Parallel Design for MP2 Calculation

Team 10
Hieu, Mouza, Olga, Artur

Steps to calculate MP2 energy

- Step 1: Change of basis

$$(pq|P) = \sum_{q}^{N+M} \left(C^{\dagger}\right)_{q\nu} \sum_{p}^{N+M} \left(C^{\dagger}\right)_{p\mu} (\mu\nu|P)$$

- Step 2: Obtain B_matrix

  (Orthogonalization)

$$B_{pq}^{P} = \sum_{P}^{N+M} (pq|P)\,(P|Q)^{-1/2}$$

$$(ia|jb) = \sum_{P} B_{ia}^{P} B_{jb}^{P}$$

- Step 3: Calculate MP2

$$E_{MP2} = \sum_{ij} \sum_{ab} \frac{(ia \mid jb)[2(ia \mid jb) - (ib \mid ja)]}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b}$$

$\left(C^{\dagger}\right)$	NBASIS NBASIS
$(\mu\nu\|P)$	NBASIS NBASIS NAUX
$(P\|Q)^{-1/2}$	NAUX NAUX
MO_energy	NBASIS

Compute kernels in the calculation

Use BLAS Level 3 DGEMM to optimize serial code

1)

```
for (size_t P = 0; P < NAUX; P ++){
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasTrans, NBASIS, NBASIS, NBASIS, 1.0,

    , NBASIS, C, NBASIS, 1.0, uqP+ P *NBASIS *NBASIS, NBASIS);
}
```

$$(pq|P) = \sum_{q}^{N+M} \left(C^{\dagger}\right)_{q\nu} \sum_{p}^{N+M} \left(C^{\dagger}\right)_{p\mu} (\mu\nu|P)$$

Compute kernels in the calculation

Use BLAS Level 3 DGEMM to optimize serial code

1)

```
for (size_t P = 0; P < NAUX; P ++){
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasTrans, NBASIS, NBASIS, NBASIS, 1.0,

    , NBASIS, C, NBASIS, 1.0, uqP+ P *NBASIS *NBASIS, NBASIS);
}
```

2)

```
cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, NBASIS*NBASIS, NAUX, NAUX, 1.0,
 pqP, NBASIS*NBASIS, overlap, NAUX, 1.0, B_pqP, NBASIS*NBASIS);
```

```
for (size_t p = 0; p < NBASIS; p++){
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, NBASIS, NAUX, NAUX, 1.0,
    pqP_new+p*NAUX*NBASIS, NAUX, overlap, NAUX, 1.0, B_pqP+p*NAUX*NBASIS, NAUX);
```

$$B_{pq}^{P} = \sum_{P}^{N+M} (pq|P)\,(P|Q)^{-1/2}$$

Compute kernels in the calculation

Use BLAS Level 3 DGEMM to optimize serial code

1)

```
for (size_t P = 0; P < NAUX; P ++){
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasTrans, NBASIS, NBASIS, NBASIS, 1.0,

    , NBASIS, C, NBASIS, 1.0, uqP+ P *NBASIS *NBASIS, NBASIS);
}
```

2)

```
cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, NBASIS*NBASIS, NAUX, NAUX, 1.0,
 pqP, NBASIS*NBASIS, overlap, NAUX, 1.0, B_pqP, NBASIS*NBASIS);
```

```
for (size_t p = 0; p < NBASIS; p++){
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, NBASIS, NAUX, NAUX, 1.0,
    pqP_new+p*NAUX*NBASIS, NAUX, overlap, NAUX, 1.0, B_pqP+p*NAUX*NBASIS, NAUX);
```

3)

