4M-3

メニーコアプロセッサの性能向上を目指すタスク配置手法

佐野 伸太郎† 佐野 正浩† 佐藤 真平† 三好 健文†,‡ 吉瀬 謙二†
†東京工業大学 ‡独立行政法人 科学技術振興機構

1 はじめに

プロセッサに搭載されるコア数が増加したメニーコアアーキテクチャでは，プログラム中の並列性を活用することで，その演算性能を引き出すことが重要になる．マルチプロセッサなどと同様に，メニーコアアーキテクチャでは，並列化したタスクのコアへの配置方法が性能に影響を与えることがわかっている [1].

しかし，プログラマが最適な並列化タスクのコアへの配置（タスク配置) を記述することは困難である．そこで，コンパイラなどのツールによって最適なタスク配置を与えることが望まれる．

本稿では，メニーコアプロセッサの性能向上を目指すタスク配置手法として，パターンに基づいた配置手法を提案し，シミュレータによる評価を行う．

2 対象アーキテクチャ

本稿では，M-Core アーキテクチャ(M-Core)[2] を対象として評価を行う．M-Core は，ノードと呼ばれる 2 次元メッシュ状に配置された計算ユニットを持ち，各ノードは他のノードと明示的なデータ転送が可能である．M-Core でのルーティングには XY 次元順ルーティングを採用している．

3 タスク配置による性能変化

プログラムの実行時間が配置によって変化する例として，図 1(a) のように 3 次元インデックスに割付けられたタスクが，各タスクの近傍 6 つのタスクと相互に 512 バイトのデータ通信を行うプログラムを考える．3 次元立方体の大きさを $3 \times 3 \times 3$ とし，各々100 回データを送受信する．

このプログラムを 2 次元メッシュに配置した場合の性能を比較する．図 1(b) はタスクを z 次元によって分割し，単純に横に並べたものである．図 1(c) は XY 次元順ルーティングによる衝突を考慮し，人が配置したものである．この 2 つを比較したところ，(c) は (b) に対して 22%の性能向上を確認した．(c) の配置は (b) に比べホップ数が大きく通信レイテンシが増加するように見える．しかし，通信の衝突が抑えられることにより，スループットが向上し，性能が向上したと考えられる．このことから，衝突を考慮して配置することで性能向上が見込まれる．

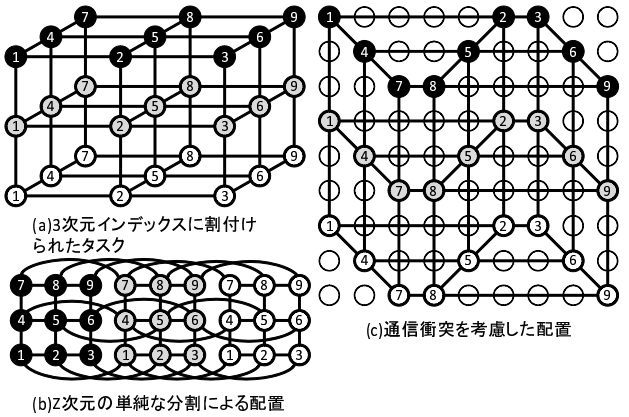

図 1: 3 次元インデックスに割り当てられたタスクの 2 次元メッシュへの配置.

4 タスク配置手法の提案

図 1(c) のような配置を求めるためには，ネットワークの通信パターンを求め，最適化アルゴリズムなどで通信衝突が最小となる配置を決定する必要がある．通信衝突とは別の問題となるが，ネットワーク通信量が最小である配置を決定する問題は NP 困難であることが知られている．通信衝突を最小化する問題は，通信量を最小化する問題と同程度以上に難しいと考えられるため，図 1(c) の配置をプログラムによって求めることは困難であることが予想される．また，例え最適な配置が求まっても，タスクの割り当てられていないノードの問題もある．本稿では，アプリケーションによらない特定のパターンを用いることで通信衝突の削減を目指す一方で，余剰ノードの活用についても検討する．

4.1 n ルーク問題

有名な最適化問題に n クイーン問題がある．これはチェスの盤上にクイーンを配置するとき，どの駒も他の駒にとられることがないように配置する問題である．このクイーンをルークに置き換えたものが n ルーク問題である．n ルーク問題によって得られた配置をタスク配置に置き換えると，X，Y の各次元に 1 つのタスクしか存在しない配置を得ることができる．

XY 次元順ルーティングであると，このような配置で，宛先の異なる通信が衝突することはない．なぜなら，X 次元に存在するタスクが 1 つであり，1 つのタスクから同時に複数の通信が発生することはないため，X 次元での衝突はない．同様に Y 次元に存在するタスクが 1 つであり，宛先が異なる通信であるため，Y 次元での衝突はないからである．

このように n ルーク問題から得られた配置を利用することで，通信衝突を削減することができる．しかし，n ルーク問題を解くためには，タスク数 n とするとノー

A method of task mapping to improve many-core processor performance
Shintaro SANO†, Masahiro SANO†, Shimpei SATO†, Takefumi MIYOSHI†,‡, and Kenji KISE†
†Tokyo Institute of Technology
‡Japan Science and Technology Agency

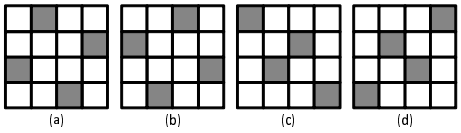

図 2: 4 ルーク問題から得られた配置.

