Parallelism for the Very Large and Very Small

Matthew Knepley

Mathematics and Computer Science Division
Argonne National Laboratory

Scientific Libraries: MPI & Multicore Issues and Plans
Sandia CSRI Workshop
Bishop's Lodge, NM
June 3, 2008

MPI Problems

Where are the problems for MPI?

MPI Problems

Where are the problems for MPI?

- Very small

MPI Problems

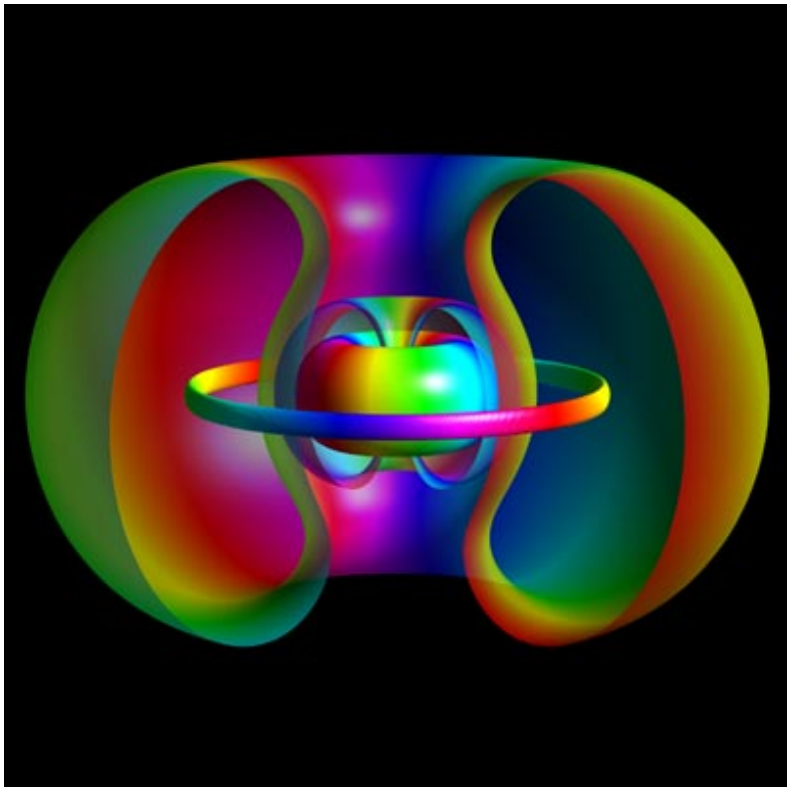
Where are the problems for MPI?

