並列計算機シミュレータライブラリの提案

若林 正樹 †　　米田 卓司 †　　天野 英晴 †

† 慶應義塾大学理工学部

ソフトウェアシミュレーションによる並列計算機の性能評価は、ハードウェアによる制約を受けないため、設計や予備評価に有効な手法である。シミュレーションを行う場合、通常は特定のネットワークや計算機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実装するというオーバヘッドを伴う。本論文では、様々な並列計算機シミュレータで共用可能な機能を抽出した、アーキテクチャ非依存なライブラリを提案する。本ライブラリにより、シミュレータ実装者の負担を大幅に軽減できる。また、本研究で提案するライブラリを用いて実装したシミュレータの性能評価を行った結果、実用的な速度で動作するシミュレータを生成できることが確かめられた。

ソフトウェアシミュレーション, 性能評価, アーキテクチャ非依存, ライブラリ

A design of Multiprocessor Simulator Library

M. Wakabayashi†　　T. Komeda†　　H. Amano†

† Keio University

Software simulation is an effective method for the performance evaluation of multiprocessors, since the target is not limit by hardware configuration. However, even in software simulator, network or memory system of the simulation target is usually fixed. Using such a simulator, engineers must build a distinct simulator for their own machine. In this paper, we propose an architecture-independent library for multiprocessor simulators. It includes various functions which can be useful to develop simulators and reduce engineers' job. We implemented a simulator using the library and evaluated its performance. As a result, we confirmed that a simulator which runs at reasonable speed can be built using the library.

Software simulation, Performance evaluation, Architecture independent, Library

1 はじめに

近年の並列計算機や並列アプリケーションは大変複雑な構成を採っているため、理論解析やシミュレーション、ハードウェアなどによる様々な性能評価が行われる。その中でソフトウェアシミュレーションによる評価は、シミュレーション対象のハードウェアによる制約を受けないため、設計や予備評価段階で有効な手法である。

並列計算機をシミュレーションによって評価する場合、重点が置かれる評価対象によって様々なシミュレータが実装される。代表的なシミュレータとしては、トレース駆動型シミュレータとして Stanford 大学の ATUM-2[1]、実行駆動型シミュレータとして Stanford 大学の Tango[2]、命令レベルシミュレータとして慶應義塾大学の MILL[7]、ISIS[8]、複合式シミュレータとして Stanford 大学の SimOS[4] 等が挙げられる。

しかし、これらのシミュレータの多くは特定のアーキテクチャを持つ並列計算機の性能評価を目的として開発されたため、対象と異なる構成の並列計算機をシミュレートすることはできない。また一般に、大規模並列計算機にはトレース駆動型シミュレーション方式、小規模並列計算機には命令レベルシミュレーション方式といったように、評価対象に応じて異なるシミュレーション方式が採用される。このような理由から、新たな並列計算機を評価する必要性が生じた場合には、新しいシミュレータを実装することが多い。このオーバヘッドは研究者にとって大きな負担となる。

そこで本研究では、並列計算機シミュレータのためのアーキテクチャ非依存なライブラリを提案する。本ライブラリは、様々な並列計算機シミュレータに共通する機能ブロックを抽出し、より汎用性を高めてライブラリ化したものである。本ライブラリを使用してシミュレータを実装する場合、ライブラリで用意された機能ブロックをそのまま使用することができるため、シミュレータ実装者は対象の並列計算機特有の機能のみを実装するだけで良い。したがってシミュレータ実装者の負担を大幅に軽減することができる。

ライブラリの実装は、命令レベル並列計算機シミュレータ ISIS を拡張する形で行う。ISIS はマルチプロセッサチップの評価を目的として開発された命令レベル並列計算機シミュレータで、ハードウェアを機能ブロック単位でモデル化し、個々の機能ブロックを結合させることにより並列計算機を構成するよう設計されている。計算機の構成を柔軟に変更できるため、既に様々な研究で利用されている [9][10][11][12]。しかし ISIS の利用目的が広がるにつれて、機能ブロックの結合方式に問題があること、大規模システムのシミュレーションが困難であることなど、拡張性の面でいくつかの問題点が浮上してきた。そこで、ISIS の抱えている問題点を解決し、より汎用性の高いライブラリとして再構築する。

