計算機アーキテクチャ 83−17
（1990. 7. 18）

OSCAR上での細粒度タスクの並列処理

笠原 博徳　本多 弘樹　Wichian Premchaiswadi
小椋 章央　茂木 章善　成田 誠之助
早稲田大学理工学部電気工学科

概要

本論文ではマルチプロセッサシステムOSCAR（Optimally Scheduled Advanced Multiprocessor）上での、細粒度タスクの並列処理手法について述べる。ここでOSCAR上での細粒度タスクとは各々が単一あるいは複数浮動小数点命令命令からなるタスクを意味する。本手法ではデータ転送を考慮したスタティックスケジューリングを用いることにより、同期及びデータ転送の最小化及び、各プロセッサのレジスタの最適使用が可能となる。本手法を用いたコンパイラはすでにOSCAR上にインプリメントされており、本論文では、OSCAR上での性能評価についても述べる。

PARALLEL PROCESSING OF NEAR FINE GRAIN TASKS ON OSCAR(Optimally Scheduled Advanced Multiprocessor)

Hironori KASAHARA, Hiroki HONDA, Wichian PREMCHAISWADI
Akio OGURA, Akiyoshi MOGI and Seinosuke NARITA
Dept. of Electrical Engineering, Waseda University
3-4-1 Ohkubo Shinjuku-ku, Tokyo, 169, Japan

ABSTRACT

This paper proposes a compilation scheme for parallel processing of near fine grain tasks, each of which consists of several operations or a statment, on a multiprocessor system called OSCAR(Optimally Scheduled Advanced Multiprocessor). The scheme generates optimized parallel machine codes which minimize synchronization overhead and data transfer overhead and optimally use registers of each processor by using static multiprocessor scheduling algorithms considering data transfer among processors. This scheme can effectively be combined with compilation scheme for macro-dataflow computation which uses parallelism among coarse grain tasks like loops, basic blocks and subroutines and for the traditional loop concurrentization which use palallelism among medium grain tasks like iterations. A compiler using the proposed scheme has been implemented on OSCAR which has been designed to take full advantage of the static scheduling. In this paper the performance evaluation of the scheme on OSCAR is also described.

１．はじめに

ＦＯＲＴＲＡＮのようなシーケンシャルな言語でかかれたプログラムをマルチプロセッサシステム[5]上で並列処理する際、ループの並列化が広く研究されてきた[3][4]。最近では強力なデータ依存解析[1][2]やプログラムのリストラクチャリング[4][6][9]により、種々の形状のループが自動的に並列化できる[4][6][7][10]ようになっている。しかしながら、イタレーション間にまたがるデータ依存やループの外への条件分岐によりＤＯＡＬＬやＤＯＡＣＲＯＳＳの手法が効果的に適用できないループも存在する。また、ループ以外の部分は従来マルチプロセッサシステム上ではシーケンシャルに実行されていた。したがって、今後マルチプロセッサシステムの実行効率を向上させるためには、シーケンシャルループやベーシックブロックをより細かい粒度のタスクを用いて並列に実行することが重要である。

命令レベルの極細粒度のタスクを用いた並列処理は、マルチプロセッサ以外のいろいろな並列マシン、例えば複数の演算ユニットを持つマシン[12]、データフローマシン[17]、ＶＬＩＷ[5][13][14]、スーパースカラーマシンなどでインプリメントされてきた。しかし複数演算ユニットマシン、データフローマシン、スーパースカラーマシンのようにダイナミックスケジューリングを用いたマシンは、たとえスケジューリングが高速なハードウェアで行われても命令実行時間と比較した相対的な実行時オーバーヘッドはかなり大きくなる。この実行時スケジューリングオーバーヘッドを削減するために、ＶＬＩＷマシンではコンパイル時のスタティックスケジューリングを採用している。一般的に、スタティックスケジューリングは内部にデータ依存しか存在しないベーシックブロックの並列処理に有効である。しかしながら、スタティックスケジューリングでは実行時でないと確定しない条件分岐や命令実行時間の変動を扱う際には困難が生じる。この条件分岐に対処するために、トレーススケジューリング[13][14]のように実行時プロファイルから得られた分岐の確率の情報を用いる手法や、パーコレーションスケジューリングにみられるようなプログラムコードの入口への命令のパーコレーションなどが提案されている[18]。

しかし、オーバーヘッドの比較的大きいダイナミックスケジューリングも細粒度でなく粗粒度タスクに適用すれば、スケジューリングオーバーヘッドが相対的に小さくなるため、分岐確率が未知であるような条件分岐を扱うために有効である。

