

High-Performance Scientific Software Development: Two Glimpses into the Future

David E. Bernholdt

Computer Science and Mathematics
Oak Ridge National Laboratory

bernholdtde@ornl.gov

Research supported by the Mathematics, Information and Computational Sciences Office, Office
of Advanced Scientific Computing Research, U.S. Dept. of Energy
and
the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory

Scientific Computing Has Come of Age...

- “Computation is now regarded as an equal and indispensable partner, along with theory and experiment, in the advance of scientific knowledge and engineering practice.”
(SIAM report on CSE Education)
- “High-speed computers and networks are enabling scientific discovery across a broad spectrum...”
(PITAC Report, 1999)

... But this Brings New Challenges

- “Research problems are becoming more complex and interdisciplinary in nature.”
(PITAC Report, 1999)
- “The demand for software has grown far faster than our ability to produce it.”
(PITAC Report, 1999)
- “Although they achieve remarkable performance in some cases, the current scalable, parallel, high-end computing systems are not well-suited to many nationally important, strategic applications.”
(PITAC Report, 1999)

Modern Scientific Software Engineering Challenges

- **Productivity**
 - Time to first solution (prototyping)
 - Time to solution (“production”)
 - Software infrastructure requirements
- **Complexity**
 - Increasingly sophisticated models
 - Model coupling – multi-scale, multi-physics, etc.
 - “Interdisciplinarity”
- **Performance**
 - Increasingly complex algorithms
 - Increasingly complex computers
 - Increasingly demanding applications

Addressing the Challenges

- Component-Based Software Engineering for High-Performance Scientific Computing
- Synthesis of High-Performance Algorithms for Electronic and Nuclear Structure Calculations
- *Just two of many ways...*

The Common Component Architecture (CCA)

<http://www.cca-forum.org>

- ANL - Lori Freitag, Kate Keahey, Jay Larson, Ray Loy, Lois Curfman McInnes, Boyana Norris, ...
- Indiana University - Randall Bramley, Dennis Gannon, ...
- JPL – Dan Katz, ...
- LANL - Craig Rasmussen, Matt Sotille, ...
- LLNL - Tom Epperly, Scott Kohn, Gary Kumfert, ...
- ORNL - David Bernholdt, Wael Elwasif, Jim Kohl, Torsten Wilde, ...
- PNNL - Jarek Nieplocha, Theresa Windus, ...
- SNL - Rob Armstrong, Ben Allan, Lori Freitag Diachin, Curt Janssen, Jaideep Ray, ...
- University of Utah - Steve Parker, ...
- And others as well ...

Research supported by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, U.S. Dept. of Energy

Component-Based Software Engineering

- CBSE methodology is emerging, especially from business and internet areas
- **Software productivity**
 - Provides a “plug and play” application development environment
 - Many components available “off the shelf”
 - Facilitates reuse and interoperability of components
- **Software complexity**
 - Components encapsulate much complexity into “black boxes”
 - “Plug and play” approach simplifies applications
 - Model coupling is natural in component-based approach

So, What are Components?

- No universally accepted definition...yet
- **A unit of software deployment/reuse**
 - i.e. has interesting functionality
 - Ideally, functionality someone else might be able to (re)use
 - Has granularity consistent with component environment
- **Interacts with the outside world *only* through well-defined interfaces**
 - Components *may* maintain state information
 - But external access to state info must be through an interface (*not a common block*)
- **Implementation is opaque to the outside world**
 - Implementation language not visible
 - Component environment maintains encapsulation of impl.

HPC Puts Different Requirements on Component Models

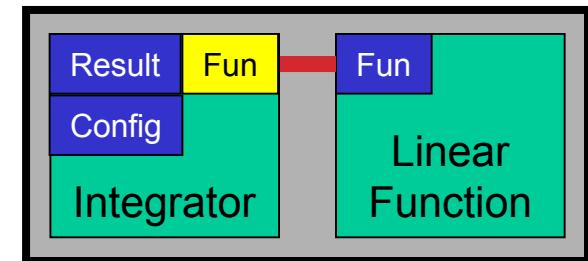
- “Commodity” environments include CORBA, COM, Enterprise JavaBeans
- **Minimize adoption overhead**
 - Make it easy to componentize legacy software
- **Minimize performance impact**
 - Allow tightly-coupled in-process components
- **Support for tightly-coupled parallel computing**
 - Commodity envs are focused on distributed
 - Distributed is nice, but parallel is critical
- **Support languages and types (and platforms) for science**
 - Fortran, complex numbers, arrays, etc.

The Common Component Architecture

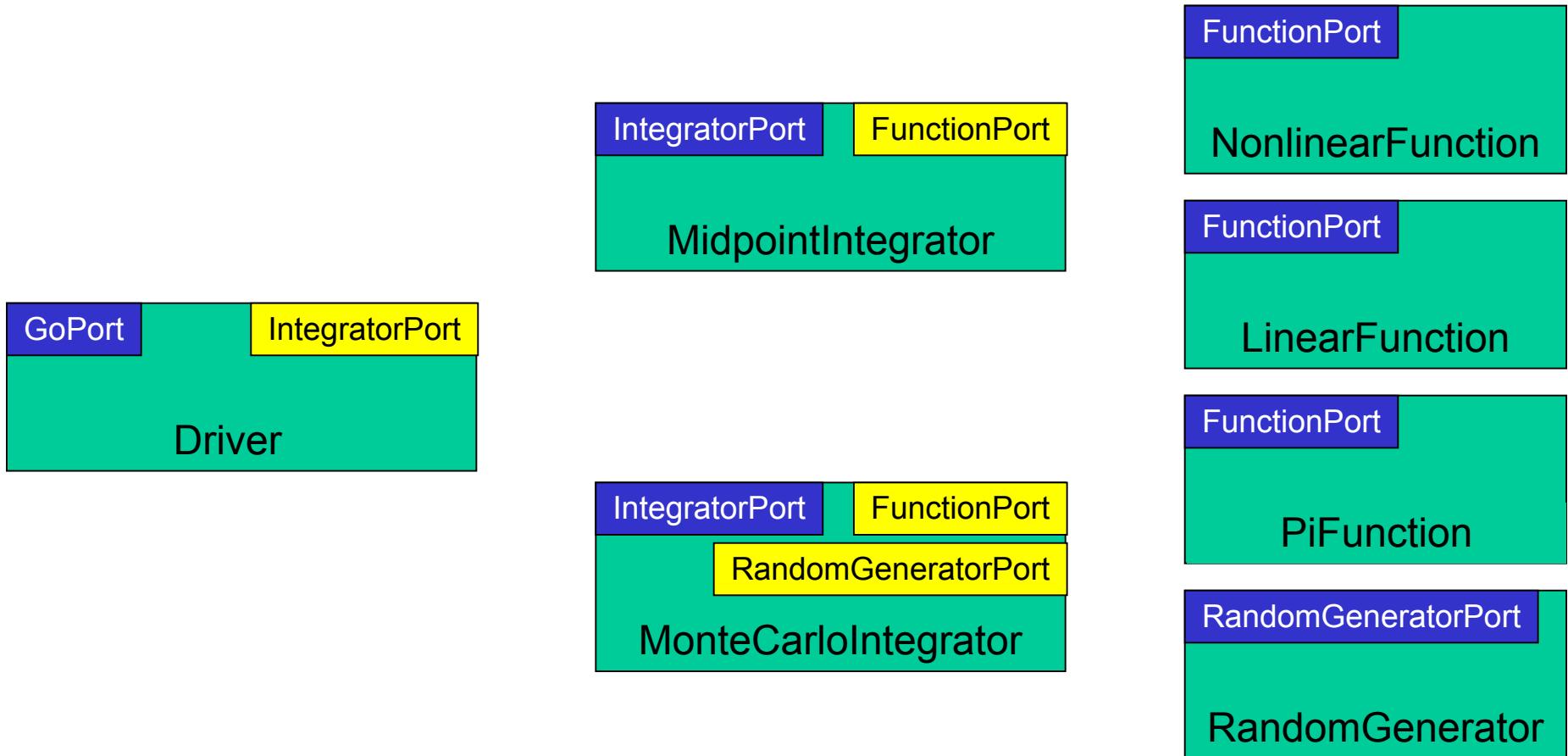
- A component model specifically designed for **high-performance scientific computing**
- Supports both **parallel** and **distributed** applications
- Designed to be implementable without sacrificing **performance**
- **Minimalist approach** makes it easier to componentize existing software
- A **tool** to enhance the productivity of scientific programmers
 - Make the hard things easier, make some intractable things tractable
 - **Not a magic bullet**

Basic CCA Terminology

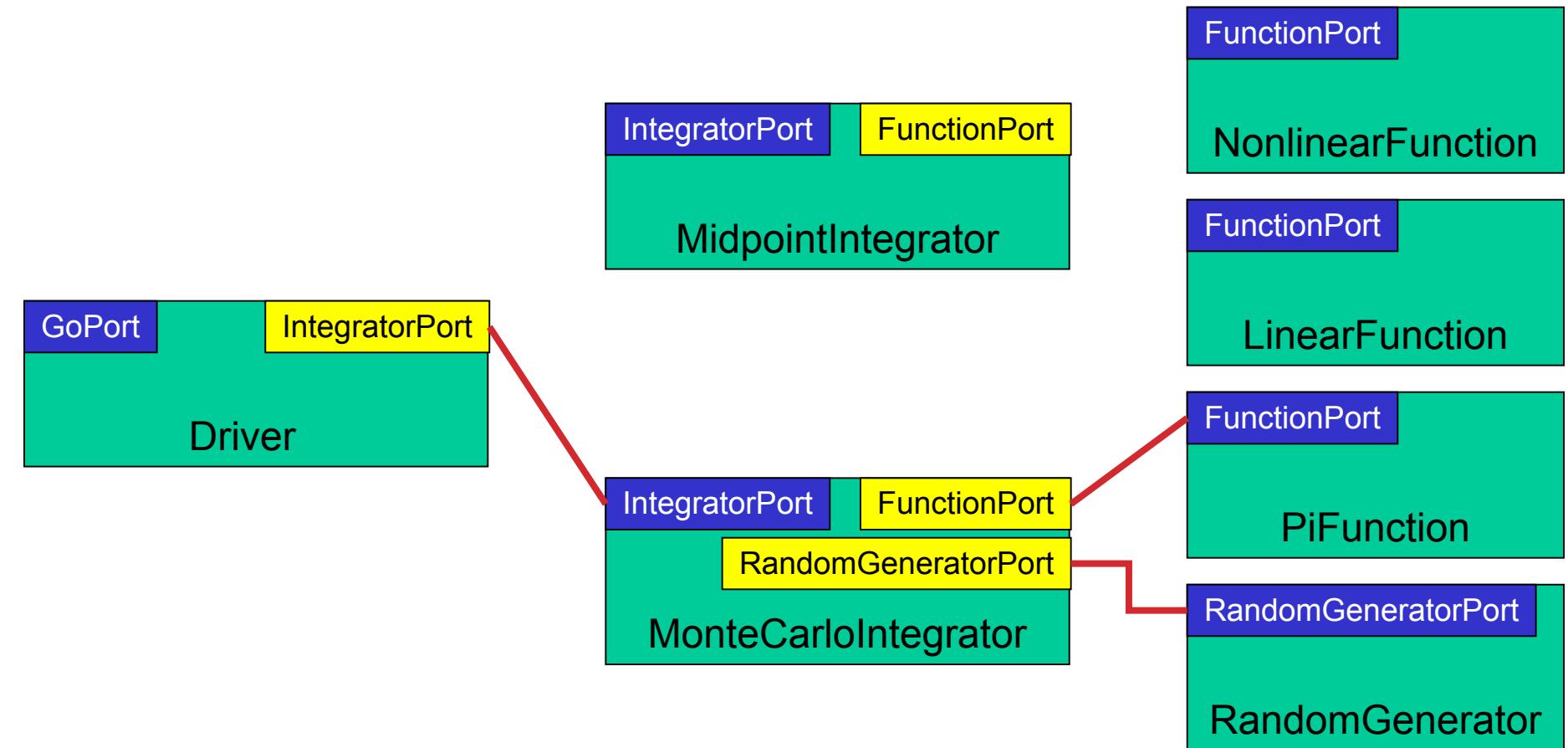
- **Port (aka *interface*)**
 - Procedural interface (not just dataflow!)
 - Like C++ abst. virtual class, Java interface
 - Uses/provides design pattern
- **Component**
 - A unit of software deployment/reuse (i.e. has interesting functionality)
 - Interacts with the outside world only through well-defined interfaces
 - Implementation is opaque to the outside world
- **Framework**
 - Holds components during application composition and execution
 - Controls the “exchange” of interfaces between components (while ensuring implementations remain hidden)
 - Provides a small set of standard, ubiquitous services to components
 - *CCA spec doesn't specify a framework per se, so components can be constructed to provide framework-like services*



