A-02

NoC ベース組込みメニーコアでのスケーラブルなデータ配置と並列化

Scalable Data Allocation and Parallel Computing on NoC-based Embedded Many Cores

丸山　雄也[1]　　加藤　真平[2]　　安積　卓也[1]

Yuya Maruyama[1]　Shinpei Kato[2]　Takuya Azumi[1]

1. はじめに

近年，自動車業界をはじめとするいくつかの領域では，高い処理要求と低電力消費を同時に実現するため，マルチ/メニーコアに向けたコンピューティングプラットフォームの進化が求められている [1] [2] [3]．例えば，自動運転システムは様々なアプリケーションを含んでおり，高性能コンピューティングにも分類される．自動運転システムの高い処理要求，予測可能性，エネルギー効率の要件を考慮すると，マルチ/メニーコアや GPU 等のヘテロジニアスなコンピューティングシステムが必要である．

しかし，マルチ/メニーコアプラットフォームを組込みシステムに適用するにはいくつかの問題が存在する [1] [4]．例えば，並列化されたプロセス同士で頻繁にアクセス競合が発生する問題や，共有資源が予測可能なタイミング動作やソフトウェア分析を妨げること等が挙げられる．これらの問題は，組込みシステム特有の厳格な要求仕様や多くのコアによって資源（例えば，メモリ及び I/O デバイス）を共有することに起因している．さらに，大容量メモリとすべてのコアを広帯域バスで接続してしまうことで，バス競合によってコア数の拡張性が失われ，大きな電力消費が必要なってしまう問題も存在する．このコア数の拡張や消費電力の問題に対応するために，Kalray によって開発された Multi-Purpose Processing Array (MPPA) -256 では，ネットワークオンチップ（NoC）を使用した NUMA（Non-Uniform Memory Access）を採用しており，256 のコアと低消費電力を実現している．

組込み要件を考慮すると，コアの数の拡張と妥当な電力効率のためには，マルチ/メニーコアには NUMA が必要と考えられる．NUMA によるスケーラブルなデータ割り当てにより，低消費電力で並列化された高性能及び汎用コンピューティングが実現可能である．MPPA-256 は，市販の組込みシステム向けマルチ/メニーコアコンポーネントの 1 つであり，NoC を用いた分散メモリアーキテクチャを採用している．しかし，NoC による分散メモリ間のデータ転送や並列化の可能性，及びメモリアクセス特性については，アプリケーション開発者には完全には明らかにされていない．本研究では，上記の項目の定量的評価と組込み向け NoC ベースメニーコアプラットフォームの実アプリケーションへの適用を取り上げる．

貢献：本研究では，MPPA-256 等の NoC に基づく組込みのメニーコアコンピューティングの検証に重点を置く．主に，行列計算によるマイクロベンチマーク評価や自動運転システムの中核をなす自己位置推定アルゴリズムを並列化することにより，下記の 2 つの貢献を導き，組込み向けメニーコアコンピューティングの実用性と今後の課題を明らかにする．

- NoC ベースのメニーコアプロセッサにおける並列化のスケーラビリティを，様々な状況での行列計算の評価と実際の複雑なアプリケーションの並列化により明らかにする
- メモリアクセス速度の特性が，データの割り当て先やメモリアクセスの主体によってどのように変化するかを定量的に明らかにする

構成：本論文は，以下のように構成される．まず，本論文で想定するシステムモデルについて 2 章で説明する．ここでは，Kalray MPPA-256 Bostan のハードウェアモデルとソフトウェアモデルを提示する．次に，3 章では，評価実験の構成やアプローチについて説明し，評価内容の考察を行う．続いて，4 章では，マルチ/メニーコアシステムに

[1] 大阪大学大学院基礎工学研究科
Graduate School of Engineering Science, Osaka University
[2] 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

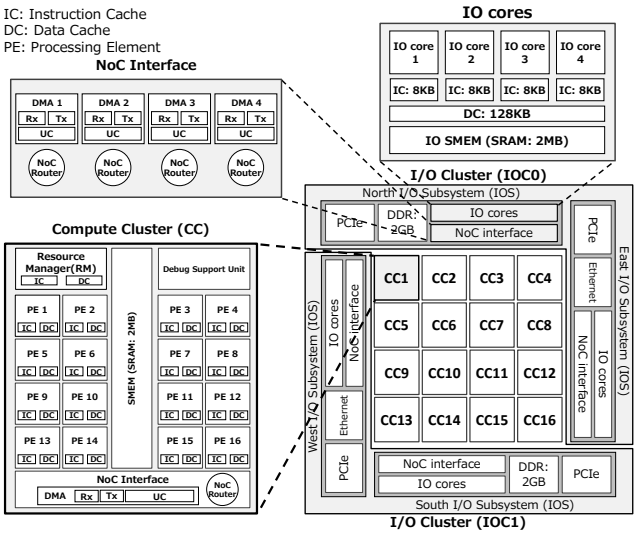

図 1　Kalray MPPA-256 Bostan のアーキテクチャの概要

焦点を当てた関連研究について議論する．最後に，5 章にて，まとめとして今後の課題と結論を示す．

2.　システムモデル

本章では，評価に使用される Kalray MPPA-256 Bostan のシステムモデルを示す．まず，2.1 節にてハードウェアモデルを紹介し，その後に 2.2 節のソフトウェアモデルの説明が続く．

