A High-Level Approach to the Synthesis of High-Performance Codes for Quantum Chemistry

Louisiana State University
J. Ramanujam

Oak Ridge National Laboratory
David E. Bernholdt
Venkatesh Choppella
Robert J. Harrison
James B. "Trey" White III

Ohio State University
Gerald Baumgartner
Alina Bibireata
Daniel Cociorva
Xioyang Gao
Sriram Krishnamoorthy
Sandhya Krishnan
Chi-Chung Lam
Qingda Lu
Russell M. Pitzer

Ohio State University (cont.)
P. Sadayappan
Alexander Sibiryakov

Pacific Northwest National Laboratory
So Hirata

University of Waterloo
Alexander Auer
Marcel Nooijen

Overview

High-quality ab initio electronic structure methods, based on coupled cluster theory or configuration interaction approaches are among the most accurate and most demanding methods in the field of computational chemistry.

Though the equations, based on contractions of tensors (the multidimensional analog of matrix multiplication) are relatively simple to express, but efficient implementation typically requires months to years of effort. Moreover, different platforms would often benefit from tailored implementations, which are impractical.

Our approach to this problem is to use a high-level domain-specific language coupled with an optimizing compiler capable of tuning the generated code based on information about the problem and hardware environment. Thus, the "Tensor Contraction Engine" (TCE) addresses both developer productivity and program performance issues.

Tensor Contractions and Coupled Cluster Methods

A typical term from the Laplace factorized CCSD(T) method:

$$A3A = \tfrac{1}{2}\big(X_{ce,af}Y_{ae,cf} + X_{\bar{c}\bar{e},\bar{a}f}Y_{\bar{a}\bar{e},\bar{c}\bar{f}} + X_{\bar{c}\bar{e},\bar{a}f}Y_{\bar{a}\bar{e},cf}$$
$$+ X_{ce,a\bar{f}}Y_{ae,c\bar{f}} + X_{\bar{c}e,af}Y_{a\bar{e},\bar{c}f} + X_{\bar{c}\bar{e},\bar{a}\bar{f}}Y_{\bar{a}\bar{e},\bar{c}\bar{f}}\big)$$

$$X_{ce,af} = t_{ij}^{ce}t_{ij}^{af} \qquad Y_{ae,cf} = \langle ab\|ek\rangle\langle cb\|fk\rangle$$

- Indices i, j, k $O(O=100)$ values, a, b, c, e, f $O(V=3000)$
- Term costs $O(OV^5) \approx 10^{19}$ FLOPs; Integrals ~ 1000 FLOPs each
- $O(V^4)$ terms ~ 500 TB memory each

