高速メッセージハンドリング機構

——AP1000 における実現——

清水俊幸† 堀江健志† 石畑宏明†

低レイテンシメッセージ通信は，高速な並列計算機実現の鍵である．本論文では，低レイテンシメッセージハンドリング方式について述べる．提案する方式は，キャッシュメモリから直接メッセージをネットワークに送信し，ネットワークからメッセージバッファに直接受信する．本方式を実現したハードウェアおよびソフトウェアの詳細について述べ，性能について評価を行い，その有効性を示す．

Low-Latency Message Communication Support for the AP1000

Toshiyuki Shimizu,† Takeshi Horie† and Hiroaki Ishihata†

Low-latency communication is the key to achieving a high-performance parallel computer. In using state-of-the-art processors, we must take cache memory into account. This paper presents an architecture for low-latency message communication and implementation, and performance evaluation. We developed a message controller (MSC) to support low-latency message passing communication for the AP1000, to minimize message handling overhead. MSC sends messages directly from cache memory and automatically receives messages in the circular buffer. We designed communication functions between cells and evaluated communication performance by running benchmark programs such as the Pingpong benchmark, the LINPACK benchmark, the SLALOM benchmark, and a solver using the scaled conjugate gradient method.

1. はじめに

これまで，多くの分散メモリ型並列計算機 (DMPP) が発表されてきた．Cosmic Cube[1]，MarkIIIfp[2]，J-Machine[3]などの研究目的ばかりでなく，iPSC[4]やnCUBE は製品として利用されている．

DMPP は，プロセッサ台数の拡張が容易で，かつ低コストであるが，実際のアプリケーションで高い性能を得ることは難しい．プロセッサの数が増えれば，通信回数が増加しオーバヘッドが顕著となる．効率を高めるには，通信が占める時間を短縮しなければならない．このため DMPP 開発にあたっては，プロセッシングエレメントの単体性能を高めるとともに，ネットワークのスループットを上げ，レイテンシを下げること，メッセージハンドリングのオーバヘッドを下げることが重要な課題である．

ネットワークスループットを上げ[5]，ネットワークレイテンシを下げる[6),7]方式が研究され，さらに，メッセージハンドリングを効率的に行うための研究も数多くなされている[3),8)~11]．

Kermani と Kleinrock は，通信経路上にあるプロセッサのインタラクションを無くすために，virtual cut-through を提案した[6]．Dally は，経路上のプロセッサによるメッセージのバッファリングを避ける wormhole routingを提案し[7]，また，デッドロックを避け，スループットを向上させる virtual channel を発表している[5]．

このように，ネットワークのレイテンシが下がり，スループットが上がると，メッセージハンドリングのオーバヘッドが有意になってくる．iWarp は，シストリックコミュニケーションを採用した[11]．これは，送信／受信がうまくスケジューリングされている場合は効率が良い．しかし，このように厳密にスケジューリングされたプログラムを書くのは難しい．

Nikhil と Papadopoulos はデータフロー計算のたトめに，*T[9]を開発した．これは，トークンのリモートアクセスとプロセスのスケジューリングのためのプロセッサ (synchronization processor) を付加している．このリモートアクセス機構はメッセージパッシングに効果的に適用できるが，トークンの長さが制限されているので，メッセージが分断され，トータルなメッセージサイズが増加し，それが，ネットワークを高

† (株)富士通研究所
Fujitsu Laboratories Limited

負荷にする可能性がある．

Hsu と Banerjee はプロセスのスケジューリング，メッセージバッファの管理，メッセージのコピーを行う message passing coprocessor (MPC) を開発し，メッセージ通信を高速化した[10]．Dally による Message-driven processor (MDP) は，受信中のメッセージを参照可能としている[3),8)]．

最新のプロセッサを使用するためにはキャッシュメモリの使用は必須であるが，以上に挙げた研究には，キャッシュメモリの使用に関して考慮されていなかった．本論文では，キャッシュの使用を考慮し，メッセージハンドリングのオーバヘッドを小さくする通信機構，line sending と buffer receiving を提案する．2章では，このメカニズムの効果を示すために，メッセージ通信のタイミングについて考察する．3，4章では line sending と buffer receiving を実現するハードウェアおよびソフトウェアの詳細について述べ，5章において本方式を実装した分散メモリ型並列計算機 AP1000[12)~14)] における性能について評価を行う．

