

Trinos for Emerging Parallel Computing Systems

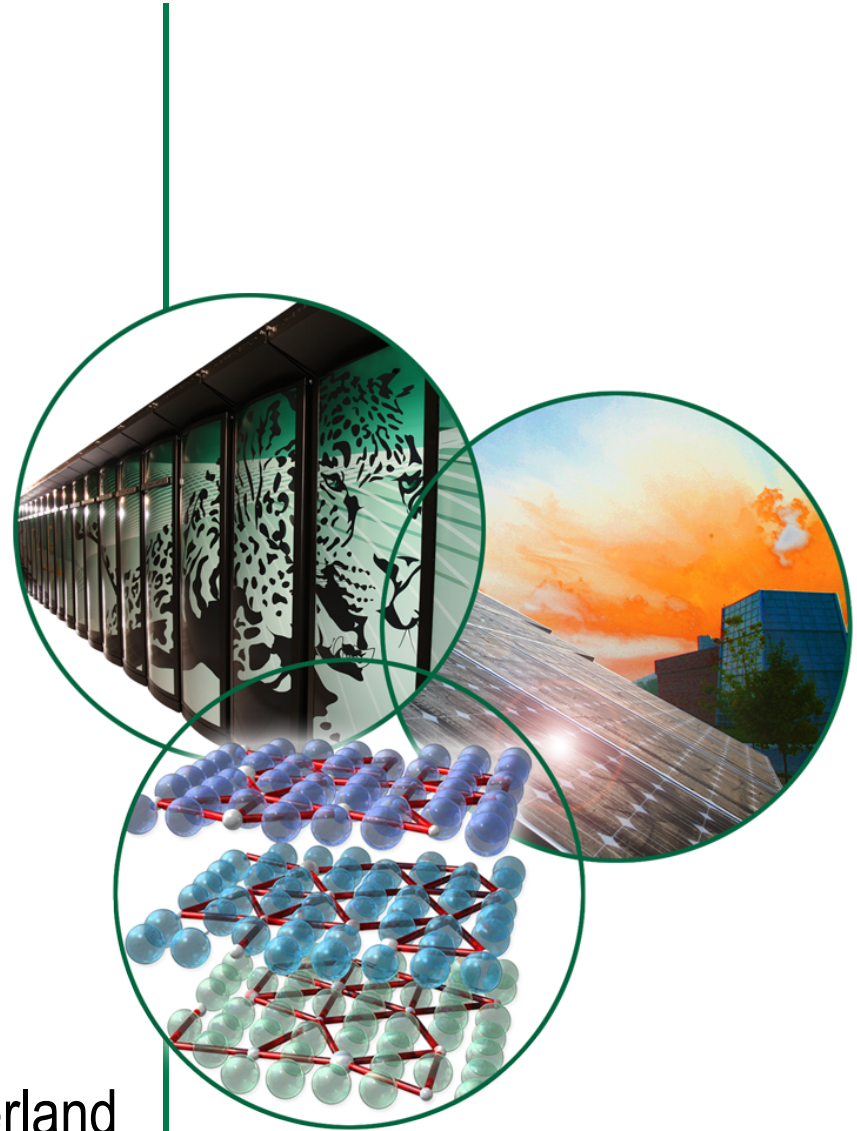
Christopher Baker

Computational Engineering and Energy Studies

Oak Ridge National Laboratory, USA

39th SPEEDUP Workshop on High-Performance Computing

September 6-7, 2010, ETH Zurich, Switzerland



My Background

- I'm a library developer in the Trilinos project.
 - I work on distributed primitives and solvers.
- Trilinos is an object-oriented software framework to enabled the solution of large-scale, complex multi-physics engineering and scientific problems.
 - Open source, implemented in object-oriented C++
 - Newer packages make increased use of generic programming
- Flexibility is key:
 - Previously (and still) necessary in order to maximize utility for **applications**.
 - Now it is necessary to maximize portability across **architectures**.



