分散メモリコンピュータにおける通信レイテンシ最小化手法

若谷彰良†

分散メモリ計算機における通信レイテンシの最小化手法について述べる．通信は計算に隠蔽することによってその通信レイテンシを最小化することができる．この手法をメッセージストリップマイニングと呼ぶ．本スキームにおける最適なブロックサイズおよびそのときのスピードアップも解析的に求められることを示し，配列再分散および配列の間接アクセスに適用した場合の効果について明らかにする．また，本スキームが既存のHPFタイプのコンパイラに容易に組み込めることを示す．

A New Approach to Reduce Communication Latency for Distributed Memory Multicomputers

AKIYOSHI WAKATANI†

This paper describes a new technique to reduce communication latency for distributed memory multicomputers. In order to hide communication behind computation, we introduce an optimization scheme, *message strip-mining*. By using this scheme, the communication overhead is almost completely overlapped with the subsequent computation. We also show that this scheme is easily implemented with HPF-type compilers, such as FORTRAN-D compiler.

1. はじめに

分散メモリマルチコンピュータが次世代スーパーコンピュータとなるためには，コンパイラの最適化能力がキーとなっている．なかでも，分散メモリマルチコンピュータ上で必要不可欠であるプロセッサ間通信コストをいかに小さくするかが重要である．しかし，従来からの通信最適化技術には必ずしも汎用性がなく，適用範囲が限られていた．

プロセッサ間通信のコストは，通信起動と通信レイテンシからなり，この通信レイテンシを削減する方法として，反復組替え法 (iteration reordering)[1]やベクタメッセージパイプライン法 (vector message pipelining)[7),8)]等がある．しかし，これらの方法は通信を計算に隠蔽することを基本としており，通信パターンがある制約を満たす場合にのみ適用される方法である．

既存の通信レイテンシ最小化手法が適用しにくい例として，2次元の拡散方程式に対するADI法[6]について考える．基礎となる偏微分方程式は次のようなものである．

$$\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{1}$$

ADIの基本的な考えは，時間ステップを $\frac{\Delta t}{2}$ の2つのステップに分け，それぞれのステップにおいて1次元の簡単な3重対角行列を解くようにすることである．したがってもう一方の次元に関してはまったく依存関係がないので，並列化は容易である[5]．

$$u_{j,l}^{n+\frac{1}{2}} = u_{j,l}^{n} + \frac{1}{2}\alpha\left(\delta_x^2 u_{j,l}^{n+\frac{1}{2}} + \delta_y^2 u_{j,l}^{n}\right) \tag{2}$$

$$u_{j,l}^{n+1} = u_{j,l}^{n+\frac{1}{2}} + \frac{1}{2}\alpha\left(\delta_x^2 u_{j,l}^{n+\frac{1}{2}} + \delta_y^2 u_{j,l}^{n+1}\right) \tag{3}$$

$$\alpha \equiv \frac{D\Delta t}{\Delta^2} \tag{4}$$

$$\Delta \equiv \Delta x = \Delta y \tag{5}$$

上式において，δ は差分演算子を表し，δ_x^2 は x 方向の2階差分，δ_y^2 は y 方向の2階差分を表す．ADI法を分散メモリコンピュータに効果的に実装するために，u に対する配列 u を2種類の方向に分散させる必要がある．最初のステップ（$n+\frac{1}{2}$ step）に対して，u は (*, BLOCK) で分散される．というのは，最初のステップは3重対角行列を第1の次元で解くものであるからである．言い換えると，u(j,l) は u(k,l) $(\forall k)$ を用いて解かれる．この計算は第2の次元において並列に行える．これに反して，第2ステップは第2の

† 松下電器中央研究所
Central Research Laboratories, Matsushita Electric Industrial, Co., Ltd.

次元に関する3重対角行列を解くものなので，配列 u は (BLOCK, *) で分散される．よって，すべての時間ステップにおいて分散された配列の再分散が必要となる．配列の再分散は本質的にはコストの大きい集合通信（collective communication）であるので，そのような通信のコストを最小化することは重要である．

しかし，反復組替え法およびメッセージパイプライニング法はこのケースには効果的には適用できない．というのは，ループのすべての計算でリモートデータが必要であり，通信を隠すべき計算部分がないからである．よって，大きな通信部分は残る．

このように，既存技術ではカバーしきれない応用分野も多い．しかし，そのための通信最適化技術は，標準プログラミング言語の普及には欠かせないものである．

そこで，本論文では，通信と計算をオーバラップしてレイテンシを隠蔽するために，通信と計算を複数の単位に分割し，依存関係に従って通信と計算をスキューイングする手法について述べる．この通信レイテンシの削減手法は，規則性の高い集団通信（collective communication）を含む広い範囲の一般のプログラムに適用可能であり，実効性能の改善が期待できるものである．

以降の章において，通信レイテンシそのものを計算に隠蔽する方法，メッセージストリップマイニング法（message strip-mining）の構成を述べ，通信をオーバーラップする計算部分の決定方法および並列化すべき配列の次元と分散配置されている次元の関係について言及する．また，最小通信単位をメッセージ集団化手法でブロック化する際の最適ブロックサイズおよび通信スピードアップについて検討する．さらに，メッセージストリップマイニング法の配列再分散および配列の間接アクセスの通信への適用した場合の効果を明かにする．最後に，メッセージストリップマイニング法のコンパイラへの実装方法についてその概要を述べ，従来の FORTRAN D コンパイラ等の HPF 系コンパイラへの実装が容易であることを示す．

