ハイパフォーマンス
コンピューティング 82−16
（2000. 8. 4）

実行時情報を用いたブロックストライド通信の静的な最適化

横田 大輔† 千葉 滋†‡ 板野 肯三†

†筑波大学工学研究科 ‡科学技術振興事業団 さきがけ研究21

ループ並列化において、通信のパターンを静的に解析する事が困難である場合、インスペクタ-エクゼキュータ方式を用いる事で容易に並列コードを得る事が出来る。しかし、インスペクタ-エクゼキュータ方式は通信パターンの解析を実行時に行なうため、理想的にカスタマイズされたコードを生成する事が困難である。特に、並列計算機が持つ通信機構を活用して通信コードを最適化する事がむずかしい。そこで我々は、インスペクタ-エクゼキュータ方式におけるインスペクタ部の実行をコンパイル時に行なって、通信とその周辺のコードを最適化する手法を提案する。我々はこの手法を応用して実際にHPFコンパイラのサブセットを実装した。本論文では、このコンパイラを使って行なった予備実験の結果も報告する。

Static optimization of block-stride communications with runtime informations

DAISUKE YOKOTA †, SHIGERU CHIBA †‡ and KOUZOU ITANO †

†Institute of Information Science and Electronics, University of Tsukuba
‡PREST,Japan Science Technology Corp.

This paper describes a derived Inspector&Executor method. In loop parallelization, we can get parallel code by Inspector&Exector, though static analysis cannot inspect communication patterns in source program. But these communication codes can not be optimized as the best customized codes, because inspector analyzes communication patterns for execution time. In addition it is difficult to use special communication hardwares, even if they are much faster than usual communication hardwares, if these devices must control under special rule. We are studying derived Inspector&Executor. We move inspector phase to compile time to get parallel codes with the best communication. This paper describes this method for CP-PACS and Pilot3, and our compiler generate good optimized communications for their block-stride RDMA communications.

1. はじめに

並列化コンパイラが分散メモリマシン用にループを並列化をする場合、プログラムの適切な位置に通信コードを挿入しなければならない。しかし、プログラムによっては、コンパイル時に通信パターンを解析する事が困難な場合がある。例えば、分散配置された配列変数へのアクセスの際、他の配列変数、関数などによってインデックスが指定される場合など、インデックスがループ制御変数の線形結合になっていない場合がそうである。このようなプログラムを並列に実行するには、インスペクタ-エクゼキュータ方式[2]という手法を用いる必要がある。

インスペクタ-エクゼキュータ方式は、通信が必要な箇所を実行時に調べ、その情報をプロセッサ間で渡し合い、通信を行なう。この方式を用いると、分散配置された配列変数にアクセスする際に使われるインデックスの値を直接知ることができるので、通信パターンを静的に解析できない場合に有効である。しかし、通信パターンの解析を実行時に行ない、その解析結果を参照しながら通信を行なうので、通信回数などに関して最適化を行なう事が難しくなったり、通信パターンが単純な場合でも、解析結果を参照し、通信内容を決定する部分のコードが冗長になりがちである。

また、特定の条件を満たせば通常の通信より高速に通信できる機構を持った並列計算機を利用する場合、通常のインスペクタ-エクゼキュータ方式ではこの機能を利用する事が難しい場合がある。また、利用できた場合でも、コンパイル時に通信パターンが分かっている場合にできるような最適化ができない。例えばCP-PACS、Pilot3[7]はブロックストライドを単位としたメモリ領域を高速に通信する機構を備えているが、インスペクタ方式でこの通信機構を効率良く利用するのは容易ではない。

我々はインスペクタ-エクゼキュータ方式のインスペクタ部分をコンパイル時に仮実行し、その解析結果をもとに、最適な通信コードを生成する方式を研究している。通常のインスペクタ-エクゼキュータ方式と比べ、解析結