2.1　ハードウェアモデル

MPPA-256 プロセッサは，16 個のコンピュートクラスタ（CC）及び 4 個の I/O サブシステム（IOS）で構成され，これらは全てトーラス状のネットワークオンチップ（NoC）によって接続されている（図 1 と図 2 参照）．

2.1.1　I/O サブシステム（IOS）

MPPA-256 には，北，南，東，西の 4 つの I/O サブシステム（IOS）がある．北と南の IOS は，DDR インタフェースと 8 レーンの PCIe コントローラに接続されている．東と西の IOS は，10GB/s イーサネットコントローラに接続されている．各 IOS は 4 つの I/O コアと NoC インタフェースで構成されており，図 1 のように 2 つの IOS によって 1 つの I/O クラスタ（IOC）が構成される．

I/O コア：I/O コアは，図 1 に示すように，合計容量が 2MB の共有メモリ（IO SMEM）に接続される．各 I/O コアは 32（8×4）KB の独自の命令キャッシュを持ち，128KB のデータキャッシュ及び外部の DDR メモリへのアクセスを 4 つの I/O コアで共有する．さらに，I/O コアは，PCIe，Ethernet，及びその他の I/O デバイスコントローラ等を管理している．

NoC インタフェース：NoC インタフェースは，図 1 のように 4 つの DMA エンジン（DMA1-4）と 4 つの NoC ルータを持つ．DMA エンジンは，NoC ルータを介したデータ通信を管理しており，受信（Rx）インタフェース，送信（Tx）インタフェース，及びマイクロコア（UC）の 3 つの NoC インタフェースを備えている．UC は，マルチスレッドエンジンで，Tx インタフェースでデータを送信するスレッドを設定するようにプログラムできる．設定されたパターンを使用してメモリからデータを抽出し，NoC でデータの送信を開始し，動作後は他のプロセッシングエンジン（PE）及び I/O コアを利用することなく自律的に実行することが可能である．

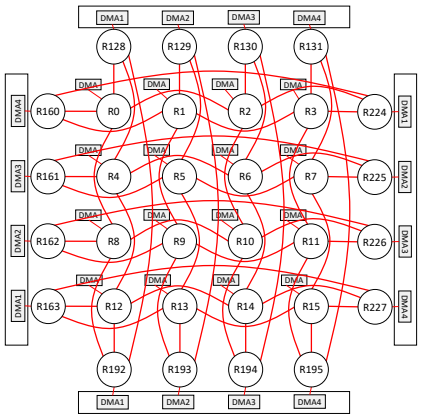

図 2　NoC（Network-on-Chip）マップ

2.1.2　コンピュートクラスタ（CC）

MPPA-256 では，16 個のコアを持つコンピュートクラスタ（CC）が 16 個存在し，合計 256 のコアによって主要な処理が行われる．NoC 上の 16 個の内部ノードが CC に対応しており，図 1 は各 CC のアーキテクチャを示している．

プロセッシングエンジンとリソースマネージャ：CC では，16 個のプロセッシングエンジン（PE）と 1 個のリソースマネージャ（RM）が 2 MB のクラスタローカルメモリ（SMEM）を共有する．PE は主に，並列処理のためにユーザーによって使用される．CC 内の PE と RM は，Kalray 独自の 32 ビット VLIW の Kalray-1 コア (600MHz または 800MHz) となっている．

デバッグサポートユニットと NoC インタフェース：SMEM のバスマスタは，PE と RM の他に，デバッグサポートユニットと NoC インタフェース内の DMA エンジンとなる．DMA エンジンと NoC ルータは，NoC インタフェース内に配置されており，CC の DMA エンジンにも，Rx，Tx，及び UC の 3 つのインタフェースが存在している．

2.1.3　ネットワークオンチップ（NoC）

16 個の CC と 4 つの IOS は，図 2 のようにネットワークによって接続されている．プロセッサ上にバスネットワークとして NoC が構築されており，各ノードにはルータが存在する．

バスネットワーク：バスネットワークはノード（CC と IOS）をトーラス状に接続している [5]．トーラス状ネットワークのメリットはメッシュ状 [6] [7] の場合と比べて平均

