ハイパフォーマンス
コンピューティング 57－6
（1995．8．24）

分散環境における共有メモリ型 SPMD プログラミングモデル Split-C/PVM の実装

南里 豪志 † 佐藤 周行 ‡ 島崎 眞昭 ‡

† 九州大学情報工学専攻
‡ 九州大学大型計算機センター

概要

ワークステーションクラスタ上に共有メモリ型SPMD プログラミング言語Split-C/PVM を構築した. Split-C/PVM は分散メモリ型並列計算機用の並列 C 言語 Split-C を分散環境システム PVM 上に構築したものである. PVM は汎用性が高いため, 広く利用されている. そのため Split-C/PVM を様々なプラットフォーム上に構築することができる. 行列積を用いたベンチマークの結果, PVM の Message Passing モデルや並列計算機 CM-5 に対して遜色のない性能が得られたため, Split-C/PVM の共有メモリ機構を効率良く構築できたことが示された.

Implementing Split-C/PVM A Shared Memory SPMD Programming Model On a Distributed Environment

Takeshi Nanri † Hiroyuki Sato ‡ Masaaki Shimasaki‡

† Department of Computer Science and Communication Engineering, Kyushu University,
‡ Computer Center, Kyushu University,

Abstract

In our work, we implemented Split-C/PVM, a shared memory SPMD programming language on workstation cluster environment. Split-C was originally developed for a distributed memory parallel machine. We ported Split-C on a distributed computing environment using PVM. Split-C/PVM can be a uniform distributed computing platform because PVM is a portable message passing library package for a distributed environment. We used matrix multiplication as a benchmark. From the result, Split-C/PVM proved to be competitive with Message Passing model of PVM and parallel computer CM-5. So we can conclude that we implemented shared memory system sufficiently efficient.

1 背景

近年の High Performance Computing において並列処理は不可欠な要素である. この並列処理環境を安価に提供するシステムとして, ネットワーク結合されたワークステーションクラスタにおける分散処理システムが注目されている. 現在開発されている分散処理システムの多くは Message Passing 型のプログラミングモデルを提供している. Message Passing モデルは非常に低水準であるため, マニュアルで効率の良いコードを生成するのは困難である. これに対し近年, 分散メモリ型アーキテクチャ上にプログラミングの容易な共有メモリ型モデルを構築する, 分散共有メモリシステムに関する研究が多くなされている.

本研究で構築する Split-C/PVM は, ワークステーションクラスタ上の分散共有メモリシステムである. Split-C/PVM は汎用性が高く, 異機種, 同機種を問わず様々なワークステーションクラスタ上に構築できる. さらに Split-C/PVM が提供する共有メモリ型 SPMD モデルは, 様々な最適化に関する研究においてターゲット言語として扱われているため, そのような最適化を利用して HPF [6] のようにより高水準な言語を構築することも可能である. 本稿では Split-C/PVM のコストモデルとして NUMA を利用し, システムを評価する.

本稿は 2 節で Split-C/PVM の構築方法を説明し, 3 節でコストモデル NUMA と Split-C/PVM の関係を示す. 4 節で分散処理システムとしての性能を初期評価し, 5 節で関連研究を紹介する.

2 Split-C/PVM の構築

Split-C/PVM は分散処理システム PVM 上に共有メモリ型 SPMD 並列 C 言語 Split-C を実現したものである. 本節ではまず PVM 及び Split-C を紹介し, 次に Split-C/PVM を構築する上で必要となる技術を説明する.

2.1 Split-C

Split-C はカリフォルニア大学で分散メモリ型並列計算機用に開発された SPMD 型並列 C 言語である [3]. 現在 CM-5 (Thinking Machines), Paragon (Intel), SP-1 (IBM) 上に構築されている. Split-C は共有メモリ型のプログラミングモデルを提供しているため, Message Passing モデルに比べてプログラミングが容易である.

Split-C におけるプロセッサ間通信は, 大域メモリに対するメモリアクセスの形で行なわれる. Split-C には, この大域メモリアクセスを開始フェーズと転送フェーズに分割した split-phase assignment と呼ばれるアクセス命令が用意されており, これを用いてプロセッサ間通信を非同期に行なうことが出来る. また, ブロック転送命令による非同期大量通信も提供されている. Split-C の通信, 同期命令を表 1 にまとめた.