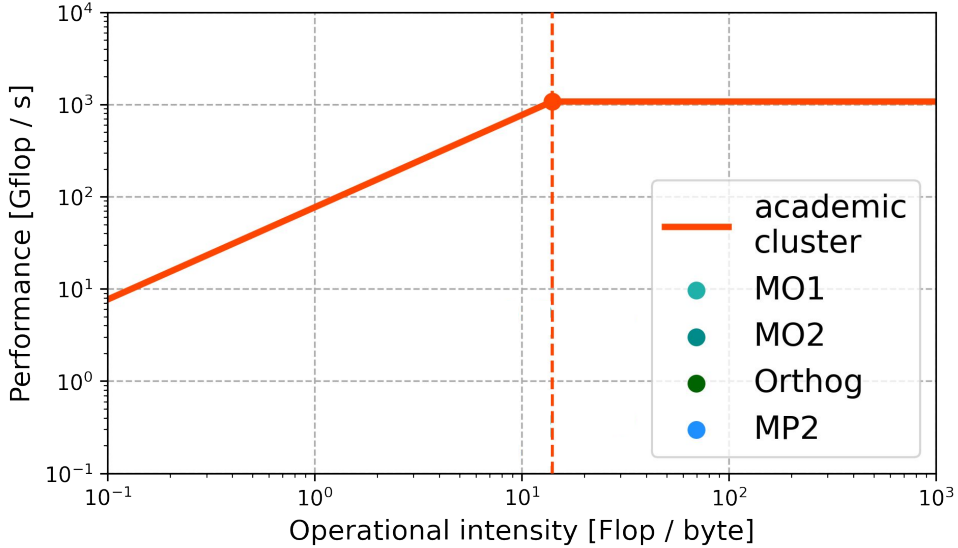
```
for (size_t i = 0; i < NELECS; i++){
    for (size_t j = 0; j < NELECS; j++){
        int NVIR = NBASIS - NELECS;
        double *ipjq = new double[NVIR * NVIR] {};
        // contract over P
        cblas_dgemm(CblasRowMajor, CblasTrans, CblasNoTrans, NVIR, NVIR, NAUX, 1.0, B_iaP + i*NVIR*NAUX, NVIR, B_iaP+j*NVIR*NAUX, NVIR, 1.0, ipjq, NVIR);
        // contract over a and b
        // Vectorization can be done here since this is matrix matrix element wise multiplication
        for (size_t a = NELECS; a < NBASIS; a++){
            for (size_t b = NELECS; b < NBASIS; b++){
                double ibja = ipjq[(b - NELECS)*NVIR + (a - NELECS)];
                double iajb = ipjq[(a - NELECS)*NVIR + (b - NELECS)];
                double delta = MO_energy[i] + MO_energy[j] - MO_energy[a] - MO_energy[b];
                E_MP2 += 1/delta * (2 * iajb *iajb - iajb * ibja);
            }
        }
```

$$(ia|jb) = \sum_P B_{ia}^P B_{jb}^P$$

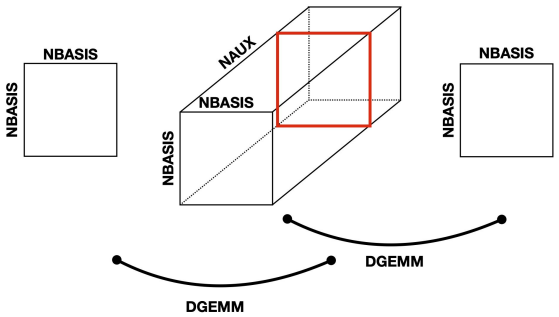
$$E_{MP2} = \sum_{ij} \sum_{ab} \frac{(ia \mid jb)[2(ia \mid jb) - (ib \mid ja)]}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b}$$