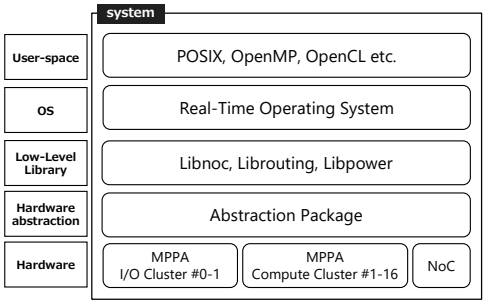

図 3　Kalray MPPA-256 のシステム構成

ホップ数が少なくなることにある．実際には，ネットワークは双方向リンク（図 2 に赤線で示される）を持つ 2 つの並列バスで構成されている．大きな容量のデータ転送に最適化された Data-NoC（D-NoC）と低レイテンシで小さなメッセージ用に最適化された Control-NoC（C-NoC）である．NoC 上のデータは，フリットと呼ばれる単位に分割される方式で転送され，ルータ間を循環するデータは可変長フリットによってパッケージ化されている．

NoC ルータ：CC ごとの 1 つノードと IOS ごとの 4 つのノードには，D-NoC ルータと C-NoC ルータがある．さらに，CC/IOS 上の NoC インタフェースの DMA エンジンは，Rx インタフェース，Tx インタフェース，及び UC を備えた D-NoC ルータを介して送受信を行う．図 1 に示されている NoC ルータは，図 2 のノードと対応する．D-NoC と C-NoC の両方で，各ネットワークノード（CC または IOS）には次の 5 つの方向へのリンクを持っている．東西南北の 4 つの隣接リンク及び，及び NoC ルータに接続されたローカルアドレススペースへのリンクの計 5 つである．NoC ルータは，各方向でフリットを FIFO キューイングし，リンク幅は各方向 4 バイト幅であり，コアが 600MHz または 800MHz の CPU クロックレートで動作するため，各ノードは合計 2.4GB/秒または 3.2GB/秒で送受信することとなる．

2.2　ソフトウェアモデル

図 3 は，本研究で扱う Kalray MPPA-256 に使用されているソフトウェアスタックを示している．Linux，リアルタイム OS（Real-Time OS：RTOS），POSIX API，OpenCL，及び OpenMP 等のいくつかのプログラミングモデルまたはランタイムをマッピングすることが可能である．

ハードウェア抽象化レイヤでは，抽象化パッケージが CC，IOS，及び NoC のハードウェアを抽象化する．抽象化は，ハードウェアリソースを分割し，ユーザー空間の OS ライブラリからのリソースへのアクセスを制御している．ハードウェア抽象化は主に RM コアで実行される．

低レベルのライブラリ層では，Kalray の提供するシステムが NoC を処理するためのライブラリを提供している．さらに，NoC のルーティングや QoS 等の機能は，プログラマによって設定可能となっている．Libnoc は，メモリマップレジスタへの直接アクセスを可能にし，最低限のオーバヘッドで設計されてる．Librouting は，MPPA の任意のクラスタ間でのデータ通信をルーティングするための最小限の機能を提供する．このライブラリではトーラスネットワーク上のルーティングは独自のポリシーで静的に実行されており，必ずしも最小のホップ数を保証するものではない．Libpower を使用すると，リモートクラスタの起動や実行終了を待機することができる．

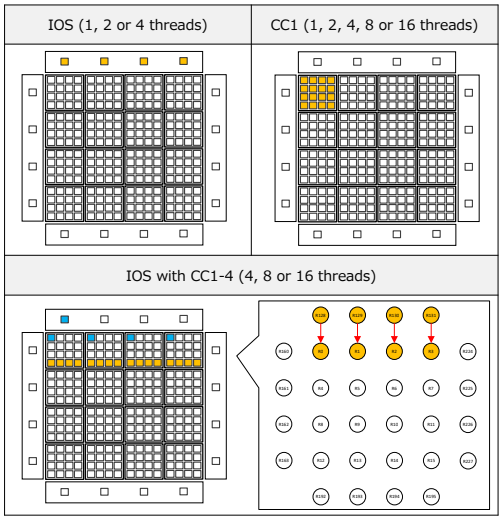

図 4　行列計算の評価構成

さらに，OS 層の抽象化パッケージは様々な OS をサポートしている．ここでは，その中から RTOS である eMCOS を紹介する．eMCOS は日本の RTOS のサプライヤである eSOL によって開発されたリアルタイム組込み OS であり，eMCOS は世界で初めて市販された組込みシステム向けメニーコア RTOS である．CC と IOS の両方で動作し，最小限のプログラミングインタフェースとライブラリを提供する．eMCOS は分散マイクロカーネルアーキテクチャを実装しており，これにより，アプリケーションは優先順位ベースのメッセージパッシング，ローカルスレッドスケジューリング，スレッド管理を IOS 及び CC で操作することが可能になっている．

