論　文

並列計算機シミュレータの構築支援環境

若林　正樹[†]　　天野　英晴[†]

Environment for Multiprocessor Simulator Development

Masaki WAKABAYASHI[†] and Hideharu AMANO[†]

あらまし　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もつため，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ごとにシミュレータを構築するという実装コストを伴う．本論文では，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，機能ブロックシミュレータをつな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．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，その実行速度と実装コストを評価した結果，実用上十分な実行速度を非常に低い実装コストで得られることがわかった．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．

キーワード　並列計算機，性能評価，シミュレーション，構築支援

1. まえがき

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きた．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，大きく分けて四つの方式がある．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一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ごとに個別の評価システムを実装しなければならない．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．にもかかわらず，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 1 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，プロセッサやメモリといった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「小さなシミュレータ」(以降ユニットと呼ぶ) 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，必要なユニットをつなぎ合わせることで所望のアーキテクチャのシミュレータを実現する．それゆえ，このシステムはターゲットアーキテク

[†] 慶應義塾大学理工学部，横浜市
Department of Computer Science, Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama-shi, 223–8522 Japan

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，各ユニット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も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，様々な研究成果に貢献している．

本論文は，**2.** で提案するシステムの設計，**3.** で実装について述べる．**4.** でこのシステム及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.** でこのシステムを使用した研究事例を紹介し，**6.** 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．以下に代表的な要素技術を示す．

[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．実行速度と精度がトレードオフの関係にある．実行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換えることで，評価面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．

[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，何らかの方式でそれらの動作を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低いとき有利で，後者はイベントの発生頻度が高いときに有利である．並列計算機の場合はイベント発生頻度が高いため，後者が好まれる．

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」を実現するためには，なるべく多くの技法をサポートし，なおかつそれらを評価対象の規模や要求される精度に応じて柔軟に組合せ可能にすることが要求される．また，構築されたシミュレータの精度や実行速度，動作環境を制限するようなものであってはならない．

そこで我々は，従来のように並列計算機全体を単一のシミュレータとして実装するのではなく，並列計算機内の機能ブロックをそれぞれ独立した小さなシミュレータとして実装する方式を採用する．この個々の小さなシミュレータをユニットと定義する．各ユニットには一定の時間間隔ごと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，クロック同期式を用いているため，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 3 方式に対応できる．

2.1 システムモデル

各ユニットを自由に相互接続できることを保証するために，機能ブロック間の接続，送受信される情報のそれぞれをポート，パケットとして抽象化し，実装方法を統一する．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の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から独立して動作する．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．

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 **3.2.2** で詳しく述べる．

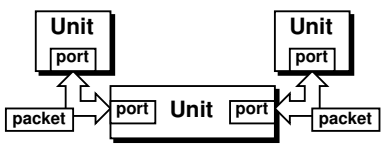

図1　システムモデル
Fig. 1　System model.

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 実装コスト

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．ゆえに，異なるシミュレータ間で同一ユニットを共有することでシミュレータの実装コストを軽減できる．そこで，多くのシミュレータで共有されると思われるプロセッサやメモリ等のユニットをあらかじめ実装しておき，ライブラリ形式でシミュレータ実装者に提供する．ライブラリ内に用意されていないユニットはシミュレータ実装者自身が記述する．

多くのユニットをライブラリ内に用意しておくこと，新規ユニットの実装を容易にすることで，シミュレータの実装コストをより軽減させることができる．

2.3 実行時コスト

シミュレータの実行時コスト（CPU 使用時間とメモリ使用量，ディスク使用量）は，シミュレータを構成する個々のユニットが必要とする資源と，それ以外のオーバヘッドからなる．個々のユニットは独立しているため，そのユニットの実行時コストはそのユニットをいかに最適化するかということで決まる．それ以外のオーバヘッドの部分は，ポートとパケットの処理を最適化することで軽減可能である．

また，動作環境に対する制約が少なければ，生成されたシミュレータは多くのホストマシン上で実行できる．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が総実行時間の短縮につながる．このように，移植性を高くすることで間接的に実行時コストを軽減することができる．

2.4 精度と実行速度

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よい．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トレードオフは支援システムには依存していないため，シミュレータ構築者が選択することができる．

