計算機アーキテクチャ 95−12
(1992. 8. 19)

AP1000の性能評価
−メッセージハンドリング, 放送, バリア同期の効果−

清水 俊幸, 堀江 健志, 石畑 宏明
(株) 富士通研究所
{*toshi, lions, hata*}*@flab.fujitsu.co.jp*

本論文では,AP1000が採用した, メッセージハンドリング機構, 放送機構, およびバリア同期機構について評価検討する. 分散メモリ (メッセージパッシング) 型並列計算機において, 通信時間を短かく抑えることは, 効率の良い並列実行のために欠かせない. プロセッサ台数を増やして性能を上げていくためには, 多くのプロセッサと効率良く通信する機能 (放送, データの分散 / 収集) が大切になる. プロセッサ間の同期も高速にとる必要がある. これらの要求を満たすために, AP1000に実装された機構について, ユーザが利用できるレベルから評価し, それが実際のアプリケーションの性能にもたらす効果について検討する.

Performance Evaluation of the AP1000
Effects of message handling, broadcast, and barrier synchronization on benchmark performance

Toshiyuki Shimizu Takeshi Horie Hiroaki Ishihata
Fujitsu Laboratories Ltd.
1015 Kamikodanaka, Nakaharaku
Kawasaki 211 Japan
{*toshi, lions, hata*}*@flab.fujitsu.co.jp*

This paper presents a performance of evaluation of the functions embedded on the AP1000 for fast execution of parallel programs. The functions include message handling, broadcasting, the barrier synchronization, and gather/scatter mechanisms. The message handling mechanism supports low-latency communication between processing elements; broadcasting and gather/scatter mechanisms support efficient communication among many processing elements; and barrier synchronization mechanism allows all processing elements to be synchronized. We benchmark these functions at the user-library level, and we discuss the impact of library function speed on the performance of large standard benchmarks such as Linpack, Slalom, and SCG.

1 はじめに

メッセージパッシング型の並列計算機においては, 並列実行に伴いメッセージのやり取りが必要であり, メッセージ通信性能がそのシステムの性能に大きく影響する. プロセッサ台数を増やすことにより, 演算性能を高めることを考えれば, 通信回数の増加, メッセージサイズの微細化が生じ, 通信のセットアップ時間が性能に大きく影響を及ぼす. 従来のシステム [1] などにおいては, 通信セットアップ時間が大きいこと, 多くのプロセッサを扱うことに対する配慮が不十分であることから, 巨大な問題を適用する場合を除き, プロセッサ台数を増やした場合の性能向上はあまり高くなかった.

AP1000 では, 多くのアプリケーションプログラムを効率的に実行させるために, アーキテクチャに, 独自のメッセージハンドリング機構, 放送機構, バリア同期機構, 分散収集機構を採用している [2, 3, 4, 5]. メッセージハンドリング機構は, キャッシュメモリから直接メッセージを送出し, リングバッファに直接受信することを特徴としている. 放送機構は, 二次元トーラストポロジを持つプロセッシングエレメント (セル) 間の通信ネットワーク上で, 領域を指定した放送を可能としている. バリア同期機構は, ツリー構造を持つ専用ネットワークを使用して, 短時間でセル間の同期を実現する. 分散収集機構は, 一回の操作によりセルに分散されたデータをホスト計算機に収集したり, ホストに保持されているデータをセルに分散させることが可能である.

これらの機構がアプリケーションの実行に及ぼす効果について評価を行なう. 2 節では AP1000 が採用している独自の機構について簡単に述べ, その性能についてまとめる. この測定にはアプリケーションを開発するユーザが利用するライブラリ (ユーザレベルライブラリ) を用いる. 3 節においてこれらの機構がアプリケーション性能に与える影響について調べ考察する.

2 AP1000 アーキテクチャ

AP1000 は, 図 1に示すように 3 つの独立なネットワーク (S-net, B-net, T-net) によってプロセッシングエレメント (セル) が接続された分散メモリ型の並列計算機である.

セル間の通信は主に T-net を介して行なわれる. T-net は, 二次元のトーラスのトポロジを持ち, ルーティング方式としてワームホールルーティングを用いている. T-net は,RTC[4] によって構成される. T-net を介してセルは,X 方向,Y 方向, および XY 方向に範囲指定の放送が可能である.