3.　評価と考察

本章ではまず，MPPA-256 の並列化可能性とメモリアクセス特性を示すための行列計算の評価を行う．その後，NUMA のメニーコアの実用性を検証するため，実用的な自動運転アプリケーションの並列化を行い，最後に全体の考察を行う．以下のすべての評価実験は，eMCOS を使用した実際のハードウェアボード上で行われるものとする．

3.1　行列計算

3.1.1　評価構成と仮定

本評価では，MPPA-256 の行列計算時間を計測する．行

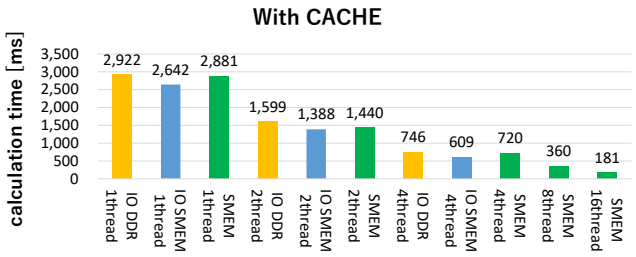

図 5　行列計算：IOS と CC（キャッシュ有効）

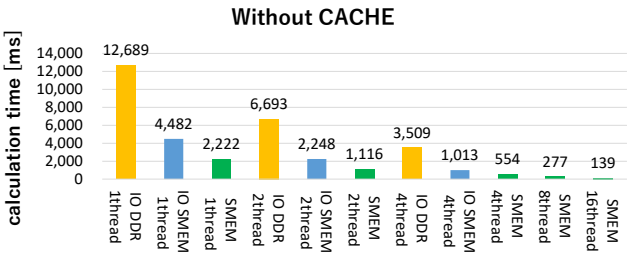

図 6　行列計算：IOS と CC（キャッシュ無効）

列計算は IOS と CC で行われ，図 4 に示すように，3 つの計算ケースが想定する．第一のケースでは，IOS 内で 4 つのコアを用いて計算される．ここでは，メモリアクセス特性を分析するために，IO DDR 及び SMEM に行列バッファを割り当てる．第二のケースでは，1 つの CC を用いて，1〜16 コアで計算を行う．

そして，第三のケースでは，IOS と 4 つの CC を使用してオフロードコンピューティングを行う．ここでは，IOS と CC のいくつかのコアは，行列計算とは別に，並列化された処理の管理を担っている．このケースでは，4 つの CC を用いるため，SMEM による 2MB の制約がなく，分散メモリを用いて 2MB を超えるデータを処理することが可能である．また，IOS 及び CC のコアは，キャッシュのコヒーレンシトラブルを回避するために，キャッシュを無効にした状態で行列バッファにアクセスする必要がある．NoC でのデータ通信については，図 4 に示すように行列バッファを 4 つの DMA で分担して並列送信を行った．

各ケースでの行列計算時間は，並列化の度合いと割り当てられたメモリの種類，キャッシュの ON/OFF の 3 つの観点で分類される．以下の評価では，OS 等のシステムアプリケーションのメモリ領域も考慮し，利用可能な合計バッファサイズは 1MB とした．したがって 314KB の行列を 3 つ準備し，行列 A と行列 B が乗算され，結果が行列 C に格納されるものとする．また，第三のケースに関しては，メモリの制約が一部解除されるため，640KB の行列による計算も行った．

3.1.2　キャッシュとメモリの種類の影響

IOS と CC でキャッシュを有効した場合の行列計算の結果を図 5 に示す．IO DDR，IO SMEM，及び CC SMEM は，キャッシュによってほとんど違いがないことが分かる．これは，IOS の 128 KB のデータキャッシュが機能し，DDR アクセスの遅延を補っていると考えられる．さらに，計算時間はスレッドの数とほぼ線形関係にあることが観察され，並列化に関する理想的な動作に相当する．

次に，IOS と CC のキャッシュを無効にした場合の行列計算の結果を図 6 に示す．キャッシュを無効にしたことで，DDR での計算時間が約 4 倍に増加し，SMEM と比較して大きな差が生じている．他の特筆すべき結果は，CC SMEM の計算速度が IO SMEM の計算速度を超えていることである．この特性は，キャッシュ有効時の結果（図 5）では隠れている特性である．計算を行っているコア自体は IOS と CC では物理的には同じなので，SMEM の特性や物理的配置が大きな影響を及ぼしていると考えられる．CC SMEM と IO SMEM に無視できない大きな差が存在することは，興味深い結果である．さらに，計算時間はスレッド数と線形関係にあることも観察され，CC SMEM におけるキャッシュ無効時の計算速度は，キャッシュ有効時の計算速度を上回っている．キャッシュ有効時にアクセス速度が遅くなることは直感に反するものであり，可能性としてはキャッシュラインの問題が考えられる．CC の PE 内の小型データキャッシュ（8KB）が十分に機能せず，アプリケーションが頻繁にキャッシュのミスヒットを起こした場合，メモリアクセスはキャッシュされていないデータアクセスの時間とキャッシュラインを補充するためのコストを同時に支払うことになる．その結果，キャッシュ無効時のメモリアクセス速度がキャッシュ有効時のメモリアクセス速度を超えていると考えられる．

