Design Issues for Numerical Libraries on Scalable Multicore Architectures

Michael A. Heroux
Sandia National Laboratories

Sandia
National
Laboratories

Preliminaries

About MPI

- MPI will be the primary inter-node programming model.
- Very few people program in MPI: Abstractions.
- Right ingredients:
  - Portable, ubiquitous.
  - Forced alignment of work/data ownership and transfer.
- Matches architectures:
  - Interconnects of best commercial node parts.
- New languages:
  - Big fan of Co-Array Fortran (Have been for 15 years: F--).
  - Chapel looks good.
  - But tough uphill climb.

- Real question: How do we program the node?

Codes Discussed Here

- HPCCG:
  - “Closest thing to an unstructured FEM/FVM code in 500 semi-colons or fewer.”
- pHPCCG:
  - Compile-time parameterized FP (float, double, etc) and int (32, 64, etc).
- Trilinos/Epetra Benchmark Tests:
  - Trilinos Performance-determining kernels.
- phdMesh Vector Multi-update:
  - Basic kernel in explicit dynamics (generalized DGEMV).
- Tramonto:
  - Polymer test case.
- LAMMPS: Molecular Dynamics.
- Trilinos/Tpetra, Trilinos/Kokkos:
  - New multicore-aware packages.

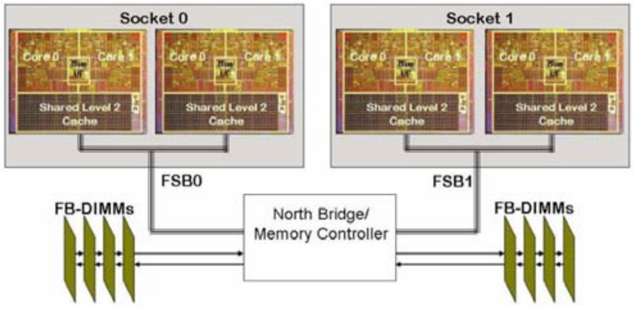
Trilinos Package Summary

http://trilinos.sandia.gov

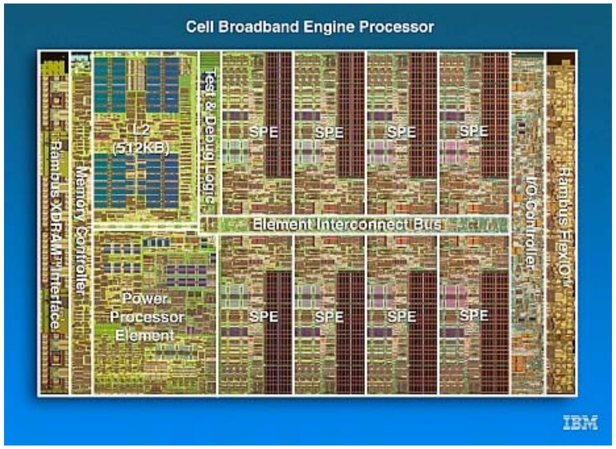
	Objective	Package(s)
Discretizations	Meshing & Spatial Discretizations	phdMesh, Intrepid, Pamgen, Sundance
	Time Integration	Rythmos
Methods	Automatic Differentiation	Sacado
	Mortar Methods	Moertel
Core	Linear algebra objects	Epetra, Jpetra, Tpetra
	Abstract interfaces	Thyra, Stratimikos, RTOp
	Load Balancing	Zoltan, Isorropia
	“Skins”	PyTrilinos, WebTrilinos, Star-P, ForTrilinos, CTrilinos
	C++ utilities, I/O, thread API	Teuchos, EpetraExt, Kokkos, Triutils, TPI
Solvers	Iterative (Krylov) linear solvers	AztecOO, Belos, Komplex
	Direct sparse linear solvers	Amesos
	Direct dense linear solvers	Epetra, Teuchos, Pliris
	Iterative eigenvalue solvers	Anasazi
	ILU-type preconditioners	AztecOO, IFPACK
	Multilevel preconditioners	ML, CLAPS
	Block preconditioners	Meros
	Nonlinear system solvers	NOX, LOCA
	Optimization (SAND)	MOOCHO, Aristos
	Stochastic PDEs	Stokhos

Node Classification

- Homogeneous multicore:
  - SMP on a chip.
  - NUMA nodes.
  - Varying memory architectures.



- Heterogeneous multicore:
  - Serial/Controller processor(s).
  - Team of identical, simpler compute processors.
  - Varying memory architectures.



Why Homogeneous vs. Heterogeneous?

- Homogeneous:
  - Out-of-the-box: Can attempt single-level MPI-only.
  - m nodes, n cores per node: p = m*n
  - mpirun -np p …
- Heterogeneous:
  - Must think of compute cores as “co-processors”.
  - mpirun -np m …
  - Something else on the node.
- Future:
  - Boundary may get fuzzy.
  - Heterogenous techniques can work well on homogeneous nodes.

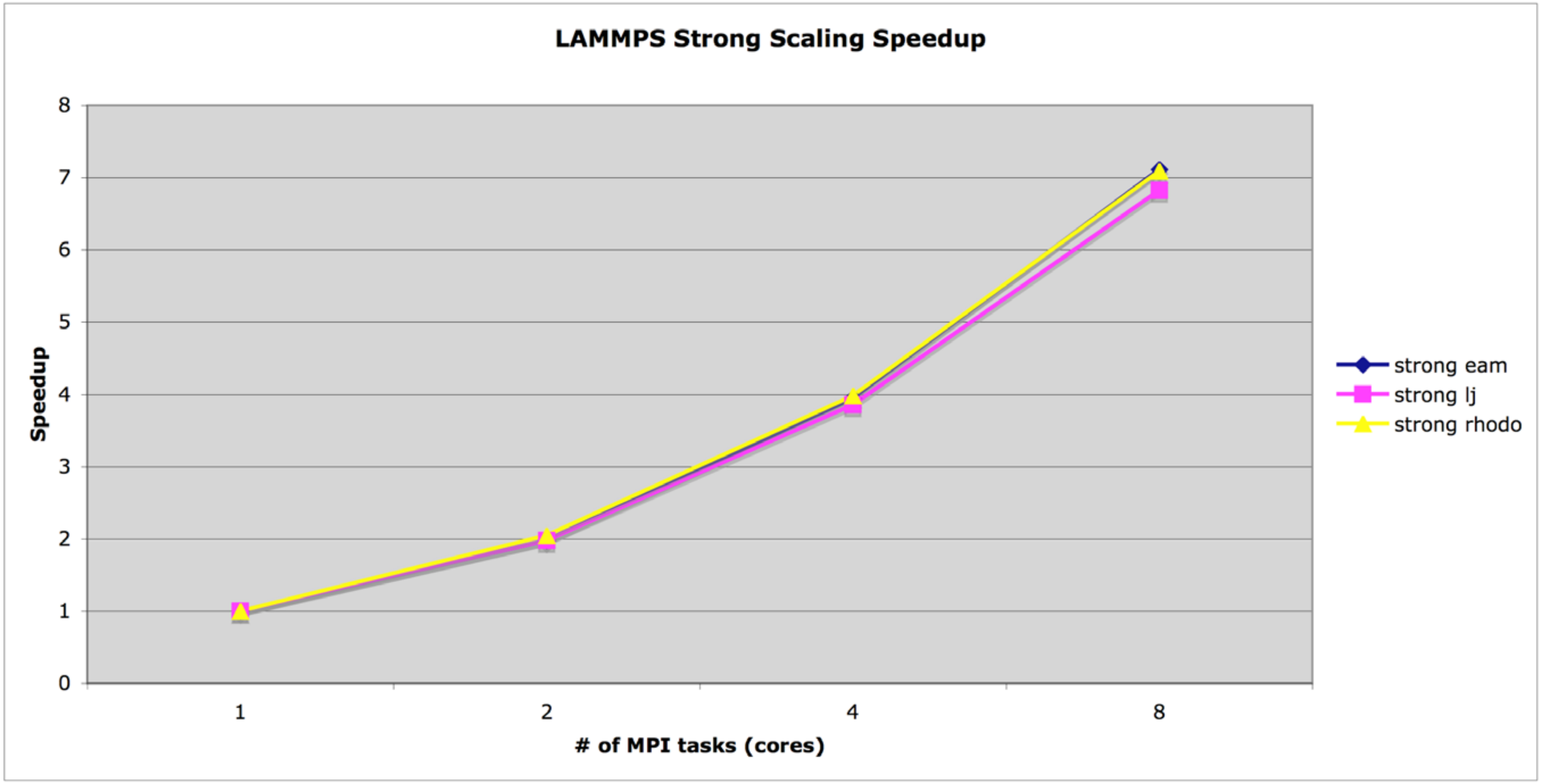
Homogeneous Multicore Issues

Single Core Performance: Still improving for some codes

- HPCCG microapp.
- Clock speeds stable: ~ 2GHz.
- FP-friendly computations stalled.
- Memory-intensive computations still improving.

