Vol.2010-ARC-188 No.8
2010/3/1

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 M-Core

植原 昂[†1] 佐藤真平[†1]
佐野伸太郎[†2] 吉瀬謙二[†1]

マルチコアおよびメニーコアプロセッサを対象としたソフトウェアおよびプロセッサアーキテクチャの教育・研究が今後ますます重要になることを想定し，我々は M-Core 基盤環境を開発している．M-Core 基盤環境はマルチコアおよびメニーコアアーキテクチャの研究者，並列プログラミングを学ぶ学生，マルチコアおよびメニーコア向けのシステムソフトウェアを開発する研究者を主なターゲットとする実用的な基盤環境である．本稿では M-Core 基盤環境 V1.0 について述べ，これを利用した事例を示し，マルチコアおよびメニーコアプロセッサの教育・研究に対する実用性を示す．

M-Core: A Practical Infrastructure for Researches and Education of Many-Core Processors

Many-core processors which have hundreds or thousands of cores on a chip will be realized. In order to help researches and education of such many-core processors, we developed an M-Core infrastructure V1.0. This paper describes two main elements provided by our infrastructure. One element is the simple many-core processor architecture. The other is software system including APIs, a processor simulator, benchmark programs, and so on. This paper also presents some case studies using our infrastructure.

†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
†2 東京工業大学工学部情報工学科
Department of Computer Science, Tokyo Institute of Technology

1. まえがき

広い分野にわたり，複数のコアを搭載するマルチコアプロセッサ（マルチコア）が主流となりつつある．半導体の集積度の向上により，搭載されるコア数は着実に増加し続けており，プロセッサアーキテクチャはさらに多くのコアを集積するメニーコアプロセッサ（メニーコア）[⋆1] へと向かっている．このため，マルチコアおよびメニーコアを対象とした研究・教育が今後ますます重要になると考えられる．

我々は，(a) メニーコアアーキテクチャ，(b) 並列プログラミング，(c) マルチコアおよびメニーコアを対象とするシステムソフトウェア，の研究・教育を支援する実用的な M-Core 基盤環境を開発している．

図 1 に，M-Core 基盤環境の構成を示す．M-Core 基盤環境は，大別して以下に述べる 2 つの要素からなる．1 つ目は，図 1 左に示す M-Core アーキテクチャ(M-Core architecture)[1)] である．これは，我々が新規に定義したメニーコアアーキテクチャである．2 つ目は，図 1 右に示す M-Core ソフトウェアシステム (M-Core software system) である．これは，並列プログラムの開発・評価環境である．

M-Core アーキテクチャは，主に，構成がシンプルで理解し易いこと，アーキテクチャを拡張することで様々な研究に応用できる実用性を備えることを考慮して設計されている．M-Core ソフトウェアシステムは次の 3 つの要素を提供する．(1) プログラム開発を支援する API (Application Programming Interface)．(2) ソフトウェアおよびハードウェアを評価するメニーコアプロセッサのシミュレータ SimMc．(3) 並列プログラミングの研究・教育を支援する豊富なアプリケーションプログラム，およびソフトウェア・ハードウェアの検証を効率的におこなう検証システム．

API として，図 1 の (1-A) MClib および (1-B) MPIMC の 2 種類を提供する．MClib は M-Core アーキテクチャ上で動作する並列プログラムを開発する上で，基本となる関数群である．また，MPIMC は，MPI (Message Passing Interface standard) のサブセットである．これは，MClib の上位層に位置する API である．これにより，MPI を利用して実装されたソフトウェアが M-Core 上で動作する．

また，アプリケーションプログラムとして，図 1 の (3-A) マイクロベンチマークプログラムと，(3-B) Nas Parallel Benchmarks（以下，NPB と記述する）を提供する．前者は，我々が記述したサンプルプログラムや小規模なベンチマークプログラムから成り，並列プロ

⋆1 1 チップに 2 個以上のコアを搭載するプロセッサをマルチコアプロセッサと呼ぶことにする．その中で，特に，1 チップに数十個以上のコアを搭載するプロセッサをメニーコアプロセッサと呼ぶことにする．

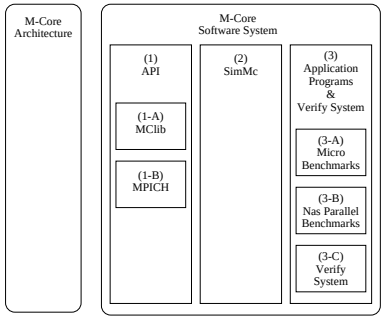

図 1 M-Core 基盤環境の構成.

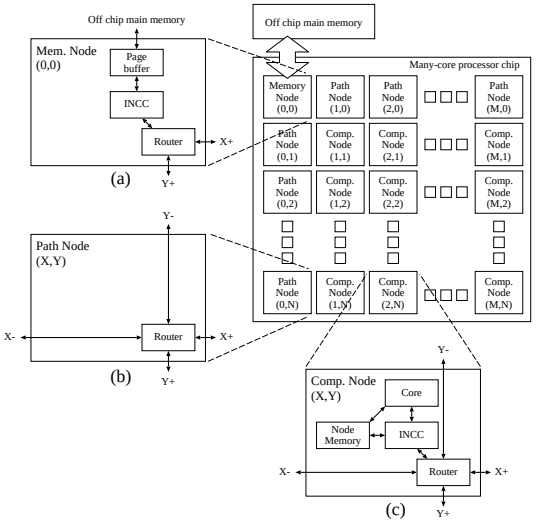

図 2 M-Core アーキテクチャ.

