情報処理学会論文誌 Vol. 50 No. 11 2665–2676 (Nov. 2009)

# 教育・研究に有用な MIPS システムシミュレータ SimMips

藤枝直輝[†1] 渡邉伸平[†1] 吉瀬謙二[†1]

近年では組み込みシステム教育への要求が高まっており，この要求を満たすためにはシンプルなシステムシミュレータの利用がおおいに有用である．本論文では，本研究室で開発している MIPS システムシミュレータ SimMips のコンセプトと実装について述べたあと，講義の題材として用いた実例をあげ，教育用途への有用性を示す．また，SimMips をインフラストラクチャとして用いた応用例として，シンプルな組み込みシステム Simplem，およびメニーコアアーキテクチャのシミュレータ SimMc の 2 例をあげ，研究用途にも適応性が高いことを示す．

# A MIPS System Simulator SimMips for Education and Research of Computer Science

Naoki Fujieda,[†1] Shimpei Watanabe[†1] and Kenji Kise[†1]

In recent years, there has been increasing demand for education of embedded system. Using a simply coded system simulator is an efficient way to meet this demand. In this paper, we describe the concept and impremention of SimMips and take an example of using it as a marerial of class, showing the suitability for educational use. We also clarify the flexibility of SimMips for research use by showing two applications, a simple embedded system Simplem and a many-core simulator SimMc, where SimMips is used as an infrastracture.

†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

## 1. はじめに

プロセッサアーキテクチャの教育・研究のツールとして様々なプロセッサシミュレータが利用されている[1),2)]．このようなプロセッサシミュレータに加えて，入出力などを含めた計算機システム全体に対するシミュレーションを行うシステムシミュレータもしばしば用いられている．

一方近年では，組み込みシステムに対する需要の増加にともない，組み込みシステム教育への要求が高まっている．この要求を満たすためには，シンプルで理解しやすく，変更や拡張が容易なシステムシミュレータの利用がおおいに有用である．このようなシステムシミュレータを用いてシステムを理解し，またそれに変更を加えることにより，より体験的に組み込みシステムに対する理解を深めることが可能となる．また，シンプルさと拡張性の高さは，これを研究のインフラストラクチャとして用いる際にも重要である．

我々は，教育・研究に有用なシステムシミュレータとして，コードのシンプルさを重視した，MIPS32 命令セット[3)] のプロセッサを含むシステムシミュレータ **SimMips** の開発を行っている．SimMips は C++を用いて 4,500 行程度と比較的少ないコード量で記述されているにもかかわらず，コードを修正していない最新の版の Linux が動作する．

本論文ではまず SimMips のコンセプトと実装について述べる．システムのハードウェア構成を意識した設計と，可読性を重視したプログラミングにより，理解しやすく学びやすい構造をなしていることを明らかにする．次に，コードのシンプルさを損なうような過度の最適化を避けているにもかかわらず，十分実用的な速度でシミュレーションが可能であることを示す．さらに SimMips を講義の題材として用いた実例をあげ，教育目的への有用性を示す．

また応用事例として，SimMips の一部を Verilog HDL に移植したソフトプロセッサを含むシンプルな組み込みシステム **Simplem**，および計算コア部分に SimMips を組み込んだメニーコアプロセッサ・シミュレータである **SimMc** についても述べる．高い可読性と拡張性を活かすことで，こうした応用を短期間で構築可能なことを示し，教育用途ばかりでなく研究用途においても高い適応性を持つことを明らかにする．

本論文の構成を述べる．2 章で開発に至った背景と SimMips のコンセプトについて述べる．3 章では設計と実装について解説し，プロセッサシミュレータからシステムシミュレータを構築するプロセスについて述べる．4 章ではシミュレーションが現実的な速度で動作することを示し，講義の題材として用いた結果も交えて SimMips の有用性を検討する．5 章

で応用例である Simplem および SimMc について述べ，6 章で本論文をまとめる．

## 2. 背景とコンセプト

教育・研究のツールとして様々なプロセッサシミュレータが用いられている．たとえば，MIPS シミュレータとしては SPIM[4] が有名である．SPIM にはテキスト形式のアセンブリを直接実行できるために，クロス開発環境が必要ないという利点がある．クロス開発環境の構築は労力を要する作業であったが，近年では Buildroot[5] などのツールにより手軽に構築できるようになっている．このため，アセンブリを直接実行できることの利点はそれほど大きくなく，逆に ELF 形式の実行ファイルが読み込めないことが欠点となる．

システムシミュレータとしては Simics[6]，M5[7]，QEMU[8] などがある．これらはいずれも数多くのプラットフォームに対応し，高速で動作することを主眼としており，動的コード変換などの手法もしばしば用いられる．このため，教育・研究において重要となるシンプルさや可読性を欠いている．

シングルプラットフォームのシステムシミュレータとしては，x86 アーキテクチャを対象とした Bochs[9] が有名である．しかしながら，x86 アーキテクチャは複雑で教育用途としては利用しにくい面がある．計算機アーキテクチャなどの講義では MIPS プロセッサを題材として用いることが多い[10]．このため，MIPS アーキテクチャを採用するシンプルなシステムシミュレータの有用性は高い．

SimMips はこれらのシミュレータとは一線を画し，シンプルさと可読性を重視して設計された MIPS システムシミュレータである．我々の知る限り，SimMips は最新の Linux が動作する国産初のオープンソースのシステムシミュレータである．可読性と実行速度との間にはトレードオフがあるが，近年のプロセッサの高速化により，可読性を維持しつつ現実的な速度で動作させることが可能となりつつある．このため我々は，動作速度ではなく可読性を重視することにした．SimMips を用いることにより，プロセッサや OS などを含む計算機システムの教育・開発・研究におおいに利点を発揮する．

## 3. SimMips の実装

SimMips は C++を用いて記述されている．同じくオブジェクト指向型の言語である Java と比較してビット単位の操作を簡潔に記述しやすく，また，Verilog などのハードウェア記述言語よりも柔軟な記述が可能であることから，C++による開発を行うこととした．

**表 1** は SimMips Version 0.5.2 のファイル構成である．ソースコードの行数とファイル名，およびそのファイルが提供する内容を列挙する．2 列目の左側の数字はコメントや空行を含んだ行数である．また，2 列目の右側で括弧の中に示す数字はコメントや空行を含まない行数である．最終行はすべてのファイルの合計行数である．ここに示すように，コード量は約 4,500 行であり，システムシミュレータとしてはとても少ない．

