Vol.2017-HPC-161 No.9
2017/9/20

SMW 公式を用いた逆行列計算における 通信のタイリングによる通信時間隠蔽効果の評価

藤田　侑弥[1]　松井 祐太[1]　福間 慎治[1]　森 眞一郎[1]

概要：本研究では, 多くのシミュレーション中に現れる線形方程式 Ax=b の求解問題に着目する. 係数行列 A が時間変化する線形方程式の高速求解法として我々は SMW 公式を用いて, 時間変化後の A の逆行列をハイブリッド並列処理により高速に求解する手法を提案してきた. 本研究ではさらなる高速化として通信を複数に分け送信することで通信と計算のオーバーラップ実行を可能とした. これをタイリングと呼び, タイリングによる通信の分割数は 2 のべき乗で行うことができる. 本手法について評価を行うためハイブリッド並列処理で用いるノード間並列数及び, 通信のタイリング数による通信時間の隠蔽効果を通信速度が 10Gbps,1Gbps,100Mbps のそれぞれについて評価を行った. タイリングを行うことで従来までのタイリングを行わない場合に比べ高速化が確認できた.

1. はじめに

本研究では, 多くのシミュレーション中に現れる線形方程式 $\boldsymbol{Ax} = \boldsymbol{b}$ の求解問題に着目する. 特に, シミュレーション中のユーザからの入力や時間経過によって係数行列 $\boldsymbol{A}$ の要素の一部のみが変化する状況での時系列シミュレーションの実現を研究のターゲットとする. このように係数行列 $\boldsymbol{A}$ が時間変化する線形方程式の高速求解法として我々は SMW 公式 (Sherman-Morrison-Woodbury formula)[1] を用いて時間変化後の $\boldsymbol{A}$ の逆行列をハイブリッド並列処理により高速に求解する手法を提案してきた [2]. 本研究では, 更なる高速化のため, 従来までの通信と計算のオーバーラップ実行を改善しより多くのオーバーラップ実行を行うことで高速化を目指す. そのために通信の最粒度化を行い通信と計算のオーバーラップ実行の割合を増やした. 通信を複数に分割することによる高速化を確認するため問題サイズ $\boldsymbol{n}$ や微小変化のサイズ $\boldsymbol{s}$, 並列数, 通信速度によって本手法における影響を調査し評価を行った.

2. 研究背景

2.1 SMW 公式

SMW 公式とは，Sherman-Morrison-Woodbury formula の略称であり, 以下のような式で表される.

ただし，ここでの $\boldsymbol{A}$，$\boldsymbol{B}$，$\boldsymbol{C}$，$\boldsymbol{D}$ はそれぞれ行列であり，なおかつ $\boldsymbol{A}$，$\boldsymbol{D}$ は逆行列が存在するという条件がある

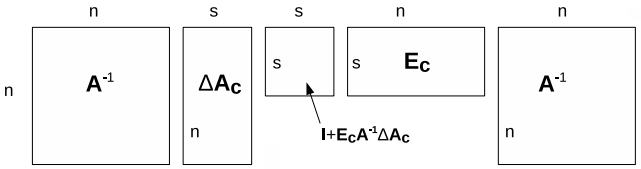


図 1: SMW 公式で用いられる行列の行列サイズ

$$(\boldsymbol{A}+\boldsymbol{BDC})^{-1} = \boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D}^{-1}+\boldsymbol{CA}^{-1}\boldsymbol{B})^{-1}\boldsymbol{CA}^{-1} \tag{1}$$

式 (1) から $\boldsymbol{A^{-1}}$ を用いて $\boldsymbol{A'^{-1}}$ を求めるため，更なる変形を行う．いま，$\boldsymbol{A}$ が時間経過によって $\boldsymbol{A'}$ へと微少変化したとして，$\boldsymbol{A'}$ を $\boldsymbol{A'} = \boldsymbol{A} + \boldsymbol{\Delta A}$ と表現する．さらに $\boldsymbol{\Delta A}$ を $\boldsymbol{\Delta A} = \boldsymbol{\Delta A_c} \times \boldsymbol{I} \times \boldsymbol{E_c}$ とし，$\boldsymbol{\Delta A_c}$ は $\boldsymbol{\Delta A}$ から全ての要素が 0 である列を取り除いた行列，$\boldsymbol{E_c}$ は列の除去に対応して 0 と 1 からなる行列とする．この時，式 (1) において $\boldsymbol{B} = \boldsymbol{\Delta A_c}$，$\boldsymbol{C} = \boldsymbol{E_c}$,$\boldsymbol{D} = \boldsymbol{I}$ とすると，

