ハイパフォーマンス
コンピューティング 72−21
（1998. 8. 7）

SMP クラスタ上でのリモートメモリ転送を用いた通信と計算のオーバーラップによる性能改善

田中良夫† 松田元彦†
久保田和人† 佐藤三久†

SMP クラスタは高性能計算のためのプラットフォームとして広がりつつある．各 SMP ノードのプロセッサ性能が向上するにつれ，ノード間通信のコストを隠すことが，SMP クラスタで高い性能を得るためにより重要となる．我々はリモートメモリ転送を用いたノード間通信と計算とをオーバーラップさせることにより，SMP クラスタ上でのプログラムの実行性能を向上させた．我々のリモートメモリ転送機能は低オーバーヘッドであり，ホストプロセッサに負荷を与えないため，通信と計算をオーバーラップさせることにより性能を大きく向上させることが可能となる．本稿では通信と計算のオーバーラップの方法およびその結果に関して報告する．我々の実験では，いくつかのベンチマークにおいてノード間通信にかかる時間のほとんどが隠蔽され，実行時間は最大で 24%短縮された．

Performance Improvement by Overlapping Computation and Communication on SMP Clusters

Yoshio Tanaka ,† Motohiko Matsuda ,† Kazuto Kubota † and Mitsuhisa Sato†

Clusters of SMPs (Symmetric Multiprocessor Systems) have emerged as important platforms for high performance computing. To achieve high performance on SMP clusters, as the performance of node processor increases, it becomes more important to reduce the inter-node communication overhead. In order to tolerate inter-node communication, we overlapped the inter-node communication and computation using remote memory based user-level communication primitives NICAM, which we designed on our SMP cluster COMPaS. In this paper, we report on the programming and its performance of COMPaS by overlapping communication and computation. Our experimental results show that communication is almost hidden and the execution time is reduced 24% in the best case.

1. はじめに

近年対称型マルチプロセッサシステム (Symmetric MultiProcessor System, SMP) は広く普及しつつあり，Myrinet などの高速ネットワークを用いた SMP クラスタは近い将来高性能計算のための重要なプラットフォームの１つとなると考えられる．SMP クラスタは共有メモリシステムと分散メモリシステムが混在したアーキテクチャであると考えられる．SMP クラスタ上でのプログラミングモデルとしては，MPI などのメッセージパッシングライブラリのみを用いて行なうものや，分散共有メモリシステムが提供するグローバルアドレスを用いた共有メモリプログラミングなどがある．我々は SMP クラスタ上でのプログラミング方法として，共有/分散融合プログラミングを提案した[5),6)]．共有/分散融合プログラミングにおいては，各ノード内での計算はマルチスレッドプログラミングにより行ない，ノード間の通信はメッセージパッシングやリモートメモリ転送により行なう．図 1に共有/分散融合プログラミングの概要を示す．

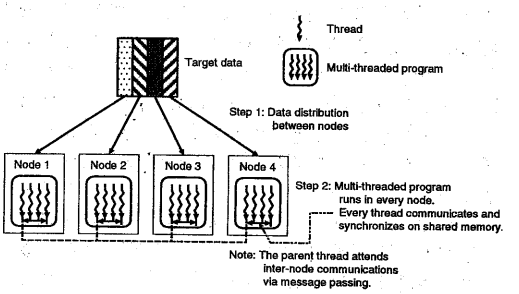

図 1 共有/分散融合プログラミング

† 新情報処理開発機構
Real World Computing Partnership

共有/分散融合プログラミングはSPMDプログラミングスタイルに基づいており，各ノード内で複数のスレッドにより計算が行なわれる分散プログラミングと考えられる．各ノード内のマルチスレッド処理においてデータの局所性を引き出してキャッシュを有効に利用することができれば，共有/分散融合プログラミングはSMPクラスタ上で高い実行性能を得ることができる．