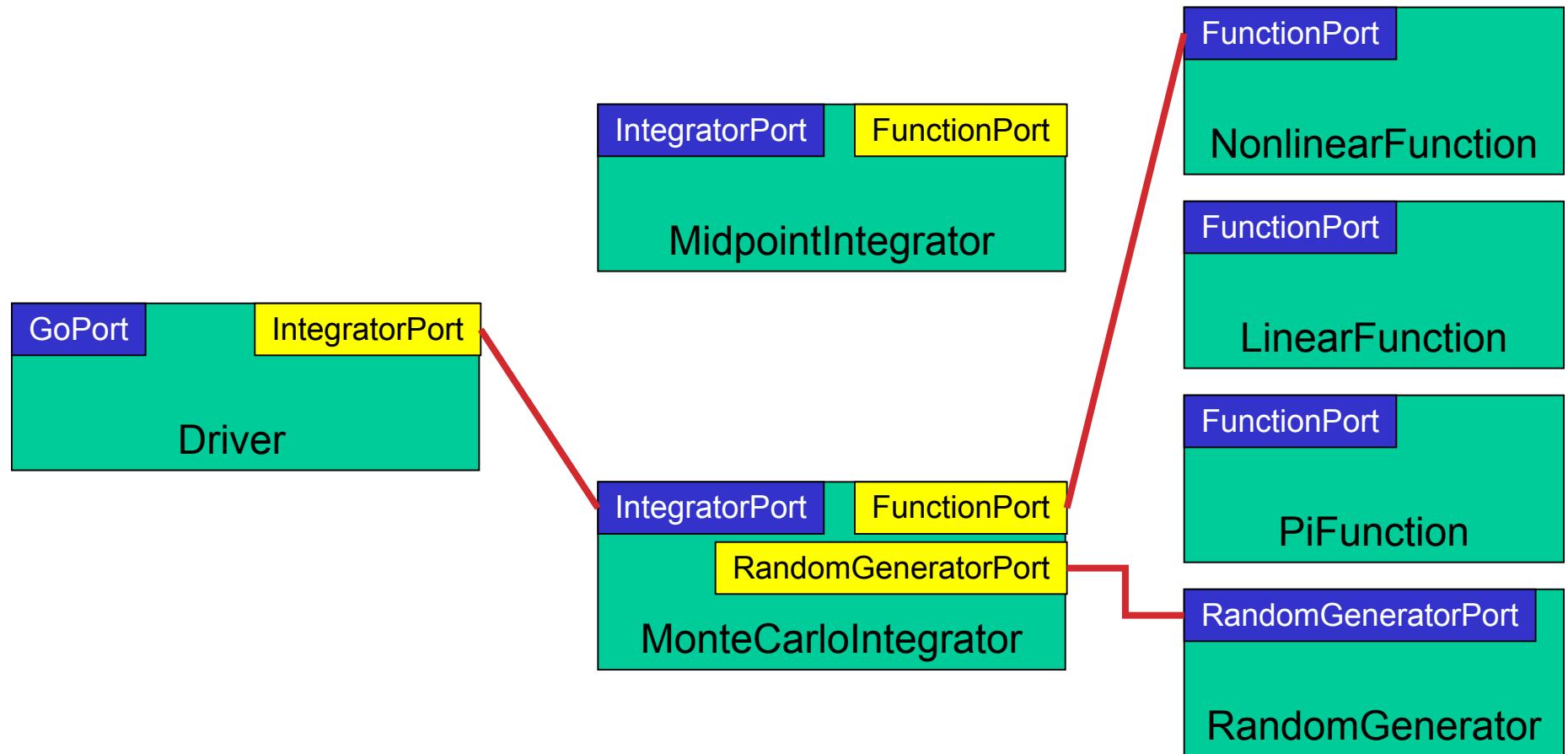
Components and Ports in the Integrator Example



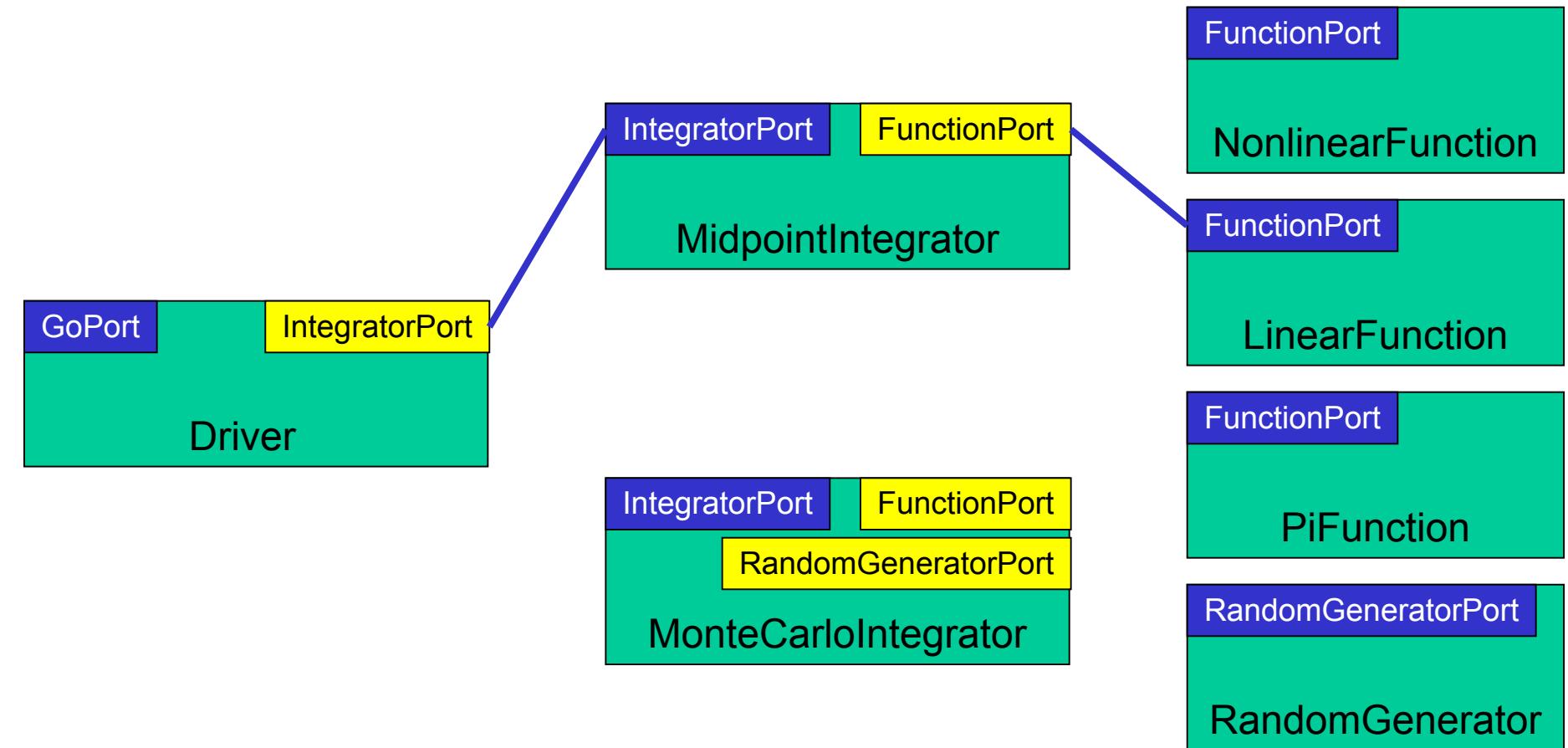
An Application Built from the Example Components



Another Application...

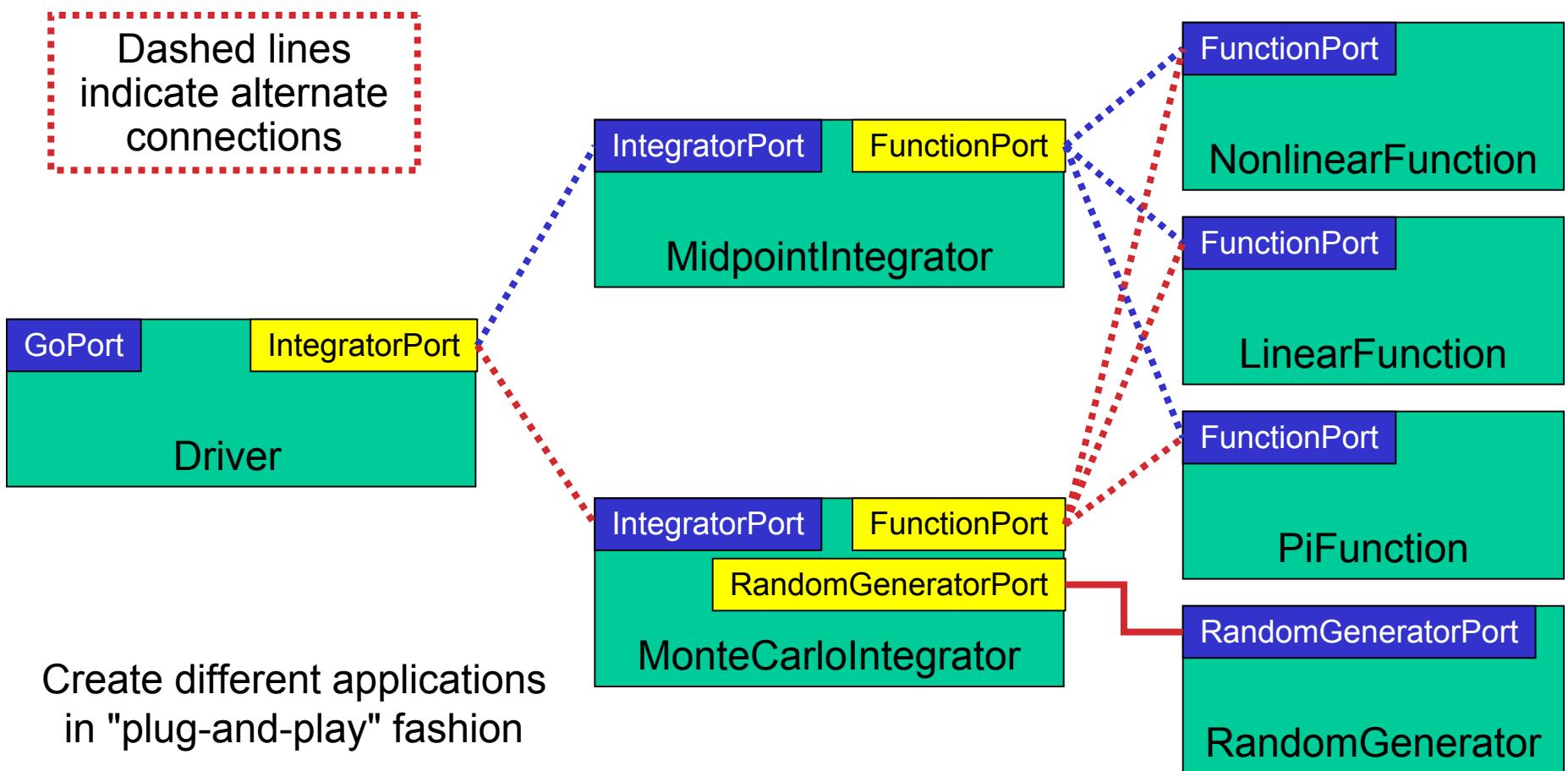


Application 3...