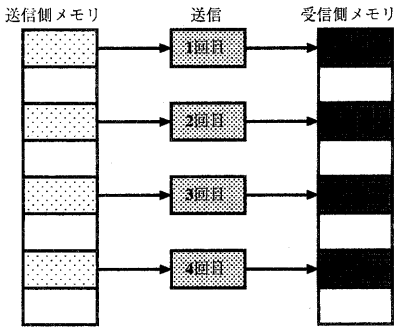
果を参照し、通信内容を決定するコードをエクゼキュータ部分から除去できるので、実行時性能を改善できる。我々はこの方式に基づいたHPFコンパイラのサブセットをCP-PACS、Pilot3向けに開発しており、実験によって性能改善を確かめようとしている。

本論文では、CP-PACS、Pilot3の持つブロックストライド通信の説明、およびインスペクタ-エクゼキュータ方式の説明を行なう。次に、本方式によって実装したHPFコンパイラについて述べ、本方式と通常のインスペクタ-エクゼキュータ方式、ブロックストライド通信の利用にある程度特化したインスペクタ- エクゼキュータ方式との比較を、我々がおこなった予備実験の結果をふまえて論じる。

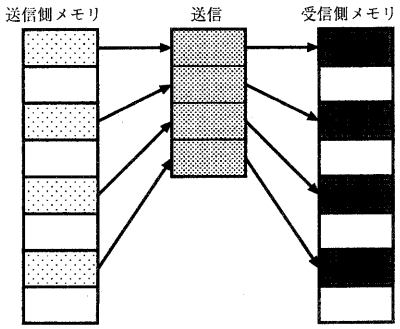
2. 研究の背景

ここでは、本研究で実験に用いたCP-PACS、Pilot3の説明と、インスペクタ-エクゼキュータ方式の説明をする。

2.1 CP-PACSとPilot3



(a) 通常の通信機構による通信



(b) ブロックストライドの機能を持つ通信機構による通信

図1 ブロックストライド通信と通常の通信

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個のプロセッサユニットを持つ分散メモリ機であり、どちらも各プロセッサユニットは同じアーキテクチャを持っている。これらの計算機はブロックストライドをサポートしたRDMA(Remote DMA)通信機構[8]を持っている。

この通信機構は、ある任意のバイト長連続し(ブロック)、ある任意の等間隔(ストライド)で繰り返されるデータを一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事ができる。図1では等間隔に並んだ4個の同サイズのメモリブロックを転送する例であるが、通常の通信機構では、4回に分けて送信しなければならない。しかし、CP-PACS、Pilot3のブロックストライドで通信を利用すれば、一回の送信命令で送信する事ができる。ブロック長は4~1020バイト、ストライド長は4~65532バイトで可変である。この通信は片側通信であるので、基本的に受信命令を必要とせずに、受信側のプロセッサのローカルメモリにデータを書き込む事が出来る。また、ターゲットのローカルメモリにバッファを介さず直接書き込むので、余分なメモリコピーが発生せず高速な通信が可能である。

2.2 インスペクタ-エクゼキュータ

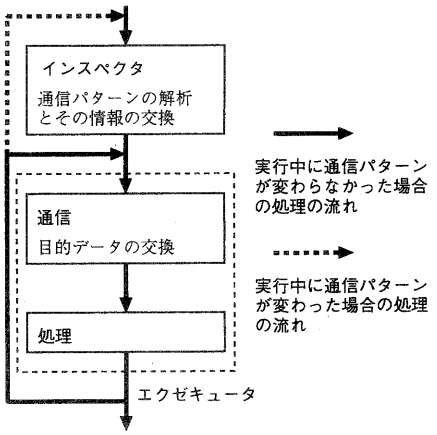


図2 インスペクタ-エクゼキュータ方式

ループを並列化する場合、それぞれの反復でどのような通信が行なわれるか解析する必要がある。しかし、このような情報をコンパイル時に解析する事ができない、または困難な場合は、通常の方法で並列化をする事ができない。このような場合、インスペクタ-エクゼキュータ[2]を用いる事で、実行時に通信の解析を行なうことで並列化をすることができる。

インスペクタ-エクゼキュータは不規則な通信パターンに強く、通信パターンに関係なく、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なうため、複数の通信の融合などの最適化を行なう事は難しい。

インスペクタ-エクゼキュータによる並列コードは次の

流れで通信を行なう (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周り、問題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、通信先と必要な要素位置などを各プロセッサ間で交換し合う。次に、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-エクゼキュータでは通信パターンの解析とその結果を各プロセッサで交換し合う作業が必要であるが、通信パターンに変化がない事をユーザの指示等で保証できれば、このインスペクタの作業は一回で済む。

