情報処理学会論文誌　コンピューティングシステム　Vol. 4　No. 4　96–109 (Oct. 2011)

メニーコアプロセッサのための通信衝突に着目したタスク配置手法

佐野　伸太郎[†1]　　吉　瀬　謙　二[†1]

Network-on-Chip で接続されたコアを持つメニーコアアーキテクチャでは，並列化されたタスクのコアへの割り当て方によって性能が大きく変化する．そこで，自動的に最適なタスク配置を求めることが望まれる．本論文では，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する．シミュレータを用いた評価から，提案手法は NAS Parallel Benchmarks において有用性を確認した．

A Task Mapping Method to Mitigate Network Contention for Many-core Processors

Shintaro Sano[†1] and Kenji Kise[†1]

The Network-on-Chip (NoC) is a promising interconnection for many-core processors. On the NoC-based manycore processors, the network performance of multi-thread programs depends on the method of task mapping. In this paper, we propose a pattern-based task mapping method in order to improve the performance of many-core processors. Evaluation of the proposed method using a detailed software simulator reveals effectiveness of the method.

1. はじめに

プロセッサのシングルスレッド性能の向上に限界が見え始めている．そのため，複数のコアを搭載して処理性能を向上させる方式が普及している．このようなマルチコアプロセッサでは，アプリケーションの並列性を活用することで，高い計算性能を得ることができる．

2 コアや 4 コアを有するマルチコアプロセッサではコア間をバスやクロスバースイッチによって接続する．コア間接続にバスを用いる場合，コア数が増えた場合に通信競合が多数発生することが予想される．また，クロスバースイッチによる接続方法はコア数が増えるほど回路に占める接続線が増えてしまう．

数十，数百といった多数のコアを持つマルチコアプロセッサ（メニーコアプロセッサ）では，ルータを用いたインターコネクションネットワーク（NoC）が有効である．NoC による接続はコア数に対して回路規模が一定の割合で済む．NoC を用いたプロセッサとして，Intel の Shingle chip Cloud Computer [1] や TILERA の TILE64 [2] が存在する．

NoC による接続は従来のバスやクロスバと違い，コア間の通信レイテンシが一定ではない．このため，どのコアにどのタスクを配置するのかという問題（タスク配置問題）が重要になる．このタスク配置問題は，様々な並列計算機の上で議論されてきた問題である [3),4)]．多くの場合，タスク配置問題はタスク間の通信量を最小化する問題として扱われる．さらに，その問題は NP 困難であると定義される [3),5)]．そのため，乱択アルゴリズム [5),6)] やヒューリステックアルゴリズム [7),8)] を使用して近似解を求めることが一般的である．しかし，乱択アルゴリズムの一種 Simulated Annealing による最適化には，100 コアの構成で 40 時間以上が必要という報告もある [7)]．このため，メニーコアプロセッサに適した，高速なタスク配置アルゴリズムの開発が必要である．

本論文ではパターンに基づいたタスク配置手法を提案する．この手法は通信衝突に着目して配置を行い，通信量の最適化は行わない．また，一定のパターンを採用するため，コア数が 1000 を超える場合でも，配置に時間がかかることはない．このように一般的なタスク配置アルゴリズムとは大きく異なっている．

本論文の構成を以下に示す．2 章で背景を述べる．3 章でタスク配置手法を提案する．4 章で提案手法をランダム通信と NAS Parallel Benchmarks によって評価する．5 章では 4 章と異なるアーキテクチャによって提案手法を評価する．6 章で関連研究について述べる．最後に，7 章でまとめる．

2. 背　　景

NoC を採用するメニーコアプロセッサをターゲットに様々なネットワークトポロジが検討されている．その中でも，2 次元メッシュネットワークは拡張性に優れ，実装も容易である．さらに，2 次元メッシュネットワークを採用したメニーコアプロセッサが多数存在す

†1 東京工業大学大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology

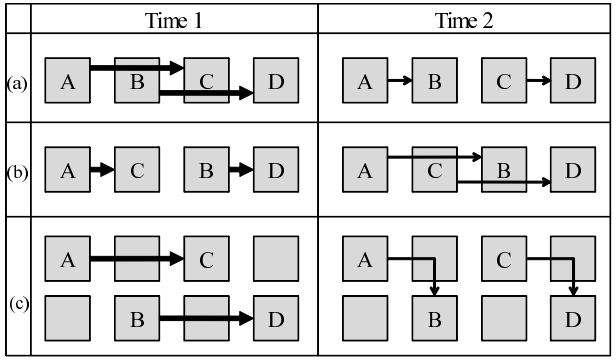

図 1　タスク配置によって変化する通信衝突の様子．図には 3 つの配置が存在している．3 つの配置それぞれに対して 2 つの時刻での通信を示す．矢印が通信方向を表し，矢印の太さが通信量を表している．(a) は Time 1 で衝突が発生する．(b) は Time 2 で衝突が発生する．(c) は衝突が発生しない

Fig. 1　An example of the effect of task mapping. Three mappings of (a), (b), and (c) are shown in figure. The arrows show direction of the communication, and the thickness of these arrows mean the network traffic.

る[2),9),10)]．これらの理由により，本論文では 2 次元メッシュネットワークを対象にタスク配置問題を考える．

タスク配置問題を考えるにあたって，通信データがどのような経路をたどるかといった問題も重要である．本論文では単純で広く用いられている XY 次元順ルーティングを考える．XY 次元順ルーティングでは，パケットは送信コアから X 軸方向を移動し，宛先コアの X 座標と同じ場所まで到達する．次に Y 軸方向を移動し，宛先コアに到達する．

上記に示したネットワーク構成で，タスク配置が性能におよぼす影響を考える．**図 1** に 3 つの配置 (a)，(b)，(c) を示す．この図では灰色の四角がコアを表している．さらに矢印が通信方向を表し，矢印の太さが通信量を表している．

配置 (a) の Time 1 ではコア A からコア C への通信が発生し，同時刻にコア B からコア D への通信が発生する．配置 (a) の Time 2 ではコア A からコア B への通信が発生し，同時刻にコア C からコア D への通信が発生している．ここで，Time 1 の太い矢印は Time 2 の細い矢印より多くの通信が発生していることを表している．配置 (a) の Time 1 では，コア B とコア C 間のネットワークで通信衝突が発生し，性能が低下する．特に Time 1 の通信量は多く，性能への影響も大きい．

この通信衝突を抑える方法は 2 つある．1 つ目の方法を図 1 (b) に示す．この手法はコア B とコア C の位置を交換する．この場合，配置 (a) と比べ全体としての性能を改善できる．Time 2 で通信衝突が発生するが，Time 2 は Time 1 に比べて通信量が少なく通信衝突を緩和できる．

2 つ目の方法を図 1 (c) に示す．ここでは 4 つのタスクを割り当てるために，8 個のコアの使用を仮定している．コア数が非常に大きいメニーコアプロセッサでは，通信ボトルネックなどの理由により並列性能がコア数にスケールしない状況が生じる．このような場合，1 つのアプリケーションのタスク数を制限することが好ましい．本論文では，このように 1 つのアプリケーションがすべてのコアを利用しないというシナリオを前提に議論する．図 1 (c) の手法はコア B とコア D の位置を下に 1 段ずらす．この配置の場合，通信衝突は発生しない．配置 (c) は (a) に対して，バイセクションバンド幅が 2 倍になる．そのため，通信スループットの向上が期待できる．しかし，通信ホップ数が増えたため，通信レイテンシが増加する．本論文では，配置 (c) のような余剰コアを使用して，通信衝突を減らす手法を考える．このような配置を求めるためには，どのようにこのような配置を作成するのか，余っているコアをどうするのかといった問題に答えなければならない．