3. 実　　装

本章では，**2.1** 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及びポート，パケットの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ユニットについて概説する．

3.1 実装形態

ソフトウェアシミュレータの実装言語は，高速に動作する実行形式を生成可能で，かつ大規模設計に耐え得るプログラミングパラダイムをサポートしている必要がある．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．**2.1** 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軽減を図る．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，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比較的容易に実装を行うことができる．記述したシミュレータはコンパイル及びリンクを経て実行ファイルに変換される．

3.2 システム基本部分

システムの根幹をなすユニット，ポート，パケットは

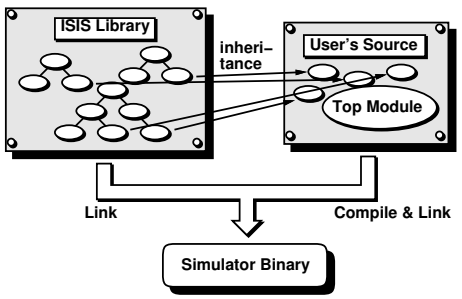

図2　ライブラリ構成
Fig. 2　Structure of the library.

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のみをメンバにも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．

なお，現時点ではISISはシミュレータ自身の並列実行をサポートしていないが，ユニット間の通信処理と順序依存解決はすべてポートに集約されているため，ポート内部の通信処理を変更することでシミュレータの並列化にも対応可能である．

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

図3 packetクラスの定義
Fig. 3 Definition of packet class.

```
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）
Fig. 4 Definition of port class.

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の三つの純粋仮想関数のみをも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，状態遷移をも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．これらの定義済デバイスを用いることで，大規模なユニットを比較的低コストで実装することができる．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．このデバイスについてはクラス階層は形成しない．

3.3 機能ブロックシミュレータ

2.2 で述べたように，シミュレータの実装コストを軽減させるためには，システムモデルに基づいた具体的な「有用な部品」となるユニットをライブラリ内に多数取りそろ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

```
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
Fig. 5 Definition of unit class.

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，パケット及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ー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．その代わり，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ひな形，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2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．したがって，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を管理する．リード及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にルータ内部の構成図を示す．

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 方式及び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御される．コントローラのルーチ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．両アルゴリズムのひな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よい．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．

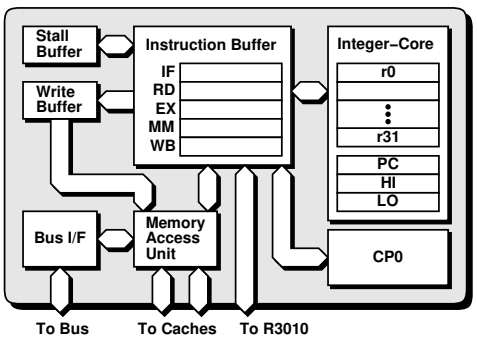

図6 R3000シミュレータ
Fig. 6 Structure of R3000 simulator.

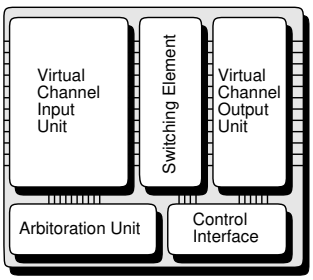

図7 ルータの構成
Fig. 7 Structure of router.

表1 動作実績
Table 1 Supported architecture and OS.

アーキテクチャ	OS
HP PA-RISC	HP-UX 10.10
Sun Sparc	SunOS 4.1.4 Solaris7
PC/AT compatibles	FreeBSD 3.5.1 Linux 2.2.17 Solaris8
SGI Origin	IRIX 6.4

表2 要求リソース量
Table 2 Required resource size.