グラミングの教育や，簡単な性能評価に用いることができる．後者は，より現実的なベンチマークプログラムとして，主にアーキテクチャの性能評価に用いることができる．

2. M-Core アーキテクチャ

メニーコアに関する教育・研究を効率的に進めるため，実用的で理解しやすいメニーコアアーキテクチャが必要とされる．ここで，メニーコアアーキテクチャとは，プログラマが理解する必要があるメモリやコアの構成やその接続方式などを含むメニーコアプロセッサの構成方式のことである．このようなアーキテクチャには，以下に述べるコンセプトが求められる．(1) シンプルで理解しやすく，拡張することで様々な研究で利用できる実用性がある．(2) 現実のハードウェアとして実装することを考慮する．(3)RISC プロセッサ，コンパイラ，ライブラリなど，既存の資産を有効に活用できる．(4) 小規模で均一のコアを多数並べる．(5) 高い並列化効率を目指すため，コア間のデータ通信オーバヘッドを削減するシンプルで効率的な通信方式を採用する．(6) 1 チップに搭載されるコア数が数千や数万のオーダまで増えることを想定し，スケーラビリティを確保する．

我々の調べた限り，これらのコンセプトを全て満たすアーキテクチャは存在しない．このため，我々はこれらのコンセプトを満たすものとして，M-Core アーキテクチャを新規に開発した．なお，様々なアーキテクチャの調査結果は，7 章で述べる．

本章では，まず M-Core のアーキテクチャを示す．続いて，M-Core アーキテクチャを実現する具体的な設計の 1 つとして M-Coreα マイクロアーキテクチャを定義する．

2.1 アーキテクチャ

図 2 右上に示すように多数のノードをタイル上に配置するタイルアーキテクチャを採用する．各ノードはチップ内の 2 次元メッシュネットワークで接続される．2 次元メッシュネットワークは，実装が比較的簡単で理解しやすい，容易にネットワークサイズを変更できる，チップ上に素直に配置配線できる，といった利点がある．このため，様々なメニーコアプロセッサの研究開発において採用されている．

各ノードにはチップ内で固有のノード ID を割り当てる．ノード ID は，ノード間でデータを通信する際に，ノードを指定するために使用する．ノード ID はチップ上の X 座標と Y 座標の組合せで表現する．本稿では，ノード ID が (X,Y) となるノードを Node (X,Y) と表記する．

ノードには，計算ノード，メモリノード，パスノードの 3 種類がある．計算ノードは，コアを格納し，アプリケーションプログラムの実行およびパケットの転送をおこなう．メモリノードはオフチップのメインメモリに接続し，計算ノードとメインメモリのインターフェイスの役割を果たす．パスノードは主にパケットの転送をおこなう．図 2 において，計算ノードが Node (1,1) から Node (M,N)，メモリノードが Node (0,0)，パスノードが Node (1,0) から Node (M,0) および Node (0,1) から Node (0,N) である．

2.1.1 ノードアーキテクチャ

図 2 (a) に計算ノードのブロック図を示す．計算ノードは，コア，ノードメモリ，INCC (Inter Node Communication Controller)，ルータで構成される．コアは軽量化したプロセッシングエレメントである．ノードメモリは小規模なローカルメモリである．ノードメモリには，プログラムの実行に必要な命令およびデータを格納する．INCC は，ノード間の通信を扱うコントローラである．また，ルータは，隣接する 4 つのノードのルータおよび自身のノードの INCC の 5 方向と接続し，パケットを転送する．ノードメモリ，INCC，ルータは，コアの命令セットとは独立している．このため，様々なコアを利用することができる．

図 2 (b) にメモリノードのブロック図を示す．メモリノードは INCC とページバッファ (page buffer) を格納し，メインメモリと接続する．ページバッファはメモリアクセスレイテンシを隠蔽するため，直近で参照したメインメモリの 1 ページをバッファリングする．

図 2 (c) に，パスノードのブロック図を示す．パスノードは主にメモリノードと計算ノー

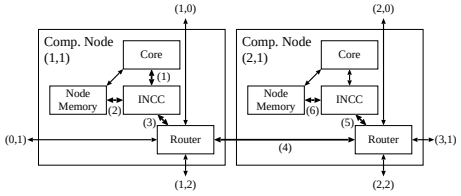

図 3 DMA_PUT の実行手順.

ドの通信のために利用されるため，ルータのみで構成される．

2.1.2 通信アーキテクチャ

コアの接続方法について，少数のコアの場合はバス接続を採用することがあるが，これはコア数のスケールに対応しない．このため，我々はネットワークで接続する方式 (Network on Chip, NoC) を採用する．そして，ノード間の通信はパケット転送方式を採用する．パケットの紛失を防ぐため，ルータでフロー制御をおこなう．フロー制御とは，ネットワークが混雑する場合に転送を中断し，混雑が解消してから転送を再開する手段である．送信元および送信先が同じパケットは，送信したパケットから順に宛先ノードのメモリに書き込まれることを保証する．これにより，アプリケーションプログラムの記述が容易になる．

パケットの送受信には DMA (Direct Memory Access) を利用する．これを用いて，自身のノードメモリにあるデータを他の計算ノードのノードメモリまたはメモリノードのメインメモリに複製する．本稿では，この DMA を DMA_PUT と呼ぶ．

DMA_PUT の起動は，計算ノードのコアが INCC に対して，DMA_PUT の実行に必要となる以下のパラメタを与えることで実現する．(1) 宛先ノード ID．(2) 読み出しメモリアドレス．(3) 書き込みメモリアドレス．(4) 読み出し時のストライド値．(5) 書き込み時のストライド値．(6) データサイズ．宛先ノード ID として，自身のノード ID を指定することも可能である．

図 3 に，DMA_PUT の処理の流れ (1) から (6) を示す．これは計算ノードである Node (1,1) が Node (2,1) に対する DMA_PUT をおこなう様子である．(1) Node (1,1) のコアが Node (2,1) に対する DMA_PUT を発行する．(2) Node (1,1) の INCC がノードメモリから転送するデータを読み出す．(3) Node (1,1) の INCC がデータをパケットに加工し，Node (1,1) のルータに転送する．(4) Node (1,1) のルータが Node (2,1) のルータにパケットを転送する．(5) Node (2,1) のルータが INCC にパケットを転送する．(6) Node (2,1) の INCC がパケットを展開し，ノードメモリにデータを書き込む．

計算ノードのコアは，自身のノードにあるノードメモリに対して，命令フェッチおよびロード/ストア命令を実行することによりアクセスする．また，他のノードのノードメモリおよびメインメモリに対して，DMA によりアクセスする．各計算ノードは，他の計算ノードのノードメモリおよびメインメモリから，DMA を利用して必要なデータを自身のノードのノードメモリに複製し，処理をおこなう．