3. 提 案 手 法

通信衝突の削減とバイセクションバンド幅の増加を達成するためのタスク配置手法を提案する．この手法では図 1 (c) のような配置を機械的に生成する．

まず，衝突数削減のために，衝突がない理想的な配置を考える．衝突のない配置は 2 つのタスクが同じ行と列に存在しない配置によって得ることができる．このような配置は n ルーク問題を解くことによって発見できる．n ルーク問題とは，$n \times n$ のチェスボード上に n 個のルークを配置するときに，どのルークも他のルークに取られることがないように配置する問題である．

n ルーク問題を用いた配置は通信衝突を完全に排除できるが，n ルーク問題を解くためにはタスク数 N_{task} とすると N_{task}^2 個のコアが必要となる．たとえば，64 コアに 8 タスクしか配置することができない．

次に，n ルーク問題の解を使用し，衝突数を抑えながらより多くのタスクを配置する方法を考える．これは，与えられたコア数よりも小さなコア数の n ルーク問題の解を考えることによって解決できる．ここでは，4×4 コアの 4 ルーク問題の解を考える．4 ルーク

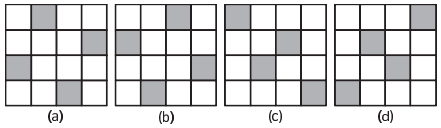

図 2　4 ルーク問題から得られた配置パターン
Fig. 2　Solution of 4-rook problem.

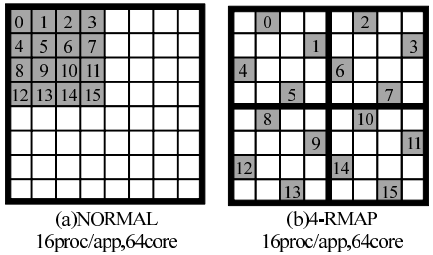

図 3　64 コアに 4-RMAP を適用した配置
Fig. 3　4-RMAP for 64 cores.

問題の解は 24 個ある．そのうちの 4 つを**図 2** に示す[⋆1]．図 2 中の灰色で示した四角がタスクを配置したコアである．先に示したように，この配置では通信衝突が起きない．4×4 コア以上の構成のプロセッサであるとき，この 4 ルーク問題の解をタイル状に敷き詰める．16 タスクを 64 コアに配置する場合，図 2 (a) を 4 つ並べたパターンにおける灰色の部分にタスクを配置する．タスク配置の結果は**図 3** (b) である．この配置は図 3 (a) を規則的に図 3 (b) へと変換する．図 3 に示す数字は対応するタスクを表す．ここで，図 3 (a) の配置を **NORMAL** と名付ける．さらに，図 3 (b) のように，n ルーク問題の解をタイル状に並べる手法を **n-RMAP**（n-Rook Mapping）と名付ける．4 ルーク問題の解を使用した場合，4-RMAP と記述する．2×2 コアの 2 ルーク問題の解をタイル状に並べ，その解にタスクを配置する手法は 2-RMAP である．2-RMAP を使用し，32 タスクを 64 コアに配置する場合，タスク配置の結果は**図 4** となる．

3.1　余剰コアを活用した n-RMAP の拡張手法の提案

提案した 4-RMAP を使用すると，たとえば 16 個のタスクに対して 64 個のコアが必要となり，48 個のコアが余る．この余剰コアを活用するために，複数の配置パターンを重ね合わせる手法を提案する．まず，余剰コアの存在しない配置を考える．**図 5** (a) は 16 プロセスのアプリケーション 4 つを NORMAL を用いて配置したものである．それぞれの模様が異なるアプリケーションを表している．このように NORMAL を用いて 4 つのアプリケーションを配置するものを NORMAL X4 と名付ける．

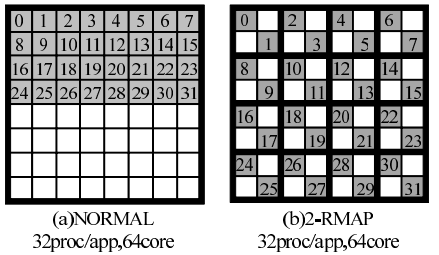

図 4　64 コアに 2-RMAP を適用した配置
Fig. 4　2-RMAP for 64 cores.

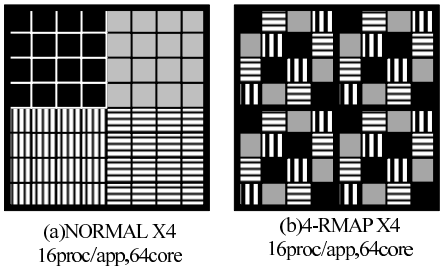

図 5　64 コアに 4-RMAP X4 を適用した配置
Fig. 5　4-RMAP X4 for 64 cores.

この NORMAL X4 に n-RMAP を適用する手法を考える．図 2 で示した 4 ルーク問題によって得られた 4 つの解 (a) から (d) はそれぞれ別々のコアを使用している．そのため，4 つの解を重ね合わせることができる．NORMAL X4 のアプリケーションごとに，4 つの解 (a) から (d) を適用し，重ね合わせると図 5 (b) となる．このように，n ルーク問題の解を重ね合わせる手法を n-RMAP Xm と名付ける．ここで m は重ね合わせるアプリケーションの数である．4-RMAP を使用して 4 つのアプリケーションを配置する手法であれば **4-RMAP X4** と表記する．同様に，2-RMAP を使用して 2 つのアプリケーションを配置する手法を **2-RMAP X2** とする．2-RMAP X2 を 64 コアに適用した配置を**図 6** に示す．

⋆1 4 つの配置は同じコアを使用しないものを選んだ．

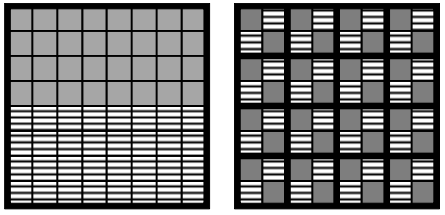

図 6　64 コアに 2-RMAP X2 を適用した配置
Fig. 6　2-RMAP X2 for 64 cores.

図 5 (a) の NORMAL X4 の場合，各アプリケーションの通信は，他のアプリケーションによる通信の混雑などによる影響を受けない．これは XY 次元順ルーティングでは，各アプリケーションの通信が 4×4 ブロックの外に出ることがないためである．しかし，提案手法による配置では複数のアプリケーションを重ね合わせたため，他のアプリケーションの通信の影響を受ける．この影響については実験によって明らかにする．

3.2　初期配置の検討

提案手法 n-RMAP を適用するタスク配置プロセスを**図 7** に示す．図 7 (a) タスクグラフは並列プログラムの通信の様子を表す．丸はタスクを表し，数字はタスクの番号である．この配置プロセスでは，タスクグラフから静的あるいは動的に図 7 (b) のような初期配置を求める．ここまではプログラム起動前の処理である．次の段階で実際のコアへ割り当てる．提案手法を用いる場合は，図 7 (c1) の流れとなる．この流れでは，n-RMAP は動的情報を用いることなく，(b) のタスクを物理コアに配置する．図 7 (c2) は密に割り当てた通常手法による配置である．これは (b) のタスクをそのままの形で物理コアに割り当てる．

3.2.1，3.2.2 項ではこれまで議論していなかった初期配置について考える．初期配置の作り方として 2 つの手法 sequential と EPAM-XY [7] を用いる．

3.2.1　sequential

本論文では MPI アプリケーションを使用して評価を行う．MPI アプリケーションは複数のプロセスによる並列プログラムであり，それぞれのプロセスには MPI ランクと呼ばれる識別子が付いている．p 個のプロセスが存在する場合，それらのプロセスは $0, 1, ..., p-1$ のランクを持つ．

