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

2007－HPC－111 (2)
2007／8／1

Sakura-C: 超並列計算機向けC言語と最適化

林崎弘成 † 西川徹 †† 菅原豊 †
稲葉真理 † 平木敬 †

本論文では，SIMASD 計算機向け言語 Sakura-C を提案する．より特化した専用言語に比べプログラムを容易にするため，C 言語に特殊構文として並列実行指定のみを追加し，通信は配列アクセスとポインタデリファレンスから自動生成する．高性能を達成するため，SIMASD 計算機に適した計算・通信パターンに記法を制限し，可能ならブロードキャストを用いる．コンパイラを実装し，評価を行った．密行列乗算の実行速度は手動最適化したアセンブリコードの 3.2 倍であった．

Sakura-C: An Extended C Language for Massively Parallel Computers

Hiroshige Hayashizaki ,† Toru Nishikawa ,††
Yutaka Sugawara ,† Mary Inaba † and Kei Hiraki †

We propose the Sakura-C programming language for massively parallel SIMASD computers. To make it easy to program SIMASD computers, Sakura-C extends C language by adding one special syntax for parallel execution and infers communications from array references and pointer dereferences. To achieve high performance, Sakura-C infers broadcast when possible, and declines programs that cannot be implemented by using computation/communication capability of SIMASD computers directly. We implemented a compiler of Sakura-C and conducted performance evaluation using dense matrix multiplication as a benchmark. The execution time of the compiler-generated code was 3.2 times longer than that of a hand-optimized assembly code.

1. はじめに

重力計算や密行列乗算など，データ量に対して計算量が多いアプリケーションに対しては，プロセッサとネットワークを単純化することでより多くのプロセッサを並列計算機内に集約し，演算ピーク性能を向上させるアプローチが有効である．

そのような設計の計算機の一つとして，Single Instruction Multiple And Shared Data (SIMASD) 型計算機がある．**図 1** に，SIMASD 型計算機の構成を示す．SIMASD 計算機は，(1) 多数のプロセッサ要素 (PE)，(2) 各 PE に付属するローカルメモリ (LM)，(3) 共有メモリ，(4) 制御プロセッサと，(5) その間のツリー状ネットワークから構成される．

SIMASD 型計算機は，プロセッサを単純化するために，全ての PE が同一の命令を実行する SIMD を採用している．ネットワークを単純化するために，PE 間，共有メモリ間の直接通信ネットワークは持たない．これらにより，演算器をより多く搭載することを可能にしている．

SIMASD 型計算機では，制御プロセッサ-共有メモリ間，共有メモリ-PE 間にはツリー状通信ネットワークが存在する．1 対 1 の通信の他，制御プロセッサから全共有メモリへ，共有メモリからその下に属する全 PE へのブロードキャスト及び共有メモリからホストへのリダクション演算が 1 対 1 通信と同コストで行える．

SIMASD 計算機は，汎用計算機であるホストマシンに接続して使用する．ホストマシンから SIMASD 計算機へデータを送信し，SIMASD 計算機の各 PE 上で並列に計算を行い，その結果をホストマシンに送信する，という形で SIMASD 計算機を用いた計算が行われる．

SIMASD 型計算機をプログラムする高級言語の設計では，SIMASD 型計算機の制約に合致するように通信・計算を記述でき，アーキテクチャの持つブロードキャストなどの機能を利用できることが重要である．

Flat-C[1] のようにアーキテクチャ特化言語とする場合は，対象アーキテクチャの特性を十分に反映でき

† 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† 有限会社 プリファードインフラストラクチャー
Preferred Infrastructure

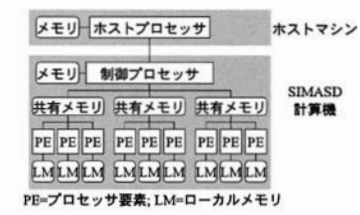

図 1 SIMASD 計算機

る．しかし，プログラムする側としては汎用言語に比べて記述しにくい．

汎用言語に近い言語とする場合は，SIMASD 型計算機に適さない通信・計算パターンを排除するか，プログラマに意識させることが，高い性能を達成するためには重要である．しかし，PE 間での相互通信が可能な環境を想定している UPC[2] などの既存の汎用並列言語では，データを各 PE に分散させ，PE 間の相互通信を行いながら計算するというパターンの記述に向いているため，そのままでは SIMASD 型計算機に適さない．

