通信機構に合わせた最適化を行う並列化コンパイラ

横田大輔[†] 千葉滋[†,††] 板野肯三[†]

分散メモリ型の並列計算機では，プロセッサ間の通信がボトルネックにならないように，典型的な通信パターンを高速化する特別なハードウェアを用意しているものがある．我々は並列計算機 CP-PACS/Pilot3（日立 SR2201）用に，HPF コンパイラのサブセットを開発し，対象計算機が持つ通信用ハードウェアを最大限に活用するコードを生成できるようにした．通信用ハードウェアを最適に利用するためには，プロセッサ間の通信パターンを正確に分析する必要がある．我々はインスペクタ・エグゼキュータを改良した方式を用いることで，これを可能にした．また，我々はベンチマーク・テストによって，開発したコンパイラが生成するコードが実際に高い実行効率を達成していることを確かめた．

A Parallelizing Compiler Optimizing for Communication Devices

Daisuke Yokota,[†] Shigeru Chiba[†,††] and Kozo Itano[†]

A number of parallel computers with distributed memory provide special hardware for boosting the speed of typical inter-processor communication. We have developed a compiler for the CP-PACS/Pilot3 parallel computers (Hitachi SR2201), which compiles a program in a subset of HPF into the code exploiting such special hardware as optimally as possible. To do this, the compiler uses our new technique based on the inspector-executor method so that it can examine the exact patterns of inter-processor communication. We have also measured the execution performance of benchmark programs compiled by our compiler.

1. はじめに

分散メモリ型の並列計算機が主流になってくるにつれ，プロセッサ間の通信機構がハードウェア設計の要点の 1 つになってきた．並列計算機の中には，典型的な通信パターンを高速化するハードウェアを用意して，プロセッサ間の通信が計算のボトルネックにならないようにしているものがある．たとえば，筑波大学の並列計算機 CP-PACS と Pilot3（日立 SR2201）[6] は RDMA（Remote DMA）[7] という通信機構を持つ．これは受信側のプロセッサのメモリに直接書き込む通信を可能にし，メモリコピーが発生せず高速な通信を実現する．さらに，ブロックストライドという単位で送信することでより高速に送信を行うことが可能である．制限付きながら不連続なメモリ空間のデータを 1 回の通信で送信することができる．

† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†† 科学技術振興事業団さきがけ研究 21
PREST, Japan Science Technology Corp.

しかしながら，このような通信機構が，実際のアプリケーションから効果的に使われているとは，必ずしもいえない．このような通信機構は，アプリケーションから直接利用するには低レベルでありすぎ，一方，自動並列化コンパイラはこのような通信機構をうまく活用できてこなかった．コンパイラがこのような通信機構をうまく活用するには通信パターンの正確な解析が必要であるが，これが容易ではないからである．

コンパイラによる静的な解析では，複雑なメモリ参照を行うプログラムの場合，通信パターンを正確に判断することが難しい．大まかな解析結果だけをもとにコンパイルしなければならないとすると，生成されたコードは，性能を犠牲にした冗長な通信を行いがちである．正確な解析結果なくては，計算機が持つ通信機構を最適化した形で利用するコードを生成することは不可能である．

本論文では，我々が開発した CP-PACS/Pilot3 用の HPF コンパイラのサブセットについて述べる．このコンパイラのために，我々はインスペクタ・エグゼキュータ方式[1] を拡張し，インスペクタ部で集められた通信パターンの情報をもとに，エグゼキュータ部が

RDMA を最大限に活用するように動的に判断しながら通信を行うようにした．また，インスペクタ部の実行後，エグゼキュータ部を最適化コンパイルし直すことで，通信速度をさらに改善する方法も開発した．

以下，2 章ではコンパイラによる通信機構の活用について述べ，3 章では我々が提案する方式による通信の最適化コンパイルについて述べる．4 章ではベンチマークによる我々のコンパイラの性能測定実験の結果を示し，5 章で関連研究を述べ，6 章で本論文をまとめる．

2. コンパイラによる通信機構の活用

分散メモリ型の並列計算機では，プロセッサ間の通信が計算のボトルネックにならないよう，ハードウェアに様々な工夫がこらされている．計算機の中には，多くのアプリケーションに見られる特定のパターンに合致する通信を高速化するために，そのような通信専用のハードウェアを備えているものもある．

