2J-04

動的負荷分散を用いた階層型行列生成の並列化

棟形 克己* 平石 拓† 伊田 明弘‡ 岩下 武史§ 中島 浩†

京都大学大学院情報学研究科* 京都大学学術情報メディアセンター†
京都大学学術情報メディアセンター/JST CREST‡ 北海道大学情報基盤センター/JST CREST§

1 はじめに

境界要素解析や多体問題シミュレーションでは，物理要素間の相互関係を表す密行列を係数行列とする連立一次方程式を解くことが必要となることが多い．しかし，N 個の物理要素の全相互関係を表す密行列の要素数は N^2 となり，これを直接扱う計算手法では必要なメモリ容量や計算時間が N を増やすにつれて急激に増大してしまう．そこで，そのような密行列の近似圧縮表現である階層型行列（$\mathcal{H}$ 行列） [1] を用いる手法が提案されている．$\mathcal{H}$ 行列とは図 1 のように区分けされた行列の各ブロック（葉行列）を，積がその葉行列の近似となるような 2 つの低ランク行列で表現したものである（ただし図中では “full” と区分けされている一部の小さな葉行列は全要素の値により直接表現される）．ブロックサイズが $l \times m$ の葉行列は，サイズが $l \times k$ と $k \times m$ の 2 つのランク k の行列で表現される．ただし k は l や m より十分小さい．$\mathcal{H}$ 行列は N 個の物理要素データから密行列の直接表現を経由することなく生成可能で，そのデータサイズは $\mathcal{O}(NK \log N)$ に抑えられる．ただし K は k の許容される上限であり，要求される近似精度に応じて変化するが，通常 N より十分小さな値に設定される．

$\mathcal{H}$ 行列を利用するアプリケーションの全体の高速化のためには，$\mathcal{H}$ 行列ベクトル積のような$\mathcal{H}$ 行列に対する演算だけでなく$\mathcal{H}$ 行列の生成そのものの高速化も重要である．そこで本研究では，$\mathcal{H}$ 行列生成の動的負荷分散を用いた MPI/OpenMP ハイブリッド並列化を行った．

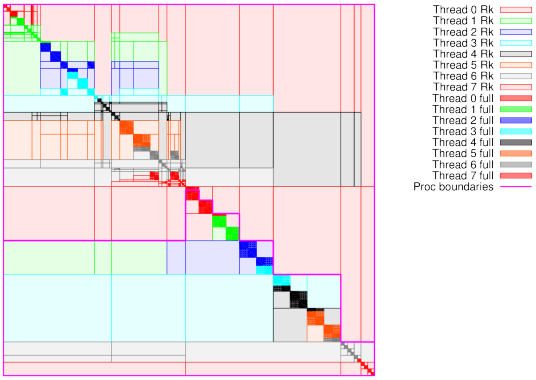

図 1 階層型行列

2 動的負荷分散の必要性とその実装

$\mathcal{H}$ 行列の生成は，行列の葉行列への区分け処理と，各葉行列に対応する低ランク行列の要素計算（フィル処理）から構成される．本研究では計算量が大部分を占めるフィル処理の並列化を考える．各葉行列のフィル処理は独立に実行できるため，各 OpenMP スレッドに処理すべき葉行列集合をタスク集合として割り当てることによってフィル処理全体のタスク並列化が可能であり，既存の$\mathcal{H}$ 行列計算ライブラリである$\mathcal{H}$ACApK [2] でも採用されている．

このようなタスク並列化では，各スレッドの負荷が均等になるようなタスク割当が求められる．各タスクに要する時間は，生成される 2 つの低ランク行列の要素数の合計にほぼ比例する．そこで$\mathcal{H}$ACApK では，経験則で定めた定数 k' を導入して，ブロックサイズ $l \times m$ の葉行列のフィル処理の計算量を $k'(l+m)$（全要素の値で直接表現される葉行列に関しては $l \times m$）と見積もり，計算量の見積値の合計が全スレッド間でできるだけ均等になるように静的にタスク割当を行う．図 1 における色分けは，この割当手法による 2 プロセス ×8 スレッド実

Parallelized Generation of Hierarchical Matrices with Dynamic Load Balancing

* Katsumi Munakata; Graduate School of Informatics, Kyoto University
† Tasuku Hiraishi, Hiroshi Nakashima; Academic Center for Computing and Media Studies, Kyoto University
‡ Akihiro Ida; Academic Center for Computing and Media Studies, Kyoto University/JST CREST
§ Takeshi Iwashita; Information Initiative Center, Hokkaido University/JST CREST

Copyright ©2015 Information Processing Society of Japan.
All Rights Reserved.

