並列化手法を用いた流況シミュレーションの高速化

解析事業部　防災情報部

菊野　拓人

原井光一郎

1. はじめに

科学技術計算において計算機性能の向上は解析モデリングの精緻化を果たす上で重要な要素となっており、その性能が飛躍的な向上をしてきたことは周知のことと思われる。例えば、マイクロプロセッサーの性能は約 40 年の間に 32 万倍の性能向上が得られたという報告もある。[1)]

この性能向上に比例して数値シミュレーションにかかる計算時間も自動的に短縮化が図れてきたことは容易に想像できる。しかしながら、このマイクロプロセッサーの性能は、消費電力の問題等により、これまでの性能向上と比べると頭打ちの状態となっている。

一方、数値シミュレーションを扱うような実務面においては、計算領域の拡大、空間分解能（Δx）の細分化（小メッシュ、多層空間）、Δx の細分化に伴う時間刻み（Δt）の細分化など計算容量および計算時間が増大するような要求事項が増えてきており、今後もその傾向は続くものと思われる。

これらの背景を踏まえると、効率よく計算を行うためには、ハード面での整備だけでなく数値シミュレーションに用いる計算プログラムを並列化していく等、ソフト面での整備が必要であると考えられる。

この並列化手法はこれまでに様々な方法が紹介されている（表 1）。代表的な手法として、PC クラスタのような分散メモリ型の計算機環境で実行可能な MPI(Message Passing Interface)が挙げられる。また、CPU のマルチコア化が進み、共有メモリ型の計算機環境で普及した並列化手法として OpenMP がある。更に近年注目される手法として画像処理のハードウェアである GPU（Graphics Processing Unit）を数値計算に利用する GPGPU（General-Purpose computing on GPUs）が挙げられる。

以上のような背景を踏まえて、本稿では社内の計算機環境のように身近な計算資源で数値計算の高速化を図ることを目的とし、MPI および OpenMP を用いて既存の流況シミュレーションモデルの並列化を試みた。

表 1　並列化手法の例

メモリアーキテクチャ	名称
分散メモリ型	MPI
	HPF
	XcalableMP
共有メモリ型	OpenMP
	Pthreads
―	GPGPU

2. 計算モデル

本稿で並列化を試みる流況シミュレーションモデルは非圧縮性流体に対する Navier-Stokes の運動方程式と流体の連続式を基礎式とした二次元

多層非定常モデルによる海域の潮流計算モデルである。また、流入河川を考慮するため、水温、塩分に関しては Fick の拡散方程式を基礎式とする平面二次元拡散モデルを用いて同時に解析している。なお、解析モデルに用いられているプログラミング言語は fortran である。

モデル化の対象は東京湾全域とし、格子分割数は水平方向に(東西)152×(南北)208 格子、鉛直層分割数は 5 層とした。座標系は直交直角座標とし、格子幅(□x)は 300mである。図 1 に格子分割図を示す。

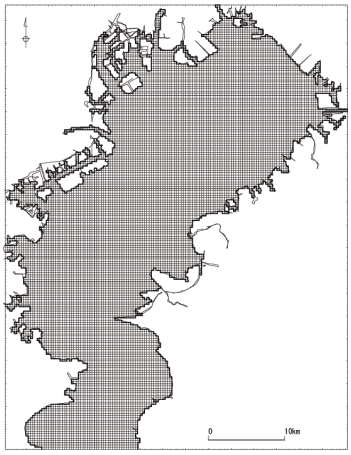

図 1 格子分割図

3. 並列化の特徴

並列計算は実行する計算機環境によって「プロセス並列」、「スレッド並列」の 2 種類に大別できる。プロセス並列は、クラスタなどの分散メモリ環境で処理が可能であり、代表的な手法として MPI がこれにあたる。一方、スレッド並列は共有メモリ環境で処理が可能であり、代表的な手法として OpenMP がこちらにあたる。

本稿では、これら MPI および OpenMP を用いて、2 章で述べたシミュレーションモデルの並列化を試みた。なお、並列計算は並列実行するプロセッサ数に計算領域を分割すること(データ並列)で行った。以下に、各並列化手法について特徴およびプログラミングの留意点について述べ、その後に並列化の概要について説明する。

3. 1 MPI(Message Passing Interface)[2)]

プロセス並列による計算の流れのイメージを図 2 に示す。分散メモリ環境は各プロセッサ(CPU)がそれぞれ独立したメモリを保有し(この状態をノードという)、ネットワークを介してつながっている。この環境で並列計算を行うと、各ノードは割り当てられたデータ量について計算を実行するが、あるプロセスが別のプロセスのデータを参照するような場合、ノード間でデータを通信する必要がある(図3)。このようなノード間でデータ(メッセージ)の通信を行うための関数(ルーチン)をライブラリとして提供している代表的な規格が MPI である。

