

並列処理最適化機構の 実用アプリケーションを用いた性能評価

秋葉 智弘 松本 尚 平木 敬

東京大学 理学部 情報科学科

〒113 東京都 文京区 本郷 7-3-1

共有メモリ型並列計算機における、並列アプリケーションの最適化を支援するために、様々なハードウェア機構が提案されている。その中で特に、本論文で取り上げるハードウェア機構は、Elastic Barrierとスヌープキャッシュにおけるプロトコルのデータオブジェクト毎の切替え（スヌープキャッシュ制御機構）である。細粒度から粗粒度まで 広い範囲のプログラムを対象に、命令パイプラインレベルの、execution-driven シミュレーションにより、これらの機能がどのような性能改善をもたらすかの評価を行った。このシミュレーション結果からElastic Barrierとスヌープキャッシュ制御機構は細粒度並列処理に対しては多大な効果があることが確認された。粗粒度並列処理に関しては、両機構は共に大きな性能改善を示すことはなかったが、システムの性能を悪化させることは全くなかった。また、シミュレーション結果より、粗粒度並列処理であってもデータ転送量が多い場合にはスヌープキャッシュ制御機構が性能改善に寄与することが予想される。

Performance Evaluation by Utility Applications of Mechanisms to Optimize Parallel Processing

Tomohiro Akiba Takashi Matsumoto Kei Hiraki

Department of Information Science, Faculty of Science, The University of Tokyo

7-3-1 Hongo, Bunkyo-ku, Tokyo, 113, Japan

Many hardware mechanisms are proposed to support optimizing parallel applications on shared-memory parallel computers. This paper concentrate especially on Elastic barrier (an extended barrier to eliminate idle time), and snoop-cache-protocols' coexistence (each data object can have its own snoop-cache protocol). This paper evaluates by execution-driven instruction-pipeline-level simulation, how much these mechanisms improve the performance of parallel applications that range over fine-grain and coarse-grain programs. The result data of simulations of SPLASH matches the result of execution on another machine. This fact assures that this simulator has high reliability.

1 はじめに

本論文ではバス結合型共有メモリ型並列計算機における二種類の並列処理最適化機構について定量的性能評価を行なう。

並列アプリケーションはプロセス間で頻繁に同期を行う細粒度並列アプリケーションから、データ分散が効率良くなされていれば計算中に同期がまれにしか発生しない粗粒度並列アプリケーションまで様々な性質のものが存在する。また、大規模なソフトウェアでは、細粒度の性質を示す部分と粗粒度の性質を示す部分を合わせ持つことも多い。粗粒度並列アプリケーションにおいては同期の頻度が少ないために、同期のオーバーヘッドによる時間的損失が少ない。このため、プロセッサ台数にほぼ比例した性能が得られる。しかし、細粒度並列アプリケーションにおいては同期が頻繁に発生するために同期による時間的損失が大きく、並列処理による実行時間短縮が難しい。

同期による時間的損失を可能な限り削減することにより性能を改善することが出来る。この同期オーバーヘッドの削減を行なう機構として筆者らはElastic Barrierを提案している [Mat89, Mat91]。

通信に関しても、データ転送路の通信容量には限界があり、容量を増大させるためにはコストがかかる。本論文では、構成が簡易でコストパフォーマンスに優れたバス結合型のマルチプロセッサを評価対象にしているため、バスの転送能力はプロセッサ台数に依らず一定と仮定した。しかし、プロセッサの処理能力が増大するにつれて、計算に必要なデータ転送量は増大する。従って、プロセッサ台数が増加するにつれてバス競合が激化し、最終的にはバスが飽和して処理能力には限界が発生する。

バスの転送容量の限界による性能向上の限界を引き上げるためには、バスの使用頻度を引き下げる必要がある。メモリとプロセッサ間のバストラフィックを削減する方式として、プロセッサにキャッシュメモリを持たせる方式が汎用性があり効果も高い。複数のキャッシュに保持されている同一アドレスのデータの一貫性(コンシステンシ)を保ちながらキャッシングを行うために、スヌープキャッシュが提案された [Goo83, JLL86]。スヌープキャッシュは、一定のプロトコルを使用して、コンシステンシを維持しながらのキャッシュの使用を可能にした。スヌープキャッシュのプロトコルとして、様々なプロトコルが提案されている。しかし、全ての状況で優位に立つプロトコルは存在しない。これに