メッセージの送信 / 受信は, ラインセンド / バッファレシーブと [5] 呼ぶ機構を用いて実現される[1]. この機構は, キャッシュメモリ上に送信データが存在する場合には, キャッシュメモリから直接データをデバイスに送出し, 存在しない場合には, メインメモリから転送を行なう. ラインセンドはキャッシュメモリのフラッシュと同等の操作によって起動され, 送信オーバーヘッドが極めて小さい. バッファレシーブは, ネットワークから到着したメッセージをメインメモリ上に確保したリングバッファに直接受信する. このため, 受信時に必要であった割り込み処理等のセットアップ時間が不要である.

[1]この機構を使用した通信を XY 通信と呼ぶ

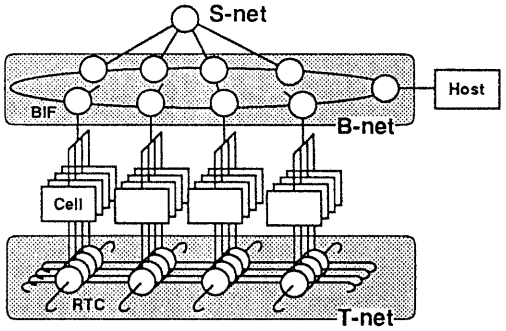

図 1: AP1000 の構成

セル間の同期は,S-net を介して行なわれる. S-net はツリー状のネットワークで, 時分割で同期要求をツリーのルートに対して送りだし, ルートにおいて AND された同期要求が, ツリーを辿って再分配される. このため, セル台数によらない高速なバリア同期が実現される.

分散収集機構は,B-net を構成する BIF の機能として実現されている. ホストからデータをセルに分散させる場合には, ホストは B-net に全てのデータを書き出す操作を 1 回起動する. B-net に現れたデータは, セルそれぞれで, フィルタリングされ, 必要な部分のデータのみが, セルに取り込まれる. セルが保持しているデータをホストに収集するには, 各セルの BIF が自セルがデータを送出する番であるかどうか B-net を監視することによってカウントしながら, データを収集していく. これらの機構により, 一回の操作でセルに分散されたデータをホスト計算機に収集したり, ホストに保持されているデータをセルに分散させることができる.

本節では, 以上の機能の基本的な性能の測定を行なう. 測定は全てユーザレベルの関数を用いて行なった. 使用した AP1000 は, 物理構成に等しいコンフィグレーション (2 次元トーラス) で動作させた.

2.1 メッセージハンドリング

メッセージハンドリング機構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 Pingpong ベンチマーク [7] によって測定した. 時間は, マスタセルが送出し, そのメッセージを受け取ったスレーブセルがマスタにメッセージを送り返し, マスタセルがそのメッ

セージを受け取るまでの時間を 1/2 したものである．ネットワークは T-net を使用した．他のセルはアイドルであり，結果に影響を与えない．

図 2 は，メッセージ長に対する時間を表している．従来の DMA 転送を用いる関数 l_asend(), crecv() の結果 (Normal) も合わせて示す．XY1, XY24 はそれぞれセル間の距離 1 と 24 の場合の XY 通信の性能をプロットしている．x バイトの通信に対して，消費される時間は，それぞれ，$T_{XY1} = 0.18 \times x + 19\mu s, T_{XY24} = 0.18 \times x + 24\mu s$ となっている．セル間の距離の影響は，設計値の距離 1 あたり 160 ナノ秒に一致することがわかる．XY 通信は種々のオーバーヘッドを削減しているため，セットアップが小さい．スループットは約 5.5MB/s である．スループットが Normal に比べ低いのは xy_recv() がユーザが指定したバッファに，受信メッセージをコピーしているためである[2]．xy_recv() の代わりに受信メッセージをコピーしないでポインタを返す関数 xy_crecv() を使用した場合 (CXY1) には，$T_{CXY1} = 0.05 \times x + 10\mu s$ となりスループットが 20MB/s になる．l_asend(), crecv() のインタフェースを使い，XY 通信の機構を利用するオプション -LSEND を与えた場合には，$T_{LS1} = 0.04 \times x + 31\mu s$ となる．パラメータ変換のオーバヘッドがセットアップ時間を大きくしている．

