An Abstract Node API for Heterogeneous and Multi-core Computing

Christopher G. Baker
Michael A. Heroux
Sandia National Laboratories

LACCS 2008
DMP vs. SMP

- Parallel computing has targeted two dominant architectures over the past decades.

- Highly scalable distributed systems:
  - programmed as a flat network of serial nodes
  - employs message passing interface, typically MPI

- Moderately scalable shared memory systems:
  - programmed indirectly using, e.g., OpenMP or directly via some threading API (e.g., Pthreads)

- The latter approach cannot be applied to systems of the former type.

- The former (MPI-based) approach can be used on systems of the latter type.
MPI-Only Programming Model

- Dominant approach: a collection of nodes communicate via message passing API such as MPI.
- In the presence of SMP nodes, possible approaches are:
  - MPI under MPI
  - employ hybrid MPI+threads approach
  - maintain the “flat” MPI-Only model
- Flat MPI: k cores each on m nodes $\rightarrow O(k \times m)$ MPI processes
- “SMP-aware” MPI implementations allowed flat MPI approach to maintain dominance
  - shared memory copies for local communication
  - single copy of application per node reduces overhead
- Full performance benefit may not be fully realizable.
Tramonto Clovertown Results

- **Super-linear speedup (Setup phase)**
- **Sub-linear speedup (Solve phase)**

- **Setup (The application code itself):** Excellent MPI-only.
- **Solve (libraries):** Much poorer. Inherent in algorithms.
Tramonto Niagara2 Results

Super-linear/linear speedup (Setup phase)

Linear/sublinear speedup (Solve phase)

Tramonto Niagara2 Timings

Setup Time

Solve Time

# MPI processes

Time (sec)
Addressing These and Other Issues

- Disappointing kernel performance is not due to poor implementation:
  - memory subsystem cannot fully exploit all cores on the node
  - solver algorithms may be handicapped by smaller domains
- General consensus is that the number of cores per node will continue to increase for a while.
  - These multicore architectures look like the SMP machines of yesterday.
  - However, now they are ubiquitous.
  - Furthermore, it seems necessary to exploit them due to slowing single-core performance gains.
  - Solution: Apply known SMP algorithms from the past decades of research.
Other Items On Our Wishlist

- Support for multi-precision:
  - Double-precision is not always questioned in scientific computing.
  - Single-prec. floating point arithmetic can be significantly faster.
  - Smaller word size puts less strain on taxed memory hierarchy.
  - Multi-precision algorithms allow combination of fast arithmetic and need for higher accuracy.

- Support for newer architectures:
  - FPGA, GPU, CBE, ???

- Can achieve these via a general purpose programming environment with runtime support for desired platforms.
  - e.g., Sequoia, RapidMind
  - Too much trouble for me.
  - Instead, narrow scope to our libraries (i.e., those pesky solvers).
Tpetra Abstract Interfaces

- We propose a set of abstract interfaces for achieving these goals in the Tpetra library of linear algebra primitives.
- Tpetra is a templated implementation of the Petra Object Model:
  - these classes provide data services for many other packages in the Trilinos project (e.g., linear solvers, eigensolvers, non-linear solvers, preconditioners)
  - successor to Trilinos’s Epetra package
- Tpetra centered around the following interfaces:
  - `Comm` for providing communication between nodes
  - `Map` for describing layout of distributed objects
  - `DistObject` for redistributing distributed objects
  - Linear algebra object interfaces (`Operator`, `Vector`)
Satisfying Our Goals: Templates

- How do we support multiple data types?
  - C++ templating of the scalar type and ordinals.
  - Not new, not difficult.
  - Compiler support is good enough now.

- This provides generic programming capability, independent of data types.

- Templating implements compile time polymorphism.

- Pro: No runtime penalty.

- Con: Potentially large compile-time penalty.
  - This is okay. Compiling is a good use of multicore! :) 
  - Techniques exist for alleviating this for common and user data types (explicit instantiation)
Example

Standard method prototype for apply matrix-vector multiply:
```cpp
template <typename OT, typename ST>
CrsMatrix::apply(const MultiVector<OT, ST> &x, MultiVector<OT, ST> &y)
```

Mixed precision method prototype (DP vectors, SP matrix):
```cpp
template <typename OT, typename ST>
CrsMatrix::apply(const MultiVector<OT, ScalarTraits<ST>::dp> &x, MultiVector<OT, ScalarTraits<ST>::dp> &y)
```

Exploits traits class for scalar types:
```cpp
typename ScalarTraits<ST>::dp; // double precision w.r.t. ST
typename ScalarTraits<ST>::hp; // half precision w.r.t. ST
ST ScalarTraits<ST>::one(); // multiplicative identity
```

Sample usage in a mixed precision algorithm:
```cpp
Tpetra::MultiVector<int, float> x, y;
Tpetra::CisMatrix<int, double> A;
A.apply(x, y); // SP matrix applied to DP multivector
```
C++ Templates

- Example was for float/double but works for:
  - `complex<float>` or `complex<double>`
  - Arbitrary precision (e.g., GMP, ARPREC)
  - The only requirement is a valid specialization of the traits class.
The Rest: C++ Virtual Functions