対して、筆者らはデータ毎にプロトコルを切替える方式(スヌープキャッシュ制御機構)を提案した [Mat89]。

本研究では、Elastic Barrierとスヌープキャッシュ制御機構の定量的効果に関して、粗粒度から細粒度まで広い範囲のアプリケーションを用いてシミュレーションによる評価を行なう。そして、これらの機構が細粒度アプリケーションの性能を向上させることと、粗粒度アプリケーションの性能を改善することを示す。

SPLASHは粗粒度処理を中心とした実用アプリケーションであり、多くの並列システムで評価用のベンチマークとして用いられている [SWG91]。一方、細粒度アプリケーションに関しては、細粒度の並列性を利用可能な並列計算機が少なく、確立したアプリケーションがまだ存在していない。しかし、並列処理が一般的になった時点では、粗粒度の並列処理が不可能な部分を含むアプリケーションも多数存在すると考えられる。細粒度並列処理はこれらのアプリケーションのボトルネック解消に役立つと考えられる。このため、細粒度アプリケーションとしては、現在当研究室で開発を進めている最適化コンパイラOP.1 [IMH93] で用いている例題(カーネルループ)を使用する。これらの実用もしくはアプリケーションの中に含まれるであろうカーネルループのシミュレーションによって、提案している機構の有効性を実証する。

2 シミュレーション方式とモデル

今回想定したマルチプロセッサシステムの構成を図1に示す。アプリケーションコードの特性を反映した厳密なシミュレーションを行うために、シミュレーション方式はトレース駆動や確率モデルによるシミュレーションではなく、実際のアプリケーションコードをクロックレベルで厳密にモデル化されたシミュレータ上で実際に実行させる実行駆動方式を採用した。アプリケーションコードの作成にはコンパイラを用い、並列化や高性能化のためのチューニングは基本的にソースコードレベルで行った。使用したコンパイラシステムは二種類あり、一つは市販のMISP用CコンパイラにPARMACS(SPLASHの節を参照)マクロライブラリを追加したシステムを開発して使用した。もう一つは自動並列化を目指して開発中のOP.1コンパイラである。前者をSPLASHアプリケーションとdoacrossループに後者を級数展開プログラムに対して使用した。シミュレータはMISCシミュレータ

(仮称) [Mat92] に SPLASH の動作のための I/O 入出力機能を付加したものを開発して使用した。なお、SPLASH の実行結果を他の機種での実行結果と比較することや実行時モニタによる動作状況の把握により、シミュレータの信頼性は確認されている。

シミュレーションモデルを簡単に説明する。CPU は、MIPS の R3000 (ただし、R4000 と同等の浮動小数点演算パイプ内蔵) をモデルとするパイプライン型 RISC プロセッサに、fetch-and-increment や test-and-set 等のアトミックメモリ操作命令と preq, rreq, aprv 等の Elastic Barrier 命令を追加している。Elastic Barrier の同期操作は専用インストラクションによる実装を仮定しているため、レジスタ演算命令と同じコストが CPU 内でもかかり、電気的伝搬による CPU 間の同期情報の遅延もクロックレベルで厳密に評価している。メモリ管理は、セグメントによって行ない、セグメント毎にキャッシュプロトコルの切替が可能である。今回のシミュレーションでは update プロトコル、invalidate プロトコル、allread_up (読み出し時は allread 動作書き込時は update 動作) プロトコルの三種類のプロトコルのみを使用した。