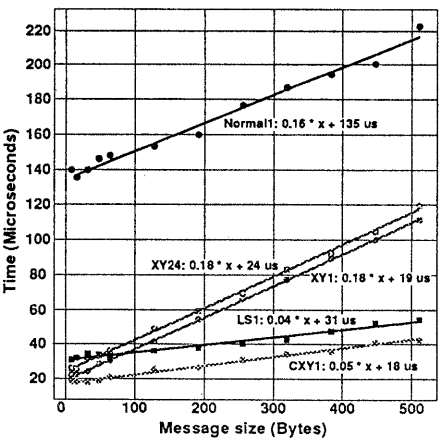

図 2: Pingpong performance

2.2 放送

放送機構は，T-net にインプリメントされている．X, Y および XY 方向の放送が可能である．ここでは，Y 方向の 8, および 16 個のセルに Y のインデックスが 0 であるセルから Y 方向に放送を行なった場合の性能を測定した．X 方向，XY 方向の速度も同じである．図 3にその結果を示す．y_brd16 は 256(16 × 16) セルの構成で Y 方向に放送したもの．y_brd8 は 64(8 × 8) 構成の場合，ybrd16, ybrd8 はそれぞれ，一対一通信を使用して 2 進木の通信によって実現した場合である．性能は，それぞれ x バイトのメッセージを送った場合に $T_{y\_brd8} = 0.11 \times x + 2.8\mu s, T_{y\_brd16} = 0.13 \times x + 2.7\mu s, T_{ybrd8} = 0.20 \times x + 27\mu s, T_{ybrd16} = 0.27 \times x + 36\mu s$, となる．前節の結果に比べ，セットアップは小さく，スループットが高くなっている．これは，本測定では送信セルと受信セルが決まっているため，複数 (1000) 回の試行による平均をとると，送信と受信がパイプラインされた影響がでるためである．

y_brd, ybrd を比較すれば，放送機構を用いることにより，セットアップは約 $\frac{1}{10}$，スループットは約 2 倍になっていることがわかる．

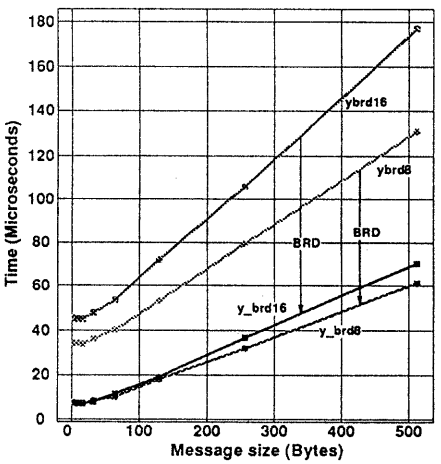

図 3: Broadcast function

2.3 グローバル関数

グローバル関数, 例えば xy_damax() や xy_dsum() は，全セルから同時に呼びだし，2 進木の通信をしながら，システム全体で，最大値や総和を計算し，最終的に全てのセルが同じ結果を保持する関数である．xy_dsum() は全セルが持つ要素 (倍精度実数:8 バイト) の総和を求める関数，xy_damax() は全セルの中での最大値を求める関数である．

これらは，表 1で示す 16 バイトの短いフォーマットのメッセージを用いて実現されている．*Index of DATA* に，どのセルが選択されたデータを持っていたかを示す．

このフォーマットは 1 キャッシュラインに収まるので効率がよい．*Routing header* の下位 9 ビットをグローバル関数の種類を指定するために割り当ててある．表 2

[2]Normal は，送信データをコピーするが，xy_recv() が行なうメインメモリからの読み出し (キャッシュミスが起きる) に比べ書き出しは高速である

表 1: グローバル関数用メッセージのフォーマット

Offset	Field	Size
0	Routing header	4 Bytes
4	Index of DATA	4 Bytes
8	DATA (float, int, double)	8 Bytes

にグローバル関数の速度を示す.

表 2: グローバル関数の性能 (マイクロ秒)

Cells	4x4	8x8	16x16
x_damax/y_damax	30	40	52
xy_damax	58	78	104
x_dsum/y_dsum	23	30	39
xy_dsum	44	59	79

xy_dsum() の方が xy_damax() に比べ僅かに速い. これは, xy_damax() では index の計算等, 処理内容が多いためである.

xy_dsum() などは 1 つの要素に対して結果を求める関数であるが, アプリケーションによっては, ベクトルとしてその要素毎の総和や, 最大値を求めることがある. ベクトルの要素毎の総和をとる場合の性能を測定した結果が, 図 4である.