**表 1**　SimMips Version 0.5.2 のファイル構成
Table 1　File organization of SimMips Version 0.5.2.

| ファイル名 | 行数 | | 提供する内容 |
|---|---|---|---|
| define.h | 744 | (618) | クラス・定数などの定義 |
| main.cc | 21 | (10) | main 関数 |
| board.cc | 622 | (518) | シミュレーション環境 |
| memory.cc | 297 | (236) | メインメモリ，メモリコントローラ |
| simloader.cc | 227 | (192) | ELF ファイルローダ |
| mips.cc | 899 | (813) | MIPS 計算コア |
| mipsinst.cc | 767 | (743) | MIPS の命令に関する情報 |
| cp0.cc | 309 | (248) | MIPS システム制御コプロセッサ |
| device.cc | 546 | (456) | I/O などのコントローラ |
| 合計 | 4,432 | (3,834) | |

### 3.1　2 つのモードを提供するシミュレータ

SimMips は ELF 形式の実行ファイルを読み込み，記述されている命令の処理をシミュレートする．SimMips は，OS カーネルを ELF 形式の実行ファイルとして読み込み，Linux などの OS を動作させる **OS モード**と，静的にリンクされたユーザプログラムを動作させる **App モード**との 2 つの実行モードを提供する．後者のモードを提供することで，SimMips をシステムシミュレータとしてだけではなく，シンプルなプロセッサシミュレータとしても利用できる．

シミュレーションにより，通常のアプリケーションや OS の出力に加えて，メモリやレジスタの値，実行された命令のトレース，統計データなどを出力する機能を持つ．

### 3.2　ソフトウェアアーキテクチャ

**図 1** に，SimMips がシミュレートするシステムのハードウェア構成を示す．特に関係の深い部分どうしは矢印で結ばれている．シミュレートするシステム全体のことを Board と呼んでいる．Board はチップ（Chip）と主記憶（MainMemory），オフ・チップの割込みコントローラ（IntController）とシリアル I/O コントローラ（SerialIO）から構成される．Chip には MIPS プロセッサ（Mips）とシステム制御コプロセッサ CP0（MipsCp0），メモリインタフェース（MemoryInterface）が集積されている．入出力にはシリアルコンソール

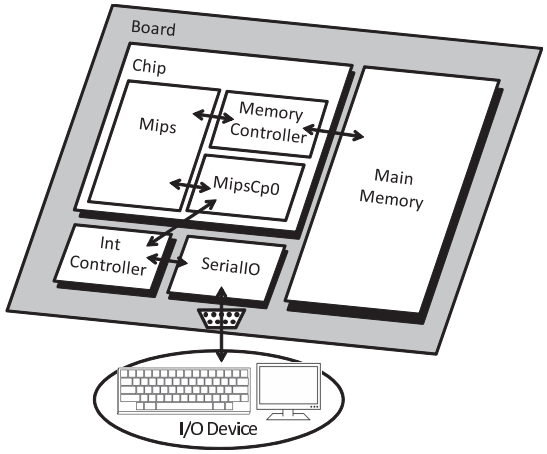


**図 1**　SimMips がシミュレートするシステムのハードウェア構成
Fig. 1　Hardware organization of a system which SimMips simulates.

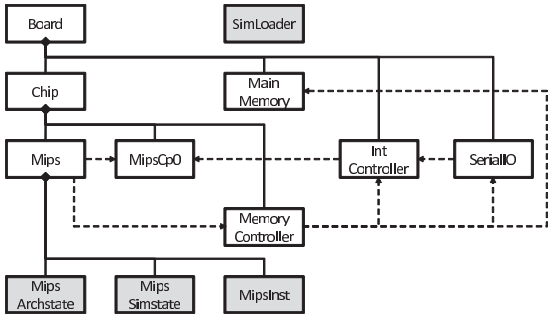


**図 2**　SimMips における主要なオブジェクト（C++）の関係図
Fig. 2　relation among major objects (C++) of SimMips.

を用いるものとする．

**図 2** にこの構成の階層構造を維持しつつ，各ユニットを C++のクラスとして実装するときの主要なオブジェクトの関係図を示す．灰色の背景で示すオブジェクトは，図 1 には存在しないが，可読性の向上のためやシミュレータとしての機能を実現するために追加されたオブジェクトである．図中の実線は，あるクラスの中で別のクラスのオブジェクトが生成される関係を表している．また点線は矢印の元のオブジェクトが，矢印の示す先のオブジェクトを参照していることを表している．たとえば Mips クラスのオブジェクトは Chip クラスによって生成され，MipsCp0 と MemoryController を参照している．なお，SimLoader クラスは初期化の際に利用される ELF ファイルローダである．

OS モードではこれらすべてのユニットが動作している．一方 App モードでは，MipsCp0, IntController，SerialIO の各ユニットは無効化される．

### 3.3　計算コアの実装

**Mips クラス**は MIPS の計算コアを実装したクラスである．通常は 1 サイクルにつき 1 命令を実行する機能レベルの実行モデルをとる．MIPS32 命令セットのうち，浮動小数点演算に関わるものを除いた約 100 種類の命令を実行可能である．このクラスは MipsArchstate クラス，MipsSimstate クラス，MipsInst クラスの 3 つのオブジェクトを生成する．MipsArchstate クラスはレジスタの値などのアーキテクチャの状態を保持し，MipsSimstate クラスは実行した命令など，シミュレータが取得した統計情報を記録する．MipsInst クラスは命令デコードやニーモニックといった命令に関わる情報をまとめたクラスである．これら 3 つのクラスを Mips クラスから切り分けて実装することで，ソースコードの可読性が向上する．

SimMips では，既存のシンプルなプロセッサシミュレータである SimCell [11] と同様に，プロセッサの各ステージを 1 つ，あるいは 2 つのメソッドとして実装している．すなわち，典型的な 5 段パイプラインのプロセッサのステージであるフェッチ，デコード，実行，メモリアクセス，ライトバックの各ステージを `fetch()`（フェッチ），`decode()`，`regfetch()`[⋆1]（デコード），`execute()`（実行），`memsend()`，`memrecieve()`（メモリアクセス），`writeback()`，`setnpc()`[⋆2]（ライトバック）の 8 つのメソッドにより実装している．このような記述方式を用いることにより，複数のステージをまとめて記述するときと比べて，プロセッサのハードウェア構成を意識しやすく，動作の理解への見通しを立てやすい．

