並列計算機シミュレータの構築環境

若林 正樹　米田 卓司　小守 継夫　天野 英晴
masaki@am.ics.keio.ac.jp
慶應義塾大学 理工学部

概 要

急激な性能向上を続ける並列計算機の開発には，実装前段階での性能予測が欠かせない．性能予測手法の一つであるソフトウェアシミュレーションは高い柔軟性を持つため，多方面で利用される．並列計算機のシミュレーションを行う場合，特定のアーキテクチャを対象としてシミュレータを実装する．しかし，この方法は異なる対象毎にシミュレータを構築するという実装コストを伴う．本稿では，並列計算機シミュレータの構築支援システムとして，並列計算機シミュレータライブラリ ISIS を提案する．ISIS は並列計算機内部の機能ブロックシミュレータの集合体であり，機能ブロックを繋ぎ合わせることでどのようなアーキテクチャの並列計算機シミュレータでも構築できる．また，旧来の手法と比較してシミュレータの実行時コストを損なわないまま実装コストを軽減できる．ISIS を用いていくつかの並列計算機シミュレータを実装し，その実行速度を評価した結果，実用上充分なシミュレーション速度を得られることが分かった．ISIS はオンチップマルチプロセッサのキャッシュシステムの研究等で実際に用いられ，その評価結果から研究上様々な知見が得られている．

Environment of Multiprocessor Simulator Development

Masaki Wakabayashi　Takuji Komeda　Tsugio Komori　Hideharu Amano
Department of Computer Science, Keio University

Abstract

Performance estimation techniques are essential for researchers and designers of multiprocessors. Software simulation is one of the most effective methods, since there is no limitation on device technology nor hardware configuration. Although lots of software simulators have been developed and used, they must be modified according to the distinct target system. When designers create a new architecture, it is sometimes a cumbersome job. In this paper, architecture independent simulation kit for multiprocessors called ISIS is proposed. It includes various small simulators called units such as processors, buses, memories, caches and I/O devices. ISIS users can build simulators for their original target architectures only connecting units each other. The implementation cost is much reduced almost no overhead in the simulation time. Experimental results of ISIS in research projects including on-chip multiprocessor design are reported.

1　は じ め に

コンピュータ・システムは，現在に至るまで急速な性能向上を続けて来た．新たに計算機を開発する際には，実装前段階で計算機の性能を予測することが成功するための一つの鍵となっている．近年急速に普及している並列計算機の開発においても実装前の性能予測は必要不可欠であり，このための手段として様々な予測手法が使用されている．中でもターゲットマシンの動作をソフトウェアを用いて模倣することで性能予測を行う方法は，実現が容易で柔軟性が高いことから幅広く用いられている．

ソフトウェアシミュレーションには，大きく分けて4つの方式がある: 確率モデルシミュレーション，トレース駆動型シミュレーション，実行駆動型シミュレーション，命令レベルシミュレーションである．評価を行う際には，ターゲットマシンの構成や要求される精度を踏まえた上で，これらのシミュレーショ

ン方式の中から1つを選択してシミュレーションシステムを構築する．Stanford 大学の SimOS[1] など，これまでに数多くのシミュレータが実装され，実用されてきた．

しかし，既存のシステムの多くは特定の並列計算機の性能評価を目的として構築されるため，実装当初に想定していたものと大きく異なるアーキテクチャを評価することは困難である．ゆえに評価対象の並列計算機毎に個別の評価システムを実装する．この評価システム構築のオーバヘッドは研究者にとって大きな負担である．このように，従来のシミュレータは実行時コストのみが重視され，実装コストが軽視されていた．

そこで我々は従来と視点を変えて，並列計算機シミュレータの実装コストを軽減させることを第一目的とし，並列計算機シミュレータの構築支援システムである並列計算機シミュレータライブラリ ISIS を提案する．ISIS は，並列計算機内の個々の機能ブロックをシミュレートする「部品」を多数集めたライブラリである．特定の並列計算機シミュレータを構築する場合は，必要な機能ブロックシミュレータを繋ぎ合わせることで所望のアーキテクチャを実現する．それゆえ，この支援システムはターゲットアーキテクチャに依存することがなく，アーキテクチャの変更によるシミュレータ再実装等のコストを大幅に削減可能である．また，各部品の制御方式と接続方式に余分なオーバヘッドを付加しないため，実行時コストの低減を目的としたシミュレータに匹敵する速度を持つシミュレータを生成できる．ISIS は既に複数の並列計算機研究・開発プロジェクトで利用されており，様々な研究成果が発表されている．