Year	Processor	Clock (GHz)	Cores per socket	MFLOPS /sec
2003	AMD Athlon	1.9	1	178
2004	AMD Opteron	1.6	1	282
2005	Intel Pentium M	2.1	1	310
2006	AMD Opteron	2.2	2	359
2007	Intel Woodcrest	1.9	4	401
2007	AMD Opteron	2.1	4	476
2007	Intel Core Duo	2.3	2	508

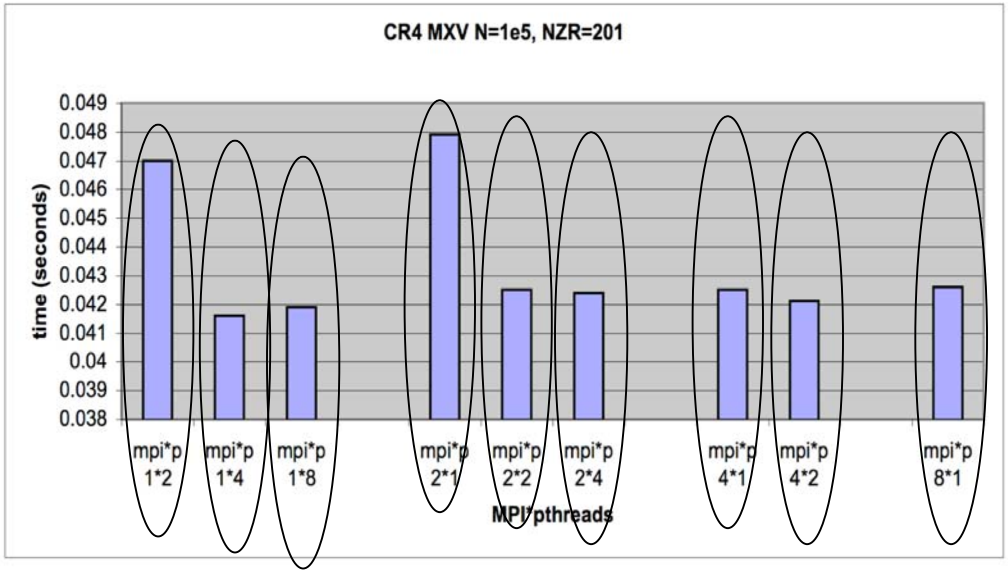
MPI-Only
(Intel Clovertown)



- The incumbent: Always present.
- Sometimes sufficient.

Programming Model Translation

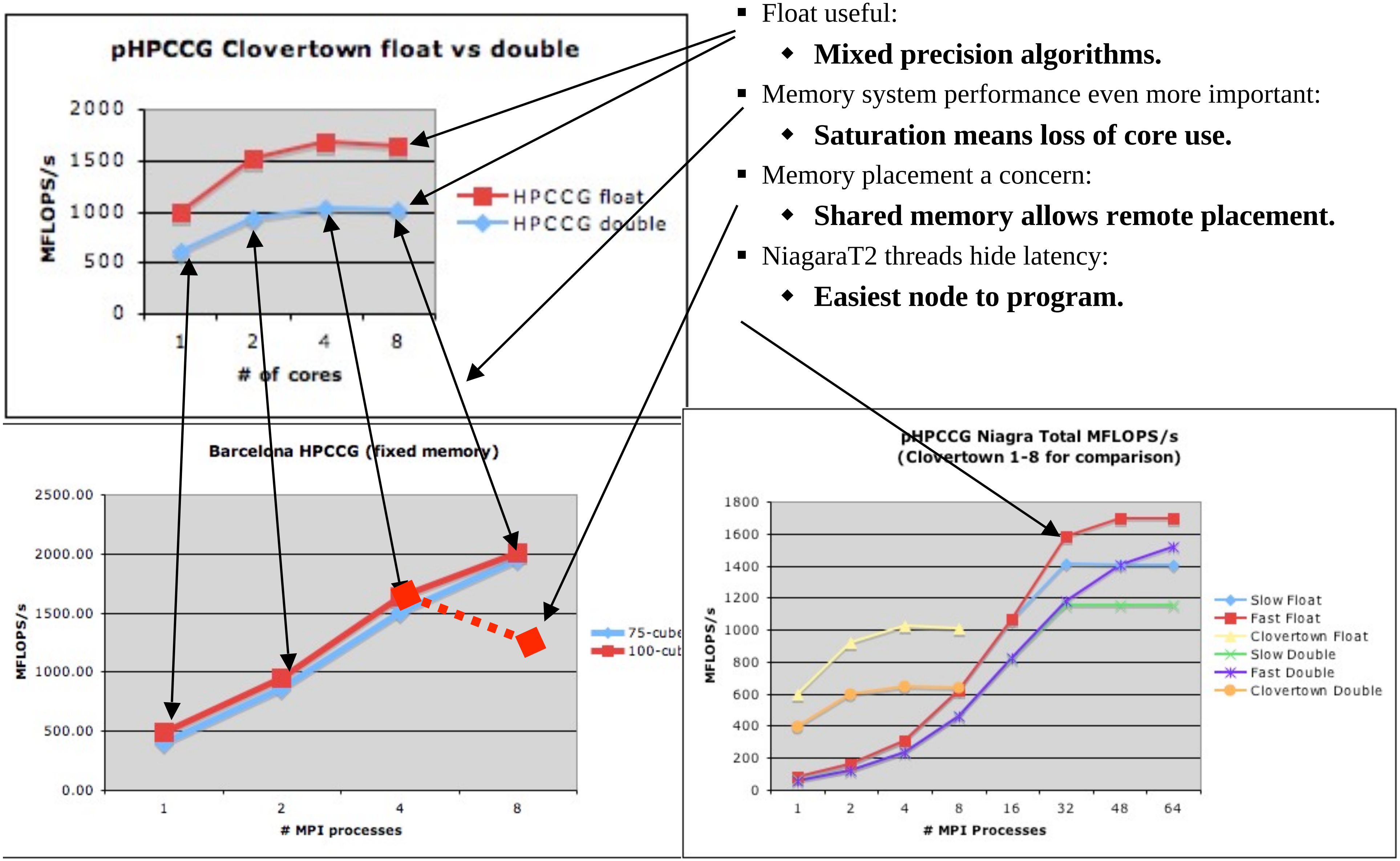
(courtesy H.C. Edwards)



- Been here before:
  - 12-15 years ago: SMP nodes.
  - MPI vs. MPI/OpenMP/Pthreads.
- Lesson learned:
  1. Nothing magic about programming model.
  2. For SMP model to matter: *Algorithms must exploit shared memory*.