The Trilinos Software Stack

- Different Trilinos packages build on each other to create a stack providing the necessary capability:

Non-linear solver

- **NOX** non-linear solver
- **Anasazi** iterative eigenvalue solvers

Linear Solver/
Preconditioner

- **ML**, **Ifpack**, **Teko** preconditioners
- **Belos**, **AztecOO** linear solvers
- **Amesos** direct sparse solvers

Distributed linear algebra

- **Epetra** vectors, sparse matrices (double/int)
- **Tpetra** templated objects (S/LO/GO/Node)

Local linear algebra

- **BLAS** and **LAPACK** kernels
- **Kokkos** parallel node interface and kernels

Current Scientific Library Paradigm

- Library provides a specific capability.
 - A user can grab the data and expand the functionality.
- Such encapsulation of work is the M.O. for many libraries:
 - e.g., ARPACK, PETSc, Trilinos
- In an **MPI-only scenario**, expansion comes via domain-specific serial kernels coded by **domain specialists**.
- With a **single memory pool**, data is commonly shared between library and app.
 - The main difficulty here regards data “ownership”.
- With a **single target architecture**, compilation is simple.
- Still, library tuning is potentially intrusive and tedious.

Enter the Hybrid Parallel Environment

- Modern supercomputers potentially employ **exotic** hardware alongside the **mundane** multi-core CPU:
 - LANL RoadRunner (#3) utilizes Cell BE in addition to multi-core CPUs.
 - Chinese Nebulae (#2) utilizes NVIDIA Fermi GPUs.
 - OLCF-3 will utilize 1 GPU:1 CPU, across thousands of nodes.
 - The path to exascale will likely employ some type of accelerator.
- Ditch the assumptions of the previous slide/paradigm:
 1. shared-memory programming augments MPI-only
 2. accelerators with their own memory space complicate data passing; even data layout for NUMA archs. requires finesse
 3. heterogeneous execution environment requires compiling our kernels multiple times, for each hardware type
 4. tuning library is more important and intrusive than ever