SMPノードのハードウェア性能が向上するにつれ，ノード間通信の性能がプログラムの実行性能により大きな影響を与えるようになる．Myrinetのような高速ネットワークを用いても，プロセッサが十分その能力を発揮することができればノード間通信が性能劣化の大きな要因となってしまう．我々の実験でも各ノード内で複数のスレッドにより高速に計算が行なわれるような場合に，ノード間通信にかかる時間がより大きな比重を占めることが示されている．我々はノード間通信にかかる時間を短縮するのではなく，ノード間通信にかかるコストを隠蔽するアプローチをとる．具体的には，ノード間通信のコストを隠蔽するため，共有/分散融合プログラミングにノード間通信と計算をオーバーラップさせる技法を導入した．いくつかの典型的な繰り返し処理を行なうデータ並列プログラムにおいては，各ノード内での計算と，次の繰り返し処理のための他ノードへのデータ転送とをオーバーラップさせることが可能である．基本的な考えは，計算が終わってからまとめてデータを転送するのではなく，計算が終わった部分から順次データ転送を行なうというものである．

本稿ではノード間通信と計算とのオーバーラップ技法およびその方法に基づくプログラミングの実行性能に関して報告する．実験は我々が作成したSMPクラスタCOMPaS(Cluster Of MultiProcessor System)上で行なった．COMPaSは4プロセッサ(Pentium Pro, 200MHz)のSMPノードをMyrinetで結合したSMPクラスタである．ノード間通信には我々が開発したMyrinet上の通信レイヤであるNICAMを用いる．NICAMは高バンド幅かつ低オーバーヘッドなリモートメモリデータ転送および同期プリミティブを提供する．NICAMはMyrinetのネットワークインタフェース上のDMAエンジンを用いてデータ転送を行なう．NICAMのCPUオーバーヘッドはとても低く，かつホストプロセッサを介さずにデータ転送を行なうことができるため，我々のオーバーラップ技法はノード間通信が性能のボトルネックとなるようなアプリケーションの性能を大きく改善することが可能となる．

次節ではオーバーラップ技法の基本的な考えについて述べる．第3節ではCOMPaSとNICAMの概要および基本性能に関して述べる．第4節では我々のオーバーラップ技法を説明し，実験結果および考察を第5節に示す．第6節では関連研究について述べ，最後にまとめを行なう．

2. データ並列プログラムにおける通信と計算のオーバーラップ

2.1 基本的な考え

典型的なデータ並列プログラムは計算フェーズと通信フェーズの2つのフェーズにより構成される。例えばCG法による連立1次方程式の解法の場合、計算フェーズは行列とベクタの乗算およびベクタの内積、加算などを含み、通信フェーズでは *complete exchange* や *reduction* が行われる。科学技術計算等で多く用いられる行列に対する数値計算をデータ並列プログラムとして実装する場合、これらの通信フェーズと計算フェーズをある回数だけ繰り返して計算を進めるものが多い。これらのプログラムでは、各ノードは通信フェーズにおいて次の繰り返しでの計算のためにリモートノードのデータを更新する。つまり、計算フェーズでは各ノードは前回の繰り返し処理において更新されたデータに対して計算を行うことになる。図2に共有/分散融合プログラミングにより実装された繰り返しを行うデータ並列プログラムの実行の様子を示す。

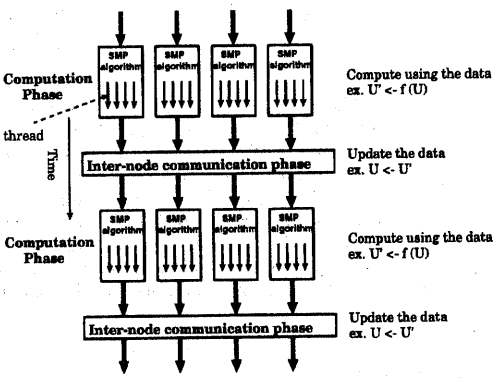

図2 共有/分散融合プログラミングによるデータ並列プログラムの実行の様子

図2では、各ノードは以下の処理を繰り返す。

計算フェーズ
データ(U)に対して計算を行い、その結果をU'に格納する。

通信フェーズ
計算結果(U')をリモートノードに転送し、リモートノードのデータ(U)を更新する。

この方法では「計算がすべて終わってから通信」という手順を踏むことになり、プログラムの実行時間は計算時間と通信時間の合計ということになる。NICAMにおけるデータ転送はホストプロセッサを介さずにネットワークインタフェース上のマイクロプロセッサ

