オンチップ制御並列プロセッサMUSCATの提案

鳥居 淳[†1] 近藤真己[†2] 本村真人[†3]
池野晃久[†2] 小長谷明彦[†4] 西直樹[†1]

1チップ上に複数のPE（Processing Element）を集積することを前提にした制御並列処理アーキテクチャMUSCAT（MUlti-Stream Control ArchiTecture）を提案する．MUSCATはスレッド管理のハードウェア化とスレッド間のレジスタ継承をサポートし，並列実行オーバヘッドを大幅に低減している．また，制御/データスペキュレーションという2種類のスレッド投機実行をサポートしている．シミュレーションによる評価の結果，4PEモデルで1.5～3.0倍の性能向上を確認した．さらに，同一ハードウェア規模のスーパスカラプロセッサの性能を上回ることが明らかになった．

On-chip Control Parallel Multi-processor: MUSCAT

SUNAO TORII,[†1] MASAKI KONDO,[†2] MASATO MOTOMURA,[†3]
AKIHISA IKENO,[†2] AKIHIKO KONAGAYA[†4] and NAOKI NISHI[†1]

An on-chip control parallel multi-threaded architecture: MUSCAT has been proposed. MUSCAT supports hardware-controlled thread management and inter-thread register inheritance mechanisms to reduce multi-thread execution overhead. It also employs control and data speculative execution. Simulation results indicate that a four processing-element multiprocessor achieves one and a half to three times better performance than single processing-element model. It is demonstrates that MUSCAT performance is better than a super-scalar processor with the same hardware resources.

1. はじめに

マイクロプロセッサの高性能化技術として，スーパスカラプロセッサに代表される命令レベル並列処理が受け入れられた要因は，広範囲な領域での性能向上を，コード互換性を保ったまま実現した点が大きい．しかしながら，命令レベル並列処理は理論的な性能の限界に近づき，徐々にリソース投入量に対して十分な効果が得られなくなりつつある．

我々は，複数のプロセッサを1チップに集積するオンチップマルチプロセッサを今後の高性能化の有力な一手法としてとらえ，オンチップマルチプロセッサ向けの順序付きマルチスレッド実行モデルを提案，評価してきた[1)~3)]．その結果，本モデルによってスレッドスケジューリング，メモリ管理オーバヘッドを従来マルチスレッドモデルに比べ大幅に低減できることを確認した．一方，並列コードの生成を人手に頼らなくてはならないことや逐次性の強い問題では性能向上が難しいなどの課題が明らかになった．

これらのことから，広範囲な領域における性能向上と自動並列化コンパイラの実現を目指したアーキテクチャMUSCAT（MUlti-Stream Control ArchiTecture）を新たに提案する．MUSCATでは，PE間の距離が短く交信コストが低いというオンチップマルチプロセッサの利点を活かした制御並列処理（Control Parallel）という処理方式を導入する．

近年，制御並列アーキテクチャの研究は活発化しており，MUSCATの他にもMultiscalar[4)]，SPSM[5)]，Superthread[6)]，PEWs[7)]，TLS[8)]，マルチスーパースカラパイプライン[9)]，OCHA-Pro[10)]，SKY[11)]などが提案されている．これらのアーキテクチャは，スレッドの単位を基本ブロックレベルとしており，スレッドの制御スペキュレーションを導入している点で共通している．一方，スレッドの生成方法，並列性抽出を実

†1 NEC C&C メディア研究所
C&C Media Res. Labs., NEC Corporation
†2 NEC 情報システムズ
NEC Informatec Systems, Ltd.
†3 NEC シリコンシステム研究所
Silicon System Res. Labs., NEC Corporation
†4 北陸先端科学技術大学院大学知識科学研究科
School of Knowledge Science, Japan Advanced Institute of Science and Technology, Hokuriku

表1 制御並列アーキテクチャの比較（SPはスペキュレーションを示す）
Table 1 Control parallel architecture comparative table.

	並列性抽出	データ転送	メモリ正依存	メモリ逆依存
MUSCAT	コンパイラ	フォーク時レジスタ継承	同期/データSP	自動解消
Multiscalar	コンパイラ	任意時レジスタ継承	データSP	ARB自動解消
SPSM	コンパイラ	フォーク時/後にマージ	データSP	自動解消
Superthread	コンパイラ	メモリバッファ経由	同期	メモリバッファ
PEWs	ハードウェア	任意時レジスタキュー経由	データSP	キャッシュ内自動解消
TLS	コンパイラ	キャッシュ経由	データSP	キャッシュ内自動解消
Multi-superscalar pipeline	ハードウェア	親レジスタ参照可	同期	ハードウェア
OCHA-Pro	ハードウェア	キャッシュ経由	データSP	SAB自動解消
SKY	コンパイラ	任意時レジスタ継承	並列化しない	並列化しない

行時に行うかコンパイラによって静的に行うかという点，レジスタ間データ転送の有無，メモリ上のデータ依存の取扱いの差，並列化対象のループ限定の有無などが相違点としてあげられる（**表1**）．MUSCATは，並列処理におけるハードウェアとソフトウェアの分担を考慮し，フォーク1回モデル，フォーク時レジスタ継承，スレッドスペキュレーション，2種類のメモリデータ依存性制御，制御並列処理向き同期命令などの新規技術を採用し，幅広い並列化を可能とすることを目標にしている．