2.1.3 メモリアーキテクチャ

アーキテクチャをシンプルにするため，ノードメモリに対するキャッシュメモリは実装しない．キャッシュメモリは，性能を最適化する際に，必要に応じて実装するものとする．また，ノードメモリおよびメインメモリでは仮想アドレス変換をおこなわない．従って，メモリは全て実アドレスでアクセスする．また，アトミック性を保証する特別なハードウェアを設けない．これについては，計算ノードのコアでは割り込みが生じないため問題は起こらない．

このように，M-Core アーキテクチャはシンプルさと拡張性を備えている．コア間の通信に DMA を採用することで，効率的な通信を実現する．

2.2 マイクロアーキテクチャ

本節では，M-Core アーキテクチャを実現する具体的な設計の 1 つとして M-Coreα マイクロアーキテクチャを定義する．

2.2.1 ネットワーク

NoC の分野において，様々なネットワークトポロジが存在する．M-Coreα では，構造がシンプルで理解しやすい 2 次元メッシュを採用する．各ノードのルータは，隣接する 4 つのノードのルータおよび自身のノードの INCC の 5 方向と通信する．

パケットのルーティング方式には，構成がシンプルである XY 次元順ルーティングを採用する．この方式では，パケットはまず X 軸方向に転送され，次に Y 軸方向に転送される．XY 次元順ルーティングではデッドロックは生じない．通信経路が固定されるため，パケットの追い越しは起きない．従って，このルーティング方式を採用することで，初めに送信したパケットから順に宛て先ノードで受信されることが保証される．

パケットサイズを増やすと，通信経路の占有期間が長くなり，他の通信が長期間停滞する．一方，パケットサイズを減らすと，パケットに占めるデータの割合が減少し，転送効率が下がる．このトレードオフを考慮し，M-Coreα ではパケットに含まれるヘッダおよびデータを合わせた最大サイズを 40 バイトと定義している．

ネットワークのバンド幅を抑え，ハードウェアコストを下げるために，パケットを分割し，複数サイクルに分けて転送する．これには複数の方式が存在するが，特にワームホール

(wormhole switching) は，転送時間やハードウェアコストの面で他の方式に比べ優れている．このため，M-Coreα ではワームホールを採用する．これは，パケットをフリット (flow control unit, flit) に分割し，パイプライン式に転送する方式である．

パケットの紛失を防ぐために，ルータはフロー制御をおこなう．我々は，実装が比較的簡単で理解しやすい Xon/Xoff を採用する．これは，隣接するルータに対して，ルータの入力バッファの空きの有無を互いに 1bit で通知する方式である．この情報を元に，パケットの送信を適切に中断する．これにより，ルータの入力バッファにおけるオーバフローを防ぎ，パケットの紛失を防ぐ．

パケット転送にはノード ID を使用する．ノード ID は，X 座標および Y 座標をそれぞれ 8bit で表すものとする．また，ノード ID は 1 ワード (32bit) で表現し，下位 16bit に ID を格納する．1 ワードで表現することで，アプリケーションプログラムから自然に扱うことができる．

このように，メニーコアに関する研究・教育を効率的に進めるため，シンプルで理解しやすいネットワーク構成を採用している．

DMA 転送は，パケット転送を用いて実現されている．一度の DMA の発行により，複数のパケットを送信することがある．さらに，パケットはフリットに分割され，M-Coreα では，1 フリットは 32bit（1 ワード）のデータと 6bit の制御情報で構成される．制御を含まないパケットのサイズを 40 バイト（10 ワード）とする．つまり，1 つのパケットには最大で 10 フリットを格納される．

フリットには，ヘッダフリット，アドレスフリット，ストライドフリット，データフリット，の 4 種類を定義する．ヘッダフリットには宛先ノード ID を格納する．これは，ルータでパケットをルーティングするために利用する．アドレスフリットには，宛先ノードでデータを書き込む先頭アドレスを格納する．ストライドフリットには，宛先ノードでデータを書き込む際のストライド値を格納する．データフリットには，転送するデータを格納する．

図 4 に，Node (1,1) から Node (2,1) に対する DMA_PUT を例に，パケットが転送されるタイミングを示す．図は，3 ワードのデータを転送する場合の例である．まず，Node (1,1) の INCC の出力バッファにヘッダフリットが格納され，続いてアドレスフリット，ストライドフリット，データフリットが格納される．INCC から出力されたフリットは，1 サイクルごとに，Node (1,1) のルータ，Node (2,1) のルータ，Node (2,1) の INCC と転送される．Node (2,1) の INCC は，データフリットを受信する次のサイクルに，データをノードメモリに書き込む．図に示すとおり，データの読み出しとフリットの出力をパイプライン式に処理しながらパケットとして送信している．これは，データを全てバッファリングしてから送信を開始する方式に比べ，実行サイクル数およびハードウェア量ともに効率的である．

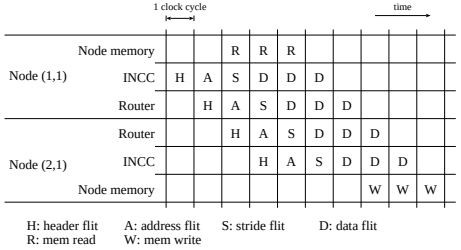

図 4 DMA 転送のタイミング.

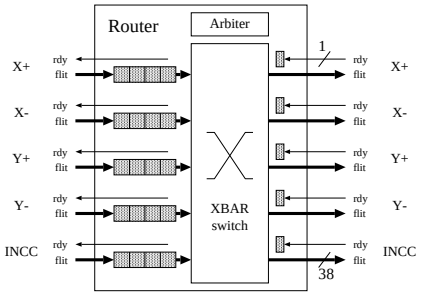

図 5 ルータのマイクロアーキテクチャ.