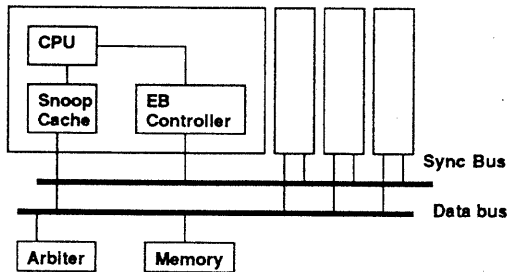


図 1: 評価対象システムの構成図

シミュレータの設定可能なパラメータのうち、シミュレーションにおいて固定したパラメータの設定内容を以下に列挙する。

- 共有バスは 1 本、バス幅が 64bit、通信はキャッシュブロック単位 (ブロックサイズ 32byte) で、で行われる。1 ブロックの転送に 4 回のバス上のデータ転送が必要で、バースト転送により 1 アクセス (4 転送) が共有バス上において 6 クロック (3+1+1+1) のコストで実行されると設定した。
- キャッシュの構成はプロセッサ内蔵の命令キャッシュが 4KB、スヌープキャッシュが 16KB、共

に 2 ウェイ。データの load および store はバスアクセスを伴わない場合が 2 クロックで、バスアクセスを伴う場合は 2 クロックに加えてバスを獲得してから 7 クロックのレイテンシがある。

- プロセッサの命令コストはレジスタ演算命令が 1 クロック、ただし、乗除算は 4 クロック、分岐命令が 2 クロック、load および store のメモリアクセスレイテンシの影響に関しては、スコアボーディングと 1 段のバストランザクションバッファにより (R3000/R4000 には存在しない) 依存関係と資源が許す限り load/store に後続する命令はブロックされない。
- Elastic Barrier のプロセッサ間の同期情報伝搬遅延はプロセッサ間 3 クロック (CPU-controller-controller-CPU) と設定されている。

3 細粒度アプリケーション

先述のように、配列を使った典型的な doacross ループ (隣合うあるいは定数回離れたループイテレーション間に依存関係が存在するプログラム) と級数展開プログラム (doacross ループの一種) を細粒度アプリケーションの例として取り上げる。これらの doacross 型のプログラムは、多数のイテレーションが鎖状に依存関係で結合されるために、並列化による高性能化が難しいタイプのプログラムである。

静的にループのイテレーションをスケジューリングしても、バスの競合やキャッシュミスと言った静的に完全に予測できない要因によって実行時のプロセッサごとのプログラムの進行状況に乱れが生じる可能性がある。この乱れが生じていても依存関係が保証されプログラムが正しく実行されるために、同期が必要とされる。doacross 型のプログラムでは、依存関係の発生頻度が多く、それに応じて多数の同期が必要とされる。このため同期のオーバーヘッドが蓄積して性能に悪影響を及ぼし易い。

3.1 使用するバリアの変更

Elastic Barrier を含む三種類のバリア方式を使って性能比較を行った。この評価に用いた Elastic Barrier 版、naive な barrier 版、Soft によるバリア版、の 3 つのプログラムは、以下の様にして得た。まず、細粒度的チューニングによる最適化、もしくはサイクリックなスケジューリングによる最適化により、Elastic Barrier 版プログラムを生成する。さらに、

```

float v1,v2,v3,v4,v5,v6,v7,v8;
integer v9,v10;
for(v9=1;v9<1000;v9++){
  v10 = v9 - (v9/100)*100;
  v2 = 1.0/v6;
  v2 = v2-1.0/(v6+2.0);
  v2 = v2+1.0/(v6+4.0);
  v2 = v2-1.0/(v6+6.0);
  v3 = v3+v7/(v6+8.0);
  v3 = v3-v7/(v6+10.0);
  v3 = v3+v7/(v6+12.0);
  v3 = v3-v7/(v6+14.0);
  v4 = v7/(v6+16.0);
  v5 = v5+v7/(v6+18.0);
  v4 = v4+v7/(v6+20.0);
  v5 = v5+v7/(v6+22.0);
  v4 = v4+v7/(v6+24.0);
  v5 = v5+v7/(v6+26.0);
  v4 = v4+v7/(v6+28.0);
  v5 = v5+v7/(v6+30.0);
  v1 = v1+v8/(v6+32.0);
  v1 = v1-v8/(v6+34.0);
  v1 = v1+v8/(v6+36.0);
  v1 = v1-v8/(v6+38.0);
  v2 = v2+v4;
  v1 = v1+v2*4.0;
  v6 = v6+40.0;
  v1 = (v1+4.0*v3) - 4.0*v5;}