本論文では，MUSCATの基本アーキテクチャとその評価結果を中心に説明し，MUSCATが採用した制御並列アーキテクチャの有効性を実証する．まず，2章では，制御並列処理とMUSCATの基本アーキテクチャについて述べる．ここでは，制御並列アーキテクチャの選択肢とMUSCATがとった選択理由についてもあわせて言及する．さらに，MUSCATのコードの生成手法を3章で説明し，4章で評価を行う．5章で得られた知見のまとめと今後の課題について述べる．

2. MUSCATアーキテクチャ

2.1 制御並列処理

制御並列処理とは，プログラムを制御フローに基づいていくつかの基本ブロックもしくはマクロブロックに分割して，このブロック間で並列実行を行う処理方式である．たとえば，**図1**に示す制御フローの場合，基本ブロックAが実行することになった時点で，基本ブロックB，C，Dの実行いかんにかかわらず，基本ブロックEの実行は確定するので，別スレッドとして基本ブロックAと並列に実行する．もちろん，並列実行のためには，これらのスレッド間のデータ依存を解消する必要がある．

2.2 並列化手法

制御並列処理におけるスレッドの粒度は，最小で基本ブロックレベル程度となり，依存解析を行う範囲が

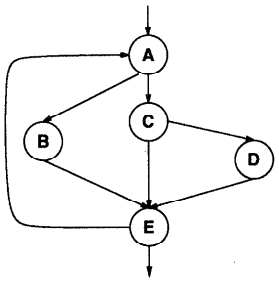


図1 基本ブロックの制御フローグラフ
Fig. 1 Control flow graph of basic blocks.

限定される．このことから，実行を行う前にコードを解析する静的自動並列化，実行時に逐次コードを解析して並列化する動的自動並列化が可能になると考えられる．

静的解析では，コンパイラによってスレッドの生成/消滅やスレッド間の依存関係を指定する．これらの指定は，拡張専用命令もしくは命令以外の情報提供手段をコードに組み込むことによって行われる．したがって，比較的粒度が大きく自由度の高い並列化が可能となるが，完全なコード互換性は保証できず，スレッド間の依存が動的にしか決定できないものや，動的に振舞いが変化するような場合には対処できない．

一方，動的解析の場合，分岐命令を基に動的予測や履歴から，逐次コードを実行時に複数のスレッドに分割する．そのため，バイナリ互換性が保たれるが，解析範囲が限定されるために性能向上が限定されることや解析用ハードウェアを付加する必要がある．

性能向上とハードウェア規模とを鑑みて，MUSCATでは，命令セットに直接スレッド制御命令を追加し，静的解析を用いた並列化を行うことにした．並列化は自動並列化コンパイラの実現を前提とし，スレッド生成時にはFORK命令，スレッド終了時にはTERM命令をコード上に付加する．

2.3 スレッドモデル

制御並列処理は，将来実行する，もしくは実行することが予想される基本ブロックを現在の実行と並列に処理する．したがって，スレッドは現在の実行しているスレッド（親スレッド）と，将来の実行を前倒しして実行するスレッド（子スレッド）に分類される．各スレッド間には先行/同一/後続の世代関係が生じ，最も先行するスレッドが親スレッドとなる．

このような実行形態では，先行スレッドから後続スレッドに対する制御やデータの依存関係は存在するが，後続スレッドから先行スレッドへの依存関係は存在しない．したがって，複数のスレッド間には逐次的な実行順序が定義されることになる．MUSCATでは，この逐次実行順序を活用し，さらにフォーク1回モデルを導入する（**図2**）．このフォーク1回モデルはスレッドは生存中にたかだか1回に限って後続スレッドを生成するというモデルである．

このスレッドモデルの利点は以下のとおりである．

- 同時存在スレッド数は（PE数+1）以内
- 各スレッドごとに世代を定義可能（同一世代スレッドは複数存在しない）
- デッドロックフリー
- フォーク先は隣接PEに限定可能
- 特別な同期機構が不要

フォーク1回モデルによって，スレッド管理が簡単化される．ただし，スレッド間で負荷が不均等な場合には，実効効率が低下することが予想される．この点については，基本ブロック単位という細粒度スレッドの並列化であり，スレッドの寿命が短いことで許容できると判断した．MUSCATでは，このフォーク1回モデルに基づいてスレッド管理のハードウェア化を実現し，スレッド管理オーバヘッドを削減することにした．

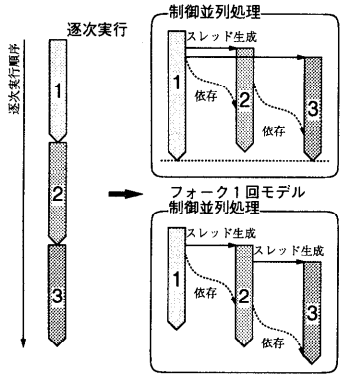


図2 制御並列処理とフォーク1回モデル
Fig. 2 Control parallel and fork once model.