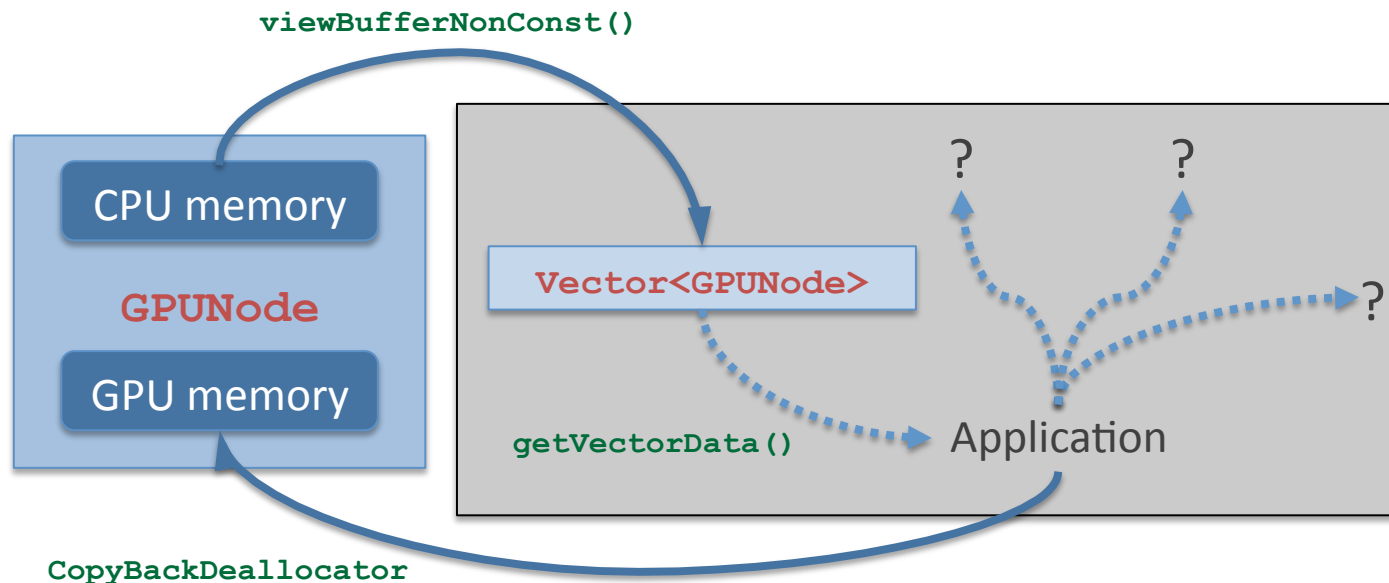
Expanding Library Functionality

1. Application expansion of code must either:
 - a) limit the app to MPI-only, *or*
 - b) introduce a serial bottleneck, *or*
 - c) match the shared memory methodology used by the node, *or*
 - d) be **somehow** handed off the to library for parallel execution

2. Data sharing between app and library must either:
 - a) **somehow** acknowledge distinct memory pools
 - data must be allocated on device and copied to host
 - modified data must be copied back to device at the appropriate time
 - b) be prohibited
 - if the user can't manipulate the data, then the library must do it for them **somehow**

Example Memory Usage Pattern

- The library components may be part of a deep software stack.
- **Problem:** explicit release of data views permeates the stack.
 - downside: now, these memory management objects permeate the stack
- **Problem:** it is not always obvious when to release the view.
- **Automatic memory management** trivially handles this case.



Heterogeneous Target Architecture

3. Heterogeneous execution environment requires either:

- a) writing the kernels for different platforms as needed
 - M kernels * N architectures == M*N implementations
- b) writing the code once and letting the compiler sort it out
 - very limited selection of solutions that are appropriate and capable
- c) writing cross-platform kernels that are **somehow** handed off to the platform for parallel execution

4. Tuning of the library by vendors/heroes requires either:

- a) writing brand new library classes for each architecture
- b) identifying the necessary functionality and leaving generic hooks for their provision

✓ **Common thread: could utilize some method for submitting **work/data** to to a shared-memory parallel node.**

Our Approaches to These Problems in Stage 2 Trilinos

- **Templated C++ code**
 - Templating data allows more efficient use of cache and bandwidth.
 - Templating data expands capability (e.g., integer limit, `complex`)
 - Template meta-programming allows some nice compile-time tricks.
- **Abstract inter-node communication**
 - Possibilities abound for use of shared memory libraries in MPI-only legacy applications.
- **Generic shared memory parallel node**
 - Our abstraction of a shared-memory parallel node.
 - Template meta-programming aiming for write-once, run-anywhere.
- **Plug-and-play structures and algorithms**
 - Expose the SMP node to apps; enable node-optimized kernels.

Case study: Arithmetic traits classes

- **Anasazi eigensolvers and Belos linear solvers code to an abstract interface for multivectors and operators.**
 - **MultiVecTraits<S, MV>** provides constructors and arithmetic for a generic multivector class MV over scalar field S
 - **OperatorTraits<S, MV, OP>** provides an interface for applying a linear operator of type OP to a multivector of type MV over a scalar field S
- **Both MV and OP are opaque; no low-level access (in general).**
- **Solvers and their utilities are templated on <S, MV, OP>:**
 - **LOBPCGSolver<S, MV, OP>**
 - **DGKSOrthoManager<S, MV, OP>**

Tpetra Package and Template Types

Standard

- **Scalar**: scalar field for a mathematical object
 - must support math and communicationdouble
- **LocalOrdinal**: index type for local elementsint
- **GlobalOrdinal**: index type for global elementsint
- **Node**: concrete node type for an MPI processSerialNode/
OpenMPNode
- **size_t**: indices into node memory
 - e.g., number local matrix entriesint
- **global_size_t**: indices into global memory
 - e.g., number of global matrix entries
 - 64-bit should be large enough for anyone, but still...int

