プログラミング 18－2
(1998. 3. 23)

超並列項書換えシステムの実装と評価

平田寛道　　五百蔵重典　　緒方和博　　二木厚吉

北陸先端科学技術大学院大学 情報科学研究科

923-12 石川県能美郡辰口町旭台1丁目1番

{h-hirata,ioroi,ogata,kokichi}@jaist.ac.jp

あらまし　項書換えシステムは，代数仕様の直接実行など等式論理を基礎としたさまざまな分野に応用可能な計算モデルである．このシステムには潜在的に並列性が内在し，それを利用して並列書換えを行なえば，逐次書換えに比べ大幅な効率向上が期待できる．本稿では，並列項書換え抽象機械 (Massively Parallel TRAM) の設計と実装について記述し，予備実験による性能評価について報告する．

キーワード　並列書換え，抽象機械，超並列計算機

Implementation and Evaluation of Massively Parallel Rewriting

Hiromichi Hirata　　Shigenori Ioroi　　Kazuhiro Ogata　　Kokichi Futatsugi

Graduate School of Information Science
Japan Advanced Institute of Science and Technology

1-1 Asahidai, Tatsunokichi, Ishikawa 923-12, JAPAN

{h-hirata,ioroi,ogata,kokichi}@jaist.ac.jp

Abstract　Term rewriting systems can be used as computational models to implement language processors and theorem provers based on equational logic such as algebraic specification languages. Since the systems inherently contain concurrency, it is possible to improve their rewriting speed drastically by implementing the systems on a massively parallel computer so as to take advantage of the concurrency. In this paper, the design and implementation of Massively Parallel TRAM are described, which has been designed to be executed on a massively parallel computer. Its preliminary performance results are also reported.

key words　parallel rewriting, abstract machines, massively parallel computers

1 はじめに

項書換えシステム [5] は，等式で記述された論理の世界を自然な形で計算の世界に結びつける事ができる計算モデルであり，等式論理の定理証明，代数仕様記述や関数プログラムの直接実行等，さまざまな分野に導入されている．また，その実際の計算機上への実装という面から見ても，その相性は非常によく，ナイーブな実装は比較的容易に行う事ができる．しかし，そのような性質とは対照的に，一般にその実行効率は悪い．特に計算原理となる書換え一つ一つに対し，数多くの書換え規則の中から入力項と一致するパターンを見つけ出す作業が必要となるため非常に効率が悪く，またこの部分の効率は項書換えシステム全体の効率にも大きく影響する．この書換え効率を改善するための研究は数多くなされているが，そのなかで TRAM [6, 7] は項書換えシステムの実行効率に重点をおいて実装された抽象機械で，弁別ネットに代表される多くの要素技術が盛り込まれている．

項書換えシステムには潜在的に並列性が内在し [13]，それを引き出しマルチプロセッサや超並列計算機に実装して並列書換えを行なえば，逐次書換えに比べ大幅な書換え効率の向上が期待できる．Parallel TRAM [8, 9] は，TRAM を並列書換えが可能なように，逐次書換えの指定しか出来なかった **E-戦略**を拡張し，並列書換えの指定を可能とした**並列 E-戦略**や並列書換えのための抽象機械命令の追加などを行ない，4 プロセスを使用して TRAM の 2 倍程度の性能向上を達成している．

しかし，項書換えシステムを超並列計算機，あるいは分散計算機環境上に実装した例はほとんどない．本稿では Parallel TRAM を基に，TRAM を分散メモリ計算機上で動作できるよう拡張した超並列 TRAM の設計，および分散メモリ型超並列計算機である Cray T3E システム上への実装に関して記述する．また，超並列 TRAM の評価について記述する．