単位要素転送

同期	=
非同期	:=, :-

ブロック転送

同期	bulk_write(), bulk_read()
非同期	bulk_put(), bulk_get(), bulk_store()

同期

転送終了確認	sync(), all_store_sync()
バリア同期	barrier()

表 1: Split-C の通信関数

2.2 PVM

PVM はネットワーク接続されたワークステーションクラスタを一つの並列計算機として利用する, 分散処理環境を提供するシステムである [5]. PVM は様々な機種で構成されるワークステーションクラスタ上に構築できるため, 広く利用されている。

PVM のシステムはデーモンプロセスとライブラリで構成されている。クラスタ中の全ワークステーションで予めデーモンプロセスが起動されている状態で、あるプロセスがデーモンプロセスに対して実行オブジェクトを spawn することによって分散処理が開始される.

PVM における通信はハンドシェイク型の Message Passing で行なわれるため, 送信側の send 命令と受信側の receive 命令が対応している必要がある. ここで送信側はメッセージをネットワークに出力後次の処理に移ることが出来る. これに対して受信側には、要求するメッセージの到着まで待つブロック受信命令と、実行された時点で到着していなければ次の処理に移る非ブロック受信命令が用意されている。

2.3 Split-C/PVM

分散メモリ型並列計算機上の Split-C はハードウェア及びベンダーから提供された Message Passing ライブラリによってサポートされている. 本節では CM-5 の Message Passing ライブラリ CMMD と PVM を比較して Split-C を PVM 上に構築する方法を説明する.

2.3.1 並列処理

CM-5 上の Split-C コンパイラは, 初期化ルーチンと並列処理コードで構成される一つの実行オブジェクトを生成する. このオブジェクトは図 1 のように, まず Partition Manager と呼ばれるプロセッサで実行される. その後並列処理コードが各プロセッサに分配され, 並列処理が開始される.

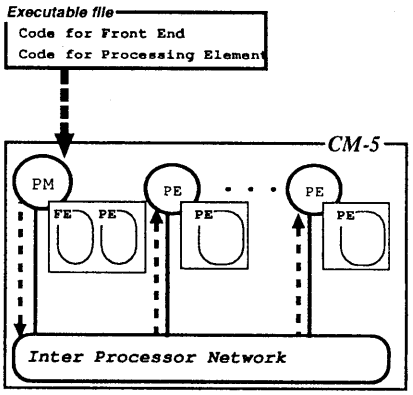

図 1: Split-C on CM-5

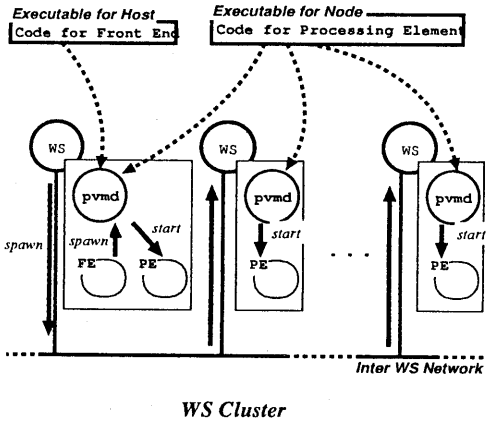

図 2: Split-C/PVM

これに対し PVM では, 各ワークステーションに実行オブジェクトを spawn することにより並列処理が実現される. そこで Split-C/PVM では, 図 2 のように, 初期化オブジェクトと並列処理オブジェクトを用意する。初期化オブジェクトを実行したプロセスが並列処理オブジェクトを spawn することによって並列処理が開始される.

2.3.2 大域アドレス空間

共有メモリ型プログラミングモデルでは, 各プロセスが大域アドレス空間の任意のオブジェクトに対して一意にアクセス可能である必要がある. CM-5 の様な分散メモリ型並列計算機では, 全プロセスでメモリイメージが同じであることが保証されているため, このような大域アドレス空間を容易に提供できる. しかし分散環境では各プロセスが同じメモリイメージを持つ保証がない.

そこで Split-C/PVM では, ベースアドレス及び相対アドレスによって大域アドレス空間を実現する. ベースアドレスは各並列プロセスの実行開始時にシステムコール sbrk(0) によって設定する. 各並列プロセスは, 自分が所有する領域のデータに対しては絶対アドレスでアクセスする. これに対し他のプロセスが所有する領域のデータに対しては, 絶対アドレスとベースアドレスの差で得られる相対アドレスを用いて所有するプロセスにデータを要求することによってアクセスする.