本稿は，2 章で提案するシステムの設計，3 章で実装について述べる．4 章でこのシステムおよび生成したいくつかのシミュレータの性能について述べる．5 章でこのシステムを使用した研究事例を紹介し，6 章で総括する．

2 設 計

並列計算機シミュレータの実装には，ターゲットマシンの構成や規模，要求される精度や実行速度に応じて様々な要素技術が用いられる．本研究の目的である「多様なシミュレータの構築支援システム」の実現には，なるべく多くの技法をサポートし，かつそれらを柔軟に組み合わせ可能にすることが要求される．以下に代表的な要素技術を以下に示す．

シミュレーション方式　シミュレーションを実行する場合の基本方式．精度と実行速度がトレードオフの関係にある．実行速度が高速なものから順に，確率モデルシミュレーション，トレース駆動型シミュレーション [2]，実行駆動型シミュレーション [3]，命令レベルシミュレーション [4] がある．複数のシミュレーション方式を実行時に動的に切り替えることで，評価を行う上で重要な部分を正確に，重要でない部分を高速に実行して精度向上と高速化の両立を図る方法 [1] も用いられる．

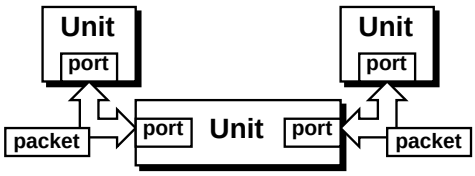

図 1: システムモデル

同期方式　シミュレータ内部の個々の機能ブロックは独立に動作するため，何らかの方式で同期させる必要がある．同期方式にはイベント同期式とクロック同期式がある．前者はイベントの発生頻度が少ない時有利で，後者はそれぞれのクロックで各要素が動作する場合に有利である．並列計算機の場合はイベント発生頻度が多いため，後者が好まれる．

これらの要素技術は評価対象の規模や要求される精度に応じて使い分けられることが望ましい．また，構築されたシミュレータの精度や実行速度，動作環境を束縛するような制御方式は望ましくない．

そこで我々は，小さな機能ブロックのシミュレータを個別実装し，それらの「小さなシミュレータ」をライブラリ形式で並列計算機シミュレータ実装者に提供することにする．各機能ブロックシミュレータには一定の時間間隔毎に状態遷移を行うクロック同期式を用い，自由に相互接続できるように構成する．こうすることで，支援システムを並列計算機アーキテクチャと独立に構築できる．また，このクロックに同期させる制御方式は確率モデルシミュレーション，トレース駆動型シミュレーション，命令レベルシミュレーションのどの方式にも対応可能である．

2.1 システムモデル

高い柔軟性を実現するために，機能ブロック，機能ブロック間の接続，送受信される情報のそれぞれを**ユニット**，**ポート**，**パケット**として抽象化する．図 1 に提案するシステムの概念図を示す．

2.1.1 ユニット

並列計算機内部の個々の機能ブロックシミュレータをユニットと定義する．外部からのクロック入力により，接続されたユニット以外のものとは独立して動作する．ユニット同士の接続には後述するポートを用いる．シミュレータ内の全ユニットの動作順序保証をするために，ユニットへのクロック入力を入力フェーズと出力フェーズに分割する．入力フェーズではユニット外部からユニット内部への情報入力，出力フェーズではその逆の動作のみを行えるものとする．また，大規模な構造を持つシミュレータに対応するためにユニットの階層構造を許す構成を採用し，ユニットの内部実装にもユニットを使用できる

ようにする．

2.1.2　ポート

ユニット間の結合路への入出力端子をポートと定義する．このポート同士を接続することで，ユニットが相互接続される．シミュレータ上での具体的な通信処理はすべてポート内で行う．これにより，ユニットの内部実装から通信処理を抽象化する．ポートの内部実装には，余分なオーバヘッドを招かないような高速な通信手段を用いる．これについては3 章で詳しく述べる．

2.1.3　パケット

ユニット間で送受信される情報をパケットと定義する．送受信される情報の管理はすべてパケット自身が行う．これにより，具体的なデータ処理を意識しないでポートの内部実装を行うことができる．

2.2　精度と実行速度

特定の並列計算機のシミュレータはユニットの集合体として構築されるため，シミュレータの精度と実行速度はそのシミュレータの構成要素をどう選択するかということに強く依存する．精度が必要なら高精度なユニット群を，実行速度が必要なら高速動作するユニット群を選択すれば良い．また，必要な部分だけを高精度にすることで実行速度と精度をある程度両立させることもできる．これらの精度と実行速度の選択権は，支援システムではなくシミュレータ構築者に与えられる．

