

アクティブ・メッセージによる並列プログラム実行性能の改善

林 憲一 堀江 健志

(株)富士通研究所 並列処理研究センター

E-mail: {woods, lions}@flab.fujitsu.co.jp

概要

メッセージ・パッシングは現在並列計算機で広く用いられているプログラミング・パラダイムである。しかし、メッセージ受信時のバッファリングに関わるオーバーヘッドがあり、これがプログラム実行時のアイドル時間を生む原因の一つとなっている。一方、アクティブ・メッセージは、メッセージ毎に指定されたハンドラをメッセージ受信時に起動し、メッセージを直接ユーザのメモリ領域に書き込んだり、リモートのリード要求に対して返事をすることによって、バッファリングに纏わるオーバーヘッドを削減する。本論文では、アクティブ・メッセージを並列計算機AP1000上に実装し、行列のかけ算やSCGなど数値計算のプログラムに適用し、その効果を検討する。さらにアクティブ・メッセージを効率的に実現するためのハードウェアのサポートについて考察する。

Improvement of parallel programs performance by Active Messages

Kenichi Hayashi, Takeshi Horie

Parallel Computing Research Center, Fujitsu Laboratories Ltd.

E-mail: {woods, lions}@flab.fujitsu.co.jp

Abstract

Message passing is a popular programming paradigm in parallel computing. But it has an intrinsic overhead of message buffering at receipt which causes idle time. On the other hand, Active messages reduce the overhead of buffering by writing messages into user memory area directly. We implemented Active messages on the AP1000 and applied it to typical numerical computation programs such as matrix multiply and SCG.

1 はじめに

メッセージ・バッシングは現在並列計算機で広く用いられているプログラミング・パラダイムである。しかし、メッセージ受信時のバッファリングやバッファ内でのメッセージのサーチなど本質的なオーバーヘッドがあり、これがプログラム実行時のアイドル時間の原因の一つとなっている。一方、アクティブ・メッセージは、メッセージ毎に指定されたハンドラをメッセージ受信時に起動し、メッセージを直接ユーザのメモリ領域に書き込んだり、リモートのリード要求に対して返事をする事によって、バッファリングに纏わるオーバーヘッドを削減する。本論文では、アクティブ・メッセージを利用したPUT/GETプリミティブを並列計算機AP1000上に実装し、行列のかけ算やSCGといった数値計算のプログラムに適用してその効果を検討する。また、PUT/GETを利用するプログラミング・パラダイムである共有メモリモデルとメッセージ・バッシングのプログラミング・パラダイムとを比較・検討し、最後にアクティブ・メッセージを効率的に実現するハードウェアによるサポートについて検討する。

2 アクティブ・メッセージ

メッセージ・バッシングでのメッセージは、受信ノードに到着した後はメッセージバッファに一時的にストアされ、受信命令によって読み出されるのを待っている。これに対してアクティブ・メッセージでは、メッセージはアクティブ(能動的)である。すなわち、アクティブ・メッセージではメッセージ・ヘッダに処理されるべき方法(割り込みハンドラ)を指定して、受信時にこれを起動してその処理をCPUに行なわせる。処理の例としては、ユーザのメモリ領域の指定されたアドレスにそのメッセージを書き込むとか、リモートのリード要求に対して返事をするなどが挙げられる。アクティブ・メッセージの最大の特徴は、受信する際のバッファリングに纏わるオーバーヘッドをなくすることである。しかしアクティブ・メッセージを利用してメッセージを受信する場合は、プログラムで明示的な受信命令がないので、既にメッセージが到着しているかどうかフラグをチェックするなどの陽な同期が必要である。

2.1 メッセージ・バッシングとの比較

メッセージ・バッシングでは受信メッセージは一旦バッファリングされ、受信側の受信命令によって、メッセージがバッファからユーザ領域にコピーされる。送信側で、

1. メッセージをストアすべきアドレス

2. 受信側の受信領域をまだ利用しているか

のいずれもが分かっている場合にはメッセージ・バッシングは有効な方法である。しかし、送信側で予め(1)

