計算機アーキテクチャ 105−13
ハイパフォーマンス
コンピューティング 50−13
（1994. 3. 11）

PVM/AP1000の実現および通信性能評価

岩下 茂信† 國貞 勝弘‡ 村上和彰‡

†九州大学 工学部 情報工学科

‡九州大学 大学院総合理工学研究科 情報システム学専攻
〒816 春日市春日公園 6−1

E-mail: {iwashita, kunisada, murakami}@is.kyushu-u.ac.jp

逐次型言語をベースとして並列計算機用のアプリケーションプログラムを作る方法には大きく分けて自動並列化コンパイラを用いる方法と，並列プログラミングライブラリを使用してユーザが並列化を行う方法の2つがあり，現在は並列プログラミングライブラリを用いる方法が一般的であると考えられる．並列プログラミングライブラリを用いてワークステーションクラスタと呼ばれる仮想的な並列計算機を構成することができる．この様な目的のためのライブラリであるPVMをAP1000上に実装した．本稿では，このPVM/AP1000の実装の方針を述べ，PVM/AP1000の通信性能について述べる．

Implementation and Evaluation of Communication Performances of a PVM/AP1000

Shigenobu Iwashita† Katsuhiro Kunisada‡ Kazuaki Murakami‡

†Department of Computer Science and Communication Engineering
Kyushu University

‡Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
6−1 Kasuga-koen, Kasuga-shi, Fukuoka 816 Japan

E-mail: {iwashita, kunisada, murakami}@is.kyushu-u.ac.jp

There are two programming paradigms for parallel applications, parallelizing Compiler and parallel programming library. Parallel programming library is one solution of developing parallel programs. Using a parallel programming library, a virtual parallel machine can be made with Workstations. We implemented PVM, one of parallel programming librarys on AP1000. In this paper, we describe the implementation and communication performance of PVM/AP1000.

1 はじめに

並列プログラミングライブラリは，プロセッサ間通信や同期など，並列処理に必要な機能を用意した関数ライブラリである．ユーザはプログラム中で陽にこれらの関数を呼び出すことにより，並列実行されるプログラムを作成する．はじめから並列処理を意識してプログラムを書くことになるため，プログラムの実行時の並列度が高くなることが期待される．一般的な並列計算機では独自に並列プログラミングライブラリが用意され，アプリケーションがライブラリ関数を呼び出すことにより並列処理を行う方式がとられていることが多い．

並列プログラミングライブラリを用いて，ワークステーションクラスタと呼ばれる仮想並列計算機を実現することができる．ワークステーションクラスタとは，独立したワークステーションをネットワークで接続し，1台のワークステーションを1つのプロセッシングエレメントと見ることで，見かけ上1台の並列計算機として使用するというものである [6]．並列プログラミングライブラリは，各ワークステーション上で実行される複数のプロセス間の通信を実現する．現在用いられているワークステーションクラスタを構成するための主な並列ライブラリとして，Ork Ridge National Laboratory(ORNL) の PVM, Parasoft 社の Express, Argonne National Laboratory の P4, Battelle Pacific Northwest Laboratory の TCGMSG などを挙げることができる．

