Massively Parallel Tensor Contraction for High-Accuracy Quantum Chemical Calculations

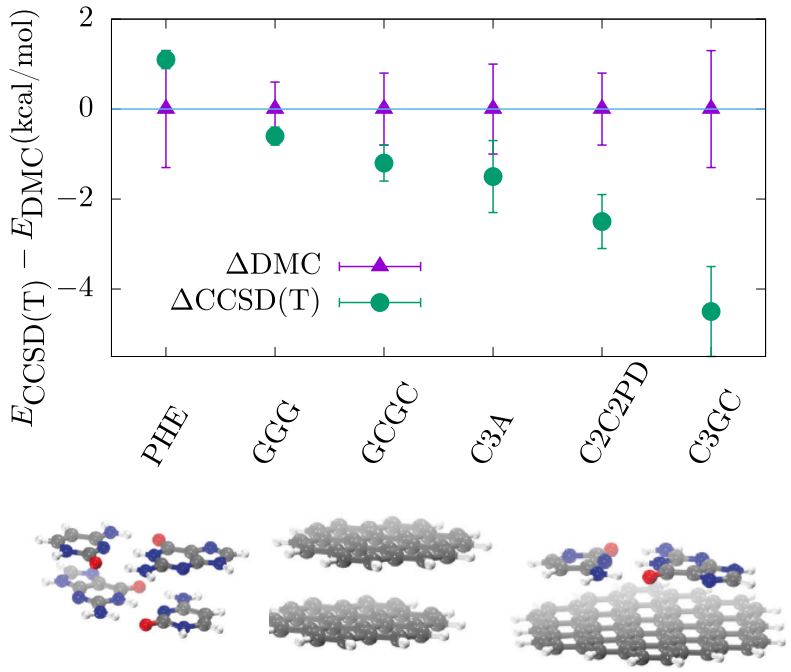
Andreas Irmler
Grüneis group
Vienna University of Technology, Austria

Accurate ab-initio calculations:

Interactions between large molecules pose a puzzle for reference quantum mechanical methods

Yasmine S. Al-Hamdani, Péter R. Nagy , Andrea Zen , Dennis Barton , Mihály Kállay ,
Jan Gerit Brandenburg ✉ & Alexandre Tkatchenko

NATURE COMMUNICATIONS (2021)



Accurate ab-initio calculations:

Interactions between large molecules pose a puzzle for reference quantum mechanical methods

Yasmine S. Al-Hamdani[1,2,9], Péter R. Nagy[3,9], Andrea Zen[4,5,6,7], Dennis Barton[2], Mihály Kállay[3], Jan Gerit Brandenburg[8,10]✉ & Alexandre Tkatchenko[2,10]✉

NATURE COMMUNICATIONS | (2021)12:3927 | https://doi.org/10.1038/s41467-021-24119-3 | www.nature.com/naturecommunications

In obtaining CCSD(T) interaction energies, the sources of error are:

- Single-particle basis representation of the CCSD(T) wavefunction.
- Local approximations of long-range electron correlation according to the LNO scheme.
- Neglected core electron correlation.
- Missing high-order many-electron contributions beyond CCSD(T).

Accurate ab-initio calculations:

Interactions between large molecules pose a puzzle for reference quantum mechanical methods

Yasmine S. Al-Hamdani[1,2,9], Péter R. Nagy[3,9], Andrea Zen[4,5,6,7], Dennis Barton[2], Mihály Kállay[3], Jan Gerit Brandenburg[8,10✉] & Alexandre Tkatchenko[2,10✉]

NATURE COMMUNICATIONS | (2021)12:3927 | https://doi.org/10.1038/s41467-021-24119-3 | www.nature.com/naturecommunications

Understanding Discrepancies of Wavefunction Theories for Large Molecules

Tobias Schäfer, Andreas Irmler, Alejandro Gallo, and Andreas Grüneis

arXiv: 2407.01442, Nat. Comm (accepted)

- plane-wave basis (VASP)
- canonical approach

In obtaining CCSD(T) interaction energies, the sources of error are:

- Single-particle basis representation of the CCSD(T) wavefunction.
- Local approximations of long-range electron correlation according to the LNO scheme.
- Neglected core electron correlation.
- Missing high-order many-electron contributions beyond CCSD(T).

