計算機アーキテクチャ　95－15
(1992. 8. 19)

科学技術計算用データ駆動計算機SIGMA-1における最適化技法の評価

平木　敬　　島田俊夫　　関口智嗣
(東京大学)　(電子技術総合研究所)

命令レベルデータ駆動計算機SIGMA-1におけるコードの最適化の効果を評価する。コードの最適化手法として、ループをアンフォールド実行するために必要な機能を統合化した命令の導入、ループ不変変数の実現を効率化する手法、命令レベルの負荷分散についてデータ駆動計算機SIGMA-1を用いて評価を行なった。

つぎに、大規模なプログラムを安全かつ効率良く並列実行するために必要なプログラム文脈に依存する最適化技法として、非同期性と多重代入、資源の回収と再利用、データ配置、大域データに基づく条件分岐などについて述べた。

Evaluation of OPtimization Methods for SIGMA-1, an Instruction Level Dataflow Computer

Kei Hiraki　　Toshio Shimada　　Satoshi Sekiguchi
(University of Tokyo)　(Electrotechnical Laboratory)

The mechanism and the performance of the code-optimization methods for an instruction-level dataflow computer SIGMA-1 are evaluated. A compound switch instruction that accelerates loop-unfolding operations, an efficient implementation of loop invariant variables, and an instruction-level load distribution mechanism are evaluated by several sample programs. Then optimization mechanisms necessary for writing numerical programs are discussed.

１．はじめに

現在、科学技術計算用データ駆動計算機ＳＩＧＭＡ－１[1]を用いた性能評価実験を行なっている。評価すべき項目は、命令レベルデータ駆動計算機の持つ基本的能力、応用プログラムにおける性能と並列性の振舞い、およびオーバーヘッドの解析である。しかしながら、これらの項目は被評価プログラムを作成し、コンパイルして評価を行なうため、プログラムの記法およびコンパイラにおける最適化の影響を強くうける。

プログラム記法とコンパイラにおける最適化は大別してプログラムの意味・内容に依存しない客観的最適化と、意味・内容に依存する主観的最適化に分類される。現在コンパイラに組み込まれている最適化機能の多くは客観的最適化機能であると考えられる。本発表ではまず客観的コード最適化機能としてループ構成に関する最適化に注目する。最適化技法としてＳｔｉｃｋｙトークンの導入、複合的分岐命令の使用、SWITCH命令を省略したループ不変変数処理についてのべ、それらの効果を測定する。次に主観的最適化技法について、２種類のプログラム実装に関して考察を行なう。

２．コード最適化

２．１　ループを並列実行する命令の最適化

SIGMA-1は、ループから並列性を引き出すため、ループの繰り返しごとにコンテキストを切り替える。この機能はSWITCH、SI_INC命令によって実現される。SWITCHは、OP1とOP2の２つのデータを受け取り、OP1の値がtrueかfalseかによってOP2の行く先をスイッチする。SI_INC命令は、通常１個のデータを受け取り、そのループカウンタを１だけ増す。これらの命令が全実行命令数に対して占める割合はかなり大きなものがある。幾つかのプログラムについて全実行命令数とこれらの命令の実行回数を調べた結果を表１に示す。

比率は、実行命令数に対してSWITCH命令とSI_INC命令の実行回数の和が占める割合である。この結果からSIGMA-1ではループ実行の際の命令オーバヘッドが平均30%程度はあり、これが性能をかなり低下させていることがわかる。したがってこれらの命令を効率良く実行することが重要である。

