社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－HPC－103 (28)
2005／8／5

flat-c：超並列計算機向けC言語の実現

西川 徹† 稲葉 真理† 平木 敬††

超並列計算機向け flat-c 言語を提案する。flat-c 言語では、ネットワークやプロセッサ内の制御を制限することにより、より多くのプロセッサを集積できるようにした超並列計算機を対象とする。これらの計算機に適合したプログラムを書くためのプログラミング言語が flat-c 言語である。本論文では、flat-c 言語で採用する超並列計算機のモデルを示し、そのモデルがどのように言語仕様に含まれているか、及び、採用するモデルの各機能を flat-c 言語からどのように利用するかを示す。そして、現在開発中の flat-c 言語のコンパイラがどのように実現されるかを示す。

flat-c : The C Programming Language for Massively Parallel Computers

TORU NISHIKAWA,† MARY INABA† and KEI HIRAKI††

We propose a flat-c language for massively parallel computers. The flat-c language is a programming language for writing programs suitable for massively parallel machines which integrates many processors by restricting network between processors and controls in each processor. In this paper, we show a model which we adopts in the flat-c language. Then we explain how this model is included in a specification of the flat-c language and how to use capabilities in this model by using the flat-c language. Lastly we present a way to realize a compiler which we are developing now.

1. はじめに

計算負荷の高い計算においては、より多くの演算器を並列計算機に集約することにより、演算性能を高めることが重要である。例えば、天文学などの分野では、重力計算といったN個の粒子に対してNの二乗の計算を行うような計算を、非常に多くの粒子に対して高速に計算を行うことが求められる。

1つの並列計算機により多くのプロセッサ要素を詰め込むことにより演算性能を向上させるアプローチでは、並列計算機と記憶領域との間のバンド幅がシステムの性能を制限する。しかしながら、行列演算や重力計算といった計算負荷の高いアプリケーションの多くは、アルゴリズムを変更することにより、並列計算機と記憶装置との間の必要なバンド幅を大幅に減らすことができる。これらのアルゴリズムでは、並列計算機と記憶領域との間のバンド幅によるシステム性能の制限を無視することができる。

† 東京大学情報理工学系研究科コンピュータ科学専攻
Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo
†† 東京大学情報理工学系研究科創造情報学専攻
Department of Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo

より多くのプロセッサ要素を並列計算機内に集約するためには、プロセッサ要素の機能を削減しプロセッサ要素自体のサイズを小さくすることと、プロセッサ要素間のネットワークを制限することによりプロセッサ間ネットワークの占めるチップ上面積を小さくする必要がある。そのために、本論文で扱う計算機は、すべてのプロセッサ要素が同一の命令を実行することにより、プロセッサ要素に必要な制御のための回路を小さくする。そして、プロセッサ要素間のネットワークは用意しない計算機を扱う。プロセッサ要素間通信の代わりに、プロセッサ要素をグループ分けし、そのグループ内に共有メモリを置くことにより、制限された形でプロセッサ要素間の通信を行えるようにする計算機を対象とする。これらの計算機では、各プロセッサ要素は、プロセッサ要素によって異なるデータを持つことができ、それに加えて、共有メモリを用いることによりグループ内のプロセッサ要素間で同一のデータにアクセスすることができる。本論文では、このような並列計算機をSIMASD(Single Instruction Multiple And Shared Data)型計算機と呼ぶ。

上記の並列計算機では、既存の並列プログラミング言語では記述できる通信パターンの自由度が高く、実際に行われる通信パターンの予測が困難となり、通信性能の予測が困難である。また、このような並列計算機に適したアルゴリズムを記述するためには、対象とする並列計算機に適していない計算・通信パターンを

