(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4082706号
(P4082706)
(45)発行日　平成20年4月30日(2008.4.30)　(24)登録日　平成20年2月22日(2008.2.22)

(51)Int.Cl.　F I
G06F 9/45 (2006.01)　G06F 9/44 322F

請求項の数 6　(全 33 頁)

(21)出願番号　特願2005-114842(P2005-114842)
(22)出願日　平成17年4月12日(2005.4.12)
(65)公開番号　特開2006-293768(P2006-293768A)
(43)公開日　平成18年10月26日(2006.10.26)
審査請求日　平成19年3月2日(2007.3.2)

早期審査対象出願

前置審査

(73)特許権者　899000068
学校法人早稲田大学
東京都新宿区戸塚町１丁目１０４番地
(74)代理人　100075513
弁理士　後藤　政喜
(72)発明者　笠原　博徳
東京都新宿区大久保３丁目４番１号　学校法人　早稲田大学　理工学部内
(72)発明者　木村　啓二
東京都新宿区大久保３丁目４番１号　学校法人　早稲田大学　理工学部内
(72)発明者　白子　準
東京都新宿区大久保３丁目４番１号　学校法人　早稲田大学　理工学部内

最終頁に続く

(54)【発明の名称】マルチプロセッサシステム及びマルチグレイン並列化コンパイラ

(57)【特許請求の範囲】
【請求項１】
単一または複数種類のプロセッサユニットを複数個有するマルチプロセッサシステムに実行させる目的プログラムを生成するコンパイラであって、
前記プロセッサユニットは、特定用途プロセッサユニットと汎用プロセッサユニットとを含み、
入力プログラムを読み込む処理と、
前記入力プログラムを解析し、前記入力プログラムを複数の粒度の単位ブロックに分割する処理と、
前記単位ブロック間の制御依存性、及びデータ依存性を解析し、前記単位ブロックの並列性を抽出する処理と、
前記マルチプロセッサシステムが具備する各プロセッサユニットで前記単位ブロックを処理するために必要な演算サイクル時間をコスト情報として求める処理と、
前記コスト情報に基づいて前記入力プログラムの処理時間が最小となるように、前記単位ブロックを処理するのに必要な前記特定用途プロセッサユニットを選択し、さらに少なくともひとつの汎用プロセッサユニットを選択し、これら特定用途プロセッサユニットと汎用プロセッサユニットを１つのグループとして前記単位ブロックを割り当てるスケジューリングコードを生成する処理と、
前記スケジューリングコードを入力プログラムに付加し、前記各プロセッサユニット毎の実行コードを生成し、目的コードとして出力する処理と、

を計算機に実行させることを特徴とするマルチグレイン並列化コンパイラ。
【請求項２】
単一または複数種類のプロセッサユニットを複数個有するマルチプロセッサシステムに実行させる目的プログラムを生成するコンパイラであって、
前記プロセッサユニットは、特定用途プロセッサユニットと汎用プロセッサユニットとを含み、
入力プログラムを読み込む処理と、
前記入力プログラムを解析し、前記入力プログラムを複数の粒度の単位ブロックに分割する処理と、
前記単位ブロック間の制御依存性、及びデータ依存性を解析し、前記単位ブロックの並列性を抽出する処理と、
前記マルチプロセッサシステムが具備する各プロセッサユニットで前記単位ブロックを処理するために必要な電力をコスト情報として求める処理と、
前記コスト情報に基づいて前記入力プログラムの処理時間が最小となるように、前記単位ブロックを処理するのに必要な前記特定用途プロセッサユニットを選択し、さらに少なくともひとつの汎用プロセッサユニットを選択し、これら特定用途プロセッサユニットと汎用プロセッサユニットを１つのグループとして前記単位ブロックを割り当てるスケジューリングコードを生成する処理と、
前記スケジューリングコードを入力プログラムに付加し、前記各プロセッサユニット毎の実行コードを生成し、目的コードとして出力する処理と、
を計算機に実行させることを特徴とするマルチグレイン並列化コンパイラ。
【請求項３】
前記スケジューリングコードを生成する処理は、
前記単位ブロックをスケジューリングによって割り当てられたプロセッサユニットで演算した際に、前記演算に必要とする時間が前記スケジューリングで許容された時間内であった場合は、
前記プロセッサユニットの演算に際して消費する電力を低減するためプロセッサユニットに与える動作電圧及び動作クロックを変化または遮断するための制御コードを生成する処理を含むことを特徴とする請求項１または請求項２に記載のマルチグレイン並列化コンパイラ。
【請求項４】
前記スケジューリングコードを生成する処理は、
前記単位ブロックをスケジューリングによって割り当てられたプロセッサユニット以外のプロセッサユニットについて、動作クロック及び動作電源の供給を遮断するための制御コードを生成することを特徴とする請求項１または請求項２に記載のマルチグレイン並列化コンパイラ。
【請求項５】
前記実行コードを生成し、目的コードとして出力する処理は、
前記各プロセッサユニットの種類に応じたローカルコンパイラを用いて前記実行コードを生成することを特徴とする請求項１または請求項２に記載のマルチグレイン並列化コンパイラ。
【請求項６】
前記マルチプロセッサシステムは、前記プロセッサユニットに複数のクロック周波数の内の一つを動作クロックとして供給するクロック供給部と、前記プロセッサユニットに複数の電圧の内の一つを動作電圧として供給する供給する電力供給部と、前記クロック供給部及び電力供給部がプロセッサユニットに供給する動作クロックと動作電圧とを設定するシステム制御レジスタを有し、
前記スケジューリングコードを生成する処理は、
前記単位ブロックをスケジューリングによって割り当てられたプロセッサユニットで演算した際に、前記演算に必要とする時間が前記スケジューリングで許容された時間内であ

った場合は、
前記プロセッサユニットの演算に際して消費する電力を低減するため前記システム制御レジスタに設定する値を変更する制御コードを生成する処理を含むことを特徴とする請求項１または請求項２に記載のマルチグレイン並列化コンパイラ。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、複数のプロセッサユニットで構成されるマルチプロセッサシステムにおいて、当該複数プロセッサユニットを効率よく動作させることを可能とするマルチプロセッサシステムのハードウェア構成及びプログラムを生成するコンパイラに関する。
【背景技術】
【０００２】
半導体製造技術の進歩による素子の微細化により、膨大な数のトランジスタを集積することが可能となっている。それと同時にプロセッサの高周波数化も進むが、動作時電力の増加、またリーク電流に起因する待機時電力の増加により、従来のプロセッサが歩んできた動作周波数の向上と論理方式の改善により達成してきた性能向上に限界が見え始めている。
【０００３】
そこで現在、性能改善と低電力化を実現する手段として、従来のＣＰＵ、ＤＳＰといったプロセッサユニット（以下、ＰＵとする）を複数個オンチップで搭載し、処理を並列で行うことで、動作周波数を向上させなくとも、高い演算性能を得ることが可能な、マルチプロセッサシステム（シングルチップ・マルチプロセッサシステム）が有望となっている。将来、微細化がさらに進むことで、ＰＵをオンチップで１００個～１０００個積載することも可能となると予測される。
【０００４】
このようなマルチプロセッサシステムにおいて、ＰＵの数に比例した演算性能を得るためには、搭載されたＰＵを同時に稼動させプログラムを処理する必要がある。しかしながら、通常の入力プログラムは処理が時系列で逐次的に記述されているため、複数のＰＵを搭載するにも拘わらず、当該複数ＰＵに比例して期待される演算性能を得ることができない。
【０００５】
この問題点を解決するための一つの方法として、プログラム開発者が自らプログラムの並列性を考慮し、当該プログラムを実行させるマルチプロセッサシステムの構成に基づいて、当該プログラムを複数のＰＵで実行させるための並列化コードを付加する必要がある。しかしながら、本手法はＰＵが数個のシステムにおいては有効であるが、将来の数十～数千といった数のＰＵが積載されたシステムにおいては、また特にＰＵが異種で構成される場合は、開発時間、実効性能の点で実用的ではない。
【０００６】
そこで、構成及び演算性能が同種の複数ＰＵで構成されたマルチプロセッサシステムにおいて、入力プログラムを解析し、当該プログラム中から並列に動作可能な部分を抽出し、当該部分を複数のＰＵに割り当て同時に実行することを可能とする、自動並列化コンパイラの研究がすでに行われている。例えば、入力ソースプログラムを解析し、当該プログラムをサブルーチンやループなどさまざまな粒度のブロック（タスク）に分割し、当該複数タスク間の並列性を解析すると共に、タスクとそれらがアクセスするデータをキャッシュあるいはローカルメモリに適合するサイズに分割し、それらを各ＰＵに対して当該タスクを最適割り当てすることによって、マルチプロセッサシステムを効率よく動かす目的プログラムを生成するコンパイル方式が、特許文献１に開示されている。またマルチグレイン並列処理の機能をサポートするチップマルチプロセッサのアーキテクチャとしては、特許文献２に開示されている。
【０００７】

また、電力消費の低減及び排熱の低減のため、マルチプロセッサシステムにおいて各ＰＵの低電力化が必須となるが、個別のプロセッサに関して低電力化する手法に関し、様々な提案が行われている。例えば、リアルタイム処理制約内でプロセッサの動作クロックを低減させ、そのクロック周波数に応じた電圧プロセッサに供給する、という周波数・電圧を動的に制御することにより低電力化を達成する手法が、特許文献３、４で開示されている。

【０００８】

また、画像処理など予め処理の手順が判明しているアプリケーションにおいて、各処理の特性に合わせてＣＰＵやＤＳＰなどの種類の異なる複数のプロセッサを組み合わせ、当該プロセッサ上での処理時間や消費電力情報を予め測定し与えておくことで、当該情報により動的に一連の処理を当該各プロセッサに割り当てる手法が、特許文献５に開示されている。

【特許文献１】特開２００４－２５２７２８号
【特許文献２】特開２００１－１７５６１９号
【特許文献３】特許第３１３８７３７号
【特許文献４】特開２００４－２３４１２６号
【特許文献５】特開２００４－２５２９００号

【発明の開示】

【発明が解決しようとする課題】

【０００９】

現在、自動車のナビゲーションシステム、携帯電話、デジタルテレビなどといった、画像、音声、データベース情報など多様なデータを同時に扱う新アプリケーションが生まれる中で、様々な種類の入力データを、当該データ夫々に最適な方法で同時に処理をするために、プロセッサは複数種類のＰＵを搭載することになると考えられる。従来のマルチプロセッサシステムでは、上記特許文献１で開示されているように同じ構成のＰＵを複数搭載したホモジニアスなプロセッサシステムであった。

【００１０】

しかしながらこのように、多様なアプリケーションが同時に多種のＰＵ上で処理される将来のヘテロジニアスマルチプロセッサシステムにおいて、搭載するＰＵの種類と数に比例して処理性能を得るためには、ＰＵの種類を考慮したプログラムの並列化と配置が必須となるという問題がある。

【００１１】

また従来、複数のＰＵ上でプログラムを効率よく実行するためには、小規模なプログラムや処理シーケンスが常に固定的に実行できる場合、スケジューリングを行うための実行時間などのスケジュール情報を得るために、一度プログラムを当該システム上で実行して処理時間などを測定することが必要となり、当該測定値を元に開発者が予め当該スケジュール情報を手動で生成する必要があり、多大な労力と時間が必要となる。

【００１２】

この場合、事前に処理の内容やシーケンスが不明な一般のプログラムの場合、特に規模が大きなプログラムの場合は、当該情報を事前に手動で生成することが困難となる。またＰＵの種類や数が増加した場合も同じく、当該情報を手動で生成することが困難となる。

【００１３】

また、多数のＰＵが搭載されるシステムでは、プロセッサ全体の消費電力増大が懸念されるため、特に携帯電話などのモバイル機器や、家庭で使用されるデジタルテレビなどに適用することを考慮すると、従来のＯＳによるＦＶ（動作周波数と駆動電圧）制御のみならず、各アプリケーションプログラム内でのソフトウェアによる各ＰＵの処理状況によりきめ細かい電源管理や動作周波数制御が必要となると。

【００１４】

特に、処理性能を落とさずに消費電力を低減する必要がある。また、実時間処理を要求するプログラムを実行するに際しても、時間制約を遵守しつつ、電力を低減する必要があ

る。

【００１５】

そこで本発明は、多様な種類のＰＵを搭載するマルチプロセッサシステムにおいて、処理対象となる入力プログラムから自動的に並列性を持つタスクを抽出し、各ＰＵの特性に合わせて当該タスクを配置することで当該ＰＵを効率よく動かし、さらに当該ＰＵの処理量を見積もることで動作周波数や電源電圧を最適化するコードを生成し、目的プログラムに付加するコンパイラ及びその最適化を可能とするマルチプロセッサシステムを提供することを目的とする。

【課題を解決するための手段】

【００１７】

本発明は、単一または複数種類のプロセッサユニットを複数個有するマルチプロセッサシステムに実行させる目的プログラムを生成するコンパイラであって、前記プロセッサユニットは、特定用途プロセッサユニットと汎用プロセッサユニットとを含み、入力プログラムを読み込む処理と、前記入力プログラムを解析し、前記入力プログラムを複数の粒度の単位ブロックに分割する処理と、前記単位ブロック間の制御依存性、及びデータ依存性を解析し、前記単位ブロックの並列性を抽出する処理と、前記マルチプロセッサシステムが具備する各プロセッサユニットで前記単位ブロックを処理するために必要な演算サイクル時間をコスト情報として求める処理と、前記コスト情報に基づいて前記入力プログラムの処理時間が最小となるように、前記単位ブロックを処理するのに必要な前記特定用途プロセッサユニットを選択し、さらに少なくともひとつの汎用プロセッサユニットを選択し、これら特定用途プロセッサユニットと汎用プロセッサユニットを１つのグループとして前記単位ブロックを割り当てるスケジューリングコードを生成する処理と、前記スケジューリングコードを入力プログラムに付加し、前記各プロセッサユニット毎の実行コードを生成し、目的コードとして出力する処理と、を計算機に実行させる。

また、単一または複数種類のプロセッサユニットを複数個有するマルチプロセッサシステムに実行させる目的プログラムを生成するコンパイラであって、前記プロセッサユニットは、特定用途プロセッサユニットと汎用プロセッサユニットとを含み、入力プログラムを読み込む処理と、前記入力プログラムを解析し、前記入力プログラムを複数の粒度の単位ブロックに分割する処理と、前記単位ブロック間の制御依存性、及びデータ依存性を解析し、前記単位ブロックの並列性を抽出する処理と、前記マルチプロセッサシステムが具備する各プロセッサユニットで前記単位ブロックを処理するために必要な電力をコスト情報として求める処理と、前記コスト情報に基づいて前記入力プログラムの処理時間が最小となるように、前記単位ブロックを処理するのに必要な前記特定用途プロセッサユニットを選択し、さらに少なくともひとつの汎用プロセッサユニットを選択し、これら特定用途プロセッサユニットと汎用プロセッサユニットを１つのグループとして前記単位ブロックを割り当てるスケジューリングコードを生成する処理と、前記スケジューリングコードを入力プログラムに付加し、前記各プロセッサユニット毎の実行コードを生成し、目的コードとして出力する処理と、を計算機に実行させる。

【発明の効果】

【００２１】

したがって、本発明により、入力プログラムをコンパイルする際に、プロセッサユニットの構成に基づきコンパイラが予め事前の処理時間を見積もり、コンパイル時に処理順序を静的に決定できる部分に関しては事前に処理手順を決定しておき、また実行時にならないと処理順序が決定できない部分に関しては、コンパイラが実行時間を含んだ処理情報に基づいたスケジューリングを動的に行うプログラムを生成することで、一般のプログラムを多種のプロセッサユニットで構成されるマルチプロセッサシステム上で効率よく処理できる。

【００２２】

また、入力プログラムをコンパイラが解析し、事前に処理手順を決定した後に処理時間を見積もり、時間制約に対する当該処理時間の余裕度を見てプロセッサユニット毎に電源

管理や周波数制御をきめ細かく行うことで、電力を大きく低減できる。

【００２３】

また、異種のプロセッサユニットで構成されるマルチプロセッサにおいて、実行時間最小となるよう入力プログラムを並列化しスケジューリングした後、各プロセッサユニット間で処理時間が最小となるよう、プロセッサユニット毎にきめ細かく動作周波数制御及び電源管理を行うことで、性能を損なわずに電力を最適化できる。

