Vol.2010-ARC-187 No.23
Vol.2010-EMB-15 No.23
2010/1/29

Feature-Packingのための ソフトウェアによるメモリ管理手法の実装と評価

三好健文[†1,†2] 入谷優[†4] 植原昂[†1]
笹田耕一[†3] 小林良太郎[†5] 吉瀬謙二[†1]

Feature-Packing（FP）プロセッサ・アーキテクチャは多数のシンプルなコアによるメニーコアアーキテクチャである．我々は，FP において各コアがローカルに持つメモリ (ノードメモリ) を効率良く利用するためにオーバレイによるメモリ管理手法を提案した．提案手法の実現には，ノードへのアクアセスの管理機構が必要であり，ハードウェアを用いる手法 (手法 1) と，ソフトウェアだけで実現する手法 (手法 2) が考えられる．手法 1 は手法 2 よりオーバヘッドが少ないと考えられるが，多量のハードウェアリソースが必要になる．手法 1，2 による管理機構を用いた場合のプログラムの実行時間を比較したところ，手法 2 を使う場合は，手法 1 を使う場合に対して，数倍から数十倍の実行時間であることが示された．また，ソフトウェアによる実装方式のオーバヘッドは最適化によって削減できることが分かった．

An Implementation and Evaluation of Software Memory Management for Feature-Packing

Takefumi MIYSOHI ,[†1,†2] Masaru IRITANI ,[†4]
Koh UEHARA ,[†1] Koichi SASADA ,[†3]
Ryotaro Kobayashi [†5] and Kenji KISE [†1]

In this paper, an implementation and evaluation of software memory management for Feature-Packing is described. Feature-Packing processor architecture is a many cores architecture, which consists of many simple cores. The authors have proposed an explicit method for memory management by overlay. An access management mechanism of node communications is required for implementing the proposed method. The mechanism is implemented by using hardware (“method 1”) or by using only software (“method 2”). It is expected that execution overhead of “method 1” is much less than “method 2”, however much hardware resource is required. As program execution time on these two mechanism, the execution time of “method 2” is about from several to dozens of times more than “method 1”. On the other hand, the overhead of “method 2” can be reduced by some optimizations.

1. はじめに

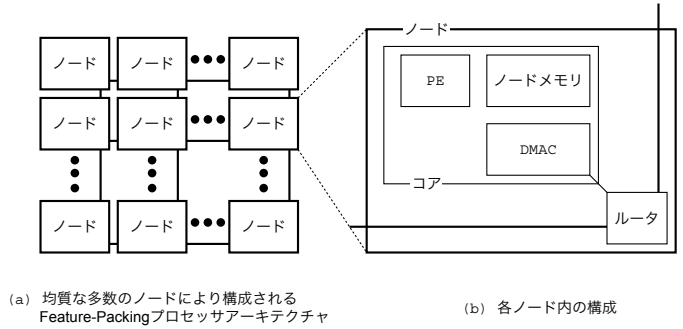
マルチコア・アーキテクチャは，低消費電力で高い計算処理能力を得ることが可能なプロセッサ・アーキテクチャとして活発に研究開発されている．消費電力や配線遅延の増加などによる大規模な単一コアの性能向上がますます困難になっていく一方で，集積度は着実に向上し続けているため，今後，プロセッサ・アーキテクチャは，数百のコアを搭載するメニーコアの時代となる．そこで，メニーコアの実現における重要な課題である高速化，省電力化，ディペンダビリティの向上，製造コストの低減を解決するために Feature-Packing（FP）プロセッサ・アーキテクチャが提案されている[1)]．FP は，アーキテクチャの技術とソフトウェアの技術を組み合わせることで，多機能なメニーコア・プロセッサを実現するプロセッサアーキテクチャ技術の枠組みである．メニーコアを効率良く活用するためには，その豊富なコア数を活用し，消費電力と配線遅延を抑制しつつプロセッサ全体のスループットを向上させることで，プログラムを効率良く実行する必要がある．また，アプリケーションの多様化に伴い，各アプリケーションの性質に合わせて，プロセッサの性能，消費電力，及びディペンダビリティの間に存在するトレードオフを解決するための，高い柔軟性が必要となる．これらを実現するために，FP では，キャッシュや割り込み，高機能なメモリアクセス機構といった機構を除いたシンプルなコアを多数用いる手法を提案している．