何らかの方法で排除する必要がある。

本論文では、上述した特徴を持つ計算機向けの言語flat-c言語を提案する。flat-c言語は、並列計算機が持つ演算・通信機能を言語レベルで提供し、それ以外の演算・通信機能を記述できなくすることで、適していない計算・通信パターンを制限する。そして、外部との通信をユーザーが容易に予測可能にするため、プログラミングモデルを固定し、通信が生じる場所と生じる通信の種類を、ユーザーがプログラムの文面から判断できる言語仕様をもつ。

本論文では、2章で本論文で対象とする計算機モデルを提示する。3章でflat-c言語が採用しているプログラミングモデルを示す。4章でそれらのモデルがどのように言語仕様に取り込まれているかを示し、5章で現在開発中のflat-c言語コンパイラの構成について述べ、6章で関連研究を示し、7章でまとめる。

2. 計算機モデル

本節では、flat-c言語で採用する超並列計算機の計算機モデルについて説明する。

マシンは、多数のプロセッサを搭載したSIMASD型計算機、SIMASD型計算機を制御する制御プロセッサ、及び、制御プロセッサにデータを提供し自分自身も計算を行うホストマシンにより構成される。ホストマシンには1つの制御プロセッサが接続されており、SIMASD型並列計算機にデータ・命令を送るときは、制御プロセッサを介して行われる。概要を図1に示す。

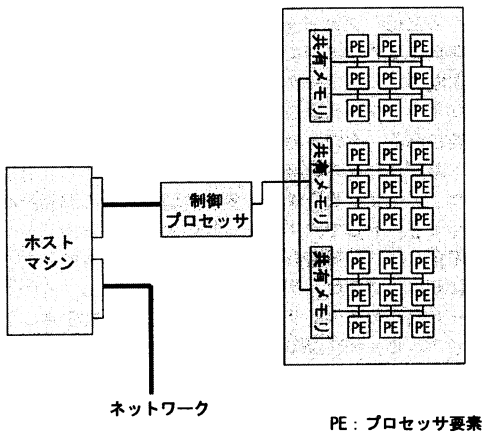

図1 マシンの概略

制御プロセッサは、ホストマシンとSIMASD型計算機とを接続し、ホストマシンからSIMASD型計算機を制御するための機能を提供する。ホストマシンは、制御プロセッサを介して、SIMASD型計算機へのデータ送信、結果の回収、命令の送信を行う。ホストマシンが制御プロセッサを通じてSIMASD型計算機に命令を送るには、まず制御プロセッサに命令列をホストマシンから登録し、その登録した命令列をSIMASD型計算機に発行するコマンドをホストマシンが制御プロセッサに送ることにより、SIMASD型計算機に命令列を送ることができる。また、制御プロセッサにデータ転送・結果回収コマンドを送ることにより、ホストマシン上のメモリからデータを読み出してSIMASD型計算機に転送したり、ホストマシン上のメモリにSIMASD型計算機の計算結果を書き込む、ということを制御プロセッサが行う。また、制御プロセッサは記憶領域と簡単なシーケンサを持つ。これは、値を一部分づつホストメモリからSIMASD型計算機に転送し計算する、という操作を繰り返す計算のために用意される。ホストマシン上のメモリからループ毎に値を転送する代わりに、あらかじめ制御プロセッサにホストマシン上のメモリのデータを転送しておき、制御プロセッサ・ホストマシン間の通信を削減する。

SIMASD型計算機は、単純な機能を持ったプロセッサ要素を集約する。プロセッサ要素の機能には、演算器・小規模の記憶装置・結果の書きこみを制御するマスク、が含まれ、分岐機構などの制御を持たない。本モデルのSIMASD型計算機では、プロセッサ要素をグループ分けし、各グループにはグループに所属するプロセッサ要素からアクセスできる共有メモリを配置する。SIMASD型計算機と制御プロセッサは、共有メモリを介して通信を行う。制御プロセッサからSIMASD型計算機にデータを転送するには、共有メモリにデータを書き込んで、その共有メモリ上の値をプロセッサ要素が読み込むことによって行われる。その際、すべての共有メモリに同一の値を書き込むこと、特定の共有メモリに値を書き込むこと、ができる。これは、どちらもO(1)の時間でできる。SIMASD型計算機から制御プロセッサにデータを送る方法には、特定の共有メモリ上の値を制御プロセッサに転送する方法と、複数グループの共有メモリ上の同じアドレスの値に対して加算などの集約演算を行いながら回収する方法が存在する。集約演算は、ツリー状に値を集約するので、グループの数をnとするとO(log n)の時間がかかる。特定の共有メモリ上の値の回収も、本計算機では集約に用いるツリーを用いるので、同様の時間がかかる。

