並列計算機 Cenju-4 上でのリモート DMA 転送とハードウェア DSM の性能比較

Performance Comparison of Remote DMA Transffer with Hardware DSM on a Parallel Computer Cenju-4

前川 勉

平成 12 年 2 月 15 日

内容梗概

大規模かつ高速な並列計算機システムを構成するためには、主記憶を物理的に分散させた上で、効率の良いプロセッサ間通信を実現する必要がある。分散メモリ型並列計算機におけるプロセッサ間通信は、大きく、ハードウェア DSM とメッセージ通信に分けることができる。前者は、プログラムがノード間のデータ転送を全く意識することなく、ハードウェア機構により、ロード/ストア命令を分散メモリ全体に拡張する方式であり、後者は、プログラムが明示的にデータ転送を指示する方式である。

メッセージ通信はさらに、システムバッファを用いた通信と、リモート DMA 転送に分類することができる。システムバッファを用いた通信では、受信側においてもソフトウェアによる受信操作を行う必要があり、かつ、オペレーティングシステムが管理する受信バッファからプログラムが使用するデータ領域へのデータ転送を伴うため、リモート DMA 転送よりも転送コストが高い。このため AP1000+および Cenju-4 では、通信専用 LSI を各ノードに装備し、効率の良いリモート DMA 転送を実現している。

本研究では、現在主流となっている、ハードウェア DSM とリモート DMA 転送の二つに着目し、各方式の性能を比較した。測定は、ハードウェア DSM とリモート DMA 転送の両方を装備した並列計算機 Cenju-4 において、SPLASH-2 の FFT、LU 分解および Radix Sort を走行させせることにより行った。

ハードウェア DSM はプログラマに単純な並列プログラミング環境を提供し、かつ、細粒度通信が要求される問題に対しては高い性能を示す。しかし、ノード間で巨大なデータを一度に転送する場合には、キャッシュライン単位の細粒度のデータ転送が繰り返されるため、個々の転送のオーバーヘッドは小さくとも、オーバーヘッドが累積し、結果的に大きなオーバーヘッドとなることが予想される。

一方、リモート DMA 転送では、一回の通信で指示できるデータ長が可変であるため、ノード間で転送されるデータを、低コストで、ある程度大きなサイズにまとめることが可能である場合、通信オーバーヘッドを低減することができると予想される。また、通信専用ハードウェアが通信処理を行い、 プロセッサは通信コマンドを発行すると直ちに後続する計算を実行できることから、計算と通信のオーバーラップによる高速化の効果が大きいと予想することができる。

SPLASH-2 に含まれるベンチマークプログラムを実際にハードウェア DSM とリモート DMA 転送それぞれの方式に基づいて実装し、走行したところ、以下の結果を得ることができた。FFT および LU 分解においては、データの転送パターンに見られる規則性を利用して転送データのサイズをより大きくすることができ、効果的なブロックデータ転送を実現することができた。さらに、計算と通信をオーバーラップさせた結果、リモート DMA 転送は、ハードウェア DSM に対して、32 ノードの場合、FFT で平均 19.8 %、LU 分解で平均 14.9 %、の高速化を達成

した。また、LU分解において用いたリモートDMA転送では、同一ノードに対するデータ転送の順序が保証されることを利用して、転送データ毎に同期フラグを用意し個別に同期を取ることにより、ハードウェアDSMが提供するバリア同期よりも、効率の良い同期操作を実現することがでた。一方、Radix Sortでは、ブロックデータ転送を実現するためのコストが大きく、逆に、リモートDMA転送による実装はハードウェアDSMに比べ平均で約1.37倍の実行時間を要する結果となった。

しかしながら、転送データが受信ノードの主記憶の連続領域に格納されるようにすることができれば、受信ノードにおける、データ配置操作に伴うオーバーヘッドを低減でき、リモートDSM転送を用いた場合でも高速化することができると考えており、今後の課題である。

Abstract

In order to compose a multiprocessor which has high scalability and high performance, it is necessary to distribute memory physically and to implement an efficient interprocessor communication. The two major communication paradigms in multiprocessors with a distributed shared memory are hardware DSM and messages passing. In the former, programs don't have to be conscious of communication and an implicit communication has done through loads and stores in a shared memory. In the latter, programs indicate explicit communication.

Messages passing can be classified communication using system buffer and remote DMA transfer. Communication using system buffer costs much,because it requests receiver an explicit receive operation and copying data from system buffer to memory area which is used by program. Therefore machines like AP1000+ and Cenju-4 provide a special LSI in each node that is responsible for handling all communication to and from remote nodes,and implement an efficient remote DMA transfer.

In this research, I pay attention to hardware DSM and remote DMA transfer, which are predominant data transfer today. And I make a comparative study of hardware DSM with remote DMA transfer. To carry out performance comparison I implement three computational kernels of the SPLASH-2 suite, FFT,LU and Radix, on a parallel computer Cenju-4 which supports both hardware DSM and remote DMA transfer.

Hardware DSM presents programmers with a favorable programming environment, and have been shown to perform well on tasks that require fine-grain communication. But when a large block of data is to be communicated from one processing node to another, it must be done through a series of fine-grain transfers in cache-line size. Although the overhead required to initiate each fine-grain transfer is small in hardware DSM, the number of transfers required for large communications will bring the cumulative overhead to levels that have a substantial impact on overall performance.

In remote DMA transfer the size of transfered data par communication is changeable. Therefore it is possible to reduce the overhead of communication, if transfered data can be gathered into one block data with small cost. And there is a good possibility that overlapping computation with communication develops performance, because with the special hardware which responsible for communication, the main processor is free to perform concurrent computation.

I actually implement SPLASH-2 benchmarks using both hardware DSM and remote DMA transfer. And I have run them in practice. Consequently I have got the following results. In FFT and LU, I can gather transfered data into one large block using the regularity which is shown in patterns of data transfer, so I implement efficient block data transfer. And overlapping computation with communication, the execution time of the remote DMA transfer version is reduced, on the average,by 19.8 % in FFT and by 14.9 % in LU as compared with the execution time of the hardware DSM version.

The remote DMA transfer assure of sequential data transfer to the same node. In LU, therefore, I can implement an efficient synchronization by using flags to synchronize processors in the remote DMA transfer version instead of barriers that are used in the hardware DSM version.

In Radix, on the contrary, the execution time of the remote DMA transfer version is about 1.37 times as long as that of the hardware DSM version on the average.

If it is possible to store transfered data in memory of a receiving node serially, the overhead of allocating data in receiving node is expected to be reduced. And remote DMA transfer should increase performance.