初期配置を求める手法として，MPI ランクを順番に配置していく方法を用いる．この配置法によって，16 プロセスの MPI アプリケーションを配置する場合，各 MPI ランクは図 3 (a) のようになる．この手法は通信量などの動的な情報を使用しない．

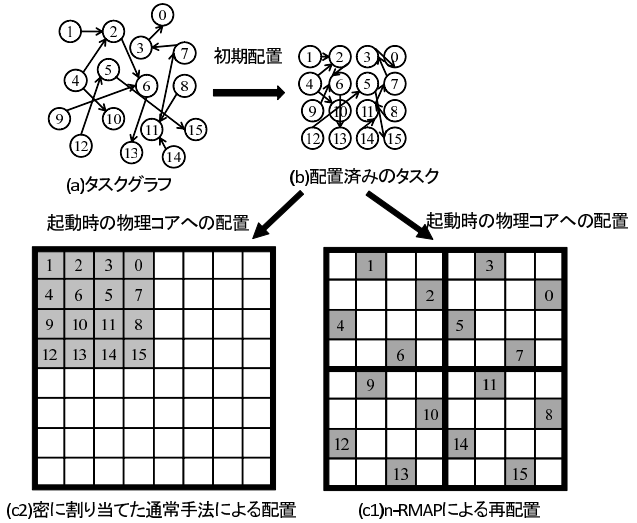

図 7　タスク配置プロセス
Fig. 7　Task mapping process.

3.2.2　EPAM-XY

2 つ目の手法として，文献 7) による手法 EPAM-XY を用いる．EPAM-XY はタスク間の通信量をプロファイルし，以下で定義される通信コストを最小化するように，最適化を行う．

$$CommCost = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} Hop(Pos(i), Pos(j)) \times Comm(i, j) \quad (1)$$

ここで i，j はタスクを表し，n は結合するタスク群の総タスク数，Hop はホップ数（単位 hop），Pos はタスクを配置した位置，$Comm$ はタスク間の通信量（単位 byte）である．

通信コストを最適化するために，EPAM-XY は幅優先探索を行う．幅優先探索ですべての配置を探索することは困難であるため，タスク配置に適した枝刈りを行う．この最適化では，求めたいコア数が十分に小さい場合，必ず最適解を求めることができる．ただし，コア

数が増えると探索に時間がかかりすぎる．そのため，幅優先探索に用いるキューがある大きさ以上になった場合，キューへの追加を制限する．この制限のため最適解が求められない場合がある．

4. 評　　価

4.1 評価環境 M-Core

提案手法の評価には M-Core アーキテクチャ（M-Core）[11] を使用する．計測には M-Core のソフトウェアシミュレータである SimMc [11] を利用する．M-Core のモデルを**図 8** (a) に示す．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つ．図 8 (a) は 8 × 8 のノードを持つ例である．各ノードはコア，ノードメモリ（各ノードが持つ小規模メモリ），INCC（InterNode Communication Controller），ルータで構成される（図 8 (b)）．コアは 2 命令発効のインオーダプロセッサである．命令セットは MIPS を使用する．コア間でデータが必要な場合には，DMA 転送を用いて明示的にデータ通信を行う．

DMA 転送はパケット転送を用いて実現されている．ワームホールスイッチングを採用し，パケットはフリットと呼ばれる転送単位に分割される．パケットは最大 40 バイトのデータを転送する．それより大きい転送は，複数のパケットに分割される．また，ルーティング方式には XY 次元順ルーティングを採用する．

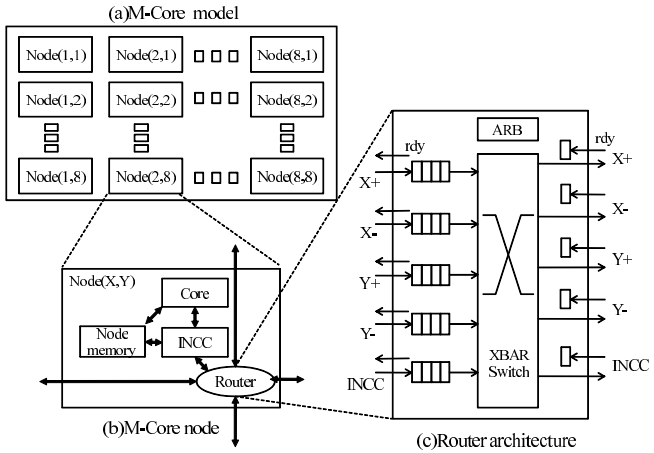

図 8　M-Core アーキテクチャ
Fig. 8　The M-Core architecture.

ルータアーキテクチャを図 8 (c) に示す．このルータはシングルパイプラインを採用している．ルータは入力ポートからのデータを格納するために，4 フリットのバッファを持つ．仮想チャネルは持っていない．パケットの紛失を防ぐために，フロー制御に Xon/Xoff 方式を採用している．調停回路（ARB）は送信可能な出力ポートに対して，ラウンドロビン方式で調停を行う．調停によって出力を許可されたパケットは，XBAR Switch を通過して出力ポートに送られる．また，ルータ間のビット幅は 32 bit である．

M-Core のノードメモリは 512 KB と規定されている．動作させたいベンチマークプログラムは，静的に確保するデータ領域と命令領域を合わせると 512 KB を超えるものが存在する．このため，本章の評価ではノードメモリの容量を増やして対処する．ノードメモリを制限し，オフチップメモリとの通信を行うものについては，5 章で評価を行う．

NAS Parallel Benchmarks を実行するために，M-Core アーキテクチャ上で動作する MPI ライブラリ，MPIMC を使用する．このライブラリは DMA 転送で実装可能な文献 12)，13) を参考にして実装されている．

4.2 ランダム通信による評価

提案手法を適用した配置のネットワーク性能を明らかにするために，ランダム通信の通信スループットと通信レイテンシを測定する．通信スループットは，データ転送量を通信プロセス数と実行サイクル数で割ったものである．通信レイテンシは，先頭パケットがネットワークに注入されてから，テイルパケットが受信コアに到着するまでの時間である．実験に用いたプログラムは，各コアが送信コアをランダムに決定し，データを送信する．

図 9 はランダム通信のスループットを示している．縦軸は 1 サイクル 1 コアあたりのスループット，横軸は 1 サイクル 1 コアあたりの注入データ量を示す．配置方法は図 3 (a) NORMAL，図 3 (b) RMAP，図 5 (a) NORMAL X4，図 5 (b) RMAP X4 を使用する．

NORMAL X4 の評価では，4 × 4 コアの中で通信を行う 4 つのランダム通信ベンチマークを用いて評価を行っている．この NORMAL X4 の配置では，4 つのランダム通信が 4 × 4 コアの外に出ることはないため，通信プロセス数で割ったスループットは NORMAL の配置と等しくなる．

各配置のスループットを注入データ量 0.64 flit で比較すると，NORMAL で 0.46 flit，4-RMAP で 0.64 flit，4-RMAP X4 で 0.22 flit である．最大スループットは 4-RMAP で向上する．4-RMAP はバイセクションバンド幅が増加し，その効果が出たものと思われる．しかし，4-RMAP X4 はスループットが減少する．これは他のアプリケーションの通信によ

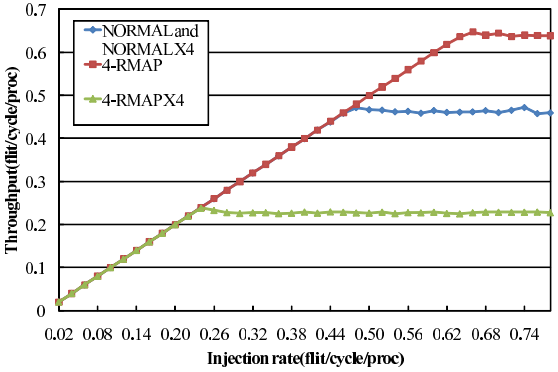

図 9　ランダム通信のスループット（16 proc/app，64 core）
Fig. 9　Throughput of random traffic (16 proc/app, 64 core).