```

図 2: example: 級数展開

naive なバリア版プログラムを PREQ の除去、及び APRV の RREQ への変換により生成する。さらにその RREQ をメモリへの atomic operation によりバリア機能を実現するルーチンで置き換えることで、ソフトによるバリア版を生成する。

3.2 細粒度的チューニング

細粒度最適化によるプログラム実行の例として、図2のループボディを持つプログラムを使用する。このプログラムを級数展開 (progression) と呼ぶことにする。このプログラムを細粒度並列化コンパイラ OP.1 [IMH93] によってコンパイルし、実行した結果を図3および図4に示す。Elastic Barrier を使用した場合、2プロセッサの時1.8倍、4プロセッサの時2.1倍、の性能向上を示す。ナイーブなバリアでは、Elastic Barrier と比較して約1割性能が低い。さらに、共有変数によるバリアでは、実行時間の約9割がバリアアイドルタイムとなり、1PE

の時よりもマルチプロセッサの時の方が多くの実行時間がかかってしまっている。なお、キャッシュプロトコルは update 型を使用している。

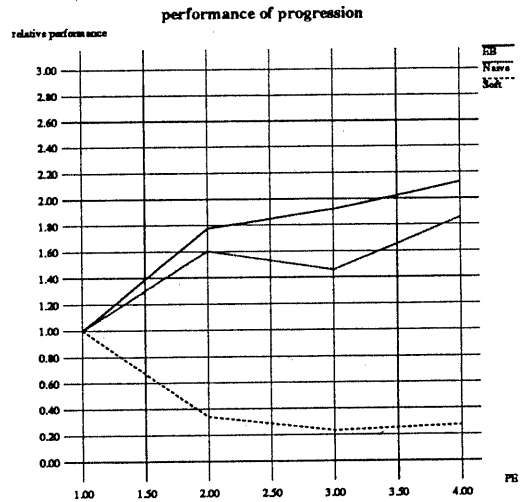


図 3: 級数展開と実行効率

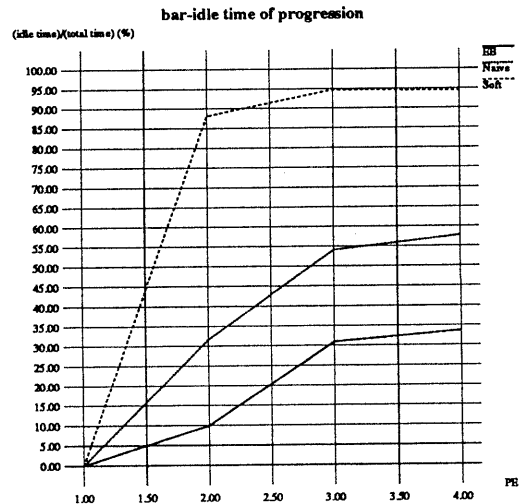


図 4: 級数展開とバリアアイドルタイム

3.3 パイプライン型並列性によるチューニング

配列演算で典型的な doacross 型の依存を持っているプログラムの例として、図5のループ部分を持つプログラム [Mat92] を使用する。このプログラムは短時間に多数の同期を必要とし、かつ同期による時間的損失が多い。

PARMACS を使用して、サイクリックなスケジューリングを行ない、このプログラムの最適化を行なっ

```

for(i=3;i<n-2;i++){
  c[i] = x[i] + w1*x[i+2]
        + w2*x[i+1] + w3*c[i-3] +w4*c[i-2];}

```

図 5: doacross example

```

for(i=3;i<n-2;i++){
  αi; t = x[i] + w1*x[i+2] + w2*x[i+2];
  βi; t = t + w3*c[i-3];
  γi; t = t + w4*c[i-2];
}

```

図 6: 3段に分けた doacross example

た。ループイテレーション間の依存は、配列 c を介して存在している (図 5)。また、ループボディの計算には、他のイテレーションに依存しない部分、2つ前のイテレーションに依存する部分、3つ前のイテレーションに依存する部分、の3つがある (図 6)。