本論文では，SIMASD 計算機向けの拡張 C 言語 Sakura-C を提案する．Sakura-C では，C 言語を拡張し，SIMASD 計算機上での並列性を汎用言語に近い形で記述可能にする．SIMASD 計算機上で可能な通信パターンを配列アクセスなどのプログラムの文面から推論することによって，アーキテクチャ特化構文や明示的な通信プリミティブの呼び出しを伴わずに，通信を含むプログラムを記述できるようにする．

一方，Sakura-C コンパイラは通信にブロードキャストが利用できるか否かを推論し，可能ならブロードキャストを用いることにより，計算機の通信能力を活用する．SIMASD 計算機に適さない通信パターンは排除する．

2 章で Sakura-C の計算機モデルを提示する．3 章で Sakura-C の言語仕様を示し，マシンの機能とどのように対応しているのかを示す．4 章で Sakura-C コンパイラの実装と最適化について述べる．5 章で評価を行い，6 章で関連研究を示し，7 章でまとめを行う．

2. 計算機モデル

現在の Sakura-C で採用している計算機モデルを**図 2** に示す．図 1 の SIMASD 計算機の制御プロセッサがホストに統合されている．また，現在共有メモリ階層を利用したプログラミングをサポートしていないため，共有メモリが省かれている．

超並列計算機は，多数の PE と，各 PE に付属するローカルメモリ（LM）から構成される．ここでローカルメモリとは，レジスタも含めた，各 PE ごとに用意されたメモリ領域の総称である．

各 PE は SIMD 的に動作し，同時に同一の命令を

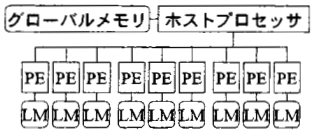

図 2 Sakura-C 計算機モデル

```
1: void func( double a[], double b[],
              double c[] ){
2:     int i;
3: #pragma sakura pfor
4:     for( i = 0; i < 512; ++i ){
5:         double x = a[i];
6:         int j;
7:         for( j = 0; j < 512; ++j ){
8:             double y = b[j];
9:             ...
10:        }
11:        c[i] = ...;
12:    }
13: }
```

図 3 Sakura-C サンプルコード

実行する．ローカルメモリ上のフラグに従って，一部の PE でのみ命令を実行することができ，これにより条件分岐を実現する．

ホストと各 PE とはネットワークで結ばれており，

(1) ホストから全 PE へ同じ値を送信 （ブロードキャスト）
(2) ホストから特定の PE へ値を送信 （個別送信）
(3) 特定の PE からホストへ値を送信 （個別回収）
(4) 全 PE から値を回収し，その総和・最大値などをホストへ送信 （リダクション）

の 4 つの通信を行うことができる．PE 間の直接通信は行えない．(1) と (2)，(3) と (4) とはそれぞれ同コストである．但し，現在のコンパイラではリダクションは記述できない．

ホストからどの値を送信し，またグローバルメモリ上のどこに回収した値を格納するかは，ホストがアドレスを計算して指定する．PE 側でどこに受信した値を格納するか，またどの値をホスト側に送信するかは，PE 側で指定する．

3. Sakura-C 言語

本章では，Sakura-C の言語仕様を示し，それがどのように 2 章に示したモデルに対応するのかを示す．**図 3** に Sakura-C のサンプルコードを，**図 4** にこのコードから生成される計算及びデータの分散を示す．

3.1 ループ並列化の記述

直前に `#pragma sakura pfor` と記述することで，

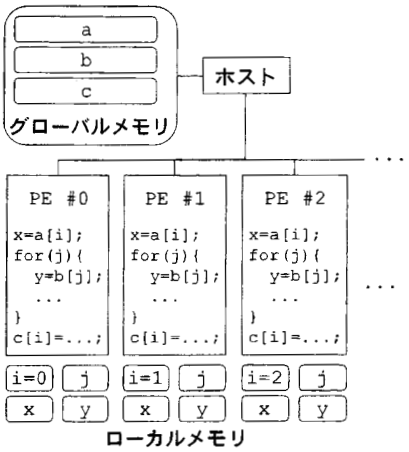

図 4 PE への割当

超並列計算機上で並列実行する for 文をプログラマが明示的に指定する（図 3 行 3-4）．この for 文を，pfor （parallel for） 文と呼ぶ．