今回，我々は PVM をメッセージパッシング型マルチコンピュータ AP1000 に実装した．実装の目的は次の2つである．

- PVM を使用した AP1000 用プログラムの作成:
  AP1000 用のプログラムは AP1000 用の通信ライブラリ関数を用いて作成するが，AP1000 用通信ライブラリを用いて作成されたプログラムは，AP1000 上でしか実行することができず汎用性がない．そこで，PVM という一種の標準化されたライブラリを用いることにより，PVM が実装されている他の計算機とのインタフェースの統一が可能となる．
- 並列プログラミングライブラリ自身の性能向上の研究:
  - 定性的性能
    並列プログラミングライブラリを用いた並列プログラムの作成においては，ライブラリ関数の機能，記述能力が重要となる．PVM を AP1000 に実装し，また実際にプログラムを作成することにより，ライブラリの定性的性能について調べる．また，実際に実装を行うことで，放送通信，同期，タスクの動的生成および実行等ライブラリの機能を実現するために必要なハードウェア，OS 等の機能も明らかになる．
  - 定量的性能
    並列プログラミングライブラリ関数を用いた通信にかかる時間は，メッセージの転送時間およびライブラリ関数自身のオーバーヘッド時間の合計である．高速な通信には，高速な通信媒体および高速なライブラリか必要となる．PVM を AP1000 に実装し，通信媒体およびライブラリの違いによる通信性能の違いを調べ，より高速かつ高機能のライブラリについて研究する．

本稿では，AP1000 上に実装した PVM に関しての報告を行う．まず 2章で PVM の説明を行い，3章で実装の方針について，4章で実装した場合の機能について述べる．5章で性能の評価およびその結果ついての考察を行い，6章で本稿のまとめとする．

2 並列プログラミングライブラリ PVM

2.1 PVM の構成

PVM は，ワークステーションクラスタを実現するための並列プログラミングライブラリである．ユーザは逐次型言語で書かれたプログラムの中で PVM の関数を呼び出し，他のワークステーション上の PVM のプロセスと通信を行うことで，複数のワークステーションをあたかも一台のメッセージ交換型マルチコンピュータのように使用することができる．以下，PVM による仮想的並列計算機をバーチャルマシン，バーチャルマシンを構成する個々のワークステーションを PVM ホストと呼ぶ．

PVM は，以下の **PVM-Daemon**(pvmd) と **PVM** ライブラリの2つで構成される．

PVM-Daemon

各 PVM ホストに起動されている．タスク間の通信はすべて pvmd を介して行われる．他の PVM ホストにあるタスクとの通信は UDP プロトコル，同一 PVM ホスト上にあるタスク間のデータ通信は TCP プロトコルの Socket を使って行われる．

PVM ライブラリ
PVM ライブラリは，バーチャルマシン上で実行するプログラムで使用する関数のライブラリである．タスク間のメッセージ通信，バリア同期，タスクの生成，PVM ホストをバーチャルマシンに加える，等の機能が関数として用意されている．

2.2 PVM の機能

バーチャルマシンの構成
バーチャルマシンを構成するホストの変更は，通常は PVM を起動した後，コンソールからマニュアルで行う．また，プログラム中で `pvm_addhosts()`, `pvm_delhosts()` 関数を用いて行うことも可能で，プログラム実行中に動的に構成を変えることができる．

タスクの生成
バーチャルマシン上で実行されるタスクには，(1) はじめに起動されるタスク (マスタタスク) と (2) マスタタスクから起動されるタスク (スレーブタスク) の 2 種類がある．タスクの生成には，`pvm_spawn()` 関数を用いる．マスタタスクは複数回スレーブタスクを生成することができ，マスタタスクが生成したスレーブタスクも，新たにタスクを生成することができる．すなわち，バーチャルマシン上のタスクは，実行時に動的に生成することができる．

メッセージの送信および受信
他のタスクにメッセージを送信するには，まずメッセージバッファを生成し (`pvm_initsend()`), 送信するデータをバッファに入れる (データのパッキング: `pvm_pk*()`). その後，バッファ中のデータを指定したタスクに送信する (`pvm_send()`). メッセージの受信には，同期式 (`pvm_recv()`) および非同期式 (`pvm_nrecv()`) があり，どちらも受信したデータをバッファに入れ，バッファ内のデータを取り出す (アンパッキング:`pvm_upk*()`) ことにより，データ転送を完了する．いくつかのデータをパッキングして送ることにより，例えば int 型と float 型といった型の違う複数のデータをまとめて送受信することができる．

タスクのグルーピング
PVM では，いくつかのタスクをまとめてタスクのグループを作ることができる．放送，同期などをグループに対して行うことも可能である．グループへの参加 (`pvm_joingroup()`) および離脱 (`pvm_lvgroup()`) は，随時行うことができる．また，一つのタスクが複数のグループに属することも許されている．