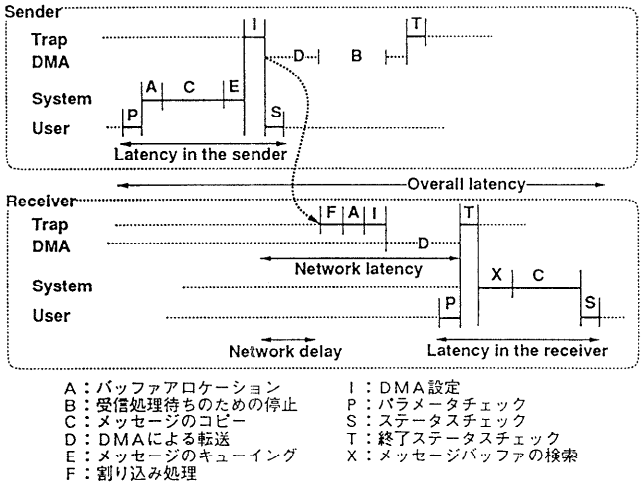


図 1 従来方式の通信タイミング
Fig. 1 Conventional steps for sending and receiving a message.

2. メッセージ通信のタイミング

本章では，従来方式の通信タイミングと，提案する方式の比較を行う．

2.1 従来方式

従来方式の通信タイミングを，**図1**に示す．このタイミングは最適の場合で，メッセージが到着しなければ受信プロセッサはアイドル状態になる．

Overall latency がもっとも重要である．これは，送信プロセスがメッセージを送り出そうとしてから，受信プロセスがメッセージを使用できるまでの時間である．

高速なネットワークを用いても，受信側プロセッサによる受信処理のセットアップが終了するまでの待ちによる停止(B)は避けられない．メッセージが到着する前に DMA をスタートさせておけば，この停止を避けることができるが，この場合は，到着したメッセージを調べてメッセージ長を知ることができないので，安全のために大きなバッファを用意しておかなければならない．また，転送終了後，DMA を再スタートさせるための割り込み処理も無視できない．

図1のDを除く記号すべてがメッセージハンドリングのオーバヘッドである．データのコピーはメッセージバッファのアドレスを渡すことによって避けられる．iPSC[4)]は，isend() と irecv() で，このようなインタフェースを提供している．

MPC[10)]は，割り込み処理，メッセージバッファ管理，メッセージコピーを処理することによって，CPU がメッセージ処理に消費する時間の削減に成功した．MPC を利用するためには，ユーザはメッセージのバッファを明示的にアロケート／デアロケートしなければならないが，この制限を設けることにより，コピーのオーバヘッドをなくしている．しかし，コピーや割り込み処理は，依然として MPC によって実行されるため，overall latency はそれほど改善されない．

MDP[8)]は，到着しつつあるメッセージに対してアクセスすることを許している．これによって，メッセージの到着完了を待たなくてもよく，パイプライン的な処理が可能となっている．しかし，キャッシュメモリに関する考慮はされていない．

2.2 Line sending と buffer receiving

コピーバック方式のキャッシュメモリは，計算性能を考慮した場合，ライトスルー方式に比べ有利であるが，メッセージを，DMA がアクセス可能なメインメモリに書き出すために，送信 DMA の起動前にキャッシュ不可の領域へコピーする必要がある．

キャッシュ不可の領域へのアクセスは非常に遅い．キャッシュのフラッシュによってコピーを避けるにも

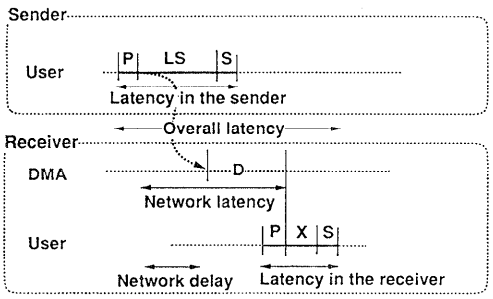


図 2 MSC によるメッセージパッシング
Fig. 2 Message passing performed by the MSC.

5章で述べるように，アプリケーションプログラムにおいて，送信するメッセージの 90% 以上はキャッシュメモリにのっており，これらを書き出すコストは高い．しかも，DMA 終了まではメッセージ領域への書き込みを制限する必要がある．

我々は，line sending と buffer receiving を提案する．**図 2** は，この方式のタイミングである．LS は，line sending である．他の記号は図 1 と同じである．

Line sending には以下の効果がある．

- Line sending は，キャッシュのフラッシュに類似の方法でデータを送出する．コピー，DMA 設定，割り込み処理が不要である．
- Line sending は，バッファリングされないので，データは直ちに転送される．
- すべての処理はユーザモードで行われるため，OS のサービスコールが必要ない．トラップが生じるのは，ネットワークが輻輳してデータの書き込みができなくなった時だけである．

メッセージの送出が高速に行われても，受信プロセッサで割り込み処理，バッファ確保，DMA 設定を行うと送出がブロックされる．Buffer receiving では，受信 DMA は常にアクティブであり，メッセージは到着すると同時にリングバッファに格納される．割り込み処理が不要で，受信処理は直ちに行われるため，送出プロセスはブロックされることがない．