2.3.3 プロセッサ間通信

Split-C では, プロセッサ間通信は大域アドレス空間へのメモリアクセスとして指示される. 大域アドレス空間への書き込みは, その領域を所有するプロセッサへの送信, 読み出しは受信をそれぞれ表現している. このように, Split-C のプログラム中ではプロセッサ間通信の送信又は受信の一方のみを指定すれば良いため, プログラミングが容易である.

このような通信命令は, CM-5 上では active message と呼ばれるメッセージで実現される. active message はハンドラ関数及びその関数への引数によって構成されるメッセージであり, 受けとったプロセッサは現在処理しているプロセスを中断してハンドラ関数を指定された引数で実行する. active message の機能は, ハードウェア及びライブラリによって提供される.

これに対して PVM では、任意のプロセッサにシグナルを送信する機能は用意されているが、プロセッサ間通信において送信と受信がプログラム中で対応している必要がある. そこで Split-C/PVM では, active message の機能を割り込みとメッセージ転送の二つのフェーズに分けて実行することにより, 大域アドレス空間へのアクセスとしてのプロセッサ間通信を実現する (3). 例えばプロセス p1 が所有するデータに対してプロセス p0 が書き込みを行なう場合, 以下のような手順でプロセッサ間通信が行なわれる.

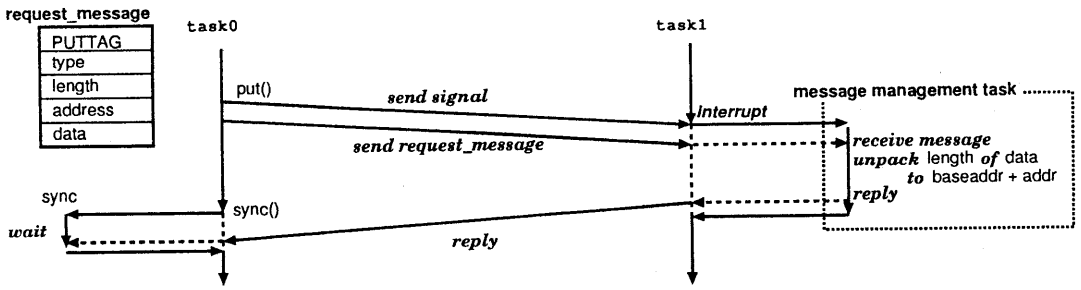

図 3: put with pvm

1. p0 から p1 に対してまずシグナルを送信し, 次に書き込み要求メッセージを送信する. 非同期通信の場合はメッセージをネットワークに出力した後次の処理に移る.

2. シグナルに割り込まれた p1 はメッセージ管理タスクに制御を移し, 要求メッセージを待つ.

3. メッセージが転送されるとそのメッセージタグから要求内容を識別し, メッセージ中のアドレスと値を用いてデータの書換えを行なう.

4. 処理が終了すると p1 は p0 に対して応答メッセージを送信し, 元の処理に制御を戻す.

5. 同期通信の場合 p0 は, p1 からの応答メッセージを待って次の処理に移る. 非同期通信の場合は応答メッセージの到着を明示的に確認する命令が必要となる. p0 がこの命令を実行すると, p1 からの応答メッセージ到着までブロックし, 書き込みの終了を確認する.

3 Split-C/PVM のコストモデル

分散メモリ型並列計算機におけるメモリアクセスは, local access と global access の二階層に分けられる. そのため, 実行コストモデルとして NUMA が適している. Split-C のコストモデルも NUMA で評価でき, Split-C/PVM は仮想分散メモリ型並列計算機上に Split-C を構築したものであるため, Split-C/PVM のコストモデルも NUMA として評価できる.

NUMA において性能を決定するパラメータは以下の通りである.

1. 各プロセッサの単体での処理速度

2. 局所メモリアクセス速度

3. 大域メモリアクセス速度

4. memory contention による影響

```
for (j = 0; j <= N - 1; j++)
  for (k = 0; k <= N - 1; k++)
    for (i = 0; i <= N - 1; i++)
      C[j * N + i] = C[j * N + i]
                   + A[j * N + k]
                    * B[k * N + i];
```

図 4: Naive Code of Matrix Multiplication

Split-C/PVM の場合, プロセッサ間ネットワークが低速であるため, 3がシステムの性能を左右する.