2 基礎概念

本節では，超並列 TRAM の基礎概念である項書換えシステム [5]，TRAM [6, 7] および Parallel TRAM[8, 9] について記述する．

2.1 項書換えシステム

項書換えシステムは項の書換えを計算の基本とした計算モデルである．与えられた項を書換え規則に基づいて書き換えることで，それ以上書き換えられない項をもとの項に対する計算結果として得ることができる．項書換えシステムは，項の集合と，項を書換えるための書換え規則の集合の対で定義される．たとえば，次の書換え規則 $\mathcal{R}$：

$$
\begin{aligned}
&padd(X,0) \to X\\
&padd(X,s(Y)) \to s(padd(X,Y))\\
&pfib(0) \to 0\\
&pfib(s(0)) \to s(0)\\
&pfib(s(s(X))) \to padd(pfib(s(X)),pfib(X))
\end{aligned}
$$

が与えられているとき，項 $padd(pfib(s(0)),pfib(0))$ をこの書換え規則に与えると，次のような書換えが起こる．

$$
\begin{aligned}
&padd(pfib(s(0)),pfib(0)) \to padd(s(0),pfib(0)))\\
&\to padd(s(0),0) \to s(0)
\end{aligned}
$$

この書換えによって，項 $s(0)$ が最終結果として得られる．

2.2 TRAM

TRAM (Term Rewriting Abstruct Machine) は項書換えを対象とした抽象機械で，項書換えシステムの実行効率に重点をおいて設計および実装されている．そのために，**E-戦略**の使用による書き換え順序の制御，**戦略リスト**を用いたリデックス検索の高速化などを行なっている．

E-戦略は，演算子ごとに書換えの順番を指定する事ができる戦略である．書換え順序の指定は数列を用いる．この数列の各要素は，

0: 全体項簡約

n: n 番目の引数項簡約 ($0 < n \leq$ 引数の個数)

となる．この E-戦略を用いることの利点は，次のような状況で起こる．

次のような書換え規則があるとする．

$$
\begin{aligned}
&if(true,X,Y) \to X\\
&if(false,X,Y) \to Y
\end{aligned}
$$

これは，書換え戦略によっては効率の悪い書換えとなり得る．例えば最内戦略で書換えを行ったとすると，3 つの引数すべてを書換えた後に全体項の書換えを行う．もしここで第 1 引数の書換えの結果が $true$ だとすると，Y の書換えは無駄になってしまう．ここで，if の戦略を (10) と指定する事で，第 1 引数の書換えを行った後，全体項の書換えを行うので，無駄な書換えを生じさせないようにできるのである．

戦略リストは，E-戦略で指定された書換えの順序を制御するために用いられるリスト構造で，マッチン

⟨ParallelLocalStrategy⟩ ::= ( ) | (⟨SerialElem⟩ * 0)
⟨SerialElem⟩ ::= 0 | ⟨ArgNum⟩ | ⟨ParallelArgs⟩
⟨ArgNum⟩ ::= 1 | 2 | ... | n
⟨ParallelArgs⟩ ::= { ⟨ArgNum⟩$^+$ }

図 1: 並列 E-戦略の定義

グプログラムのラベル列となっており，入力項のコンパイラで，マッチングプログラムと E-戦略から構築され，SL 領域に格納される．抽象命令のインタプリタは，この戦略リストの順番に従ってマッチングプログラムを実行し，書換えを行ってゆく．

2.3 Parallel TRAM

Parallel TRAM は TRAM をマルチプロセッサ上で実行できるよう拡張したもので，E-戦略をユーザが並列性を明示できるように拡張した**並列 E-戦略**を用いている．また，戦略リストはこの並列 E-戦略で指定された並列書換えを反映できるように拡張されている．

並列 E-戦略は，E-戦略に並列書換えの指定を追加したもので，アリティnの演算子fに対して，図1のように定義されている．

⟨ParallelArgs⟩ が並列書換えのために追加した定義で，その要素を並列に書換えを行わせることを表す．また，⟨SerialElem⟩ は，通常の E-戦略と同様に，その要素を左から順に書換えを行う．

Parallel TRAM はマルチプロセッサを対象としており，その構成は図2のようになる．

それぞれのプロセッサには，抽象命令のインタプリタと，書換えの際動的に内容が変化する 4 つの領域 (コード，スタック，戦略リスト，変数バインディング) を複製し，抽象機械のレベルで仮想的に割り付ける (この1ブロックを「プロセスユニット」とする)．また，内容が動的には変化しない領域 (弁別ネット，右辺の雛型) は，グローバル領域に格納される．また，各ユニットの動作状態を保持するテーブルとして「プロセス状態」というテーブルをグローバル領域に配置している．

