Vol.2012-ARC-199 No.4
2012/3/27

高機能ルータを利用したDMR実行メニーコアにおける効率的なタスク配置手法の検討

高前田（山崎） 伸也[†1,†2] 佐 藤 真 平[†1,†2] 吉 瀬 謙 二[†1]

本稿では高機能ルータにより多重実行を行うメニーコアプロセッサを対象に，高い信頼性と高いアプリケーション性能を両立するタスク配置手法を検討する．高機能ルータは，オンチップルータが通常持つ機能に加えて（1）パケットの複製,（2）パケット宛先の変更，および（3）パケットの比較の機能を有するルータである．メニーコアプロセッサにおいてはタスク割り当ての方式によってアプリケーション性能が変化する．特に，高機能ルータによる冗長実行系においては，通常の通信に加えて，冗長実行のための通信が付加されるため，各タスクの配置方法に加えて冗長実行を行うペアの配置方法を配慮する必要がある．通信密度の高いタスクを隣接させて配置することで通信に起因する性能低下を抑制することが可能である．一方で，冗長実行のペアを隣接させて配置すると，電源電圧の変動やノイズおよび宇宙線など，同一要因で両方のコアが同時に故障モードに入る可能性が高くなる．そのため，より高い信頼性を実現するには，冗長実行のペアをできるだけ離して配置する必要が出てくる．本稿では，単一のアプリケショーンではなく，複数のアプリケーションを同時に多重実行した場合に有効なタスク配置手法を検討する．冗長実行のペア間距離が小さい配置と大きい配置をそれぞれ数種類のタスク配置の性能を計測し，冗長実行による性能低下を抑えながらペア間距離を確保することが出来る配置を探る．実験の結果，通信頻度の大きいアプリケーションでは，アプリケーション間の競合を減らすRMAPの配置をベースに冗長実行ペアの距離を離すことで，アプリケーション性能と高信頼性を両立するタスク配置が実現可能であることがわかった．

†1 東京工業大学 大学院情報理工学研究科
†2 日本学術振興会 特別研究員

1. はじめに

単一チップに複数のコアを搭載するマルチコアプロセッサは高性能計算にはもちろん，携帯電話などの電力制約のより厳しい組み込み機器まで，幅広い範囲で用いられている．更なる半導体プロセスの進歩および三次元実装技術の発展により，さらに多くのコアを搭載するメニーコアプロセッサの登場が期待されている．実用的なメニーコアプロセッサの実現には，回路・アーキテクチャ・ソフトウェア，いずれかのレベルでプロセッサの信頼性を保証する必要がある．以前より我々は，オンチップルータが通常持つ機能に加えて，パケットの複製，パケット宛先の変更およびパケットの比較の機能を持つ**高機能ルータ**を軸とした冗長実行機構SmartCoreシステム[1),2)]を提案している．メニーコアプロセッサがコア間接続に用いるネットワークオンチップ（Network on Chip, NoC）に冗長実行を支援する機能を追加することで，メニーコアが持つコア冗長性を活用した多重実行を可能とする．SmartCoreシステムでは，プロセッサ中の2つのコアがペアを形成し同一のスレッドを実行する．そして，各コアが出力するパケットをルータのレベルで比較することで，コアで発生した誤りを検出する．

メニーコアプロセッサにおいては，どのタスク・スレッドをどのコアに割り当てるかにより，アプリケーションおよびプロセッサ全体のスループットが変化する[3)]．特に，SmartCoreシステムによる冗長実行系においては，通常のアプリケーションの枠組み内で発生する通信に冗長実行のための通信が加わる．そのため，アプリケーションおよびプロセッサのスループットの観点から，各タスクの配置方法はもちろん，冗長実行を行うペアの配置方法を配慮する必要が出てくる．一般的には，通信密度の高いスレッドを隣接させて配置することで，通信に起因する性能低下を抑制することが可能である．しかし，冗長実行のために頻繁に通信を行う冗長実行のペアを隣接させて配置すると，電源電圧の変動やノイズおよび宇宙線など，同一要因で両方のコアが同時に故障モードに入る可能性が高くなる．そのため，より高い信頼性を実現するには，冗長実行のペアをできるだけ離して配置する必要が出てくる．

本稿では，高機能ルータにより二重実行を行うメニーコアプロセッサを対象に，高い信頼性と冗長実行による性能低下を抑えるタスク配置手法を検討する．我々は以前，単一のアプリケーションを二重実行する場合のタスク配置方式に起因するアプリケーション性能への影響について，いくつかのマイクロベンチマークを用いて調査した[4)]．その結果，単一アプリケーションの場合には，ペア間の距離よりも，ペア間通信の重複が少ない配置が性能には有効であることを確認した．本稿では，単一のアプリケショーンではなく，複数のアプリケー