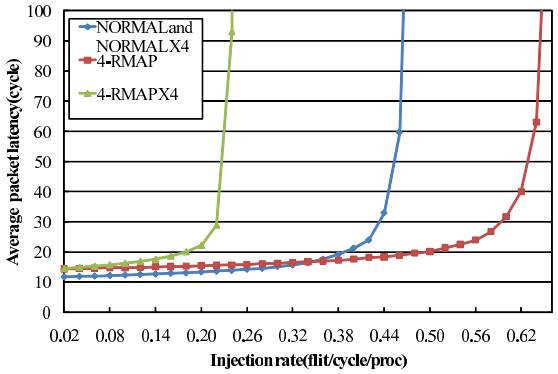

図 10　ランダム通信のレイテンシ（16 proc/app，64 core）
Fig. 10　Latency of random traffic (16 proc/app, 64 core).

るネットワークの混雑が大きく影響したからである．

図 10 はランダム通信のレイテンシを示している．縦軸はパケットの平均通信レイテンシ，横軸は 1 サイクル 1 コアあたりの注入データ量を示している．各配置のレイテンシを注入データ量 0.1 flit で比較すると，NORMAL で 12 cycle，4-RMAP で 15 cycle，4-RMAP X4 で 16 cycle である．注入データ量 0.4 flit で比較すると，NORMAL で 21 cycle，4-RMAP で 17 cycle，4-RMAP X4 で 18261 cycle である．

注入データ量が 0.36 flit 未満では NORMAL のレイテンシが最も短い．注入データ量が 0.36 flit 以降から 4-RMAP のレイテンシが最も短くなる．これは，注入データ量が少ない範囲では，NORMAL と 4-RMAP のホップ数の差がそのまま出たと考えられる．そして，注入データ量を増やすにつれ，NORMAL，4-RMAP ともにネットワークの混雑による通信レイテンシが増加する．提案手法による通信衝突削減によって，RMAP は通信レイテンシが小さくなったと考えられる．

4-RMAP X4 は NORMAL に比べてつねに通信レイテンシが増加している．これは通信混雑の影響とホップ数が増えたことの両方が考えられる．

4-RMAP X4 は NORMAL X4 と比べスループット，レイテンシが悪い．しかし，ここで用いたランダム通信は一定の間隔で通信を行うプログラムであり，これは実アプリケーションの通信挙動とは異なる．4.3 節では実際のアプリケーションを用いて評価を行う．

4.3　NAS Parallel Benchmarks による評価結果

実アプリケーションにおける提案手法の効果を測定するために，NAS Parallel Benchmarks 3.3 [14]（NPB）による評価を行う．ベンチマークはカーネルから mg，cg，ft，is を用いた[⋆1]．問題サイズはクラス W を使用した．クラス W は 32 MB 以下の少ないメモリシステムのためのベンチマークである．

まず，使用するベンチマークの台数効果を**図 11** 示す．配置方法は NORMAL，初期配置は sequential を採用した．n-RMAP と n-RMAP Xn の評価に使用する 16 から 256 プロセスでは，is 以外のベンチマークにおいて性能向上を示している．is は 256 コアにおいて性能が低下しているが，極端な性能低下ではなく，16 から 256 プロセスを使用し評価することは妥当であると考える．

4.3.1　初期配置の評価

NAS Parallel Benchmarks の cg，ft，is，mg ベンチマークに 2 つの手法，sequential と EPAM-XY を適用したときの式 (1) で与えられる通信コストを求める．この通信コストはホップ数（hop）と通信量（byte）の積を足し合わせたものであり，単位は hop×byte となる．この通信コストは低いほど良い．**表 1**，**表 2** に評価結果を示す．この実験では，EPAM-XY のキューサイズは探索時間が 10 分程度になるように調節している．

⋆1 カーネルベンチマークには ep も存在する．ep は通信時間の割合が他のベンチマークと比較して極端に少ない．配置変更による性能の変化はないと考えたため，除外した．

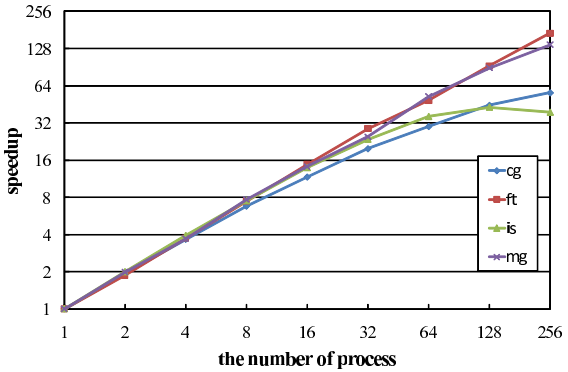

図 11　M-Core アーキテクチャで NPB を実行したときの台数効果
Fig. 11　The speedup of NPB.

表 1　最適化による通信コスト（4 × 4 コア）
Table 1　Communication Cost (4 × 4 core).

ベンチ名	sequentail（hop×byte）	EPAM-XY（hop×byte）
cg	4.82×10^8	3.29×10^8
ft	1.47×10^8	1.47×10^8
is	1.09×10^8	1.10×10^8
mg	5.70×10^7	5.66×10^7

表 2　最適化による通信コスト（8 × 8 コア）
Table 2　Communication Cost (8 × 8 core).

ベンチ名	sequentail（hop×byte）	EPAM-XY（hop×byte）
cg	2.15×10^9	2.15×10^9
ft	3.32×10^8	3.54×10^8
is	2.64×10^8	2.90×10^8
mg	1.80×10^8	2.06×10^8

表 1 から 16 コアの cg ベンチマークで，sequential よりも EPAM-XY の通信コストが低くなっていることが分かる．実際のシミュレータで評価したところ，EPAM-XY は sequential に比べ 2.2%性能向上を示した．しかし，それ以外のベンチマークに対しては sequential とほぼ同じか悪くなっている．特に 8 × 8 コアを使用した実験では，sequential よりも良いものは存在しない．この原因としては，64 コアが EPAM-XY の探索空間として広すぎることがあげられる．

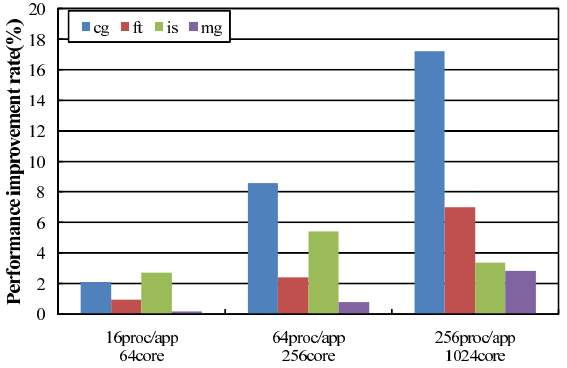

図 12　NORMAL と比較したときの 4-RMAP の性能向上率
Fig. 12　Performancd improvement rate of 4-RMAP compared with NORMAL.

表 3　NPB の通信スループット（NORMAL 64 プロセス）
Table 3　Throughput of NPB (NORMAL 64 process).