3.1.3　4 つの CC による並列化

最後に，IOS と CC でのオフロードコンピューティングによる行列計算の結果を図 7 に示す．この場合，行列バッファの合計容量が 1MB を超えると仮定し，計算速度の結果は IO DDR（キャッシュ有効時）と比較することになる．並列化に伴い，IOS では，行列バッファの CC へ転送処理や送信バッファに詰めるための行列の転置計算（inverse matB），各 CC での計算結果を受け取った後に結果を行列バッファに集約するための処理（input matC）等が余分に必要になる．これらは，図 7 と図 8 に示されるように，一定のオーバヘッドを生んでいる．しかし，オーバヘッドを考慮しても，オフロードによる計算速度は，IO DDR（キャッシュ有効時）の速度を上回っている．この結果は，いくつかの重要な事実を示している．第一に，D-NoC データ転送は計算時間に比べてオーバヘッドをほとんど発生させていない．第二に，DDR への DMA メモリアクセスの速度は，I/O コアのメモリアクセスの速度を超えている．オフロードの場合，DMA は DDR の行列バッファにアクセスし，IO DDR から各 CC SMEM にデータを転送する．その後，CC 内の PE は，キ ャッシュを無効にした状態で行列バッファにアクセスすることになる．MPPA-256 の場

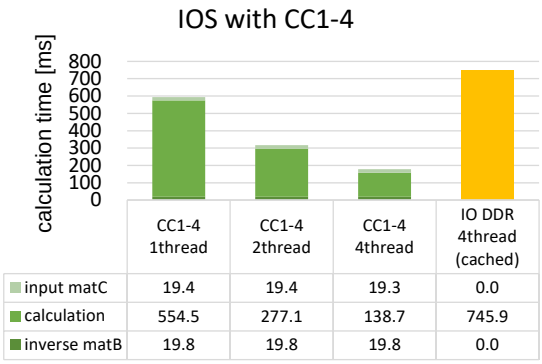

図 7 行列計算：IOS + 4 CCs（314KB 行列 × 3）

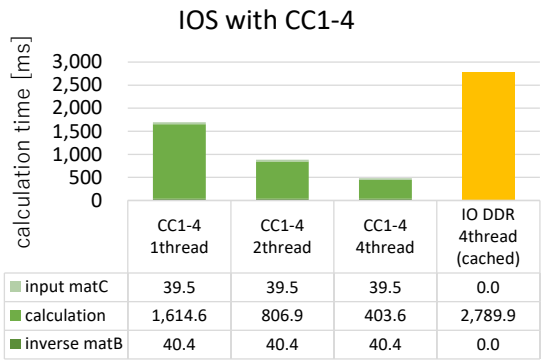

図 8 行列計算：IOS + 4 CCs（640KB 行列 × 3）

合，NoC のデータ通信と DMA によるメモリアクセスのオーバヘッドが少なく，並列化された NoC データ通信と複数の CC による分散メモリが実用的であるといえる．計算結果が格納されている行列 C を DDR に配置した場合，DDR のメモリアクセス遅延により NoC ルータの FIFO がオーバフローし，エラーが発生することが観測された．図 7, 図 8 の評価結果は，行列 C が DDR で割り当てられている場合の結果であるが，このエラーは伝送プロトコルによって防止されるべきであり，NoC 上でフリットのフロー制御は MPPA-256 の今後の課題である．現在，このエラーを確実に回避するには，行列 C を IO SMEM に割り当てる必要がある．

3.2 実アプリケーション

本節では，自動運転システムの一部のアルゴリズムを採用し，組込み向けメニーコアプラットフォームの実用性を検証する．本研究では，市街地自動運転のためのオープンソースソフトウェアである Autoware [8] の C++で記述された車両の自己位置推定アルゴリズムの並列化を行った．アルゴリズムは，主に Point Cloud Library [9] に実装されているいる正規分布変換マッチングアルゴリズム [10] を採用している．

自己位置推定アルゴリズムのボトルネック部分は主に *computeTransform* 関数であり，レーザセンサからのスキャンクエリごとに最大 8 個の近傍点を地図データから探索し，ニュートン法により位置推定結果が収束するまで処理を繰り返す関数になっている．この評価実験では，*computeTransform* の一部を 16 個の CC で並列化し，残りの部分を IOS 上の 4 つのコアで並列化した．探索対象となる地図データが 1MB を超えるため，*computeTransform* 関数全体を CC で並列化するには，最近傍点探索のアルゴリズム自体を大きく再設計する必要がある．このアルゴリズムの再設計は NUMA 採用によるデメリットといえ，今後の課題のひとつである．