コアがシンプルであることは，開発コストを抑制し，多数のコアをチップに配置するメニーコアの実現を助ける．また，キャッシュや割り込みがないシンプルなプロセッサコアはプログラムの動作を解析，予測することが比較的容易であり，各種最適化やコア間の柔軟な連携を実現しやすい．しかしその一方で，プログラマにコア間の通信やデータ管理などといったアーキテクチャ上の制約を考慮したプログラム記述を強いる．

そこで，FP を対象とするプログラミングを容易にするために，コンパイラ及びリンカによってサポートすることが可能なソフトウェアによるメモリ管理手法を考える．我々は，文献 2) で効率良いデータアクセスを実現するために，プログラムの命令領域とデータ領域をそれぞれセクションに分割し，それらをノードメモリ上にオーバレイによって割当てる手法を提案した．また，必要に応じて，外部メモリを用いたスタック領域の退避及び復帰を実現する命令を挿入する．セクションの分割及びメモリ管理コードの挿入は，コンパイル及びリンク時に自動的に実行することができるため，この手法によりプログラマの開発コストを削減することができる．

†1 東京工業大学大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology
†2 独立行政法人 科学技術振興機構
Japan Science and Technology Agency
†3 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†4 東京工業大学工学部
School of Engineering, Tokyo Institute of Technology
†5 豊橋科学技術大学 情報工学系
Dept. of Information and Computer Science, Toyohashi University of Technology

Vol.2010-ARC-187 No.23
Vol.2010-EMB-15 No.23
2010/1/29



（a） 均質な多数のノードにより構成される Feature-Packingプロセッサアーキテクチャ　　（b） 各ノード内の構成

図 1 Feature-Pcaking プロセッサアーキテクチャ概要

本論文では，文献 2) で提案したソフトウェアによるメモリ管理手法を FP アーキテクチャ上に実装し評価する．オーバレイによるメモリアクセスを実装する場合に限らず，FP アーキテクチャ上では，特定のノードからのデータの読み出しには，アクセスの管理機構が必要となる．オーバレイでは，グローバルメモリとローカルメモリの間で頻繁にデータのコピーが行われるため，この管理機構による実行速度の低下は重要な問題である．しかし，その一方で，多数のコアをチップ内に搭載するため，必要となるリソース量がコア数にスケールすることが求められる．本論文では，アクセス管理の専用ハードウェアがある場合と，専用のハードウェア機構をもたずソフトウェアで排他制御を行う場合の二種類の実装を行いリソース量および性能の評価を行う．

第 2 節で対象とする FP について述べ，第 3 節で FP におけるメモリ管理の問題と，ソフトウェアによる明示的なメモリ管理手法について述べる．第 4 節で，アクセス管理機能の実現手法として，専用のハードウェアによる機構を用いる場合と，ソフトウェアのみで行う場合について述べる．第 5 節で，オーバレイを用いて実行される 4 種類のプログラムによって，実装方法による実行時間の差異を示し，第 6 節では，結果の考察および必要となるハードウェアリソース量の見積りを示す．

2. FP プロセッサ・アーキテクチャ

図 1 に Feature-Packing プロセッサアーキテクチャ（FP）を示す．FP は，2 次元メッシュ状に配置される多数の均一な計算ノードと，それ以外のモジュールから構成される．各計算ノードは，(1) コア：2-Way スーパスカラ程度の RISC プロセッサである演算処理器（PE），PE から直接アクセス可能な数百 KB 程度のメモリ（ノードメモリ），と，コア間のデータ転送を行う Direct Memory Access Controller（DMAC）で構成される，及び (2) ルータ：ノード間の Network-on-Chip 機構を提供する，から成る．モジュールとしては，外部 I/O や割込み処理を行う汎用プロセッサや，メインメモリがある．この ID によって特定のノードあるいはモジュール間の通信を実現する． ノード間の通信には，コア A がコア B にデータを送る PUT アクセスと，コア A がコア B のデータを読み出す GET アクセスがある．

各コアから大規模なキャッシュ，複雑な分岐予測機構，大規模な投機処理機構などを排除することで，メニーコアの利点である消費電力と配線遅延の抑制をさらに高め，また，タイル・アーキテクチャ[3] と同様，設計と検証のコストを抑えることができる．また，ノードの多様性，融合性，独立性などを利用し，柔軟に制御する．これによりアプリケーションの性質に合わせて，様々なトレードオフを考慮したプロセッサの規模や機能を動的に決定することができる．