3　実 装

本章では，2 章で述べたシステムモデルの実装形態と，システムの基幹をなすユニットおよびポート，パケット具体的な実装方法について述べる．また，この基幹部を用いて実装した具体的な機能ブロックシミュレータについて概説する．

3.1　実装形態

ソフトウェアシミュレータが満たすべき条件に，実行時の効率がある．そのため，C 言語あるいはそれに匹敵する実行速度を確保できる言語を選択する必要がある．また，大規模設計に耐え得るプログラミング環境をサポートしている方が望ましい．これらを踏まえて，我々はシステムの実装言語にANSI 標準のC++言語を選択した．2 章で述べたユニット，ポート，パケットそれぞれをクラスとして表現し，クラスライブラリの形でユーザに提供する．

また，クラス間の共通性を基底クラスとして抽出し，クラス階層を形成する．これにより個々のクラス実装のコスト低減を図る．シミュレータ構成要素の最も基本となるユニット，ポート，パケットは，それぞれ個別のクラス階層を形成する．シミュレータ実装者は，必要ならばそれぞれのクラス階層から必要なクラスを取り出し，その派生クラスで意図した機能を実装する．

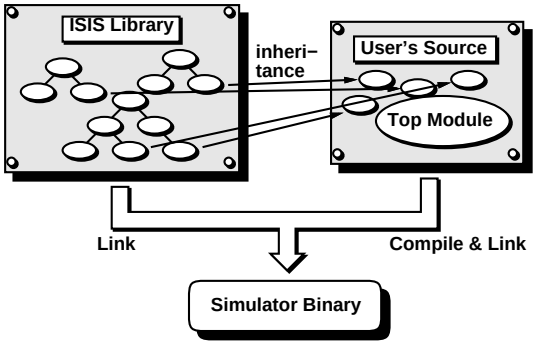

図 2: ライブラリ構成

図 2にライブラリ構成を示す．シミュレータ実装者はライブラリ内のクラス群を用いてシミュレータのトップモジュールを記述する．用意されているクラスならばそのまま使用でき，用意されていないクラスについても継承を用いることで容易に実装を行うことができる．

3.2　システム基本部分

システムの根幹をなすパケット，ポート，ユニットはそれぞれ C++のクラスとして定義し，派生クラスのインタフェースを統一する役割を担う．また，シミュレータの実装コストと実行時コストに大きく影響する具体的なユニットの実装をサポートするために，ユニット内部の構造物をデバイスとして定義する．

3.2.1　パケット

パケットの基底クラスとして，packet クラスを定義する．図 3にpacket クラスの定義を示す．このクラスはパケットの自己複製を行う純粋仮想関数new_packet のみをメンバに持つ抽象クラスである．バス上の信号やルータ内部を通過するフリットがこのクラスの派生クラスになる．パケット内の具体的な情報を一切定義しないことで，派生クラスの実装に高い自由度を与えている．

3.2.2　ポート

ポートの基底クラスとして，port クラスを定義する．図 4にport クラスの定義の一部を示す．例えばバスやルータチップ間のリンクへの入出力端子はこのクラスの派生クラスになる．ポート同士の相互接続と遮断（connect，disconnect），パケットの送受信（put，get），通信制御（have_packet: パケットの有無の確認等）を行う．複数のポートを接続す

```
class packet {
public:
  virtual packet* new_packet() const = 0;
}
```

図 3: packet クラスの定義

ると，そのポート間には仮想的な通信路が自動生成される．この通信路はただ一つのパケットのみを格納することができる．実行速度を向上させるために，パケットの送受信はパケットそのものの複写を行わず，パケットへのポインタの受け渡しを用いる．

3.2.3 ユニット

ユニットの基底クラスとして，unit クラスを定義する．図 5にunit クラスの定義を示す．入力フェーズの状態遷移関数clock_in，出力フェーズの状態遷移関数clock_out，状態初期化関数reset の 3 つの純粋仮想関数のみを持つ抽象クラスである．シミュレータ内のプロセッサやメモリモジュールといった機能ブロックは，このクラスの派生クラスとして実装される．

3.2.4 デバイス

プロセッサ等の大規模ユニットの実装をサポートするために用意された概念である．例えばキャッシュのバッファ部，レジスタファイル，命令バッファなど，状態遷移を持たないような機能ブロックはこのデバイスに分類される．現時点で約 20 のデバイスがライブラリ内に定義されている．