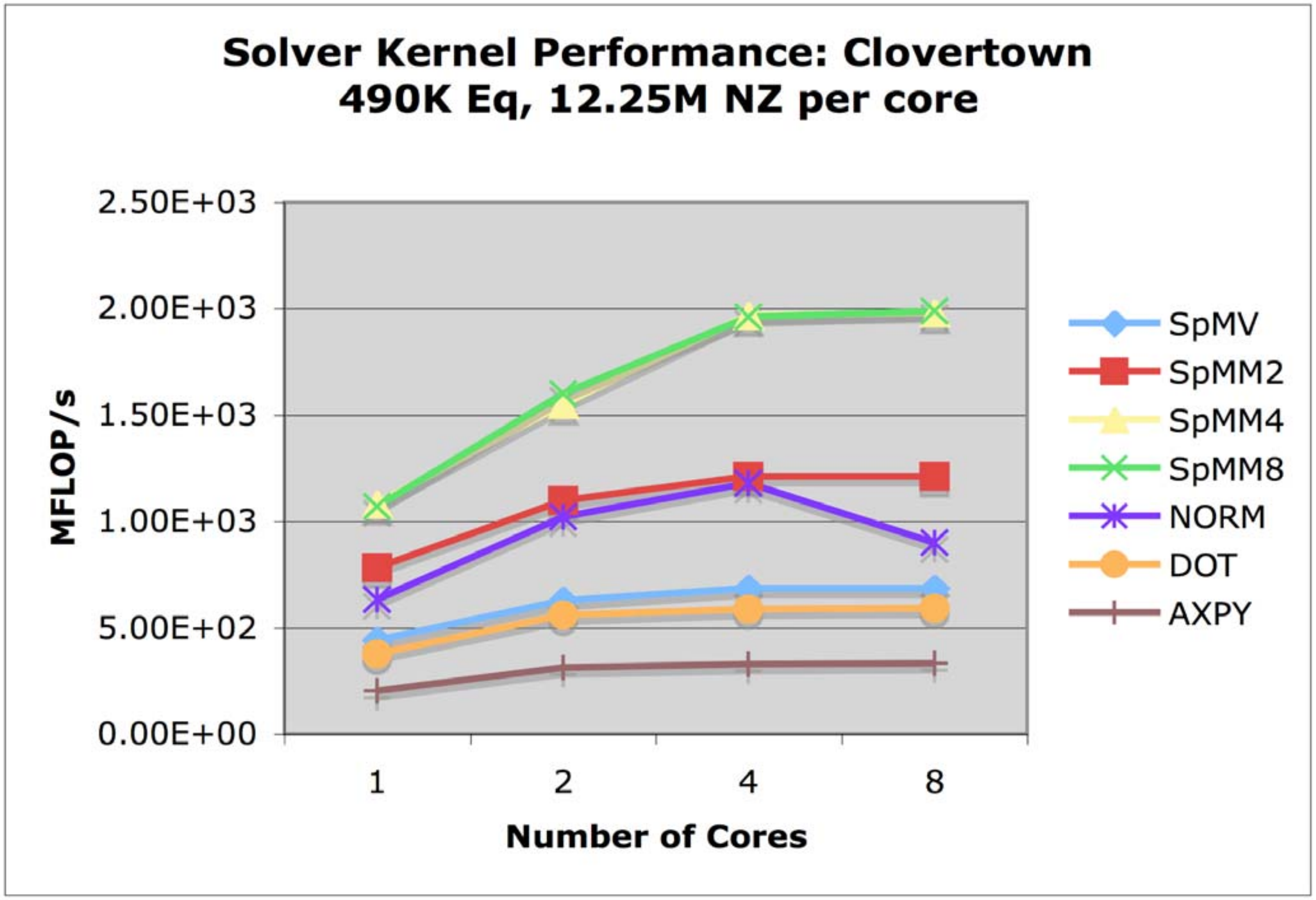
A Few HPCCG Multicore Results

- Float useful:
  - **Mixed precision algorithms.**
- Memory system performance even more important:
  - **Saturation means loss of core use.**
- Memory placement a concern:
  - **Shared memory allows remote placement.**
- NiagaraT2 threads hide latency:
  - **Easiest node to program.**

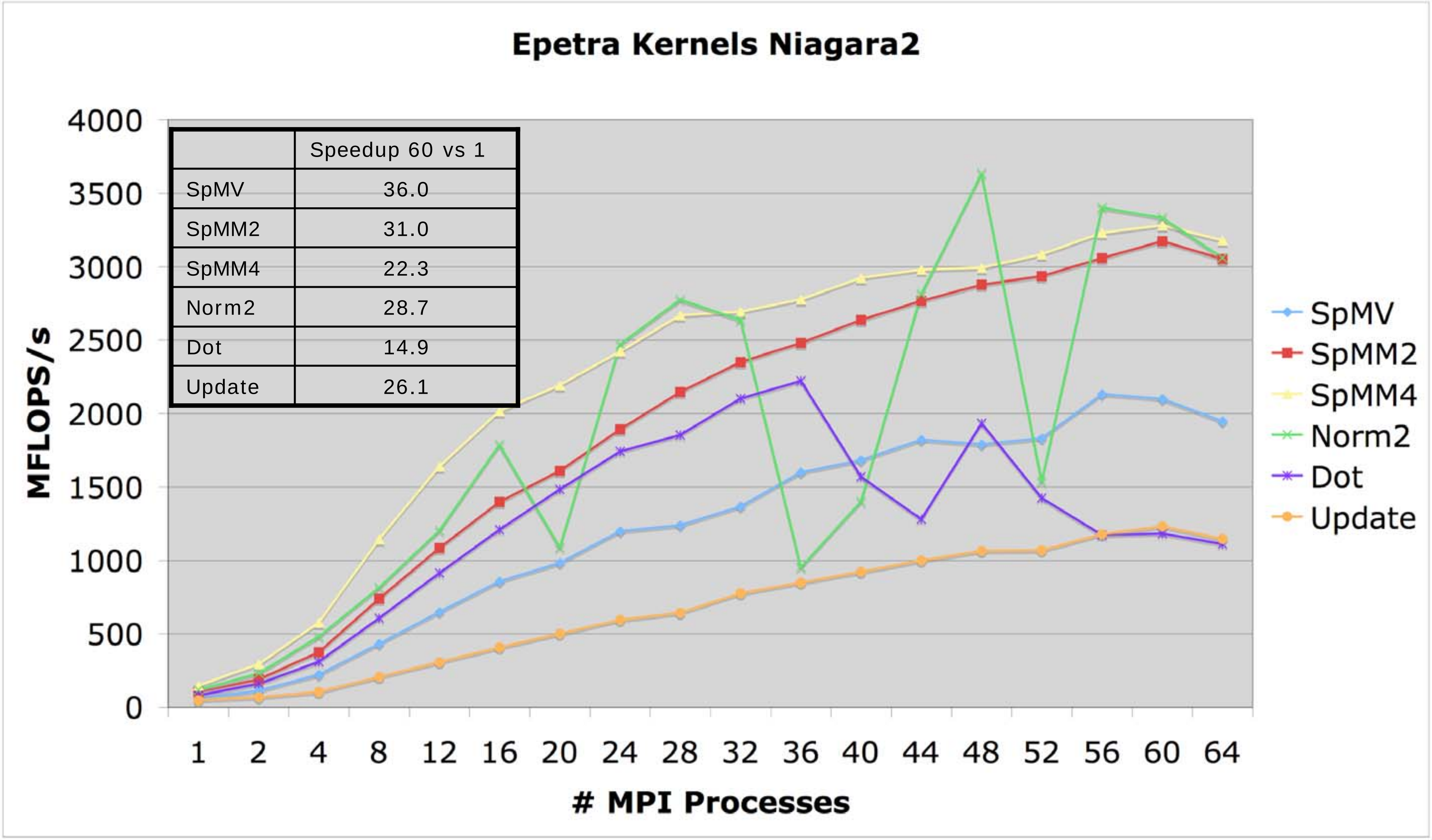


Epetra Benchmark Tests

- Focused on core Epetra kernels:
  - Sparse MV, MM.
  - Dot products, norms, daxpy's.
- spMM:
  - Better performance.
  - Better core utilization.