本論文では，FP アーキテクチャからさらに具体的に検討されたメニーコアプロセッサアーキテクチャである M-Core[4] アーキテクチャを対象として，ソフトウェアによるメモリ管理手法を検討する．M-Core アーキテクチャでは，

- 各ノードは，DMA 通信を管理する Inter Node Connection Controller(INCC) をもつ．DMAC は INCC 内に実装される．
- メインメモリは，INCC を持つメモリノードによって 2 次元メッシュ上に配置される．メモリノードは計算ノードと同様に ID を持ち，計算ノードは，DMA 通信による PUT/GET アクセスができる．

となる．

3. ソフトウェアによるメモリ管理手法

FP はメニーコア・アーキテチャの実現が目標であるから，各コアが潤沢なメモリを持つことは現実的ではない．そこで各ノードメモリのサイズを小さくすることが求められる．しかし，プログラム実行時に頻繁に外部メモリへのアクセスが発生し，高い性能を得ることができない．そのため，効率良くプログラムを実行可能な適切なメモリ管理が必要となる．

一般に，効率良くプログラムを実行するためのメモリ管理として，ハードウェアによるキャッシュやページ管理といった支援機構が用いられる．しかし，ハードウェアによる実行支援では，必要となる多量のハードウェア資源による面積や消費電力の増加及びコアの複雑化に伴う開発コストの増加によって，メニーコアの実現が難しくなる．一方でソフトウェアによるメモリ管理は，ハードウェア資源の増加を必要としない．しかし，プログラマに余計な負担をかけ，ソフトウェア開発コストが増加する．

そこで，ハードウェア資源の追加を必要としない完全なソフトウェアによるメモリ管理を，プログラマに意識せずに実現することが求められる．我々は，ソフトウェアによる明示的なメモリ管理によって，FP アーキテクチャのコア内のノードメモリを有効に利用する手法を検討した[2]．文献 2) の手法では，効率良いデータアクセスを実現するために，プログラムの命令領域及びデータ領域をセクションに分割し，それらをノードメモリ上にオーバレイによって割当てる．また，必要に応じて外部メモリを用いたスタック領域の退避及び復帰を実現する管理コードを挿入する．セクションの分割とメモリ管理コードの挿入はコンパイル及びリンク時に自動的に実行することができるため，この手法によりプログラマの開発コストを削減することができる．

オーバレイは，外部メモリ上の一部分を高速なローカルメモリ上のアドレス空間に重ねて割当て，実行時に切り替えて使用する手法である．アクセスに時間のかかる外部メモリのデータをアクセス時間の短いローカルメモリ上に置くことで効率良くプログラムを実行す

Vol.2010-ARC-187 No.23
Vol.2010-EMB-15 No.23
2010/1/29

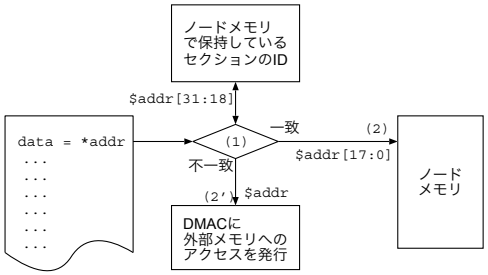


図 2 ソフトウェアによる明示的な管理下でのノードメモリアクセス

ることができる．また，プログラム中では，ローカルメモリ上のアドレスが実効アドレスであり，複雑なアドレス変換機構を必要としない．これは FP のシンプルなコアを多数並べるという設計方針に沿っている．

ここで，グローバルアドレス空間をセクションと呼ぶ単位に分割しノードメモリへ割当てる．ハードウェアに手を入れず効率良く実装するために，アドレスビットを分割して，セクション識別番号（セクション ID）とノードメモリアドレスを表現する．すなわち，アドレス空間 32bit，ノードメモリ 256kB の場合，下位 18bit がノードメモリアドレスに相当し，上位 14bit でセクション ID を示す．