並列化された自己位置推定アルゴリズムの評価を，図 9 に示す．推定結果収束までの平均実行時間を計測し，並列

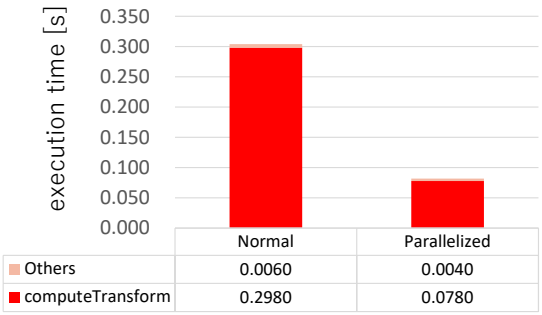

図 9 MPPA-256 による自動運転システムの自己位置推定処理の並列化

化によって *computeTransform* 関数が高速化されているを示している．現状では多くの自動運転システムにおいて，レーザセンサからスキャンクエリは 10Hz となることが多く，今回の並列化によりデッドラインとなる 100 ミリ秒以下を達成したことになる．この並列化されたアルゴリズムは，シミュレーションと実車実験の両方で動作実験が行われ，テストコース内で正常に動作することが確認されている．

3.3 考察

本章では，NoC ベースの組込みメニーコアのデータ転送と並列コンピューティングの特性を明らかにしてきた．これらの実験結果から，MPPA-256 を始めとする NoC ベースのメニーコアプラットフォーム使用者及び開発者のためのガイドラインを得ることができる．

行列計算のマイクロベンチマークからは，SMEM の特徴，キャッシュの影響，D-NoC のデータフロー制御の 3 つのメモリアクセスに関する考察を得ることができる．まず，CC SMEM のメモリアクセス速度は，図 6 のように IO SMEM のメモリアクセス速度を超えていることである．ここには無視できない差異が存在し，CC で積極的に処理をするための大きな根拠となる．次に，図 6 の CC SMEM でのキャッシュ無効時の計算速度が，図 5 のキャッシュ有効時のものを上回っていることを定量的に示したことが挙げられる．キャッシュのミスヒットが頻繁に発生した場合，オーバヘッドが存在することは一般的に知られているが，

キャッシュラインの影響によって無視できない大きな差異が発生することは特筆すべき項目といえる．最後に，データが複数の CC から IO DDR に並列転送される場合，DDR へのメモリアクセス遅延のために IOS 内の NoC ルータの FIFO がオーバフローする問題である．これは，今後，伝送プロトコルとフロー制御によって防止する必要がある．

3.2 章の実アプリケーションによる評価では，自動運転システムの自己位置推定アルゴリズムを並列化し，実用化可能性を示した．CC SMEM をスクラッチパッドメモリとして使用した NoC による IOS から CC へのデータ転送や IOS/CC での並列コンピューティングは実際の複雑なアプリケーションにも応用可能であるといえる．

4. 関連研究

本章では，MPPA-256 と他のマルチ/メニーコアプラットフォームを比較し，メニーコアに関連する既存研究の紹介を行う．初めに，本研究で利用した Kalray MPPA-256 を他のマルチ/メニーコアコンポーネントと比較して MPPA-256 の特徴を整理し，続いて本研究と既存研究との比較を行う．

現在，多くのマルチ/メニーコアコンポーネントが開発され，様々なベンダーによってリリースされている．例えば，Kalray の Multi-Purpose Processing Array（MPPA）-256 [11] や Tilera の Tile-Gx [12] [13]，Tile64 [14]，Intel の Xeon Phi [15] [16]，Single-chip Cloud Computer（SCC）[17]）等が挙げられる．本研究では，リアルタイム組込みアプリケーション向けに設計された Kalray MPPA-256 に焦点を当てた．MPPA-256 は，エネルギー効率の高い 256 個の汎用コアを内包しており，NoC ベースのクラスタ構造を持っていることが大きな特徴である．

MPPA-256 は，コア数の拡張性と電力効率の面で，他の COTS マルチ/メニーコアコンポーネントよりも優れている（表 1 参照）．他のプラットフォームとコア数を比較すると，MPPA-256 は 256 コアであるのに対し，他のマルチ/メニーコアコンポーネントは多いものでも 64 コアとなっている．この優位性は，16 のクラスタがそれぞれ独自のローカル共有メモリを持っている NUMA メモリアーキテクチャに起因している．MPPA-256 を除く多くのプラットフォームでは，すべてのコアがグローバルな DDR メモリを共有しており，特定のバスルートに大きな負荷がかかる．このため，メモリへのアクセス競合が頻繁に発生し，コア数増加の妨げとなっている．MPPA-256 の NUMA メモリアーキテクチャは上記の問題を軽減し，コア数の拡張性を向上させている．