2. メッセージストリップマイニング法 (message strip-mining)

配列再分散の通信の場合は，文献11) で述べたように，概念プロセッサを物理プロセッサに効果的にマッピングする手法（螺旋配置）によって，再分散が必要とする大域通信を近接通信に変換できた．しかし，通信時間自体は陽に残ったままである．

そこで，計算および通信において多次元配列の1次元部分を最小単位として分割し，その最小単位の通信と計算部分とがオーバーラップできるアプリケーションに対しては，その配列再分散の通信レイテンシが隠蔽できることを示すものである．これをメッセージストリップマイニング法（message strip-mining）と呼ぶ[10]．

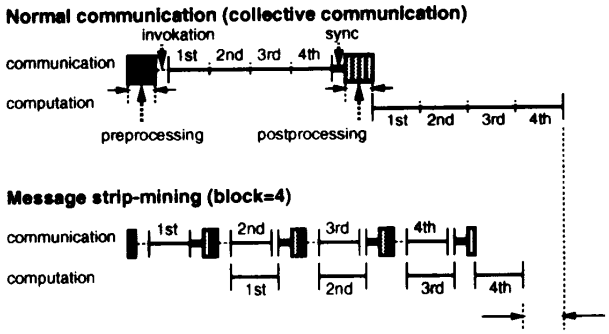


図1 従来の（collective）communication とメッセージストリップマイニング法のタイムチャート
Fig. 1 Typical time-charts of normal (collective) communication and message strip-mining.

さらに，このメッセージストリップマイニング法は，配列再分散だけでなく規則性の高い通信を持つ任意のアプリケーションに対して適用できるものである．次節において，一般的なアプリケーションに対するメッセージストリップマイニング法の構成に関して述べ，適用例として配列再分散のほかに配列の間接アクセスの通信に応用した場合を紹介する．間接アクセスは，本来，不規則通信であるが，inspector/executor 法を用いると，executor フェーズにおける通信は規則性の高い通信に変更可能であり，メッセージストリップマイニング法を適用することが可能となる[9]．

2.1 基本構成

まず計算部分は，いくつかの最小部分に分割できるとする．前章で扱った ADI では，多次元配列の連続する1次元部分が最小単位として扱われた．この最小単位に対応する通信単位を1つのメッセージと考える．

受信したデータはその後の計算において使用されると仮定する．すなわち，i 番目のメッセージで受信されたデータは，i 番目の計算部分で使用されるとする．さらに，受信する通信単位☆の数およびループの繰り返し回数を N とする．

図1の上に，典型的な通信および計算のタイムチャートを示してある．

通信モデルは，前処理（preprocessing），通信起動（invoking），通信（traveling），同期（synch），後処理

☆ 1つの通信単位は1つ以上のデータからなる．

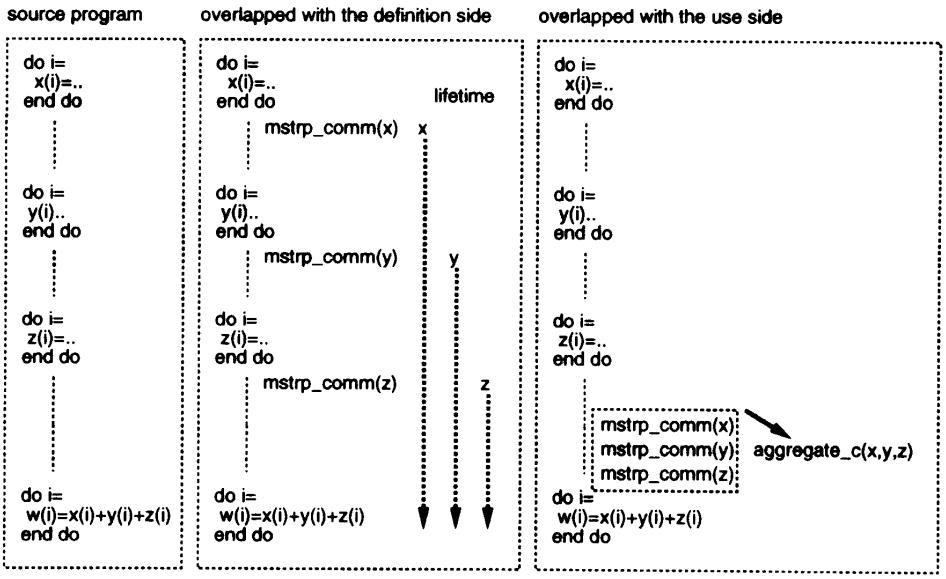


図2 メッセージストリップマイニングの実現方法
Fig. 2 Two alternatives to implement message strip-mining.

(post-processing) の5つの部分からなる.

なお, これらのフェイズのうちで, 通信フェイズだけが計算とオーバーラップできる. というのは他のフェイズは通信プロセッサではなく計算プロセッサによって実行されるからである. しかし, 通信フェイズは他のフェイズに比べてかなり大きいと考えられる.