領域分割(データ並列)によって MPI 並列計算を行う場合、各ノードが担当する計算量は

(全体のデータ量)÷(領域分割数)

となる。したがって、並列化により各ノードの計算量が節約され、かつそれらが並列に実行されるため計算の効率化が図れる。ただし、MPI プログラミングの注意点として以下のことが挙げられる。上述のようにあるプロセスが別のプロセスのデータを参照する場合、MPI 通信関数を指定する必要があるが、この関数を正しく指定しなければ誤った解を算出するおそれがある。また、MPI によって効率よく計算を行うためには、このデータの通信量を減らすようなプログラミングが必要である。したがって、対象

とする計算の規模(格子数、タイムステップ等)、地形形状などによってデータの分割方法を考慮する必要がある。さらに、各ノードに割り当てるデータ量に差があると、ロードバランスが不均等になり効率的な計算が行えない。したがって、各ノードに割り当てるデータにできるだけ偏りが無くなるような領域分割が必要である。

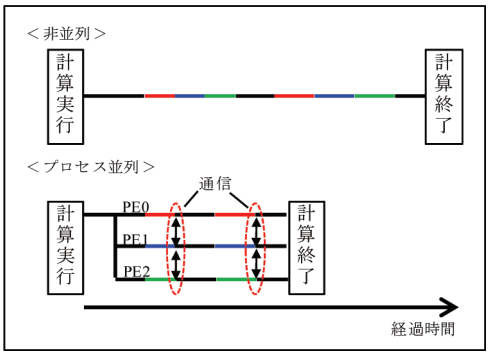

図 2 プロセス並列のイメージ

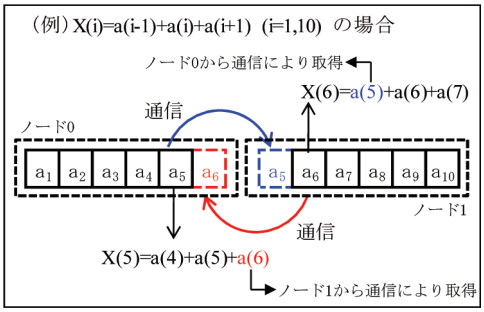

図 3 データ通信の例

3. 2 OpenMP[3)]

OpenMP はプログラム中にコンパイラへの並列化指示文を挿入(スレーブスレッドの生成)することで並列計算が可能になる。また共有メモリ上で計算を行うため、各プロセスは同じメモリ上のデータにアクセスでき、MPI のようにノード間のデータ通信を意識したプログラミングの必要がない。ただし、スレーブスレッドを生成することで、すべての変数は各スレッド内だけでアクセスできる変数(プライベート変数)もしくはすべてのスレッドで共有な変数に分類されるため、各変数がどちらにあたるかを定義する必要がある。

OpenMP プログラム(スレッド並列プログラム)の計算の流れを図 4 に示す。計算を実行するとマスタースレッドによりファイルの入力など逐次計算が行われる。そして、OpenMP 指示文を挿入した部分に達すると、予め設定した数のスレッドが立ち上がり並列計算を実行する。並列計算が終了するとマスタースレッドに戻り逐次計算に戻る。したがって、計算負荷の大きい do ループのみを並列化するなど、ユーザーによって意図的な並列化が可能である。ただし、do ループのたびに指示文を挿入するなど計算粒度が小さくなる並列化は計算効率を悪くするおそれがある。これは、スレッドの立ち上げが計算上のオーバーヘッドとなるため、粒度が小さければオーバーヘッドも増大するためである。したがって、OpenMP による並列化は、計算粒度をなるべく大きくするプログラミングが必要となる。(ただし、計算粒度が粗すぎると、計算負荷の不均衡により計算性能が低下するおそれがある。)

