アルゴリズム 61－6
（1998．3．20）

PRAMアルゴリズムのマルチスレッドアーキテクチャへのインプリメントと評価

佐藤和博 黒澤隆之 本田健治 中野浩嗣 林達也
名古屋工業大学電気情報工学科

これまでに多くの並列アルゴリズムがPRAM上で開発されてきている．しかし、PRAMは複数のプロセッサから均一にアクセス可能な共有メモリをもっているので、現実的な並列計算機のモデルではない．本稿では、PRAMよりも現実的なマルチスレッドアーキテクチャを提案する．そして、マルチスレッドアーキテクチャの能力を明らかにするために、PRAMアルゴリズムをインプリメントし、その性能評価を行う．

Implementing the PRAM Algorithms in the Multithread Architecture and Evaluating the Performance

Kazuhiro Sato, Takayuki Kurosawa, Kenji Honda, Koji Nakano, and Tatsuya Hayashi

Many parallel algorithms on the PRAM have been developed, so far. However, the PRAM is an unrealistic model of parallel computers, because it has a shared memory uniformly accessed by processors. The main contribution of this work is to introduce a multithread architecture, which is more realistic than the PRAM. To clarify the power of the multithread architecture, we implement PRAM algorithms in it and evaluate their performance.

1 はじめに

PRAM(Parallel Random Access Machine)は、複数のプロセッサと共有メモリから構成される並列計算機の理論モデルである[1]．各プロセッサは、共有メモリ上の任意のアドレスに単位時間でアクセス可能である．PRAMは、逐次計算機の理論モデルであるRAM(Random Access Machine)を自然に拡張した単純な並列計算機のモデルであり、並列計算の本質をとらえるためにもっとも適している．また、超立方体結合やメッシュ結合などのネットワーク結合の並列計算機などは、共有メモリへのアクセスを一部制限したPRAMとみなすことができるので、PRAMは最も抽象度の高い並列計算機であると考えられる．そのため、グラフ、幾何学、数値計算などの様々な問題を解くPRAM上の並列アルゴリズムが開発されてきており、並列計算に関する様々な知見が得られている．しかし、複数のプロセッサから均一にアクセスが可能な共有メモリ

が実現不可能であるため、PRAMアルゴリズムは、単なる机上だけのものであると考えられてきた.

本研究では、PRAMを想定して設計されたアルゴリズムを効率良く実行できるより現実的な並列計算機アーキテクチャであるマルチスレッドアーキテクチャを提案する. このマルチスレッドアーキテクチャの各プロセッサは、複数のスレッドを時分割で実行することにより、仮想的に並列計算を行うことがができる. １９７８年ごろから様々なマルチスレッドをベースにした並列計算機が実際に開発されている [3]. マルチスレッドアーキテクチャでは、プロセッサ台数以上のスレッドの処理を仮想的に同時に行うことができる. よって、多くのプロセッサが利用可能であることを前提とした超並列プログラムの実行にむいていると考えられる.

本研究では、マルチスレッドアーキテクチャの性能を解明するために、ワークステーション上にマルチスレッドアーキテクチャのレジスタトランスファレベルのシミュレータを開発した. そして、超並列プログラムとして、PRAM(厳密には、同一メモリセルへの同時読みだし同時書き込みを許すCRCW-PRAM) を想定して設計された並列アルゴリズムをマルチスレッドアーキテクチャにインプリメントし、その性能評価を行った. 具体的には、prefix sums問題とlist ranking問題を解く並列アルゴリズムをマルチスレッドアーキテクチャの機械語命令でコーディングし、その実行に要するサイクル数を評価した.

2 マルチスレッドアーキテクチャ

通常の逐次計算用のプロセッサは、命令ポインタを含んだレジスタセットを１つもち、命令ポインタの指す命令を順に実行するシングルスレッドアーキテクチャである. これに対して、マルチスレッドアーキテクチャで用いるプロセッサ (マルチスレッドプロセッサと呼ぶ) は、命令ポインタを含んだレジスタセットを複数持ち、各命令ポインタの指す命令を時分割で仮想的に同時実行するアーキテクチャである (図1). プロセッサが持つレジスタセットを $R_1, R_2, \ldots, R_m$ と表す. 図1に示したように、レジスタセットはプログラムである機械語命令列 $I_1, I_2, I_3, \ldots$ を共有し、各命令ポインタの指す機械語命令が順に処理される. プロセッサ