また、本研究で提案するライブラリを用い、ある簡単なアーキテクチャを持つ並列計算機のシミュレータを構築し、シミュレータ自身の実行速度の性能評価を行った。

2 命令レベルシミュレータ ISIS

慶応義塾大学で開発された命令レベル並列計算機シミュレータ ISIS は、マルチプロセッサチップに代表される小規模並列計算機の性能評価を目的として開発された。ハードウェアの機能ブロック (以降ユニット呼ぶ) を構成単位とし、ユニットを結合させることにより評価対象の並列計算機を構成する。各ユニットはクロック単位で状態遷移するため、きわめて正確に動作がシミュレートされる。図 1に並列計算機の構成例を示す。この例ではプロセシングユニットやキャッシュ、メモリ、バスがユニットにあたる。それぞれのユニットが接続されたユニットと通信を行いながら状態遷移を繰り返し、系全体のシミュレーションが行われる。

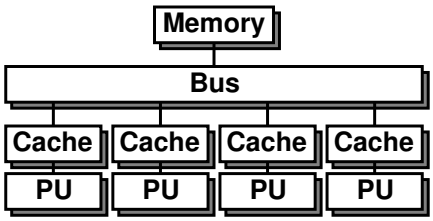

図 1: 並列計算機構成例

ユニットとして MIPS 社の R3000 プロセッサ [5]、R3010 浮動小数点演算コプロセッサ [6]、バス結合型並列計算機用のスヌープキャッシュ等のユニットが実装されている。また、SPLASH ベンチマーク [3] をシミュレータ上で実行するための環境も整備されている。したがって、R3000 プロセッサからなるバス結合型並列計算機をシミュレートし、定量的な性能評価を行うことができる。さらに、シミュレーションの実行結果としてアドレストレースファイルを生成する機能も備えているため、他のトレース駆動型シミュレータへのデータ入力システムとしても有用である。

ISIS の利用例として、“共有キャッシュ対スヌープキャッシュ: シングルチップマルチプロセッサにおけるキャッシュ機構の評価”[9] が挙げられる。この研究では、マル

チプロセッサを 1 チップ上に構成した場合のキャッシュの構成について検討している。図 2に評価対象のアーキテクチャの例を示す。(a) はそれぞれのプロセッサに対してスヌープキャッシュを接続したアーキテクチャ、(b) はチップ内の全プロセッサで 1 つのキャッシュを共有するアーキテクチャである。これらの各アーキテクチャ上で SPLASH ベンチマーク集のアプリケーションを実行することで、マルチプロセッサチップの定量的な性能評価を行っている。このように、様々な構成のマルチプロセッサシステムをシミュレートするために ISIS の柔軟な構成変更機能が利用された。

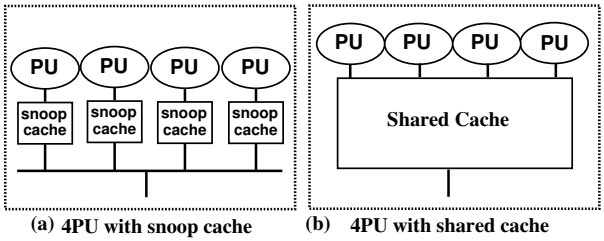

図 2: 共有キャッシュ対スヌープキャッシュ

また、直接 ISIS を使用しているわけではないが、トレース出力機能を使用した研究としては、“多段結合網に基づくキャッシュコヒーレンスネットワークの設計と実装”[10]、“分散共有メモリを持つ WS クラスタ: JUMP-1/3[11]、超並列マシン JUMP-1 のための分散共有メモリ管理プロセッサ”[12] 等が挙げられる。

この様に ISIS は多方面で利用されているが、幾つかの問題点も抱えている。

1 つは、ユニット同士の接続に明確な規定が存在しないことである。一定の規則が存在しないため、ユニットによっては接続可能なユニットが制限される場合がある。

もう 1 つは、命令レベル方式のシミュレーションであるため、対象となる並列計算機のプロセッサ数に比例した実行時間がかかることである。また、命令レベル方式ではシミュレートするシステムの資源の状態も全て把握する必要があるため、大規模な並列計算機をシミュレートする場合にはホストマシンに莫大なメモリを用意する必要がある。