が分かっている、且つ、既に(2)で受信領域の利用が終わっていることも分かっている場合には、メッセージのバッファリングは無駄であり、プログラム実行時のアイドル時間の原因になる[1]。一般的に分散メモリ型の並列計算機上でメッセージ・バッシングで書かれたプログラムでは(1)、(2)とも分からない。だが、行列の演算を扱う数値計算のプログラムのように、各プロセッシング・エレメントがほぼ同じ大きさの計算を担当し、同期しながら計算を実行して行くような場合には、メッセージをストアすべきアドレスも、受信領域が使用中かどうかも予め分かることが多い。このような場合にはアクティブ・メッセージを利用してメッセージのバッファリングに纏わるオーバーヘッドを削減することで、プログラムの実行性能が向上することを期待できる。また、アクティブ・メッセージとメッセージ・バッシングは相反する考えではないので、同一のプログラムの中で、メッセージの送受信のスケジューリングが難しい部分にはメッセージ・バッシングを用い、またメッセージの振舞いが分かっている場合にはアクティブ・メッセージを使う、といったように、両者の長所をうまく組み合わせてプログラムを作ることが可能である。

3 AP1000への実装

ここでは、アクティブ・メッセージのAP1000への実装について説明する。AP1000上に実装したのはスプリットフェーズ・リモートメモリ処理のPUTとGETである。PUT/GETは共有メモリモデルのプリミティブであるが、AP1000の場合、MMUはないが、同じプログラムをロードした場合には、各プロセッシング・エレメント(セル)でのアドレスの割り付けは同一になる。具体的には、同じプログラムをロードした時に、セル0で宣言した配列a[0]とセル1で宣言した配列a[0]のアドレスは同じになる。このため、AP1000ではハードウェアで共有メモリをサポートはしていないが、PUT/GETを利用することは可能である。

3.1 PUT/GET

PUT/GETは図1に示したような動作を行なう。それぞれの働きは以下の通りである。

PUT ローカルメモリのブロックを送り手に指定されたリモートメモリの特定のアドレスにコピーする。

GET 送り手によって指定されたリモートメモリ上のブロックを取ってきて、ローカルコピーを作る。

また、PUT/GETはフラグをチェックすることで陽に同期をとる。

3.2 AP1000への実装方式

AP1000では、短いメッセージ(8B以下)の場合と任意の長いメッセージの場合の2通りに分けて実装した。短い

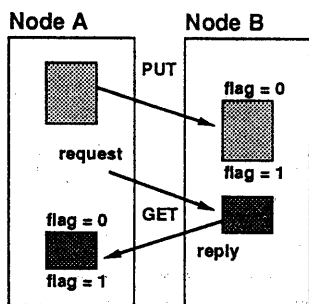


図 1: PUT/GET オペレーション

メッセージの場合には、AP1000の特徴の一つであるラインセンドを利用し、また長いメッセージの場合には、DMAを利用して送信している。受信にはリングバッファを用いず、割り込みでハンドラーを起動し、PUTの場合は受信メッセージを直接ユーザ領域に書き込み、GETの場合には、要求に対する返事をするようになっている。

8バイト以下の場合にラインセンドを、また任意の長いメッセージの場合にDMAを利用して実装した理由は以下の通りである。

1. ラインセンドで一度に送れる16バイトのメッセージの中に、ヘッダ情報などを除いて詰め込めるデータは8バイトまでであること。
2. 1キャッシュラインのラインセンドはアトミックに実行できること。
3. ラインセンドは短いメッセージを小さいオーバーヘッドで送信することが可能であるが、送信中はCPUをブロックするため、長いメッセージの送信には向いていないこと。
4. 長いメッセージを送る場合にはCPUをブロックせず、且つラインセンドより高い転送レートが期待できるDMA転送が有利であること。
5. リングバッファでメッセージを受信し、その後に割り込み用のメッセージで、リングバッファからメッセージをユーザ領域にコピーする方法も考えられるが、リングバッファからのコピーのオーバーヘッドが大きいこと。

ラインセンドは1キャッシュラインの送信毎にインストラクションを実行してメッセージを送信し、最後にラインセンドエンドを送信して全メッセージの送信を終るようになっている。このため、その間CPUをブロックする(3)。またメッセージが通過する各ルーティング・コントローラ(RTC)はラインセンドエンドが来るまで、ヘッダに書かれた目的地までの通信路を確保している。このため、ラインセンドの途中でGETの割り込みが入って、