また，メインとなるプロセスユニットを 1 つ決め，このユニットに書換え規則のコンパイラ，入力項のコンパイラを配置し，書換えの前処理をこのユニットで逐次に行っている．

Parallel TRAM の戦略リストは，並列 E-戦略で指定された並列書換えを反映させるために，TRAM の戦略リストに対し次のような拡張が行われている．

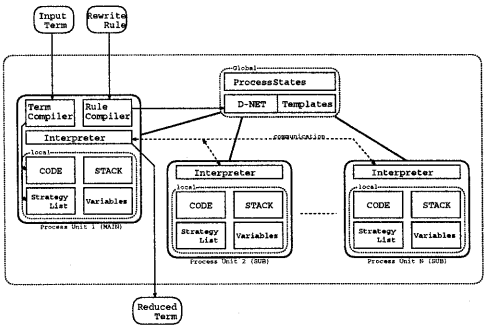

図 2: Parallel TRAM の構成

1. 引数項書換えは，その区切りを識別できるようにブロック化されている．
2. 並列指定された引数項書換えは，その最後の引数項書換えを除いて全て先頭に「*FORK*」が付加される．
3. 並列書換えの終りには，「*JOIN*」が付加される．

FORK および *JOIN* は，並列書換えを実現するために新たに付け加えられた抽象命令であり，並列書換えの指定時にはこれらの命令に処理を移せるよう，戦略リストの適切な位置にこれらの命令へのラベルが挿入される．そして，これらのラベルの挿入は上記 2 および 3の作業を機械的に行うことで可能となる．また，戦略リストの雛型も同様の作業で作成できる．

3 超並列項書換えシステムの設計

本節では Parallel TRAM を基に，TRAM を分散メモリ型計算機上で動作出来るよう拡張した超並列 TRAM [10, 11] の設計について記述する．Parallel TRAM はマルチプロセッサを対象としているが，本稿では分散メモリ型計算機を対象とするため，Prallel TRAM の構造・動作をそのまま用いることは出来ない．従って，Parallel TRAM の設計を基に，分散メモリ型の計算機上で動作する超並列 TRAM を設計する．

3.1 基本構成

Parallel TRAM はプロセス間のデータ移動を出来るだけ最小化するために，共有メモリ型の計算機ではどのプロセスからでもすべてのメモリを直接参照することが可能であることを利用し，可能な限りデータそ

のものでなくデータへのポインタなどを渡している．しかし，分散メモリ型の計算機では，プロセスごとに独立したメモリを持っており，プロセス間の直接メモリ参照は不可能であるか，または複雑な処理を用いなければならない．本研究で使用する超並列計算機 Cray T3E では，分散メモリを共有メモリとして利用する機構も用意されているが，システムに大きく依存するため，それを用いた場合プログラムの移植性が落ちる．本研究は分散計算機環境での TRAM の高速化に応用が可能であると考えられるため，システムに依存しない形での設計を行う．そこで，超並列 TRAM では分散計算機環境でよく利用されているメッセージパシングを利用して必要なデータを全て送受信する事で，プロセス間の直接メモリ参照を無くし，プロセス間のデータ授受の処理をできるだけ単純にさせる．これにより，各プロセスを書換えにできるだけ専念させ，処理効率の向上を図ることにする．

次にプロセス管理であるが，Parallel TRAM ではプロセス管理領域を共有メモリ上に置き，全てのプロセスからそれを操作していた．しかし，分散メモリではやはりこの方法は使えない．そこで，メッセージパシングシステムでよく利用されるプロセス管理手法である，Master-Slave (あるいは Master-Worker) モデルを利用したプロセス管理を行なう．このモデルは，仕事を行う Slave と，Slave に仕事を割り当てる Master とで構成される．超並列 TRAM では Master に，Slave のプロセス状態を管理するための機構を備え，これを用いて各 Slave に仕事を割り当てる．