Line sending と buffer receiving は，キャッシュを有効に利用して，効率良いメッセージパッシングを実現するが，以下の問題を持っている．

- Line sending は CPU が起動するので，メッセージ送信中は計算処理ができない．
- ネットワークが輻輳した時には，line sending はブロックされ，トラップが生じる．これは，アプリケーションのネットワークの使用具合による．
- メッセージ送受信は，キャッシュラインサイズ単位で行われるため，アライメントが合わないメッセージの扱いを考慮しなくてはならない．
- リングバッファによる受信操作は複雑である．メッセージのサーチや，ラップアラウンドしたメッセージの取り扱いを考慮しなくてはならない．

以上の問題点が性能にどう関わるかは 5 章において検証する．

3. MSC

本章では，line sending と buffer receiving を実現するメッセージコントローラ (MSC) について述べる．MSC はキャッシュコントローラと DMA コントローラを内蔵する．**図 3** に MSC の構成を示す．

メモリ，デバイス，MSC はローカルバス (LBUS) で接続されている．CPU とキャッシュメモリは MSC に接続されている．通常動作時は，MSC はコピーバック方式のダイレクトマップのキャッシュメモリコントローラとして動作する．Wbuf はメモリやデバイスにデータを書き込む時に使用するライトバッファである．1 キャッシュラインは 4 ワード (16 バイト) である．

3.1 Line sending

Line sending の起動は，キャッシュメモリ制御コマンドと同様，普通の store 命令で行われる．アドレス 32 ビットの上位 6 ビットが制御コマンドとしてデコードされる．CPU が上位 6 ビットを line sending に指定して store を実行すると，下位 26 ビットのアドレスが指すデータが 1 キャッシュライン分キャッシュメモリからネットワークへ送出される (**図 4**：HIT)．これは，プログラムモードの転送とは異なる．プログラムモードは，レジスタを介して数回の load

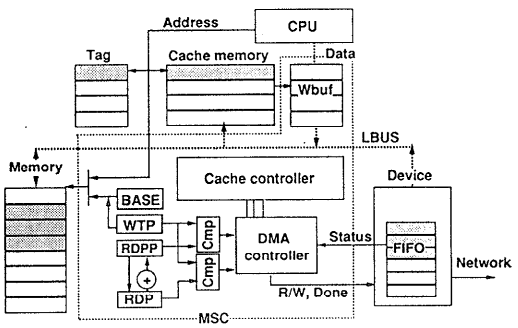


図 3 MSC の構成
Fig. 3 MSC configuration.

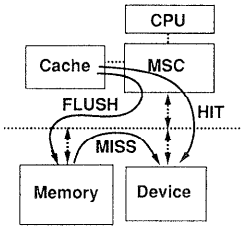


図 4 Line sending によるデータ移動
Fig. 4 Data movement during line sending.

と store を繰り返す必要がある.

CPU が line sending を起動すると MSC が，キャッシュの Tag メモリをチェックし，データがキャッシュにあるかどうかを調べる．同時に，出力デバイスのステータスを見て送信可能か調べ，LBUS が他のデバイスから使用されていないこと，DMA が同じデバイスに対して起動されていないことを確かめる．キャッシュメモリ上にある場合は，データは Wbuf に書き込まれる (HIT)．ない場合には，DMA が自動的に起動され，メモリからデバイスにデータが転送される (MISS).

CPU は，すべてのステータスがチェックされ，Wbuf にデータが書き込まれるまで，または，DMA が起動されるまでブロックされる．デバイスに書き込まれるまで待つことはない．この機構を実現するためには，デバイスはキャッシュラインサイズよりも深い FIFO を持たなければならない．なぜなら，すべてのデータが転送される保証がなければ，デッドロックを生じる可能性があるからである．MSC は，FIFO の状態を調べ十分な空きがなければ，データアクセス例外によって CPU にトラップを起こし再実行を可能とする.

Line sending は，キャッシュエントリを無効化しない．メインメモリにも書き戻さない．これは，これらのデータは再度アクセスされる可能性が高いからである．アプリケーションプログラムは，メッセージを書き替え再度送り出すことが多い.

3.2 Buffer receiving

Buffer receiving (**図 5**) は，非同期的なメッセージ受信に対応する．リングバッファのオーバフローはハードウェアで監視される．バッファフルになると，CPU に割り込みをかけて報告するか，または，メッセージの読み込みによってバッファに空き領域ができるまで受信を停止する.

データ転送単位はキャッシュラインサイズと同じ

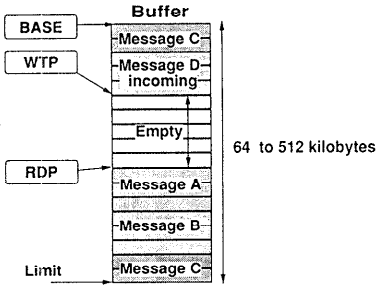


図 5 受信バッファの構成
Fig. 5 Receive buffer.