Very small

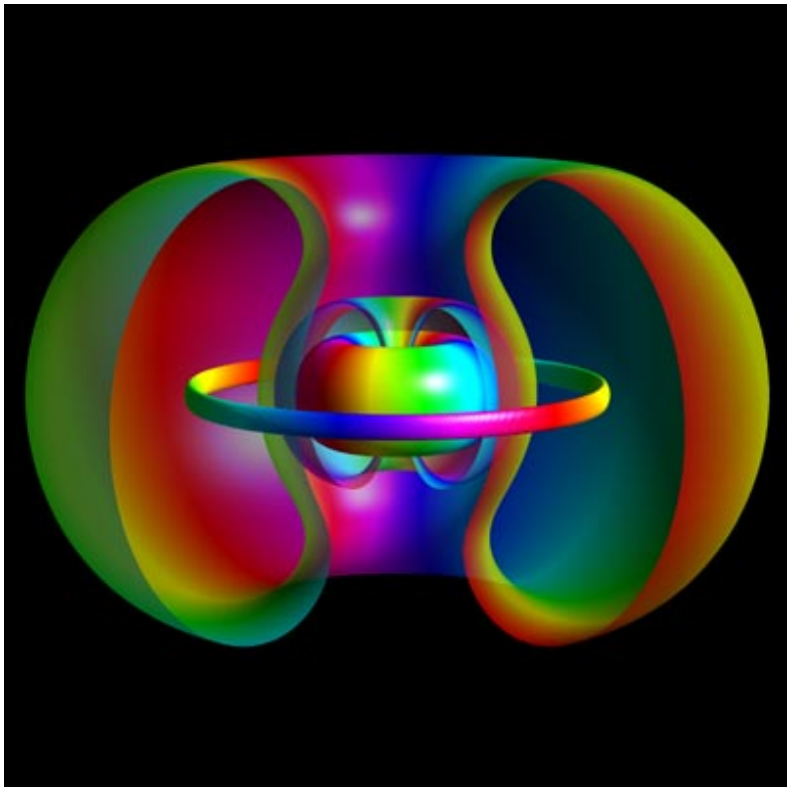
Very Large

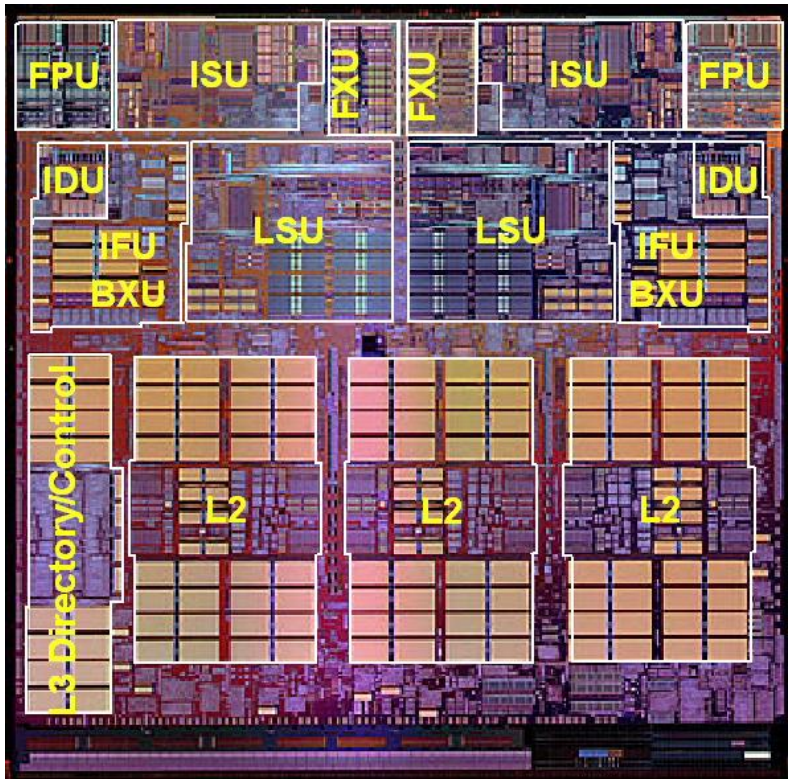
Outline

Very Small



Very Small




FPU
ISU
FXU
FXU
ISU
FPU
IDU
IFU
BXU
LSU
LSU
IDU
IFU
BXU
L3 Directory/Control
L2
L2
L2

Code Generation

Big Idea: **Code Generation**

- Compile time examples
  - Dense linear algebra
  - Digital Signal Processing
  - FEM Accumulation/Assembly
  - Model coupling
- Runtime support
  - Inspector-Executor
  - PGAS
  - Libraries, if written correctly, can be oblivious