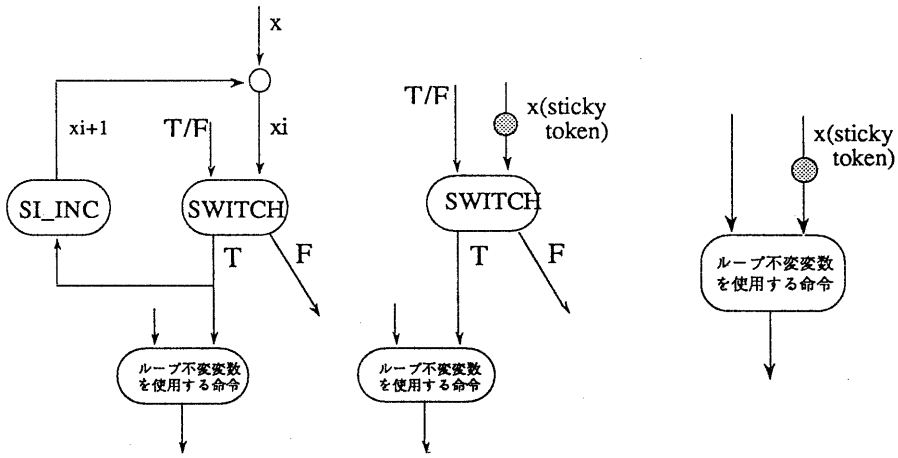
SWITCH命令は条件文の記述のために単独で使用されることがあるが、SI_INC命令は必ずSWITCH命令と組み合わせて使用するので、SWITCH命令とSI_INC命令を複合すれば、実行命令数を平均で15%程度減らすことができ、命令フェッチ時間の減少につながる。さらにSWITCH命令とSI_INC命令の間のトークンの発生が押さえられるため、実行パイプラインのバブルの減少、ネットワーク内のトラフィックの減少につながり性能向上が期待で

プログラム名	全実行命令数	SWITCH命令	SI_INC命令	比率
階乗	253	62	60	48%
内積	37080	8008	4000	32%
台形則積分	33148	9018	3000	36%
行列乗算	30133	6941	2640	32%
コイン問題	16120	2782	358	19%
クイーン	10353	2311	496	27%

表１　SIGMA-1の命令オーバヘッド

プログラム名	性能向上率（１ＰＥ）		性能向上率（１グループ）	
	計算値	実測値	計算値	実測値
階乗	16.3%	17.0%	21.0%	12.0%
内積	7.3	6.1	9.4	6.2
台形則積分	6.0	2.1	7.8	2.1
行列乗算	5.8	2.5	7.5	10.0
コイン問題	1.4	0.0	1.8	0.0
クイーン	3.2	1.8	4.1	1.8

表２　SWTIFC命令使用の効果



(a) stickyトークンを使用しない場合　(b) SWITCHにstickyトークンを貼り付ける　(c) 使用命令に直接貼り付ける

図1　不変変数の保持

きる。この目的のためSIGMA-1にはSWTIFC命令という複合命令が導入された。SWTIFC命令の実行時間は、trueの場合4クロック、falseの場合3クロックである。SWITCHとSI_INCの2命令の実行時間は、true、falseのどちらの場合でも5クロックである。2命令を同じプロセッサで逐次実行する場合パケット転送に1クロック、異なるプロセッサで実行する場合2クロックかかる。ループの繰り返し回数が多い場合（trueの場合）を比較すると4クロック対6または7クロックとなる。予測できる性能向上の割合はSWITCH、SI_INC命令の比率に対して1PEの場合4／6、1グループ（4PE）の場合4／7を乗じた値である。これらの予測される計算値と実際の測定結果を表2に示す。実際には、プログラムに並列性があれば、SWITCH命令とSI_INC命令の実行の間に他の命令の実行が入る可能性があり、この場合パケット転送の遅延は隠蔽されるので、実際の向上率は期待値より小さくなる。また1グループで実行する場合、負荷分散をランダムに行なっているので、SWITCH命令とSI_INC命令が同じプロセッサで実行される場合もあり、これも転送の遅延が小さくなるので実際の向上率が下がる原因となる。