Roofline analysis for compute kernels

$$\pi = 1075.2 \text{ Gflop } /\text{s}$$
$$\beta = 76.8 \text{ GB } /\text{s}$$



Roofline analysis for compute kernels

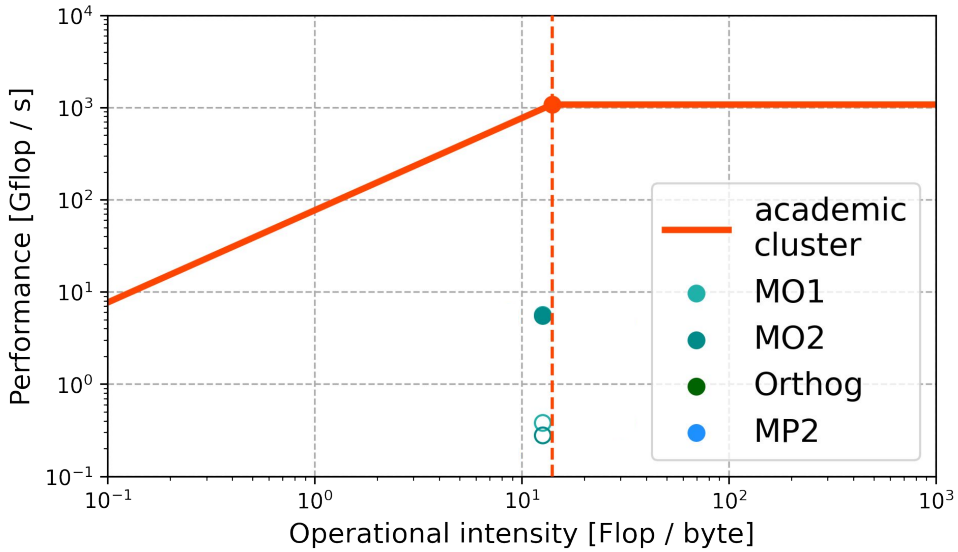


$$F = 2 \cdot \mathrm{NBASIS}^3 \cdot \mathrm{NAUX}$$

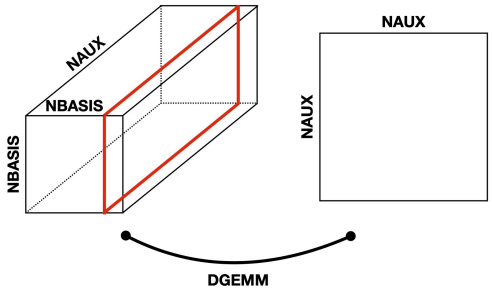
$$I = \frac{\mathrm{NBASIS}}{16}$$

Time in sec

Molecule	MO 1	MO 2
C_3H_8 for-loops	20.99	28.67
C_3H_8 CBLAS	1.42	1.45



Roofline analysis for compute kernels

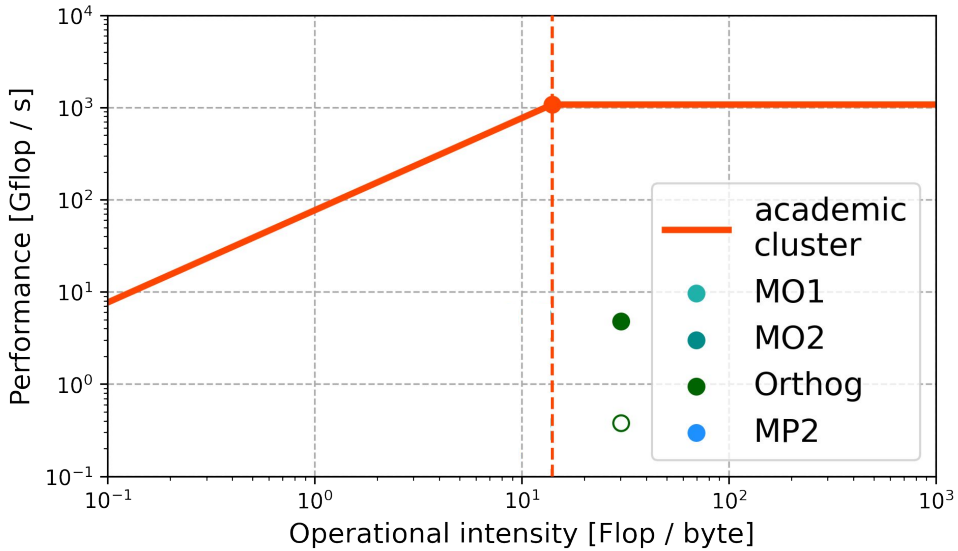


$$F = 2 \cdot \mathrm{NBASIS}^3 \cdot \mathrm{NBASIS} \cdot \mathrm{NBASIS}$$

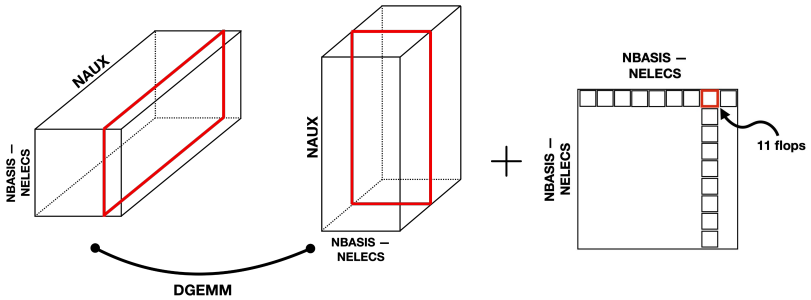
$$I = \frac{\mathrm{NAUX}}{16}$$

Time in sec

Molecule	Ortho
C_3H_8 for-loops	50.54