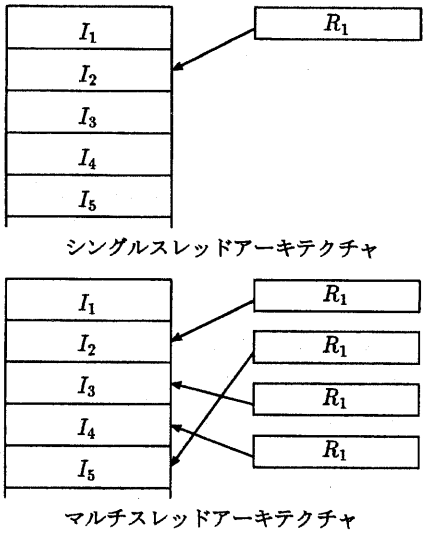

図 1: シングルスレッドアーキテクチャとマルチスレッドアーキテクチャ

は複数のレジスタセットを持つが、実行制御部は１つだけであり、各レジスタセットの命令ポインタが指す機械語命令を時分割で処理する.

各レジスタセット R_i $(1 \leq i \leq m)$ が t 番目 $(1 \leq t)$ に実行する機械語命令を I_i^t と表す. マルチスレッドアーキテクチャでは、次に表される順序で、機械語命令を処理する：

```
for t:=1 to ∞ do
   for i:=1 to m do
      processor executes I_i^t
```

つまり、１機械語命令ごとに処理されるスレッドが順に切り替わる. 内側のforループによる機械語命令の実行を１順の実行と呼ぶことにする.

通常、プロセッサアーキテクチャでは、サイクル時間を短くし、機械語命令の１つあたりの処理時間を短縮するために、パイプライン処理が行われる. 例えば、機械語命令の実行は、次の４つのステージにに分けることができる. ステージ毎にオーバーラップして、複数の命令を同時実行するのがパイプライン処理である.

IF 命令フェッチ.

ID 命令デコードおよびレジスタフェッチ.

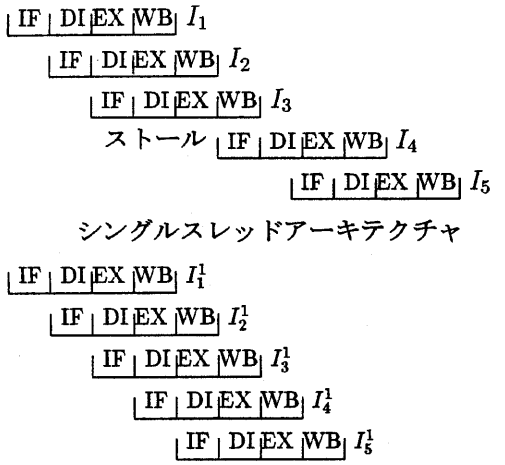

図 2: パイプライン処理

EX 実行.

WB メモリアクセス.

シングルスレッドアーキテクチャでは、前後の命令の依存関係や条件分岐命令などにより、パイプラインの乱れがおこり、効率が上がらないことがある (図 2). これに対して、マルチスレッドアーキテクチャでは、実行制御部に投入される命令の前後に直接依存関係がない. また、条件分岐命令で、分岐するか否かは、1 順前の実行で判定可能なので、パイプラインの乱れがおきない. また、パイプラインハザードの検出や回復のための回路や、分岐予測のための複雑な機構が必要ない. また、1 順に実行される命令に依存関係がないので、実行制御部での処理をさらに細分化してパイプラインの段数を増やすことができる. これにより、サイクル時間、つまり 1 命令あたりの実行時間を短縮することができる. 但し、アクティブなレジスタセットがパイプラインの段数より少なくなることが起きやすくなる. この場合、実行制御部に間断ない命令の投入ができなくなり、効率が悪くなるという欠点を持つ.

マルチスレッドプロセッサを複数ならべた並列計算機を考える. p 台のプロセッサ $P_1, P_2, \ldots, P_p$ がそれぞれ m 個のレジスタセットを持つとする. P_i の m 個のレジスタセットを $R_{i,1}, R_{i,2}, \ldots, R_{i,m}$ と表すことにする. レジスタセットの総数は mp 個なので、全体で mp 台のプロセッサが仮想的に同時に動作しているとみなすことができる. 但し、実際には p 台のプロセッサしかなく、各レジスタセットの命令は m サイクルに 1 度実行されるだけなので、並列処理による加速の効果は、高々 p 倍である.

