Vol.2009-ARC-185 No.3
2009/10/26

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

高前田 伸也[†1] 渡邉 伸平[†1] 姜 軒[†1]
藤枝 直輝[†1] 植原 昂[†1]
三好 健文[†1,†2] 吉瀬 謙二[†1]

メニーコアアーキテクチャの動作を現実的な時間でシミュレーションするために，我々はハードウェアによるシミュレーション環境 ScalableCore を提案している．これは，シミュレーションノード (ScalableCore Unit) を，共通の接続インターフェース (ScalableCore Board) を用いて接続することで，高い拡張性を実現する．本論文では，小容量の FPGA を複数用いた ScalableCore システムの実装方法を検討する．この実装方法は，メニーコアプロセッサ内の各コアを各 FPGA に対応づけることで，ハードウェアによるシミュレーション環境の実装で問題となる，複雑さの軽減を可能とする．我々は，試作した ScalableCore Unit と ScalableCore Board を用いた ScalableCore システム上に，メニーコアアーキテクチャM-Core の実装を行っている．実装中の ScalableCore システムにおいて，シンプルな M-Core 用アプリケーションを動作させ，実際にプロセッサの構築ができることを確認した．

ScalableCore system: Scalable HW Evaluation Environment for Many-core Architecture Researches

Shinya Takamaeda,[†1] Shimpei Watanabe,[†1]
Ken Kyou,[†1] Naoki Fujieda,[†1] Koh Uehara,[†1]
Takefumi Miyoshi[†1,†2] and Kenji Kise [†1]

In order to practically simulate many-core processor, the authors proposed ScalableCore, which is a simulator by using hardware. ScalableCore consists of Simulation nodes named ScalableCore Unit and common interfaces between ScalableCore Units named ScalableCore Board. Each of them corresponds to cores in the target processor and the buses between cores, respectively. Since each ScalableCore Board can connect four ScalableCore Units around it, ScalableCore can realize high scalability. This paper shows an implementation method of prototyping system with a lot of small-sized FPGAs. The proposed method reduces that the implementation complexity which is a major problem for constructing a simulator by using hardware. The authors implement Many-core architecture or M-Core on a preliminary system of ScalableCore with commercial FPGAs and our designed printed-circuit boards. On the current system, some simple applications for M-Core work well. It is confirmed that the actual construction of a Many-core processor on a ScalableCore system is possible.

1. はじめに

半導体製造プロセスの微細化により，1 チップに複数のコアを集積するマルチコアプロセッサが実現されている．1 チップに複数のコアを搭載することにより，処理速度 · 電力効率の向上を達成するマルチコアプロセッサは，現在広く普及している．継続的なプロセスの微細化により，今後はさらに多くのコアを 1 チップに集積するメニーコアプロセッサへと向かうと考えられる．

メニーコアプロセッサアーキテクチャや，メニーコアプロセッサ時代のソフトウェアに関する研究 · 開発を効率的に行う為には，様々なアイディアを迅速に検証できる環境が必要となる．構成の変更が柔軟であること，特定のハードウェアが必要でないことから，現在はソフトウェアによるシミュレータが用いられることが多い．しかし，メニーコアプロセッサを対象とするシミュレーションには，シミュレーション対象のコア数の増加に伴い，シミュレーション速度が大幅に低下してしまうという欠点がある．その為，それを解決するより高速なシミュレーション環境が求められている．

そこで我々は，ハードウェアによる高速プロトタイピングシステム ScalableCore を開発している．シミュレーション対象の一構成要素を実装した”ScalableCore Unit” を共通の接続インターフェースの”ScalableCore Board” を用いて複数接続することで，スケーラビリティを満たしながら，柔軟かつ高速なシミュレーション環境の実現を目指す．

本稿では，現在実装中の ScalableCore システムについて述べる．まず 2 章で関連研究に

†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 科学技術振興機構
Japan Science and Technology Agency(JST)

ついて述べ，3 章では実装の対象とするメニーコアアーキテクチャとそのソフトウェアシミュレータついて述べる．4 章では我々の提案する ScalableCore について述べ，5 章で現在の実装とその動作について解説し，その評価を行う．最後に，6 章で本稿をまとめる．

2. 関連研究

コンピュータアーキテクチャの研究やソフトウェアの性能向上のための解析の道具として，プロセッサのシミュレーション環境の研究は数多くある．近年では，コア数増加の流れに伴い，マルチコア/メニーコアを対象としたシミュレーション環境の研究が多く行われている．マルチコア/メニーコア上の複数のコアを用いて効率良くプログラムを動作させるためには，コンピュータアーキテクチャだけではなく，その上で動作するミドルウェアやアプリケーションを含めた研究開発が必要である．そのため，特にマルチコア/メニーコア向けのフルシステムシミュレーション環境の構築が今後ますます重要になってくる．ソフトウェアによるフルシステムシミュレーションをイベント駆動型で実現する場合，これは，現実的な時間で終了しない．そのための，functional-simulation と timing-simulation を組み合わせたシミュレーション環境については数多くの研究があり，GEMS[1]，M5[2] などは広く使われている．“How to Simulate 1000 Cores”[3] では，functional-simulator が生成するトレースを元に，シングルコア上で動作するマルチスレッドアプリケーションの各スレッドをシミュレーション対象プロセッサの各コアにマップ·シミュレーションし，timing-simulator などで構成されるバックエンド側でスレッド間の同期をとるという，シミュレーションモデルが提案されている．このモデルでは，既存の full-system simulator で動作するマルチスレッドアプリケーションをメニーコアプロセッサ上で実行した場合の性能を評価することが可能である．また，AMD のプロセッサシミュレータである SimNow をコアとして用いたフルシミュレーション環境[4] などもある．これらは，それまでの手法と比較すると高速なシミュレーションを実現することができ，またソフトウェアならではの高い柔軟性で容易に様々なアーキテクチャ構成を試行することができる．その一方で，ミドルウェアやアプリケーション開発において，大規模なプログラムを実行させるには，十分な速度を実現できていない．