Theory	Interaction energy
MP2	-38.5 ± 0.5
CCSD	-13.4 ± 0.5
CCSD(T)	-21.1 ± 0.5
LNO-CCSD(T)	-20.6 ± 0.6
DLPNO-CCSD(T_0)	-20.9 ± 0.4
PNO-LCCSD(T)-F12	-20.0
DMC	-18.1(8)
DMC	-17.5(14)

- Gaussian type basis sets
- local approaches

Accurate ab-initio calculations:

Interactions between large molecules pose a puzzle for reference quantum mechanical methods

Yasmine S. Al-Hamdani[1,2,9], Péter R. Nagy[3,9], Andrea Zen[4,5,6,7], Dennis Barton[2], Mihály Kállay[3], Jan Gerit Brandenburg[8,10]✉ & Alexandre Tkatchenko[2,10]✉

NATURE COMMUNICATIONS | (2021)12:3927 | https://doi.org/10.1038/s41467-021-24119-3 | www.nature.com/naturecommunications

In obtaining CCSD(T) interaction energies, the sources of error are:

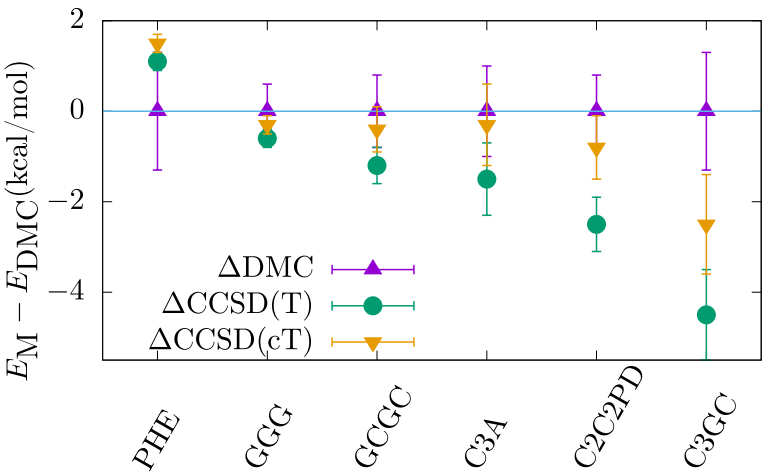
- Single-particle basis representation of the CCSD(T) wavefunction.
- Local approximations of long-range electron correlation according to the LNO scheme.
- Neglected core electron correlation.
- Missing high-order many-electron contributions beyond CCSD(T).

CCSD(cT)

Understanding Discrepancies of Wavefunction Theories for Large Molecules

Tobias Schäfer, Andreas Irmler, Alejandro Gallo, and Andreas Grüneis

arXiv: 2407.01442, Nat. Comm (accepted)

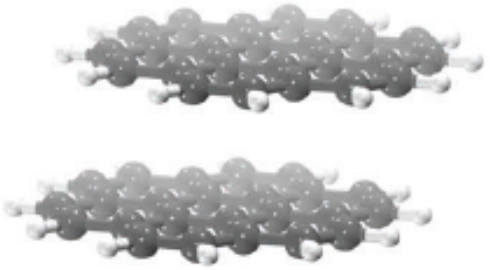
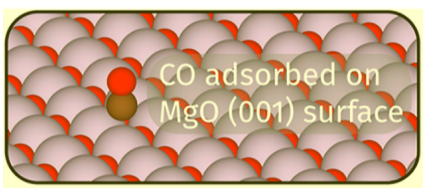


PHYSICAL REVIEW LETTERS **131,** 186401 (2023)

Averting the Infrared Catastrophe in the Gold Standard of Quantum Chemistry

Nikolaos Masios , Andreas Irmler , Tobias Schäfer , and Andreas Grüneis

Large-scale CCSD(T) calculations:

	local approaches	canonical approach
	~220GB ~50k CPUh*	~6TB 250k CPUh*
	~20GB 20k CPUh*	~3TB 200k CPUh*