$$\begin{aligned}(\boldsymbol{A'})^{-1} &= (\boldsymbol{A}+\boldsymbol{\Delta A})^{-1} \\ &= \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{\Delta A_c}(\boldsymbol{I}+\boldsymbol{E_c}\boldsymbol{A}^{-1}\boldsymbol{\Delta A_c})^{-1}\boldsymbol{E_c}\boldsymbol{A}^{-1}\end{aligned} \tag{2}$$

となる．式 (2) は $(\boldsymbol{A'})^{-1} = \boldsymbol{A}^{-1} -$ (補正項) と見ることができ，変化後の行列 $(\boldsymbol{A'})$ の逆行列が $\boldsymbol{A}^{-1}$ の補正により導出できることを表している．式 (2) において，右辺第二項で行う行列積計算に使用する行列サイズの概要を図 1 に示す．なお，$\boldsymbol{\Delta A_c}$ の列の数を $\boldsymbol{s}$ とする．ここで，行列 $(\boldsymbol{I} + \boldsymbol{E_c}\boldsymbol{A}^{-1}\boldsymbol{\Delta A_c})^{-1}$ は図 1 の中央部分にある大きさ $\boldsymbol{s} \times \boldsymbol{s}$ の行列に対応し，この逆行列を求める計算量は $\boldsymbol{O(s^3)}$ となる．その他の行列積の計算での計算量は $\boldsymbol{O(sn^2)}$ とな

[1] 福井大学大学院　工学研究科

1. $S_1 = A^{-1} \times \Delta A_c$
2. $S_2 = E_c \times S_1$
3. $S_3 = I + S_2$
4. $S_4 = (S_3)^{-1}$ を直接法で計算
5. $S_5 = E_c \times A^{-1}$
6. $S_6 = S_4 \times S_5$
7. $S_7 = S_1 \times S_6$
8. $S_8 = A^{-1} - S_7$
9. $S_9 = S_8 \times b$

図 2: SMW 公式の計算ステップ

る．しかしながら，変化する要素数が微少 ($n >> s$) であると仮定すると，計算量は $O(sn^2)$ である．図 2 に SMW 公式を用いた計算ステップを示す.

2.2 ハイブリッド並列処理

ハイブリッド並列処理とはマルチコアプロセッサを用いたノード内並列処理と複数のノードを用いたノード間並列処理を組合せた並列化のことである.SMW 公式の計算ステップの大半が行列積計算であり並列化が容易である. 我々の過去の研究でハイブリッド並列処理と各計算ステップの順序を入れ替えて通信と計算のオーバーラップ実行を可能とした手法 [2] を用いることによる高速化が確認されている. 計算順序を入れ替えたアルゴリズムを Hybrid とし図 3 に示す.S'_5,S'_2 はそれぞれ部分和であり, データの再配置を回避するために部分和を求める処理と合成の処理が存在している. ここでのオーバーラップは通信に依存関係のない計算と通信のオーバーラップである. 並列度が高くなると 1 ノード当たりの計算量が減少する一方で, ノード間通信の回数が多くなり通信時間の増大に伴う速度低下が発生するという問題がある.

ステップ 1. $S'_5 = E_c \times A^{-1}$
ステップ 2. S'_5 の通信命令発行
ステップ 3. $S_1 = A^{-1} \times \Delta A_c$
ステップ 4. $S'_2 = E_c \times S_1$
ステップ 5. S'_5 の送受信完了
ステップ 6. S'_2 の通信命令発行
ステップ 7. S_5 の部分和合成
ステップ 8. S'_2 の送受信完了
ステップ 9. S_2 の部分和合成
ステップ 10. $S_3 = I + S_2$
ステップ 11. $S_4 = (S_3)^{-1}$ を直接法で計算
ステップ 12. $S_6 = S_4 \times S_5$
ステップ 13. $S_7 = S_1 \times S_6$
ステップ 14. $S_8 = A^{-1} - S_7$

図 3: ハイブリッド並列処理での計算ステップ

2.3 関連研究

しかし本研究で用いたデータサイズでは通信と計算のオーバーラップ実行が正常に動作しないことが判明した.