3 PVM/AP1000 の実現方針

以下，我々が AP1000 上に実装した PVM を PVM/AP1000 と呼ぶ．PVM/AP1000 を実装するにあたって，方針として決めるべき項目には，

1. PVM/AP1000 と他の PVM ホストを接続するか
2. ライブラリ関数をどのレベルで移植するか

があり，それぞれについて次のような選択肢が存在する．

3.1 他の PVM ホストとの接続

PVM/AP1000 を他の PVM ホストと接続するか否かで，AP1000 のホスト上で実行されるプログラムが以下のように異なる．

1. PVM/AP1000 を他の PVM ホストと接続しない場合:
   AP1000 単独で，PVM のバーチャルマシンと同等のものを構成する．すなわち，AP1000 のセル 1 つ 1 つがバーチャルマシンの 1 ワークステーションに対応し，AP1000 で閉じた世界の中で PVM を実現する (図 1)．この時,「AP1000 のホスト上では PVM のマスタプログラムが実行され，マスタプログラムによりセル上にスレーブプログラムが生成および実行される」という AP1000 での通常のプログラム実行方法と同じになる．
2. PVM/AP1000 を他の PVM ホストと接続する場合:
   AP1000 は，バーチャルマシンを構成するワークステーションの 1 つとなり，AP1000 のセル上で実行される PVM のスレーブタスクは，外部のワークステーション上のマスタプログラムから直接的

に生成および実行することが可能となる．この時，AP1000 のホスト上には外部のワークステーションとのインタフェースとなる pvmd が動くことになる (図 2)．文献 [3] では，この方法で PVM を AP1000 に実装している．

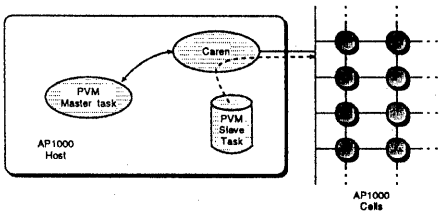

図 1: PVM/AP1000 を他の PVM ホストと接続しない場合

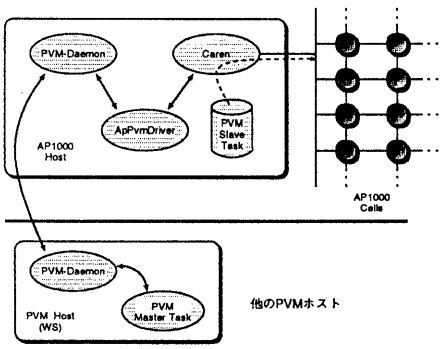

図 2: PVM/AP1000 を他の PVM ホストと接続する場合

3.2 ライブラリ関数の移植レベル

PVM ライブラリ関数を，AP1000 のライブラリ関数を用いて実現する．これには，次の 3 つのレベルが考えられる．

レベル 1: 各セルに pvmd は載せず，スレーブタスクのみを走らせる．PVM の通信用関数を，それぞれ直接 AP1000 のライブラリ関数で実現する (図 3)．これは，レベル 1 および 2 がどちらも基本的にワークステーション用 PVM をそのまま AP1000 に載せようとするのに対し，PVM 用に書かれたプログラムが AP1000 で動くように，PVM のライブラリ関数を AP1000 独自のライブラリ関数を用いて作るというものである．

レベル 2: 各セルにスレーブタスクと pvmd を走らせ，PVM の内部で，使用されるシステムコールを AP1000 のライブラリ関数に置き換える (図 4)．レベル 3 と似ているが，PVM の内部のソースプログラムを変更し，PVM が呼び出すのはシステムコールではなく AP1000 のライブラリ関数となっている点が異なる．