各レジスタセットの情報交換のための通信を行うが、送信元と受信先のレジスタセットが同一プロセッサに属していれば、プロセッサ内の通信により実現することができる. プロセッサは 1 つの異なるプロセッサにあれば、プロセッサ間での通信が必要となる. マルチスレッドプロセッサでは、各レジスタセットの命令が処理されるのは m サイクルごとである. よって、あるレジスタセットの命令が送信受信命令であった場合、その送受信命令は m サイクル以内に完了すればよい. よって、プロセッサ間通信のネットワークのレイテンシが大きくてもよいことが予想される.

3 マルチスレッドアーキテクチャのシミュレータ

本節では、マルチスレッドアーキテクチャの性能評価のために、ワークステーション上に作成したマルチスレッドアーキテクチャのシミュレータについて述べる.

マルチスレッドプロセッサの命令セットとして、基本 RISC アーキテクチャである DLX [2] を採用し、通信命令と同期命令を追加することにより拡張した. 通信命令は、送受信先のプロセッサとレジスタセットの番号をそれぞれ指定することにより行われる. 同期命令では、全レジスタセットが sync 命令を実行するまで、一時停止するバリア同期を実現する.

パイプライン処理の詳細については決めなかったが、現在のプロセッサアーキテクチャのパイプラインが 10 段程度なので、各ステージをさらに細分化することを想定し、4 倍の 40 段と仮定した. つまり、1 つの機械語命令が実行制御部に投入されてから、処理が完了するまで 40 サイクル必要となる. パイプラインの段数が大きくなるほど、サイクル時間を短縮することができる. しかし、アクティブなレジスタセットがパイプラインの段数より少ないと、実行制御部に間断無く機械語命令を供給することができなくなり、アルゴリ

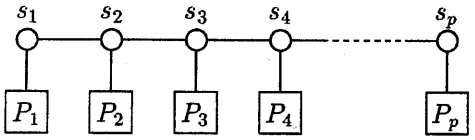

図 3: ネットワーク

ズム実行のサイクル数での評価は増大する．つまり、パイプラインの段数が増えるにしたがって、プログラム実行に要するサイクル数は増加する．サイクル時間は、デバイス技術に依存するので今回は評価しないが、パイプラインの段数が増えるにしたがって、減少することが期待できる．よって、パイプラインの段数を増やすことによるサイクル数の増加より、サイクル時間の短縮の度合の方が多ければ、段数を増やした方がよいと言える．

マルチスレッドアーキテクチャのシミュレータでは、もっとも通信能力が弱い１次元アレイ状の結合を採用した．図 3は、本稿で用いた１次元アレイのネットワークを示している．p 個のプロセッサ $P_1, P_2, \ldots, P_p$ はそれぞれスイッチ $s_1, s_2, \ldots, s_p$ と接続しており、プロセッサ間通信は、スイッチを経由しておこなわれる．スイッチ間の通信用リンクは双方向であり、１語＝３２ビットのデータに送信先のプロセッサを示すヘッダがついたものがそれぞれの向きに同時に転送可能である．この通信用リンクは、プロセッサを構成する VLSI チップ外に置かれることを想定し、プロセッサ内部の通信の４倍のペナルティを科すことにする．つまり、隣接したスイッチ間のヘッダ付き１語のデータ転送に４サイクル必要であると仮定する．したがって、s_i にあるデータを s_j のデータに転送するには、少なくとも $4|i-j|$ サイクル必要となる．また、例えば、あるスイッチが、プロセッサと右側のスイッチから同時にデータが送られ、その２つのデータを左に転送しなければならない場合が考えられる．通信用リンクには一度に１つのデータしか送れないので、転送できないデータが発生する．このようなデータはスイッチで保存し、各スイッチは送信すべきデータのうち、最も目的地の遠いものを優先的に転送するものとする．また、各スイッチで保存できるデータ数には上限を設けない．