ソフトウェアシミュレーションのかわりに，柔軟に構成を変更可能な FPGA を用いることで，高速なシミュレーションの実現を目標とする研究には，RPM Project[5] をはじめとして，FAST[6] や，ATLAS[7]，ProtoFlex[8] などがある．FAST は，4 個の Processor Tile を持つシミュレーションプラットフォームであり，チップマルチプロセッサにおけるメモリシステムの研究に適していた．RAMP[9] は，ソフトウェアとハードウェアの両面からマルチコア/メニーコアの研究を加速するためのプラットフォームの研究を推進している．ATLAS では，トランザクショナルメモリをサポートした CMP についての実用的なシミュレーション環境を提案している．これは，ソフトウェアシミュレーションである TASSEL と比較して 100 倍の高速化を実現している．また，ProtoFlex では，OS の動作をも模倣するフルシステムシミュレーションの 38 倍の高速化を実現している．これらは，機能レベルのシミュレーションをソフトウェアに任せることで，複雑な実装を回避している．

しかし一方で，ソフトウェアとのハイブリッドシミュレーション環境では，現実的なハードウェアリソース量では実現しえないような不当な仮定をおいてしまう可能性がある．そのため，ハードウェアリソース量を評価するために，全てをハードウェアとして実現可能な RTL ベースのシミュレーション環境と，それを現実的な時間で運用するための FPGA によるプラットフォームが必要である．

3. メニーコアアーキテクチャとソフトウェアシミュレータ

メニーコアプロセッサのシミュレーションの高速化について考える．本稿では簡単のため，トポロジを 2 次元メッシュに限定する．高速化の題材として M-Core アーキテクチャと，そのシミュレータ SimMc[10] を取り上げる．

3.1 M-Core アーキテクチャ

図 1 に M-Core アーキテクチャの構成を示す．M-Core アーキテクチャは，小規模で均一なコアを多数並べる構成を採用し，システムソフトウェアとの協調によりチップの高性能化を目指す．

M-Core の各構成要素はノードと呼ばれる．各ノードにはノード識別子 (ノード ID) が割り当てられており，これを用いて通信を行う．各ノードは 2 次元メッシュネットワークで接続されており，DMA を用いて互いにデータ転送が可能である．

図 1 において，タイル状に配置されたノードは，計算専用のコアを含む計算ノードである．左上の大きなノード (0,0) は，外部 I/O 処理などの割り込みや OS が動作するオペレーションノード，上部に並んでいる y 座標が 0 のノードは，オフチップメモリとの通信を受け持つメモリノードである．

図 2 に，計算ノードの詳細を示す．計算ノードは，Core(演算処理ユニット)，ノードメモリ (各ノードが持つ小規模なメモリ)，INCC(Inter Node Communication Controller)，ルータとで構成される．各ノードは独立したメモリアドレス空間をもち，他のノードあるいはメインメモリへのアクセスは INCC を介した DMA 転送により行う．

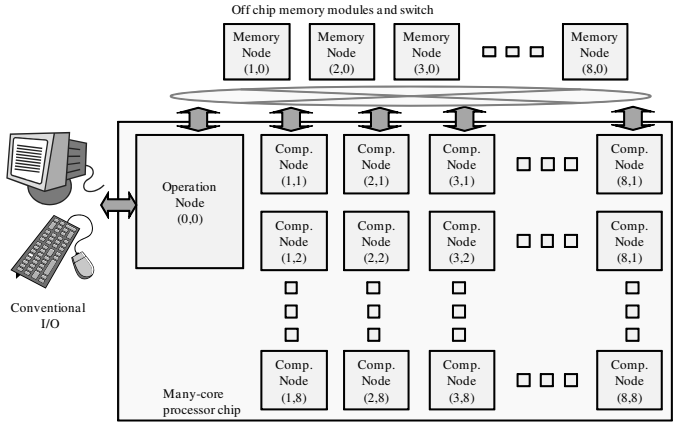

図 1 M-Core アーキテクチャのモデル

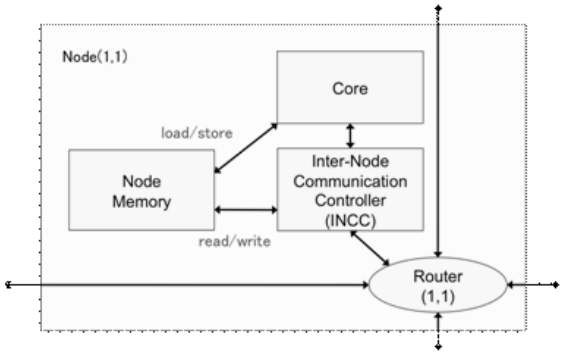

図 2 計算ノードの詳細

M-Core アーキテクチャの cycle-accurate なシミュレータとして SimMc がある．SimMc はアーキテクチャの構成を容易に変更できることを指針として設計されている．コマンドラインオプションにより，プロセッサの構成を容易に変更できる．また，ソースコードの可読性を意識したコーディングにより，シミュレータ自体に手を入れやすいよう配慮されている．これにより，柔軟性のあるシミュレーションを実現する．

