ICOT Technical Report: TR-563

TR-563

疎結合並列マシンMulti-PSI上でのKL1分散処理系におけるプロセッサ間通信の評価

中島克人，稲村　雄，市吉伸行

May, 1990

©1990, ICOT

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

疎結合並列マシン Multi-PSI 上での KL1 分散処理系における プロセッサ間通信の評価

中島克人
三菱電機（株）
稲村 雄、市吉 伸行
（財）新世代コンピュータ技術開発機構

1 はじめに

第５世代コンピュータプロジェクトの一環として開発されたマルチ PSI [7] には２つの目的がある。即ち、並列ソフトウェア研究のためのツールを提供することと、並列言語 KL1 の処理系実装などのための実験マシンに用いることである。

マルチ PSI はハードウェア実装が容易なことから２次元格子（メッシュ）結合の非メモリ共有方式を採用しているが、これは拡張性のある並列マシンの実現手法の研究に格好のテストベッドといえる。

マルチ PSI には GHC[8] をベースにしたストリーム AND 型の並列論理型言語 KL1[2] の処理系が実装されたが [5]、設計に当たってはマルチ PSI の最大プロセッサ数（64PE）を意識することなく、より大規模な並列システムに適用する方式が各種採用されている。即ち、一回の通信コストが少々大きくなろうとも、通信の回数が削減できることが優先されている。

本稿では非メモリ共有マシン故の通信コストの大きさと、大規模システム向きの処理方式を採用したことによる通信コストの大きさを分析し、また、それらによるオーバヘッドがプログラム実行時にどの程度になるかを評価した。また、PE 間接続ネットワークのトラフィック量の評価も行なった。

以降、２節では PE 間通信のためのネットワーク制御アーキテクチャと KL1 処理系に関して簡単に紹介し、３節では評価結果を示し、４節ではまとめを行なう。

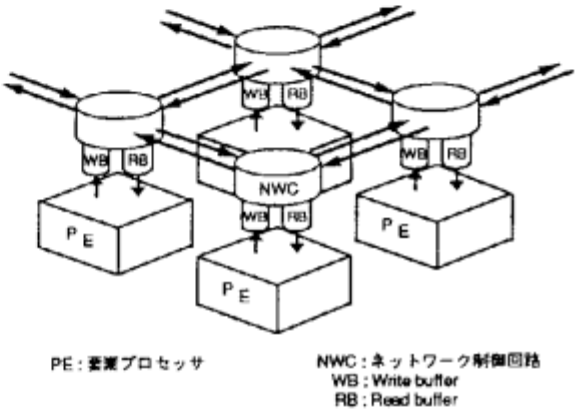

図 1: マルチ PSI おける PE 間結合方式

2 マルチ PSI システムの概要

2.1 ネットワーク制御アーキテクチャ

2.1.1 PE とネットワーク制御ハードウェア

マルチ PSI の要素プロセッサ（PE）は PSI-II[6] の CPU と同一で、４K 語のキャッシュメモリ、最大 16 M 語の主記憶（ローカルメモリ）を備えた水平型マイクロプログラム制御のプロセッサである。PE の速度は KL1 の append プログラムで約 130 KRPS（Kilo Reduction Per Second）である。

それぞれの PE は PE 間メッセージ通信をサポートするための専用のネットワーク制御回路を備えている。この回路は隣接 PE と自 PE とを接続するために５組の入出力チャネル（各１バイト幅）を備え

Evaluation of Inter-processor Communication in the KL1 Implementation on the Multi-PSI
Katsuto NAKAJIMA†,
Yū INAMURA‡, Nobuyuki ICHIYOSHI†
†Mitsubishi Electric Corporation
‡ICOT

ている（図 1）。メッシュ結合では種々のトポロジに比べ、遠隔の PE 同士が通信する場合の途中ノードの通過数が大きい。そこで、通過メッセージによる PE 処理の妨害防止と転送速度向上のため、各ネットワーク制御回路は自動ルーチング機能を備えている。即ち、自 PE 宛以外のメッセージ転送は PE の処理とは独立に高速に行なわれる。サイクル時間は PE と同じ 200ns （5 M バイト/s）であり、各ノードでの通過メッセージの遅延は理想状態（チャネル待ちなしの状態）で 5 サイクルである。