超並列項書換えシステムの構成は図3のようになる．Master には，入力項のコンパイラと，それに必要な書換え規則のコンパイラ，および4つの領域 (コード，戦略リスト，弁別ネット，右辺の雛型) を用意している．さらに，Slave の状態を管理するための機構を持ち，Master はこれを利用して，各 Slave の状態の管理，仕事の割当てを行う．また各 Slave には，書換え規則のコンパイラ，抽象命令を解釈・実行するインタプリタと，書換えに必要な7つの領域 (コード，スタック，戦略リスト，変数，弁別ネット，右辺の雛型，書換え候補リスト) を用意している．

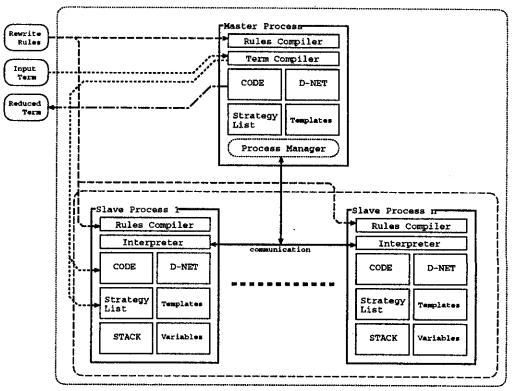

図 3: 超並列項書換えシステムの構成

3.2 INFO

超並列 TRAM ではメッセージパシングを利用してデータの授受をおこない，プロセス間の直接メモリ参照をなくしている．したがって，並列項書換えに際し，部分項の書換えに必要なデータは全て相手プロセスに渡される．Parallel TRAM ではこの部分項は，それを持っている全体項のアドレスを持っており，部分項の書換えが終了すると結果はそのアドレスに返される．しかし，分散メモリの場合この方法では，全体項と部分項が異なるプロセス上に存在する事になり，異なるプロセスのアドレスを部分項が持つ事になる．この場合，全体項を持つプロセスで GC (ガーベージコレクション) が起き項のアドレスが変わる時，部分項が保持しているアドレスも変更しなければならないため，GC 時にプロセス間で同期をとらなければならず，またアドレスの変更と言う作業も必要となる．このため，GC のオーバーヘッドが非常に大きくなり，書換え効率が非常に落ちる．このようなオーバーヘッドを回避するために参照テーブルを導入し，部分項ではアドレスでなくこのテーブルの参照を行わせる．これにより，プロセスごとに非同期に GC が行えるようになり，また部分項が保持する全体項の参照先を変更せずに済む．

超並列 TRAM ではこの領域を戦略リスト中に「INFO」として埋め込む事にする．これは戦略リストの構築時に FORK の数だけ戦略リスト中に埋め込まれる．この領域を独立して確保する場合，必要な領域のサイズが不明であり，最大 FORK 可能数だけ確保しなければならないが，このようにする事で，不必要な参照テーブルの領域を確保せずに済む．

3.3 プロセス管理

超並列 TRAM では，Master が各 Slave の状態を管理し，その状態に応じて次の動作を決定し，それを Slave に知らせる．Slave 状態には *Idle*, *Wait*, *Busy* があり，その定義は次のようになり，その状態は図4

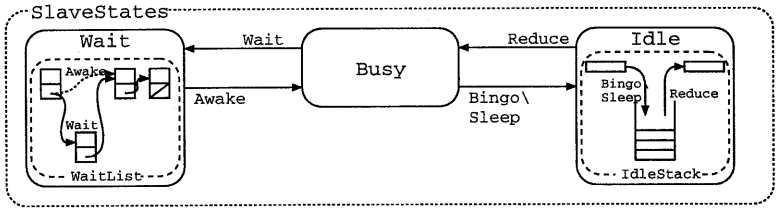

図 4: Slave の状態遷移

のように変化する.

Idle 何も処理をせず, 他プロセスからの書換えの割当を待っている状態. Slave から FORK の要求があった場合, 部分項の書換えを行わせる子プロセスとして優先的に割り当てられる. Idle Stack で管理する.