3 設計

前述した ISIS の問題点を踏まえ、より汎用性の高いシミュレーション環境を構築するために、まずライブラリへの要求事項を列挙する。

1. 並列計算機のアーキテクチャに依存しない
2. シミュレーション方式に依存しない
3. シミュレータ実装者に負担をかけない
4. 生成したシミュレータが実用的な速度で動作する

ライブラリを並列計算機アーキテクチャとシミュレーション方式に依存させないためには、並列計算機内の個々の機能ブロックをユニットとして個別に実装する方法が有効である。特定の並列計算機のシミュレータを実装する場合には、個々の機能ブロックを結合することにより並列計算機を構成すればよい。シミュレーション方式によって機能が異なる部分—例えばプロセッサなど—については、その方式に応じたユニットを個別に用意することで対応が可能である。さらにユニット単位で独立にカプセル化することでユニット別にメンテナンスすることが可能になるため、実装や機能拡張のコストを軽減できる。

シミュレータ実装者にかかる負担を最小限に留めるためには、様々な並列計算機に共通する機能ブロックをなるべく多く実装済みのユニットとして提供することはもちろんだが、ユーザ自身が新しいユニットを実装する際の負担も抑える必要がある。これを実現するために、ユニット間の共通機能を継承を用いて記述することにする。継承を利用すると、基本機能を持つユニットをライブラリ側で用意し、ユーザが基本機能を持つユニットを拡張することが可能となる。さらに、ライブラリ内のユニットを実装する際のコストを低くすることができるというメリットもある。

生成したシミュレータを実用的な速度で動作させるためには、ライブラリを実装する際に使用する言語が高速に動作する実行ファイルを出力できる言語を選択する必要がある。一般に、ソフトウェアによる並列計算機シミュレーションは実行時間が長くかかるため、シミュレータの実装言語には C 言語など、高速に動作する実行ファイルを出力できる言語を選択するのが普通である。実装言語の選択については 4 章で詳しく述べる。

3.1 システムのモデリング

様々なユニットを相互結合させることで並列計算機をシミュレートするためには、ユニットの動作方法を規定しなければならない。また、ユニット同士を柔軟に接続できる様にするために、ユニット間の結合と通信のインタフェースを定義する必要がある。

ユニットを動作させる方法には、以下の 2 つが考えられる。

- ユニット毎に次の状態遷移時刻を計算する方法

- 全ユニットを一定時間毎に状態遷移させる方法

前者の方法は、状態遷移時刻を正確に扱えるが、発生するイベントの時刻管理を行う必要がある。後者の方法は時刻管理が容易だが、状態遷移時刻は設定した時間間隔毎にしか行えないという制限がある。しかし計算機をシミュレートする場合、計算機内部のハードウェアはそのほとんどがクロックに同期して動作しているため、後者の方法でも性能評価を行うには十分正確なシミュレーションが可能である。そこで、時刻管理の容易さから後者の方法を選択する。ユニットそれぞれに状態遷移機能を持たせ、システム内の全ユニットを一定時間毎に状態遷移させることにする。

次に、ユニット間の結合と通信のインタフェースを規定するために、情報を仲介する存在としてパケット、結合を仲介する存在としてポートを導入する。

パケットは、相互接続されたユニット間でやり取りされる全ての情報を表現する。例えばルータ間で送受信されるフリット、プロセッサがバスに出力するメモリアクセス要求などがパケットである。ポートは、ユニット同士を結合している通信路への入出力端子を表現する。接続したいユニット同士のポートを接続することで、ユニット間の通信路が構築される。

図 3に結合と通信のモデル化の概念図を示す。この図では、2 つのユニットが 1 つのユニットの仲介によって通信を行っている。パケットはユニット間でやり取りされる情報を表す。ポートはパケットの受渡しを仲介するための入出力端子である。また、ユニット同士の結合も全てポートによって行われる。ユニットはポートとパケットを介して他のユニットとの結合と通信を行う。