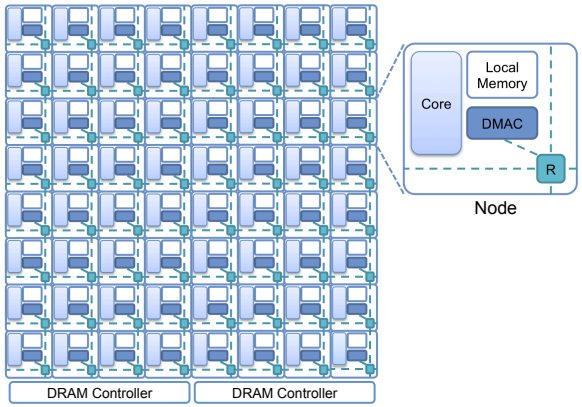


図 1 M-Core アーキテクチャ

ションを同時に多重実行した場合に有効なタスク配置手法を検討する．冗長実行のペア間距離が小さい配置とペア間距離の大きい配置をいくつか試すことで，冗長実行による性能低下を抑えつつ，高信頼性のためのペア間距離を確保することができる配置を探る．

本稿の構成を以下に示す．2 章では，ベースとするメニーコアプロセッサのアーキテクチャ，および高機能ルータにより二重実行を行う SmartCore システムのアーキテクチャについて述べる．3 章では，タイル型メニーコア上に複数のアプリケーションを同時に配置する手法について述べる．4 章では，いくつかのタスク配置方式について，FPGA システムを用いてアプリケーション性能への影響を評価する．そして 5 章で本稿をまとめる．

2. 高機能ルータによる冗長実行

本稿では特に，Cell/B.E[5] と同様に各コアがスクラッチパッドメモリを持ち，明示的な DMA 転送によりデータ共有を行う M-Core アーキテクチャ[6),7)] を対象に，高機能ルータによる冗長実行およびタスク配置の議論を行う．

2.1 ベースとするプロセッサアーキテクチャ

図 1 に M-Core アーキテクチャ全体の構成を示す．M-Core では，コア (図中，Core)，ローカルメモリ (図中，Local Memory)，DMA コントローラ (図中，DMAC) およびオンチップネットワークのルータ (図中，R) をそれぞれ 1 つずつ含むノード (図中，Node) を単位とする．これらを 2 次元メッシュネットワークで接続し，各ノードはそれぞれの位置に応

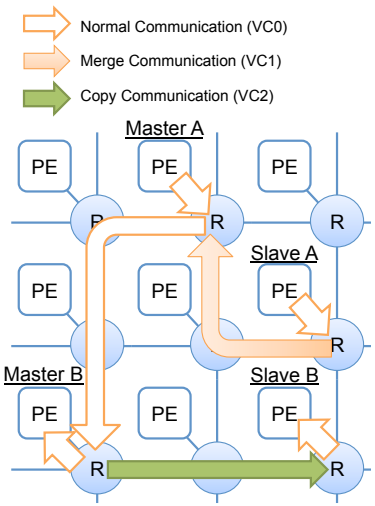


図 2 高機能ルータによる DMR 実行

じたユニークな ID(ノード ID) を持つ．各ノードのコアは，自ノードのローカルメモリに格納されている命令を実行し，処理を行う．各ローカルメモリのアドレス空間は独立しており，各コアはそれぞれ自ノードのローカルメモリに対しては，ロード・ストア命令を介して直接アクセスする．ローカルメモリのアドレスは仮想化されていない．また，他ノードのローカルメモリまたはオフチップのメインメモリに対しては DMA コントローラを介して DMA 転送を発行することでアクセスすることができる．DMA 転送には，ノード ID により，どのノードのローカルメモリに対する DMA なのかを指定し，加えて転送元と転送先それぞれのローカルメモリ中のアドレスと転送サイズを指定する必要がある．DMA コントローラは DMA 転送が発行されると，指定されたローカルメモリアドレスから指定されたサイズをゼロコピーで送信する．すなわち，送信用バッファに送信データの退避は行わない．

2.2 SmartCore システム

次に我々が提案する，高機能ルータの支援により二重実行を行う冗長系の SmartCore システムについて述べる．本稿では特に高機能ルータの支援により 2 つのコアが DMR (Dual Modular Redundant) ペアを形成する場合の議論を行う．**図 2** に，SmartCore システムにおいて，2 つのコアが DMR ペアを形成し冗長実行する場合の通信パケットが流れる様子