目 次

1 はじめに

マルチプロセッサシステムにおいては、プロセッサ間通信に伴うコストがシステムの性能を大きく左右するため、より低いレイテンシ及び、より高いスループットが要求される。効率の良いプロセッサ間通信を実現するためのデータ転送方式は以下の二方式に大別される。一つはハードウェアを工夫することにより、プロセッサの load/store 命令を遠隔メモリにまで拡張し、起動コストの小さいメモリアクセス単位の細粒度通信を多頻度で実現するもの (以下、ハードウェア DSM と呼ぶ)、もう一つはメモリ空間を共有することなく、ソフトウェアによるメッセージ通信により、起動コストは大きいものの高いスループットを実現するものである。

従来、前者は細粒度の通信パターンを有する並列アプリケーションに対して、また後者は粗粒度の通信パターンを有する分散アプリケーションに対して有効であるとされ、住み分けがなされてきた。

ところが近年、多くの並列計算機において専用ハードウェアが開発され [2, 5]、OS の介在を必要としないユーザーレベルのメッセージ通信を実現するとともに、リモートノードのメモリを直接操作するリモート DMA(Direct Memory Access) 転送機構も装備されるようになった。これらのシステムにおいては、通信のための処理からプロセッサを解放することにより、計算と通信のオーバーラップが可能であり、さらなる性能向上が期待できるため、メッセージ通信によるデータ転送方式は、その適用範囲を拡大しつつあると言える。

また、ソフトウェア分散共有メモリと最適化コンパイラ技術の進歩により、SPLASH (Stanford ParalleL Applications for SHared memory) 等に現れる細粒度の通信パターンをより大きい粒度の通信パターンに変換できる可能性が報告されるなど [13, 14]、細粒度通

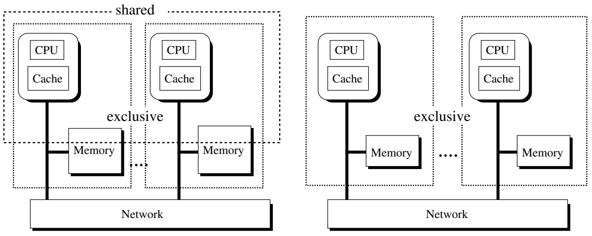
信の必要性は縮小する方向にある。実際、SPLASH-2 [12] の数値計算アプリケーションの中には、規則的で反復的な通信パターンを有するものがいくつかある。このような問題に対しては通信パターンの粒度をある程度粗くし、ブロックデータ転送を適用することにより通信の起動コストと通信量を削減し、さらに計算と通信をオーバーラップさせることにより性能向上を図ることができる。

本研究では、SPLASH-2 を用いて、ハードウェア DSM とリモート DMA 転送の性能比較を行う。以下、2 章において本研究の背景としてマルチプロセッサにおけるプロセッサ間通信の概要について述べる。次に、3 章において性能比較の手段として用いた並列計算機 Cenju-4 の概要について説明する。続いて 4 章において、Cenju-4 上での両データ転送を用いた SPLASH-2 の実装について示し、5 章で測定結果と考察を述べ、6 章でまとめる。

2 プロセッサ間の通信方式

拡張性のある並列計算機を構成するためには、主記憶を物理的に分散配置する必要がある。本章ではマルチプロセッサの構成方式を分散共有メモリマルチプロセッサと分散非共有メモリマルチプロセッサの二つに大別し、両システムにおけるプロセッサ間通信の特徴について述べる。

2.1 マルチプロセッサのメモリアーキテクチャ



分散共有メモリ　　　　　　　　分散非共有メモリ

図 1: マルチプロセッサのメモリアーキテクチャ

分散共有メモリ　分散共有メモリマルチプロセッサでは、各プロセッサは、固有のローカルアドレス空間と共有アドレス空間を持ち、ローカルメモリ及び共有メモリに対する参照を高速化するために、一般にキャッシュが設けられる。さらに、主記憶の一部を利用した大容量遠隔キャッシュを有するシステムも存在する。

分散非共有メモリ　分散非共有メモリマルチプロセッサにおいては各プロセッサはローカルアドレス空間のみを有する。他のプロセッサは主記憶に対する通常のロード・ストア命令による参照は不可能である。

2.2 ハードウェア DSM の概要

ハードウェア DSM では、データ転送は共有メモリに対するロード及びストア操作により行われ、キャッシュのコヒーレンス制御もキャッシュラインを単位としてハードウェアによって高速に行われる。このため、一回の通信当りの起動および管理コストは小さい。また共有空間に対するメモリアクセスに応じて必要な処理をハードウェアが自動で行うために、ソフトウェアからはプロセッサ間通信を意識する必要がないため、プログラマに対して単純な並列プログラミング環境を提供することができる。このためデータの転送パターンが動的に変化するような細粒度通信を行うプログラムに適している。

2.3 ハードウェア DSM の特徴

ハードウェア DSM においては、キャッシュシステムを利用してノード間のデータ転送に伴なうレイテンシの隠蔽を図っている。即ち、キャッシュシステムこそハードウェア DSM を特徴づけるものであると言える。以下にハードウェア DSM の特徴としてハードウェア DSM におけるキャッシュシステムの特徴を示す。

- キャッシュディレクトリの分散配置

  ハードウェア DSM ではシステムの拡張性を増すために、コヒーレンス制御に必要となるキャッシュディレクトリも各ノードに分散配置されている。

- コヒーレンスプロトコル

  コヒーレンス制御の手法としては主に、write-invalidate 方式と write-update 方式がある。write-invalidate では、共有メモリに書き込みが

  あった時に当該領域のキャッシュコピーを全て無効化する。一方、write-update では、当該領域のキャッシュコピーを全て更新する。通常、各並列計算機おいてコヒーレンス制御方式は固定されている。

- データ転送とコヒーレンス制御の単位

  ハードウェア DSM においては、データ転送とコヒーレンス制御の単位はキャッシュラインと呼ばれる固定サイズで行われ、高速に処理が行われる。しかし、キャッシュラインは通常 32KByte ~ 128KByte と比較的小さいため、スループットに限界があるほか、本来は必要のないキャッシュのコヒーレンス制御に伴う無駄なデータ転送が生じる場合がある。

2.4 リモート DMA 転送の概要

リモート DMA 転送では、遠隔メモリ操作要求コマンドをソースコードに記述することにより、送受信プロセッサ間でお互いのアドレス空間に直接アクセスすることができる。ただし、同期操作を受信側で明示的に行う必要がある。送信プロセッサが受信プロセッサのメモリ上にデータを書き込むリモートメモリライトや、受信プロセッサが送信プロセッサのメモリ上からデータを読み出すリモートメモリリードなどがある。

近年多くの並列計算機において、従来 OS やユーザープログラムが行っていたメッセージ処理の一部を専用ハードウェアが高速に処理するユーザーレベルのリモート DMA 転送に関する研究が盛んに行われており、AP1000+や Cenju-4 などの並列計算機に実装されている。

2.5 リモート DMA 転送の詳細

図 2は典型的な分散メモリ型並列計算機の構成とリモート DMA 転送によるデータ転送の様子を示したものである。各ノードは PE(Processing Element)、Node Controller(以下 NC)、 Memory 及び Network Interface から構成される。

リモート DMA 転送では図 2の Node A の PE で Node B のメモリに対してリモート書き込み要求が生じた場合、以下に示す過程を経てデータ転送が実行される。