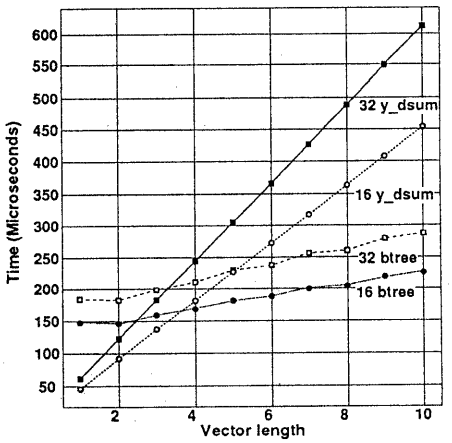

図 4: Binary tree operation vs. y_dsum

使用したプログラムを図 5,6に示す. 2 つのプログラムは共にベクトル毎に y 方向の総和をとる. それぞれ y_dsum, btree に対応する. 32,16 は Y 方向のセル台数 (ncely) である. xy_send(rx,ry,type,msg,size) は, 二次元セル id(cidx,cidy) のセルが (cidx+rx,cidy+ry) のセルにメッセージ (msg) を送信する関数である.

図 4において, ベクトル長が 3 ～ 4 のところで性能の逆転がある. y_dsum() が, 1 回の実行が高速なのは,1 回当たりのメッセージ長をヘッダを含め 16 バイトに収めていることと, 関数呼び出しが 1 回であるためオーバヘッドが小さいことによる. xy_send() では, データに 32 バイトのヘッダを付加して通信するため, データ転送量はベクトル長 4 の点で y_dsum() の場合と等しくなる. このことから, ヘッダを特別に用意し, メッセージ長を抑えた効果が主であることがわかる.

```
/*
** Global sum of each vector elements.
*/
for (i=0; i < vlen; ++i) /* vlen: vector length */
   /* sum of y-direction */
   y_dsum(results[i],&results[i]);
```

図 5: Binary tree sum

```
/*
** Global sum of each vector elements.
**   Binary tree sum and broadcast
*/
length = vlen * sizeof(double);
for (i = 1; i < ncely; i += i) {
   if (cidy & i) {
      k = xy_send(0, -i, MSGTYP, results, length);
      break;
   }
   else if (i+cidy < ncely) {
      k = xy_recvs(0, i, MSGTYP, tmp, length);
      qadd_(tmp, results, vlen);
      /* vector add: results += tmp */
   }
}
y_brd(cidx,results,length); /* broadcast */
```

図 6: Binary tree sum

2.4 分散収集

分散収集機能は,B-net にインプリメントされている. ホストから各セルへデータを 1 回の操作で分割保持させたり, セルに分割保持されているデータをホストに収集することができる.

ホストから, セルに分割保持させる時の性能 (Scat), セルから, ホストに収集する時の性能 (Gathc,Gathf) を, 64 セル, 512 セルのシステムで測定した結果を図 7に示す. 横軸にホストが収集する, または分配する総データ量. 縦軸に必要な時間を示している.

Poll64 は,64 セルシステムで収集機構を用いない場合の性能である. 512 セルの場合は, 時間が長いため, 図

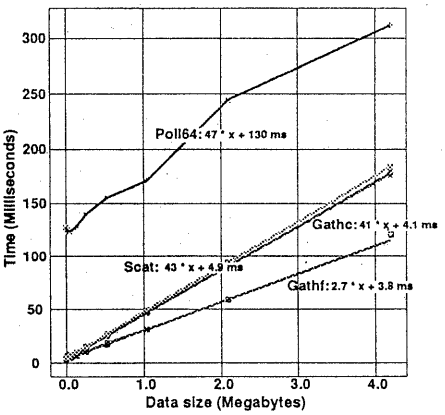

図 7: Scatter and gather performance

7には表示できない. Gath, Scat の性能はセル台数に影響されない. 性能は Scat では, ホストインタフェースのメモリ読みだし速度によって制限を受ける. Gathf, Gathc はそれぞれ収集するデータの要素が小さな場合 (各セルが 1 度に 4 バイトのデータを順番に送出する), 大きな場合 (データサイズをセル数で除した量のデータを各セル一回送出する) である.1 セルが一度に出すデータのサイズが大きな場合, セルごとに順番に大きなデータブロックが読み出されるため, セルメモリの読みだし速度で性能が抑えられる. 小さい場合, 各セルが出すべきデータはセルの BIF の FIFO で待たされている. これらのデータは B-net 上でマージされ, ホストに送り出される. セル台数を p とすると p-way のインターリーブアクセスと同等の動作となり, 高いスループットが得られる. この時のスループットは 38MB/s であり, ホストインタフェースのメモリ書き込み速度の理論値, 40MB/s に近い.