【発明を実施するための最良の形態】

【００２４】

以下、本発明の一実施形態を添付図面に基づいて説明する。

【００２５】

＜実施形態の全体構成＞

図１は、本発明の一実施形態であるマルチプロセッサシステムの構成を示す。図１において、マルチプロセッサシステムは、複数の異種のプロセッサユニット（以下、ＰＵとする）１０～１７と、これら共有メモリ（以下、ＳＭとする）１８を主体にして構成される。当該各ＰＵ１０～１７は、それぞれのバスインタフェース（ＢＩＦ）２７を介し、ローカルバス（ＬＢＵＳとする）１９に接続される。また、ＳＭ１８はＬＢＵＳ１９に接続され、各ＰＵ１０～１７からアクセスすることができる。当該各ＰＵ１０～１７は、当該ＰＵに対し電源電圧及び動作クロックを供給する電源電圧生成回路（ＤＣＧＥＮ）２０、及びクロック生成回路（ＣＬＫＧＥＮ）２１が接続されている。なお、本マルチプロセッサシステムは、異種のプロセッサユニットで構成されたヘテロジニアスマルチプロセッサシステムの例を示す。

【００２６】

本実施形態ではＰＵの種類と個数を、２個の汎用処理プロセッサ（以下、ＣＰＵ）１０、１１と、２個の信号処理プロセッサ（以下、ＤＳＰ）１４、１５と、２個の動的再構成可能プロセッサ（以下、ＤＲＰ）１６、１７と、及び２個のビット演算処理プロセッサ（以下、ＢＭＰ）１２、１３で構成した例を示す。なお、上記ＰＵの種類や数は本実施形態で示した限りではなく、さまざまな構成を取り得る。また、従来の同種のＰＵのみによる構成（例えばＣＰＵのみ４個で構成）としても良い。また、本実施形態では、ＤＳＰ（０、１）１４、１５と、ＤＲＰ（０、１）１６、１７と、ＢＭＰ（０、１）１２、１３を特定用途プロセッサユニットとし、ＣＰＵ（０、１）１０、１１を汎用プロセッサユニットとする。

【００２７】

また、動的再構成可能プロセッサ１６、１７は、処理回路を動的に再構成することが可能なプロセッサを指し、限られたコア内で仮想的に回路を変更することができる。

【００２８】

上記各ＰＵ１０～１７は、各ＰＵで処理されるプログラムやデータを一時的に保存するローカルメモリ（またはキャッシュ）（ＬＭ）２４及び、当該ＰＵに対する供給電圧（ＶＬ）や動作周波数（ＦＬ）を決定する周波数・電源電圧（ＦＶ）制御、及びＰＵ間の同期制御を行うためのシステム制御レジスタ（Ｒ）２５を具備する。なお、ローカルメモリ（ＬＭ）２４は、他のＰＵ及び当該ＬＭを持つ自ＰＵからアクセス可能なグローバルアドレスがマッピングされている領域と、当該自ＰＵのみアクセス可能なプライベートアドレスがマッピングされている領域に分割される。なお、ＬＢＵＳ１９に接続された共有メモリ（ＳＭ）１８はグローバルアドレスがマッピングされており、複数のＰＵからアクセス可能である。なお、以上は本発明における適用構成の一例に過ぎず、実施の形態としてはこの限りでない。例えば図１では表現されていないが、ＬＢＵＳ１９には入出力処理、割り込み処理、タイマ、デバッグ回路、等の周辺回路を必要に応じて接続することとなる。また、バスブリッジを介して、同種類または異種類のバスを階層的に接続してもよい。

【００２９】

また、各ＰＵ１０～１７で共有されるＳＭ１８には、ＳＭ１８に供給する供給電圧（ＶＬ）や動作周波数（ＦＬ）を設定して、周波数・電源電圧（ＦＶ）制御を行うためのシス

テム制御レジスタ（Ｒ）１８１を具備する。また、前記ＰＵ及び前記ＳＭを相互に接続するローカルバスＬＢＵＳ１９には、ＬＢＵＳ１９に供給するＶＬやＦＬを設定して、ＦＶ制御を行うためのシステム制御レジスタＲ１９１を具備する。なお、このシステム制御レジスタ（Ｒ）１８１に代わって、各ＰＵ１０～１７のシステム制御レジスタ（Ｒ）２５のそれぞれに、ＳＭ１８及びＬＢＵＳ１９のＦＶ制御用のレジスタを設けても良い。また、システム制御レジスタ（Ｒ）１８１は、ＰＵ１０～１７のいずれか一つによって設定される。

【００３０】

なお、上記ＰＵ１０～１７は、一つのチップ（ＬＳＩ）上に構成された場合を示すが、ＰＵ１０～１７を構成する複数のＬＳＩを結合し、一つのチップまたはモジュールとしたものであっても良い。

【００３１】

また、ＳＭ１８は、プロセッサの種類毎（ＣＰＵ、ＤＳＰ、ＤＲＰ、ＢＭＰ毎）に共有するようにしても良く、例えば、後述する図１０のように、ＳＭ１８の領域をバンクに分割し、各バンクをプロセッサの種類毎に共有することもできる。あるいは、ＳＭ１８の一部を複数のＰＵで共有するようにいしても良い。

【００３２】

＜電圧・周波数可変回路＞

次に、各ＰＵに接続された電源電圧生成回路（ＤＣＧＥＮ）２０及びクロック生成回路（ＣＬＫＧＥＮ）２１の構成について説明する。

【００３３】

図２は、ＤＣＧＥＮ２０の構成を示す。ＤＣＧＥＮ２０は外部より供給された通常の電源電圧（ＶＤ）２２を予め指定した複数の供給電圧（ＶＬ）２０５に降圧（または昇圧）する回路（ＤＣＣＮＶ）２０６、生成した複数の当該供給電圧よりＰＵへ供給する電圧を選択する供給電圧選択回路（ＶＤＳＥＬ）２０３、及び電源電圧を遮断する回路（後述）で構成する。

【００３４】

なお降圧（または昇圧）回路（ＤＣＣＮＶ）２０６は、複数の電圧降下部を含み、例えば、図２のように、電源電圧ＶＤをそのまま供給する回路２２’と、電源電圧ＶＤを３／４に低下する降圧回路（ＬＶＣＮＶ）２０１と、電源電圧ＶＤを１／２に低下する降圧回路（ＬＶＣＮＶ）２０２とから構成される。

【００３５】

各ＰＵ１０～１７は、当該ＰＵが持つシステム制御レジスタ（Ｒ）２５内のＦＶ制御レジスタに設定されたＦＶ（駆動周波数及び駆動電圧）モードにより制御線（ＤＣＣＬ）２０４を介して供給電圧選択回路（ＶＤＳＥＬ）２０３を制御することで、降圧回路（ＤＣＣＮＶ）２０６で生成された複数の電圧のうちいずれかひとつを選択し、当該選択された供給電圧ＶＬが対応するＰＵ１０～１７に供給される。なお、ＦＶ制御レジスタはシステム制御レジスタ２５の内の所定の領域に設定されるものである。

【００３６】

電源電圧を降圧する回路（ＬＶＣＮＶ）の構成の一例を図３に示す。上記図２の降圧回路ＬＶＣＮＶ２０１、２０２は、供給電圧（３／４V_D）を決定する参照電圧生成回路（ＶＲＥＦ）２０７、及び参照電圧で指定した電圧まで降圧しＰＵに対し電流を供給する電源生成回路（ＶＧＥＮ）２０８で構成され、電源生成回路２０８の出力が図２の供給電圧選択回路２０３に入力される。

【００３７】

降圧回路ＬＶＣＮＶの出力電圧（V_{CNV}）は、参照電圧を決定する参照電圧生成回路（ＶＲＥＦ）２０７内のｎＭＯＳＦＥＴの段数により決まり、ＦＶ制御レジスタで指定する電圧を供給するよう、ＬＶＣＮＶの構成は決定される。以上の回路で降圧した複数の電圧から、ＰＵからの制御線２０４により指定された電圧を選択し、各ＰＵ１０～１７へ出力（２０５）する。

【００３８】
図４は、供給電圧選択回路（ＶＤＳＥＬ）２０３の構成の一例を示す。供給電圧選択回路（ＶＤＳＥＬ）２０３は制御信号デコード部２０３１と電圧選択スイッチ２０３２で構成する。また、ＰＵのソース電圧供給部にしきい値の高いｎＭＯＳＦＥＴ２０９を挿入することで、当該ＰＵの電源遮断時に流れるリーク電流を低減することができる。なお、図３、図４の構成は、降圧回路ＬＶＣＮＶの機能を実現するための一つの構成に過ぎず、他の様々な電源電圧生成回路方式を適用しても良い。
【００３９】
続いて図５に、クロック生成回路（ＣＬＫＧＥＮ）２１の構成の一例について説明する。ＣＬＫＧＥＮ２１は、内部クロックＦＣ（２３）を当該内部クロックの１／２、１／４など内部クロックＦＣを整数分の１に低減する分周回路２１２、２１３、及び分周生成した複数のクロックから、当該ＰＵへ供給するクロック（ＦＬ）２１６を選択する、クロックパルス選択器（ＣＬＫＳＥＬ）２１４で構成する。なお、内部クロックＦＣ（２３）は、ＰＬＬ（Ｐｈａｓｅ　Ｌｏｃｋｅｄ　Ｌｏｏｐ）回路２１１にて、外部から入力されたシステムクロックを、指定した逓倍率で逓倍することで生成される。
【００４０】
クロックパルス選択器（ＣＬＫＳＥＬ）２１４の構成の一例を図６に示す。ＣＬＫＳＥＬ２１４は入力制御信号ＣＫＣＬのデコード部２１７１と、クロック信号選択部２１７２で構成する。各ＰＵは、当該ＰＵが持つＦＶ制御レジスタのモードにより制御線（ＣＫＣＬ）２１５を介してＤＣＳＥＬ２１４を制御することで、生成された複数のクロックより指定されたＦＬが選択され、当該クロックＦＬ２１６の供給を受ける。
【００４１】
なお、上記図１に示した構成の他にも図７に示すように、電源電圧、クロックパルスを変換する回路（ＤＣＣＮＶ２０６、ＣＬＫＣＮＶ２１７）を複数ＰＵ（またはＰＵ全体）に対し各１個付加し、各ＰＵ側に生成した複数種類の電源電圧及びクロックパルスを選択する回路（ＶＤＳＥＬ２０３、ＣＬＫＳＥＬ２１４）を付加する構成としても良い。図７の例では、２組の電源電圧生成回路２０６とクロック生成回路２１７が、それぞれ４つのＰＵに電力とクロックを供給する場合を示す。
【００４２】
このように、電源電圧及びクロックパルスを生成、供給する回路は様々な構成を取り得ることができ、その構成は以上に示した限りではない。例えば、電源電圧を変換する回路ＤＣＣＮＶ２０６を複数ＰＵ（またはＰＵ全体）に対し１個付加し、クロックパルスを生成する回路ＣＬＫＣＮＶ２１７はＰＵ側にＰＵ毎に付加する構成とすることもできる。また、例えば、クロックパルスを生成する回路ＣＬＫＣＮＶ２１７を複数ＰＵ（またはＰＵ全体）に対し１個付加し、電源電圧を変換する回路ＤＣＣＮＶ２０６はＰＵ側にＰＵ毎に付加する構成とすることもできる。また、例えばチップ内にＤＣＣＮＶ２０６を搭載せず、チップ外部にて生成した複数種類の電源電圧を入力し、ＶＤＳＥＬにて所望の供給電圧ＶＬ２０５を選択供給する構成としてもよい。
【００４３】
また、ＦＶ制御モードを設定する手段として、図１や図７に示した各ＰＵが当該ＰＵのＦＶ制御を行うＦＶ制御レジスタ（システム制御レジスタ（Ｒ）２５）を持つ構成とする他に、図８に示すように各各ＰＵ１０～１７のＦＶ制御モードを一括して保持するＦＶ制御テーブル（ＦＶＴＢＬ）２６をＬＢＵＳ１９に接続する構成としてもよい。図８においては、図１においてシステム制御レジスタ２５に含まれていたＦＶ制御レジスタを、ローカルバスＬＢＵＳ１９に接続されたＦＶ制御テーブル２６とし、集約したものである。
【００４４】
ＦＶ制御テーブル（ＦＶＴＢＬ）２６は各ＰＵ１０～１７からアクセス可能であって、共有メモリＳＭの一部に設定されてもよいし、共有メモリＳＭから独立したメモリ（またはレジスタ）で構成されても良い。
【００４５】

＜バスＩＦ＞
以上のように、各ＰＵ１０～１７は電源電圧を個別に設定することが可能なため、当該ＰＵと接続されたＬＢＵＳ１９間においては、信号の電圧レベルが異なることになる。そこで、ＰＵ１０～１７とＬＢＵＳ１９間に接続されたバスインタフェース（ＢＩＦ）２７は信号レベル変換回路（図示省略）を具備し、バスＬＢＵＳ１９とＰＵ１０～１７間の信号レベル変換を行う。

【００４６】
＜ＰＵに対する電圧・周波数モードの与え方＞
次に、電源電圧生成回路（ＤＣＧＥＮ）２０及びクロック生成回路（ＣＬＫＧＥＮ）２１で生成する電源電圧（ＶＬ）２０５及び動作クロック（ＦＬ）２１６を決定するハードウェア機構について説明する。

【００４７】
電源電圧生成回路ＤＣＧＥＮ２０及びクロック生成回路ＣＬＫＧＥＮ２１の動作モード（供給するＶＬ、ＦＬ値）は設計時に予め決定されており、各ＰＵは前記回路に対し当該ＰＵが具備するＦＶ制御レジスタの値によって制御線ＤＣＣＬ、ＣＫＣＬを介して指定する。ＶＬ、ＦＬの設定方法の詳細に関しては後述するが、コンパイラがＦＶ制御レジスタをセットする制御コードを生成し、当該コードを実行する汎用プロセッサＣＰＵ０またはＣＰＵ１が、メモリマップされたＦＶ制御レジスタにアクセスして値を書き換える。

【００４８】
本実施形態におけるＰＵ１０～１７のＦＶモードは、ＶＬ、ＦＬの組み合わせを４段階（ＦＶ制御レジスタ内の２ビット）として設定する。図９にＰＵ１０～１７の動作モード一覧を示す。つまり、ＦＶ制御レジスタの２ビットの値が“００”においてはＶＬ＝０、ＦＬ＝０の電圧・周波数遮断となるＯＦＦモードに設定され、上記レジスタ値が“１１”においては、ＶＬ＝ＶＤでシステムの電源電圧と等価、ＦＬ＝ＦＣでシステムの動作周波数と等価とするＦＵＬＬモードが設定される。

【００４９】
またレジスタ値が“０１”ではＶＬ＝（１／２）ＶＤ、ＦＬ＝（１／４）ＦＣとなるＬＯＷモードに設定され、レジスタ値が“１０”ではＶＬ＝（３／４）ＶＤ、ＦＬ＝（１／２）ＦとなるＭＩＤＤＬＥに設定される。なお、ＶＬ・ＦＬモードの数、またＶＬ値・ＦＬ値は、構築するシステムの形態やアプリケーション、使用するプロセス技術、等により、決定される。

【００５０】
＜ＰＵ内の部分的なＦＶ制御＞
以上ではＦＶ制御を対象とする範囲をＰＵ全体とし、一括でＦＶ制御モードを設定するとして説明したが、ＰＵ１０～１７に搭載するローカルメモリ（ＬＭ）やＦＶ制御レジスタ、また他のプロセッサ周辺回路に対し、夫々異なるＦＶ制御モード設定を行っても良い。これは、ＦＶ制御レジスタのビットフィールドを拡張し、被ＦＶ制御部に対応したＦＶ制御モードを設定するフィールドを持たせることで実現できる。例えば、ローカルメモリＬＭまたはシステム制御レジスタ（Ｒ）２５といったデータ保持が必要な回路部に対しては、独立にＦＬ、ＶＬを設定する機構とすることが考えられる。つまりローカルメモリＬＭやＦＶ制御レジスタに対し独立してＦＶ制御を行うことで、前記ＰＵのＦＶが遮断状態としても、当該ＬＭ及びＲのデータが保持され、また対象ＰＵが遮断状態においても、他のＰＵから当該ＰＵのＬＭに対しアクセスすることも可能となる。

【００５１】
またさらに、ローカルメモリＬＭの構成によりＦＶ制御対象を複数設定することができる。図１０にローカルメモリＬＭに対するＦＶ制御方式を示す。

【００５２】
例えば、図１０（ａ）に示す通りローカルメモリＬＭをバンク構成とし、各バンク毎（Ｂａｎｋ０～Ｂａｎｋ３）に対してそれぞれＦＶ制御を実施する。このため、各Ｂａｎｋ０～３にはそれぞれ供給電圧選択回路（ＶＤＳＥＬ）２０３が接続される。

