MPI 及び OpenMP を用いた並列処理行列計算の性能評価

Performance evaluation of parallel processing matrix calculations using MPI and OpenMP

Abstract - In this paper, I applied parallel programs using OpenMP and OpenMPI to matrix calculations and investigated their execution time trends.

Keywords : OpenMP, MPI, Matrix calculation

1. はじめに

行列計算はあらゆるところで用いられる．3DCG 分野ではアファイン変換，機械学習分野では学習させるデータ数が非常に多くなり，通常の連立方程式を用いた計算では手に負えなる．そのために行列を用いた計算が利用される．しかし，大量のデータや変数を扱う行列計算を愚直に行ってしまえば途方もない計算時間を要してしまい，プログラムのパフォーマンス低下を招いてしまう可能性が生じる．

そこで本稿では行列計算の並列化における性能向上についての調査を行った．2節で今回利用した環境の紹介．3 節で行列計算アルゴリズムの概要説明．4 節で逐次処理プログラムと N×N 行列の逐次処理における実行結果の確認．5 節で並列化によって性能改善を目指すプログラムの実装方法の説明．6 節で MPI と OpenMP で並列化したプログラムの性能評価や逐次処理と比べた性能評価割合などの説明を行う．7 節で今回並列化プログラムによって得られた結果から考えられることや今後の課題について述べる．

2. 評価環境

評価に利用した CPU は IntelCore i5，1.4GHz，4 コア．メインメモリ 16GB，2 次キャッシュ(コア単位)256KB，3 次キャッシュ 6MB．コンパイラ clang version 13.0.0．コンパイルオプション -fopenmp，OS は macOS Monterey version12.4 となる．

3. 行列計算アルゴリズム

3.1 概要

今回の行列計算は行列 A と行列 B を掛け合わせた値を ans という変数に入れ，値を足し続ける行列積プログラムであり，数式は ans += $\sum_{k=0}^{N} A_{ik}B_{kj}$ になる．

3.2 行列計算の概要

表記法は行列 A がサイズ N×N の正方行列．行列 B も同様に，サイズが N×N の正方行列である．N の値は 100～800 まで 100 刻みで変化させている．プログラミング言語は C 言語で記述した．

4. 逐次処理性能評価

図 1 に今回実装した逐次処理プログラム概要を示す．

```
1 for(int i = 0; i < N; i++){
2   for(int j = 0; j < N; j++){
3     for(int k = 0; k < N; k++){
4       ans += A[i][k] * B[k][j];
5     }
6   }
7 }
```

図 1 のプログラムでは A と B の行列積を計算している．図 2 に逐次処理時の実行時間推移を示す．

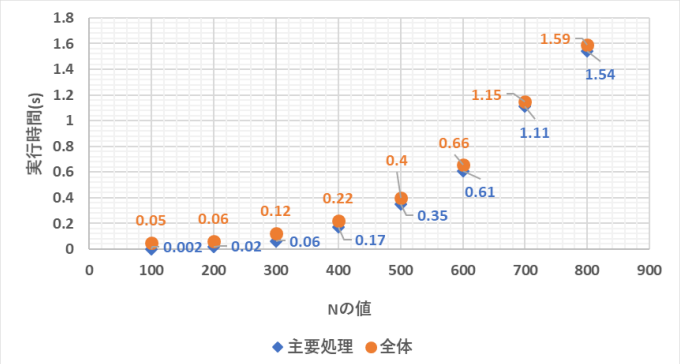

図 2 逐次処理時の実行時間の推移グラフ

図 2 より N の大きさが増えていくたびに実行時間が増加していることが分かる．

5. 並列化プログラムの概要

並列化プログラムの性能向上目標として M ノード時の実行時間が 1 ノード時に比べて 1/M の実行時間になるよう目標を定めた．また並列化プログラムを行う上で，行列計算をマルチコア CPU の複数コアを利用し，複数のノードでプログラムを行う MPI を用いた並列化プログラムの実装と OpenMP を用いたスレッド並列プログラムの

実装も行った.

6. MPI と OpenMP を用いた並列プログラム性能評価

6.1 MPI 性能改善結果

図 3 に MPI を利用し並列処理を行うためのプログラム概要を示す.

```
1  int myrank; // 自ノード番号を入れる変数
2  int size; // 使用するノード数を入れる変数
3  for(int i = N/size *myrank; i < N/size*(myrank+1); i++){
           // 逐次処理時の 2~6 行目
9  }
```

図 3

図 3 のプログラムは外側の for ループで並列化を試みている．2 ノード時，N＝100 とすると 0~49，50~99 までの計算がノード 1 とノード 2 でそれぞれ並列化し計算される．次に図 4 に図 3 で用いた並列化プログラムの実行時間結果を示す.