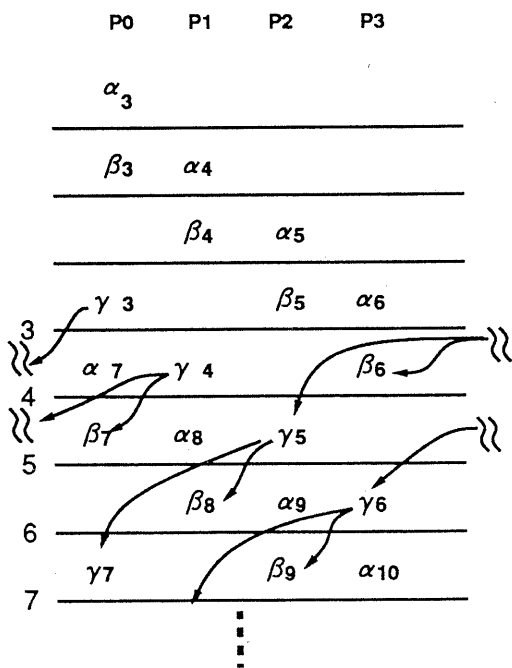


図 7: 4PE の場合の doacross のパイプライン型並列性

これらの3つのフェーズをオーバーラップさせることにより、並列性を引き出した (図 7)。その並列性と Elastic-Barrier の命令との対応を図 8 に示す。な

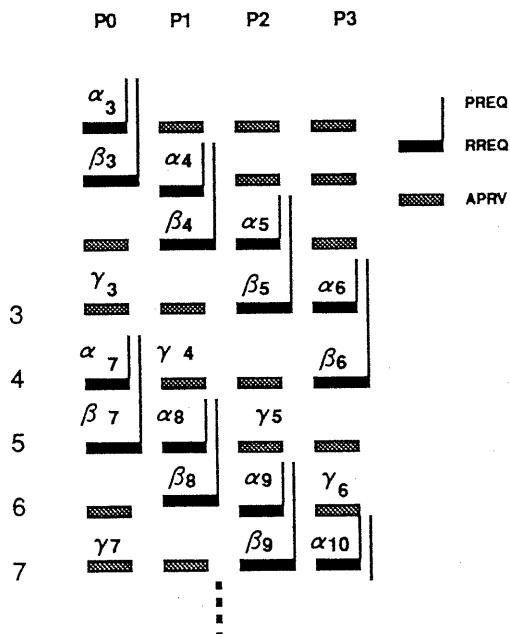


図 8: 4PE の場合の doacross の Elastic Barrier

お、こちらのプログラムを doacross と書くことにする。

そしてその実行した結果を図 9、図 10、に示す。なお、これらの図に対応するシミュレーションではスヌープキャッシュプロトコルを update に固定している。

4 プロセッサの時に、ナイーブなバリアで、バリアによるアイドルタイムが 41% を占めているのに対し、Elastic Barrier では 19.3% を占めている。1 プロセッサの時と比較すると、2.38 倍の速さで、厳しい条件下で性能改善を達成している。

同期によるアイドルタイムが 19.3% に抑えられた。しかしながら、 $4 \times (100 - 19.3) / 100 = 3.22$ 倍に達していない。

さらに性能を引き出すために、プロトコルの切替を行なった。プロセッサローカルなスタック領域に関しては常にプロセッサ間共有が起らないプロトコルを使用し、PARMACS の gmalloc で獲得される共有メモリ領域 (つまり配列 x と配列 c) に関してのみプロトコルを invalidate, update, allread_up に変更してシミュレーションを行った (図 11)。すると、最も速い allread_up が 最も遅い invalidate より 37% 程速いことが分かった。