Example: mixed-precision eigensolve

```
// helpful typedefs
typedef Kokkos::          OpenMPNode          Node;
typedef Tpetra::          Map<int,long,Node>   ilMap;
typedef Tpetra::CrsMatrix<float,int,long,Node> fMatrix;
typedef Tpetra::  Vector<double,int,long,Node> dVector;
typedef Tpetra::Operator<double,int,long,Node> dLinOp;
typedef Anasazi::BasicEigenproblem<double,dVector,dLinOp> dEproblem;
// Tpetra setup
RCP< Teuchos::Comm<int> > comm = ...;
long numGlobal = ...; // <= 2^63
RCP< const ilMap > map = Tpetra::createUniformContigMap<int,long,Node>
                          ( numGlobal, comm );
RCP< fMatrix > Af = Tpetra::createCrsMatrix<float>( map, nnz, StaticProfile );
// ... fill Af with float data
Af->fillComplete( OptimizeStorage );
RCP< dLinOp > Ad = Tpetra::createCrsMatrixMultiplyOp<double>( Af );
// setup Tifpack preconditioner (T10.8?)
RCP< dLinOp > P = Tifpack::createPreconditioner<double>( Af );
// setup Anasazi eigensolver
RCP< dEproblem > prob = rcp( new dEproblem( Ad ) );
prob.setHermitian( /* true or false */ );
prob.setPrec( P );
LOBPCGSolMgr<double,dVector,dLinOp> solver( prob );
solver.solve();
RCP< Vector<double,int> > x = prob->getSolution();
```

Generic Shared Memory Node

- Provides two main components:
 - **Generic memory model** addresses data issues
 - Allocation, deallocation and efficient access of memory
 - **compute buffer**: special memory used for parallel computation
 - Use of automatic memory management classes
 - **Generic compute model** addresses work issues
 - Description of kernels for parallel execution on a node
 - Provides skeletons for common parallel work constructs
 - Currently, we describe **parallel for loop** and **parallel reduce**
 - User-provided kernels provide the body of these skeletons
- Library developed around a compile-time polymorphic **Node** object.
- Supporting a platform requires only implementing a new node type.

Example Kernels: `axpy()` and `dot()`

```
template <class WDP>
void
Node::parallel_for(int beg, int end,
                  WDP workdata );
```

```
template <class WDP>
WDP::ReductionType
Node::parallel_reduce(int beg, int end,
                    WDP workdata );
```

```
template <class T>
struct AxyOp {
    const T * x;
    T * y;
    T alpha, beta;
    void execute(int i)
    { y[i] = alpha*x[i] + beta*y[i]; }
};
```

```
template <class T>
struct DotOp {
    typedef T ReductionType;
    const T * x, * y;
    T identity()      { return (T)0;      }
    T generate(int i) { return x[i]*y[i]; }
    T reduce(T x, T y) { return x + y;    }
};
```

```
AxyOp<double> op;
op.x = ...; op.alpha = ...;
op.y = ...; op.beta = ...;
node.parallel_for< AxyOp<double> >
    (0, length, op);
```

```
DotOp<float> op;
op.x = ...; op.y = ...;
float dot;
dot = node.parallel_reduce< DotOp<float> >
    (0, length, op);
```

Fusing Nodes and Kernels: How it all comes together

- API dev/vendor writes **SMP nodes**; user writes **serial kernels**.

- C++ template meta-programming fuses them (à la TBB, Thrust)

- `TBBNode::parallel_reduce< DotOp<double> >(0, vecLen, wdp);`
- `ThrustGPUNode::parallel_for< ComputeForces<3D,LJ> >(0, numParticles, wdp);`

- Nodes and kernels are fused at compile time:

- `OpenMPNode<AxyOp>` equivalent to hand-coded OpenMP axpy (YMMV)

- Certain classes expose this interface (T10.6):

- `Tpetra::execute< UserKernel<S> >(Tpetra::Vector<S> &x);`
- `S ans = Tpetra::reduce< FunkyNorm<S> >(Tpetra::Vector<S> &x);`

- Other classes expose more specialized interfaces (T10.6):

- `class Tpetra::CrsMatrix<Scalar, LO, GO, Node, LocalMatOps> {
 LocalMatOps<Scalar, LO, GO, Node> lclSparseMatVec_
};`

Deprecated/Modified Functionality Under the New Paradigm

- **Epetra allows fine control over/access to object memory:**
 - `Epetra_Vector v(View, my_vec_pointer);`
 - `assert(v.Values() == my_vec_pointer);`
- **User-allocated memory is not appropriate:**
 - GPU arch requires allocation in GPU memory pool.
 - Multi-core CPU may require special placement for performance.
- **Access to object memory is still possible, but requires additional consideration.**
 - In generic code, view may require copy from device to host.
 - For non-const view, this also incurs copy-back.
 - “Object modes” (View/Edit/Compute) to address this by allowing user to specify intention.