4 ワードであり，バッファは 4 ワード単位でアライメントされる．このことにより，高速な転送が保証される.

図 3 において，ベースレジスタ (BASE)，ライトポインタ (WTP) とリードポインタ (RDP) はそれぞれ，バッファアドレス，リングバッファのポインタである．WTP と RDP で受信したデータが保持されている領域を指す．WTP は次に受信されるデータが書き込まれるアドレス，RDP は CPU が現在アクセス中のアドレスを保持している．不要となった受信データを捨てるために，CPU は RDP に対して任意の値を書き込むことができる.

ネットワークからデータが到着すると，MSC はバッファがオーバランしないことを確かめる．もし WTP と RDP が同じ値であった場合，データ転送を停止し，CPU に割り込み信号が加えられる.

WTP は，バッファの先頭アドレスに初期化される．RDP はバッファの終了アドレス (Limit : base address + buffer size − 1)* を指すように初期化される．バッファのサイズは 64K から 512K バイトである.

受信されたメッセージはキャッシュメモリに直接書き込まれない．これは，メッセージの参照に先だって必要なデータがキャッシュされていて，そのデータがメッセージをキャッシュメモリに書き込むためにリプレイスされてしまうのを避けるためである.

4. ソフトウェア

ここでは，line sending と buffer receiving を用いた基本的なコミュニケーション関数である send() と recvs() の実現について述べる.

表 1 は使用するメッセージのフォーマットである．*Routing header* は，行き先を示すハードウェア

* Base address はバッファサイズのアライメントを守る.

表 1 メッセージフォーマット
Table 1 Message format.

Offset	Field	Size
0	Routing header	4 Bytes
4	Message length	4 Bytes
8	Alignment of message	4 Bytes
12	Cell IDx of sender	4 Bytes
16	Cell IDy of sender	4 Bytes
20	not used	4 Bytes
24	Message ID for trace	4 Bytes
28	Message type	4 Bytes
32	Message body	Any[a]

[a] Buffer receiving の制限により 512 KB が上限

ヘッダである．このヘッダだけがハードウェアで使用される．他はソフトウェアで使用する．

Cell IDx と *Cell IDy* は送り元のセル ID であり，それぞれ二次元平面における X, Y 座標の論理的な位置を保持する．*Alignment of message* はメッセージが始まるアドレスへのオフセットを保持する．アライメントはアドレス 32 (十進) からの変移である．これによってバイトアライメントのメッセージを扱うことができる．アライメントは 0 から 15 の値を持つ．*Message length* はメッセージのサイズを示す．

4.1 Send

send() はアドレス(p)，サイズ (size) のメッセージを送信先セル (cid) に送る関数で，以下のように実現される．

```
send (cid, type, p, size)
{
  Set message info. to the header in buffer q;
  LSEND(q);
  LSEND(q+16);
  do size/cache_line_size_1 times{
    LSEND(p);
    Increment p;
  }
  LSENDE(p);
}
```

LSEND(p) と LSENDE(p) は，line sending を起動する．前者はエンドビットを付けず，後者はエンドビットを付ける．次の命令で実現されている．このエンドビットの付加によりメッセージの終了を指定することで，複数キャッシュラインのメッセージを転送することができる．

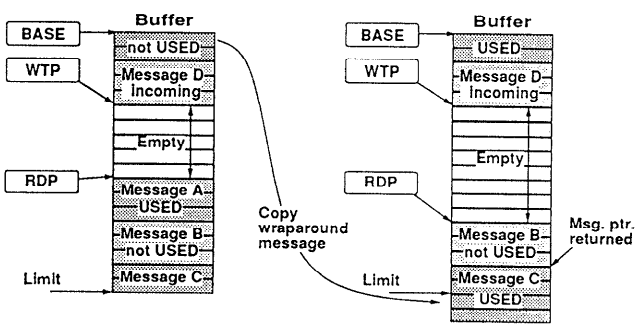
```
;; LSEND(p):
st %r0,[%r1+%r2];; %r1==p, %r2==0xec000000
;; LSENDE(p):
st %r0,[%r1+%r2];; %r1==p, %r2==0xf0000000
```

4.2 Receive

recvs() は，指定したセル (cid) からメッセージを受信する関数で，以下のように実現されている．これは，不必要なコピーをせずにアドレスを返す．メッセージがない場合には NULL を返す．

```
recvs (cid, type)
{
  Return NULL if no message has been received;
  Pick up a header from top of the ring buf. (RDP);
  forever {
    if header is marked 'USED'{
      Invalidate this message area;
      Increment RDP to skip this message;
    } else if message is found{
      Mark a header 'USED';
      Wait until the whole message is received;
      Copy data only if wraparound occurred;
      Return the message address;
    }
    if no more messages{
      Return NULL;
    }
    Pick up the next message header;
  }
}
```

図 6 はリングバッファがどのように扱われているか



(a) 初期状態 (a) Initial state
(b) 復帰状態 (b) Return state

図 6 リングバッファの扱い
Fig. 6 Ring buffer operation.