- CC-based methods have **tens to hundreds** of such terms
- *Example:* **CCSD Doubles Equation**
  - CCSD is the *simplest* CC method in routine use
  - This equation, together with the singles equation are iterated until convergence

```
hbafa,b,i,j == sum[fb,c,d*f[a,c,d]] -sum[fb,c,d*f[c,a,d]] +sum[fb,c,d*f[a,c,d]] -sum[fb,c,d*f[c,a,d]] -sum[fb,j*f[a,c,a,b],k]] -
  sum[fb,k*f[a,c,a,b],j]] +sum[fb,k*f[a,c,a,j]] -sum[fb,k*f[c,a,j]] +sum[fb,j*f[a,c,a,k]] +sum[fb,j*f[c,a,k]] -sum[fb,k*f[a,c,i,j]] -
  sum[fb,j*f[a,c,i,k]] +sum[fb,j*f[a,c,i,j]] +sum[fb,k*f[a,c,i,k]] +sum[fb,j*f[a,c,j,k]] -sum[fb,k*f[a,c,j,j]] -sum[fb,j*f[a,c,k,k]] +
  sum[fb,k*f[a,c,k,j]] -sum[fb,j*f[a,c,i,a,b]] +sum[fb,k*f[a,c,i,a,j]] -sum[fb,j*f[a,c,i,a,k]] -sum[fb,k*f[a,c,j,a,j]] +sum[fb,j*f[a,c,j,a,k]] -
  sum[fb,k*f[a,c,k,a,j]] +sum[fb,j*f[a,c,k,a,k]] -sum[fb,j*f[a,c,i,a,b],k]] +sum[fb,k*f[a,c,i,a,b],j]] -sum[fb,j*f[a,c,i,a,j],k]] +
  sum[fb,k*f[a,c,i,a,j],j]] +sum[fb,j*f[a,c,i,a,k],k]] -sum[fb,k*f[a,c,i,a,k],j]] -sum[fb,j*f[a,c,j,a,k],k]] +sum[fb,k*f[a,c,j,a,k],j]] -
  sum[fb,j*f[a,c,k,a,k],k]] +sum[fb,k*f[a,c,k,a,k],j]] -sum[fb,j*f[a,c,i,a,j],i]] +sum[fb,k*f[a,c,i,a,j],k]] +sum[fb,j*f[a,c,j,a,j],i]] -
  sum[fb,k*f[a,c,j,a,j],k]] -sum[fb,j*f[a,c,k,a,j],i]] +sum[fb,k*f[a,c,k,a,j],k]] -sum[fb,j*f[a,c,i,a,k],i]] +sum[fb,k*f[a,c,i,a,k],j]] +
  sum[fb,j*f[a,c,j,a,k],i]] -sum[fb,k*f[a,c,j,a,k],j]] +sum[fb,j*f[a,c,k,a,k],i]] -sum[fb,k*f[a,c,k,a,k],j]] -sum[fb,c,d*f[a,c,d]] -
  sum[fb,c,d*f[c,a,d]] +sum[fb,c,d*f[a,c,d]] -sum[fb,c,d*f[c,a,d]] +2*sum[fb,c,d*f[a,c,d,b],c]] -sum[fb,c,d*f[a,c,d,c],b]] -sum[fb,c,d*f[a,c,d,b],c]] -
  +2*sum[fb,c,d*f[c,a,d,b],c]] -sum[fb,c,d*f[c,a,d,c],b]] +sum[fb,c,d*f[c,a,d,c],b]] +sum[fb,c,d*f[a,c,d,b],c]] -sum[fb,c,d*f[a,c,d,c],b]] -
  sum[fb,c,d*f[c,a,d,c],b]] +sum[fb,c,d*f[a,c,d,b],c]] -sum[fb,c,d*f[a,c,d,c],b]] +2*sum[fb,c,d*f[a,c,d,b],c]] +2*sum[fb,c,d*f[a,c,d,c],b]] -
  +2*sum[fb,c,d*f[c,a,d,b],c]] -sum[fb,c,d*f[c,a,d,c],b]] -sum[fb,c,d*f[a,c,d,b],c]] +sum[fb,c,d*f[a,c,d,c],b]] +sum[fb,c,d*f[a,c,d,b],c]] -
  sum[fb,c,d*f[a,c,d,c],b]] -sum[fb,c,d*f[c,a,d,b],c]] +sum[fb,c,d*f[c,a,d,c],b]] +sum[fb,c,d*f[a,c,d,b],c]] -sum[fb,c,d*f[a,c,d,c],b]] -