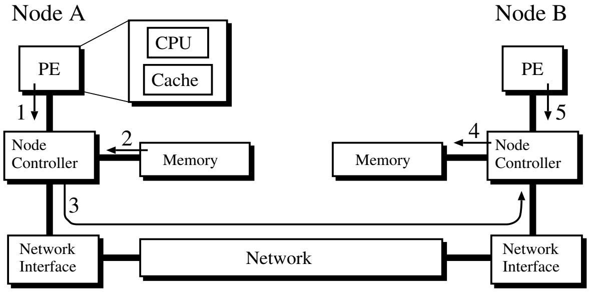


図 2: リモート DMA 転送の流れ

1. PE は NC に対して転送データの自ノードメモリ上でのアドレス、データサイズ、Node B のノード ID 及び Node B のメモリ上での格納アドレス等を通知する (Step1: 図 2の矢印 1)。
2. Node A の NC はデータ転送に対する操作を PE から引き受け、PE がデータ転送に関連する初期化を完了するとローカルメモリから 転送データを取り出す (step2: 図 2の矢印 2)。
3. NC は転送データに PE により通知された情報をヘッダとして転送データに付加してネットワークへの送信を開始する。このときネットワークのハードウェア的な制約から、通常、転送データは固定長の大きさに分割され、それぞれにヘッダを付加したパケットと呼ばれる単位で転送される (Step3: 図 2の矢印 3)。
4. Node B にデータが到着すると Node B の NC は転送データ (パケット) からヘッダを取り除きヘッダの情報をもとに Node B のローカルメモリの指定されたアドレスに転送データを格納する (Step4: 図 2の矢印 4)。
5. 最後に Node B の PE はフラグをチェックし、データ転送の完了確認を行なう (Step5: 図 2の矢印 5)。

2.6 リモート DMA 転送の特徴

以下にリモート DMA 転送の特徴を示す。各項目は性能向上に対して最も効果的であると予想されるものから列挙してある。

- ブロックデータ転送による起動コストの低減

ハードウェア DSM の起動時オーバヘッドがキャッシュライン単位に行われるデータ転送毎に生じるのに対し、リモート DMA 転送の場合は転送対象となるメモリオブジェクト毎で済む。このためラインサイズより大きいデータを転送する場合、リモート DMA 転送の起動時のオーバーヘッドは転送データ総量に対して相対的に小さくなる。

- データ転送のパイプライン化

  リモート DMA 転送では Step2 及び Step3 はパイプライン化して実行される。即ち、NC はパケットをネットワークに送出すると同時に次のデータをメモリから取り出す。このパイプライン化により更にオーバーヘッドを隠蔽することができる。

- 演算と通信のオーバーラップ

  前述の通り、リモート DMA 転送では、Step1 においてプロセッサが NC に対して通信の起動通知を行うと、データ送信のための一連の操作 (Step2,3) は NC が担当する。プロセッサは送信作業から解放され、次命令の実行が可能となる。即ち、通信と演算をオーバーラップさせることが可能であり、通信レイテンシを隠蔽することができる。

- 生産者による効果的な通信指示

  一般にデータの生産者によるデータ転送はブロッキングではない。つまりデータの生産者はデータの送信が完了していなくとも後続する処理を続けられる。これに対し、データの消費者はデータが必要になった時にはそのデータを得るまで後続の処理を続けられない場合が多い。よって、データの生産者がそのデータの消費者に対してデータを転送した方が性能向上につながる可能性が高いと考えられる。

  送信ノードでのデータ転送の指示を受信側での転送データ依存命令の発行に対して十分先行させることで、受信ノードから見た通信のレイテンシを完全に隠蔽することが可能である。

- 通信量の削減

  ハードウェア DSM による通信の場合、遠隔ノードであるデータに対してリードミスが発生すると、まずリクエストメッセージがネットワークを通じてホームノードに転送され、次に、ホームから転送データを含むリプライがリクエストメッセージを発信したノードに返される。即ち、二つのメッセージがネットワーク上を通過する。更に、キャッシュコヒーレンス制御が加わると多くのメッセージがネットワークを飛び交うことになる。ホームノード以外の PE がキャッシュ上のデータを更新するまえに、そのデータを含むキャッシュラインが置き替え対象となった場合、ホームへの書き戻しメッセージがホームに対して送信される。また複数のノードがキャッシュ上にデータを共有している場合、ある PE がこれを更新したとき無効化 / 更新メッセージ及び応答メッセージが発生する。

  一方、リモート DMA 転送においては、データ転送を可能な限り生産時に行うことにより、リクエストメッセージを省略することが可能である。送信側と受信側のメモリ空間は互いに異なるため、ハードウェア DSM において必要であったノード間のキャッシュコヒーレンス制御が必要なくなり、関連するメッセージの転送も行われない。

3 性能比較の手段

並列計算機 Cenju-4 はキャッシュ一貫性制御付分散共有メモリとユーザーレベルのメッセージ通信機構を持った分散非共有メモリの二つのメモリアーキテクチャをもつ。ノード間のデータ転送方式として各メモリアーキテクチャを反映して、ハードウェア DSM を用いたキャッシュライン単位のデータ転送とリモート DMA 転送を実現しており、両データ転送方式 (ハードウェア DSM 及びリモート DMA 転送) の性能比較を行うのに最適の並列計算機であると言える。本章では、実験環境である Cenju-4 の概要と、Cenju-4 におけるハードウェア DSM 機構及びリモート DMA 転送機構について述べる。

3.1 Cenju-4 の概要

Cenju-4 は、1024 個までのノードが、4 × 4 のネットワークスイッチから構成される多段ネットワークによって接続された構成をとる。(図 3)

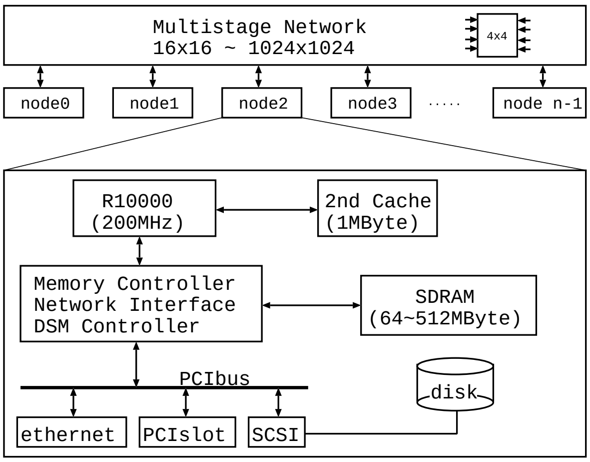


図 3: Cenju-4 の構成

各ノードは、CPU として MIPS の R10000 を搭載している。R10000 は 1 サイクルに 4 命令のデコードと、5 命令までの同時実行が可能である。R10000 は 32KByte の命令キャッシュと 32KByte のデータキャッシュを内蔵しており、SSRAM の 1MByte の 2 次キャッシュが接続されている。主記憶は、512MByte までの容量の SDRAM で構成されており、8 Byte × 80 MHz のピークスループットを持つ。