の例を示す．図において，**Master A** と **Slave A**，**Master B** と **Slave B** がそれぞれ DMR のペアを形成しており，それぞれのペアは同一のスレッドを実行する．Master A と Master B は本来スレッドを実行するノードとし，Slave A と Slave B は冗長実行のために割り当てられたノードとし，それぞれを **Master ノード**，**Slave ノード**[⋆1] と呼ぶ．ここで Master A から Master B へと通信をする場合を考える．信頼性向上のためには，冗長実行の結果を適切に比較する必要がある．SmartCore システムでは，ペアが生成するパケット列をルータのレベルで比較することで，ペアの片方に誤りが発生したことを検出する．図の例の場合には，Slave A の PE が出力したパケット列は Slave A のルータ (R) により宛先が書き替えられ，Master A のルータへとフォワードされる．Master A のルータでは Master A の PE が出力するパケット列と Slave A からフォワードされるパケット列の 2 つを待ち合わせし，2 つのパケット列の内容を先頭から順にフリットレベルで比較する．このとき，PE において発生したフォールトにより計算結果が変化し，それぞれの PE が出力するパケット列の内容が一致しないことがある．SmartCore システムではこのことにより PE で発生した誤りを検出する．もし，ルータでのパケット比較の結果，誤りが検出された場合には，当該スレッド・アプリケーションの実行停止および再実行などの回復処置が必要となる．比較の結果，パケット内容に差異がない場合には，2 つのパケットをマージして，通常のパケットと同じように本来の宛先コア (Master B) へとパケットは移動する．宛先コアの Master B へ到着したパケットは Master B の PE へと移動しつつ，Slave B へとコピーされる．これにより Master B と Slave B は同一のパケット列を受信し，同じデータ列を利用して計算を継続することができる．

ここで，SmartCore システムで発生する通信を 3 つに分類する．図の例において，Slave A から Master A への通信はマージ通信 (Merge Communication)，Master A から Master B への通信は通常の通信 (Normal Communication)，Master B から Slave B への通信はコピー通信 (Copy Communication) と呼ぶ．二重実行しない通常系の場合では，この 3 種のうちマージ通信とコピー通信は発生せず，通常の通信のみが発生する．一方，二重実行する場合には，これに加えてマージ通信とコピー通信が発生する．これらの通信の経路は，X-Y 次元順ルーティングの場合には DMR ペアの配置に従って一意に決定できる．

2.3 高機能ルータアーキテクチャ

図 3 に高機能ルータのアーキテクチャを示す．2 つのノードが DMR 実行のペアを構成するために，SmartCore システムにおける高機能ルータは，通常のオンチップルータの機能に加えて，次の 3 つの機能を持つ．

- パケットの複製 （図中 (a)）
- パケット宛先の変更 （図中 (b)）
- パケットの比較 （図中 (c)）

Master ノードに届いたパケットは，ペア間で同一の実行フローをたどるために，Slave ノードへとコピーする必要がある．Master ノードのコア方向（出力 C）へフリットが移動するときに，同時に図中 (a) の Copy Buffer へとフリットがコピーされる．このとき，Copy Buffer に格納されるパケットの宛先は図中 (b) の ID Translator により Slave ノードを指し示すように書き替えられる．一方，Slave ノードのコア方向から入力されたパケットの宛先は，入力 C の入力段にある ID Translator により Master ノードへと変更される．その後，入力されたパケットは一般のルータと同様に，入力バッファに格納され，仮想チャネルおよ

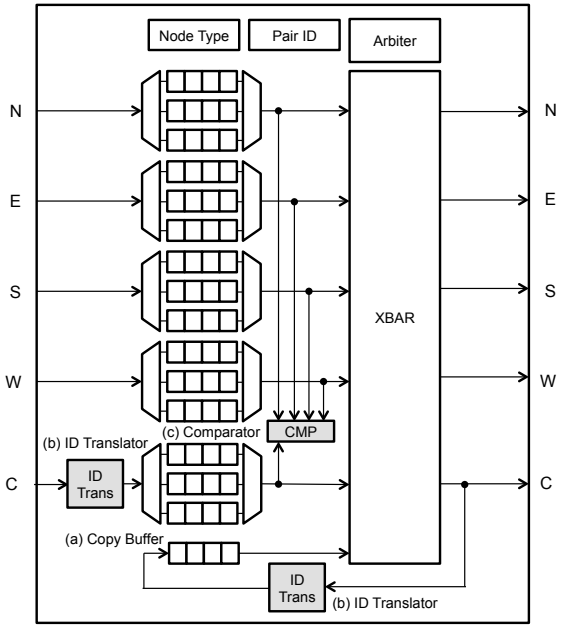


図 3 高機能ルータアーキテクチャ

⋆1 または Mirror ノード．本稿では略語の分かりやすさを優先し Slave ノードと呼ぶこととする．

びクロスバーの調停が行われた後で，出力段へと移動する．クロスバーはコピーされたパケットを格納する Copy Buffer からの入力も含むため，一般のルータよりも入力ポート数の多い，6 入力-5 出力のものに拡張されている．冗長実行の結果が正しいかどうかは，図中 (c) の Comparator により，パケット・フリットの内容を比較することで確認する．Master ノードにおいて，コア方向からの入力パケットと Slave ノードからフォワードされてきたパケットを Comparator で順次比較する．