  2*sum[fb,c,d*f[a,c,d,b],c]] +2*sum[fb,c,d*f[a,c,d,c],b]] -sum[fb,c,d*f[a,c,d,b],c]] +sum[fb,c,d*f[a,c,d,c],b]] -sum[fb,c,d*f[a,c,d,b],c]] +
  2*sum[fb,c,d*f[a,c,d,c],b]] +sum[fb,c,d*f[a,c,d,b],c]] -sum[fb,c,d*f[a,c,d,c],b]] -2*sum[fb,k,a,c*f[b,k,a,c]] -2*sum[fb,k,a,c*f[b,k,c,a]] +
  2*sum[fb,k,a,c*f[c,k,a,c]] +2*sum[fb,k,a,c*f[c,k,c,a]] -2*sum[fb,k,a,c*f[a,k,a,c]] +2*sum[fb,k,a,c*f[a,k,c,a]] -2*sum[fb,k,a,c*f[b,k,a,c]] +
  2*sum[fb,k,a,c*f[b,k,c,a]] -2*sum[fb,k,a,c*f[c,k,a,c]] -2*sum[fb,k,a,c*f[c,k,c,a]] +2*sum[fb,k,a,c*f[a,k,a,c]] -2*sum[fb,k,a,c*f[a,k,c,a]] -
  2*sum[fb,k,a,c*f[b,k,a,c]] +2*sum[fb,k,a,c*f[b,k,c,a]] +2*sum[fb,k,a,c*f[c,k,a,c]] +2*sum[fb,k,a,c*f[c,k,c,a]] -2*sum[fb,k,a,c*f[a,k,a,c]] +
  2*sum[fb,k,a,c*f[a,k,c,a]] +2*sum[fb,k,a,c*f[b,k,a,c]] -2*sum[fb,k,a,c*f[b,k,c,a]] -2*sum[fb,k,a,c*f[c,k,a,c]] -2*sum[fb,k,a,c*f[c,k,c,a]] +
  2*sum[fb,k,a,c*f[a,k,a,c]] -2*sum[fb,k,a,c*f[a,k,c,a]] +2*sum[fb,k,a,c*f[b,k,a,c]] -2*sum[fb,k,a,c*f[b,k,c,a]] +2*sum[fb,k,a,c*f[c,k,a,c]] +
  2*sum[fb,k,a,c*f[c,k,c,a]] -2*sum[fb,k,a,c*f[a,k,a,c]] +2*sum[fb,k,a,c*f[a,k,c,a]] +2*sum[fb,k,a,c*f[b,k,a,c]] -2*sum[fb,k,a,c*f[b,k,c,a]] -
  2*sum[fb,k,a,c*f[c,k,a,c]] -2*sum[fb,k,a,c*f[c,k,c,a]] +2*sum[fb,k,a,c*f[a,k,a,c]] -2*sum[fb,k,a,c*f[a,k,c,a]] -2*sum[fb,k,a,c*f[b,k,a,c]] +
  2*sum[fb,k,a,c*f[b,k,c,a]] +2*sum[fb,k,a,c*f[c,k,a,c]] +2*sum[fb,k,a,c*f[c,k,c,a]] -2*sum[fb,k,a,c*f[a,k,a,c]] +2*sum[fb,k,a,c*f[a,k,c,a]] +
  2*sum[fb,k,a,c*f[b,k,a,c]] -2*sum[fb,k,a,c*f[b,k,c,a]] -2*sum[fb,k,a,c*f[c,k,a,c]] -2*sum[fb,k,a,c*f[c,k,c,a]] +2*sum[fb,k,a,c*f[a,k,a,c]] -
  2*sum[fb,k,a,c*f[a,k,c,a]] +2*sum[fb,k,a,c*f[b,k,a,c]] -2*sum[fb,k,a,c*f[b,k,c,a]] -2*sum[fb,k,a,c*f[c,k,a,c]] -2*sum[fb,k,a,c*f[c,k,c,a]] +
  2*sum[fb,k,a,c*f[a,k,a,c]] -2*sum[fb,k,a,c*f[a,k,c,a]] +2*sum[fb,k,a,c*f[b,k,a,c]] -2*sum[fb,k,a,c*f[b,k,c,a]] +2*sum[fb,k,a,c*f[c,k,a,c]] +
  2*sum[fb,k,a,c*f[c,k,c,a]] -2*sum[fb,k,a,c*f[a,k,a,c]] +2*sum[fb,k,a,c*f[a,k,c,a]] +2*sum[fb,k,a,c*f[b,k,a,c]] -2*sum[fb,k,a,c*f[b,k,c,a]] -
  2*sum[fb,k,a,c*f[c,k,a,c]] -2*sum[fb,k,a,c*f[c,k,c,a]] +2*sum[fb,k,a,c*f[a,k,a,c]] -2*sum[fb,k,a,c*f[a,k,c,a]] -2*sum[fb,k,a,c*f[b,k,a,c]] +