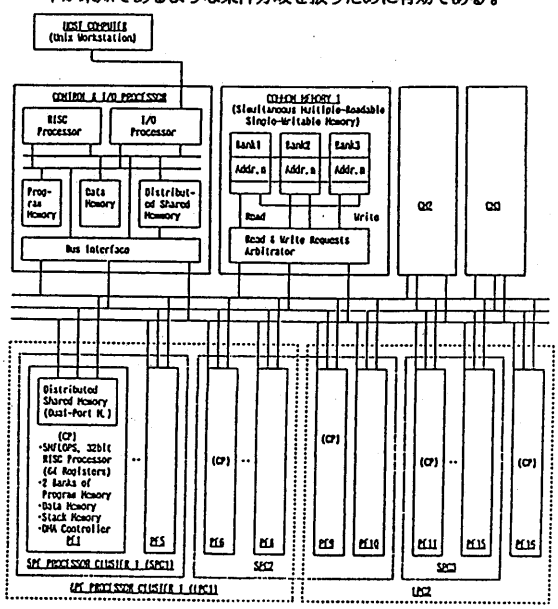


図１　ＯＳＣＡＲのアーキテクチャ

以上のことを考慮して、筆者らはダイナミックスケジューリングを用いたマクロデータフロー処理[11]、ダイナミック及びスタティックスケジューリングを用いたループ並列化[10]、スタティックスケジューリングを用いた細粒度タスクの並列処理を階層的に統合したマルチプロセッサ用のコンパイラを開発してきた。このコンパイラにおいては、ベーシックブロックやシーケンシャルループの並列処理は複数命令あるいはステートメントレベルの細粒度タスクを用いて行われる。

２．ＯＳＣＡＲのアーキテクチャ

本章ではＯＳＣＡＲ(Optimally Scheduled Advanced Multiprocessor)のアーキテクチャ[8]について述べる。ＯＳＣＡＲはスタティックスケジューリングを用いることにより細粒度タスクを低オーバーヘッドで並列処理できるように設計されている。図１にすでに製作されたＯＳＣＡＲのアーキテクチャを示す。図１のように、ＯＳＣＡＲは最大１６個のプロセッサエレメント（ＰＥ）と、ダイナミックスケジューラ及びワークステーションのようなホストコンピュータとのインターフェースとして利用されるコントロール＆Ｉ／Ｏプロセッサ（ＣＩＯＰ）とが、３つのコモンメモリと３本のバスによって結合されているメモリ共有型のマルチプロセッサシステムである。

ＯＳＣＡＲのアーキテクチャは、スタティックスケジューリングを用いた細粒度タスクの並列処理を効率よくインプリメントするために以下のようなサポートを行う。

（１）ＲＩＳＣプロセッサとバス

ＯＳＣＡＲのＲＩＳＣプロセッサは浮動小数点の加算、減算、乗算を含む全ての命令を１クロックで、他の浮動小数点演算を固定されたクロック数で実行する。全命令を１クロックで実行することにより、従来、スタティックスケジューリングを実マルチプロセッサシステムへ適用する際の問題点となっていたタスク処理時間の正確な推定の困難さが解決され、各タスクの正確な処理時間の推定が可能となる。

さらに、ＯＳＣＡＲにおいては、全ＰＥとバスが１つの参照クロックの下で動作し、全てのＰＥの実行を同時に開始することができる。この特徴により、高精度のスタティックスケジューリングを用いることにより、コンパイラが、各ＰＥのバスアクセスタイミングをクロックレベルで制御することが可能となる。具体的には、ＯＳＣＡＲコンパイラは緊急でないバスアクセス要求に対してはＮＯＰをＰＥ上のプログラムコードに挿入しバスアクセスを遅らせる等の処理を行うことにより、ＰＥ間でのバスアクセスを制御することができる。

現在までにインプリメントされているコンパイラでは、デバッグ等の目的のために同期コードをマシンコード中に挿入しているが、上述の最適化技術により最終的にはベーシックブロック中のほとんど全ての同期コードを削除することが可能となる。

（２）デュアルポートメモリ（ＤＰＭ）と３種のデータ転送モード

ＯＳＣＡＲでは、ＰＥ上のＤＰＭすなわち分散型共有メモリを用いたＰＥ間の１対１の直接データ転送モード、あるＰＥから他のＰＥへのデータのブロードキャスト転送モードという２種類のデータ転送モードが用意されている。さらに、あるＰＥから複数のＰＥへのＣＭ経由による間接データ転送モードも用意されている。間接データ転送では２回のデータ転送が必要となるのに対しＤＰＭを用いた直接データ転送では、あるＰＥから他ＰＥへの３２ビットデータの転送に１回のデータ転送のみを要する。また、データのブロードキャストはＣＭ経由のデータ転送と比較して転送時間を大幅削減することを可能とする。すなわちスタティックスケジューリングを用いて、これらの３種類のデータ転送モードを最適に選択することにより、データ転送によるオーバーヘッドを大幅に削減することが可能となる。特に、このＰＥ間の１対１直接データ転送モードは、スタティックスケジューリングが採用されたときにのみ有効に使用することができるデータ転送方式である。さらに、ＤＰＭを使用したデータ転送及び同期では、各ＰＥがＤＰＭ上に書き込まれた同期フラグをプロセッサ内部で独立してチェックするので、ＣＭを使用した場合のビジーウェイトのために生じるバスのバン