計算とオーバーラップさせるために, 通信を最小の単位に分割するが, その最小単位をいくつかまとめて集団化 (aggregation)[8]する. したがって, ストリップマイニングのために, N 個の通信単位からなる通信全体を複数の通信単位からなる B 個のブロックに分けるとする. すなわち, それぞれ通信ブロックには N/B 個の個別の通信を含んでいるので, i 番目の通信ブロックには $((i-1)\times\frac{N}{B}+j)$ 番の個別通信が含まれている $(0\leq j\leq\frac{N}{B}-1)$. 同様に, ループも B 個のブロックに分ける. それぞれの計算ブロックには対応する通信ブロックで受信したデータを参照するような計算が含まれている. なお, B が1の場合は, それは従来方式の通信と同じである.

i 番目の計算ブロックを開始する前に, i 番目の通信ブロックは完了していなければならない. しかし, $(i-1)$ 番目の計算ブロックは i 番目の通信ブロックと並列に実行できる. つまり, i 番目の通信ブロックを $(i-1)$ 番目の計算ブロックとオーバーラップさせることにより, 通信オーバーヘッドはほとんど隠蔽できる.

2.2 基本構成に対する考察

2.2.1 通信を隠蔽する計算ループの選択

通信を計算とオーバーラップする方法として2つ考えられる. 1つは受信データを使用している計算部分と通信をオーバーラップする方法で, もう1つは送信データを定義する計算部分と通信をオーバーラップする方法である. それぞれの長所短所を次に示す.

- 通信をデータの使用部分とオーバーラップする場合, 他の通信と集団化できる可能性がある. それによりメッセージの数を減らすことができる. しかし, 十分な計算がない場合, 通信オーバーヘッドは残る.
- 一方, 定義側と通信をオーバーラップする場合は, 使用側に十分な計算は必要ない. しかし, 受信データをバッファに保持している時間 (lifetime) は長くなり, より多くのメモリを必要とする.

これらを**図2**の例を用いて説明する. 配列 `x`, `y`, `z` は最後のループで使用されている. これらは配列 `w` と align されてなく, 通信が必要であると仮定する. 定義側でオーバーラップした場合, 配列 `x` は第2, 第3のループで使用されることなくバッファに格納したままにしなければならない. これに対して, 使用側でオーバーラップした場合は, 使用される直前で通信されるので, バッファのためのメモリを少なくすることができる. さらに, 配列 `x`, `y`, `z` が互いに align されていたら, これらのメッセージは1つのメッセージに集団化でき, 通信起動コストを低減できる.

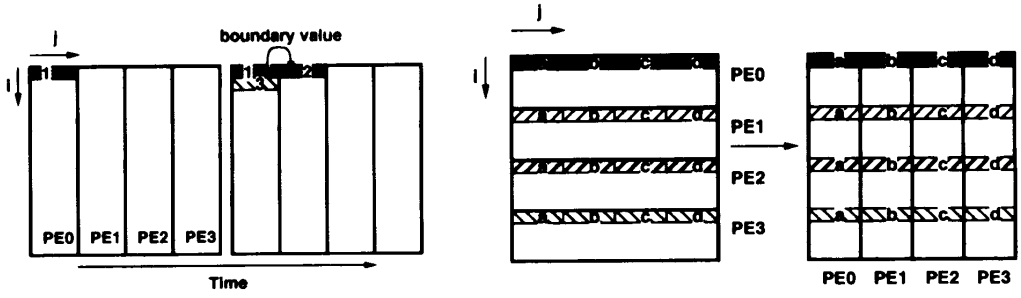
これ以降の議論においては, つねに使用側計算部分に十分な計算量があると仮定し, 使用側でオーバーラップするとする.

2.2.2 並列化ループとデータ分散

ここまでは, 最外側ループが並列化され, かつ, そ

表1 分散次元と並列化次元
Table 1 Distributed dimension and parallelized dimension.

Outer loop is parallelized		Inner loop is parallelized	
Outer loop is distributed	Inner loop is distributed	Outer loop is distributed	Inner loop is distributed
プログラム 1 !hpf redistribute + (*,block):: A do j=1,100 do i=1,100 A(i,j)= + A(i-1,j)+A(i,j) end do end do	プログラム 2 !hpf redistribute + (*,block):: A do i=1,100 do j=1,100 A(i,j)= + A(i,j-1)+A(i,j) end do end do	プログラム 3 !hpf redistribute + (*,block):: A do j=1,100 do i=1,100 A(i,j)= + A(i,j-1)+A(i,j) end do end do	プログラム 4 !hpf redistribute + (*,block):: A do i=1,100 do j=1,100 A(i,j)= + A(i-1,j)+A(i,j) end do end do



(a) 従来方式 (b) 遅延メッセージストリップマイニング法

図3 内側ループの並列化および外側ループの分散
Fig. 3 Parallelized inner loop and distributedouter loop.