2.4 制御スペキュレーション

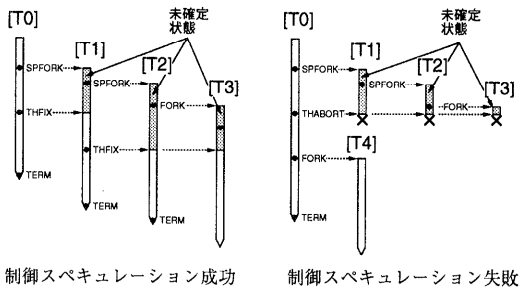
制御並列処理では，実行の確定した先行スレッドを並列に実行することを基本とするが，実際の問題では実行の確定するスレッドが十分に得られない場合も多い．そこで，MUSCATでは，実行する可能性の高いスレッドを実行確定前に投機的に実行する制御スペキュレーションをサポートする．投機状態のスレッドは，実行の取消しがハードウェア上可能である範囲内で仮実行を行う．制御スペキュレーションが失敗した場合は，すべての後続スレッドの実行を破棄する．MUSCATではこれらの指定を拡張命令によって行う．**図3**に示したように，投機的にスレッドを生成する場合にはSPFORKを，その後のスレッド確定/破棄にはTHFIX，THABORT命令を用いる．

2.5 レジスタ継承

並列処理において処理速度を向上させるためには，並列化にともなうオーバヘッドを上回る利得を得ることが必要である．このオーバヘッドは，複数のスレッドのスケジューリング等の管理コストとともに，スレッドに分割することによって生じるスレッド間の交信による同期，命令数の増加などによるものである．このスレッド間の交信コストを低減するためには，交信自身の高速化と交信による命令数の増加をおさえることが重要である．

新規に起動されたスレッドは実行に必要な初期データをレジスタ上に揃える必要が有る．この初期データはそのスレッドを生成する先行スレッド側のレジスタに揃っていることが多い．この性質を利用して，MUSCATでは，フォーク命令実行時点のレジスタセットの内容を，新たに生成される後続スレッドに継承するレジスタ継承機能をハードウェアでサポートすることにした（**図4**）．

MUSCATでは，先行スレッド側のフォーク時点のレジスタ内容が後続スレッドの初期状態となる．これにより，スレッド生成側は，継承データのメモリスト



制御スペキュレーション成功　制御スペキュレーション失敗

図3 制御スペキュレーション
Fig. 3 Control speculation.

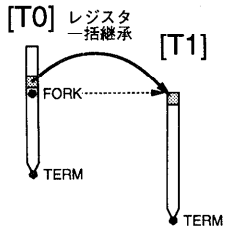


図4 フォーク時のレジスタ継承
Fig. 4 Register inheritance at fork time.

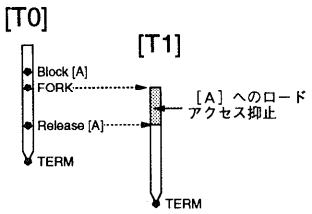


図5 ブロックによるデータ依存制御
Fig. 5 Data dependency control using block instruction.

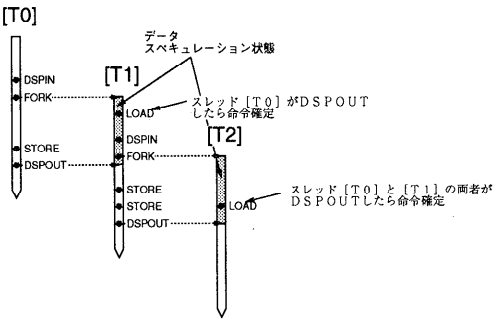


図6 データスペキュレーション
Fig. 6 Data speculation.

アを，被生成側では必要データのメモリロードを省くことが可能になり，スレッド間の交信コストが低減される．

さらに，フォーク後に先行/後続スレッド間でデータを転送する必要が生じる場合についても，レジスタ間で直接データを転送することも考えられる．しかしながら，フォーク時の継承と異なり，コード上に必要なデータ転送の指定や同期の指定等を付加する必要が生じる．MUSCATはon-chip multiprocessorで，共有オンチップデータキャッシュを持つことを仮定しており，レジスタ間直接転送命令を導入してそれらを行うことは，データキャッシュを介することに比してメリットが少ないと判断し，サポートしないことにした．

2.6 メモリ上のデータ依存制御

複数の基本ブロックを並列実行するためには，制御フロー上実行が確定している場合でも，基本ブロック間でデータの依存を解消する必要がある．MUSCATでは，スレッド間の世代定義により，データ依存も先行スレッドから後続スレッドへの単一方向のみとなる．したがって，先行スレッドがデータ生産者，後続スレッドがデータ消費者となるデータ依存が存在する場合が生じる．

MUSCATでは，フォーク命令実行時点で，自スレッドがフォーク後にストアするメモリアドレスが解析可能か否かによって，複数のデータ依存制御を使い分ける．解析可能な場合には，このメモリアドレスをフォークより前にBLOCK命令によって宣言し，後続スレッドが同一アドレスをアクセスするのを抑止する（**図5**）．アクセスの抑止の解除はフォーク後に先行スレッド側でRELEASE命令を実行することによって行う．

フォーク時点で解析不可能な場合には，フォーク命令挿入位置を解析可能になるまで遅らせるか，スレッド間のデータ依存がないと仮定して，投機的にロード/ストアを実行するデータスペキュレーションを用いる（**図6**）．データスペキュレーションはスレッドをフォークする前に，DSPIN命令を実行することによって指定する．この後に生成されたスレッドにおいてロードした同一アドレスに親スレッドが書き込みを行った場合は，スペキュレーション実行失敗となり，レジスタ値を復帰させたうえで衝突したロード命令以降の命令の再実行を行う．このため，状態が復帰できる範囲内でのみ，データスペキュレーションによる実行が可能になる．状態の復帰は，out-of-order実行を行うスーパスカラプロセッサの状態復帰機構を用いてレジスタ値を復元することによって行われる．

なお，MUSCATでは親スレッドがメモリロードを行う前に子スレッド側で同一アドレスに対してストアオペレーションが発生するような逆依存関係については，子スレッドの書き込みをすべてストアバッファに保持しておき，主記憶の書換えを親スレッドになるまで遅らせるハードウェアを用いて依存を解消し，順序関係を保証することにした．これによって，コンパイラは正依存のみ解析すればよいことになり，解析の負担と命令数の増加をおさえることが可能になる．

2.7 同　　期

MUSCATでは逐次順序関係を維持する範囲内で並列処理を行うので，スレッド間の同期は暗黙的に行われる．しかしながら，データの依存性などの関係で，先行/後続スレッド間で明示的に同期をとる方が性能上有利な場合もあるため，すべての先行スレッドが終了するまで実行を一時的に中断するための同期命令

表2 MUSCAT拡張命令セット一覧
Table 2 MUSCAT extended instruction set.