通信と計算のオーバーラップの実装においては,MPI の非同期通信関数を用いた実装と通信専用のスレッドを別に起動して計算と通信のオーバーラップを行う方法が提案 [4], [5] されている. メッセージサイズが小さい場合,MPI の非同期通信関数を用いると容易に通信と計算のオーバーラップ実装が可能であるが, メッセージサイズが一定以上になると MPI の非同期通信のプロトコルが Eager Protocol から Rendezvous Protocol に切り替わり, プロトコル処理のための CPU 利用と計算のための CPU 利用が競合し計算と通信のオーバーラップ実行が正常に機能しないことが知られている.

そこで, 我々も文献 [4], [5] 等と同様に通信専用のヘルパースレッドを用いた実装を行った. この際,OpenMP を用いた計算処理で使用するスレッドの 1 つをヘルパースレッドに割り当てるのではなく, 通信専用の POSIX スレッドを別途起動して通信に専念させる実装を行った. また, 実際のノード間通信は MPI 同期通信を用いて実装し, ノード内の通信用スレッドと計算用スレッドの同期は簡単なフラグを用いて実装を行った. これにより, 計算処理と通信処理のプログラムの独立性があがりプログラム最適化の自由度が向上した.

文献 [4], [5] では, 通信用スレッドの割り当てによる物理コアの浪費を回避するために, 通信用スレッドにも計算を割り当てる実装が提案されている. 文献 [5] では dynamic スケジューリングを併用した実装が, 文献 [4] ではノード間通信に必要なデータを生成・消費する計算空間上の袖領域の計算を通信用スレッドに割り当てる実装が提案されている. 一方, 我々の実装では仮想スレッドが利用できる環境あるいはノード内の物理コアに余裕がある状況 [*1] を想定し, 通信用スレッドへの計算の割り当てはおこなっていない.

3. 通信のタイリングによる通信時間隠蔽

前節では本質的な依存関係のない通信ステップと計算ステップのオーバーラップ実装について, 逆行列計算の実行ステップ単位で考えてきた. しかしながら, 通信時間を十分に隠蔽するだけの計算が存在しなければ通信時間の隠蔽はできない. 我々のケースでは図 3 における,S'_5 の通信時間隠蔽が重要であるが，これと依存関係にない計算は S_1 および S'_2 であり十分な隠蔽効果が得られない.

そこで, 通信ステップ $S_{通}$ と依存関係にある後続の計算ステップ $S_{計}$ の一部とのオーバーラップ実行を行うことによる通信時間隠蔽の可能性を考えてみる. いま通信ステップで送信するデータを d 分割して送信すること（これを通信のタイリングと呼ぶ）を考える. i 番目のデータ受信により後続計算ステップの一部 $S_{計\ i}$ が実行可能になるのであ

[*1] 利用可能なメモリバンド幅等との関係でノード内並列度を抑制したほうがよい場合など

れば,$i+1$ 番目のデータ送受信処理と $S_{計\ i}$ のオーバーラップ実行が可能となる. これにより理想的には, 図 4 のように通信時間の隠蔽が可能となる.

実際には通信時間と計算時間は等しいとは限らないので，それぞれ $T_{通}$ および $T_{計}$ とし, 通信の分割数を d としてタイリング後の実行時間を考えると,$T_{通} \leq T_{計}$ の場合 $T_{計} + \frac{1}{d}T_{通}$ のような $T_{通} \geq T_{計}$ の場合 $T_{通} + \frac{1}{d}T_{計}$ と近似できる. しかしながら, 実際には分割数 d を増やすと通信や計算の起動・停止に伴うオーバヘッドならび同期処理が増加するため, 適切な分割数 d の選択が重要となる.