のループの次元が配列の分散している次元に一致しているような場合だけを扱ってきた．しかし，他のケースも実アプリケーションにおいては存在する．つまり，外側ループが並列化されない，もしくは，分散次元と一致しない場合である．そのような場合に対処するために，ループ置換 (loop interchanging)[12]および遅延通信 (delayed communication) を用いる．表1に示されるような2次元配列を用いたプログラムを考えよう．配列 A は当初 (BLOCK,*) で分散されていると仮定する．

プログラム1は2.1節で扱った典型的なループで，配列の並列化すべき次元と分散されている次元が一致するものである．よって，メッセージストリップマイニング法は容易に適用できる．

プログラム2では，内側ループに関して依存関係 (true dependence) があるので，外側ループが並列化されるべきであるが，配列 A の第2の次元が分散されている．したがって，第2の次元を DOACROSS 型で並列化する必要がある．

図3の左図に4プロセッサの場合の実行シーケンスを示す．まず，プロセッサ0がインデックス (i=1, j=1:25) に関する計算を行い，配列 A の部分配列 (i=1, j=26:50) をプロセッサ1に送る．その後，プロセッサ0は，境界値 (i=1, j=25) をプロセッサ1に送る．次の時間スロットで，プロセッサ0はインデックス (i=2, j=1:25) の計算を行い，部分配列 (i=1, j=51:76) をプロセッサ2に送り，部分配列 (i=2, j=26:50) をプロセッサ1に送る．同時にプロセッサ1はインデックス (i=1, j=26:50) の計算を行う．

しかしながら，この計算スキームはプログラム1よりも多くの通信を必要とする．というのは，それぞれの通信は多対多ではなく1対多通信であり，また，隣接プロセッサへの境界値の通信のために追加の点対点通信が必要である．さらに，同期をとらなければいけないポイントも増える．それゆえ，プログラム2に，直接，メッセージストリップマイニング法を適用すれば，性能劣化を招く．

通信オーバーヘッドおよび同期ポイントを軽減するために，通信と計算の順序を入れ替える方法 (delayed communication) を用いる．すなわち，計算は分散前の位置で行い，その後配列を分散するのである．なお，ここでは，再分散ディレクティブの意味合いを若干拡

大している．

前の処理と同様に，4プロセッサを用いた実行シーケンスについて説明する．pとqをプロセッサ番号(p, q=0..3)とする．すべてのプロセッサpは，第1の部分計算(i=1+p*25, j=1:100)を通信なしに行う．その後，第2の部分計算(i=2+p*25, j=1:100)を行いながら，それぞれのプロセッサから部分配列(i=1+p*25, j=1+25*q:25*(q+1))をプロセッサqに送信する．この通信は多対多である．この通信を図3の右図で示す．このスキームは計算のあとに通信起動を行っている点を除けばプログラム1に適応したメッセージストリップマイニング法に似ている．よって，プログラム2 (delayed communication) に適用したメッセージストリップマイニング法の有効性はいままでの議論から容易に引き出せる．

ループ置換を用いるとプログラム3および4は容易にプログラム2および1に変換できるので，すべてのケースでメッセージストリップマイニング法が効果的に適用できることが分かる．

3. 最適ブロックサイズと理想スピードアップ値

この節では，通信単位の最適粒度の観点より通信コストを最も小さくするブロックサイズを求め，そのサイズのメッセージストリップマイニング法によって得られる通信時間のスピードアップについて考察する．

$nproc$ をメッセージを交換しあうプロセッサの数とする．この値は通常 (全プロセッサ数 -1) である．また，α をメッセージあたりの通信起動と同期処理の時間の合計とし，β を通信単位あたりの通信レイテンシとする．さらに，δ を通信単位あたりの前処理および後処理の時間とし，γ を通信単位あたりの計算時間とする．従来の通信と計算のコストは次のように見積もれる．

$$T_{comm} = nproc \times \alpha + N \times (\beta + \delta) \quad (6)$$

$$T_{comp} = N \times \gamma \quad (7)$$

$$T_{total} = nproc \times \alpha + N \times (\beta + \delta + \gamma) \quad (8)$$

メッセージストリップマイニング法を適用した場合，トータルコストは，1番目の通信ブロックの時間，i $(2 \le i \le B)$ 番目の通信ブロックの起動/同期の時間，i $(1 \le i \le N)$ 番目の通信単位に対する前処理，後処理，および計算の時間である．

$$T_{st} = \left(\frac{N \times \beta}{B} + nproc \times \alpha\right) + (B-1) \times nproc \times \alpha + N \times (\delta + \gamma) \quad (9)$$

式(8)および(9)から，もし，次の条件を満たす整数 B (>1) が存在すれば，T_{st} は T_{total} よりも小さい．

$$B < \frac{N \times \beta}{nproc \times \alpha} \quad (10)$$

つまり，適当なブロック数をとることにより，メッセージストリップマイニング法は十分大きい N に対して有効である．

次に，ストリップマイニング法の通信コストを最小にする最適なブロック数 B_{opt} を求める．B_{opt} は T_{st} を B に関して微分することによって，次のように表される．

$$\frac{dT_{st}}{dB} = nproc \times \alpha - \frac{N \times \beta}{B^2} = 0$$

$$\Rightarrow B_{real} = \left(\frac{N \times \beta}{nproc \times \alpha}\right)^{\frac{1}{2}} \quad (11)$$