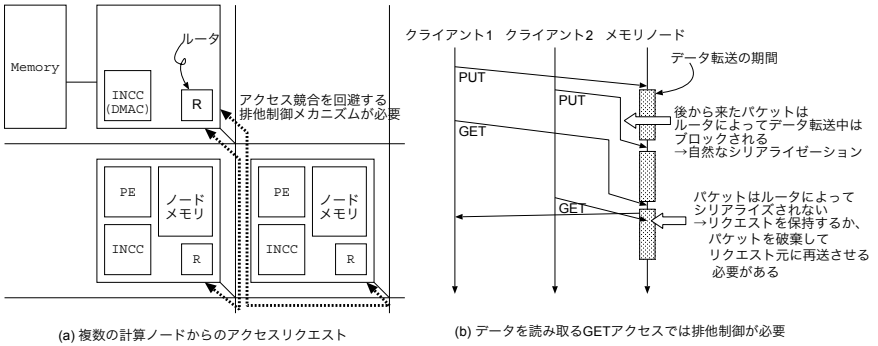
ノードメモリには，命令領域及びデータ領域の各セクションのデータがオーバレイによって割当てられる．プログラム実行中に，ノードメモリ上に保持されているデータのセクション ID をそれぞれレジスタ上に保存し管理する．ソフトウェアによる明示的な管理下でのノードメモリへのアクセスを**図 2** に示す．ノード内の PE で動作するプログラムコードは，データにアクセスする際，まずノードメモリに現在格納されているセクション ID と，アクセスしようするデータのセクション ID を比較する (図 2(1))．

ここで，ノードメモリに格納されているデータのセクション ID がアクセスしたいデータのセクション ID に等しい場合，図 2 中の矢印 (2) のように，ノードメモリ上のデータに直接アクセスする．しかし，現在ノードメモリ上にあるデータのセクション ID とアクセスしたいデータのセクション ID が異なる場合，図 2 中の矢印 (2)' で示すように，DMAC へメモリ転送要求を発行する．

本論文では，さらに，ノードメモリ上でオーバレイする領域をダイレクトマップド方式によって分割することを考える．これは分割数に応じたセクション ID の下位ビットを用いてノードメモリ上の，オーバレイ先のエリアを指定する．外部メモリ上の幾つかの離れた領域を同時にアクセスするようなプログラムを実行する場合，ノードメモリがサポートするエリアを分割することで，オーバレイの発行回数を削減することができると考えられる．これにより，実行時間の削減が期待できる．

4. アクセス管理制御

第 3 節で述べた手法を FP アーキテクチャ上に実装するためには，メインメモリに接続



(a) 複数の計算ノードからのアクセスリクエスト　(b) データを読み取るGETアクセスでは排他制御が必要

図 3 メモリアクセスの競合

したメモリノードの DMAC へのメモリ転送要求が，メモリ転送中に破壊されないように，管理する必要がある．**図 3** に複数の計算ノードがメモリノードからデータを読み出す例を示す．FP アーキテクチャでは，DMA 転送を用いてノード間でデータを授受する．ここで，二つのノードがメモリノードからデータを取得しようとするとき，互いのリクエストが衝突する．

単純な双方向の 5 ポートのバッファを有する FP のルータでは，外からノードへ入ってくるパケットは，ルータで排他制御される．そのため，複数のノードからのアクセスは，自然にシリアライズされ，図 3(b) のように 2 つの PUT アクセスが正しく動作する．一方で，GET アクセスの場合には，メモリノードから GET リクエストを行ったノードにデータを転送している時に，他のノードからきた GET リクエストがノードメモリに到着する可能性がある．この場合，後続の GET リクエストに対して正しくデータを返すためには，すべてのリクエストを保持しておくか，あるいは，リクエストは破棄し，リクエスト元に再送要求をだす必要がある．

以降，本節では，複数の GET リクエストを正しく処理するために管理する機構を，ハードウェアとして実装する場合と，専用のハードウェアがなくソフトウェアで排他制御を行う場合の二種類の実装について述べる．

4.1 ハードウェアによる GET リクエスト管理手法 (手法 1)

ハードウェアによってリクエストを管理する場合，すべてのリクエストのデータを保持しておくことは，リソース量の観点から現実的ではないと考えられる．そのため，受理できなかったリクエストは ID だけをバッファに保存し，それらに対し，再送要求を発行する方式を考える．