2．2　ループ不変変数処理の最適化

ループの実行には、しばしば値が変わらない変数（ループ不変変数）が使用される。これらの変数は、データフローグラフ上では1個のトークンで表現され、ループの繰り返しの世代を示すため、図1に示したようにSI_INC命令でループカウンタを更新しながらループを循環する。この処理は変数のデータ値が変わらないにもかかわらず2命令を実行しなければならない。さらにループ不変変数のトークンがネットワークの通信量を増し、他の処理の実行を遅延させる可能性がある。

この問題は、ループ不変変数を、繰り返しの各世代で共有することにより解決できる。これはノイマン方式のレジスタの概念をデータフローグラフに導入することである。これを実現するためにSIGMA-1の実行モデルではstickyトークンを考案した。データ駆動方式では命令の入力アークにトークンが到着するとその命令は発火し、入力アーク上のトークンは消滅する。stickyトークンが入力トークンの場合、ノードが発火した後もstickyトークンは消滅せず、その命令の入力アークに残る。

しかし、stickyトークンをループ不変変数を使用する命令に直接貼り付ける場合には問題がある。ループ不変変数は、その前に条件文があると生成されるかどうかが実行時に決定され、実行前には分からない。しかし、ループ終了時には、消去トークンをstickyトークンを貼り付けた命令に送り、stickyトークンを消去する必要がある。しかしstickyトークンの生成が実行時に決まるため、消去トークンを送るべきかどうかの判断が実行前にできない。stickyトークンが存在しない所に消去トークンを送ればエラーとなる。そこで条件文の入り口にSWITCH命令を置き、それにstickyトークンを貼り付けることにすれば、stickyトークンは必ず生成されるので消去トークンも関数終了時に必ず送ることにすればこの問題は解決できる。このようにしても、図1のSI_INC命令とその入出力トークンの生成が抑さえられるので、実行効率をかなり改善できる。しかし、ループ不変変数を使用する命令に直接貼り付ける方法はさらに効率が良いので、これを実現する方法を考えた。

まずデータフローモデルを拡張し「相手のトークンが存在しなくても発火し、次の命令に出力トークンを送る」という発火規則を導入し、消去トークンをこのタイプのトークンとする。このタイプのstickyトークンを、

	stickyトークン数	（1）の方法	（2）の方法
内積	3	0.62	0.19
流体方程式	7	0.70	0.38
台形則積分	7	0.65	0.41

（1）の方法：stickyトークンをSWITCH命令に貼り付ける。
（2）の方法：stickyトークンを不変変数使用命令に直接貼り付ける。

表3　stickyトークンの効果

関数終了時に送れば、stickyトークンが生成されていなくてもエラーは起こらない。問題は、stickyトークンが目的の命令に貼り付く前に、消去トークンがその命令に到着する場合である。この場合stickyトークンが関数終了後も残りエラーの原因となることがある。この問題は、グラフ上では、関数の終了をストリクトに検出した後、消去トークン生成命令を実行すればよい。ハードウェア上は、ネットワークの閉塞などで順序が逆転することを防ぐためstickyトークン生成命令と消去トークン生成命令を同一のプロセッサで実行し、ネットワークを経由しないように命令のスケジューリングを行なうことで解決できる。このようなstickyトークンの効果をSIGMA-1の１ＰＥで測定した結果を表3に示す。表3の値はstickyトークンを使用しない場合を１としたときの実行時間の比である。（1）の方法でもstickyトークンの効果は非常に大きいが、（2）の方法はさらに優れていることが明確に示されている。

3．　負荷分散

SIGMA-1は命令レベルと関数レベルの並列実行を行なうことができる。実際のプログラムでは命令レベル、関数レベルとも並列性は大きく変動するため、効率の良い負荷分散を実現することが高い性能を得るために重要である。SIGMA-1には、静的符負荷分散と動的負荷分散の機能があるが、本稿ではまず命令レベルの静的負荷分散について検討する。

命令レベルの負荷分散の問題は、マルチプロセッサのスケジューリング問題と同じであり、多くの場合ＮＰ完全な問題に属し[2]、ヒューリステイックなアルゴリズムが研究されている。多くの方法は、グラフの各ノードから出口ノードまでの距離を計算し、距離に基づいてプライオリティを決めて割り付ける。