自 PE 宛、もしくは自 PE 発のメッセージは 4 K バイトの 2 つのバッファ、*Read Buffer* と *Write Buffer* を介して、PE のマイクロプログラムで処理される。

2.1.2 メッセージの基本操作

基本的なメッセージ操作のためにマイクロコード化された低レベルのルーチン（以後、Basic message handler と称する）が用意されている。これは、メッセージのヘッダやテールを付加もしくは解読したり、32 ビットのデータをバイトシリアルに変換してメッセージを組み立てたり、その逆の操作を行なったりする。これらの処理は後述する KL1 の言語処理方式とは無関係の機能で、言い替えれば、マルチ PSI のネットワーク制御回路に対して特有の処理である。

一連のメッセージバイト列が目的地のノードに到着すると、それらはネットワーク制御回路により一旦 Read Buffer に蓄えられる。そして、割り込みにより PE 内の Basic message handler が呼び出され、このルーチンがメッセージ単位で *Read Packet Buffer* と呼ばれるメモリ上の十分に大きな領域に移動する。これは Read Buffer が満杯の状態を避けるためである。Read Buffer が満杯であると、このノードへのメッセージが滞り、その停滞がネットワーク上で伝搬することにより、著しい性能低下やデッドロックを引き起こすことがあるからである。

Read Packet Buffer の中のメッセージはリダクションの切れ目でデコードされ、メッセージに応じた処理、即ち KL1 処理系に依存した処理が、別のルーチン（以後、KL1 Encoder/decoder と称する）で行なわれる。メッセージ送出に際しては、もしそのメッセージ全体を格納するスペースが Write Buffer に無ければ、そのスペースができるまで Busy 待ちする。その場合でも、Read Buffer に到着するメッセージの Read Packet Buffer への取り込みだけは行なわれる。

表 1: 代表的な PE 間メッセージ

メッセージ	機能
throw_goal	KL1 ゴールの送出
read	外部参照データの値の読出し要求
answer_value	read メッセージに対する返答
unify	外部参照データのユニファイ要求
release	外部参照データの WEC の返却

2.2 KL1 分散処理系

2.2.1 ゴールの送出と輸出入

KL1 のゴールを他の PE に向けてフォークする場合には、以下のようなゴール送出プラグマをボディ・ゴールに指定する。

$$\ldots, \mathrm{body}(\text{引数})@\mathrm{processor}(\mathrm{PE}), \ldots$$

マルチ PSI は非メモリ共有マシンであるため、ゴール送出に際しては実際にデータが移動される。KL1 は既定義変数への破壊的書き込みを許さない論理型言語であるため、ゴール引数が既定義データの場合はコピーを作ることが許されるが、未定義変数の場合は変数の同一性を保つためその変数へのポインタ（外部参照ポインタと称する）を生成し、それをゴールに付加して送出することになる。外部参照ポインタを生成することを輸出と称し、それを他の PE が受け取ることを輸入と称している。送出ゴールは throw_goal メッセージにより他 PE に運ばれる。なお、引数が定義済みであってもそれが構造体である場合も外部参照ポインタを輸出することにしている。行き先 PE でその引数の値が用いられるとは限らないからである。

行き先 PE で輸入した外部参照ポインタの値が必要になった場合には、read メッセージが輸出元に送られ、輸出元でその変数の値が定義された時点で answer_value メッセージにより値の（コピーの）返信が行なわれる。また、輸入した外部参照ポインタへのアクティブ・ユニフィケーションでは unify メッセージが発信され、輸出元でユニフィケーションが行なわれる。表 1に代表的な PE 間メッセージを示す。