【００５３】
つまり、データの保持が必要なバンクのみ通常電圧、またはデータ保持に必要な最低限の電圧を供給し、当該バンク以外の他のバンクは電源を遮断することで電力を削減すると共に、データの退避処理を行う必要がなくなるためＰＵの電源遮断時から通常動作時への復帰を高速に行うことが可能となる。
【００５４】
またさらには、図１０（ｂ）に示す通り、ローカルメモリＬＭのアドレス空間を一定の連続したアドレス区間（Ａｒｅａ１～Ａｒｅａ４）で分割し、当該区間単位でＦＶ制御を行うことで、不要なアドレス区間（記憶領域）に対する電源を遮断することで電力を削減できる。
【００５５】
このため、ローカルメモリＬＭの各アドレス区間（Ａｒｅａ１～Ａｒｅａ４）毎に供給電圧選択回路（ＶＤＳＥＬ）２０３が接続される。
【００５６】
また、図１０（ｃ）に示す通り、ローカルメモリＬＭがバンク構成としたとき、各バンク（Ｂａｎｋ０～３）に跨る一定の連続したアドレス区間（Ａｒｅａ１～Ａｒｅａ４）で分割した単位でＦＶ制御を行う。
【００５７】
このため、ローカルメモリＬＭの各Ｂａｎｋ０～３に跨る各アドレス区間（Ａｒｅａ１～Ａｒｅａ４）毎に供給電圧選択回路（ＶＤＳＥＬ）２０３が接続される。この構成により、バンク構成を活用しメモリアクセスを高速化するメモリインタリーブを実現しつつ低電力化が可能となる。
【００５８】
なお、ローカルメモリＬＭは、機能的に当該ＬＭを搭載するＰＵのみからアクセスできる部分（非共有メモリ）と、当該ＰＵのみならず他のＰＵからもアクセス可能な部分（分散共有メモリ）に分割して実装してもよく、以上の２つのメモリ機能単位別でＦＶ制御を行うことも考えられる。このため、図示はしないが、上記非共有メモリと分散共有メモリの領域毎にそれぞれ供給電圧選択回路（ＶＤＳＥＬ）が接続される。
【００５９】
また、図１０に示したメモリ分割手法は、前記メモリ機能単位、また共有メモリＳＭなど、システムが搭載する様々なメモリや機能部位単位に対しても、同様に適用することが可能である。例えばＰＵ外に配置した共有メモリＳＭに対しても複数バンク構成とし、各バンクに対するアクセス頻度やシステム状態（スタンバイ、スリープなど）に対応して、別途ＦＶ制御を行うようにしてもよい。
【００６０】
例えば、図１０（ａ）の構成をローカルメモリＬＭに代わって共有メモリＳＭ１８に適用し、共有メモリＳＭ１８をバンク（Ｂａｎｋ０～３）に分割して、各バンク毎に供給電圧選択回路（ＶＤＳＥＬ）２０３を接続することでバンク単位で電力制御を行うことができる。また、図１０（ｂ）の構成をローカルメモリＬＭに代わって共有メモリＳＭ１８に適用し、共有メモリＳＭ１８のアドレス空間を一定の連続したアドレス区間（Ａｒｅａ１～Ａｒｅａ４）で分割し、当該区間単位でＦＶ制御を行うことで、不要なアドレス区間（記憶領域）に対する電源を遮断することで電力を削減できる。また、図１０（ｃ）の構成をローカルメモリＬＭに代わって共有メモリＳＭ１８に適用し、各バンク（Ｂａｎｋ０～３）に跨る一定の連続したアドレス区間（Ａｒｅａ１～Ａｒｅａ４）で分割した単位でＦＶ制御（電力制御）を行うことも可能である。なお、図１０（ａ）～（ｃ）を共有メモリＳＭ１８に適用する場合には、図中「ＬＭ」を「ＳＭ」と読み替えるものとする。
【００６１】
＜ローカルメモリＬＭに対するＦＶ制御モード＞
ＦＶ制御対象をＰＵ内の複数部分（機能部分）とする例として、ローカルメモリＬＭに対してＰＵとは独立してＦＶ制御を行う方法を以下に説明する。本例ではローカルメモリ

ＬＭは図１０（ａ）で示したように４バンク（Ｂａｎｋ０～３）で構成し、ＦＶ制御対象はＰＵ及びＬＭバンク毎とする。

【００６２】

ＰＵのＦＶ制御モードは上記図９に示した通りである。図１３にローカルメモリＬＭのＦＶ制御モードの一覧を示す。ＬＭの動作モードに関しては対象がメモリとなるため、本実施形態では通常メモリアクセス及びデータ保持が可能である通常動作モード（ＶＬ＝ＶＤ、ＦＬ＝ＦＣ、レジスタ値“１１”）と、メモリアクセスは不可であるがデータを保持可能なデータ保持モード（ＶＬ＝１／２ＶＤ、ＦＬ＝０、レジスタ値“０１”）と、データ保持を行わず完全に電源を遮断する、電源遮断モード（ＶＬ＝０、ＦＬ＝０、レジスタ値“００”）の３モードとする。

【００６３】

続いて、ローカルメモリＬＭを図１０（ａ）または図１０（ｃ）のように複数のバンクで構成し、バンク毎のＦＶ制御に対応したＦＶ制御レジスタのフォーマットを図１４（ａ）に示す。本例では、各ＰＵ１０～１７毎に、ＦＶ制御レジスタ１２５０を一メモリアドレスにマップし、そのフィールド（マップしたアドレス）で制御対象毎にＦＶ制御モードを決定する。１フィールドが図示のように３２ビットの場合、つまりビット１、０をＰＵのＦＶ制御モード（ＰＵＦＶ）、ビット３、２をＬＭバンク０（Ｂａｎｋ０）のＦＶ制御モード（ＬＭ０ＦＶ）、ビット５、４をＬＭバンク１（Ｂａｎｋ１）のＦＶ制御モード（ＬＭ１ＦＶ）、ビット７、６をＬＭバンク２（Ｂａｎｋ２）のＦＶ制御モード（ＬＭ２ＦＶ）、ビット９、８をＬＭバンク３（Ｂａｎｋ３）のＦＶ制御モード（ＬＭ３ＦＶ）として、所望のフィールドをアクセスし、図１３のテーブルからローカルメモリＬＭの動作モードを決定し、ＦＶ制御を行う。なお、ＰＵの動作モードは、図９のテーブルに基づいて決定する。

【００６４】

ＦＶ制御レジスタ１２５０の設定例を図１４（ｂ）に示す。本例では、ＰＵのモードは図１４（ａ）のＰＵＦＶにＭＩＤＤＬＥ（”１、０”）が設定され、ローカルメモリＬＭのバンク０のみを活性化し通常動作させるようＬＭ０ＦＶには”１、１”が設定され、ＬＭのバンク１はデータ保持モードとなるようにＬＭ１ＦＶには”１、０”が設定され、バンク２及びバンク３はＬＭ２ＦＶ、ＬＭ３ＦＶに”０、０”を設定し電源遮断としている。

【００６５】

また、上記の他にもＦＶ制御レジスタに対し、制御対象毎にアドレスを割り振る構成としても良い。例えば、図１４（ｃ）で示すようにＦＶ制御対象毎にレジスタをメモリアドレスにマップしたＦＶ制御レジスタのフォーマットを示す。図１４（ｃ）では、アドレスの順にＰＵのＦＶ制御モードとＬＭのＦＶ制御モードを示すＰＵＦＶ、ＬＭ０ＦＶ～ＬＭ３ＦＶが格納され。

【００６６】

上記図１４（ａ）で示したように、ＦＶ制御レジスタのビットフィールドで制御対象のＦＶ制御モードを切り替える場合、当該フィールドの値を設定するためのビット演算が必要となるが、図１４（ｃ）の構成では直接当該制御対象レジスタがマップされたアドレスに直接アクセスすれば良く、ＦＶ制御レジスタの設定に関する命令数を削減することができる。しかしその反面、アドレスリソースが図１４（ａ）と比較し多く必要となる。

【００６７】

以上では、各ＰＵが持つＦＶ制御レジスタを設定することで、ＰＵやＬＭのＦＶ制御モードを設定する例として説明したが、図８で説明したように各ＰＵ１０～１７のＦＶ制御モードを示すＦＶ制御レジスタを、ローカルバスＬＢＵＳ１９に接続されたＦＶ制御テーブル（ＦＶＴＢＬ）２６として持たせる場合では、図１５に示すように当該ＦＶＴＢＬを構成する。

【００６８】

図１５は、図８で示したＦＶ制御テーブルＦＶＴＢＬのフォーマットを示す。当該ＦＶ

ＴＢＬの１ラインは、ＰＵ番号（ＰＵＮ２５０）、当該ＰＵのＦＶ制御モード（ＰＵＦＶ２５１）、ＬＭのＦＶ制御モード（ＬＭ０ＦＶ２５２、ＬＭ１ＦＶ２５３、ＬＭ２ＦＶ２５４、ＬＭ３ＦＶ２５５）、後述するローカルバスＬＢＵＳのＦＶ制御モード（ＢＵＳＦＶ）に対応し、任意のＰＵに対する当該ＰＵ、ＬＭ、ＬＢＵＳのＦＶ動作モードを決定できる。

【００６９】

そして、ＣＰＵ０等がこのＦＶ制御テーブルＦＶＴＢＬを読み込んで、各ＰＵ１０～１７毎にＰＵとローカルメモリＬＭ（各バンク毎）及びローカルバスのＦＶ制御モードを決定し、電圧生成回路（ＤＣＧＥＮ）２０及びクロック生成回路（ＣＬＫＧＥＮ）２１を制御する。

【００７０】

＜バスに対する電源制御＞

また、各ＰＵ１０～１７を接続するローカルバス（ＬＢＵＳ）１９に対しても、マルチプロセッサシステムの機能部位単位のＦＶ制御として、部分的に電源制御（電源遮断）することができる。

【００７１】

例えば、ＰＵ１０～１７の何れかが非動作時で電源遮断されている場合、該当するＰＵのバスインタフェース（ＢＩＦ）はアクセスされないため、当該ＢＩＦの電源遮断を行うことができ、その結果リーク電流を削減することができる。また、バス構成をクロスバとした場合、当該ＰＵに接続するバスを決定するスイッチ部の電源を制御し遮断することもできる。

【００７２】

図１１に、クロスバ構成のバスに対する電源制御の概念を示す。例えば、ＤＳＰ０（１４）を電源遮断状態としＤＳＰ０に対する通信トラフィックが無いとすると、ＤＳＰ０に付随するＢＩＦ（２７）、及びＤＳＰ０に対し他のＰＵ及び共有メモリＳＭからのバスネットワークを接続するスイッチ群（１９２）、に対する電源も遮断する。

【００７３】

これにより、非動作状態となったＤＳＰ０のスイッチ群１９２の電力消費を削減できる。

【００７４】

図１２に、クロスバネットワークに対する電源制御を実現するための回路構成を示す。なお本図では、ＣＰＵ０、ＤＳＰ０、ＤＳＰ１及び共有メモリＳＭを、クロスバネットワークで構成されたローカルバスＬＢＵＳ１９に接続した構成を示す。本回路は、各ＰＵ１０～１７が送出したパケットデータを解析し、図１１に示したスイッチ群１９２の制御を行うネットワーク制御部ＮＷＣＲＬと、当該パケットデータの送出元と送出先のネットワークを接続するネットワークスイッチ部（ＮＷＳＷ）で構成する。

【００７５】

ネットワーク制御部ＮＷＣＲＬは、ＰＵ１０～１７が送出したパケットを解析しパケット処理の優先度を決定するＳＨＣＴＬ１９５と、ＳＨＣＴＬ１９５により優先度決定された当該パケットを選択するセレクタ（ＳＥＬＣ）１９６と、当該パケットを一時的に保持するキュー１９７と、当該パケットを解析し送出先及び送出元のネットワークを接続するセレクタスイッチ１９１～１９４を制御するＳＷＣＴＬ１９８から構成される。

【００７６】

また、ネットワークスイッチＮＷＳＷは各ＰＵ間のネットワークを接続するセレクタスイッチ（ＳＥＬ）１９１～１９４で構成する。

【００７７】

各ＰＵ１０～１７とネットワーク制御部ＮＷＣＲＬには、電源生成回路ＤＣＧＥＮと選択的に接続するスイッチ（ＤＣＳＥＬ）１９９がそれぞれ設けられる。そして、スイッチ（ＤＣＳＥＬ）１９９から各ＰＵ１０～１７及びＢＩＦ２７と、当該ＰＵが接続されるセレクタスイッチ１９１～１９４に対し電源供給を行う。

【００７８】
例えばＤＳＰ０を電源遮断状態とし、ＤＳＰ０に対する通信トラフィックが発生しないとすると、ＤＳＰ０に付加したスイッチＤＣＳＥＬ１９９は、当該ＤＳＰ０のみならず当該ＤＳＰ０に接続されたＢＩＦ２７及び、当該ＤＳＰ０へのネットワークを選択するセレクタスイッチＳＥＬ１（１９２）に対する電源を遮断する。これにより、電源遮断状態としたＤＳＰ０のみならず、周辺の回路への電力を遮断することで電力消費をさらに削減できる。なお、ネットワーク全体を待機状態、つまり電源遮断状態とするときは、ＮＷＣＲＬに対しても当該ＮＷＣＲＬへ電源供給するスイッチＤＣＳＥＬ１９９により、電源遮断を行う。
【００７９】
＜ＦＶ制御レジスタの設定方法＞
次に、ＦＶ制御レジスタ１２５０を設定する具体的な方法について説明する。なお以下では、レジスタフォーマットを図１４（ａ）の構成として説明する。
【００８０】
各ＰＵのＦＶ制御レジスタ２５には、全ＰＵから一意にアクセス可能なグローバルアドレスがそれぞれ割り振られ、コンパイラが予め決定したタスク管理用のＰＵ（つまり、スケジューラまたはＯＳを実行するＰＵ）が当該アドレスにアクセスし、当該レジスタ値を変更することでＦＶ制御モードを設定する。
【００８１】
図１６にマルチプロセッサシステム全体のグローバルアドレス空間マップを示す。本実施形態では、先頭アドレスより所定のアドレスまでＰＵ自身のローカルリソース（ＬＭ、ＦＶ制御レジスタ１２５０を含むシステム設定レジスタ）が見える領域、ブロードキャスト（ＢＣ）領域を定義する。ＢＣ領域に関しては後述する。そして、ＢＣ領域の後に各ＰＵ毎のアドレス空間を割り振り、さらに各ＰＵ内のアドレス空間にローカルメモリＬＭアドレス、及びシステム設定レジスタアドレスを割り振る。ＦＶ制御レジスタは、前記システム設定レジスタアドレス内の１アドレスを持ち、当該アドレスをアクセスすることでＦＶ制御モードを設定できる。なお図１６では、ＣＰＵ０、ＣＰＵ１、ＤＳＰ０、ＤＳＰ１、の順に先頭空間よりアドレスがマップされ、例えばＤＳＰ０のＦＶ制御レジスタを設定する際は、アドレス“ＤＳＰ０＿ＦＶＲＥＧ＿ＡＤＲＳ”をアクセスすることとなる。また、当該空間の各ＰＵ領域の後には共有メモリＳＭのアドレスを割り振る。
【００８２】
ＦＶ制御レジスタの設定は、コンパイラが決定したタスク管理用のＰＵがレジスタアクセス用のオブジェクトコードを実行し、ローカルバスＬＢＵＳ１９を介して制御先ＰＵのＦＶ制御レジスタをアクセスすることで行う。コンパイラが当該オブジェクトコードを生成する具体的な方法に関しては後述するが以下簡単に説明すると、コンパイラがタスクを複数ＰＵに割り当てる際、当該複数のＰＵをグループ化し、グループ内のタスクの起動や同期処理を行うタスク管理ＰＵを決定する。コンパイラは、当該管理ＰＵ上でＦＶ制御を行うコードを生成し、当該管理ＰＵは当該コードを実行することで、グループ内のＰＵのＦＶ制御を行う。なお、上記コンパイラは図示しない計算機上で実行されるものである。
【００８３】
図１７（ａ）にＦＶ制御レジスタを設定する例を示す。コンパイラがＣＰＵ０、ＤＳＰ０、ＤＳＰ１をグループ化し、ＣＰＵ０がタスク管理を行い、当該ＣＰＵ０がＤＳＰ０に対しＦＶ制御を行うとする。ＣＰＵ０は、コンパイラが生成したＦＶ設定を行うオブジェクトコードを実行することで、ＤＳＰ０のＦＶ制御モードを設定する。
【００８４】
図１７（ｂ）にＤＳＰ０のＦＶ制御モードを設定するオブジェクトコードの例を示す。本例ではＤＳＰ０内のＦＶ制御レジスタのアドレスを予め定義しておき、ＣＰＵ０内の汎用レジスタに当該ＦＶ制御レジスタのアドレス、及びＦＶ設定値を転送し、当該設定値を当該アドレスで指定されたレジスタに書き込むことで、設定を完了する。
【００８５】

