Software implementation of correlated quantum chemistry methods. Exploiting advanced programming tools and new computer architectures

Evgeny Epifanovsky
Q-Chem

Septermber 29, 2015

Acknowledgments

Many thanks to my collaborators:

- **Michael Wormit** (Heidelberg)
- Ilya Kaliman and Anna Krylov (USC)
- Edgar Solomonik (ETH)
- Khaled Ibrahim and Samuel Williams (LBL)

Q-CHEM

U.S. DEPARTMENT OF ENERGY | Office of Science

Anatomy of a QC computation

Single point energy

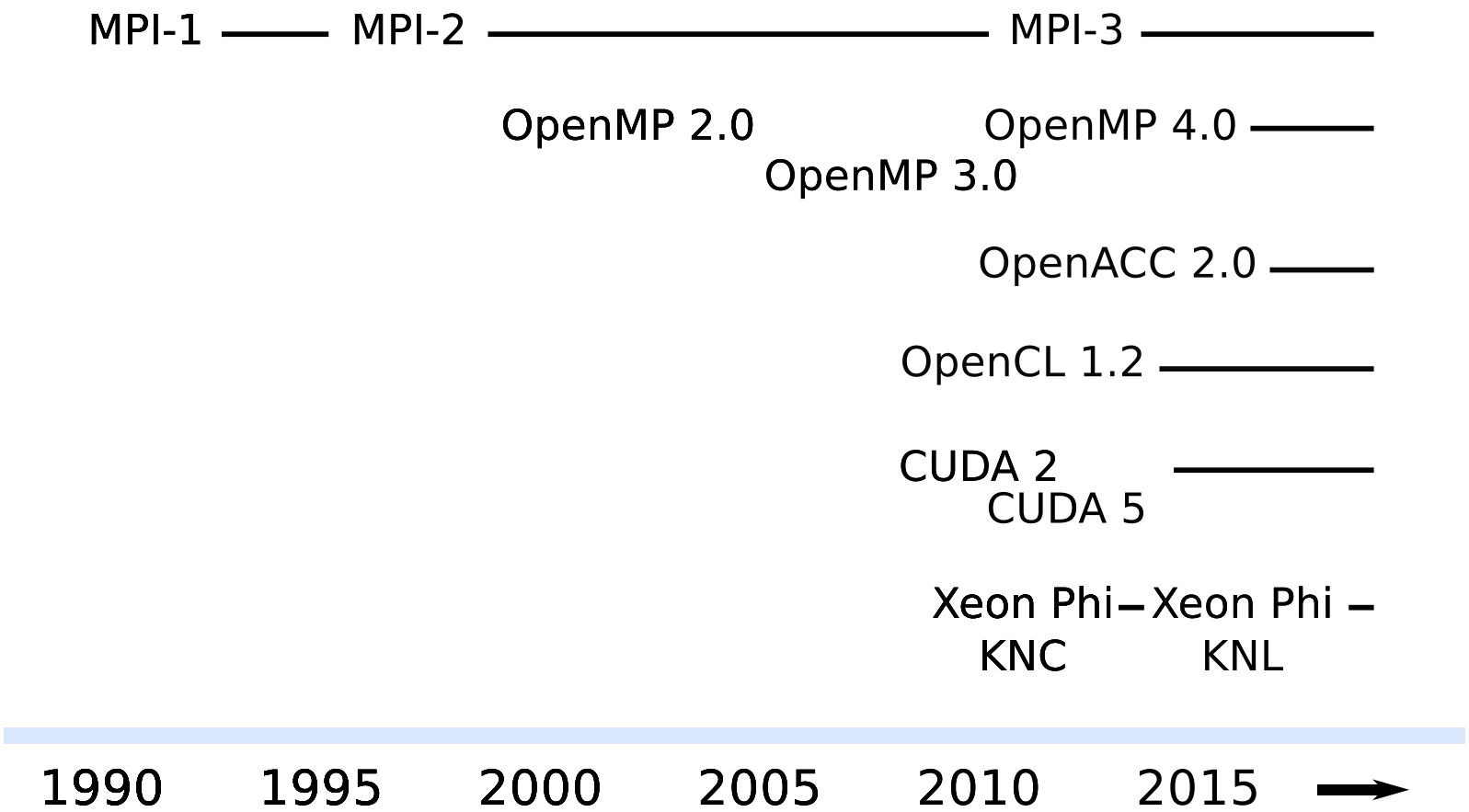
Iterative solver

Programmable tensor expressions

Tensor contractions

BLAS and its extensions

Programming technologies



Coupled cluster methods in Q-Chem

	Ground state	Excited state	Properties
2010–2012	MP2 QCISD CCD, CCSD CCSD(T), (dT), (fT)	CISD EOM-CCSD (EA, EE, IP, SF, DIP, DSF) IP-CISD, EA-CISD	OPDM, TPDM Properties (all methods) Gradient (CC, EOM)
2013	RI-CCSD	RI-EOM-CCSD (EA, EE, IP, SF)	RI-OPDM, RI-TPDM RI properties
2013–2015	CS/CX-MP2 CS/CX-CCSD	CS/CX-CISD CS/CX-EOM-CCSD (EA, EE, IP, SF)	CS/CX-OPDM Real, complex Dyson orbitals Two-photon absorption Spin-orbit coupling