このように，M-Coreα は，高い並列化効率を目指すため，コア間のデータ通信オーバヘッドを削減するシンプルで効率的な通信方式を採用している．加えて，シンプルなパケットの構成を採用し，現実のハードウェアに実装することを考慮し定義している．

2.2.2 コ　　ア

コアの命令セットアーキテクチャはプロセッサアーキテクチャの教育・研究において多くの実績がある MIPS32 を採用する．浮動小数点演算をサポートする．割り込み制御機構は備えない．初期の MIPS プロセッサは 5 段程度のパイプライン処理をおこなうが，1 サイクルにほぼ 1 命令を実行する効率が良いアーキテクチャである．このため，マイクロアーキテクチャの複雑化を避けるために，シングルサイクルのプロセッサを採用する．

DMA の発行にはメモリマップド I/O を使用する．すなわち，ストア命令で特定のアドレスにデータをストアすることで DMA を起動する．このため，DMA をサポートするためのコアの変更は非常に少ない．

2.2.3 ノードメモリ

M-Coreα では，ノードメモリは典型的には 512KB の容量をもつものとする．32bit アドレスでアクセスするが，アドレスの下位 19bit を使用し上位 13bit は無視する．メモリアクセスは，(1) コアによる命令フェッチ，(2) コアによるロード/ストア，(3) INCC による読み出し，(4) INCC による書き込み，を同時に保証するため 4 ポートを必要である．ノードメモリへのアクセスは 1 ワード（32bit）を単位としておこない，1 サイクルで完了する．

2.2.4 メインメモリ

メインメモリは 64bit のメモリアドレス空間をもつ．高速にアクセスするためにページ

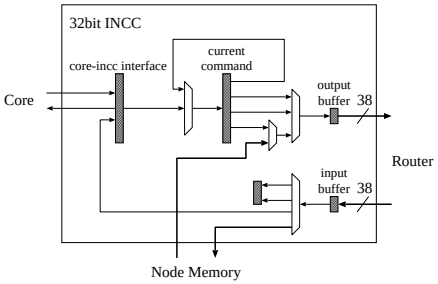

図 6 32bit INCC のマイクロアーキテクチャ.

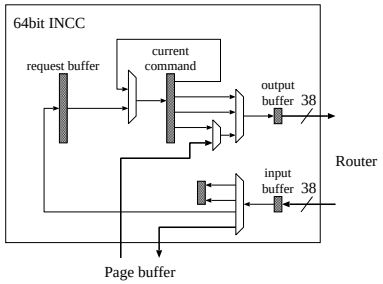

図 7 64bit INCC のマイクロアーキテクチャ.

バッファをもつ．ページバッファの容量は典型的には 4KB とし，アクセスレイテンシは 1 サイクルとする．メインメモリからページバッファへの複製には 40 サイクルを要するものとする．

2.2.5 ルータ

図 5 にルータのマイクロアーキテクチャを示す．パケットは入力線を経由して入力バッファに格納され，XBAR switch を通り，適切な方向へと出力される．各入出力ポートは 1 フリット (38bit) 分のビット幅を備える．Arbiter がラウンドロビン方式でパケットの調停を行う．入力バッファは FIFO であり，最大 4 フリットを格納する領域を備える．

一般的に，ルータは，動作周波数を向上させるために，3 段程度にパイプライン化して実装することが多い．しかし，M-Coreα では，マイクロアーキテクチャを簡潔にするためにシングルサイクルの方式を採用した．すなわち，ルータはフリットを 1 サイクルで転送する．

2.2.6 INCC

INCC は，計算ノードとメモリノードで，アクセスするメモリの仕様が異なる．ノードメモリは 32bit でアクセスするが，メインメモリは 64bit でアクセスする．このため，2 種類の INCC が必要になる．以降では，計算ノードの INCC を 32bit INCC，メモリノードの INCC を 64bit INCC と記述することとする．

まず，32bit INCC について述べる．**図 6** にマイクロアーキテクチャを示す．図の Core-INCC interface はメモリマップされたレジスタである．ノード間で通信する際には，このレジスタに対して，コアが DMA の実行に必要なデータをストアする．INCC はこの情報を図 6 の current command レジスタに複製する．

INCC は current command レジスタが格納する情報を元に，フリットを構成し，output buffer に書き込む．この際，送信するフリットの種類に応じて current command やノードメモリに格納されている値を取得し，フリットに格納する．output buffer に格納されたフリットは，次のサイクルでルータに転送される．パケットの宛先ノードでは，INCC はルータからパケットを受信し，input buffer に格納する．アドレスフリットおよびストライドフリットから，書き込むメモリアドレスを取得する．そして，データフリットを受信すると，データをノードメモリに書き込む．

次に，64bit INCC のマイクロアーキテクチャを**図 7** に示す．request buffer はメモリマップされたレジスタである．これは，32bit INCC の core-incc interface と同様に，DMA の要求を書き込むために使用する．計算ノードがメインメモリのデータを取得するには，メモリノードの request buffer がマップされているアドレスに対して要求を書き込む．これは，メモリノードに対する DMA を用いて実現する．

3. API

M-Core 基盤環境は 2 種類の API，MClib および MPIMC を提供する．いずれのライブラリもスクラッチから実装している．MClib (M-Core library) は M-Core アーキテクチャ上で動作する並列プログラムを開発する上で基本的な関数群から成る．MPIMC は Message Passing Interface (MPI) のサブセットとして実装されているライブラリである．これは MClib を利用して実装されている．

3.1 MClib

本節では MClib について述べる．MClib を用いることにより，M-Core 上で動作する並列プログラムを容易に開発できる．ライブラリのコードの大部分は C で記述し，一部をアセンブラで記述している．

MClib を使うことで，アプリケーションプログラムは C 言語により記述でき，初心者が容易にプログラミングできる．これについては，6 章で議論する．加えて，文献 1) で，MClib を用いたサンプルプログラムを示し，アプリケーションプログラムを容易に開発できることを述べた．

3.2 MPIMC

並列プログラムの開発においては，MPI が利用されることがある．これは主に，分散メモリシステムを想定して定義された API である．M-Core アーキテクチャでは，メインメモリは全ての計算ノードで共有するが，ノードメモリは各計算ノードで独立して利用するため，分散メモリシステムとして捉えることができる．そこで，M-Core アーキテクチャ向けライブラリとして，MPIMC (MPI library for M-Core Architecture) を提供する．これに

より，MPI で記述された並列プログラムを M-Core 上で動作させることができる．

現在の MPIMC が提供する関数は MPI のサブセットであるが，NPB のソースコードで利用している関数を全て提供している．このため，教育・研究向けには十分実用的である．