3. 実行時情報を用いた静的な最適化

並列計算機によっては、うまく工夫して使用すると高性能を発揮する通信機構を持っている場合がある。しかし、通常のインスペクタ-エクゼキュータでは、工夫による性能向上が、工夫を行なうための解析に必要な時間を上まわらないと、このような通信機構を利用できない。しかし、インスペクタ-エクゼキュータ方式でないと並列化ができない、あるいは困難なソースコードを実行する場合でも、これらの機構を利用したい。

そこで、インスペクタ-エクゼキュータ方式の一部を変更して、そのような通信機構の利用を可能にする方法を提案する。また、可能ならばエクゼキュータ自体が持っている、通信パターンの解析結果を参照するためのコードを除去する。本論文で実装したコンパイラは、そのような通信機構に特化したコードを出すために、インスペクタに相当する処理をコンパイル時に行ない、得られた結果を並列化と通信の最適化のための情報として利用する。

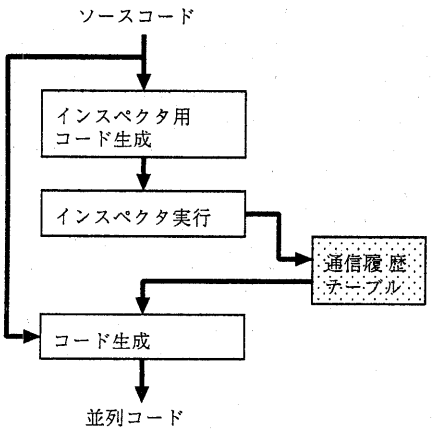


図 3　本方式による処理の流れ

我々が CP-PACS、Pilot3 用に実装したコンパイラは図 3に示される流れで処理される。まず、もとのプログラムの一部分を実行してその通信履歴を記録する。このためにコンパイラは前処理として、通信パターンを記録するコードを生成し、そのコードを実際に実行し通信履歴を記録する。その後、コンパイラは記録された通信履歴を参照しながら、プログラムの実行中に通信パターンの変化がないものと仮定して、最終的な並列コードの生成をおこなう。これにより、コンパイラはコンパイル時に実行時の通信パターンを知る事ができ、通信に関するコードの最適化を行なう事ができる。

3.1 本方式の利点と欠点

本研究の方式の利点は、最終的に得られるコードの実行速度の向上である。これは、得られるコードにはインスペクタ部がなく、また実行時の通信パターンを利用して通信関連のコードを最適化できるからである。通信パターンが比較的単純な場合、例えば複数回の通信を 1 回の通信に融合できる可能性がある。また、速度向上以外にも、通常のインスペクタ-エクゼキュータと違って、対象となる配列と通信履歴を同時にメモリ上に置かずに済む。

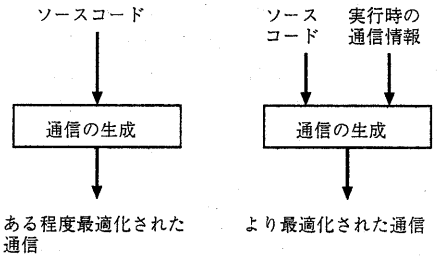


図 4　実行時情報を利用できる利点

本方式の欠点は、プロセッサ間の通信パターンがループの実行中に変化していないことを仮定している点である。これはインスペクタに相当する部分をコンパイル時にループ 1 回分しか実行しないためである。しかしながら現実の並列計算では、このような仮定が成り立つ場合が多いと考えられる。また、ループの実行中に通信パターンが頻繁に変化する場合、通常のインスペクタ-エクゼキュータ方式も実行性能が大幅に低下するので、実用的でない。さらに、将来、本方式を拡張して、ループの実行中に通信パターンが変化した場合には、再コンパイルをおこない、実行中のコードを直接置き換えられるようにすれば、本方式の欠点は回避できると我々は考えている。