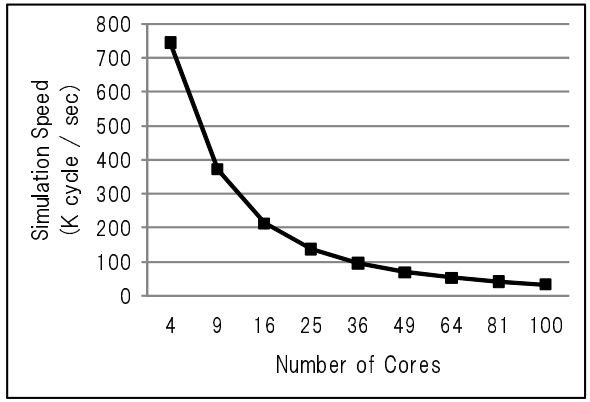

図 3 メニーコアプロセッサのシミュレータ SimMc のシミュレーション速度．シミュレーション対象のコア数が増大するとともに速度が低下し，100 コアのシミュレーションで約 32Kcycle/sec となっている

3.2 シミュレーション速度の評価

図 3 は，SimMc において並列アプリケーション Equation solver kernel を動作させた際の，シミュレーション対象コア数とシミュレーション速度の関係である．実行環境は Intel Xeon X5365 3GHz，メモリ 16GB のマシンである．グラフの横軸はノード数，縦軸は単位時間あたりのシミュレーションサイクル数を表す．シミュレーション対象のノード数が増加するに従い，おおよそそれに反比例して速度が低下し，100 コアのシミュレーションでは 32K サイクル/秒と非常に低速である．したがって，メニーコアプロセッサを大規模なアプリケーションを用いて評価するためには，シミュレーション環境の高速化が不可欠である．

3.3 シミュレーションの高速化

シミュレーションの高速化手法について考える．シミュレーションの高速化をする際，まず考えられるのがソフトウェアの最適化 · 並列化によるものである．特に現行の SimMc では，各コア及びコア間の通信を直列にシミュレーションしているため，並列化 · 最適化により一定の効果は期待できる．しかし，ソフトウェアの可読性 · 柔軟性が並列化 · 最適化により失われ，SimMc の利点である柔軟性が失われてしまう可能性が高い．また，グリッドコンピューティングなどを用いて，大規模に並列化を行うことも，SimMc のような細粒度の通信を必要とするアプリケーションでは難しい．

一方でハードウェアによるシミュレータの実装を考える．これは，ソフトウェアシミュレータの持つ柔軟性を犠牲にしつつも，ソフトウェア比で大幅なスピードアップが見込める．

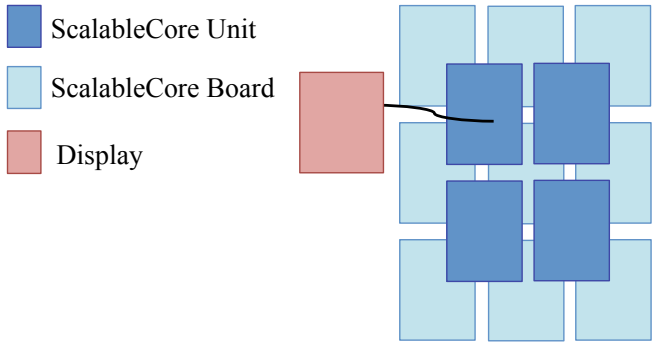

図 4 2 × 2 の ScalableCore システムにおける接続方法.

もちろん，HDL 記述に気をつければある程度はシミュレーションの柔軟性を確保することができる．また，ロジックの複雑さやレジスタの容量，クリティカルパスの長さなど「実際にハードウェアとして動作する」適切なアーキテクチャであることを検証できるメリットがある．

4. ScalableCore

我々は，ソフトウェアシミュレーションの速度の問題を解決し，メニーコアプロセッサに関する研究を加速させるための，ハードウェアによるシミュレーション環境 ScalableCore を提案する．

ScalableCore システムの主な構成要素は，メニーコアプロセッサのコアなどの構成要素単位に対応するシミュレーションノードの ScalableCore Unit と，それらを接続するためのインターフェースの ScalableCore Board(接続用ボード) である．各 Unit にシミュレーション対象の一構成要素と Unit 間の通信機構を実装し，それらを Board を用いて接続することにより，メニーコアプロセッサの評価環境を実現する．図 4 のように 4 枚の Board が 1 つの Unit を取り囲むように配置し，各 Unit の I/O ポートを隣接する 4 つの Unit の I/O ポートと接続する．Unit 間のポート数が制限されるため，Unit 間の通信はシリアル転送とし，シリアル通信モジュールを上下左右の 4 方向分実装することとする．