2.2.2 輸出入表

KL1 では変数の書き換え（再代入）や、バックトラックによるメモリの再利用が出来ないので、KL1

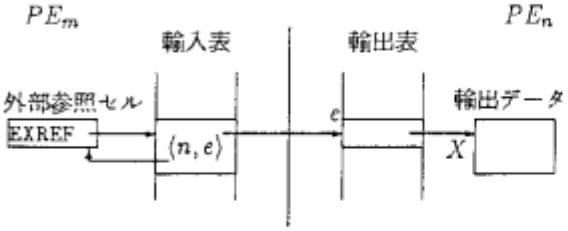

図 2: 外部参照ポインタと輸出入表

処理系の設計においてはメモリ消費速度を極力抑えること、即ち効率の良いGCを実現することが最も重要な課題の一つである。システム全体のゴミを一度に回収する大域GCはいずれ必要であるとしても、これは全てのPEでの計算を停止させる必要があり、オーバヘッドが大きい。従って、大域GCの頻度はできるだけ抑えて、GCをできる限り局所的に行なうことが得策となる。

PE内部の局所GCとしてマルチPSIではMRB方式[1]と一括GCを併用しているが、輸出した外部参照ポインタも一括GC時のマーキング・ルートとしなければならないことから、これを管理するために輸出表を設けている。また、外部参照ポインタをPEの内部アドレスとは独立な $< n, e >$（ただし、nは輸出PEの番号、eは輸出表のエントリ位置）なる形で表現することにより、一括GCによるPE内部でのデータの移動が他PEなどに影響を与えないようにしている（図 2）。輸出表は外部参照ポインタからPEの内部アドレスへの変換テーブルの機能も果たしている。

2.2.3 PE間即時GC

輸出表から指されたゴミのセル、即ち、他のすべてのPEが使用済みとなった輸出データを即時に回収し、大域GCの頻度を削減するためには、輸出表のエントリ自身の回収をしなくてはならない。そのため、重み付きリファレンス・カウンティング方式[9]を応用した、WEC（Weighted Export Counting）方式によるGCが採用されている[4]。この方式は通常のレファレンス・カウント方式に比べカウント値のメンテナンスのためのメッセージを半分程度にできるという特長がある。

この方式では、参照カウント値（WEC）を輸入側でも保持するため、輸入表なるものも設けられている（図 2）。輸入側で使われなくなった外部参照ポインタのWECは`release`メッセージにより輸出側に返却される。

3 計測と評価

本節では1PE, 4PE, 8PE, 16PE, 32PE, 64PEのマルチPSIを用いた、プロセッサ間通信のコスト、プログラム実行時のそれらによるオーバヘッド、プロセッサの稼働率、通信トラフィックなどの評価結果について述べる。

3.1 メッセージ処理のコスト解析

図 3 に主なメッセージの処理コストを示す。これらは、マイクロ命令の所要ステップ数にサイクル時間、即ち200 nsを乗じる事により得た所要時間であり、キャッシュ・ミスによる待ち時間などは含まれていない。

図において、Basic message handlerは前述のようにKL1言語処理系の方式とは無関係なネットワーク制御ハードウェアを操作するために必要な処理ステップを表している。KL1 encoder/decoderとはゴール引数のデータタイプに応じてエンコード／デコード（可変長バイトの処理）したり、ゴールの呼びだしコードアドレスなどをエンコード／デコードしたりする処理、また、輸出入表などへの登録参照などのステップを表す。

図 3(a) は、引数がアトムと2つの外部参照ポインタの3つであるようなゴールのための`throw_goal`メッセージの送受コストを表す。送出に85 μs、受信に130 μsを要している。Basic message hanlderでの所要時間がKL1 encoder/decoderでの所要時間とほぼ等しく、かなり大きいことが分かる。また(b)において、この65バイトのメッセージをRead BufferからRead Packet Bufferに移動するためのステップ *Copy_RPKB* が全体の14%に及ぶことも分かった。

本来ネットワーク・チャネルのバンド幅からいうと、65バイトのメッセージの送受に13μs（(a)の15%、(b)の10%）しか必要ないことから、Basic message handler、KL1 encoder/decoderの何れにおいても相当なオーバヘッドが存在する。

後に表 2 で示すように、Pentominoプログラムでは1PE上で約40 KRPS（25μs／リダクション）であるので、(a)および(b)は、3.4及び5.2リダクション分のコストがかかることになる。