以下では実際のコードを用いて，Mips クラスの概要を示す．**図 3** は，MIPS プロセッサの実行ステージにあたる `execute()` メソッドのコードであり，**図 4** は，App モードにおいてシステムコールの代行を行う `syscall()` メソッドのコードである．

`execute()` メソッドでは，それ以前のステージで取得した命令のオペレーションやレジ

⋆1 Register Fetch の略.
⋆2 Set Next Program Counter の略.

```
 1 void Mips::execute()
 2 {
 3   switch (inst->op) {
 4   case ADDU_____:
 5     rrd = rrs + rrt;
 6     break;
 7   case BEQ______:
 8     npc = inst->pc +
 9           (exts32(inst->imm, 16) << 2) + 4;
10     cond = (rrs == rrt);
11     break;
12   case SYSCALL__:
13     if (cp0)
14       exception(EXC_SYSCALL);
15     else
16       syscall();
17     break;
18   case ...
19   }
20 }
```

**図 3**　execute メソッドの実装（一部略）．実行ステージに相当する
Fig. 3　A part of the impremention of execute method, corresponding with execute stage.

スタ値をもとに，適切な処理を行う．たとえば，4〜6 行目の `addu`（ADD Unsigned）命令は，レジスタ `rs`，`rt` の値 (`rrs`, `rrt`) の和 `rrd` をレジスタ `rd` に格納する命令である．また，7〜11 行目の `beq`（Branch EQual）命令は，レジスタ `rs`，`rt` の値が等しいときに限り，命令の即値（`imm`）フィールドによって相対的に指定されたアドレスに分岐させる命令である．

12〜17 行目の `syscall`（SYStem CALL）命令の実装は，OS モードと App モードとで異なっている．App モードでは，16 行目で `syscall()` メソッドを呼び出す．このメソッドはシステムコールに対応する処理を SimMips が代行するもので，図 4 に示すとおり実装されている．システムコールの種類がレジスタ `$v0`，引数がレジスタ `$a0` 以降に記録されており，レジスタ `$v0` の値を見て適切なシステムコールの処理を行う．たとえば `exit` がコールされれば，5 行目でコアを停止状態にすることにより，シミュレーションを終了させる．`write` システムコールの処理は 7〜19 行目に記述されている．8 行目で `write` システムコールが標準出力に対するものかをチェックする．もしそうであれば，指定されたバイト数だけ 10〜15 行目の処理，すなわち MemoryController クラスを経由して MainMemory

```
 1 void Mips::syscall()
 2 {
 3   switch (as->r[REG_V0]) {
 4   case SYS_EXIT:
 5     state = CPU_STOP;
 6     break;
 7   case SYS_WRITE:
 8     if (as->r[REG_A0] == STDOUT_FILENO) {
 9       for (uint i = 0; i < as->r[REG_A2]; i++) {
10         int mcid = mc->enqueue(as->r[REG_A1] + i,
11                                sizeof(char), NULL);
12         if (mcid < 0) break;
13         mc->step();
14         if (mc->inst[mcid].state == MCI_FINISH)
15           putchar((int) mc->inst[mcid].data008);
16       }
17     }
18     as->r[REG_V0] = as->r[REG_A2];
19     as->r[REG_A3] = 0;
20     break;
21   case ...
22   }
23 }
```

**図 4**　syscall メソッドの実装（一部略）．App モードにおいてシステムコールの動作を代行する
Fig. 4　A part of the impremention of syscall method, which substitutes the behavior of system call in App mode.

クラスから 1 文字ずつ読み出し，SimMips の標準出力に書き込む，という処理を行う．最後に，18〜19 行目で返り値をセットして処理を終える．こうしたシステムコールの代行処理は，現在のバージョンでは最小限にとどめているが，`syscall()` メソッドに記述を追加することにより，ファイル操作などの広範なシステムコールに対応するよう拡張できる．

OS モードでは，14 行目の `exception()` メソッドが呼ばれ `SYSCALL` 例外がセットされる．この例外によって PC は例外ハンドラのアドレスへと書き換えられ，シミュレーション対象の OS に記述されたシステムコール・ハンドラが実際のシステムコールの処理を行う．

### 3.4　システムシミュレータ特有部分の実装

本節では，プロセッサシミュレータをシステムシミュレータへと発展させていくための特徴的な部分の実装について述べる．SimMips の開発においては App モードのみ，すなわち前節で述べた計算コアとメインメモリのみを含むシンプルなプロセッサシミュレータを構築し，それをベースとして本節で述べる OS モードの部分を追加するという方法をとっている．

OS の動作するシステムシミュレータには，例外（割込みを含む）の制御，TLB の制御とアドレス変換，I/O コントローラなどの機能が必要となる．以下では OS モードにおいて重要な役割を果たす 4 つのクラスについて説明する．

**MipsCp0 クラス**　MIPS のシステム制御コプロセッサ CP0（コプロセッサ 0）を実装したクラスである．CP0 は制御レジスタや TLB エントリといった情報を保持し，例外の取扱い，TLB の制御，アドレス変換などを行う．MipsCp0 クラスは，Mips クラスから参照されるほかに，一定サイクルごとに内部カウンタを更新している．このカウンタがあらかじめ指定された値に達すると，MIPS 内部のタイマ割込みが発生する．

**MemoryController クラス**　メモリコントローラを実装したクラスである．ロード・ストアは必ずこのクラスを介して行われる．このクラスはあらかじめ設定ファイルによって指定されたメモリマップを持っており，ロード・ストアの際には対象の物理アドレスが渡される．これらをもとに，そのアクセスがどのユニットに対するものかを判別して，適切なユニットにアクセスする．

**IntController クラス**　割込みコントローラの機能を提供するクラスである．内部で Intel 8259 相当の割込みコントローラ 2 個をシミュレートする．接続されたデバイスからの割込み要求（IRQ）を統合して，CP0 へと送信する．

**SerialIO クラス**　シリアル I/O コントローラをシミュレートするクラスである．SerialIO クラスでは，SimMips の標準入力を読み込んでシリアルコンソールの入力とする．同様に，シリアルコンソールの出力は SimMips の標準出力へ書き出される．割込みは IRQ 4 に接続されている．一定サイクルごとに標準入力をチェックし，入力が存在しかつ割込みが有効ならば割込みを発生させる．