4. シミュレータ SimMc

メニーコアを対象とした並列プログラムの評価環境として，またはメニーコアのアーキテクチャ研究における検証および性能評価のための環境として，M-Core アーキテクチャのシミュレータ SimMc を提供する．本章ではシミュレータ SimMc の設計と実装ついて述べる．

4.1 設計指針

一般に，プロセッサシミュレータに求められる要素として，拡張性，可読性，評価精度，豊富な機能，シミュレーション速度，などが挙げられる．しかし，これらの要素の間にはいくつかのトレードオフが存在し，全ての要望を同時に満たすのは困難である．本節では，SimMc の利用目的に合わせてこれらの要素から適切なものを選択する．

プロセッサアーキテクチャの研究・教育では，アーキテクチャの評価環境としてシミュレータを利用する．また，アーキテクチャを深く正確に理解する目的でも，シミュレータのソースコードを利用できる．このためには，サイクルレベルのシミュレーション精度，高い拡張性・可読性が求められる．

ソフトウェアの研究・教育においては，開発したソフトウェアの実行環境としてシミュレータを利用できる．ここでは，プログラムの並列化効率を高めるために，実行時の統計データを出力する様々な機能が求められる．また，実用的なアプリケーションプログラムが豊富に動作することも必要である．

シミュレーション速度も重要な要素ではあるが，これまで示した他の要素に比べ優先度は低いと考えている．従って，シミュレータの可読性を損なう可能性がある並列化などの高速化技術は適用しない．

4.2 機　　能

SimMc は，シミュレータとしての基本的な機能に加え，コアによる出力機能，DMA の履歴の表示機能，ネットワークトラフィックの可視化機能，マルチバイナリ機能，を提供する．

シミュレータの基本的な機能として，実行サイクル数を表示する．これは，プロセッサやソフトウェアの性能評価に利用できる．また，コマンドラインオプションによるノード数の変更などの機能も提供する．

アプリケーションプログラムのデバッグや計算結果の確認を容易にするために，計算ノードのコアが文字列を出力する機能を提供する．これは，SimMc がソフトウェアで実装されていることを利用して実装したもので，本来，計算ノードが備えていない機能である．

```
1       cycle = 0;
2
3       while (finishcond()) {
4           for (int i = 0; i < nr_node; i++)
5               node[i].core->chip->step_funct();
6
7           for (int i = 0; i < nr_node; i++) {
8               node[i].incc->update();
9               node[i].router->update();
10          }
11
12          for (int i = 0; i < nr_node; i++)
13              node[i].router->switching();
14
15          for (int i = 0; i < nr_node; i++) {
16              node[i].incc->comm();
17              node[i].router->comm();
18          }
19
20          ++cycle;
21      }
```

図 8　SimMc のソースコードのメインループ．

並列プログラムの開発では，通信の部分でバグが混入することが多い．加えて，性能に大きく影響を与えるネットワークの挙動は，プロセッサの重要な統計情報である．このため，ノード間通信に関する情報を出力する機能を備える．出力する粒度として，DMA の発行単位，パケットの転送単位，フリットの転送単位，の 3 種類を提供しており，必要に応じて選択できる．これにより，ノード間通信の情報を大局的にあるいは詳細に取得できる．

通信の情報は，テキストだけでなく，視覚的に表現することで，アプリケーションプログラムのバグや，通信のボトルネック等を効率的に発見できる．メニーコアアーキテクチャのネットワークの挙動を視覚的に理解できる．このため，ネットワークのトラフィックを視覚化する機能を提供する．これは，ある時刻のネットワークにおけるフリットの存在位置を出力するものである．対話的に操作することができ，表示する時刻を前後させたり，通信が発生する時刻までスキップする等の機能がある．

複数の異なるアプリケーションを別々の計算ノードで同時に実行することによる影響や，タスクの配置方法が性能に与える影響を調査するため，マルチバイナリ機能を提供する．これにより，ノード単位で実行するプログラムを指定できる．

4.3 実　　装

図 8 に SimMc のソースコードのメインループを示す．見やすいよう，コードの一部を整形している．1 行目の変数 cycle は実行サイクル数を表すカウンタである．3-21 行目がメイ

ンループである．1 回のループの実行で 1 サイクル分のシミュレーションを行う．繰り返しの条件となる finishcond() は，全ての計算ノードのコアがアプリケーションプログラムを終了した場合に偽を返し，それ以外は真を返す関数である．4-5 行目で，全ての計算ノードのコアについて 1 サイクル分のシミュレーションをおこなう．7-18 行目は INCC とルータのシミュレーションをおこなう．7-10 行目では，INCC およびルータの内部にあるレジスタを更新する．12-13 行目では，ルータにおいて，パケットのルーティングや XBAR switch の調停をおこなう．15-19 行目で，ルータ-ルータ間及び INCC-ルータ間の通信をおこなう．20 行目で，カウンタをカウントアップする．

SimMc は C++で実装されている．可読性を高めるため，実装するユニットごとにソースコードを分割して記述している．また，空行やコメント行を含めても約 3,500 行と，非常に少ないコード量で実装している．このため，利用者は短期間でソースコードを理解できる．

このように，マルチコアおよびメニーコアの教育・研究に有効に活用するため，SimMc は，高い可読性，高い拡張性，サイクルレベルのシミュレーション精度，ハードウェアの統計情報を出力する豊富な機能，を考慮して実装されている．

5. サンプルプログラムと検証機能

5.1 豊富なサンプルプログラム

M-Core システムの新規利用者が，MClib や MPIMC を用いた並列プログラムの開発方法を効率良く学習できるよう，豊富なサンプルプログラムを提供している．

サンプルプログラムには，マイクロベンチマークと，NPB を収録している．前者には，Hello, World，DMA を実行するサンプル，バイトニックソート，Equation Solver Kernel，行列積，姫野ベンチマーク，N-queen，を提供する．また，後者には，NPB 3.3 で公開されているものの中で MPI を用いて実装されている 9 種類のベンチマークを提供する．