その処理(GET要求に対するリブライの送信)を行なうとメッセージの送出先が間違ってしまう。これを避けるためにはラインセンドの途中で割り込みが入らないようにする必要がある。ラインセンドを排他的に実行するためには、現在のAP1000のOSでは、システムコールで行うのが現実的な方法だが、これはオーバーヘッドが大きいのでラインセンドの特徴が活かせない。このため、排他的に実行できる1キャッシュラインのラインセンドで送れる8バイトのPUT/GETを実装することにした(2)。16バイトや24バイトのデータのPUT/GETを利用したい場合には、8バイトのPUT/GETを連続して用いる方が、システムコールを利用して排他的にラインセンドを行なうより現在の実装では有利である。

一方、並列計算のプログラムでは行列の演算のように大きなデータを一括して送信する場合がある。このような場合には、CPUをブロックしないDMA転送を用いた方が有利である。このため、任意の長さのPUT/GETにはDMA転送を送信に利用する方式を採用した(4)。AP1000のDMA転送はシステムコールの中でのみ実行できるようにになっているので、DMA転送中に他の割り込みのための転送が行なわれることはない。また長いメッセージの受信には、DMAを用いる方法の他に、AP1000の特徴の一つである、リングバッファを用いる方法も考えられる。送信時にメッセージヘッダに割り込みではなく、リングバッファを指定することで、メッセージはリングバッファに自動的に入る。メッセージ本体を送信した後に割り込み用のメッセージを送信し、リングバッファに既に入っているデータをユーザ領域にコピーするようなハンドラを起動することで、アクティブ・メッセージの機能は実現できるが、リングバッファからのコピーはCPUをブロックするため長いメッセージに対してはオーバーヘッドが大きいと判断した。このため、任意の長いメッセージの受信には、割り込みでDMA転送を利用する方式を採用した。

3.3 8バイト以下のPUT/GET

8バイト以下の短いメッセージの場合のPUT/GETのメッセージの構成を図2に示す。AP1000のラインセンドでは一度に1キャッシュライン(16B)を送ることができる。まずPUTのメッセージ構成では、ヘッダにはRTC用のデータと起動する割り込みハンドラの種類が入る。次の4Bには書き込みリモートのアドレス(24ビット)とフラグを指定するための情報(8ビット)が入る。残りの8BにはPUTするデータが入る。AP1000のローカルメモリは16MBあるので、アドレスを一意的に示すには24ビットの情報量が必要である。しかし、1キャッシュラインに8バイトのデータを入れて送る場合には、フラグのために24ビットを用意することができない。このため、フラグはそのアドレスで送らずに、フラグを区別するための識別子(添字)を8ビットの情報で送ることとする。具体的に

はフラグを予め初期化の段階で必要な数だけ宣言し(例えば `int flag[10]` など), PUT/GETではその添字を送ることでフラグを区別する。8ビットの情報なので最大で256種類のフラグしか利用できないが、普通はこれだけあれば充分である。

次にGETのメッセージ構成は、ヘッダにRTC用のデータと割り込みハンドラの種類が入り、次の4BにGETを要求したローカルノードのID、次の4BにはGETしたデータを書き込むアドレス(24ビット)とフラグを指定するための情報(8ビット)が入る。最後の4BにはGETするリモートのアドレスが入る。

PUTで、データを受信するノードでは、RTCのレジスタからデータを読んで、直接指定されたユーザ領域にストアする。

ただし、PUT/GETいずれの場合も送信するメッセージのアラインメントは揃うように予め作っておく。

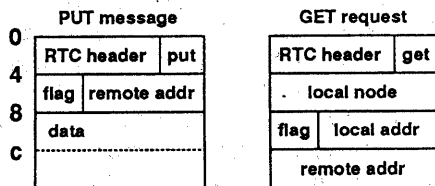


図 2: メッセージの構成(LSENDの場合)

3.4 任意の長さのPUT/GET

次に任意の長さのメッセージをPUT/GETするためにDMA転送を利用した場合のメッセージ構成を図3に示す。AP1000の通常のDMAベースの送信では、まず送信するユーザのデータをシステム領域のメッセージヘッダの後ろのコピーして、連続領域にしてDMA送信を行なっている。しかし、送信時のメッセージのコピーはオーバーヘッドが大きいので、PUTによる送信ではこのコピーを行わずに、直接ユーザ領域からDMA転送を行なう。また受信の際にもユーザ領域に直接DMAでメモリ上に書き込みを行なう。このため、PUTする側もされる側もDMAが起動している間、メッセージを送受信している領域にアクセスしないようにする必要がある。今回の実装ではPUTする側には `send_dma_complete_flag`, PUTされる側は `put_flag` を設けて送受信領域をプログラムの責任で保護することになっている。すなわち、PUTする側では `send_dma_complete_flag` をチェックして、このフラグが変わるまで送信領域にアクセスしない。またPUTされる側では `put_flag` をチェックして、このフラグが変わるまで受信領域にアクセスしない。