たとえば CP-PACS/Pilot3 は RDMA と呼ばれる通信機構を備えている．この通信機構はブロックストライドという単位の送信を高速化する．ブロックストライドとはある任意のバイト長連続し（ブロック：4〜1020 Bytes），ある任意の等間隔（ストライド：4〜65532 Bytes）で繰り返されるデータを 1 度の送信命令で片側送信する通信機構である（**図 1**）．この機構は，等間隔で繰り返される複数のメモリブロックを一度に転送することができる．

このような特定のパターンの通信を高速化するハードウェアを，一般のユーザが活用してプログラムすることは現実的には難しい．活用するためには，そのハードウェアの詳しい使い方，およびアプリケーション実行中に現れるプロセッサ間の通信のパターンを知っていなければならない．しかし，並列計算機の一般ユーザである物理学者や天文学者などの計算機の非専門家に対し，これらの知識を持ってアプリケーション・プログラムを書くように要求することは現実的ではない．CP-PACS/Pilot3 の利用状況に関する我々の経験でも，計算機の専門家との共同作業で開発されたアプリケーションを除き，RDMA のブロックストライド通信のようなハードウェアに密着したライブラリはほとんど使われていない．一般ユーザは MPI などのより抽象度の高い汎用ライブラリを使う傾向にあり，特定の並列計算機固有の機能の利用を避ける傾向にある．

一般のユーザが通信の高速化のためのハードウェアを活用して計算を実行できるようにするためには，自動並列化コンパイラによる支援が現実には不可欠である．しかしながら，従来の自動並列化コンパイラでは，このようなハードウェアを活用したコードを生成することに，これまでのところあまり成功していない．最小限のデータを，最小限の回数で，最も高速に送受信するようにコンパイルするには，コンパイル時に通信パターンを正確に分析する必要がある．しかしながら通常の静的解析技術では，制御の流れを確定する条件式や，分散配置される配列の添字式が複雑な場合，正確な通信パターンの特定が困難であるからである．大まかな通信パターンしか分からない場合，コンパイラは最適な通信を行うコードの生成をあきらめ，冗長だが正しく計算を行うことが保証できるコードを生成しなければならない．

実際，4 章で示す我々の実験では，HPF 準拠の商用コンパイラが生成したコードは，通信速度がボトルネックとなり十分な性能を達成できなかった．十分な性能を達成するには，JAHPF [4]が提案している，明示的な通信の抑制（LOCAL）や明示的なシャドウ領域の制御（SHADOW）など，補助的なディレクティブをプログラマがプログラム中に埋め込み，コンパイラによる通信パターンの静的な解析や最適化コンパイルを助けてやらなければならない．

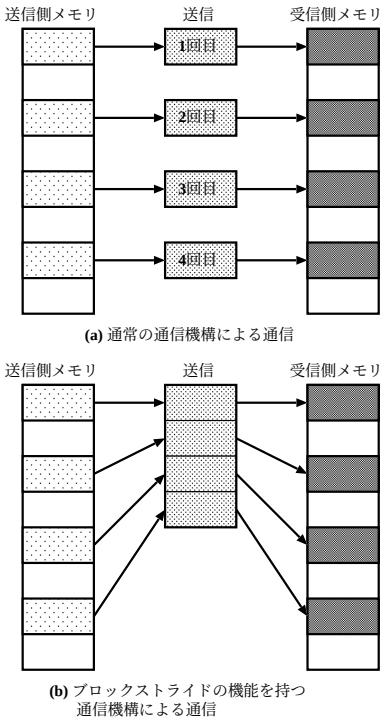

図 1 ブロックストライド通信と通常の通信
Fig. 1 Block-stride communication and normal communication.

3. 動的な解析による通信の最適化

我々はCP-PACS/Pilot3用にHPFコンパイラのサブセットを開発し，CP-PACS/Pilot3が持つ通信機構であるRDMAを最適な形で利用するようにコード生成を行えるようにした．このためには，プロセッサ間の通信パターンの正確な解析が必要である．我々はインスペクタ・エグゼキュータ方式を改良し，実行時に得られる通信パターンを解析して，動的に通信を最適化する方法を開発した．静的な解析による方法と比較して，実行時に解析を行う分，性能が低下するが，正確な通信パターンを容易に得ることができるので，通信に関してより強力な最適化を行い，全体として実行時性能を改善することができる．