参考までに表 1 に update プロトコル使用時のバス使用率を示す。

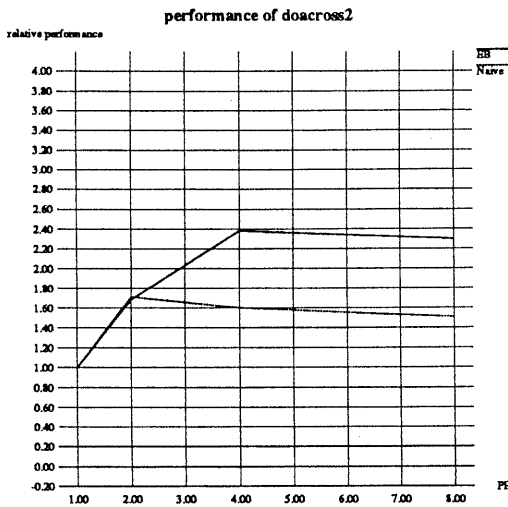


図 9: doacross と実行効率

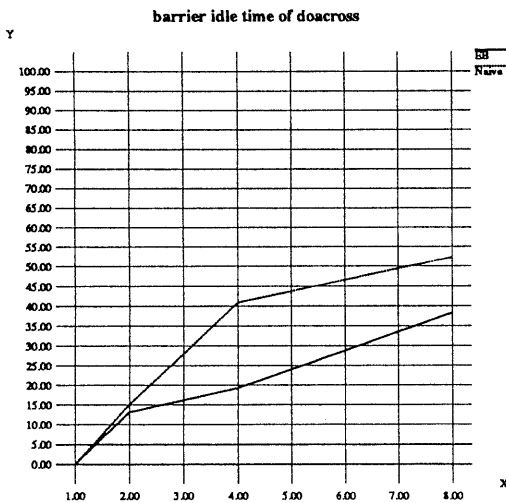


図 10: doacross とバリアアイドルタイム

| | IFETCH | DREAD | DWRITE | TOTAL |
|---|---------|---------|---------|---------|
| 1 | 1.3739 | 1.8416 | 0.3069 | 3.8708 |
| 2 | 2.4810 | 2.6471 | 10.3598 | 16.2069 |
| 4 | 5.9791 | 9.2613 | 15.4077 | 32.9617 |
| 8 | 10.5583 | 16.4528 | 17.1637 | 47.1827 |

表 1: doacross の共有バスの使用量 (%)

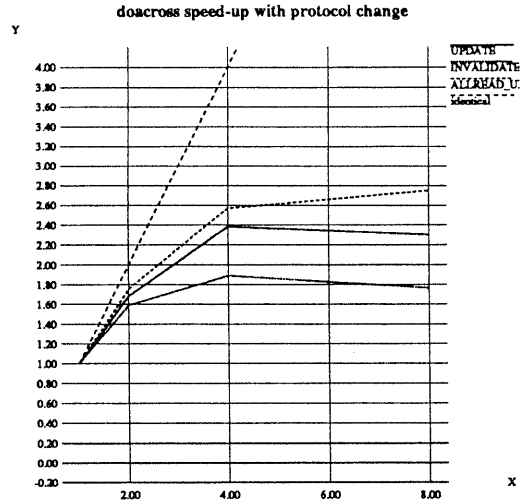


図 11: doacross とプロトコル

3.4 細粒度のまとめ

1つ目には同期が錯綜し、同期によるアイドルタイムがしばしば大きくなることである。2つ目には、ローカルキャッシュに乗りにくいために、データアクセスのコストが高くなることである。特にバストラフィックの増大により、バスが飽和した状態では性能を厳しく制限することになる。

これらの問題は、計算の構造自体に由来するものである。従って根本的に解決することは出来ない。しかしながら、Elastic Barrier やスヌープキャッシュ制御機構によってアイドルタイムを軽減することは出来る。また、このアイドルタイムの削減に伴って、現行の並列計算機で比較的効率良く動作するアプリケーションも、以前よりより高速に実行することができる。

4 粗粒度プログラム

4.1 SPLASH の概要

SPLASH(Stanford Parallel Applications for SHared-memory) は従来使われていたものよりも完全な形の、共有メモリ型のマルチプロセッシングシステムの設計及び評価の中で利用するための、並列アプリケーションを提供しようとするものである。これらのアプリケーションは、共有メモリマルチプロセッシングの分野で働くアーキテクトとプログラマの助けになるように意図して、様々な科学技術計算の分野から選ばれた。

これらのアプリケーションのコンパイルについては、原則としてソースコードに手を加えないで、実行が出来ることを目標に、普通の C 言語と PAR-

| Application | Synchronization | Granularity |
|-------------|----------------------|-------------|
| Ocean | barrier | large |
| Water | barrier | large |
| MP3D | barrier | large |
| LocusRoute | task queues, barrier | medium |
| PTHOR | task queues, barrier | medium |
| Cholesky | task queue | large |

表 2: Summary of some behavioral characteristics.