ド幅低下を防ぐことができ、同期のオーバーヘッドが軽減することを可能とする。

３．データ転送を考慮したスタティックスケジューリングを用いた細粒度タスクのコンパイル手法

本章ではデータ転送を考慮したスタティックスケジューリングを用いた、ベーシックブロック内での細粒度タスクの並列処理のためのコンパイル手法について述べる。本手法は、細粒度タスクの生成及びタスクグラフの作成、ＰＥへのタスクのスケジューリング、生成されたスケジュールに基づく並列化マシンコードの生成という３つの部分からなっている。

３．１　タスク及びタスクグラフの生成

ベーシックブロックを効率良く並列処理するためには、並列性が最大になるように、またデータ転送及び同期によるオーバーヘッドが最小になるようにベーシックブロック内の計算をタスク（ＰＥに割り当てられる単位）に分割してやらねばならない。一般的には、１命令、複数命令、１ステートメント、複数ステートメントなどがベーシックブロックの並列処理におけるタスクとなり得る。しかしながら、効率良く処理するためには、ベーシックブロックの大きさ、ベーシックブロック内部の並列性、処理能力、プロセッサ間のデータ転送能力、同期のオーバーヘッド、タスクスケジューラの能力等のような様々な要因を考慮にいれて最善のタスクグラニュラリティを選ばなければならない。つまり、命令レベルの極細粒度タスクは最大の並列性をもたらすかも知れないが、使用するマシンによっては同時に大きなオーバーヘッドを生じる可能性がある。

本論文においては、ＯＳＣＡＲの処理能力、データ転送能力を考慮した場合の最適なグラニュラリティとして、ステートメントレベルあるいは数命令レベルのグラニュラリティを選択するが、以下に示すコンパイル手法自体は他のグラニュラリティに対しても同様に適用できる。

図２にステートメントレベルのタスク、すなわち細粒度タスクの例として、ランダムなスパース係数行列を持つ線形システムをクラウト法により解く[15][19]ベーシックブロックの例を示す。図２のような計算パターンを持つ大きなベーシックブロックは、電子回路シミュレーション中のスパース線形システムを解くために提案されたシンボリックジェネレーションテクニックによりしばしば生成される。生成されたタスク間にはフロー依存、出力依存、逆依存[2][3][4]などのデータ依存が存在する。しかしながら、ほとんどの出力、逆依存は変数のリネーミングにより容易に取り除くことができる[3][4]。

データ依存あるいは先行制約は図３のような”タスクグラフ”[8][20]と呼ばれる無サイクル有向グラフ中のアークとして表され、図中で各々のタスクは１つのノードに対応している。図３において、ノード中の数字はタスク番号ｉを表し、その横にかかれた数字はＰＥ上でのタスク処理時間t_iを表す。ノードN_iからN_jに向かって引かれたエッジはタスクT_iがT_jに対して先行するという部分的な順序制約を表す。またタスク間のデータ転送時間を考慮する場合には、エッジは通常可変のウェイトを持つものとする。このウェイトは、タスクT_iとT_jが別々のＰＥに割り当てられた場合には２つのＰＥ間でのデータ転送時間t_{ij}となる。タスクが同一のＰＥに割り当てられた場合には、０またはレジスタあるいはローカルメモリへのアクセス時間となる。２重線の線分はクリティカルパスを表している。クリティカルパス長t_{cr}は、オーバーヘッド無しで任意台数のＰＥを用いてタスク集合を処理した際に達成できる最小実行時間を表す。しかし、ここでタスク処理時間は、必ずしも決定的な値（一意に決定できる値）であるとは限らない。それは、浮動小数点演算の実行時間がそのオペランドの値によって変動する可能性があるからである。この問題に対しては各々の命令の平均値を用いて解決することが提案されている[15][16]。しかし、ＯＳＣＡＲでは、第２章で述べたように全ての命令を１クロックで実行するＲＩＳＣライクプロセッサを使用することによりタスク処理時間をコンパイル時に正確に求めることを可能とし、この問題を解決している。

