相互結合網評価用命令レベルシミュレーション

小守 継夫 † 若林 正樹 ‡ 天野 英晴 ‡

† 日立公共システムエンジニアリング

‡ 慶應義塾大学理工学部

慶應義塾大学では並列計算機の性能評価を目的に, 並列計算機シミュレータライブラリ ISIS と相互結合網シミュレータライブラリ SPIDER を開発している. 本研究では, これらのライブラリを利用して分散共有メモリ型並列計算機モデルである NUMA のシミュレーションを行うため, 新たな機能ブロックとしてネットワークインタフェースを開発してライブラリの拡張を行った. 本稿では, ライブラリの拡張方法と, 実装した幾つかのシミュレータ上で SPLASH2 プログラム集を実行して性能評価を行った結果を示す. 本研究により, 様々な相互結合網について命令レベルシミュレーションによる性能評価を可能にした.

An Evaluation System of Interconnection Networks

Tsugio Komori† Masaki Wakabayashi‡ Hideharu Amano‡

†Hitachi Government and Public Corporation System Engineering
‡Keio University

An instruction level simulation library “ISIS” which is developed by Keio University supports to simulate a bus connected multiprocessor. On the other hand, “SPIDER” which is also developed by us supports to simulate an interconnection networks. Here, these two simulation libraries are combined to simulate a large scale multiprocessor connected with an interconnection network. We also implemented several distributed shared memory mulitprocessor simulator based on the new library. In this paper, the new extension method of library is described, and performance evaluation of the implemented simulator is shown.

1 はじめに

近年, 数百・数千台以上のプロセッサ要素から構成される並列計算機に関した研究, 実装が盛んに行われている. このような並列計算機では, 各プロセッサ要素間を結合する手段として直接網や間接網などの相互結合網を用いて構成することが多い. 相互結合網は, トポロジ, ルーティング方式, フロー制御方式等で特徴づけられ, これらのパラメータが適切に選択されない場合, ネットワークがシステム全体のボトルネックとなり, 高い並列処理効果を得ることができない. そこで, 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ている. その中で計算機を使った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段となる. シミュレーションによって並列計算機を評価する場合, 確率モデルシミュレーションを用いて概略の評価を行い, 詳細な評価はケースを絞ってトレース駆動型シミュレーションや命令レベルシミュレーションを行うことが理想である.

本研究室ではこのような目的で, 並列計算機シミュレータライブラリ ISIS[1] と相互結合網シミュレータライブラリ SPIDER[2] を開発している. ISIS はシミュレーション方式に非依存なライブラリであり, ISIS のライブラリを利用することで, 確率モデルや命令レベル方式による並列計算機のシミュレーショ

ンが行える．しかし, 結合網がバスに限定されているため, シミュレーションが行える計算機モデルはバス結合型マルチプロセッサのみである．一方, SPIDER のライブラリを利用することで, 様々な相互結合網についてのシミュレーションが行えるが, シミュレーション方式は確率モデルに限られる．現在, シミュレータを記述する際には, ISIS と SPIDER のライブラリを同時に利用することがはできないが, この 2 つのライブラリを結合する機能ブロックがあれば, 相互結合網で構成された並列計算機の命令レベルシミュレータを構築することができる．

本研究では, ISIS と SPIDER を結合する機能ブロックとしてネットワークインタフェースを開発し, ライブラリの拡張を行う．その際, シミュレーション対象とする計算機モデルは分散共有メモリ型並列計算機モデルである NUMA (Non-Uniform Memory Access model) を想定する．また, ライブラリを拡張した後に, 幾つかの相互結合網についてシミュレータを構築し, SPLASH2 プログラム集による性能評価を行う．

2 並列計算機シミュレータライブラリ ISIS

慶應義塾大学で開発された並列計算機シミュレータライブラリ ISIS は, 並列計算機の性能評価を目的としたシミュレータライブラリであり, 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である．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, 並列計算機内の個々の機能ブロックはユニットとして個別に実装されている．

ISIS は実装言語として, オブジェクト指向をサポートし高速動作が可能な C++言語を用い, 各ユニットをクラスライブラリの形で提供している．ユーザは, これらのユニットを評価する対象に合わせて選択, 使用することによりシミュレータの実装を簡単に行うことができる．また, ユニット間の共通機能に継承を用いることにより, 新しいユニットを簡単に実装することができるよう設計されている．

ユニット間の結合と通信のインタフェースには, 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の概念が導入されている．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これに相当する．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 1 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．