- Over 1000 programmable expressions implemented
- Work by a single academic research group (Krylov @ USC)
- 14 contributors
- 4–5 persons working on method development at a given time

Coupled-cluster doubles (CCD) equations

$$D_{ij}^{ab} = \epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b$$

$$\begin{aligned} T_{ij}^{ab} D_{ij}^{ab} = \langle ij||ab\rangle &+ \mathcal{P}_-(ab)\left(\sum_c f_{bc}\, t_{ij}^{ac} - \frac{1}{2}\sum_{klcd}\langle kl||cd\rangle\, t_{kl}^{bd}\, t_{ij}^{ac}\right) \\ &- \mathcal{P}_-(ij)\left(\sum_k f_{jk}\, t_{ik}^{ab} + \frac{1}{2}\sum_{klcd}\langle kl||cd\rangle\, t_{jl}^{cd}\, t_{ik}^{ab}\right) \\ &+ \frac{1}{2}\sum_{kl}\langle ij||kl\rangle\, t_{kl}^{ab} + \frac{1}{4}\sum_{klcd}\langle kl||cd\rangle\, t_{ij}^{cd}\, t_{kl}^{ab} + \frac{1}{2}\sum_{cd}\langle ab||cd\rangle\, t_{ij}^{cd} \\ &- \mathcal{P}_-(ij)\mathcal{P}_-(ab)\left(\sum_{kc}\langle kb||jc\rangle\, t_{ik}^{ac} - \frac{1}{2}\sum_{klcd}\langle kl||cd\rangle\, t_{lj}^{db}\, t_{ik}^{ac}\right) \end{aligned}$$