なお、ＦＶ設定を行う方法として、タスク管理ＰＵがレジスタを直接アクセスするオブジェクトコードを実行するとしたが、例えば、図１７（ｃ）に示すように、ＯＳが各ＰＵのＦＶ動作モードを管理するとした場合では、ＯＳのＦＶ制御用ＡＰＩをコールすることで、ＯＳの管理下でＰＵのＦＶ制御モード設定を行うことができる。

【００８６】

なお、システム全体のＦＶ制御用レジスタを設け、このレジスタ内にすべてのＰＵ１０～１７のＦＶ制御モードを設定するレジスタを設けて、各レジスタにモードを設定すると、全ＰＵ１０～１７のＦＶモードが自動で設定される機構を持たせても良い。例えば、図８のＦＶ制御テーブル２６のように全ＰＵ１０～１７で共有するレジスタを設け、当該レジスタを更新することで、全ＰＵ１０～１７のＦＶ制御モードを変更することが可能となる。

【００８７】

また、当該システム全体の制御用レジスタに汎用処理ＰＵ群（ＣＰＵ）、専用処理ＰＵ群（ＤＳＰ、ＤＲＰ、ＢＭＰ）など、プロセッサの種類に対応して、当該種類ごとのＰＵについてＦＶモードを同期して設定する複数のＦＶ制御レジスタを設けるようにしても良い。この場合、図８のＦＶ制御テーブル２６にプロセッサの種類毎にＦＶ制御レジスタを設け、各ＣＰＵ、ＤＳＰ、ＤＲＰ、ＢＭＰ毎にＦＶ制御レジスタを共有することにより、一つのレジスタを変更することにより、プロセッサの種類毎にＦＶ制御モードを変更することができる。

【００８８】

＜ＢＣ領域＞

続いて、上記図１６で示したメモリマップの先頭領域に設けたブロードキャスト（ＢＣ）領域について、以下に説明する。当該ＢＣ領域は書き込み専用の領域であり、一ＰＵが当該ＢＣ領域の一アドレスに対しデータの書き込みを行うと、当該アドレスに予め対応させた全ＰＵの各ＬＭエントリに当該データを、ローカルバスＬＢＵＳ１９を介して同時に書き込む。これにより、全ＰＵで共有するデータを同時に各ＰＵのローカルメモリＬＭが持つこととなり、各ＰＵ１０～１７は共有メモリＳＭにアクセスしなくとも高速にＬＭ上の当該データにアクセスすることが可能となる。

【００８９】

また、ＦＶ制御などのシステム制御を行うに際しても、当該ブロードキャストを行うことで全ＰＵに対し一斉に制御情報を送信することが可能となる。また、ＰＵ範囲を指定するマスク情報を併せてブロードキャスト送信することで、ＰＵの範囲を限定してデータや制御情報を送信するマルチキャスト機能を実現することも可能である。この結果、例えばコンパイラが、あるタスクを並列処理する複数ＰＵで構成するＰＵグループを定義する場合、当該ＰＵグループ内で一斉にタスク処理を開始したり、ＦＶ制御を実行したりすることが可能となり、システム全体のスループットが向上する。

【００９０】

＜タスク処理時のＦＶ制御概念＞

次に、図１８を用いて各ＰＵ１０～１７におけるタスク処理時の電源電圧、動作周波数（ＦＶ）制御方法の概念について説明する。各ＰＵで実行するタスク（プログラム）は、後述するコンパイラによって入力プログラムから生成される。つまり、当該入力プログラムは、まず後述するコンパイルにより、当該プログラムの構造を解析することで、代入文のみからなるベーシックブロック（ＢＢ）、繰り返しブロック（ＲＢ）、サブルーチン（ＳＢ）といった粒度が大きなステートメント・ブロックをマクロタスク（ＭＴ）として分割する。本例では、３個のマクロタスクＭＴ３０１～３０３に分割されるとする。ここで、マクロタスクは、入力プログラム（ソースコード）を複数の粒度の単位ブロックに分割したものである。つまり、マクロタスクに分割することで、サブルーチン等の粗粒度タスク間の並列処理を利用するマクロデータフロー処理、ループレベルの並列処理である中粒度並列処理に、基本ブロック内部のステートメントレベルの並列性を利用する近細粒度並列処理とを階層的に組み合わせて並列処理を行うマルチグレイン並列処理を行う。また、

本実施形態では、プログラムの構成要素をマクロタスクとし、マクロタスクの構成要素をタスクとし、タスクをＰＵ１０～１７に割り当てるものとする。

【００９１】

続いて、当該マクロタスクＭＴの任意のＰＵにおける演算コスト等の特性情報を算出することで、当該マクロタスクＭＴをどのＰＵ上で実行するかを決定し、また当該マクロタスクＭＴ間のデータ依存性や制御依存性を解析することで、タスクの実行順序を決定する。

【００９２】

図１８（ａ）は、タスク間の並列実行依存性を示したマクロタスクグラフである。本グラフは、マクロタスクＭＴ１（３０１）とマクロタスクＭＴ２（３０２）は同時に実行できることを示し、さらにマクロタスクＭＴ３（３０３）は、マクロタスクＭＴ１（３０１）及びＭＴ２（３０２）の実行が終了後に、実行できることを示している。また本例では、マクロタスクＭＴ１はＰＵ１０（ＣＰＵ０）に、マクロタスクＭＴ２はＰＵ１２（ＢＭＰ０）１２に、マクロタスクＭＴ３はＰＵ１０（ＣＰＵ０）に配置される。

【００９３】

以上のようにスケジューリングされたマクロタスクを通常処理（電圧・周波数制御なし）したときの処理ガントチャートを図１８（ｂ）に示す。

【００９４】

図１８（ｂ）において、マクロタスクＭＴ１とＭＴ２は並列実行可能なため、ＭＴ１はＣＰＵ（ＣＰＵ０）にて（３０５）、ＭＴ２はＢＭＰ（ＢＭＰ０）にて（３０６）同時に処理が開始される。通常処理時はＣＰＵ、ＢＭＰ共に供給される電圧は通常のＶＤ、また動作周波数も通常のＦＣが供給されている。本例では、ＣＰＵにおけるマクロタスクＭＴ１の処理サイクル数は、ＢＭＰにおけるマクロタスクＭＴ２の処理サイクル数より小さいため、ＣＰＵにおけるマクロタスクＭＴ１の処理（３０５）が、ＢＭＰにおけるＭＴ２の処理（３０６）に先行して終了する。

【００９５】

ＣＰＵはマクロタスクＭＴ１の処理（３０５）を終了したため、次にマクロタスクＭＴ３を処理（３０７）することになるが、マクロタスクＭＴ間の依存関係からＢＭＰにおけるＭＴ２の処理（３０６）が終了するまでは、ＣＰＵにおいて次に処理すべきマクロタスクＭＴ３を実行することができない。そのため、ＣＰＵはＢＭＰでのマクロタスクＭＴ２の処理が終了するまでアイドル状態となる。当該アイドル状態においても、ＣＰＵに対しては通常の電源電圧ＶＤ及びクロックＦＣが供給されているため、余分な電力を消費することになる。

【００９６】

そこで以上を解決する一方法として、ＣＰＵにおいてマクロタスクＭＴ１を実行する際に、ＢＭＰがマクロタスクＭＴ２の処理に必要とする時間と、ＣＰＵで実行するマクロタスクＭＴ１の処理時間が等しくなるよう、ＣＰＵの動作周波数を通常時（ＦＵＬＬモード）よりも低減させるＬＯＷモード（図９の１／４ＦＣを供給するモード）で駆動する（３０８）。つまり、ＢＭＰを通常時のＦＶ制御モード（ＦＵＬＬ）で駆動する一方、ＣＰＵのＦＶ制御モードをＬＯＷモードとし、処理が早く終了する方のＰＵの動作周波数及び低減し、並列処理を行うＰＵ間でＦＶ制御モードが異なるようにＦＶ制御レジスタの設定を行う。本手法によるＦＶ制御適用時のガントチャートを図１８（ｃ）に示す。コンパイラは、ＣＰＵ（ＣＰＵ０）におけるマクロタスクＭＴ１の処理サイクル数及び、ＢＭＰ（ＢＭＰ０）におけるマクロタスクＭＴ２の処理サイクル数を見積もることで、双方の処理時間が等しくなるようＣＰＵの動作周波数を決定する。この結果、ＣＰＵの動作周波数が低減されるため、当該ＰＵに対する電源電圧ＶＬも低減することが可能となり、消費電力を最適化できる。

【００９７】

つまりこの例では、マクロタスクＭＴ１について、ＣＰＵのＦＶ制御モードを図９の「ＬＯＷモード」に設定し、並列的に処理が行われるＢＭＰのＦＶ制御モードを「ＦＵＬＬ

」に設定する。

【００９８】

また、上記図１８（ｂ）のＣＰＵアイドル状態を解決する別の方法として、ＣＰＵにおけるマクロタスクＭＴ１の処理（３０９）が終了した時点で、ＣＰＵの電源電圧及び動作クロックの供給を遮断し待機状態とする（３１０）。つまり、ＣＰＵがマクロタスクＭＴ１の処理を完了すると、ＣＰＵのＦＶ制御モードを図９の「ＯＦＦ」に設定する。そして、マクロタスクＭＴ３の開始時に、ＦＶ制御モードを「ＦＵＬＬ」に設定し、処理を再開する。本手法によるＦＶ制御適用時のガントチャートを図１８（ｄ）に示す。

【００９９】

図１８（ｄ）において、ＣＰＵによるマクロタスクＭＴ１の処理はＢＭＰにおけるマクロタスクＭＴ２（３０６）より先行して終了するが、当該終了時点でＣＰＵを待機状態（ＯＦＦ）とし、ＢＭＰがマクロタスクＭＴ２の処理（３０６）が終了した時点で再びＣＰＵを通常の電源電圧及び動作クロックを供給することで通常状態に復帰させ、マクロタスクＭＴ３の処理（３０７）を開始する。この結果、ＣＰＵをアイドルさせることなく動作が停止するため、消費電力を低減できる。

【０１００】

このように、プログラム（タスク）をコンパイルする時、ＰＵの構成に基づきコンパイラが予め事前の処理時間を見積もり、コンパイル時に処理順序を静的に決定できる部分に関しては事前に処理手順を決定し、ＰＵ毎にきめ細かく動作周波数制御及び電源管理を行うことで、マルチプロセッサシステムの性能を損なわずに電力を最適化することが可能となるのである。

【０１０１】

＜タスク処理時（リアルタイム制約時）のＦＶ制御概念＞

以上では、マクロタスク（ＭＴ）の実行条件が他のマクロタスクＭＴに依存する場合のＦＶ制御方法を説明したが、他にもタスクがある一定時間内に処理すべき制約を持つ、すなわち図１９（ａ）で図示するような、当該タスクの処理期限（許容時間）が決定されているリアルタイム処理タスクを対象としたＦＶ制御方法も考えることができる。

【０１０２】

図１９（ａ）で例示したマクロタスク（ＭＴ１）３１１では、「ＦＵＬＬ」モード、つまり通常の電源電圧及びクロック周波数で動作するＣＰＵで処理した場合、処理期限（Ｄｅａｄｌｉｎｅ）よりも先行して処理が終了する。この場合、処理が終了した時点でＣＰＵはアイドル状態となるが、本来の処理期限に対して余裕があるため、処理期限制約内でＣＰＵの動作周波数を低減させる（３１２）ことが可能となる。

【０１０３】

本手法によるＦＶ制御適用時のガントチャートを図１９（ｂ）に示す。コンパイラは処理期限までに処理を完了可能な動作周波数を見積もり、図示のようにＣＰＵのＦＶ制御モードを「ＬＯＷ」モードに決定する。その結果、供給電圧ＶＬも低減することができ、消費電力を最適化できる。

【０１０４】

また、同様にマクロタスクＭＴ１の処理（３１３）が終了した時点でＣＰＵの電源及び動作クロックを遮断する（３１４）ことで、消費電力を低減するようにしてもよい。本手法によるＦＶ制御適用時のガントチャートを図１９（ｃ）に示す。この場合、ＣＰＵは「ＦＵＬＬ」モードでマクロタスクＭＴ１を処理し、処理期限前に処理を完了することになるが、マクロタスクＭＴ１の完了時に動作モードを「ＯＦＦ」とすることで、無駄な電力消費を回避することができる。

【０１０５】

＜コンパイラの処理フロー＞

次に前述したマルチプロセッサ・アーキテクチャ上で、プロセッサユニット（ＰＵ）の特性に合わせたスケジューリングと電圧・周波数の動的変化による消費電力の最適制御を行うコードを生成するコンパイル方法と、当該手法を実装したコンパイラの処理について

以下順を追って説明していく。図２０に本手法を実装したコンパイラ４０の処理フローを示す。

【０１０６】

＜マクロタスクの生成＞

ＣやＦｏｒｔｒａｎ等の高級言語で記述された逐次構造の入力プログラム４００は、まず、当該プログラム構造を解析することで、繰り返しブロック（ＲＢ：Ｒｅｐｅｔｉｔｉｏｎ Ｂｌｏｃｋ）、サブルーチン（ＳＢ：Ｓｕｂ Ｒｏｕｔｉｎｅ）、擬似代入文ブロック（ＢＰＡ：Ｂｌｏｃｋ ｏｆ Ｐｓｅｕｄｏ Ａｓｓｉｇｎｍｅｎｔ ｓｔａｔｅｍｅｎｔｓ）の３種類の粒度が大きなマクロタスク（ＭＴ）に分割し、各マクロタスクを生成する（４０１）。ＲＢはループブロックで各階層での最も外側のループであり、ＢＰＡはスケジューリングオーバーヘッドあるいは並列性を考慮し、代入文からなる複数の基本ブロックを融合あるいは分割したブロックである。図２１に、上記のような入力プログラム（ソースプログラム４００）の一例を示す。

【０１０７】

＜Ｄｉｒｅｃｔｉｖｅ指定＞

なお、当該ソースプログラム４００中には、予めＰＵの割り当てを記述することが可能であり、例えばあるサブルーチンをＤＳＰに、またあるサブルーチンをＣＰＵに割り当てることを明示的に指定することも可能である。その結果、当該ＤＳＰサブルーチンはコンパイラ４０によりさらに並列性の解析が行われ、例えばＤＳＰが４個ある場合、当該４個のＤＳＰに対してコンパイラ４０は並列化スケジューリングを実施する。

【０１０８】

＜データ依存・制御フロー解析＞

続いて、分割生成された当該マクロタスク間の制御フロー及びデータ依存性を解析し、マクロタスクＭＴの実行順序関係を抽出する（４０２）。図２１の入力プログラム４００は、逐次的に記述されているため、通常のコンパイラ４０によって生成される実行コードは、当該プログラムの構造と同様に逐次的な順序で実行されるが、マクロタスクＭＴ間で見ると必ずしも記述された順序で実行する必要がないことが多い。

【０１０９】

つまり、マクロタスクＭＴ間において、制御またはデータ参照の依存性がない場合、特にマルチプロセッサシステムにおいては、複数のＰＵに複数のマクロタスクＭＴを配置して同時に、または順序を変更して、全体の実行時間が短くなるようスケジューリングすることが重要となる。

【０１１０】

このようなスケジューリングを行うためには、ＭＴ間の並列性を解析する必要がある。そこで、この解析に向けた準備として、データ依存・制御フロー解析処理４０２により、マクロタスクＭＴ間の実行順序関係を抽出する。

【０１１１】

＜ループレベル並列性解析＞

続いて、マクロタスクＭＴ内の中粒度レベルの並列性解析として、ループレベルの並列化を行う（４０３）。ループレベル並列化４０３では、ループの繰り返し（イタレーション）単位間のデータ依存性を解析して、各イタレーションが独立に処理できるかを判断し、可能な場合は各イタレーションを複数のＰＵに割り当てて並列処理を行う。

【０１１２】

また、単一のループを複数のループに分割して並列性を高めたり、データのコピーや配列変数の拡張により、各ループ間のデータ依存性を削除することで並列化したり、また複数のループを単一のループに融合することでループ制御に必要なオーバーヘッドを軽減したり、といった様々な手法によりループの並列化を実現する。

【０１１３】

＜処理コスト解析＞

次に、上記生成されたマクロタスクＭＴを各ＰＵで実行した際に必要となる処理サイク

ルを見積もる、処理コスト解析を行う（４０４）。処理コスト（演算コスト）の見積もり方法としては、例えばＣＰＵなどに関しては、乗算や加算など命令レベルで必要とするサイクル数を処理コストテーブル４２０にプロファイリング情報として保持しておき、当該テーブル４２０を参照することで、マクロタスクＭＴを当該ＰＵで実行する際の逐次処理サイクル数を見積もることができる。