pfor 文の中では，pfor 文の外側で宣言された変数あるいは配列の各要素に対し，書き込みと読み込みの両方を行ってはいけない．これは，イテレーション間にフロー依存・逆依存が存在しないことを含意する．ある変数に対し複数のイテレーションから書き込みがある場合，その実行順序は不定である．

pfor 文は for(i=定数; i<定数; i+=定数) の形で書く（i は任意の整数変数）．以下，説明のために pfor 文のループ変数は i であるとする．pfor 文内でループ変数 i に代入することはできない．pfor 文は break 文で抜けることはできない．

pfor 文の各イテレーションは各 PE で並列に実行される．現在は pfor 文の繰り返し回数と PE の数とが等しい場合のみをサポートし，各 PE にイテレーション 1 回を割り当てる．4.1 節で述べるように，実際の実装においては一部のコードがホスト側で実行されるが，意味的には PE 上で実行されると考えてプログラムしてよい．

pfor 文の外側のコードはホスト上で実行される．

3.2 ストレージの割当

変数宣言の場所によって，各変数をどのメモリ上に割り付けるかを区別する．

(1) pfor 文の外側で宣言された変数（図 3 a, b, c）はホスト側のグローバルメモリ上に確保される．pfor 文の内側でグローバルメモリ上の変数にアクセスすると通信が発生する．

(2) 内側で宣言された変数（図 3 x, y, j）は各 PE 上のローカルメモリ上に確保される．

(3) pfor 文のループ変数は，pfor 文の外側で宣言されているが，pfor 文の実行中はローカルメモリ上に確保され，PE から通信なしでアクセスできる．

3.3 通信の推論と生成

pfor 文内でグローバルメモリ上のデータに対して読み込みか書き込みを行う（図 3 行 5, 8, 11）と，通信が生成される．

読み込みの場合はホストから PE への通信が生成されるが，その際ブロードキャストと個別送信のどちらが生成されるかは以下の原理に従う：

(1) アクセスするデータのアドレスが PE ごとに異なる（可能性のある）場合，個別送信で各 PE に送信する．

(2) それ以外の場合（アドレスが全 PE で同一の場合），ブロードキャストで送信する．

この判定にはデータ依存及びコントロール依存関係を用いる．pfor 文のループ変数 i に，アドレス値が直接または間接に，データ依存またはコントロール依存する場合，(1) と判定する．ローカルメモリ上の配列変数に依存する場合は，現在ではサポートせず，エラーとする．これ以外の場合は (2) と判定する．配列添字にループ変数 i が出現するか否かで通信種別を判定する方法は，Li and Chen[3] が提案した．

図 3 の場合，j の値，b[j] のアドレスは全 PE で同一であり，8 行目の b[j] へのアクセスはブロードキャストが推論される．i の値，a[i]，c[i] のアドレスは各 PE で異なり，5, 11 行目の a[i]，c[i] へのアクセスはそれぞれ個別通信，個別回収が推論される．

ポインタ演算を使用することもできる．その場合，簡単なポインタ解析を用い，確実にグローバルメモリ上を指しているポインタのデリファレンスについて通信を生成する．グローバルとローカルのどちらのメモリを指しているか解析できない場合はエラーとする．

4. コンパイラ

4.1 通信に関するコード生成

ホストと PE 間の通信において，ホスト側の送信データ読出元及び受信データ格納先のアドレスはホスト側で計算する必要がある（2 章）．そのため，pfor 文中に書かれているプログラムから，ホスト側データのアドレス計算を抽出してホスト側で実行するようにした．この変換手順を以下に示す．この変換は 4 つ組に変換された中間表現に適用される．**図 5** に適用例を示す．

(1) pfor 文中の全てのコード及び変数のコピーをホスト側に作成し，ホスト側で実行するようにする．ホスト側で PE 上の並列実行を全てエミュレートすることになる．図 5 (b) 左で ' が付けられている変数はホスト側に作成されたコピー変数であることを示す．