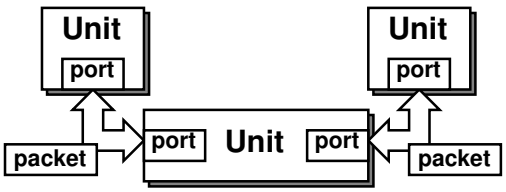

図 1: 結合と通信のモデル

現在は, MIPS 社の R3000 プロセッサ, R3010 浮動小数点演算コプロセッサ, バス結合型並列計算機用のスヌープキャッシュ[3] 等のユニットが実装されている．また, SPLASH2 プログラム集 [4] 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．しかし, ISIS は小規模並列計算機の性能評価を目的として開発されたため, 各ユニットの結合形態は, バス結合しか行うことができないという問題を抱えている．

3 相互結合網シミュレータライブラリ SPIDER

慶應義塾大学で開発された SPIDER は, 相互結合網の性能評価を目的に開発されたシミュレータライブラリである．相互結合網シミュレータを実装する上で再利用可能な機能ブロックをユニットとして提供している．図 2 に SPIDER が想定している並列計算機システムのモデルを示す．

SPIDER は, 結合網間のメッセージをパケット, 結合網の中間ノードをルータとしてクラス化している．パケットおよびルータは様々なルーティングアルゴリズム, パケット転送方式, メッセージのマルチキャスト等をサポートしており, さらにユーザ自身の手によって容易に機能変更が可能となっている．また, ルータ間の結合を自由に構成できるため, あらゆる

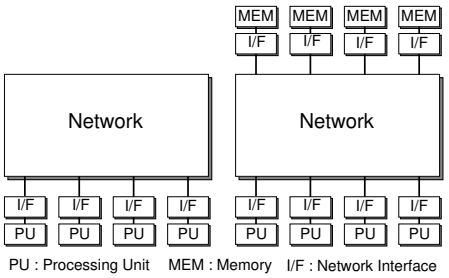

図 2: 評価対象モデル

トポロジに対応することができる. SPIDER のライブラリを使用すれば, 様々な相互結合網のシミュレーションが可能である. しかし, シミュレーション方式は確率モデル方式のみで, トレース駆動型や, 命令レベルによるシミュレーションは行えない.

4 シミュレーション対象モデル

本研究の目的は, 相互結合網で構成された並列計算機の命令レベルまたはトレースレベルのシミュレーションである. 本研究ではその中でも分散共有メモリ型並列計算モデルである NUMA をシミュレーション対象とする. NUMA は, プロセッサとメモリからなる PU (Processing Unit) を, 直接網や階層バスを用いて接続し, 他 PU のメモリも自分のメモリと同一の空間でアクセスできる機構をつけたシステムである. 図 3 に直接網を用いた NUMA の例を示す. ここではコンシステントなキャッシュを持たない単純な NUMA を想定している.

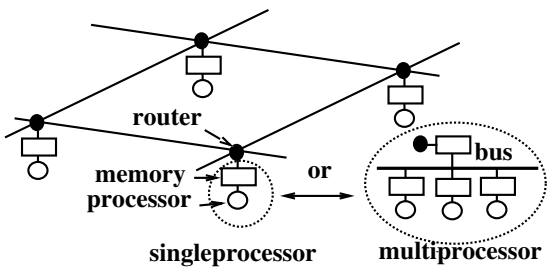

図 3: 直接網を用いた NUMA

ノードの基本的な構成要素は, 単一の PU の場合と, バス結合型マルチプロセッサの場合がある. この形態は, すべてのプロセッサから, アクセスできる共有メモリを持つが, アクセスに要する時間がアドレスによって異なる.

5 設計

5.1 ライブラリ拡張に必要な機能ブロック

4 章で述べた NUMA のシミュレータを構築する場合に必要となる機能ブロックを図 4 に示す.

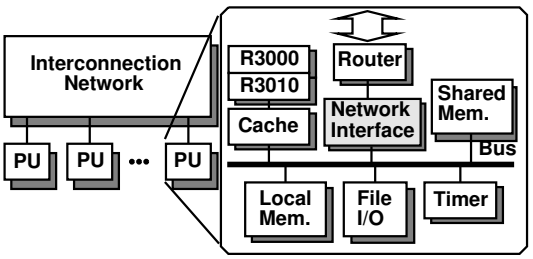

図 4: NUMA シミュレータの機能ブロック図

ノードの構成には, R3000 プロセッサ, R3010 浮動小数点演算演算コプロセッサ, キャッシュ, メモリ等の ISIS で提供されている機能ブロックを使用する. また, 相互結合網部分は SPIDER で提供されている機能ブロックであるルータを使用する. したがって, NUMA のシミュレーションを行うには, ノードをネットワークに結合するための機能ブロックであるネットワークインタフェースが必要となる.

