Final Project Presentation MP2 Calculation

Team 10
Hieu, Mouza, Olga, Artur

Intro

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

(Orthogonalization)

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

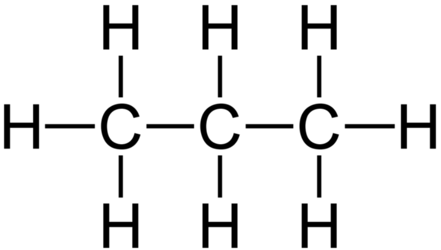
Step 3: Calculate MP2

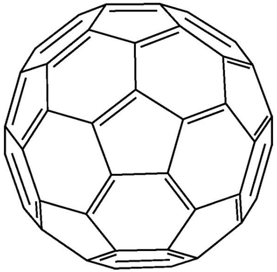
$$E_{MP2} = \sum_{ij}\sum_{ab} \frac{(ia \mid jb)[2(ia \mid jb) - (ib \mid ja)]}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b}$$

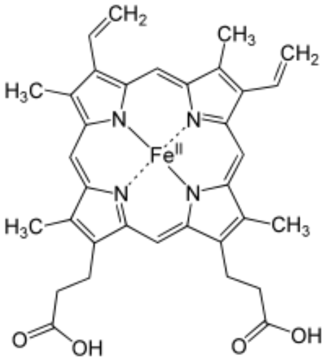
$(C^{\dagger})$	NBASIS NBASIS
$(\mu\nu\|P)$	NBASIS NBASIS NAUX
$(P\|Q)^{-1/2}$	NAUX NAUX
MO_energy	NBASIS

Systems of Interest

	C_3H_8	C_{60}	Heme
N_{BASIS}	202	1800	632
N_{AUX}	483	4860	1618
N_{ELEC}	13	180	161
Memory	157 MB	110 GB	5.17 GB
Total node hours	2.96 s	6.61 hours	0.0903 hours







Serial Implementation

Use BLAS Level 3 DGEMM to optimize serial code

1)

MO1 and MO2

```
for (size_t P = 0; P < NAUX; P ++){
        cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, NBASIS, NBASIS, NBASIS, 1.0,
                    C, NBASIS, uqP + P *NBASIS *NBASIS, NBASIS, 1.0, pqP+ P *NBASIS *NBASIS, NBASIS);
    }
```

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

2)

3)

Serial Implementation

Use BLAS Level 3 DGEMM to optimize serial code

1)

MO1 and MO2

```
for (size_t P = 0; P < NAUX; P ++){
        cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, NBASIS, NBASIS, NBASIS, 1.0,
                    C, NBASIS, uqP + P *NBASIS *NBASIS, NBASIS, 1.0, pqP+ P *NBASIS *NBASIS, NBASIS);
    }
```

2)

Orthogonalization

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

Orthogonalization

```
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)

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

Serial Implementation

Use BLAS Level 3 DGEMM to optimize serial code

1)

MO1 and MO2

```
for (size_t P = 0; P < NAUX; P ++){
        cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, NBASIS, NBASIS, NBASIS, 1.0,
                    C, NBASIS, uqP + P *NBASIS *NBASIS, NBASIS, 1.0, pqP+ P *NBASIS *NBASIS, NBASIS);
    }
```

2)

Orthogonalization

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

Orthogonalization

```
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)