  2*sum[fb,k,a,c*f[b,k,c,a]] +2*sum[fb,k,a,c*f[c,k,a,c]] +2*sum[fb,k,a,c*f[c,k,c,a]] -2*sum[fb,k,a,c*f[a,k,a,c]] +2*sum[fb,k,a,c*f[a,k,c,a]] +
  2*sum[fb,k,a,c*f[b,k,a,c]] -2*sum[fb,k,a,c*f[b,k,c,a]] -2*sum[fb,k,a,c*f[c,k,a,c]] -2*sum[fb,k,a,c*f[c,k,c,a]] +2*sum[fb,k,a,c*f[a,k,a,c]] -
  2*sum[fb,k,a,c*f[a,k,c,a]] +2*sum[fb,k,a,c*f[b,k,a,c]] -2*sum[fb,k,a,c*f[b,k,c,a]] -2*sum[fb,k,a,c*f[c,k,a,c]] -2*sum[fb,k,a,c*f[c,k,c,a]] +
  +sum[fb,c,d*f[b,c,d]] -sum[fb,c,d*f[c,b,d]] -sum[fb,c,d*f[b,c,d]] +sum[fb,c,d*f[c,b,d]] +2*sum[fb,c,d*f[b,c,d,a],c]] -2*sum[fb,c,d*f[b,c,d,c],a]] -
  +sum[fb,c,d*f[c,b,d,a],c]] -sum[fb,c,d*f[c,b,d,c],a]] -2*sum[fb,c,d*f[a,c,d,a],c]] +2*sum[fb,c,d*f[a,c,d,c],a]] -2*sum[fb,c,d*f[b,c,d,a],c]] +
  +sum[fb,c,d*f[b,c,d,c],a]] +sum[fb,c,d*f[c,b,d,a],c]] -sum[fb,c,d*f[c,b,d,c],a]] +2*sum[fb,c,d*f[a,c,d,a],c]] -2*sum[fb,c,d*f[a,c,d,c],a]] -
  +sum[fb,c,d*f[a,c,d,c],a]] -sum[fb,c,d*f[b,c,d,a],c]] +sum[fb,c,d*f[b,c,d,c],a]] +sum[fb,c,d*f[c,b,d,a],c]] -sum[fb,c,d*f[c,b,d,c],a]] -
  2*sum[fb,c,d*f[a,c,d,a],c]] +2*sum[fb,c,d*f[a,c,d,c],a]] -sum[fb,c,d*f[b,c,d,a],c]] +sum[fb,c,d*f[b,c,d,c],a]] -sum[fb,c,d*f[c,b,d,a],c]] +
  +sum[fb,c,d*f[c,b,d,c],a]] +sum[fb,c,d*f[a,c,d,a],c]] -sum[fb,c,d*f[a,c,d,c],a]] -sum[fb,c,d*f[b,c,d,a],c]] +sum[fb,c,d*f[b,c,d,c],a]] -
  2*sum[fb,c,d*f[c,b,d,a],c]] +2*sum[fb,c,d*f[c,b,d,c],a]] +sum[fb,c,d*f[a,c,d,a],c]] -sum[fb,c,d*f[a,c,d,c],a]] -sum[fb,c,d*f[b,c,d,a],c]] +
  +sum[fb,c,d*f[b,c,d,c],a]] -sum[fb,c,d*f[c,b,d,a],c]] +sum[fb,c,d*f[c,b,d,c],a]] +sum[fb,c,d*f[a,c,d,a],c]] -sum[fb,c,d*f[a,c,d,c],a]] -
  2*sum[fb,c,d*f[b,c,d,a],c]] +2*sum[fb,c,d*f[b,c,d,c],a]] +sum[fb,c,d*f[c,b,d,a],c]] -sum[fb,c,d*f[c,b,d,c],a]] +sum[fb,c,d*f[a,c,d,a],c]] -
  +sum[fb,c,d*f[a,c,d,c],a]] +sum[fb,c,d*f[b,c,d,a],c]] -sum[fb,c,d*f[b,c,d,c],a]] -sum[fb,c,d*f[c,b,d,a],c]] +sum[fb,c,d*f[c,b,d,c],a]] -
  2*sum[fb,c,d*f[a,c,d,a],c]] +2*sum[fb,c,d*f[a,c,d,c],a]] -sum[fb,c,d*f[b,c,d,a],c]] +sum[fb,c,d*f[b,c,d,c],a]] -sum[fb,c,d*f[c,b,d,a],c]] +
  +sum[fb,c,d*f[c,b,d,c],a]] +sum[fb,c,d*f[a,c,d,a],c]] -sum[fb,c,d*f[a,c,d,c],a]] -sum[fb,c,d*f[b,c,d,a],c]] +sum[fb,c,d*f[b,c,d,c],a]] -
  2*sum[fb,c,d*f[c,b,d,a],c]] +2*sum[fb,c,d*f[c,b,d,c],a]] +sum[fb,b,k*f[k,a,c*f[a,c]]] +sum[fb,b,k*f[k,a,c*f[c,a]]] +*[fb,b,i]
```