Wait FORK した部分項の書換えの結果が戻って来るのを待っている状態. Slave から FORK の要求があった場合, Idle Stack に Idle 状態のプロセスがない時, 部分項の書換えを行わせる子プロセスとして割り当てられる. Wait 状態はいつ Busy 状態に遷移するか不定であるため, Idle 状態のように Stack でなく, List で管理する. これを Wait List と呼ぶ. Master から Awake コマンドを受け取る事で, Busy 状態となる.

Busy 書換えを行っている状態. 全ての書換えが終れば Idle 状態になり, Wait 命令が実行されると Wait 状態となる.

3.4 動作の詳細

本節では, 超並列 TRAM の動作の詳細を述べてゆく. 超並列 TRAM の動作は以下のようになる.

1. 書換え規則のコンパイル
   書換え規則が与えられると, それは全てのプロセスに渡され, それぞれのプロセスでコンパイルされる.

2. 入力項のコンパイル
   入力項が与えられると, それはまず Master に渡され, そこでコンパイルされる. コンパイルされたコードと戦略リストは, Slave の一つ ($Slave_a$ とする) に渡され, $Slave_a$でコンパイルが始まる.

3. 書換え
   Master からコードを受け取った $Slave_a$は, 戦略リストからコードのラベルを取り出し, そのコードの実行を行ってゆく. 戦略リスト中に並列書き換えに関する命令 (Fork, Wait, Exit), 及び書き換え終了を表す命令 (Bingo) が現れた場合の動作は以下のようになる.

   (I) Fork
   書換え中に $Slave_a$で Fork 命令が実行されると, $Slave_a$ は Master に FORK コマンドを送り, 部分項の書換えを割り当てるための, 子プロセスを要求する. Master は空きプロセスを探し, あれば ($Slave_b$とする) その番号を $Slave_a$に返す. また, $Slave_b$に, $Slave_a$から部分項の受け取りを指示する. 空きプロセスが無い場合, -1 を $Slave_a$に返す. $Slave_a$は $Slave_b$に, 部分項の書換えを割り当てる. $Slave_b$は部分項を受け取り, その書換えを始める.

   (II) Exit
   書換え中に $Slave_b$で Exit 命令が実行されると, 部分項の書換えを割り当てた $Slave_a$に, 書換えの結果を返す. 次に Master に, 部分項の書換えが終わり, Idle 状態に移った事を知らせる. Master は $Slave_a$が Wait 状態なら $Slave_a$に Awake コマンドを送り, $Slave_a$に書換えを続けさせる.

   (III) Wait
   書換え中に $Slave_a$で Wait 命令が実行されると, Fork した書換えを割り付けた子プロセス (この場合 $Slave_b$) から, 書換えの結果が送られているか確認する. 送られてきている場合, それを受け取る. Fork した書換えがすべて戻ってきていれば, 次の書換えに移る. そうでなければ, Master に Wait 状態に

SlaveNumber \ WaitNum	1	2	------	i	------	m
1	-1	-1		-1		-1
2	1	0		-1		-1
.....						
j	1	2		k		-1
.....						
n	-1	-1		-1		-1

図 5: ForkTable

移った事を知らせる．Master は ForkTable (図 5) を参照し，$Slave_a$のこの Wait がいくつの Fork の結果を待っているかを調べ，それが 0 なら $Slave_a$に Awake コマンドを送る．また，また 1 つ以上の結果を待っているのなら $Slave_a$を WaitList に入れる．$Slave_a$では，Master から Awake コマンドが送られて来るまで，Fork した書換えの結果を受け付ける．

(IV) Bingo
書換え中に Bingo 命令が実行されると，その時点でのマッチングプログラムを Master に返し，Master はそれを入力項の書換え結果として出力する．

4 実装と評価

以上のように設計された超並列 TRAM により，どの程度書換え効率が改善されるかを見るために，SGI Cray Research 社の T3E システム (プロセス数: 128)[14] 上に，分散計算機環境で良く利用されるメッセージパシングライブラリである MPI[1, 2, 3, 4] を用いて実装し，いくつかの性能評価を行なった．