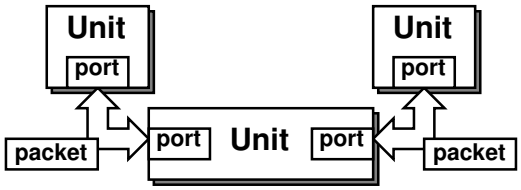

図 3: 結合と通信のモデル化

3.2 大規模システムへの対応

一般的には、大規模システムにはトレース駆動型シミュレーションを使用する場合が多い。その理由は、比較的正確で高速な評価結果が得られるためである。そこで、ライブラリをトレース駆動型シミュレーションに対応させることにする。

トレース駆動型シミュレーションでは、トレースファイルを読み込んでネットワークの評価を行う。そこで、プロセッサユニットに代わるトレースリーダユニットを導入する。図 4にトレース駆動型シミュレータの構成図を示す。トレースリーダはトレースファイルを読み込んでプロセッサと同様のパケットを送出する機能を持つ。

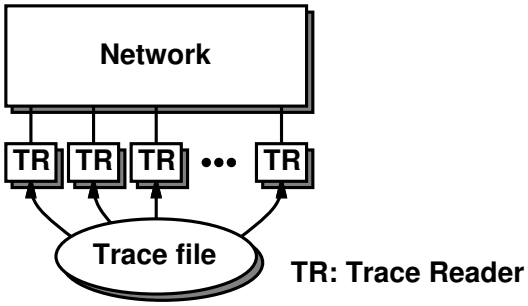

図 4: トレース駆動型シミュレーション

ネットワークユニットはプロセッサユニットとトレースリーダユニットを区別する必要がないため、同一のネットワークユニットで命令レベルシミュレーションとトレース駆動型シミュレーションを行うことが可能になる。ユーザはまず命令レベル方式で 1 回実行を行ってトレースファイルを生成する。その後は生成したトレースファイルを入力とし、トレース駆動方式で短時間で数多くの評価を行うことができる。

4 実装

3 章ですでに述べた様なユニットのカプセル化と継承を行うためには、ライブラリを実装する際に使用する言語がオブジェクト指向プログラミングをサポートしていることが望ましい。一方、ソフトウェアによる並列計算機シミュレーションは一般に実行時間が長くかかるため、シミュレータの実装言語には C 言語など、高速に動作する実行ファイルを出力できる言語を選択するのが普通である。オブジェクト指向プログラミングと実行時の効率を両立させる必要性から、ライブラリの実装言語に C++言語を採用する。それぞれのユニット、パケット、ポートをクラスとして実現し、クラスライブラリの形でユーザに提供することとする。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。

以下、それぞれのクラスの実装について述べる。

4.1 ユニット

クロック入力によって状態遷移を行うオブジェクトのクラスの基底クラスとして、synchronous_unit クラスを定義する。並列計算機内のユニットを表すクラスは全てこのクラスの派生クラスとして実装される。synchronous_unit クラスは抽象クラスであり、その派生クラスに対して状態遷移 (クロック入力)、状態の初期化 (リセット) 機能のインタフェースを規定する働きを持つ。これらの機能は全て仮想関数とし、派生クラスによって実装がなされるよう設計されている。

4.2 パケット

パケットを表す全てのクラスの基底クラスとして、packet_base クラスを定義する。バス上を通過する情報、ルータ内を通過するパケットなどはこの packet_base クラスの派生クラスとして実装される。

4.3 ポート

ユニット同士を結合するポートを表すクラスの基底クラスとして、port_base クラスを定義する。このクラスはポート同士の接続とパケットの転送処理機能を提供する。複数のポートを接続すると、接続された全てのポート間で共有される通信路が確保される。この通信路は、ただ 1 つのパケットだけを格納することができる。