また，実行時の解析にともなう性能低下を回避するため，実行時に得られた通信パターンをもとに，プログラム全体を静的に最適化コンパイルし直す方法も開発した．理想的には，実行中にコードが最適になるように自己書き換えするようなコードをコンパイラが生成するべきだが，我々のコンパイラは，並列処理される最外ループの繰返しごとに，通信パターンが変わらないとユーザが保証すると仮定して，オフラインで静的に最適化する．

以下では順にCP-PACS/Pilot3の持つRDMAの説明，インスペクタ・エグゼキュータ方式の説明，我々の提案する手法を説明する．

3.1 CP-PACS/Pilot3

CP-PACS，Pilot3は筑波大学計算物理学研究センターが所有する超並列計算機である．CP-PACSは2048，Pilot3は128個のプロセッサを持つ分散メモリ機であり，どちらも各プロセッサは同じアーキテクチャを持っている．各プロセッサはクロスバーで接続されており，CP-PACSでは3次元，Pilot3では2次元に接続されている．これらの計算機はブロックストライド（2章参照）をサポートしたRDMA（Remote DMA）通信機構[7]を持っている．

3.2 インスペクタ・エグゼキュータ方式

ループを分散メモリ機用に並列化する場合，それぞれの反復でどのような通信がプロセッサ間で必要になるか解析する必要がある．しかし，通信パターンをコンパイル時に静的に解析することができない，または困難な場合は，そのままでは並列化ができない．このような場合には，インスペクタ・エグゼキュータ[1]方式を用い，実行時に通信の解析を行うことで並列化をすることができる．

インスペクタ・エグゼキュータ方式は不規則な通信パターンに強く，実際の通信パターンの複雑さに関係なく，定型的なコードの変換で並列化をすることができる．ただし，通信パターンの解析を実行時に行うため，解析結果に基づいた冗長なコードの除去などの最適化は行わない．

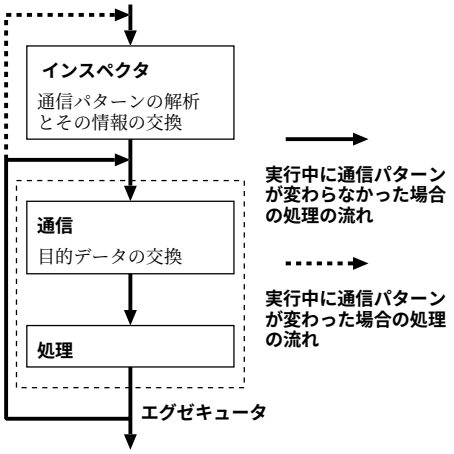

図2 インスペクタ・エグゼキュータ方式
Fig. 2 The inspector-executor method.

インスペクタ・エグゼキュータ方式によって並列化されたコードは次の流れで処理を行う（**図2**）．まず，実行時に並列に処理されるループと同じ構造のループを実行し，通信が必要になる配列の添字をチェックして通信先を特定する．次に，計算に必要な配列要素を持っているプロセッサや配列要素のメモリ上の位置などの情報を各プロセッサ間で交換しあう．その後，自分の持っているデータを他のプロセッサが必要とする場合，これを送信する．また，自分が必要なデータを受信する．最後に本来のループを並列に実行する．インスペクタ・エグゼキュータ方式では通信パターンの解析とその結果をループの繰返しのたびに各プロセッサで交換しあう処理が必要であるが，通信パターンに変化がないことをユーザの指示などで保証できれば，このインスペクタの作業は1度で済む．

3.3 通信機構の特性の活用

我々はまずインスペクタ・エグゼキュータ方式を改良し，RDMAの特性を最適に利用するようにした並列化コンパイラを開発した．具体的にはインスペクタによって得られた通信パターンの情報を用いて，通信を可能な限りブロックストライドの塊にまとめ，エグゼキュータ部の通信回数を減らすようにした．通信をまとめる処理はコスト高であるが，通信の解析はインスペクタ部実行時の1度で済むので，通信の最適化によるエグゼキュータ部の実行速度の向上の効果の方が

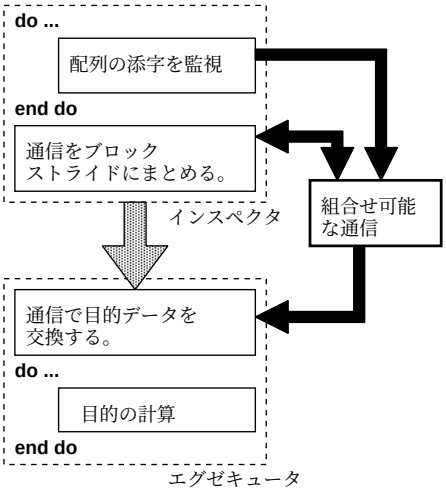

図 3 通信機構の特性の活用
Fig. 3 The optimized communication with the RDMA mechanism.