まず，TRAM と超並列 TRAM における逐次書換えの性能の比較を行う．そのために，次のような計算を行わせた．

1. TRAM におけるフィボナッチ数列 $fib(34)$ の計算．
2. 超並列 TRAM，逐次書換えにおけるフィボナッチ数列 $fib(34)$ の計算．

その結果は表 2のようになる．

この結果から，逐次書換えに関して，超並列 TRAM は TRAM とほぼ遜色ない処理能力を有している事が

	書換え時間 (s)	r/s	speed up
TRAM	223.3	247911	1
超並列 TRAM	235.7	234908	0.95

表 2: 逐次書換え性能の評価

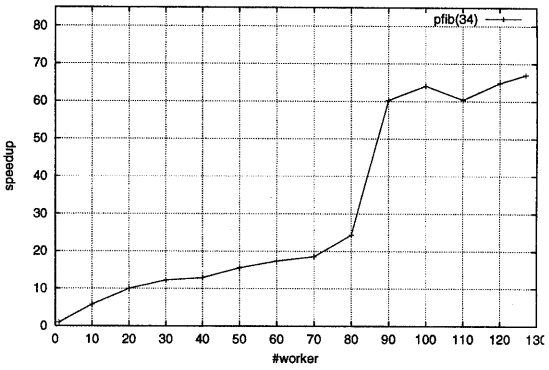

図 6: $pfib(34)$ の速度向上比

確認できる．若干の速度低下は，Master-Slave 間のデータ転送にかかるオーバーヘッドのためと考えられる．

次に，基本性能を評価するために，以下の点に注目した性能評価を行なった．

1. 使用 PE 数による速度向上の変化
使用 Slave 数を 1〜127 個で変化させ，$fib(34)$ を並列計算させた．その結果は表1のように，また，使用 Slave 数と Speed UP の関係は図6のようになる．この結果から，超並列 TRAM は最大で TRAM の 67 倍の速度向上を得られることがわかる．また，Slave 数が 80 個と 90 個の間で格段に性能が向上しているが，これは FORK の成否によるものと考えられる．FORK が失敗した場合，その部分項の書換えは FORK しようとしたプロセスで行なわれるので，そのプロセスの書換え時間は FORK が成功した場合に比べ，単純計算で 2 倍の時間を要する．FORK の失敗が複数のプロセスで生じた場合，これらの時間が微妙にずれて積み重なり，グラフ上に現れるような差となるものと考えられる．

2. FORK 数による速度向上の変化
評価1で用いた書換え規則では，その並列度を制御するために書換え規則中にしきい値を設けて

	TRAM	超並列 TRAM													
Slave数	1	1	10	20	30	40	50	60	70	80	90	100	110	120	127
書換え時間 (秒)	223	232.3	38.9	22.4	18.2	17.4	14.3	11.5	12.0	9.1	3.72	3.50	3.71	3.45	3.80
r/s (x10,000)	24.7	23.83	142.5	247.6	304.0	317.5	386.5	480.2	460.6	603.0	1489	1583	1493	1603	1656
speed up	1	0.96	5.77	10.0	12.3	12.9	15.6	17.4	18.6	24.4	60.3	64.1	60.4	64.9	67.0
FORK 失敗回数	-	88	65	44	44	38	31	28	18	9	0	0	0	0	0