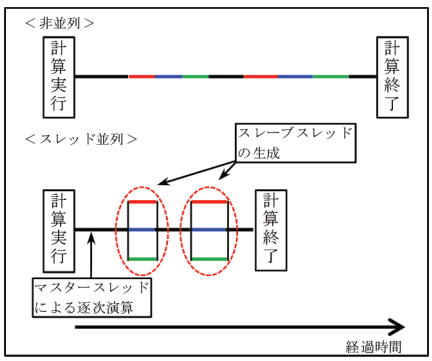

図 4 スレッド並列のイメージ

3. 3　並列化概要

並列計算の方法は、並列プログラミングが比較的容易であること、かつ効果的な計算速度の向上を期待してデータ並列による計算を採用した。データ並列による計算は、各プロセス(スレッド)に担当するデータを割り当てることで行う。本稿では、計算領域を並列実行するプロセス(スレッド)数に分割し、分割した領域のデータを各プロセス(スレッド)に割り当てることで並列計算を行うようにした。ここで計算領域の分割は、図 1 に示した計算範囲を y 方向(南北方向)に分割することで行った。これは、MPI は 3.1 節で述べたように、隣り合う領域と接続する境界位置のデータを互いに通信する必要があり、本稿のモデル領域が南北方向に格子数が多い配置となっているため、このように分割することで通信量を抑えることができると考えられるためである。ここで領域を x、y 方向それぞれ 2 分割した場合の通信量のイメージを図 5 に示す。

OpenMP では、プログラム内の y 方向に関する do ループを並列に実行するよう指示文を挿入し、計算を行った。

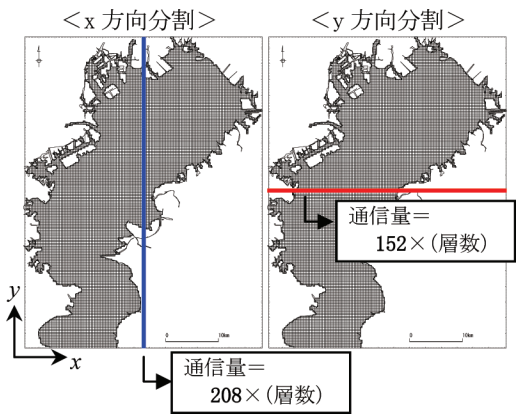

図 5　データ通信量

4. 並列化による性能検討

4. 1　検討条件

計算を実行するにあたり、使用した計算機環境を表 2 に示す。計算はマルチコア PC(1 台)の環境下で実施した。まず、基準となる計算結果算出のために 1 プロセスのみを使用した計算を実行した。並列化による性能検討のために実施した並列計算のプロセス数(スレッド数)は MPI、OpenMP 共に 2,4,8 プロセス(スレッド)とした。

表 2　計算環境一覧

OS	Windows Vista Ultimate SP1
CPU	Core i7 CPU 870 2.93GHz
RAM	8.00GB
コンパイラ	Intel Fortran 11.0
MPIライブラリ	Intel MPI Library 4.0

並列化に伴い、各手法の計算精度の検証を行い、計算時間の変化について検討した。計算精度は、MPI はデータの通信が不十分な場合、OpenMP は変数の扱い方(プライベート/共有変数の指示)などに誤りがある場合、逐次計算を進めると誤差(1 プロセスの結果との差)が蓄積し、最終的に過った解を得ることになる。したがって、基準となる 1 プロセスの計算結果と並列化によって得られる計算結果との差を確認した。並列化によって得られた結果に問題が無いことを確認した上で、計算速度の比較による並列化の性能検討を行った。流況シミュレーションの計算条件を表 3 に示す。

表 3　計算条件

対象潮汐	平均大潮 (M2+S2分潮) (＝1周期あたり12時間)
計算期間	助走計算期間：12時間 (1周期) 周期計算期間：60時間 (5周期) 計 72時間
タイムステップ	2.0 秒

4．2　計算精度の検証

並列化による計算精度の確認を行った。確認方法は、まず計算実行中に逐次出力するモニター地点の数値結果リストを 1 プロセスによる計算時に出力されたリストと比較した。また、計算終了後は 1 プロセスによる計算結果と各並列計算結果の差（流速、水温、塩分）をとることで行った。以上の確認を、実施したすべての並列計算結果について行ったところ、結果に差が無いことが確認できた。これにより、MPI、OpenMP 共に正しく並列化プログラミングが行えていることを確認できた。

4．3　並列化による性能検討

続いて、各並列化による計算速度の検証を行った。まず、計算速度の速度向上率 S を、

$$S=\frac{1プロセス使った場合の計算所要時間}{並列化による計算所要時間}$$

とし、表 4 に各並列パターンの計算時間およびその速度向上率を示し、計算時間のグラフを図 7 に示した。プログラムには非並列化部分が存在するため単純にプロセッサ数（スレッド数）倍の効果は得られない（アムダールの法則）。本計算では、MPI について 2 プロセスで 1.7 倍、4 プロセスで 2.9 倍の速度向上率となり、一定の効果が得られたものと思われる。ただし、8 プロセスの場合、4 プロセスの場合と計算所要時間はほぼ同じであり、効果が得られにくかった。この原因として、使用した計算マシンのプロセッサが 4 コアのハイパースレッディングであり、高負荷な状況では HP の効果が得られず、8 スレッドの処理が同時に出来なかったためと考えられる。OpenMP においては、MPI 程度の計算効率は得られなかったものの、8 スレッドまで順調に計算速度の短縮が図れている。本稿では、OpenMP による並列化は、モデルの中でも計算負荷が大きいと考えられる部分（運動方程式、拡散方程式）について行った。したがって、OpenMP を用いて更なる計算効率化を図るためには、モデル全体の並列可能部分の見直しや計算粒度を考慮した並列化などオーバーヘッドを減らすプログラミングを行うことで可能になると考えられる。