この接続方法により，シミュレーション速度を保ちつつ，ScalableCore Unit の数の増減が容易となる．このことにより，柔軟に様々なアーキテクチャの試行することができる．

また，コマンドインタプリタ型の小型液晶ディスプレイを用いて，シミュレーションの状況などを表示することが可能である．

5. 実装と評価

前章で述べたコンセプトのもと，ScalableCore システムを実装し，評価する．

オペレーションノードやメモリノードを含めた M-Core アーキテクチャの全実装を行うことを目標とし，本実装ではその第一歩として，DMA 転送の一部機能およびメモリノードやオペレーションノードなどを省いた M-Core アーキテクチャを ScalableCore システム上に実現し，その上でシンプルなアプリケーションを実行させることを目指す．

5.1 実装方法の検討

ScalableCore システムの実装方法について検討する．ScalableCore システムでは，現実的な時間でのシミュレーションの実現と共に，アーキテクチャ検証のための柔軟性や，実装コストの低減が必要となる．そのため ScalableCore Unit には柔軟に回路構成を変更することが可能な FPGA を採用する．

単一の FPGA チップに搭載しきれない規模のシステム評価環境を構築する場合，選択肢として，大容量の FPGA を少数接続する，小容量の FPGA を多数接続する，の 2 通りが考えられる．もちろん，大容量の FPGA を多数接続することも可能であるが，必要以上の容量の FPGA を用いることはコストの点で現実的ではない．

大容量の FPGA を少数接続する場合，高速で動作をするというメリットをもつ一方で，1 つのユニットが高価になるというデメリットが考えられる．小容量の FPGA を多数接続する場合，動作速度は大容量 FPGA を少数用いたシステムに劣ると考えられる一方，1 つのユニットあたりでは安価となり導入しやすい，というメリットがある．また，大容量の FPGA を用いた場合，1 つの FPGA チップに多数のコアを搭載する都合上，メモリのポート数がネックとなる，1 つのユニットに対して大きなメモリを搭載する必要があり，コストや物理的なサイズが大きくなってしまう可能性がある．小容量の FPGA を用いた場合，1 つの FPGA チップあたりのコア数が少ない為，システム全体の設計をシンプルにすることができる．

以上を踏まえて，我々は小規模の FPGA を多く接続する ScalableCore システムを開発している．

5.2 基板の製作

図 5 は，制作した ScalableCore Board である．図中左側の Board は，左方向への通信ができない代わりに，DC5V の電源および MMC(MultiMedia Card) が接続でき，システ

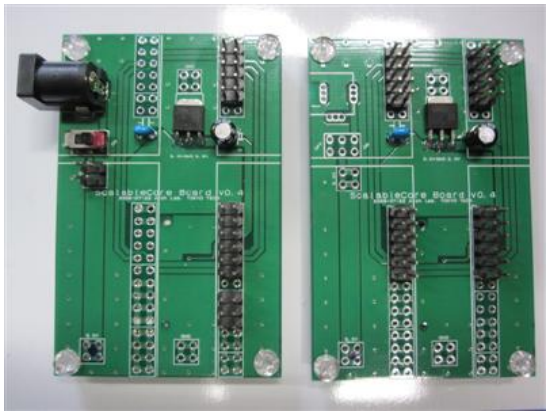

図 5 制作した ScalableCore Board．左が電源 ·MMC リーダが実装されている最左端用，右がその他用の実装．

図 6 制作した ScalableCore Unit． FPGA チップは Xilinx XC3S500E

ム中では最左端に配置する．右側の Board は，上下左右 4 方向との通信が可能なものであり，最左端以外に配置される．電力は最左端では電源から，それ以外では後述の Unit を通じて DC5V が供給され，Board 上の 3 端子レギュレータで DC3.3V に降圧され FPGA に供給されている．

図 6 は，制作した ScalableCore Unit である．Xilinx Spartan 3E の XC3S500E(50 万

図 7 Board と Unit を接続し，4 × 4 の ScalableCore システムとしたところ．各 board 裏についている磁石を利用し，ホワイトボードを用いて縦に設置している．

ゲート)，512KB の SRAM を搭載する．DC5V 電圧を横方向に供給するワイヤの役目も担っている．

図 7 に，実際に Board と Unit を接続した状態を示す．本実装では Board の裏に磁石を実装し，図のようにホワイトボードに貼り付けることで占有面積を小さくしている．

これらのプリント基板は全て 2 層で実装されており，そのコストはそれぞれ 1 枚あたり 300 円程度，実装費を含めたコストは Unit1 枚およそ 4000 円，Board1 枚約 500 円ほどであった．

5.3 モジュール構成

本実装では，各 ScalableCore Unit に M-Core アーキテクチャの計算ノードを実装し，適切な数を ScalableCore Board を用いて接続することで M-Core アーキテクチャのプロトタイプを構成する．各 Unit には識別子 ID が割り振られており，システム起動時に設定される．