ベンチ名	実行サイクル（cycle）	DMA スループット（flit/cycle/proc）
cg	6.36×10^7	5.44×10^{-2}
ft	2.12×10^7	2.71×10^{-2}
is	0.69×10^7	4.27×10^{-2}
mg	2.62×10^7	1.11×10^{-2}

この評価から，動的情報を用いない sequential が十分に最適化されていると考えられる．そのため，以降の評価には sequential を初期配置として用いる．n-RMAP も動的情報を用いないため，アプリケーションの配置は静的に求めることができる．

4.3.2　n-RMAP の評価

4-RMAP を適用した配置の性能を評価する．**図 12** は 4-RMAP を適用した配置を NORMAL と比較したときの性能向上率である．グラフの横軸はコア数を示している．図 12 の一番左 16 proc/app，64 core は，図 3 (a) と (b) を比較した結果である．このグラフからコア数にかかわらず，すべてのベンチマークで性能向上を示していることが分かる．この結果から 4-RMAP は実アプリケーションにおいて効果的であるといえる．特に cg ベンチマークは性能向上率が高い．これは cg ベンチマークの通信スループットが高いためであると考えられる．**表 3** に各ベンチマークのプロセス数 64 における通信スループットを示す．cg

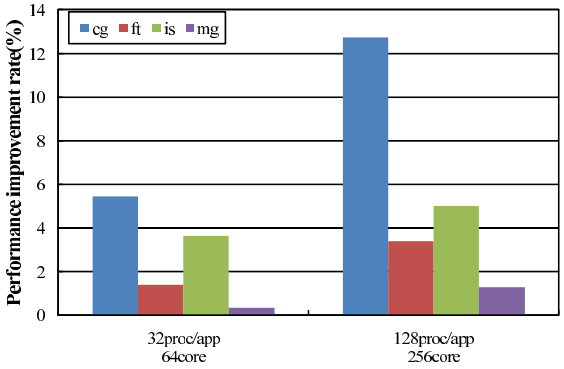

図 13　NORMAL と比較したときの 2-RMAP の性能向上率
Fig. 13　Performancd improvement rate of 2-RMAP compared with NORMAL.

ベンチマークと is ベンチマークにおいて通信スループットが高く，図 12 の 64 proc/app, 256 core と比べても，通信スループットの順と性能向上率の順が一致していることが分かる．

また，コア数が増えるにつれて性能が向上している．これは，アプリケーションの並列度が上がったことにより，プログラム中の実行時間のうち通信に関係する時間が増えたことが考えられる．

図 13 に 2-RMAP を適用した配置の性能向上率を示す．縦軸は NORMAL と比較した性能向上率であり，横軸はコア数をである．図 12 の一番左 32 proc/app，64 core は，図 4 (a) と (b) を比較した結果である．すべてのベンチマークで性能向上を示している．

4.3.3　n-RMAP Xm の評価

n-RMAP Xm の評価を行う．4-RMAP X4 を評価する場合，図 5 の各模様に 1 つのアプリケーションを割り当て，複数のアプリケーションを同時に実行する．各ベンチマークはプログラムの実行時間が異なり，プログラムを 1 度だけ実行すると，プログラムが存在しない期間ができてしまう．そのため，各ベンチマークプログラムを繰り返し実行するように変更した．プログラムの初期化なども複数回行われる．すべてのベンチマークを 3 回以上実行し，各ベンチマークが出力する結果によって評価する．

図 14 に 4-RMAP X4 を適用した性能向上率を示す．実験には 64 コア，256 コア，1024 コアを使用した．図 14 の一番左の cg，ft，is，mg 16 proc/app，64 core は図 5 (a) と (b) を比較したものである．複数回実行したため，性能向上率の最大，最小，平均による性能向上率を示す．

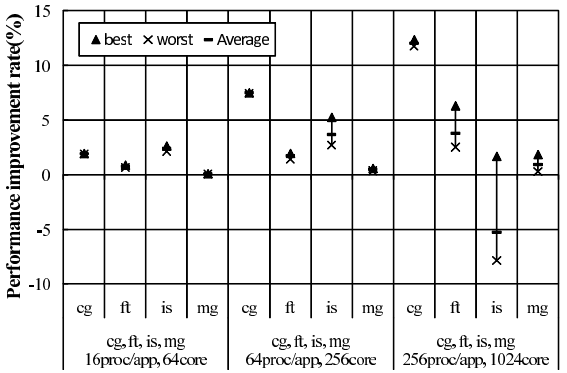

図 14　NORMAL X4 と比較したときの 4-RMAP X4 の性能向上率
Fig. 14　Performancd improvement rate of 4-RMAP X4 compared with NORMAL X4.

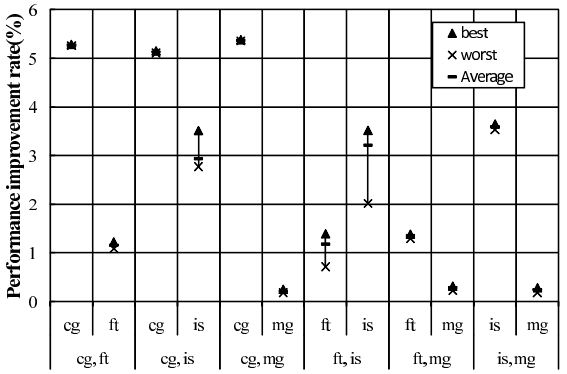

図 15　NORMAL X2 と比較したときの 2-RMAP X2 の性能向上率（32 proc/app，64 core）
Fig. 15　Performancd improvement rate of 2-RMAP X2 compared with NORMAL X2 (32 proc/app, 64 core).

4-RMAP X4 の 4 つのアプリケーションの平均の性能向上率は，64 コアで平均 1.3%，256 コアで平均 3.4%である．64 コアと 256 コアでは大きな性能低下がなく，4-RMAP X4 が効果的であるといえる．しかし，1024 コアを用いた実験では is が平均 5.2%性能が低下している．この理由については 4.4 節で考察する．

図 15，**図 16** に 2-RMAP X2 の性能向上率を示す．性能評価方法は 4-RMAP X4 と同

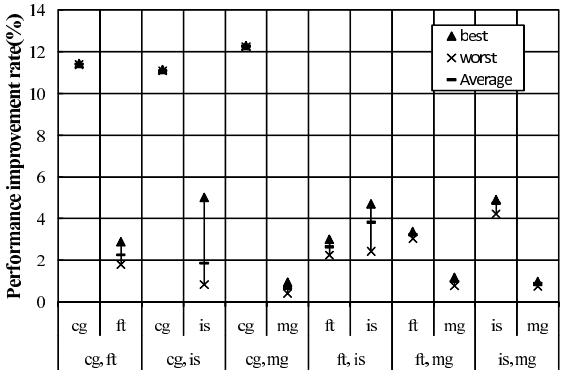

図 16　NORMAL X2 と比較したときの 2-RMAP X2 の性能向上率（128 proc/app，256 core）
Fig. 16　Performancd improvement rate of 2-RMAP X2 compared with NORMAL X2 (64 proc/app, 128 core).

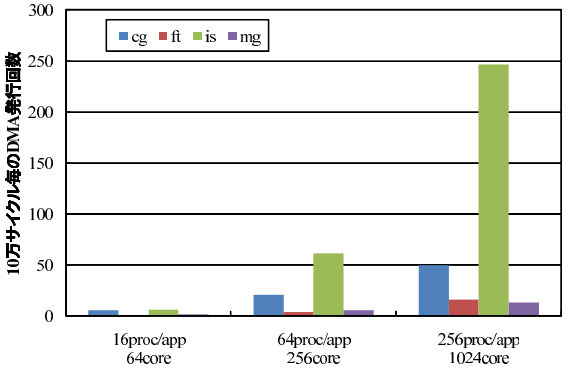

図 17　10 万サイクルごとの DMA 発行回数（NORMAL 64 プロセス）
Fig. 17　The number of DMA transfer every 100,000 cycle (NORMAL 64 proc).