システムの鍵となる通信専用 LSI は、メモリの制御 (メモリコントローラ) と、プロセッサ間通信の機能である、分散共有メモリ (DSM コントローラ) と ユーザーレベルメッセージ通信 (ネットワークインターフェース) の機能を持つ。

Cenju-4 は、1024 ノードまで対応できるディレクトリベースの無効化型プロトコルによる DSM システムを持っている。R10000 の物理アドレスの最上位ビットで共有領域と非共有領域を区別している。メッセージ通信は非共有領域のみで行うことができる。

3.2 Cenju-4 のハードウェア DSM 機構

Cenju-4 は各要素プロセッサ毎に主記憶を分散配置している。主記憶は、全プロセッサで共有している共有アドレス空間と、各要素プロセッサ毎に独立したローカルメモリとして使用する空間にあらかじめ分割されている。

共有空間はハードウェア DSM によって管理されている。また、要素プロセッサ間で、ローカルメモリ領域においてメッセージ通信を用いた通信も可能である。図 4に Cenju-4 の分散共有メモリの構成とキャッシュシステムの様子を示す。

Cenju-4 における共有メモリは物理アドレス共有によって実現されているため、他プロセッサ上の共有メモリの内容を、自プロセッサ上の共有メモリ又はローカルメモリ上にキャッシュすることはできず、直接 1 次及び 2 次キャッシュにキャッシュされる (図 4中矢印 2, 3)。1 次及び 2 次キャッシュ上では、共有空間とローカル空間の区別なくキャッシュされる。キャッシュの一貫性制御方式は write-invalidate 型であり、主記憶領域 128 バイト毎に、ノード間の共有状態を示すディレクトリを管理している。

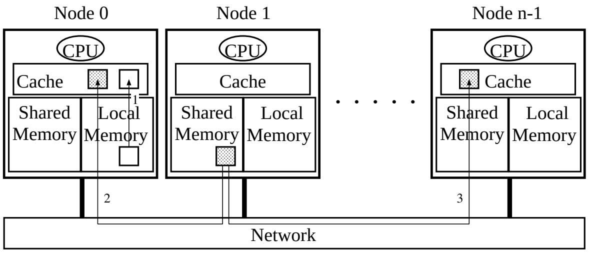


図 4: 分散共有メモリの構成とキャッシュシステム

3.3 Cenju-4 のリモート DMA 転送機構

Cenju-4 では、リモート DMA 転送の機能を持った通信ライブラリとして Paralib/CJ4 が実装されている。プログラマは Paralib/CJ4 の関数をソースコード上に記述することで明示的に通信を指示することが可能である。今回の性能評価においては、Paralib/CJ4 の関数の内、リモートメモリ書き込み及びリモートメモリ読み出しと、リモート関数呼び出しを使用している。

リモート DMA 転送においては、自ノード上のアドレス、相手ノード上のアドレス、転送データサイズを指定する。仮想アドレスを用いて転送を指示するので、相手ノード上のメモリでの保護もユーザレベルで保証できる。

リモートメモリ書き込み、及びリモートメモリ読み出しの動作を、図 5 に示す。左図においては Node P_A が、自ノードのローカルメモリ上のアドレス A から始まる L バイトのデータを、Node P_B のアドレス

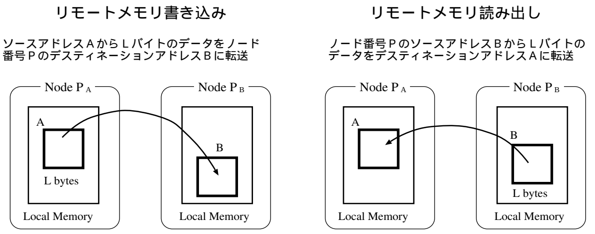


図 5: リモートメモリアクセス

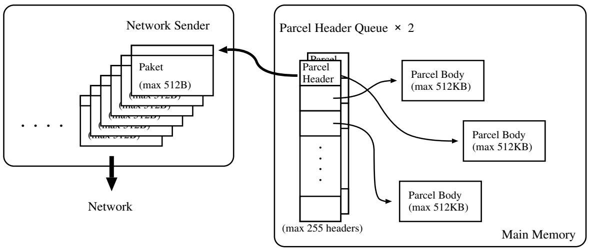


図 6: Cenju-4 の送信インターフェースの機能

B から L バイトにコピーすることを示している。右図においては Node P_A が、Node P_B のアドレス B から始まる L バイトのデータを、Node P_A のアドレス A から L バイトにコピーする事を示している。

図 6は Cenju-4 の送信インターフェース (以下 Sender) の機能を表わしている。Cenju-4 のメッセージ通信 (DMA 通信も含む) ではデータ転送の対象となるメモリ上のデータの単位を Parcel と呼び、Sender は Parcel 単位で転送データを管理する。Parcel のサイズは可変で最大 512KB まで大きくできる。送信時においてプロセッサは Pacel Header に Pacel 本体へのポインタ、転送先ノード ID、転送先ノード上でのアドレス及び転送データサイズを格納し Parcel Header Queue につなげる。Parcel Header Queue には最大で 255 個の Parcel Header をキューイングすることができる。プロセッサが送信に関わるのはここまでで、このあと Sender は順次 Parcel Header Queue から Parcel Header を読み込みその情報をもとに Parcel 本体をメモリから取り出し、約 512Byte 単位で Parcel を分割しヘッダを付加して Packet を生成する。生成された Packet は次々とネットワークに送出される。つまり、プロセッサはネットワークの状態に関係なく最大で約 256MB(128KB × 255 個 × 2) のデータを Sender に登録でき、後は Sender がプロセッサに代わって自動的に送信を行う。したがって、Cenju-4 の Sender 機構はリモート DMA 転送を効率的に実現する機能を備えていると言える。

4 SPLASH-2 の実装

本章では、Cenju-4 を用いて、ハードウェア DSM とリモート DMA 転送の性能比較を行うためのプログラム及び実装方法について述べる。実装するプログラムとしては、SPLASH-2 に含まれるベンチマークプログラムのうち、FFT、LU 分解および Radix Sort を選択した。

SPLASH-2 は元来、共有メモリ型マルチプロセッサの性能評価を目的に開発されたベンチマーク集であるため、当然、ハードウェア DSM を用いて実装した方が、良い結果が得られると予想される。本研究では SPLASH-2 の中でも比較的中粒度の通信パータを持つと思われるプログラムを選択し、ハードウェア DSM が得意とする細粒度から中粒度の通信パターンを持つ問題において、リモート DMA 転送とハードウェア DSM の性能比較を行うことを念頭に、上記の三つのプログラムを選択した。

4.1 FFT

SPLASH2 の FFT は優れた負荷分散を実現する一方で、プロセッサ間通信が最小となるように最適化されている。

4.1.1 データ構造とアルゴリズム

本アプリケーションで使用されるデータ構造は N 個 (N は 2 の偶数乗) の複素入力データが格納された $\sqrt{N} \times \sqrt{N}$ の 2 次元行列 (Source Matrix)[1] と、ひねり係数を格納した $\sqrt{N} \times \sqrt{N}$ の 2 次元行列 (Roots of Unity) 及び、計算結果の中間値と出力を格納するための $\sqrt{N} \times \sqrt{N}$ の 2 次元行列 (Scrach Matrix) である。Source Matrix と Scrach Matrix はプログラムの進行に伴い繰り返しリード、ライトされるのに対して、Roots of Unity は一度初期化されるとリードされるのみで、その内容は変更されることはない。

