Coupled-Cluster

Andrey Asadchev

Iowa State University

October 26, 2011

Outline

- Tensor Contraction
- Coupled Cluster
- Programming

Tensor Contraction

- tensor "is a" multidimensional array, G^{ij}_{kl} eg $G(i,j,k,l)$
- vectors and matrixes are tensors
- contraction is a multiplication: rank can be lower, same, higher after contraction

Einstein notation is a compact way to represent contractions:
inner product: $a = x^i y_i$
outer product: $t^i_j = x^i y_j$
matrix product: $t^i_j = a^i_k b^k_j$
transpose: $t^i_j = a^j_i$
integral transformation: $v^{ij}_{kl} = C^i_p C^j_r C^q_k C^s_l G^{pr}_{qs}$

Integrals and Amplitudes

Let $i, j, k, l, \ldots$ refer to #OCCUPIED indices, $a, b, c, d, \ldots$ to active, $p, q, r, s, \ldots$ to atomic indices

- AO tensor, G^{pr}_{qs}, has 8-fold symmetry, eg one of them G^{rp}_{sq} which is $G(p, r, q, s) = G(r, p, s, q)$
- MO integral V^{ab}_{cd} has likewise 8-fold symmetry.
- MO integral V^{ab}_{ci} has 2-fold symmetry: V^{cb}_{ai}
- MO integral V^{ij}_{ab} has 2-fold symmetry: V^{ji}_{ba}
- MP2 $t^{ij}_{ab} = V^{ij}_{ab}/(\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b)$, same symmetry as the integral
- General MBPT amplitudes follow the same symmetry: $t^{..i..j..}_{..a..b..} = t^{..j..i..}_{..b..a..}$
- in general, contractions do not preserve symmetry

Contracting Tensors

- Factorize:
  $g^{ij}_{qs} = C^i_p C^j_r G^{pr}_{qs}$ is N^6
  $g^{ij}_{qs} = C^i_p (C^j_r G^{pr}_{qs})$ is N^5
- Use BLAS:
  $g^{ir}_{qs} = C^i_p G^{pr}_{qs}$ stored as $g(i,r,q,s) = C(i,p)G(p,r,q,s)$
  gemm$(C(i,p),\ G(p,rqs),\ g(i,rqs))$

For clarity only terms in question are shown explicitly:

- $u_{ab}^{ij} = V_{ab}^{ij} + P(ia/jb)[...0.5 v_{ef}^{ab} t_{ij}^{\prime ef} ... - t_{mj}^{ae} I_{ie}^{mb} + I_{ie}^{ma} t_{mj}^{eb} ...]$
- $P(ia/jb)(u_{ab}^{ij}) = u_{ab}^{ij} + u_{ba}^{ji}$ - symmetrizer
- $0.5 v_{ef}^{ab} t_{ij}^{\prime ef}$ is direct

Memory Considerations

Assume #VIRTUAL to #OCCUPIED ratio is 10:1

- $o(ijab)$: $N^2N^2/100$
  V=1000 $80G$
  V=2000 $1.3T$
- $o(iabc)$: $N^3N/10$
  V=500 $50G$
  V=1000 $800G$
- $o(abc)$: N^3
  V=500 $1G$
  V=1000 $8G$ V=2000 $64G$
- $o(iab)$: $N^3/10$
  V=1000 $0.8G$
  V=2000 $6.4G$ ‘
- $o(ijkl)$: $N^4/10^4$
  V=1000 $0.8G$
  V=2000 $12.8G$

Tricks

- $u(i,j,a,b) = t(a,e,m,j)I(m,b,i,e)$
  $u(i,j,a,b) = t(m,j,a,e)I(m,b,i,e)$
- $u(i,j,a,b) = t(e,b,m,j)I(m,a,i,e)$
  $u(i,j,a,b) = t(m,j,e,b)I(m,a,i,e)$
  $u(j,i,b,a) = t(j,m,b,e)I(m,a,i,e)$
- $u(i,j,a,b)$ and $u(j,i,b,a)$ can be added as is since they will be symmetrized
- Result - all operations are local to a single virtual index, memory requirements relative to index are $o(ija)$

CC implementation

- All $O(4)$ storage, except $O(ijkl)$ is out-of-core
- out-of-core storage can be disk, distributed memory, local memory
- all get/put operations are $O(3)$ and contiguous
- in-core requirements are $O(iab)$
- Multithreaded
- GPU/CUBLAS enabled

Performance

- Memory - improvements by magnitude
  Calculation which takes 24GB distributed and 8GB per node memory only requires 500MB (according to top)
- CCSD takes longer than DDI - smaller matrix size and disk overhead are most likely the culprits, decrease on the order of 20%-30%.
- (T) is faster by $\tilde{\%}25$.
- CUBLAS suffers from memory transfer overhead and smaller matrix sizes, only gives a moderate improvement
- Plenty of room for improvement and parallel implementations

Programming

- HDF5 - data storage model, allowing for multidimensional data sets and attributes
  for example: `read(buffer, start(0,5,7,7), finish(n,n,9,8))`
  groups/dataset: `openDataset(‘‘/cc/diis/t2’’)`
  surprisingly, writes can be more efficient than reads due to buffering
- Functionality for manipulating tensors: multiplication, permutation, symmetry operations
- C++/C/Fortran integration, for example: `call cchem_runtime_set_double(‘‘/cc/convergence’’, 1e-10)`
- Optional ublas library usage instead of BLAS for debugging purposes

Aknowledgements

- Dr. M.S.Gordon
- Dr. R.Olson