【０１１４】

また、ＤＲＰやＤＳＰなど、プログラム中から表現される命令レベルでのサイクル数の見積もりが困難な場合は、一度当該ブロックのプログラムをＤＲＰやＤＳＰ用の実行コードを生成する夫々のローカルコンパイラを呼び出し、変換した実行コードを元にプロファイリングを行い、処理コストを求める（４１３）。なお、ローカルコンパイラは、ＰＵの種類に応じて予め設定され、例えば、ＤＳＰであればＤＳＰのローカルコンパイラプロファイリング情報を、ＤＲＰであればＤＲＰのローカルコンパイラプロファイリング情報を用いる。

【０１１５】

また、例えば分岐を含む場合、あるいはループのサイズや配列のサイズが、マクロタスクＭＴの実行前では定まらないような場合においは、ローカルなコンパイラで一度プロファイリングすることにより、精度を高めた処理コストの算出を行うこともできる。なお、本プロファイリングを行わずマクロタスクＭＴが分岐を含む場合は、分岐確率を５０％としてコスト算出を行う。また、同じくプロファイリングを行わずループや配列のサイズが定まらない場合は、例えばループを固定回数とし、また配列宣言時の最大サイズとする、等の方法を適用する。

【０１１６】

以上では処理コストを、処理サイクル数（時間）として定義したが、他にコストを電力として定義することで、電力が最小となるようなスケジューリングを行うことも可能である。例えば、処理サイクル数とマクロタスクＭＴを完了すべき処理期限から、設定可能なＦＶ制御モード（周波数、動作電圧）のうち処理期限内で最小の消費電力となる動作モードを選択する。あるいは、ＰＵの消費電力は、

消費電力＝動作電圧^2×駆動周波数

と見積もることができるので、処理期限内で消費電力が最小となるＦＶ制御モードの組み合わせを選択し、スケジューリングすればよい。例えば、図１９（ｂ）のようにＬＯＷモードのみで、処理を実行する場合や、図１９（ｃ）のように複数のＦＶ制御モードを組み合わせる。あるいは、処理時間が最小かつ消費電力が最小となるＦＶ制御モードの組み合わせを選択することもできる。

【０１１７】

＜マクロタスク間並列性解析＝最早実行条件解析＞

コンパイラ４０はマクロタスクＭＴの処理コストが決定した後、データ依存・制御フロー解析処理４０２で抽出したマクロタスクＭＴ間の制御フローと、データ依存性を同時に解析した結果から、マクロタスクＭＴ間の並列性、つまり各マクロタスクＭＴの実行を最も早く実行してよい条件（最早実行条件）を決定する（４０５）。

【０１１８】

この最早実行条件をグラフで可視的に示したものがマクロタスクグラフ（ＭＴＧ）である。図２１の入力プログラムを解析し生成されたＭＴＧを、図２２に示す。なお、マクロタスク間並列性解析結果は、マクロタスクグラフテーブル４２１として外部記憶装置に保持され、後段のコンパイル処理で使用される。なお、この外部記憶装置は、コンパイラ４０を実行する図示しない計算機のものを指す。

【０１１９】

＜マクロタスクグラフの説明＞

以下、図２２を参照しながら、マクロタスクグラフＭＴＧについて説明する。本グラフ中の各ノードはマクロタスクＭＴを示し、ノード間の実線はマクロタスク間のデータ依存関係を、ノード間の破線はマクロタスク間の制御依存関係を、ノード内の小円が条件分岐

を表している。例えば、マクロタスクＭＴ１＿１（５０１）からＭＴ１＿２（５０２）及びＭＴ１＿３（５０３）に対して実線が伸びているが、これは粒度の大きいマクロタスクＭＴ１＿２及びＭＴ１＿３がＭＴ１＿１を実行した結果生じたデータを入力データとして用い、処理を実行しているという依存関係があることを示している。そのため、実行順序として、ＭＴ１＿２及びＭＴ１＿３はＭＴ１＿１の終了後、実行できることを示す。

【０１２０】

また、入力プログラム４００から求めたマクロタスクＭＴ１＿２（５０２）は、複数のループやサブルーチンで構成された粒度の大きいブロックなので、コンパイラ４０は当該マクロタスクＭＴをさらに複数のマクロタスクＭＴに階層的に分割する。よって、当該マクロタスクグラフＭＴＧではＭＴ１＿２中に、別階層でさらにマクロタスクグラフＭＴＧ１＿２を構成する。マクロタスクＭＴ１＿３（５０３）も同様に別階層のマクロタスクグラフＭＴＧＭＴＧ１＿３を構成する。

【０１２１】

マクロタスクＭＴ１＿２（５０２）内のマクロタスクグラフＭＴＧ１＿２（５１０）を見ると、タスクＭＴ１＿２＿１（５１１）からはタスクＭＴ１＿２＿２（５１２）、タスクＭＴ１＿２＿４（５１４）、タスクＭＴ１＿２＿５（５１５）に実線が延びているため、ＭＴ１＿２＿１（５１１）終了後、同時にこれら３つのタスク５１２、５１４、５１５を実行することが可能である。

【０１２２】

また、タスクＭＴ１＿２＿２（５１２）はさらにタスクＭＴ１＿２＿３（５１３）へ依存があるが、これはＭＴ１＿２＿２（５１２）のタスクが終了後、ＭＴ１＿２＿３（５１３）のタスクを実行すればよい。また、タスクＭＴ１＿２＿４（５１４）及びＭＴ１＿２＿５（５１５）からタスクＭＴ１＿２＿６（５１６）に対して実線が延びているため、タスクＭＴ１＿２＿４（５１４）及びタスクＭＴ１＿２＿５（５１５）双方の実行が終了した時点でＭＴ１＿２＿６（５１６）が実行できる。

【０１２３】

以上のように、マクロタスクＭＴ１＿２は実行前に並列順序関係は確定されており、あらかじめ固定的（静的）にタスクをスケジューリング可能である。

【０１２４】

続いて、マクロタスクＭＴ１＿３（５０３）内のマクロタスクグラフＭＴＧ（５２０）を見ると、マクロタスクＭＴ１＿３＿１（５２１）には小円が存在するが、これはタスクＭＴ１＿３＿１（５２１）が条件分岐を含むことを示している。小円からタスクＭＴ１＿３＿２（５２２）、ＭＴ１＿３＿５（５２５）へは矢印のある破線が伸びており、また制御依存のＯＲ条件を表す点線のアーク５２９が重なっているため、当該条件がタスクＭＴ１＿３＿２またはＭＴ＿１＿３＿５の何れかに分岐することを示している。

【０１２５】

また、タスクＭＴ１＿３＿３（５２３）へは矢印のない破線が延びており、タスクＭＴ１＿３＿３（５２２）への矢印のない当該破線と、タスクＭＴ１＿３＿２（５２３）に接続される破線上には、制御依存のＡＮＤ条件を表す実線のアーク５２８が重なっている。これは、もし当該条件で、矢印で指されているタスクＭＴ１＿３＿２の方向に分岐すると、同一の分岐に制御依存しているタスクＭＴ１＿３＿３が同時に実行可能なことを示している。なお、図中破線はタスクの実行が確定される制御依存関係と、データ依存しているタスクが実行されない場合の条件を表している。また、矢印がついた破線は、データ依存・制御フロー解析（４０２）で求めた制御フローと同一（つまり、オリジナル）であることを表す。

【０１２６】

なお、タスクＭＴ１＿３＿１（５２１）の条件分岐は、当該タスクＭＴ１＿３＿１を実行しないと分岐方向が確定しないため、実行時の状況に応じたスケジューリングを行う必要がある。当該条件分岐が確定し、タスクＭＴ１＿３＿２（５２２）及びタスクＭＴ１＿３＿３（５２３）方向に分岐した場合、タスクＭＴ１＿３＿２（５２２）及びタスクＭＴ

１＿３＿３（５２３）からタスクＭＴ１＿３＿４（５２４）にデータ依存関係を示す実線があるため、双方のタスクＭＴ（５２２、５２３）処理の実行が終了した時点で、タスクＭＴ１＿３＿４（５２４）の処理を実行できる。

【０１２７】

また、タスクＭＴ１＿３＿５（５２５）方向に分岐した場合、同様にデータ依存関係から当該タスク５２５が終了した時点で、タスクＭＴ１＿３＿６（５２６）及びタスクＭＴ１＿３＿７（５２７）の処理を実行できる。

【０１２８】

＜プロセッサグルーピング＞

次にコンパイラ４０は、生成されたマクロタスクグラフＭＴＧテーブル４２１を参照し、マクロタスクグラフの形状や並列性に応じた、またはユーザからの指定に応じたプロセッサのグループ化を行う（４０６）。

【０１２９】

つまり、マクロタスクグラフＭＴＧで表現された上位階層のマクロタスクＭＴ、例えばマクロタスクＭＴ１＿２（５０２）、ＭＴ１＿３（５０３）の形状、並列性を解析し、当該マクロタスクＭＴの処理に必要となるＰＵ１０～１７をグループ化し、当該グループに対してタスクＭＴを割り当てる。なお、１つのグループは、当該グループ内の特定用途プロセッサ（ＤＳＰ、ＤＲＰ、ＢＭＰ）を制御するための汎用プロセッサ（ＣＰＵ）を少なくとも一つ含む集合となる。具体的には、マクロタスクグラフＭＴＧの構成によりにコンパイル時に適切なプロセッサＰＵのグループ構成が判断できる場合、コンパイラ４０は特定用途プロセッサを含めたグルーピングを行う。

【０１３０】

また、マクロタスクグラフＭＴＧの構成によりコンパイル時にプロセッサグルーピングができない場合、つまり特殊用途プロセッサが使用すべきタスクが多階層に存在し、グルーピングが適切に行えない場合では、コンパイル時にコンパイラ４０、または実行時にスケジューラとしての役割を果たす汎用プロセッサが、タスクを特定用途プロセッサグループに対し割り当てる。この際、特定用途プロセッサへの負荷が大きい場合、汎用プロセッサで処理を代行する。なお、決定したグルーピング情報は、プロセッサグルーピングテーブル４２２として外部記憶装置に保持され、後段の処理で使用される。

【０１３１】

本実施形態では、図２２ のマクロタスクグラフＭＴＧ１（５００）においてマクロタスクＭＴ１＿２（５０２）とＭＴ１＿３（５０３）の処理コストが同等であり、これらマクロタスク内部のマクロタスクグラフＭＴＧ１＿２（５１０）、ＭＴＧ１＿３（５２０）の実行には汎用プロセッサによる特定用途プロセッサの制御が必要であるため、２個のプロセッサグループを定義する。

【０１３２】

次に、マクロタスクＭＴ１＿２、ＭＴ１＿３の各ＰＵに対する処理コストや並列性に応じ、各プロセッサグループ内で必要とされる特定用途プロセッサの構成を決定する。本実施形態においては、必要とされるプロセッサ構成がマクロタスクグラフＭＴＧ１上の利用可能なリソースで実現できるため、プロセッサグループを、ＣＰＵ０、ＤＳＰ０、ＤＲＰ０、ＤＲＰ１、及びＣＰＵ１、ＤＳＰ１、ＢＭＰ０、ＢＭＰ１とコンパイル時に決定する。

【０１３３】

＜スタティックスケジュールとダイナミックスケジュール＞

コンパイラ４０は次にスケジューリング方法として、スタティックスケジュールかダイナミックスケジュールかを判断する（４０７）。夫々のスケジューリング方法の詳細は後述するが、以下簡単に説明する。まず、もし、タスクＭＴに条件分岐がなく最早実行条件が予め決定できるＭＴフローであれば、前者のスタティックスケジュールを適用し、コンパイラ４０が予めタスクＭＴのスケジューリングを行い、同期コード及びＦＶ制御コード（スケジューリングコード）をタスク間に挿入する。

【０１３４】
また、もし、タスクＭＴに条件分岐がある、あるいはタスクＭＴの処理時間が実行時に変動するなど、コンパイル時に予測できないＭＴフローであれば、後者のダイナミックスケジュールを適用し、コンパイラ４０は、分岐などの状況に応じた制御を実行時に行うスケジューリングプログラムを生成する。コンパイラ４０がスケジューリングプログラム（スケジューリングコード）を生成する利点は、従来のマルチプロセッサのようにＯＳあるいはライブラリに祖粒度タスクの生成、スケジューリングを依頼すると、数千から数万クロックのオーバーヘッドが生じてしまう可能性があり、それを避けるためである。
【０１３５】
＜スタティックスケジューリング＞
まず、スタティックスケジューリングの処理フローについて説明する。スタティックスケジューリングでは、コンパイル時にタスクＭＴの割り当てと実行順序がすでに決定しているため、まず、マクロタスクスケジューリング処理４０８により、マクロタスクテーブル４２１及びプロセッサグルーピングテーブル４２２情報を参照し、ＰＵ間における実行タスク間の同期や他タスクの起動等を行う制御情報を生成し、当該制御情報の挿入箇所を決定する。
【０１３６】
また、さらに、タスクプログラムまたはタスクが必要とするデータを、他ＰＵのローカルメモリＬＭや共有メモリＳＭより当該ＰＵのローカルメモリＬＭにロードするといったデータ転送情報も併せて生成する。このような、データローカライゼーション手法により、ローカルメモリＬＭを有効に活用し、データの転送量を最小化する。以上の処理により生成されたスケジュール情報は、マクロタスクスケジュールテーブル４２３として、外部記憶装置に保持される。
【０１３７】
続いて、コンパイラ４０は、スタティックＦＶスケジューリング処理４０９を行う。本処理では、図１８で説明したタスク並列処理実行時のＦＶ制御概念に基づき、マクロタスクスケジューリング４０８で生成されたスケジュール情報（マクロタスクスケジュールテーブル）より、タスクの処理コスト及びスケジュール期限（処理期限）で決定される余裕度を判定し、当該余裕度に応じた動作周波数・供給電圧を決定するＰＵ動作モードを設定するため、ＦＶ制御情報を生成する。当該ＦＶ制御情報は、電源・ＦＶ制御スケジュールテーブル４２４として、外部記憶装置に保持される。なお、余裕度は、例えば、実行開始から実行完了までの処理時間と、処理期限までの時間から求まるものであり、各ＰＵ１０～１７の性能上の余裕の度合いを示す。例えば、図１９（ａ）のように、ＣＰＵ０の実行完了から処理期限までの時間が長い場合には余裕度が大きいと判定し、逆に実行完了から処理期限までの時間が短い場合には余裕度が小さいと判定できる。
【０１３８】
また、処理期限（許容時間）は、入力プログラムに記述したり、コンパイラ４０の処理時に図示しないコンソールから入力してもよい。
【０１３９】
＜ダイナミックスケジューリング＞
次に、ダイナミックスケジューリングの処理フローについて説明する。ダイナミックスケジューリング時は、タスクＭＴ内の条件分岐等の不確定要素によりコンパイル時にスケジューリング内容を決定することができない。このためコンパイラ４０は、タスクＭＴの処理結果に基づき、プログラム実行時に動的にスケジューリングを行うスケジューリングプログラムを生成する（４１０）。本プログラムは、処理結果に基づきタスクＭＴの起動や同期、及び当該ＭＴが必要とするデータのロードを行う。また、さらに図１８で説明したタスク並列処理実行時のＦＶ制御概念に基づき、ＰＵの動作周波数・供給電圧を決定する動作モード（ＦＶ制御モード）を設定する。生成したスケジューリングプログラムは、電源・ＦＶ制御スケジュールテーブル４２４として、外部記憶装置に保持される。
【０１４０】

＜スケジューリングコードの生成＞

コンパイラ４０は以上の処理により、入力プログラム４００のＰＵ１０～１７へのスケジューリングを完了する。これまでの処理で生成したスケジューリング情報は外部記憶装置にテーブル（４２１～４２４）として保持されている。コンパイラ４０は、当該テーブル（４２１～４２４）にアクセスし、タスクの実行順序やデータ転送情報、及びＦＶ制御情報を読み込み、入力プログラムに対してスタティックスケジュール部には制御コード（スケジューリングコード）を、またダイナミックスケジュール部にはスケジューリングプログラムを付加する（４１１）。制御コードは、例えば分散メモリ型並列計算機における並列プログラミング標準インタフェースであるＭＰＩ（Ｍｅｓｓａｇｅ Ｐａｓｓｉｎｇ Ｉｎｔｅｒｆａｃｅ）などを用いて生成する。当該コードが付加された入力プログラムは、さらに各ＰＵの種類毎に用意されたローカルコンパイラで処理することにより、当該ＰＵにおける実行バイナリコードに変換される（４１２）。なお、ローカルコンパイラは、上述のようにＣＰＵ、ＤＳＰ、ＤＲＰ、ＢＭＰのＰＵの種類毎に予め用意され、コンパイラ４０から各ＰＵの種類に応じたローカルコンパイラを呼び出して実行バイナリコードを生成するものである。