このようにDMAを用いたPUTではヘッダとメッセージ本体が連続領域ではないので、プリアンブル付きのDMA転送を利用している。プリアンブル付きのDMA転送はAP1000のハードウェアがサポートしている機能であ

る。プリアンブル付きのDMA転送では、図3に示すようにHDPTRの指すヘッダとMADDRの指すメッセージ本体が別の場所にあり、それぞれのアドレスとサイズを指定することで、DMA転送が連続的に行なえる。

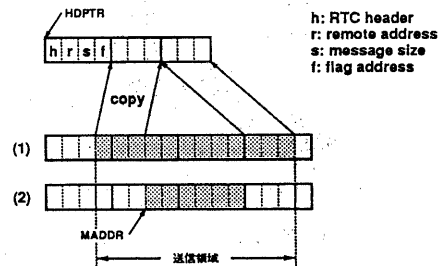
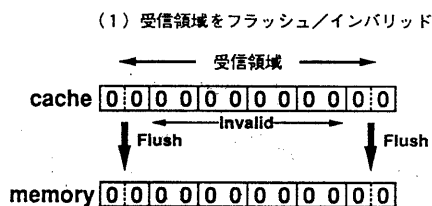


図 3: メッセージの構成(DMAの場合)

3.5 ミスアラインメントの問題

前節で述べたようにDMA転送中は、送受信を行なっている領域へのアクセスはプログラムの責任で行なわないことにするが、その前後の領域へのアクセスをプログラムの責任にすることはできない。もし受信領域のアラインメントがそろっていない場合、DMA転送開始後に受信領域の前後にアクセスがあると、図4に示すような不具合が生じる可能性がある。DMAでメッセージを受信する場合には、まず図4の(1)に示したように受信領域のキャッシュのインバリデートを行なう。この時、受信領域のアラインメントが揃っていない場合には、その部分はメモリに反映するようにフラッシュを行なう。その後(2)のようにDMAの転送を開始する。図4の(2)ではメモリに1が書き込まれている様子を示す。DMAは転送を開始すると処理をシステムモードからユーザに返し、ユーザプログラムの実行が始まる。この時、(3)に示したように受信領域に連続する部分に対して、アクセスが行なわれる可能性がある。受信領域のアラインメントが揃っていない場合には、受信領域に連続する部分に対するアクセスがあると、受信領域の最後(あるいは最初)の部分がキャッシュに乗ってしまう。このあとDMAによる書き込みがメモリに対して行なわれた場合には、キャッシュとメモリとでデータの整合性がとれなくなってしまう。

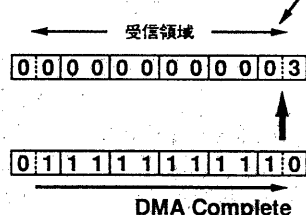
これを避けるために、DMAを利用したPUTでは図3に示すような工夫をしている。プリアンブル付きDMA転送のヘッダにはヘッダ情報の他に、データ本体の先頭と末尾の3ワードのデータを含んでおり、10ワードある。またデータ本体はデータの先頭と末尾の3ワードを除いた領域としている。PUTの割り込み受信では、まずヘッダの10ワードをRTCのレジスタを読んで受信する。そしてそこに含まれているデータ本体の先頭と末尾の3ワードをストアした後、DMA転送によって残りの部分のデータの書き込みを行なう。これによって、受信領域がミスアライン



(2) DMA GO



(3) 受信領域の後ろにアクセス



最後のキャッシュラインでは、前半はメモリがキャッシュより新しく、後半はキャッシュの方が新しいという不具合が生じる

図 4: ミスアラインによる不具合

メントの場合も図4に示したようなことはなくなる。

3.6 任意の長さのGET

任意の長さのGETの要求のメッセージは短いのでラインセンドで送っている。それに対する返事は割り込みハンドラで任意の長さのPUTを起動して、前に述べたのと同じ方法でPUTを行なっている。