	ランダム(us)	通信時間最小(us)	比
階乗	29.4	25.3	0.86
台形則積分	327	694	2.1

表4　負荷分散を用いた場合の実行時間

これらの方法をSIGMA-1で適用することを考えた場合、次の点が問題となる。まず第１に、ハードウェアに関する不確定要素として、ネットワークの閉塞、構造プロセッサのＤＲＡＭのリフレッシュ、命令の実行時間の変動など動的要素が強く、実際の実行では静的レベルからのズレが大きいことが予想される。第２にデータフロー方式では関数レベルの並列性に同期を取らないため、他の関数が割り当てられるタイミングを静的に予測できない。このため、同じグループに別の関数が重畳して割り当てられると、静的スケジューリングによるクリティカルパス割り付けが、同じプロセッサに重なり、ランダムなスケジューリングより悪くなることも考えられる。

ここでは以上のことを考慮しつつ、ループを含むプログラムを用いて、負荷分散の実験を行なった。用いた方法は、文献[3]の中の通信時間最小を目標としたスケジューリング法である。上述したようにSIGMA-1は動的要素が強いため、レベルに基づくスケジューリングより通信オーバヘッドの減少を優先する方法が有効であろうと考えたからである。この方法を用いてSIGMA-1の１グループを用いて実験した結果を表4に示す。

階乗のプログラムはループの繰り返し間の依存性が強く、ほぼ逐次処理されるのでループボディが並列に展開され重畳的に実行されることがない。また構造プロセッサへのアクセスもない。このため、静的な環境と実際の実行環境がかなり良く一致していると考えられる14%の速度向上結果が得られた。一方、台形則積分は、ルー

プの繰り返し間の依存性が弱いため、ループボディが自動的に展開され、並列実行される。この場合には、クリティカルパスを割り付けたプロセッサが重なるため、ランダムに比べて実行時間は2.1倍と遅くなった。このような場合には、命令レベルの割り付けを動的に変更する機能が必要となる。このような機能として、SIGMA-1はプロセス識別子とループの繰り返し回数に基づいて命令の割り付けを変更できる機能を持っている。これらの機能を用いた実験は別の機会に述べることとする。

4．行列計算プログラムにおける細粒度プログラムの最適化

ここまで、SIGMA-1システムにおけるプログラムの最適化要素について述べた。前節までに述べた項目である（1）関数内の命令分散が適切でない、（2）ループのアンフォールディングに関するオーバーヘッドが大きい、（3）命令設計が適切でなく複数の命令で実現する場合、パイプラインを巡回するオーバーヘッドが大きい、（4）関数呼びだしに伴うオーバーヘッドが逐次計算機と比較して大きいこと、はプログラムの文脈に依存していないため、コンパイラの最適化機能により安全に効率が改善される。

しかし、実際の数値計算アルゴリズムを並列実行する場合には利用可能な並列性を引き出すことが重要である。並列性はアルゴリズムおよびプログラム技法に強く依存するため、安全かつ効率的に並列性を引き出すためには文脈・プログラムの意味内容に依存した最適化を行なうことが必要である。

本節では、例題を通して安全は並列化に関する方式を扱う。例題は線形方程式の解法であるウェーブフロントアルゴリズムによるＳＯＲ法とＣＧ法の一種であるマルチグリッド法を前処理に使う共役勾配法（ｍｇｃｇ法）[4]である。なお、両プログラムともＤＦＣ言語と基本ライブラリを用いて記述されている。

ウェーブフロント・アルゴリズムに基づくＳＯＲプログラムでは、格子状に配置された節点の値を、上方および左方からのすでに更新されている値、当該節点、右方および下方からの更新前の値を使用して求める。従って並列度は行列の対角線方向に存在し、隣接する節点から送られる更新された値により評価が開始される。前節点の更新が終った状況で収束の判定を行ない、以後条件を満たすまで繰り返す。この例題の特徴は次の通りである。