ベースのアーキテクチャのルーティングには X-Y 次元順ルーティングを採用している．しかし，Slave ノードから Master ノードへの通信のマージ通信，Master ノード間の通常の通信，ならびに Master ノードから Slave ノードへのコピー通信のこれら 3 種の通信の切り替わるポイントでは，次元順ではあり得ないターンをする可能性があり，ネットワークにデッドロックが発生する可能性がある．そこで，デッドロックを回避するために，本稿で取り扱う高機能ルータでは 3 本の仮想チャネルを用いる．通信種類毎に専用の仮想チャネルを利用し，種別が切り替わるポイントで仮想チャネルを切り替えることで，デッドロックを回避する．

3. 複数アプリケーションのタスク配置

メニーコアプロセッサにおいて，「どのタスクをどこに配置するか」という**タスク配置**はアプリケーション性能およびプロセッサのスループットに影響を与える．特に，2 つのコアがペアを構成し冗長実行を行う SmartCore システムの場合には，アプリケーションで定義された通常の通信に加えて，冗長実行のためのマージ通信とコピー通信が発生するため，より注意してタスク配置を決定する必要がある．冗長実行による性能低下を回避するシンプルな方法として，冗長実行のペアをできるだけ近くに配置するという方法があげられる．しかし，冗長実行のペアを近くに配置すると，電源電圧の変動や GND ノイズ，および宇宙線など，同一の要因で両方のコアが故障モードに突入する可能性が高くなると考えられる．そのため，信頼性確保の観点からはペアはできるだけ離して配置することが好ましい．

本稿では，性能と信頼性を両立する効率的な複数アプリケーションのタスク配置手法を探るために，いくつかの配置について実験を行う．16 スレッドの並列アプリケーションを 2 つ同時にプロセッサ上に配置し，すべてのスレッドを SmartCore システムにより冗長実行する場合のタスク配置を考える．そのため全体でのスレッド数は，16 スレッド・2 アプリケーションの 2 重実行のため，合計 64 となる．

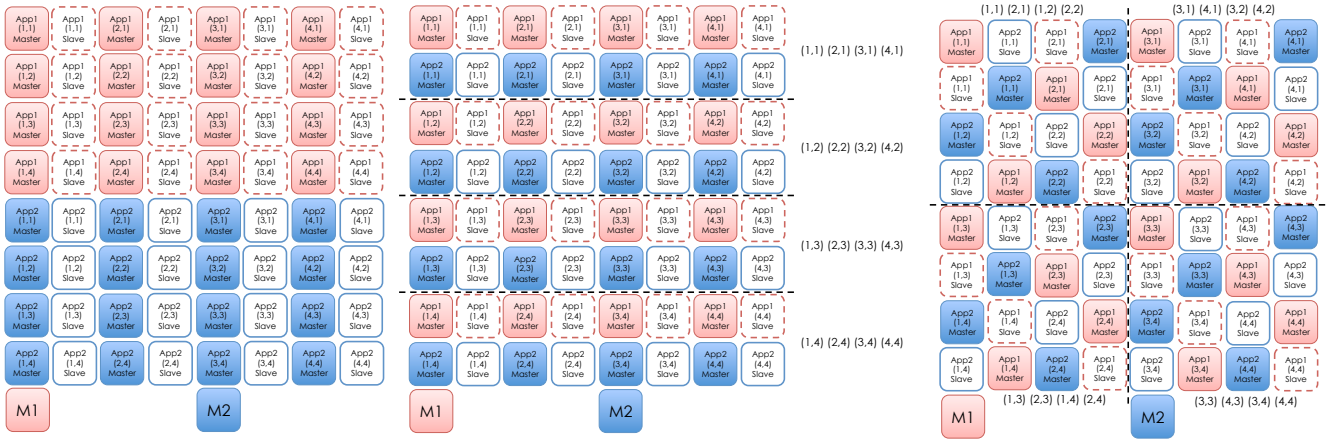
まず，**図 4** に Master ノードと Slave ノード間の通信ホップ数[⋆1]が 1 である配置を 3 種示す．すべての配置において，図中の **M1** および **M2** はそれぞれ，アプリケーション 1 が利用するオフチップメモリ，アプリケーション 2 が利用するオフチップメモリを示す．図 4(a) は 2 つのアプリケーションを分離した配置，図 4(b) は (a) の配置を Y 軸上で交互の配置したもの，図 4(c) は文献 3) で提案している RMAP X4 を適用したものである．具体的には，4 × 4 の 16 ノードのブロックを 4 つの 4 ルーク問題の 4 つ解の重ね合わせとして見て，それぞれの解に，2 つのアプリケーションの Master ノード，Slave ノード，計 4 種類のノードを 1 種類ずつ割り当てた配置である．これら 3 つの配置は Master ノードと Slave ノード間の通信ホップ数が 1 と隣同士に配置されているため，冗長実行のためのマージ通信ならびにコピー通信が性能に与える悪影響は少ないと考えられる．