Epetra Kernels on Niagara2


Epetra Kernels Niagara2

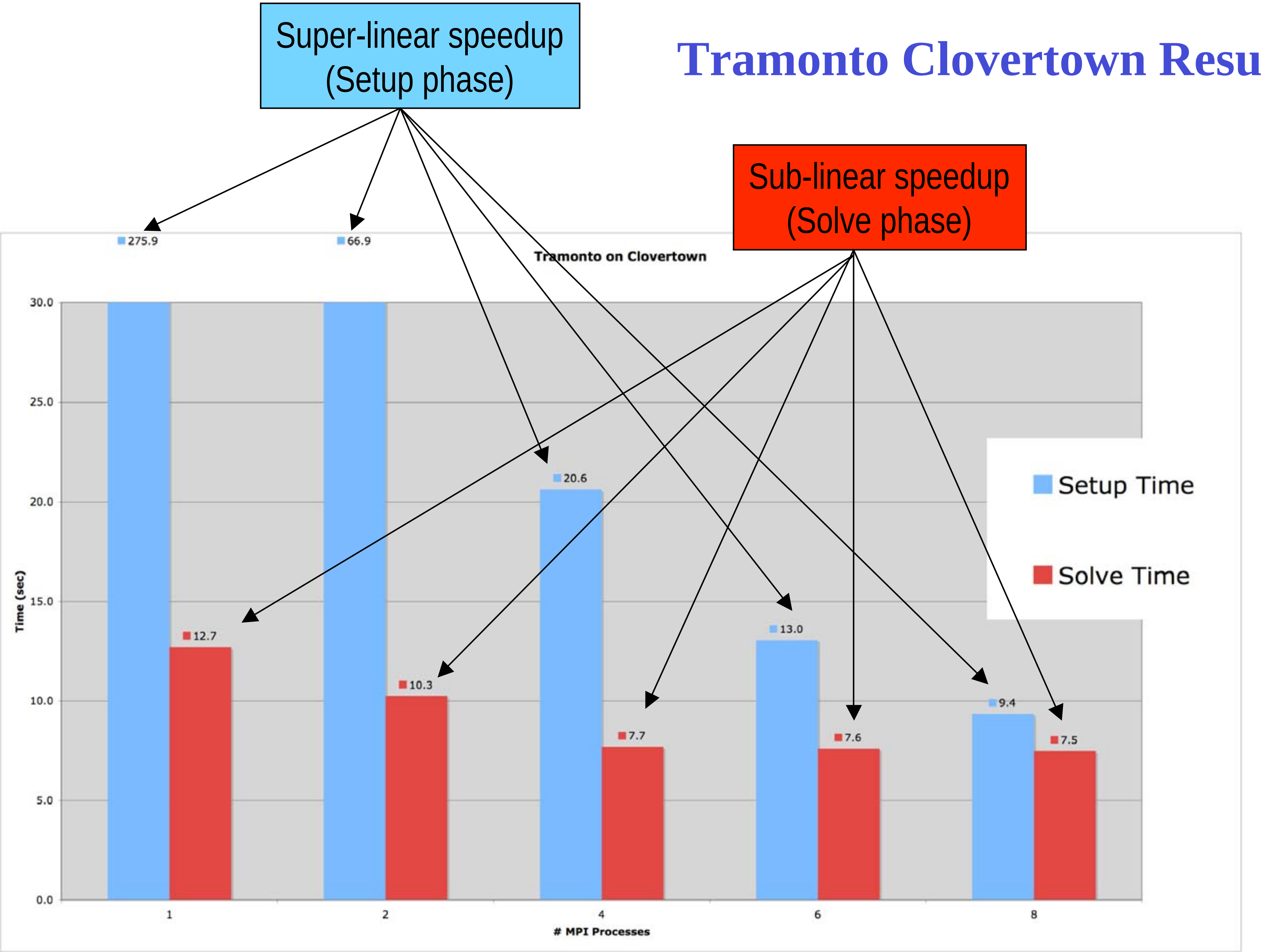
	Speedup 60 vs 1
SpMV	36.0
SpMM2	31.0
SpMM4	22.3
Norm2	28.7
Dot	14.9
Update	26.1

Sandia National Laboratories

Tramonto Clovertown Results

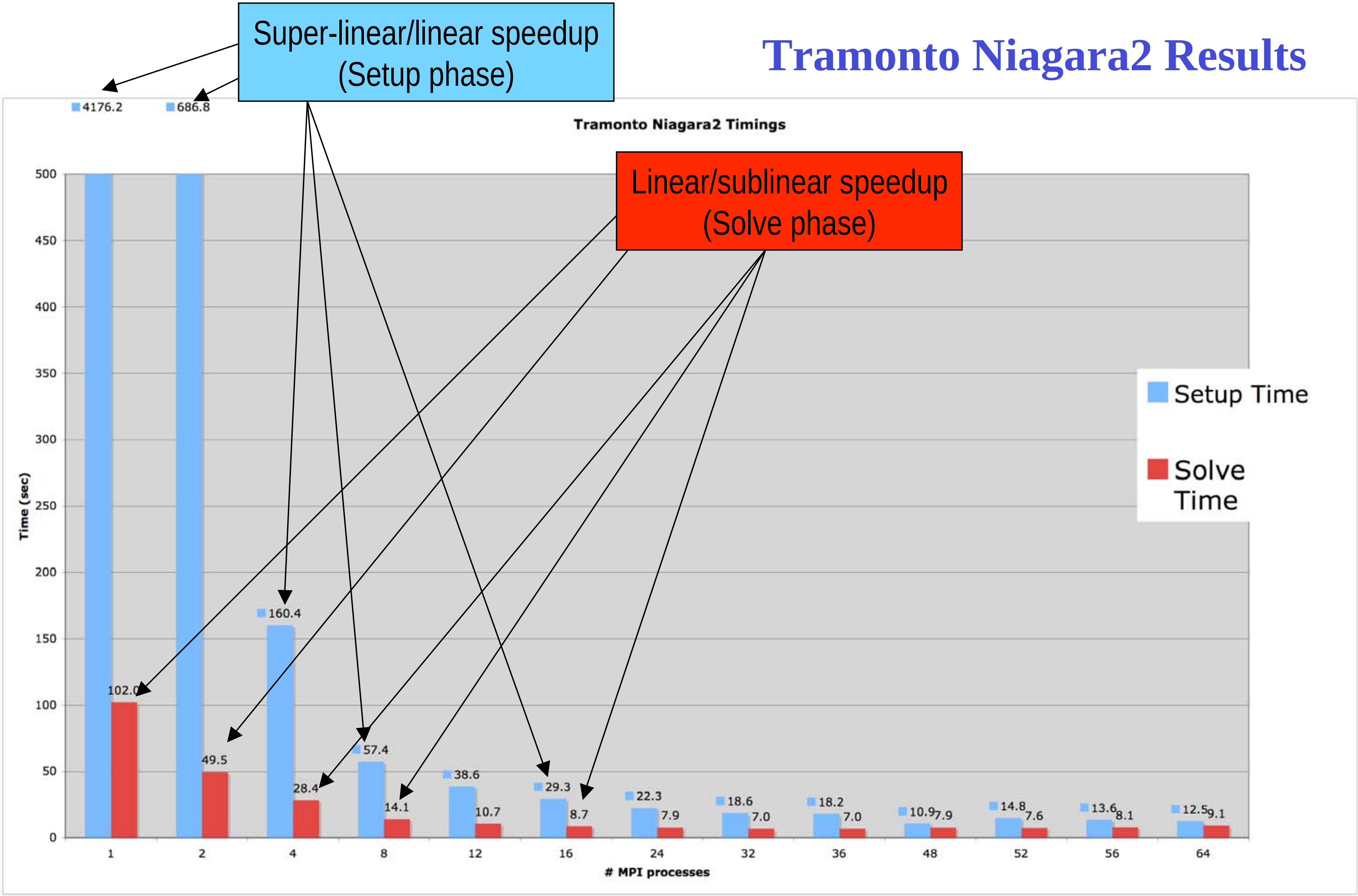
Super-linear speedup (Setup phase)

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)

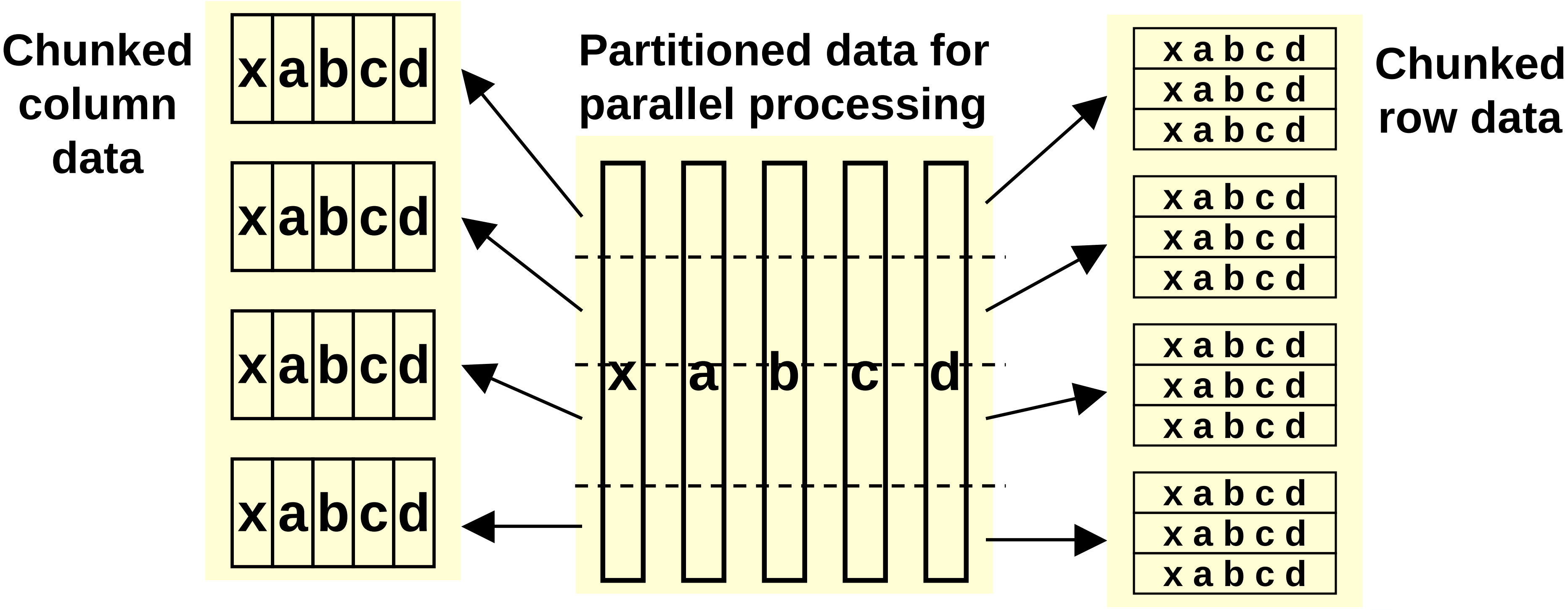
Linear/sublinear speedup (Solve phase)