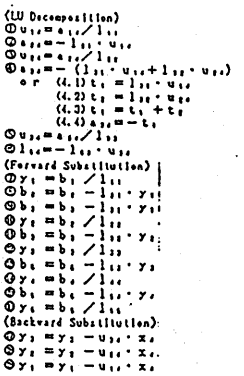


図２　細粒度タスクの例

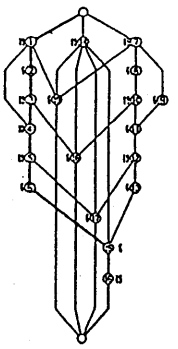
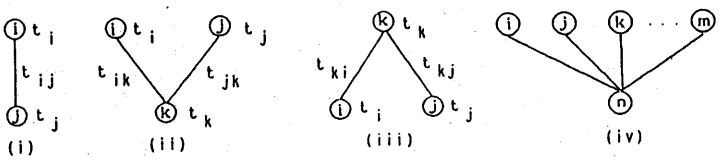
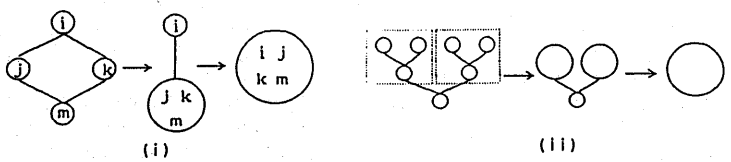


図３　タスクグラフ



（ａ）　タスク融合の基本パターン



（ｂ）　タスク融合の適用例

図４　タスク融合法

また本手法では生成されたタスクが細かすぎた場合に、ヒューリスティックなタスク融合手法を用いて並列性をあまり犠牲にする事なく自動的に複数のタスクを融合し１つの粗いタスクを生成する。図４（ａ）にこのヒューリスティックタスク融合法が自動的に適用される基本パターンを示す。この手法では、タスクグラフ中で図４（ａ）の様な部分パターンをもつタスクのうち、その最小並列処理時間がシーケンシャル処理時間より大きい場合にそれらのタスクを１つのタスクに融合するのである。例えば図４（ａ）の(i)のパターンにおいて、もしT_iがT_jとは異なるＰＥに割り当てられると、シーケンシャルの場合には処理時間が（t_i+t_j）であるのに対し、処理時間は（$t_i+t_j+t_{ij}$）となってしまう。したがって、T_iとT_jを１つのタスクに融合する。パターン(ii)の場合、最小並列処理時間ｍｉｎ｛（$t_i+t_k+t_{ik}$），（$t_j+t_k+t_{jk}$）｝がシーケンシャル処理時間（$t_i+t_j+t_k$）より大きい場合には３つのタスクを１つに融合する。パターン(iii)の場合も基本的にパターン(ii)と同じである。パターン(iv)の場合も、最小並列処理時間ｍｉｎ｛（$t_i+t_n+t_{in}$），（$t_j+t_n+t_{jn}$），．．．，（$t_m+t_n+t_{mn}$）｝がシーケンシャル処理時間（$t_i+t_j+\ldots+t_m+t_n$）よりも大きい場合にはこれらのタスクを１つに融合する。この単純な方法を部分タスクグラフに繰り返し適用してやることにより、図４（ｂ）のパターン(i)の様な構造を持つ複数タスクを１つに融合することが可能となる。同様に、図４（ｂ）(ii)のような木構造を持つタスクは、タスク間のデータ転送時間に対するタスク処理時間の割合に応じて複数のあるいは１つの粗いタスクに融合される。この単純なタスク融合手法は、不必要なデータ転送を防ぐと共に各タスク内でＰＥ上のレジスタを有効利用することを可能とする。

３．２マルチプロセッサ上でのスタティックスケジューリングアルゴリズム

与えられたマルチプロセッサ上で最小時間でタスク集合の実行をするためには、プロセッサへのタスクの最適な割当て、及び、同一プロセッサに割当てられたタスクの最適な実行順序の決定を行うことが重要である。最適な割当て、及び実行順序の決定は実行時間（スケジュール長）最小化マルチプロセッサスケジューリング問題として扱うことができる[20]。すなわち、このスケジューリング問題は、ｎ個のタスクからなる先行制約があるタスク集合を、ｍ台の能力の等しいプロセッサ上で並列処理する際に、スケジュール長を最小とするようなノンプリエンティブなスケジュールを求める問題である。

しかし、このスケジューリング問題は、強ＮＰ困難として知られている。すなわち、Ｐ＝ＮＰ，でなければ多項式時間最適化アルゴリズムはもちろん、疑似多項式時間近似スキームも求めることもできない。このような見地から、様々なヒューリスティックアルゴリズムや実用的な最適化アルゴリズムが研究されてきた。計算時間と、生成されるスケジュールの質の両面から考え、ＯＳＣＡＲコンパイラでは、ＣＰ／ＭＩＳＦ法[20]をデータ転送時間を考慮するように改良したヒューリスティックアルゴリズムであるＣＰ／ＤＴ／ＭＩＳＦ法（Critical Path / Data Transfar /Most Immediate Successors First）が採用されている。なお、ＣＰ／ＭＩＳＦ法は、データ転送時間を考慮しないスケジューリング問題において従来までに開発された最も優れたヒューリスティックアルゴリズムの一つである。