MACS [Boy87] によって書かれたソースコードを、シミュレーター用のオブジェクトに変換するシステムを開発した。これは、RISC News の C コンパイラ、ディスアセンブラを利用し、それをフィルタで調整して、シミュレータ用のオブジェクトを生成する。

4.2 SPLASH の実行結果

Elastic Barrier と、キャッシュプロトコルのデータオブジェクト毎の切替えを評価するために、以下の 3 つのバージョンを用意した。

Original: オリジナル

CPC: キャッシュのプロトコルを切替える。

EB: Elastic Barrier を利用して、バリア同期に幅を持たせる。

そして、実行した結果が、表 4 である。また、初期のデータ入力部については、実行時間から省いてある。

| Application | Original % | CPC % | EB % |
|-------------|------------|-------|-------|
| Water | 100 | 100.6 | 100.3 |
| MP3D | 100 | 106.8 | 100.2 |

表 4: 実行効率 (オリジナルを 100% とした)

| Application | Original | CPC | 削減率 |
|-------------|----------|----------|-------|
| Water | 2027548 | 1752309 | 13.6% |
| MP3D | 15664814 | 14886029 | 5.0% |

表 5: バス使用時間

Water, MP3D の様に、基本的にバリアを使い、適当にロックを挿入した、粒度の粗いアプリケーションにおいては、もともとバリアの同期におけるタイムロスが、実行時間に比べて非常に小さいので、Elastic Barrier による性能改善があまり認められなかった。これを裏付ける表が表 6 である。

また、WATER, MP3D においては、共有メモリの特定の位置に対して、スタティックに担当する

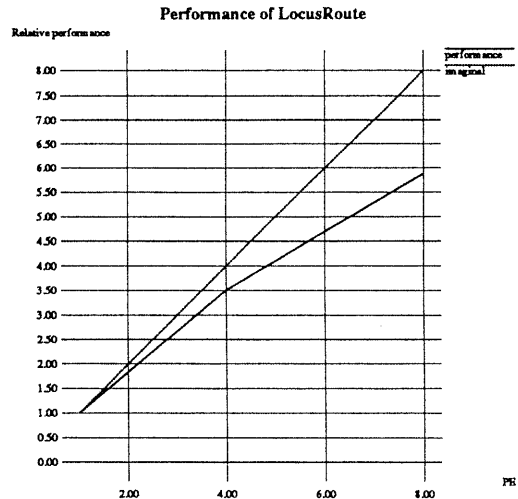


図 12: LocusRoute と実行効率

| EB: | Water | MP3D | locus |
|-----------|-------|-------|-------|
| lock idle | 0.00% | 0.00% | — |
| bar idle | 3.92% | 0.75% | — |
| Naive: | Water | MP3D | locus |
| lock idle | 0.00% | 0.00% | — |
| bar idle | 3.95% | 0.87% | 8.1% |

表 6: アイドルタイムの占める割合

プロセッサが決まってしまうような部分の大きいので、あるメモリオブジェクトに対しては、あるプロセッサが高い確率でアクセスするのに対し、他のプロセッサは非常に低い確率でしかアクセスしない。そのため、update 用の情報が共有バスへわざわざ出ていかなないように、キャッシュプロトコルを update から invalidate へと変えて性能を検証した。図 4 は、その性能の表である。スヌープキャッシュプロトコルの切替えによって、性能が微増している。表 3 は、そのときのバス上のメモリアクセスに要したクロックの表である。表 3 より、メモリアクセスのロードが高くなく、実行に大きな影響を与える状況ではない。そのため、性能の大幅な向上が見られなかったものと思われる。

4.3 粗粒度のまとめ

SPLASH は、粗粒度のアプリケーションに偏っているために、共有変数によるバリア同期で十分小さいアイドルタイムが達成できる。また、表 3 の様に、共有バスの使用が、比較的少ないものが多い。すなわち、Elastic Barrier や、キャッシュプロトコル切替えが実行時間を改善できる余地が、もともと

| | IFETCH | DREAD | L_DREAD | DWRITE | L_DWRITE | TOTAL |
|-------|--------|-------|---------|--------|----------|-------|
| Water | 2.0 | 0.7 | 0.6 | 1.2 | 0.2 | 4.7 |
| MP3D | 1.2 | 7.8 | 9.9 | 18.0 | 4.8 | 41.7 |
| pthor | 0.9 | 4.7 | 15.5 | 0.9 | 7.9 | 29.9 |
| Locus | 3.4 | 4.3 | 12.9 | 6.8 | 3.6 | 31.0 |