【０１４１】

＜スケジューリングの結果＞

本スケジューリング結果の例を図２３に示す。図２３は、ＭＴのスケジューリング結果をＰＵ毎に時間軸で示している。本例ではＣＰＵ０がタスクの起動や同期、ＦＶ制御などの管理を行うため、最初にＣＰＵ０上にコンパイラ４０が生成した制御コード５５０がロードされる。当該制御コード５５０は、各ＰＵのＦＶ制御を行うコード、及びＣＰＵ０においてマクロタスクＭＴ１＿１を起動するコードで表現されている。つまりＣＰＵ０は、マクロタスクＭＴ１＿１を実行するＣＰＵ０に対し動作モードをＦＵＬＬにするようＦＶ制御レジスタの値を“３”に設定する。また、他のＰＵに対しては処理すべきＭＴがないため、動作モードをＯＦＦにするようＦＶ制御レジスタの値を“０”に設定する。次に、ＣＰＵ０は自ＣＰＵ０にてマクロタスクＭＴ１＿１の処理を開始する。

【０１４２】

ＣＰＵ０でマクロタスクＭＴ１＿１処理終了後、コンパイラ４０が生成した制御コード５５１を実行し、ＦＶ制御モードの設定を行う。次に処理すべきタスクは、マクロタスクＭＴ１＿２及びＭＴ１＿３の下位層のマクロタスクグラフＭＴＧで指定されたタスクＭＴ１＿２＿１及びタスクＭＴ１＿３＿１であり、前者はＣＰＵ０にて、後者はＣＰＵ１にて実行される。

【０１４３】

従ってＣＰＵ０はタスクＭＴ１＿３＿１を処理するＣＰＵ１のＦＶ制御レジスタを通常モード（ＦＵＬＬ）に設定する。またＣＰＵ０に関しても、すでにＦＶ制御モードは通常モードであるが、引き続きＭＴ１＿２＿１を処理するため通常モード（ＦＵＬＬ）としておく。

【０１４４】

このとき、前記タスクＭＴ１＿２＿１及び前記タスクＭＴ１＿３＿１の処理中は、他のＰＵ上での処理タスクは存在しないため、当該他のＰＵは引き続き動作モードをＯＦＦとしておく。続いてＣＰＵ０は、自ＣＰＵ０においてタスクＭＴ１＿２＿１を、またＣＰＵ１においてタスクＭＴ１＿３＿１の処理を開始する。

【０１４５】

なお前述した通り、マクロタスクＭＴＧ１＿２は条件分岐を含まないためコンパイル時ですでにスケジューリングが行われており（スタティックスケジューリング）、ＣＰＵ０、ＤＳＰ０、ＤＲＰ０、ＤＰ１グループへタスクＭＴを割り振る。またマクロタスクＭＴＧ１＿３は条件分岐を含むため、実行時にスケジューリングを行うプログラムを付加し、実行結果によってＣＰＵ１、ＤＳＰ１、ＢＭＰ０、ＢＭＰ１のグループへタスクＭＴを割り振る。

【０１４６】

続いてマクロタスクグラフＭＴＧ１＿２のタスクＭＴグループのスケジューリングについて説明する。タスクＭＴ１＿２＿１の実行がＣＰＵ０において終了すると、ＣＰＵ０は制御コード５５２を実行し、タスクＭＴ１＿２＿２を処理するためにＤＳＰ０を通常動作モードＦＵＬＬとするよう、ＦＶ制御レジスタを“３”に設定する。またタスクＭＴ１＿２＿４及びＭＴ１＿２＿５はコンパイル時のＦＶスケジューリングによるタスクの余裕度判定によりＤＲＰ０、ＤＲＰ１を低電力動作モードＭＩＤＤＬＥとするよう、ＦＶ制御レジスタを“２”に設定する。また、ＣＰＵ０においては実行すべきＭＴはないが、スケジューラを実行しＤＳＰ０、ＤＲＰ０、及びＤＲＰ１のタスクの同期管理を行っているため、スケジューラはＣＰＵ０に対して低電力動作モードＬＯＷとするよう、ＦＶ制御レジスタを“１”に設定する。そして、タスクＭＴ１＿２＿２、タスクＭＴ１＿２＿４、及びタスクＭＴ１＿２＿５の処理を開始する。なお、タスクＭＴ１＿２＿２は当該ＭＴ処理終了後、同じくＤＳＰ０においてタスクＭＴ１＿２＿３を実行することとなるため、この時点でタスクＭＴ１＿２＿２終了後にタスクＭＴ１＿２＿３を起動するようタスクの起動予約（タスクレディキューへのタスクの登録）をしておく。

【０１４７】

次に前記タスクＭＴ１＿２＿４及びタスクＭＴ１＿２＿５の処理が双方とも終了したことをＣＰＵ０が判定すると、当該ＣＰＵ０は制御コード５５３を実行し、ＣＰＵ０はタスクＭＴ１＿２＿６を処理する自ＣＰＵ０の動作モードをＭＩＤＤＬＥに、またＤＲＰ０及びＤＲＰ１は処理すべきタスクが存在しないため、動作モードをＯＦＦに設定し、通常よりも電圧と動作周波数を低減した状態で前記タスクＭＴ１＿２＿６の処理を開始する。前記タスクＭＴ１＿２＿６の処理が終了すると、ＣＰＵ０は制御コード５５４を実行しタスクＭＴ１＿２＿３の処理終了を判定すると、ＭＴＧ１＿２グループ内のタスク処理が完了したため、ＣＰＵ０はＤＳＰ０の動作モードをＯＦＦに設定する。

【０１４８】

続いてＭＴＧ１＿３のタスクＭＴグループについて説明する。マクロタスクＭＴ１＿３は内部に分岐を持つため、タスクを実行しないと分岐方向がわからない。従って、実行時の状況によりタスク、データのロード、タスクの起動、同期及びＦＶ制御コードの管理を行うダイナミックスケジューラ５５５を、ＣＰＵ１において実行する。当該スケジューラ５５５は、まずタスクＭＴ１＿３＿１を実行するためＣＰＵ１の動作モードをＦＵＬＬに設定し、タスクＭＴ１＿３＿１を起動する。タスクＭＴ１＿３＿１実行終了後、スケジューラ５５６は分岐方向を判定し、次に起動するタスクを判断する。

【０１４９】

図２３に示したスケジューリングでは、タスクＭＴ１＿３＿２がＤＳＰ１にて、タスクＭＴ１＿３＿３がＢＭＰ０にて処理される。当該処理の起動前に、スケジューラ５５６はＦＶスケジューリング結果に従いＤＳＰ１の動作モードをＦＵＬＬに、またＢＭＰ０の動作モードをＬＯＷに設定する。ＣＰＵ１においては、実行すべきタスクＭＴはないが、スケジューラ５５６を実行しＤＳＰ１及びＢＭＰ０のタスク管理を行っているため、スケジューラ５５６はＣＰＵ１の動作モードを低電力モードＬＯＷに設定する。

【０１５０】

次に、ＣＰＵ０上のダイナミックスケジューラ５５７は、前記タスクＭＴ１＿３＿２及びタスクＭＴ１＿３＿３の双方の処理終了を判定すると、次にタスクＭＴ１＿３＿４を処理する自ＣＰＵ０の動作モード（ＦＶ制御モード）をＦＵＬＬに、また実行すべきタスクが存在しないＤＳＰ１及びＢＭＰ０の動作モードをＯＦＦに設定する。そして、自ＣＰＵ０において、前記タスクＭＴ１＿３＿４を実行する。

【０１５１】

以上により、異種のＰＵ１０～１７で構成されるマルチプロセッサにおいて、マルチグレイン並列処理により、実行時間が最小となるよう入力プログラムを並列化しスケジューリングした後、各ＰＵ間で処理時間が均一となるよう、ＰＵ毎にきめ細かく動作周波数制御及び電源管理を行うことで、最小の実行時間で性能を損なわずに消費電力の低減を図ることが可能となるのである。

【０１５２】
＜コンパイラが生成する目的プログラム＞
コンパイラ４０により、以上で説明したようなスケジュールを行うスケジューリングコードを、入力プログラム４００に付加しＰＵ毎に出力した出力プログラム（目的コード）の一例を図２４及び図２５に示す。図２４、図２５は、上記図２３で示した処理を記述したもので、図２４はマクロタスクＭＴＧ１＿２を処理するプロセッサのグループとしてＣＰＵ０、ＤＳＰ０、ＤＲＰ０、ＤＲＰ１の各ＰＵ用の実行コードを示す。また、図２５は、マクロタスクＭＴＧ１＿３を処理するプロセッサのグループとしてＣＰＵ１、ＤＳＰ１、ＢＭＰ０、１の各ＰＵ用の実行コードを示す。コンパイラ４０の生成コードは擬似コードで表現されているが、前述したように実際はＭＰＩ等のインタフェース規格に従って表現される。

【０１５３】
＜まとめ＞
以上のように、本発明によれば、多種のプロセッサユニットＰＵを集積するマルチプロセッサシステムにおいて、当該ＰＵを効率よく動作させるプログラムの分割配置および制御コードを生成するコンパイラ４０により、当該プロセッサシステムの性能を最大限活用しつつ、最小限の処理時間内に低電力にて効率よく処理することが可能となる。また、さらにはソフトウェア開発者がプロセッサの構成を意識することなく、短時間で極めて効率の良いプログラム作成が可能となる。

【０１５４】
＜補足＞
なお、上記実施形態においては、一つのチップに複数のＰＵ１０～１７を備えたマルチプロセッサシステムに本発明を適用した例を示したが、図１に示したチップを複数備えた並列計算機に適用することも可能であり、上記と同様の作用効果を得ることができる。

【０１５５】
また、請求項１４、請求項２２、請求項３０において、前記プロセッサユニット毎に設定されたプロファイリング情報に基づいて前記単位ブロックを処理する処理サイクル数を求め、当該処理サイクルから前記演算サイクル時間を求めることを特徴とするマルチグレイン並列化コンパイラ。

【０１５６】
また、請求項１４、請求項２２、請求項３０において、前記演算サイクル時間は、前記プロセッサユニット毎に設定されたローカルコンパイラにより前記単位ブロックを実行コードに変換し、前記プロセッサユニット毎に設定されたプロファイリング情報に基づいて前記単位ブロックを処理する処理サイクル数を求め、当該処理サイクル数に基づいて前記演算サイクル時間を求めることを特徴とするマルチグレイン並列化コンパイラ。

【０１５７】
また、請求項１５、請求項２３、請求項３１において、前記プロセッサユニット毎に設定されたプロファイリング情報に基づいて前記単位ブロックを処理する処理サイクル数を求め、当該処理サイクルから前記電力を求めることを特徴とするマルチグレイン並列化コンパイラ。

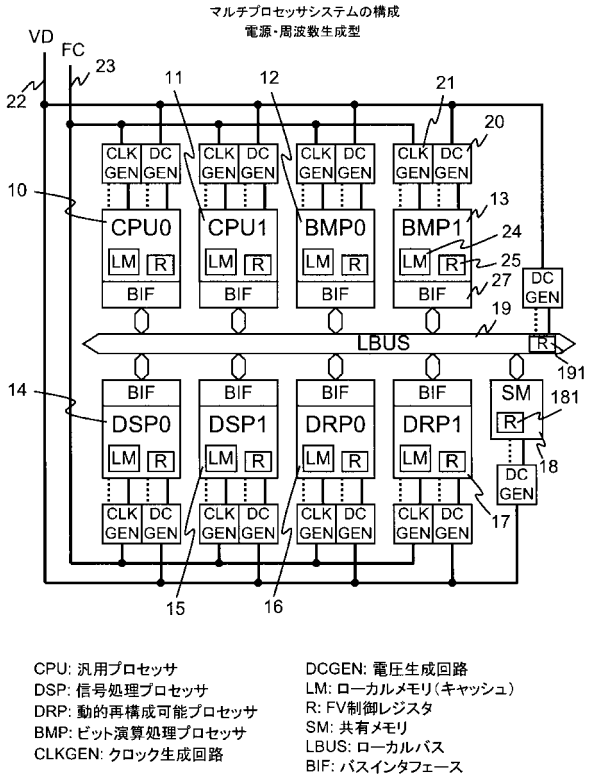
【０１５８】
また、請求項１５、請求項２３、請求項３１において、前記演算サイクル時間は、前記プロセッサユニット毎に設定されたローカルコンパイラにより前記単位ブロックを実行コードに変換し、前記プロセッサユニット毎に設定されたプロファイリング情報に基づいて前記単位ブロックを処理する処理サイクル数を求め、当該処理サイクル数に基づいて前記電力を求めることを特徴とするマルチグレイン並列化コンパイラ。

【０１５９】
また、請求項１３、請求項２１、請求項２９において、前記単位ブロックの並列性を抽出する処理は、前記抽出した単位ブロックをマクロタスクグラフテーブルに格納し、前記単位ブロックを処理するのに必要なプロセッサユニットをグループ化してプロセッサグル

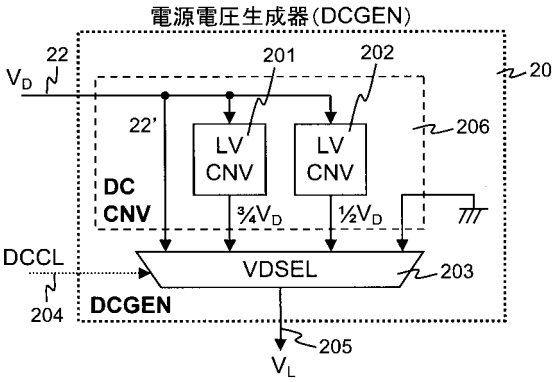
ーピングテーブルに格納する処理とを含み、
前記スケジューリングコードを生成する処理は、前記マクロタスクグラフテーブルとプロセッサグルーピングテーブル及びコスト情報を参照してプロセッサユニットの制御コードを生成する処理と、前記単位ブロックの実行順序に応じて前記制御コードを挿入する処理を含むことを特徴とするマルチグレイン並列化コンパイラ。
【０１６０】
また、請求項１３、請求項２１、請求項２９において、前記単位ブロックの並列性を抽出する処理は、前記抽出した単位ブロックをマクロタスクグラフテーブルに格納し、前記単位ブロックを処理するのに必要なプロセッサユニットをグループ化してプロセッサグルーピングテーブルに格納する処理とを含み、
前記スケジューリングコードを生成する処理は、前記マクロタスクグラフテーブルとプロセッサグルーピングテーブル及びコスト情報を参照して、動的に前記単位ブロックのスケジューリングを行うスケジューリングプログラムを生成する処理と、前記単位ブロックの実行順序に応じて前記スケジューリングプログラムを挿入する処理を含むことを特徴とするマルチグレイン並列化コンパイラ。
【産業上の利用可能性】
【０１６１】
以上のように、本発明は、多様なＰＵを集積するマルチプロセッサシステムにおいて、当該ＰＵを効率よく動作させるプログラムの分割配置および制御コードを生成するコンパイラにより、当該プロセッサシステムの性能を最大限活用しつつ、低電力にて効率よく処理することが可能となる。また、さらにはソフトウェア開発者がプロセッサの構成を意識することなく、短時間で効率良いプログラム作成が可能となる。その結果、高い演算性能を持ちかつ低電力に処理することが強く望まれる、カーナビゲーションシステムや携帯電話、情報家電向けのＬＳＩに本発明を適用することができ、高品質の動画像や音声処理、また画像認識や音声認識といった機能を実現することが可能となる。また、自動車における情報系、制御系システム向けＬＳＩに対しても適用することができ、自動運転や安全運転システム等を実現することが可能となる。またさらには、将来非常に高い演算性能を有しつつ低電力化が必須となる、スーパーコンピュータへの適用も可能である。
【図面の簡単な説明】
【０１６２】
【図１】本発明の一実施形態を示すマルチプロセッサシステムのブロック図。
【図２】電源電圧生成回路のブロック図。
【図３】降圧回路のブロック図。
【図４】供給電圧選択回路のブロック図。
【図５】クロック生成回路のブロック図。
【図６】クロックパルス選択器のブロック図。
【図７】マルチプロセッサシステムの他の例を示し、電源電圧生成回路とクロック生成回路を集約したマルチプロセッサシステムのブロック図。
【図８】マルチプロセッサシステムの他の例を示し、各プロセッサユニットのＦＶ制御レジスタを集約したＦＶ制御テーブルを有するマルチプロセッサシステムのブロック図。
【図９】ＦＶ制御によるプロセッサユニットの動作モードを示す説明図。
【図１０】ローカルメモリのＦＶ制御を示す説明図で、（ａ）はローカルメモリＬＭを複数バンクで構成してバンク毎にＦＶ制御を行う例を示し、（ｂ）はローカルメモリＬＭを複数のアドレス区間毎にＦＶ制御を行う例を示し、（ｃ）複数バンクに設定したアドレス区間毎にＦＶ制御を行う例を示す。
【図１１】ローカルバスＬＢＵＳをクロスバで構成した場合の電源制御の説明図。
【図１２】図１１に示したローカルバスＬＢＵＳの構成図。
【図１３】ＦＶ制御によるローカルメモリＬＭの動作モードを示す説明図。
【図１４】ＦＶ制御レジスタフォーマットを示す説明図で、（ａ）はローカルメモリＬＭを４バンクで構成した場合のレジスタフォーマットを示し、（ｂ）は同じくレジスタの設