C_3H_8 CBLAS	3.29



Roofline analysis for compute kernels

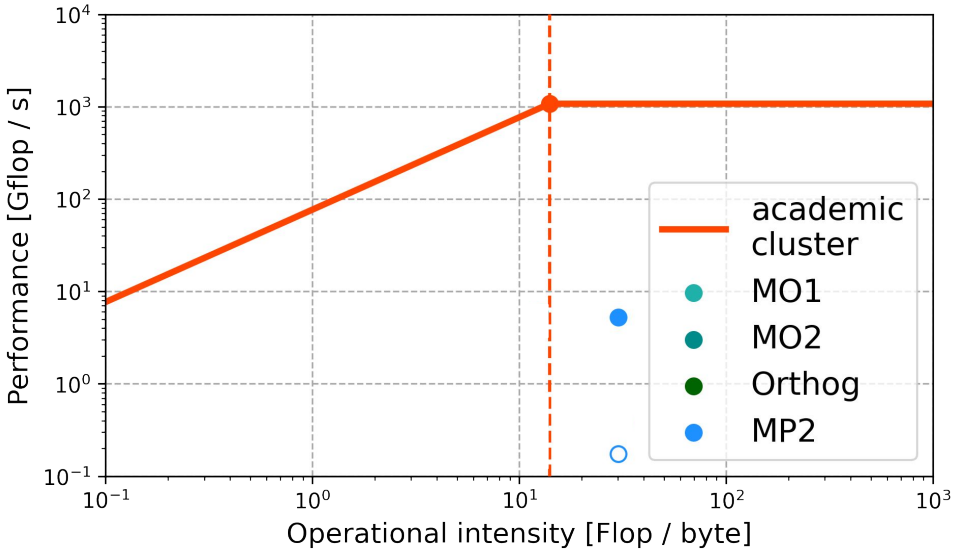


$$F = 2 \cdot (\text{NBASIS - NELECS})^2 \cdot \text{NAUX} + 11(\text{NBASIS - NELECS})^2$$

$$I = \frac{\text{NAUX}}{16}$$

Time in sec

Molecule	MP2
C_3H_8 for-loops	34.04
C_3H_8 CBLAS	1.18



Prelim speed-up (32c): 1a ~ 16x
1b ~ 10x

Shared memory model parallelization

Step 1 - MO Transformation

$$(pq|P) = \sum_{q}^{N+M} \left(C^{\dagger}\right)_{q\nu} \sum_{p}^{N+M} \left(C^{\dagger}\right)_{p\mu} (\mu\nu|P)$$

```
#pragma omp parallel for num_threads(OMP_NUM_THREADS)
```

for $p < N + M, P < NAUX, \nu < N + M$ **do**
  $(p\nu|P) \leftarrow (C^{\dagger})_{p\mu}(\mu\nu|P) + (p\nu|P)$;
end