Code Generation

Big Idea: **Code Generation**

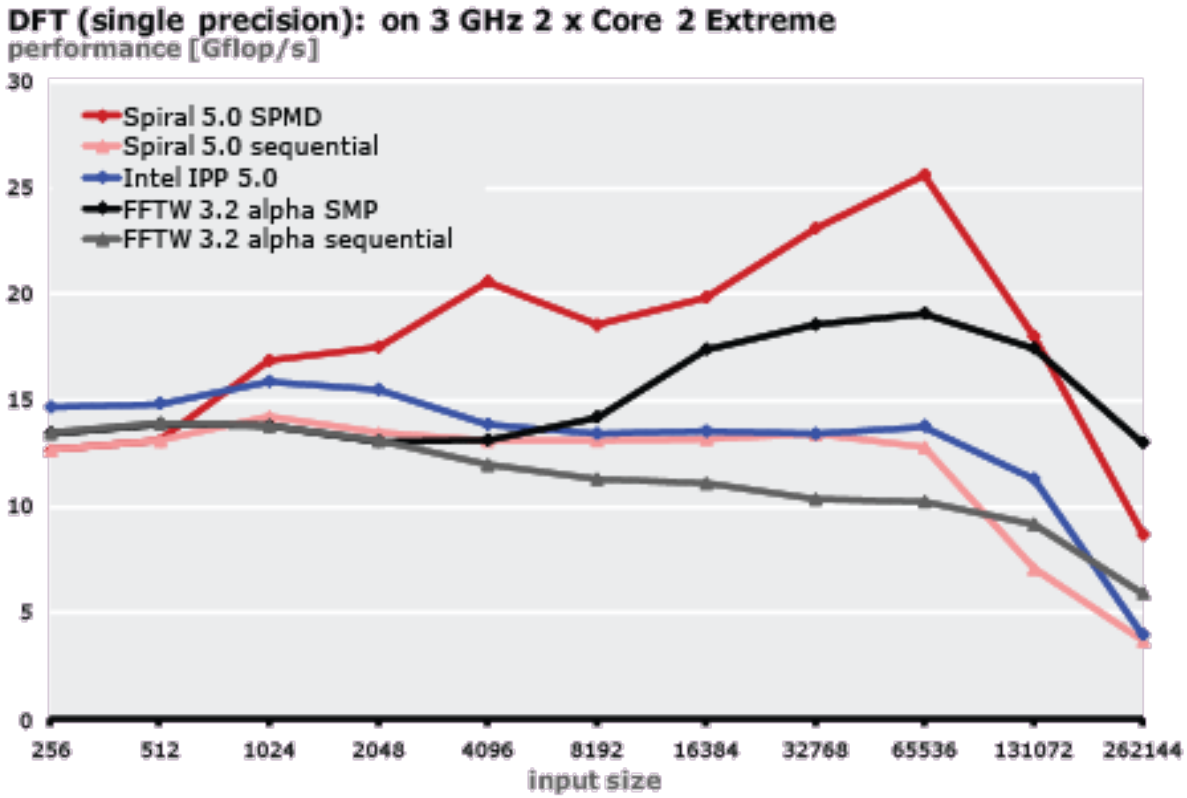
- Compile time examples
  - Dense linear algebra
  - Digital Signal Processing
  - FEM Accumulation/Assembly
  - Model coupling
- Runtime support
  - Inspector-Executor
  - PGAS
  - Libraries, if written correctly, can be oblivious