Mips クラスとこれらのクラスにより，OS モードの機能を実現する．以下に一例として，シリアルコンソールに入力が現れたことによる割込みが発生したとき，各クラスのオブジェクトがどのように振る舞うかを述べる．SerialIO が入力の存在を検出して，割込みを IntController 経由で MipsCp0 に送る．Mips は例外の発生をチェックしており，もし発生していればそれ以降のステージにおける命令の処理をキャンセルする．MipsCp0 は，自らの制御レジスタに例外の状況や発生理由（ここでの原因は割込みである）といった情報を記録し，特権モードを表すフラグをセットする．その後で例外ハンドラのアドレスを Mips に通知し，Mips はこのアドレスから実行を再開する．

IntController や SerialIO のようにメモリマップド I/O を持ち，MemoryController を介してアクセスされるクラスは，すべてインタフェース MMDevice（Memory Mapped Device）を実装している．これにより，コードの見通しの良さと拡張性の高さを確保している．

## 4. SimMips の有効性

### 4.1　Linux の起動による OS モードの検証

**図 5** に，SimMips で Linux を動作させたときの出力の様子を示す．実行開始から 1 分ほど待つとシェルが起動し，ユーザからの操作を受け付ける状態となる．この状態からコマンドを実行することができる．また，Ctrl-c を入力することでシミュレーションを終了し，実行した命令数などの情報を表示（図 5 では最後の 5 行）する．

SimMips は機能レベルのシミュレータなので，各命令の実行終了ごとにアーキテクチャ・ステートが正しいかどうかを検証すればよい．アーキテクチャ・ステートはメモリの内容なども含むが，これは命令ごとに比較するには大きすぎるため，我々はレジスタファイルとプログラムカウンタの値を検証の対象とした．App モードでは，デバッガ（GDB）においてステップ実行とレジスタ値表示を繰り返して取得したログと，SimMips で命令ごとにアーキテクチャ・ステートを表示させて取得したログとを比較することで検証を行った．検証には Hello World やクイックソート，N クイーンなどを用い，これらのプログラムを実行して終了するまでの数万〜数百万ステップの検証を行った．

OS モードでは割込みの発生タイミングにより決定性が失われるため，先に述べたような厳密な検証作業は行っていない．しかしながら，実行開始直後から最初の割込みが発生するまでの約 300 万命令については App モードと同様の方法で既存シミュレータとの比較・検

```
## SimMips: Simple Computer Simulator of MIPS Version 0.5.2 2009-01-09
Linux version ...
(中略)
Freeing unused kernel memory: 132k freed
Algorithmics/MIPS FPU Emulator v1.5



BusyBox v1.1.3 (...) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

/bin/sh: can't access tty; job control turned off
~ # echo hello
hello
~ #
## interrupt
## cycle count: 1122195456
## inst count: 403379006
## simulation time:   54.616
## mips: 7.386
```

**図 5**　SimMips で Linux を動作させたときのコンソール出力の一部
Fig. 5　A part of the output when running Linux on SimMips.

証を行った．また，各種コントローラについては単体の動作検証を行った．その結果，図 5 に示すようにコマンドを実行できるところまで到達した．デバッグ中は実装に誤りがあり，kernel panic や無限ループで処理が進まない状況が頻発したことから，OS モードは正しく動作していると考えられる．

**図 6** は，先の実行におけるシミュレーション開始からシェルが起動するまでの約 4 億命令間の命令ミックス（Linux）と，クイックソートプログラムの命令ミックス（Quick Sort）とを比較したものである．OS のコードには論理命令やロード命令が多く含まれていることが見てとれる．また，クイックソートはそのアルゴリズムから想像できるように，算術命令や比較命令で約半数を占めている．このように，アプリケーションだけではなく，特権モードを含む統計情報を容易に取得することができることも SimMips を用いる利点の 1 つである．

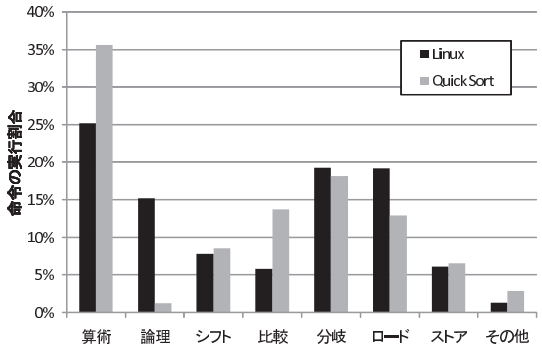


**図 6**　Linux の起動やクイックソートを動作させたときの命令ミックス
Fig. 6　Instruction mix of starting Linux and quick sort.

### 4.2　詳細な動作速度の検討

先に Linux が 1 分程度で起動することを述べた．このことから SimMips は十分現実的な速度で動作するといえる．本節では，SimMips のシミュレーション時間を MIPS 実機と比較することによって，より詳細な動作速度に関する検討を行う．用いた計算機環境は以下のとおりである．

- Xeon X5365（3.0 GHz，Quad-core，4 MB L2）x2
- メインメモリ 16 GB
- Red Hat Enterprise Linux 5（Kernel 2.6.18）
- GCC Version 4.1.2
- Intel Compiler Version 10.0

MIPS Technology 社の評価ボードである Malta ボード[12] を MIPS 実機として用いる．CPU は MIPS 4KEc コア（240 MHz），メインメモリは 128 MB である．ベンチマークプログラムにはクイックソートを用い，SimMips は App モードで動作させる．

**図 7** に，SimMips と実機との動作速度をグラフに示す．横軸は MIPS（Million Instruction Per Second）値を，縦軸は実機（Real）の動作速度と，コンパイラおよび最適化オプションを変化させたときの SimMips の動作速度とをとっている．SimMips は，最適化オプションを-O3 とした gcc を用いたとき最も高速に動作し，-O3 最適化を施した icc や-O2 最適化を施した gcc に対し 1.2 倍，最適化オプションを用いない gcc に対しては約 3 倍高速であ

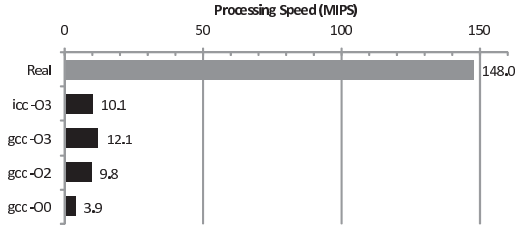


**図 7**　SimMips と実機との動作速度の比較
Fig. 7　Comparison of processing speed between SimMips and real machine.

**表 2**　課題解答者の割合と，課題に要した平均時間
Table 2　Percentage of the students who solve the problem and average time taking for it in hours.