Sandia National Laboratories

Vector Multi-update

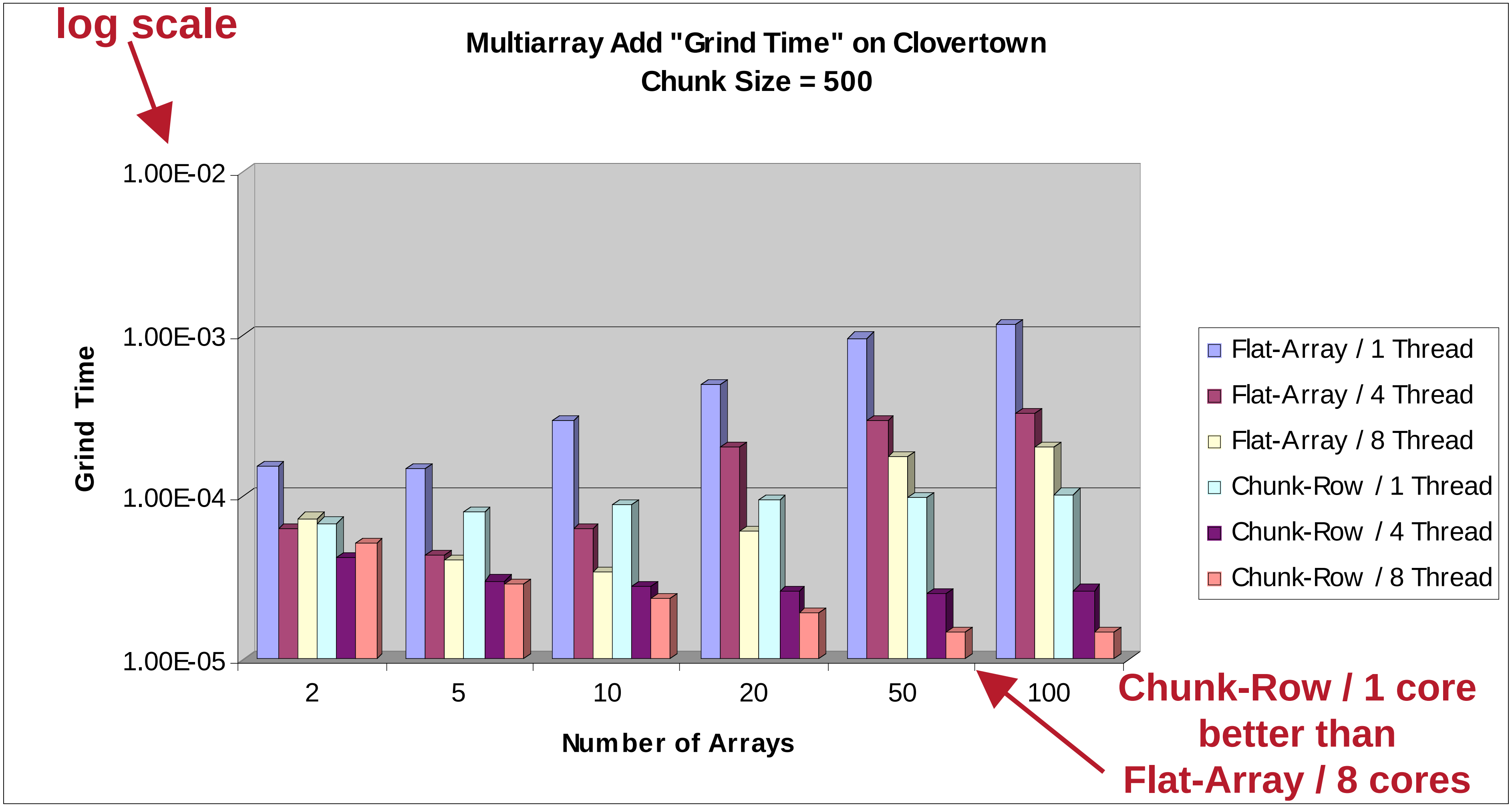
(courtesy H.C. Edwards)

- Cores compete for access to main memory
- Consider: x[i] = f(a[i], b[i], c[i], d[i], …); parallel on ‘i’
  - Compare performance of ‘Array’ versus ‘Chunk’ data structures