3. プログラミングモデル

本節では、flat-c言語で採用するプログラミングモデルを示す。

このプログラミングモデルでは、プログラマは、SIMASD型計算機で計算を行うには、1．SIMASD型計算機に分散させたい値をホストマシン上に配置し、2．SIMASD型計算機の各プロセッサ要素でどのような計算を行うかを記述し、3．どの結果をホス

トマシンに転送したいかを指定することによりプログラムを構成する。つまり、ユーザーは、ホストマシン上のメモリ領域（flat-c 言語では配列で表現する）とプロセッサ要素に存在する変数との対応を記述し、プロセッサ要素に存在するそれらの変数に対する演算を記述する。

ホストマシン上のメモリ領域とプロセッサ要素上の変数の対応、および各プロセッサ要素での計算を記述することにより、実際には、ホストマシンから値の転送、結果の回収、計算が行われる。その一連の流れをトランザクションという単位で扱う。1つのトランザクションは、1．SIMASD 型計算機・制御プロセッサ内の記憶領域へのデータの転送（フェーズ1）　2．SIMASD 型計算機による計算（フェーズ2）　3．結果の回収（フェーズ3）　から構成される。

トランザクションの例を図2に示す。

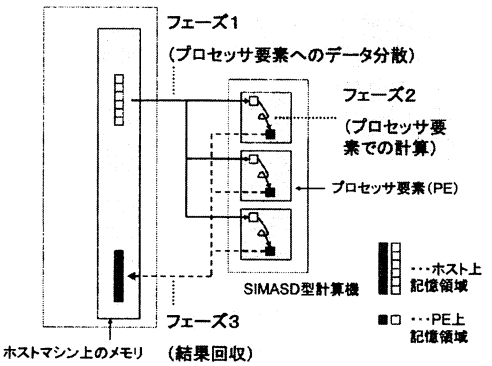

図 2　トランザクションの例

4. flat-c 言語

SIMASD 型計算機は、ホストマシン・SIMASD 型計算機間で必要な通信量に対して演算量が大きい計算に適している。例えば、(1) あらかじめ各プロセッサ要素に異なった値を置く、(2) すべてのプロセッサ要素に同一の値をブロードキャストしながら計算する、(3) 結果を集約しながら回収しホストマシンに送り返す、という計算である。これらの計算で性能を出すためには、ホストマシンと SIMASD 型計算機間で生じる通信を意識する必要がある。また、性能を出すためには、プロセッサ要素向けに記述したプログラムがどの程度の実行時間を消費するか、を意識する必要がある。既存の並列プログラミング言語では自由度が高く、各計算がどのように実際の計算機で実行されるか意識することが困難である。

flat-c 言語では、並列関数という概念を導入し、プログラマにこの2つの問題を意識させる。それにより、プログラムの文面から実際の計算機での性能を容易に推定できるようにする。並列関数の持つ構造は、ホストマシン・SIMASD 型計算機間での通信を明確にする。並列関数内での言語要素の拡張と制限は、計算が SIMASD 型計算機でどのように実行されるかを明確にする。まず、並列関数の構造を説明する。次に、SIMASD 型計算機に合わせた機能拡張・制限が並列関数内でどのように実現されているかを示す。