$$\mathcal{P}_-(ij)A_{ij} = A_{ij} - A_{ji}$$

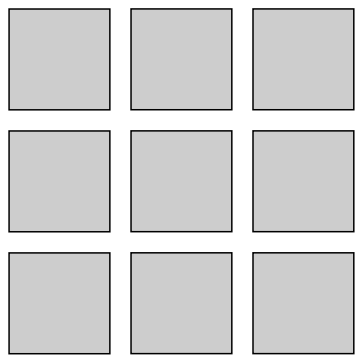
Tensor expressions for CCD

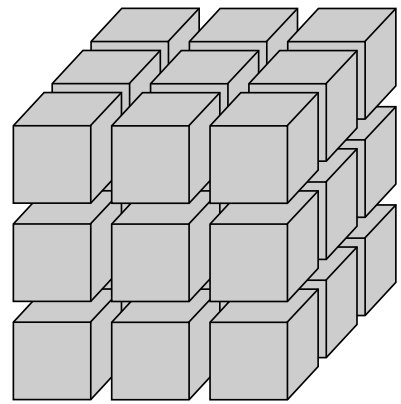
```
void ccd_t2_update(...) {

    letter i, j, k, l, a, b, c, d;
    btensor<2> f1_oo(oo), f1_vv(vv);
    btensor<4> ii_oooo(oooo), ii_ovov(ovov);

    //  Compute intermediates
    f1_oo(i|j) =
          f_oo(i|j) + 0.5 * contract(k|a|b, i_oovv(j|k|a|b), t2(i|k|a|b));
    f1_vv(b|c) =
          f_vv(b|c) - 0.5 * contract(k|l|d, i_oovv(k|l|c|d), t2(k|l|b|d));
    ii_oooo(i|j|k|l) =
          i_oooo(i|j|k|l) + 0.5 * contract(a|b, i_oovv(k|l|a|b), t2(i|j|a|b));
    ii_ovov(i|a|j|b) =
          i_ovov(i|a|j|b) - 0.5 * contract(k|c, i_oovv(i|k|b|c), t2(k|j|c|a));

    //  Compute updated T2
    t2new(i|j|a|b) =
          i_oovv(i|j|a|b)
        + asymm(a, b, contract(c, t2(i|j|a|c), f1_vv(b|c)))
        - asymm(i, j, contract(k, t2(i|k|a|b), f1_oo(j|k)))
        + 0.5 * contract(k|l, ii_oooo(i|j|k|l), t2(k|l|a|b))
        + 0.5 * contract(c|d, i_vvvv(a|b|c|d), t2(i|j|c|d))
        - asymm(a, b, asymm(i, j,
            contract(k|c, ii_ovov(k|b|j|c), t2(i|k|a|c))));
}
```

Block tensors in libtensor

Three components:

- Block tensor space: dimensions + tiling pattern.
- Symmetry relations between blocks.
- Non-zero canonical data blocks.





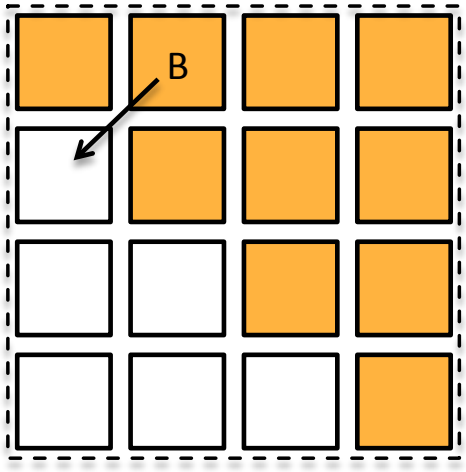
Block tensors in libtensor

Three components:

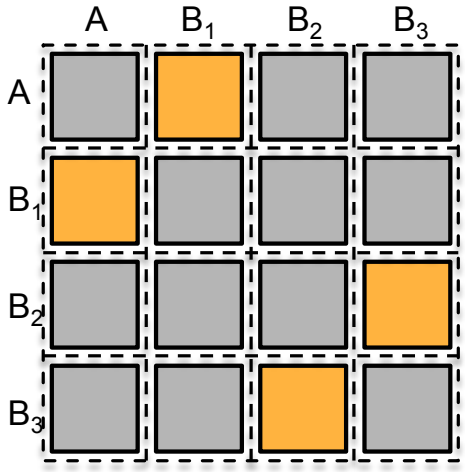
- Block tensor space: dimensions + tiling pattern.
- Symmetry relations between blocks.
- Non-zero canonical data blocks.