マイクロベンチマークは，容易に理解できるプログラムを収録している．並列プログラミングのスキルを徐々に高めていけるよう構成しているため，チュートリアルとしても利用できる．一方，NPB は，より現実的なベンチマークプログラムとして利用できる．

5.2 動作の検証システム

研究・教育を効率的により進めるため，動作検証システムを提供する．これは，前節で述べたマイクロベンチマークの正しい出力結果と，検証を自動でおこなうスクリプトで構成される．スクリプトにより，特定のテストケースを用いて提供する全てのマイクロベンチマークを実行し，正しい出力結果と照合する．結果が一致しない場合は，その旨を出力する．

このシステムは，アプリケーションプログラムの出力のみを比較する．このため，出力に影響を与えない変更であれば，このシステムを利用できる．これには，例えば，SimMc を変更・拡張する場合，または，API の一部を書き換える場合などが挙げられる．

このシステムを利用することで，動作検証が自動化され，プロセッサアーキテクチャおよびシステムソフトウェアの研究を効率的に進めることができる．

6. M-Core 基盤環境の評価

M-Core 基盤環境の特徴を以下に列挙する．(a) 多くのプラットフォームで動作する．(b) 多数のアプリケーションプログラムが動作する．(c) 新規利用者が理解しやすく扱いやすい．(d) カスタマイズが容易である．(e) 現実的な時間で評価を完了する．本章では，これらの項目について明らかにする．また，研究に対する実用性を示すため，M-Core 基盤環境を用いた研究の例を述べる．

6.1 プラットフォーム

M-Core ソフトウェアシステムが様々な評価環境で動作するよう，特定のプラットフォームに依存する記述は極力省いている．ただし，リトルエンディアンのマシンで動作することを前提として実装している．動作を確認したプラットフォームを以下に列挙する．

- Linux Redhat 5.2, x86_64, gcc 4.1.2
- Linux Debian lenny 5.0.3, x86_64, gcc 4.3.2
- TSUBAME (SUSE Linux Enterprise Server 10, x86_64, gcc 4.1.2)
- FreeBSD 8.0, x86, gcc 4.2.1
- Cygwin 5.1, x86, gcc 4.3.2
- Mac OS X 10.6.2, x86, gcc 4.2.1
- LinuxLink 6.2, MIPS 4KEc, gcc 4.2.0

なお，動作確認には，姫野ベンチマーク，Equation Solver Kernel，NPB の FT，の 3 種類のプログラムを使用し，ノード数は 64 ノードとした[⋆1]．

一方，以下に示すプラットフォームでは，シミュレータやアプリケーションプログラムはコンパイルできるが，シミュレーションは正常に実行されない．これは，プロセッサのエンディアンの違いに起因している．

- Linux Fedora 9, Cell/B.E., gcc 4.3.0

このように，一部動作しないマシンはあるものの，M-Core ソフトウェアシステムは多く

⋆1 ただし，LinuxLink 6.2, MIPS 4KEc, gcc 4.2.0 については，マシンのメモリ容量が不足したため，姫野ベンチマークは 16 ノードの評価をおこなった

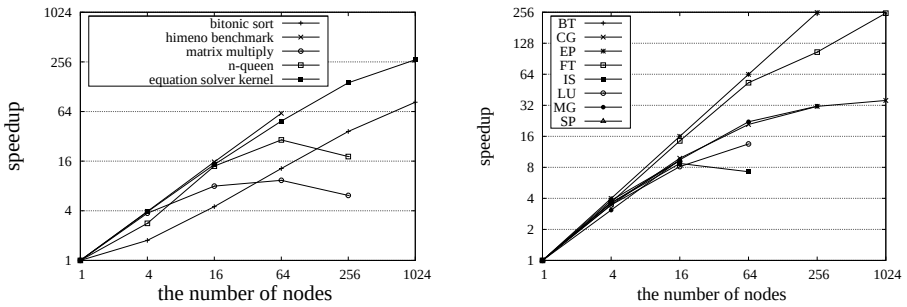

図 9 マイクロベンチマークと NPB を用いた，コア数の増加に伴う速度向上.

の主要なプラットフォーム上で動作する.

6.2 豊富なアプリケーションプログラム

豊富なアプリケーションプログラムが動作することで，以下の利点が得られる．(1) M-Core 基盤環境を用いた研究において，豊富なベンチマークプログラムとして利用できる．(2) 並列プログラミングの教育において，豊富な例題プログラムを用いて指導できる．本節では，豊富なベンチマークプログラムとして，我々が並列化したマイクロベンチマークと NPB が，M-Core ソフトウェアシステムで動作することを示す.

図 9 に，NPB を用いた，コア数の増加に伴う速度向上の評価結果を示す．ここでは，問題サイズとしてクラス S を採用している．グラフに示すとおり，1,024 ノードのシミュレーションも可能である．1,024 ノードまでの結果を載せていないものも一部あるが，これは，クラス S の仕様によりノード数が制限されるためである．また，一部の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 512KB のノードメモリでは不足するものがあり，本論文の評価ではノードメモリの容量を増やして対処している．将来的には，仮想アドレスまたはオーバレイを適用することで対応することを予定している．また，グラフの傾向が，一般的な計算機環境で評価した結果と同様の傾向であること，計算結果が正しいことから，評価は正確におこなわれている.

6.3 新規利用者にもやさしい基盤環境

M-Core 基盤環境の開発目的のひとつに，マルチコア・メニーコアに関する教育用途への適用がある．本節では，講義の題材として M-Core 基盤環境を利用した実例を示し，教育用途への有用性を明かにする.

コンピュータアーキテクチャの講義を受講している学部の学生および修士課程の学生に対し，M-Core アーキテクチャ，および M-Core ソフトウェアシステムの利用法を解説した．その後，行列積を求める逐次プログラムを並列化し高速化するという課題を課し，さらに修士課程の学生に対しては，難易度の高いクイックソートの並列化も課題に加えた．その際，解答に用した時間と，並列化により得られた速度向上率を報告するよう求めた.