図 3(c),(d),(e) および (f) は`read`メッセージの送受および、`answer_value`メッセージの送受のコストを示す。`answer_value`メッセージでの返答データは典型例としてCARがアトムで、CDR

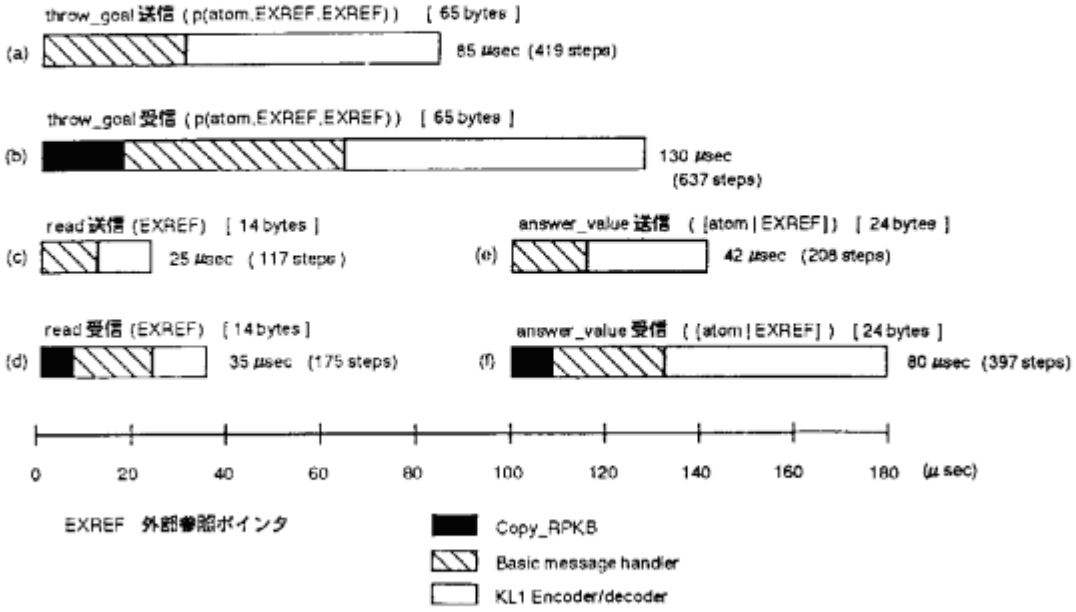

図 3: メッセージ処理コスト

が外部参照ポインタであるようなリストを選んだ。

14 バイトから成る read メッセージの送受にはそれぞれ 25μs と 35μs 要している。これは Pentomino でのリダクション・コストの 1 ～1.4 倍である。また、24 バイトの answer_value メッセージの送受にはそれぞれ 42μs と 80μs 要している。こちらは Pentomino リダクションの 1.7 ～3.2 倍である。

14 バイトまたは 24 バイトのメッセージの送受のみに 2.8 μs または 4.8 μs かかるが、それらは全体の処理コストの 6 ～11% に相当する。この比率は throw_goal メッセージの場合と大差無い。つまり、メッセージ処理コストの大部分はマイクロルーチンの処理によるものであり、そのうち約半分はハードウェア操作のための処理、残り半分は KL1 処理系独特の処理で占めてられていることが分かる。

なお、Copy_RPKB はネットワーク上の渋滞を避けるための処理であり、Read Buffer に新たに到着するメッセージのためのスペースがある間（従って殆んどの場合）はこの処理は省略が可能である。これにより各メッセージの受信処理コストは 10～18% 程度削減できることも分かる。

3.2 ベンチマークプログラムを用いたプロセサ間通信の測定

前項ではメッセージ処理の個々のコスト解析を行なった。本節では 2 つの異なったタイプのベンチマークプログラムを用いて、メッセージ処理に関する動的特性を評価する。

3.2.1 ベンチマークプログラム

以下の 2 つのプログラムを使用した。

- **Pentomino:** 5 × 8 の大きさの詰め込みパズルの解を全て求める。即ち、5 × 8 の大きさの長方形に 1 × 1 の大きさの正方形を幾つかつないだ様々な形の部品をピッタリとはめ込む方法を全て探す。
- **Bestpath:** 160 × 160 のグリッド状のネットワークでの最短経路を求める。ネットワークの各ノード間は非負のランダムなコストを持つエッジで結ばれており、プログラムではあるノードを出発点として、それ以外の全てのノードに対する最短経路を求める。