次に、ＣＰ／ＤＴ／ＭＩＳＦ法を簡単に説明する。ＣＰ／ＤＴ／ＭＩＳＦ法はタスク割当てに優先順位を用いているリストスケジューリングアルゴリズムであり本質的には一種のデータ駆動型スケジューリングアルゴリズムである。
その手順は、以下の通りである。

手順１　各タスクのレベルｌｉを計算する。ｌｉはタスクグラフ上でのタスクＮｉから出口ノードへの最長パス長である。

手順２　プロセッサ間のデータ転送時間を考慮したリストスケジューリングを行う。

手順２．１
あるスケジューリング時点において実行待ちタスクの内で最もレベルの高いタスクをアイドルプロセッサの一つへ割り当てる際に生じるデータ転送時間を、考えられる全ての割当てに対して推定する。

手順２．２
データ転送時間が最小になるようなタスクのプロセッサへの割当てを一つ選び出しその時点までの部分スケジュールにこの割当を加える。このとき、データ転送時間が最小となるタスク割当てが２組以上存在するときには、直接の後続タスクが最も多いタスクを優先する。
アイドルプロセッサが無くなるまで手順２．２を繰り返す。

手順２．３
データ転送時間を考慮し、次のスケジューリング時点を算出し実行待ちタスクと空きプロセッサを見つける。実行待ちタスクが一つも無くなれば終了、それ以外の場合は手順２．１へ戻る。

このアルゴリズムの性能は、１００～４０００のタスクを含む約１５００例のタスクグラフに適用し評価されている。これらのタスクグラフは、実在のプログラムのタスクグラフや、ランダムに作り出されたタスクグラフから得られたものである。これらのタスクグラフにおいては、平均タスク間データ転送時間（エッジ上の重みの平均）は、平均タスク処理時間の１０％～２０％と小さい時間にもかかわらず、データ転送時間を考慮したＣＰ／ＤＴ／ＭＩＳＦ法で求められるスケジュールによる平均実行時間は、ＦＩＦＯタイプのスケジューリング、すなわち、タスク割当ての優先度を用いないスケジューリング法により求められるスケジュールによる平均実行時間（データ転送時間を含めた実行時間）に比べて約１０％程短くなっている。また、スケジューリングに要する時間も短く、約１０００程度のタスク数の問題をワークステーション上で数十秒で解くことができる。

３．３マシンコード生成

実際のマルチプロセッサシステム上で実効性能をあげるためには、スケジューリング結果を用いて最適な並列マシンコードを生成することが必要である。スケジューリング結果により以下のような様々な情報が得られる。すなわち、各ＰＥでどのタスクが実行されるのか、各ＰＥに割当てられたタスクの実行順序、他のＰＥに割当てられたタスクから転送されるデータを待つ時間の推定値、同期の有無、等である。従って、ＰＥに割当てられたタスクのコードを順番に並べ、データ転送時間及び、同期コードをつけ加えることにより、各ＰＥの並列化マシンコードを生成することができる。タスク間の同期には、バージョンナンバー法が用いられている。

スタティックスケジューリングによって得られる情報を最大限に利用することにより、コンパイラは低オーバーヘッドのマシンコードを生成することができる。例えば、タスク割当て及び、実行順序に関する情報を用いることにより、同一のプロセッサに割り当てられたタスクがデータの受け渡しを行う際に、そのＰＥ内のレジスタを最適に利用することができる。このレジスタの最適利用により、実行時間は著しく短縮される。更に、同期を必要とするタスクや、タスク割当て、実行順序に関する情報を熟慮する事により、コンパイラは同期のオーバーヘッドを最小にすることができる。

更に、ＯＳＣＡＲ上では、２章で述べた通り、ＰＥによるパ

スアクセスのタイミング、データ転送モードを、最適化する事が可能である。図5は、ＰＥのバスアクセスのタイミング、データ転送モード、及び６４個のレジスタ利用が最適化されて生成されたマシンコードの実行イメージである。この図は、ＯＳＣＡＲ上で実際に並列処理を行う際の厳密なシュミレーションとみなすこともできる。図中ＰＥ３の列において、”ＬＤ３０”、”２５Ｒ”、”ＷＡＩＴ”、及び、”ＰＥ５”などの文字列の意味は、”ＤＭからタスク３０の出力データをレジスタにロード”、”タスク２５を実行しその出力データをレジスタに保存”、”タスク２５の出力データをＰＥ５に直接転送する間しばらく待つ”、ということである。このとき、ＰＥ３はＰＥ１がブロードキャストでバスを使っている間バスが空くのを待っている。