Code Generation

Big Idea: **Code Generation**

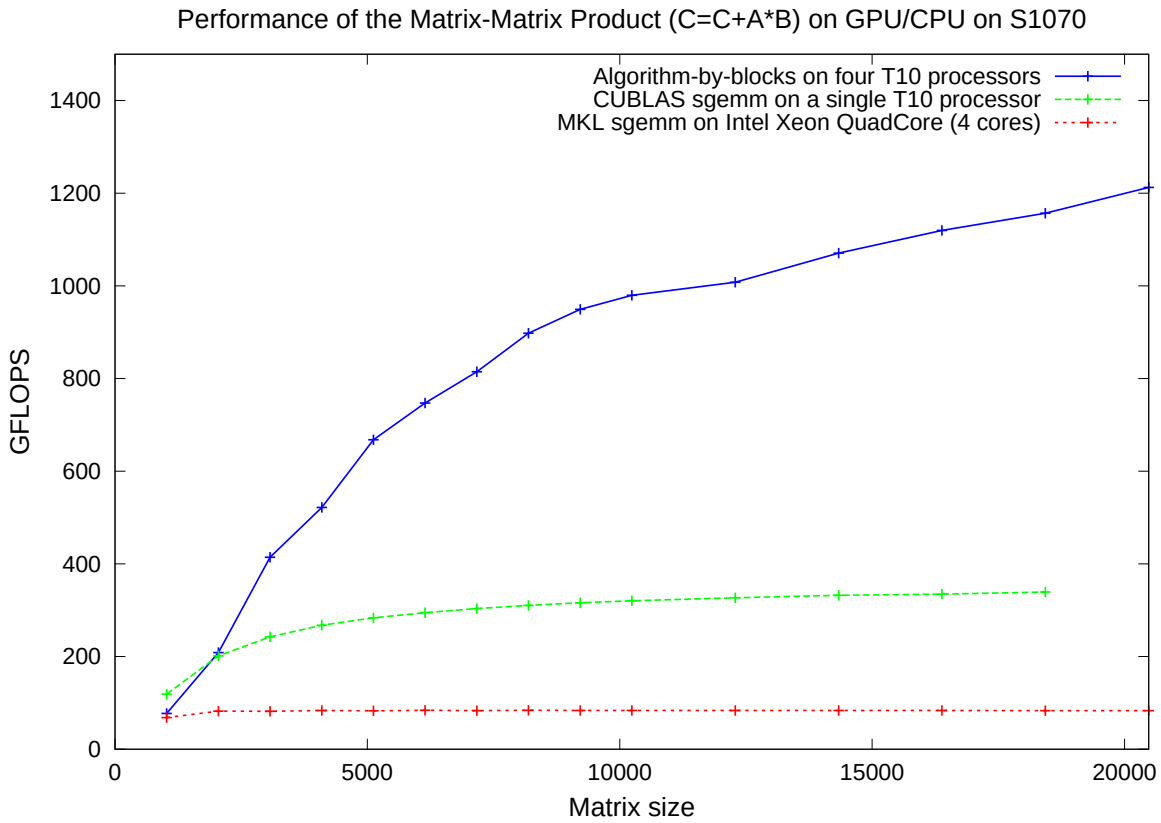
- Compile time examples
  - Dense linear algebra
  - Digital Signal Processing
  - FEM Accumulation/Assembly
  - Model coupling
- Runtime support
  - Inspector-Executor
  - PGAS
  - Libraries, if written correctly, can be oblivious

Spiral



- Spiral Team, http://www.spiral.net
- Uses an intermediate language, SPL, and then generates C
- Works by circumscribing the algorithmic domain

FLAME & FLASH



- Robert van de Geijn, http://www.cs.utexas.edu/users/flame
- FLAME is an Algorithm-By-Blocks interface
- FLASH/SuperMatrix is a runtime system

Conclusions

- Circumscribe algorithmic domain
- Specialize to algorithm/hardware with code generation
- Runtime decisions informed by high level information

Outline

Very Large

Very Large

Hierarchical Design

Big Idea: **Hierarchy**

Multilevel Method

- Solve local problems
  - Locality of operations is key for efficient implementation
  - Should enable reuse of serial implementation
- Stitch together to form a global solution
  - Manifold or Domain Decomposition idea: local pieces w/ overlap
  - Global complexity is encoded in the (small) Overlap

Hierarchical Design

Big Idea: **Hierarchy**

Multilevel Method

- Solve local problems
  - Locality of operations is key for efficient implementation
  - Should enable reuse of serial implementation
- Stitch together to form a global solution
  - Manifold or Domain Decomposition idea: local pieces w/ overlap
  - Global complexity is encoded in the (small) Overlap

Hierarchical Design

Big Idea: **Hierarchy**

Multilevel Method

- Solve local problems
  - Locality of operations is key for efficient implementation
  - Should enable reuse of serial implementation
- Stitch together to form a global solution
  - Manifold or Domain Decomposition idea: local pieces w/ overlap
  - Global complexity is encoded in the (small) Overlap

Hierarchical Design

Big Idea: **Hierarchy**

Multilevel Method

- Solve local problems
  - Locality of operations is key for efficient implementation
  - Should enable reuse of serial implementation
- Stitch together to form a global solution
  - Manifold or Domain Decomposition idea: local pieces w/ overlap
  - Global complexity is encoded in the (small) Overlap