Symmetry:

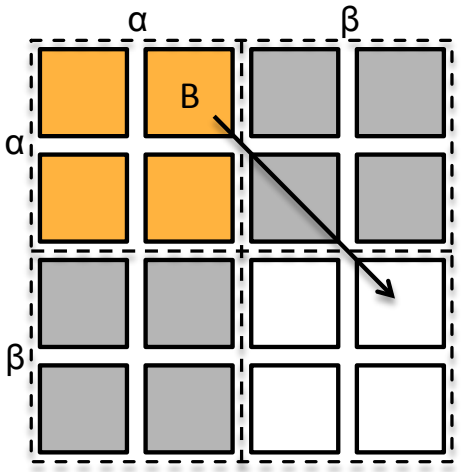
$$S : SB_i \mapsto (B_j, U_{ij})$$



Permutational



Point group



Spin

Front end	**Middleware**	**Back end**
Architecture-independent programming interface	Preparation of platform-specific tasks	Platform-specific optimized kernels

Front end	**Middleware**	**Back end**
Architecture-independent programming interface	Preparation of platform-specific tasks	Platform-specific optimized kernels

TCE in NWChem

Equation specification via GUI	Equation factorization and code generation	Autogenerated Fortran code

Front end	**Middleware**	**Back end**
Architecture-independent programming interface	Preparation of platform-specific tasks	Platform-specific optimized kernels

TCE in NWChem

Equation specification via GUI	Equation factorization and code generation	Autogenerated Fortran code

libtensor in Q-Chem

Tensor expressions	Runtime expression AST optimization	One of back-ends (native, XM, CTF)

Algorithms

1. Virtual memory (RAM + disk) based block tensors (native)
   Targets large-memory machines with fast disk. Most efficient in-core, lacks efficiency when spillover to disk is significant

2. Disk based tensor contraction algorithm (XM by Ilya Kaliman)
   Targets machines with fast disk, lacks efficiency when job fits in RAM

3. Distributed parallel in-core memory tensor library
   (CTF by Edgar Solomonik)
   Targets highly parallel machines with low memory per node and no disk

AST Optimizations

$$I^{(1)}_{iajb} = \sum_c \langle ia||bc\rangle t^c_j$$

$$t^{ab}_{ij} = \mathcal{P}(ij)\mathcal{P}(ab)\sum_{kc} I^{(1)}_{kbic} t^{ac}_{jk} + \mathcal{P}(ij)\sum_c \langle jc||ba\rangle t^c_i$$

```
{ = i1(i,a,j,b) { * ovvv(i,a,b,c) t1(j,c) } }
{ = t2(i,j,a,b) { +
    { asym(i,j; a,b) { * i1(k,b,i,c) t2(j,k,a,c) } }
    { asym(i,j) { * ovvv(j,c,b,a) t1(i,c) } }
} }
```

AST Optimizations

$$I^{(1)}_{iajb} = \sum_c \langle ia||bc\rangle t^c_j$$

$$t^{ab}_{ij} = \mathcal{P}(ij)\mathcal{P}(ab)\sum_{kc} I^{(1)}_{kbic} t^{ac}_{jk} + \mathcal{P}(ij)\sum_c \langle jc||ba\rangle t^c_i$$

```
{ = i1(i,a,j,b) { * ovvv(i,a,b,c) t1(j,c) } }
{ = t2(i,j,a,b) { +
    { asym(i,j; a,b) { * i1(k,b,i,c) t2(j,k,a,c) } }
    { asym(i,j) { * ovvv(j,c,b,a) t1(i,c) } }
} }
```

For disk-based block tensors:

```
{ = i1(i,a,j,b) { * ovvv(i,a,b,c) t1(j,c) } }
{ = x(i,j,a,b) { * ovvv(j,c,b,a) t1(i,c) } }
{ += x(i,j,a,b) { asym(a,b) { * i1(k,b,i,c) t2(j,k,a,c) } } }
{ = t2(i,j,a,b) { asym(i,j) x(i,j,a,b) } }
```