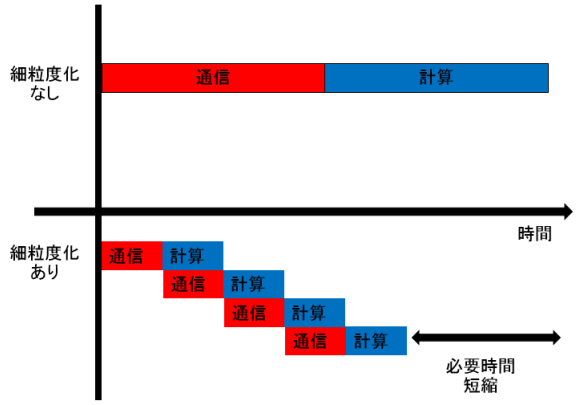


図 4: タイリングを用いた通信と計算の
オーバーラップによる通信時間隠蔽

各ステップでの計算において, 計算時間の多くを占めているのはステップ 13 および 14 である. ここで, 通信中にこのステップでの計算を行えるようなアルゴリズムを考える. ステップ 12 の計算は S_5 を必要とするが, 図 5 のように S_5 の列方向ブロック分割された部分行列で計算が可能である.

その後のステップ 13 もステップ 12 の部分行列で計算が可能であり，ステップ 14 も同様である．この点に着目し，S'_5 の通信を細粒度化し，後に続く計算を通信とオーバーラップすることで高速化を行う.

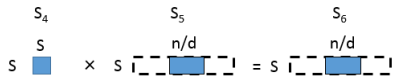


図 5: ステップ 12 の部分行列を用いたモデル

4. 評価

タイリングの効果は MPI でノード間通信を行うノード数, 各ノードの計算速度, ノード間通信の通信速度によって異なるため, これらのパラメータの変化がタイリングの効果に与える影響を評価する. 評価実験には 2 つの実験環境を利用した. 実験環境 1 (表 1 参照) は研究室で所有するクラスタシステムの一部 (8 ノード) であり, 実験環境 2 は福井大学が機関契約 (準優先) で利用している京都大学のスーパーコンピュータの一部 (4 ノード) である.

表 1: 実験環境 1 のノード仕様

OS	Linux 2.6.32-358.11.1.el6.x86_64
CPU	Intel Core i7-3770 3.40GHz
L3Cache	8MB
compiler	mpiicc,icc Version 12.0.0
option	-O3
MPI	Intel MPI
NIC	Intel X540T2 (10Gbps)
SW	NETGEAR XS708E (10Gbps)
	Buffalo LSW-GT-16NSR(1Gpbs)

実験環境 1 では,8 台のノードを接続するイーサネットのスイッチを物理的に交換することで通信速度を 10Gbps,1Gbps および 100Mbps と変化させ通信時間と計算時間の比率を変更する実験を可能にした.

行列サイズ n が 3722 ならびに 7397 の 2 種類の行列に対し, 係数行列の変化 s を 5 から約 500 まで変化させた場合の逆行列導出時間を測定した.

表 2: 実験環境 2 のノード仕様

プロセッサ数 (物理コア数)	2(2x18=36)
CPU	Intel Xeon Broadwell 2.1GHz
L3Cache	45MB
メモリ	128GB
コンパイラ	icc Version 17.0.2
MPI	Intel MPI

4.1 Trace_Analyzer を用いた動作分析

実験環境 1 の 4 台のノードを $10Gbps$ のスイッチで接続し, $N = 7397$, $s = 492$ の問題に対して逆行列計算を行い, インテルの Trace_Analyzer を用いてトレースデータを取得した結果を図 6 および図 7 に示す. タイリングを行わない場合が図 6, タイリング数 4 の場合が図 7 である. 横軸は時間であり, 各ノードでの計算スレッド群とヘルパースレッドの動作が上下ペアで表示されている. 待ち時間を含むが計算部分が青色で, 通信時間が赤色で表示されている. 各計算ステップに対応する場所を図中に示しているが,S_{34678} はステップ $10 \sim 14$ まで,S_{678} はステップ $12 \sim 14$ までをまとめたものである.

実際に通信や計算を行っている部分と待ち時間の区別ができていないが, タイリングに伴う通信パターンの変化やオーバーラップ実行による計算時間の短縮が確認できる.