によって行われるため、各スレッドはデータ転送を行っている間に計算を行うことが可能である。つまり、通信と計算をオーバーラップさせることができる。通信と計算をオーバーラップさせるための基本的なアイデアは、計算と通信を細かく分けて交互に実行することにある。具体的には計算が終わった部分から少しずつリモートノードに計算結果を送ることになる。図3は通常の方法とオーバーラップによる方法の実行の様子を比較したものである。

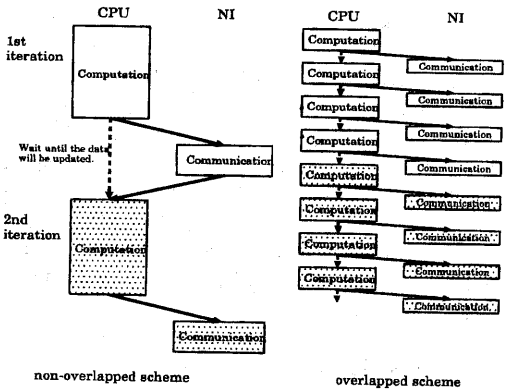

図3　非オーバーラップ 対 オーバーラップ

オーバーラップさせない場合は，計算が終わってからデータ転送を行ない，データが到着してから次の繰り返しの計算を開始する．一方，オーバーラップさせる場合には計算が終わった部分から順次データ転送を行ない，同時に計算も進めていく．次の繰り返しに入った時点で，計算に必要なデータはすでに到着しているか，あるいは到着していなくてもすぐに到着することが期待され，各スレッドは即座に次の計算フェーズを開始することができる．我々のオーバーラップ技法は通常の分散プログラミングに対しても適用することができるが、SMP クラスタのように各ノード内での高い演算性能を提供するプラットフォームの方がより高い効果を引き出すことができる。

2.2 適用できるプログラム

我々のオーバーラップ技法はすべてのデータ並列プログラムに適用できるわけではない．基本的には前述のような，ある領域に対する計算フェーズとリモートノードのデータ領域を更新するための通信フェーズを繰り返すようなデータ並列プログラムに対して適用可能である．問題になるのは，リモートノードのデータ領域に対してリモートコピーを行なって良いかどうかを如何に知るかということである．リモートノードで計算を行なっている最中にその領域にデータをリモートコピーすることは許されない．この問題の解決方法は2通りある．1つは，書き込み許可のフラグを用意し，書き込みを行なう前にリモートリードによってフラグの値を検出する方法である．しかし，この方法ではデータ転送を行なう際に必ずフラグの値をリモートノードから読み出す必要があり，転送コストが増加してしまう．もう1つの方法は，データ領域を2つ用意して交互に利用する方法である．共有/分散融合プログラミングはSPMD プログラミングスタイルに基づいており，各スレッドはリモートノードでどちらのデータ領域を計算の対象としているのかを知ることができる．現在の計算対象とはなっていないデータ領域に対してリモートコピーを行ない，次の繰り返しではそちらの領域に対して計算を行なう．この方法では通常の方法に比べて2倍のデータ領域が必要となってしまう場合もあるが，もともと2つのデータ領域を利用しているようなプログラムの場合には問題がない．

3. COMPaS と NICAM の概要

3.1 COMPaS の概要

本節では我々の実験のプラットフォームである COMPaS と、我々が開発した COMPaS 上の通信レイヤである NICAM の概要および基本性能に関して述べる。図 4に COMPaS の構成を示す。