ホスト側に x バイトのデータを収集する場合に必要な時間は, $T_{Gathf} = 0.027 \times x + 3,800\mu s$, $T_{Gathc} = 0.041 \times x + 4,200\mu s$, $T_{Scat} = 0.043 \times x + 4,900\mu s$, $T_{Poll64} = 0.047 \times x + 130,000\mu s$, $T_{Poll512} = 0.16 \times x + 1,900,000\mu s$, となる. $T_{Poll512}$ は,512 セルで収集を行なった場合である. 分散収集を使用しない場合には, 特にセル台数が増加した場合に時間がかかることがわかる. T_{Poll} は,1 セルが 1 度しかホストに送信を行なわないが, 例えば T_{Gathf} が行なっているような通信パターンを実現するには, データを受信後さらにデータの並び替えが必要になることに注意する.

ホストは Sun4/330 であり,UNIX が動作している. このため, セットアップ等のオーバヘッドは大きい. ホストが関係する操作ではその回数を減らすことが大切である.

2.5 バリア同期

同期専用のネットワーク S-net によって, 全セルでの同期および, ホストを含んだ同期が可能である.

従来メッセージパッシング型の並列計算機において, 同期は, メッセージパッシングによって確立されていたが, この方法を用いた場合, 遅いこと, メッセージがデータ通信と混じるために混乱を生じるという欠点があった.

AP1000 では, 専用のネットワークの採用により速度, メッセージの通信と同期の切りわけを解決している. 同期に必要な時間は, セル構成によらず一定で, $10.6\mu s$ である.

メッセージ通信によって同期を取るとセル台数が p の時, $T_{xovr} = 33 \times \log(p) + 6.8\mu s$ 必要である.

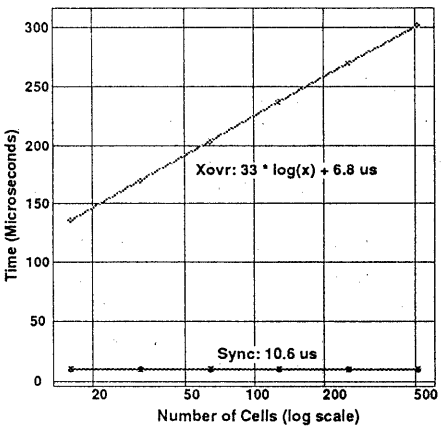

図 8: Syncronization time

3 アプリケーションの性能

いくつかのベンチマークでの性能を測定し, 通信機構, 放送機構, 分散 / 収集機構のが性能に与える影響について調べる. それぞれベンチマークについて簡単に解説した後, 考察を行なう.

3.1 LINPACK

LINPACK [9] 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 $n \times n$ 行列を表す. 測定対象は通常の LINPACK ベンチマーク [8] 同様, ガウス消去と後退代入の部分であり, マトリックスの作成, 解の書き出しは含んでいない.

LU 分解は partial pivot 付きのブロック LU 分解である. Pivot 行の選択を行なうために, グローバル関数 (xy_damax) を使用する. Pivot 行のデータを他のセルに送るためと, 前進後退代入時に T-net の X, および Y 方向のブロードキャスト (x_brd, y_brd) を使用する.

表 3: LINPACK ライブラリコール解析

Cells	Size	BRD	GLB	XY	L(%)	T(s)
64	1000	6861	1000	17		3.59
		1.3	0.2	–	32	4.67
	2000	13679	2000	22		22.5
		4.0	0.42	–	18	25.0
	4000	27493	4000	32		159
		14.0	1.2	—	9	164
256	1000	6972	1008	11		1.4
		1.1	0.2	–	49	2.5
	2000	11868	2000	17		7.0
		4.0	0.38	–	39	9.5
	4000	27721	4000	22		45
		9.0	0.91	–	20	50

表 3にライブラリの使用状態の解析結果を示す. セル台数 64, 256 においてマトリックスサイズ 1000×1000, 2000×2000, 4000×4000 を解いた場合の結果である. BRD, GLB, XY それぞれ放送, グローバル関数,XY 通信に関するデータを示す. 列の上段は, 各セルにおける呼びだし回数の平均であり, 下段はその実行に要した時間 (秒) の平均である. L は通信に費やされた実行時間の全体の時間に対する割合である. T は実行時間, 下段は, 解析のためのトレースを行なった場合の実行時間である. この値によりトレースの結果と, 実際の実行の差を見積もることができる.