3.3 機能ブロックシミュレータ

シミュレータ実装者の負担を軽減させるには，既に述べたシステムモデルに基づいた具体的な「有用な部品」をライブラリ内に多数取り揃えておかなければならない．既に実装されているこれらの部品について簡単に解説する．

3.3.1 プロセッサ

実在のプロセッサのシミュレータとして，R3000 プロセッサシミュレータが実装されている．図 6にこのシミュレータの内部構造を示す．

このシミュレータは MIPS R3000 プロセッサのほぼ完全なクロックレベルシミュレータであり，5 段命令パイプライン，レジスタファイル，1 次キャッシュ，ライトバッファ，バスインタフェースの動作をクロック単位で正確に模倣する．また，同様に R3000 の浮動小数点演算コプロセッサである R3010 のクロックレベルシミュレータもライブラリ内に定義されている．

3.3.2 バス

バスは，プロセッサやメモリ等，相互結合網の構成要素以外の多くのユニット間の通信を仲介する存在であるので，ポート間に作成される通信路をバスとして使用する．このバスを実現するために，パケットおよびポートの派生クラスを用いる．バスパケットは，実際のバスのアドレス線，データ線，コントロール線上の情報を格納する．バスポートは，バスパケットの送受信やバスのオーナ制御等を行う．

3.3.3 キャッシュ

キャッシュは接続するプロセッサの構成や実装ポリシによってその内部構造がまちまちであるため，ISIS は特定のキャッシュユニットは提供しない．そのかわり，キャッシュを実装するために必要なタグメモリやデータメモリの雛形，バスインタフェースなどの「キャッシュの部品」がデバイスとして提供される．これらの部品は，その構造をシミュレータ実装者が決定するべき部分がクラスのテンプレート引数によって自由に制御できる．

3.3.4 メモリ

メモリシミュレータは，データを記憶するバッファ部と，バッファとバスの制御を行うコントローラ部の二部構成になっている．

バッファ部は内部で動的にページ単位の記憶管理を行っており，シミュレータ記述時に要求されたサイズのメモリを実行開始時には確保しない．シミュレーション実行中にアクセスが行われた時点で，必要なだけの容量のメモリ領域を動的に確保していく．従って，シミュレータ記述時に巨大な記憶領域を要求したとしても，シミュレータ上で実行されるアプリケーションがアクセスしたメモリ容量以上のリソースは消費しない．

コントローラ部はバスからの要求に応じてバッファ部のデータ管理を行う．リードおよびライトアクセスの遅延の個別設定，バースト転送やスプリットトランザクションの有無等，きめ細かな設定が可能である．

3.3.5 ルータ

相互結合網の構成要素となる，ルータのクロックレベルシミュレータである [5]．図 7にルータの内部構造を示す．

様々なターゲットマシンに対応するために，入出力帯域幅，通信遅延，仮想チャネル数，バッファ長等はすべて可変となっている．パケット転送方式は wormhole 方式および virtual cut through 方式をサポートする．ルータ内部には仮想チャネルバッファ，クロスバ，アービタがあり，コントローラによって制

```
class port {
public:
  void put(packet*);
  packet* get();
  void connect(port&);
  void disconnect();
  bool have_packet() const;
}
```

図 4: port クラスの定義（一部）

```
class unit {
public:
  virtual void clock_in() = 0;
  virtual void clock_out() = 0;
  virtual void reset() = 0;
}
```