PRAM アルゴリズムをマルチスレッドアーキテクチャへインプリメントするときには、n 台の PRAM プロセッサ $PE(0), PE(1), \ldots, PE(n-1)$ を p 台のマルチスレッドプロセッサ $P_0, P_1, \ldots, P_{p-1}$ に均等に割り当てる．つまり、各 $P_i (0 \leq i \leq p-1)$ は、$\frac{n}{p}$ 個のレジスタセット $R_{i,0}, \ldots, R_{i,\frac{n}{p}-1}$ を用い、それぞれ、$PE((i-1)\cdot\frac{n}{p}), \ldots, PE(i\cdot\frac{n}{p})$ の動作を実現する．PRAM の共有メモリをマルチスレッドプロセッサのローカルメモリへの割り当ても重要な問題である．しかし、後述の今回扱うアルゴリズムは、PRAM プロセッサが担当する共有メモリのメモリセルが明白である．例えば、大きさ n の配列 $a(0), a(1), \ldots, a(n-1)$ に対して、$PE(i)$ は $a(i)$ を担当する．よって、この PRAM プロセッサの担当をもとに、共有メモリのローカルメモリへの割り当てを行うことにする．また、入力データは、あらかじめ各ローカルメモリに与えられているものとし、入力や出力のために要する時間は無視する．

4 PRAM アルゴリズムとマルチスレッドアーキテクチャへのインプリメント

ここでは、インプリメントの対象とした prefix sums 問題と list ranking 問題を解く PRAM アルゴリズムについて説明する．

prefix sums とは、n 個の要素 $a(0), a(1), \ldots, a(n-1)$ が与えられたときに、i 番目の prefix sum $a(0)+a(1)+\cdots+a(i)$ を全ての i $(0 \leq i \leq n-1)$ を求める問題である．ここでは、各要素は１語＝３２ビットの整数とする．

prefix sums 問題を解く並列アルゴリズムには様々なものがあるが、今回は、単純で効率良い次の PRAM アルゴリズムを採用する．

```
PREFIX-SUMS1
for j:=0 to log n-1 do
   for i:=1 to n-1 do in parallel
      a(i):=a(i)+a(i-2^j)
```

図 4に PREFIX-SUMS1 の動作を示す．アルゴリズムの動作の詳細は [4](pp.32–24) を参照されたい．各 $a(i)$ の計算に $PE(i)$ を割り当てると、このアルゴリズムは n 台のプロセッサで、$O(\log n)$ 時間で prefix sums を

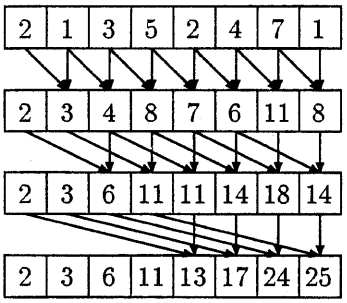

図 4: PREFIX-SUMS1 の動作

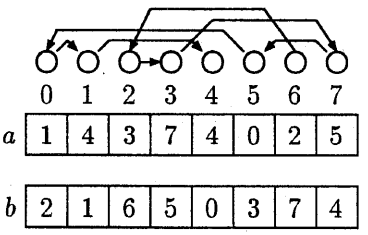

図 5: List ranking 問題

求めることができる．この n 台の PRAM アルゴリズムをマルチスレッドアーキテクチャにインプリメントすると、外側の for ループの最後の実行で、離れたプロセッサにあるデータをアクセスすることになり、通信に多くの時間が必要であることが予想される．

PREFIX-SUMS1 を PRAM で実行すると、n 個の要素に対して、n 台のプロセッサを用い、$O(\log n)$ 時間で prefix sums を計算する．プロセッサ台数と計算時間の積である並列アルゴリズムのコストは $O(n \log n)$ であり、自明な最適逐次アルゴリズムの計算時間 $O(n)$ より大きい．つまり、オーダ的に最適な加速を達成していない．しかし、各プロセッサに複数個のデータを担当させるとコストを最適逐次アルゴリズムの計算時間と一致させることができる．n 個の入力を $\frac{n}{s}$ 個のデータからなる s 個のグループに分割する．つまり、i 番目のグループ $(0 \le i \le n-1)$ を $A_i = a(i \cdot \frac{n}{s}), a(i \cdot \frac{n}{s} + 1), \ldots, a((i+1) \cdot \frac{n}{s} - 1)$ $(0 \le i \le s-1)$ と表す．次の並列アルゴリズム PREFIX-SUMS2 は s 台のプロセッサを 1 台ずつ A_i に割り当て、prefix sums を求める．