And Many More....

Dashed lines indicate alternate connections



Existing Code → Components

- Component environments **rigorously** enforce interfaces
- Can have **several versions** of a component loaded into a single application
- Component needs add'l code to interact w/ framework
 - Constructor and destructor methods
 - Tell framework what ports it *uses* and *provides*
- Invoking methods on other components requires slight modification to “library” code

Framework interaction
code (*new*)

Integrator library code
(*slightly modified*)

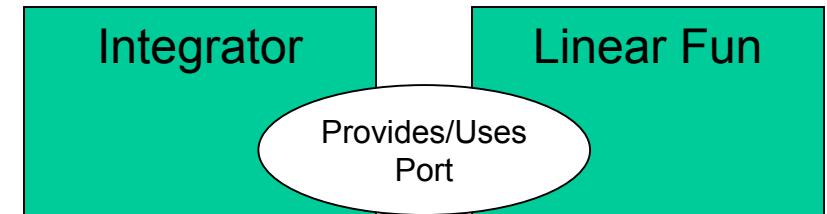
Integrator

Ports, Interoperability, and Reuse

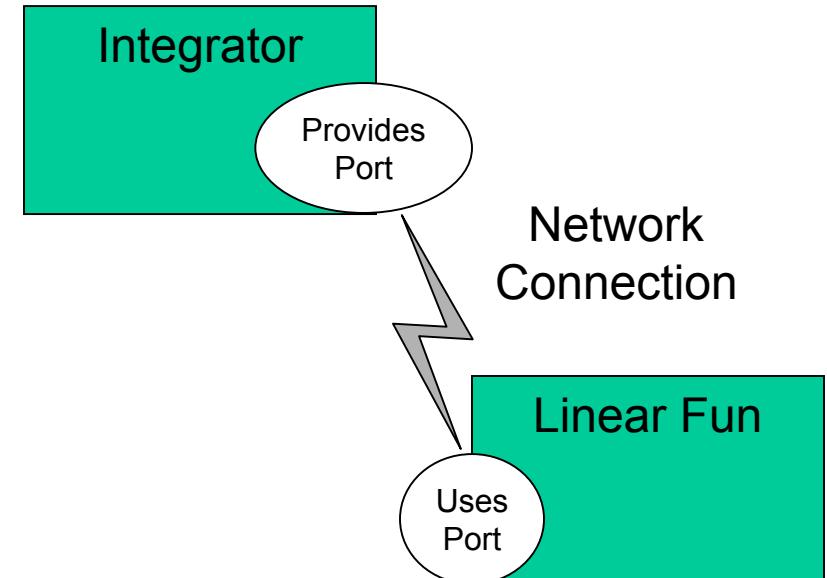
- Ports (interfaces) define how components interact
- Generality, quality, robustness of ports is up to designer/architect
 - “Any old” interface is easy to create, but...
 - Developing a robust domain “standard” interface requires thought, effort, and cooperation
- General “plug-and-play” interoperability of components requires multiple implementations conforming to the same interface
- Designing for interoperability and reuse requires “standard” interfaces
 - Typically domain-specific
 - “Standard” need not imply a formal process, may mean “widely used”

Importance of Provides/Uses Pattern for Ports

- Fences between components
 - Components must **declare** both what they provide and what they use
 - Components **cannot interact** until ports are connected
 - No mechanism to call anything not part of a port
- Ports preserve high performance **direct connection** semantics...
- ...While also allowing **distributed computing**



Direct Connection



Network Connection

“Direct Connection” Maintains Local Performance

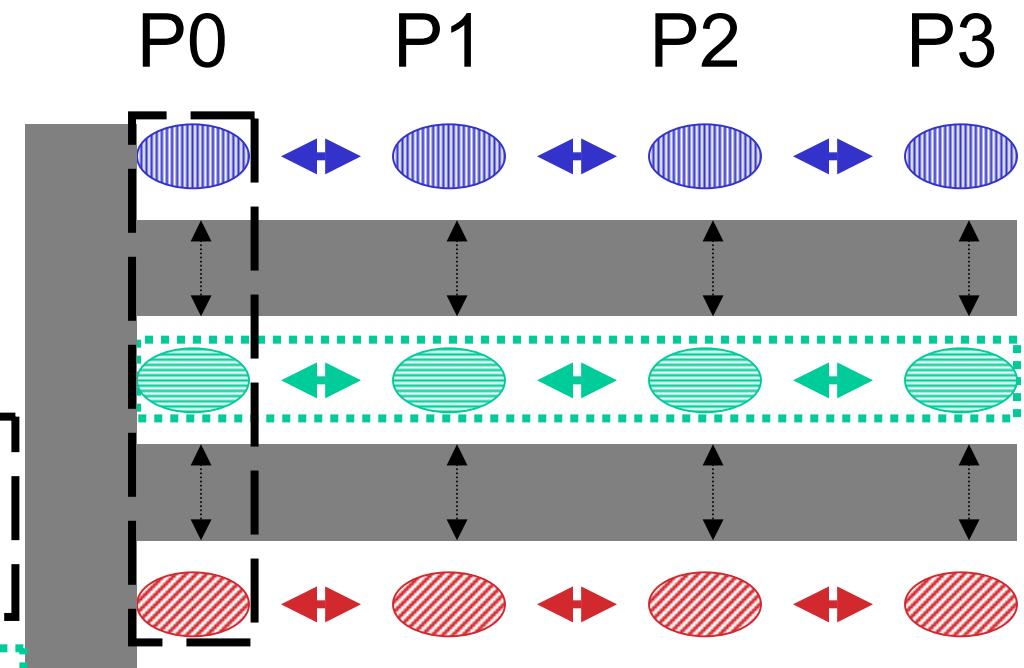
- Components loaded into separate namespaces in the same address space (process) from shared libraries
- `getPort` call returns a pointer to the port's function table
- Calls between components equivalent to a C++ virtual function call: lookup function location, invoke
- Cost equivalent of ~2.8 F77 or C function calls
- All this happens “automatically” – user just sees high performance
- *Description reflects Ccaffeine implementation, but similar or identical mechanisms are in other direct connect fwks*

Framework Stays “Out of the Way” of Component Parallelism

- Single component multiple data (SCMD) model is component analog of widely used SPMD model
- Each process loaded with the same set of components wired the same way

• Different components in same process “talk to each” other via ports and the framework

• ***Same component in different processes talk to each other through their favorite communications layer (i.e. MPI, PVM, GA)***



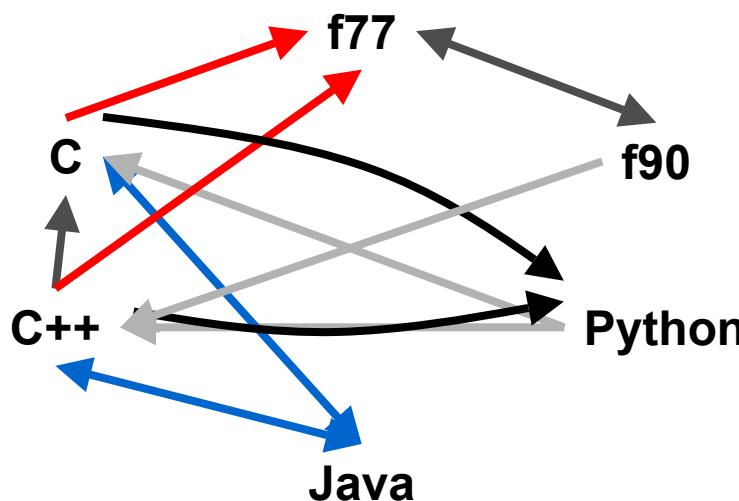
Components: Blue, Green, Red

Framework: Gray

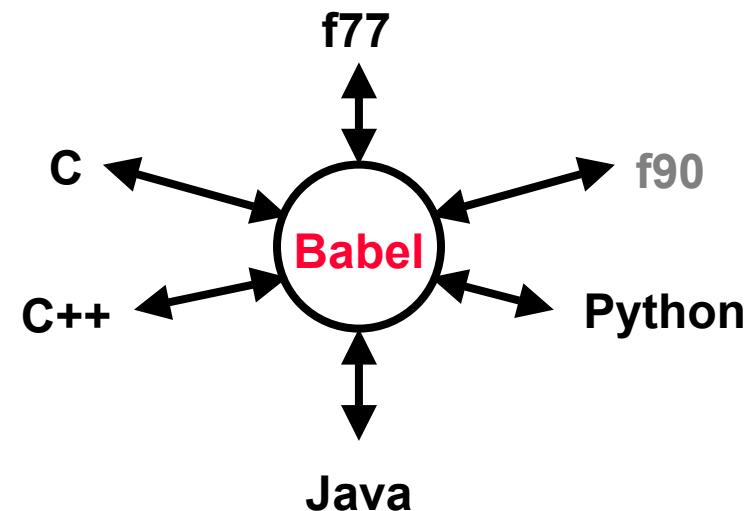
MCMD/MPMD also supported

Language Interoperability

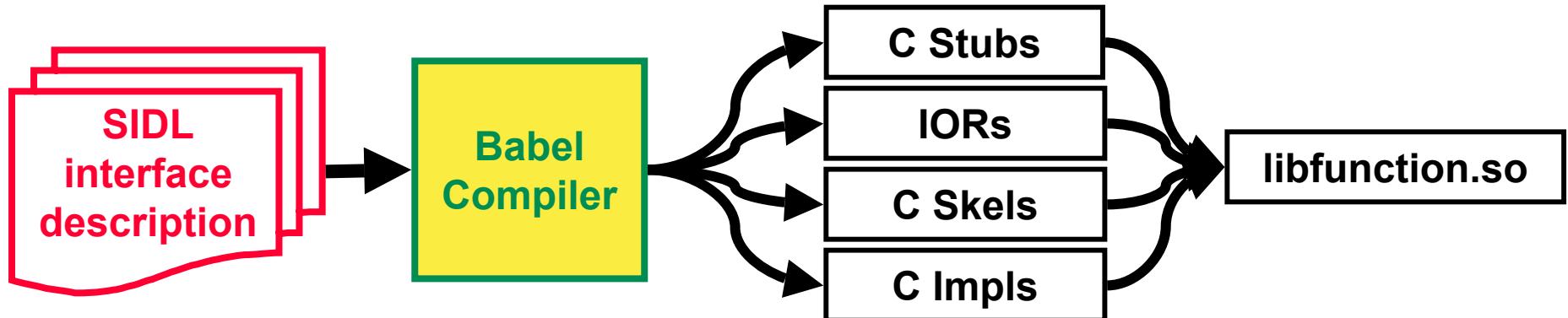
- Existing language interoperability approaches are “**point-to-point**” solutions