M-Core アーキテクチャ[4] では，各ノードの INCC にリクエストを保持するためのバッファが用意されている．このバッファを用いたアクセスの管理を**図 4** に示す．GET の処理中に送られてきた後続のリクエストは，そのノード ID が NACK バッファに格納され，リクエストデータは破棄される．INCC は，バッファ中の ID を用いて後続のノードにリクエストデータを破棄したことを通知する．

Vol.2010-ARC-187 No.23
Vol.2010-EMB-15 No.23
2010/1/29

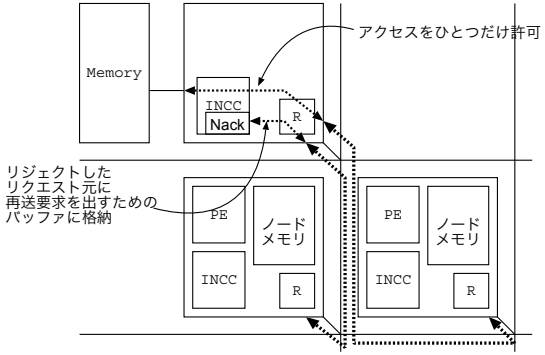


図 4 ハードウェアによる排他制御の実現手法

しかし，すべてのノードからの GET リクエストに対して正しい動作を保証しなければならないとき，必要となるバッファの量は，ノード数に比例すると考えられる．そのため，この機構を現実のハードウェア上にこのまま実装することは，ハードウェアリソースの観点から許容できないものとなる可能性がある．

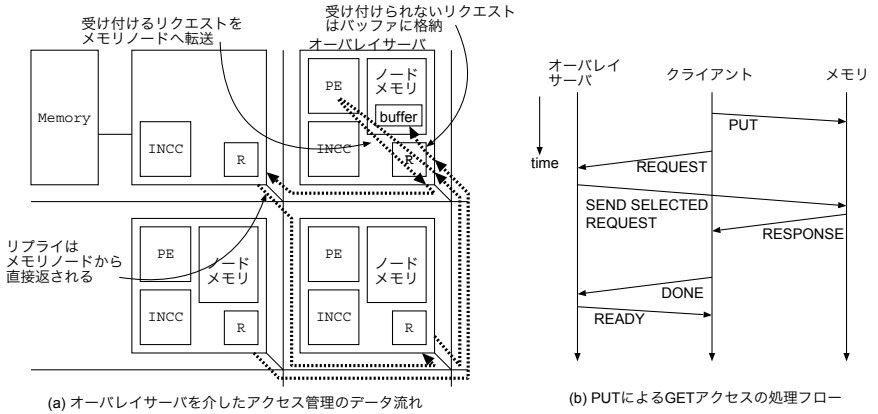
4.2 ソフトウェアによる GET リクエスト管理手法 (手法 2)

一方ハードウェアによる管理機構を用いず，ソフトウェアでリクエストを管理することを考える．これは，メニーコアプロセッサ上の多数のノードの一つ或いは複数で動作するソフトウェアプロセスとして実現される．そのため，特別なハードウェア機構は必要としない．リクエスト管理のプロセスを実行する計算ノードをオーバレイサーバと呼ぶ．

図 5 にオーバレイサーバを介したアクセスにおけるリクエストの流れをを示す．メインメモリに対してデータの送受信を行う計算ノード (クライアント) は，オーバレイサーバに対して GET リクエストに相当するデータのリクエスト情報を送る．オーバレイサーバでは，リクエスト情報を保持し，定められた優先順位で選択しメモリノードへ転送する．メモリノードへのリクエストの転送は，メモリノードの DMA 制御レジスタに制御コマンドを書き込むことで実現できる．リクエストからのレスポンスはメモリノードから直接，リクエスト元の計算ノードに返される．ここで，ノード間の DMA アクセスは，すべて PUT アクセスであることに注意されたい．すなわち，これらのアクセスはルータによって排他制御がされるため，通信の欠損が生じない．