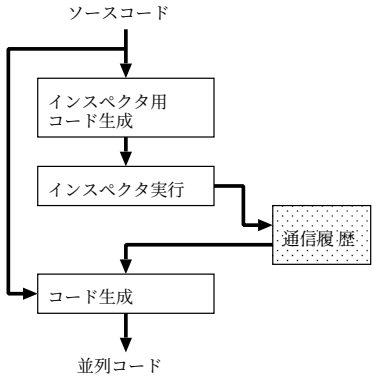

図 4 エグゼキュータ部の最適化コンパイルによる処理の流れ
Fig. 4 Optimizing compilation of the executor.

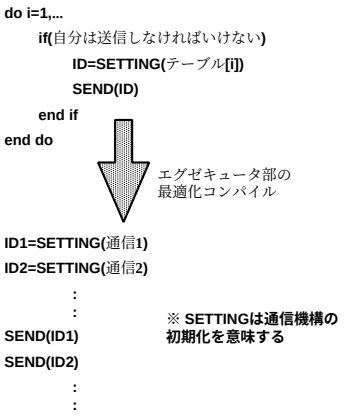

図 5 エグゼキュータ部の最適化コンパイル
Fig. 5 The optimized communication by the executor.

支配的で全体として性能が改善される．

本コンパイラは通信の回数を減らすために通信をHPF の INDEPENDENT 命令で指定された範囲でコードを移動し，可能な限りブロックストライドの単位にまとめるようにした．HPF の INDEPENDENT 命令とはプログラマがコンパイラに与えるヒントの 1 つである．INDEPENDENT と指定されたループは，ループが運ぶ依存がないかまたはそれが無視できるということを，プログラマがコンパイラに保証するために使われる．したがってこの指定がされているループでは反復をインデックス変数の示す順番で行う必要がなく，このループ中でなされる通信も反復の順番に従って行う必要がない．この範囲では異なる反復で行われるべき複数の通信をブロックストライドを単位とした 1 回の通信に置き換えて実行することができる．

図 3 はインスペクタ・エグゼキュータ方式を拡張して RDMA の特性を活用するようにしたコードの流れである．まず通常のインスペクタ・エグゼキュータ方式と同様にインスペクタ部で通信パターンを調べる．この後，INDEPENDENT 命令が指定する，通信の移動が可能な範囲を調べて，ブロックストライドにまとめられる通信の組合せを見つけ出して，テーブルに保存する．この後，エグゼキュータ部はこのテーブルに保存された情報をもとに通信を行いデータの交換を行う．

3.4 エグゼキュータ部の最適化コンパイル

前節の方法では，インスペクタ部がブロックストライドにまとめられる通信の組合せを見つけ，テーブルに保存し，エグゼキュータ部が実行時にこのテーブルを参照して通信を行う．実行時にエグゼキュータ部がテーブルを参照するオーバヘッドを取り除くため，我々のコンパイラは，コンパイル時にインスペクタ部だけを取り出して実行し，その結果得られた通信パターンをもとに，エグゼキュータ部をコンパイルして通信に関するコードを最適化する機能も持つ．この場合のコンパイラの処理の流れは**図 4** のようになる．理想的には，実行時にエグゼキュータ部を再コンパイルして，もとのコードを最適化されたコードで置き換えられるようにするべきであるが，現在の我々のコンパイラの実装では，インスペクタ部をコンパイル時に 1 回だけ実行し，その結果に基づいてエグゼキュータ部を静的に最適化コンパイルする．したがって，ループの繰返しごとに，通信パターンが変化しないことをユーザが保証しなければならない．