PREFIX-SUMS2

Step 1 各 A_i でローカルに prefix sums を逐次計算する．ここで、$sum(A_i) = a(i \cdot \frac{n}{s}) + a(i \cdot \frac{n}{s} + 1) + \cdots + a((i+1) \cdot \frac{n}{s} - 1)$ $(0 \le i \le s-1)$ とする．各 $sum(A_i)$ も同時に計算されていることに注意せよ．

Step 2 $sum(A_0), sum(A_1), \ldots, sum(A_{s-1})$ を入力とする各 prefix sum $S_i = sum(A_0) + sum(A_1) + \cdots + sum(A_i)$ を PREFIX-SUMS1 を用いて計算する．

Step 3 Step 1 で求めた $A_i (1 \le i \le n-1)$ の prefix sums に S_{i-1} を加算する．

PRAM は、Step 1 と Step 3 を $O(\frac{n}{s})$ 時間、Step 2 を $O(\log s)$ 時間でで実行できる．よって、総計算時間は $O(\frac{n}{s} + \log s)$ である．したがって、$s \le \frac{n}{\log n}$ のとき、s 台のプロセッサ用い、$O(\frac{n}{s})$ 時間で prefix sums 問題を解くことができる．このときのコストは $O(n)$ であり、最適逐次アルゴリズムの計算時間と一致する．

list ranking 問題は、n 個のノード $0, 1, \ldots, n-1$ のリストが与えられたときに、各ノードのランク、つまり、リストの先頭までのパスの長さを求める問題である．各ノード i $(0 \le i \le n-1)$ の次のノードが $a(i)$ に与えられる．ノード i がリストの先頭の場合、$a(i) = i$ である．

次の並列アルゴリズム LIST-RANKING は、各 $b(i)$ にノード i のランクを求める．詳細は [4](pp.34-35) を参照されたい．

```
LIST-RANKING
for i:=0 to n-1 do in parallel
   if a(i)=i then b(i)=0 else b(i)=1
for j:=1 to log2n do
   for i:=1 to n-1 do in parallel
      b(i):=b(i)+b(a(i))
      a(i):=a(a(i))
```

この並列アルゴリズムは、$a(i)$ と $b(i)$ を更新するために、それぞれ、$a(a(i))$ と $b(a(i))$ をアクセスする．ここで、配列 a と b にほぼランダムにアクセスするため、多くの通信が必要である．List ranking 問題について

も、コスト最適化の手法が知られているが、かなり複雑であり、ここでは扱わない.

5 シミュレータによるサイクル数の評価

マルチスレッドアーキテクチャに PREFIX-SUMS1、PREFIX-SUMS2、及び、LIST-RANKING をインプリメントしたときの実行に要するサイクル数を様々な入力データ数とプロセッサ台数に対して測定した. 明らかに、PREFIX-SUMS1 と PREFIX-SUMS2 の動作はデータ数 n とプロセッサ台数 p のみに依存し、データの中身に依存しない. よって、各 n と p の組合せに対して、ランダムな入力データに対して1度だけシミュレーションを行った. LIST-RANKING の動作は入力データに依存するので、5通りの完全ランダムなリストに対してサイクル数の測定を行い、その平均を求めた. ただし、測定の結果異なるリストによるサイクル数の差はいずれの場合も 1% 以下であった.

表1に各データ数 n、プロセッサ台数 p、各プロセッサのレジスタセット数 $\frac{n}{p}$ の場合の PREFIX-SUMS1 の実行に要したクロック数を示す. また、図6に加速率、つまり、

$$\frac{\text{プロセッサ台数1のときのクロック数}}{\text{プロセッサ台数}p\text{のときのクロック数}}$$

の各データ数における変化を示す. この値は、理想的な並列計算が行われた場合、p となる. データ数 n が大きくなるほど、理想的に近い加速率が得られている. $n = 512$ のとき、p を増加させると、$p = 8$ のときに加速が鈍化し、$p = 512$ のときはクロック数が増加に転じる. これは、プロセッサ間の通信に多くの時間が必要となるためである. $n = 16\text{k} = 16384$ のときは、$p = 64$ のときに加速が鈍化する. データ数が増えるほど、加速が止まるプロセッサ台数は大きくなる. これは、同じプロセッサ台数では、データ数が増えるほど、各プロセッサのでアクティブなレジスタセットの数 $\frac{n}{p}$ が増加し、制御実行部に間断無く命令が供給されるためだと考えられる.