を説明するものである．バッファの状態が同図(a)の時，メッセージCを探す場合を考える．recvs() が呼ばれると，まずメッセージAを取り出すが，既に使用済みで 'USED' とマークされているので，RDP をメッセージBの先頭まで進める．Bはまだ使用されていないがCではないので，次のメッセージCを取り出す．これは探しているものであるので 'USED' とマークする．これはラップアラウンドしているので，残りの部分をバッファの最後 (limit) にコピーし，メッセージの先頭アドレス (msg. ptr.) を返す．recvs() が終了した時点のバッファの状態は同図(b)となる．

5. 性能評価

5.1 Pingpong ベンチマーク

Line sending と buffer receiving の効果を調べるために，メッセージ長を変化させた時の通信時間を通信方式をかえて AP1000 で測定した．時間は，マスタプロセッサが送出したメッセージを受けとったスレーブプロセッサがマスタにメッセージを送り返し，マスタプロセッサがそのメッセージを受けとるまでの時間を 1/2 したものである[15]．この時間は2章の overall latency に相当する．AP1000 はワームホールルーティングを用いているためプロセッサ間の距離は結果にほとんど影響しない[14]．他のプロセッサはアイドル状態であり，結果に影響を与えない．

図7は，メッセージ長に対する時間を表している．すべての測定において，受信メッセージはアドレスで与えられるため，受信においてコピーのオーバヘッドはない．Normal は従来の DMA 転送を用いた場合，

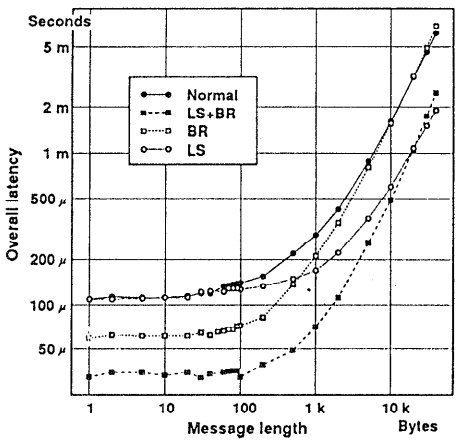


図 7 メッセージ通信遅延
Fig. 7 Message overall latency.

表 2 オーバヘッド解析 (マイクロ秒)
Table 2 Summary of overhead (microseconds).

Mode	Overall latency	10 B	10 KB
Normal	$P, S_s, N_d, S_r, D, X, S$	112	1590
LS	$P, N_d, S_r, max(LS, D), X, S$	111	597
BR	P, S_s, N_d, D, X, S	61.3	1550
LS+BR	$P, N_d, max(LS, D), X, S$	32.8	491

$S_s = A+E+C+I$, $N_d = Network\ delay$, $S_r = F+A+I+T$

BR は DMA の代わりに buffer receiving を用いた場合，LS は line sending を用いた場合，LS+BR は BR と LS 両方を用いた場合の結果である．

サイズの大きなメッセージの場合，送信時のコピーの時間が有意である．LS および LS+BR は，コピーがないので性能がよい．サイズが小さな場合，受信割り込み処理のため LS および Normal の性能は悪い．これに対し，BR は受信割り込みがないため性能がよい．LS+BR は全体にわたって良い結果である．サイズが 30 K バイト以上で BR の性能が低下するが，これはラップアラウンドしたメッセージのコピーのためである．DMA によって小さなメッセージの送信が行われた場合はセットアップ時間が，大きなメッセージではコピーの時間が性能を悪くしている．

表2は転送におけるオーバヘッドをまとめたものである．記号は2章と同じである．表2はメッセージ長が 10 バイトと 10 K バイトの場合の時間である☆．

データが小さいときコピーは無視できる．BR と LS+BR の差は送信のセットアップ時間の差であり，約 28 μs (=61.3−32.8) である．Normal と LS ではデータが小さい場合，時間は受信のセットアップ時間によって抑えられている．受信のセットアップ時間の差は約 78 μs (=111−32.8) である．

5.2 ベンチマークプログラムでの性能比較

表3はいくつかのベンチマークでの実行時間を測定したものである．LS, LS+BR には，line sending のときのメッセージがキャッシュにのっていた率，およびトラップが生じた率を（ヒット率，トラップ率）で示してある．**図8**は normal モードでの実行時間で正規化したものである．ベンチマークについて簡単に解説した後，考察を行う．

LINPACK[16]は，密行列方程式を LU 分解によって解くプログラムである．問題の大きさ n は $n \times n$ 行

☆ この測定において，BR, LS, LS+BR モードは normal モードのユーザインタフェースに合わせているため BR と LS を用いる AP 1000 の関数 xy_send, xy_recv に比べて遅い．10バイトのデータを xy_send と xy_recv を用いてやり取りした場合には，19.4 μs である．

表 3 計算時間
Table 3 Performance results.