| 課題内容 | 学部 | 修士 |
|---|---|---|
| データ値予測のヒット率測定機構 | 76%（6.7 時間） | 92%（5.2 時間） |
| データキャッシュのヒット率測定機構 | 64%（7.7 時間） | 92%（5.5 時間） |

る．また，最適化オプションを-O3 とした gcc と実機との動作速度を比較すると，SimMips は実機に対し 10 倍強のスローダウンとなっている．しかしながらこの程度のスローダウンであれば，より小さなデータセットを用いるなどの工夫をすることで，十分実用的な時間でのシミュレーションが可能である．

### 4.3　講義の題材としての利用

これまでに SimMips のコンセプトと実装，動作例および実行速度をあげ，教育への有用性を明らかにしてきた．本節ではより実際的な教育用途への適用として，SimMips を講義の題材として利用した実例について述べる．

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，SimMips の実装や動作例を解説した．その後，SimMips に対しデータ値予測[13] やデータキャッシュのヒット率を測定する機構を追加するという課題を課し，レポートを提出させた．その際，課題を解くまでに要した時間の記述を求めた．これらの課題に対し，学部の学生 25 人，修士課程の学生 26 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を持っている．ただし，必ずしも全員が C++によるプログラミングに精通しているわけではない．

**表 2** に学部，および修士課程のそれぞれについて，各課題に対する解答がなされた者の割合と，課題に要した時間の平均を示す．課題を課した時期に差異があるために割合どうしの比較には必ずしも大きな意味はないが，多くの学生がコードを理解し，所望の機能を追加できた．また，所要時間は学部の学生で 6〜8 時間，修士課程の学生で 5〜6 時間であった．他のシミュレータを用いた場合との比較は行っていないが，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．特にコメントやドキュメント，チュートリアルなどの充実を求める意見が最も多く見受けられる．こうした意見も参考に，SimMips をより扱いやすく，コードの理解や拡張がより容易なシミュレータへと改良していくことが求められている．

## 5.　インフラストラクチャとしての SimMips

本章では，SimMips をインフラストラクチャとして用いた 2 つの応用例について述べる．前半ではシンプルな MIPS 組み込みシステム Simplem （**Simple M**ips **Em**bedded system）[14] の開発について述べる．後半では SimMips の拡張性の高さを利用し，計算コア部分に SimMips を組み込んだメニーコアプロセッサ・シミュレータ SimMc の開発をあげる．

### 5.1　シンプルな組み込みシステム Simplem

FPGA（Field Programmable Gate Array）デバイスの大容量化にともない，ソフトプロセッサ（ソフトマクロのマイクロプロセッサ）の利用が広がっている．我々は，SimMips がハードウェアを意識した構成となっていることを利用して，その一部を Verilog HDL に移植することにより，シンプルでカスタム可能なソフトプロセッサ MipsCore を開発した．MipsCore はパイプライン化されていないシンプルなマルチサイクルプロセッサである．

OpenCores[15] では，既存の MIPS ソフトプロセッサとして Plasma[16]，YACC-Yet Another CPU CPU，UCore などが登録されている．このうち有名なものが Plasma である．独自の OS が動作しパイプライン処理を行う Plasma と比較すると，これらを持たない MipsCore は機能面に課題がある．しかし MipsCore は約 1,000 行と，Plasma の約 1,600 行と比較してコンパクトに記述されており，可読性が高くカスタマイズが容易である．

MipsCore におけるステージの分割は，3.3 節で述べた Mips クラスにおけるメソッドの切り分けを踏襲しているが，ライトバックに相当する `writeback` と `setnpc` は 1 つのステージに統合している．すなわち MipsCore は，`fetch`，`decode`，`regfetch`，`execute`，`memsend`，`memrecieve`，`writeback` の 7 ステージから構成されている．各ステージは基本的に 1 サイクルで通過するが，乗除算の `execute` ステージのみ 32 サイクルかけて通過する．したがっ

て，ロード/ストア命令は 7 サイクル，乗除算には 36 サイクル，その他の命令には 5 サイクルを要する．

**図 8** に，SimMips と MipsCore デコードステージのうち，符号なし加算命令 addu をデコードしている部分を示す．各行の記述内容はそれぞれ対応している．まず 3 行目から 13 行目で，命令を各フィールドに分解している．SimMips ではこれをシフト演算とマスク演算により行うが，MipsCore では単純に命令を該当するビット幅に切り分けている．次に 15 行目から 17 行目で初期化作業を行う．SimMips では，latency を変更することにより 1 命令の実行にかかる時間を変更することができるが，MipsCore では前述したとおりの固定のレイテンシを持つ．19 行目以降がデコード操作である．opcode や funct に合った命令を選び，適切な操作を行っている．

MipsCore の実装について述べる．コードの半分以上を占めるデコードおよび実行ステージにおける各命令の記述は，大部分が図 8 のような機械的な置き換えにより実装を行った．その他のステージ，すなわちフェッチ，レジスタフェッチ，メモリアクセス，ライトバックにおいては，レジスタファイルやメモリコントローラとのインタフェースに違いがあるので単純には置き換えができない．しかしながら，これらは SimMips による検証を行いながら実装をすすめることにより，効率的な実装が可能であった．

MipsCore の検証は以下の手順による．MipsCore を Verilog シミュレーションし，各命令の終了時におけるアーキテクチャ・ステートを取得する．次に SimMips で同じアプリケーションを動作させ，同様にアーキテクチャ・ステートを取得する．そしてこれら 2 つを比較することにより，MipsCore と SimMips との動作の等価性を検証する．この作業は簡単なシェルスクリプトを記述して行った．

MipsCore の実装と検証に要した人員は修士課程の学生 1 人と学部学生 2 人，要した期間は計 1 週間であった．ハードウェアの構築には長い時間を要することが多いが，SimMips を利用することにより比較的容易に Verilog HDL への移植が可能であった．