図 5: unit クラスの定義

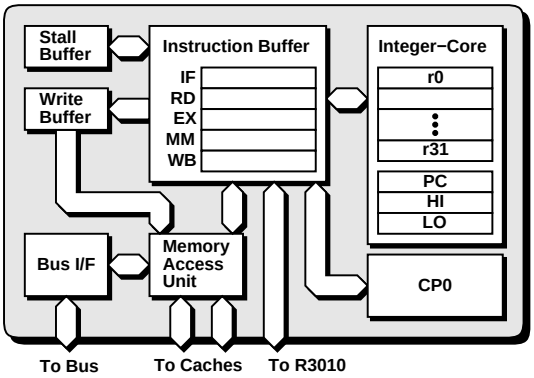

図 6: R3000 シミュレータ

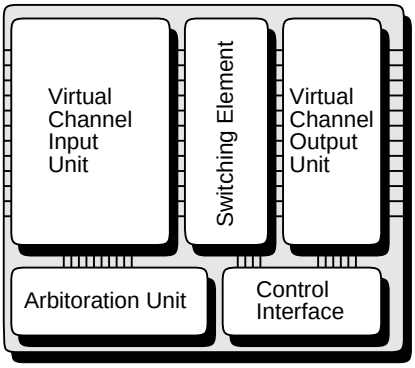

図 7: ルータの構成

御される．コントローラのルーティングアルゴリズムおよび内部制御アルゴリズムはISISが用意する基底ルータクラスでは定義されておらず，派生クラス実装者が定義する．両アルゴリズムの雛形がライブラリ内に多数用意されているので，通常はそれらの中から意図した関数を選択するだけで良い．シミュレータ実装者自身が手続き指向で記述することもできる．

3.3.6 ネットワークインタフェース

ISISで提供しているユニットで構成したノードをルータに接続するために，ネットワークインタフェース [6] が定義されている．このユニットは，ノード内バスとルータ間のパケットの相互変換，フロー制御を行う．また，分散共有メモリをサポートする．これにより，NUMAの命令レベルシミュレーションが可能である．

3.3.7 入出力装置

ターゲットマシン上のアプリケーションがホストマシン上の周辺機器を利用できるように，ファイル入出力や時刻管理等を行うためのユニットが用意されている．このユニットに対して処理要求パケットを送信すると，そのアクセスはシミュレータ外部のホストマシン上のOSに対する処理要求に写像される．

表 1: 動作実績

アーキテクチャ	OS
HP PA-RISC	HP-UX 10.10
Sun Sparc	SunOS 4.1.4 Solaris 2.7
PC/AT compatibles	FreeBSD 3.4 Linux 2.2.14 Solaris 2.7-x86
SGI Origin	IRIX 6.4

表 2: 要求リソース量

ユニット	リソース（bytes）
R3000 プロセッサ	624
R3010 コプロセッサ	1,936
メモリコントローラ	140
I/O その他	2,476
ルータ	140
Network I/F	9,208
ユニプロセッサマシン	6,296

3.4 実装方法

既に述べた通り，ISISの実装にはANSI C++言語を用いた．極力多くの実行環境を確保できるようにするため，C++言語標準のもの以外のライブラリは一切使用していない．また，シミュレータ上で動作するアプリケーションのために，完全なANSI CおよびANSI C++，ANSI FORTRAN 77，FORTRAN 90をサポートする．アプリケーションのサポート環境はGNUの`gcc`と`newlib`を用いている．

現時点のISISは並列動作するシミュレーションをサポートしていないが，そのかわり多数のワークステーションやPC上で動作する．命令レベルシミュレーションといった長いシミュレーション時間を要する評価では，多くのホストマシン上で異なるパラメータの評価を並列実行することで総実行時間を短縮することができる．

4 評 価

まず初めに動作実績について触れる．続いて機能ブロックシミュレータそれぞれの使用リソース量と実行速度，およびそれらを用いて実装した並列計算機シミュレータの性能評価を行い，本稿で提案するシステムを実行性能面で評価する．

4.1 動作実績

表 1に現在までに動作が確認されたアーキテクチャとOSを示す．表 1に含まれていないホストであっても，C++コンパイラがあれば問題なく動作する．

表 3: ホストマシンの諸元

アーキテクチャ	PC/AT compatibles
プロセッサ	Pentium-II(450MHz) × 1
メモリ	128MBytes
OS	FreeBSD 3.4-RELEASE

表 4: ユニットの実行速度

	loop	copy
合計 step 数	10,000,702	5,941,462
総実行時間 [sec]	63.25	34.33
実行時間/step[μsec]	6.325	5.778
実行 step 数/秒	158,113	173,069

4.2 機能ブロックの性能評価

4.2.1 要求リソース量

表 2に，システム内のユニットの要求リソース量を示す．単位はバイトである．表中のリソース量はプログラム中でそれぞれのユニットが要求する主記憶領域サイズを測定したものなので，ユニット間の接続状況や送受信されるパケット数，およびホストマシンと OS によって実際に使用されるリソース量は若干変化する．また，二次記憶等は一切要求しない．なお，表中最下のユニプロセッサマシンは，R3000 プロセッサ，R3010 コプロセッサ，メモリ，I/O 装置を搭載した，ほぼ最小規模の命令レベルユニプロセッサシミュレータである．ルータやネットワークインタフェースは搭載していない．