Hierarchical Design

Big Idea: **Hierarchy**

Multilevel Method

- Solve local problems
  - Locality of operations is key for efficient implementation
  - Should enable reuse of serial implementation
- Stitch together to form a global solution
  - Manifold or Domain Decomposition idea: local pieces w/ overlap
  - Global complexity is encoded in the (small) Overlap

MPICH-G2

Early Attempt at Hierarchy

- Communicator hierarchy, topology depth
- Only exposed to the user through Comm attributes
  - Still have to support flat model
- Hierarchy information is buried too deep
  - Only really accessible in the implementation (collectives)

MPICH-G2

Early Attempt at Hierarchy

- Communicator hierarchy, topology depth
- Only exposed to the user through Comm attributes
  - Still have to support flat model
- Hierarchy information is buried too deep
  - Only really accessible in the implementation (collectives)

MPICH-G2

Early Attempt at Hierarchy

- Communicator hierarchy, topology depth
- Only exposed to the user through Comm attributes
  - Still have to support flat model
- Hierarchy information is buried too deep
  - Only really accessible in the implementation (collectives)

Sieve Overview

- Hierarchy is the centerpiece
  - Strip out unneeded complexity (dimension, shape, . . .)
- Single relation, covering, handles all hierarchy
  - Rich enough for FEM
- Single operation, completion, for parallelism
  - Enforces consistency of the relation

Sieve Overview

- Hierarchy is the centerpiece
  - Strip out unneeded complexity (dimension, shape, . . .)
- Single relation, covering, handles all hierarchy
  - Rich enough for FEM
- Single operation, completion, for parallelism
  - Enforces consistency of the relation

Sieve Overview

- Hierarchy is the centerpiece
  - Strip out unneeded complexity (dimension, shape, . . .)
- Single relation, covering, handles all hierarchy
  - Rich enough for FEM
- Single operation, completion, for parallelism
  - Enforces consistency of the relation

Uses

Completion has many uses:

FEM accumulating integrals on shared faces

FVM accumulating fluxes on shared cells

FDM setting values on ghost vertices

- distributing mesh entities after partition
- redistributing mesh entities and data for load balance
- accumlating matvec for a partially assembled matrix

Uses

Completion has many uses:

FEM accumulating integrals on shared faces

FVM accumulating fluxes on shared cells

FDM setting values on ghost vertices

- distributing mesh entities after partition
- redistributing mesh entities and data for load balance
- accumlating matvec for a partially assembled matrix

Uses

Completion has many uses:

FEM accumulating integrals on shared faces

FVM accumulating fluxes on shared cells

FDM setting values on ghost vertices

- distributing mesh entities after partition
- redistributing mesh entities and data for load balance
- accumlating matvec for a partially assembled matrix

Uses

Completion has many uses:

FEM accumulating integrals on shared faces

FVM accumulating fluxes on shared cells

FDM setting values on ghost vertices

- distributing mesh entities after partition
- redistributing mesh entities and data for load balance
- accumlating matvec for a partially assembled matrix

Uses

Completion has many uses:

FEM accumulating integrals on shared faces

FVM accumulating fluxes on shared cells

FDM setting values on ghost vertices

- distributing mesh entities after partition
- redistributing mesh entities and data for load balance
- accumlating matvec for a partially assembled matrix

Uses

Completion has many uses:

FEM accumulating integrals on shared faces

FVM accumulating fluxes on shared cells

FDM setting values on ghost vertices

- distributing mesh entities after partition
- redistributing mesh entities and data for load balance
- accumlating matvec for a partially assembled matrix

Uses

Completion has many uses:

FEM accumulating integrals on shared faces

FVM accumulating fluxes on shared cells

FDM setting values on ghost vertices

- distributing mesh entities after partition
- redistributing mesh entities and data for load balance
- accumlating matvec for a partially assembled matrix