Pentomino では部品の置き方に関する OR 木の全面探索を行なう。一つのマスタ PE がルート・ノー

ド、即ち空箱の状態から始めて探索木のある深さまで進み、それから先のサブ木を各PEに出来るだけ公平に分配する、いわゆる動的負荷分散を行なう[3]。

64PEを用いて実行する場合、8PEずつのグループ分けを行ない、マスタPEはまずサブ木をそのPEグループごとに分配する。各PEグループ内ではそこで新たに生成される子サブ木を各PEに分配し、負荷調整を行なう。

Bestpathではまず乱数を用いて問題とするグリッドを生成し、その後最短経路のアルゴリズムを実行する。本稿の測定では後者のフェーズのみを対象とした。

負荷分散は静的に行なっている。即ち、グリッド全体を小さな領域に分割して、各PEに固定的に割り付けている。具体的には、グリッドを 40×40 のブロックを単位として、16個のブロックに分け、それぞれのブロック内部をPEの数に応じて平面的に分割してマッピングする。例えば64 PEの場合、一つのPEが担当するブロック内の広さは 5×5 で、それが16個あることになる。

3.2.2 メッセージの動的頻度

表2にPE間メッセージの種類別頻度を示す。リダクション総数やリダクション速度なども合わせて示す。

Pentominoではread, answer_value, releaseそしてunifyの4つのメッセージが大部分を占める。Bestpathではそれらにthrow_goalが加わる[1]。これらのプログラム実行中における各PEでのメッセージの発信頻度は64 PE上で88リダクション（Pentomino）もしくは6リダクション（Bestpath）に一回の割合である。

3.2.3 通信オーバヘッド

本節では通信によるオーバヘッドがベンチマークプログラムの実行速度にどの程度の影響を与えているかを解析する。

プログラムの実行時間を以下のように詳細化して解析した。マルチPSIの各PEは、予め指定されたアドレスのマイクロ命令を何回実行したかを計るカウンタをそれぞれ備えている。PEはまた、システム立ち上げ時に同時に初期化されるカレンダクロック（1/16 msの精度）も備えている。これらを用い、また、各PEがアイドル状態を出入りする時刻を記録することにより、以下のような項目を測定した。

(a) アイドル状態以外でのマイクロ命令の実行ステップ総数（各PE）

(b) メッセージ処理に要したマイクロ命令の実行ステップ数（各PE）

(c) 実行時間（プログラムの開始から全PEでの終了まで）

(d) アイドル時間合計（各PE）

PEの稼働率は $(c-d)/c$ として定義される。c と $d + a \times$（サイクル時間 $= 200$ ns）の違いはキャッシュミス・ペナルティによるものである。$b \times$（サイクル時間）はキャッシュミス・ペナルティを除いた通信オーバヘッド時間である。

図4にPEの実行時間の解析結果を示す。図は全PEの平均値であり、棒グラフの中のComputing（$a \times 200$ ns）は問題を解くための本来の計算処理、Msg handling（$b \times 200$ ns）はPE間通信のためのオーバヘッド、Cache miss（$c - d - a \times 200$ ns）は計算処理とPE間通信処理におけるキャッシュミスによるメモリ待ち時間の合計、Idle（d）はそれ以外の時間、即ち、アイドル時間を表している。

Pentominoではプロセッサ間通信オーバヘッドとキャッシュミス・ペナルティは非常に小さく、PE台数に比例した速度向上を得られない主な理由はPEのアイドルであることが分かる。しかし、負荷分散が比較的うまくいっているため、アイドルの比率はそれほど大きくない。

Bestpathではも64 PEでのアイドルの比率は小さいが、計算時間（Computing）の比率はかなり小さい。また、通信オーバヘッドのみならずキャッシュミス・ペナルティが相当大きい。これはメモリ上のワーキングセットが非常に大きいことによると考えられる。また、PE数が増加するにつれ通信オーバヘッドの比率が増大している。これはPE間を跨る最短経路情報メッセージの数が、グリッドブロック内のPE分割の境界の長さ、即ち、PE数の平方根に比例することから説明される。これは表2におけるメッセージ数にも表れている。