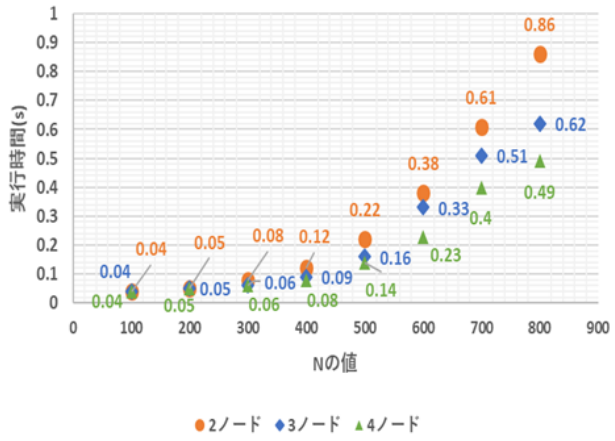

図 4 並列プログラムの強スケーリング結果

図 4 は MPI を用いた行列並列計算の実行時間推移グラフになる．2 ノード，3 ノード，4 ノード時の場合を調査した．図 4 を見ると N＝100 から N＝400 まではノード数によって実行時間が大きく変わることはなかった．しかし, N = 800 の場合の実行時間は 2 ノードと 4 ノードでは約 1.75 倍の実行時間の差がある.

以上より MPI 並列化プログラムのノード数増加における性能向上は問題サイズが小さい場合, 並列プログラムの恩恵を得られにくい．しかし，サイズが大きくなると並列化のプログラムの恩恵を得やすいという結果が分かる．一方，2 ノードと 4 ノードの実行時間は 1/2 になることはなかった．ということがこのグラフから分かる.

6.2 OpenMP の性能改善結果

OpenMP は MPI と異なり pragma と呼ばれる指示行を書き込み，スレッド並列化を行う．1 行目で並列化するプログラムブロックを指定し，5 行目で並列化を行いたい for 文を指定している．図 5 には OpenMP のプログラム全体の概要．図 6 に OpenMP と MPI の性能比較を示す.

```
1  #pragma omp parallell
2  {
3     int id = omp_get_thread_num();
4     int max = omp_get_num_threads();
5     int start = id * N / max;
6     int end = (id+1) * N / max;
7  #pragma omp for
8     {
9           for(int i = start; i < end; i++){
               // 逐次処理時の 2~6 行目
15    }
16 }
```

図 5

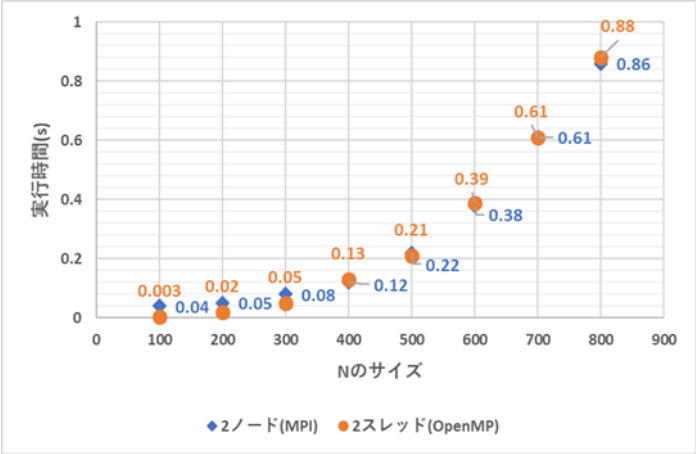

図 6 OpenMP と OpenMPI の実行時間推移グラフ

図 6 を見ると MPI と OpenMP の実行時間に大きな差はなかった．ただ N の値が小さい時(100~300)の場合は OpenMP の実行時間が若干速くなった．この理由として OpenMP の並列化はスレッドを用いているが，MPI はノードで分けるときに通信処理を行っている．そのため通信処理時間の分の MPI 実行時間が若干多くなったと考察される．また MPI の並列プログラムと同様に，OpenMP のスレッド並列化も逐次処理時の 1/M の実行時間になることはなかった.

7. まとめ

本稿では行列計算プログラムを逐次処理を MPI と OpenMP によって並列化し実行速度の向上を目指した．N=800 時に逐次処理が 1.59 秒, MPI では 0.86 秒, OpenMP は 0.88 秒となり，当初の実行時間削減目標の 1/M の実行時間にはいたらなかった．しかし，MPI と OpenMP は愚直な並列化プログラムでもある程度実行速度の向上が見られた．また，本プログラムの実行速度向上の手法として for 文内の外側ループアンローリングの 2 段展開やブロック化アルゴリズムを用いるなどの for 文内処理の高速化が考えられる.