4.1 並列関数の構造

並列関数では、3節で示したプログラミングモデルを用いてプログラムを記述する。つまり、トランザクションという概念を並列関数に導入する。トランザクションの各フェーズに、並列関数の構成要素を対応させる(図 3)。トランザクションの各フェーズで生じる通信の種類は明確に分かれているため、並列関数の記述をみることにより、ユーザーはホストマシン・SIMASD 型計算機間の通信の種類を特定することができる。特に、ホストマシンと SIMASD 型計算機との間に発生する通信は、関数宣言部を見ることによってすべて知ることができる。flat-c では、関数が並列関数であることを示すために、関数パラメータ宣言の直後に as parallel 修飾子を付ける。

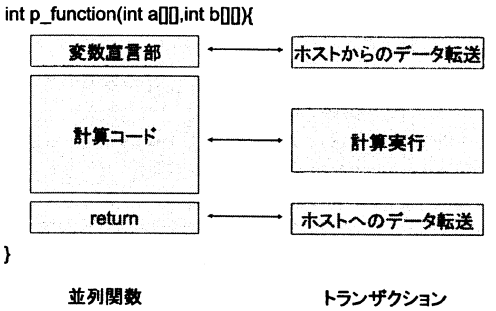

図 3　並列関数とトランザクションとの対応

以下では、トランザクションの各フェーズの並列関数内での記述法を示す。

4.1.1 ホストマシンからのデータ転送

ホストマシンから転送されるデータは、並列関数のパラメータ部に記述される。並列関数の先頭では、並列関数内で用いる変数を宣言できる。その際に、変数に、ホストマシンから転送されたデータを格納するということを明示的に指定することにより、ホストマシンから指定した変数へのデータ転送を表現できる。例を示すと、図4のようになる。

parallel 修飾子は、p_a がプロセッサ要素内の変数であることを示す。block 修飾子は、p_b が共有メモリ上の変数であることを示す。flat-c では、転送するデータは配列に置く。配列の次元と転送先の変数の型から、どのようにデータを分散させるかを決定する。

```
void p_func(int a[][],int b[])
 as parallel{
  parallel int p_a as a;
  block int p_b as b;
}
```

図 4　ホストマシンからのデータ転送の記述例

プロセッサ要素内の変数は、SIMASD 型計算機全体では、グループ数とグループ内のプロセッサ要素数を乗算した数存在し、2次元配列としてみることができる。ホストマシン上の2次元配列がプロセッサ要素に分散される場合、配列の第1次元をグループのインデックス、第2次元をプロセッサの要素のインデックスとみなして、対応するプロセッサ要素にデータを転送する。flat-c では、ホストマシン上の転送する配列の次元が転送先の次元よりも小さい場合にはある階層で値がブロードキャストされる、などのルールも規定する。

4.1.2 計算実行

並列関数内の変数宣言部と結果回収部との間に、トランザクションのフェーズ2（計算実行部）を記述する。ここには、プロセッサ要素内で行われる計算を記述する。flat-c では、計算実行部に記述できる C 言語の文法を定めている。計算機モデルの機能と対応した文法を提供し、文法と計算機モデルの機能との対応を明確にする。そして、プロセッサ要素・制御プロセッサの機能では実現できない機能の記述を制限する。例えば、一般の制御文を記述することはできない。

並列関数内では、一般の制御文の代わりに、並列制御文を用いることができる。並列制御文では、プロセッサ要素・制御プロセッサの機能を用いて実現できる制御を記述できる。例えば、固定回数のループを実行する制御文を記述できる。

4.1.3 結果回収

並列関数内には、return 文を、関数の最後に記述することができる。return 文では、プロセッサ要素内・共有メモリや制御プロセッサ内の記憶領域からホストマシンに転送するデータを指定する。転送するデータの型は、並列関数の返値の型と一致していなければならない。return 文に指定されたデータは、それが制御プロセッサ内の一時記憶領域に存在するデータでない限り、並列に存在する。よって、各プロセッサ要素や各共有メモリ上のデータは配列の1要素とみなされ、ホストマシン上のメモリ上の指定された配列にマッピングされ、転送が行われる。