次に，**図 5** に Master ノードと Slave ノード間の通信ホップ数が 1 より大きい配置を 4 種示す．図 4 と同様に，すべての配置において，図中の **M1** および **M2** はそれぞれ，アプリケーション 1 が利用するオフチップメモリ，アプリケーション 2 が利用するオフチップメモリを示す．図 5(a) は各アプリケーションの Master ノード群，Slave ノード群をそれぞれ隣接させた配置，図 5(b) は図 4(b) の Slave ノードの位置を Y 軸方向で入れ替えたもの，図 5(c) は図 5(b) の Slave ノードの位置を更に X 軸方向で入れ替えたものである．図 5(d) は図 4(d) の Slave ノードの位置をブロック単位で入れ替えたものである．それぞれの Master ノードと Slave ノードとの間の平均距離は (a) は 4, (b) は 5, (c)(d) は 8 である．これらの配置では，Master ノードと Slave ノード間の通信ホップが図 4 のものよりも大きいため，冗長実行のためのマージ通信とコピー通信による性能低下幅が大きくなると考えられる．

4. 性 能 評 価

図 4，図 5 に示したの計 7 つのタスク配置のアプリケーションの性能を FPGA シミュレータを用いて評価する．評価には，多数の FPGA で構成したメニーコアプロセッサシミュレータである ScalableCore システム 3.4[8),9)] を用いた．高機能ルータを搭載する M-Core メニーコアプロセッサを Verilog HDL で記述し，ScalableCore システム上に実装した．**表 1** に評価時のプロセッサ構成を示す．16 スレッドの並列アプリケーションを 2 つ同時にプロセッサ上に配置し，すべてのスレッドを SmartCore システムにより DMR 実行し，タスク配置の影響を観測した．シミュレーションサイクル数をおよそ 350M サイクルとし，2 つのアプ

⋆1 X-Y 次元順ルーティングのホップ数



(a) Separate (b) Interleave (c) RMAP

図 4 Master-Slave 間の距離が 1 であるタスク配置

表 1 評価時のプロセッサ構成

Core	MIPS32 ISA, 5-stage, Single-issue Memory access port: 2 (Fetch, Load/Store)
DMA Controller	Memory access port: 2 (32-bit DMA Read, 32-bit DMA Write)
Router	5-input/output (North, East, West, South, Core), 4-stage (NRC/VA: Next Routing Computation and Virtual Channel Allocation, SA: Switch Allocation, ST: Switch Traversal, LT: Link Traversal), 3-Virtual-Channel, FIFO depth: 4, Credit-base flow control X-Y Dimension Routing
Local Memory	Access Latency: 1, 512KB, 32-bit 4-port (Core Fetch, Core Load/Store, DMA Read, DMA Write)
# Node	64 (8 × 8)
# DRAM Controller	2 (Location (X,Y): App1 (1,9), App2 (5,9))

リケーションを繰り返し実行した[⋆1⋆2]．また，本実行サイクル数は，アプリケーション配布およびシミュレーション結果の通知[⋆3]に要する時間は含まない，アプリケーション実行開始から完了までの正味のサイクル数である．評価用アプリケーションには，図 4 および図 5 の App1 として並列ソートの **Bitonic Sort** を，App2 として Cannon のアルゴリズムにより並列化された行列積 (**Matrix Multiply**) を用いた．アプリケーション構成を以下に示す．

Bitonic Sort: 並列ソート[10]．データサイズは 1M エントリー，4MB とした．シミュレーションした 350M サイクル中で 3 回繰り返し実行した．