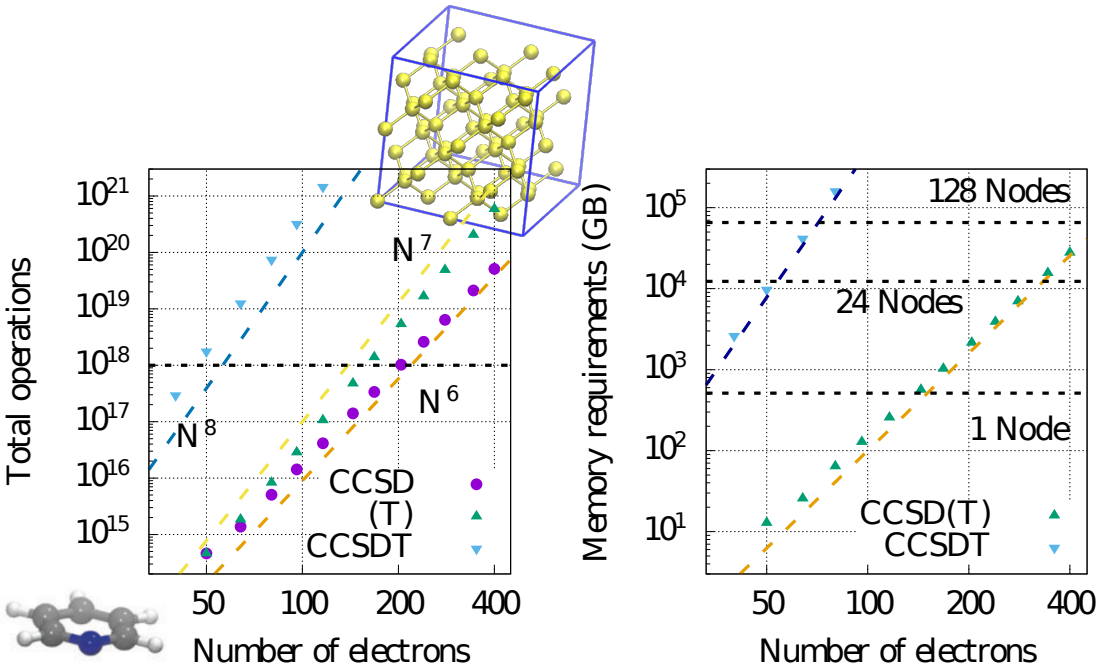
Large canonical calculations

Distributed parallel tensor frameworks

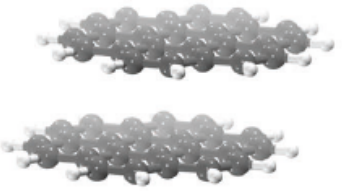
Hirata:	**Solomonik, Matthews, Hammond:**	**Valeev:**
TCE/ NWChem	**CTF/ Aquarius**	**TiledArray/ MPQC**