- Babel** provides a unified approach in which all languages are considered **peers**
- Babel used primarily at interfaces



Babel Features



- **Babel includes...**
 - Code generator
 - Runtime (linked into CCA framework)
- Implemented using C-based **internal object representation** (IOR)
- **Server side:** wrap implementation in Babel-generated code to allow calling from any language via IOR
- **Client side:** Use SIDL interface definition to generate stubs to call from client language to IOR
- Strives to allow **natural-looking code** in each supported language

Scientific Interface Def. Lang. (SIDL)

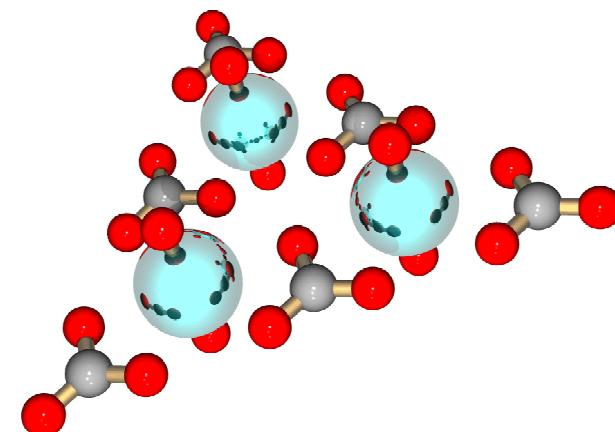
- **Objects:** Interfaces, Abstract Classes, Concrete Classes
- **Methods:** *all public*; virtual, static, final
- **Mode:** in, out, inout (like CORBA)
- **Types:** bool, char, int, long, float, double, fcomplex, dcomplex, array<Type,Dimension>, enum, interface, class

CCA Research Thrusts and Application Domains

- Frameworks
 - Framework interoperability
 - Language interoperability
 - Deployment
- Scientific Components
 - Data Components
 - Linear Algebra
 - Visualization & Steering
 - ...
- MxN Parallel Data Redistribution
 - Component-based
 - Framework-based
- Application Outreach
 - Education
 - Best practices for use
 - Chemistry, Climate
- *Chemistry*
- *Combustion*
- *Climate Modeling*
- *Meshing Tools*
- *(PDE) Solvers*
- Supernova simulation
- Accelerator simulation
- Fusion
- ASCI C-SAFE
- ...

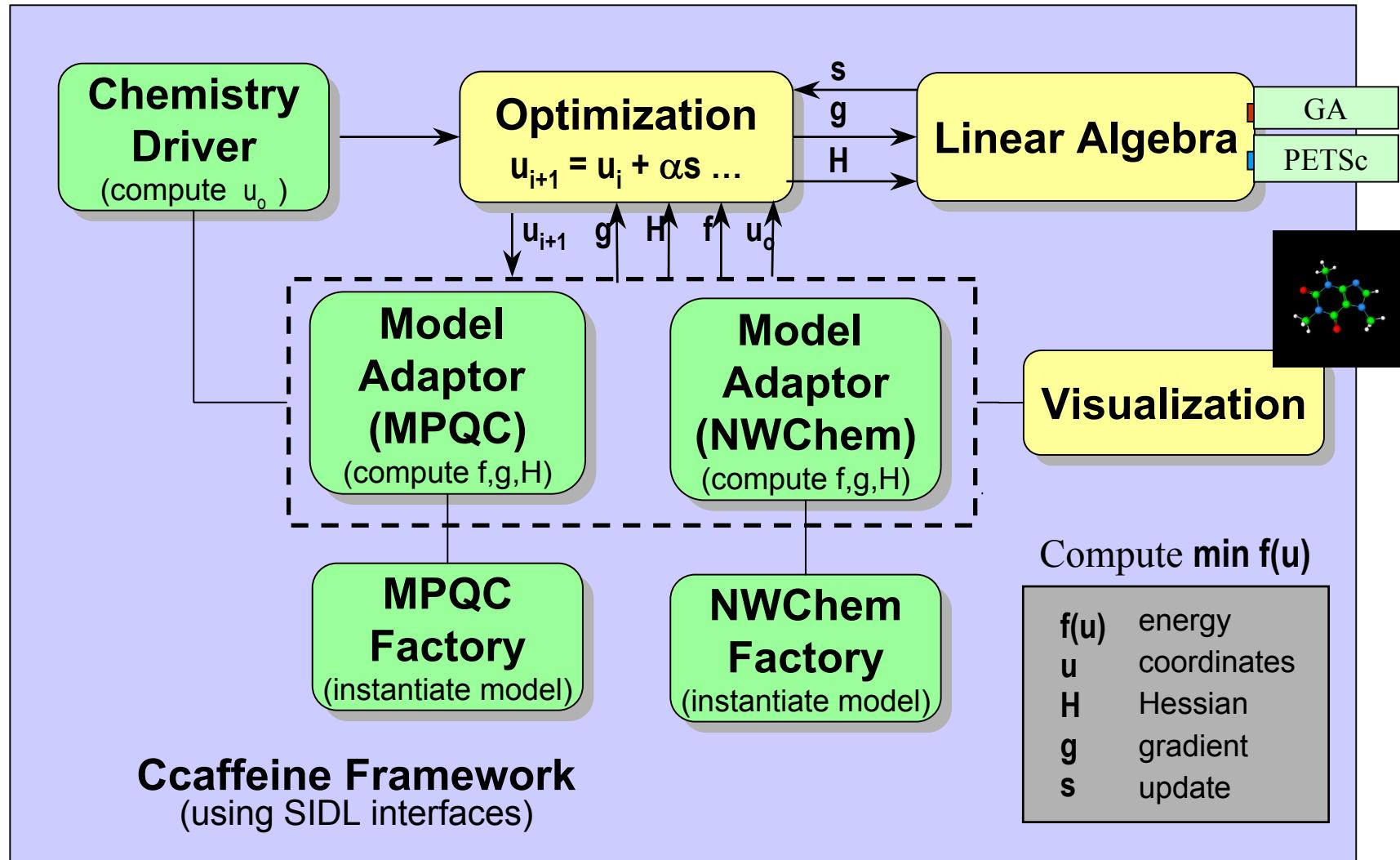
A Basic Chemistry Application

- Decouple geometry optimization from electronic structure
 - Take advantage of modern optimization technology
- Demonstrate interoperability of electronic structure components
 - Can even change packages in mid-optimization
- Software:
 - Electronic structure: MPQC, NWChem,
 - Optimization: TAO,
 - Linear algebra: Global Arrays, PETSC
- Liz Jurrus, Manoj Krishnan, Jarek Nieplocha, Theresa Windus (PNNL)
- Curtis Janssen (SNL)
- Steve Benson, Lois McInnes, Jason Sarich (ANL)
- David Bernholdt (ORNL)



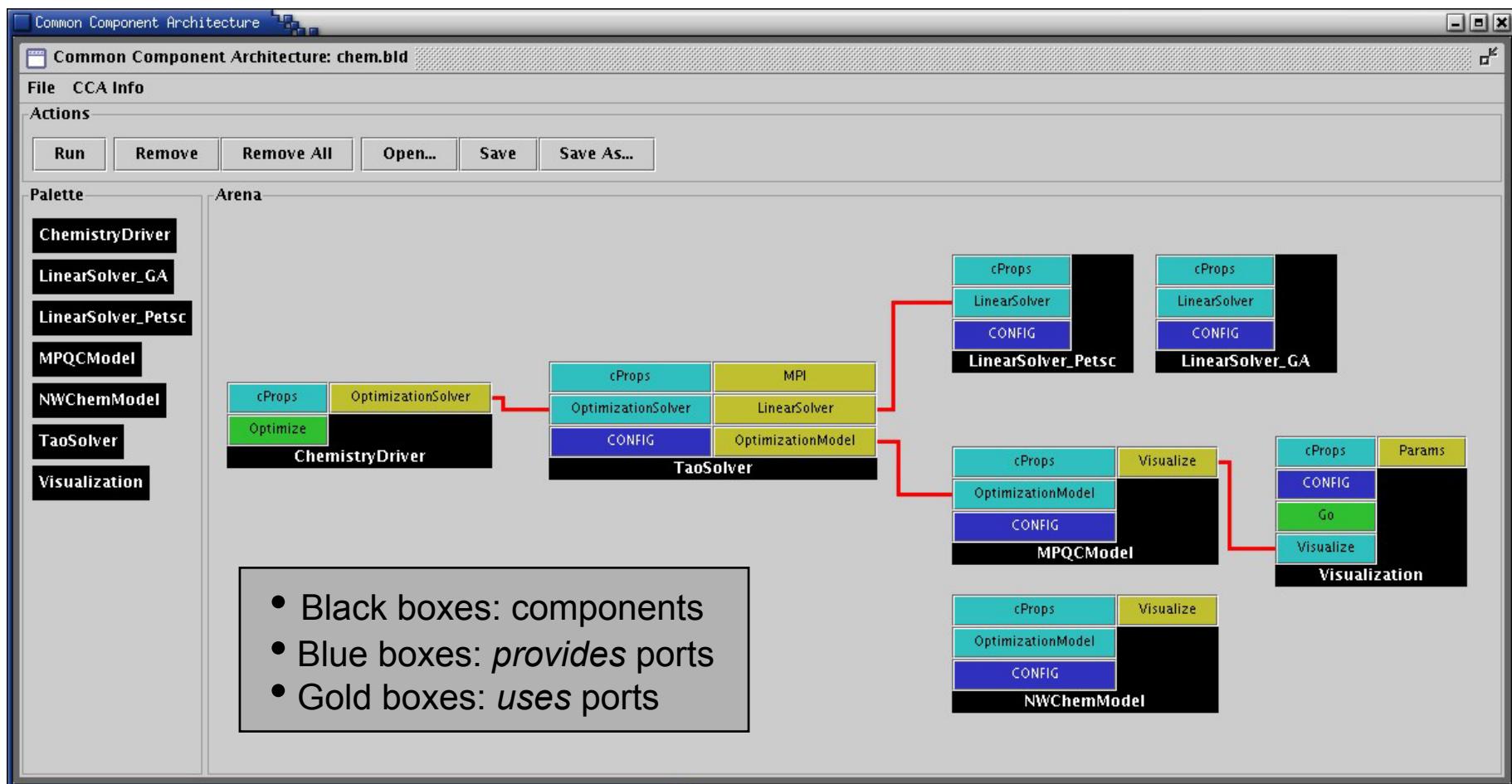
Molecular Geometry Demo

Compute the molecular geometry with minimum energy, i.e. solve $\min f(u)$, where $f: R^n \rightarrow R$.