各 PE で値が異なる式（a=i+1 など）は，PE 数

```
p = a + i;
x = *p;
q = b + 1;
y = *q;
z = f(x,y);
r = c + i;
*r = z;
```

(a) 変換前

ホスト側	PE 側
p̃' = a + i;	~~p = a + i;~~
x̃' = *p̃';	
SEND x̃';	x = RECEIVE;
q' = b + 1;	~~q = b + 1;~~
y' = *q';	
SEND y';	y = RECEIVE;
~~z' = f(x',y');~~	z = f(x,y);
r̃' = c + i;	~~r = c + i;~~
*r̃' = RECEIVE;	SEND z;

(b) コピー，dead code elimination 後

ホスト側	PE 側
for(i) p'[i] = a + i;	
for(i) x'[i] = p'[i];	
for(i) SEND x'[i];	x = RECEIVE;
q' = b + 1;	
y' = *q';	
SEND y';	y = RECEIVE;
for(i) r'[i] = c + i;	z = f(x,y);
for(i) *(r'[i]) = RECEIVE;	SEND z;

(c) 最終出力コード

図 5 アドレス生成に関するコード生成

をベクトル長とするベクトル値として扱い（例: $\vec{a'}$=i+1），最終的には for 文として出力される（図 5 (c) 左）．

通信の生成も行い，通信を生成すべきポインタのデリファレンスを SEND と RECEIVE の組に置き換える．

(2) Dead code elimination を適用することで，(1) でコピーしたコードのうちアドレス計算に不要な部分を削除する．PE 側のコードからも不要になったアドレス計算が削除される．

このような方式を用いたのはポインタ演算を可能にするためである．

4.2 通信プリミティブ呼出回数の削減

このままでは **図 6** (a) のように，メモリアクセス一回につき通信プリミティブが一回呼び出されることになる．ホストから PE 方向の通信では，通信プリミティブの呼び出しにかかるオーバーヘッドが問題になったため，以下のように複数の通信をまとめて通信

```
for( j = 0; j < n; j ++ ){
    send a_j;
    send b_j;
}
```

(a) 最適化前

```
for( j = 0; j < n; j ++ ){
    send {a_j, b_j};
}
```

(b) 同一ブロック内をまとめる

```
send {a_0, b_0, ..., a_{n-1}, b_{n-1}};
for( j = 0; j < n; j ++ ){
    ...
}
```

(c) ループ内をまとめる

図 6 通信回数の削減

プリミティブの呼び出し回数を削減した．PE からホストへの通信に関しては，性能上の問題になっていないため，適用していない．

(1) 同一ブロック内の，通信パターン（ブロードキャストか個別送信か）が一致する複数の通信を一つにまとめる（図 6 (b)）．

(2) ループ内の各繰り返しで発生する通信を，ループ開始前の一回にまとめる（図 6 (c)）．これは，繰り返し回数が定数の場合のみ，かつ 図 1 におけるホストから制御プロセッサへの通信にのみ適用する．制御プロセッサと共有メモリ間の通信に適用しないのは，まとめて通信した場合共有メモリ側のバッファ領域が不足するためである．

ここで，読み込みアクセスのあるデータに対する書き込みは pfor 文内には無いので（3.1 節より），以上のように pfor 文内で通信のタイミングを移動させても通信される値は変わらず，正当な最適化である．

5. 評価実験

1536 × 1536 正方行列の密行列乗算をベンチマークとして性能評価実験を行い，実行時間を測定した．

5.1 評価環境

評価には，SIMASD 計算機である GRAPE-DR[4] を用いた．**表 1** に現在の GRAPE-DR の仕様を示す．倍精度浮動小数は 72bit で表現される．

5.2 比較対象

以下の 3 つのコードを比較した．

(a) Sakura-C で記述されたコード（**図 7**）からコンパイラにより生成されたコード

(b) (a) と同じ方式（ループ構造・データ分散レ

☆ 倍精度，乗算:加算=1:1 の時

ホストマシン CPU	Opteron 3.0 GHz * 2
ホストマシン メモリ	2 GB
ホストマシン OS	Linux 2.6.9
GRAPE-DR 動作周波数	500 MHz
共有メモリ数	16
PE 数	512
	(1 共有メモリ当たり 32)
ピーク性能	256 GFLOPS ☆
制御プロセッサメモリサイズ	72bit * 1048576
共有メモリサイズ	72bit * 1024
ローカルメモリサイズ	72bit * 32 (レジスタ)
	+ 72bit * 256 (メモリ)
ホスト-制御プロセッサ間帯域	900MB/s (実効)
制御プロセッサ→共有メモリ帯域	4.2GB/s (72bit/1 clock)
制御プロセッサ←共有メモリ帯域	2.1GB/s (72bit/2 clock)
共有メモリ-PE 間帯域	4.2GB/s (72bit/1 clock)