以上まとめると、Pentominoでは通信オーバヘッドは小さく、これはOR並列型の問題一般に言えるであろうこと、また、静的負荷分散を行なっているBestpathでは稼働率低下防止のためにはグリッドを多くの、従ってより小さなブロックに分割するの

[1]何故なら、グリッドのノード間を行き来する最短経路情報の伝達メッセージはゴールの形で実現されているからである。

表 2: メッセージの動的頻度とリダクション数

Pentomino (39 3 KRPS / 1 PE)			
メッセージ	4 PE	16 PE	64 PE
read	10,408 (27.7 %)	24,746 (31.9 %)	30,736 (32.4 %)
answer_value	10,408 (27.7 %)	24,704 (31.9 %)	30,581 (32.3 %)
release	6,843 (18.2 %)	12,333 (15.9 %)	13,641 (14.4 %)
unify	2,379 (6.3 %)	7,504 (9.7 %)	11,129 (11.7 %)
throw_goal	1,191 (3.2 %)	1,677 (2.2 %)	1,768 (1.9 %)
etc.	6,396 (16.9 %)	6,542 (8.4 %)	6,895 (7.3 %)
合計	37,625 (100 %)	77,506 (100 %)	94,750 (100 %)
msg. 平均長	21.1 byte/msg.	20.4 byte/msg.	20.2 byte/msg.
実行時間	54,634 ms	14,615 ms	4,345 ms
総リダクション数	8,317 K	8,332 K	8,340 K
リダクション /s	152.2 KRPS	570.1 KRPS	1,919.4 KRPS
リダクション /msg.	221	108	88
msg. バイト数 /s	14.5 K	108.1 K	440.5 K

Bestpath (23.4 KRPS / 1 PE)			
メッセージ	4 PE	16 PE	64 PE
read	12,396 (27.5 %)	28,780 (27.8 %)	66,899 (27.7 %)
answer_value	10,156 (22.6 %)	23,980 (23.2 %)	56,980 (23.6 %)
unify	8,439 (18.8 %)	19,190 (18.6 %)	43,370 (18.0 %)
release	6,198 (13.8 %)	14,390 (13.9 %)	33,446 (13.9 %)
throw_goal	6,198 (13.8 %)	14,390 (13.9 %)	33,446 (13.9 %)
etc.	1,618 (3.5 %)	2,694 (2.6 %)	7,263 (2.9 %)
合計	45,005 (100 %)	103,424 (100 %)	241,404 (100 %)
msg. 平均長	27.0 byte/msg.	27.2 byte/msg.	27.0 byte/msg.
実行時間	10,655 ms	4,062 ms	1,691 ms
総リダクション数	987.7 K	1,213.6 K	1,505.2 K
リダクション /s	92.7 KRPS	298.8 KRPS	890.1 KRPS
リダクション /msg.	21.9	11.7	6.2
msg. バイト数 /s	114.0 K	692.5 K	3,854.3 K

が望ましいが、ブロック数の平方根に比例して通信オーバヘッドが大きくなるため、トレードオフが存在し、今回の規模の問題では台数効果が 64PE で頭打ちになっている。

3.3 ネットワーク・ハードウェアの評価

3.3.1 ベンチマーク実行におけるネットワーク・トラフィック

Pentomino では PE グループ内でのメッセージのやりとりが大きい比率を占めるため、メッセージの平均移動距離は、隣の PE に移動するのを距離 1 とすると約 2 となる。システム全体でのチャネル数は $L = \sqrt{(\text{PE数}=64)} = 8$ とすると、

$$(\text{チャネル数}) = 4 \cdot L \cdot (L-1) = 224$$

で表わされるので、64 PE での平均ネットワーク・トラフィックは、

$$\frac{441\ \text{KB} \times 2}{(\text{チャネル数}) = 224} = 3.9\ \text{KB/s}$$

となる。これは 5MB/s のチャネルバンド幅の 0.08% と非常に小さい。それぞれの PE グループの負荷分散を行なうグループ内マスタ PE にはメッセージの送受が平均の 8 倍だけ集中するが、それでもなおトラフィックは十分小さい。