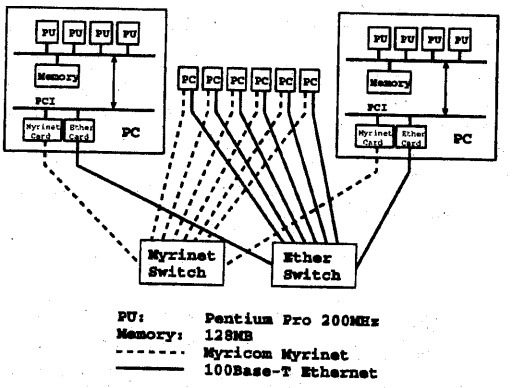

図4　COMPaS の構成

COMPaS は Pentium Pro プロセッサ (200MHz, 16KB L1 cache, 512KB L2 cache, 128MB memory) を4台搭載した SMP システム (東芝 GS700, 450GX chipset) をノードプロセッサとするクラスタシステムである。各ノードは Myrinet および 100Base-T Ethernet により結合されている。Myrinet は計算のために使用し、100Base-T Ethernet は主にファイルシステムの共有のために利用している。各ノードのオペレーティングシステムは Solaris 2.5.1 である。

3.2 NICAM の概要と基本性能

SMP ノードを用いてクラスタを構成した場合，メッセージパッシングによる通信では各ノード上のスレッドに対してバッファ管理に排他制御が必要になりバッ

ファへのコピーが共有資源であるバスに負荷をかけるといった問題がある．これらは通信プリミティブをリモートメモリ転送にすることで避けることができる．NICAM はネットワークインタフェース上のアクティブメッセージ通信を使用した通信レイヤであり，データ転送はネットワークインタフェース上の DMA を起動することにより実行され，ホストプロセッサが関与する必要がなくなる．NICAM のバンド幅は最大で約 82.5MB/s，レイテンシは約 16μ秒である．

4. 共有/分散融合プログラミングにおけるオーバーラップ技法

本節ではラプラス方程式の陽解法を例に共有/分散融合プログラミングにおけるオーバーラップ技法について説明する。ラプラス方程式の陽解法においては、各繰り返しにおいて行列 U から新しい行列 U' が以下の式に基づいて計算される。

$$u'_{i,j} = \frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}}{4} \quad (1)$$

図 5に 4 ノード、4 スレッドの場合の分割方法を示す。

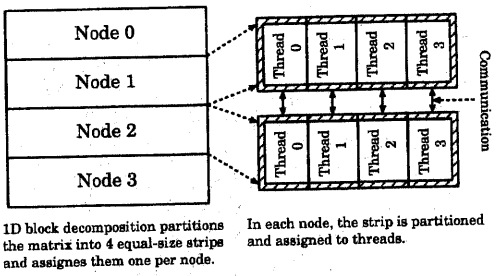

図 5 4 ノード，4 スレッドの場合の行列の分割方法

まず始めに、行列は同じ大きさに分割されて各ノードに分配される。各ノードは分配されたデータを、隣接ノードが保持する境界部分のデータを格納するためのゴーストセルと共に保持する。各ノードにおいて、行列データはさらに分割されて複数のスレッドに割り当てられる。先頭行と末尾の行を更新するには、ゴーストセルの値が必要であり、隣接ノードによってゴーストセルの値が更新されている必要がある。それ以外の行に関しては、ノード内固有のデータのみでデータの更新が可能である。従って、まずは先頭行と末尾の行の計算を行ない，その結果をリモートノードに転送している間にそれ以外の行に関する計算を行なうことができる．図 6にラプラス方程式の陽解法をオーバーラップ技法を用いて実装した場合の処理の流れを示す。

本実装では、各スレッドは以下の処理を繰り返す。

(1) 先頭行を計算し、その結果を隣接ノードに転送する。

(2) 末尾の行を計算し、その結果を隣接ノードに転送する。

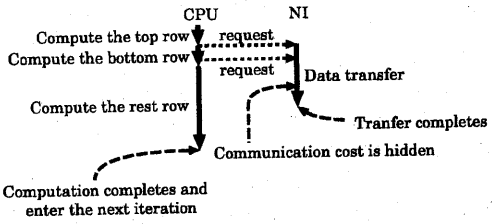

図 6 ラプラス方程式の陽解法におけるオーバーラップの方法

(3) それ以外の行を計算する。

まず始めに境界部分の計算を行って即座に隣接ノードに転送するため、次回の繰り返し処理において先頭および末尾の行を計算する際には、すでに隣接ノードによってゴーストセルの値が更新されていると期待でき、結果としてノード間通信にかかる時間をほとんど隠蔽することができる。発生するノード間通信のコストはノード間通信を行う関数呼び出しのコストのみであり、NICAM においてはこのコストはほとんど無視することができる。