- How do we address our desire to support multiple implementations for these objects?
  - C++ virtual functions and inheritance.
- This provides runtime polymorphism.
- Use abstract base classes to encapsulate data and behavior.
- Specific concrete implementations of these interfaces provide adapters to target architectures.
- We will “abstract away” communication, data allocation/placement and computation.
Tpetra Communication Interface

- Teuchos::Comm is a pure virtual class:
  - Has no executable code, interfaces only.
  - Encapsulates behavior and attributes of the parallel machine.
  - Defines interfaces for basic comm. services between “nodes”, e.g.:
    - collective communications
    - gather/scatter capabilities
  - Allows multiple parallel machine implementations.
  - Generalizes Epetra_Comm.

- Implementation details of parallel machine confined to Comm subclasses.

- In particular, Tpetra (and rest of Trilinos) has no dependence on any particular API (e.g., MPI).
Comm Methods

- `getRank()`
- `getSize()`
- `barrier()`
- `broadcast<Packet>`(Packet *MyVals, int count, int Root)
- `gatherAll<Packet>`(Packet *MyVals, Packet *AllVals, int count)
- `reduceAll<Packet>`(ReductionOp op, int count, const Packet *local, Packet *global)
- `scan<Packet>`(ReductionOp op, int count, const Packet *send, Packet *scans)

Comm Implementations

- SerialComm simultaneous supports of serial and parallel coding.
- MpiComm is a thin wrapper around MPI communication routines.
- MpiSmpComm allows use of shared-memory nodes.
Abstract Node Class

- Trilinos/Kokkos: Trilinos compute node package.
- Abstraction definition in progress.
  - Node currently envisioned as an abstract factory class for computational objects.

Example:
```cpp
Kokkos::LocalCrsMatrix<int,double> lclA;
lclA = myNode.createCrsMatrix(...);
lclA.submitEntries(...); // fill the matrix
Kokkos::LocalMultiVector<int,double> lclX = myNode.createMV(...),
   lclY = myNode.createMV(...);
lclA.apply(lclX,lclY); // apply the matrix operator
```
Abstract Node Class (2)

- Node handles platform-specific details, such as:
  - how to allocate memory for the necessary data structures?
    - significant in the case of attached accelerators with distinct memory space.
  - How to perform the necessary computations?
    - Tpetra is responsible only for invoking global communication, via the abstract Comm class.
    - In addition to supporting multiple architectures, Tpetra/Kokkos becomes a test bench for research into primitives.

- These abstractions (*hopefully*) provide us with flexibility to tackle a number of platforms.

- Cons:
  - \( m \) kernels, \( p \) platforms \( \rightarrow m \times p \) implementations
  - Heros can’t improve code they don’t have access to.
Sample Code Comparison: MV::dot()

**MPI-only:**

```c
double dot(int len,
            double *x,
            double *y)
{
    double lcl = 0.0, gbl;
    for (int i=0; i<len; ++i)
        lcl += x[i]*y[i];
    MPI_ALLREDUCE(lcl, gbl,...);
    return gbl;
}
```

**Tpetra/Kokkos:**

```cpp
template <typename ST>
ST Tpetra::MultiVector<ST>::dot(
    Comm comm,
    Kokkos::LocalVector<ST> x,
    Kokkos::LocalVector<ST> y)
{
    Scalar lcl, gbl;
    lcl = x.dot(y);
    reduceAll<ST>(comm, SUM, lcl, gbl);
    return gbl;
}
```

- For appropriate choices of Node and Comm, both implementations are equivalent.
- Right hand example is limited only by the available implementations of these classes:
  - can determine whether library was compiled with support for GPU, MPI, etc.
  - can compose different nodes for heterogeneous
Example: MPI-Only $Ax = b$

- App passes matrix and vector values to library data classes.
- All ranks store $A$, $x$, $b$ data in locally-visible address space.
- Library solves $Ax=b$ using distributed memory algorithm, communicating via MPI.

Using: SerialNode, MpiComm
Example: MPI+Threads $Ax = b$

App passes matrix and vector values to library data classes

All ranks store $A$, $x$, $b$ data in memory visible to rank 0

Library solves $Ax=b$ using shared memory algorithms on the node

Using: SmpNode, SerialComm
Example: Multicore + GPU $Ax = b$

App passes matrix and vector values to library data classes.

Kokkos objects allocate data according to concrete implementation.

Library solves $Ax=b$ using appropriate kernels.

Using: SmpNode, CudaNode, SerialComm, CudaComm
Conclusion

- I wish I had some results.
- C++ polymorphism:
  - allows library user to run apps on a wide variety of platforms
  - allows/\textit{requires} library developer to isolate implementation from interface
  - gives hacker/hero a hook for experimentation/salvation
- Abstract Comm has been in use in Epetra for years; some use of abstract interfaces to hide implementation have already seen experimental use.
- Development of this API for Tpetra is currently in progress; expect limited release in March ‘09, general release in Sept ‘09.