ユニット	リソース (bytes)
R3000プロセッサ	1,052
R3010コプロセッサ	1,936
メモリコントローラ	140
I/O その他	464
ルータ	244
Network I/F	1,052
ユニプロセッサマシン	3,668

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

命令レベルシミュレーションを行う際に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エミュレートするためのユニットが定義されている．このユニットに対して処理要求パケットを送信すると，そのアクセスはホストマシン上のOSに対する処理要求に写像される．

3.4 実装方法と動作実績

既に述べたとおり，ISISの実装にはANSI C++言語を用いた．移植性を高くして間接的に実行時コストを軽減させる目的で，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．表1に現在までに動作が確認された主なアーキテクチャと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．

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及びANSI C++，ANSI FORTRAN 77，FORTRAN 90をサポートしている．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．

4. 評価

各ユニットの使用リソース量と実行速度，及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，本論文で提案するシステムの実行性能を評価する．また，性能評価に用いたシミュレータのコード共有度を求め，実装コストについて評価する．

4.1 ユニットの性能評価

4.1.1 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，及びホストマシンと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，2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，入出力装置を搭載した，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，最小規模のシミュレータはわずか4キロバイト弱のリソースしか消費しないことがわかる．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わかる．

なお，シミュレーションを行う際には，ターゲットマシンがもつキャッシュやメモリ等をシミュレートするため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**で述べたとおり，ターゲットマシンが大量のメモリ領域を要求しても、シミュレータ上で実行されるアプリケーションが実際にアクセスしたメモリ容量以上のリソースは消費しない．

4.1.2 実行速度

4.1.1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する．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16k/データ4kバイトの1次キャッシュを追加搭載している．なお，シミュレーションホストマシンの諸元は表3に示すとおりである．

このアーキテクチャ上で二つの単純なアプリケー

表3　ホストマシンの諸元
Table 3　Simulation host machine.

アーキテクチャ	PC/AT compatibles
プロセッサ	Pentium-III(600MHz) × 1
メモリ	512MBytes
OS	Solaris8

表4　ユニットの実行速度
Table 4　Unit speed.

	loop	copy
合計step数	2,002,590	2,002,641
総実行時間[s]	6.34	5.37
実行時間/step[μs]	3.166	2.681
実行step数/秒	315,866	372,931

ションを実行した．一つは100 万回の空ループを実行するもの (loop)，もう一つは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 の方がメモリアクセス待ちでプロセッサが待ち状態になる分，実行速度が速くなっている．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31 万から38 万ステップ実行できることがわかる．

文献 [1] によれば，SimOS をR4400 150MHz のホストマシン上で実行し，ホストマシンと同一のプロセッサをパイプラインレベルまでシミュレートした場合，実プロセッサで直接実行した場合との実行時間比は180 倍から232 倍である．これは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65 万から85 万ステップ実行できる速度である．このことから，ISIS によるシミュレータはSimOS の2 分の1 から3 分の1 程度の実行速度をもつものと思われる．

4.2　シミュレータの性能評価

4.2.1　命令レベルシミュレーション

小規模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，共有メモリにプロセッサ台数分のポートをもつ理想的なメモリを用いた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．なお，シミュレーション

表5　命令レベルシミュレーションの負荷
Table 5　Workloads for instruction-level simulation.

プログラム	問題サイズ
BARNES	512 bodies
FFT	65,536 complex doubles
LU	256 × 256 matrix
OCEAN	66 × 66 grid
RADIX	2,097,152 keys

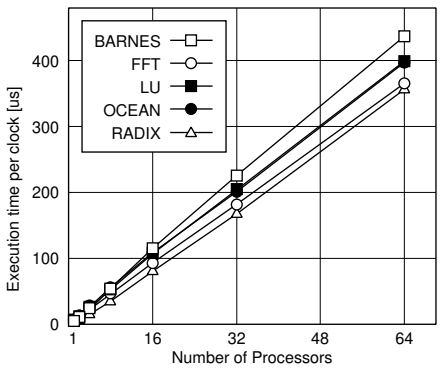

図8　命令レベルシミュレータの実行速度
Fig. 8　Performance of instruction-level simulators.

ホストマシンの諸元は先ほどと同じく表 3に示すとおりである． 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．グラフからわかるように，この1 ステップ当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ほとんど左右されない．これは4 プロセッサの並列計算機であれば1 秒間に4 万から7 万ステップを処理する能力に相当する．

4.2.2　確率モデルシミュレーション

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，評価を行う．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ており，ノード数は16 から4,096 まで変化させる．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のとおりである．

図 9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．トポロジーによって差はあるが，1,024 ノード構成の並列計算機を1 ステップ実行させるのに要する時間は7 から15 ミリ秒程度である．これは1 秒間に60 から130 ステップを処理する能力に相当する．トポロジー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チングアルゴ

表6　確率モデルシミュレーションの評価条件
Table 6　Parameters of probabilistic-model simulation.