（1）一回の繰り返しに関して節点当たりの計算量がすくなく、相対的に条件判定の比重が高い。
（2）演算が行列の対角線方向にウェーブフロントとして広がるため、演算並列度が行列の要素数の平方根程度に抑えられる。
（3）並列計算機による実行では、行列の要素レベルにおける細粒度同期が必要であり、通信／同期オーバーヘッドが問題となる。

また、ｍｇｃｇ法プログラムでは、３重ないしは５重対角行列に対してレッドブラックＳＯＲ法をグリッド間隔を変化させながら前処理をおこない、その後共役勾配法で繰り返しを行なう。行列は対角線方向のベクトルとして持っている。この例題の特徴は：

（1）一回の繰り返しに関する演算量が多い。（繰り返し回数が他の方法と比較して少ない。）
（2）対角方向に要素を持つデータ構造を用いるため、行列とベクトルの積演算に対角方向走査が必要である。
（3）グリッド間隔が広がった状態における並列性が低下する。

この両者をＤＦＣ言語を用いてＳＩＧＭＡ－１上で実行する場合に問題となることを以下に示す：

［非同期性と多重代入］ 総和計算や行列積の計算に見られる可換な算術操作の繰り返しを、ループとして直線的に解くことが構造体に対する単一代入性、データの一貫性を維持する為に行なわれる。その結果、本来持っている並列性が失われ、かつ多くの構造体割り付けとコピーが必要となる。例えばｍｇｃｇプログラムでは、行列とベクトルの積、レッドブラックＳＯＲ処理において、対角方向に行列を走査するために発生する積和計算が問題となる。

現在のインプリメンテーションでは、構造体メモリ状にＦＩＦＯキューを作成するＢ構造を用いて非同期性を実現してプログラムの並列化を行なっている。

［並列性制御］ 処理装置数に依存しないオブジェクトコードにコンパイルするばあい、過大な並列性による効率の低下／資源の枯渇を防止する必要がある。今回の例題の場合には主要な並列構造がループによる繰り返しで記述されているため、ループ内並列性に固定的な制限を付けることにより比較的容易に制御することが可能である。しかし、ノンストリクトな入力引数評価による爆発の防止は縦方向並列性抽出を妨げるため、現状では制御が非常に困難である。（例題では場所毎に場当たり的手法を用いている。）

［メモリ資源の回収・再利用］ 資源を回収・再利用するため、処理の開始／終了を明確に定義する必要がある。関数型言語であるＤＦＣではその本質上構造体は単一代入的に消費され、資源の回収・再利用にはガーベジコレクションが必要である。しかし、構造体データ全てをガーベジコレクションの対象にすることは、特に効率的に好ましくなく、局所的資源の回収・再利用が必要となる。

ウェーブフロントの例題では、各繰り返しにおいて最後に値が設定される場所が指定される。これを利用して多重パイプライン的に実行されるウェーブフロントアルゴリズムで用いられる行列の再利用を行なっている。一方ｍｇｃｇ法では領域の回収に際してバリア動作が必要であるが、ＤＦＣの基本構造とあい入れないため、現

在は全体の動作に強制的順序付けを行なうことにより実現している。

［関数資源の回収・再利用］ 関数インスタンスは、メモリフレーム等の物理的資源を消費するとともに、トークンタグの一部である並列識別子資源を消費する。並列識別子資源を回収・再利用するためには自分自身の実行が終り自らがクリーンになるだけでなく、そこに源を発する関数トリー全体およびトリーに属する関数が発した構造体操作の実行が終了しかつクリーンになっている必要がある。

関数トリーがクリーンであることを保証する容易な方法の一つは、関数をトークンレベルで関数的にする、すなわち、入力と出力をストリクトに評価することである。しかしながら、並列実行環境ではストリクトにすることにより関数実行の開始、関数からの返値が遅れることにより性能が低下する（表5参照）。今回の例題では、関数を要求されるストリクトネスに基づいて分類し、適宜コンパイルすることで解決を図った。