レベル 3: 各セルにスレーブタスクと pvmd を走らせ，PVM が使用する UNIX のシステムコールを AP1000 のライブラリを用いて疑似的に実現する (図 5)．PVM が使用する UNIX のシステムコール (主に Socket 通信用システムコール) を AP1000 の各セル上で使えるようになるので，ワークステーション用の PVM を何の変更もなしに各セルの上にそのまま載せて実行することができる．

今回の PVM/AP1000 では，他の PVM ホストと接続せず，レベル 1 での実現方法をとった．これは，以下の理由による．

- 他の PVM ホストと接続しないため，ワークステーションとのインターフェースを考える必要がなく，実現が比較的簡単である．
- PVM-Daemon を使わず，直接 AP1000 のライブラリ関数を使用しているため，ライブラリ関数のオーバーヘッドが小さくなる．

将来的には，他の PVM ホストと接続することを考えている．

4 PVM/AP1000 の機能

4.1 制限・変更される機能

PVM/AP1000 では，AP1000 の機能の関係上，以下のような制限・変更される機能がある．

PVM のホスト構成の変更
PVM ではバーチャルマシンのホスト構成を動的に変えることができるが，AP1000 の使用セル構成はプログラム実行中に動的に変更することはできない．

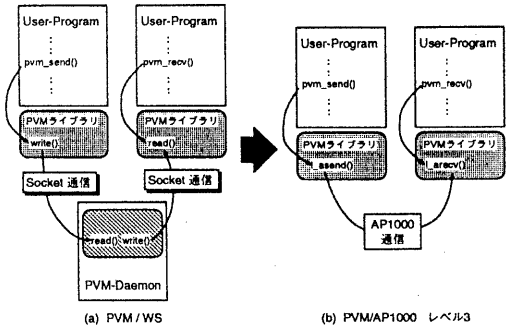

図 3: レベル 1:PVM の関数を直接 AP1000 の関数で実現する

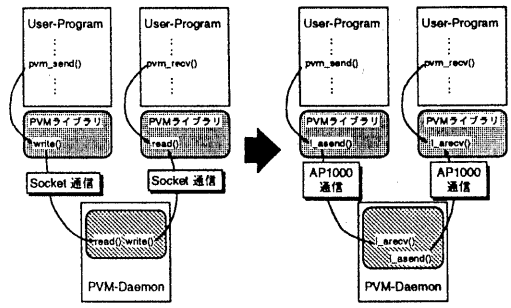

図 4: レベル 2:PVM のソースプログラム中のシステムコールを AP1000 の関数に置き換える

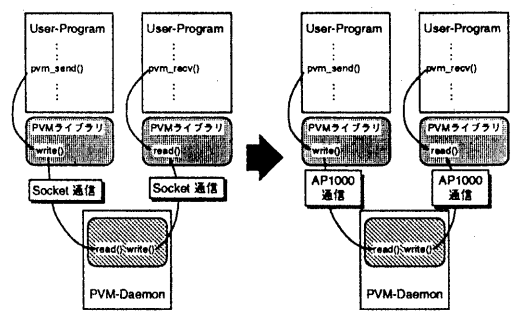

図 5: レベル 3:疑似システムコールを作る

表 1: PVM/AP1000 で実現した関数

pvm_barrier()	pvm_bcast()	pvm_bufinfo()
pvm_exit()	pvm_freebuf()	pvm_getinst()
pvm_getrbuf()	pvm_getsbuf()	pvm_gettid()
pvm_gsize()	pvm_initsend()	pvm_joingroup()
pvm_lvgroup()	pvm_mcast()	pvm_mkbuf()
pvm_mytid()	pvm_nrecv()	pvm_pk*()
pvm_parent()	pvm_perror()	pvm_recv()
pvm_send()	pvm_setrbuf()	pvm_setsbut()
pvm_spawn()	pvm_upk*()	

表 2: PVM/AP1000 で実現しなかった関数

pvm_addhosts()	pvm_config()	pvm_delhosts()
pvm_kill()	pvm_mstat()	pvm_notify()
pvm_pstat()	pvm_sendsig()	pvm_serror()
pvm_tasks()		