命令	意味
スレッド生成命令	
FORK	制御確定フォーク
SPFORK	制御投機フォーク フォーク先静的指定
スレッド制御命令	
THFIX	子スレッド実行確定
THABORT	子スレッド実行中止
TERM	自スレッド終了
データ依存制御命令	
BLOCK	子スレッド指定アドレスアクセス禁止
RELEASE	指定アドレスアクセス禁止解除
DSPIN	子スレッドデータスペキュレーション指定
DSPOUT	子スレッドデータスペキュレーション解除
同期命令	
CWAIT	自スレッド実行確定状態待ち合わせ
PWAIT	親スレッド終了待ち合わせ

(PWAIT)，スレッドの実行が確定するまで，自スレッドの実行を抑制するための同期命令（CWAIT）を用意することにした．

PWAITは，先行スレッドがアクセスする資源に自スレッドもアクセスすることが明白な場合に用いるとデータ依存制御命令を省くことが可能になり，CWAITは特定資源への書き込みなど取り消されては問題が生じるアクセスを取消しが生じないと保証できるところまで抑止することが可能になる．

2.8 拡張命令セット

以上で述べてきたスレッドモデルを実現するための拡張命令を，**表2**にまとめる．

3. コード生成手法

3.1 フォークポリシー

性能向上のためには，投機的実行を極力減らすことが重要である．このためには，条件分岐にできるだけ依存しないスレッドをフォークする必要がある．たとえば，**図7**(a)のような制御フローの場合には，条件分岐に依存しないようなパスをフォークする戦略をとる必要がある．

この場合は，基本ブロックA，Cの分岐いかんにかかわらず基本ブロックEは実行できるため，基本ブロックA，Cの分岐の失敗によっても基本ブロックEは実行を継続できる．しかしながら，基本ブロックA，B，C，Dで生成した値を子スレッドに引き継ぐためには，メモリオペレーションを介する必要があり，データ依存制御命令か同期命令が必要になる．したがって，スレッド粒度が小さいときには継承する値が確定後にレジスタ継承を用いてフォークを行うようにコードを生成した方が有利な場合もあり，コード生成上のトレードオフとなる．

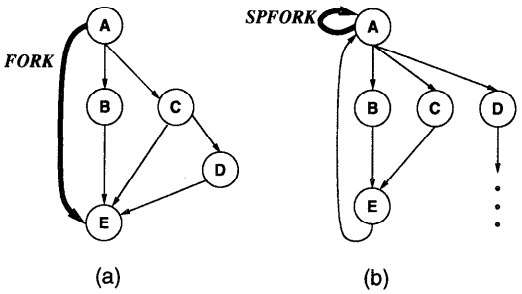


図7 フォーク箇所の決定ポリシー
Fig. 7 Fork instruction insertion policy.

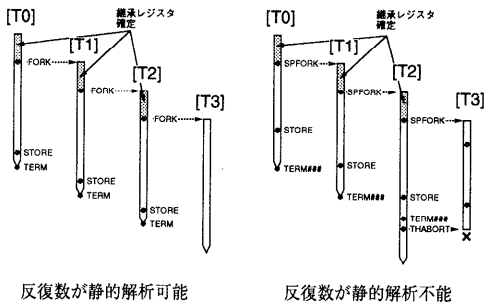


図8 ループのスレッド化
Fig. 8 Conversion loop to threads.

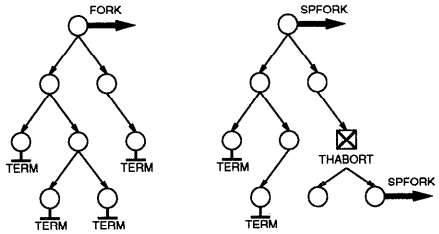
また，図7(b)は，ループのスレッド展開事例である．ループには並列性が潤沢に存在し，最も性能向上の期待できる処理である．ループの具体的な処理方法は**図8**に示すように，ループ間で継承すべきレジスタ値確定後にフォークを行う．静的には反復数が判明しない場合は，SPFORK命令を用い制御スペキュレーション状態のスレッドとしてループを構成する．

3.2 データ依存の取り扱い

フォーク命令の挿入位置候補が決まった後に，メモリ上のデータ依存の制御命令を挿入する．これは，フォークの位置からTERM命令およびTHABORT命令までのすべての制御パス上のメモリストアオペレーションで行われているアドレスについての依存を調べ，親スレッド側にデータ依存制御命令か子スレッド側に同期命令を挿入する．

3.3 フォーク1回モデルの保証

フォーク箇所決定後にフォーク1回モデルを保証するための措置を行う．これは，**図9**に示すように，親スレッドのフォーク命令後に，とりうるすべての制御フローを解析し，逐次コードフローが子スレッドにつながる最後の地点にスレッド終了命令を，子スレッドにつながらないことが判明した地点にスレッドアボート命令を挿入することによって行う．



FORK命令の場合　　SPFORK命令の場合

図9 フォーク1回モデルの保証
Fig. 9 Guarantee fork once model.