5. シミュレーションと評価

ハードウェアとソフトウェアによる 2 つのメモリアクセス管理手法を用いて実装したオーバレイ手法をメニーコアアーキテクチャのシミュレータである SimMc[4] を用いて評価する．シミュレータの実行速度の制約および評価の簡単のため，オーバレイの処理に用いる各ノードメモリ内の記憶領域 (ローカルストレージ) を 2^8B と制限し，最大 2^{12}B のデータにアクセスするプログラムをベンチマークとして用いる．またプログラムコードおよび使用するスタックはノードメモリに十分収まるサイズでありオーバレイの対象とならないと仮定する．同一のプログラムにおいて，処理するデータ量とローカルストレージの比が等しいとき，プログラム中のオーバレイの発行回数は等しい．そのため，この制約の下の評価は，ローカルストレージ 2^{18}B に対して，2^{22}B の実データをオーバレイして実行するプログラムの評価としてみることができる．これは一般的なプログラムの同一フェーズにおいて使用するデータとして現実的な評価対象であると考えられる．



(a) オーバレイサーバを介したアクセス管理のデータ流れ　(b) PUTによるGETアクセスの処理フロー

図 5 ソフトウェアによる排他制御の実現手法

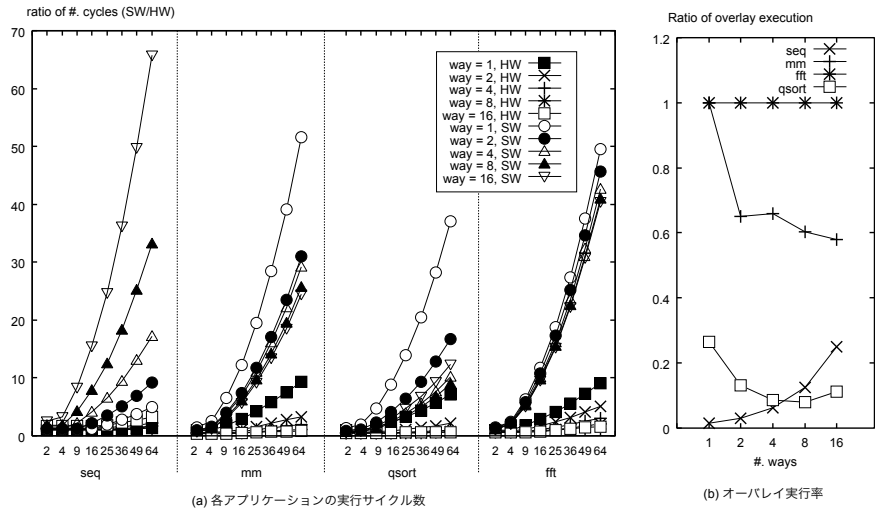
コア内のノード数は，2，4，9，16，25，36，49，64 個の場合について評価を行う．ただし，1 個のノードは，オーバレイサーバとして予約してあるため，実際にベンチマークプログラムを実行するノードは，1，3，8，15，24，35，48，63 個である．

ベンチマークとして，メモリアクセスパタンが異なるプログラムである逐次アクセス (seq)，クイックソート (qsort)，行列積 (mm)，FFT(fft) の 4 種類を選択した．これらのプログラムはチップ中のノードで独立に実行する．ローカルストレージを 1，2，4，8，16 の way 数の領域に分割した場合の実行時間を評価する．

FP アーキテクチャを実現可能なプロセッサアーキテクチャとしてとらえる時，メインメモリとノードメモリ間のレイテンシは，重要なパラメタである．メインメモリは，オフチップのメモリであり，そのレイテンシは非常に大きい．しかし，本論文では，ノードメモリは，計算ノードからリクエストされたメインメモリの内容を遅延なしで返すことができると仮定する．レイテンシを考慮する場合，オーバヘッドがメモリアクセスレイテンシに隠蔽されると考えられる．そのため，オーバレイの実現手法を検討する場合には，この仮定により実装手法による差異が明確化され議論は容易になると考えられる．

図 6(a) に専用ハードウェア (HW)/ソフトウェア (SW) によるアクセス管理手法を用いてオーバレイを実行した各アプリケーションの実行サイクル数を示す．実行サイクル数は，各アプリケーションを専用ハードウェアによる管理手法を用いてローカルストレージを分割