表 1 課題解答者の割合，課題に要した平均時間，および 16 ノード実行時の速度向上率

課題内容	学部			修士		
	解答率	解答時間	速度向上率	解答率	解答時間	速度向上率
行列積の並列化	47%	9.0 時間	4.9 倍	73%	8.9 時間	3.9 倍
クイックソートの並列化		-		53%	10.2 時間	4.6 倍

これらの課題に対し，学部の学生 14 人，修士課程の学生 22 人からのレポートの提出があった．なお，学部の学生は最低 1 年，修士課程の学生は最低 3 年のプログラミングの経験がある．ただし，必ずしも全員が並列プログラミングの経験があるわけではない．特に学部の学生は並列プログラミングに関する講義の受講歴が無く，多くの学生は未経験である.

表 1 に，各課題に対して解答した学生の割合，課題に要した平均時間，および 16 ノード実行時の速度向上率を示す．修士課程の学生に関しては過半数の学生が解答し，学部の学生も約半数の学生が解答している．また，所要時間は学部・修士課程の学生ともに，各課題につき 10 時間前後であった．他の評価環境を用いた場合との比較はおこなっていないが，これは十分に短い時間であると考えられる．また，この短い期間の中で，16 ノード実行時に 4，5 倍の性能向上を達成している.

この課題を通じた感想や要望など多数のフィードバックを受けた．苦労した点として，共有変数の扱い方，複数コアの同期処理の実装などが見受けられた.

このように，M-Core 基盤環境を利用することで，並列プログラミングにおける重要な要素を効率的にかつ実践的に教育することができる.

6.4 カスタマイズの容易性

性能上のボトルネックとなる部分を見いだし，性能を改善する手がかりを得るめには，プロセッサの統計情報を収集する．ここでは，シミュレータを変更し，求める情報を容易に収集できることが重要である．本節では，M-Core 基盤環境の使用例として，SimMc の機能を利用し，あるいは SimMc の一部を変更して，プロセッサの統計データを収集する．そして，マルチコアおよびメニーコアの教育・研究における M-Core 基盤環境の有用性を示す.

図 10 に，ノード数と DMA の発行回数の関係を計測した結果を示す．DMA に関する基礎的なデータは SimMc の機能を用いて容易に採取できる.

アプリケーションプログラムの書き方によっては，ノード間の通信が混雑し，性能が出な

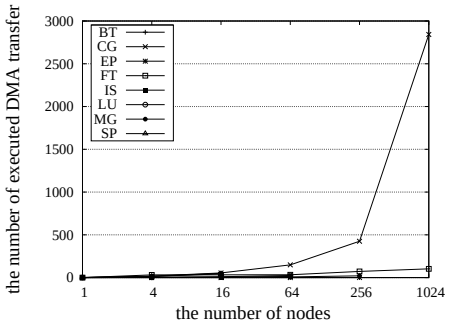

図 10 DMA の発行回数.

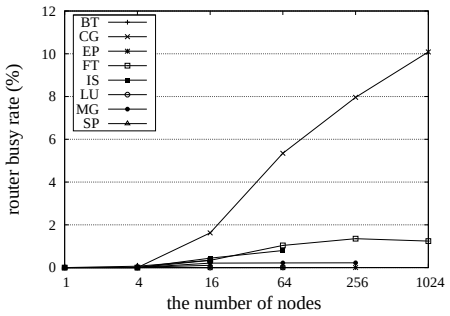

図 11 最も通信が集中するルータにおける混雑度.

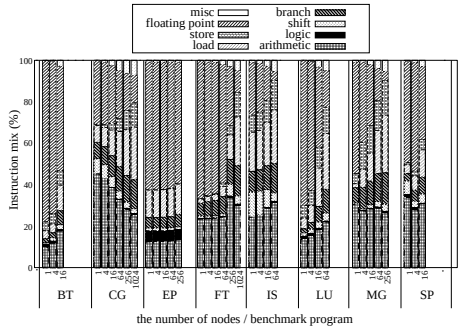

図 12 命令ミックス.

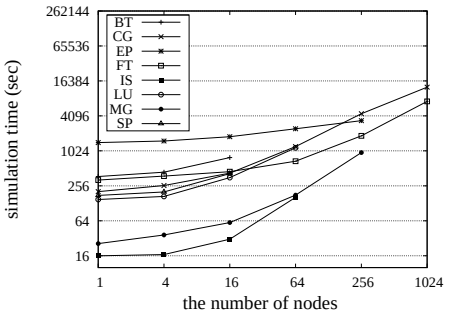

図 13 シミュレーション時間.

い場合がある．これは，パケットが効率よく転送されていないためである．この場合，ルータのフロー制御によって，パケットが停滞していると考えられる．

そこで，パケットが最も混雑したルータにおいて，パケットが停滞した期間が全実行時間に占める割合を求めた．**図 11** に評価結果を示す．1,024 ノードで実行すると，一部の混雑するルータでは，混雑する期間が 5%程度になるプログラムがあることがわかる．

図 12 に，実行時の命令ミックスを示す．全体的に，実行するノード数が増加するに従い，計算に関与する命令の割合が減少している．これは，タスクを分割することで，各ノードが処理する計算量が減少するためである．また，ノード数が増えるに伴い，分岐命令やロード命令の割合が増加している．これは，実行するノード数が多くなると DMA の通信レイテンシが増大し，データの受信待ちでメモリをポーリングする回数が増加するためである．

ここで紹介した統計データはいずれも，SimMc に実装されている機能を利用して，あるいはシミュレータのソースコードに高々10 行程度のコードを加えて計測したものである．

メニーコアプロセッサを対象とするアーキテクチャやシステムソフトウェアの研究において，M-Core 基盤環境を用いて様々な評価をおこなうことで，性能向上のヒントが得られる．本節では，M-Core 基盤環境を用いてプロセッサの統計情報を取得し，コア数の増加に伴うプロセッサの性能や挙動の変化を考察した．そして，ここで示した様々な統計情報は，SimMc を利用することで簡単に計測できることを示した．

6.5 実用的な時間での評価

我々はシミュレーション速度にある程度の犠牲を払っても，可読性・拡張性を第一の指針としてシミュレータ SimMc を実装した．しかし，評価を現実的な時間で完了しなければ，優れた評価環境とは言えない．

本節では，マイクロベンチマークおよび NPB を用いて，SimMc によるシミュレーションが実用的な時間で完了することを示す．評価に用いるマシンの仕様は，次のとおりである：Intel(R) Core i7 2.80GHz，8GB memory，Debian GNU/Linux 5.0.3，gcc 4.3.2.