5. 実験結果

我々はラプラス方程式の陽解法，行列の乗算および CG カーネルをオーバーラップの技法を用いて実装した．ラプラス方程式における行列の大きさは 640×640，行列の乗算における行列の大きさは 1800×1800，CG カーネルは Nas Parallel Benchmarks の実装に基づくものであり，サイズは Class A である．ラプラス方程式の陽解法に関しては，前節で述べた方法で通信と計算をオーバーラップさせた．行列の乗算は分散メモリのアルゴリズムとして Canon のアルゴリズムを用いている．Canon のアルゴリズムでは，行列の乗算 $C = A \times B$ において各行列 A，B および C を $n \times n$ のブロックに分割し，各ブロックをノードに割り当てる．各ノードは分担するブロック行列の乗算と，そのブロックのリモートノードへのシフト操作 (転送) を繰り返して行列の乗算を行なう．今回は 8 ノードで実験を行なうため，各行列を 4×4 のブロックに分割し，各ノードに 2 ブロックずつ割り当てた．リモートノードに転送するデータは計算結果ではなく，値が変わらない A および B のブロック行列であるので，ブロック行列の乗算 (計算) を行なう前にブロック行列をリモートノードに転送することにより，通信と計算がオーバーラップする．各ノードはブロック行列を保持する領域を二重に持つことにより，参照中の領域にリモートノードから上書きされるのを防ぐことができる．CG カーネルにおいては，行列とベクタの乗算を行なう際に，結果が得られた部分を少しずつリモートノードに転送することによりオーバーラップさせる．

実験はCOMPaS上で行ない，ノード数は8に固定し，スレッド数を1,2および4に変化させてプログラムの実行時間を測定した．各ベンチマークプログラムにつき，オーバーラップさせないプログラムの実行時間(regular)，計算フェーズのみにかかる時間(comp.)，およびオーバーラップさせたプログラムの実行時間(overlap)を測定した．表1に実験結果を示す．

表1　オーバーラップさせない場合の実行時間(regular)，計算フェーズのみにかかる時間(comp.)，オーバーラップさせた場合の実行時間(overlap)．かっこ内の数字はregularを1とした場合の割合．ノード数は8ノードに固定し，スレッド数を1,2,4に変化させた．

Laplace solver			
threads	regular	comp.	overlap
1	0.176	0.160(0.91)	0.164(0.95)
2	0.085	0.072(0.85)	0.075(0.88)
4	0.041	0.030(0.73)	0.031(0.76)
Matrix multiply			
1	17.85	17.02(0.95)	17.33(0.97)
2	9.42	8.59(0.91)	8.90(0.95)
4	4.46	4.46(0.84)	4.74(0.90)
CG Kernel			
1	9.57	8.74(0.91)	9.17(0.96)
2	7.67	6.47(0.84)	7.26(0.94)
4	7.55	5.83(0.77)	7.11(0.94)

図7に各ベンチマークの速度向上率を示す．

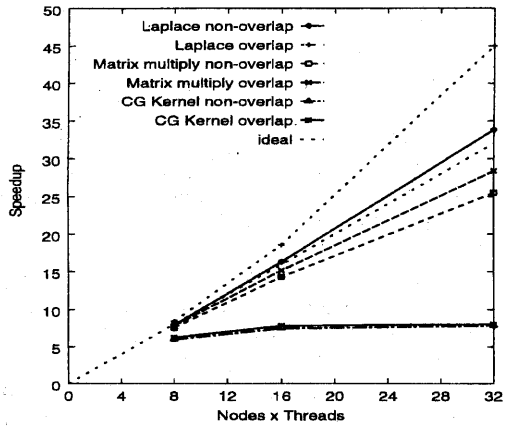

図7　速度向上率

すべてのベンチマークに関して，オーバーラップさせた場合の速度向上率はさせない場合のそれを上回っている．特にラプラス方程式の陽解法においては，4スレッドの場合オーバーラップさせた場合はさせない場合に比べて1.36倍速くなっている．