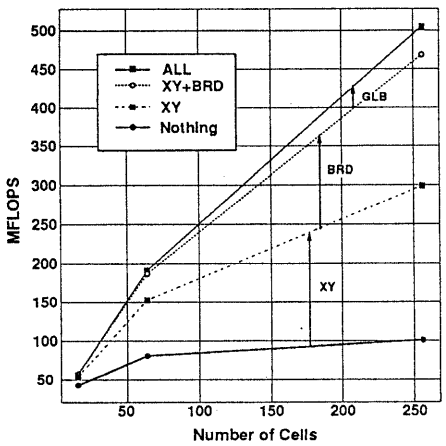

図 9: Linpack performance

図 9に 1000×1000 の解を求めた時のセル台数に対する FLOPS 値を示す. この結果において, Nothing は, 通常のメッセージ通信機構 (l_asend, crecv) を使用し, 放送およびグローバル関数は複数回の通信で実現した場合の結果である. XY は, XY 通信 (xy_send, xy_recv) を使用した場合. XY+BRD は, 放送機構を利用した結果, ALL は XY 通信, 放送機構, およびグローバル関数を利用した結果である.

XY 通信, および放送機構は性能向上に大きく寄与していることがわかる. これらの機構を持たない場合, セル台数を増やしても, 問題の大きさが等しい場合には, 性能向上が得られないことが,Nothing の結果からわかる. グローバル関数の効果はあまり顕著ではない. これは, 関数呼び出し回数が 1000 回と多くないためである. XY 通信の回数は少ないが, 放送やグローバル関数のエミュレーションにも使用するため, XY の性能は大きな比重を持っている.

3.2 SCG

SCG は, 2 次元のポアソン方程式を SCG 法 [10] で解くものである. 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.

表 4: SCG ライブラリコール解析

Cells	Size	GLB	XY	L(%)	T(s)
64	100	445	888		0.64
		0.10	0.21	34	0.93
	200	893	1784		4.1
		0.27	0.66	19	4.8
	400	1789	3576		43
		1.2	2.5	8	45
256	100	445	1332		0.25
		0.10	0.24	54	0.64
	200	893	2662		1.3
		0.32	0.69	45	2.3
	400	1789	3576		8.3
		0.86	1.8	27	9.6

本プログラムでは, 内積の総和の計算時にグローバル関数 (xy_dsum) を使用している. 表 4にライブラリの利用状態の解析結果を示す. 問題の大きさは 100×100, 200×200, 400×400 のメッシュ分割を計算した場合の結果である. 表の意味は表 3と同じである.

図 10に 200×200 のメッシュを解いた時のセル台数に対する FLOPS 値を示す. LINPACK と同様な結果が得られている. XY と XY+BRD の差は, グローバル関数のエミュレーションに放送機能を使うかどうかの差である. エミュレーションには, 図 6のプログラムと同じアルゴリズムを使用している. LINPACK と比べ, グローバル関数の効果が大きいのは, 全体時間に占めるグローバル関数の消費時間が大きいためである.