行時の各スレッドへのタスクの割り当て結果を示している．しかし，この手法では，以下の原因により良好な負荷均衡が達成されない．

(1) 実際に生成される低ランク行列のランク数 k は葉行列ごとに異なり，計算量の見積値 $k'(l+m)$ と実際の計算量 $k(l+m)$ の間に無視できないずれがある．k の事前予測も困難である．
(2) 計算量が全スレッドの平均計算量を上回る大きなタスクが存在する．

そこで本研究では，従来実装の計算量見積もりを活用しつつ，(1) タスクをスレッドに実行時に割り当てる動的負荷分散を行い，さらに (2) 負荷均衡の阻害要因となりうる大きな計算量見積値を持つタスクには MPI プロセス内の全スレッドを割り当てて並列に処理することで負荷均衡の改善を試みた．提案実装におけるフィル処理の具体的な計算手順は以下の通りである．なお，S は計算量見積値の全タスクの合計，n_t は全スレッド数，α は「大きなタスク」の閾値を決めるパラメータ，c は MPI プロセスが一度に取得するタスク集合のサイズを決めるパラメータである．本論文の数値実験では $\alpha = 0.1$ とし，c は取得したタスク集合を 1 スレッドで処理したときに 1 秒程度かかるような値に設定した．

1. 全タスクを計算量見積値に基づいて降順にソートし，その結果をグローバルタスクキューとする．
2. グローバルタスクキューの先頭部分のうち， 計算量見積値が $\alpha S/n_t$ 以上の全タスクを各 MPI プロセスにサイクリックに割り当てる．
3. 各 MPI プロセスで，2. で割り当てられたそれぞれのタスクを，プロセス内の全スレッドを使って並列に処理する．
4. 3. の後，計算量見積値の合計が c 以上となる最小の数のタスクをグローバルタスクキューからまとめて取得し，得られたタスク集合をローカルタスクキューとする．プロセス内の各スレッドは，ローカルタスクキューからタスクを 1 つずつ取得・処理するということをキューが空になるまで繰り返す．
5. 4. をグローバルタスクキューが空になるまで繰り返す．

なお，本研究の実装では，ランク 0 の MPI プロセスのうち 1 スレッドを各 MPI プロセスからのタスク要求の処理やグローバルタスクキューの管理のために割り当てている．また，4. のプロセス内のタスク並列処理は，OpenMP の並列ループにおけるダイナミックスケジューリング機能を用いて実装した．

表 1　評価環境

	ApproGreenBlade8000
Processor	Xeon Sandy Bridge 2.6GHz (8 cores × 2/node)
Memory	DDR3-1600 64GB/node
Network	InfiniBand FDR×2, Fat tree
Compiler	Intel Fortran Compiler 15.0 (-O3 最適化)
MPI	Intel MPI Library 5.0 (8 threads/process)

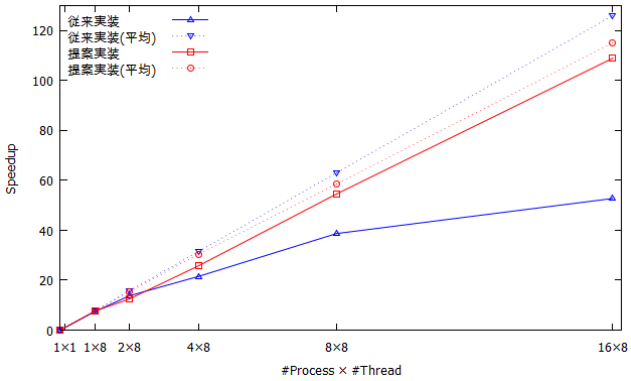

図 2　従来手法と提案手法の並列性能の比較

3　数値実験

提案実装の性能を評価するため，表面電荷法（1,188,000 要素）に現れる係数行列を$\mathcal{H}$ 行列として生成し，フィル処理にかかる時間を従来実装（静的タスク割当）と比較した．評価環境には京都大学のスーパーコンピュータ Laurel を用いた．表 1 に評価環境の詳細を示す．

測定結果を図 2 に示す．図中の実線は逐次実行に対する性能向上率を示す．破線は全スレッド間のタスク処理時間の平均を逐次実行時間で割った値，すなわち負荷が完全に均衡したと仮定した場合の性能向上率を示す．この図から，提案実装では動的負荷分散のコストがかかるものの，負荷均衡は大幅に改善されることがわかる．その結果，提案実装は 32 コア以上での実行で従来実装を大幅に上回る性能を達成し，特に 128 コア実行では従来実装に対して 2.07 倍の性能が得られた．

参考文献

[1] Börn, S. et al., Hierarchical Matrices, Lecture Note, Max-Planck-Institut für Mathematik, 2006.
[2] A, Ida. et al., Parallel Hierarchical Matrices with Adaptive Cross Approximation on Symmetric Multiprocessing Clusters, JIP, Vol. 15, No. 9, 2014.

Copyright ©2015 Information Processing Society of Japan.
All Rights Reserved.