表 4 計算所要時間と計算速度向上率

プロセス数（スレッド数）	MPI		OpenMP	
	計算所要時間（分）	速度向上率	計算所要時間（分）	速度向上率
1	49.5	—	49.5	—
2	28.6	1.7	36.9	1.3
4	17.3	2.9	32.0	1.5
8	16.2	3.1	26.0	1.9

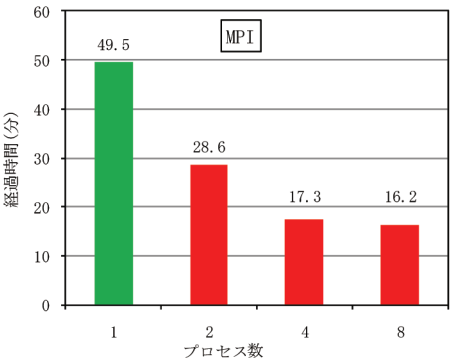

図 7-1 MPI モデルの計算所要時間

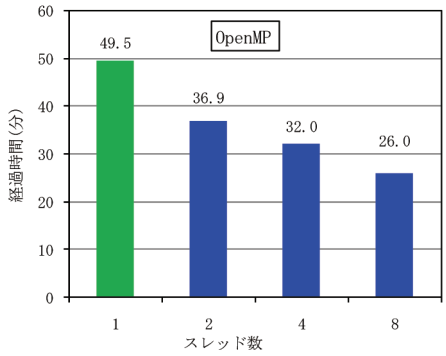

図 7-2 OpenMP モデルの計算所要時間

5. まとめと今後の課題

本稿では、社内の PC 環境など身近な計算資源で数値シミュレーションの高速化を図ることを目的に、マルチコア PC を用いて既存の流況シミュレーションモデルを MPI および OpenMP によって並列化した。その結果、MPI、OpenMP 共に計算時間を短縮することができた。特に、MPI は OpenMP よりも計算速度向上率が高く、身近な計算資源環境において、より効率的に数値計算が実施できる並列化手法と考えられる。また、プロセス間通信を極力減らすようなデータの分割（MPI）、計算粒度が小さくならないようにすること（OpenMP）など各並列プログラミングを改善することで、本稿と同じような計算機環境において更なる効率化が可能であると考えられる。

また、本稿のシミュレーションモデルの並列化および更なる高速化を図るために今後の課題として以下のことが挙げられる。

計算モデルの並列化に関しては、今回の計算は東京湾のように南北方向に長い計算地形であったため、同方向に分割することで便宜的にデータの通信を抑えるようにした。しかしながらデータ通信量を極力抑えることや各プロセスのロードバランスを均等に保つための最適な領域分割方法は、本稿の地形条件と違い東西方向に長い場合や水深に偏りがある場合が想定されるように、対象とする地形条件によって変わってくる。したがって、データの分割方法を、対象とする地形ごとに設定および計算できるような並列プログラミングが必要であると考えられる。

計算時間の高速化の課題としては次のことが考えられる。今回行った計算では、1 台のマルチコア PC を使って並列計算を実施した結果、MPI において、4 プロセスまでの計算速度向上率の増加割合と比べて 8 プロセスのそれは低減する結果となった。このことを解消するために複数 PC を繋げたクラスタの構築が必要であると考えられる。

また、はじめに述べた GPU を用いた並列計算（GPGPU）や、MPI および GPGPU を組み合わせた並列計算などにより、更なる計算高速化が図れるものと考えられる。

以上の課題を改善することで、今後、より計算負荷の大きい問題に対して効率的にシミュレーションを実施できることが期待される。

＜参考文献＞

1）池井満、林浩史、田中智子「インテル Parallel Studio プログラミングガイドーParallel Composer Parallel Inspecter Parallel Amplifier の活用ー」（2009, 株式会社カットシステム）

2）青山幸也 「並列プログラミング入門 MPI 版」（2008,理化学研究所情報基盤センター）

3）牛島省「OpenMP による並列プログラミングと数値計算法」（2006,丸善株式会社）