図8に，オーバーラップさせない場合の実行時間に対する計算時間およびオーバーラップさせた場合の実行時間の割合を示す．

このグラフより，オーバーラップさせることによる効率改善はノード間通信が占める割合に依存し，メモリバスの性能の影響を受けることが分かる．

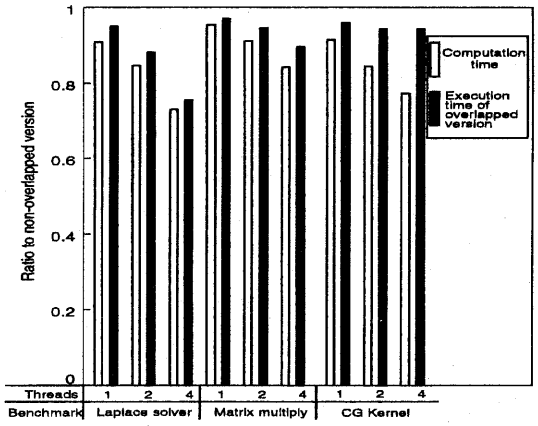

図8　オーバーラップさせない場合の実行時間に対する，計算時間とオーバーラップさせた場合の実行時間の割合(8ノード，1,2,4スレッド)．

ノード間通信の割合:

完全に通信がオーバーラップして隠蔽された場合の実行時間は計算時間で示される値であることは明らかである．つまり，オーバーラップさせた場合の理想的な性能改善の割合は通信時間の割合ということになる．例えば通信時間が実行時間の5%しか占めないようなアプリケーションでは，通信と計算をオーバーラップさせても最大で5%しか効率が改善されない．逆に通信時間が実行時間の30%を占めるようなアプリケーションでは，オーバーラップさせることにより最大で30%の効率改善が期待される．ラプラス方程式の陽解法においては，1スレッドの場合は計算時間が約90%であるのに対し，4スレッドの場合は約73%となる．ラプラス方程式の陽解法においては通信時間をほとんど隠蔽することができるため，通信と計算をオーバーラップさせることによって4スレッドの場合で約24%実行時間が短縮される．これは，通信コストの約93%が隠蔽されていることを意味する．行列の乗算は分散プログラミングのアルゴリズムとしてCannonのアルゴリズムを用いており，行列を4×4のブロックに分割して，各ノードに2ブロックずつ割り当てる方法を採用している．そのためノード間通信は全部で4回発生するが，最初の計算を行なう前にまず1回ノード間通信を行なう必要がある．今回の実装ではこの部分に関しては計算とオーバーラップさせていないため，最大でも全通信時間の75%しか隠蔽することができない．実験結果によると通信時間の約70%が隠蔽されていることが分かるが，実装を工夫して全通信時間を隠蔽することができるように改善する余地がある．いずれにせよ，SMPノード内で複数のスレッドを用いて計算時

間が短縮されると，通信時間が占める割合が増加し，オーバーラップによる効果が増加することになる．

メモリバスボトルネック:

CG カーネルの場合はオーバーラップさせることによって効率が改善されない．プログラミングの見地においては通信時間はほとんど隠蔽されるはずであるが，結果として実行時間が短縮されていない．NICAM はホストプロセッサおよびプロセッサバスを介さずにデータ転送を行なうが，DMA エンジンがデータ転送の際にメモリバスを使用する．CG カーネルの計算のほとんどは行列とベクタの乗算に費やされるが，この処理はメモリバスに対して高いスループットを必要とし，複数のスレッドを起動してもメモリバスの性能がボトルネックとなり，高い効率が得られないものである．この処理に通信をオーバーラップさせた場合，データ転送がメモリバスの負荷をさらに増加させ，オーバーラップさせない場合に比べて計算時間が増加させてしまうのが，効率改善が見られない原因であると考えられる．