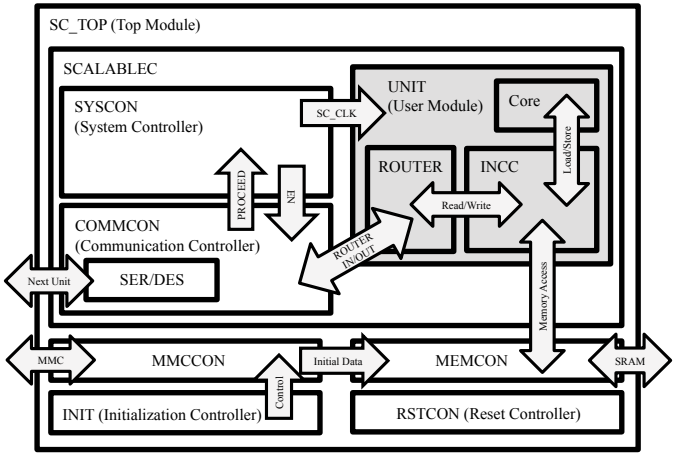

図 8 ScalableCore Unit に実装するモジュールの構成．モジュール UNIT 内に，ユーザモジュールである M-Core の計算ノードの Core, ROUTER, INCC などが実装され，System Controller から供給されるクロックに同期して動作する．

ここで計算ノードなどのユーザーが定義する，シミュレーション対象のモジュールを**ユーザモジュール**，ScalableCore システムを実現する上で必要となるモジュールを **ScalableCore モジュール**と呼ぶことにする．図 8 にそれぞれの ScalableCore Unit に実装するモジュールの構成を示す．また，図 9 は ScalableCore Unit 内モジュールの階層図である．図中の UNIT がユーザモジュールに相当し，それ以外のモジュールが ScalableCore モジュールに相当する．シミュレーション対象のモジュールはユーザモジュールとして構成し，ScalableCore モジュールとは分離されている．また，ScalableCore モジュール内の SER/DES や MEMCON などはシステムの実装に合わせて変更が可能である．

ScalableCore システムをソフトウェアシミュレータと同等に cycle-accurate に動作させるキーとなるのが Communication Controller と System Controller である．これらが連携して隣接 Unit 間と処理の同期を行っている．

本実装ではユーザモジュールとして，M-Core アーキテクチャの計算ノード内の Core, ROUTER および INCC を実装している．ノードメモリには，試作した ScalableCore Unit に実装されている SRAM を用いることにする．本実装の INCC はメモリコントローラの役目も担っており，Core，ROUTER からのメモリアクセスはすべて INCC を経由し行われる．メモリポート数の都合上，INCC のメモリコントローラ部分は Core，ROUTER などの 4 倍速で動作させる．

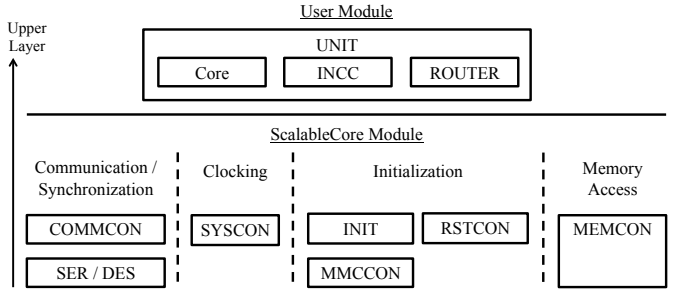

図 9 ScalableCore Unit の階層図．シミュレーション対象はユーザモジュールの階層に割り当てられる．ScalableCore システムを構成する為に必要なモジュールは ScalableCore モジュールとして実装し，MMCCON や SER/DES などの低い階層のモジュールは必要に応じて変更できる．

以下に，各 ScalableCore モジュールがどのような役割を果たしているかを述べる．

- **Reset Controller (RSTCON)**
  外部からの Reset 入力と，後述する Initialization Controller の初期化完了に合わせて Reset 信号を制御し，各モジュールに供給するモジュールである．
- **Memory Controller (MEMCON)**
  各ノードメモリには試作した ScalableCore Unit に実装されているデータバス幅 8bit の SRAM を利用する．INCC からのアクセスは 32bit 幅で行われるため，Memory Controller は SRAM の制御とデータバス幅の変換を行っている．
- **Initialization Controller (INIT)·MMC Controller (MMCCON)**
  M-Core のソフトウェアシミュレータである SimMc は各ノードメモリにデータが配置された状態からシミュレーションを開始する．本実装では，Unit(1,1) の SRAM の初期イメージを MMC(MultiMedia Card) から読み出し，そこからソフトウェアによるプログラムローダで他の Unit にデータおよびプログラムを転送することで，SimMc に近い環境を実現している．その為に，ScalableCore Board に差し込まれた MMC から，メモリの初期イメージを SRAM へ転送するハードウェアが組み込まれている．
  MMC Controller は MMC のコマンド初期化やデータを読み書きができる．そして，Initialization Controller が MMC Controller を制御し，MMC のある特定のアドレス

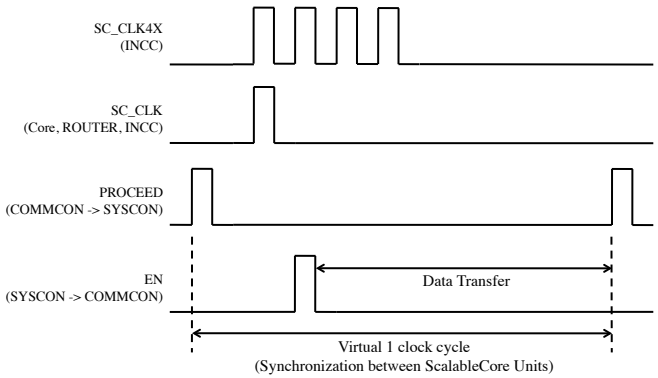

図 10 System Controller および Communication Controller が生成する信号のタイミングチャート．PROCEED は Communication Controller が隣接 ScalableCore Unit 間の同期が完了したことを示す信号，EN は System Controller が隣接する ScalableCore Unit へのデータ転送を Communication Controller に開始させる信号である．