---

[1] 実際には複素数の実部、虚部を各々double 型でとるため N 個の複素入力データに対して $2 \times N \times \mathrm{sizeof(double)}$ のメモリ領域が確保される。

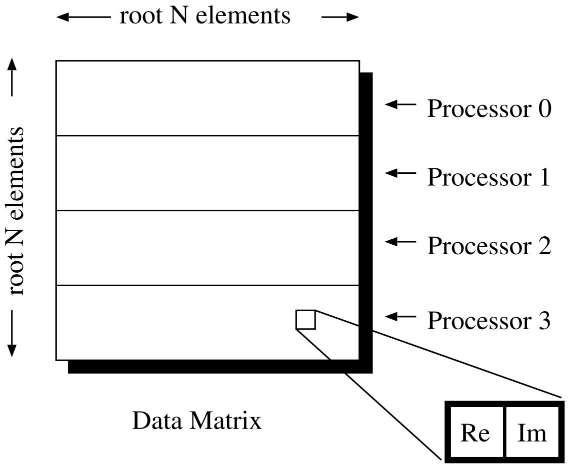


図 7: FFT のデータ構造

また各行列は、行方向に等分割されてプロセッサに割当てられるため、C 言語のような行優先の言語の場合、各プロセッサが担当する要素はアドレス空間上に連続して配置されることになる。(図 7参照)。

一方、アルゴリズムは以下に示す 6 つのステップから構成される。

- *Step*1　Source Matrix を転置し Scrach Matrix に格納する。
- *Step*2　Scrach Matrix の各行に対して 1 次元 FFT を行なう。
- *Step*3　Scrach Matrix の各要素にひねり係数を乗じる。
- *Step*4　Scrach Matrix を転置し Souse Matrix に格納する。
- *Step*5　Source Matrix の各行に対して 1 次元 FFT を行なう。
- *Step*6　Source Matrix を転置し Scrach Matrix に出力として格納する。

先に述べたように各プロセッサが担当する行列要素は行方向に連続しているため、*Step2*、*Step3*、*Step5* の操作は各プロセッサが自分の担当領域に対してのみ行うことになる。従って本アルゴリズムにおいてプロセッサ間で通信が行われるのは *Step1*、*Step4*、*Step6* における行列の転置操作のときのみに限定される。

4.1.2　ハードウェア DSM による実装

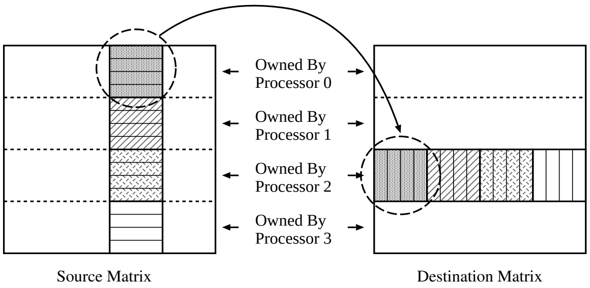


図 8: FFT の通信パターン

図 8は *Step1*、*Step4*、*Step6* において行列の転置操作を行うときのプロセッサ間通信の様子を示したものである。いま図 8の円で囲まれた部分 (以下この単位を patch と呼ぶ) に注目すると、行列全体の転置操作はこの patch を転置したものを Destination Matrix の適切な位置に転送することにより実現されることがわかる (図 8の矢印)。各 PE が担当する領域は PE 数 (= Pとする) に等しい数の patch に分割される。図 8より明らかなように、各ノード上の patch のうち 1 つは転置の後に自ノードのメモリ上に再配置され、他の $P-1$ 個の patch はプロセッサ間通信の対象となる。

個々の patch の転置には以下に述べる 4 通りの方法がある。

1. リモートノードの patch を列方向に読み、自ノードの patch に行方向に書き込む方法 (図 9(a))、
2. リモートノードの patch を行方向に読み、自ノードの patch に列方向に書き込む方法 (図 9(b))、
3. 自ノードの patch を列方向に読み、リモートノードの patch に行方向に書き込む方法 (図 9(c))、
4. 自ノードの patch を行方向に読み、リモートノードの patch に列方向に書き込む方法 (図 9(d)) の 4 つである。

なお、データの有効性を保証する同期操作は、消費時の転送であれば転置操作の直前に、生産時の転送であれば転置操作の直後に行う必要がある。

ハードウェア DSM による実装においては図 9(b) の方法を採用する。この理由は、コストの大きいリ

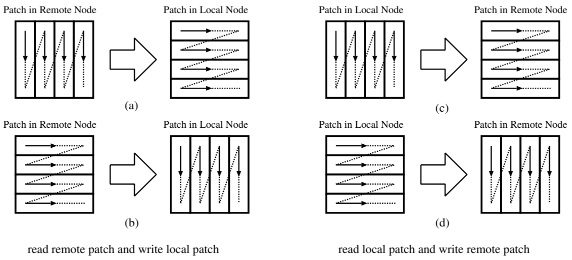


図 9: patch の転置方法

モートメモリアクセスにおいてキャッシュを有効利用するためと、生産時の転送においては更新データが生産者側のキャッシュに残ってしまい、消費者(データを割り付けられたPE)がそのデータを参照する場合、生産者のキャッシュからデータを取り出すこととなり、低速だからである。である。

4.1.3 リモートDMA 転送による実装

リモートDMA 転送による実装においては先のDSMによる実装において各PEが担当していた共有メモリ上の担当部分を自ノードのローカルメモリ上に配置する。

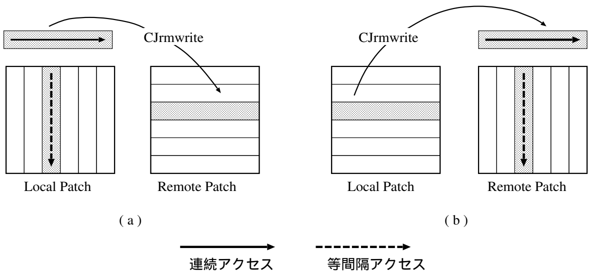


図 10: リモートDMA 転送によるデータ転送

patchの転送方式は生産時の転送を意識し、図9の(c)または(d)の方法を採る。(c)および(d)に則したpatchの転置の様子をそれぞれ図10(a)及び図10(b)に示す。図10(a)では送信側にバッファを用意し、自ノードのpatchを列方向に読んでバッファに格納する。次にリモートメモリライト関数(CJrmwrite)を用いて受信側のpatchの行にブロック転送する。

一方、図10(b)では受信側にバッファを用意し、自ノードのpatchの1行をリモートメモリライト関数を用いて受信側バッファにブロック転送する。次に受信側PEがバッファを読んで自ノードのpatchに列方向に書き込む。この方法では受信側PEに転送データのアロケート操作を強いることになり効率的ではない。したがって、本実装においては図10(a)の方法を採用する。