4. 評　価

4.1 評価モデル

MUSCATの性能評価を行うために，**図10**に示すシミュレーション用のハードウェアモデルを構築した．

演算器やロードストアユニットはオンチップである利点を活かしてPE間で共有構造とした．制御並列処理は命令レベル並列の並列性をあわせて利用できることから，各PEは最新のスーパスカラプロセッサに合わせて4命令/サイクルの命令デコード，完了を可能とし，PE内の分岐スペキュレーション実行をサポートした．また，スレッドの生成は隣接PEに対して行い1サイクルで実行可能とした．また，データスペキュレーション状態のスレッドは，分岐スペキュレーションと同様，最初のメモリオペレーション命令から命令ウィンドウサイズ相当の命令数の仮実行を行う．

評価は，トレースドリブンのパイプラインシミュレータを用い，**表3**に示すパラメータを使用した．

4.2 評価プログラム

idct, p_block, compress, eqntottの4種類のプログラムを用いて評価を行った．これらのコードの並列化は3章で示したコード生成手法に基づいて，自動並列化可能な範囲で，逐次コードを並列コードに書き換えることによって行った．

idct

2次元逆離散コサイン変換処理．反復間の依存関係は存在しないので，1反復を1スレッドとして並列化を施し，1次元ごとに終了時に同期をとった．今回の問題では，8回の反復なので並列度は8となった．

p_block

SPECベンチマークgccの一関数（propagate_block）で，基本ブロック間のレジスタの生存期間を調べる．基本ブロック間でのデータの依存性が強い．forループのイテレーションを1スレッドで実行している間に，子スレッドが依存のない部分まで先行し，同期で待ち合わせるという手法で並列化を行った．

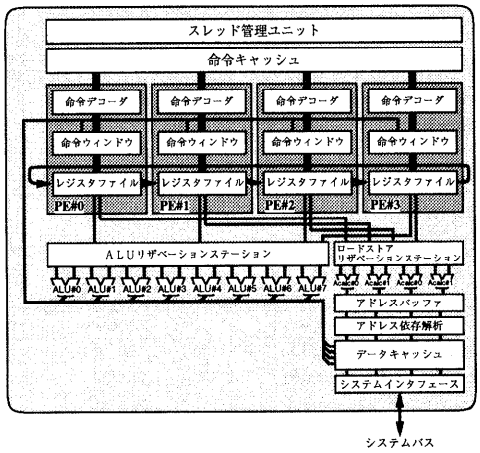


図10 MUSCATシミュレーションモデル（4PE構成）
Fig. 10 MUSCAT simulation model (4PE structure).

表3 シミュレーションパラメータ
Table 3 Simulation parameters.

項目	パラメータ
各PEの構造	
パイプライン	IF, ID, Issue/Reg, EX, WB, Graduate Issue/Reg～WB out-of-order実行
命令ウィンドウ	PE毎32命令（整数/ロードストア各16）
演算リソース	ALU 2 × PE数 L/Sパイプ1 × PE数
ロード/ストア	3サイクルレイテンシ
スーパスカラ度	4命令同時デコード/終了
分岐投機	4分岐まで仮実行
分岐履歴	2048 entry 4状態（PE間共有）
スレッド生成	IDステージ検出1サイクル
キャッシュ	
キャッシュ方式	命令/データ分離
キャッシュ容量	各32 Kbyte（64 byte × 512 entry）
マッピング方式	4 Way Set Associative LRU追出し
ミスペナルティ	20サイクル

compress

SPECベンチマークcompressから1 MBのファイル圧縮部を抽出．制御フローが動的に決定され，ポインタアクセスが多いためスペキュレーションを多用した並列化を行った．

eqntott

SPECベンチマークeqntott．処理時間のほとんどがクイックソートの比較関数の処理に費される．比較関数のループを中心に，スペキュレーションを用いて並列化した．

これらの並列化したコードの特性を**表4**に示す．平均命令数は実行された1スレッドあたりの動的な命令実行数で，スペキュレーション失敗にともなう取り

表4 評価プログラムの特性
Table 4 Characteristic of evaluation programs.

プログラム	平均命令数	制御 Spec.	データ Spec.
idct	140.1	なし	なし
p_block	99.1	あり	なし
compress	16.3	あり	あり
eqntott	11.1	あり	あり

表5 トレースデータの特性
Table 5 Characteristic of trace data.

	実行命令数	分岐	Load	Store
idct	32,838	0.2 (%)	15 (%)	10 (%)
p_block	95,941	11 (%)	25 (%)	12 (%)
compress	92,462,899	14 (%)	24 (%)	9 (%)
eqntott	130,723,959	27 (%)	22 (%)	1 (%)

消されたスレッドについては含んでいない．また，シミュレーションに用いた逐次実行トレースデータの動的特性を**表5**に示す．

4.3 評価結果

図11に1PEモデル逐次コード実行に対する2PE，4PE，8PEの各モデルの性能向上率，**図12**にデータキャッシュのヒット率，**図13**にPE稼働率☆，**図14**にスレッド生存率を示す．

idctは並列性が潤沢に得られるので，PE数に相当する性能向上が得られている．しかし，8PE時には性能向上が鈍っている．これはデータキャッシュの容量的な制約と，連想数がPE数を下回りPE間のアクセス競合によるスラッシングによってPE稼働率が低下したためと考えられる．なお，128 Kbyte，16 way構成という容量，連想度ともに1PEの4倍にしたデータキャッシュの場合には，逐次実行に対して約4.95倍の性能向上を実現できた．このときに，理想性能に達しない要因としては，並列化できない部分が残っていることや，共有資源のアクセスにともなう調停によるスレッド間の処理速度差によって，同期待ち時間が生じることなどがあげられる．