表2に、各データ数 n、プロセッサ台数 p、の場合に、LIST-RANKING の実行に要したクロック数を示す. また、図7にプロセッサ台数と加速率の関係を示す. プロセッサ台数1のとき、サイクル数は PREFIX-SUMS1 とさほど変わらないが、プロセッサ台数が増えても加

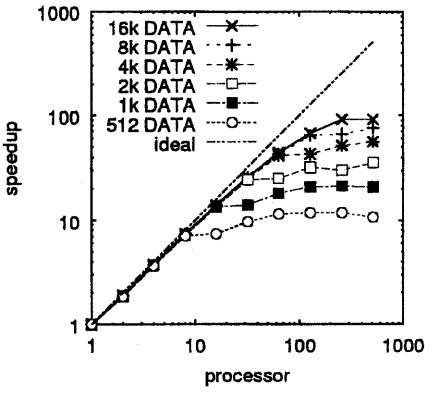

図 6: PREFIX-SUMS1 のプロセッサ台数と加速率

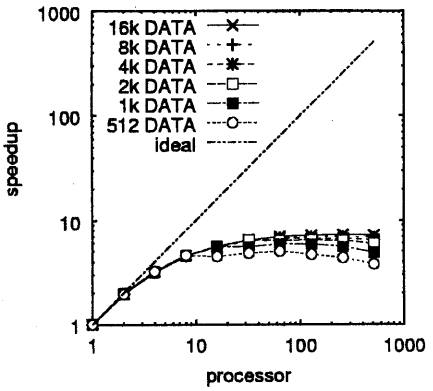

図 7: LIST-RANKING のプロセッサ台数と加速率

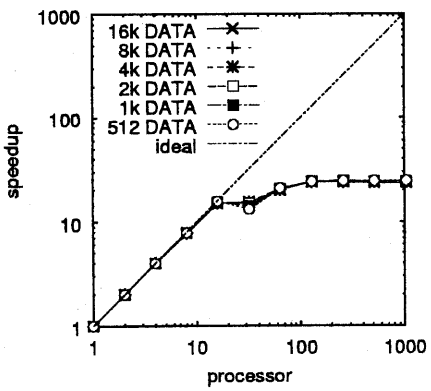

図 8: PREFIX-SUMS2 のプロセッサ台数と加速率 (総レジスタセット数 =1k)

表 1: PREFIX-SUMS1 の性能評価

n \ p	1	2	4	8	16	32	64	128	256	512
512	95250	51712	26175	13471	12999	9798	8318	8108	8092	8922
1k	211986	114176	57727	29631	15711	15175	11672	10094	9991	10124
2k	466962	249856	126207	64639	34111	18975	18439	14631	15462	13099
4k	1019922	542720	273919	140031	73599	40639	24478	23783	19783	18181
8k	2211858	1171456	590847	301567	157951	86655	51646	34140	33255	29239
16k	4759918	2514944	1267711	646143	337407	184063	108414	70717	51871	51366

表 2: LIST-RANKING の性能評価

n \ p	1	2	4	8	16	32	64	128	256	512
512	124416	63637	38808	26993	27683	26071	24777	26637	28523	32771
1k	275456	141096	86402	60277	49437	49541	47166	47080	49424	56433
2k	614160	309452	190037	133000	109457	95010	96277	94585	95948	102629
4k	1314816	675009	415114	291202	240157	207311	195000	197509	197324	201621
8k	2879457	1462270	899493.5	632238	522749	451621	423620	410643	415445	419412
16k	6111232	3129343	1919327	1339389	1101826	947328	882453	853613	841498	851325

表 3: PREFIX-SUMS2 の性能評価：総レジスタ数 1k

n/d \ p	1	2	4	8	16	32	64	128	256	512	1024
1k/1	319486	158720	79999	40767	21279	20743	15848	13528	13467	13608	13783
2k/2	370675	184320	92799	47167	24479	23943	18247	15571	15487	15628	15763
4k/4	429031	213504	107391	54463	28127	27716	20983	17923	17805	17940	18075
8k/8	511951	254976	128127	64831	33311	34244	24871	21331	21149	21244	21379
16k/16	643999	321024	161151	81343	41567	45380	31063	26827	26517	26532	26667
32k/32	874303	436224	218751	110143	55967	65732	41863	36499	35933	35788	35923
64k/64	1301119	649728	325503	163519	82655	104516	61879	54523	53445	52980	53115