表 2から，各ユニットの要求リソース量は少なく，この点からは本システムが高いスケーラビリティを有していることが分かる．

実際のシミュレーションを行う際には，シミュレータが使用するキャッシュやメモリ等の容量分のバッファが別途必要となる．ただし，メモリシミュレータについては3.3.4 項で述べた通り，シミュレータ上で実行されるアプリケーションがアクセスしたメモリ容量以上のリソースは消費しない．

4.2.2 実行速度

4.2.1 項で述べた最小規模の命令レベルユニプロセッサシミュレータを用いて，シミュレーション時間を測定した．ただし，なるべく現実的なシミュレーションを行うために，命令/データ合わせて 20k バイトの 1 次キャッシュを追加搭載している．シミュレーションホストマシンの諸元は表 3に示す通りである．

このアーキテクチャ上で 2 つの単純なアプリケーションを実行した．1 つは 100 万回の空ループを実行するもの（loop），もう 1 つは 1Mbytes の領域コピーを行うもの（copy）である．表 4にその実行結果を示す．copy の方がメモリコントローラに対する負荷が高いため，実行速度が若干遅くなっている．この結果から，1 プロセッサの命令レベルシミュレーションを 1 秒間に 16 万から 17 万ステップ実行できることが分かる．この速度は実プロセッサのクロックレベルシミュレータとしては現実的である．

表 5: 命令レベルシミュレーションの負荷

プログラム	問題サイズ
BARNES	512 bodies
FFT	65536 complex doubles
LU	256 × 256 matrix
OCEAN	66 × 66 grid
RADIX	2097152 keys

表 6: 相互結合網の評価条件

パケット転送方式	virtual-cut-through
リンクのバンド幅	1 flit/clock
メッセージ長	6 flit
転送パターン	一様乱数
アクセス生起確率	0.1%
ルーティング	固定ルーティング
総ステップ数	100,000

4.3 シミュレータの性能評価

4.3.1 命令レベルシミュレーション

まず，小規模なマルチプロセッサを精密にシミュレーションする命令レベルシミュレータを構成した場合について評価する．評価に用いた構成は，オンチップマルチプロセッサ上にプロセッサ台数分のポートを持つマルチポートメモリを実装した小規模なマルチプロセッサで，プロセッサコアには R3000 を用いる．この並列計算機の命令レベルシミュレータを ISIS を用いて実装し，その上で実用的な並列アプリケーションを実行した．実行させた並列アプリケーションには，SPLASH2 アプリケーション集 [7] の中から表 5に示すプログラム，問題サイズを選択した．プロセッサ数は 1 から 64 まで変化させた．シミュレーションホストマシンの諸元は先ほどと同じく表 3に示す通りである．

図 8に，この並列計算機の時刻を 1 クロック進めるのに要する時間を示す．グラフから分かるように，この 1 ステップ当たりの実行時間はほぼプロセッサ数に比例し，アプリケーションにあまり左右されない．これは 4 プロセッサの並列計算機であれば 1 秒間に 2 万から 3 万ステップを処理する能力に相当する．

4.3.2 確率モデルシミュレーション

次に，相互結合網を用いた大規模な NUMA 型並列計算機の性能を確率モデルを用いてシミュレーションして測定するためのシミュレータを構築し，評価を行った．各ノードは 2 次元トーラス，ハイパキューブ，多段結合網のいずれかの相互結合網により接続され