一方で，NUMA メモリアーキテクチャはコア数の拡張は容易だが，利用可能なメモリ容量を制限し，NoC での DDR からのデータ転送を必要とすることになる．特に，多くのメモリを必要とするアプリケーションの場合，既存コードの移植が困難になることが多い．そのため，表 1 に示されているように，他の COTS プラットフォームに比べてコードの移植性の観点では劣っている．

表 1　マルチ/メニーコアプラットフォームの比較

	コア数の拡張性	電力効率	コードの移植性
Kalray MPPA-256 [11]	○	○	△
Tilera Tile series [14]		△	○
Intel Xeon Phi [15] [16]			○
Intel SCC [17]			○

電力効率の面では，MPPA-256 はコアの数が多いにもかかわらず，優れたエネルギー効率を実現する [18]．MPPA-256 の 1 ワットあたりの合計クロック数は，現在のマルチ/メニーコアコンポーネントの中で最も高く，消費電力は，600MHz の場合で 16W，800MHz で 24W であり，組込み向けの側面が強くなっている．MPPA-256 は自動車や航空機といった組込み向けメニーコアプラットフォームとして広く受け入れられており，これまでも多くの既存研究で利用されてきている [19] [3] [20]．

既存研究では，MPPA-256 等のメニーコアプラットフォームでのリアルタイムアプリケーションの研究がいくつか行われてきた．参考文献 [4] では，マルチ/メニーコアプラットフォームの可能性と今後の課題について述べており，リアルタイム/組込みシステムにおけるマルチ/メニーコアへの移行についても議論している．参考文献 [19] [2] [3] では，マルチ/メニーコアシステムのためのタスクマッピングアルゴリズムが提案されており，Airbus は，MPPA-256 を使用したハードリアルタイムアプリケーションのための有向非循環グラフ（DAG）スケジューリングの方法を提案している [3]．参考文献 [2] は，AUTOSAR（車載用組込みソフトウェアシステムを開発するための標準アーキテクチャ [21]）に基づいたマッピングフレームワークが提案しており，共有リソースの競合を考慮した AUTOSAR タスクスケジューリングについては，参考文献 [1] の中で紹介されている．

MPPA-256 を対象にした既存研究は主に参考文献 [22] [23] [18] [20] 等が挙げられる（表 2）．MPPA-256 の性能やエネルギー効率の高さは，参考文献 [18] の中で紹介されている．しかし，このレポートには評価がほとんど含まれておらず，NoC によるデータ転送やメモリアクセス特性については触れられていない．MPPA-256 の NoC によるデータ転送や NoC の保証するサービスについては参考文献 [22], [23] の中で分析されている．これらの研究では，理論的な分析は十分に行われているが，評価実験等は少なく，データの並列送信は取り上げられていない．参考文献 [20] は，予測可能なメモリアクセスに焦点を当てている．研究では，MPPA-256 等のメニープラットフォームでのパフォーマンスと予測可能性の主なボトルネックとして，外部 DDR との NoC 通信が挙げている．この分析では，外部 DDR へのメモリアクセス特性が取り上げられい

表 2　関連研究との比較

	性能評価	NoC 通信	メモリアクセス	実アプリケーション	並列送信
Kalray clusters calculate quickly [18]	△				
Network-on-Chip Service Guarantees [23]		○			
Predictable composition of memory accesses [20]		○	○		
本研究	○	○	○	○	○

くつかの考察を得ているが，ボトルネックに対する解決策は検討されておらず，実用的な評価は欠けている．

5. おわりに

本研究では，NoC を用いた行列計算によるマイクロベンチマーク，NUMA のメニーコアプラットフォームによる実アプリケーションの並列化を行った．評価結果は，メモリごとのデータ割り当ての影響やメモリアクセスに与えるキャッシュの影響，並列化の拡張性，実用化可能性等を示している．これらの実験結果は，システム設計者が適切な設計を行う手助けとなり，実アプリケーションの並列化は NoC ベースの組込み向け NUMA メニーコアプラットフォームの実用性を実証している．

今後の研究では，メニーコア上で参考文献 [24] 等のリアルタイムシステムソフトウェアを設計することや 3.2 節で挙げた最近傍探索等の大容量データを扱うアルゴリズムの並列化が挙げられる．

謝辞

本研究の一部はトヨタ自動車とイーソルの支援により実施された．

参考文献

[1] Becker, M., Dasari, D., Nicolic, B., Akesson, B., Nolte, T. et al.: Contention-Free Execution of Automotive Applications on a Clustered Many-Core Platform, *in Proc. of 28th IEEE ECRTS*, pp. 14–24 (2016).