3.7 AP1000でのPUT/GETの性能

表1にAP1000でのPUT/GETに要する時間を示す。比較のために nCUBE2 と CM-5 でのPUT/GETの時間も示す[2]。

まず、8バイト以下のPUT/GETでは、AP1000の性能はCM-5とはほぼ同じ性能になっている。PUT/GETともそれぞれメッセージの送出にかかる時間は約1.2 μ sである。PUT/GETにかかっている時間の大半は割り込みハンドラのオーバーヘッドの時間である。現在までの実装では、アセンブラで書かれた割り込みルーチンからC言

	PUT	GET
AP1000	6 μ (8B以下) 68 μ +0.081 μ /B	12 μ (8B以下) 73 μ +0.081 μ /B
nCUBE2	26 μ +0.5 μ /B	52 μ +0.5 μ /B
CM-5	7 μ (16B以下) 19 μ +0.1 μ /B	12 μ (16B以下) 25 μ +0.1 μ /B

表 1: PUT/GETの性能

語で書かれたハンドラを呼んでいるが、この部分に冗長なコード、SPARCのレジスタウィンドウをフラッシュする操作、が含まれている。C言語の部分の全てをアセンブラで記述し、レジスタウィンドウのフラッシュを省略することで、PUT/GETにかかる時間はかなり減ると予想している。

次にDMA転送による任意の長さのPUT/GETはnCUBE2やCM-5と比べて、オーバーヘッドがかなり大きくなっている。この原因としては以下のものが考えられる。

- DMA転送の設定はシステムコールを利用するが、その中のアセンブラのルーティンからC言語のルーティンと呼んでいる部分にレジスタウィンドウを操作するための冗長なコードがある。
- DMA転送のスタートのためにセットするレジスタの数が多。
- DMA転送を開始する前に送信領域のキャッシュのポスト、また受信領域のキャッシュのフラッシュ/インバリッドが必要であり、このキャッシュ操作を1キャッシュライン分ずつ命令を発行している。

ソフトウェアの部分に関しては、8バイト以下の時と同様に全てアセンブラで記述し、レジスタウィンドウのフラッシュを省略することで、PUT/GETともそれぞれ10数 μ sの削減が可能である。それ以外の部分は現在のAP1000のハードウェアの限界である。また表1でPUT/GETでは1バイトあたり0.081 μ sかかっている。これはDMAによる転送レートが約12.3MB/sであることを示している。AP1000のネットワークの転送レートは25MB/sであるので、ピーク性能の約半分である。これはDMA転送を行なう前にキャッシュの操作を必要とするからである。試みにDMA転送を行なう前にキャッシュの操作を行なわなかった場合の転送レートを測定すると24.2MB/sであり、ほぼピーク性能に近くなっている。すなわち、キャッシュの操作のために転送レートが半減していることが分かる。これらハードウェアの問題については、6章のハードウェアによるサポートで検討する。

4 アプリケーションでの利用

アクティブ・メッセージによって書き換えたのは行列のかけ算と二次元ボアソン方程式をSCG法[3]で解くプロ

グラムである。

4.1 行列のかけ算の場合

行列のかけ算のプログラムを任意の長さのPUTを使って実現し、その性能を評価した。計算は $N \times N$ の行列 A, B, C について $C = A \times B$ を求めるものである。プログラムの一部をList1に示す。

List 1 行列のかけ算

```

1  cell_main()
2  {
3      int jb, j0, dj, next_cell;
4      int put_flag = 0;
5      int dcf = 0;
6
7      cid = getcid();
8      ncel = getncel();
9
10     Rp = N/ncel;
11     j0 = cid * Rp;
12     lput(cid, A, N*Rp*sizeof(double),
13         nV, &put_flag, &dcf);
14
15     sync(0,0);
16
17     for(dj = 0; dj < ncel; dj++){
18         jb = (j0 + dj * Rp) % N;
19         next_cell = (cid-dj-1+ncel)%ncel;
20
21         AMcheck(&put_flag);
22         tmpV = V; V = nV; nV = tmpV;
23
24         if(next_cell != cid){
25             lput(next_cell, A,
26                 N*Rp*sizeof(double), nV,
27                 &put_flag, &dcf);
28         }
29
30         for(i=0; i<Rp; i++){
31             for(j=0; j<Rp; j++){
32                 tmpC = 0.0;
33                 for(k=0; k<N; k++){
34                     tmpC += V[i*N+k] * B[j*N+k];
35                 }
36                 C[j*N+jb+i] = tmpC;
37             }
38         }
39         sync(0,0);
40     }
41 }