#pragma omp parallel for num_threads(OMP_NUM_THREADS)

for $p < N + M, P < NAUX, q < N + M$ **do**
  $(pq|P) \leftarrow (C^{\dagger})_{q\nu}(p\nu|P) + (pq|P)$;
end

Prelim speed-up (32c): ~ 14x

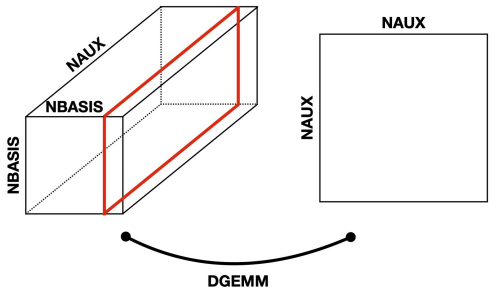
Shared memory model parallelization

Step 2 - Orthogonalization

$$B_{pq}^{P} = \sum_{P}^{NAUX} (pq|Q)\,(P|Q)^{-1/2}$$

$B_{pq}^{P} = 0;$
for $p, q < N + M, P < NAUX$ **do**
  for $Q < NAUX$ **do**
    $B_{pq}^{P} = B_{pq}^{P} + (pq|Q)\,(Q|P)^{-1/2};$
  end
end

```
#pragma omp parallel for num_threads(OMP_NUM_THREADS)
```



Prelim speed-up (32c): ~ 16x

Shared memory model parallelization

Step 3 - Calculate MP2

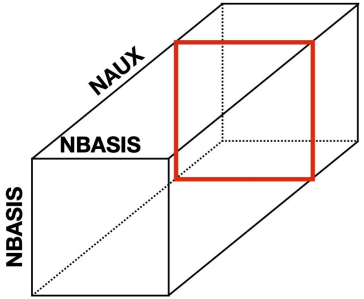
```
#pragma omp parallel for num_threads(OMP_NUM_THREADS)
collapse(2) reduction(+:E_MP2)
```

$$(ia|jb) = \sum_P B_{ia}^P B_{jb}^P$$

for $i, j < N$ **do**
  for $N \leq a, b < N+M$ **do**
    $E_{MP2} = E_{MP2} + \frac{1}{\Delta_{ij}^{ab}} \times [(ia|jb)\,[2\,(ia|jb) - (ib|ja)]];$
  end
end

$$E_{MP2} = \sum_{ij} \sum_{ab} \frac{(ia \mid jb)[2(ia \mid jb) - (ib \mid ja)]}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b}$$

Distributed memory model parallelization (if time permits)



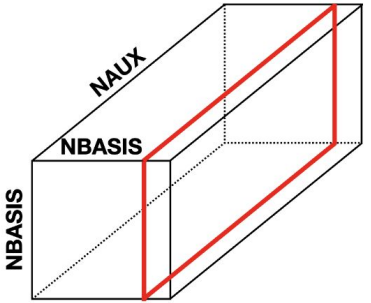
- Each rank has C (NBASIS x NBASIS) and (P|Q) (NAUX x NAUX). Rank 3 tensor is distributed across multiple ranks along dimension P.

$$(pq|P) = \sum_{q}^{N+M} \left(C^{\dagger}\right)_{q\nu} \sum_{p}^{N+M} \left(C^{\dagger}\right)_{p\mu} (\mu\nu|P)$$

- Change of basis is done independently on each rank with DGEMM.

- Change the data layout so that either p or q is the slowest changing index.