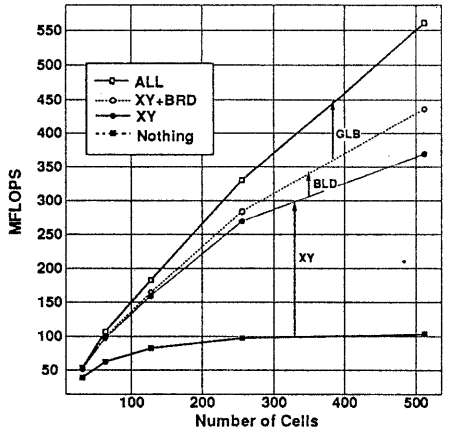

図 10: SCG performance

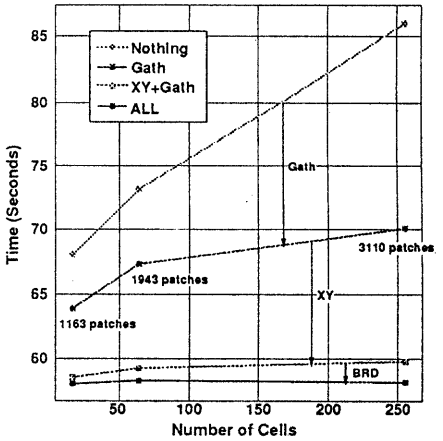

図 11: SLALOM performance

3.3 SLALOM

SLALOM は直方体内部の光の分布をラジオシティ法で解くベンチマークプログラムである [6]. 直方体の内部の壁をパッチに分割して, そのパッチの輝度 (色) を計算する. ベンチマークは, 初期データファイルの読み込みから, 計算結果ファイルの書き出しまでの全体の時間を 60 秒間に固定した場合に, 処理可能なパッチ数で評価する.

プログラムは, 大きく, 初期条件ファイル読み込み, マトリックスの作成:(3 ステップ), LDL 法による求解, 結果の書き出し, の 4 つの部分から構成されている.

表 5: SLALOM ライブラリコール解析

Cells	Size	BRD	XY	Gath	L(%)	T(s)
16	700	7800	2400	1		58
		5.8	1.8	1.2	14	63
64	1943	12387	2200	1		59
		10	2.8	1.5	22	68
256	3110	19243	1850	1		58
		16	4.1	2.1	31	72

ファイルの読み込みは, 0.2 秒程度で一定である. マトリックスの作成:(3 ステップ) の計算量は, n をパッチ数,p をプロセッサ数とした場合に $O(n^2/p)$. 全体に対して 3 割から 1 割程度の時間が消費される. LDL 法による求解の計算量は,$O(n^3/p)$. 全体に対して 7 割から 9 割程度の時間が消費される. 結果の書き出しは, セルに分割保持された解をホストに集め, ファイルに書き出す. 収集機能を用いた場合, ホストは 1 回の関数呼び出しにより, 結果の収集を行なう. 収集機能を用いない場合には $\sqrt{p}$ 回の関数呼び出しが必要である. 全体に対して 5% 程度の時間が消費される.

このベンチマークを特別な機能を用いずに, メッセージ通信だけを用いて計算した場合 (Nothing), 収集機能を用いた場合 (Gath), XY 通信を加えた場合 (XY+Gath), さらに放送機能を加えた場合 (All) にについてプロセッサ台数を 4, 16, 64, 256 台と変化させた場合の時間を測定した. パッチ数 n を,All の場合に処理時間が約 60 秒となるように選んだ.

収集機能を使用しない場合 (Nothing) には, セル台数を増やして計算可能なパッチ数を増加させると, 結果の書きだし時にホスト - セル間の通信回数, および通信量が増加するために処理時間が増加する.

XY 通信を使用すれば, パッチの数が増えることによる通信回数の増加に伴って, 性能向上が観測できる. さらに放送を使用することによって, 性能向上が見られるが, 顕著ではない. これは, 放送の回数が多くなく全体に占める消費時間が少ないためである. 他のベンチマーク同様, 放送を Normal でのエミュレーションから, XY 通信によるエミュレーションに置き換える効果が目だっている.

表 6にベンチマークの性能をまとめる.

4 まとめ

AP1000 に用意されている機構が, アプリケーションの性能向上にどれほど寄与しているかをいくつかのベンチマークにより評価した.

メッセージ送信および受信のオーバヘッドを小さくし, セットアップ時間を短くすることは, アプリケーションの高速実行に大きく寄与することがわかった. 問題が小さい場合には, 実行時間のうち, 通信に消費される時間の割合が大きいため, メッセージ送受信のセットアップ時間の削減効果が大きい. また, セル台数を増やすと通信

表 6: 性能のまとめ

ベンチマーク	セル台数	問題のサイズ	性能 MFLOPS	時間 (秒)
LINPACK	8x8	1000	190	3.51
	8x8	2000	237	22.5
	8x8	4000	268	159
	16x16	1000	505	1.36
	16x16	2000	755	7.07
	16x16	4000	951	44.9
SCG	16x16	100	212	0.251
	16x16	200	341	1.26
	16x16	400	412	8.32
	16x32	100	267	0.200
	16x32	200	536	0.799
	16x32	400	678	5.06
SLALOM	4x4	1067	28.3	58.1
	8x8	1666	123	58.7
	16x16	2205	492	58.2

回数が増加し, メッセージサイズが小さくなるため, セットアップ時間が性能を大きく左右する. 放送機構も通信回数を減らすのと同等の効果がある. このため, 実行時間の短縮に貢献する.