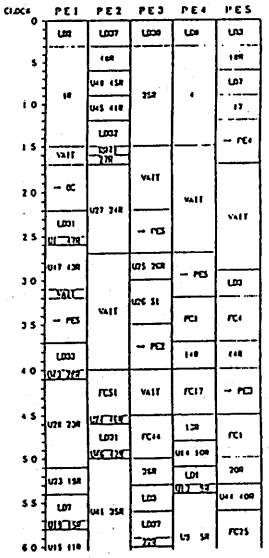
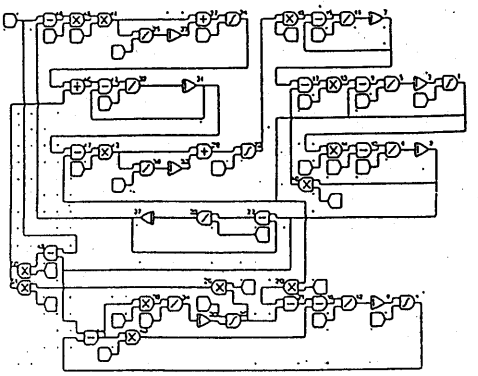


図5　ＯＳＣＡＲ上でのマシンコードの実行イメージ

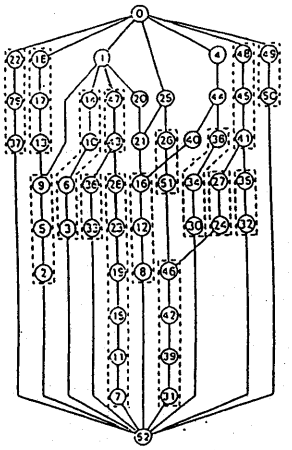
４．ＯＳＣＡＲの性能評価

この章では、ＯＳＣＡＲにおける細粒度レベルでの並列処理の例を示す。図６（ａ）は常微分方程式を解くための一種のデータフローグラフである[32]。グラフ中では、各ノードは単一あるいは、複数の浮動小数点演算を表す。図中のサイクルは、ＦＯＲＴＲＡＮなどで記述した場合に（ｉ－１）番目のイタレーションからｉ番目のイタレーションにデータ依存が存在することを表す。言い替えれば、この計算は，ＤＯＡＣＲＯＳＳ処理が効果的でないＤＯループである。図６（ｂ）は５１タスクからなるループボディを表すタスクグラフである。各タスクは、図６（ａ）におけるノードに対応する。各ノードは上記のように単一あるいは複数の浮動小数点演算からなり、これを細粒度（Near Fine Granularity）タスクと呼ぶ。図６（ｃ）はＯＳＣＡＲ上での実際の実行時間（実線）とシュミレーションによる推定実行時間（点線）を表している。”タスク融合前（Before Task Fusion）”の見出しのついた曲線群は細粒度タスクの並列処理時間を表している。他の曲線は、３．１項で述べたタスクフュージョンによって生成された粗粒度タスクでの実行時間を表している。実際の実行時間は、１ＰＥでそれぞれ１０８．７［μs］、１０５．８［μs］に対し、７ＰＥで３７．２［μs］（１／２．９２）、３６．８［μs］（１／３．０１）までに減少している。

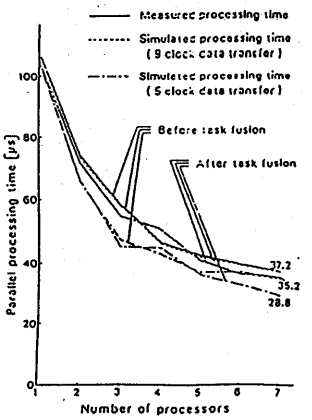
また図中シミュレーションによる実行時間、すなわちマシンコードから推定される実行時間、が二本の点線で表されている。これを見ると、実際の実行時間と推定実行時間との間にはほとんど差がみられない。このことから、ＯＳＣＡＲ上では、スケジューリングの結果を用いた厳密なマシンコードの最適化が有効であることがわかる。ＯＳＣＡＲはノーマル運転モードでは５クロックで２ワードのデータを転送することができるのであるが、現在ハードウェアテストの段階であり上記の実際の実行時間は２ワードのデータを９クロックかけて転送している状態で測定されている。従って、ノーマル運転モードで実行が行われれば、データ転送によるオーバーヘッドは半減すると考えられる。一点鎖線の曲線はタスク融合前及び、融合後の細粒度タスクでのノーマル運転モードにおける正確なシュミレーションによる推定実行時間が表されている。このシミュレーション、タスク融合後の推定実行時間は、１ＰＥで１０４．８［μs］から７ＰＥで２８．８［μs］（１／３．６４）に減少している。