MP2 calculation

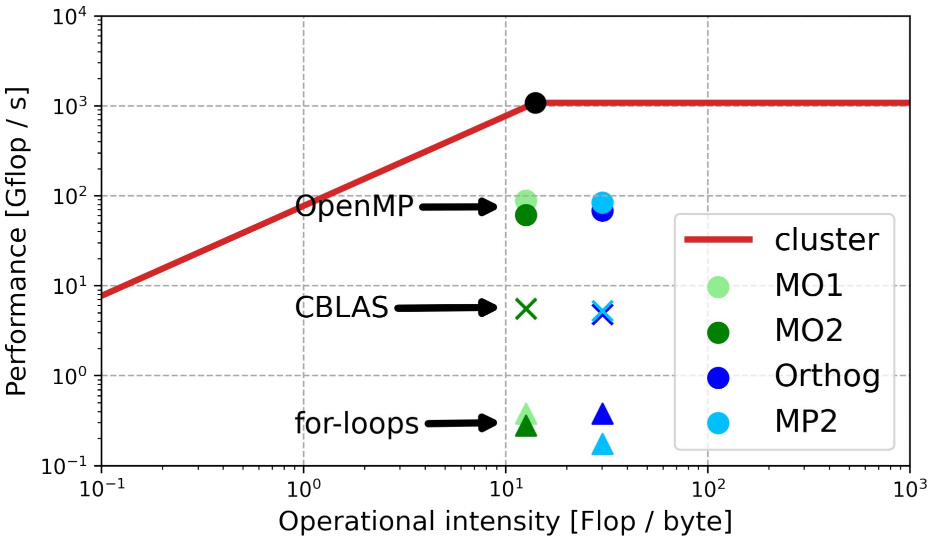
```
    for (size_t i = 0; i < NELECS; i++){
        for (size_t j = 0; j < NELECS; j++){
            int NVIR = NBASIS - NELECS;
            double *ipjq = new double[NVIR * NVIR] ();
            // contract over P
            cblas_dgemm(CblasRowMajor, CblasTrans, CblasNoTrans, NVIR, NVIR, NAUX, 1.0, B_iaP +
                        i*NVIR*NAUX, NVIR, B_iaP+j*NVIR*NAUX, NVIR, 1.0, ipjq, NVIR);

            // 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

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



Roofline Analysis

$$F_{\mathrm{MO1,\,2}} = 2 \times N_{\mathrm{BASIS}}^3 \times N_{\mathrm{AUX}}$$

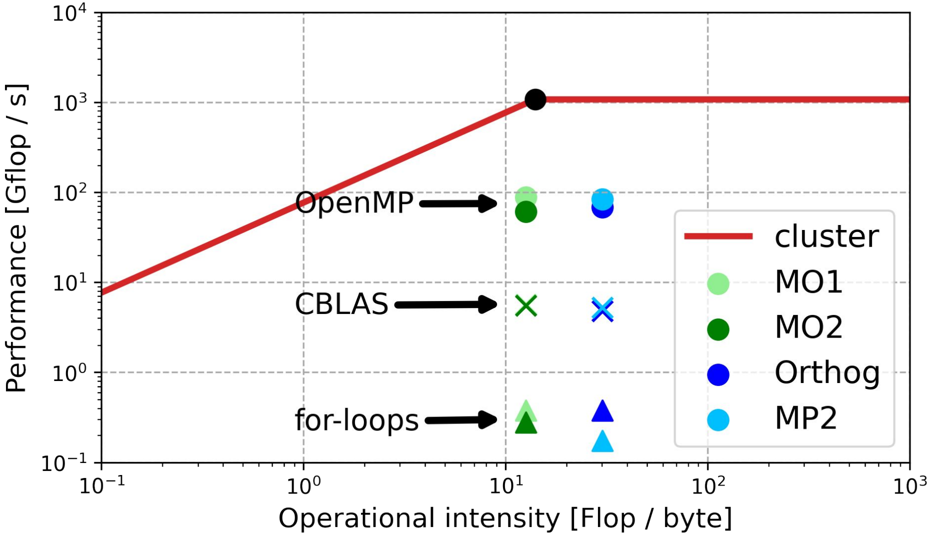
$$F_{\mathrm{Orthog}} = 2 \times N_{\mathrm{AUX}}^3 \times N_{\mathrm{BASIS}}$$

$$F_{\mathrm{MP2}} = 2 \times (N_{\mathrm{BASIS}} - N_{\mathrm{ELECS}})^2 \times N_{\mathrm{AUX}} + (N_{\mathrm{BASIS}} - N_{\mathrm{ELECS}})^2 \times 11)$$

$$I_{\mathrm{MO1,\,2}} = \frac{F_{\mathrm{MO1,\,2}}}{8 \cdot 4 \cdot N_{\mathrm{BASIS}}^2 N_{\mathrm{AUX}}} = \frac{1}{16} N_{\mathrm{BASIS}}$$