```

このプログラムを16セルで実行した時のセルあたりのFLOPS値を計算したものを図5に示す。またメッセージ・パッシングで同じプログラムを実行した結果も示す。図5でサイズとは行列のサイズ $N \times N$ の N を表す。

セルのFPUのピーク性能は倍精度で5.56MFLOPSだが、ここでの実行最高性能はPUTを使った場合は3.30MFLOPS、ラインセンド・バッファレシープ(LS-BR)を使った場合は2.92MFLOPSになっており、約13% PUTを使った方が速くなっている(表2)。

またList1とは別に1セルで通信を行なわない行列のかけ算のプログラムを実行したら実行最高性能は3.40MFLOPSであった。このことから、PUTを利用した場合には単体での実行最高性能の97.1%の性能を実現

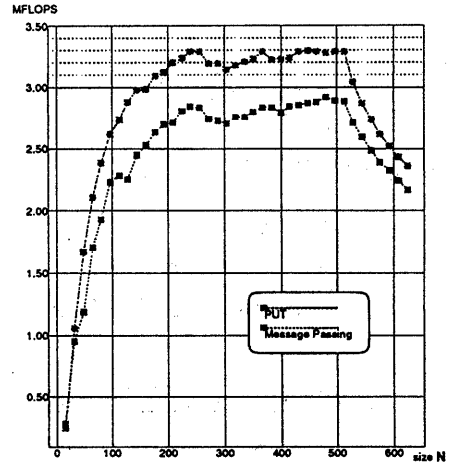


図 5: 行列のかけ算

	MFLOPS	割合
FPU のピーク性能	5.56	-
セル単体での実行最高性能	3.40	100%
PUTでの実行最高性能	3.30	97.1%
LS-BRでの実行最高性能	2.92	85.6%

表 2: 行列のかけ算の実行性能

していることが分かる。

メッセージ・パッシングのプログラムはList1と基本的にアルゴリズムは同じである。性能の差は、PUTを使った場合には、次の計算に必要なデータが計算をしている間に既に入力されているのに対し、メッセージ・パッシングでは、メッセージは既にリングバッファに届いているが、受信命令を発行するまではユーザ領域にコピーされないために、コピーにかかる時間だけ、CPUがアイドルしてしまうためである。このようにアクティブ・メッセージを使ったプログラムでは計算と通信をオーバーラップさせることが可能であるが、メッセージ・パッシングではバッファからのコピーが必ず必要となるために通信と計算をオーバーラップさせることができない。すなわち、メッセージ・パッシングの問題である受信時のバッファリングに纏わるオーバーヘッドが全体の性能を下げていることが分かる。

4.2 SCGの場合

次にSCG法にPUTを適用した場合の性能を図6に挙げる。SCGは二次元ポアソン方程式を差分法で解いたものである。グラフの横軸は問題のサイズで、二次元の領域を $n \times n$ に分割した時の n の値である。また縦軸は16プロセッサで実行した時のMFLOPS値である。SCG法では

通常の通信の他にグローバル演算を利用しているが、今回の性能測定ではこの部分にはアクティブ・メッセージベースのグローバル演算は利用していない。ここで図6にはAP1000の通信方式の内、割り込み送受信をベースにしたものを利用した場合とラインセンド・バッファレシーブをベースにしたものを利用した場合の2通りについても性能を測定している。PUT は基本的に割り込み送受信をベースにしているが、送信時のコピーと受信時のバッファリングを削減する効果が大きく出ていることが分かる。また、ラインセンド・バッファレシーブを用いた場合に対してもほぼ同等な性能を発揮している。