定例を示し、（ｃ）は他のレジスタフォーマットを示す。
【図１５】図８に示したＦＶ設定テーブルのフォーマットの例を示す説明図。
【図１６】ＦＶ制御レジスタのマッピングを示すマルチプロセッサシステム全体のメモリマップ。
【図１７】ＦＶ制御レジスタアクセスの例を示す説明図で、（ａ）はＣＰＵ０がＤＳＰ０のＦＶ制御レジスタにアクセスする例を示し、（ｂ）はＣＰＵ０がＤＳＰ０のＦＶ制御レジスタにアクセスする場合のオブジェクトコードを示し、（ｃ）はＯＳのＡＰＩを用いてＦＶ制御レジスタを操作する場合のオブジェクトコードを示す。
【図１８】タスクを並列処理する場合のＦＶ制御の概念を示し、（ａ）はマクロタスクの並列実行依存性を示すマクロタスクグラフで、（ｂ）はＦＶ制御を行わない場合（ＦＵＬＬモード）のガントチャートを示し、（ｃ）はＣＰＵを継続して動作させる場合のＦＶ制御の例を示すガントチャートで、（ｄ）はＣＰＵを一時的に停止させる場合のＦＶ制御の例を示すガントチャート。
【図１９】リアルタイム制約時のＦＶ制御を示し、（ａ）は通常処理によるリアルタイム処理のガントチャート、（ｂ）はＣＰＵを継続して動作させる場合のＦＶ制御の例を示すガントチャートで、（ｄ）はＣＰＵを一時的に停止させる場合のＦＶ制御の例を示すガントチャート。
【図２０】コンパイラの処理を示すフローチャート。
【図２１】入力プログラムの一例を示すソースコード。
【図２２】コンパイラが生成したタスク間の依存関係図。
【図２３】タスク・ＦＶ制御スケジューリング結果を示す時系列的なチャート。
【図２４】コンパイラが生成するコードの例を示し、CPU0、 DSP0、 DRP0、 BRP1向けのコード。
【図２５】同じくコンパイラが生成するコードの例を示し、CPU1、 DSP1、 BMP0、 BMP1向けのコード。
【符号の説明】
【０１６３】
１０、１１　ＣＰＵ（プロセッサユニット）
１２、１３　ＢＭＰ（プロセッサユニット）
１４、１５　ＤＳＰ（プロセッサユニット）
１６、１７　ＤＲＰ（プロセッサユニット）
１８　共有メモリＳＭ
１９　ローカルバス
２０　電源電圧生成回路（ＤＣＧＥＮ）
２１　クロック生成回路（ＣＬＫＧＥＮ）
２４　ローカルメモリ（ＬＭ）
２５　システム制御レジスタ

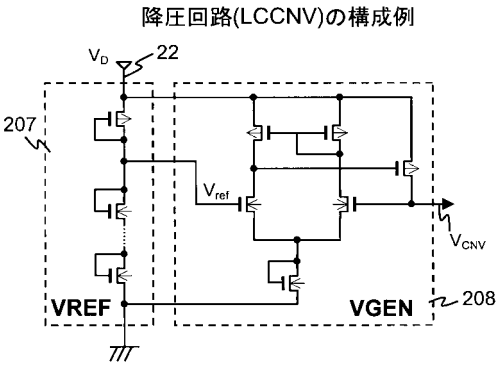
【図１】


マルチプロセッサシステムの構成
電源・周波数生成型
VD
FC
CPU0
CPU1
BMP0
BMP1
DSP0
DSP1
DRP0
DRP1
SM
LBUS
CLK GEN
DC GEN
LM
R
BIF
CPU: 汎用プロセッサ
DSP: 信号処理プロセッサ
DRP: 動的再構成可能プロセッサ
BMP: ビット演算処理プロセッサ
CLKGEN: クロック生成回路
DCGEN: 電圧生成回路
LM: ローカルメモリ（キャッシュ）
R: FV制御レジスタ
SM: 共有メモリ
LBUS: ローカルバス
BIF: バスインタフェース

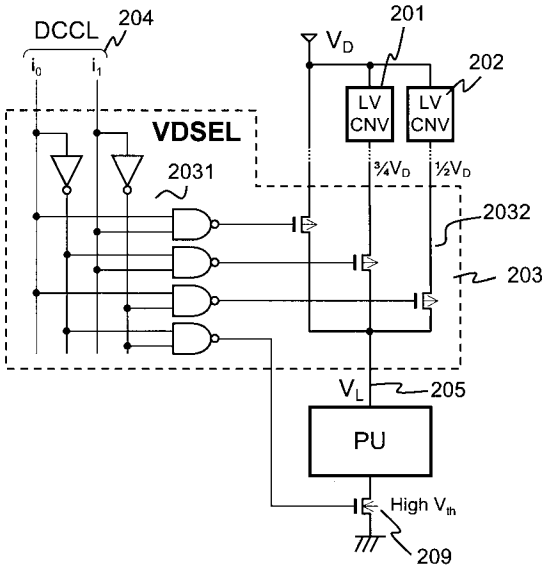
【図２】


電源電圧生成器(DCGEN)
LV CNV
DC CNV
¾VD
½VD
VDSEL
DCCL
DCGEN
VL

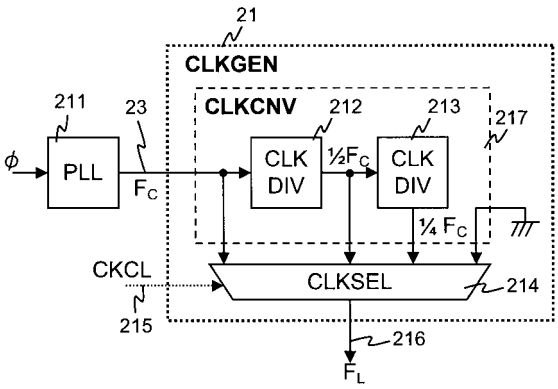
【図３】


降圧回路(LCCNV)の構成例
VD
Vref
VCNV
VREF
VGEN

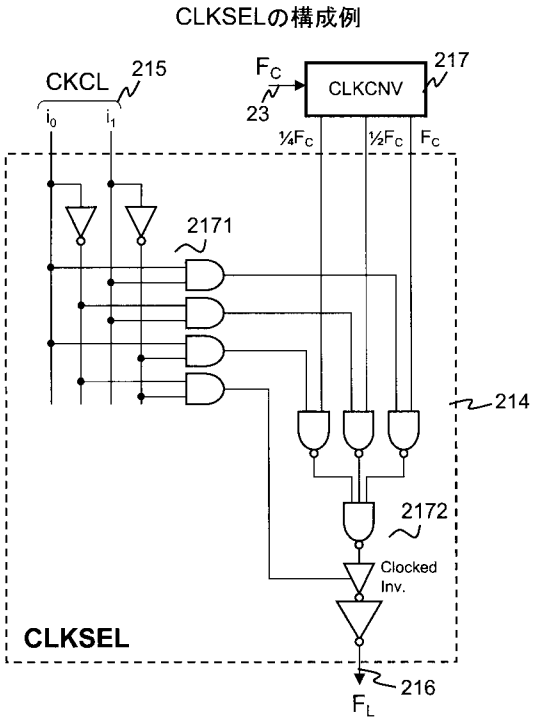
【図４】


VDSELの構成例
DCCL
VD
LV CNV
VDSEL
¾VD
½VD
PU
High Vth

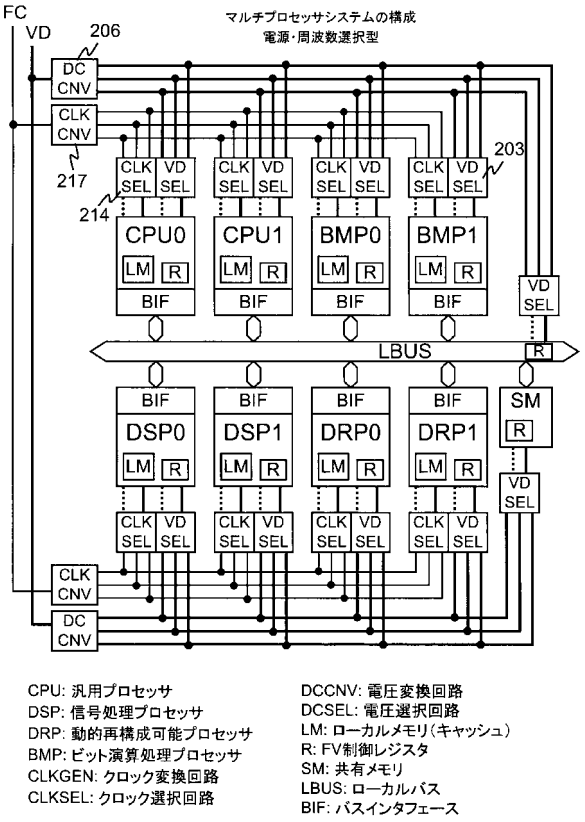
【図５】


周波数生成器(CLKGEN)の例
CLKGEN
CLKCNV
PLL
CLK DIV
½FC
¼FC
CKCL
CLKSEL
FL

【図６】



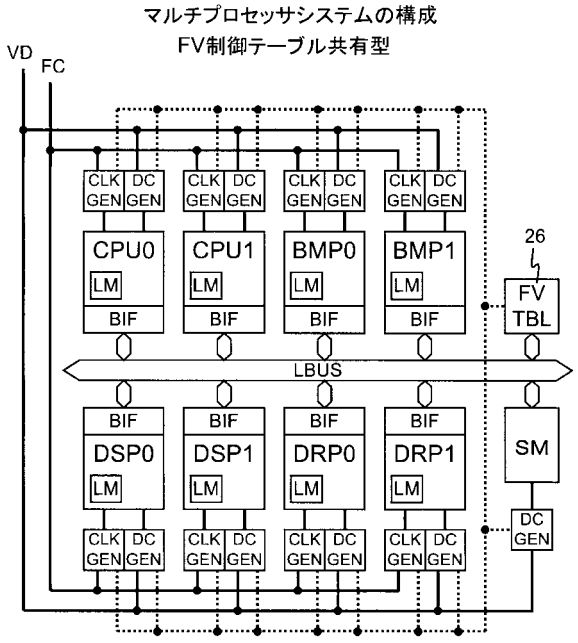
【図７】



CPU: 汎用プロセッサ
DSP: 信号処理プロセッサ
DRP: 動的再構成可能プロセッサ
BMP: ビット演算処理プロセッサ
CLKGEN: クロック変換回路
CLKSEL: クロック選択回路

DCCNV: 電圧変換回路
DCSEL: 電圧選択回路
LM: ローカルメモリ(キャッシュ)
R: FV制御レジスタ
SM: 共有メモリ
LBUS: ローカルバス
BIF: バスインタフェース

【図８】



CPU: 汎用プロセッサ
DSP: 信号処理プロセッサ
DRP: 動的再構成可能プロセッサ
BMP: ビット演算処理プロセッサ
CLKGEN: クロック生成回路

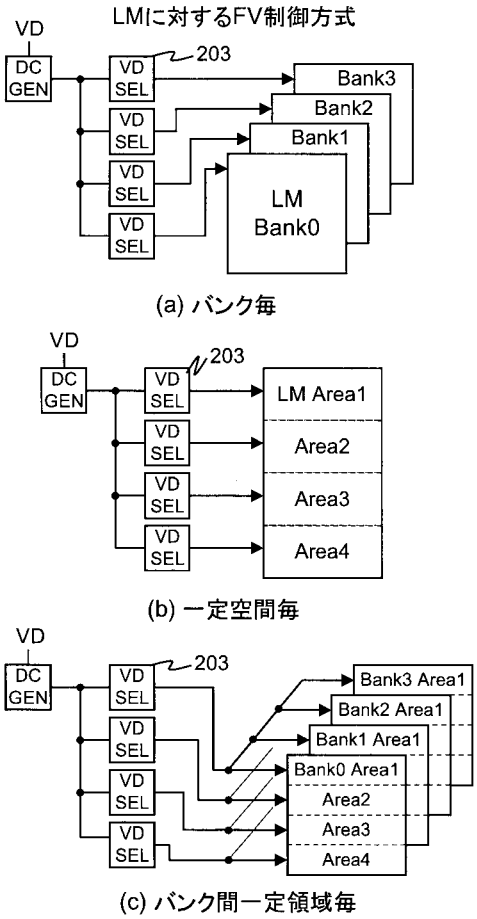
DCGEN: 電圧変換回路
LM: ローカルメモリ(キャッシュ)
FVTBL: FV制御テーブル
SM: 共有メモリ
LBUS: ローカルバス
BIF: バスインタフェース

【図９】

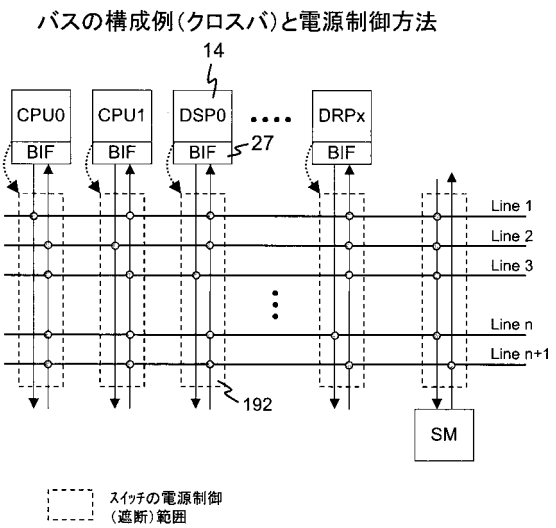
プロセッサ・ユニットの動作モード

モード名	FV制御レジスタ値	供給電圧(VL)	供給クロック周波数(FL)
OFF	00	0 (遮断)	0 (遮断)
LOW	01	½ VD	¼ FC
MIDDLE	10	¾ VD	½ FC
FULL	11	VD (システム電圧)	FC (システムクロック)

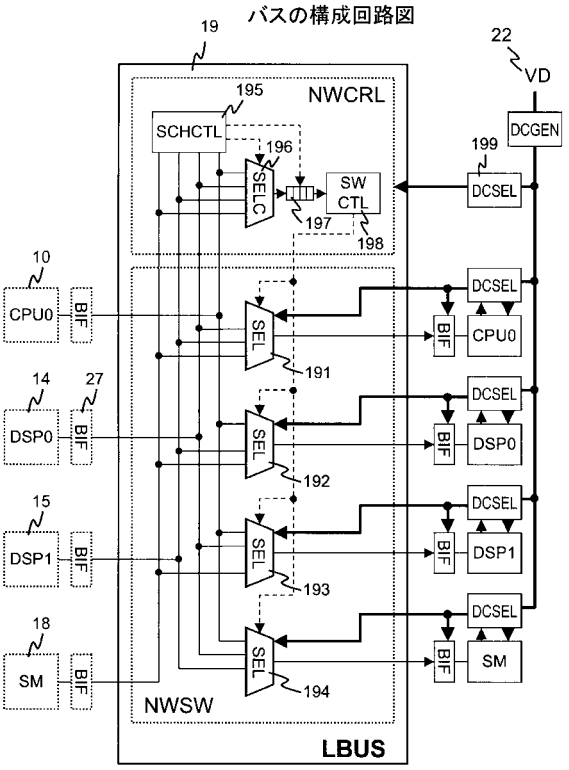
【図１０】



【図１１】



【図１２】



【図１３】

LMの動作モード

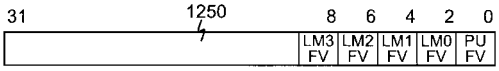
モード名	FV制御レジスタ値	供給電圧(VL)	供給クロック周波数(FL)
OFF 完全遮断	00	0 (遮断)	0 (遮断)
SLP データ保持	01	½ VD	0
FULL 通常	11	VD (システム電圧)	FC (システムクロック)