4.1.4 トランザクションの発行

並列関数により、トランザクションを flat-c でどのように定義するかを示した。実際にトランザクションを発行するためには、並列関数を、逐次関数から呼び出す必要がある。並列関数の呼び出し部では、転送したいデータ、結果を回収するメモリ上の領域を配列の形で渡す。並列関数呼び出しの例を図5に示す。

```
int[][] p_func(int a[][],int b[][])
 as parallel{
  parallel int pa as a, pb as b, pc;
  pc = pa + pb;
  return pc;
}
int main(void){
  int ha[][],hb[][],hc[][];
  ...
  hc = p_func(ha,hb);
}
```

図 5　並列関数の呼び出し例

トランザクションが発行されると、配列 ha と配列 hb がホスト上のメモリから読み出され、SIMASD 型計算機に転送される。そして、計算が実行され、結果が配列 hc に格納される。並列関数のユーザーは、ホスト上の配列にデータを配置して、並列関数を呼び出すことにより、その配列に対して並列計算を行う。

4.2 並列関数の言語要素

ここでは、SIMASD 型計算機・制御プロセッサの各機能を利用するための並列関数の言語機能を示す。SIMASD 型計算機の中の通信機能や演算機能の、言語での表現を説明する。flat-c 言語では、原則として、並列関数の計算部分では SIMASD 型計算機内でハードウェアの機能を用いて実現できる機能のみをを言語レベルで提供する。それにより、計算部分の各記述に対して、ユーザーは利用する計算機モデル機能の種類を、プログラムの文面から判断することができる。

4.2.1 並列型

SIMASD 型計算機にはプロセッサ要素上の変数・共有メモリ上の変数が存在し、制御プロセッサにも一時記憶領域が存在する。並列関数内でそれらを区別するために、それぞれ、parallel,block,external という修飾子を変数に指定することにより、どの記憶領域に存在するかを指定する。

4.2.2 並列制御文

並列関数内の、計算部分には、計算式、および、並列制御文を記述することができる。並列制御文には、pif と pfor があり、pif はマスクの制御、pfor は、単純な繰り返しループなどを表現するために用いられる。pfor は、ネストすることができない。pfor は、繰り返し回数として、ホストマシンから受け取ったデータを利用することができる。pfor 内では、ループの先頭で変数宣言を行うことにより、制御プロセッサ内の記憶領域上のデータを、各ループの先頭で SIMASD 型計算機内の記憶領域に分散させることができる。これは、制御プロセッサ内の記憶領域から一部分づつデータを取り出して繰り返し計算を行うための機能である。たとえば、図6のように記述することができる。

```
void p_func(int a[][],int b,int c[][])
 as parallel{
  external int e_a[][] as a;
  parallel int p_c as c;
  pfor(int t = 0 ; t < b ; t ++){
    block int b_a as e_a[t];
  }
  pif(p_c > 0) p_c = 0;
}
```

図 6 並列制御文の記述例

4.2.3 共有メモリ

採用する計算機モデルでは、SIMASD 型計算機の各グループには、所属するグループ内のプロセッサ要素からアクセスできる共有メモリが存在することは既述したとおりである。しかし、共有メモリ上の変数に対して、直接演算を施すことはできない。プロセッサ要素上の変数に移してから計算する必要がある。また、共有メモリ上の変数にプロセッサ要素から値を書き込む場合、どのプロセッサ要素から書き込むかを指定する必要がある。flat-c では、block 変数に対しての演算はコンパイルエラーとなる。また、block 変数に対して、プロセッサ要素上の値を代入するときは、@演算子を用いて、どのプロセッサ要素が値を書き込むかを指定する。プロセッサ要素上の変数から共有メモリ上の変数へ代入する例を図 7 に示す。

```
int[] p_func(int a[][],int b[][])
 as parallel{
  parallel int p_a as a, p_b as b;
  block int result;
  result = (p_a + p_b)@0;
  return result;
}
```