B_{up} を B_{real} よりも大きいか等しい最小の整数とし，B_{low} を B_{real} よりも小さいか等しい最大の整数とする．$(T_{st}|_{B=B_{up}})$ が $(T_{st}|_{B=B_{low}})$ より大きい場合は，B_{opt} として B_{low} を選び，そうでない場合は B_{up} を選ぶ．

さらに，B_{opt} が B_{real} に等しく，α や β に比べて δ が無視でき，B_{opt} 個のブロックに通信を分けた場合，通信理想スピードアップは次のように容易に計算できる．

$$T_{comm\_st} \equiv T_{st} - T_{comp} \quad (12)$$

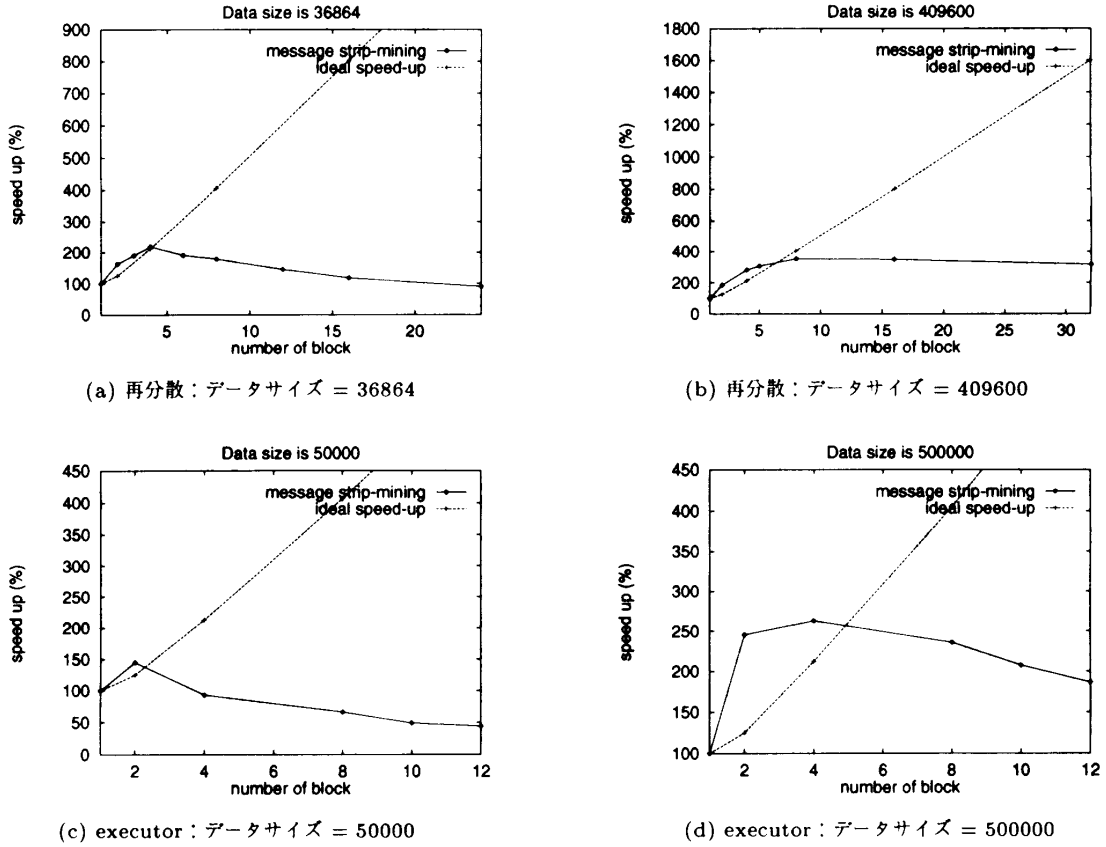
$$\Rightarrow ideal\_speedup = \left.\frac{T_{comm}}{T_{comm\_st}}\right|_{B=B_{opt}} = \frac{1 + B_{opt}^2}{2B_{opt}} \quad (13)$$

ここで，T_{comm_st} は計算に隠れなかった通信時間である．この評価によると，B_{opt} が2の場合，スピードアップは125%であり，B_{opt} が4の場合は，スピードアップは212.5%であり，B_{opt} が8の場合は，スピードアップは406.3%である．

4. 実　　験

本方式の効果を例証するために，iPSC/860上に実装した．特に，ブロック数が通信時間にどう影響するかについて調べる．前述した評価から分かるように，ブロック数の最適値が存在し，最適値より大きいもしくは小さいブロック数の場合は通信時間が増加すると予想される．

ここでは配列の再分散および間接アクセスに対する実験を取りあげる．アプリケーションは特に限定していないが，通信コストを隠蔽するのに十分な演算量がある計算とし，ノンブロッキング通信の通信本体は計



(a) 再分散：データサイズ = 36864

(b) 再分散：データサイズ = 409600

(c) executor：データサイズ = 50000

(d) executor：データサイズ = 500000

図4 16ノードiPSC/860上での実験結果時間（sec）

Fig. 4 Communication time in seconds for redistribution and executor on an 16-node iPSC/860.