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…

Funding

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

Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the US Dept. of Energy under contract DE-AC-05-00OR22725.

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

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

The Tensor Contraction Engine

- 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

Operation Minimal Form

Algebraic transformations to minimize operation count

```
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
```

Inputs

External function calls

Output

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

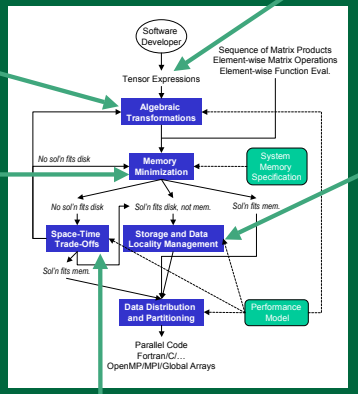
a .. f: range V = 1000 .. 3000
i .. k: range O = 30 .. 100

Memory Minimal Form

Fusion of loops allows reduction of rank of arrays while **preserving cost**

```
for a, f, b, k
  T2_afbk = f2(a, f, b, k)
for c, e
  for b, k
    T1_bk = f1(c, e, b, k)
  for a, f
    for i, j
      X += T_ijae T_ijcf
    for b, k
      Y += T1_bk T2_afbk
  E += X Y
```

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



Space-Time Trade-Offs

Full fusion comes at the cost of increased computation

```
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^4O^2
T1	1	$C_{f1}V^5O$
T2	1	$C_{f2}V^5O$
Y	1	V^5O
E	1	V^4

Tiling provides a ***controlled*** compromise between minimal operations and minimal memory (full fusion)

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

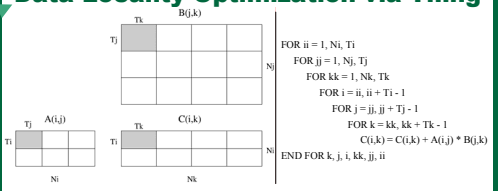
A Language for Tensor Contractions

```
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
```

$$A3A = \tfrac{1}{2}\big(X_{ce,af}Y_{ae,cf} + X_{\bar{c}\bar{e},\bar{a}f}Y_{\bar{a}\bar{e},\bar{c}\bar{f}} + X_{\bar{c}\bar{e},\bar{a}f}Y_{\bar{a}\bar{e},cf}$$
$$+ X_{ce,a\bar{f}}Y_{ae,c\bar{f}} + X_{\bar{c}e,af}Y_{a\bar{e},\bar{c}f} + X_{\bar{c}\bar{e},\bar{a}\bar{f}}Y_{\bar{a}\bar{e},\bar{c}\bar{f}}\big)$$

$$X_{ce,af} = t_{ij}^{ce}t_{ij}^{af} \qquad Y_{ae,cf} = \langle ab\|ek\rangle\langle cb\|fk\rangle$$

Data Locality Optimization via Tiling