一である．実験は cg，ft，is，mg から 2 つを選択し，すべての組合せを試した．2-RMAP X2 は 64 コアで平均 2.5%，256 コアで 4.7%性能が向上した．2-RMAP X2 は効果的であるといえる．

4.4 考　　察

4-RMAP X4 はランダム通信においてスループット，レイテンシともに性能低下が見られた．しかし，NPB を用いた実験では大きな性能低下が見られず，性能向上を示すものがほとんどであった．ランダム通信ではつねに通信が発生するのが，実アプリケーションでは通信は局所的に発生し，アプリケーションが通信を行っていない期間が存在する．この期間を利用し，アプリケーションの性能が向上したと考えられる．

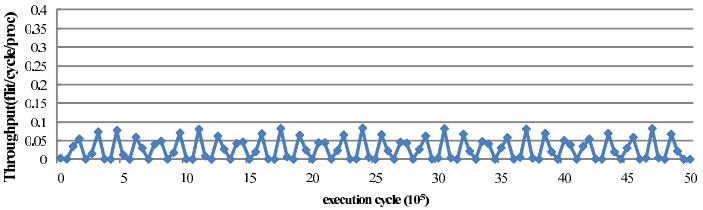

図 18　cg の通信挙動（NORMAL 64 proc）
Fig. 18　Network usage of cg (NORMAL 64 proc).

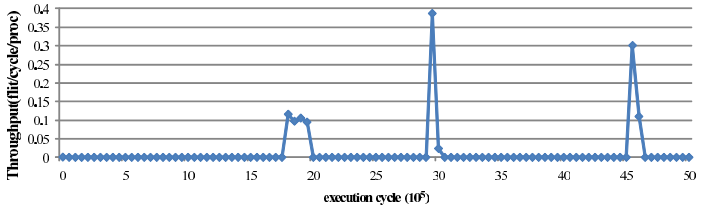

図 19　ft の通信挙動（NORMAL 64 proc）
Fig. 19　Network usage of ft (NORMAL 64 proc).

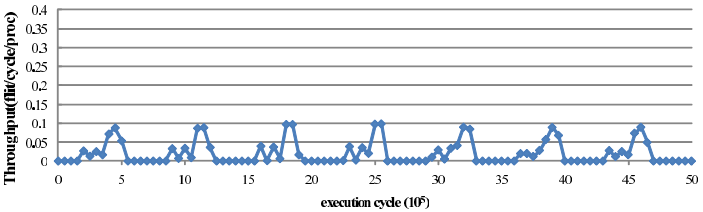

図 20　is の通信挙動（NORMAL 64 proc）
Fig. 20　Network usage of is (NORMAL 64 proc).

そこで，ネットワークの挙動を調べ，通信の局所性がどの程度存在するか示す．**図 18**，**図 19**，**図 20**，**図 21** に cg，ft，is，mg の通信挙動を示す．測定には 64 プロセス，配置 NORMAL を使用した．グラフは 10 万サイクルごとのデータ転送量の総和を測定し，その平均スループットを示している．4 つの図から，すべてのベンチマークで通信が局所的に発

生している様子が見られる．特に，ft，mg は通信が発生していない割合がかなり高い．この局所性を利用したため，4-RMAP X4 で性能が向上したと考えられる．

ただし，図 14 の 1024 コアの is ベンチマークは性能低下を示している．この原因を探るため，各ベンチマークの 10 万サイクルごとの DMA 発行回数を測定する．その結果を**図 17** に示す．この結果，256 プロセスの is が突出して DMA 発行回数が多いことが分かる．さらに，is 中のどの MPI 関数が DMA 発行回数を増やしているのかを探るために，3 つの MPI 関数（MPI_Allreduce，MPI_Alltoall，MPI_Alltoallv）に対して通信サイズと時間を計測した．is ベンチマークにおいて，これらの 3 つ以外の関数は無視できるほど通信時間が短い[15]．3 つの関数に関する評価結果を**表 4** に示す[⋆1]．この結果から，MPI_Alltoall と MPI_Alltoallv が遅くなっていることが分かる．これらの関数は転送サイズが小さいことが分かる．このように小さなサイズの Alltoall 通信には 4-RMAP X4 は向かないことが分かる．

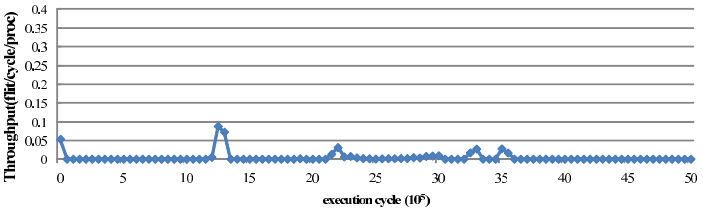

図 21　mg の通信挙動（NORMAL 64 proc）
Fig. 21　Network usage of m (NORMAL 64 proc).

表 4　is ベンチマークの通信時間
Table 4　Communication time of is benchmark.

	1 回の転送サイズ (byte)	NORMAL 通信時間	4-RMAP X4 通信時間
MPI_Allreduce	4096	4.14×10^8	3.90×10^8
MPI_Alltoall	4	4.40×10^8	5.06×10^8
MPI_Alltoallv	100 前後	4.73×10^8	5.57×10^8

⋆1 MPI_Alltoallv は通信サイズが通信ごとに異なるため，サイズを 100 前後とした．

5. オフチップメモリを含めた評価

M-Core アーキテクチャを用いた実験では，容量制限をなくした理想的なローカルメモリを想定した実験を行った．この章では，ローカルメモリを制限し，オフチップメモリとの通信を行うアーキテクチャによる評価を行う．

5.1 アーキテクチャ概要

実験に使用したアーキテクチャを説明する．**図 22** に 16×16 コアの構成を示す．小さな四角はコアを表し，大きな四角はメモリコントローラを表している．コアとメモリコントローラはチップ内の 2 次元メッシュネットワークで接続される．コアとメモリ間の通信とコアどうしの通信は，1 つの 2 次元メッシュネットワークを使用する．

コア内部のアーキテクチャを**図 23** に表す．各コアは命令キャッシュ，データキャッシュ，L2 キャッシュ，キャッシュコントローラ，DMA コントローラ（DMAC），スクラッチパッドメモリ，ルータを持つ．キャッシュコントローラは L2 キャッシュとメモリ間の操作を行う．キャッシュ容量などの詳細なパラメータは**表 5** に示す．

メモリモデルについて述べる．メモリモデルの全体像を**図 24** に示す．各コアのメモリ空

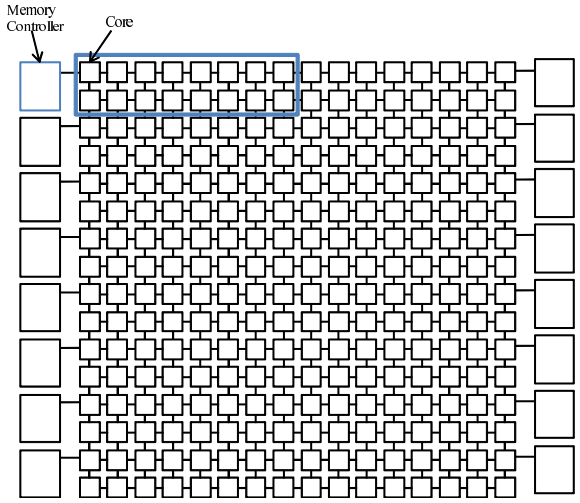

図 22　評価に用いた構成
Fig. 22　Architecture for evaluation.