Bestpath では大部分のブロック間メッセージは隣接 PE からのものであるので、平均の移動距離は

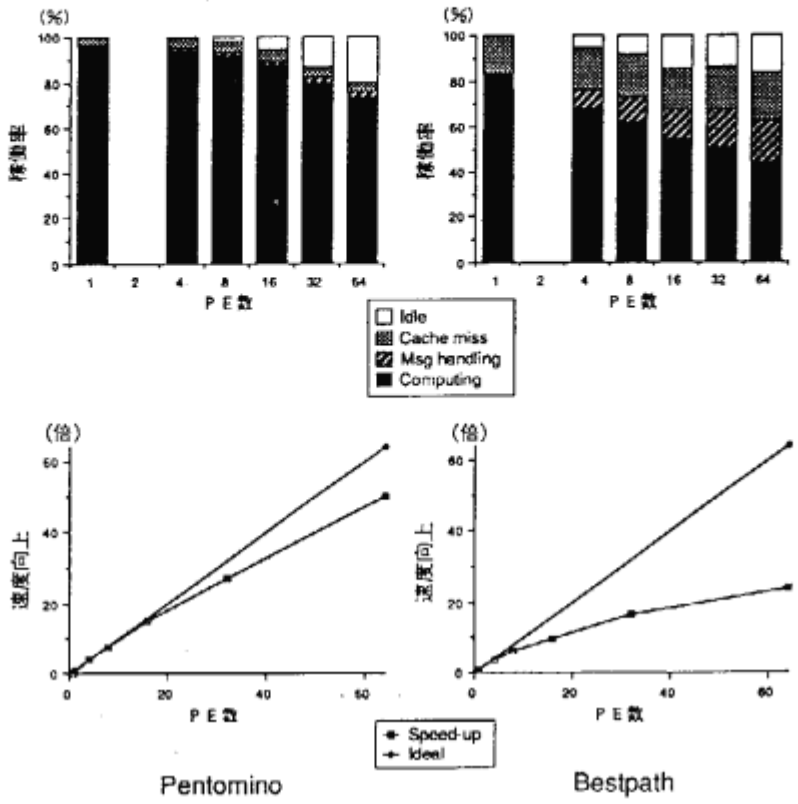

図 4: 稼働率とスピード・アップ

1余りと見積もられる。従って、平均トラフィックは以下のように計算される。

$$\frac{3{,}854\ \mathrm{KB} \times 1}{224} = 17.2\ \mathrm{KB/s}$$

これは、チャネル・バンド幅の 0.3% に相当する。問題のグリッドは静的に PE にマッピングされるので、最短経路情報のメッセージの伝達する波面に沿ってトラフィックのホットスポットができると考えられる。その波面の長さはおおよそ PE 総数の平方根に比例するため、ホットスポットもその程度分散する。従って、実際のトラフィックが上記の計算値よりも悪いとしてもやはり十分に小さい。これは、これらのベンチマークプログラムの実行に関しては、ネットワーク制御回路が、要求されるよりもかなり上回る性能を有していることを意味する。

3.3.2 大規模システムに向けての見積もり

次にマルチ PSI を大規模化（例えば 1,000PE 規模）した場合のネットワーク・トラフィックに関する見積もりを試みる。ここでは、Bestpath を 64 PE 上で実行した時のように、 6 リダクションに 1回の PE 間メッセージが発行される小粒度な並列プログラムの実行を仮定する。これは PE 当たりのメッセージ発行が、

$$27\mathrm{B} \times (890\mathrm{KRPS}/64)/6 = 63\mathrm{KB/s}$$

に相当する。さらに 2 次元メッシュのトポロジの欠点を考慮するため、PE 間通信はランダムに発生するものと仮定する。この場合のメッセージ平均移動距離は PE 数の平方根に比例[2]し、以下のようになる。

$$2 \cdot (L-1) \cdot (L+1)/(3 \cdot L) = 21.3$$

ここで $L = \sqrt{(\mathrm{PE}\text{数}=1{,}024)} = 32$ である。これは 3.3.1の Pentomino や Bestpath による評価に比べ厳しい（トラフィックに関して悲観的な）見積もりとなっている。