次にp_blockであるが，idct同様4PEまでは性能向上を果たしている．p_blockで行った並列化では，依存している場所に親スレッドを待ち合わせるため，同期箇所までに生成されるスレッド数によって並列性が制限される．図13に示された8PE時のPE稼働率の低下からも，このことが明らかになっている．したがって，スペキュレーションを多用すれば，並列度を高めることが可能にはなるが，この場合には，4PE以下の構成時の処理効率悪化が予想される．

☆ PEにスレッドが割り付けられている状態の割合．

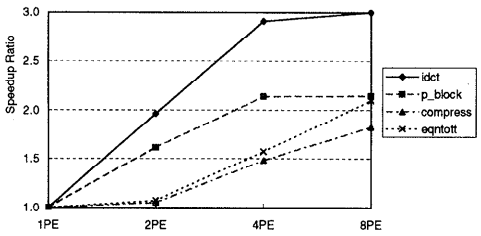


図11 MUSCATの性能向上率
Fig. 11 MUSCAT performance improvement.

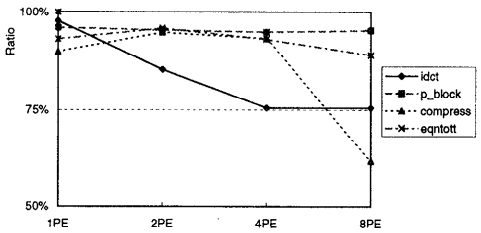


図12 データキャッシュのヒット率
Fig. 12 Data cache hit rate.

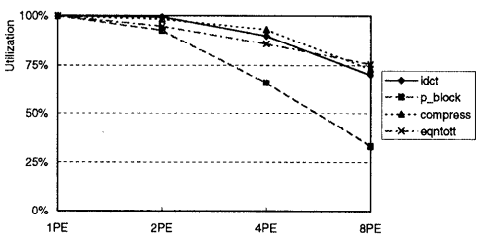


図13 PE稼働率
Fig. 13 PE utilization.

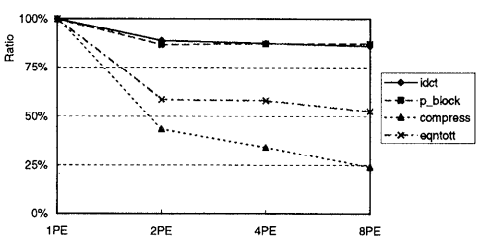


図14 スレッド生存率
Fig. 14 Thread survival ratio.

compress，eqntottではMUSCATのターゲットとする1スレッドの粒度が10命令程度の微細粒度スレッド並列化を行った．これらのコードでは，2PEでは性能がほとんど向上していないが，4PE時では1.5倍程度性能が向上した．スレッドの起動と終了が頻繁に行われると，**図15**に示すようなスレッド内の最後の命令が終了するまで，新規スレッドの命令フェッチが開始

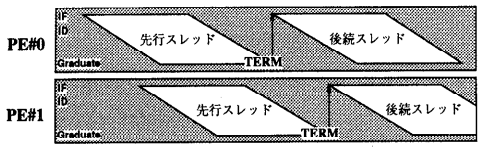


図15 スレッド起動/終了時のオーバヘッド
Fig. 15 Overhead of thread creation and termination.

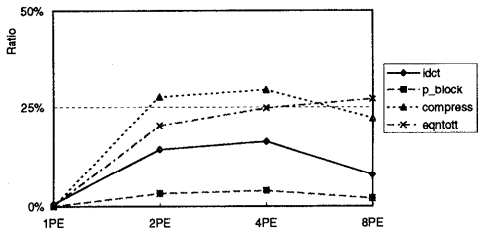


図16 デコーダ停止率
Fig. 16 Decoder stall ratio.

できないというオーバヘッドが顕在化する．**図16**に，全実行時間に対して，新規後続スレッドが存在しながら先行親スレッドの完全終了を待ち合わせてデコーダ停止している割合を示す．2PE実行時のcompressにおいては，これらの時間が全実行時間の25%にも及び，eqntottでも20%程度となっていることが分かる．このオーバヘッドが並列化による利得を上回るため，2PE実行時には性能が向上しなかった．この問題は，idct，p_blockでも生じるが，スレッドの粒度が大きく生成/終了回数が少ないために，この割合も低くなり性能にはほとんど影響を与えない．

またcompressでは8PEにした場合，4PEと比較して性能向上率が鈍っているが，この原因はデータの依存による並列性の減少と，データキャッシュのヒット率の低下，スレッドの生存率の減少によるものと考えられる．図14からもcompressではPE数が増えると完了するスレッドの割合が徐々に減少していることが分かる．compressでは，後続スレッドが多数生成された後に，スペキュレーションが失敗することが多くなるため，PE数が増加すると同時に存在するスレッド数が増加し，スレッドの生存率が低下する．このように，compressのような制御の依存性が大きい問題では，3章に示したコード生成手法を用いても，並列度は最高4程度でおさえられてしまう．

MUSCAT制御並列モデルと命令レベル並列処理を強化したプロセッサモデルとを比較するために，**表6**に示したスーパスカラプロセッサ強化モデルの性能比較を行った．このモデルは，MUSCATの2PE，4PE，8PEのケースと同じ機能ユニット数/命令同時デコード数を持ったスーパスカラモデルとして，命令同時デコード数に合わせて，4SS（4並列Super Scalar），8SS，16SS，32SSの4種類を用意した．このときの性能向上率を**図17**に，MUSCATモデルとの性能比較を**図18**に示す．

表6 スーパスカラ強化モデルのパラメータ
Table 6 Parameter of extended superscalar models.