最大FORK数：88

表 1: 逐次型 TRAM と超並列 TRAM の性能評価：$fib(34)$ の演算

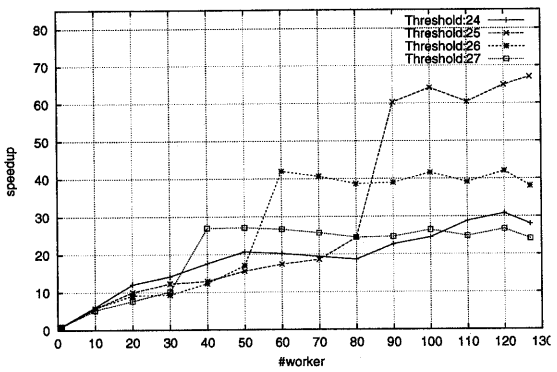

図 7: しきい値を変化させた時の Speed UP の変化

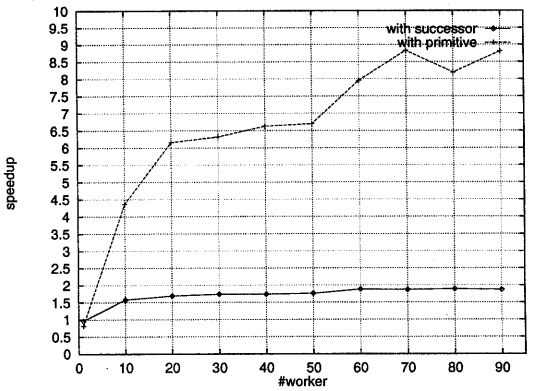

図 8: 転送データ量による Speed UP の変化

ある．評価1では，fib の引数が 25 以下になった場合逐次の書換えを行なうようにして，$fib(1)$ などの過度の FORK を抑えてある．このしきい値によって計算中に発生する FORK の回数が変化するが，この FORK の回数の違い，つまり書換え規則中のしきい値の設定によって，書換え速度がどのように変化するかを観察した．その結果は図7のようになる．$fib(34)$ を入力項として与え，しきい値を 24〜27 で変化させた．しきい値が 24 の場合，FORK は最大 143 回発生した．これにより，Slave を 127 個用いた場合でもすべての FORK が成功することはなく，その速度向上は低い伸びとなる．しきい値を 25，26，27 と変化させた場合，FORK の最大回数はそれぞれ 88，54，33 となる．従って，グラフ上でも Slave をそれらの個数用いた場所で速度向上が高く伸びている．この伸びの高さは，FORK の最大回数が多いほど高くなっている．つまり FORK がすべて成功するなら，その回数が多いほど書換え性能が向上することになる．次に，FORK がすべて成功した場合の速度向上について考えてみる．この場合，単純に考えてその FORK 数倍だけ性能が向上して良いはずである．しかし，実際に得られた値はそれよりもかなり小さい．この差は，メッセージ転送にかかるオーバーヘッドのせいであると考えられる．さらに，FORK の回数が増すにつれ，この差が増大している．これは，FORK の回数が増すにつれ，Slave から Master へのメッセージが増加し，そのメッセージ量に対して Master の処理が追い付かない状況となるため，Slave での Master のメッセージの処理待ちが長くなってゆくためではないかと考えられる．

3. メッセージ転送量による速度向上の変化

最後に，項のサイズによってどのように性能に差が出るかを観察した．入力項として $fib(25)$ をプリミティブと successor 関数を用いて与えた場合，どのように性能に差が出るかを観察した．なお，しきい値は 14 とし，これにより FORK は 88 回発生する．この結果は図8のようになる．successor 関数を用いた場合，項のサイズがプリミティブを用いた場合に比べ非常に増大するため，プロセス間を転送するデータのサイズが大きくなる．そのため，プロセス間のデータ転送にかかるオーバーヘッドが非常に大きくなり，このように性能に大きな差が出るものと考えられる．

5 関連研究

項書換えシステムは，様々な分野への応用が可能であるという性格上，理論・実装両面にわたり盛んに研究が行われている分野の一つである．特に，項書換えを並列に動作させ，その実行効率をあげる研究は近年盛んに行われている．

Gouguen らは，これまでの von Neumann の計算モデルに代わる新しい計算モデルとして，Concurrent Term Rewriting [13] を提唱し，これを RRM (Rewrite Rule Machine) の計算モデルとして用いている．RRM は，書換えを直接を行う計算機である．RRM は非常に多くのプロセッサで構成されており，特別な技術を用いずに通信コストを抑えるため，プロセスの構成をプロセス数で 4 つの粒度に分け，粗粒度の要素は細粒度の要素を複数合わせる構造となっている．また，そのレベルによって書換えのモードを MIMD モードと SIMD モードで使い分けている．ここで用いられている Concerrent E-strategy は，本研究で用いた並列 E-戦略をサブセットとして含んでいる．