Vol.2010-ARC-187 No.23
Vol.2010-EMB-15 No.23
2010/1/29



(a) 各アプリケーションの実行サイクル数　(b) オーバレイ実行率

図 6　二つの実装方法による実行サイクル数の比とオーバレイ実行率

しない場合 (way=1) の結果で正規化して示している．

すべてのベンチマークにおいて，ソフトウェアによる GET アクセス管理手法を用いる場合には，ハードウェアによる機構を用いる場合と比べ，数倍から数十倍の実行サイクル数となることがわかる．現状のソフトウェアによる実装では，リクエストの選択のための検索にかかる時間が線形に増加する．実行結果から，この傾向が見てとれる．また，コア数の増加に伴う，ネットワーク中の衝突も実行時間の増加に起因する．

また，図 6(b) は，オーバレイ対象領域へのメモリアクセス中にオーバレイを実行する必要があった回数を全アクセス数に対する割合で示す．オーバレイの実行率が低い程，GET リクエストの管理機構によるオーバヘッドが発生しないことを意味する．そのため，図 6(b) においてソフトウェア・オーバレイの実行率が低い場合には，図 6(a) における専用ハードウェアの有無による実行時間の差が小さいことが確認できる．

6. 考　　察

シミュレーションによる実験の結果から，ソフトウェアによるアクセス管理機構による実装では，すべてハードウェアによってアクセス管理機構を実装した場合より実行時間が数倍から数十倍大きいことが示された．

しかし一方で，ハードウェアによるアクセス管理機構は，高価なものであるから，その実行時間の差は高々数倍から数十倍でしかない，とも考えることができる．VHDL で記述して，Xilinx XC3S500E[5] を対象に ISE WebPack 11.3[6] で合成した場合に，必要となるハードウェアリソース量と信号遅延を**表 1** に示す．XC3S500E で使用可能なスライス数は

表 1　専用ハードウェアの実現に必要なスライス数と信号遅延

コア数	2	4	9	16	25	36	49	64	81	100
スライス数	33	40	72	88	115	141	185	250	318	367
信号遅延 (ns)	5.613	6.050	6.383	6.552	6.466	6.888	7.535	8.816	9.493	10.094

4656 個であるので，必要なリソースは，コア数 64 の場合でも 5%程度となり，コア数 400 の場合には 37%程度となる．これは，同じ XC3S500E 上に実装した MIPS コアのリソース量が 50%程度，M-Core アーキテクチャを FPGA 上に実装した ScalableCore ノード[7] のリソース量が 90%程度であることを考えると，無視できない大きさである．

さらに，必要となるリソースの量が，プロセッサ中のノード数に対し $O(N)$ で増加することに注意されたい．これは，プロセッサ中のノード数 N に対して，プロセッサ全体で必要となるリソース量が $O(N^2)$ で増加することを意味する．

また，オーバレイの発行回数が少ない場合には，アクセス管理機構手法のハードウェアとソフトウェアの実装方法による実行時間の差異は小さくなる．すなわち，今回行ったようなオーバレイ領域の分割や，各種最適化によってデータの局所性の抽出と活用によってオーバレイの発行回数を削減することで，ソフトウェアによるアクセス管理手法のオーバヘッドの相対的な削減が実現可能である．

これらから，ソフトウェアのみによるアクセス管理機構は，FP アーキテクチャにとって十分に現実的であると考えられる．

7. 関 連 研 究

マルチコア，メニーコアを対象として，計算に必要なメモリを近くに配置する手法にはいくつかの研究がなされている．同じく二次元メッシュ型のメニーコアプロセッサである Raw Machine[8] のメモリ管理手法として Maps[9] が提案されている．この手法では，コンパイル時に ECU とモジュロアンローリングによる静的なデータのコア割当て及び，実行時の直列化について述べられている．しかし，ローカルメモリへのアクセスミス時の，他のコアのメモリあるいは外部メモリへのアクセス手法については述べられていない．FP の各ノードは割込み機構を持たないため，ノードメモリへのアクセスミス時に明示的なメモリ管理手法が必要となる．1024 コアを実現する Rigel アーキテクチャ[10] のためのメモリ管理手法が提案されている．Rigel では，グローバルメモリ[11] に対するアトミックなアクセスが提供されていることが仮定されており，本論文で検討したような機構が必要ない．

また，メモリの局所性解析やメモリ管理の研究には，チップ内のスクラッチパッドメモリを有効に利用するための手法[12)–14)] や，配列のアクセス範囲の解析手法[15)] が研究されている．これらの最適化手法を，提案手法であるソフトウェアによるメモリ管理手法にも適用することが可能であり，これは今後の課題である．