$$I_{\mathrm{Orthog}} = \frac{F_{\mathrm{Orthog}}}{8 \cdot 4 N_{\mathrm{AUX}}^2 N_{\mathrm{BASIS}}} = \frac{1}{16} N_{\mathrm{AUX}}$$

$$I_{\mathrm{MP2}} = \frac{F_{\mathrm{MP2}}}{3 N_{\mathrm{BASIS}}^2} = \frac{1}{16} N_{\mathrm{AUX}}$$



Roofline Analysis: BLAS routines

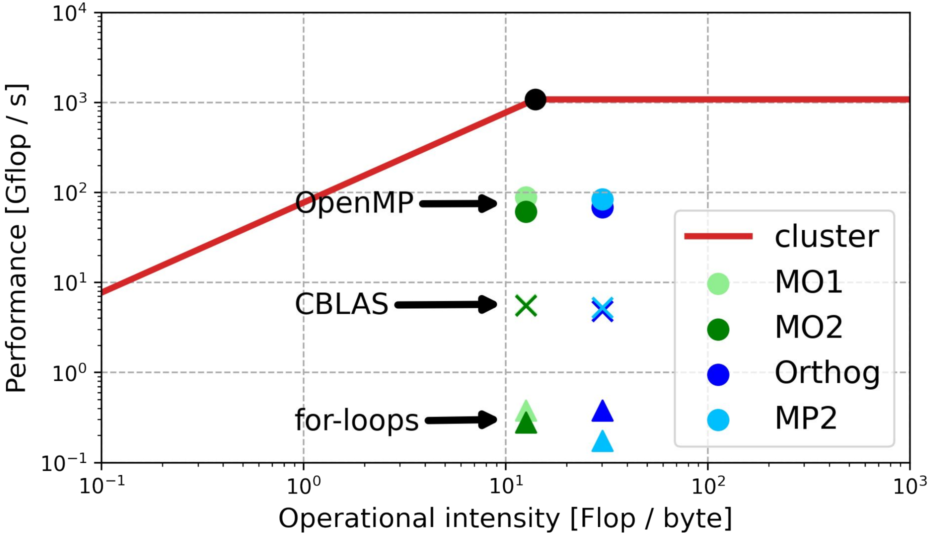
$$F_{\mathrm{MO1,\,2}} = 2 \times N_{\mathrm{BASIS}}^3 \times N_{\mathrm{AUX}}$$

$$F_{\mathrm{Orthog}} = 2 \times N_{\mathrm{AUX}}^3 \times N_{\mathrm{BASIS}}$$

$$F_{\mathrm{MP2}} = 2 \times (N_{\mathrm{BASIS}} - N_{\mathrm{ELECS}})^2 \times N_{\mathrm{AUX}} + (N_{\mathrm{BASIS}} - N_{\mathrm{ELECS}})^2 \times 11)$$

* cache optimization

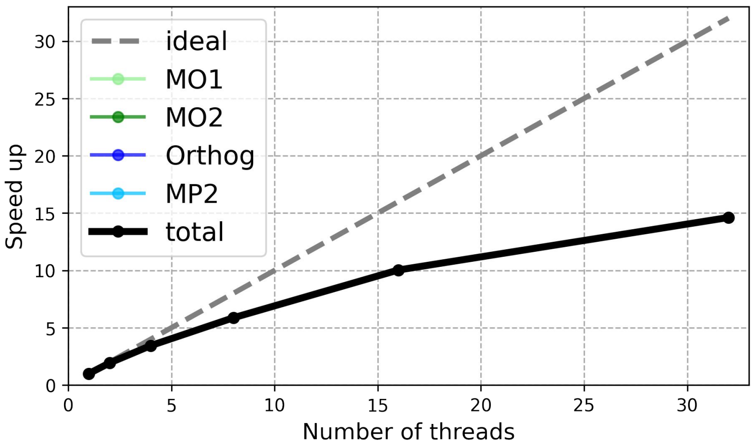
Step	for-loops	CBLAS	OpenMP
MO1	20.99	1.42	0.09
MO2	28.67	1.45	0.13
Orthogonalization	50.54	3.29	0.28
MP2	34.04	1.18	0.07



Performance Benchmarks: OpenMP implementation / strong scaling

#pragma omp parallel for

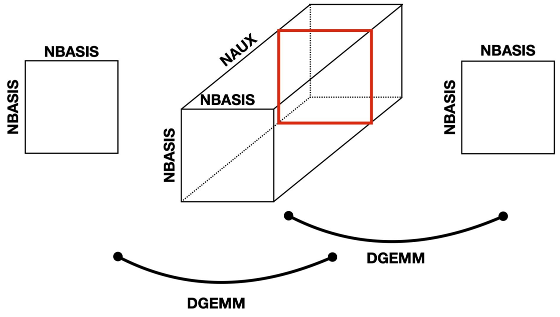
- scheduling
- **first touch policy**