また，項書換えシステムをあらゆるレベルで並列に行わせるような並列書換えの実装についての研究 [12] が，Kirchner らによって行われている．書換え規則はその構造が木構造あるいは DAG (Directed Acyclic Graphs) 構造となるが，そのあらゆる深度で同時に書換えを行える部分を探しだし，その書換えを行わせるものである．この設計にはいくつかの問題があり，特に各深度間でのメッセージ通信が非常に膨大な量となるという問題がある．

6 おわりに

本稿では，Parallel TRAM を基に，TRAM を超並列計算機上で並列書換えを行なわせるよう拡張した超並列 TRAM を設計した．また，この超並列 TRAM を Cray T3E システム (プロセス数:128) 上に MPI を用いて実装し，その性能を評価した．その結果，TRAM と比較して最大 67 倍の速度向上が確認できた．また，評価 2から，プロセス数，FORK 数が増すほど，並列書換え時のオーバーヘッドが増すことがわかった．さらに評価 3から，超並列 TRAM は通信メッセージ処理時間に対し，書換え処理時間の割合が大きい計算ほど書換え効率が向上することが確認できた．今後は，並列書換え時のオーバーヘッドを減少させること，つまり Master の負化の軽減やメッセージ通信のさらなる効率化をはかる必要がある．

参考文献

[1] Gropp, W., Lusk, E. and Skjellum, A.: USING MPI: Portable Parallel Programming with the Message-Passing Interface. The MIT Press. (1996)

[2] Snir, M., Otto, S. W., Lederman, S. H., Walker, D. W. and Dongarra, J.: MPI: The Complete Reference. The MIT Press. (1996)

[3] Pacheco, P. S.: Parallel Programming with MPI. Morgan Kaufmann Publishers. (1997)

[4] MPI: http://www.mcs.anl.gov/mpi/index.html

[5] 二木厚吉，外山芳人：項書換え型計算モデルとその応用．情報処理 Vol. 24，No. 2．情報処理学会．(1983)

[6] Ogata, K., Ohhara, K. and Futatsugi, K.: TRAM: An Abstract Machine for Order-Sorted Conditional Term Rewriting Systems. Proc. of the 8th Intl. Conf. on Rewriting Techniques and Applications. LNCS **1232** Springer-Verlag. (1997) 335–338

[7] Ogata, K. and Futatsugi, K.: Implementation of Term Rewritings with the Evaluation Strategy. Proc. of the 9th Intl. Sympo. on Programming Languages: Implementations, Logics, and Programs. LNCS **1292** Springer-Verlag. (1997) 225–239

[8] 近藤勝：並列項書換え抽象機械：Parallel TRAM の設計と実装．修士論文．JAIST．(1997)

[9] Ogata, K., Kondo, M., Ioroi, S. and Futatsugi, K.: Design and Implementation of Parallel TRAM. Proc. of the Third Intl. Euro-Par Conf. LNCS **1300** Springer-Verlag. (1997) 1209–1216

[10] 平田寛道，緒方和博，二木厚吉：超並列計算機を用いた項書換えシステムの高速化．第 14 回 (1997 年度) 大会論文集．日本ソフトウェア科学会．(1997) 413–416

[11] 平田寛道：Parallel TRAM を基にした超並列 TRAM の実装と評価．修士論文．JAIST．(1998)

[12] Viry, P. and Kirchner, C.: Implementing Parallel Rewriting. Proc. of the Intl Workshop on Programming Language Implementation and Logic Programming. LNCS **456** Springer-Verlag. (1990) 1–15

[13] Goguen, J., Kirchner, C. and Meseguer, J.: Concurrent Term Rewriting as a Model of Computation. Proc. of the Workshop on Graph Reduction. LNCS **279** Springer-Verlag. (1986) 53–93

[14] Cray T3E: http://www.cray.com/products/systems/crayt3e