4.4 バス

バスは、ポート間に構成される通信路をそのままバスとして使用する。図 5にバスの構成方法を示す。

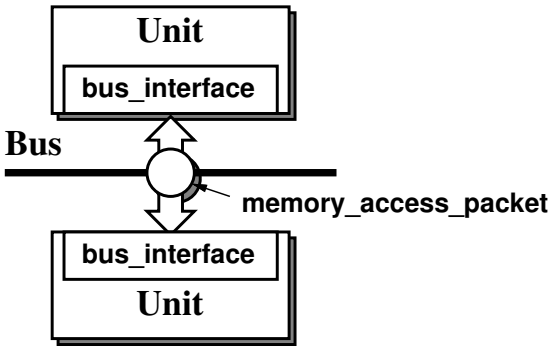

図 5: バスの構成方法

ユニット間の結合には、port_base クラスの派生クラスである bus_interface クラスを使用する。また、結合路上に存在するバスのデータを表すクラスとして、packet_base の派生クラスである memory_access_packet クラスを使用する。memory_access_packet は計算機内のバスのアドレス線、データ線、コントロール線をカプセル化したものである。また、bus_interface クラスは、ユニット間に構成された通信路と通信路上のパケットをバスとしてアクセスできる様にするためのインタフェースを提供する。例えばあるアドレスのデータに対するリード要求を発行する場合は、bus_interface クラスに対して指示を行うだけでよい。実際のパケットの送信などの処理は bus_interface クラスが行う。

4.5 プロセッサ

プロセッサを表す抽象クラスとして、全てのプロセッサの基底クラスとなる processor クラスを定義する。ユニットクラス synchronous_unit の派生クラスである。また、プロセッサ内のレジスタファイルへのアクセス、様々な状態問い合わせを行う機能を仮想関数として持つ。状態の問い合わせ機能により、現在プロセッサがストール中であるかどうか、メモリアクセス中であるかどうかなどが細かく調査できるようになっている。

実際のプロセッサを表すクラスとして、r3000 クラスが実装されている。このクラスは MIPS 社の R3000 プロセッサ [5] をクロックレベルで正確にシミュレートする。図 6に r3000 クラス内部のブロック図を示す。また、R3010 浮動小数点演算コプロセッサ [6] を表すクラス、R3000 プロセッサに接続する 1 次キャッシュを表すクラスもすでに実装されている。これらのクラスは命令レベル並列計算機シミュレータ ISIS[8] 内のユニットを移植したものである。

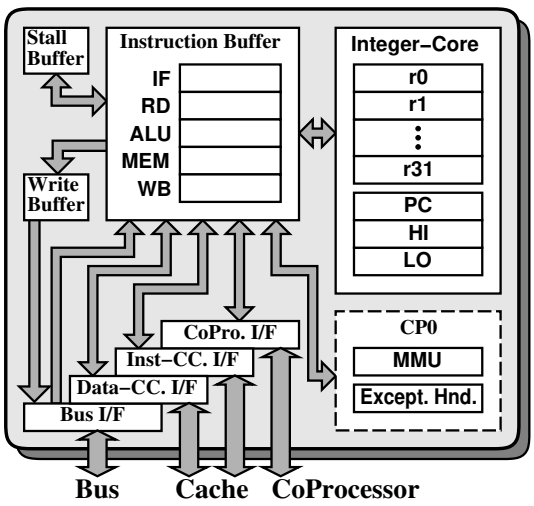

図 6: r3000 クラス内部のブロック図

4.6 メモリ

メモリは、実際の記憶装置としてのメモリを表すクラスと、そのメモリを制御するメモリコントローラクラスの 2 つのクラスで実装されている。

memory クラスは並列計算機の主記憶装置を表現するクラスである。このクラスは内部で動的にページ単位の記憶領域管理を行っており、要求されたサイズのメモリをすぐには確保しない。値が書き込まれた時点でページ毎に実体が確保される。従って莫大な量の記憶領域を要求しても、実際に巨大な領域を使用しなければホストマシンの資源が不足することはない。この機能により、ホストマシンよりも巨大な資源を持つ並列計算機を容易にシミュレートすることができる。

memory_control_unit クラスは、ユニットクラスの具体派生クラスである。バスとメモリを接続して使用する。バス上のアクセス要求を受け取り、要求されたアドレスに対応するメモリのデータに対してアクセスを行い、バス上に応答を返す機能を持つ。単一ワード転送、バースト転送、スプリットトランザクションを行うことができる。また、リードアクセス時間、ライトアクセス時間を個別に設定できる。