エグゼキュータ部を最適化コンパイルすることで，直感的には，プロセッサ間の通信について**図 5** のような最適化がなされる．前節の方法では，インスペクタ

部が作成したテーブルを参照するために余分なループや条件分岐が必要であったが，本節の方法では，それらが不要になる．またテーブル自体をメモリ上に置かないようにして，全体で必要となるメモリ量を抑制することができる．ただし本方式ではコードの大きさが増大するので，必ずメモリの使用量を抑制できるわけではない．また，RDMAでは，通信の総数があまり多くない場合，プログラムの先頭で通信に必要なパラメータをあらかじめ設定しておき，実行中は，設定したパラメータのIDを指定するだけで，通信をより高速に実行することも可能である．エグゼキュータ部を最適化コンパイルすることで，RDMAのこのような機能を利用することも可能になる．

本節の方法では，コンパイル時にインスペクタ部を実行するために，コンパイル時間が増加する．そこで我々のコンパイラは，-EXECUTE命令を提供して，インスペクタ部の実行時間を短縮するためのヒントをユーザが与えられるようにした．この命令は，通信パターンがループ不変であることを，ユーザがコンパイラに伝えるための命令である．インスペクタ部の実行時には，この指定がなされたループを1反復しか実行しない．これ以外は，プログラム全体を最初から最後まで忠実に実行する．この命令を使うことにより，たとえばポアソン方程式の反復解法のように，同じパターンの通信を引き起こす反復を繰り返し行うプログラムのコンパイル時間が短縮される．

現在の我々の実装ではインスペクタ部を1プロセッサ分しか実行しない．これはコンパイルを非並列機上のクロスコンパイル環境で行うからである．このためコンパイル可能なソースプログラムはすべてのプロセッサが同じパターンで通信しなければならないという制限が加わる．結果として，分散処理される配列はトーラス状でなければならない．なお，この制限はコンパイラ自身も並列計算機上で動かすように改良し，すべてのプロセッサ分についてインスペクタ部を実行するようにすれば回避できる．

4. 実　　験

我々が開発したコンパイラの性能を調べるために，従来方式と比較実験を行った．我々の開発したコンパイラは2つの方式をサポートしている．本章では，3.3節の方式を実行時の通信最適化，3.4節の方式をエクゼキュータの最適化コンパイルと呼ぶ．

4.1 実験環境

我々のコンパイラは，FreeBSD4.2上に実装されている．このコンパイラは，入力が基本的なディレクティブだけに対応したHPFのサブセット，出力がCP-PACS/Pilot3用のRDMAを利用したSPMDコードである．また実験ではコンパイルをPentiumIII 933 Mhz，メモリ1 GBytesのPC環境で行い，並列計算機での実行はPilot3で行った．

実験に用いたベンチマークはpde1（GENESIS Distributed Memory Benchmarks）とshallow（Shallow Water Benchmark）ベンチマークのHPF版[5]である．pde1は3次元ポアソン方程式の反復解法で，規模Nを7とし，10^4回の反復で測定した．Nは並列化の対象になる配列変数の大きさを決めるパラメータで，並列化の対象になる配列変数の各次元の大きさは2のN乗になり，総計算量は8のN乗に比例する．shallowは2次元の配列を水面に見立てて波紋の広がりをシミュレートする計算である．規模Nを1024とし10^3回の反復で測定した．Nは並列化の対象になる配列変数の大きさで，総計算量はNの二乗に比例する．

4.2 実行速度

まず，我々が開発したコンパイラによって生成されたコードの実行時間を，それぞれのベンチマークについて測定した．比較のため，RDMAの活用を考えない素朴なインスペクタ・エグゼキュータ方式のコンパイラも開発し，生成されたコードの実行時間を測定した．また，日立製のHPF準拠のコンパイラ（98年バージョン02-05）が生成したコードについても測定した．日立製のコンパイラには，通信ライブラリとしてRDMAを，同期にはspinを使うことを指定した．コンパイラが未対応であったため，シャドウ領域の使用は指定しなかった．我々が開発したコンパイラおよび日立製のコンパイラともに，日立製のFortran90コンパイラ（バージョン02-06-/C+02-06-XF）をバックエンドとして用いた．

プロセッサ数が64の場合の測定結果を**表1**，**表2**に示す．またプロセッサ台数を増やした場合の，プロセッサ台数1の場合との速度向上比を**図6**，**図7**に示す．HITACHIは日立製のコンパイラ，I&Eは素朴なインスペクタ・エグゼキュータ方式，Opt.Commは我々の実行時の通信最適化方式，Opt.C&Eは我々のエグゼキュータの最適化コンパイル方式を，それぞれ表す．日立製コンパイラの場合の実行時間は，仕事の規模を1/10にして実行し，これを10倍した値である．

いずれのベンチマークでも，我々の方式は他方式に比べて顕著な速度向上を達成した．これはCP-PACS/Pilot3の場合，RDMAの機能をうまく活用できた場合とそうでない場合とで，通信速度が大きく変化してしまうためである．また，エグゼキュータ部

表 1 実行時間（pde1：$N = 7$，ロセッサ数 $= 64$）
Table 1 The execution time (pde1: $N = 7$, 64 processores).