3は, 一度に転送されるデータ量がネットワークのピーク性能に近いほど高くなる. そこで Split-C/PVM では, 大量通信によってより効率の良いメモリアクセスを実現する.

4 性能評価

ワークステーションクラスタ上に構築した Split-C/PVM の処理速度を計測し, PVM 上で Message Passing によるプログラミングを行なった場合, 及び CM-5 上で Split-C を用いた場合と比較した. 利用したワークステーションクラスタ及び CM-5 の計測環境は表 2の通りである. 計測には行列積を用いた.

4.1 通信速度

Split-C/PVM 及び PVM, CM-5 の通信速度は表 3 のようになる. ここで Naive は通信の最適化を行なわなかった場合 (図 4), Opt は最適化を行なった場合 (図 5) である. 最適化としては, 非同期大量通信を用いるための communication aggregation を行なった.

Split-C/PVM の通信速度は, CM-5 のように高速ネットワークを用いている並列計算機より非常に遅い. しかし, Message Passing による PVM コードとほとんど同じ速度を得ることが出来た. そのため, Split-C/PVM はネットワークを最大限に利用していることが分かる.

	WS Cluster	CM-5
Machine	SPARC Station10	CM-5 *without Vector Unit*
Processors	8	
Memory	48MByte / WS	32 MByte / PE
CPU Speed	5.8 MFLOPS/WS	2.4 MFLOPS/PE
OS	SunOS 4.1.3	CMOST Version 7.3
Network	Ethernet	

表 2: Environment of Performance Evaluation

		Split-C/PVM	PVM	CM-5
Naive		0.001	0.001	0.037
Opt	100	0.16	0.16	3.43
	200	0.16	0.16	3.67
	300	0.18	0.18	3.84
	400	0.16	0.17	3.47
	500	0.16	0.17	3.79

表 3: Global Memory Access Speed (MB/sec)

Size	Single (A)	Split-C/PVM (B)	B/A	PVM (C)	C/A
100	5.43	2.54	0.47	3.31	0.61
200	5.38	4.43	0.82	5.77	1.07
300	5.42	8.47	1.56	10.34	1.91
400	5.36	9.65	1.80	12.29	2.29
500	5.38	10.87	2.02	14.16	2.64

表 4: ワークステーションクラスタ上の処理速度 (MFLOPS)

Size	Single(D)	CM-5(E)	E/D
100	2.88	11.00	3.82
200	2.41	12.23	5.07
300	2.43	12.59	5.18
400	2.43	12.95	5.33
500	2.43	12.95	5.33

表 5: CM-5 上の処理速度 (MFLOPS)

```
for (p = 0; p < PROCS; p++)
  bulk_store(&(g_BUFF[p][buff_location * N])
           , l_B, ll_N * N * sizeof(double));

all_store_sync();

on_one{
  for (l_j = 0; l_j <= ll_N - 1; l_j++)
    for (k = 0; k <= N - 1; k++)
      for (i = 0; i <= N - 1; i++)
        l_C[l_j * N + i] = l_C[l_j * N + i]
                         + l_A[l_j * N + k]
                          * l_BUFF[k * N + i];
}
barrier();
```

図 5: Benchmarking Code

4.2 処理速度

ワークステーションクラスタ, 及び CM-5 おける行列積の処理速度はそれぞれ表 4,5のようになる. この処理速度は全体の浮動小数点計算回数 $2N^3$ を処理時間で割ったものである.

なお, ワークステーションクラスタにおける処理時間としては消費時間を計測したのに対し, CM-5 における処理時間としては CPU 時間を計測した. そのため厳密には両者を比較出来ないが, 計測は真夜中に行なったためワークステーションクラスタをほとんど占有して利用できたので, この結果を用いた評価は無意味ではない.

500 × 500 の行列では, Split-C/PVM の処理速度は 10.87 MFLOPS であり, PVM の 14.16 MFLOPS, CM-5 の 12.95 MFLOPS と比較しても十分な性能が得られた.