	4SS	8SS	16SS	32SS
デコード/サイクル	4	8	16	32
ALU数	2	4	8	16
LS-Pipe数	1	2	4	8
命令ウィンドウサイズ	32	64	128	256
分岐Speculation	4	8	16	32

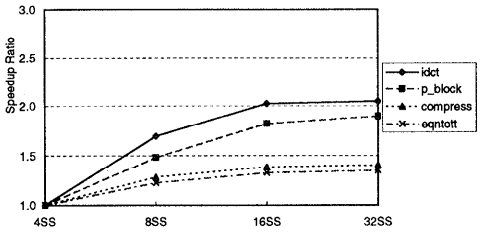


図17 スーパスカラ強化モデルの性能向上率
Fig. 17 Extended superscalar models performance improvement.

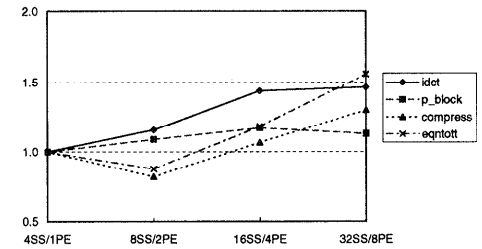


図18 MUSCATとスーパスカラ強化モデルの性能向上比較
Fig. 18 Performance comparison MUSCAT with superscalar extended models.

compress，eqntottの制御並列の2PEの場合よりも同一資源量の8SSモデルの方が高い性能を示しているが，それ以外はスーパスカラプロセッサを強化したモデルよりもMUSCATモデルの方が高い性能を示している．この要因としては，3章で示したコード生成手法によって，実行が確実なもの，もしくは実行される確率の高いものが優先してスレッドとして起動するため，分岐予測ミスペナルティが課せられない効果，および，命令ウィンドウが分割されたことによって，実行できる命令が有効に命令ウィンドウ内に格納できるプリフェッチ効果が大きいことがあげられる．

スーパスカラの強化モデルの性能向上を妨げる要因

は，idctなどでは，命令ウィンドウサイズがループの1反復の範囲内に限られることによる．この問題は，ループアンローリングなどの広域コードスケジューリングを効果的に行うことによって解消できると考えられるが，使用可能な論理レジスタ本数がアンローリング数を制限するという問題は依然として残る．

また，基本ブロック間の制御の依存性が強いcompress，eqntottなどでは分岐スペキュレーションの失敗にともなう再実行コストが大きいことなどがあげられる．分岐ミスの影響を確認するために，分岐予測の成功率を100%としてcompressの32SSモデルと4SSモデルを実行したところ32SSモデルは4SSモデルの1.61倍の性能向上が確認された．このとき，MUSCATの8PEモデルでは1.88倍の性能向上となり性能差が縮小する．

5. まとめ

本論文では，制御並列アーキテクチャMUSCATを提案した．MUSCATでは以下のアーキテクチャ的な工夫を行った．

- フォーク1回モデルの導入とコンパイラによる並列化
- フォーク時レジスタ継承
- メモリ上のデータ依存性制御方法を複数提供
- 制御並列処理向き同期命令の導入

フォーク1回モデルの導入によりスレッド管理/制御が簡単化され，スレッド管理オーバヘッドの低減が可能となった．また，レジスタ継承をフォーク時に行うことにより，データ交信オーバヘッドの低減と命令コードに加えるデータ量の抑制を両立した．さらに，複数のデータ依存制御/同期命令の追加によって，コードの性質にあわせて最適な制御が可能になった．

これらの効果は，シミュレーションによって確かめられ，以下の特性が明らかになった．

- 制御の依存性が強く他の並列化手法では速度向上が困難な問題に対しても高速化が実現可能である．この要因は確実もしくは高い確率で実行されるスレッドの先行実行による．また，細粒度スレッドへの分割オーバヘッドは，高速スレッド起動とフォーク時レジスタ継承によって削減可能である．
- 粒度が比較的大きなidctやp_blockなどの問題も，効率的に実行可能である．命令ウインドウの範囲では命令間の依存が存在するが，その範囲を拡大すれば命令間の依存が存在しない命令を実行できるという性質を利用可能である．
- 制御並列のターゲットとする細粒度スレッド処理では，スレッドごとの粒度差がそれほど大きくならない範囲で並列化可能であり，フォーク1回モデルによる性能劣化は特に認められなかった．

一方，スーパスカラ方式では，制御の依存性の強い問題の場合，コードスケジューリングの自由度も低くなり，結局性能は分岐予測精度によるものが大きくなる．1サイクルあたりの並列度の向上にはより高度な分岐予測か条件付き実行命令の導入が考えられる．しかしながら，分岐予測率を100%にするのは不可能であり，条件付き実行命令が適応できる分岐も限られる．

また，粒度の大きな問題では，命令ウィンドウ中に同時発行可能な命令を格納するコードスケジューリングが重要となる．しかしながら，大域的な命令の移動は，コンパイラ解析能力，論理レジスタ本数，大域移動にともなう補償コードの挿入などの問題が生じる．