方式	実行時間 (sec)
HITACHI	138400
I&E	9720
Opt.Comm	538
Opt.C&E	512

表 2 実行時間（shallow：$N = 1024$，プロセッサ数 $= 64$）
Table 2 The execution time (shallow: $N = 1024$, 64 processors).

方式	実行時間 (sec)
HITACHI	11355
I&E	1021
Opt.Comm	201
Opt.C&E	191

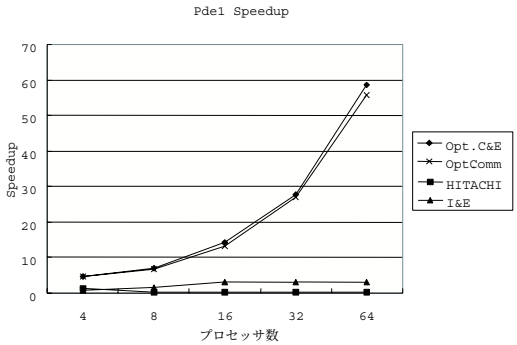

図 6 速度向上比（pde1）
Fig. 6 Speed up (pde1).

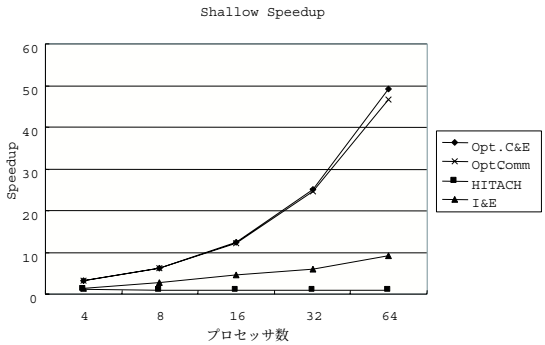

図 7 速度向上比（shallow）
Fig. 7 Speed up (shallow).

を静的に最適化コンパイルすることで，実行時の通信最適化方式に比べても，pde1 で最大 6%，shallow で最大 5%ほど高速になった．なお，実行時の通信最適化方式の場合も，ループの繰返しごとに，通信パターンが変化しないことをユーザからコンパイル時にヒントとして与えられたものとして実行している．このためインスペクタ部はループの最初の反復のときにだけしか実行されない．

実験に用いたベンチマークでは，配列を各プロセッサのメモリ上に分散配置する際，配列の最終次元を block 分割している．このため，仕事の規模 N が小さい場合には，RDMA のブロックストライド通信機能を使わなくても，単純なブロックの一括送信ですんでしまう．しかしながら N が大きくなると，ブロック長がハードウェアが許す最大長 1020 Byte を超えてしまい，1 回のブロック送信で送信できなくなってしまう．我々が開発したコンパイラが生成するコードでは，このような場合にも，複数のブロック送信を 1 回のブロックストライドにまとめて送信するように最適化される．なお，日立製のコンパイラが生成するコードがどのような通信を行っているか詳細は不明である．

4.3 コンパイル時間

3.4 節で述べたエグゼキュータの最適化コンパイル方式は，インスペクタ部をコンパイル時に実行するので，コンパイル時間が非常に長くなる傾向がある．そこで我々は，開発したコンパイラによるコンパイル時間についても，それぞれのベンチマークについて測定した．

図 8，**図 9** は，それぞれのベンチマークのコンパイル時間である．仕事の規模 N を大きくすると，それにあわせてコンパイル時間も増加していくことが分かる．ちなみに実行時の通信最適化方式の場合のコンパイル時間は，仕事の規模 N にかかわらずつねに一定で，pde1 の場合 0.7 秒，shallow で 6 秒であった．

しかしながらエグゼキュータの最適化コンパイル方式ではプロセッサの台数を増やすと，コンパイル時間は大きく減少した．これは，コンパイル時には，インスペクタ部をプロセッサ 1 台分しか実行しないため，プロセッサ台数を増やすとプロセッサ 1 台あたりに割り当てられる仕事量が減少し，その結果インスペクタ部の実行時間も短縮されるためである．