（ａ）　細粒度タスクからなる一種のデータフローグラフ



（ｂ）　図６（ａ）のタスクグラフ



（ｃ）　ＯＳＣＡＲ上での並列処理時間

図６　ＯＳＣＡＲ上での細粒度タスクを用いた常微分方程式求解の並列処理

図7（a）は大規模なベーシックブロックの例としてスパースな連立1次方程式求解のためによく使用されるシンボリックジェネレーション法によって生成されたFORTRANプログラムの例があげられている。PE台数に対するプログラムの実行時間が図7（b）に示されている。タスクのグラニュラリティはステートメントレベルである。実行時間は、図のように、1PEで765［μs］、3PEで314［μs］、7PEで179（1／4．27）［μs］に減少している。

5．むすび

本論文では、CP／DT／MISF法と呼ばれるスケジューリングアルゴリズムを用いたOSCAR上での細粒度タスクの並列処理のためのコンパイレーション手法を提案した。本手法は最適化された並列化マシンコードを生成することによってプログラムの実行時間を最小化する事ができる。この最適化された並列化マシンコードでは、データ転送及び、同期オーバーヘッドを最小化できると共に、同一のPEに割り当てられたタスク間でのデータ授受のためにレジスタの最適利用を図ることができる。OSCARのアーキテクチャーサポートによりコンパイラは、タスクの処理時間を正確に推定する事が可能であり、その結果、優れたスタティックスケジュールを生成する事ができる。本手法によるマシンコードスケジューリングをさらに正確なものとすることにより、ベーシックブロック内の同期コードを全て削除する事が今後の研究課題である。

また、本手法によるFORTRANコンパイラ、並びに複数の特定アプリケーション用コンパイラがOSCAR上ですでに動作している。

6．参考文献

[1] A.V.Aho, R.Sethi and J.D.Ullman, Compilers : Prinsples, Techniques , and Tools,Addison Wesley,1988.

[2] U.Banerjee,Dependence Analysis for Supercomputing, Kluwer Academic, Publisher,1988.

[3] D.A.Padua, D.J.Kuck and D.H.Lawrie, "High-speed multiprocessor and compilation techniques," IEEE Trans., Vol.C-29, No.9, pp.763-776, Sep.1980

[4] D.A.Padua and M.J.Wolfe,"Advanced Compiler Optimizations for Supercomputers," C.ACM, Vol.29, No.12, pp.1184-1201, Dec.1986.


```
A03P18=A03P18/A03P03
A03P57=A03P57/A03P03
A06P48=A06P48/A06P06
A07P13=A07P13/A07P07
A07P38=A07P38/A07P07
A11P51=A11P51/A11P11
A17P53=A17P53/A17P17
A23P48=A23P48/A23P23
A27P45=A27P45/A27P27
A31P53=A31P53/A31P31
A32P38=A32P38/A32P32
A32P53=A32P53/A32P32
A33P48=A33P48-A33P06*A06P48
A33P48=A33P48/A33P33
A34P55=A34P55/A34P34
A37P50=A37P50/A37P37
A53P38=A53P38-A53P32*A32P38
A38P52=A38P52-A38P05*A05P52
A38P52=A38P52/A38P38
A44P46=A44P46/A44P44
A52P46=A52P46-A52P44*A44P46
A50P51=A50P51/A50P50
A53P52=A53P52-A53P38*A38P52
A53P53=A53P53-A53P32*A32P53
A54P55=A54P55/A54P54
A54P56=A54P56/A54P54
Z001=B001/A01P01
Z001=B001/A01P01
Z002=B002/A02P02
Z003=B003/A03P03
Z004=B004/A04P04
Z005=B005/A05P05
Z006=B006/A06P06
Z007=B007/A07P07
Z008=B008/A08P08
Z009=B009/A09P09
Z010=B010/A10P10
Z011=B011/A11P11
Z012=B012/A12P12
Z013=B013/A13P13
Z014=B014/A14P14
Z015=B015/A15P15
Z016=(B016-A16P01*Z001)/A16P16
Z017=B017/A17P17
Z018=B018/A18P18
Z019=B019/A19P19
Z020=B020/A20P20
Z021=B021/A21P21
Z022=(B022-A22P20*Z020)/A22P22
Z023=B023/A23P23
Z024=B024/A24P24
Z025=B025/A25P25
Z026=B026/A26P26
Z027=B027/A27P27
Z028=(B028-A28P19*Z019)/A28P28
Z029=(B029-A29P15*Z015)/A29P29
Z030=(B030-A30P04*Z004-A30P10
&*Z010)/A30P30
Z031=B031/A31P31
Z032=B032/A32P32
Z033=(B033-A33P02*Z002-A33P06
&*Z006)/A33P33
Z034=(B034-A34P07*Z007)/A34P34
Z035=B035/A35P35
Z036=(B036-A36P30*Z030)/A36P36
Z037=B037/A37P37
Z038=(B038-A38P05*Z005)/A38P38
Z039=(B039-A39P09*Z009)/A39P39
Z040=B040/A40P40
Z041=B041/A41P41
Z042=B042/A42P42
Z043=B043/A43P43
Z044=B044/A44P44
Z045=(B045-A45P37*Z037)/A45P45
Z046=B046/A46P46
Z047=B047/A47P47
Z048=B048/A48P48
Z049=(B049-A49P47*Z047)/A49P49
Z050=(B050-A50P35*Z035-A50P45
&*Z045)/A50P50
Z051=B051/A51P51
Z052=(B052-A52P44*Z044-A52P46
&*Z046)/A52P52
Z053=(B053-A53P32*Z032-A53P38
&*Z038-A53P52*Z052)/A53P53
Z054=(B054-A54P11*Z011-A54P52
&*Z052)/A54P54
Z055=(B055-A55P53*Z053)/A55P55
Z056=B056/A56P56
Z057=(B057-A57P25*Z025)/A57P57
Z058=B058/A58P58
Z059=B059/A59P59
Z060=B060/A60P60
X054=Z054-A54P55*Z055-A54P56*Z056
X044=Z044-A44P46*Z046
X038=Z038-A38P52*Z052
X033=Z033-A33P48*Z048
X032=Z032-A32P38*Z038-A32P53*Z053
X017=Z017-A17P53*Z053
X006=Z006-A06P48*Z048
X005=Z005-A05P52*Z052
X003=Z003-A03P57*Z057
END
```