4.2 並列処理の効果

予備実験として実験環境 1 において, ノード数を 2,4,8 と変化させたときの計算速度を測定した. ノード内並列処理

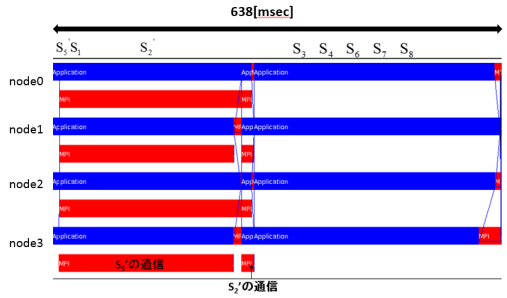


図 6: trace_analyzer を用いた動作分析図 (タイリングなし)

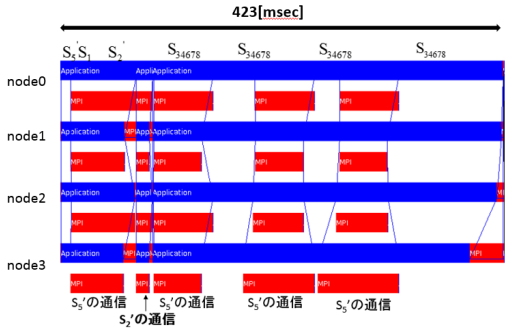


図 7: trace_analyzer を用いた動作分析図 (タイリング数 4)

のための OpenMP のスレッド数は 2, タイリング数 d も 2 として計測を行った. 図 8 に行列サイズ $n = 3722$ の場合について, 通信速度を変化させた場合の実行結果を示す. $n = 7393$ の場合の結果は付録に示す.

通信速度が 1Gbps ではノード数 2 の場合が最速で, ノード数を増やすと速度低下が見られるのに対して, 通信速度 10Gbps では s の値が小さい場合を除きノード数 4 の場合が概ね最速となった. ただしノード数 4 と 8 の差はノード数 2 と 4 の差ほど大きくないことがわかる.$n = 7393$ の場合においてもほぼ同様の結果がえられた.

参考までに通信速度を 100Mbps とした場合の実験も行ったが, 実行時間の大半が通信時間となり並列化の効果は得られなかった.

4.3 タイリングの効果

実験環境 1 において, タイリング数 d を 1(なし),2,4,8,16 と変化させたときの逆行列導出時間への影響を測定した.

行列サイズ n=3722, ノード数を 4 として, 通信速度を 10Gbps ならびに 1Gbps とした場合の実験結果を図 9 ならびに図 10 に示す. ノード数 2 および 8 の場合, ならびに $n = 7397$ の場合については付録に示す.

いずれの場合も, タイリングを行うことにより高速化の効果を確認できた. また, ノード数 4 の場合は, 両方の通信速度とも, 大きな s に対してはタイリング数の増加とともに実行時間の短縮が確認でき, タイリング数 16 が概ね最速となった.

ノード数 2 の場合は,1Gbps の場合ノード数 4 の場合と同様の傾向がえられたが, 10Gbps の場合は, タイリングの有無を含めタイリング数を変えても実行時間の有意差はわずかで, 全体を通して概ねタイリング数 4 がベストな結果となっている.

一方ノード数 8 では, タイリングの効果は見れるものの, 多くの場合タイリング数 2 がベストであるという計測結果以外に, タイリング数と速度に関して規則性をよみとることは困難である. ノード数 8 では通信パターンも複雑化しタイリングに伴う通信起動のオーバヘッドとタイリングの効果が複雑に影響しあっているものと考えられる.

4.4 実験環境 2 での評価

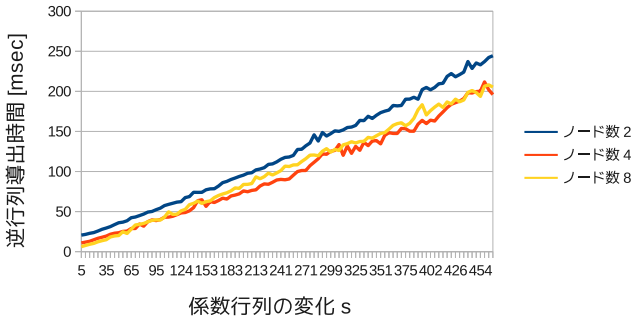
通信性能と計算性能の比が変わると, タイリングの効果も変わるため, 実験環境 1 とは異なる計算機環境での評価実験を行った. 実験環境 1 とことなり実験環境 2 は共用利用のため現在までに十分な評価データを得ることはできていないが,OpenMP のスレッド数を 32 とし行列サイズ n=7397 とした場合について, これまでに得られている結果を図 11 ならびに図 12 に示す. 図 11 はタイリング数 2 としてノード間並列処理の効果を測定したものである. いずれの場合もノード数 2 の場合が高速であったため, 図 12 ではノード数 2 の場合についてタイリングの効果を測定した結果である.