コンパイル時間と実行時間の合計で比較すると，エグゼキュータの最適化コンパイル方式による実行速度の改善は，実行時の通信最適化方式に比べて，pde1（$N = 7$，プロセッサ数 $= 64$）の場合，1.4%の速度向上を得られた．一方，shallow（$N = 1024$，プロセッサ数 $= 64$）では，エグゼキュータの最適化コンパイル方式の方がかえって遅くなってしまった．しかし，コンパイル時間の増加の影響は，プログラムの最外ループの反復の回数を増やせば，相対的に小さくなる．たとえば shallow の場合，反復回数をもとの 10^3 から，1.6×10^3 以上に増やせば，エグゼキュー

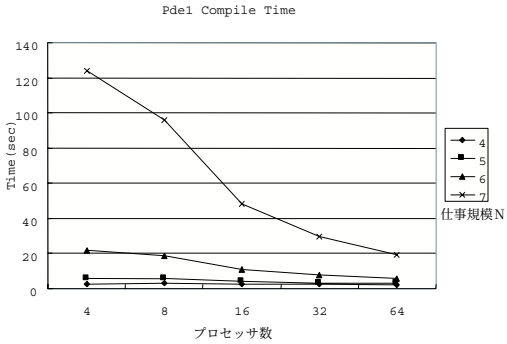

図 8 エグゼキュータ部の最適化コンパイルによるコンパイル時間（pde1）
Fig. 8 The compilation time (pde1).

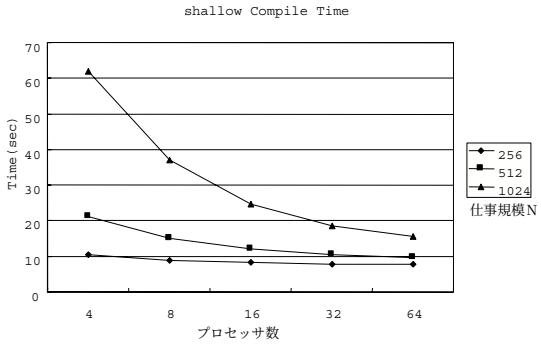

図 9 エグゼキュータ部の最適化コンパイルによるコンパイル時間（shallow）
Fig. 9 The compilation time (shallow).

タの最適化コンパイル方式の方が速くなる．我々のコンパイラの対象分野は計算科学であり，典型的なアプリケーションは，時間の経過にともなう物理現象の変化のシュミレーションである．このようなアプリケーションでは，反復回数が非常に大きく，全体の実行時間も数十時間から数百時間に及ぶ．このため，コンパイル時間の増加の影響は相対的に小さくなると考えられる．

5. 関連研究

TEA Expert [2]は動的な情報を用いて並列プログラムを最適化するツールである．我々のエグゼキュータ部の最適化コンパイル方式と同様に，実行時の情報を用い，あらためてコードを生成する方式である．このシステムは実行，実行時間の測定，パラメータの調整を繰り返し行い，最適なパラメータを求めるためのシステムである．パラメータとは，ループを mining する反復数，最内周ループの unrolling の段数，並列プログラムでのデータの分割法とサイズ，スケジューリングを静的に行うか動的に行うかなどである．TEA Expert の最適化の対象は通信ではなく，メモリアクセスやデータ分割に関するものである点が我々の研究と異なる．

Voss ら[8]も実行時情報を用いた最適化の研究を行っている．本研究との違いは最適化の対象がことなる点と，実行時情報を用いて最適化したコードをあらためて生成しない点である．Voss らの研究ではループ並列化の可否，tilling，serializing を対象にしている．また，Diniz ら[10]の研究では実行時情報を用いた同期の最適化を取り扱っている．並列計算ではないが，ATLAS [11]や，窪田らによる Java の実行時最適化[12]では，実行時情報を用いたループの最適化を研究している．

6. まとめ

本論文では，分散メモリ型の並列計算機である CP-PACS/Pilot3 用に我々が開発した自動並列化コンパイラについて述べた．CP-PACS/Pilot3 が持つプロセッサ間の通信機構 RDMA を最大限に利用するコードを生成するため，我々のコンパイラはインスペクタ・エグゼキュータ方式を改良した方法に基づいてコンパイルを行う．これにより，プロセッサ間の通信パターンについての正確な情報が得られるようになり，生成コードが効率良く通信を行えるようになる．