5.2 機能概要

ネットワークインタフェースはバス上に送信されるパケットを監視し, パケットの内容が他ノードのメモリを対象にしたメモリアクセスの場合, 該当するノードの宛先情報などを付加してネットワークパケットを生成し, ネットワークに送信する. また, ネットワークからメモリアクセス要求のパケットを受信した場合, バスに接続されているメモリに対して該当するメモリアクセス処理を行う.

5.3 結合と通信モデル

ネットワークインタフェースは図 4 に示す通り, バスとルータに結合される. 結合する際は, ISIS で提供されている bus_port クラス, virtual_channel_ input クラス, virtual_channel_output クラスを使用する. 図 5 にバス, ルータとの結合と通信モデルを示す.

5.3.1 バス上での通信方法

バス上で通信を行う場合, 通信ポートを表す bus_port を結合することで, バスとしての通信路が

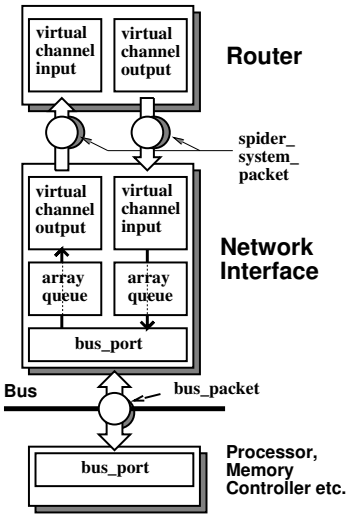

図 5: 結合と通信モデル

構築される. bus_port クラスにはバス上の通信に必要なパケット転送などの関数が実装されている. バス上の通信は, バスのデータを表す bus_packet クラスを利用して行う. 表 1 に bus_packet が表すパケットの一覧を示す. バストランザクションにはリードトランザクションとライトトランザクションがある. 図 6 にトランザクション手順を示す.

表 1: packet クラスのパケット一覧

read_request	read 要求
read_grant	read 要求を受理可能
read_nack	read 要求を受理不能
read_ack	read 要求を受理する準備完了
read_data	read データ
write_request	write 要求
write_grant	write 要求を受理可能
write_nack	write 要求を受理不能
write_ack	write 要求を受理する準備完了
write_data	write データ

リードトランザクションでは, プロセッサがバスのオーナになり, リード要求を示す read_request パケットをバスに送信する. ネットワークインタフェースは, 他ノードのメモリを対象にした要求の場合, 要求の受理を示す read_grant パケットをバスに送信する. ただし, ネットワークインタフェース内のバッファが一杯の場合は受理不能であることを示す read_nack パケットをバスに送信する. read_grant パケットを受信したプロセッサは, 一旦バスを開放して, 応答準備が整ったことを示す read_ack パケットを待つ. ネットワークインタフェースは, この要求を該当するノードに送信し, リードデータが受信できた段階でバスのオーナになり, 応答準備が整ったことを示す read_ack パケットをバスに送信する. その後リードデータを示す read_data パケットをバスに送信してリードトランザクションが終了する.

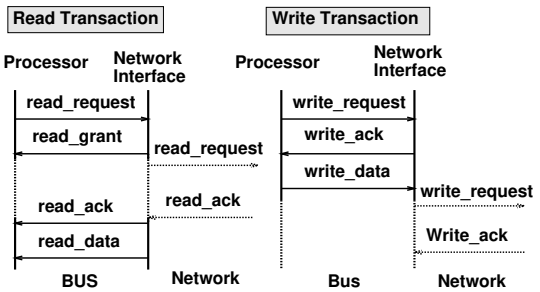

図 6: トランザクション手順

ライトトランザクションについては, 基本的には図 6 に示す手順で行われる. ただし, write_request パケットを受信した時に, 格納するバッファが一杯の場合には, 受理可能を示す write_ack パケットではなく, 受理不能を示す write_nack パケットをバスに送信する. また, プロセッサがライトデータである write_data を送信した時点でバスは解放される.