[5] 富田,末吉,"並列処理マシン",オーム社、1989.

[6] M.Wolfe, " Optimizing Supercompilers for Supercomputers," MIT Press,1989.

[7] M.D.Guzzi,D.A.Padua,J.P.Hoeflinger and D.H.Lawrie, "Cedar Fortran and other Vecter and Parallel Fortran Dialects , " Univ.of Illinois at Urbana-Champain CSRD Rpt.No.731, Aug.1988

[8] 笠原, 成田, 橋本, " OSCARの アーキテクチャ", 信学論D,Vol.J-71D, No.8 ,Aug.1988.

[10] C.D.Polychronopoulos, Parallel Prpgramming and Compilers, Kluwer Academic Pub.,1988.

[11] D.Gajski, D.Kuck, D.Lawrie and A.Sameh,"CEDAR," Report UIUCDCS-R-83-1123, Dept. of Computer Sci., Univ.Illinois at Urbana-Champaign, Feb.1983.

[12] R.M.Tomasulo, "An Efficient Algorithm for Exproiting Murtiple Arithmetic Units ," IBM Journad of Res. and Dev., Vol.11,No.1, pp.25-33,Jan.1967.

[13] J.A.Fisher, "Trace Scheduring : A Technique for Global Microcode Compaction," IEEE Trans. Comput., Vol.C-30, No.7,pp.478-490, Jul.1981.

[14] J.R.Ellis, " Bulldog : A Compiler for VLIW Architectures," MIT Press,1985.

[15] H.Kasahara, T.Fujii, H.Nakayama and S.Narita, " A parallel prosessing scheme for the solution of sparse linear equations using static optimal multiprocessor scheduling algorithms, " Proc. 2nd Int. Conf. on Supercomputing,May 1987.

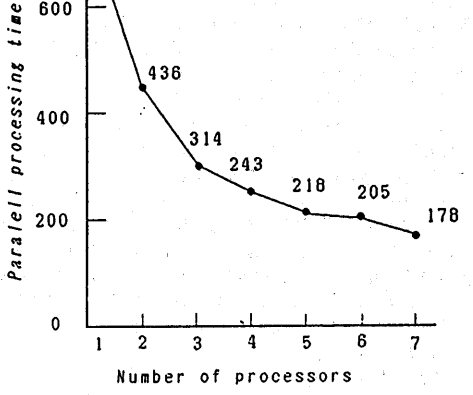
[16] H.Kasahara and S.Narita, " An aproach to supercomputing using multiprocessor scheduling algorithms, " in Proc. IEEE 1st Int. Conf.on Supercomputing, pp.139-148, Dec.1985.

[17] J.B.dennis, " Dataflow Supercomputer, " IEEE Computer, Vol.13,No.11, pp.48-56,

[18] A.Nicolau, "Uniform Parallelism Exploitation in Ordinary Programs," Proc.1985 Int. Conf. Parallel Processing, Aug.1985.

[19] F.G.Gustavson, W.Liniger and R.A.Willoughby, " Symbolic generation of an optimal Crout algorithm for sparse systems of liniar equations." J.ACM, Vol.17, pp.87-109, Jan1970.

[20] H.Kasahara and S.NArita,"Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing," IEEE Trans. Comput., Vol.c-33,No.11, pp.1023-1029,Nov.1984.



（a） シンボリック生成法により生成されたFORTRANプログラム　（b）OSCAR上での並列処理時間

図7 線形スパース方程式求解における並列処理