からメモリ初期イメージを読み出し，SRAM に転送している．

- **System Controller (SYSCON)**
  System Controller はユーザモジュールにクロック信号を供給するモジュールである．図 10 に System Controller と，後述する Communication Controller が生成する信号のタイミングチャートを示す．SC_CLK1X と SC_CLK4X がユーザモジュールに供給されるクロック信号である．
  System Controller は，Communication Controller が生成する，隣接 ScalableCore Unit 間との同期完了を示す信号の PROCEED に同期して，ユーザモジュールに提供するクロック信号を生成する．本実装では，INCC のメモリポート数の都合上，別系統のクロックを INCC が必要とするため，System Controller は 2 系統のクロックを生成している．System Controller は隣接 ScalableCore Unit への転送すべきデータが揃うタイミングで EN を 1 にし，Communication Controller にデータ転送を開始させる．
- **Communication Controller (COMMCON)**
  Communication Controller は隣接 Unit との通信および処理の同期を行う．各 Unit の識別子の決定と，隣接 Unit の有無の検出はシステム起動時実行される．詳しくは，節で述べる．ScalableCore システムで各コアが正しいデータを授受するためには，処理に用いるデータの世代が隣接 Unit 間で同じである必要がある．すべてのコアでデータ

```
wait until SYSCON.EN = '1'
for i in [N, E, W, S]{
   send_data_to(i)
}
for i in [N, E, W, S]{
   recv_data_from_with_block(i) if exist?(i)
}
PROCEED <= '1'
```

図 11 ローカル同期方式による隣接コア間でデータの世代を保証する擬似コード

の世代を管理する変数を共有することは現実的でないため隣接するコア間で各コアが自律的に同期を取るローカル同期方式を採用する．図 11 に，ローカル同期方式の擬似コードを示す．
まず，各コアは，SYSCON(System Controller) の出力する信号 EN が 1 になるタイミングで，隣接するユニットへデータを送信する．データの送信が完了すると，次に隣接コアからのデータを受信する．ここでデータ受信処理”recv_data_from_with_block”では，データの受信をブロックする．また，存在しないユニットからの受信処理は実行しない．すべての隣接ユニットからデータを受信できた場合，PROCEED を’1’ にし，ユニットに次の処理への移行を許可する．次の処理へユニットが移行した時点で隣接コアと最新の世代のデータ授受が完了していることは，この一連の処理により，保証される．なお，受信バッファと送信バッファは独立でありデータをそれぞれ上書きすることはない．
ここで，send_data_to および，recv_data_from_with_block 内で通信を行うモジュールは任意に選択することができる．本稿執筆時点での実装では，スタート/ストップビット各 1bit，データ本体 40bit の 1bit 調歩同期通信であり，高速化のために LVDS やロケット I/O などを用いることを検討している．

5.4 システム起動からアプリケーション実行まで

ここでは，システムを起動してから，実際にシミュレーション用のアプリケーションが実行されるまでの流れについて述べる．

ScalableCore システム中の各 ScalableCore Unit は，初期化からアプリケーション実行開始までに，以下の 5 つのフェーズを経ている．

(1) **SRAM 初期化**:

MMC から各 Unit の SRAM の初期イメージを読み込む

(2) **ID 設定コマンド待機**:
Unit のスタートボタンが押される，または隣接 Unit から ID 設定コマンドが転送されてくるのを待機

(3) **ID 設定コマンド・レスポンス発行**:
隣接 Unit に ID 設定コマンドを発行し，合わせて自らが存在することを隣接 Unit に知らせるレスポンスを発行

(4) **レスポンス待機**:
隣接 Unit からレスポンスコマンドが転送されるのを一定時間待機

(5) **実行**:
ユーザモジュールにクロック供給し，シミュレーションを開始する

(1) の SRAM 初期化は，MMC Controller と Initialization Controller によって行われる．初期化は MMC が挿入された Unit のみが行われ，MMC が挿入されていない Unit はタイムアウトの後，次の ID 設定コマンド待機のフェーズに移行する．

ScalableCore システムの各 Unit には ID という識別子が割り振られており，(x,y) の形式で表される．最も左上の Unit を (1,1) として，右方向に進むと x の値が 1 増え，下方向に進むと y の値が 1 増える．ID は (2)〜(4) のフェーズを経て，シミュレーション開始時に決定される．ID の設定は図 12 のように，Tree 状に行われる．この ID は，ScalableCore システム上に実装するユーザモジュールで，識別子として扱うことを想定している．本実装では，各 Unit に実装した計算ノードの ID として用いている．