［データの配置］ データの配置は古くから大きな問題であり、データパラレル計算モデルを支える大きな論点であった。ＳＩＧＭＡ－１では関数の実行される処理装置は、当該時点で負荷量が最低であるクラスタに割り付ける動的負荷分散方式を用いている。その結果、データ配置と関数配置が独立に決定され、システム能力を低下させる主要な原因となり、静的な配置が必要と考えられた[5]。

［大域データに基づく条件分岐］ ＤＦＣを用いたＳＩＧＭＡ－１における大域的条件分岐には２種類の非効率が伴っている。第一の問題は、条件の収集と分岐のブロードキャストに時間がかかり実行平均並列性を低下させることである。特にＤＦＣでは大域条件の収集はすべて子関数との通信を伴うため大きいオーバーヘッドとなる。第二の問題点は、関数出力のストリクトネスである。ＤＦＣにおける関数構造は入力／出力が分離し、出力を用いた結果を入力に戻すことが不可能である。従って、条件分岐は異なる関数を生起することにより実現される。この結果、関数内部で生成した内部値を再利用することが困難となっている。例題においては、これらの問題点を回避する目的で構造体を使用してことなるレベルの双方向通信を実現している。

このように、安全な範囲で並列性を損ねないプログラムを実現するためには、（１）大局的決定性を持った非同期アクセスの実現、（２）入力と出力におけるストリクトネスの詳細な指定、（３）レベルを越えた関数間での双方向通信、（４）大域的同期の実現、（５）許される並列性に関する詳細な指定、（６）データ配置を考慮した動的負荷分散の効率的実現が必要である。

フォンノイマン計算機による逐次処理はこれらの条件を文脈依存なしで解決する極端な例であり、命令レベルのデータ駆動計算機における並列処理は上記問題点を全て抱えた逆の極端な例であると考えられる。従って、データ駆動計算機ではプログラミング技法とコンパイラ機能の協力によりこれらの問題を解決する必要がある。

プログラム名	非ストリクト実行時間	ストリクト実行時間	オーバヘッド
台形則積分	96.1 (msec)	98. (msec)	12%
行列乗算	6.1	6.43	5%
ウエーブフロント	111	122	10%

表5　出力ストリクト化によるオーバーヘッド

４．おわりに

データ駆動計算機ＳＩＧＭＡ－１を対象としたプログラム記述およびコンパイル時における最適化技法について述べ、最適化による性能向上の測定を行なった。本発表で述べた方法を総合して用いるとかなり大きな性能向上が見られ、これらの技法が有効であることが示された。しかしながら、よりプログラムの意味・内容に関連した並列性にかんする最適化はプログラム作者が手で行なっていることが現状である。これらの機能を自動的に最適化する方式の開発が強く望まれる。

本研究遂行にあたり、討論に参加して頂いた計算機方式研究室の皆様、研究を支えて頂いた弓場情報アーキテクチャ部長に深く感謝いたします。

参考文献

[1] Hiraki, Nishida, Sekiguchi, Shimada, "SIGMA Dataflow Supercomputer: A Challenge for New Generation Supercomputing Systems", J. of Information Processing, 10, 4, 1988.

[2] 笠原, "並列処理技術", コロナ社, 1991年.

[3] 大塚, 坂井, 弓場, "データ駆動計算機における命令水準の負荷分散", 電子通信学会, 電子計算機研究会 CAS86-136, 1986年.

[4] 建部修見、小柳義夫：マルチグリッド前処理付共役勾配法、ＳＷｏＰＰ’９２、数値解析研究会資料

[5] 島田俊夫、平木敬、関口智嗣：データフロー計算機ＳＩＧＭＡ－１の性能評価、ＪＳＰＰ９２論文集