Chunked Data Structures Experiment
Clovertown – Scaling

Flat-Array 1,4,8 threads vs. Chunk-Row 1,4,8 threads

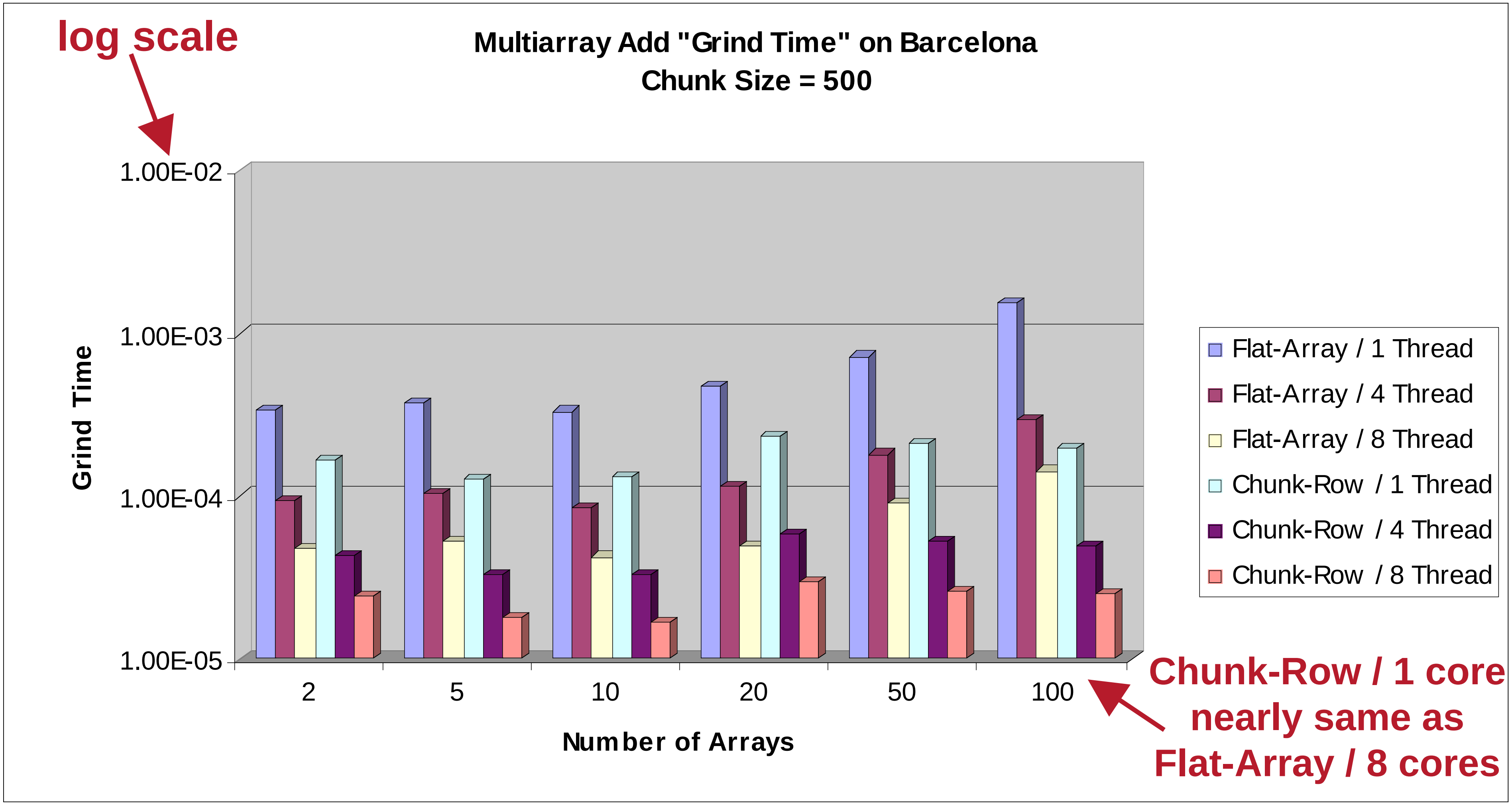


log scale

Chunk-Row / 1 core better than Flat-Array / 8 cores

Chunked Data Structures Experiment
Barcelona – Scaling

Flat-Array 1,4,8 threads vs. Chunk-Row 1,4,8 threads

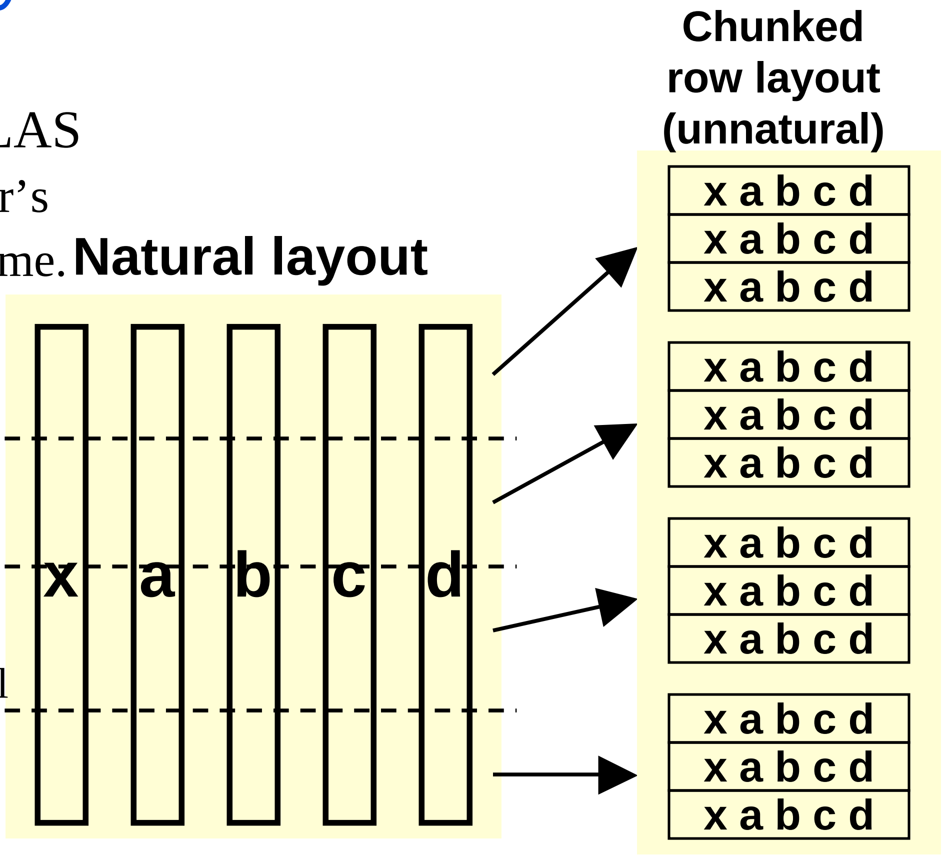


log scale

Chunk-Row / 1 core nearly same as Flat-Array / 8 cores

Unnatural Data Layouts: Observations

- Unnatural layouts are troublesome.
- Have been around a long time: Dense BLAS
  - Actual compute layout different than user’s
  - Compute rich: Translation done in real time.
- Sparse, vector computations much more challenging:
  - Translation (from natural to unnatural) cannot be done in real time.
  - Forces:
    - User to deal with unnatural layout or
    - Abstraction layer with temporal or spatial overheads.
  - Unnatural layout may have fastest kernel performance, but:
    - Overhead of translation.
    - Complexity of use.
  - Require careful interface design.



Observations (So Far) for MPI Applications

1. MPI-only is a legitimate approach and the default.
2. Multicore will change how we program the node, eventually.
   - Opinions on time frame vary greatly.
   - Uncomfortable defending MPI but: Bold predictions of MPI-only demise so far have proved false.
3. Simple programming model translation is ineffective.
4. Runtime environment is fragile: process/memory placement.
5. Memory-system-intensive code problematic: Ineffective core use.
6. Multithreading helps us: performance and simpler code.
7. Data placement: Huge performance impact, abstraction a challenge.

Library Efforts for Multicore

Library Preparations for New Node Architectures (Decision Made Years Ago)

- We knew node architectures would change…
- Abstract Parallel Machine Interface: Comm Class.
- Abstract Linear Algebra Objects:
  - Operator class: Action of operator only, no knowledge of how.
  - RowMatrix class: Serve up a row of coefficients on demand.
  - Pure abstract layer: No unnecessary constraints at all.
- Model Evaluator:
  - Highly flexible API for linear/non-linear solver services.
- Templated scalar and integer types:
  - Compile-time resolution float, double, quad,… int, long long,…
  - Mixed precision algorithms.

Library Effort in Response to Node Architecture Trends

- Block Krylov Methods (Belos & Anasazi):
  - Natural for UQ, QMU, Sensitivity Analysis…
  - Superior Node and Network complexity.
- Specialized sparse matrix data structures:
  - Sparse diagonal, sparse-dense, composite, leverage OSKI.
- Templated Kernel Libraries (Tpetra & Tifpack):
  - Choice of float vs double made when object created.
  - High-performance multiprecision algorithms.
- Shared memory node-only algorithms:
  - Triangular solves, multi-level preconditioner smoothers.
- Kokkos Node class
  - Intel TBB support, compatible with OpenMP, Pthreads, …
  - Clients of Kokkos::TbbNode can access static, ready-to-work thread pool.
  - Code above the basic kernel level is unaware of threads.
- MPI-only+MPI/PNAS
  - Application runs MPI-only (8 flat MPI processes on dual quad-core)
  - Solver runs:
    - MPI-only when interfacing with app using partitioned nodal address space (PNAS).
    - 2 MPI processes, 4 threads each when solving problem.

Sandia National Laboratories

C++ Templates

Standard method prototype for apply matrix-vector multiply:

```
template<typename OT, typename ST>
CisMatrix::apply(Vector<OT, ST> const& x, Vector<OT, ST>& y)
```

Mixed precision method prototype (DP vectors, SP matrix):

```
template<typename OT, typename ST>
CisMatrix::apply(Vector<OT, ScalarTraits<ST>::dp()>  const& x,
                 Vector<OT, ScalarTraits<ST>::dp()> & y)
```

Sample usage:

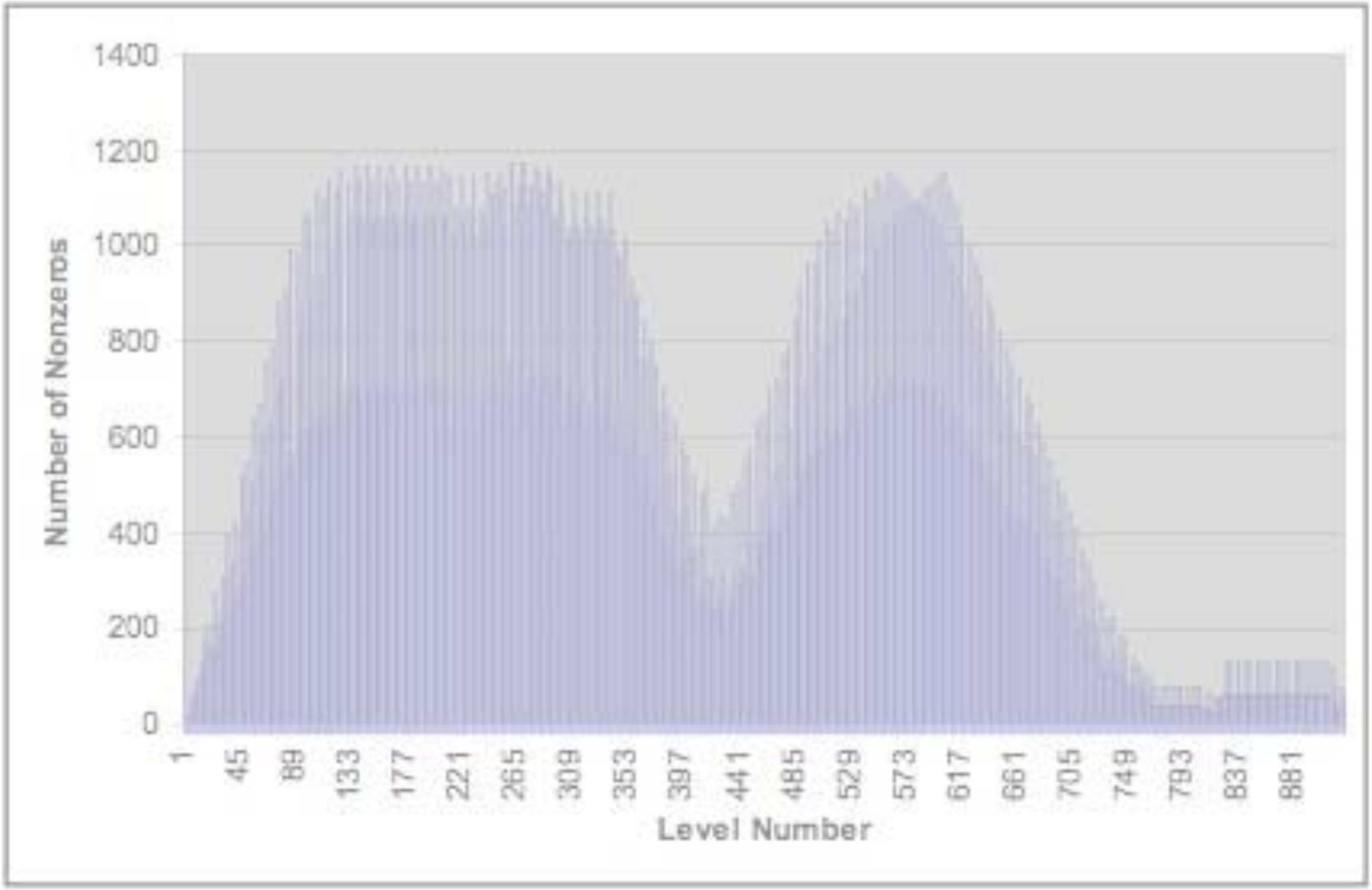
```
Tpetra::Vector<int, double> x, y;
Tpetra::CisMatrix<int, float> A;
A.apply(x, y);  // Single precision matrix applied to double precision vectors
```

C++ Templates

- Compile time polymorphism.
- True generic programming.
- No runtime performance hit.
- Huge compile-time performance hit:
  - But this is OK: Good use of multicore :)
  - Can be reduced for common data types.
- Example was for float/double but works for:
  - complex<float>/complex<double>.
  - Arbitrary precision.

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 & 0 \\ l_{31} & 0 & 1 & 0 & 0 \\ 0 & 0 & l_{43} & 1 & 0 \\ l_{51} & 0 & l_{53} & 0 & 1 \end{bmatrix} \quad \text{Solve } Ly = x.$$

Shared Memory Algorithms

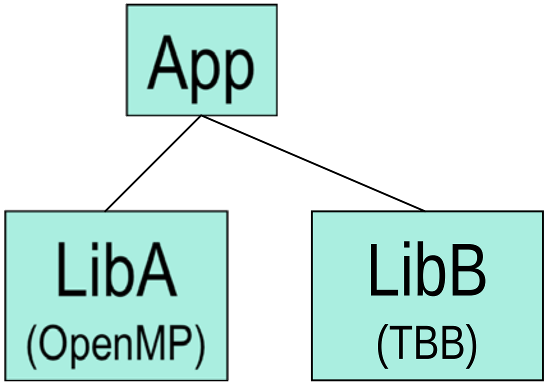


Programming Models for Scalable Homogeneous Multicore (beyond single-level MPI-only)

Sandia National Laboratories

Threading under MPI

- Default approach: Successful in many applications.
- Concerns:
  - Opaqueness of work/data pair assignment.
    - Lack of granularity control.
  - Collisions: Multiple thread models.
    - Performance issue, not correctness.



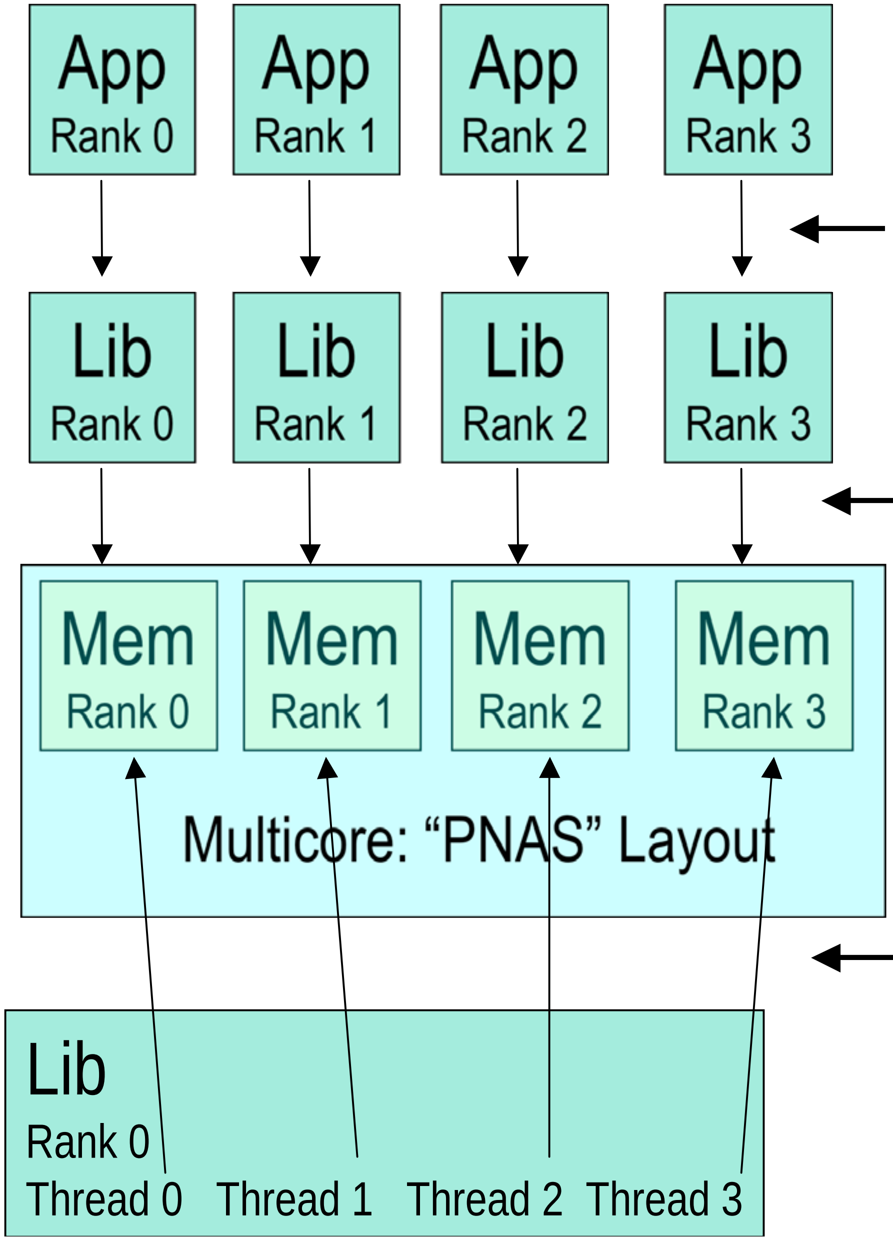
- Bright spot: Intel Thread Building Blocks (TBB).
  - Iterator (C++ language feature) model.
  - Opaque or transparent: User choice.