Unit に取り付けられたスタートボタンを押すと，その Unit の ID は (1,1) に設定され，(3) の ID 設定コマンド・レスポンス発行のフェーズに移行する．そして，Communication Controller が隣接 Unit に ID 設定コマンドを発行する．ID 設定コマンドは各 Unit に実装されたシリアル通信機構，ScalableCore Board を経由し隣接 Unit に転送される．そして，ID 設定コマンドを受信した Unit は (2) の待機フェーズから (3) の ID 設定コマンド・レスポンス発行フェーズに移行する．そして，ID の設定が図 12 に示すように Tree 状にシステム全体で行われる．

(3) の ID 設定コマンド・レスポンス発行のフェーズが完了した Unit は (4) のレスポンス待機のフェーズに移行する．このフェーズでは，送信されれてくるレスポンスを元に，Communication Controller は隣接 Unit が存在するかどうかを判断する．判断結果は Communication Controller 内の 4 ビットのレジスタに格納され，シミュレーション実行時に処理と通信の待ち合わせに使用する．このフェーズは十分な時間を待機した後タイムアウトし，(5) の実行フェーズに移行する．

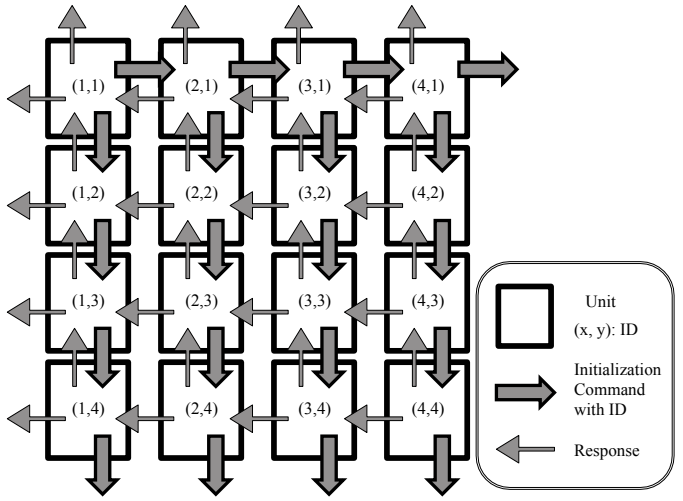

図 12 ID 設定コマンドの方向とそれに対するレスポンスコマンドの方向．ID は Unit(1,1) を基準に Tree 状に設定される．レスポンスは上方向と左方向に送信し，隣接 Unit に自らの存在を知らせる．

そして，(5) の実行フェーズで System Controller がユーザモジュールにクロックの供給を開始し，シミュレーションを開始する．そして前述の通り，Communication Controller と System Controller が協調動作し，隣接 Unit 間で処理と通信の同期をとりながらシミュレーションを進めていく．

5.5 動　　作

原稿執筆時現在，ScalableCore システム v0.6.2 が動作している．v0.6.2 では M-Core 用のテストアプリケーションの Quick Sort や Node(1,1) に対してそれ以外ノードが DMA PUT を行うアプリケーション，Equation Solver Kernel などが動作している．図 13 は 4×3 の構成で Quick Sort を実行したときの様子である．

まず初期化の段階で，Unit(1,1) は実行バイナリを MMC から SRAM に読み込む．そして，Unit(1,1) に実装された計算ノード (1,1) が，そのバイナリを実行し，ソフトウェアで構成されたプログラムローダで計算ノード (1,1) から他のノードに DMA 転送を用いてプログラムを配布することで動作している．

図 13 4 の ScalableCore システムで Quick Sort が実行されている様子．左がマスターコア (1,1) の出力，右がワーカコア (4,3) の出力である

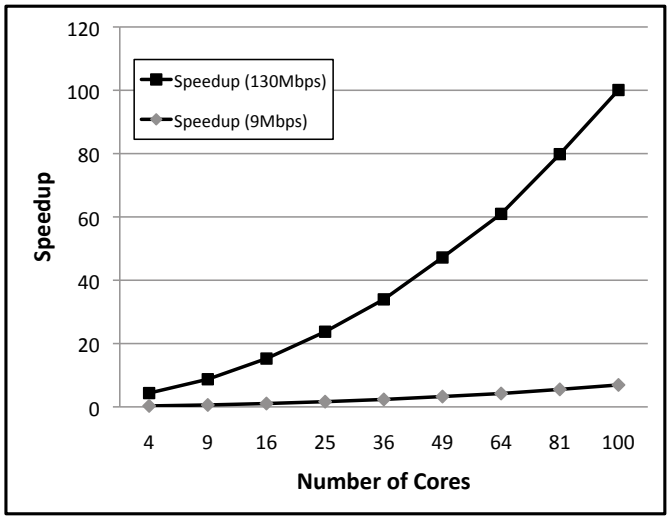

図 14 ScalableCore v0.6.2 によるスピードアップの見積もり

5.6 評　　価

ScalableCore システム v0.6.2 を元に，シミュレーション速度向上の見積もりを行った．

図 14 に SimMc に対するシミュレーションの高速化の見積もりを示す．Unit 間の通信速度はシステム全体の動作速度を決める重要な要因である．そして，ScalableCore v0.6.2 における Unit 間の通信速度は 9Mbps と低速である．この通信速度の場合，図中の下のグラフに示すように，100 コアのシミュレーションでも約 7 倍の高速化で留まってしまう．