タスクの動的割り付け

AP1000 ではセル上で実行されるタスクの生成はホスト上のマスタタスクだけが 1 回のみ行うことができる．よって PVM/AP1000 では，スレーブタスクの生成は 1 回きりで，タスクの動的生成および割り付けはできない．

タスクのグルーピング

PVM ではいくつかのタスクでグループを作り，グループ内でバリア同期や放送を行うことができる．AP1000 ではメッセージ放送およびバリア同期を実行中のすべてのタスクについて高速に行う機能があるが，これを用いて特定タスク間のみでメッセージ放送およびバリア同期を直接行うことはできない．したがって PVM/AP1000 では，グルーピングの機能は AP1000 のハードウェアがもつ放送およびバリア同期の機能は使わずに，通常の通信のみを用いてソフトウェア的に実現する．

4.2 PVM/AP1000 のプログラミングスタイル

PVM/AP1000 で実現可能なプログラミングは以下の様になる．

- PVM の書式 (C 言語+PVM ライブラリ関数) に従って書かれたプログラムを AP1000 上で実行することができる．

- 実行の方法は，通常のAP1000でのプログラムの実行方法と同じである．マスタタスクをAP1000のホスト上で起動し，マスタタスクがAP1000のセル上にスレーブタスクを生成および実行させる．
- マスタタスクによるスレーブタスクの生成は1度だけ行うことができる．スレーブタスクは，新たにタスクを生成することはできない．
- タスクのグルービングに関しては，PVMでの機能がそのまま使用できる．

5 性能評価

基本的な通信パターンにおける性能を測定することにより，PVM/AP1000の関数を用いることが通信性能へどの程度の影響を与えるかを調べる．

5.1 性能評価方法

作成したPVM/AP1000の性能を，5.2で述べる項目について調べる．同様の項目について，ワークステーション上のPVM(PVM/WS), AP1000独自のライブラリのそれぞれで測定し，性能を比較する．

PVM/AP1000 およびAP1000独自のライブラリ

- ホスト-セル間およびセル-セル間についてそれぞれタスク数4および64で行う．
- ホスト-セル間については，1ホスト+3セルおよび1ホスト+63セルで行う．
- セル-セル間については，(1ホスト+)4セルおよび(1ホスト+)64セルで行う．

PVM/WS

- タスク数4で行う．4タスクがすべて同じPVMホスト上にある場合およびそれぞれ異なるPVMホスト上にある場合について行う．

時間の測定は，プログラム中の測定範囲の前後に時刻をとる特定の関数を埋め込むことにより行っている．通信時間を測定する場合，受渡しされるメッセージのサイズを変えながら測定を行った(0〜4000バイト)．

5.2 評価項目

評価項目として，以下を定義する．

- **1対1通信:** 1つのタスクが，他の1つのタスクに対してメッセージを送信する．

  送信タスクがメッセージ送信を開始してから，受信タスクがメッセージを受信し終るまでの時間を測定する．2つのタスク間でメッセージを交互に送受信し，かかった時間を通信回数で割って1回の通信時間を求める．

- **全対全放送:** すべてのタスクがそれぞれ他のすべてのタスクに対して同じメッセージを送信する．

  すべてのタスクが一斉に放送用関数を用いて自分を除く全タスクにメッセージを送信し，自分を除く全タスクからのメッセージを受信するまでの時間を測定する．

- **バリア同期:** すべてのタスク間でバリア同期をとる．

  はじめのタスクがバリア同期に入ってから，バリア同期が完了してすべてのタスクが実行を再開するまでの時間を測定する．

5.3 結果

5.3.1 1対1通信

結果を図6に示す．グラフにおいて，メッセージサイズが0の時の値が(関数のオーバーヘッド+通信路の速度)，線の傾きが通信路のバンド幅を表していると考えられる．また，同一通信路における値の差がライブラリ関数自身のオーバーヘッドの差を表していると考えられる．