【図１４】

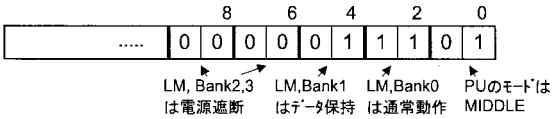
FV制御レジスタフォーマット

PUFV: PU本体のFVモード
LM*FV: LM(ﾊﾞﾝｸ番号*)のFVモード

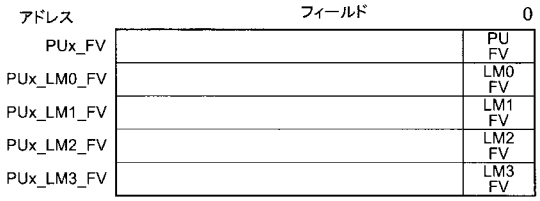
(a) FV制御レジスタフォーマット
（LMを4バンク構成とした例）



(b) FV制御レジスタ設定例



(c) 制御対象毎にアドレスを割り振ったFV制御レジスタフォーマットの例　（LMを4バンク構成とした例）



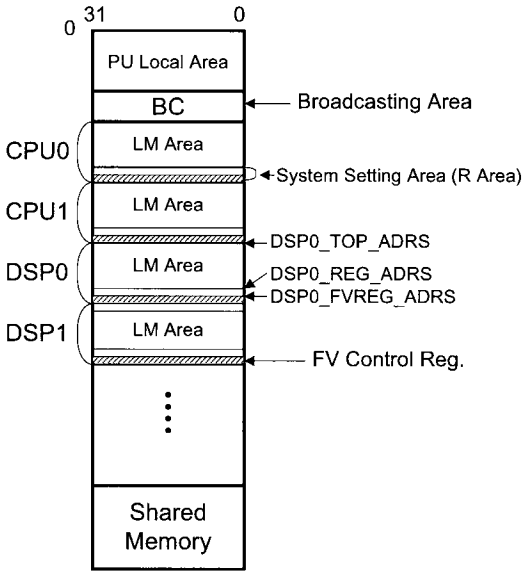
【図１５】

FVTBLのフォーマット

250	251	252	253	254	255	256
PUN	PUFV	LM0 FV	LM1 FV	LM2 FV	LM3 FV	BUS FV
0	01	11	01	00	00	11
⋮	⋮	⋮	⋮	⋮	⋮	⋮

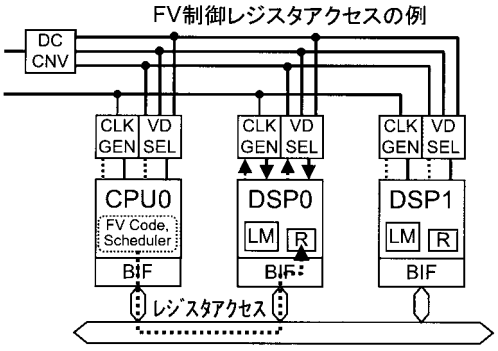
【図１６】

FV制御レジスタのマッピング



【図１７】

FV制御レジスタアクセスの例



(a) FV制御レジスタアクセス

(b) コンパイラが挿入する、レジスタセットを行うオブジェクトコードの例

```
; FV REGS ADDRESS SET

; Obtain a registers top address for each PU
CPU0_REG_ADRS       .EQU        CPU0_TOP_ADRS+SYSREG_OFFSET
DSP0_REG_ADRS       .EQU        DSP0_TOP_ADRS+SYSREG_OFFSET
DSP1_REG_ADRS       .EQU        DSP1_TOP_ADRS+SYSREG_OFFSET

; Obtain an FV register address for each PU
CPU0_FVREG_ADRS     .EQU        CPU0_REG_ADRS+FVREG_OFFSET
DSP0_FVREG_ADRS     .EQU        DSP0_REG_ADRS+FVREG_OFFSET
DSP1_FVREG_ADRS     .EQU        DSP1_REG_ADRS+FVREG_OFFSET

; Set the target FV register for DSP0
MOV.L       #DSP0_FVREG_ADRS,R0         ; Set target reg. address to R0
MOV.W       #H'0000001D,R1              ; Set fv reg. value to R1
MOV.W       R1,@R0                      ; Determine the mode setting
```

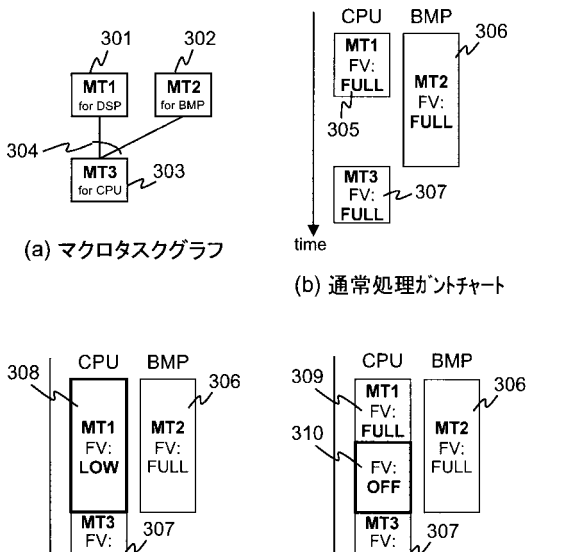
(c) OS APIを用いたレジスタセットの例

set_fv_regs(PUの種類, PUのFVﾓｰﾄﾞ, LMﾊﾞﾝｸ0のFVﾓｰﾄﾞ, ﾊﾞﾝｸ, ﾊﾞﾝｸ2, ﾊﾞﾝｸ3);

例） set_fv_regs(CPU0, MID, FULL, SLP, OFF, OFF);

【図１８】

タスク並列処理時のFV制御概念



(a) マクロタスクグラフ

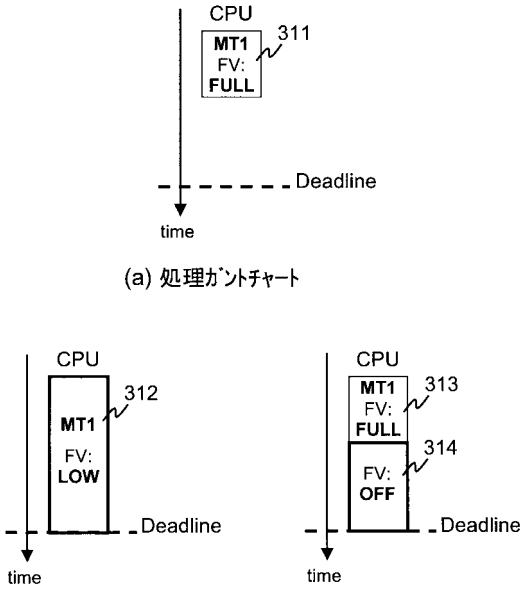
(b) 通常処理ｶﾞﾝﾄﾁｬｰﾄ

(c) FV制御時ｶﾞﾝﾄﾁｬｰﾄ(1)

(d) FV制御時ｶﾞﾝﾄﾁｬｰﾄ(2)

【図１９】

リアルタイム制約時のFV制御概念



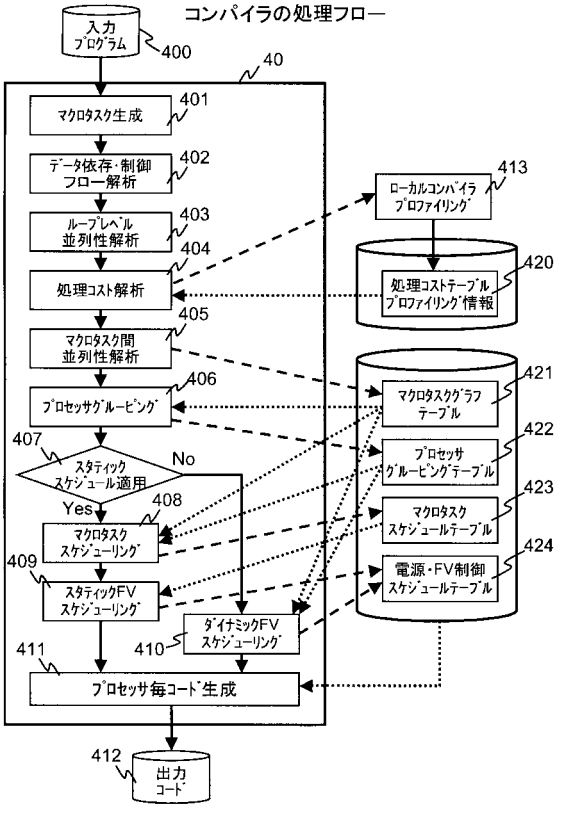
(a) 処理ｶﾞﾝﾄﾁｬｰﾄ

(b) FV制御時ｶﾞﾝﾄﾁｬｰﾄ(1)

(c) FV制御時ｶﾞﾝﾄﾁｬｰﾄ(2)

【図２０】

コンパイラの処理フロー



【図２１】

入力プログラム例

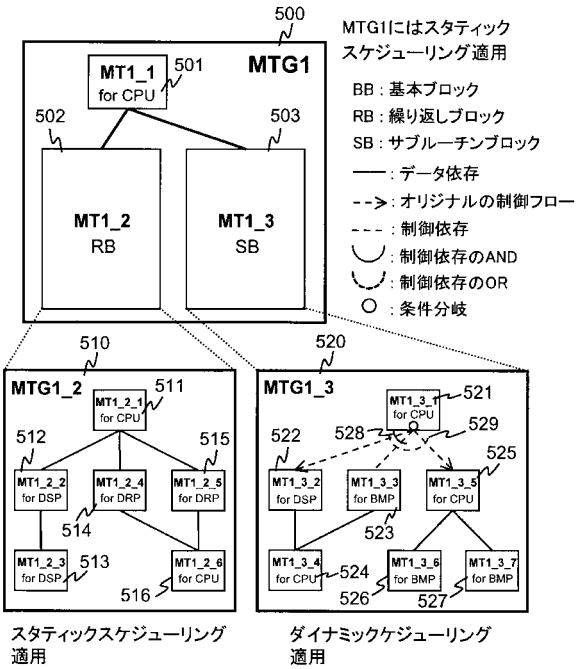
```
    /* MT1_1 */
    for (j = 0; j < m; j++)
       scanf( "%d %d" , &a[0][j], &x[j]);

    /* MT1_2 */
    for (i = 1; i < n; i++) {
       /* MT1_2_1 */
       for (j = 0; j < m; j++) a[i][j] = a[i-1][j];

       /* MT1_2_2 */
       func1(a, b);

       /* MT1_2_3 */
       func2(b, c);

       /* MT1_2_4 */
       for (j = 0; j < m; j++) d[i][j] = a[i][j] * 2;

       /* MT1_2_5 */
       for (j = 0; j < m; j++) e[i][j] = a[i][j] / 2;

       /* MT1_2_6 */
       func3(d, e, f);
    }

    /* MT1_3 */
    func4(x, y, z);
}

void func4(x, y, z)
{
    /* MT1_3_1 */
    t = (x(0) + x(m)) / 2
    if (t > g(m/2)) {
       /* MT1_3_2 */
       for (j = 0; j < m; j++) y[j] = x[j] * 2;

       /* MT1_3_3 */
       for (j = 0; j < m; j++) z[j] = x[j] + 1;

       /* MT1_3_4 */
       func4a(x, y, z);
    } else {
       /* MT1_3_5 */
       func4b(x, y, z);

       /* MT1_3_6 */
       for (j = 0; j < m; j++) y[j] = x[j] + 2;

       /* MT1_3_7 */
       for (j = 0; j < m; j++) z[j] = x[j] + 3;
    }
}
```

/* ～ */ は
コメント文

400

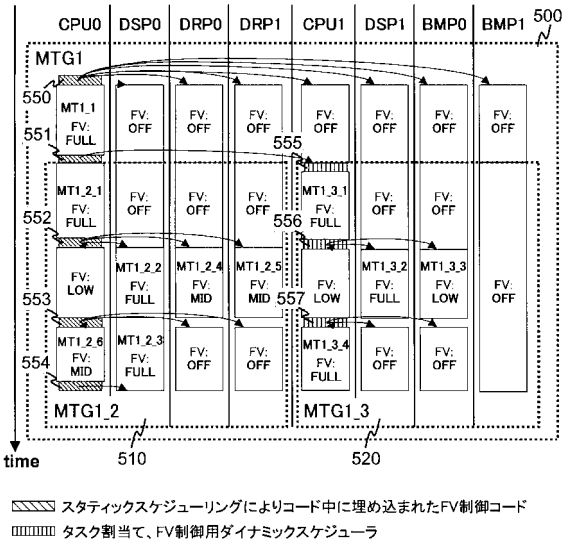
【図２２】

コンパイラが生成したタスク間依存関係図



【図２３】

タスク・FV制御スケジューリング結果



【図２４】

コンパイラが生成するコード例（1）

（CPU0, DSP0, DRP0, BRP1向けコード）

CPU0用コード

```
電源・FV制御指示文
( V_CPU0= full, V_DSP0= off,
 V_DSP1= off, V_DRP0= off,
 V_DRP1= off, V_BMP0= off,
 V_BMP1= off )

/* MT1_1 */
for( … ) scanf( … );

電源・FV制御指示文
( V_CPU1= full )

/* MT1_2 */
for( … ) {
  /* MT1_2_1 */
  for( … ) a[i][j] = …,

  電源・FV制御指示文
  ( V_CPU0= low, V_DSP0= full,
   V_DRP0= mid, V_DRP1= mid )

  タスクの起動コマンド発行
  実行終了の待機
  ( MT1_2_4, MT1_2_5
   実行終了まで待機 )

  電源・FV制御指示文
  ( V_CPU0= high, V_DRP0= off,
   V_DRP1= off )

  /* MT1_2_6 */
  func3( … );

  電源・FV制御指示文
  ( V_DSP0= off )

}
```

DSP0用コード

```
for( … ) {

  /* MT1_2_2 */
  func1( … );

  /* MT1_2_3 */
  func2( … );

  電源・FV制御待機

}
```

DRP0用コード

```
for( … ) {

  /* MT1_2_4 */
  for( … ) b[i][j] = …;

  電源・FV制御待機

}
```

BRP1用コード

```
for( … ) {

  /* MT1_2_5 */
  for( … ) b[i][j] = …;

  電源・FV制御待機

}
```

【図２５】

コンパイラが生成するコード例（2）

（CPU1, DSP1, BMP0, BMP1向けコード）

CPU1用コード

```
タスク割り当て、電源・FV
制御スケジューラ起動
( CPU1にMT1_3を割当 )

  /* MT1_3_1 */
  n = …;

タスク割り当て、電源・FV
制御スケジューラ起動
( DSP1にMT1_3_2を、
 BMP0にMT1_3_3を割当、
 FV制御コマンド発行 )

( V_CPU1= low, V_DSP1= full,
 V_BMP0= low )

タスクの起動コマンド発行
 実行終了の待機
 ( MT1_3_2, MT1_3_3
  実行終了まで待機 )

タスク割り当て、電源・FV
制御スケジューラ起動
( CPU1にMT1_3_4を割当、
 FV制御コマンド発行 )

( V_CPU1= full, V_DSP1= off,
 V_BMP0= off )

  /* MT1_3_4 */
  func4a( … );
```

DSP1用コード

```
/* MT1_3_2 */
for( … ) d[i] = … ;

スケジューラの待機
```

BMP0用コード

```
/* MT1_3_3 */
for( … ) e[i] = …;

スケジューラの待機
```

BMP1用コード

フロントページの続き

(72)発明者　伊藤　雅樹
東京都町田市本町田９６９番地７号
(72)発明者　鹿野　裕明
東京都国分寺市西恋ヶ窪３丁目８番地１号

審査官　久保　光宏

(56)参考文献　特開２００４－１９９１３９（ＪＰ，Ａ）
特開２００５－０６２９５６（ＪＰ，Ａ）
特開平０７－２８７６９９（ＪＰ，Ａ）
特開平０４－１９５６１９（ＪＰ，Ａ）
小出洋・他，「メタスケジューリング」，ｂｉｔ，日本，共立出版株式会社，２００１年　４月１日，Vol.33，No.4，pp.36-41，ISSN:0385-6984
笠原博徳，「最先端の自動並列化コンパイラ技術」，情報処理，日本，社団法人情報処理学会，２００３年　４月１５日，Vol.44，No.4，pp.384-392，ISSN:0447-8053

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ９／４５，
Ｇ０６Ｆ９／４６－９／５４，
Ｇ０６Ｆ１５／１６－１５／１７７，
Ｇ０６Ｆ１／０４－１／１４，
ＪＳＴＰｌｕｓ（ＪＤｒｅａｍ２），
ＣＳＤＢ（日本国特許庁）