さらに，MipsCore をプロセッサとして用いたシンプルな組み込みシステム Simplem を開発した．Simplem では，MipsCore をアットマークテクノ社の FPGA ボード SUZAKU-S（Spartan3E XC3S1200E，スピードグレード-4）に載せ，MipsCore 上で動いているアプリケーションにより，インテグラル電子社のコマンドインタプリタ液晶 ITC-2432-035H にコマンドを送り，画面表示をする．現在のバージョンでは，アプリケーションはコンフィグファイル内に格納されており，ブロック RAM の初期化の際に書き込む形となっている．FPGA 上には MipsCore に加え，メインメモリや I/O コントローラモジュールなどが実装

```
 1 void MipsInst::decode()                  | /* MipsInst::decode() */
 2 {                                        | always@ ( DATA_IN ) begin
 3   opcode = (ir >> 26) &      0x3f;       |   IDOPCODE = DATA_IN[31:26];
 4   rs     = (ir >> 21) &      0x1f;       |   IDRS     = DATA_IN[25:21];
 5   rt     = (ir >> 16) &      0x1f;       |   IDRT     = DATA_IN[20:16];
 6   rd     = (ir >> 11) &      0x1f;       |   IDRD     = DATA_IN[15:11];
 7   shamt  = (ir >>  6) &      0x1f;       |   IDSHAMT  = DATA_IN[10:6];
 8   funct  =  ir        &      0x3f;       |   IDFUNCT  = DATA_IN[5:0];
 9   imm    =  ir        &    0xffff;       |   IDIMM    = DATA_IN[15:0];
10   addr   =  ir        & 0x3ffffff;       |   IDADDR   = DATA_IN[25:0];
11   code_l = (ir >>  6) &   0xfffff;       |   /* IDCODE_L   not used in MipsCore now
12   code_s = (ir >> 16) &     0x3ff;       |      IDCODE_S   (used in OS mode)        */
13   sel    =  ir        &       0x7;       |   IDSEL    = DATA_IN[2:0];
14                                          |
15   op = UNDEFINED;                        |   IDOP     = ‘UNDEFINED;
16   attr = READ_NONE | WRITE_NONE;         |   IDATTR   = ‘READ_NONE | ‘WRITE_NONE;
17   latency = 1;                           |   /* changing instruction latency is
18                                          |                not supported in MipsCore */
19   switch (opcode) {                      |   case ( IDOPCODE )
20   case 0:                                |   6’d0: begin
21     switch (funct) {                     |     case ( IDFUNCT )
22     ...                                  |     ...
23     case 33:                             |     6’d33: begin
24       op = ADDU_____;                    |       IDOP   = ‘ADDU_____;
25       attr = READ_RS | READ_RT | WRITE_RD; |       IDATTR = ‘READ_RS |‘READ_RT |‘WRITE_RD;
26       break;                             |     end
27     ...                                  |     ...
```

**図 8**　SimMips（左）と MipsCore（右）における表現の差異の例
Fig. 8　An example of the difference of expression between SimMips (left) and MipsCore (right).

**表 3**　Simplem Version 0.9.7 のファイル構成
Table 3　File organization of Simplem Version 0.9.7.

| ファイル名 | 行数 | | 提供する内容 |
|---|---|---|---|
| define.v | 169 | (157) | 定数の定義 |
| MipsR.v | 28 | (23) | トップモジュール |
| MIPSCORE.v | 1,044 | (939) | MipsCore |
| memcon.v | 68 | (55) | メモリコントローラ |
| kbcon.v | 90 | (77) | キーボードコントローラ |
| lcdcon.v | 45 | (41) | 液晶コントローラ |
| freqsyn.v | 49 | (47) | 逓倍器 |
| 合計 | 1,530 | (1,339) | |

されている．これらすべてを合計した記述量は**表 3** に示すとおり 1,500 行強である．なお表 1 と同様に，表 3 では空行やコメントを含む行数と含まない行数とを併記している．

Simplem の論理合成には Xilinx ISE 10.1.03 を用いた．Simplem は Xilinx 社のツールで論理合成をする際に論理の大きさの尺度となるスライスを 1,474 使用する．これは XC3S1200E における使用可能スライスの 16%ほどであり余裕がある．一方ブロック RAM は 26 個使用しており，これは XC3S1200E において使用可能なブロック RAM の 92%にあたる．動作周波数は論理合成ツールによれば最高 127 MHz であるが，実際にはやや余裕をとって 81 MHz で動作させている．

Simplem の検証のために，MipsCore で行ったアーキテクチャ・ステートの比較に加えて，各種コントローラ単体の動作確認，タイマアプリケーションなどを動かし続けることによる振舞いの確認を行った．Simplem の実装と検証に要した期間は計 2 週間であった．

**図 9** に，Simplem 向けのアプリケーションの例として，電源を入れてからの時間を表示するタイマアプリケーションが動作している様子を示す．扱いやすいコード量で，20 cm 四方のハンディな組み込みシステムが動作しているさまは，組み込みシステムの学習者に対するモチベーションを高めることが期待される．

### 5.2　メニーコアプロセッサ・シミュレータ SimMc

近年の高性能汎用プロセッサのトレンドは，スレッドレベルの並列性を活用するマルチコアに移ってきており，特に数十〜数百のコアを集積するメニーコアアーキテクチャ[17),18)]の研究が活発に行われている．このようなメニーコア研究の初期評価の段階では，短期間でシミュレータを構築することが重要となる．我々は，メニーコアアーキテクチャ・シミュレータ SimMc の構築にあたり，計算コア部分に SimMips を組み込んで開発を行っている．これにより，短期間で，しかも重要となるネットワーク部分に集中してメニーコアプロセッサ・シミュレータを開発できるというメリットを享受している．