この結果から，PVM/AP1000における通信にかかる時間は以下の式で計算できる．

$$host - cell : t_{c-c} = 1.72 \times 10^{-4} \times n + 0.10(msec)$$

$$cell - cell : t_{h-c} = 5.36 \times 10^{-4} \times n + 2.45(msec)$$

PVM/WSとPVM/AP1000を比較すると，PVM/WSではUNIXの通信手段であるSocket通信を使っているため，ライブラリ関数のオーバーヘッドが大きい．また，通信路のバンド幅もAP1000と比較して小さいため，グラフの傾きが大きくなっている．

表 3: バリア同期にかかる時間

ライブラリ	タスクの位置	タスク数	時間
PVM /AP1000	ホスト-セル	4	5.71[msec]
		64	33.49
	セル-セル	4	2.78
		64	24.63
PVM/WS	同 PVM ホスト	4	21.15
	異 PVM ホスト	4	34.31
AP1000	ホスト-セル	4	1.35
		64	1.35
	セル-セル	4	0.18
		64	0.18

関数のオーバーヘッドが一定であるとすると，メッセージサイズが大きくなるにつれ転送時間が長くなり，通信時間全体に占める関数のオーバーヘッドの割合が小さくなると考えられる．実際，セル-セル間通信の場合メッセージサイズ 0 バイトのときのオーバーヘッドがセル-セル間で約 70%およびホスト-セル間で約 9%であるのに対し，4000 バイトのときにはセル-セル間で約 3%およびホスト-セル間で約 5%となっている．ある程度大きなメッセージの通信には十分実用的であるといえる．

5.3.2 全体全放送

結果を図 7，7に示す．

PVM/AP1000 では pvm_mcast()，AP1000 では cbroad() といった放送用関数を使用している．AP1000 の cbroad() は放送用ネットワーク B-Net を用いて全タスクに一度に放送を行うが，PVM の pvm_mcast() では指定したタスクに 1 つ 1 つ順番に 1 対 1 通信を用いて送信する．このため，pvm_mcast() ではタスク数が増えると通信回数が増えるため，グラフの傾きが大きくなる．

5.4 バリア同期

結果を表 3に示す．

PVM/AP1000 では sync()，PVM では pvm_barrier() といったバリア同期用関数を使用している．AP1000 の sync() は同期用ネットワーク S-Net を用いて全タスクが高速に同期を行う．PVM/AP1000 の pvm_barrier() では指定したタスク間でのみで同期をとるため，メッセージ通信を用いてソフトウェア的に同期をとっている．このため，バリア同期にかかる時間はタスク数に比例して増える．

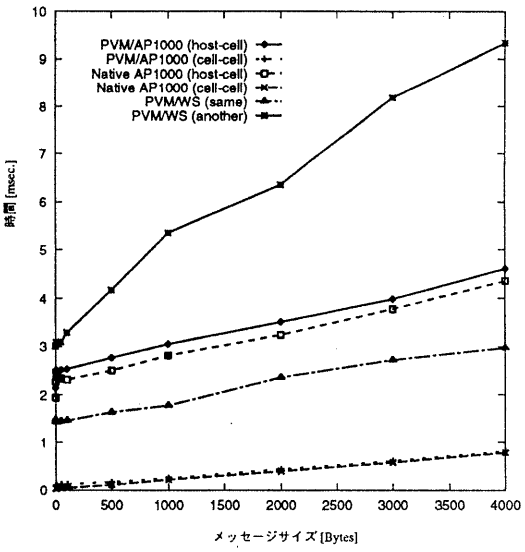

図 6: 1 対 1 通信

6 おわりに

ワークステーションクラスタを構成するための並列プログラミングライブラリ PVM を AP1000 上に実装した．本稿では，ライブラリを実装するときの関数の実現の方法などの選択肢について述べ，基本的な通信性能を測定した結果について述べた．