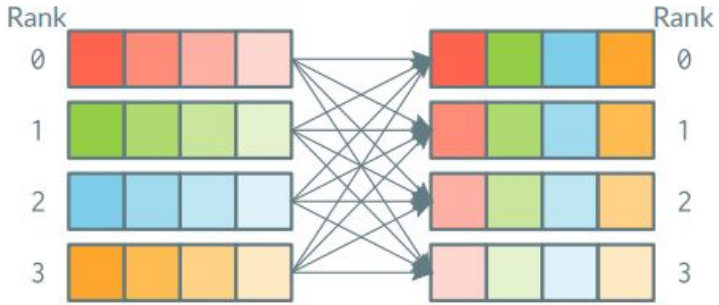
Distributed memory model parallelization (if time permits)



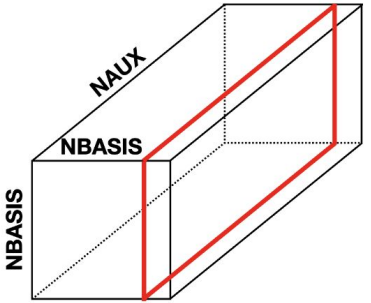
- MPI_Alltoall to get the transpose of rank-3 tensor. If the dimension does not work out, we will use MPI_Send and MPI_Recv

$$B_{pq}^{P} = \sum_{P}^{NAUX} (pq|Q)\,(P|Q)^{-1/2}$$

- Orthogonalization is done on each node with DGEMM.



Distributed memory model parallelization (if time permits)

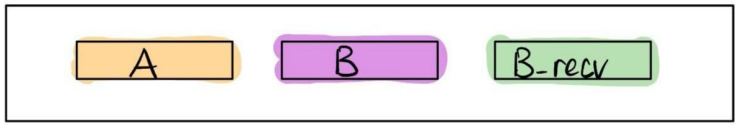


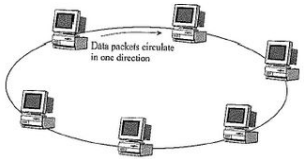
- Ideally, we do not change the memory layout from step 2. We now do matrix multiplication to get the rank-4 tensor.

```
for (size_t i = 0; i < NELECS; i++){
    for (size_t j = 0; j < NELECS; j++){
```

$$(ia|jb) = \sum_P B_{ia}^P B_{jb}^P$$

- Each rank has access to some i (A below). We can allocate extra memory (B and B_recv). Using idea from Cannon Algorithm, we can hide latency while doing calculation between A and B. Synchronization is enforced after the computation between A and B is done.





- After a full cycle of sending matrix B around, use MPI_Reduce to get the final MP2 energy.

Your team needs to present the design of your parallel application that covers the following sections:

- You should have a sequential baseline of your application at this point for which you can present results for a simple test case. Results may be presented using visuals such as simple graphs, contour plots, volume renderings or movies for example.
- Present a profiling of your sequential baseline to identify the bottlenecks and present a simple roofline analysis of the identified compute kernels.
- Based on your analysis above propose the forms of parallelism you want to exploit in your application and which parallel programming models you will use. (This may deviate from the draft you presented in the previous presentation.)
- Elaborate on how you plan to implement the parallel code in terms of logic. What is the sequence of computational steps? Where are synchronization points? Comment on the communication overhead you expect and whether you expect load imbalance issues. Propose methods to hide these latencies.

Outline

- Brief summary of the step to get the final MP2 energy (Artur)
- Optimization of the sequential code
  - Naive implementation (Artur)
  - Implementation of with BLAS (two version for further optimization (Artur)
- Roofline plot to show the current working baseline model (Olga)
  - Show the calculation/assumption – similar to HW2. Focus on the variable.
  - Comment on the naive implementation and BLAS
- Suggestion for parallelize design
  - Shared memory model (OpenMP) - parallelize each of the for loop in step 1,2, and 3 (Mouza)
  - Small vectorization can be done for step 3 where we unroll the two most inner loop (Mouza))
  - Distributed memory model (MPI for the first two steps) – We will potentially use MPI AlltoAll to get the right matrix multiplication for step 1 and 2. If we have enough time then we will move on to step 3 with MPI Reduction. (Hieu)
- Challenges (Hieu)
  - File readings
  - Reordering the data to do MPI AlltoAll

Challenges

- Read files and distribute to multiple ranks