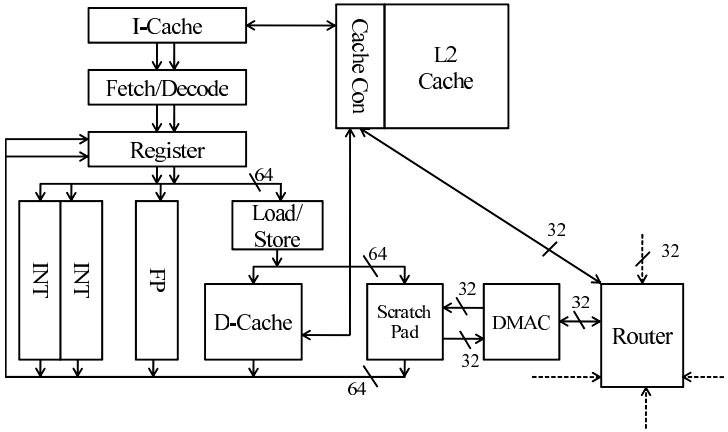

図 23　コアアーキテクチャ
Fig. 23　Core architecture.

表 5　プロセッサのパラメータ
Table 5　Processor parameters.

Number of cores	256
Number of Memory Controllers	16
Main Memory Latency	100 cycle
Main Memory Bandwidth	32 bit/cycle
L1I, L1D size	32 KB
L1I, L1D associativity	2-way
L2 size	256 KB
L2 associativity	8-way
ScratchPad Size	32 KB

間はコアごとにプライベートである．コアは 32 ビットアドレス空間をすべて使用できる．各コアは 1 つのメモリコントローラを使用する．8 × 2 コアが 1 つのメモリコントローラを共有する．図 22 の青い枠と青いメモリコントローラが 1 つのメモリコントローラを共有するコア群を表す．

メモリ空間の一部はコアが所有しているスクラッチパッドメモリにマッピングされている．このスクラッチパッドメモリを使用して，コア間の通信を行う．

通信の基本的な動作を示す．まず，送信コアはデータをスクラッチパッドメモリにコピーする．スクラッチパッドへのコピーが完了したら，DMA コントローラにデータの転送要求

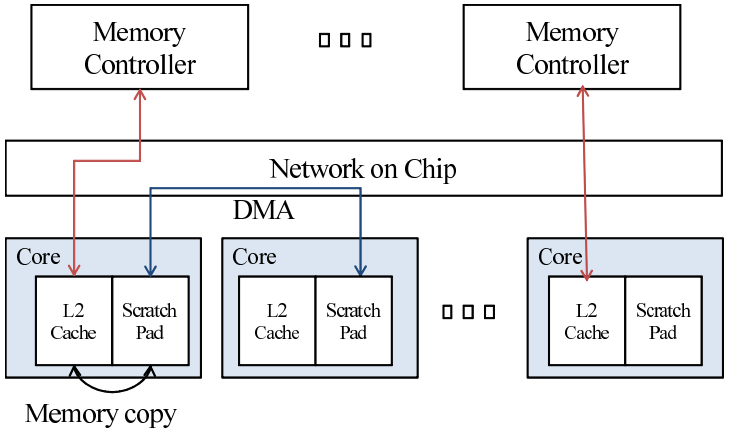

図 24　メモリモデル
Fig. 24　Memory model.

を出す．通信データは受信コアのスクラッチパッドに転送される．スクラッチパッドは容量が制限されるため，大きなデータの転送は複数回に分けて行われる．

5.2　NAS Parallel Benchmarks による評価

4.3 節と同様に NAS Parallel Benchmarks による評価を行う．評価には cg，ft，is，mg を用いる．評価対象は NORMAL X4 と 4-RMAP X4 である．計測方法は 4.3 節と同様である．実験に用いたコア数は 16 × 16 であり，図 22 と同様の構成である．

実験結果を**図 25** に示す．図は，4-RMAP X4 と NORMAL を比較したときの性能向上率である．cg ベンチマークで 2.3%の性能向上，ft ベンチマークで 1.6%の性能向上，mg ベンチマークで 27.4%の性能向上である．しかし，is ベンチマークでは 0.18%の性能低下が見られた．

mg ベンチマークが高い性能向上を示している理由について考察する．まず，各ベンチマークのキャッシュミスによる通信と DMA 通信による通信スループットを**表 6** に示す．測定は NORMAL X4 で行った．表 6 は表 3 と同じプロセス数での測定結果であるが，DMA スループットに差が見られる．これはキャッシュミスやスクラッチパッドへのコピーオーバヘッドによって，実行サイクルが増加するためである．

表 6 から mg ベンチマークはキャッシュミスによるメモリ通信が多いことが分かる．これはメモリコントローラへの通信が競合し，ボトルネックになっていると考えられる．このシ

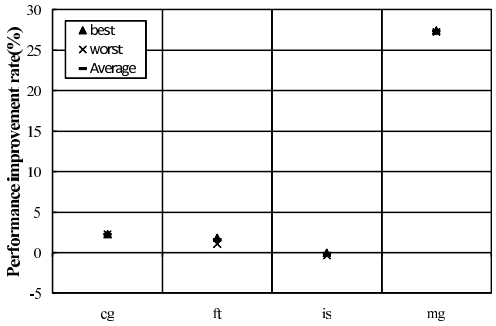

図 25　オフチップメモリ通信を含めた実験結果
Fig. 25　Evaluation result including off chip communication.

表 6　キャッシュミスによる通信スループットと DMA による通信スループット（NORMAL X4，64 プロセス）
Table 6　Cache miss throughput and DMA throughput (NORMAL X4, 64 process).

ベンチ名	実行サイクル (cycle)	Cache スループット (flit/cycle/proc)	DMA スループット (flit/cycle/proc)
cg	9.81×10^{7}	0.00×10^{-2}	3.79×10^{-2}
ft	2.91×10^{7}	1.24×10^{-2}	1.57×10^{-2}
is	1.14×10^{7}	0.00×10^{-2}	2.78×10^{-2}
mg	7.29×10^{7}	8.54×10^{-2}	0.43×10^{-2}
total	-	9.78×10^{-2}	8.52×10^{-2}

ステムは 16 コアで 1 つのメモリコントローラを使用している．このため，複数のコアが同時にキャッシュミスした場合，メモリコントローラでの競合が発生する．NORMAL のようにコアを密に配置した場合，1 つのアプリケーションが 4 つのメモリコントローラを占有して扱う．4-RMAP X4 を使用することにより，4 つのアプリケーションが 16 個のメモリコントローラを共有して使用することができる．このため，4-RMAP X4 を使用し，コア間通信の集中とメモリコントローラへの通信の集中が緩和できる．

is ベンチマークの性能が低下している理由について考察する．図 14 の評価では 256 コアで is は 3.8%の性能向上を示していた．4.3 節での評価との違いは，オフチップメモリとの通信である．表 6 からキャッシュミスによる通信が 9.78×10^{-2}（flit/cycle/proc）であり，DMA による通信は 8.52×10^{-2}（flit/cycle/proc）である．このキャッシュミスと DMA を合わせた通信量は 18.3×10^{-2}（flit/cycle/proc）は，表 3 の合計 13.5×10^{-2}（flit/cycle/proc）よりも大きい．4.4 節での考察のとおり，is はネットワークの混雑の影響を受けやすいアプリケーションである．通信量の増加により is の性能が落ちたと考えられる．

6. 関 連 研 究

タスク配置に関連する研究は多数存在する．ここではタスク配置アルゴリズムを記述した研究と，複数アプリケーションのタスク配置に関する研究について記述する．

6.1 タスク配置アルゴリズム

本論文で使用した EPAM-XY のように，タスク間の通信量から最適な配置を決定するためのタスク配置手法が数多くある．動的情報を用いたヒューリスティックな配置最適化アルゴリズムとして，PMAP [16]，NMAP [8]，ONYX [17]，CGMAP [18]，BMAP [19] などが存在している．特に BMAP は $O(N^2 \log n)$ のアルゴリズムである．本論文で使用した EPAM-XY が $O(N^4 \log n)$ であるため，BMAP を使用することで，大きなコアの最適化ができる可能性がある．しかし，BMAP は NoC のルータバッファの削減を狙うものであり，本論文が対象とするアプリケーションの性能向上とは異なるものである．