図中の上のラインは，仮に通信が 130Mbps で行えたとした場合のスピードアップの見積もりである．130Mbps は，LVDS などを用いたより高速な通信機構を実装することで，十分に実現可能である伝送速度である．130Mbps での伝送ができれば，100 コアのシミュレーションにおいて約 100 倍の高速化が見込める．伝送速度の向上の為には，高速な通信機構の実装が必須と言える．

6. ま と め

メニーコアプロセッサに関する研究 · 開発を効率的に行う為には，様々なアイディアを迅速に検証できる環境が必要となる．我々は，ハードウェアによるシミュレーション環境 ScalableCore を提案している．これは，シミュレーションノード (ScalableCore Unit) を，4 方向に拡張可能な共通の接続インターフェース (ScalableCore Board) を用いて接続することで，高い拡張性を実現する．本論文では，小容量の FPGA を複数用いた ScalableCore システムの実装方法を検討した．この実装方法により，メニーコアプロセッサ内の各コアを各 FPGA に対応づけることで，ハードウェアによるシミュレーション環境の実装で問題となる複雑さを軽減させることができる．

初期評価として，ScalableCore Unit と ScalableCore Board を試作し，それらを用いてメニーコアアーキテクチャM-Core の実装を ScalableCore システム上に行っている．システムを構成する各 ScalableCore Unit 間の処理と通信の同期を行うために，Communication Controller と System Controller を実装し，ソフトウェアシミュレータと同等の cycle-accurate 性を確保した．内部構成は，ユーザモジュールと ScalableCore モジュールの大きく 2 つに分けられており，シミュレーション対象とそれをサポートするモジュールの分離が行われている．実装したシステムにおいては，100 コアのシミュレーションで，約 7 倍程度の高速化であれば可能であるが，Unit 間の通信機構をより高速なものに変更することで，100 コアのシミュレーションで約 100 倍の高速化と，大幅な高速化が見込める．

今後の課題を述べる．まず，ScalableCore システム上に M-Core アーキテクチャを全実装することを目指す．現在の実装 v0.6.2 では，DMA 転送の機能が制限されているため，その実装が必要となる．さらには，外部割り込みなどの処理を行うオペレーションノードの実装を行う．これは Unit(1,1) に専用のマイコンボードなどを接続することで対応することを検討している．加えて，オフチップのメインメモリを扱うためには，メモリノードを実装する必要がある．

高速化の点では，SimMc 比で 100 倍の高速化を目指し，シリアル転送を高速化する必要がある．さらに動作速度や消費電力，価格について詳細な評価を行う必要がある．また，使

いやすいシミュレーション環境とする為に，任意のタイミングでデバッグ情報を出力する機構の実装を検討する必要がある．合わせて，コンフィグレーション作業を効率化する為に，まとめて複数の FPGA をコンフィギュレーションする機構の実装の検討をしている．

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) Martin, M. M.K., Sorin, D.J., Beckmann, B.M., Marty, M.R., Xu, M., Alameldeen, A.R., Moore, K.E., Hill, M.D. and Wood, D.A.: Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset, *SIGARCH Comput. Archit. News*, Vol.33, No.4, pp.92–99 (2005).
2) Binkert, N.L., Dreslinski, R.G., Hsu, L.R., Lim, K.T., Saidi, A.G. and Reinhardt, S.K.: The M5 Simulator: Modeling Networked Systems, *IEEE Micro*, Vol.26, No.4, pp.52–60 (2006).
3) Monchiero, M., Ahn, J., Falcon, A., Ortega, D. and Faraboschi, P.: How to Simulate 1000 Cores, Workshop on Design, Architecture and Simulation of Chip Multiprocessors (dasCMP ' 08) (2008).
4) Argollo, E., Falcón, A., Faraboschi, P., Monchiero, M. and Ortega, D.: COTSon: infrastructure for full system simulation, *SIGOPS Oper. Syst. Rev.*, Vol.43, No.1, pp.52–61 (2009).
5) Öner, K., Barroso, L.A., Iman, S., Jeong, J., Ramamurthy, K. and Dubois, M.: The design of RPM: an FPGA-based multiprocessor emulator, *FPGA '95: Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*, New York, NY, USA, ACM, pp.60–66 (1995).
6) Davis, J.D., Richardson, S.E., Charitsis, C. and Olukotun, K.: A chip prototyping substrate: the flexible architecture for simulation and testing (FAST), *SIGARCH Comput. Archit. News*, Vol.33, No.4, pp.34–43 (2005).
7) Wee, S., Casper, J., Njoroge, N., Tesylar, Y., Ge, D., Kozyrakis, C. and Olukotun, K.: A practical FPGA-based framework for novel CMP research, *FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, New York, NY, USA, ACM, pp.116–125 (2007).
8) Chung, E.S., Papamichael, M.K., Nurvitadhi, E., Hoe, J.C., Mai, K. and Falsafi, B.: ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs, *ACM Trans. Reconfigurable Technol. Syst.*, Vol.2, No.2, pp.1–32 (2009).
9) http://ramp.eecs.berkeley.edu/: RAMP: Research Accelerator for Multiple Processors.
10) 植原昂，佐藤真平，高前田伸也，渡邉伸平，吉瀬謙二：メニーコアプロセッサの HW/SW 研究開発を加速する実用的な基盤環境，先進的計算基盤システムシンポジウム SACSIS2009 論文集 (2009).