算にオーバーラップできていると仮定する．

4.1 配列の再分散

16ノードのiPSC/860上で2次元配列の再分散をブロッキング通信と非ブロッキング通信を用いて実装した．配列は(Block,*)から(*,Block)へ再分散する．タイミングは**dclock**組込み関数で計測する．

実験の結果を図4(a)，(b)に示す．ここで，**data size**は分散配列のローカルなサイズを示す．つまり，配列のトータルサイズは**data size**の16倍である．縦軸には通信時間のスピードアップ値を示している．スピードアップとは（従来方法の場合の通信時間/計算に隠蔽できなかった通信時間）で定義されるものである．また，Ideal speed-upは式(13)で表されるもので，そのブロック数が最適ブロック数であった場合の理想的なスピードアップ値を示す．なお，横軸の左端は従来方法の再分散方式を示す．

予想されるように，それぞれの場合に対してブロック数の最適値が存在する．データサイズが409600の場合はブロック数8，データサイズが36864の場合はブロック数4が最適である．スピードアップはそれぞれ354%，219.1%である．最適ブロック数が8のときに理想スピードアップは406.3%であり，データサイズが409600の場合はそれに近い値になっている．一方，ブロック数が4のときの理想スピードアップは212.5%と予想されたが，計測されたスピードアップは理想スピードアップよりも高い．その理由は再分散の従来方法がより多くのネットワークコンテンションを産み出していることに加えて，メッセージストリップマイニング法ではキャッシュメモリのヒット率が向上しているからであると考えられる．というのは，メッセージストリップマイニング法で使用されているメッセージの長さが従来方法よりも短いからである．

4.2 間接アクセスに対するexecutorアルゴリズム

分散メモリシステムにおいては，分散された配列への間接アクセスは，しばしば，inspector/executor法[2)~4)]で実現されている．この手法は2つの部分からなる．前半（inspector）は，間接アクセスが通信を生成するかどうかを判定し，さらに通信すべき相手先，受け取り先のプロセッサを決定する．また，バッファエリアを獲得し，グローバルインデックスをローカルインデックスに変換し，executorのための通信スケジュールを生成する．後半（executor）は，そのスケジュールに従って，他のプロセッサとメッセージを交換するものである．

8 ノードの iPSC/860 上で executor プログラムを NX-2 ライブラリを用いて実現した．通信するデータサイズとしては，5×10^4 と 5×10^5 を用いた．なお，それぞれのメッセージの送信先は実行時にランダムに与えた．よって，すべてのデータが通信されたわけではないが，$\frac{15}{16}$ $(=\frac{\text{プロセッサ数}-1}{\text{プロセッサ数}})$ の確率で通信が発生したと考えられる．

実験結果を図 4 に示す．なお，前の実験と同様に，横軸の左端は従来の executor を示す．

図 4 (c)，(d) が示すように，最大のスピードアップはデータサイズが 5×10^4，5×10^5 のとき，それぞれ，ブロック数が 2，4 の時に得られる．また，そのときのスピードアップはそれぞれ 144.9%および 262.9%で，これらは前の実験と同様に，式 (13) で得られる理想スピードアップよりも高い．理由も同様で，メッセージの長さが短くなったことによる効果と考えられる．

さらに，これらのスピードアップのプロット図は，再分散の場合よりもギザギザになっている．これは，executor の通信が不規則性を含んでおり，その通信パターンの変化が規則通信 (regular communication) よりも激しいからである．

5. コンパイラへの実装

ここまでの章でメッセージストリップマイニング法による通信コストの削減の効果が確かめられた．ここでは，実際のコンパイラにおいてメッセージストリップマイニング法を組み込むための概要をプログラム例を用いて述べた後，一般的な適用方法を表す．

5.1 コンパイラ動作例

この節では HPF プログラムを 4 プロセッサの並列コンピュータ用 SPMD FORTRAN プログラムにコンパイルする場合を考える．ここでは FORTRAN D コンパイラ[8]でのコンパイラ手法に従う．

FORTRAN D コンパイラでは，ループに関する依存性に従って生成する通信メッセージを 3 種類に分類している．ループに関して依存性のないメッセージを independent と呼び，ループの前にメッセージを配置する．ループに依存するメッセージのうち，その依存するループが全プロセッサで実行される場合，そのメッセージを carried-all と呼び，依存のあるループのループヘッダ内にメッセージを配置する．また，依存するループがプロセッサ間に分散されている場合は，そのメッセージを carried-part と呼び，ループの前後に send と receive のメッセージを配置する．たとえば下記の例を考える．

```
do k=1,M
  do i=1,N
   A(i) = B(i+1)
        + A(i+1)
        + A(i-1)
  enddo
enddo
```

分散されている配列 A，B が互いに align され，ループ i が分散された場合，オペランドになっている配列要素 B(i+1)，A(i-1)，A(i+1) のいずれも通信が必要となる．B(i+1) はいずれのループにも依存しないので (independent)，ループ k の前で通信が行われ，A(i+1) はループ k に依存しているがそのループは分散されていないので (carried-all)，ループ k のループボディの最初で通信が行われ，A(i-1) はループ i に依存しており，そのループは分散されている (carried-part)，ループ i の前後に通信が配置される．

さて，下記の HPF プログラムを 4 プロセッサの並列コンピュータ用 SPMD FORTRAN プログラムにコンパイルする場合を考える．

```
!hpf$ distribute a(block)
!hpf$ distribute b(cyclic)
      :
do i=1,6400
 a(i)=func(b(i))
end do
```