```
FOR ii = 1, Ni, Ti
  FOR jj = 1, Nj, Tj
    FOR kk = 1, Nk, Tk
      FOR i = ii, ii + Ti - 1
        FOR j = jj, jj + Tj - 1
          FOR k = kk, kk + Tk - 1
            C(i,k) = C(i,k) + A(i,j) * B(j,k)
END FOR k, j, i, kk, jj, ii
```

Choose Ti, Tj, and Tk such that Ti * Tj + Ti * Tk + Tj * Tk < cache size

Number of cache misses:

- A(i,j): Ni * Nj
- B(j,k): Nj * Nk * Ni/Ti
- C(i,k): Ni * Nk * Nj/Tj

Same algorithm used to manage disk/memory locality or memory/cache locality

Current TCE Capabilities and Uses

Capability	Prototype TCE	Optimizing TCE
Basic sequential code generation for CC-based methods	Yes	Yes
QC Packages Interfaced:		
• File based	NWChem, UTChem	NWChem
• General (file, memory, direct)		Under development
Symmetry Support:		
• Spin	Spin orbitals	General, in progress
• Spatial	Abelian	Abelian, in progress
• Permutational	Fermions	General, in progress
Optimizations:		
• Operation Minimization	Partial	Yes
• Memory Minimization	Partial	Yes
• Space-Time Transformation	No	Yes
• Data Locality	Partial	Yes
Parallel code generation	Limited general	General, in progress

Prototype TCE

- Parallel
  - MBPT(2), MBPT(3), MBPT(4)
  - CCD, CCSD, CCSD(T), QCISD, CCSDT, CCSDTQ
  - CISD, CISDT, CISDTQ
  - CCSD, CCSDT, CCSDTQ lambda equations
  - CCSD, CCSDT, CCSDTQ dipole moments
  - Parallel EOM-CCSD
  - Local/AO-based CCSD
- Sequential
  - Relativistic 2/4-component CI, CC, & MBPT

Optimizing TCE

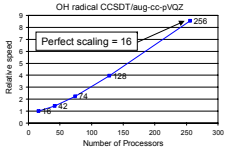
- *Parallel (limited)*
  - *CCD, CCSD*

Code availability

- Prototype TCE to be released with NWChem 4.5
- Optimizing TCE still under intense development
- Some prototype-generated implementations to appear in …
  - NWChem 4.5
  - UTChem 2003

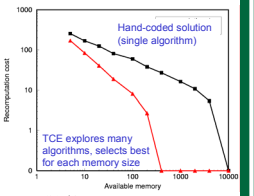
Early Performance Results

Parallel Scalability of Prototype TCE-Generated Code



Courtesy of So Hirata, PNNL. Obtained on the HP Supercluster at PNNL's Molecular Science Computing Facility

Effect of Space-Time Trade-Off Optimization



$$A3A = \tfrac{1}{2}\big(X_{ce,af}Y_{ae,cf} + X_{\bar{c}\bar{e},\bar{a}f}Y_{\bar{a}\bar{e},\bar{c}\bar{f}} + X_{\bar{c}\bar{e},\bar{a}f}Y_{\bar{a}\bar{e},cf}$$
$$+ X_{ce,a\bar{f}}Y_{ae,c\bar{f}} + X_{\bar{c}e,af}Y_{a\bar{e},\bar{c}f} + X_{\bar{c}\bar{e},\bar{a}\bar{f}}Y_{\bar{a}\bar{e},\bar{c}\bar{f}}\big)$$

$$X_{ce,af} = t_{ij}^{ce}t_{ij}^{af} \qquad Y_{ae,cf} = \langle ab\|ek\rangle\langle cb\|fk\rangle$$

On the Drawing Board…

- More flexibility in sequencing and controlling optimizations
- Common sub-expression elimination
- Global factorization (across equations)
  - Complex problem
- Improving parallel code generation
  - Multi-level parallelism
    - Threads
    - Multiple loosely coupled tasks
- More sophisticated performance models
- Develop approximate algorithms for opt.
  - Address situations where exhaustive search too expensive
    - i.e. Deliver best result spending at most 3 min on code gen.
    - … or 60 min … or 3 days …
- Generalizations beyond electronic structure

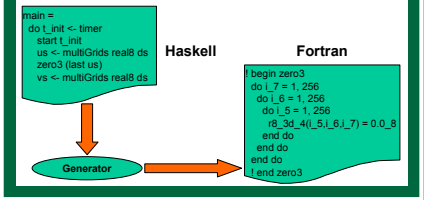
Summary

- Automatic generation of code from high-level algebraic expressions
  - Approach problem like a compiler
  - Use of "high-level language" allows automation of design decisions usually made by human software developer
  - Produce robust, reliable code
- Addresses challenges of modern HPC software development
  - ***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
- Strong interdisciplinary collaboration between chemists and computer scientists
  - Problem from chemists, solutions from computer scientists (w/ significant help from chemists)

Program Generation via Monadic Embedded Meta-Programming

James B. White III and Venkatesh Choppella

- Meta-language is the functional programming language Haskell
- Target language (Fortran) is a datatype in the meta language
- High-level optimizations implemented using higher-order functions (monads)
- Advantages:
  - Leverages the full power of the meta language in defining target language programs. Mixes evaluation, code generation, and macro expansion
  - Integrates application, library, and compiler development
  - Allows expression of high-level domain-specific optimizations

Application: NAS Parallel MultiGrid Benchmark

- Automatic synthesis of multigrid code
- Optimization of temporary usage specified using monads
- Output is optimized Fortran code
- Implementation
  - ***Main multigrid algorithm: ~60 lines***
  - ***Stencil operators: a few 10s of lines***
  - Generating and representing Fortran datatype: ~2000 lines
  - ***Generated code: 6000 lines***