6.2 複数アプリケーションのタスク配置

文献 20) では，メッシュ接続されたマルチプロセッサのためのタスク配置について記述している．この文献では複数のアプリケーションを配置するときに，1 つのアプリケーションが分断されないように配置を行う．n-RMAP Xm のように複数のアプリケーションを均等に混ぜて配置する手法とは異なる．

マルチコアクラスタでの複数アプリケーションのタスク配置について記述しているものに，文献 21) があげられる．この文献の提案手法は，1 つのクラスタノードに複数のアプリケーションを割り当てる．そして 1 つのネットワークカードを複数のアプリケーションで共有することで性能向上を達成している．この文献での実験は，1 つのネットワークカードを 16 コアが使用している．これは M-Core のような 1 つのコアに 1 つのルータがある状況と異なっている．

7. ま と め

NoC によって接続されたコアを持つメニーコアアーキテクチャではタスク配置によって性能が変化する．特に通信衝突の削減は大きな課題である．

単一アプリケーションの通信衝突削減とバイセクションバンド幅向上のために，n ルーク問題の解をタイル状に並べるタスク配置手法 n-RMAP を提案した．さらに複数アプリケーションの性能向上のために，n ルーク問題の解を重ね合わせる手法 n-RMAP Xm を提案

した．

4-RMAP をランダム通信によって評価し，通信スループットの向上を示した．また，4-RMAP X4 はランダム通信では性能低下を引き起こすことを示した．NAS Parallel Benchmarks を対象に提案手法の評価を行ったところ，ベンチマークの性能向上率から，実アプリケーションにおいて提案手法が効果的であることを示した．また，オフチップメモリとの通信を含めた評価を行い，提案手法によってメモリコントローラの競合を抑えることができることを示した．

謝辞　本研究の一部は，科学技術振興機構・戦略的創造研究推進事業（CREST）の「アーキテクチャと形式的検証の協調による超ディペンダブル VLSI」の支援による．

参 考 文 献

1) Howard, J., Dighe, S., Hoskote, Y., et al.: A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS, *International Solid-State Circuits Conference* (*ISSCC*), pp.108–109 (2010).
2) Wentzlaff, D., Griffin, P., Hoffmann, H., Bao, L., Edwards, B., Ramey, C., Mattina, M., Miao, C.-C., III, J.F.B. and Agarwal, A.: On-Chip Interconnection Architecture of the Tile Processor, *IEEE Micro*, Vol.27, pp.15–31 (2007).
3) Bokhari, S.H.: On the Mapping Problem, *IEEE Trans. Comput.*, Vol.30, pp.207–214 (1981).
4) Yu, H., Chung, I.-H. and Moreira, J.: Topology mapping for Blue Gene/L supercomputer, *SC '06: Proc. 2006 ACM/IEEE conference on Supercomputing*, p.116 (2006).
5) 三好健文，笹田耕一，植原　昴，佐野伸太郎，森　洋介，吉瀬謙二：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，計算機アーキテクチャ研究会報告（ARC-184），Vol.2009, No.10, pp.1–7 (2009).
6) Ascia, G., Catania, V. and Palesi, M.: A Multi-objective Genetic Approach to Mapping Problem on Network-on-Chip, *Journal of Universal Computer Science*, Vol.12, No.4, pp.370–394 (2006).
7) Hu, J. and Marculescuauthor, R.: Energy- and performance-aware mapping for regular NoC architectures, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.24, No.4, pp.551–562 (2005).
8) Murali, S. and Micheli, G.D.: Bandwidth-Constrained Mapping of Cores onto NoC Architectures, *Design, Automation and Test in Europe Conference and Exhibition* (*DATE*), Vol.2, pp.896–901 (2004).
9) Hoskote, Y., Vangal, S., Singh, A., Borkar, N. and Borkar, S.: A 5-GHz Mesh Interconnect for a Teraflops Processor, *IEEE Micro*, Vol.27, No.5, pp.51–61 (2007).
10) Sankaralingam, K., Nagarajan, R., Mcdonald, R., et al.: Distributed Microarchitectural Protocols in the TRIPS Prototype Processor, *International Symposium on Microarchitecture* (*MICRO-39*), pp.480–491 (2006).
11) 植原　昴，佐藤真平，吉瀬謙二：メニーコアプロセッサの研究・教育を支援する実用的な基盤環境，電子情報通信学会論文誌，Vol.93, No.10, pp.2042–2057 (2010).
12) 建部修見，児玉祐悦，関口智嗣，山口喜教：リモートメモリ書き込みを用いた MPI の効率的実装，情報処理学会論文誌，Vol.40, No.5, pp.2246–2255 (1999).
13) 森本健司，松本　尚，平木　敬：メモリベース通信を用いた高速 MPI の実装と評価，情報処理学会論文誌，Vol.40, No.5, pp.2256–2268 (1999).
14) Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Schreiber, R.S., Simon, H.D., Venkatakrishnan, V. and Weeratunga, S.K.: The NAS parallel benchmarks, *NASA Technical Report* (1991).
15) Faraj, A. and Yuan, X.: Communication Characteristics in the NAS Parallel Benchmarks, pp.729–734 (2002).
16) Koziris, N., Romesis, M., Tsanakas, P. and Papakonstantinou, G.: An Efficient Algorithm for the Physical Mapping of Clustered Task Graphs onto Multiprocessor Architectures, *Proc. 8th Euromicro conference on Parallel and distributed processing*, pp.406–413 (1999).
17) Janidarmian, M., Khademzadeh, A. and Tavanpour, M.: Onyx: A new heuristic bandwidth-constrained mapping of cores onto tile-based Network on Chip, *IEICE Electronics Express*, Vol.6, No.1, pp.1–7 (2009).
18) Moein-darbari, F., Khademzade, A. and Gharooni-fard, G.: CGMAP: A new approach to Network-on-Chip mapping problem, *IEICE Electronics Express*, Vol.6, No.1, pp.27–34 (2009).
19) Shen, W.-T., Chao, C.-H., Lien, Y.-K. and Wu, A.-Y.A.: A New Binomial Mapping and Optimization Algorithm for Reduced-Complexity Mesh-Based On-Chip Network, *Proc. 1st International Symposium on Networks-on-Chip* (*NOCS*), pp.317–322, Washington, DC, USA, IEEE Computer Society (2007).
20) Kim, G. and Yoon, H.: On Submesh Allocation for Mesh Multicomputers: A Best-Fit Allocation and a Virtual Submesh Allocation for Faulty Meshes, *IEEE Trans. Parallel and Distributed Systems*, Vol.9, pp.175–185 (1998).
21) Koop, M., Luo, M. and Panda, D.: Reducing network contention with mixed workloads on modern multicore, clusters, *IEEE International Conference on Cluster Computing and Workshops* (*CLUSTER*), pp.1–10 (2009).

(平成 23 年 1 月 28 日受付)
(平成 23 年 6 月 14 日採録)

佐野伸太郎（学生会員）

2010 年東京工業大学工学部情報工学科卒業．現在，同大学大学院情報理工学研究科修士課程在学中．コンピュータアーキテクチャに関する研究に従事．

吉瀬　謙二（正会員）

1995 年名古屋大学工学部電子工学科卒業．2000 年東京大学大学院情報工学専攻博士課程修了．博士（工学）．同年電気通信大学大学院情報システム学研究科助手．2006 年東京工業大学大学院情報理工学研究科講師．2011 年同准教授．計算機アーキテクチャ，並列処理に関する研究に従事．電子情報通信学会，IEEE-CS，ACM 各会員．