Component Wiring Diagram

- Electronic structure components based on NWChem (PNNL) and MPQC (SNL)
- Optimization components based on TAO (ANL)
- Linear algebra components based on Global Arrays (PNNL) and PETSc (ANL)



Chemistry: Future Plans

- Use current infrastructure as basis for more sophisticated protein/ligand binding, molecular dynamics codes
- Develop generalized interfaces for electronic structure/reaction dynamics applications
- Explore lower-level interoperability of electronic structure codes
 - Sharing property evaluation capabilities

Synthesis of High-Performance Algorithms for Electronic and Nuclear Structure Calculations

- **David Bernholdt**, Venkatesh Choppella, David Dean, **Robert Harrison**, Thomas Papenbrock, Michael Strayer, Trey White (ORNL)
- **So Hirata** (PNNL)
- Gerald Baumgartner, Daniel Cociorva, Russ Pitzer, P Sadayappan, a small army of graduate students (OSU)
- J Ramanujam (LSU)
- **Marcel Nooijen**, Alexander Auer (Princeton)

Bold = QTP connection

Research at ORNL supported by the Laboratory Directed Research and Development Program

Research at PNNL supported by the Office of Basic Energy Sciences, U. S. Dept. of Energy

Research at OSU, Princeton, and LSU supported by the National Science Foundation Information Technology Research Program

The “Tensor Contraction Engine” (TCE) Concept

- User describes computational problem (tensor contractions, a la many-body methods) in a simple, high-level language
 - Similar to what might be written in papers
- Compiler-like tools translate high-level language into traditional Fortran (or C, or...) code
- Generated code is compiled and linked to libraries providing computational infrastructure

Addressing Programming Challenges

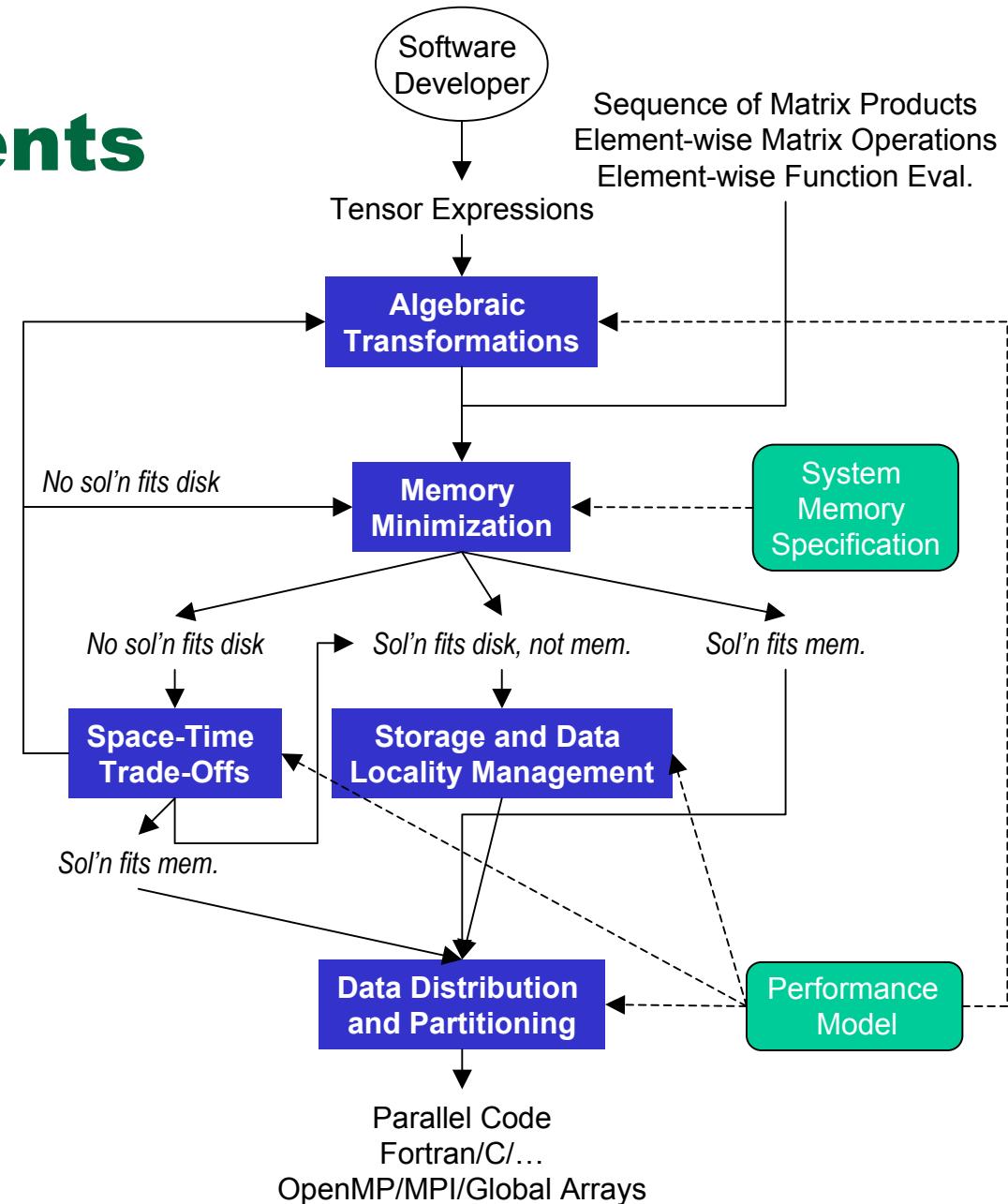
- **Productivity**
 - User writes simple, high-level code
 - Code generation tools do the tedious work
- **Complexity**
 - Significantly reduces complexity visible to programmer
- **Performance**
 - Perform optimizations prior to code generation
 - Automate many decisions humans make empirically
 - Tailor generated code to target computer
 - Tailor generated code to specific problem

So What's New About This Project?

- The creation of “little languages” and code generation tools has a long history in chemistry and other domains
- **Usually viewed only as productivity tools**
 - Imitate what researcher would do – but quicker
- **We treat it as a computer science problem**
 - Similar to (not identical to) an optimizing compiler
 - Algorithmic choices are explored and evaluated rigorously and (in most cases) exhaustively
 - Make use of machine architecture & performance models to specialize generated code to target system
- **Target applications**
 - Rapid experimentation with new many-body methods
 - Implementation of high-complexity methods
 - Improving computational efficiency on parallel machines
 - Also for nuclear physics...

TCE Components

- Algebraic Transformations
 - Minimize operation count
- Memory Minimization
 - Reduce intermediate storage
- Space-Time Transformation
 - Trade-offs btw storage and recomputation
- Storage Management and Data Locality Optimization
 - Optimize use of storage hierarchy
- Data Distribution and Partitioning
 - Optimize parallel layout



TCE Input Language

```
range V = 3000;  
range O = 100;  
  
index a,b,c,d,e,f : V;  
index i,j,k : O;  
  
mlimit = 1000000000000;
```

```
function F1(V,V,V,O);  
function F2(V,V,V,O);
```

```
procedure P(in T1[O,O,V,V], in T2[O,O,V,V], out X)=  
  
begin  
    X == sum[ sum[F1(a,b,f,k) * F2(c,e,b,k), {b,k}]  
              * sum[T1[i,j,a,e] * T2[i,j,c,f], {i,j}],  
              {a,e,c,f}];  
end
```

$$\begin{aligned} A3A = & \frac{1}{2} (X_{ce,af} Y_{ae,cf} + X_{\bar{ce},\bar{af}} Y_{\bar{ae},\bar{cf}} + X_{\bar{ce},\bar{af}} Y_{\bar{ae},cf} \\ & + X_{\bar{ce},af} Y_{ae,\bar{cf}} + X_{\bar{ce},\bar{af}} Y_{\bar{ae},\bar{cf}} + X_{\bar{ce},af} Y_{\bar{ae},\bar{cf}}) \\ X_{ce,af} = & t_{ij}^{ce} t_{ij}^{af} & Y_{ae,cf} = \langle ab \| ek \rangle \langle cb \| fk \rangle \end{aligned}$$