このプログラムにおいて配列 a,b は align されていないので do ループの計算においては通信が必要となる．

FORTRAN D コンパイラで用いられている通信メッセージ生成手法により，このプログラムは集団通信を用いた下記に示す SPMD プログラムにコンパイル可能である．

```
send (b,1:1600)
receive (tmp,1:1600)
do i=1,1600
 a(i)=func(tmp(i))
end do
```

上記のプログラムの通信はループ i に関してデータ依存がないので，independent メッセージとなり，ループの前に通信が配置されている．なお，send，receive は集団通信を示すマクロ命令である．

この independent メッセージにメッセージストリップマイニング法を適用するために，最適ブロックサイズを式 (11) を用いて決定する．

$nproc$ は 3, N は 1200☆である. α, β はハードウェアに依存するが, ここでは $\alpha = 25 \times \beta$ と仮定する.

$$B = irint\left(\left(\frac{N \times \beta}{\alpha \times nproc}\right)^{\frac{1}{2}}\right) = irint\left(\left(\frac{1200}{25 \times 3}\right)^{\frac{1}{2}}\right) = 4 \quad (14)$$

よって, 通信および計算を 4 つのブロックに分割する.

```
      do iB=1,4
       send (b,1+(iB-1)*400:iB*400)
       receive (tmp,1+(iB-1)*400:iB*400)
      end do
      do iB=1,4
       do i=1+(iB-1)*400,iB*400
        a(i)=func(tmp(i))
       end do
      end do
```

4 個の通信のうち, 最初の通信はブロッキング通信とする. 2 番目以降の通信をノンブロッキング通信とし, 計算とスキューイングする.

```
      send (b,1:400)
      receive (tmp,1:400)
      do iB=1,4
       if(iB.not.4)
c       send (b,1+iB*400:(iB+1)*400)
       do i=1+(iB-1)*400,iB*400
        a(i)=func(tmp(i))
       end do
       if(iB.not.4)
c       receive (tmp,1+iB*400:(iB+1)*400)
      end do
```

なお, 実プロセッサ等パラメータの一部がコンパイル時に不明の場合も, 最適ブロックサイズを実行時に計算することにより, 同様にメッセージストリップマイニング法を用いてコンパイル可能である.

以上のように, メッセージストリップマイニング法により通信時間の削減を図るコンパイリングができる. この効果は式 (13) により,

$$\frac{1 + B_{opt}^2}{2 \times B_{opt}} = \frac{1 + 4^2}{2 \times 4} = 2.13$$

となり, 理想的には約 2.13 倍の通信のスピードアップが得られると予想される.

☆ 補助配列 tmp のサイズは 1600 であるが, そのソース (配列 b) の 1/4 は自プロセッサが保持しているので通信の必要がない. したがって, 通信サイズは 1200 である.

```
if(explicit な集団通信がある){/* (1) */
    - receive される配列が最初に使用される do
      ループの直前に通信を移動する
    - メッセージストリップマイニング法を適用
}
else{
  if(参照代入関係より通信が生成される){/* (2) */
    switch(メッセージタイプ){
      case independent:/* (3) */
        if(予想される通信コスト<一定値){/* (4) */
          - 最初に使用される do ループの前に通信
            を移動し, そのループより外側のルー
            プが初期値の時のみ通信起動されるように
            if 文で囲む
          - メッセージストリップマイニング法を適用
        }
        break;
      case carried-all:/* (5) */
       - メッセージストリップマイニング法を適用
       break;
      case carried-part:/* (6) */
       loop 交換や配列の再分散を行い, carried-part
       部を解消し, 最初から適用し直す.
       break;
    }
  }
}
```

図 5 メッセージストリップマイニング法の一般的適用手順
Fig. 5 Generalized scheme of message strip-mining.

5.2 コンパイラ実装の手順

前節の例以外のプログラムの場合も同様に適用可能である. すなわち, 集団通信のような規則的な通信が発生するプログラムに対して, 上記の方法でコンパイラで処理ができる. 生成されるメッセージタイプの違いによるメッセージストリップマイニング法の一般的適用手順を図 5 に示す.

まず, 通信が陽に集団通信である場合 (1), たとえば配列の再分散通信のような場合, メッセージをオーバーラップできる do ループの直前に移動し, その計算にメッセージストリップマイニング法を適用する. また, 通信が陽に現れていないが, 分散されている配列を参照するなどしてコンパイラによってメッセージが生成される場合 (2) には, そのメッセージのタイプによって処理を変える.

independent の場合 (3) は依存するループがないので, 最外側のループの外側に通信が配置されている. したがって, 全計算時間から見た場合, コストが必ずしも大きくないので, まず予想される通信コストを計算し, それがある一定値以上の場合 (4) のみメッセージストリップマイニング法を行うことを考える. その