タイリングによる若干の速度向上は見られたが, 実験環境 1 の 10Gbps の場合とほぼ同様に, タイリング数と計算速度に規則性はみられず, s の値ごとに最適なタイリング数はランダムに変動する結果となった. 実験環境 1 とくらべても通信速度が非常に高速であり, 隠蔽すべき通信時間が小さくなったことでタイリングの効果が見えにくくなったものと考える.

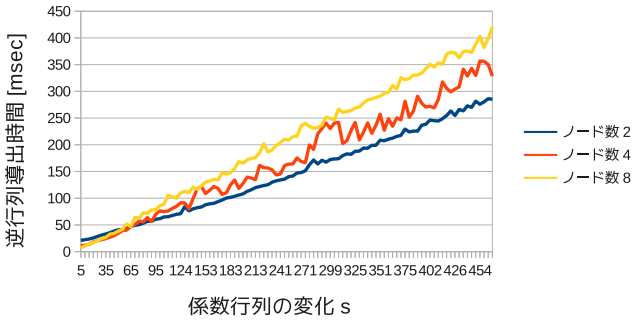
5. まとめと今後の課題

従来までの通信と計算のオーバーラップ実行の割合を増加させるためタイリングとよばれる通信の最粒度化を行った. 通信速度がそれぞれ 10Gbps,1Gbps の場合で並列処理とタイリングによる高速化の効果について評価を行った.n の値, 係数行列の変化 s, 通信速度によって並列化の効果, タイリングの効果が異なりそれぞれの環境に合わせた最適なパラメータが必要であることが分かった. 当初本研究は, 問題サイズ n, 係数行列の変化 s が与えられたときに実行時間最小とするような並列度, ならびにタイリング数を決定するための性能モデル式を導出することを目的としていたが, 明確な規則性を見つけることは困難であり, 静的な最適化よりは実行時にタイリング数を動的に最適化する方法のほうが有望と考えられる. 今後は, 性能モデル式の検討と平行して動的な最適化についても検討を行っていく予定である.

謝辞



(a) 10Gbps



(b) 1Gbps

図 8: 並列処理の効果 ($n = 3722, d = 2$)

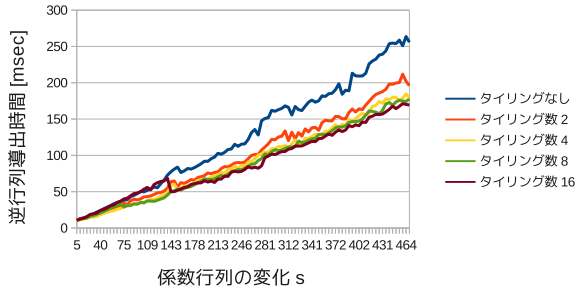


図 9: タイリングの効果 (ノード数 4,n=3722,10Gbps)

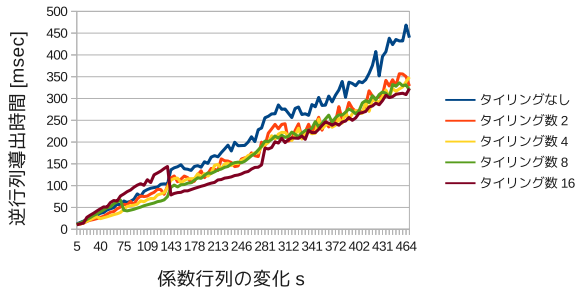


図 10: タイリングの効果 (ノード数 4,n=3722,1Gbps)