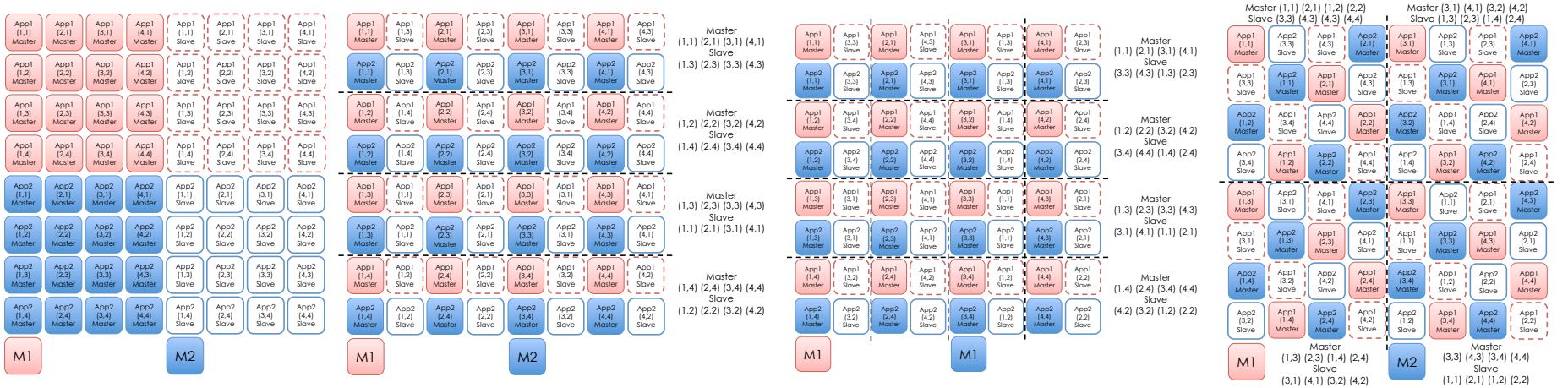
Matrix Multiply: Cannon アルゴリズム[11] により並列化された行列積．データサイズは 262144 エントリー (512×512)，1MB とした．シミュレーションした 350M サイクル中で 4 回繰り返し実行した．

まず，図 4(a) Separate，(b) Interleave，(c) RMAP ならびに図 5(a) Block の 4 つ配置

⋆1 評価に用いた FPGA システムでは，アプリケーション中の標準出力の影響により，実行毎に若干の実行サイクル数の差異が発生することがある．しかし，本実験においては評価の正当性のために，2 回同じシミュレーションを行ったが，アプリケーションの実行サイクル数は変化しなかった．

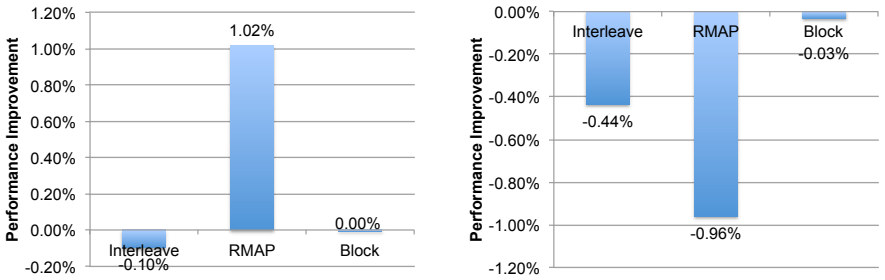
⋆2 350M サイクルのシミュレーションおよびシステム・アプリケーションの初期化に要する時間はおおよそ 6 分程度であり，ソフトウェアシミュレータと比較して 80 倍以上高速に同様のシステムをシミュレーションする．

⋆3 1 つ目のアプリケーションをプロセッサ上の当該コアにプログラムローダにより配布したあと，2 つ目のアプリケーションを当該コアに配布し，その後，すべてのコアを順次起動しアプリケーションの実行を開始した．各アプリケーションは規定回数の実行完了後にホスト PC に接続されている FPGA ノードに実行サイクル数を通知した．



(a) Block (b) Interleave ROT (c) Interleave ROTx2 (d) RMAP ROT

図 5 Master-Slave 間の距離が 1 より大きいタスク配置



(a) Bitonic Sort (b) Matrix Multiply

図 6 DMR 実行しない場合の各タスク配置適用による性能の変化

において DMR 実行しない場合の性能への影響を評価する．各配置において，Slave ノードの位置には何も実行しないアイドルスレッドを配置し，Saparate の性能を基準にしたときの性能の変化を**図 6** に示す．Bitonic Sort は，Interleave の配置で通信レイテンシが増加し，わずかではあるが 0.1%性能が低下した．Block の配置では有意な性能の変化は観測できなかった．RMAP の配置では通信の衝突が削減され 1.02%の性能向上を達成した．一方，Matrix Multiply ではすべての配置で性能が低下した．特に，RMAP の適用により 0.96%性能低下した．これは Matrix Multiply は元々通信衝突が少なく，RMAP の配置による通信ホップ数の増加および Bitonic Sort の通信と経路を共有することによる通信衝突の増加が性能に悪影響を与えたと考えられる．