図 7 共有メモリへの代入例

4.2.4 共有メモリ上の値の集約

結果の回収は return 文で記述することができるが、その際に、SIMASD 型計算機内に存在する、共有メモリ上の集約機能を用いることができる。また、共有メモリ上の値だけでなく、プロセッサ要素上の値の回収時に、同じプロセッサ要素インデックスの値に対して同様の集約演算を行うことができる。flat-c では、reduce 演算子を指定することにより、この機能を表現する。図 8 に reduce 演算子の記述例を示す。

reduce 演算子は、ブラケットの中に集約演算の種類を指定し、共有メモリにまたがる値の集約方法を記述する。上の例で、変数 result のグループ x・プロセッサ要素のインデックスが y の値を papluspb[x][y] とすると、返される配列 hostresult には図 9 の C 言語プログラムの計算結果が入る。

```
int[] p_func(int a[][],int b[][])
 as parallel{
  parallel int p_a as a,p_b as b,result;
  result = p_a + p_b;
  return reduce<sum> result;
}
```

図 8 reduce 演算子の記述例

```
int hostresult[], papluspb[][];
for(int y=0 ; y < max_pe ; y ++){
 for(int x=0 ; x < max_group ; x++){
  hostresult[y] += papluspb[x][y];
 }
}
```

図 9 集約演算の動作

5. コンパイラ

本節では、現在開発中の flat-c 言語コンパイラについて述べる。

flat-c 言語コンパイラは、flat-c 言語で記述されたプログラムを処理し、GCC などの C 言語コンパイラ用のコードを生成する。生成された C 言語のコードには、制御プロセッサを制御するためのインターフェイス関数を呼び出すコードを埋め込む。

flat-c 言語のプログラムは、採用する計算機モデルに沿っているならば、特定のアーキテクチャを想定しない。flat-c コンパイラは、対象とする制御プロセッサやプロセッサ要素が持つ機能を使えるかどうか推論し、実際の計算機に適したコードを生成する。pfor 文は、コンパイル時に静的に変数の値が決定するものなら記述できるが、その際、ループの形によって制御プロセッサのループ機能を利用できるならば、それを利用するコードを生成する。これは、プログラム文面からハードウェアの利用を見えにくくするので、コンパイラは、推論結果をレポートする機能をもつ。

SIMASD 型計算機用のコード部分は、対象とする SIMASD 型計算機のアセンブリ言語にコンパイルされる。flat-c 言語コンパイラは、制御プロセッサに、SIMASD 型計算機に送信するマシンコードを登録するインターフェイス関数、及び、登録したマシンコードを実際に SIMASD 型計算機に送信するインターフェイス関数を持つと過程する。コンパイラは、SIMASD 型計算機のアセンブリ言語を生成し、スケジューリングを行い、マシンコードを生成する。そして、そのコードを登録するインターフェイス関数の呼び出し、登録したコードを送信する関数の呼び出し、を生成される C 言語プログラムに埋め込む。

コンパイラは、コンパイル対象とするアーキテクチャのもつ SIMASD 型計算機のグループ数・プロセッ

サ要素数を保持しており、どのモードでコードが動作するかを特定する。そして、仮想モードでコードが動作する場合、並列関数呼び出し部分に、データを分割して計算を行うためのコードを埋め込む。

6. 関連研究

SIMD 型計算機用のプログラミング言語としては、C*[6] や MPL[1] がある。これらの言語では、プログラム内で並列型を持つ変数を定義することができる。これらの言語では、変数の型の種類に基づき自動的に値の分散が行われる。プログラミングの自由度は高いが、文面からホストマシン・SIMD 計算機間で生じる通信を推定することは困難である。

リストに対してデータ並列演算を行う言語としては、NESL[2] や*Lisp[4] が挙げられる。これらは関数型言語をベースとした言語で、リストの要素に対し並列演算を施すことができる。これらの言語では、リストの利用により、可変長のデータに対するプログラムを記述しやすい。しかし、実行時にプロセッサ要素へのデータの分散がどのように行われるか予測できない。