MPI Under MPI

- Scalable multicores:
  - Two different MPI architectures.
  - Machines within a machine.
- Exploited in single-level MPI:
  - Short-circuited messages.
  - Reduce network B/W.
  - Missing some potential.
- Nested algorithms.
- Already possible.
- Real attraction: No new node programming model.
- Can even implement shared memory algorithms (with some enhancements to MPI).

“Ping-pong” test	**Latency (microsec)**	**Bandwidth (MB/sec)**
Inter-node machine	0.71	1082
Intra-node machine	47.5	114

MPI-Only + MPI/Threading: *Ax=b*



App Rank 0 | App Rank 1 | App Rank 2 | App Rank 3

Lib Rank 0 | Lib Rank 1 | Lib Rank 2 | Lib Rank 3

Mem Rank 0 | Mem Rank 1 | Mem Rank 2 | Mem Rank 3

Multicore: “PNAS” Layout

Lib Rank 0

Thread 0 Thread 1 Thread 2 Thread 3

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.

Sandia National Laboratories

Heterogeneous Multicore Issues

Excited about multimedia processors

- Inclusion of native double precision.
- Large consumer market.
- Qualitative performance improvement over standard microprocessors…
- If your computation matches the architecture.
- Many of our computations do match well.
- But a long road ahead…

APIs for Heterogeneous Nodes (A Mess)

Processor	API
NVIDIA	CUDA
AMD/ATI	Brook+
STI Cell	ALF
Intel Larrabee	Ct
Most/All?	Sequoia
Most	RapidMind (Proprietary)
Apple/All	OpenCL

Commonality: Fine-grain functional programming.

Our Response: A Library Node Abstraction Layer

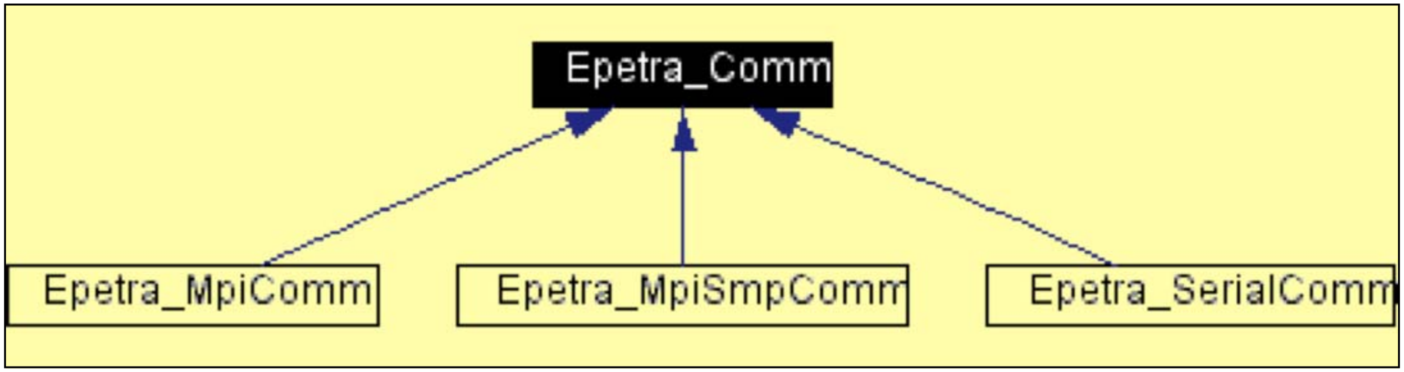
Epetra Communication Classes

- Epetra_Comm is a pure virtual class:
  - Has no executable code: Interfaces only.
  - Encapsulates behavior and attributes of the parallel machine.
  - Defines interfaces for basic services such as:
    - Collective communications.
    - Gather/scatter capabilities.
  - Allows multiple parallel machine implementations.
- Implementation details of parallel machine confined to Comm classes.
- In particular, rest of Epetra (and rest of Trilinos) has no dependence on any particular API, e.g. MPI.

Comm Methods

- CreateDistributor() const=0 [pure virtual]
- CreateDirectory(const Epetra_BlockMap & map) const=0 [pure virtual]
- **Barrier**() const=0 [pure virtual]
- **Broadcast**(double *MyVals, int Count, int Root) const=0 [pure virtual]
- **Broadcast**(int *MyVals, int Count, int Root) const=0 [pure virtual]
- **GatherAll**(double *MyVals, double *AllVals, int Count) const=0 [pure virtual]
- **GatherAll**(int *MyVals, int *AllVals, int Count) const=0 [pure virtual]
- **MaxAll**(double *PartialMaxs, double *GlobalMaxs, int Count) const=0 [pure virtual]
- **MaxAll**(int *PartialMaxs, int *GlobalMaxs, int Count) const=0 [pure virtual]
- **MinAll**(double *PartialMins, double *GlobalMins, int Count) const=0 [pure virtual]
- **MinAll**(int *PartialMins, int *GlobalMins, int Count) const=0 [pure virtual]
- **MyPID**() const=0 [pure virtual]
- **NumProc**() const=0 [pure virtual]
- **Print**(ostream &os) const=0 [pure virtual]
- **ScanSum**(double *MyVals, double *ScanSums, int Count) const=0 [pure virtual]
- **ScanSum**(int *MyVals, int *ScanSums, int Count) const=0 [pure virtual]
- **SumAll**(double *PartialSums, double *GlobalSums, int Count) const=0 [pure virtual]
- **SumAll**(int *PartialSums, int *GlobalSums, int Count) const=0 [pure virtual]
- **~Epetra_Comm**() [inline, virtual]

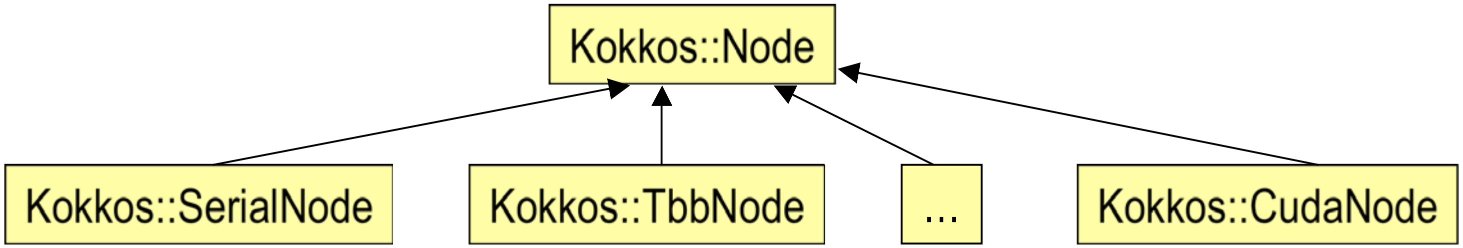
Comm Implementations



Three implementations of Epetra_Comm:

- Epetra_SerialComm:
  - Allows easy simultaneous support of serial and parallel version of user code.
- Epetra_MpiComm:
  - OO wrapping of C MPI interface.
- Epetra_MpiSmpComm:
  - Allows definition/use of shared memory multiprocessor nodes.

Abstract Node Class



- Trilinos/Kokkos: Trilinos compute node package.
- Abstraction definition in progress: Will look a lot like TBB.
- Composition needed:
  - Node with quadcore and GPU.
  - Kokkos::TbbNode uses Kokkos::SerialNode.
- Trilinos/Tpetra:
  - Tpetra::Comm constructor takes Kokkos::Node object.

Sandia National Laboratories

Going Forward: Changing the Atomic Unit

- Now:
  Single-level MPI-only OK for many apps.
- Future:
  Hiding network heterogeneity beneath single MPI level too hard.
- Philisophical approach:
  Node becomes the new atomic unit.
- Key Requirement:
  Portable standard node API.
- Hard work:
  Changes are ubiquitous (unlike MPI).

Summary

- **Exciting times:** for architecture and software design.
- **MPI-only sufficient**:
  - 3-5 years for many existing apps.
  - Except for Roadrunner apps, and similar.
- **Reducing B/W requirements:** even more important.
- **C++ is the right language** for new development:
  - Templates, compatibility with node SDKs, advanced features.
  - Fortran still OK for single core performance.
  - Fortran apps should be linkable with C++.
- **If Libs do a good job**: Some Apps can delay multicore awareness.
- **Multimedia processors:** seem to have right mix for next qualitative performance improvement.
- **Possible scenario for some apps/libs**:
  - Heterogenous API superior on homogeneous nodes.
  - Go directly from single-level MPI-only to MPI+heterogenous node?
- **A common, standard API for multicore:** Most critical need.