次に，**図 7** に Bitonic Sort の各タスク配置における性能向上率を，**図 8** に Matrix Multiply の各タスク配置における性能向上率を示す．各タスク配置において，Master ノードと Slave ノード間の平均距離を X 軸に，図 4(a) Separate の配置で DMR 実行しない場合の性能を基準としたときの性能向上率を Y 軸にプロットした．

Bitonic Sort の場合，図 4(c) RMAP を除くすべての配置において冗長実行により性能が低下している．Separate および Interleave においてはそれぞれ 0.83%，0.79%の性能低下が発生しており，以前の報告と同様にこれは DMR 実行の影響と考えられる[4]．RMAP の配置においては同一アプリケーション内での通信衝突が削減され，DMR 実行にも関わらず 0.41%の性能向上を達成した．Master ノードと Slave ノードのペア間距離が大きい配置では，配置によって性能低下幅にばらつきがみられた．図 5(a) Block，(b) Interleave ROT，

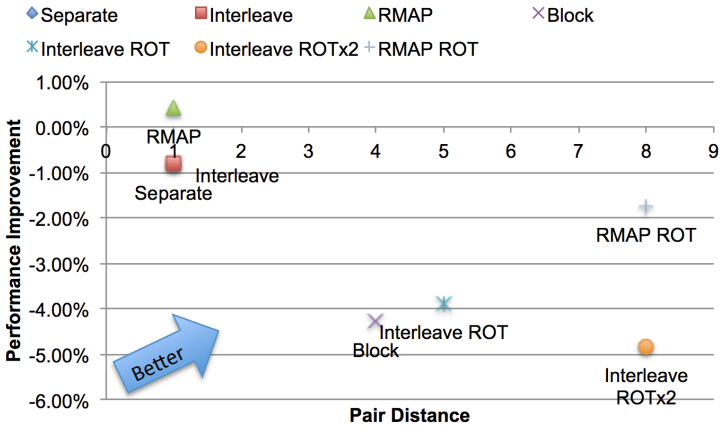


図 7 各タスク配置におけるペア間距離と性能の関係 (Bitonic Sort)

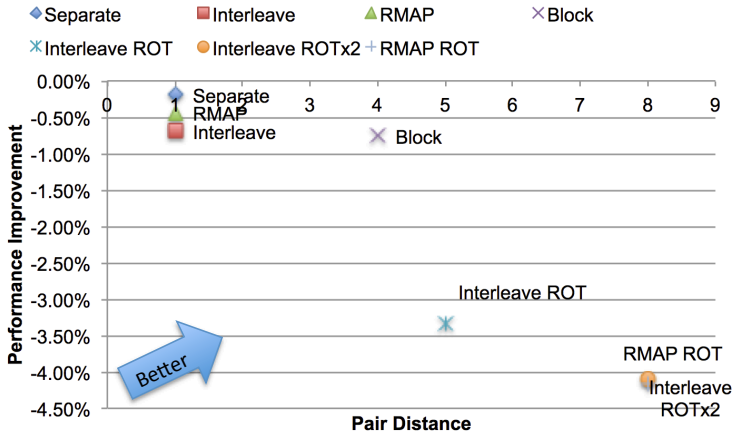


図 8 各タスク配置におけるペア間距離と性能の関係 (Maxrix Multiply)

(d) Interleave ROTx2 の 3 つにおいては，それぞれ 4.28%，3.88%，4.83%の性能低下が確認された．Interleave ROT の方が Block よりも性能低下が少ない理由としては，マージ通信およびコピー通信が利用できるバイセクションバンド幅がより増加したからであると考えられる．一方で図 5(d) RMAP ROT では，Interleave ROTx2 と同等のペア間距離にも関わらず，わずか 1.75%の性能低下であった．RMAP ROT は RMAP の Slave ノードの位置を Master ノード-Slave ノード間の距離が遠くなるように入れ替えた配置であるが，RMAP と同様に同一アプリケーション内の通信衝突は少ない配置である．そのため，同量のコピー通信・マージ通信が発生するにも関わらず，Interleave ROTx2 よりも通信の集中が軽減され，性能低下幅が軽減できたと考えられる．