High Performance Fortran(HPF)[3] では、配列上のデータをどのように各プロセッサに分散するか指定する。HPF では、ユーザーが仮想的な計算機モデルを設定することができる。そのためさまざまな計算機モデル向けにプログラムを書くことができるが、実際の計算機をユーザーに意識させることが困難となる。

Warp 用のシストリックアレイコンパイラ[5] では、計算機モデルを設定しそれに合わせた言語仕様を設定している。このコンパイラでは、採用している計算機モデルはシストリックアレイマシンである。

7. まとめ

本論文では、超並列計算機向けプログラミング言語 flat-c を提案した。flat-c 言語は、プロセッサ間及び外部とのネットワークを制限することによりより多くのプロセッサを集積する計算機向けに、必要なデータ通信量に対して計算量が大きくなるようなアプリケーションを記述する言語である。これらの計算機向けに効率的なプログラムを書くためには、ユーザーは通信がボトルネックとなる場所を意識してプログラムを書く必要がある。flat-c は、これらの計算機のために、外部ネットワークとの通信量を文面から判断でき、並列計算機の構成をユーザーが意識してプログラムを記述できるような言語仕様を提供する。そのために flat-c では、並列関数という、関数の形により通信量・通信方法が決定し、計算機モデルに合わせた文法をその中に記述することのできる新しい関数概念を提供する。

現在は、flat-c 言語の仕様の見直し及びコンパイラの開発中である。今後はコンパイラの検証環境を整備し、コンパイラのコード生成部までを作成し動作させる予定である。コンパイラは、Free Software Foundation gcc3.3.3 及び Microsoft Visual C++ .Net 2003 で動作する C 言語コードを生成する予定である。

仕様の策定に関して今後行うことは、並列制御文の仕様の明確化、ホスト上の配列と並列関数内の変数のマッピング表記の見直しである。前者は、実際に記述するアプリケーションの性質によるので、どのようなループを記述する必要があるかを調査する必要がある。並列関数内の変数のマッピングは、現在の仕様では、配列の型と並列関数内の変数の型からマッピング方法を推論する。この方式ではユーザーのプログラムミスを検知しにくいため、マッピングをユーザーが明示的に指定する機能を追加する必要がある。

今後は、最適化の手法の開発が必要である。ホストマシンで実行される部分は、既存のコンパイラの最適化機能を利用できる。flat-c 言語コンパイラが考慮する最適化は、トランザクション内・トランザクション間での通信のオーバーラップ、・SIMASD 型計算機で実行されるコードの最適化等である。

8. 謝辞

本研究を行うにあたって、東京大学理学系研究科の牧野淳一郎博士、東京大学情報理工学系研究科の菅原豊博士より貴重な助言や指導を頂いた。ここに感謝の意を表する。

本研究は、文部科学省科学技術振興調整費　重要課題解決型研究等の推進「分散共有型研究データ利用基盤の整備」により実施された。

参考文献

1) *MasPar Parallel Application Language User Guide (Software Version 3.2)*. MasPar Computer Corporation, 1993.
2) Guy E. Blelloch. Nesl: A nested data-parallel language. *CMU-CS-95-170, School of Computer Science Carnegie Mellon University*, 1995.
3) High PerformanceFortran Forum. High performance fortran language specification version 2.0. 1997.
4) Guy L.Steele Jr. and W.Daniel Hillis. Connection machine lisp: Fine-grained parallel symbolic processing. *LISP and Functional Programming*, pp. 279–297, 1986.
5) MonicaS. Lam. *A Systolic Array Optimizing Compiler*. Kluwer Academic Publishers, 1989.
6) J.R. Rose and G.Steele. C*: An extended c language for data parallel programming. *Proc. Second Int'l Conf. Supercomputing, Vol.2*, pp. 2–16, 1987.