オーバーラップさせることによる効率改善はノード間通信の割合に依存し，ノード数およびスレッド数が増えるとその効果が増加する．しかし，その効果はメモリバスの性能に制限される場合もある．

6. 関連研究

SMP クラスタ上でのプログラミングモデル，プログラミング方法に関していくつかの報告がなされている．Bader は SMP クラスタ上での SIMPLE と呼ばれるプログラミング方法を提案している[1]．彼らは共有メモリと分散メモリを融合した環境を有効に利用したコレクティブ通信のプリミティブのためのカーネルを提供している．ノード内のプロセスおよびクラスタシステム全体に対する通信プリミティブを提供しており，基本的な考えは我々の共有/分散融合プログラミングと同じである．Fink は SMP クラスタ上で通信と計算をオーバーラップさせる技法を提案している[3]．彼らのアプローチは各スレッドに対して非均等なデータ割り当てを行なうことにより，計算と通信をあわせた処理が均一に各スレッドに分配されるようにするというものである．具体的には，通信に参加するスレッドに対しては参加しないスレッドに比べて少ないデータを割り当てる．通信を行なうスレッドは行なわないスレッドに比べて早く計算が終了し，通信を行なわないスレッドが計算をしている最中にノード間通信を行なう．この方法における n ノードで p プロセッサの SMP クラスタ上においては，通信コストは最大で $\frac{p-1}{p}$ 削減される．しかし，彼らのアプローチはノード間通信に参加するスレッドを特別に扱うというものであり，プログラムコードがスレッドに対して均一ではなくなる．また，彼らは最適な分割方法を発見するための手法を提案してはいるが，実際には実験的に最適な分割方法を見つけ出している．我々の手法はスレッドに対して均一なプログラムコードであり，全スレッドが通信に参加するようなプログラムに対しても適用することができ，データ転送のための関数呼び出しのコストを除くほとんどの通信コストを隠蔽することが可能である．

7. 結　論

SMP ノードの性能が増加するに従い，ノード間通信のコストをいかに削減するかが重要になる．本稿ではノード間通信の時間を短縮するのではなく，ノード間通信を計算とオーバーラップさせることにより，ノード間通信を隠蔽する手法を採用した．我々のオーバーラップ技法はいくつかの典型的な繰り返し処理を行なうデータ並列プログラムに対して適用可能であり，計算対象となる領域とリモートノードからのデータを受信する領域とが別々にとられている場合にはノード間で同期をとる必要もなくなる．我々の方法ではノード間通信のコストのほとんどを隠蔽することが可能であり，実験ではラプラス方程式の陽解法において，8 ノード，4 スレッドの場合に最大でノード間通信の約 93%が隠蔽され，実行時間の約 24%が短縮された．データ転送の際にメモリバスへの負荷が増加し，CG カーネルのように計算の際に主記憶に対する高いバンド幅を必要とするアプリケーションに対しては効率改善は得られなかった．我々のオーバーラップ技法はノード数やスレッド数が増えるほど，大きな効果を発揮すると期待される．

参考文献

1) David A. Bader et.al., "SIMPLE: A Methodology for Programming High Performance Algorithms on Clusters of Symmetric Multiprocessors (SMPs)", UMIACS Technical Report 97-48, May 1997.
2) S. J. Fink et.al., "Non-Uniform Partitioning for Finite Difference Methods Running on SMP Clusters", http://now.cs.berkeley.edu/clumps/.
3) 松田元彦 他, "SMP クラスタ向けネットワーク・インタフェース上 AM 通信", 情報処理学会計算機アーキテクチャ研究会報告, Vol. 97, No. 76, pp. 55–60 (1997).
4) Y. Tanaka et.al., "COMPaS: A Pentium Pro PC-based SMP Cluster and its Experience", IPPS Workshop on PC-NOW'98, LNCS, Vol. 1388, pp. 486–497, Springer-Verlag (1998).
5) 田中良夫 他, "COMPaS: Pentium Pro を用いた SMP クラスタとその評価", JSPP'98, pp. 343–350 (1998).