- **thread affinity**

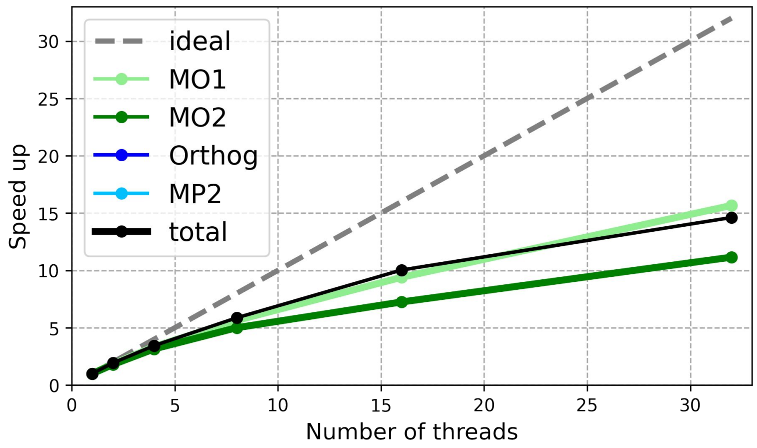


Performance Benchmarks: OpenMP implementation / strong scaling

#pragma omp parallel for

- MO2 has worse cache locality





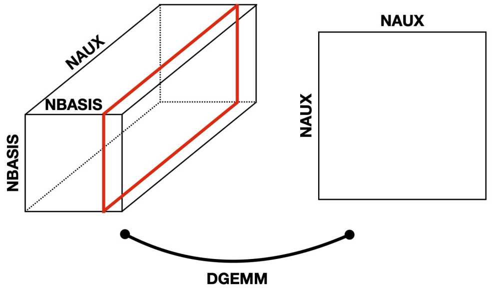
Performance Benchmarks: OpenMP implementation / strong scaling

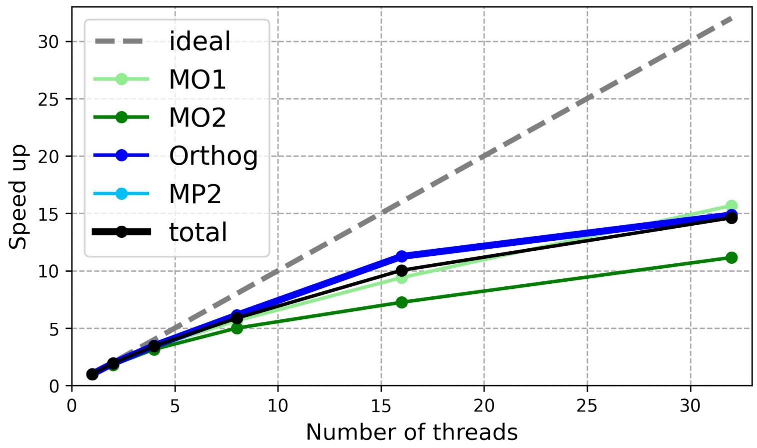
#pragma omp parallel for collapse(3)

- reshaping

#pragma omp parallel for

- DGEMM





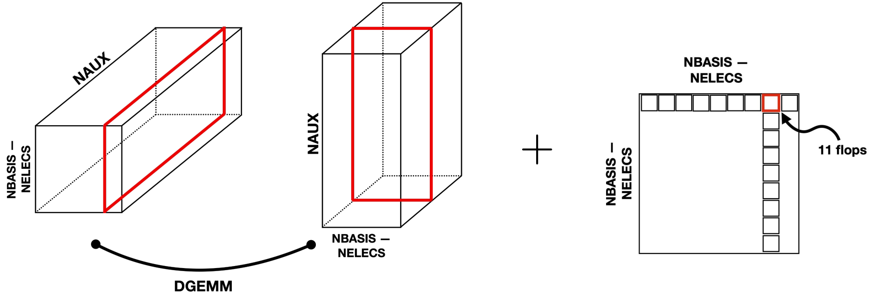
Performance Benchmarks: OpenMP implementation / strong scaling

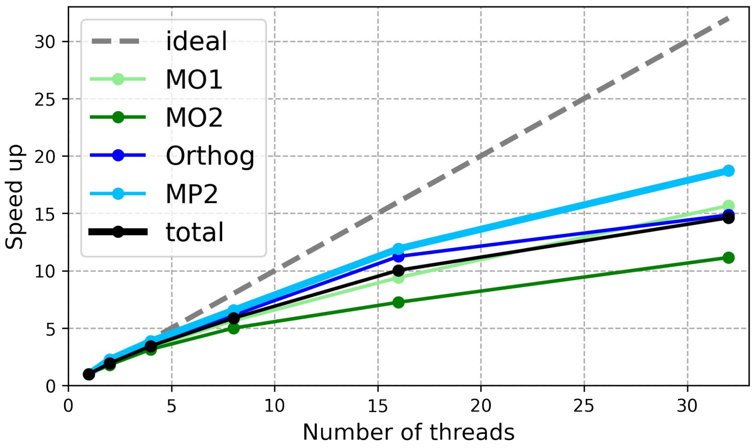
#pragma omp parallel for collapse(3)

- reshaping

#pragma omp parallel for collapse(2) reduction(+:E_{MP2})

- DGEMM + element-wise calculations





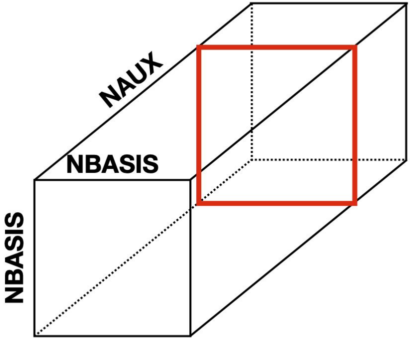
Distributed Memory Parallelization with MPI

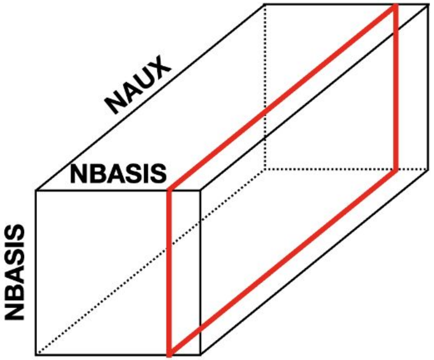
- Rank three tensor can be distributed to each rank and the change of basis can be done separately on each node.

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