Hierarchical Interfaces

Global/Local Dichotomy is the Heart of DD

Software interfaces do not adequately reflect this

- PETSc DA is too specialized
  - Basically 1D methods applied to Cartesian products
- PETSc Index Sets and VecScatters are too fine
  - User “does everything”, no abstraction
- PETSc Linear Algebra (Vec & Mat) is too coarse
  - No access to the underlying connectivity structure

Conclusions

- Have concise, abstract, flexible interface for hierarchy
- Need support for interaction with communication primitives
- Specialized networks cannot currently implement sophisticated tree algorithms

Outline

Conclusions

- Multicore performance should be improved with:
  - Better code generation and runtime tools
  - Algorithmic specificity
- Multiprocess scalability should be improved with:
  - Explicitly hierarchical interfaces/libraries
  - Better interaction of algorithms with communication

Outline

Question 15

Are there extensions that can be made to MPI so that MPI is more amenable to writing scalable applications and to building next-generation libraries and languages?

Answer 15

Hierarchy is the key notion in nearly every optimal algorithm known. For example, the solvers Multigrid (MG), Fast Multipole Method (FMM), and FETI are all based upon a hierarchical decomposition of the problem domain, which is then used to aggregate the effects of local, usually linear, operations. Use of these algorithms, particularly in large community codes, has been impeded by the high cost of implementation. However, they are arguably key to the scalability and efficiency of application codes on next-generation exascale architectures. Simple hierarchical extensions to MPI could greatly ease the implementation process, and result in much faster and more scalable applications. For instance, a hierarchical relation between communicators could be directly mapped onto tree algorithms, such as FMM.

Moreover, modern implementations of MPI must reduce the data stored per node in order to avoid explosion of local storage for the implementation. Space efficient implementations of MPI would make

Outline

Question 23

HPC is small compared to the commercial software market. What are commercial leaders like Microsoft and Google doing to prepare for an era of multicore/manycore parallelism, and how will this affect the scientific HPC world?

Answer 23

The most significant development in the commercial software market for HPC is the recent practicality of outsourced computation. The enabling technology for this development is exactly the same as that which enabled the huge growth in portable numerical libraries over the past two decades, namely abstraction of a large set of community problems to a common algorithmic domain. With a common algorithmic language, users can encode individual problems which can then be run by any computation service.

The best known example of this paradigm in the Google MapReduce implementation. However, other large players now offer much the same service to any computing customer, for instance Yahoo with Hadoop and Amazon with EC2. Outsourced computing greatly expands the notion of computing facilities, today embodied by the national centers for computation such as NERSC. This also opens the door to centralized storage of and computation on large scientific data sets. In essence, bringing computing to the data, rather than data to

Outline

Question 29

Will hierarchical problem decomposition (I call it fractal or self-similar computing) get around the billion thread programming problem (nobody is smart enough to develop billion thread codes that do anything significant)?

Answer 29

Yes, hierarchy is the key to better, more scalable algorithms. However, without sufficient computation to occupy each thread, we will not make efficient use of the machine. Migration to algorithms which have better balance between computation and communication/memory bandwidth will likely entail refactoring current applications and production of high quality middleware encapsulating both the dependency structure for computations and the task scheduling and dispatch procedure. Machine hierarchies will also play a role in managing exascale execution. We believe that specialized networks will be a key component of scalable performance for these algorithms. For example, the reduction network on BG/L allows Krylov methods to continue scaling to thousands of processors while utilizing many dot products. Extension of these networks to support scans with matrix operations would enable an even wider array of scalable algorithms, such as FMM or MG.

A New Standard?

MPI provides a good interface for data parallel algorithms. However, the extensions to task parallelism are confusing, incomplete, and sometimes slow. OpenMP does provide an interface for task parallelism. However, it does not abstract the main operations and relegates much of the user control to environment variables, rather than an API. Moreover, basic operations are absent. For example, we would like the system to accept a computation DAG from the application and use this to schedule taks dynamically at runtime. Thus, we might consider an effort to produce a standard, similar to MPI, which encapsulated task parallel algorithms.