ブロックデータ転送の対象となるバッファの大きさはpatchの1行(列)の長さに等しく、$\sqrt{N}/P$である。従ってPE数が固定の場合、問題サイズに比例してバッファが大きくなるためブロックデータ転送の効果がより顕著に現われると予想される。

なお、バッファリングとデータ転送はオーバーラップ可能である。即ち、PEはpatchのある列をバッファリングしSenderに対して通信の通知を行うと直ちに次の列のバッファリングを開始することができる。一方、SenderはPEからの送信指令を受けPEのバッファリング作業と並行してデータ転送を行う。

4.2 LU 分解

LU分解は行列Aを$L \times U = A$をみたす下三角行列Lと上下三角行列Uに分解することである。SPLASH-2のLU分解は外積形式ガウス法に基づいている。

4.2.1 データ構造とアルゴリズム

SPLASH-2のLU分解では分解対象となる$N \times N$の行列を$B \times B (B < N)$の小行列(ブロックと呼ぶ)に分割し、すべての演算と通信はこのブロックを単位として行われる。ブロックの大きさは,1次キャッシュの容量に適合することと、十分な負荷分散が可能であることを考慮して選択される[2]。

図11はブロック分割と各プロセッサへのブロックの割当ての様子及び、本アプリケーションにおける通信パターンの様子を示している。プロセッサへのブロックの割当てには図11に示すマッピング用のメッシュを繰り返し用いてサイクリックに割り当てていく。

一方、分解の過程は以下に示す3ステップを繰り返すことにより進行する。

- *Step*1　Diagonal Block の分解
- *Step*2　Perimeter Block の更新

---

[2] 通常$B = 8$または$B = 16$が最良とされる [12]

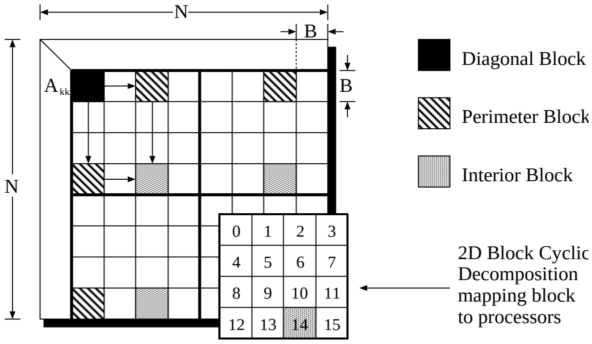


図 11: LU 分解の通信パターン

- *Step3*　Interior Block の更新

Step1 では Diagonal Block を所有するプロセッサがそのブロックを単に分解するだけであるため、プロセッサ間通信は行われない。一方、*Step2* 及び *Step3* においては自ノードのブロック (Perimeter/Interior) を更新するのにリモートノードのブロック (Diagonal/Perimeter) の参照を必要とするためプロセッサ間通信が発生する。

なお、本プログラムにおいては *Step3* の処理にほとんどの時間が費やされることになる。

4.2.2　ハードウェア DSM による実装

ハードウェア DSM による実装においては、各ブロックは消費時に転送される。即ち、当該ブロックの所有者でない遠隔 PE が Diagonal Block または Perimeter Block として当該ブロックを参照するときにリモート参照の形で転送される。

また、本実装では参照の局所性を高めてキャッシュを有効利用するための最適化を行った。分解対象となる行列を表現するのに 4 次元配列を用いている。

通常、2 次元行列を表現するためのデータ構造として 2 次元配列が用いられる。しかし、データ構造として 2 次元配列を用いた場合、各ブロックのデータがアドレス空間上に不連続に格納されるため参照の局所性が低くなるばかりか、異なるブロックに属する要素が同一キャッシュライン上に存在する可能性が高くなるため false sharing が増大することが予想される。

一方、データ構造として 4 次元配列を用いた場合、各ブロックの全データがアドレス空間上の連続領域に収まるため、参照の局所性が高まる。また各ブロックをメモリの各ページの境界を跨がないように配置することにより false sharing の発生を抑えられる。

4.2.3　リモート DMA 転送による実装

リモート DMA 転送による実装では生産時転送を強く意識した実装方針を採る。即ち、*Step1* において Diagonal Block の所有者が担当ブロックの分解作業を終えるとただちに送信コマンドを起動させて Perimer Block(同一行及び同一列のブロック) を所有するプロセッサに対し Diagonal Block のコピーを転送する。次に *Step2* において Perimer Block を所有するプロセッサは Diagonal Block のコピーが到着次第 Perimer Block の更新に取り掛る。更新が完了すると Interior Block を所有するプロセッサに対し Perimer Block のコピーを送信すると同時に次の Perimer Block の更新を開始する。最後に Interior Block を所有するプロセッサは *Step3* において行方向、列方向の Perimer Block のコピーの到着を待って Interior Block を更新する。

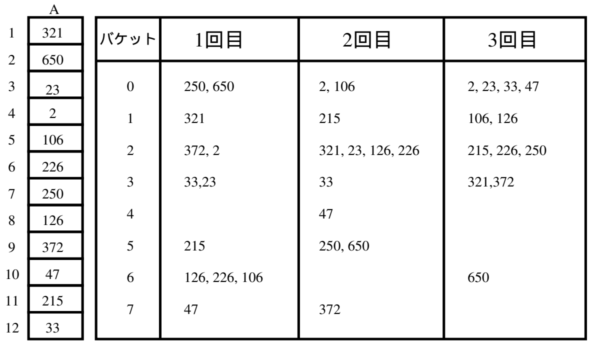
本実装方式では *Step2* において計算 (Perimer Block の更新) と通信 (コピーの転送) をオーバーラップさせることにより通信レイテンシを隠蔽できる。また、バリア同期のかわりに同期フラグを用いて各転送データに対して個別に同期をとることで同期コストを削減できるため、同期操作の最適化による性能向上も期待できる。

4.3　Radix Sort

各要素が 0 以上 $m^k - 1$ 以下の整数値、すなわち k 桁の m 進数であるとき、各桁ごとに合計 k 回のバケットソートを繰り返し適用すると、すべての要素が整列できる。図 12 にその様子を示す。まず、1 桁目の数値によりバケットソートを行う。その結果を基に 2 桁目の数値によってバケットソートを行う。この操作を最大桁 (図では 3 桁) まで繰り返すことで全要素を整列させることができる。

4.3.1　データ構造とアルゴリズム

SPLASH-2 における Radix Sort の並列化では、まず要素数に等しい大きさの int 型配列を 2 つ用意し、各配列をプロセッサ数で等分割し、各プロセッサに



	A
1	321
2	650
3	23
4	2
5	106
6	226
7	250
8	126
9	372
10	47
11	215
12	33

バケット	1回目	2回目	3回目
0	250, 650	2, 106	2, 23, 33, 47
1	321	215	106, 126
2	372, 2	321, 23, 126, 226	215, 226, 250
3	33,23	33	321,372
4		47	
5	215	250, 650	
6	126, 226, 106		650
7	47	372	