5.3.2 ネットワーク上での通信方法

ネットワーク上で通信を行う場合は, ネットワークインタフェースとルータの virtual_channel_input クラスと virtual_channel_output クラスのオブジェクトをそれぞれ結合する . 送受信されるパケットは, ネットワークパケットを示す spider_system_packet を使用する. パケットを出力する場合は, 接続先のルータの入力バッファに空きがあるかどうかを virtual_channel_output クラスの is_full() 関数を使用して確認し, put() 関数を使用してパケットを送信する. また, ルータからパケットが到着しているのを確認するには, virtual_channel_input クラスの is_empty() 関数を使用し, パケットを入力する場合は get() 関数を使用する.

5.4 メモリコンシステンシ

本研究では, マルチプロセッサでの共有メモリへのアクセスの正当性を, ハードウェア , つまりネットワークインタフェースだけで実現させるのではなく, 実行するソフトウェア側で対応するようにする. 具

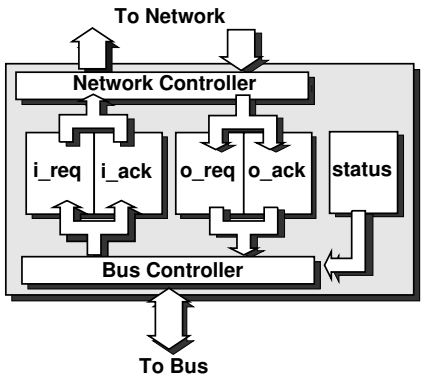

図 7: ネットワークインタフェースの内部構造

体的には, 共有メモリや同期メモリについてアクセスの完了を保証しなければならない場合, ソフトウェア側でそれまでのアクセスが全て完了しているかどうかネットワークインタフェースに問い合わせ, アクセスが完了したことを確認した時点で次のステップの実行に移るようにする. これを実現するため, ネットワークインタフェースに, 要求を受理したがまだそのアクセスが完了していないトランザクションの数を格納するステータスレジスタを設け, 同期関数内でそのレジスタを参照することによって共有メモリへの正当性を保証する.

6 実装

5 章の設計に基づき, ネットワークインタフェースを実装した. 図 7 に本研究で実装したネットワークインタフェースの内部構造を示す. ネットワークインタフェースはバスとの通信を制御するバスコントローラ (BusController), ネットワークとの通信を制御するネットワークコントローラ (NetworkController), 4 つのパケット格納バッファ(i_req,i_ack,o_req,o_ack) と, ステータスレジスタ (status) から構成される. なお, シミュレータで使用する場合は, 共有メモリのアドレスマップを作成し, そのアドレスマップをネットワークインタフェースに接続する必要がある. ネットワークインタフェースはこのアドレスマップによって, バス上に送信されるメモリアクセス処理について, どのノードに接続されているメモリを対象にしているかを判別する. 以下, 構成要素の説明を行う.

6.1 バスコントローラ

バスコントローラは, バスとの通信を制御する. バス上のパケットを監視し, パケットの内容が他ノードに接続されたメモリを対象にしたメモリアクセス要求の場合に, その要求を受理し, ネットワークインタフェース内の入力要求バッファ(i_req) へその要求を格納する. また, 出力要求バッファ(o_req) にネットワークからのメモリアクセス要求パケットが格納されている場合は, バッファからパケットを取りだし, バスに対して該当するメモリアクセス要求を送信する. そして, その結果を入力応答バッファ(i_ack) へ格納する. 他ノードからのメモリアクセス結果は出力応答バッファ(o_ack) に格納されるので, バッファからそのパケットを取り出し, 要求を出したプロセッサに対してその結果を送信する.

6.2 ネットワークコントローラ

ネットワークコントローラはネットワークとの通信を制御する. バスからの要求パケットや応答パケットが入力要求バッファ(i_req) や入力応答パケット (i_ack) に格納されている場合, バッファからそのパケットを取り出し, ネットワークに送信する. また, ネットワークからパケットを受信した場合, 要求パケットは出力要求バッファ(o_req) へ, 応答パケットのは出力応答バッファ(o_ack) へ格納する.

6.3 パケット格納バッファ

パケット格納バッファはバスとネットワークのパケットの中継に使用されるバッファであり, バスからの要求パケットを格納する入力要求バッファ(i_req), バスからの応答パケットを格納する入力応答バッファ (i_ack), バスへの要求パケットを格納する出力要求バッファ(o_req), バスへの応答パケットを格納する出力応答バッファ(o_ack) がある. パケットを到着順に処理するため, FIFO 形式のバッファとなっている.