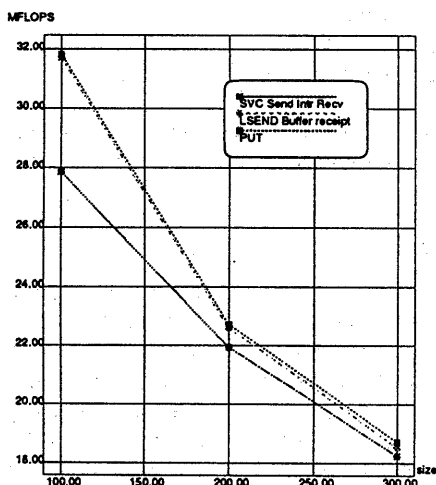


図 6: SCG にPUTを適用した場合

ここで、ラインセンド・バッファレシーブを用いたものの性能差が行列のかけ算の場合と比べて小さいのは、送信しているデータのサイズが小さいため、長いメッセージの転送に有利なDMAの特徴が発揮されていないからである。例えば図6のサイズ300の場合に一回に転送するメッセージのサイズは2400バイトであるのに対し、行列のかけ算の場合(図5)のサイズ512の場合のメッセージサイズは131Kバイトである。ちなみに、サイズが大きくなったときに性能が落ちるのは、キャッシュのヒット率が落ちるからである。

4.3 評価

行列のかけ算やSCGで見てきたようにアクティブ・メッセージを利用するとプログラムの演算性能が改善されることが分かった。SCG の所でふれたように、AP1000には割り込みベースとラインセンドベースの2つの通信方式がある。AP1000の場合にはラインセンドベースの通信を高速に行なうためのハードウェアを備えているが、割り込みベースの通信には特別のハードウェアは持っていない。それにも拘らず、割り込みベース

の通信を利用しているPUT/GETを用いると、特別のハードウェアを持っているラインセンドを利用した場合とほぼ同等の性能を実現している。これはアクティブ・メッセージによってメッセージのバッファリングに纏わるオーバーヘッドをなくすことが、特別のハードウェアと同等の効果を持つことを示している。

5 プログラミング・パラダイム

アクティブ・メッセージの最大の特徴は受信時にメッセージをバッファリングしないことである。これによってプログラムの実行性能が向上することは前節までに見てきた通りである。しかし、このことが短所になる場合がある。アクティブ・メッセージ、例えばPUTでは割り込み先で無条件にデータの書き込みを行なう。これはまだ利用しているデータやこれから使おうとしているデータを破壊する可能性がある。これに対してメッセージ・バッファリングではデータは一旦、メッセージ・バッファにバッファリングされ、必要な時にバッファから取り出すため、データの破壊が起きることはない。このようにアクティブ・メッセージの最大の特徴はまた、データを破壊する可能性があるという欠点を合わせ持つ。逆に言えば、メッセージ・バッファリングではデータを壊さない代わりに、バッファリングのオーバーヘッドが必要である。

PUTによるデータの破壊を避けるためには行列のかけ算の場合に示したように、書き込み先のアドレスを交互に換える方法などを利用してプログラミングを工夫する必要がある。

またList1の39行目に示したようにPUTを使う場合には、ループ一回毎にバリア同期(sync(0,0))をとる必要がある。GETではこの同期は必要ないが、割り込みが二回必要となるので、効率が落ちる可能性がある。しかし、プログラムの分かり易さから言えばGETを使った方がいかもしれない。

アクティブ・メッセージではデータを破壊する可能性があることと陽に同期を取る必要があるという大きな欠点を持つが、AP1000のようにアクティブ・メッセージを実装するには必ずしも有利でないアーキテクチャのマシンでもプログラムの実行性能が向上することは大きな魅力である。また、行列の演算を扱うような数値計算のアプリケーションの場合には、予め送信元でデータをストアするアドレスが分かっているし、計算は各プロセッシング・エレメントで同期的に進むので、アクティブ・メッセージの欠点はあまり影響がないはずである。

6 ハードウェアによるサポート

ここではAP1000でアクティブ・メッセージを効率的に実行するためのハードウェアによるサポートについて考察する。

まず、現在AP1000でアクティブ・メッセージを実装する上でオーバーヘッドの原因となっているのは以下の間

題である。

1. DMA 転送のセットアップ
2. DMA 転送を開始する前のキャッシュの操作
3. 割り込み処理

(1)のDMA転送のセットアップによるオーバーヘッドの問題であるが、これはDMA転送をする前にセットするレジスタの数が多いことが原因である。現在のPUT/GETの実装では、DMAによる転送を開始する前にセットするレジスタの数は5個あり、またそれぞれセットするデータのフォーマットが複雑で、これを一つ一つソフトでセットしているので時間がかかる。AP1000のもう一つの通信方式であるラインセンドではメッセージ・コントローラ(MSC)によって、1回のストアでMSCが自動的に送信を行なうようになっているので、低オーバーヘッドを実現している。これと同様なインターフェースがDMA転送に関しても実現できるはずである。