ベンチマーク	セル台数	問題のサイズ	実行時間 (秒)			
			Normal	LS	BR	LS+BR
LINPACK	8×8	1000	8.73	7.95(90.4, 18.9)	6.65	4.83(98.0, 2.69)
	8×8	2000	35.8	33.0(95.4, 12.8)	31.4	26.2(97.4, 2.12)
	8×8	4000	197	187(96.4, 8.38)	186	170(97.0, 1.72)
	16×16	1000	7.44	6.51(95.8, 25.1)	5.12	3.05(98.9, 2.43)
	16×16	2000	21.0	18.5(88.7, 18.9)	15.8	10.8(98.3, 2.30)
	16×16	4000	80.4	71.7(95.1, 12.6)	69.2	54.9(97.7, 1.66)
SCG	16×16	100	1.65	1.70(99.9, 28.5)	0.93	0.61(100, 11.8)
	16×16	200	4.22	4.21(99.9, 14.5)	2.69	1.98(100, 6.48)
	16×16	400	14.3	13.7(99.9, 5.48)	11.58	9.97(99.9, 1.41)
	16×32	100	1.71	1.76(99.8, 27.8)	0.93	0.58(100, 15.3)
	16×32	200	3.89	3.98(99.9, 24.5)	2.29	1.63(100, 16.4)
	16×32	400	12.2	12.4(82.9, 13.3)	9.34	7.51(65.5, 9.62)
SLALOM	4×4	1067	64.9	65.4(99.2, 30.5)	61.9	59.8(99.3, 6.17)
	8×8	1666	66.9	67.7(99.2, 27.8)	63.2	60.1(99.5, 10.22)
	16×16	2205	68.3	70.2(99.2, 28.5)	64.7	60.2(99.2, 11.2)

(,) 内の数字はそれぞれヒット率（左側），トラップ率（右側）を示す．

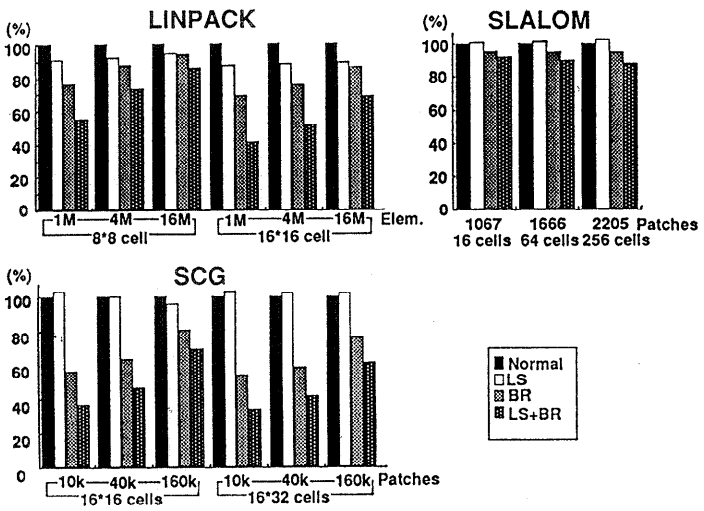


図 8 Normal によって正規化した実行時間
Fig. 8 Normalized performance results.

列を表す．測定はガウス消去と後退代入部分を測定した．セル内のみの計算時間である．

SCG は，2次元のポアソン方程式を SCG 法[17]で解いたものである．問題の大きさ n は分割されるメッシュの数 $n\times n$ を表しており，$n^2\times n^2$ の疎行列の解法に相当する．セル内のみの計算時間で，初期マトリックスの生成等の時間は含まない．

SLALOM[18]は直方体内部の光の分布をラジオシティ法で解くものである．問題の大きさ n は，LS+BR を用いて計算を行った場合に 60 秒を要するパッチ数である．初期値のファイルからの読み出しから結果の書き出しまで，ホストとセルのすべての経過時間を測定した．

表 4 に通信時間と計算時間の解析結果を示す．この結果は1セルあたりの平均通信回数と，LS+BR における実行時間に対する通信時間の割合の平均である．SLALOM についてはホスト間の通信を (,) 内に内数として示してある．

プロセッサの数が増えても通信回数は問題の大きさが同じであればほぼ同じである．これに対し，処理する問題の大きさが同じであれば，1プロセッサあたりの計算時間はプロセッサの数が増えれば小さくなる．その結果，通信時間が全体に占める割合が増加し，通信を高速化した効果が顕著となる．

SLALOM の結果にあまり変化が見られないのは，SLALOM はホストとセルのすべての経過時間を測定した結果であるため，セル間の通信時間の割合（16, 64, 256 セルでそれぞれ 12, 22, 27%）が比較的小さいことによる．

SLALOM や SCG の一部など，LS のみではかえ

表 4 ベンチマークプログラムの通信と計算の解析
Table 4 Statistics of benchmark program.