[2] Faragardi, H. R., Lisper, B., Sandström, K. and Nolte, T.: A communication-aware solution framework for mapping AUTOSAR runnables on multi-core systems, *in Proc. of IEEE ETFA*, pp. 1–9 (2014).

[3] Perret, Q., Maurère, P., Noulard, É., Pagetti, C., Sainrat, P. and Triquet, B.: Mapping hard real-time applications on many-core processors, *in Proc. of the ACM 24th RTNS*, pp. 235–244 (2016).

[4] Saidi, S., Ernst, R., Uhrig, S., Theiling, H. and de Dinechin, B. D.: The shift to multicores in real-time and safety-critical systems, *in Proc. of IEEE CODES+ISSS*, pp. 220–229 (2015).

[5] Dally, W. J. and Towles, B.: Route packets, not wires: on-chip interconnection networks, *in Proc. of IEEE DAC*, pp. 684–689 (2001).

[6] Vangal, S., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J., Finan, D., Iyer, P., Singh, A., Jacob, T. et al.: An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS, *in Proc. of IEEE ISSCC*, pp. 98–99 (2007).

[7] Taylor, M. B., Kim, J., Miller, J., Wentzlaff, D., Ghodrat, F., Greenwald, B., Hoffman, H., Johnson, P., Lee, J.-W., Lee, W. et al.: The Raw microprocessor: A computational fabric for software circuits and general-purpose programs, *IEEE Micro*, Vol. 22, No. 2, pp. 25–35 (2002).

[8] Kato, S., Takeuchi, E., Ishiguro, Y., Ninomiya, Y., Takeda, K. and Hamada, T.: An Open Approach to Autonomous Vehicles, *IEEE Micro*, Vol. 35, No. 6, pp. 60–68 (2015).

[9] : Point Cloud Library (PCL), http://pointclouds.org/.

[10] Magnusson, M.: The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection, PhD Thesis, Örebro universitet (2009).

[11] de Dinechin, B. D., Van Amstel, D., Poulhiès, M. and Lager, G.: Time-critical computing on a single-chip massively parallel processor, *in Proc. of IEEE DATE*, pp. 1–6 (2014).

[12] Ramey, C.: TILE-Gx100 manycore processor: Acceleration interfaces and architecture, *in Proc. of the 23th IEEE HCS*, IEEE, pp. 1–21 (2011).

[13] Schooler, R.: Tile processors: Many-core for embedded and cloud computing, *in Proc. of HPEC* (2010).

[14] Bell, S., Edwards, B., Amann, J., Conlin, R., Joyce, K., Leung, V., MacKay, J., Reif, M., Bao, L., Brown, J. et al.: Tile64-processor: A 64-core soc with mesh interconnect, *in Proc. of IEEE ISSCC*, IEEE, pp. 88–598 (2008).

[15] Chrysos, G.: Intel® Xeon Phi Coprocessor-the Architecture, *Intel Whitepaper* (2014).

[16] Chrysos, G. and Engineer, S. P.: Intel Xeon Phi coprocessor (codename knights corner), *in Proc. of the 24th Hot Chips Symposium* (2012).

[17] Baron, M.: The single-chip cloud computer, *Microprocessor Report*, Vol. 24, p. 4 (2010).

[18] Kanter, D. and Gwennap, L.: Kalray clusters calculate quickly, *Microprocessor Report* (2015).

[19] Carle, T., Djemal, M., Potop-Butucaru, D., De Simone, R. and Zhang, Z.: Static mapping of real-time applications onto massively parallel processor arrays, *in Proc. of the 14th IEEE ACSD*, pp. 112–121 (2014).

[20] Perret, Q., Maurère, P., Noulard, E., Pagetti, C., Sainrat, P. and Triquet, B.: Predictable composition of memory accesses on many-core processors, *in Proc. of the 8th ECRTS* (2016).

[21] Fürst, S., Mössinger, J., Bunzel, S., Weber, T., Kirschke-Biller, F., Heitkämper, P., Kinkelin, G., Nishikawa, K. and Lange, K.: AUTOSAR–A Worldwide Standard is on the Road, *in Proc. of the 14th ELIV*, Vol. 62 (2009).

[22] de Dinechin, B. D., Durand, Y., van Amstel, D. and Ghiti, A.: Guaranteed Services of the NoC of a Many-core Processor, *in Proc. of ACM NoCArc*, pp. 11–16 (2014).

[23] de Dinechin, B. D. and Graillat, A.: Network-on-Chip Service Guarantees on the Kalray MPPA-256 Bostan Processor, *in Proc. of the 2nd AISTECS* (2017).

[24] Maruyama, Y., Kato, S. and Azumi, T.: Exploring the Performance of ROS2, *in Proc. of the 13th ACM EMSOFT*, pp. 5:1–5:10 (2016).