巨大な実際の問題を解く場合, セットアップ, 結果の書き出しにおいて, ホストとセルの通信性能は大変重要である. これらは AP1000 では, B-net の分散 / 収集機能によって高速に実行できる. しかし, I/O によるシリアライズによって, その速度は B-net のバンド幅 (50MB/s), ホストインタフェースのメモリバンド幅 (ホスト → セル: 25MB/s, セル → ホスト:40MB/s) に抑えられる. AP1000 のオプション機能を利用してセルそれぞれに I/O を付加することにより I/O 処理を並列に行なえば, これらのボトルネックが改善される可能性がある.

同期機構により, セル台数によらず高速に同期をとることができる. 今回評価したアプリケーションでは, 実際に明示的な同期をとっていないため, アプリケーションの性能に与える効果を評価できていない. しかし, メッセージ通信ネットワークと分離された同期機構は, プログラムの開発段階において, 問題の切りわけが容易となるだけでも意義が大きい. 実際, プログラムの開発時には, 多くの同期が使用されるため, 高速に同期がとれることも重要である.

AP1000 でサポートしたセットアップ時間の小さいメッセージハンドリング方式, 放送機構, 分散 / 収集機構は, アプリケーションの高速実行, 特にセル台数を増やした場合のスケーラビリティを保つのに効果を示すことがわかった. これらの機構を持たない場合, セル数を増やしても加速されない場合が生じる.

本稿で扱ったプログラムは, どちらかといえば, 負荷分散, 通信時間と計算時間の割合などの面で, 性質の良いものであった. ベンチマーク性能は, 問題のサイズを大きくすることによって, 効率が高くなり, FLOPS 値も高くなる. しかし, 広い範囲の問題において効率的な実行を可能とするためには, 種々のオーバーヘッドを削減する必要がある. それらを評価するために,(通信オーバーヘッドが隠れてしまうような) 巨大な問題でなく, ある程度小さな問題について検討した.

その結果, 問題のサイズが小さい場合にも高い性能が得られたことから, 多くのアプリケーションで高い性能が得られると予想される.

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, アーキテクチャ研究部 白石部長, 並列処理研究センター第二研究室 佐藤室長, ならびに研究室の同僚の諸氏に感謝致します.

参考文献

[1] Ramune Arlauskas, "iPSC/2 System: A Second Generation Hypercube," in *Third Conf. on Hypercube Concurrent Computers and Applications*, pp. 33–36, ACM, 1988.

[2] H. Ishihata, T. Horie, S. Inano, T. Shimizu, and S. Kato, "An Architecture of Highly Parallel Computer AP1000," in *IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing*, May 1991, pp.13–16.

[3] H. Ishihata, T. Horie, S. Inano, T. Shimizu, S. Kato, and M. Ikesaka, "Third generation message passing computer AP1000," in *International Symposium on Supercomputing*, Nov. 1991, pp. 46–55.

[4] T. Horie, H. Ishihata, T. Shimizu, and M. Ikesaka, "AP1000 Architecture and Performance of LU Decomposition," in *Proc. of 1991 Int'l Conf. on Parallel Processing*, August 1991, pp.634–635.

[5] T. Shimizu, H. Ishihata, T. Horie, "Low-latency message communication support for the AP1000," in *The 19th Annual Int'l Symp. on Computer Architecture*, May 1992, pp.288–297.

[6] J. Gustafson et al., "SLALOM UPDATE," *Supercomputing Review*, March 1991, pp.56–61.

[7] R. Hockney, "Performance parameters and benchmarking of supercomputers," *Parallel computing*, Vol 17, 10&11, Dec. 1991, pp.1111–1130.

[8] J. J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software," *Technical Report*, University of Tennessee, May 8, 1991.

[9] R. P. Brent, "The LINPACK benchmark on the AP1000," *In Fourth Symposium on the Frontiers of Massively Parallel Computation*, Oct. 1992.

[10] 速水 原田, "対角法スケーリングを施した共役勾配法のベクトル計算機における有効性について" 情処論文誌, Vol. 30 No.11, pp.1364–1375, 1989.