図 12: Radix ソート

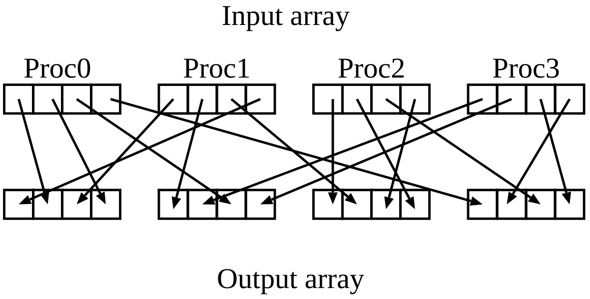


図 13: Radix Sort の通信パターン

割り当てる。次に 2 つの配列の一方に対して、各プロセッサが自分の担当領域の初期化を行う。ソートの過程は 2 つの配列のうち、最新のデータを保持している方を Input array とし、他方を Output array として、Input array から各要素を読み出し Output array へ書きこむ過程で各桁ごとにバケットソートを行う (図 13参照)。この操作を繰り返すことによって、最終的に正順された配列を得ることができる。1 回の操作は下記の 3 つのフェーズから構成され、各プロセッサは Input array の担当領域の要素に対して下記の 3 つのフェーズを実行する。

- $Phase1$　各プロセッサごとに local histogram を作成する。
- $Phase2$　個々の local histogram を結合して global histogram を作成する。
- $Phase3$　global histogram を基に Input array からデータを読み込み Output array へ書き込む。

4.3.2　ハードウェア DSM による実装

ハードウェア DSM を用いた実装においては、Input array と Output array を共有メモリに確保し、各プロセッサに割り当てる。$Phase3$ の操作は図 13に示すように各プロセッサに割り当てられた共有領域から、共有領域全体に対する書き込みとなる。

4.3.3　リモート DMA 転送による実装

リモート DMA 転送を用いた実装においては、先の DSM における Input array と Output array の各プロセッサの担当領域に相当する配列を各ノードのローカルメモリ上に確保する。1 回のバケットソートは各プロセッサが自ノードのローカルメモリ上の要素に対して先の 3 つのフェーズを実行するわけであるが、Output array に対する書き込みは自ノードあるいは遠隔ノードのローカルメモリに対して行われる。

フェーズ 3 におけるデータ転送では、送信ノード側の転送対象データの連続する部分に着目した場合、転送先ノードは不特定であり、また同一ノードに転送されるデータであってもその格納アドレスは不連続である。(図 13参照)、このため、効果的にリモート DMA 転送を行なうために一回の転送におけるデータサイズを大きくしようとすれば、送信ノードにおいて同一ノードに転送されるデータを一旦バッファリングし転送データがある大きさを越えたところで一括して転送し、受信側では受信用バッファを用意し、転送データを一旦バッファで受けて、転送データに付加されたアドレス情報を基に適切な位置に配置する必要がある。(図 14参照)。

ブロック転送の対象となるバッファのデータ構造は、転送データ本体とアドレス情報の各 4Byte から成り (図 15参照)、本来の転送データの 2 倍の大きさのデータを転送することになる。また受信ノード側にアロケート操作を強いることになる。

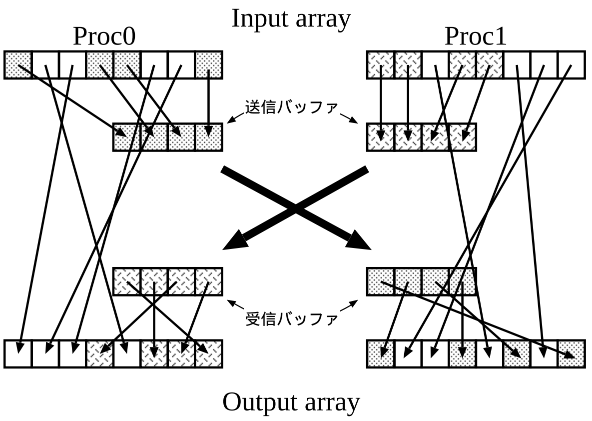


図 14: リモート DMA 転送によるデータ転送

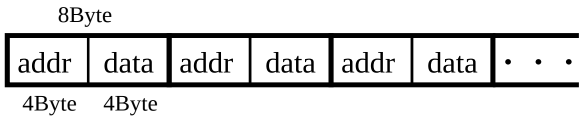


図 15: バッファのデータ構造

5　測定結果と考察

本章では、4 章で実装したプログラムの実行時間の測定結果を示し、考察を行う。

5.1　FFT

図 16にデータサイズ $N = 2^{16}$及び $N = 2^{18}$に対する実行時間を示す。いずれのデータサイズにおいてもリモート DMA 転送はハードウェア DSM を凌ぐ性能を示しているが、特に注目されるのはリモート DMA 転送においては転置時間が全体に占める割合が DSM と比較して小さいことである。これは粒度変換によるブロックデータ転送が効果的であることを示している。

また、先の予想通り固定 PE 数においては問題サイズが大きくなるに従い DSM に対する性能差が拡大している。例えば、PE 数が 32 のとき、リモート DMA 転送に対するハードウェア DSM の実行時間は、$N = 2^{16}$で約 1.2 倍、$N = 2^{18}$で約 1.4 倍となっている。

5.2　LU 分解

図17に $P = 32$、行列サイズ 512×512、1024×1024 及び 2048×2048 としたときのハードウェア DSM とリモート DMA 転送の実行時間を示す。

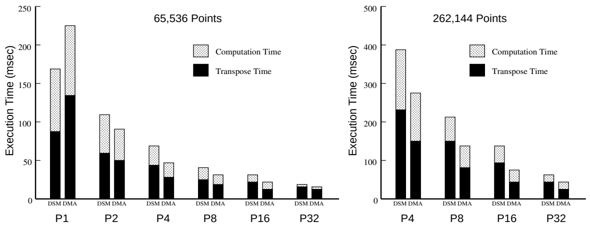


図 16: FFT 計測結果

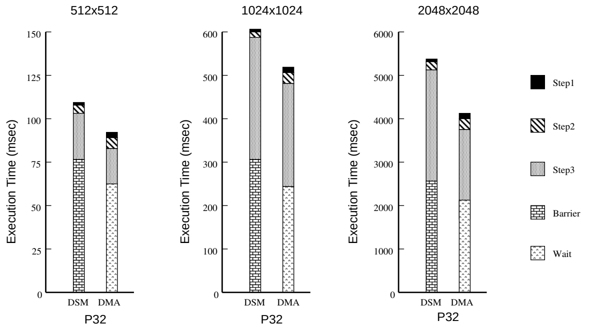


図 17: LU 分解計測結果

図 17に示されるようにリモート DMA 転送ではハードウェア DSM に比べ *Step1* と *Step2* で多くの時間を消費している。これは DMA では両ステップにおいてデータの消費者に対してデータ転送が行われるためである。一方、*Step3* においては DMA では消費データが事前に消費ノードに転送済みであるため、消費時転送の DSM と比較して高速に処理が完了していることがわかる。前述したように本アプリケーションでは *Step3* の処理に最も多くの時間が割かれるため、本ステップを高速に実行することでアプリケーション全体の実行時間を大幅に短縮することができる。リモート DMA 転送による実装はこの意図に合致している。