4.7 入出力デバイス

シミュレータ上で実行されるソフトウェアに対してファイル入出力機能を提供するために、file_io_unit クラスが用意されている。

file_io_unit クラスは、シミュレートされるソフトウェア上でオープンされているファイルのファイル記述子番号に対応するストリームを保持している。ソフトウェア側からファイルに対する操作が発行されると、指定されたファイル記述子に対応するストリームに対する操作に変換する。また、ファイルオープン要求が発行されると、新たにファイルストリームをオープンし、新しいファイル記述子を割り当てる。

4.8 トレース出力用シミュレータ

他のトレース駆動型シミュレータへのデータ入力システムとして、アドレストレースデータを収集するための命令レベル並列計算機シミュレータが実装されている。

図 7にトレース出力用シミュレータのブロック図を示す。各プロセシングエレメント内には R3000 プロセッサ、ローカルメモリ、ファイル入出力ユニットなど、単体で計算機として動作できるだけのデバイスが全て格納されている。共有メモリはプロセッサ台数分だけの入出力ポートを有するマルチポートメモリで、1 つの memory

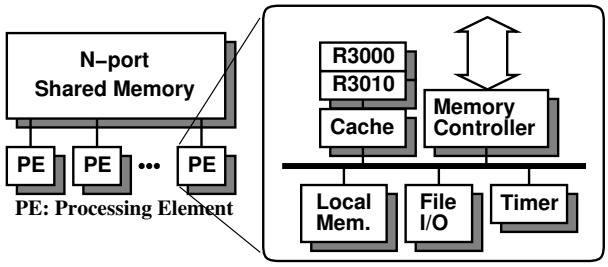

図 7: トレース出力用シミュレータ

クラスと複数の memory_control_unit クラスで実装されている。この共有メモリはプロセッサ台数が多い場合には非現実的なハードウェアであるが、結合網の性質を排除してアドレスパターンのみを正しく抽出することができる。

5 評価

提案するライブラリが実用的な速度で動作することを確認するために、本ライブラリを用いて実装したシミュレータの実行速度の評価を行った。

性能評価に使用したシミュレータは、図 7に示すトレース出力用シミュレータである。プロセッサ数は 1 台から 64 台まで変化させた。

シミュレータ上で実行するアプリケーションには SPLASH ベンチマーク集 [3] の RADIX を使用した。RADIX は整数の基数ソートを行うプログラムである。問題のサイズである要素数は 2M とした。

シミュレータを実行するホストマシンには、Pentium-II 300MHz の IBM PC 互換機を使用した。使用した OS は FreeBSD 2.2.5-RELEASE である。

以上の様な条件で、以下の 2 つの値を各プロセッサ台数について測定した。

- アプリケーションの実行に要した総クロック数
- シミュレーションに要した実行時間

アプリケーションの実行に要した総クロック数はシミュレートされた並列計算機上での実行時間であり、並列計算機の性能を表している。また、シミュレーションに要した実行時間はホストマシン上での実行時間であり、シミュレータの性能を表している。さらに、測定した値を元に台数効果を算出した。

また、これとは別に命令レベルシミュレータ ISIS を用いてシミュレーションに要した実行時間を測定し、実行時間の比較を行った。ISIS の対象となるアーキテク

チャには、“共有キャッシュ対スヌープキャッシュ: シングルチップマルチプロセッサにおけるキャッシュ機構の評価”[9] の共有キャッシュアーキテクチャを使用した。この比較はプロセッサ台数が 4 の場合について行った。