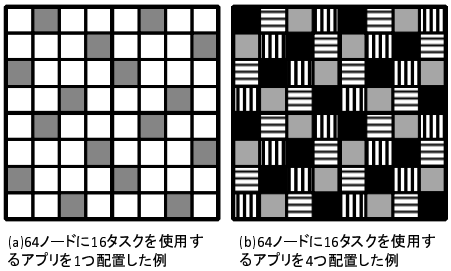

図 3: 64 ノードにパターンを適応した配置.

ド数 n^2 が必要となり現実的ではない．そこで，タスク数 4，ノード数 16 の 4 ルーク問題の解を繰り返し用いる．4 ルーク問題の解は 24 個あるが，そのうちの 4 つを図 2 に示した．これらが本稿で使用する配置パターンである．図 3(a) はノード数 64 の例である．基礎となるパターンを 4 枚貼り合わせた形となっている．

4.2 複数アプリケーションへの適応

提案した配置パターンを使用すると，タスク数 n に対してノード数 $4n$ が必要となる．タスクの割り当てられていないノードを利用をするために，複数のパターンを使用する．図 2 で示した 4 ルーク問題によって得られた配置は，同じノードを使用していない．そのため，重ね合わせて複数のプログラムを実行することが可能である．図 3(b) に 64 ノードに 4 つのプログラムが混在している様子を示す.

5 提案手法の評価

M-Core のソフトウェアシミュレータである SimMc[2] を用いて評価する.

5.1 ランダム通信による評価

まず，ネットワーク性能の指標として一般的によく用いられる，ランダム通信を行ったときのスループットを測定する．図 4 は，64 タスクのランダム通信でのスループットを示し，横軸が注入バイト数，縦軸がスループットである．図中の normal は，64 タスクを 8×8 ノードに配置している．pattern は図 2(a) のパターンを使用し，64 タスクを 16×16 ノードに配置している．この条件で，スループットが 47.4%向上することが確認できた.

5.2 NAS Parallel Benchmarks による評価

NAS Parallel Benchmarks による評価結果を示す．アプリケーションは mg，cg，ft，is を用いた．問題クラスは W である．図 5 中の 1 application は，1 つのベンチマークに対してパターンを適応した時の相対性能を示している．64node は図 3(a) の配置である．1024 ノードを使用した場合，性能が最大で 30.7%，平均で 15.9%向上した.

図 5 中の 4 application は，4 つのベンチマークに対してパターンを適応し，混在させて実行した時の相対性

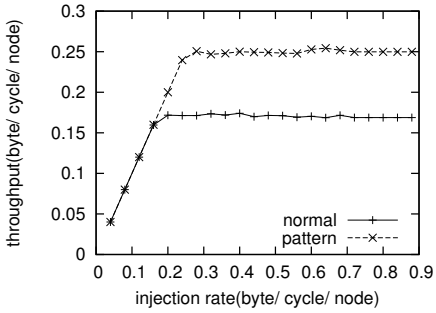

図 4: ランダム通信におけるスループット.

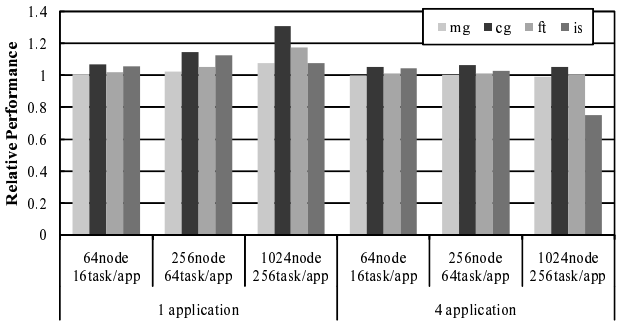

図 5: NAS Parallel Benchmarks による実験結果.

能を示している．64node は図 3(b) の配置である．64, 256 ノードでは性能低下は見られないが，1024 ノードでは is において性能低下が見られる.

この原因の一つとして，通信レイテンシの増加が挙げられる．is は実行時間に対する通信回数の多いベンチマークである．今回提案した手法では，タスク間を広げたことによる通信レイテンシの増加と，他のアプリケーションとの通信衝突によるレイテンシ増加があり，それらが性能低下の原因であることが予想される.

cg も通信回数の多いベンチマークであるが，1 回の通信量が多く，通信レイテンシの影響よりも，スループットの影響を大きく受けると考えられる.

6 おわりに

本稿では，n ルーク問題を応用したタスク配置手法を提案した．ランダム通信でスループットが 47.4%向上した．NAS Parallel Benchmarks を対象に評価を行ったところ，1024 ノードを使用し，256 タスクのベンチマークを 1 つ実行させたときに，性能が最大で 30.7%，平均で 15.9%向上した．ただし，複数のアプリケーションを実行した場合，性能低下を示すものも存在した．今後は本手法を用いて性能向上が見込めるものと，そうでないものの条件を見つけ出すことが必要であると考える.

参考文献

[1] 三好他. メニーコア向けタスクスケジューリングシステムの検討. 情報処理学会研究報告 2009-ARC-184, pp. 1–7, August 2009.

[2] Koh Uehara et al. A Study of an Infrastructure for Research and Development of Many-Core Processors. *UPDAS 09*, December 2009.