今後，実際のベンチマークプログラムを用いての評価などを行い，並列ライブラリの問題点などについての研究を行う予定である．また，他の PVM ホストとのインターフェースに関する部分の実装を行う予定である．

謝辞

日頃ご討論頂く，九州大学 大学院総合理工学研究科 安浦寛人 教授をはじめ，安浦研究室の諸氏に感謝し

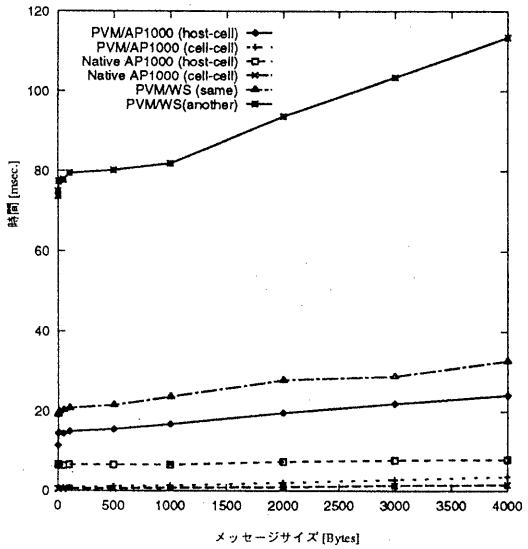

図 7: 全体全放送 (4 タスク)

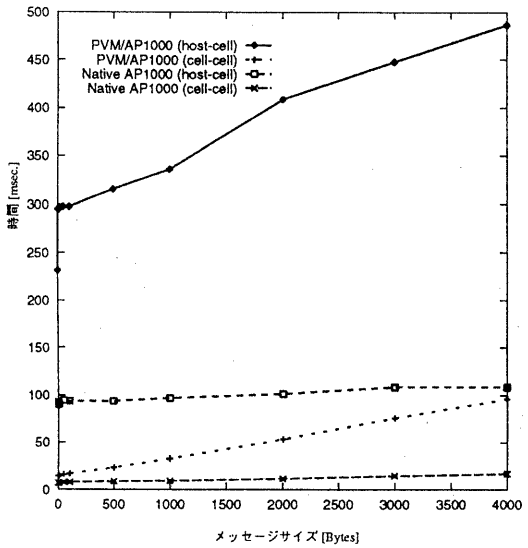

図 8: 全体全放送 (64 タスク)

ます.

本研究は一部，文部省科学研究費補助金 重点領域研究「超並列原理に基づく情報処理体系」，および，EAGL 事業推進機構 育成研究助成金による.

参考文献

[1] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam, *PVM 3.0 USER'S GUIDE AND REFERENCE MANUAL*, ORNL/TM-12187, Feb. 1993.

[2] 富士通研究所, AP1000 プログラム開発手引書 (I) C 言語インタフェース, 第 2 版, 1992 年 5 月.

[3] C. W. Johnson, D. Walsh, "Porting the PVM Distributed Computing Environment to the Fujitsu AP1000" *Proceedings of 2nd Parallel Computing Workshop*, P1-D-1–P1-D-12 ,Nov. 1993.

[4] Shigenobu Iwashita, Katsuhiro Kunisada, Kazuaki Murakami, "Implementing the PVM (Parallel Virtual Machine) on the AP1000," *Proceedings of 2nd Parallel Computing Workshop*, P2-E-1–P2-E-6 , Nov. 1993.

[5] 佐藤三久, 児玉祐悦, 坂井修一, 山口喜教, "並列計算機 EM-4/X の RICA によるメッセージ通信の実現," 情処研報, 93-ARC-102-7, pp.49–56, 1993 年 10 月.

[6] 関口智嗣, 長嶋雲兵, 日向寺祥子, "ワークステーションクラスタとメッセージパッシングライブラリ," 情処研報, 93-HPC-47-3, pp.13–20, 1993 年 6 月.