AST Optimizations

$$I^{(1)}_{iajb} = \sum_c \langle ia||bc\rangle t^c_j$$

$$t^{ab}_{ij} = \mathcal{P}(ij)\mathcal{P}(ab)\sum_{kc} I^{(1)}_{kbic} t^{ac}_{jk} + \mathcal{P}(ij)\sum_c \langle jc||ba\rangle t^c_i$$

```
{ = i1(i,a,j,b) { * ovvv(i,a,b,c) t1(j,c) } }
{ = t2(i,j,a,b) { +
    { asym(i,j; a,b) { * i1(k,b,i,c) t2(j,k,a,c) } }
    { asym(i,j) { * ovvv(j,c,b,a) t1(i,c) } }
} }
```

For CTF:

```
{ = i1(i,a,j,b) { * ovvv(i,a,b,c) t1(j,c) } }
{ = t2(i,j,a,b) { asym(i,j; a,b)
    { * i1(k,b,i,c) t2(j,k,a,c) } } }
{ += t2(i,j,a,b) { asym(i,j) { * ovvv(j,c,b,a) t1(i,c) } } }
```

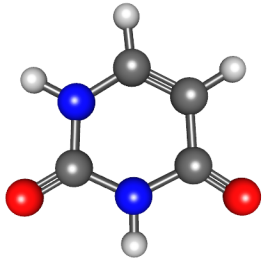
Benchmarks

Benchmarks

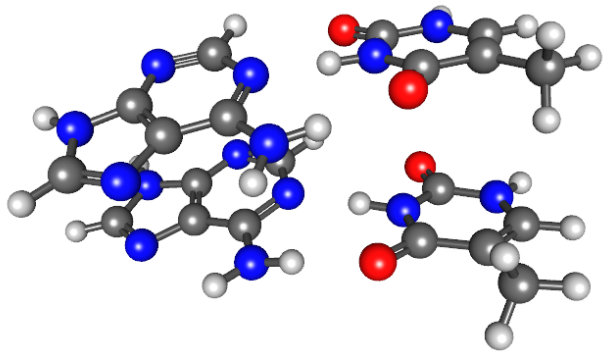
Tests performed on 2 × 8-core Sandy Bridge, 384 GB

Time to solve equations

		Steps	BT	XM	CTF
Uracil/cc-pVDZ	CCSD	10	15 s	66 s	169 s
21 O, 103 V, Cs	EOM-EE	63	46 s	661 s	869 s
Uracil/cc-pVTZ	CCSD	10	273 s	1174 s	1248 s
21 O, 267 V, Cs	EOM-EE	74	537 s	6074 s	3047 s
AATT/cc-pVDZ	CCSD	12	160 h	92 h	
98 O, 506 V, C1	EOM-IP	32	64 m	196 m	



Uracil



AATT

Benchmarks

Tests performed on NERSC Hopper system:
2 $\times$ 12-core AMD Magny Cours, 32 GB (64 GB*)

Time to solve equations

		Steps	BT	CTF-1	CTF-4
Uracil/	CCSD	10	64 s	179 s	139 s
cc-pVDZ	EOM-EE	64	144 s	809 s	696 s
			BT*	CTF-16	CTF-64
Uracil/	CCSD	10	14 m	9 m	4.6 m
cc-pVTZ	EOM-EE	64	39 m	39 m	21.8 m
				CTF-256*	
AATT/	CCSD	12		2.9 h	
cc-pVDZ	EOM-IP	32		235 s	

Benchmarks

Tests performed on NERSC Babbage system:
2 $\times$ 8-core Sandy Bridge, 64 GB, 2 Knight's Corner cards

Time to solve equations

			Sandy Bridge		Intel KNC
		Steps	BT	XM	XM (AO)
Uracil/ cc-pVDZ	CCSD	10	15 s	74 s	83 s
	EOM-EE	63	44 s	462 s	468 s
Uracil/ cc-pVTZ	CCSD				
	EOM-EE				

Conclusions

- Changing landscape in computer technology forces us to make choices about developing and supporting scientific software
- Following appropriate software design and development methodologies enables efficient use of computer and human resources