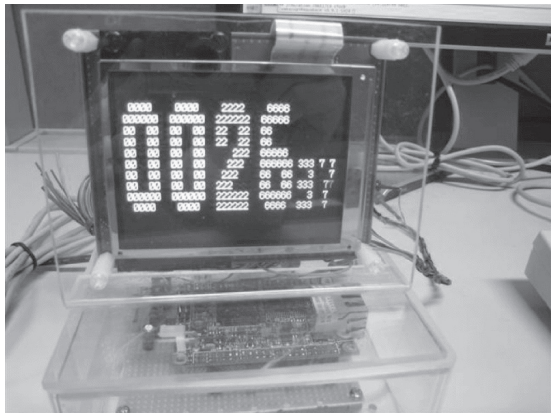

**図 9**　Simplem 上でタイマーアプリケーションが動作している様子
Fig. 9　Timer application running on Simplem.

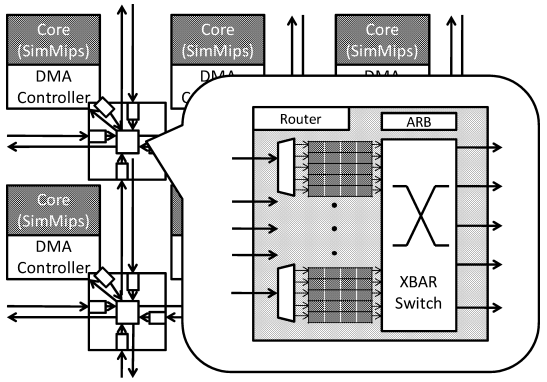


**図 10**　メニーコアアーキテクチャ・シミュレータ SimMc のネットワークモデル
Fig. 10　Network model of many-core architecture simulator SimMc.

**図 10** に SimMc のネットワークモデルを示す．各コアはルータを介してメッシュ結合により上下左右のコアと接続されている．コアには SimMips の App モードをそのまま利用する．DMA コントローラは，SimMips のメインメモリ中にあるいくつかの特定アドレス

**表 4**　SimMc Version 1.0.0 のファイル構成
Table 4　File organization of SimMc Version 1.0.0.

| ファイル名 | 行数 | | 提供する内容 |
|---|---|---|---|
| network.h | 352 | (289) | クラス・定数などの定義 |
| main.cc | 55 | (20) | main 関数 |
| env.cc | 404 | (345) | シミュレーション環境 |
| dmac.cc | 467 | (401) | DMA コントローラ |
| router.cc | 128 | (96) | オンチップルータ |
| inbuf.cc | 86 | (67) | ルータの入力バッファ |
| 合計 | 1,492 | (1,218) | |

を監視している．特定アドレスへの書き込みがあれば，それをコア間の DMA 転送のリクエストとして処理することによりデータの送受信を実現している．ユーザプログラムのために，あらかじめライブラリとして特定アドレスへの書き込みを行う関数を用意しておく．なお，将来は SimMips の OS モードを利用し，OS の動作や入出力などを司るコアを追加することを検討している．このように OS モードを利用することで，複数のプロセスをスケジューリングしながら動作をする現実的な計算機システムの性能評価が可能になる．

SimMc の構築のためには，ネットワーク，ルータ，および DMA コントローラを追加する必要がある．これらのハードウェアとして低機能のユニットを想定しているため，現在のバージョンではこれらの構築に要したコード量は**表 4** に示すとおり 1,500 行程度にとどまっている．また，浮動小数点ベンチマークの動作のために浮動小数点演算ユニットを追加したほかは，SimMc の計算コア部分は SimMips のソースコードを変更することなく利用している．このため，SimMips がアップデートされても，大きな仕様変更がない限り，単純に変更部分のコードを差し替えるだけで計算コア部分を最新のものに保つことができる．このように，SimMips は，シンプルさと可読性を重視しているだけではなく，大規模なメニーコアプロセッサ・シミュレータなどの構成要素としても利用しやすいようにモジュール化されている．

### 5.3　メニーコアシミュレータの動作例

**図 11** は，SimMc においてパケットを視覚化するツール FlitView のスクリーンショットである．SimMc におけるパケットはフリットと呼ばれる単位に分割されている．FlitView では 1 つのフリットに 1 つの値が割り振られており，その移動する様子を視覚的に確認できる．ここでは，16 コアがメッシュ接続されており，すべてのコアが同時に左上のコアに複数のパケットを送信する様子を示している．左上のコアへの経路にパケットが集中し，この部分がボトルネックとなっていることが確認できる．このツールを用いることで，パケットが不意に複製されたり消滅したりするといった，発見の難しいバグを短期間で発見することが容易になる．

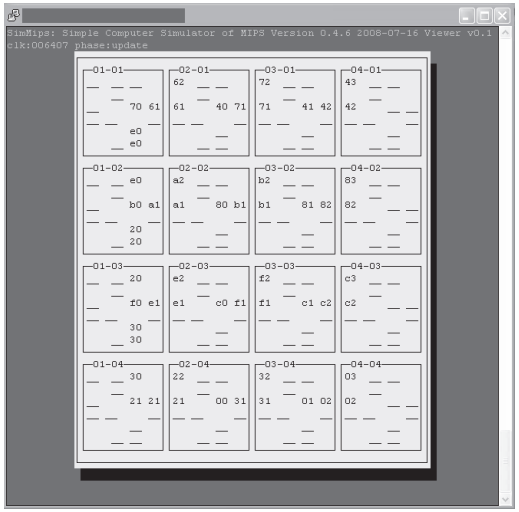


**図 11**　SimMc のパケット可視化ツールのスクリーンショット
Fig. 11　Screenshot of SimMc's packet visualization tool.

次に，2 次元配列のデータにおいて 4 近傍の平均を求める並列計算プログラム Equation Solver Kernel [19] を用いて，SimMc の動作を検証する．このプログラムは，ステップごとに隣接するコアへの通信を行う．

**図 12** に，Equation Solver Kernel の速度向上をまとめる．コア数を 1 から 64 まで変化させ，計算が終了するまでのサイクル数を計測する．1 コア時の性能を基準として，速度向上を求めた．この並列計算プログラムは，近傍のコアのみと通信を行うので，コア数に応じた性能向上を期待できる．図 12 に示すように，64 コアの場合に 60.8 倍という期待される性能向上の結果が得られている．

**図 13** に SimMc の速度低下を示す．グラフにはシミュレーション速度（三角のマーカ，Simulation Speed）と Equation Solver Kernel のシミュレーション時間（四角のマーカ，Simulation Time）の速度低下とを示す．これらはそれぞれ SimMc の 1 コア時の速度・時間で正規化している．また，縦軸・横軸は対数表示である．グラフ左端は，同等の逐次プロ