***Amd Zen3 core hour**

Coupled-cluster methods: Computational cost and performance



$$r_{ij}^{ab} = \sum_{ck} v_{ic}^{ak} t_{kj}^{cb}$$



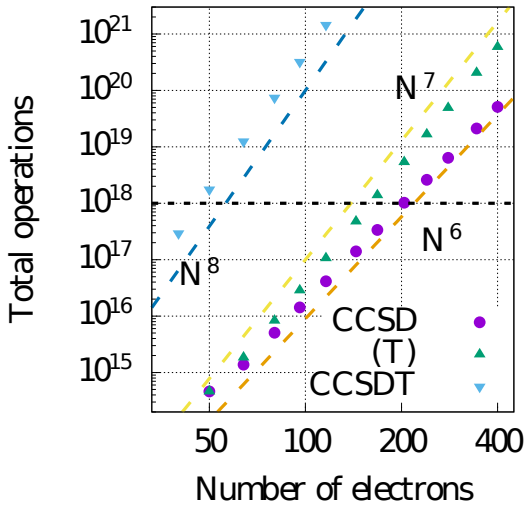
i=*j*=108

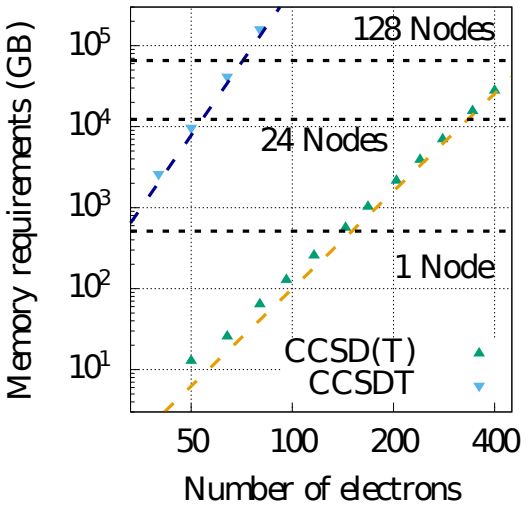
a=*b*=1164 *

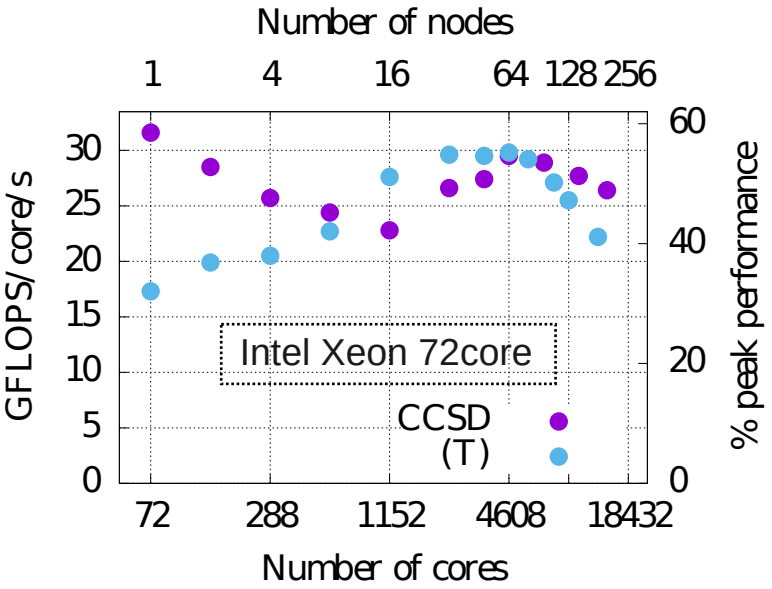
360GB mem.
~1h Zen3 128core

*aug-cc-pvdz

Coupled-cluster methods: Computational cost and performance

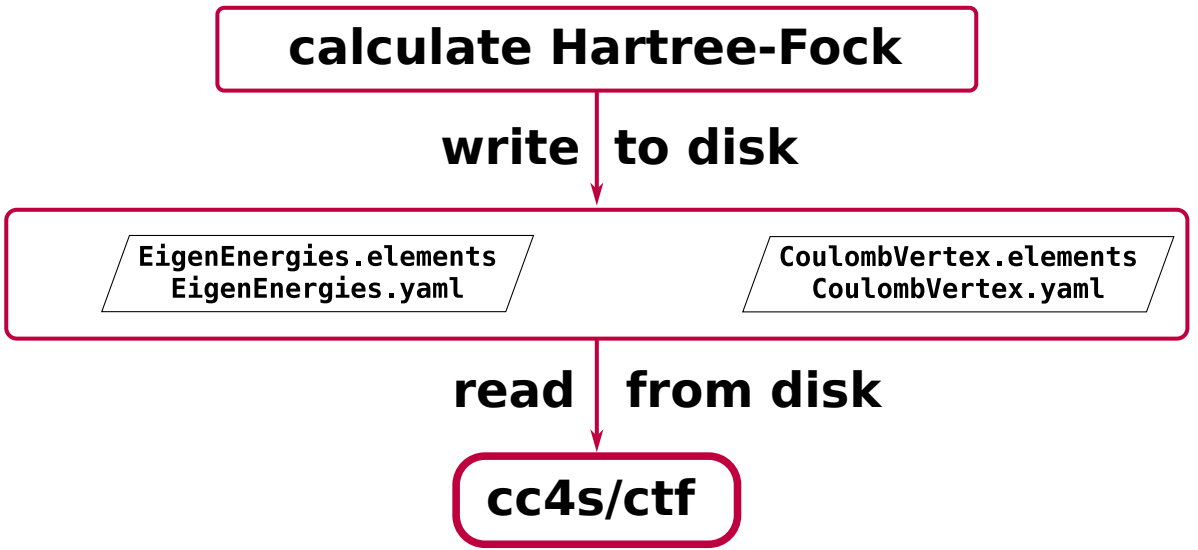






cc4s: massively parallel coupled-cluster code for solids and molecules

exciting VASP MRCC abinit FHI-aims
The ab initio materials simulation package



ctf - cyclops tensor framework

- tensors are distributed over all ranks (cyclic distribution)
- transpose tensor indices (local op.)
- parallel contraction using variants of the summa algorithm

$$r_{ij}^{ab} = \sum_{ck} v_{ic}^{ak} t_{kj}^{cb}$$

```
#include <ctf.hpp>

int main(int argc, char ** argv){
  MPI_Init(&argc, &argv);
  int No(10), Nv(100);

  CTF::Tensor<> r(4, {Nv,Nv,No,No});
  CTF::Tensor<> v(4, {Nv,No,No,Nv});
  CTF::Tensor<> t(4, {Nv,Nv,No,No});

  r["abij"] = v["akic"] * t["bckj"];

  MPI_Finalize();
  return 0;
}
```

github.com/cyclops-community/ctf

ctf - cyclops tensor framework
Simple Example

$$C_{ijkl} = \sum_G A^G_{ik} B^G_{jl}$$

Contraction performance model: find the best cyclic distribution which allows contraction:

- redistribute all tensors ($N^4 + N^3$)
- transpose rhs-tensors (N^3)
- parallel contraction via summa (N^5)
- transpose result tensor (N^4)
- bring result tensor in orignal distribution (N^4)

ctf - cyclops tensor framework
Simple Example

$$C_{ijkl} = \sum_G A^G_{ik} B^G_{jl}$$

Contraction performance model: find the best cyclic distribution which allows contraction:

- redistribute all tensors ($N^4 + N^3$)
- transpose rhs-tensors (N^3)
- parallel contraction via summa (N^5)
- transpose result tensor (N^4)
- bring result tensor in orignal distribution (N^4)

Kernel	Relative time
redistribute	5-30%
transpose	0-5%
gemm	40-80%
bcast/reduce	15-40%

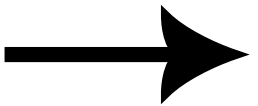
ctf - cyclops tensor framework
Simple Example

$$C_{ijkl} = \sum_G A^G_{ik} B^G_{jl}$$

Contraction performance model: find the best cyclic distribution which allows contraction:

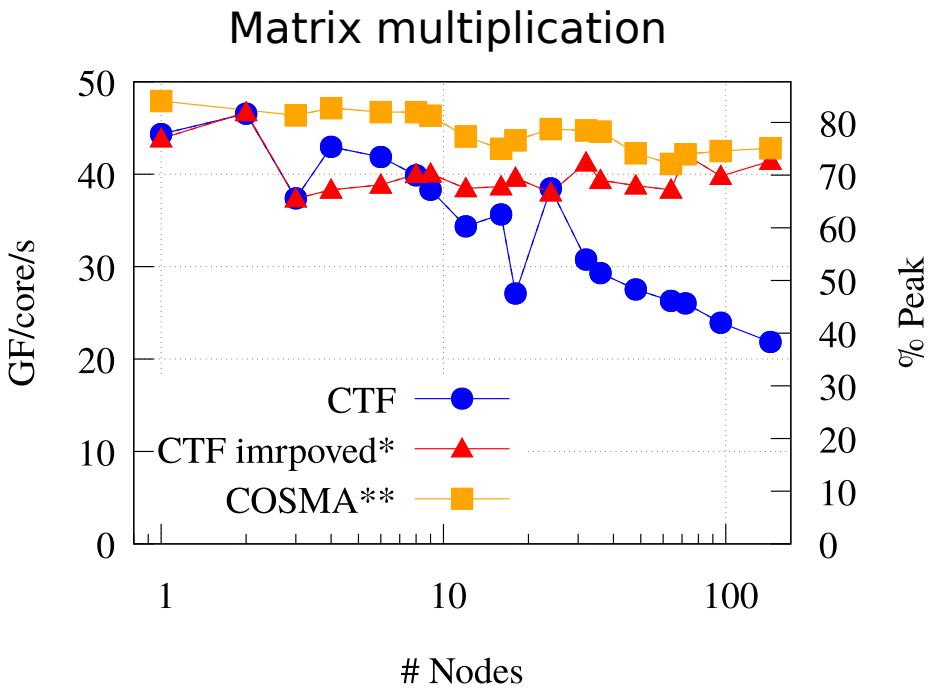
- redistribute all tensors ($N^4 + N^3$)
- transpose rhs-tensors (N^3)
- parallel contraction via summa (N^5)
- transpose result tensor (N^4)
- bring result tensor in orignal distribution (N^4)