表 1 GRAPE-DR の仕様

```
#define L 512
#define M 256
#define N 1536
void func( double A[L][M], double B[N][M],
double C[N][L] )
{
    int i;
#pragma sakura pfor
    for( i = 0; i < L; i ++ ){
        int j, k;
        double a[M];

        for( k = 0; k < M; k ++ )
            a[k] = A[i][k]; // (1)

        for( j = 0; j < N; j ++ ){
            double d = 0;
            for( k = 0; k < M; k ++ )
                d += a[k] * B[j][k]; // (2)

            C[j][i] = d; // (3)
        }
    }
}
```

図 7 Sakura-C による行列乗算

イアウト) を用いて，アセンブリ言語で記述し最適化したコード

(c) (a) とは異なる，共有メモリ階層を利用した方式を用いてアセンブリ言語で記述し最適化したコード．理論上の上限性能を達成できる方式であるが，現在のコンパイラでは共有メモリ階層をサポートしていないため，このようなデータ分散レイアウトは記述できない．

図 7 に，(a) で用いた Sakura-C ソースコードの概略を示す．512×256 の行列 A と，256×1536 の行列 B とを乗じて結果を C に格納する．B と C は転置されている．1536×1536 の行列をブロック化し，のこルーチンを複数回呼び出しての乗算を行った．実際に用いたソースコードは，これに手動で最内ループのアンロール等の最適化を施したものである．

このコードの計算及びデータの分散方法は以下の通りである (図 7 の (1) ～ (3) に対応する)：

(1) 行列 A の第 i 行 ($0 \le i < 512$) を PE i 上に個別送信し，ローカルメモリ上のバッファ a に格納する．
以下を $0 \le j < 1536$ に対して実行する:

(2) 行列 B の第 j 列を全 PE に放送する．各 PE では，ローカルメモリ上に格納されている A の第 i 行と放送された B の第 j 列との内積 d を計算する．

(3) d が C の i 行 j 列の要素に対応するので，各 PE から d の値を個別回収し，ホスト側メモリの C の対応する場所に格納する．

ここで，A の行数 512 は PE 数から，列数 256 は 1 列全体がローカルメモリに乗る最大サイズが 256 であることから設定した．

5.3 実行時間測定結果

表 2 に，それぞれのコードを GRAPE-DR 上で実行した場合の 1536×1536 正方行列乗算の実行時間を以下の項目ごとに示す．測定は 10 回行い平均値を取った．

	(a)	(b)	(c)
総実行時間	2.156	0.670	0.561
メイン演算	0.499	0.155	0.067
A 転送	0.176	0.178	0.174
B 転送	0.480	0.158	0.169
バッファ操作	0.764	0.147	0.131
初期化	0.235	0.029	0.020

表 2 密行列乗算実行時間 (単位: 秒)

A 転送 行列 A をホストから各 PE のローカルメモリまで転送する時間

B 転送 行列 B をホストから制御プロセッサのメモリまで転送する時間

メイン演算 行列 B を制御プロセッサから各 PE まで転送し，行列積を計算し，結果をホストまで送信する時間

バッファ操作 ホスト上で，GRAPE-DR へ行列 A, B を送信する際にバッファにコピーする時間，及び GRAPE-DR から回収したデータをバッファから行列 C にコピーする時間

初期化 GRAPE-DR の初期化，通信パターン・実行命令列の登録等の時間

5.4 考　　察

コンパイラによるコード (a) は，同方式でのアセンブラコード (b) と比べ以下の項目で性能差が見られる．

(1) 行列 B の転送時間
(2) メインの演算部の実行時間
(3) 転送用バッファへのコピー時間
(4) 初期化時間

(1) 行列 B の転送時間については，行列 B の同じ領域を複数回使用する際に，(b) では最初に一回だけホストから制御プロセッサのメモリに送信し，以降は制御プロセッサ-PE 間の通信のみを行うのに対し，(a) では毎回ホストから送信しているためである．これは，(b) を毎回 B をホストから送信するように改変すると B 転送の時間が 0.475 秒となり，(a) とほぼ同じ B 転送時間となることから裏付けられる．