6.4 ステータスレジスタ

ステータスレジスタはメモリコンシステンシを実現するために設けられたレジスタであり, ソフトウェアからレジスタの内容を参照することができる. レジスタには, 要求を受理したが, まだそのアクセスが完了していないトランザクション数が格納される.

7 評価

本研究の目的である, NUMA のシミュレーションが行えることを確認するため, 実装した NUMA シミュレータ上でアプリケーションを実行して性能評

価を行った. 性能評価には図 4 のシミュレータを使用する. その際の構成は, 1 ノードに 1 プロセッサとして 16 ノードを相互結合網で接続する. 相互結合網としては, ハイパキューブ, 2 次元トーラス, 多段結合網 (MIN) を使用する. ネットワークに関する設定は仮想チャネル数 1, 但し 2 次元トーラスはデッドロックフリーを保証するために 2, バッファ数 1, パケット方式は virtual-cut-through 方式, バンド幅 1(flit/clock), メッセージ長は 5 フリットとする. シミュレータ上で実行するアプリケーションには SPLASH2 プログラム集の FFT を使用した. FFT は高速フーリエ変換を行うプログラムである. 問題のサイズである要素数は 4K とした. シミュレータを実行するホストマシンには, Pentium-II 450MHz の IBM PC 互換機, メモリ容量は 128MB, OS は FreeBSD 2.2.8-RELEASE である.

本シミュレーションでは, ネットワークインタフェースのバッファ容量を変化させて全体の性能にどれだけ影響があるかを評価する. 図 8 にバッファ容量の変化に伴う, アプリケーションの計算に要する時間を示す. 横軸はバッファ容量, 縦軸はイニシャライズを除く, 並列処理部分の計算に要した時間を表しており, 単位は 100 万クロックである. 図から分かる通り, ネットワークインタフェースのバッファ容量を大きくすることで計算に要する時間が短くなる. しかし, バッファの容量を 20 以上にしてもネットワークの転送能力などの問題によりそれ以降の計算時間はほとんど変わらない. また, シミュレーションの実行時間は, ハイパーキューブのバッファ容量が 20 の時に, イニシャライズを含めたトータル時間 9643630 クロックを実行するのに, 1639 秒要している. これは,16 プロセッサの並列計算であれば 1 秒間に 5883 クロックのシミュレーションが行えることを示している.

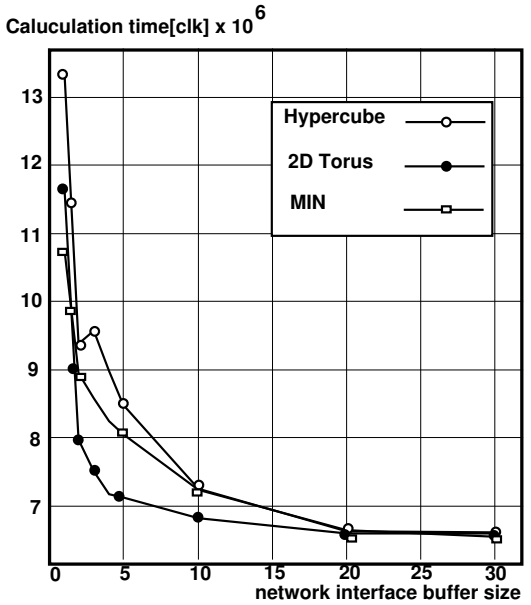

図 8: バッファ容量による計算時間の評価

8 おわりに

本研究では NUMA の命令レベルシミュレーションを目的としてネットワークインタフェースを開発してライブラリの拡張を行った. また, 実装した NUMA シミュレータ上で,SPLASH2 プログラム集を実行して性能評価を行うことにより, 本研究の目的が達成されたことを確認した.

今後の課題としては, CC-NUMA を実現するための , キャッシュ機能を有したネットワークインタフェースの開発, また, 共有メモリを持たずにメッセージパッシング方式によって他プロセッサと通信を行う NORA(NO Remote Memory Access model) を実現するためのネットワークインタフェースの開発が挙げられる.

参考文献

[1] 若林 正樹, 米田 卓司, 天野 英晴,“並列計算機シミュレーションライブリの提案”, 電子情報通信学会, CPSY 97, Dec. 1997

[2] 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, CPSY 98, Jan. 1998

[3] 井上 敬介, 若林 正樹, 木村 克行, 天野 英晴,“シングルチップマルチプロセッサ用半共有型スヌープキャッシュ”, 電子情報通信学会技術研究報告, CPSY98-59, Aug. 1998

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