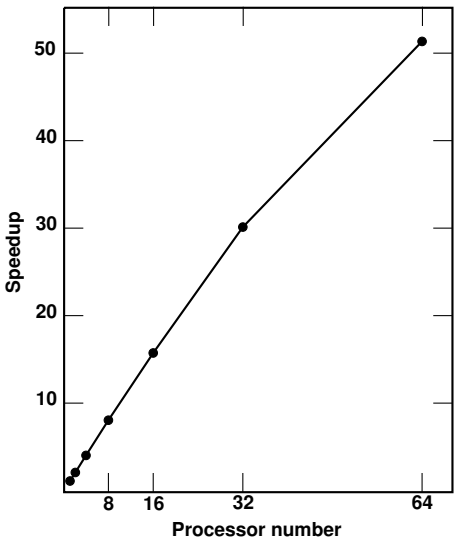

図 8: 台数効果

図 8に、台数効果の測定結果を示す。この測定結果から、実行したアプリケーションが 32 台以下の台数ではほぼ理想的な台数効果を示していることが分かる。

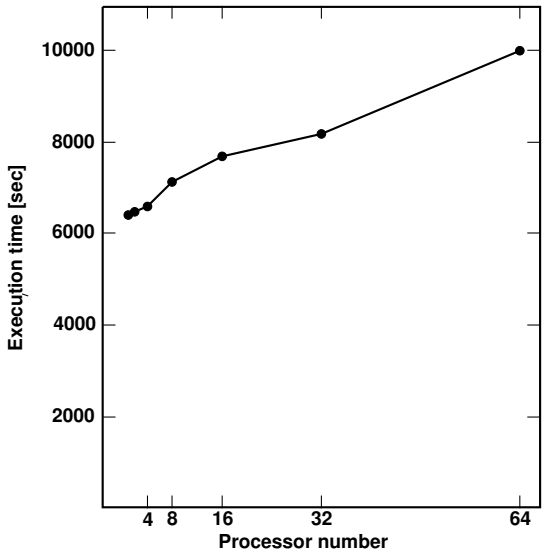

図 9: シミュレーションの総実行時間

図 9は、シミュレーションに要した総実行時間である。1 プロセッサでは 6000 秒、64 プロセッサでは 10000 秒程度であった。実際の並列計算機の性能評価を行う際には、アプリケーションの計算量によってはこの数倍程度の実行時間がかかることが予想されるが、長くても 1 日程度で実行できることが分かる。

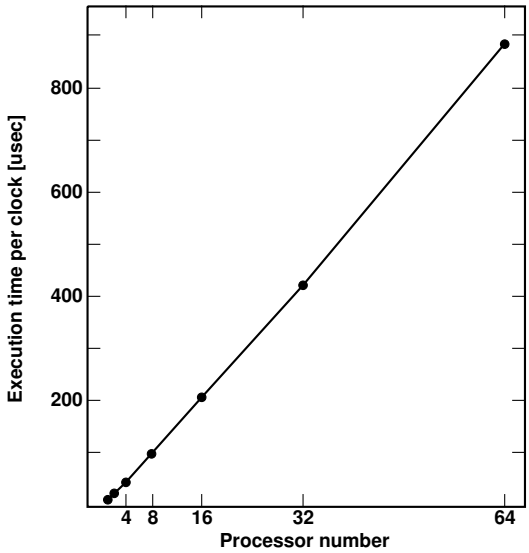

図 10: 1 クロックのシミュレーションに要する時間

図 10は、シミュレートされる並列計算機の時刻を 1 クロック進める処理に要する実行時間である。グラフからも明らかなように、この実行時間はプロセッサ台数にほぼ比例している。また、アプリケーションの台数効果にほとんど依存していない。したがって、このシミュレータの総実行時間はアプリケーションを実行するのに要する総クロック数とシステムのプロセッサ台数によってほぼ決定されることが分かる。

アーキテクチャ	実行時間 [μs]
ISIS — 共有キャッシュ	25.495
トレース出力用シミュレータ	45.151

表 1: ISIS との比較

表 1は、命令レベルシミュレータ ISIS との実行時間の比較結果である。対象となるアーキテクチャが異なり、アプリケーションの実行に要する総クロック数も異なっているため、1 クロックの処理に要する実行時間を比較した。この結果から、提案するライブラリを用いた方が実行時間が 77%長いことが分かる。対象の並列計算機が大きく異なるために一概には言えないが、ユニット間の通信方法を規定したこと、仮想関数を使用したことなどによるオーバヘッドによるものと思われる。これについては今後の検討課題としたい。

これらの結果から、提案するライブラリを使用して実装したシミュレータが実用的な速度で動作することが確認された。この評価では 64 プロセッサまでの並列計算機しか実行時間を確認していないが、シミュレーション時間がプロセッサ台数に比例していることから、256 プロセッサ程度までであれば現実的な時間でシミュレーションが可能であると思われる。