5 関連研究

分散メモリ型アーキテクチャ上のプログラミングモデルについては、様々な研究がされている。

最もアーキテクチャを反映したモデルは Message Passing であり、PVM [5] や MPI [4] は共通のインターフェイスを提案することによって広く使われている。また、Jade [10] のように言語として構築されているものもある。

これに対して近年特に注目されているのが共有メモリモデルである。Message Passing モデルと共有メモリモデルは記述能力や簡便性などを争点とした議論が行なわれているが、[2] では注意深くプログラミングされたコードではどちらも同程度の性能を得られることが主張されている。

分散メモリアーキテクチャにおける共有メモリモデ

ルについては様々なプラットフォームでの実装例が報告されている [9] が、それらのほとんどは Message Passing モデル上でのプログラミング言語の実装である。中でも分散メモリ型並列計算機上での SPMD 型言語という形での実装が多い。代表例として Split-C [3] がある. Split-C については複数のプラットフォームに実装されている [8] ことが報告され, さらに Split-C をターゲットにした HPF [6] の実装も報告されている [12].

分散メモリ型並列計算機における最近の研究は, データ局所性の向上 [14], 通信コストの低下 [1], 同期頻度の低減 [7, 13] などによる性能の向上に向けられ多くの結果を得ている. これらの評価のためには具体的なコストモデルが必要である. 特に [11] では抽象度を高く保ち, 並列計算におけるコストをメモリアクセスに帰着して論じることができることを示している.

これに対してワークステーションクラスタ上への実装例は、HP の Medusa など、特定のシステムに限られている。

6 結び

本稿では, 比較的安価なワークステーションクラスタ上に, 分散メモリ型並列計算機と同等の処理速度を持つ共有メモリ型 SPMD プログラミング環境を構築した. 現在は Ethernet を利用しているが, 将来, より高速な FDDI や ATM などで接続されたワークステーションクラスタを利用することにより, さらに性能の良いシステムになることが期待できる.

参考文献

[1] Amarasinghe, S.P. and Lam, M.S. *"Communication Optimization and Code Generation for Distributed Memory Machines."* Proc. ACM SIGPLAN '93 Conference on Programming Language Design and Implementation, June 1993.

[2] Chandra, S. *et al. "Where is Time Spent in Message-Passing and Shared-Memory Programs?,"* Proc. 1994 Architectural Support for Programming Languages and Operating Systems, 1994. pp. 61–75.

[3] Culler, D.E. *et al. "Introduction to Split-C."* University of California, Berkeley, November 9, 1993.

[4] Dongarra, J.J., *et al. "A Proposal For A User-Level, Message Passing Interface In A Distributed Memory Environment,"* Oak Ridge National Laboratory, June, 1993.

[5] Geist, Al *et al. "PVM3 User's guide and reference manual."* Oak Ridge National Laboratory, Sep, 1994.

[6] High Performance Fortran Forum: *High Performance Fortran Language Specification, version 1.0,* 1993.

[7] Kägi, A., *et al., "Techniques for Reducing Overheads of Shared-Memory Multiprocessing"* Proc. 1995 ACM International Conference on Supercomputing, 1995.

[8] Luna, S. *"Implementing an Efficient Portable Global Memory Layer on Distributed Memory Multiprocessors,"* UCB Technical Report UCB/CSD 94-810, 1994.

[9] Ramachandran, U., Yousef, M. and Khalidi, A. *"An Implementation of Distributed Shared Memory,"* Software-Practice and Experience 21, 1991. pp. 443–464.

[10] Rinard, M.C., Scales D.J. and Lam M.S. *"Jade:a high-level, machine-independent language for parallel programming,"* Supercomputing 92, 1992. pp. 245–256.

[11] Sato H. Analyzing DMPC using NUMA as the Performance Model, manuscript, 1995.

[12] Sato H., Nanri T. and Shimasaki M. *"Using Asynchronous and Bulk Communication to Construct an Optimizing Compiler for Distributed-Memory Machines with Consideration Given to Communication Costs,"* Proc. 1995 ACM International Conference on Supercomputing, 1995.

[13] Stricker T.,*et al. "Decoupling Synchronization and Data Transfer in Message Passing Systems of Parallel Computers"* Proc. 1995 ACM International Conference on Supercomputing, 1995.

[14] Wolf, Michael E. and Lam, Monica S. *"A Data Locality Optimizing Algorithm."* Proc. ACM SIGPLAN '91 Conference on Programming Language Design and Implementation, June 1991.