ベンチマーク	セル台数	問題サイズ	平均通信回数	平均通信時間 (%)
LINPACK	64	1000	7770	52
		2000	15491	33
		4000	31089	17
	256	1000	8415	72
		2000	16768	60
		4000	33510	40
SCG	256	100	2990	70
		200	6011	62
		400	10271	42
	512	100	2787	75
		200	6070	69
		400	12171	58
SLALOM	16	1067	10876(266)	27(15)
	64	1666	14357(208)	39(17)
	256	2205	16608(137)	60(33)

(,) 内はホスト間の通信回数と時間の内数

って遅くなってしまうプログラムもある．これらのプログラムでも BR を併用することによって処理時間が短くなることから，割り込みなどの受信オーバヘッドを削減すると送信オーバヘッドの削減が効果を示すことがわかる．これは，LS と LS+BR でトラップ率を比較して BR によってトラップ率が下がることからも理解できる．

2.2 節で取り上げた問題，line sending の CPU 時間の消費，トラップの影響，リングバッファの処理の複雑さ等は得られる効果に比べ無視できる．Line sending のメッセージのほとんどがキャッシュにのっていることもわかる．SCG ベンチマークでは，16×32 セル構成，問題サイズ 400 のヒット率は 65.5% と低い．しかし，この場合でも送信のオーバヘッドを削減した効果が現れていることに注意する．

一般に，同じ大きさの問題を高速に解くためには，多くのプロセッサを使用する必要があり，その場合，台数効果を十分得るためには通信時間の増加を極力抑える必要がある．台数を増やすと一回あたりの通信データ量は減少し，通信時間のうち一定時間を占めるセットアップ時間が優位となる．セットアップ時間の短縮を実現した本方式により，ベンチマークプログラムが高速化されている．

6. まとめ

分散メモリ型並列計算機のプロセッサ間通信方式において，line sending と buffer receiving という方式を提案した．いくつかのベンチマークを実行して，この低レイテンシ通信方式がアプリケーションを高速化することを示した．メッセージ送信のオーバヘッドを小さくすることは，受信オーバヘッドが小さい場合に効果が大きいことがわかった．

メッセージパッシングの機能を実現するにあたり，キャッシュメモリの存在を考慮することは非常に大切である．多くのプログラムにおいて送るべきメッセージはキャッシュメモリ上に存在しているため，line sending はメッセージ送信をキャッシュメモリから直接，遅延なく送り出すことができる．メッセージがキャッシュ上に存在しない場合にも，自動的にメモリからの送出が起動されるため，ペナルティは生じない．

Buffer receiving はリングバッファの構成をとることにより，受信を常に可能としている．これにより，受信側プロセッサに到着したメッセージは速やかにメモリに格納される．送信側プロセッサをブロックすることがないため，送信側のトラップ率を下げることができ，送信オーバヘッドが小さい送信方式を活かすことができる．

本稿で述べた line sending と buffer receiving はシングルタスクのプログラムを仮定している．マルチユーザ，マルチタスクに対する考慮がされていない．これに対しては，メッセージにタスク ID を付加し，受信時には，その ID に従いタスクごとのバッファにメッセージを格納することにより対処できる．送信時には送信タスクごとに異なったアドレスに対して line sending を起動させ，同時に送信可能なアドレスを 1 つに制御することにより，複数のタスクのメッセージ

の混在を避けることができる．送信可能でないタスクが，line sending を起動した場合には，バッファがフルの時に生じるトラップと同等なトラップを起こさせタスクスイッチさせる．

Buffer receiving は，メッセージサーチに時間がかかる．これは，現在の実現が一番古いメッセージから順に探すためである．メッセージの到着がうまくスケジューリングされていれば，はじめに到着したメッセージが最初に読み出されるため，オーバヘッドは小さい．しかし，メッセージが読み出されずにリングバッファに残ってしまう場合，サーチのオーバヘッドは大きくなる．リングバッファのオーバフローも引き起こす可能性がある．本稿で調べたプログラムでは問題にならなかったが，メッセージのスケジューリングや，サーチの方法について検討する必要がある．