本方式に従って現在までに我々が実装したコンパイラでは、通常のインスペクタ-エクゼキュータにはない制限がある。本方式を利用するにあたって、プログラマは、全てのプロセッサが同じ通信パターンで通信する事を保証しなければならない。この理由は次のためである。本方式はインスペクタにあたる処理をコンパイル時に行なう。通常、インスペクタの処理は、実行中に並列に行なわれる。しかし、本方式ではインスペクタをクロスコンパイラの環境で実行するので、インスペクタの処理は単

一プロセッサで行なわなければならない。この時、ループと分散配置される配列変数の添字の全てを解析すると、コンパイル時間が非実用的なほど増加してしまう。このため、我々が実装したコンパイラでは、全てのプロセッサが通信に関して同じ振舞をするとして、プロセッサ1台分の解析のみ行なうようにした。

3.2 インスペクタに相当するコードの生成

我々が実装したコンパイラでは、通常のインスペクタ-エクゼキュータ方式でインスペクタ部を生成するのと同様の手順で、ループをプロセッサ毎に切り分け、分散配置される配列変数にアクセスする式の値を監視するコードを埋め込む。また、配列の添字以外の情報は不要なので、添字の値を求める計算に関係しない部分をデータフローで探し削除する。

通常のインスペクタ-エクゼキュータ方式のインスペクタ部と異なる点は、実行されるループの範囲がコマンドラインなどから指定されたプロセッサが担当する範囲だけという点である。監視用のコードは、そのプロセッサが受け持つ配列の領域の外の要素を参照した場合、その参照をストレージに書き出すようになっている。このインスペクタ部に相当するコードはコンパイル中にgccによって実行コードに変換され、本コンパイラから呼び出され実行される。

3.3 エクゼキュータに相当するコードの生成

インスペクタに相当するコードによって生成された通信履歴はストレージ上に記録される。この履歴は本コンパイラによって回収され、プロセス間で交換が必要になる配列要素の集合を特定する。

CP-PACS/Pilot3が持つブロックストライド通信は4～1020バイトのブロック長、4～65532バイトのストライド長を持つ。配列の要素の交換は、この範囲で表現出来る可能な限り少ない数の通信で実現する必要があので、回収された通信履歴と、静的に解析されたソースコードの情報を用いて最適化を行なう。

通信は、可能な限り少ない回数で行ないたいので、HPFのINDEPENDENT指示文を持つループが多重である場合、最も外のINDEPENDENT指示を持つループの直前に通信命令を配置するようにした。

本コンパイラは、同じ箇所に配置される通信の集合を、それより外部のループで同じ反復の集合で実行される通信毎にグループに分ける。次に各グループ内で、全ての通信を通信先、書き込みか読み込みかで小グループに分類する(図5)。この後、この表を整理する。この状態で、各小グループ内の通信は、実行順に関係なく実行できる。

その後、集合から、ブロックの値が最大になるように通信を必要とする配列要素を要素の若い方から探す。この後、繰り返しが最大になるようなストライド幅を探していく。ブロックストライドの回数がこれ以上とれないところまで走査が終ると、ブロック幅を縮める事によって走査が先に進み、送受信される面積がこれ以上大きくなるならば、走査を進め、そうでない事を確認できれば、一組の通信命令にパックする。このとき、この通信が扱う配列要素の集合が属する小グループに、外側のループの特定の反復の時にのみ実行される指定がある場合は、IF文で通信命令が実行される反復を制限する。

```
REAL AR(1:1000,1:1000)
!HPF$ PROCESSORS procs(4)
!HPF$ DISTRIBUTE (BLOCK,*) procs AR
 INTEGER I,J
 DO J=1,999
!HPF$ INDEPENDENT
 DO I=1,1000
     IF(J.LE.500)THEN
 AR(I,J)=AR(I-3,J+1)
     ELSE
AR(I,J)=AR(I+3,J+1)
     END IF
 END DO
 END DO
```

(a) HPFソースコード

配列	Jの範囲	R/W	通信先PE	要素
AR	1	R	-1	(-2,2)(-1,2)(0,2)
AR	2	R	-1	(-2,3)(-1,3)(0,3)
:	:	:	:	:
AR	500	R	1	(251,501)(252,501)(253,501)
AR	501	R	1	(251,502)(252,502)(253,502)
:	:	:	:	:

(b) 通信履歴の分類

配列	Jの範囲	R/W	通信先PE	要素
AR	1~499	R	-1	(-2,J+1)(-1,J+1)(0,J+1)
AR	500~999	R	1	(251,J+1)(252,J+1)(253,J+1)

(c) 整理された通信履歴の分類

配列	Jの範囲	R/W	通信命令
AR	1~499	R	左のPEにブロック3
AR	500~999	R	右のPEにブロック3

(d) 挿入される通信命令

図5 本方式による通信コードの整理例

3.4 コンパイル例

本コンパイラによって図6(a)のプログラムをコンパイルすると、図6(b)のコードが一番外側のループのループボディの先頭に挿入される。この例では、外側のループインデックスが偶数になった場合のみ通信が行なわれる。インスペクタ-エクゼキュータ方式で何も工夫せずにコンパイルすると、細切れの通信を500回実行しなくてはならない。また、ある程度それを見越して改良した場

合でも、エクゼキュータはインスペクタが解析した通信パターンを追うためのコードを含まなくてはならない。本コンパイラでコンパイルした所、1回の通信で済むように、RDMA機構を利用できた。

このコードには通常のインスペクタ-エクゼキュータ方式に必要な、インスペクタ部が存在しないだけでなく、エクゼキュータが解析された通信を追うためのコードや、それに必要な記憶領域を削除する事ができた。

```
REAL AR(1:1000,1:1000)
!HPF$ PROCESSORS procs(4)
!HPF$ DISTRIBUTE (BLOCK,*) procs AR
 INTEGER I,J,K
!HPF$ EXECUTOR
DO K=1,1000
!HPF$ INDEPENDENT
 DO J=1,1000
!HPF$ INDEPENDENT
 DO I=1,1000
     IF(MOD(J,2).EQ.0)THEN
 AR(I,J)=AR(I-3,J)+I*J
     END IF
 END DO
 END DO
END DO
 STOP
 END
```

(a) HPFソースコード

```
CALL $RSETSTR(.....,4*3,500,4*2000,.....)
CALL $RSETOFF(.....)
CALL $RSTRID(.....)
CALL $RREAD(.....)
```

(b) 挿入される通信

図6　本方式による通信コードの生成例

図6の例では、外側のループが偶数の時のみ通信を行なう。また、この時の通信は配列変数ARの右側ののりしろ3要素分である。この通信は結果として、2000要素毎に3要素の通信が必要になる。本方式ではこれを検出する事ができて、一組のブロックストライド通信で処理する事ができた。この時のブロック幅は4*3バイト、ストライド幅は4*2000バイト、ブロックストライドの繰り返し回数は500回である。

4. 実　　験

我々は本論文で述べた方式の有効性を確かめるため、この方式に基づいたHPFコンパイラのサブセットを実装した。このコンパイラはRDMA通信機構を使ったSPMD並列なFORTRANプログラムをCP-PACSまたはPilot3用に出力する。出力されたFORTRANプログラムは、CP-PACSおよびPilot3用に最適化されたFORTRAN90コンパイラでコンパイルされる。さらに比較のために、我々は通常のインスペクタ-エクゼキュータ方式に基づいたHPFコンパイラ、およびインスペクタ部で詳細な解析をおこない、複数の通信を一つにまとめるように工夫したインスペクタ-エクゼキュータ方式に基づくHPFコンパイラも実装した。

我々はこれらの3つのコンパイラを用いて、予備的な実験を行なった。図7の1000×1000の配列を周り、x軸に関して-3の距離でデータをやりとりするプログラムを3つのコンパイラでコンパイルし、実行にかかった時間を比較した。評価に用いた環境は次のような環境である。我々の方式によるコンパイルはPentiumII300Mhz、メモリ128MによるFreeBSD3.4環境である。実行はPilot3上で行なった。Pilot3は150Mhzのカスタム CPUで、評価ではこれを4台利用した。

```
REAL AR(1:1000,1:1000)
!HPF$ PROCESSORS procs(4)
!HPF$ DISTRIBUTE (BLOCK,*) procs AR
 INTEGER I,J,K
!HPF$ EXECUTOR
DO K=1,1000
!HPF$ INDEPENDENT
 DO J=1,1000
!HPF$ INDEPENDENT
 DO I=1,1000
 AR(I,J)=I*J+AR(I-3,J)
 END DO
 END DO
END DO
 STOP
 END
```