CCSD Doubles Equation

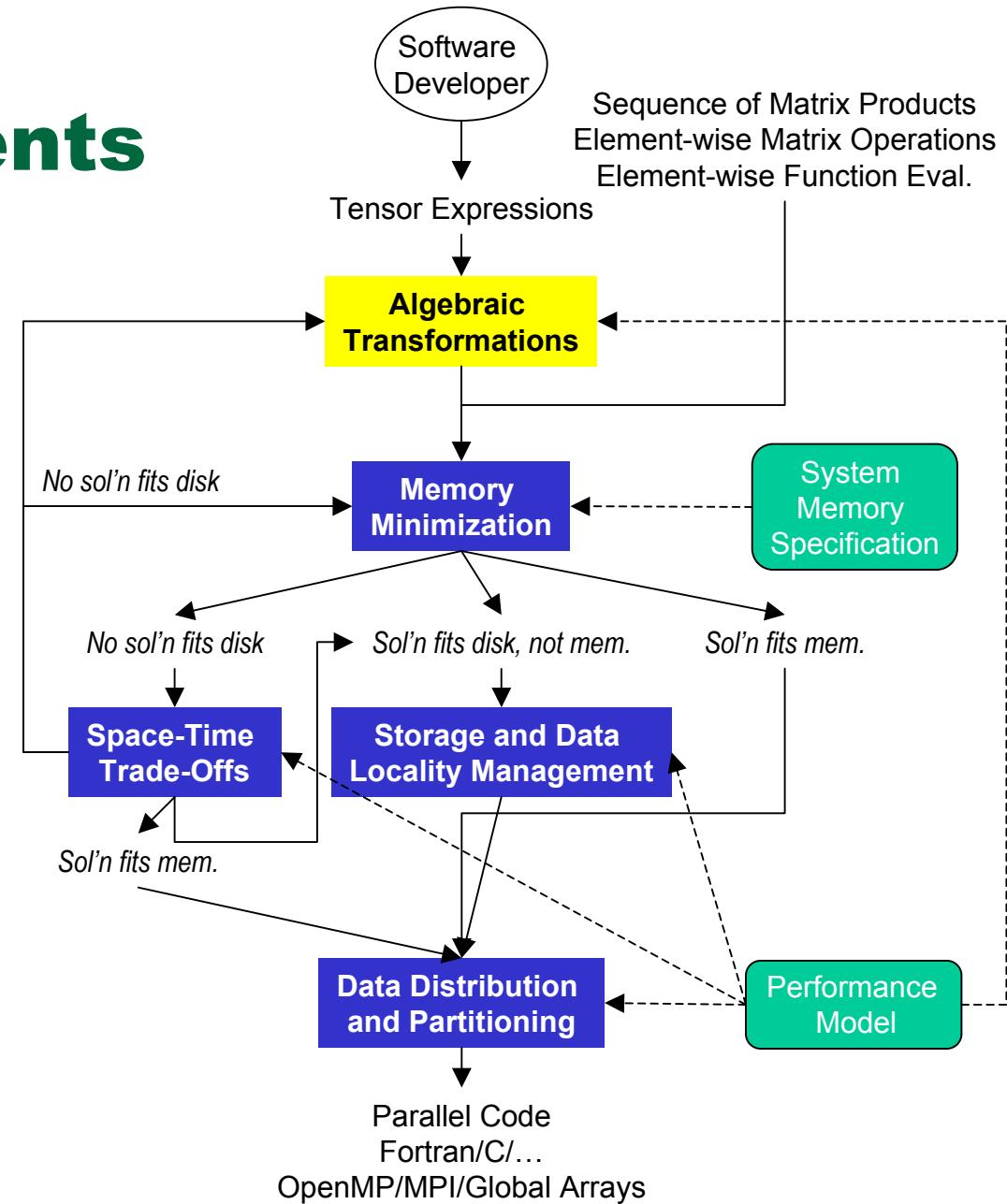
```

hbar[a,b,i,j] == sum[f[b,c]*t[i,j,a,c],{c}] -sum[f[k,c]*t[k,b]*t[i,j,a,c],{k,c}] +sum[f[a,c]*t[i,j,c,b],{c}] -sum[f[k,c]*t[k,a]*t[i,j,c,b],{k,c}] -
sum[f[k,j]*t[i,k,a,b],{k}] -sum[f[k,c]*t[j,k,a,b],{k,c}] -sum[f[k,i]*t[j,k,b,a],{k}] -sum[f[k,c]*t[i,c]*t[j,k,b,a],{k,c}]
+sum[t[i,c]*t[i,d]*v[a,b,c,d],{c,d}] +sum[t[i,j,c,d]*v[a,b,c,d],{c,d}] +sum[t[j,c]*v[a,b,i,c],{c}] -sum[t[k,b]*v[a,k,i,j],{k}]
+sum[t[i,c]*v[b,a,j,c],{c}] -sum[t[k,a]*v[b,k,j,i],{k}] -sum[t[k,d]*t[i,j,c,b]*v[k,a,c,d],{k,c,d}] -sum[t[i,c]*t[j,k,b,d]*v[k,a,c,d],{k,c,d}] -
sum[t[j,c]*t[k,b]*v[k,a,c,i],{k,c}] +2*sum[t[j,k,b,c]*v[k,a,c,i],{k,c}] -sum[t[j,k,c,b]*v[k,a,c,i],{k,c}] -sum[t[i,c]*t[j,d]*v[k,b,a,d,c],{k,c,d}] -
2*sum[t[k,d]*t[i,j,c,b]*v[k,a,d,c],{k,c,d}] -sum[t[i,c]*t[j,k,d,b]*v[k,a,d,c],{k,c,d}] -sum[t[j,k,b,c]*v[k,a,i,c],{k,c}] -
sum[t[i,c]*t[k,b]*v[k,a,j,c],{k,c}] -sum[t[i,k,c,b]*v[k,a,j,c],{k,c}] -sum[t[i,c]*t[j,d]*t[k,a]*v[k,b,c,d],{k,c,d}] -
sum[t[k,d]*t[i,j,a,c]*v[k,b,c,d],{k,c,d}] -sum[t[k,a]*t[i,j,c,d]*v[k,b,c,d],{k,c,d}] +2*sum[t[j,d]*t[i,k,a,c]*v[k,b,c,d],{k,c,d}] -
sum[t[j,d]*t[i,k,c,a]*v[k,b,c,d],{k,c,d}] -sum[t[i,c]*t[j,k,d,a]*v[k,b,c,d],{k,c,d}] -sum[t[i,c]*t[k,a]*v[k,b,c,j],{k,c}]
+2*sum[t[i,k,a,c]*v[k,b,c,j],{k,c}] -sum[t[i,k,c,a]*v[k,b,c,j],{k,c}] +2*sum[t[k,d]*t[i,j,a,c]*v[k,b,d,c],{k,c,d}] -
sum[t[j,d]*t[i,k,a,c]*v[k,b,d,c],{k,c,d}] -sum[t[i,c]*t[k,a]*v[k,b,i,c],{k,c}] -sum[t[j,k,c,a]*v[k,b,i,c],{k,c}] -sum[t[i,k,a,c]*v[k,b,j,c],{k,c}]
+sum[t[i,c]*t[j,d]*t[k,a]*t[l,b]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[k,b]*t[l,d]*t[i,j,a,c]*v[k,l,c,d],{k,l,c,d}] -
2*sum[t[k,a]*t[l,d]*t[i,j,c,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[k,a]*t[l,b]*t[i,j,c,d],{k,l,c,d}] -
2*sum[t[j,c]*t[l,d]*t[i,k,a,b]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[j,d]*t[i,l,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,a]*v[k,l,c,d],{k,l,c,d}] +
sum[t[j,d]*t[i,l,b]*t[k,c,a]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,c]*t[l,d]*t[j,k,b,a]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,a]*v[k,l,b,d]*v[k,l,c,d],{k,l,c,d}]
+sum[t[i,c]*t[l,b]*t[k,c,a]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,d]*t[j,k,l,b,a]*v[k,l,c,d],{k,l,c,d}] +4*sum[t[i,k,a,c]*t[j,l,b,d]*v[k,l,c,d],{k,l,c,d}] -
2*sum[t[i,k,c,a]*t[j,l,b,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,k,a,b]*t[j,l,c,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,k,a,c]*t[j,l,d,b]*v[k,l,c,d],{k,l,c,d}] +
sum[t[i,k,c,a]*t[j,l,d,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,d]*t[k,l,a,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,j,c,d]*t[k,l,a,b]*v[k,l,c,d],{k,l,c,d}] -
2*sum[t[i,j,c,b]*t[k,l,a,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,j,a,c]*t[k,l,b,d]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,c]*t[k,b]*t[l,a]*v[k,l,c,i],{k,l,c}]
+sum[t[i,c]*t[j,k,b,a]*v[k,l,c,i],{k,l,c}] -2*sum[t[l,a]*t[j,k,b,c]*v[k,l,c,i],{k,l,c}] +sum[t[l,a]*t[j,k,c,b]*v[k,l,c,i],{k,l,c}] -
2*sum[t[i,k,c]*t[j,l,b,a]*v[k,l,c,i],{k,l,c}] +sum[t[k,a]*t[j,l,b,c]*v[k,l,c,i],{k,l,c}] +sum[t[k,b]*t[j,l,c,a]*v[k,l,c,i],{k,l,c}]
+sum[t[i,c]*t[j,k,a,b]*v[k,l,c,i],{k,l,c}] +sum[t[i,c]*t[k,a]*t[l,b]*v[k,l,c,j],{k,l,c}] +sum[t[i,c]*t[i,k,a,b]*v[k,l,c,j],{k,l,c}] -
2*sum[t[l,b]*t[i,k,a,c]*v[k,l,c,j],{k,l,c}] +sum[t[i,b]*t[i,k,c,a]*v[k,l,c,j],{k,l,c}] +sum[t[i,c]*t[k,l,a,b]*v[k,l,c,j],{k,l,c}]
+sum[t[j,c]*t[i,d]*t[i,k,a,b]*v[k,l,d,c],{k,l,c,d}] +sum[t[j,d]*t[i,b]*t[i,k,a,c]*v[k,l,d,c],{k,l,c,d}] +sum[t[j,d]*t[i,a]*t[i,k,c,b]*v[k,l,d,c],{k,l,c,d}] -
2*sum[t[i,k,c,d]*t[j,l,b,a]*v[k,l,d,c],{k,l,c,d}] -2*sum[t[i,k,a,c]*t[j,l,b,d]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,c,a]*t[j,l,b,d]*v[k,l,d,c],{k,l,c,d}] +
sum[t[i,k,a,b]*t[j,l,c,d]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,c,b]*t[j,l,d,a]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,a,c]*t[j,l,d,b]*v[k,l,d,c],{k,l,c,d}] +
sum[t[k,a]*t[l,b]*v[k,l,i,j],{k,l}] +sum[t[k,l,a,b]*v[k,l,i,j],{k,l}] +sum[t[k,b]*t[l,d]*t[i,j,a,c]*v[k,l,c,d],{k,l,c,d}] +
sum[t[k,a]*t[l,d]*t[i,j,c,b]*v[i,k,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,d]*t[j,k,b,a]*v[i,k,c,d],{k,l,c,d}] -2*sum[t[i,c]*t[l,a]*t[j,k,b,d]*v[i,k,c,d],{k,l,c,d}] -
sum[t[i,c]*t[i,a]*t[j,k,d,b]*v[i,k,c,d],{k,l,c,d}] +sum[t[i,j,c,b]*t[k,l,a,d]*v[i,k,c,d],{k,l,c,d}] +sum[t[i,j,a,c]*t[k,l,b,d]*v[i,k,c,d],{k,l,c,d}] -
2*sum[t[i,c]*t[i,k,a,b]*v[i,k,c,j],{k,l,c}] +sum[t[i,b]*t[i,k,a,c]*v[i,k,c,j],{k,l,c}] +sum[t[i,a]*t[i,k,c,b]*v[i,k,c,j],{k,l,c}] +v[a,b,i,j]

```

TCE Components

- Algebraic Transformations
 - Minimize operation count
- Memory Minimization
 - Reduce intermediate storage
- Space-Time Transformation
 - Trade-offs btw storage and recomputation
- Storage Management and Data Locality Optimization
 - Optimize use of storage hierarchy
- Data Distribution and Partitioning
 - Optimize parallel layout



Algebraic Transformations: Operation Minimization

$$S(a,b,i,j) = \sum_{c,d,e,f,k,l} A(a,c,i,k)B(b,e,f,l)C(d,f,j,k)D(c,d,e,l)$$

- Requires $4 * N^{10}$ operations if indices $a-l$ have range N
- Using associative, commutative, distributive laws acceptable
- Optimal formula sequence requires only $6 * N^6$ operations

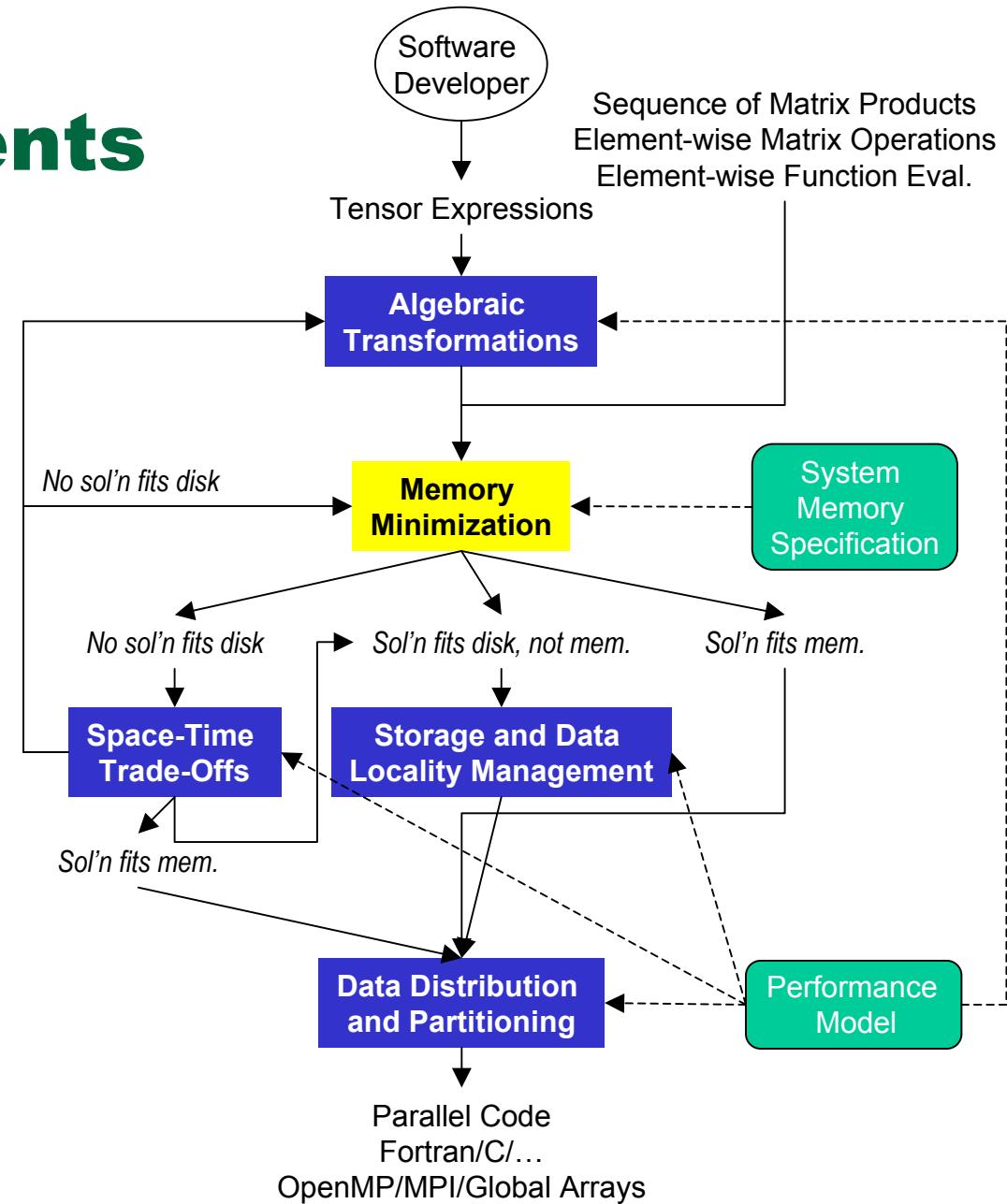
$$T1(b,c,d,f) = \sum_{e,l} B(b,e,f,l)D(c,d,e,l)$$