MultiVecTraits and OperatorTraits on Hybrid Platforms

- **Recall the MultiVecTraits class:**
 - Simple interface to vector class arithmetic implies easy port of arithmetic to new platforms
 - Opaque data storage means no conflicts moving to hybrid memory architecture.
- **The consequence is that neither Belos nor Anasazi required any changes for port to hybrid architectures.**
 - A few minor changes for efficiency purposes.
 - This result is mostly true for other iterative solvers packages due to the abstraction of linear operators.

How to Program Hybrid Clusters

- Programming for a single GPU is well studied.
- What about more than one GPU?
 - Distributed memory → distributed memory model
 - Significant code investment needed to handle communication.
 - Techniques exist for avoiding communication and load balancing.
 - **One MPI process per GPU.**
 - This currently comes at the cost of at least one CPU core per physical node.
 - Have to be even more careful with communication than before.
 - Even then, you may have to pay twice (PCIe + node interconnect).
 - Greater exposure of asynchronous kernels and memory transfer will increase the performance here.
 - Other MPI processes marshal other shared-memory nodes, such as multi-core CPUs.

Further Questions

- **What about reliability?**
 - What mechanisms are necessary to support goals of reliability?
 - How do these mechanisms affect library interfaces?
 - Specifically, if the library is responsible for reliability (or marshals it on behalf of the user), how might this limit what the app is allowed to do?
- **What is the proper balance of generic kernels and architecture specific kernels?**
 - Currently learning this as we go.
- **Trilinos effort currently focused on leveraging generic classes and abstract interfaces for flexible implementations and easy composition for larger problems.**

Conclusion

- Thanks to SPEEDUP for hosting this workshop and inviting a presentation on Trilinos.
- Please contact if you have any questions:

bakercg@ornl.gov

trilinos-users@software.sandia.gov

<http://trilinos.sandia.gov/>



- Any questions?

Shared Memory Results

- Tests of a simple iterations:
 - **power method**: one sparse mat-vec, three vector operations
 - **conjugate gradient**: one sparse mat-vec, five vector operations
- Matrix is a simple 3-point discrete Laplacian with 1M rows in compressed row (CRS) format
- Comparison of
 - Trivial serial node
 - Pthreads-based node
 - Thrust-based CUDA node
- Physical node includes:
 - one NVIDIA Tesla C1060
 - four 2.3 GHz AMD Quad-core CPUs

Node	PM (mflop/s)	CG (mflop/s)
SerialNode	101	330
TPINode(1)	116	375
TPINode(2)	229	735
TPINode(4)	453	1,477
TPINode(8)	618	2,020
TPINode(16)	667	2,203
ThrustGPUNode	2,584	8,178

Distributed Memory Results

- Tests of a simple iterations:
 - **power method**: one sparse mat-vec, two vector operations
 - **conjugate gradient**: one sparse mat-vec, five vector operations
- DNVS/x104 from UF Sparse Matrix Collection (100K rows, 9M entries)
- NCCS/ORNL **Lens** node includes:
 - one NVIDIA Tesla C1060
 - one NVIDIA 8800 GTX
 - Four AMD quad-core CPUs
- Results are **very tentative!**
 - suboptimal GPU traffic
 - bad format/kernel for GPU
 - bad data placement for threads

Node	PM (mflop/s)	CG (mflop/s)
Single thread	140	614
8800 GPU	1,172	1,222
Tesla GPU	1,475	1,531
Tesla + 8800	981	1,025
16 threads	816	1,376
1 node		
15 threads + Tesla	867	1,731
2 nodes		
15 threads + Tesla	1,677	2,102