理由は，メッセージストリップマイニング法を適用するために通信を内側のループ内に移動する場合があり，各種のオーバーヘッドを増加させてしまう可能性があるからである．移動した通信に対してストリップマイニング法を適用するが，内側のループに対しては依存関係がないメッセージなので，通信起動が1回だけになるように通信起動をif文で囲まねばならない．

carried-allの場合(5)，メッセージストリップマイニング法を適用すべきdoループは，通信が配置された位置の直後のdoループであるので，その位置のまま適用する．最後に，carried-partの場合(6)，そのままではメッセージストリップマイニング法は適用できない．したがって，loop interchangeや配列再分散を行って，分散すべきdoループを変更したり実行順序を変更してcarried-part部分を解消しなければならない．しかし，このような変更は他のdoループにも影響を及ぼすので，全体の通信コストをかんがみて合理的な処理判断が必要である．

なお，メッセージストリップマイニング法自体の適用方法は前述のとおり，1) 最適ブロックサイズを決定し，2) オーバーラップ先の計算部分および通信を分割し，3) 最後に通信をブロッキング通信/ノンブロッキング通信に分けて，2で分割した計算部分とオーバーラップさせることによって行う．

6. おわりに

次世代のスーパーコンピュータは分散メモリ型マルチコンピュータであるといわれており，すでにその範疇に含まれる製品も現れている．しかし，現状では必ずしも多数派とはなっておらず，そのための標準プログラミング環境の確立が急務であると考える．そのためにもコンパイラ通信最適化が重要である．コンパイラ実装の最大のポイントは，性能劣化の原因となるプロセッサ間通信時間の最小化をいかに行うかである．通信レイテンシを削減することにより，計算機の最大性能（ピーク性能）に近い実効性能を得ることができる．

本論文では，通信と計算を分割しオーバーラップする手法（メッセージストリップマイニング法）を述べた．さらに，規則性の大きい通信を持つ任意のアプリケーションにメッセージストリップマイニング法を適用できることを示し，iPSC/860を用いた実験も行った．主な結果を次に示す．

- メッセージストリップマイニング法において，通信時間を最小化する最適なブロックサイズを解析的に決定する手法を示し，その場合の理想スピードアップ値も示した．
- 任意のアプリケーション例として，配列の間接アクセスにメッセージストリップマイニング法を適用した場合，実験では，通信時間のスピードアップが260%を越えた．
- メッセージストリップマイニング法のコンパイラへの実装方法についてその概要を述べ，従来のFORTRAN Dコンパイラ等のHPF系コンパイラへの実装は一部の通信パターンを除き容易であること考えられる．

以上のように，本論文で述べた通信レイテンシの削減手法は，広い範囲の一般のプログラムに容易に適用可能であり，実効性能の改善が期待できる．今後はこのような手法を実用コンパイラ上に実現し，実アプリケーションによる評価が重要となる．

謝辞 本研究において多大なるご指導とご助言を賜わりました米国Oregon Graduate Institute教授Michael Wolfe博士ならびに有益なご助言を賜わりました京都大学工学部教授富田眞治博士および京都大学工学部助教授野木達夫博士に感謝の意を表します．また本研究の機会を与えていただいた松下電器中央研究所所長新田恒治博士ならびに有益なご助言をいただいた同研究所主幹研究員廉田浩博士に深謝いたします．

参考文献

1) Koelbel, C. and Mehrotra, P.: Programming Data Parallel Algorithms on Distributed Memory Machines Using Kali, *International Conference on Supercomputing*, 1991.
2) Koelbel, C.: Compile-Time Generation of Regular Communications Patterns, *Supercomputing '91*, 1991.
3) Koelbel, C., Mehrotra, P., Saltz, J. and Berryman, H.: Parallel Loops on Distributed Machines. *DMCC-5*, 1990.
4) Koelbel, C. and Mehrotra, P.: Compiling Global Name-Space Parallel Loop for Distributed Execution, *IEEE Tran. Parallel and Distributed Systems*, Vol.2, No.4, 1991.
5) Nogi, T.: Parallel Computation, *Patterns and Waves - Qualitative Analysis of Nonlinear Differential Equations*, pp.279–318, North-Holland, 1986.
6) Press, W.: *Numerical Recipes in C*, pp.665–666, Cambridge, 1988.
7) Rogers, A. and Pingali, K.: Process Decomposition through Locality of Reference, *Conf. on Program Language Design and Implementation*, 1989.
8) Tseng, C.-W.: An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines.

PhD Dissertation, Rice University, Jan. 1993
9) Wakatani, A. and Wolfe, M.: Effectiveness of Message Strip-Mining for Regular and Irregular Communication, *Proceedings of 7th Int'l Conf. on Parallel and Distributed Computing Systems*, 1994.
10) Wakatani, A. and Wolfe, M.: A New Approach to Array Redistribution: Strip Mining Redistribution, *Proceedings of Int'l Conf. on Parallel Architectures and Languages Europe*, 1994.
11) Wakatani, A. and Wolfe, M.: Optimization of the Redistribution of Arrays for Distributed Memory Multicomputers, *Parallel Computing*, Vol.21, pp.1495–1490, 1995.
12) Wolfe, M.: *Optimizaing Supercompilers for Supercomputers*, The MIT Press, 1989.

(平成7年 7 月31日受付)
(平成7年11月 2 日採録)

若谷 彰良（正会員）

昭和37年生．昭和61年京都大学大学院工学研究科数理工学専攻修士課程修了．同年松下電器（株）入社．並列マシンのソフトウエア研究開発に従事．平成4年から6年にかけて米国 Oregon Graduate Institute 客員研究員．工博．