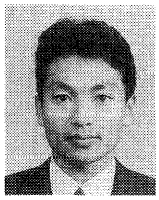
メッセージがラップアラウンドした場合には，バッファの最後にコピーされる．これは，メモリ効率とコピーオーバヘッドの点で良くない．読み終ったメッセージは，ソフトウェアでインバリデートしなくてはならない．Line sending は，ネットワークが輻輳した場合に，ブロックされ，データアクセス例外によってトラップが生じる．これらのコストは大きくないが，今後検討すべき課題である．

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，第二研究室池波室長，佐藤主任研究員ならびに研究室の同僚の諸氏に感謝いたします．

参 考 文 献

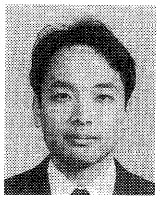
1) Seitz, C. L.: The Cosmic Cube, *Communications of the ACM*, Vol. 28, No. 1, pp. 22-33 (1985).
2) Tuazon, J., Peterson, J. and Pniel, M.: Mark IIIfp Hypercube Concurrent Processor Architecture, *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications*, pp. 71-80 (Jan. 1988).
3) Dally, W. J. et al.: The J-Machine: A Fine-Grain Concurrent Computer, *Information Processing 89*, Elsevier North Holland (1989).
4) Arlauskas, R.: iPSC/2 System: A Second Generation Hypercube, *Third Conf. on Hypercube Concurrent Computers and Applications*, pp. 33-36, ACM (1988).
5) Dally, W. J.: Virtual-Channel Flow Control, *Proc. of the 17th Int'l Symp. on Computer Architecture*, pp. 60-68 (May 1990).
6) Kermani, P. and Kleinrock, L.: Virtual Cut Through: A New Computer Communication Switching Technique, *Computer Networks*, pp. 267-286 (1979).
7) Dally, W. J. and Seitz, C. L.: The Torus Routing Chip, *Distributed Computing*, pp. 187-196 (1986).
8) Dally, W. J. et al.: Message-Driven Processor Architecture Version 11, *MIT Concurrent VLSI Architecture Memo 14*, A. I. Memo No. 1069 (Aug. 1988).
9) Nikhil, R. S. and Papadopoulos, G. M.: *T: a Killer Micro for a Brave New World, *Technical Memorandum of MIT*, Computation Structures Group Memo 325.
10) Hsu, J. M. and Banerjee, P.: A Message Passing Coprocessor for Distributed Memory Multicomputers, *Proc. of Super Computing '90*, pp. 720-729 (1990).
11) Borkar, S. et al.: Supporting Systolic and Memory Communication in iWarp, *Proc. of the 17th Int'l Symp. on Computer Architecture*, pp. 70-81 (May 1990).
12) Horie, T., Ishihata, H., Shimizu, T. and Ikesaka, M.: AP1000 Architecture and Performance of LU Decomposition, *Proc. of 1991 Int'l Conf. on Parallel Processing*, pp. 634-635 (Aug. 1991).
13) Ishihata, H., Horie, T., Inano, S., Shimizu, T. and Kato, S.: An Architecture of Highly Parallel Computer AP1000, *IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing*, pp. 13-16 (May 1991).
14) Ishihata, H., Horie, T., Inano, S., Shimizu, T. Kato, S. and Ikesaka, M.: Third Generation Message Passing Computer AP1000, *International Symposium on Supercomputing*, pp. 46-55 (Nov. 1991).
15) Hockney, R.: Performance Parameters and Benchmarking of Supercomputers, *Parallel Computing*, Vol. 117, 10 & 11, pp. 1111-1130 (Dec. 1991).
16) Dongarra, J. J.: Performance of Various Computers Using Standard Linear Equations Software, *Technical Report*, University of Tennessee (May 8, 1991).
17) 速水 謙：SCG 法による拡散方程式の解法，コンピュートロール 26，コロナ社 (1989).
18) Gustafson, J. et al.: SLALOM UPDATE, *Supercomputing Review*, pp. 56-61 (March 1991).

(平成4年9月14日受付)
(平成5年1月18日採録)



清水 俊幸

1964 年生．1986 年東京工業大学工学部電子物理工学科卒業．1988 年同大学院理工学研究科情報工学専攻修士課程修了．同年(株)富士通研究所入社，現在に至る．計算機アーキテクチャ，並列処理システムなどに興味を持つ．電子情報通信学会会員．



堀江 健志（正会員）

1962 年生．1984 年東京大学工学部電気工学科卒業．1986 年同大学院修士課程修了．同年(株)富士通研究所入社，現在に至る．並列計算機に関する研究開発に従事．

石畑 宏明

1957 年生．1980 年早稲田大学理工学部電子通信学科卒業．同年(株)富士通研究所入社，現在に至る．並列計算機に関する研究開発に従事．元岡賞受賞．電子情報通信学会会員．