これらのことから，MUSCATが命令レベル並列処理の次の世代のマイクロプロセッサアーキテクチャとして有力な選択肢になりうることが明らかになった．一方で

- 細粒度スレッドを少PE数モデルで実行した場合，スレッドが完全に終了しないと，次のスレッドの起動ができないため性能向上がおさえられる．
- PE数が増えた場合には，キャッシュの容量や連想数の不足からスラッシングが生じ，性能向上がおさえられる．

というような問題点もあわせて明らかになった．前者の問題については，先行スレッドの最終命令デコード後に，後続スレッドのデコードを開始させることで解消できるが，1PEに複数のスレッドが同時に存在することになるため，スレッド管理が複雑化する．

今後は，今回の問題点等の検討を行い，さらなる性能向上を目指すべくアーキテクチャ的な改良，およびコンパイラの検討を継続し，フォーク1回モデルやスペキュレーション等の是非を含めて，さらに詳細な評価を行う予定である．あわせて，ハードウェア規模，実現性についても検討を進める予定である．

参考文献

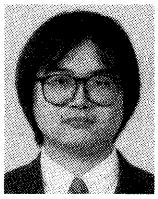
1) Motomura, M., Inoue, T., Torii, S. and Konagaya, A.: Ordered Multithreading: A Novel Technique for Exploiting Fine-Grained Parallelism., *Proc. International Conference on Parallel Architectures and Compilation Techniques*, pp.37–48 (1995).
2) 本村，井上，鳥居，小長谷：順序付きマルチスレッド実行モデルの提案とその評価，情報処理学会論文誌，Vol.37, No.7, pp.1335–1366 (1996).

3) 鳥居，本村，近藤，鈴木，小長谷：順序付きマルチスレッドアーキテクチャのプログラミングモデルと評価，情報処理学会論文誌，Vol.38, No.9, pp.1706–1716 (1997).
4) Sohi, G.S., Breach, S.E. and Vijaykumar, T.N.: Multiscalar Processor, *Proc. 22nd Annual International Symposium on Computer Architecture*, pp.414–425 (1995).
5) Dubey, P.K., O'Brien, K., O'Brien, K.M. and Barton, C.: Signle-Program Speculative Multithreading (SPSM) Architectire: Compiler-assisted Fine-Grained Multithreading, *Proc. International Conference on Parallel Architectures and Compilation Techniques*, pp.109–121 (1995).
6) Tsai, J.-Y. and Yew, P.-C.: The Superthreaded Architecutre: Thread Pipelining with Run-time Data Dependence Checking and Control Speculation, *Proc. International Conference on Parallel Architectures and Compilation Techniques*, pp.35–46 (1996).
7) Kemp, G.A. and Franklin, M.: PEWs: A Decentralized Dynamic Scheduler for ILP Processing, *Proc. International Conference on Parallel Proccssing*, Vol.1, pp.239–246 (1996).
8) Oplinger, J., Heine, D., Liao, S.-W., Nayfeh, B.A., Lam, M.S. and Olukotun, K.: Software and Hardware for Exploiting Speculative Parallelism with a Multiprocessor, *Stanford University Computer Systems Lab Technical Report CSL-TR-97-715*, pp.239–246 (1997).
9) 朝生，高田，森，清水，林：マルチスーパスカラパイプラインによるブロック並列実行方式，情報処理学会アーキテクチャ研究会報告，I19-11, pp.19–24 (1996).
10) 玉造，松本，平木：On-Chip parallel processorにおける大規模投機実行，情報処理学会アーキテクチャ研究会報告，I25-24, pp.139–155 (1997).
11) 小林，岩田，安藤，鳥田：制御依存解析と複数命令流実行を導入した投機的実行機構の提案と予備的評価，情報処理学会アーキテクチャ研究会報告，I25-23, pp.133–138 (1997).

(平成 9 年 11 月 4 日受付)
(平成 10 年 4 月 3 日採録)

鳥居　淳（正会員）

1967 年生．1992 年慶應義塾大学大学院理工学研究科修士課程修了．同年 NEC 入社．同社 C&C メディア研究所にてマイクロプロセッサ，並列処理アーキテクチャ，並列処理プログラミングの研究に従事．



近藤　真己（正会員）

1962 年生．1984 年山形大学理学部物理学科卒業．同年日本電気技術情報システム開発入社．現在 NEC 情報システムズ技術システム事業部第一技術部にて自動並列化コンパイラの研究に従事．

本村　真人

1962 年生．1987 年京都大学大学院理学研究科修士課程修了．同年 NEC 入社．同社シリコンシステム研究所にてプロセッサのアーキテクチャ等の研究に従事．工学博士．1992 年度 IEEE JSCC Best Paper Aword 受賞．電子情報通信学会，IEEE Computer Society 各会員．

池野　晃久

1968 年生．1995 年都立科学技術大学大学院工学研究科修士課程修了．同年 NEC 情報システムズ入社．同社技術システム事業部第一技術部にて性能評価技術，シミュレーション技術の研究に従事．

小長谷明彦（正会員）

1955 年生．1980 年東京工業大学大学院情報科学専攻修了．同年 NEC 入社．1996 年東京工業大学大学院知能システム科学専攻客員助教授．1997 年より北陸先端科学技術大学院大学知能科学研究科教授．遺伝子情報処理，進化的システム，コンピュータシステムの研究に従事．工学博士．

西　直樹（正会員）

1959 年生．1984 年広島大学大学院システム工学専攻修了．同年 NEC 入社．スーパコンピュータや並列コンピュータの研究/製品開発，また，マイクロプロセッサ研究開発に従事現在，NEC C&C メディア研究所研究課長．