[2]PE の 2 次元メッシュが正方形であると仮定する。

メッセージの総頻度はPE数に比例する。また、システム全体のネットワーク・バンド幅を決めるPE間のチャネル数は $4 \cdot L \cdot (L-1) = 3,968$ （ただし、$L = 32$）となるため、ネットワーク・トラフィックは以下のように計算される。

$$\frac{63\ \mathrm{KB} \times 1,024 \times 21.3}{3,968} = 346\ \mathrm{KB/s}$$

これは5 MB/sのバンド幅の6.9%であり、1,000PE程度の大規模システムとしても、通信トラフィックはなお十分に小さいことが分かる。しかしながら、プログラムによりPE間通信の局所性やホットスポットの回避を考慮しなければ、ネットワークは簡単に飽和してしまうことも認識しなくてはならないであろう。

4 おわりに

マルチPSIのPE間通信処理性能の評価を、ベンチマークプログラムなどを用いて、KL1言語処理系を含めたシステムレベルで行なった。

システムの拡張性に重きをおいたハードウェアおよび言語処理系のため、効率の良いGCの実現などのためのPE間通信コストの増大と、それによるオーバヘッドが懸念されていた。今回の評価により幾つかの定量的なデータが得られた。主なものとして、通信コストはリダクションの1～5倍である、OR木探索問題では通信オーバヘッドは十分小さくできる、64PEでのBestpathのように、6リダクションに1回のPE間通信を行なうと台数効果は50%に満たない、ネットワークトラフィックは十分に小さく、今のハードウェアで1,000 PE程度まで接続できる、などである。

しかしながら、今回のマルチPSI、および、その上での並列プログラムに関する評価から得た結論は非常に限られたものである。スケーラブル・マシンの可能性を追求する上で、これからも、異なる実行特性を持つ各種のプログラムを用いて詳細な測定と解析を続ける必要があろう。

謝辞

日頃御指導頂くICOT内田俊一研究部長、齋 和男第一研究室長に感謝します。また、ICOTおよび関連メーカのマルチPSIの処理系やベンチマークプログラムの開発チームの皆様にも感謝致します。

参考文献

[1] T. Chikayama and Y. Kimura. Multiple Reference Management in Flat GHC. In *Proc. of the Fourth International Conference on Logic Programming*, 1987.

[2] T. Chikayama, H. Sato and T. Miyazaki. Overview of the Parallel Inference Machine Operating System (PIMOS). In *Proc. of the International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, 1988.

[3] M. Furuichi, N. Ichiyoshi and K. Taki. A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Programs on the Multi-PSI. In *Proc. of the Second ACM SIGPLAN Symposium on Principles and Practice of Paralle Programming (PPoPP)*, 1990.

[4] N. Ichiyoshi, K. Rokusawa, K. Nakajima and Y. Inamura. A New External Reference Management and Distributed Unification for KL1. In *Proc. of the International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, 1988.

[5] K. Nakajima, Y. Inamura, N. Ichiyoshi, K. Rokusawa and T. Chikayama. Distributed Implementation of KL1 on the Multi-PSI/V2. In *Proc. of the Sixth International Conference on Logic Programming*, 1989.

[6] H. Nakashima and K. Nakajima. Hardware Architecture of the Sequential Inference Machine : PSI-II. In *Proc. of 1987 Symposium on Logic Programming*, Sept. 1987.

[7] K. Taki. The Parallel Software Research and Development Tool: Multi-PSI system. Programming of Future Generation Computers, Elsevier Science Publishers B.V. (North-Holland), 1988.

[8] K. Ueda. Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard. Technical Report TR-208, ICOT, 1986.

[9] P. Watson and I. Watson. An Efficient Garbage Collection Scheme for Parallel Computer Architectures. In *Proc. of Parallel Architectures and Languages Europe*, June 1987.