表 4: PREFIX-SUM2 の性能評価：総データ数 64k

m \ p	1	2	4	8	16	32	64	128
1	16256573	16259137	8135003	4073865	2044111	1029941	523480	270808
2	16259338	8135202	4074060	2044298	1030112	523618	270940	145160
4	8299205	4156107	2085383	1050711	533967	276139	147763	84227
8	4287404	2151115	1083675	550554	284539	152066	86435	54523
16	2301318	1197335	609191	315750	169606	97205	61879	45782
32	1572103	812456	421191	226279	129543	104516	82173	67389
64	806591	421440	226655	130015	82655	70748	61982	60446
128	840063	452992	259263	163519	117183	105500	99230	105277

速率は上がらない．これは、LIST-RANKINGがほぼランダムなプロセッサ間通信を行うためだと考えられる．さらに、データ数nが増えても、加速率はごくわずか増加するだけである．

表3に、総レジスタセット数を1kに固定し、各データ数n、プロセッサ台数p、各プロセッサのレジスタセット数を$\frac{1k}{p}$として、PREFIX-SUMS2の実行に要したクロック数を示す．ここで、$d = \frac{n}{1k}$が1レジスタセットあたりの入力データ数である．PREFIX-SUMS1のPREFIX-SUMS2の同一条件である$n = 1k$の場合で比較すると、PREFIX-SUMS2の方が複雑な処理を行っているため、約50%サイクル数が増加している．データ数nが大きくなるにしたがって、PREFIX-SUMS1よりPREFIX-SUMS2の方がサイクル数が大幅に少なくなる．$n = 16k$、$p = 16$の場合、PREFIX-SUMS1の337407サイクルに対して、PREFIX-SUMS2は41567サイクルであり、約1/8にサイクル数が短縮した．これは、Step 1とStep 3で行われる逐次加算は高速に行うことができ、また、Step 2でのprefix sumsの並列計算は1k個だけのデータを対象として行われるためである．したがって、PRAMで行われるコスト最適化の手法は、マルチスレッドアーキテクチャでも有効であるといえる．また、図8にPREFIX-SUMS2のプロセッサ台数と加速率の関係を示す．プロセッサが16台までは、データ数に関係なく、理想的な加速を達成しており、16台より多くなると、加速が止まる．これは、64プロセッサの場合、レジスタセット数は$\frac{1k}{32} = 32$なので、パイプラインの段数40より大きく、実行時にパイプラインを埋めることができないためである．16プロセッサの場合、レジスタセット数は$\frac{1k}{16} = 64$なので、パイプラインはほとんど埋まっていると考えられる．

表4に、入力データ数を$n = 64k$に固定し、プロセッサ台数p、各プロセッサのレジスタセット数をmとして、PREFIX-SUMS2の実行に要したクロック数を示す．いずれのプロセッサ台数においても、レジスタセット数が64のときに最小となっている．これは、パイプラインの段数である40段より大きく、かつ、1スレッドあたりのデータ数が最大(つまり、総レジスタセットが最小)であるためと考えられる．

6 むすび

マルチスレッドアーキテクチャのシミュレータを用いて、prefix sums問題とlist ranking問題を解く超並列アルゴリズムの性能評価を行った．prefix sums問題を解く並列ルゴリズムは、理想的に近い加速率が得られることがわかった．list ranking問題を解く並列アルゴリズムは、ほぼランダムな通信が頻繁行われるため、通信のオーバヘッドが大きく、理想的な加速は得られなかった．また、PRAMで用いられるコスト最適化の手法がマルチスレッドアーキテクチャでも有効であることがわかった．

謝辞

本研究は、(財)電気通信普及財団の助成金による．

参考文献

[1] A. Gibbons and W. Rytter. *Efficient Parallel Algorithms.* Cambridge University Press, 1988.

[2] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 1990.

[3] Robert A. Iannucci. *Multithreaded Computer Architecture: A Summary of the State of the Art.* Kluwer Academic, 1994.

[4] Michael J. Quinn. *Parallel Computing:Theory and Practice.* McGraw-Hill, Inc., 1990.