$$T2(b,c,j,k) = \sum_{d,f} T1(b,c,d,f)C(d,f,j,k)$$

$$S(a,b,i,j) = \sum_{c,k} T2(b,c,j,k)A(a,c,i,k)$$

TCE Components

- Algebraic Transformations
 - Minimize operation count
- Memory Minimization
 - Reduce intermediate storage
- Space-Time Transformation
 - Trade-offs btw storage and recomputation
- Storage Management and Data Locality Optimization
 - Optimize use of storage hierarchy
- Data Distribution and Partitioning
 - Optimize parallel layout



Memory Minimization: Loop Fusion

$$\begin{aligned} T1_{bcdf} &= \sum_{e,l} B_{befl} D_{cdel} \\ T2_{bcjk} &= \sum_{d,f} T1_{bcdf} C_{dfjk} \\ S_{abij} &= \sum_{c,k} T2_{bcjk} A_{acik} \end{aligned}$$

Formula sequence

T1 = 0; T2 = 0; S = 0
 for b, c, d, e, f, l
 [$T1_{bcdf} += B_{befl} D_{cdel}$
 for b, c, d, f, j, k
 [$T2_{bcjk} += T1_{bcdf} C_{dfjk}$
 for a, b, c, i, j, k
 [$S_{abij} += T2_{bcjk} A_{acik}$

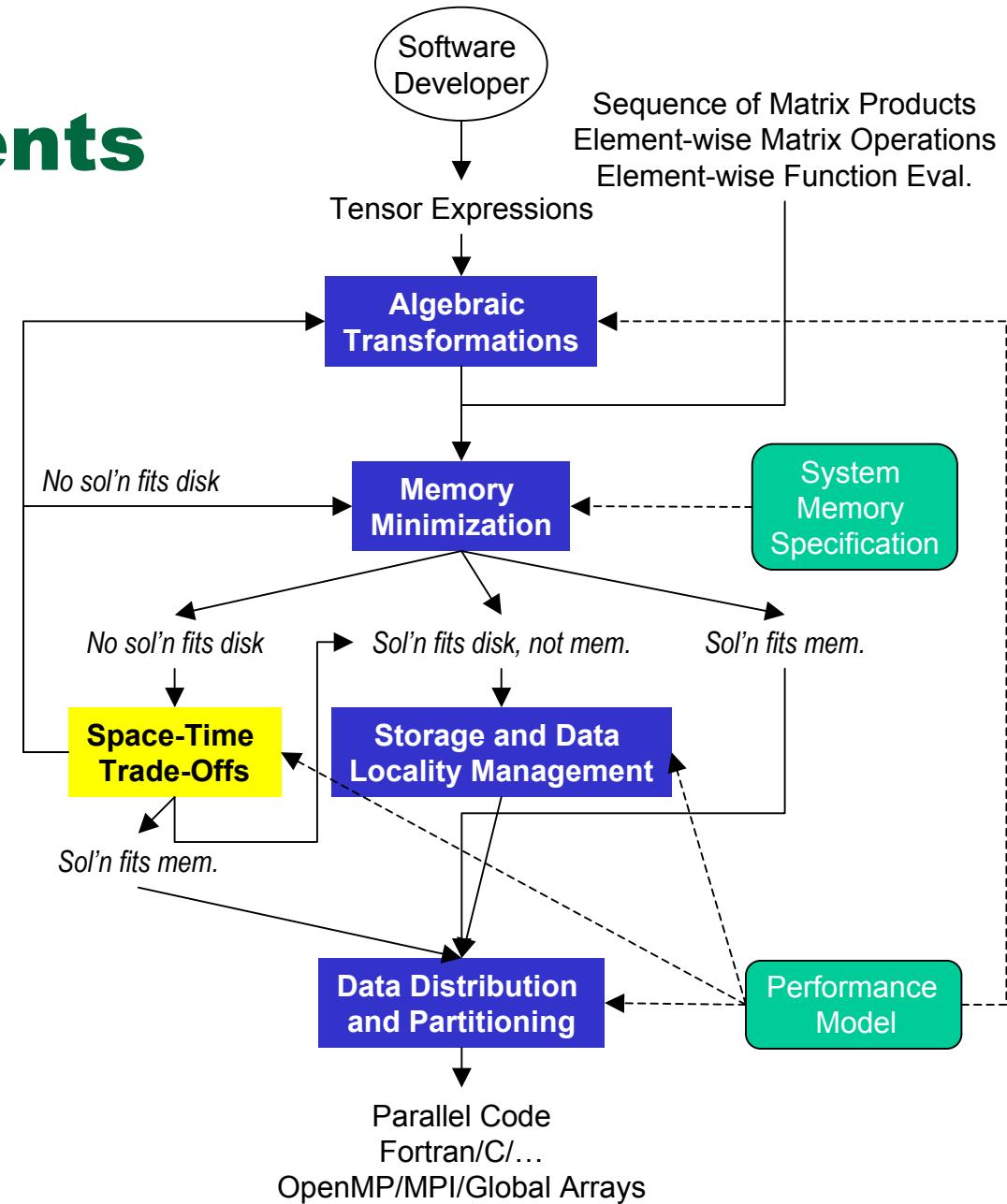
Unfused code

S = 0
 for b, c
 [T1f = 0; T2f = 0
 for d, f
 [for e, l
 [T1f += B_{befl} D_{cdel}
 for j, k
 [T2f_{jk} += T1f C_{dfjk}
 for a, i, j, k
 [S_{abij} += T2f_{jk} A_{acik}

Fused code

TCE Components

- Algebraic Transformations
 - Minimize operation count
- Memory Minimization
 - Reduce intermediate storage
- Space-Time Transformation
 - Trade-offs btw storage and recomputation
- Storage Management and Data Locality Optimization
 - Optimize use of storage hierarchy
- Data Distribution and Partitioning
 - Optimize parallel layout



Space-Time Trade-Off Example

for a, e, c, f

- └ for i, j
 - └ $X_{aecf} += T_{ijae} T_{ijcf}$

for c, e, b, k

- └ $T1_{cebk} = f1(c, e, b, k)$

for a, f, b, k

- └ $T2_{afbk} = f2(a, f, b, k)$

for c, e, a, f

- └ for b, k
 - └ $Y_{ceaf} += T1_{cebk} T2_{afbk}$

for c, e, a, f

- └ $E += X_{aecf} Y_{ceaf}$

array	space	time
X	V^4	$V^4 O^2$
T1	$V^3 O$	$C_{f1} V^3 O$
T2	$V^3 O$	$C_{f2} V^3 O$
Y	V^4	$V^5 O$
E	1	V^4

a .. f: range $V = 1000 .. 3000$
 i .. k: range $O = 30 .. 100$

Memory-Minimal Form

for a, f, b, k

$$\boxed{\quad \quad T2_{afbk} = f2(a, f, b, k)}$$

for c, e

for b, k

$$\boxed{\quad \quad \quad T1_{bk} = f1(c, e, b, k)}$$

for a, f

for i, j

$$\boxed{\quad \quad \quad \quad X += T_{ijae} T_{ijcf}}$$

for b, k

$$\boxed{\quad \quad \quad \quad Y += T1_{bk} T2_{afbk}}$$

$$\boxed{\quad \quad \quad E += X Y}$$

array	space	time
X	1	$V^4 O^2$
T1	VO	$C_{f1} V^3 O$
T2	$V^3 O$	$C_{f2} V^3 O$
Y	1	$V^5 O$
E	1	V^4

a .. f: range $V = 3000$

i .. k: range $O = 100$

Redundant Computation Allows Full Fusion

for a, e, c, f

 for i, j

$X += T_{ijae} T_{ijcf}$

 for b, k

$T1 = f1(c, e, b, k)$

$T2 = f2(a, f, b, k)$

$Y += T1 T2$

$E += X Y$

array	space	time
X	1	$V^4 O^2$
T1	1	$C_{f1} V^5 O$
T2	1	$C_{f2} V^5 O$
Y	1	$V^5 O$
E	1	V^4

