - For performance & accuracy.
  - For memory use & and power consumption.

System Interrogation

- Information:
  - Memory available.
  - CPU features: FP units, L/S overlap
  - $ info: size, hierarchy, r/w policies.
  - DGEMM peak.
  - More.
- PAPI-like approach for uniformity.

What Apps need

- Serguei:
  - Standard CSE software environment:
    - Autotools, BLAS, LAPACK, etc.
    - Fortran compiler.
    - Minimal set: RedHat package set.
    - Would enable binary distribution.
  - Installability
- Windows install tool.
- Binary distribution.

Matlab-like APIs

- Needed for Petascale?
- How seriously should we think about Matlab (Star-P, Python, Octave) as the API? YES!
- Productivity issue.
- Used natively or to generate code, or both?

Apps needs

- Tools:
  - Are our internal tools (autotuning, utilities) useful to you?
- Debugging, optimized (speed, memory) version of code.
- Reproducibility of results option:
  - Debug mode.
  - MPI_AllReduce differences.

Apps needs

- Rich:
  - Global sparse triangular solve is present bottleneck.
  - Can we develop an alternative at any level:
    - Better implementation.
    - Brand new algorithmic approach.
- Marc: Standard benchmark targets for some critical functionalities:
  - Global sparse triangular solve.
  - SpMV for several app areas.
  - Bakeoffs?
- Improved feedback loop from users:
  - Usage, problems.
  - Formal observation events of usage.
- Julien:
  - Good software engineering practices need to be transmitted to apps developers.
  - From library developers to apps developers: good design, best practices, etc.

Transition to Manycore

- Libraries migrate first.
  - Need a standard mechanism to go from flat MPI to MPI+shared, dynamically.
  - App will be running MPI-only.
- Translation tools for app:
  - Help migration.
  - Can it be transparent to the app?

Manycore concerns

- HW model is still vague:
  - Shared memory, local memory, cache coherent?
- SW model not clear.
- Parallel changes ubiquitous:
  - Transition from serial to MPI: MPI forced app framework changes, but left vast majority of complex physics code unchanged.
  - Vectorization: Happened automatically.
  - Manycore parallel will not be automatic (?).
  - Transition from MPI-only to MPI+manycore: Changes will be more disruptive, pervasive.

Manycore concerns

- Large-scale regeneration of libraries is easy to justify:
  - impacts thousands of users
  - only so many libs.
  - Small relative total cost.
- Similar rewrite of apps less broad impact:
  - may impact fewer users,
  - 100s or 1000s of apps.
  - Large total cost.
  - Need tools to reduce this cost.
- Typical programmer in MPI code does not need expert knowledge of MPI.
- Can we abstract the parallelism of manycore so the average programmer does not need to think in parallel?

Autotuning

- Need both static and dynamic tuning.
  - Need mechanism for informing tuning: e.g., number of iterations. See Zoltan.
- Language support (e.g., C++) helpful for:
- Polymorphism.
- Code generation (esp for fine-grain).

MPI needs

- Better support for overlapping comm & and comp.
- Becomes more important for manycore because of bandwidth issues.
- Asynch doesn't work all the time.
- Even parallel language extensions (CAF, UPC) don't give user control over process for most efficient execution.

Memory Requirements

- Memory size is scalability limiting factor.
  - Max/node is the issue.
  - Doubling the number of nodes for a fixed size problem should halve the node memory use (ideally).
- Out of core is an acceptable solution?
  - Is it possible on a petascale system?
  - Presumes a collection of local disks.
- New algorithms should have optimal memory usage (scalable use).
- Can data compression be used?
  - Provided easily to users? MPI tools?
  - Both lossy and non-lossy. User tunable.

Complete app rewrite? E.g. In Chapel/Fortress

- Ron:
  - Small codes:
    - Common in some areas:
      - Dynamics (chem), 100s-1000s LOC.
    - Possible. Weeks to months to rewrite.
  - Large codes:
    - Gaussian comps.
    - O(100K-1M) LOC. (GAUSSIAN - O(2M) LOC)
    - NWCHEM - O(1M) LOC. Requires 50 man-years to rewrite.
- Rich:
  - Too large, too costly to verify correctness.
- Serguei:
  - Just a few small codes.
  - Most important codes too expensive.

Debugging/profiling parallel codes

- Still really hard.
- Especially large-PE-count-only failures.
  - Runs on 10s or 100s of Pes, not on 1000s or more.
- Profiling:
  - Performance.
  - Memory use: Sampling capabilities.
    - Esp. non-virtual memory machines.