図7　実験に用いたコード

実験結果は図8のとおりである。インスペクタ部の実行時間は1000回繰り返した場合の平均値である。今回の実験では、本方式は通常のインスペクタ-エクゼキュータ方式に比べ1.8倍、改良したインスペクタ-エクゼキュータ方式に比べ1.2倍の速度向上を得る事ができた。一方、本方式の問題点として、コンパイル時間が増加する点があるが、今回の実験ではコンパイル時間は27.2秒なので、容認できる時間だと考えられる。

	本方式	インスペクタ-エクゼキュータ	改良インスペクタ-エクゼキュータ
コンパイル時間	**27.2sec**	**9.8sec**	**9.8sec**
インスペクタ時間		**25.5msec**	**42.5msec**
1000回実行した時間	**23.5sec**	**42.8sec**	**27.2sec**

図8　評価に用いたコード

5. おわりに

本論文では、インスペクタ-エクゼキュータ方式のイン

スペクタ部をコンパイル時に移すことで、並列計算機のもつ通信機構を活かして最適化された通信が可能になることを示した。我々はこの手法に基づいたHPFコンパイラのサブセットを並列計算機CP-PACS、Pilot3用に実装し、予備的な実験では、インスペクタ-エクゼキュータ方式に比べて1.2倍ないしは1.8倍の実行速度の改善が見られることを確かめた。また、本論文の方式ではコンパイル時間が長くなるという心配があるが、コンパイル可能なプログラムに制約を課すことで、コンパイル時間を実用的に許容できる範囲におさえることができた。

今回の実験ではPilot3を用いて4並列で行なったが、より現実的な状況での利用を想定した実験をおこなう必要がある。我々は本コンパイラの有用性を明らかにするために、CP-PACS上でのLINPAK等を用いた実験の準備を進めている。

インスペクタ-エクゼキュータ方式の持つ利点は、並列化が困難なコードを並列化できる点であるが、本方式利点はコンパイラの開発を容易にする点にもある。インスペクタ部で明らかになるような実行時の通信情報は、実際の通信情報であり、反復と通信先の情報が直観的で明らかである。特定の並列計算機固有の通信機構に対応したコンパイラは他に流用できないため、少ない労力で開発できる事が望ましい。本方式では、解析結果によって得られる通信の情報が、反復、通信先などについて明示的である。そのため、このような特定の並列計算機固有の通信機構を利用するコンパイラを実装する場合、開発期間を短くする事が可能であると考えられる。今回のコンパイラの対象はCP-PACS、Pilot3のRDMA通信であったが、他の通信機構を持っている並列計算機用に転用する場合でも、変更する箇所は少なくて済む。

参考文献

1) David F. Bacon, Susan L. Graham, Oliver J. Sharp: "Compiler Transformations for High-Performance Computing" *ACM '94 Computing Surveys pp.345 − 419*
2) Charles Koelbel, Piyush Mehrotra: "Compiling Global Name-Space Parallel Loops for Distributed Execution" *IEEE '91 Trans. on parallel and distributed systems pp.440 − pp.451*
3) Michael Wolfe: "HIGH PERFORMANCE COMPILERS FOR PARALLEL COMPUTING" The Addison-Wesley Publishing Company
4) B. K. Rosen, M. N. Wegman, F. K. Zadeck: "Global Value Numbers and Redundant Computations" *ACM SIGPLAN '88 POPL pp.12 − 27*
5) R. sytron, J. Ferrante, B. K. Rosen, M. N. Wegman: "Efficiently Computing Static Single Assignment Form and the Control Dependence Graph" *ACM SIGPLAN '88 PLDI pp.451 − 490*
6) High Performance Fortran Forum, 富士通株式会社, 株式会社日立製作所, 日本電気株式会社 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
7) "計算物理学研究センター" http://www.rccp.tsukuba.ac.jp
8) 株式会社日立製作所: "リモートDMA転送利用の手引-FORTRAN-" 株式会社日立製作所
9) Steven Brawer, 大森 健児 訳: "並列プログラミングの基礎" 丸善株式社