図 13 に，NPB のシミュレーション時間を示す．グラフより，実用的な時間でシミュレーションを完了することがわかる．例えば，1,024 ノードにおいても，高々10,000 秒程度である．プログラムが並列化されているため，ノード数が増加するに従って実行サイクル数は一般に減少する．このため，シミュレーション時間はそれほど悪化していない．

このように，評価に用いたマイクロベンチマークや NPB 程度であれば，1,024 ノードのシミュレーションにおいても実用的な時間で評価できる．

6.6 実研究への応用

M-Core 基盤環境は，既に実際の研究で利用されている．以降では，それらの研究について簡単に述べる．

文献 2) では，多数のコアの利用と高機能ルータによって性能向上を目指す SmartCore システムを提案している．ルータに対して，パケットの複製，送信先の変更，比較をおこなう機能を追加することで，複数コアの多重実行を支援し，ネットワークのバンド幅の向上，およびディペンダビリティの向上を目指す．この研究では，M-Core アーキテクチャを拡張し，M-Core ソフトウェアシステムを用いて評価をおこなっている．

文献 3) では，マルチコアおよびメニーコアにおける豊富なコアの有効な活用方法として，データ供給支援コアを提案している．他のコアに対してデータを供給する専用のコアを用意

することで，メインメモリのアクセス回数を削減，メモリアクセスレイテンシの削減を達成する．この研究では，M-Core 基盤環境を利用し，性能評価をおこなっている．

文献 4) では，メニーコアアーキテクチャの高速なシミュレーション環境 ScalableCore を提案している．これは，小容量の FPGA を多数接続し，高速なシミュレーションと拡張性を実現する．この研究では，M-Core アーキテクチャをシミュレーション対象のメニーコアアーキテクチャとして利用している．そして，HDL を記述するにあたり，SimMc のソースコードを利用している．また，動作検証には M-Core ソフトウェアシステムを用いている．

このように，M-Core 基盤環境は既に実研究において利用されている．上で述べた事例から，M-Core 基盤環境は，メニーコアプロセッサのアーキテクチャ研究およびシステムソフトウェア研究に対する実用性が高いといえる．

7. 関 連 研 究

M-Core アーキテクチャの関連研究について述べる．

商用のマルチコアプロセッサとして，Intel Core2 Duo，Intel Core i7，AMD Phenom，などが広く普及している．これらは，命令セットアーキテクチャに x86 を採用している．x86 は命令列が複雑な CISC 方式で設計されており，教育向けではない．また，これらのプロセッサでは，メモリアーキテクチャとして共有キャッシュ方式を採用している．この方式はチップ上に搭載するコア数が増加するにつれ，ハードウェアが複雑化し，コストもかかる．このため，高々十数個のコアを搭載するマルチコアプロセッサでは有効であるが，数千個，数万個のコアを搭載するメニーコアには対応しきれない．

別の商用プロセッサとして，Cell Broadband Engine (Cell/B.E)，がある．これは，API が初心者には扱い難く，教育用途としては不向きである．

研究レベルのマルチコア・メニーコアとして，RAW マイクロプロセッサ Tile64 TRIPS がある．これらは，データフローモデルをベースとしており，RISC プロセッサとはアーキテクチャが大きく異なる．そのため，シンプルな構造の RISC プロセッサに比べ，アーキテクチャの教育用途としては不向きである．これらの中には，効率的に分散処理をおこなうために，独自に命令セットを定義したり，ネットワークを多重化しているものもある．

これらの関連研究と比較して，我々は教育・研究に活用することを目的に定義しており，プロセッサアーキテクチャの教育・研究の素材として最も実績がある MIPS を採用している．これにより，コンパイラやライブラリなどの開発環境の構築も比較的容易であり，既存の技術を再利用できる．また，教育・研究に使用することを考慮し，理解しやすく拡張性のあるシンプルなネットワークアーキテクチャを採用している．

8. む す び

マルチコアおよびメニーコアプロセッサを対象としたソフトウェアおよびプロセッサアーキテクチャの教育・研究を支援するため，我々は M-Core 基盤環境 V1.0 を開発した．本稿では，M-Core 基盤環境の構成要素について述べた．そして，様々な評価結果に基づき，マルチコアおよびメニーコアの研究・教育に対する有用性を示した．

今後の課題について述べる．M-Core アーキテクチャをベースとする様々なマイクロアーキテクチャを定義するとともに，豊富な種類のハードウェア構成要素の提供を目指す．具体的には，異なる命令セットのコア，パイプライン化や Out of Order 化したコア，ノードメモリおよびメインメモリに対するキャッシュメモリ，パイプライン化されたルータなどをハードウェア要素として提供したい．これらを取捨選択することで，必要とする様々なマイクロアーキテクチャの構築が容易になり，実用性が向上する．

開発した M-Core 基盤環境 V1.0 はフリーソフトウェアとして公開する予定である．

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) 「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．ルータに関して適切な御指導を賜わりました，電気通信大学の吉永努先生，慶応義塾大学の松谷宏紀博士に感謝いたします．

参 考 文 献

1) 植原　昂，佐藤真平，森谷　章，藤枝直輝，高前田伸也，渡邉伸平，三好健文，小林良太郎，吉瀬謙二：シンプルで効率的なメニーコアアーキテクチャの開発，情報処理学会研究報告 2008-ARC-180，Vol.2008, No.101, pp.39–44 (2008).
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) Mori, Y. and Kise, K.: The Cache-Core Architecture to Enhance the Memory Performance on Multi-Core Processors, *Workshop on Ultra Performance and Dependable Acceleration Systems held in conjunction with PDCAT'09*, pp. 445–450 (2009).
4) 高前田伸也，渡邉伸平，　姜軒，藤枝直輝，　植原昂，三好健文，吉瀬謙二：メニーコアアーキテクチャ研究のためのスケーラブルな HW 評価環境 ScalableCore システム，情報処理学会研究報告 2009-ARC-185，pp.1–10 (2009).