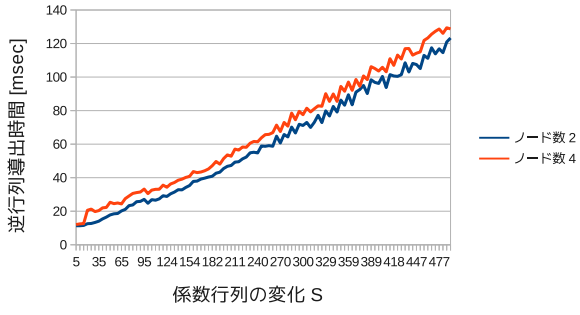


図 11: 並列処理効果 (n=3722,d=2)

本研究の一部は京都大学学術情報メディアセンターのスーパーコンピュータを利用して実施した.

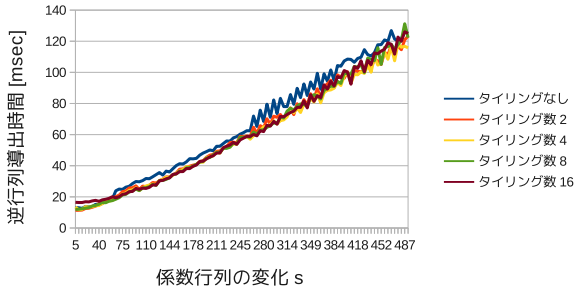


図 12: タイリングの効果 (ノード数 2,n=7397)

参考文献

[1] G.H. Golub and C.F.Van Loan, *Matrix Computations(3rd Ed.)*, Johns Hopkins Univ. Press(1996).

[2] 松井 祐太, 福間 慎治, 森 眞一郎, 実時間シミュレーションへの応用を前提とした SMW 公式を用いた逆行列計算のハイブリッド並列処理の評価, 研究報告ハイパフォーマンスコンピューティング（HPC）,2013-HPC-138(9),1-6 (2013-02-14).

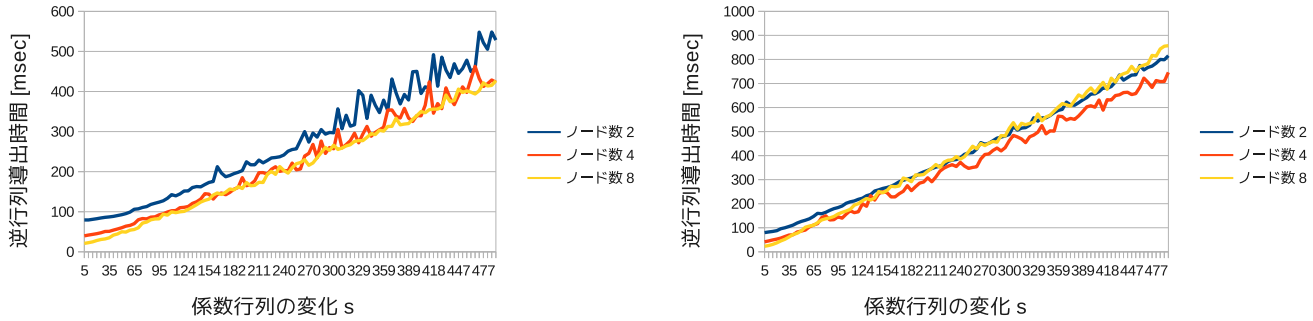
[3] 岩永翔太郎, 福間慎治, 森眞一郎：実時間シミュレーションへの応用を前提とした SMW 公式を用いた逆行列計算のハイブリッド並列による高速化, ハイパフォーマンスコンピューティングと計算科学シンポジウム HPCS2012 (2012).

[4] 深沢圭一郎, 森江善之, 曽我武史, 高見利也, 南里豪志,Halo スレッドと Halo 関数を用いた MHD シミュレーションの高効率並列化, ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集,2017,36-43 (2017-05-29).

[5] 井戸村泰宏, 中田資季, 山田進, 町田昌彦, 今村俊幸, 渡邉智彦, 沼波政倫, 井上昇, 堤重信, 三吉郁夫, 志田直之：プラズマ乱流シミュレーションにおける通信マスク手法開発, ハイパフォーマンスコンピューティングと計算科学シンポジウム HPCS2013 (2013).