Matrix Multiply の場合，すべての配置で性能が低下した．Master ノード-Slave ノードペア間の距離が小さい Separate，Interleave，RMAP の配置では 0.19%，0.69%，0.44%の性能低下が発生した．また，ペア間距離の大きい配置のうち，Interleave ROT，Interleave ROTx2，RMAP ROT では 3.33%，4.09%，4.17%と大きく性能低下が発生した．しかし，これらの配置と同様にペア間距離の大きい Block においては，0.76%と性能低下幅が小さかった．これは，Block の配置は他のアプリケーションとの経路共有が少なく通信衝突増加が少ないからであると考えられる．Matrix Multiply は DMR 実行しない場合には RMAP の適用により性能低下しており，DMR 実行した場合でもマージ通信・コピー通信についても通信衝突があまり増加せず，RMAP を適用することにより通信レイテンシ増加し，性能低下がより顕著になったと考えられる．一方，DMR 実行しない場合に RMAP 適用で性能向上した Bitonic Sort では，ペア間ホップ数の多い RMAP ROT においてもマージ通信・コピー通信を含む通信衝突が解決できたため，性能低下が小さくなったと予想される．

これらのことから，通信量・通信衝突の多いアプリケーションに対しては，冗長実行のペア間距離を増加させても，RMAP などの通信衝突を減らすタスク配置により性能低下を抑えることができることがわかった．一方で，通信衝突の少ないアプリケーションに対しては，冗長実行のペア間距離の増加による通信レイテンシ増加および他のアプリケーションとの通信衝突の増加による性能低下が顕著に現れるため，各アプリケーションを分離して配置する方が高い性能を得られることがわかった．

5. ま と め

本稿では，高機能ルータにより二重実行を行うメニーコアプロセッサを対象に，いくかのタスク配置を試すことにより，高い信頼性と冗長実行による性能低下を抑えるタスク配置手法を検討した．その結果，通信量および通信衝突の多いアプリケーションに対しては，冗長

実行ペア間距離を増加させても，タスクを混ぜた通信衝突を減らすタスク配置により，性能低下を抑えることが可能であることを確認した．

今後の課題として，通信ホップ数増加および他のアプリケーションとの通信衝突による通信レイテンシの増加がもたらす性能の低下が顕著なアプリケーションに対して有効な，ペア間距離を保ちつつ性能低下を抑えることができるタスク配置を検討する．具体的には，通信衝突の多いアプリケーションの性能向上と両立するために，アプリケーションを混ぜる方式と分離する方式のハイブリッドを検討する．また，本評価では冗長実行ペア間距離は大きいほどより高信頼であるとしたが，実際には通信経路上で発生する故障等を考慮する必要がある．そのため，NoC の回路面積比率や宇宙線の特徴などから，冗長実行ペア間距離と信頼性の関係を考察する必要がある．

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) Takamaeda, S., Sato, S., Miyoshi, T. and Kise, K.: Smart Core System for Dependable Many-Core Processor with Multifunction Routers, *International Conference on Networking and Computing (ICNC2010)*, pp.133 –139 (2010).
2) 佐藤真平，植原　昂，吉瀬謙二：メニーコアプロセッサのオンチップネットワーク性能を向上させる SmartCore システム，先進的計算基盤システムシンポジウム SACSIS2009 論文集，pp.27–35 (2009).
3) 佐野伸太郎，吉瀬謙二：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，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, pp.96–109 (2011).
4) 池田貴一，佐藤真平，吉瀬謙二：冗長実行時の SmartCore システムの性能評価，情報処理学会研究報告 2011-ARC-197， Vol.2011, No.197, pp.1–8 (2011).
5) : Cell Broadband Engine. http://cell.scei.co.jp/.
6) 植原　昂，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 システム開発論文特集号， Vol.J93-D, No.10, pp. 2042–2057 (2010).
7) : M-Core Project. http://www.arch.cs.titech.ac.jp/mcore/.
8) Takamaeda-Yamazaki, S., Sano, S., Sakaguchi, Y., Fujieda, N. and Kise, K.: ScalableCore System: A Scalable Many-core Simulator by Employing Over 100 FPGAs, *International Symposium on Applied Reconfigurable Computing (ARC 2012)* (2012).
9) 高前田伸也，佐藤真平，藤枝直輝，三好健文，吉瀬謙二：メニーコアアーキテクチャの HW 評価環境 ScalableCore システム，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.1 (2011).
10) Nassimi, D. and Sahni, S.: Bitonic Sort on a Mesh-Connected Parallel Computer, *IEEE Trans. Comput.*, Vol.28, pp.2–7 (1979).
11) Cannon, L.E.: A cellular computer to implement the kalman filter algorithm, PhD Thesis, Bozeman, MT, USA (1969). AAI7010025.