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	1%
ルーチング	固定ルーチング
総ステップ数	100,000

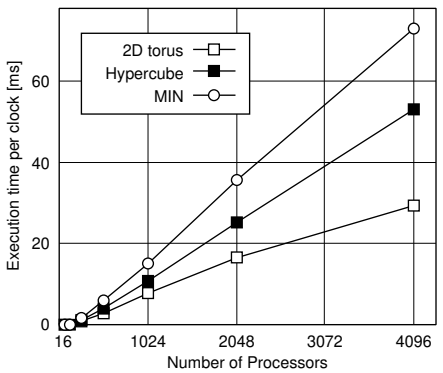

図9　確率モデルシミュレータの実行速度
Fig. 9　Performance of probabilistic-model simulators.

リズムの演算量の関係がトポロジーによって異なっているためである．

4.3　実装コスト

性能評価に用いたシミュレータのソースコード量の内訳を調べ，ISIS による実装コストの軽減の度合を評価する．評価に用いるのは，**4.1.2** で述べた命令レベルユニプロセッサシミュレータ（UNI），**4.2.1** で述べた命令レベル並列計算機シミュレータ（MULTI），**4.2.2** で述べた確率モデル並列計算機シミュレータ（2D，HC）である．2D はトポロジーに 2D トーラス，HC はハイパキューブを用いている．

表 7に，各シミュレータのソースコード量の内訳を示す．表中の値は行数である．なお，「システムモデル部」はユニット，パケット，ポートの基底クラス等のシステム基幹部分に関するコードを示している．「当該マシン専用コード」は各シミュレータのためだけに記述されたコードである．

UNI 及び MULTI は同一プロセッサを用いた命令レベルシミュレータであるため，システムモデル部，デバイス，プロセッサ，メモリコントローラ，I/O については完全に同一コードを用いている．UNI のコード

表7　ソースコード量の内訳
Table 7　Breakdown of source code.

	UNI	MULTI	2D	HC
システムモデル部	2,710	2,710	2,467	2,467
デバイス	1,914	1,914	637	637
プロセッサ	10,683	10,683	-	-
メモリコントローラ	543	543	-	-
I/O その他	1,460	1,460	-	-
Network I/F	-	-	2,183	2,183
ルータ	-	-	906	906
複数プロセッサ管理	-	617	-	-
当該マシン専用コード	203	756	233	172
専用コードの比率	1.16%	3.98%	3.63%	2.70%

のうちの 99%が MULTI と共有されている．

2D と HC は確率モデルシミュレータであるため，命令レベルシミュレータと共通する機能はほとんどない．そのため，命令レベルシミュレータである UNI や MULTI とはシステムモデル部を除いてコードをほとんど共有していない．しかし，2D と HC 間では類似点が多いため，専用コード以外は完全に同一コードとなっている．2D のコードのうちの 96%が HC と共有されている．

この結果から，特定の並列計算機のシミュレータを実装するために新規に記述しなければならないコード量は少なく，また複数シミュレータに共通する機能のための記述は容易に共有できることがわかる．このことから，ISIS はシミュレータの実装コストを大幅に軽減できることがわかる．

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．

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]．pSAS キャッシュはキャッシュラインのコピーの数が減るため，半ば共有キャッシュの状態になり，実際にキャッシュされるデータの量を増やすことができる．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もつ．キャッシュは更に他のキャッシュを自分のキャッシュの擬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．プロセッサはこの 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．

pSAS キャッシュのステートマシンは，まず自分のキャッシュでヒット判定を行い，キャッシュミスとなった場合は BDP アービタに request を出す．BDP を取得後，BDP 経由でアドレスを全キャッシュタグへ流して他のキャッシュでのヒット判定を行う．キャッシュヒットした場合は，プロセッサは BDP を通してそのキャッシュへ直接アクセスする．このときキャッシュラインの移動やコピーは行わない．

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及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．

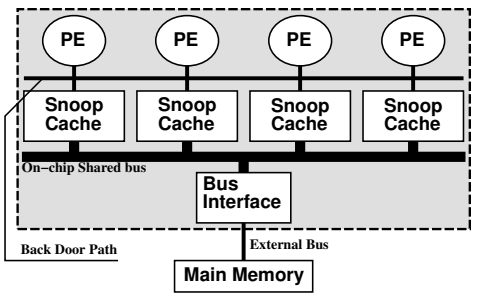

図 10　pSAS キャッシュの構成
Fig. 10　Structure of pSAS cache.