6 まとめと今後の予定

様々な並列計算機シミュレータで共用可能な機能を抽出した並列計算機シミュレータのためのアーキテクチャ非依存かつシミュレーション方式非依存なライブラリを提案した。本ライブラリを使用することで、シミュレータ実装者の負担を大幅に軽減することができる。

並列計算機内の機能ブロックをユニットとして定義し、様々なユニットを相互結合させることで並列計算機シミュレータを実現するよう設計した。また、ユニット間の結合と通信機構を規定するために、パケットとポートを定義した。

オブジェクト指向プログラミングと実行時の効率を両立させる必要性から、ライブラリの実装言語に C++言語を採用した。ユニット、パケット、ポートをそれぞれクラスとし、クラスライブラリの形で実装を行った。クラス間の共通性を継承を用いて記述することにより、実装コストの低減を図った。

また、本研究で提案するライブラリを用いてシミュレータを実装し、シミュレータの性能評価を行った。その結果、実用的な速度で動作するシミュレータを生成できることが確かめられた。

現在、本ライブラリを使用した並列計算機シミュレータの実装が行われている。1 つはマルチプロセッサチップを想定した、スヌープキャッシュを持つバス結合型並列計算機のシミュレータである。スヌープキャッシュ、共有バスを実装中である。もう 1 つは中規模並列計算機を想定した多段結合網シミュレータで、現在動作試験を行っている。これらの実装が終了し、マニュアルが整備された時点で本ライブラリを一般に公開する予定である。

参考文献

[1] Richard L. Sites, Anant Agarwal, “Multiprocessor Cache Analysis Using ATUM”, Proceeding of 15th International Symposium on Computer Architecture, 1988.

[2] Helen Davis, Stephen R. Goldschmidt, John Hennessy, “Multiprocessor Simulation and Tracing Using Tango”, Proceeding of International Conference on Parallel Processing, 1991.

[3] Rothberg. E, Smith. J. P, Gupta. A, “Working sets, Cache Sizes, and Node Granularity for Large Scale multiprocessors”, Proc. of the 20th ISCA, 1993.

[4] Mendel Rosenblum, Stephen A. Herrod, Emmett Witchel, Anoop Gupta, “Complete Computer System Simulation: The SimOS Approach”, IEEE Parallel and Distributed Technology, Fall 1995.

[5] “IDT R30xx Family Software Reference Manual”, Integrated Device Technology, Inc., 1994.

[6] “IDT79R3081 RISController Hardware User’s Manual”, Integrated Device Technology, Inc., December 11, 1992.

[7] 寺澤 卓也, “マルチプロセッサチップのためのキャッシュメモリに関する研究”, 平成 7 年度 慶應義塾大学理工学部計算機科学専攻 博士論文.

[8] 若林 正樹, 寺澤 卓也, 山本 淳二, 天野 英晴, “並列計算機シミュレータ ISIS の実装”, 平成 8 年 電子情報通信学会総合大会講演論文集, D-80.

[9] Toru Kisuki, Masaki Wakabayashi, Junji Yamamoto Keisuke Inoue, Hideharu Amano, “Shared vs. Snoop: Evaluation of Cache Structure for Single-chip Multiprocessors”, Euro-Par ’97.

[10] 亀井 貴之, “多段結合網に基づくキャッシュコヒーレンスネットワークの設計と実装”, 平成 8 年度 慶應義塾大学理工学部計算機科学専攻 修士論文.

[11] 安生 健一朗，中條 拓伯，小野 航，工藤 知宏，山本 淳二，西 宏章，木透 徹，天野 英晴, “分散共有メモリを持つ WS クラスタ: JUMP-1/3”, 並列処理シンポジウム JSPP’97.

[12] 佐藤 充，天野 英晴，安生 健一朗，周東 福強，西 宏章，工藤 知宏，山本 淳二，平木 敬, “超並列マシン JUMP-1 のための分散共有メモリ管理プロセッサ”, 並列処理シンポジウム JSPP’97.