(2)のキャッシュの操作は、DMA送信を開始する前のキャッシュのポストと受信を開始する前(あるいは後)のキャッシュのインバリデートのことであるが、これがDMAの転送レートを半減させていることは既に述べた。これは、現在の実装では送受信領域のキャッシュのポストとインバリデートを全て実行してから、DMA転送を開始しているためである。しかし、DMA要求に対してMSCがキャッシュのスヌープ機能を持っていれば、これらの操作はハードウェアで実現できる。すなわち、DMAで送信を行なおうとしているデータがキャッシュに乗っていればキャッシュから、そうでなければメモリからデータを転送することによって送信時のキャッシュのポストは必要ないし、受信の時には、MSCが必要な領域のフラッシュまたはインバリデートを行えば、ソフトウェアでこれらの操作は必要ない。これによってネットワークのピークに近い転送レートが達成できるはずである。

(3)の割り込み処理については、現在の実装では、PUT/GETによるメッセージが到着した時に割り込みを発生してハンドラを起動しているが、これをCPUが実行しているので、この間ユーザプログラムの実行はブロックされている。アクティブ・メッセージでは計算と通信をオーバーラップさせるのがその特徴の一つであるが、割り込みハンドラをCPUが実行しているのでは、本来の意味で通信と計算はオーバーラップできない。またAP1000のプロセッサに用いているSPARCプロセッサはウィンドウを切替えるだけで、割り込み処理を実行できるため、割り込み処理に対するコストは小さいが、現在のスーパースカラーのプロセッサでは割り込みのコストは大きくなっている。これらを考慮すると、メッセージが到着した時の処理はCPUが実行すべきではなく、メッセージハンドラ用の簡単なプロセッサを用いて実行すべきだと思われる。このようなプロセッサを用意すること

で計算と通信を完全にオーバーラップさせることが可能となる。

これまで述べてきたようなハードウェアのサポートによって、アクティブ・メッセージを実現するためのオーバーヘッドを小さくすることが可能となる。しかし、PUT/GETのようにソフトウェアでグローバルメモリに対するアクセスを行なうためのオーバーヘッドは、特に小さなメッセージの場合には、依然として無視できないと考えられる。このようなグローバル領域に対するアクセス、特に小さなサイズの場合は、PUT/GETのようにソフトウェアで実現するのではなく、キャッシュのコヒーレンスの機能をハードウェアで実現することによって行なうべきであろう。

小さなサイズのグローバル空間へのアクセスはキャッシュのコヒーレンスによってハードウェアで、また大きなサイズのグローバル空間へのアクセスはPUT/GETで、それぞれ効率的に実現することができると考えられる。

7 まとめ

アクティブ・メッセージを並列計算機AP1000に実装し、行列のかけ算やSCGなどの数値計算のプログラムに適用して演算性能が向上することを示した。また、アクティブ・メッセージを効率的に実行するためのハードウェアのサポートについて考察した。今後は共有メモリモデルとメッセージ・パッシングモデルの長所を活かし、これらを統合するアーキテクチャ並びにプログラミング・パラダイムについて検討していく予定である。

謝辞

日頃御指導、御助言いただく、並列処理研究センター石井センター長、白石担当部長、池坂主任研究員、ならびに研究室の同僚諸氏に感謝いたします。

参考文献

- [1] T. Horie, K. Hayashi, T. Shimizu, and H. Ishihata. Improving AP1000 parallel computer performance with message communication. In *The 20th Annual International Symposium on Computer Architecture*, pp. 314-325, May 1993.
- [2] T. von Eicken, D. E. Culler, et al. Active Messages: a mechanism for integrated communication and computation. In *19th International Symposium on Computer Architecture*, pp. 256-266, 1992.
- [3] 速見 謙. SCG法による拡散方程式の解法. コンピュータ・ロール, Vol. 26, pp. 32-39, 1989. コロナ社.
- [4] 石畑宏明, 稲野聡, 堀江健志, 清水俊幸, 池坂守夫. 高並列計算機AP1000のアーキテクチャ. 電子情報通信学会論文誌D-I, Vol. J 75-D-I, No. 8, pp. 637-645, 1992.