また実行時に通信のやり方を決定することによるオーバヘッドを回避するため，インスペクタ部をコンパイル時に実行し，その結果得られた通信パターンをもとに，エグゼキュータ部を静的に最適化コンパイルする方法も開発した．2 種類のベンチマークで実験したところ，この方法を用いると，さらに 5%程度の性能向上が達成できた．

この方法では，インスペクタ部をコンパイル時に実行するため，コンパイル時間が増加してしまう．現在の実装では，これを少しでも短縮するため，コンパイラは 1 プロセッサ分のインスペクタ部しか実行しない．このため，すべてのプロセッサは同じパターンで通信しなければならず，分散処理される配列は必然的にトーラス型であることが要求されてしまう．この制限は，コンパイラ自体も並列機で実行し，全プロセッサ分のインスペクタ部を実行するようにすれば回避できる．我々のコンパイラをそのように拡張することは，今後の課題である．また理想的には，インスペクタ部もコンパイル時ではなく実行時に動かし，エグゼキュータ部を実行中に動的に最適化コンパイルするようにすべきである．この点も今後の研究課題である．

参 考 文 献

1) Koelbel, C. and Mehrotra, P.: Compiling Global Name-Space Parallel Loops for Distributed Execution, *IEEE Trans. Parallel and Distributed Systems*, pp.440–451 (1991).
2) 佐藤三久，建部修見，関口智嗣，朴 泰祐：自動適応並列プログラム性能最適化ツール TEA Expert，情報処理学会 HPC 研究会報告，pp.13–18 (1998).
3) High Performance Fortran Forum，富士通，日立製作所，日本電気（訳）：High Performance Fortran 2.0 公式マニュアル，シュプリンガーフェアラーク東京 (1999).
4) 高度情報科学技術研究機構東京事業所 Home Page. http://rist03.tokyo.rist.or.jp/jahpf/hug2000/
5) Portland Group. ftp://ftp.pgroup.com/pub/HPF/examples/
6) 計算物理学研究センター. http://www.rccp.tsukuba.ac.jp
7) 岩崎洋一，中澤喜三郎ほか：計算物理学と超並列計算機—CP-PACS 計画，情報処理，Vol.37, No.1, pp.10–42 (1996).
8) Voss, M. and Eigenmann, R.: Dynamically adaptive parallel programs, *Proc. Int'l Symposium on Highperformance Computing*, LNCS, Vol.1615, pp.109–120, Springer (1999).
9) Weise, D., Crew, R.F., Ernst, M. and Steensgaard, B.: Value Dependence Graphs: Representation Without Taxation, *ACM SIGPLAN POPL*, pp.297–310 (1994).
10) Diniz, P. and Rinard, M.: Dynamically feedback: An effective technique for adaptive computing, *ACM SIGPLAN PLDI*, pp.71–84 (1997).
11) Whaley, R.C. and Dongarra, J.J.: Automatically tuned linear algebra software, *Proc. SC*, Orlando, FL (Nov. 1998).
12) 窪田昌史，坂口陽祐，津田孝夫：実行時情報を用いた最適化手法，情報処理学会 HPC 研究会報告，pp.23–28 (1999).

(平成 12 年 9 月 18 日受付)
(平成 13 年 2 月 1 日採録)

横田 大輔

1972 年生．1995 年慶應義塾大学理工学部計測工学科卒業．1998 年同大学院理工学研究科計算機科学専攻修士課程修了．現在筑波大学工学研究科電子・情報工学博士課程在学中．

千葉 滋（正会員）

1968 年生．1991 年東京大学理学部情報科学科卒業．1993 年同大学院理学系研究科情報科学専攻修士課程修了．1996 年同専攻博士（理学）取得．1996〜97 年同専攻助手．1997 年より筑波大学電子・情報工学系講師．言語処理系およびオペレーティングシステム等システムソフトウェアの研究に従事．日本ソフトウェア科学会，ACM 各会員．

板野 肯三（正会員）

1948 年生．1977 年東京大学大学院理学系研究科物理学専門課程博士課程単位取得後退学．1993 年より筑波大学電子・情報工学系教授．計算機のアーキテクチャ，分散処理システム，プログラミングシステム等に関する研究に従事．理学博士．日本ソフトウェア科学会，電子情報通信学会，ACM，IEEE 各会員．