また、消費時転送である DSM においては *Step1* と *Step2* の直後にバリア同期をとっている。この場合、遅いノードに引きずられ同期操作に多くの時間を費やすことになる。一方、生産時転送であるリモート DMA 転送においては同期フラグを用いることで各転送データについて個別に同期を取ることができ低コストの同期操作が実現できた。図 17において、

ハードウェア DSM の Barrier Time に対しリモート DMA 転送の Wait Time は平均で 18.5 %小さくなっている。

5.3 Radix Sort

要素数 2^{18} ~ 2^{21} の実行時間計測結果を図 18 に示す。ハードウェア DSM、リモート DMA 転送ともに問題サイズが大きくなるに従って台数効果がより顕著に現れているのがわかる。

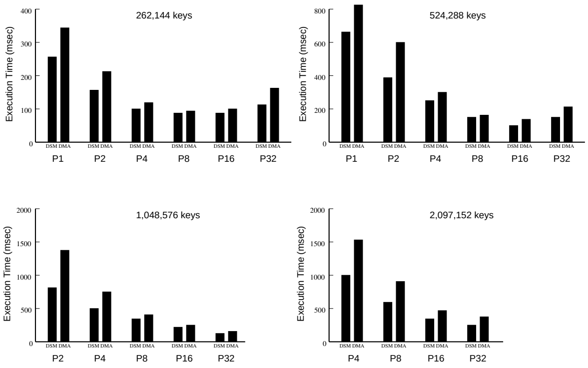


図 18: Radix Sort 計測結果

一方、図に示されるようにリモート DMA 転送はハードウェア DSM に比べ平均で 1.37 倍程度の実行時間を要している。これは、Radix Sort の場合、先の FFT や LU 分解と異り、ハードウェア DSM による実装においてもデータの生産者が共有メモリにデータを書き込むかたちでデータ転送が行われるため、生産時転送によるレイテイシの隠蔽が DSM の場合にもなされているためであり、また、リモート DMA 転送における送信ノードでの転送データのバッファリング操作と受信ノードでのアロケート操作のコストが大きいためである。

6 おわりに

本研究では、分散メモリ型並列計算機における二つのデータ転送方式、リモート DAM 転送とハードウェア DSM の性能比較を試みた。

性能比較には並列計算機 Cenju-4 を用い、SPLASH-2 の数値計算カーネルである FFT、LU 分解及び Radix Sort を両データ転送方式を用いて実装し、実行時間の計測を行った。

ハードウェア DSM では、プロセッサの load/store をそのまま遠隔メモリアクセスに拡張しており、ノード間のデータ転送はキャッシュライン単位 (固定長) で行われる。従って、ラインサイズを越えるような大きなデータをノード間で転送する場合、余分なオーバヘッドが発生する。一方、リモート DMA 転送ではパケット内のデータサイズを大きくすることにより一回の通信オーバーヘッドを低減することが可能である。このことは FFT 及び LU 分解を用いた性能比較により確認することができた。

また、ハードウェア DSM では「共有メモリの一様性」を強く意識するため、逆にメモリの配置や、同一ノードへのデータ転送の順序が保証されることを利用した、同期操作のための使い易い手段をユーザーに提供しないものが多い。このため、メモリバリアによる同期が主流であり、余分な同期オーバーヘッドを発生する。LU 分解による性能比較では、リモート DMA 転送による実装において、転送データ毎に同期をとる手法を用い同期コストを低減することが可能であることを示した。

さらに、データの受け渡し方法の観点から見た場合、現在主流である invalidate 型のキャッシュシステムを装備したハードウェア DSM では (Cenju-4 もこのタイプである)、データは必要とされた時にしかノード間で転送されないため、レイテンシの隠蔽効果は薄い。データの消費者が事前に分るような場合は、invalidate 型のハードウェアキャッシュよりは、ソフトウェアからの指示で生産時に直接リモートノードのメモリ領域に転送した方が望ましく、この観点からもリモート DMA 転送はハードウェア DSM よりも有効であることが確認できた。

一方、Radix Sort のようにブロックデータ転送を実現するのに余分なコストが必要な場合には依然として、ハードウェア DSM による細粒度のデータ転送を利用した方が有効であることが判明した。しかしながら、転送データが受信ノードの主記憶上の連続領域に転送されるようにすれば、受信ノードでのデータの再配置操作を省略でき、リモート DMA 転送を用いた場合でも高速化することができると思われ、今後の課題である。

謝辞

本研究の機会を与えて下さり、適切な御指導を賜わった富田眞治教授に深甚なる謝意を表します。

また、貴重な御助言をいただいた森眞一郎助教授、中島康彦助教授、五島正裕助手に深く感謝致します。

最後に、日頃から様々な角度からご助力頂いた、京都大学大学院情報学研究科通信情報システム専攻富田研究室の諸兄に心より感謝致します。

参考文献

[1] 広瀬 哲也 他: ”並列コンピュータ Cenju-3 のプロセッサ間通信方式とその評価” 情報処理学会論文誌 ′ 96 Vol.37 No.7

[2] 加藤 研 他: ”並列コンピュータ Cenju-4 のユーザーレベルメッセージ通信機構” JSPP ’99 論文集

[3] 細見 岳生 他: ”並列計算機 Cenju-4 の分散共有メモリ機構” JSPP ’99 論文集

[4] 山本 考伸: ”ソフトウェアからのヒントを利用するキャッシュの一貫性制御方式の提案とその予備的性能評価” 修士論文、京都大学工学部情報工学科 (1999).

[5] 白木 長武 他: ”高並列計算機 AP1000+のメッセージハンドリング機構” JSPP ’95 論文集

[6] 松本 尚 他: ”共有メモリ vs メッセージパッシング” ’97 ARC-126

[7] 松本 尚 他: ”中粒度メモリベース通信を支援する Memory-Based Processor 2” 情報処理学会 98-ARC-130-18

[8] 富田 眞治: ”並列コンピュータ工学” 昭晃堂 1996

[9] 津田孝夫: ”数値計算プログラミング” 岩波書店 1988

[10] T. von Eicken et al.: ”Active Messages: a mechanisum for integrated communication and computation” ISCA 1992

[11] Steven Cameron Woo et al.: ”The Performance Advantages of Integrating Message Passing in Cache-Coherent Multiprocessors” Tecnical Report CSL-TR-93-593

[12] Steven Cameron Woo et al.: ”The SPLASH-2 Programs: Characterization and Methodological Considerations” ISCA June 1995

[13] J.Niwa et al.: ”Efficient Implementation of Software Release Consistency on Asymmetric Distributed Shared Memory” ISPAN December 1997

[14] T.Matsumoto et al.: ”Compiler-Assisted Distributed Shared Memory Schemes Using Memory-Based Communication Facilities” PDPTA July 1998