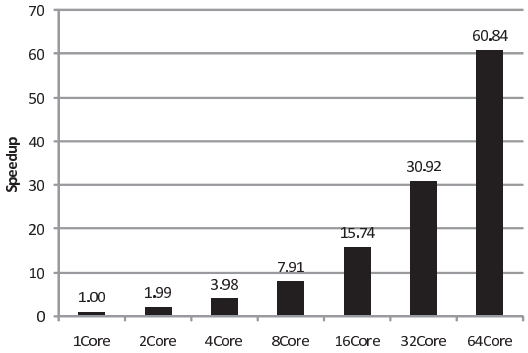


**図 12**　SimMc を用いて測定した Equation Solver Kernel の速度向上
Fig. 12　Speedup of Equation Solver Kernel measured by SimMc.

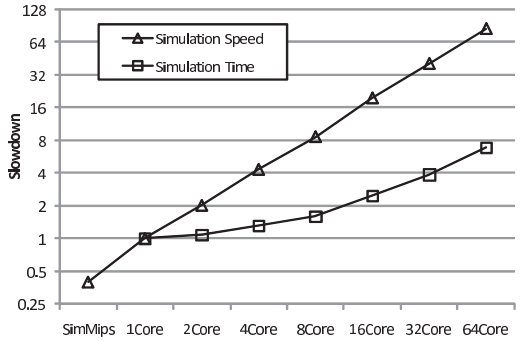


**図 13**　SimMc の速度低下
Fig. 13　Slowdown of SimMc.

グラムを SimMips で動作させたときのシミュレーション速度である．

SimMips と SimMc の 1 コア時とでシミュレーション速度を比較すると，SimMc は SimMips の 4 割ほどの速度に低下している．これはルータを詳細にシミュレートしていることによるオーバヘッドである．また，SimMc においてコア数を増加させると，それに応じてシミュレーション速度は低下する．64 コアのシミュレーションでは SimMips の逐次実行に対して 200 倍，1 コアのシミュレーションに対しても 80 倍もの速度低下を招く．しかしながら，64 コアのシミュレーションそのものに要した時間は 1 コアの場合の約 6.9 倍にとどまっている．各コアの初期化や終了のルーチンはすべてのコアが実行する必要があるため，コア数に比例したシミュレーション時間を要する．一方で，Equation Solver Kernel の計算部分では図 12 で示したように性能向上も同時に達成する．シミュレーション時間は 1 サイクルあたりのシミュレーション時間とシミュレーションに要するサイクル数の積で表される．このため Equation Solver Kernel では，64 コアでもシミュレーションに要した時間は約 6.9 倍にとどまっており，それほど悪化しない．

以上に示したとおり，メニーコア研究に有効なメニーコアシミュレータを，効率良くかつ少ないコード量の追加で構築することができた．このように，SimMips は研究用途でも有用である．

## 6. おわりに

シンプルで拡張性の高いシステムシミュレータは，組み込みシステムの教育やコンピュータシステムの研究に非常に有用である．我々は，MIPS32 命令セットのプロセッサを含むシステムシミュレータ SimMips を開発している．本論文では，SimMips がこうした教育・研究の分野において高い可用性を持つことを明らかにした．SimMips は，2009 年 2 月現在 Version 0.5.2 を Web サイト http://www.arch.cs.titech.ac.jp/SimMips/ で公開している．

今後は，ストレージなどを含むより実用的なシステムが動作するよう SimMips を拡張していくことや，システムやソースコードの理解を助けるためにドキュメントを整理することなどが課題となる．

また，コンピュータシステムの教育のためには，シミュレータだけではなく，OS やハードウェアなどを含めて包括的に学べることが望ましい．我々は，そのためのプラットフォームづくりを進めている．そして，それらの成果を積極的に発信，公開していくことを考えている．

## 参 考 文 献

1) Burger, D. and Austin, T.M.: The Simplescalar Tool Set, Version 2.0, Technical Report CS-TR-1997-1342, University of Wisconsin-Madison (1997).
2) 吉瀬謙二，片桐孝洋，本多弘樹，弓場敏嗣：SimCore/Alpha Functional Simulator の設計と実装，電子情報通信学会論文誌，Vol.J88-D-I, No.2, pp.143–154 (2005).
3) Sweetman, D.: *See MIPS Run Linux 2nd Edition*, Morgan Kaufmann (2006).
4) Larus, J.R.: SPIM S20: A MIPS R2000 Simulator, Technical Report, Computer Sciences Department, University of Wisconsin-Madison (1990).

5) Andersen, E.: Buildroot (online). available from http://buildroot.uclibc.org/
6) Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A. and Werner, B.: Simics: A Full System Simulation Platform, *IEEE Computer*, Vol.35, No.2, pp.50–58 (2002).
7) 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, pp.52–60 (2006).
8) Bellard, F.: QEMU: Open source processor emulator (online). available from http://bellard.org/qemu/
9) Lawton, K.P.: Bochs: A Portable PC Emulator for Unix/X, *Linux Journal* (1996).
10) Patterson, D.A. and Hennessy, J.L.（著），成田光彰（訳）：コンピュータの構成と設計，第 3 版，日経 BP 社 (2006).
11) Sato, S., Fujieda, N., Moriya, A. and Kise, K.: SimCell: A Processor Simulator for Multi-Core Architecture Research, *IPSJ Trans. Advanced Computing Systems*, Vol.2, No.1, pp.146–157 (2009).
12) MIPS Technologies, Inc.: *Malta*（*TM*）*User's Manual Revision 1.05* (2002).
13) Lipasti, M.H., Wilkerson, C.B. and Shen, J.P.: Value locality and load value prediction, *ACM SIGOPS Operating Systems Review*, Vol.30, No.5, pp.138–147 (1996).
14) 渡邉伸平，藤枝直輝，若杉祐太，高前田伸也，森　洋介，吉瀬謙二：MIPS システムシミュレータ SimMips を活用した組込みシステム開発の検討，情報処理学会研究報告 2008-EMB-10，pp.23–28 (2008).
15) OpenCores (online). available from http://www.opencores.org/
16) Rhoads, S.: Plasma CPU (online). available from http://plasmacpu.no-ip.org:8080/
17) Kim, C., Sethumadhavan, S., Govindan, M.S., Ranganathan, N., Gulati, D., Burger, D. and Keckler, S.W.: Composable Lightweight Processors, *Proc. 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.381–394 (2007).
18) 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，pp.39–44 (2008).
19) Culler, D.E., Gupta, A. and Singh, J.P.: *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann (1999).

(平成 21 年 2 月 2 日受付)
(平成 21 年 9 月 11 日採録)

**藤枝　直輝**（学生会員）
2008 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．MieruPC 株式会社代表取締役．プロセッサアーキテクチャ，組み込みシステムに関する研究に従事．

**渡邉　伸平**（学生会員）
2008 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．計算機アーキテクチャ，メニーコアプロセッサアーキテクチャ，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．