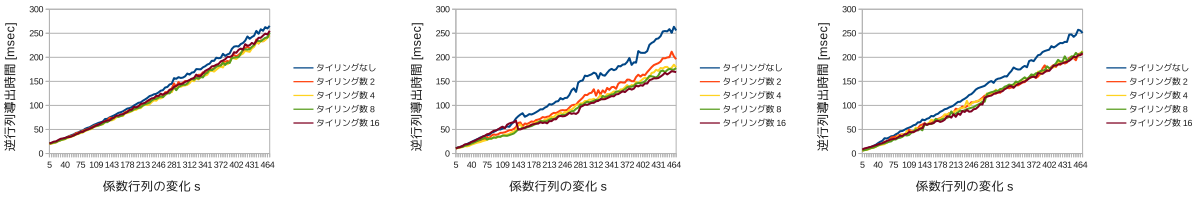
付　　録

A.1 測定結果



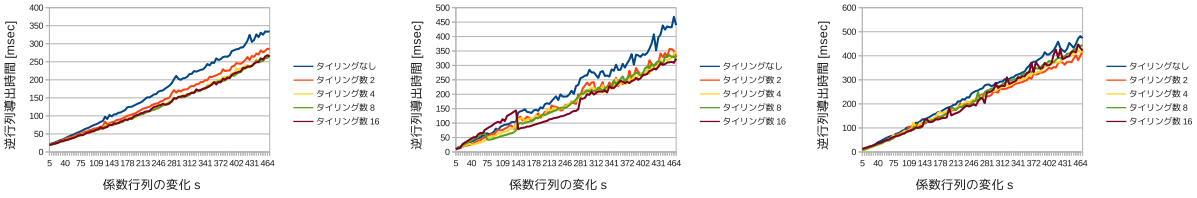
(a) 10Gbps (b) 1Gbps

図 A·1: 並列処理の効果 ($n = 7397, d = 2$)



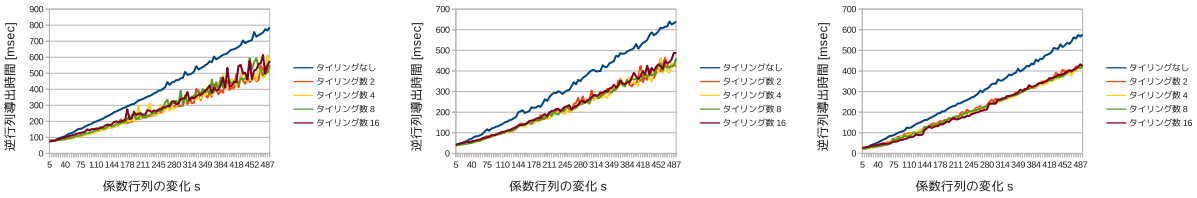
(a) ノード数 2 (b) ノード数 4 (c) ノード数 8

図 A·2: タイリングの効果 ($n = 3722, 10Gbps$)



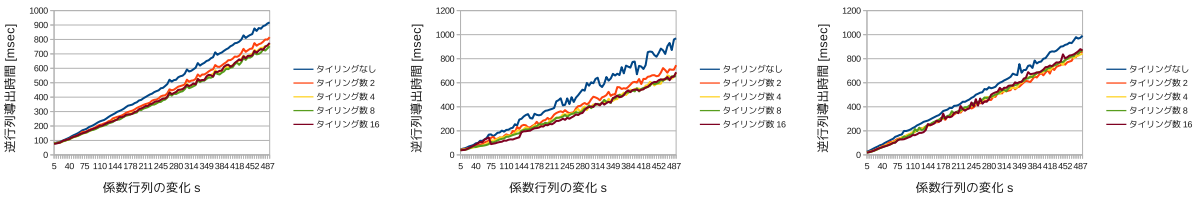
(a) ノード数 2 (b) ノード数 4 (c) ノード数 8

図 A·3: タイリングの効果 ($n = 3722, 1Gbps$)



(a) ノード数 2 (b) ノード数 4 (c) ノード数 8

図 A·4: タイリングの効果 ($n = 7397, 10Gbps$)



(a) ノード数 2 (b) ノード数 4 (c) ノード数 8

図 A·5: タイリングの効果 ($n = 7397, 1Gbps$)