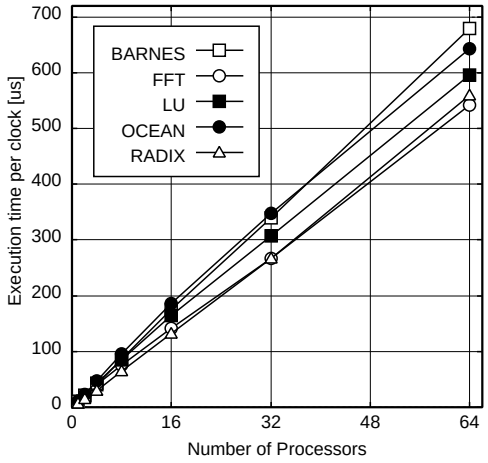

図 8: 命令レベルシミュレータの実行速度

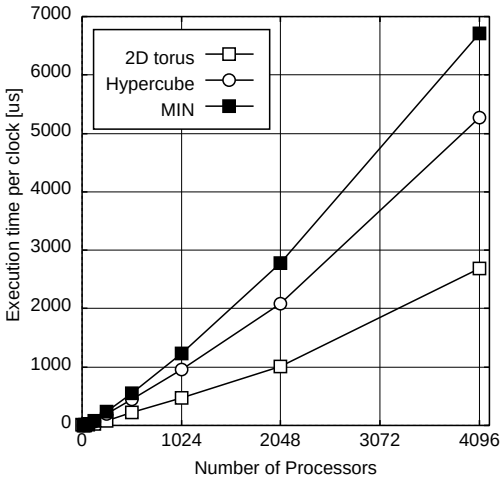

図 9: 相互結合網シミュレータの実行速度

ており，プロセッサ数は16から4,096まで変化させた．その他の評価条件を表 6に示す．シミュレーションホストマシンの諸元は表 3の通りである．図 9に実行結果を示す．トポロジによって差はあるが，1,024ノード構成の相互結合網を1ステップ実行させるのに要する時間は0.5から1.2ミリ秒程度である．これは1秒間に80から200ステップを処理する能力に相当する．トポロジによってグラフの形状が異なっているのは，ネットワークサイズとルーティングアルゴリズムの演算量の関係がトポロジによって異なっているためである．

5 研 究 事 例

1996年から開発が開始されたISISは，既に多くの研究で並列計算機の性能評価システム構築に利用されている．ここでは，pSASキャッシュの評価を例にISISによるシミュレータ構成方法を示した後，他のISISを用いた評価例についても概説する．

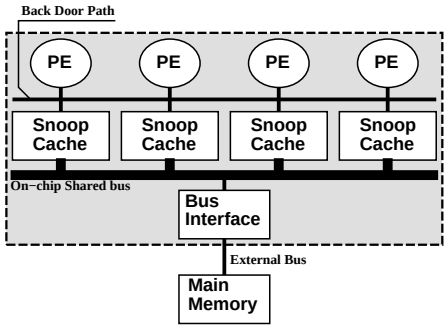

図 10: pSASキャッシュの構成

5.1 pSASキャッシュ

慶應義塾大学の井上らは，オンチップマルチプロセッサにおけるチップ内のキャッシュメモリの利用効率を上げる手法として，他のプロセッサのキャッシュを擬似的に自分のwayに見せるpSAS（Pseudo Set Associative and Shared）キャッシュを提案している[8]．オンチップマルチプロセッサ向けスヌープキャッシュプロトコルである新Keioプロトコルキャッシュに対し，pSASキャッシュは全体で10%，最大で16%性能が向上し，その効果が確認されている．

5.1.1 構成と動作

図 10にpSASキャッシュの構成を示す．各プロセッサは共有バスに接続されたキャッシュを持つ．キャッシュはさらに他のキャッシュを自分のキャッシュの疑似セットとして扱うためのデータパスであるBack Door Path（BDP）にも接続されている．pSASキャッシュはこのBDPを通して自分のキャッシュより数クロックの遅延で他のキャッシュにアクセスすることができる．

5.1.2 評価環境の実装

pSASアーキテクチャは将来のチップ内アーキテクチャの候補であるため，代替ハードウェアで評価を行うには多くのコストがかかる．また，記憶階層を正確に取り扱う必要がある．そこで，評価にはシステム全体をクロック単位でソフトウェアシミュレーションする方式が採用された．このシミュレータの実装にISISを用いている．

図 10に示されているシステム構成要素のうち，キャッシュコントローラとバスインタフェース以外はすべてISISのライブラリ内にあるユニットである．また，共有バスを実装するためにポートおよびパケットの派生クラスを用いている．また，性能比較のためにスヌープキャッシュで構成されたバス結合型並列計算機のシミュレータ等も実装された．

シミュレータ開発を開始してから評価に耐え得る安定動作に至るまでの開発期間は約4ヶ月である．また，このシミュレータのために新たに記述されたコード量は約4,000行，このシミュレータが用いているISIS内のコード量は約22,000行である．この開発期間と

コード量がISISを用いた場合と用いなかった場合とでどのくらいの差があるのかを客観的に判断するのは難しいが，クロック単位で記憶階層を正確にシミュレートするシステムとしては，開発期間4ヶ月/コード量4,000行という値は極めて短い/小さい値であると思われる．このことから，ISISの再利用性が高いことがうかがわれる．

5.2 他の利用例

5.2.1 共有キャッシュ対スヌープキャッシュ

慶應義塾大学の木透らは，1チップ上に従来のバス結合型マルチプロセッサを搭載した構成と，共有キャッシュを搭載した構成の比較を行った[9]．両アーキテクチャの命令レベルシミュレータ上でSPLASH2アプリケーション集を用いた定量的な評価を行った結果，共有キャッシュへのアクセス遅延を考慮すると，スヌープキャッシュを用いたアーキテクチャの方が良い性能を示すことが確認された．この評価に使用した2つの命令レベルシミュレータの実装にISISが用いられた．