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 か月である．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，その大半は **4.2.1** と **4.3** で述べた命令レベル並列計算機シミュレータに用いられたコードを再利用したものである．また，このシミュレータのために記述されたコード量は約 4,000 行で，ほとんどが対象アーキテクチャの専用ハードウェアである BDP アービタや pSAS キャッシュのための記述である．

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 か月/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 はコードの再利用性が高く，旧来の手法では実現不能だった実装コストの軽減に非常に有用である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 チップ上に従来のスヌープキャッシュによる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二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

本論文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. むすび

本論文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，例として64プロセッサまでのシステムを対象とした命令レベルシミュレータと，4,096プロセッサまでのシステムを対象とした確率モデルシミュレータのシミュレーション時間を評価した．その結果，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4万から7万ステップ処理する性能，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60から130ステップ処理する性能をもつことがわかった．これらの動作速度は他の専用シミュレータの数分の1程度である．また，これらのシミュレータは可能な限りコードを共有していて，共有不能なコード量は全体のわずか数%にしか満たないことがわかった．このことから，ISISは実行時コストをそれほど損なわずに実装コストを大幅に軽減する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 [10]のカスタムプロセッサMAPLE [11]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・分散処理研究推進機構（PDC）の成果物であり，2000年10月からフリーソフトウェアとして `http://www.am.ics.keio.ac.jp/isis/` にて一般公開している．

文　献

[1] M. Rosenblum, S.A. Herrod, E. Witchel, and A. Gupta, “Complete computer system simulation: The SimOS approach,” IEEE Parallel and Distributed Technology: Systems & Applications, vol.3, no.4, pp.34–43, Winter 1995.

[2] R.L. Sites and A. Agarwal, “Multiprocessor cache analysis using ATUM,” Proc. 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,” Proc. International Conference on Parallel Processing, vol.I, pp.33–40, 1989.

[4] E. Witchel and M. Rosenblum, “Embra: Fast and flexible machine simulation,” Proc. ACM SIGMETRICS ’96, International Conference on Measurement and Modeling of Computer Systems, vol.24, no.1, pp.68–79, May 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,” 信学技報, CPSY97-110, 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,” 情処研報, HOKKE-99, pp.1–6, March 1999.

[7] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” Proc. 22nd International Symposium on Computer Architecture, pp.24–36, June 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,” 情処学論, vol.40, no.5, pp.2008–2015, May 1999.

[9] T. Kisuki, M. Wakabayashi, J. Yamamoto, K. Inoue, and H. Amano, “Shared vs. snoop: Evaluation of cache structure for single-chip multiprocessors,” Proc. 3rd International Europian Conference on Parallel Processing – Euro-Par’97, pp.793–797, Feb. 1997.

[10] K. Iwai, T. Morimura, T. Fujiwara, K. Sakamoto, T. Kawaguchi, K. Kimura, H. Amano, and H. Kasahara, “Interconnection network of ASCA: A multiprocessor for multi-grain parallel processing”, Proc. 16th IASTED International Conference Applied Informatics – AI’98, pp.262–264, March 1998.

[11] T. Fujiwara, K. Sakamoto, T. Kawaguchi, K. Iwai, and H. Amano, “A custom processor for the multiprocessor system ASCA”, Proc. 16th IASTED International Conference Applied Informatics – AI’98, pp.258–261, March 1998.

（平成12年6月8日受付，10月2日再受付）

若林　正樹

平8慶大・理工・電気卒．平10同大大学院理工学研究科修士課程了．現在，同大大学院理工学研究科博士課程在学中．計算機アーキテクチャの研究に従事．

天野　英晴　（正員）

昭56慶大・工・電気卒．昭61同大大学院理工学研究科電気工学専攻博士課程了．現在，慶應義塾大学理工学部情報工学科助教授．工博．計算機アーキテクチャの研究に従事．