(2) メインの演算部の実行時間については，PE 上で実行される命令数の差によるものと考えられる．PE 上で実行される命令数は，(b) では約 9.7×10^6 命令，(a) では約 6.2×10^7 命令であった．これは，現在のコンパイラのコード生成・スケジューリングの性能が低いためである．GRAPE-DR においては，命令の並列実行やベクトル長 4 のベクトル命令が使用できる．(b) はこれらの機能を有効に利用しているのに対し，(a) はこれらを全く利用していない．

(3) 転送用バッファへのコピー時間については，図 5 のように，アドレス計算を四つ組に分解しそれぞれ `for` 文に展開されていることが一因である．これは，手動でこれらのアドレス計算の `for` 文の一部を融合・式展開することで，(a) のバッファ操作の時間が 0.764 秒から 0.675 秒に短縮できたことから確かめられた．

(4) 初期化については，(b) は 1 回で済む初期化は 1 回のみ実行しているのに対し，(a) は同じ初期化を繰り返しているためである．

今回の結果では，GRAPE-DR の性能を十分に利用できていない．例えば，(b) の B 転送の項では行列 B の全要素を 1 回ずつ転送しているので転送量は $1536 * 1536 * 8 = 18$MB であり，ホストから制御プロセッサで 900MB/s のバンド幅が達成できていれば約 0.02 秒で転送できるはずである．これは，PCI-X 転送ルーチンの最適化が不十分なためである．

6. 関連研究

SIMASD 型計算機用のプログラミング言語としては，Flat-C[1] がある．Flat-C では，SIMASD 型計算機がサポートする通信・計算パターンと対応する記法を用意しており，SIMASD 型計算機の機能をプログラマ側が指定できるが，ANSI C 言語にはない特殊な構文を用いる必要がある．

命令制御機構やキャッシュを削減し演算器数を増加させた計算機向けの言語として，GPU プログラミング用言語 Cuda[5] がある．Cuda では，汎用言語に近い形でプログラミングが可能であり，対象とするアーキテクチャは SIMD である．しかし，Cuda の対象とする GPU の通信アーキテクチャは SIMASD 型計算機とは異なり，通信の最適化方法も異なる．

7. まとめ

本論文では，SIMASD 計算機向けの拡張 C 言語 Sakura-C を提案した．拡張 C 言語を用い，配列アクセスなどから通信を自動生成することで，より明示的に通信や計算の分散を記述する専用言語に比べ，プログラムのしやすさを確保した．ブロードキャスト利用の可否を依存関係グラフを用いて判定し，可能ならブロードキャストを利用することにより，効率の良い通信の記述を可能にした．

Sakura-C コンパイラを実装し，密行列乗算を用いて速度性能評価を行った．コンパイラ生成コードの実行時間は，同じ方式を手動でアセンブラで記述した場合の実行時間に比べ約 3.2 倍であった．

課題としては，PE 上で実行するコードの最適化，通信の削減，共有メモリ階層のサポートが挙げられる．

参考文献

1) 西川 徹，稲葉真理，平木 敬：flat-c: 超並列計算機向け C 言語の実現，情報処理学会研究報告，2005-HPC-103, Vol.2005, No.81, pp.163–168 (2005).
2) Carlson, W.W., Draper, J.M., Culler, D.E., Yelick, K., Brooks, E. and Warren, K.: Introduction to UPC and language specification, Technical ReportCCS-TR-99-157, IDA Center for Computing Sciences (1999).
3) Li, J. and Chen, M.: Compiling Communication-Efficient Programs for Massively Parallel Machines, *IEEE Trans. Parallel Distrib. Syst.*, Vol.2, No.3, pp.361–376 (1991).
4) GRAPE-DR Project: GRAPE-DR Project, http://grape-dr.adm.s.u-tokyo.ac.jp/.
5) NVIDIA Corporation: *CUDA Programming Guide 1.0* (2007).