5.2.2 相互結合網シミュレータ

本稿で提案しているシステムの構築過程において，相互結合網で接続された並列計算機の確率モデルシミュレータ，命令レベルシミュレータの実装と予備評価が行われた[5, 6]．確率モデルシミュレータは1,024ノード，命令レベルシミュレータは16プロセッサ構成での評価が可能であった．

6 おわりに

本稿では並列計算機シミュレータの構築支援システムISISを提案・実装し，その有効性を評価した．ISISは並列計算機内の機能ブロックを単位とした小さなシミュレータの集合体であり，並列計算機シミュレータ実装者にシミュレーションメカニズムのみを提供し，シミュレータの構築ポリシを拘束しない．これにより，従来のシミュレータの問題点であったシミュレータ実装時のコストを削減することができる．

ISISを用いたいくつかの並列計算機シミュレータを実装し，そのシミュレーション時間を評価した．その結果，確率モデルシミュレーションの場合は1,024プロセッサのシステムを毎秒80から200ステップ，命令レベルシミュレータの場合は4プロセッサのシステムを毎秒2万から3万ステップ処理する性能を持つことが分かった．これらの動作速度は他の実用されているシミュレータに比べて遜色ない性能である．このことから，ISISは実装コストを削減する一方，実行時のコストも犠牲にしていないことが明らかになった．

ISISはオンチップマルチプロセッサのキャッシュシステムの研究等で既に活用されている．また現在，超並列計算機JUMP-1のネットワークであるRDTの相互結合網命令レベルシミュレータ，マルチグレイン並列処理用マルチプロセッサシステムASCAのカスタムプロセッサMAPLE[10]の命令レベルシミュレータの開発に本システムが用いられている．本システムは並列分散コンソーシアム（PDC）の成果物であり，2000年10月にフリーソフトウェアとして `http://www.am.ics.keio.ac.jp/isis/` にて一般公開する予定である．

参考文献

[1] Mendel Rosenblum, Stephen A. Herrod, Emmett Witchel, and Anoop Gupta. Complete Computer Simulation: The SimOS Approach. *IEEE Parallel and Distributed Technology*, 1995.

[2] Richard L. Sites and Anant Agarwal. Multiprocessor Cache Analysis Using ATUM. *Proceedings of 15th International Symposium on Computer Architecture*, pp. 186–195, 1988.

[3] C. B. Stunkel and W. K. Fuchs. Analysis of Hypercube Cache Performance Using Address Trace Generated by TRAPEDS. *Proceedings of International Conference on Parallel Processing*, Vol. I, pp. 33–40, 1989.

[4] Emmett Witchel and Mendel Rosenblum. Embra: Fast and Flexible Machine Simulation. *ACM Sigmetrics '96: Conference on Measurement and Modeling of Computer Systems*, 1996.

[5] 米田卓司, 若林正樹, 緑川隆, 西村克信, 天野英晴. 並列計算機のための相互結合網シミュレータSPIDER. 電子情報通信学会技術研究報告 CPSY97-110, pp. 59–66, Jan 1998.

[6] 小守継夫, 若林正樹, 天野英晴. 相互結合網評価用命令レベルシミュレーション. 情報処理学会研究報告 HOKKE-99, pp. 1–6, Mar 1999.

[7] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. *Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 24–36, Jun 1995.

[8] 井上敬介, 若林正樹, 木村克行, 天野英晴. オンチップマルチプロセッサ用半共有型疑似連想キャッシュ. 情報処理学会論文誌, Vol. 40, No. 5, pp. 2008–2015, May 1999.

[9] Tohru Kisuki, Masaki Wakabayashi, Junji Yamamoto, Keisuke Inoue, and Hideharu Amano. Shared vs. Snoop: Evaluation of Cache Structure for Sigle-chip Multiprocessors. In *Euro-Par*, Feb 1997.

[10] Takashi Fujiwara, Katsuto Sakamoto, Takahiro Kawaguchi, Keisuke Iwai, and Hideharu Amano. A CUSTOM PROCESSOR FOR THE MULTIPROCESSOR SYSTEM ASCA. In *Proceedings of Sixteenth IASTED International Conference Applied Informatics – AI'98*, pp. 258–261, Mar 1998.