Tiling to Reduce Recomputation

for a^t, e^t, c^t, f^t

for a, e, c, f

for i, j

$$X_{aecf} += T_{ijae} T_{ijcf}$$

for b, k

for c, e

$$T1_{ce} = f1(c, e, b, k)$$

for a, f

$$T2_{af} = f2(a, f, b, k)$$

for c, e, a, f

$$Y_{ceaf} += T1_{ce} T2_{af}$$

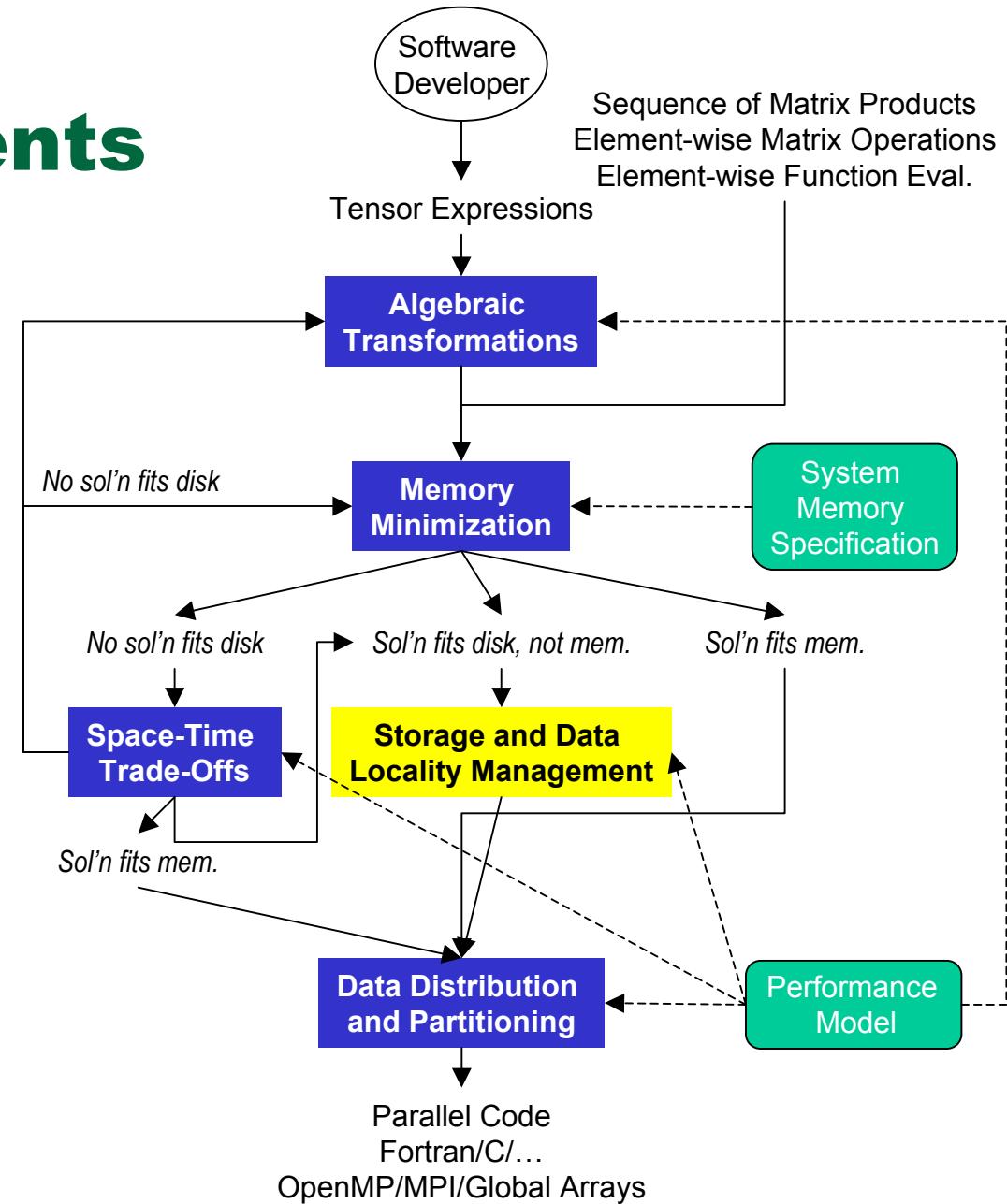
for c, e, a, f

$$E += X_{aecf} Y_{ceaf}$$

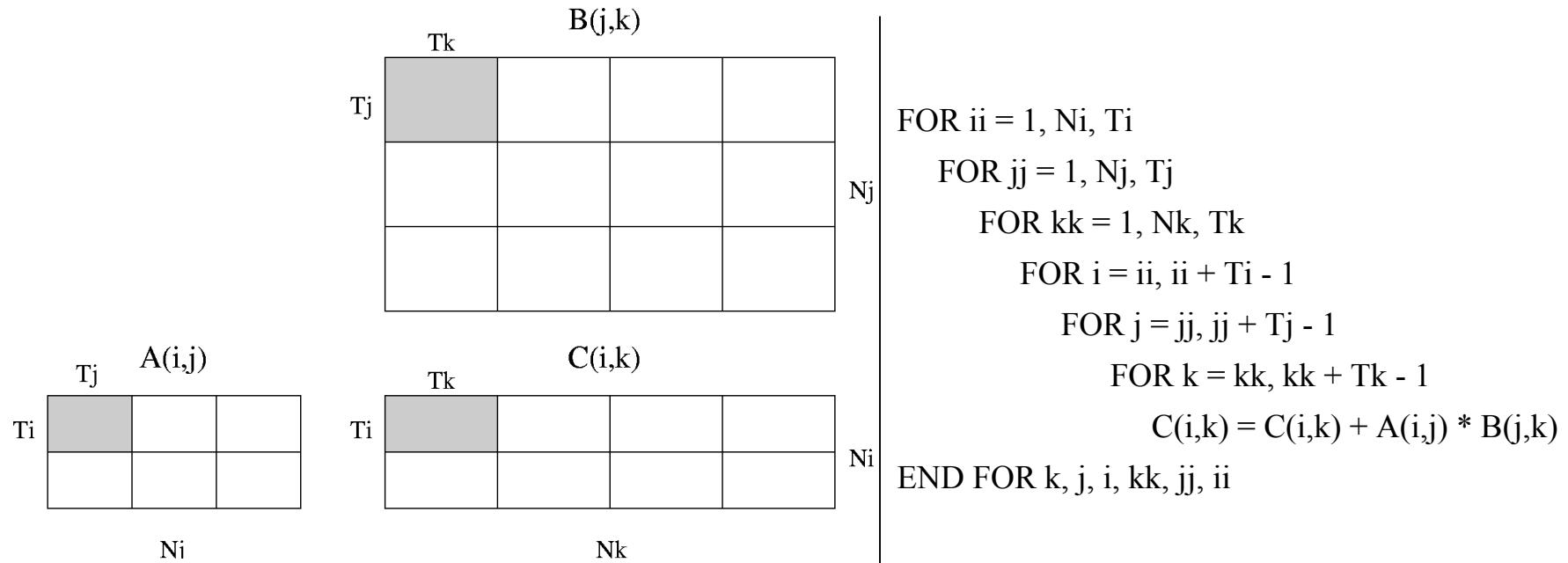
array	space	time
X	B^4	$V^4 O^2$
T1	B^2	$C_{f1}(V/B)^2 V^3 O$
T2	B^2	$C_{f2}(V/B)^2 V^3 O$
Y	B^4	$V^5 O$
E	1	V^4

TCE Components

- Algebraic Transformations
 - Minimize operation count
- Memory Minimization
 - Reduce intermediate storage
- Space-Time Transformation
 - Trade-offs btw storage and recomputation
- Storage Management and Data Locality Optimization
 - Optimize use of storage hierarchy
- Data Distribution and Partitioning
 - Optimize parallel layout



Tiling to Minimize Memory Access Time



Choose T_i , T_j , and T_k such that $T_i * T_j + T_i * T_k + T_j * T_k < \text{cache size}$
 Number of cache misses:

- $A(i,j)$: $N_i * N_j$
- $B(j,k)$: $N_j * N_k * N_i / T_i$
- $C(i,k)$: $N_i * N_k * N_j / T_j$

The TCE in Operation

```

range V = 3000;
range O = 100;

index a,b,c,d,e,f : V;
index i,j,k : O;

mlimit = 1000000000000;

function F1(V,V,V,O);
function F2(V,V,V,O);

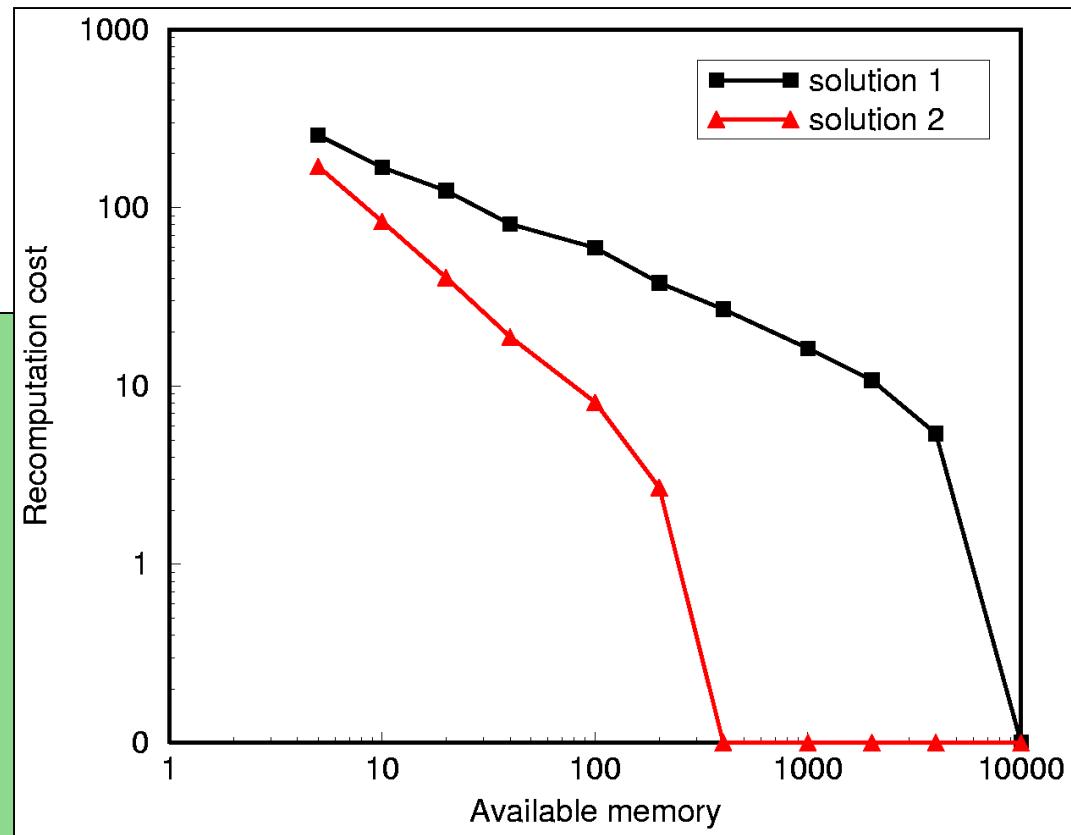
```

procedure P(in T1[O,O,V,V], in T2[O,O,V,V], out X)=

```

begin
  X == sum[ sum[ F1(a,b,f,k) * F2(c,e,b,k), {b,k}]
            * sum[ T1[i,j,a,e] * T2[i,j,c,f], {i,j}],
            {a,e,c,f}];
end

```



$$\begin{aligned}
 A3A &= \frac{1}{2} (X_{ce,af} Y_{ae,cf} + X_{ce,a\bar{f}} Y_{ae,c\bar{f}} + X_{ce,\bar{a}f} Y_{\bar{a}e,cf} \\
 &\quad + X_{ce,a\bar{f}} Y_{ae,\bar{c}\bar{f}} + X_{ce,\bar{a}f} Y_{\bar{a}e,\bar{c}\bar{f}} + X_{ce,\bar{a}\bar{f}} Y_{\bar{a}\bar{e},\bar{c}\bar{f}}) \\
 X_{ce,af} &= t_{ij}^{ce} t_{ij}^{af} \qquad \qquad \qquad Y_{ae,cf} = \langle ab \| ek \rangle \langle cb \| fk \rangle
 \end{aligned}$$

Work in Progress and Planned

- Parallel code generation
- Intermediate identification
 - Greatly increases complexity of operation min.
- Automatic use of permutational symmetry
- Code generation with geometric symmetry
 - May include in cost analysis for extreme calcs.
- Develop approximate algorithms for opt.
 - Address situations where exhaustive search too expensive
 - Deliver best result spending at most 15 min on code gen.
 - Deliver best result spending at most 3 days on code gen.

Summary

Common Component Architecture

- Component model for HPC
- Bringing advantages of emerging software engineering approach to HPC
- Addresses productivity and complexity while remaining performance-neutral
- Highly interdisciplinary project, driven by needs of domain scientists
 - Broad user base

<http://www.cca-forum.org>

Algorithm Synthesis

- Generation of highly-optimized code from high-level domain-specific language
- Addresses productivity, complexity, and performance
- Strong interdisciplinary collaboration between chemists and computer scientists
 - Problem from chemists, solutions from computer scientists (w/ significant help from chemists)

<http://www.cis.ohio-state.edu/~gb/TCE>