- Orthogonalization can also be done separately on each node after some data transfer

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



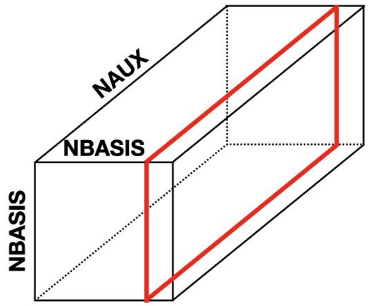


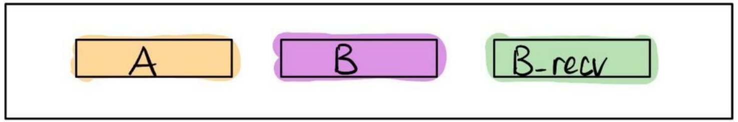
Distributed Memory Parallelization with MPI

Algorithm 4: Non-blocking MPI Implementation

Initialize 1D Cartesian Grid;
for $k = 1; k < NumRanks$ **do**
  MPI_Isend(B) to the RIGHT neighbor;
  MPI_Irecv(Brecv_) from the LEFT neighbor ;
  MP2 Energy Calculation kernel (3) + Update index ;
  MPI_Waitall(2);
  Swap B and Brecv_ ;
end
MP2 Energy Calculation kernel (3);
MPI_Reduce(&MP2_partial, &MP2_energy, root rank);

- Allocate extra memory buffer to do compute/transfer overlap
- The number of MPI Communication is minimized.
- Need an extra bookkeeping index to get the correct MO energy.





Weak scaling analysis for MPI Implementation

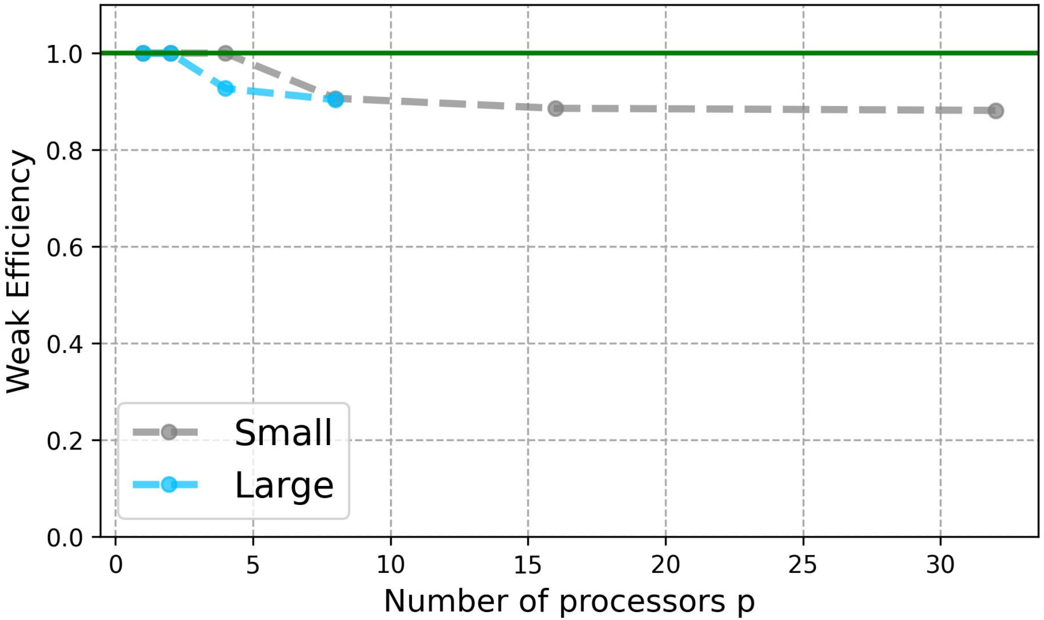
Number of processors	1	2	4	8	16	32
Time for small test	0.17	0.34	0.68	1.50	3.07	6.17
Time for large test	5.37	10.56	23.17	47.54	_	_

- Each processor is map to one core on one node
- Runtime needs to be normalized for weak scaling analysis

Weak scaling analysis for MPI Implementation

Number of processors	1	2	4	8	16	32
Time for small test	0.17	0.34	0.68	1.50	3.07	6.17
Time for large test	5.37	10.56	23.17	47.54	_	_

- Each processor is map to one core on one node
- Runtime needs to be normalized for weak scaling analysis



Final thoughts

- **Integrate BLAS/OpenMP Implementation** into MPI implementation
- Modify the MPI implementation to **divide the work between different processors.**
- Continue working on optimizing the vectorized kernel for MP2 energy calculation.

Final thoughts

- **Integrate BLAS/OpenMP Implementation** into MPI implementation
- Modify the MPI implementation to **divide the work between different processors.**
- Continue working on optimizing the vectorized kernel for MP2 energy calculation.

- Ab initio calculation requires a lot **computing resources**
- MPI implementation is useful when we do ab initio calculation for **solid state system** (discretize k point in the 1st Brillouin zone)