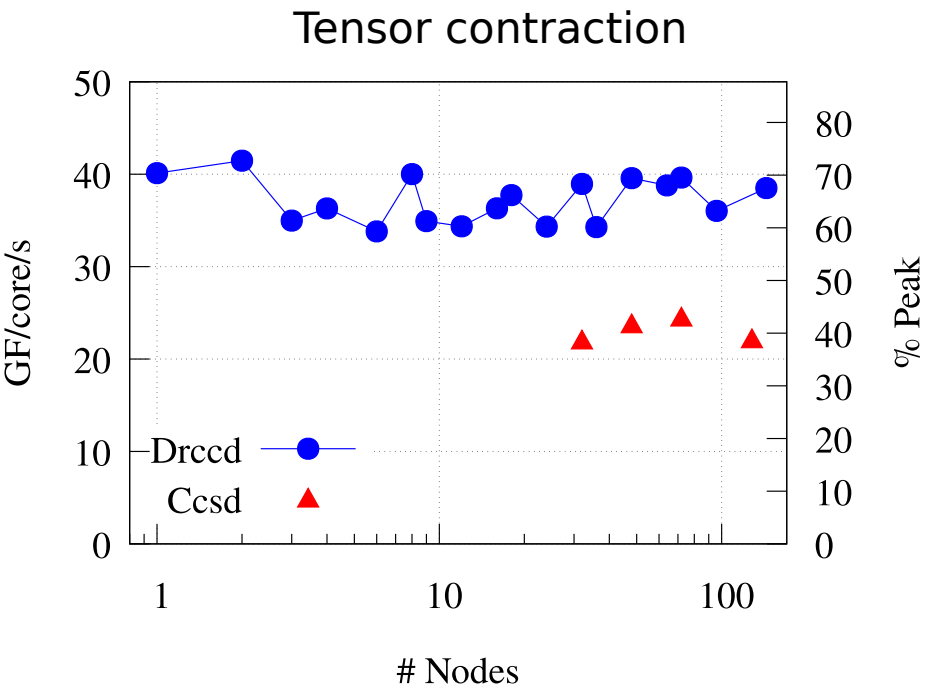
Kernel	Relative time
redistribute	5-30%
transpose	0-5%
gemm	40-80%
bcast/reduce	15-40%



GEMM dominates runtime and runs at peak efficiency

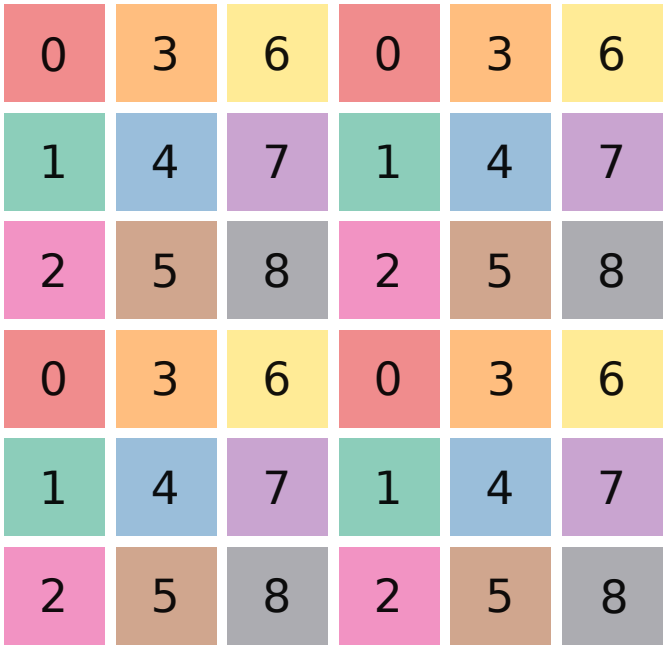
ctf - cyclops tensor framework
Performance





* Irmler, ..., Solomonik; EuroPar 23'
**Kwasniewski, ..., Hoefler; SC'19

ctf: cyclic distribution



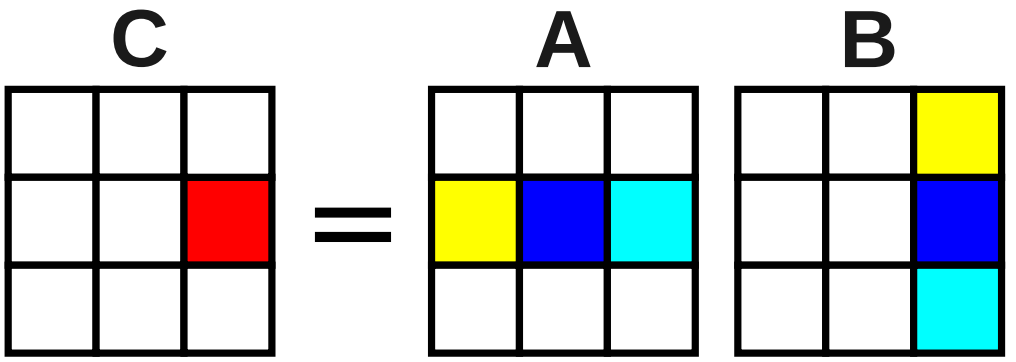
cyclic phase among all edges of each tensor

ctf: cyclic distribution

0	3	6	0	3	6
1	4	7	1	4	7
2	5	8	2	5	8
0	3	6	0	3	6
1	4	7	1	4	7
2	5	8	2	5	8

cyclic phase among all edges of each tensor

SUMMA algorithm:

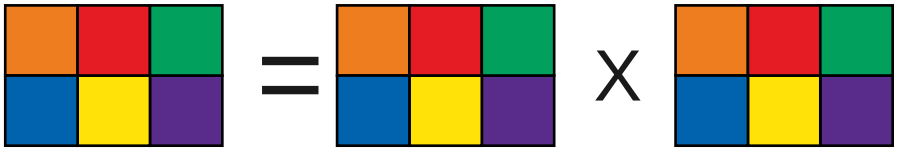


ctf: cyclic distribution

0	3	6	0	3	6
1	4	7	1	4	7
2	5	8	2	5	8
0	3	6	0	3	6
1	4	7	1	4	7
2	5	8	2	5	8

cyclic phase among all edges of each tensor

In a tensor contraction - the phases of the tensors in the contraction have to be consistent:

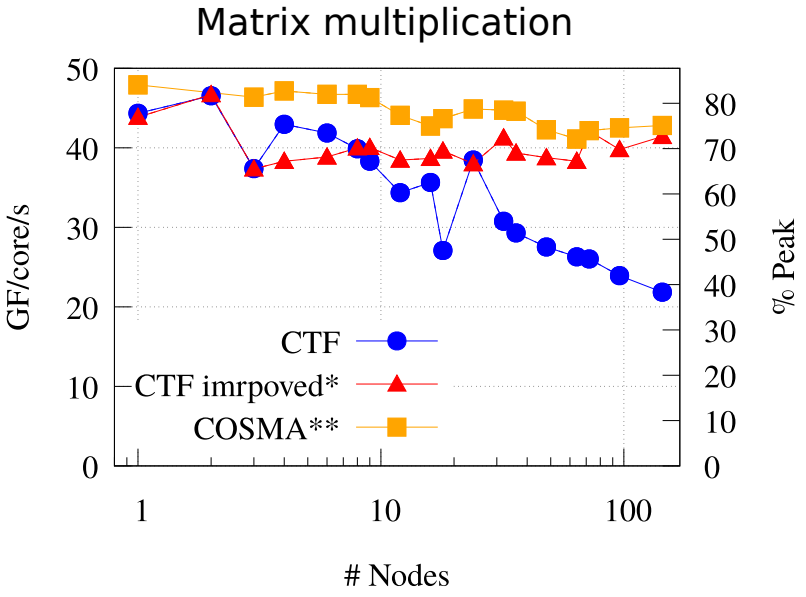


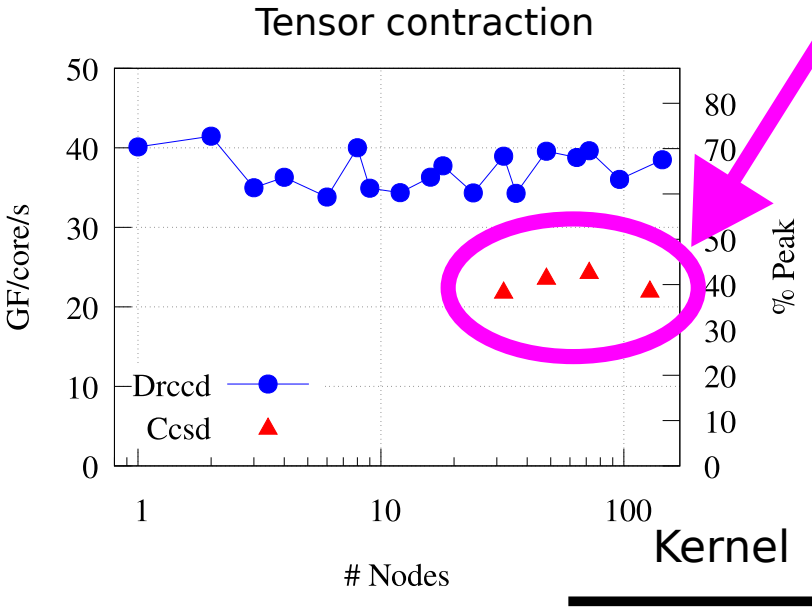
Redistribute!