表 3: バス上のメモリアクセスタイム (%)

| WATER | FETCH | READ | WRITE | TOTAL |
|--------|-------|------|-------|-------|
| Normal | 2.21 | 2.01 | 2.13 | 2.04 |
| CPC | 2.21 | 2.09 | 2.05 | 2.04 |

| MP3D | FETCH | READ | WRITE | TOTAL |
|--------|-------|------|-------|-------|
| Normal | 3.69 | 2.18 | 4.07 | 2.65 |
| CPC | 4.04 | 2.17 | 2.31 | 2.42 |

表 7: WATER, MP3D における 1 アクセスあたりの必要なクロック数

あまりなかったアプリケーションに偏っていると言える。しかしそうした中で、共有バスの使用率がかなり低いので、実行時間の差として現れることがなかったのだが、キャッシュのプロトコル切替えが共有バスの使用量を減らすことが出来たのは、潜在的に性能を向上させる可能性があると言える。

5 まとめ

共有メモリ共有バス型マルチプロセッサに関して、Elastic Barrier とスヌープキャッシュにおけるプロトコルのデータオブジェクト毎の切替え（スヌープキャッシュ制御機構）のアプリケーションに基づいた定量評価を行った。

細粒度から粗粒度まで 広い範囲のプログラムを対象に、命令パイプラインレベルの execution-driven シミュレーションにより、機構の精密な評価を行った。また、現在のコンパイラによる最適化および並列化の技術水準をも反映するために、アプリケーションコードの生成は並列処理用マクロを持つコンパイラシステムか自動並列化コンパイラを使用し、最適化はソースコードレベルのチューニングのみで行った。アプリケーションプログラムとしては、粗粒度の性質を持つ SPLASH と細粒度の性質を持つ doacross ループと級数展開プログラムを使用した。

シミュレーションの結果、細粒度プログラムに対して、Elastic Barrier やスヌープキャッシュ制御機構の性能向上効果が大きいことが示された。粗粒度プログラムに対しては、大きな性能向上は得られなかったが、バスの使用回数削減にスヌープキャッシュ制御機構の効果が現れていることがわかった。このため、粗粒度のプログラムであっても、バス使用頻度の高いものに対しては性能向上が見込まれ

る。

参考文献

- [Boy87] James Boyle. *Portable programs for parallel processors*. Holt, Rinehart and Winston, Inc., 1987.
- [Goo83] J.R. Goodman. Using cache memory to reduce processor-memory traffic. *Proc. 11th Int. Symp. on Computer Architectuer*, pp. 124-131, June 1983.
- [IMH93] 稲垣達氏, 松本尚, 平木敬. 細粒度並列計算機用最適化コンパイラ:op.1. 情報処理学会研究報告, Vol. 93, No. 73, pp. 1-7, August 1993.
- [JL86] J.Archibald and J.-L.Baer. Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Trans. Computer Systems*, 1986.
- [Mat89] 松本尚. 細粒度並列実行支援機構. 情報処理学会計算機アーキテクチャ研究会報告, Vol. 32, No. 77-12, pp. 91-98, July 1989.
- [Mat91] 松本尚. Elastic barrier: 一般化されたバリア型同期機構. 情報処理学会論文誌, Vol. 32, No. 7, pp. 886-896, July 1991.
- [Mat92] 松本尚. スヌープキャッシュ制御機構の doacross ループへの適用. 並列処理シンポジウム JSPP '92 論文集, pp. 297-304, June 1992.
- [SWG91] Jaswinder P. Singh, Wolf-Dietrich Weber, and Anoop Gupta. Splash: Stanford parallel applications for shared-memory. Technical report, Computer Systems Laboratory, Stanford University, CA 94305, 1991.