In practice, redistribution is required in every contraction!! (true for CC calculations)!

ctf - cyclops tensor framework
Performance

1000 M gemm operations
vs.
M redistribute/transpose





N^3 gemm operations
vs.
N^2 transpose/redistribute

Kernel	Relative time
redistribute	5-30%
transpose	0-5%
gemm	40-80%
bcast/reduce	15-40%

ctf: symmetries

$$t_{ij}^{ab} = -t_{ji}^{ab} = -t_{ij}^{ba} = t_{ji}^{ba}$$

```
int asas[] = {AS, NS, AS, NS};
int abij[] = {nocc, nocc, nvir, nvir};
Tensor<> T(4, abij, asas);
Tensor<> R(4, abij, asas);
{
   int abcd[] = {nvir, nvir, nvir, nvir};
   Tensor<> V(4, abcd, asas);
   R["abij"] = V["abcd"] * T["cdij"];
}
```

Often, we cannot store V["abcd"] in memory

$$\text{for } x \text{ in blocks}(a),\ y \text{ in blocks}(b):$$
$$V[\text{"xycd"}] = X[\text{"Gxc"}] * X[\text{"Gyd"}];$$
$$R[\text{"xyij"}] += V[\text{"xycd"}] * T[\text{"cdij"}];$$

ctf: symmetries

$$t_{ij}^{ab} = -t_{ji}^{ab} = -t_{ij}^{ba} = t_{ji}^{ba}$$

```
int asas[] = {AS, NS, AS, NS};
int abij[] = {nocc, nocc, nvir, nvir};
Tensor<> T(4, abij, asas);
Tensor<> R(4, abij, asas);
{
   int abcd[] = {nvir, nvir, nvir, nvir};
   Tensor<> V(4, abcd, asas);
   R["abij"] = V["abcd"] * T["cdij"];
}

{
  int nsns[] = {NS, NS, NS, NS};
  int akic[] = {nvir, nocc, nocc, nvir};
  Tensor<> V(4, akic, nsns);
  R["abij"] = V["akic"] * T["cbkj"];
}
```

Often, we cannot store V["abcd"] in memory

$$\text{for } x \text{ in blocks}(a),\ y \text{ in blocks}(b):$$
$$V[\text{"xycd"}] = X[\text{"Gxc"}] * X[\text{"Gyd"}];$$
$$R[\text{"xyij"}] += V[\text{"xycd"}] * T[\text{"cdij"}];$$

- transform T to non-symmetric tensor
- contraction without symmetry
- transform R to symmetric tensor

ctf: porting to GPUs

Contraction performance model: find the best cyclic distribution which allows contraction:

- redistribute all tensors ($N^4 + N^3$)
- transpose rhs-tensors (N^3)
- parallel contraction via summa (N^5)
- transpose result tensor (N^4)

} Relatively straightforward to port to GPUs.

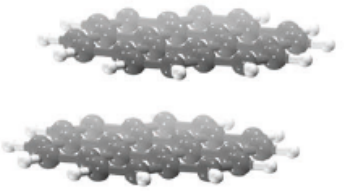
- bring result tensor in orignal distribution (N^4)

In practice, redistribution is required in every contraction!! (true for CC calculations)!

Redistribution step can not be simply overlapped with other parts of the calculation.

Atrip: evaluation of (T)

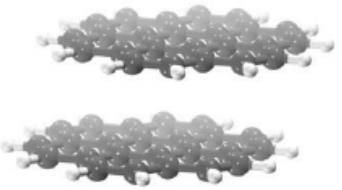
$$\delta E_{(T)} = \sum_{ijk} \sum_{abc} \hat{t}_{ijk}^{abc} \bar{t}_{ijk}^{abc}$$



$$t_{ijk}^{abc} \approx 10000\mathrm{TB}$$

Atrip: evaluation of (T)

$$\delta E_{(T)} = \sum_{ijk} \sum_{abc} \hat{t}^{abc}_{ijk} \bar{t}^{abc}_{ijk}$$



$$t^{abc}_{ijk} \approx 10000\text{TB}$$

Instead:

Solve CCSD to get T_1, T_2

for occupied (i, j, k) :

$\hat{t}^{abc} = \text{build\_intermediate}(T_1, T_2, \text{integrals})$#for fixed ijk

$\bar{t}^{abc} = \text{build\_intermediate}(T_1, T_2, \text{integrals})$

$\delta E_{(T)} \leftarrow \sum_{abc} \hat{t}^{abc} \bar{t}^{abc}$

→ Memory footprint does not exceed that of CCSD

⚠ Not efficiently implementable in CTF

Conclusions:

- CTF allows efficient CCSD calculations up to 100 nodes
- Careful assessment of whether symmetries are efficient
- Not all coupled-cluster variants can be efficiently implemented with CTF
- Efficiently porting CTF to GPUs is challenging