8. ま と め

文献 2) で提案した Featcure-Packing プロセッサアーキテクチャを対象としたオーバレイによるソフトウェアによる明示的なメモリ管理手法を実装した．実装に必要となるノード間アクセスの管理機構に専用のハードウェアを用いる手法 (手法 1) と，ソフトウェアのみ

Vol.2010-ARC-187 No.23
Vol.2010-EMB-15 No.23
2010/1/29

によって実現する手法 (手法 2) が考えられる．手法 1 は手法 2 より，オーバヘッドは小さいが必要となるリソース量が大きくなると予想される．異なるメモリアクセスパタンを持つベンチマークプログラムによって評価したところ，手法 1 に対し，手法 2 での実行時間は数倍から数十倍程度となることが示された．考察として，手法 1 の実現に必要となるハードウェアリソース量の見積りを行い，追加で必要となるハードウェア量が，プロセッサ内のノード数 N に対し $O(N^2)$ で増加することを示した．また，オーバレイの発行回数が少ないときには，手法 2 と手法 1 の実行時間の差は小さくなることを確認した．すなわち，各種最適化によって発行回数を小さくできるとき，FP アーキテクチャに対してソフトウェアのみでアクセス管理を行う手法 2 は，十分実用的であることが分かる．

今後の課題として，メモリアクセスの局所性などを考慮した静的，動的な最適化によるオーバレイ発行回数を削減する最適化手法の実現が考えられる．また，今回はオーバレイ実行時の通信衝突およびメインメモリへのアクセスレイテンシについての考慮を行っていない．この解析と最適化について検討することも今後の課題である．

謝　　辞

本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (CREST) 「アーキテクチャと形式的検証の協調による超ディペンダブル VLSI」の支援による．

参 考 文 献

1) 小林良太郎, 吉瀬謙二. 多機能メニーコアを実現するアーキテクチャ技術 feature-packing の構想 (inventive and creative architecture 特別セッション i). 情報処理学会研究報告. 計算機アーキテクチャ研究会報告, Vol. 2007, No. 115, pp. 11–15, 20071121.
2) 三好健文, 笹田耕一, 小林良太郎, 植原昂, 吉瀬謙二. ”feature-packing のためのソフトウェアによるメモリ管理手法の検討”. 情報処理学会研究報告 2008-ARC-180, pp. 45–48, October 2008.
3) MichaelBedford Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ilp and streams. *SIGARCH Comput. Archit. News*, Vol.32, No.2, p.2, 2004.
4) Koh Uehara, Shimpei Sato, Takefumi Miyoshi, and Kenji Kise. A study of an infrastructure for research and development of many-core processors. In *Workshop on Ultra Performance and Dependable Acceleration Systems held in conjunction with PDCAT'09*, December 2009.
5) Spartan-3E. http://www.xilinx.com/support/documentation/spartan-3e.htm.
6) ISE WebPackDesign Software. http://www.xilinx.com/tools/webpack.htm.
7) 高前田伸也, 渡邉伸平, 姜軒, 藤枝直輝, 植原昂, 三好健文, 吉瀬謙二. メニーコアアーキテクチャ研究のためのスケーラブルな hw 評価環境 scalablecore システム. 情報処理学会研究報告 2009-ARC-177, October 2009.
8) M.B. TAYLOR. Evaluation of the raw microprocessor : An exposed-wire-delay architecture for ilp and streams. *Proc. ISCA-31, 2004*, 2004.
9) Rajeev Barua, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Maps: a compiler-managed memory system for raw machines. *SIGARCH Comput. Archit. News*, Vol.27, No.2, pp. 4–15, 1999.
10) JohnH. Kelm, DanielR. Johnson, MatthewR. Johnson, NealC. Crago, William Tuohy, Aqeel Mahesri, StevenS. Lumetta, MatthewI. Frank, and SanjayJ. Patel. Rigel: an architecture and scalable programming interface for a 1000-core accelerator. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pp. 140–151, New York, NY, USA, 2009. ACM.
11) JohnH. Kelm, DanielR. Johnson, StevenS. Lumetta, MatthewI. Frank, and SanjayJ. Patel. A task-centric memory model for scalable accelerator architectures. In *PACT '09: Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, pp. 77–87, Washington, DC, USA, 2009. IEEE Computer Society.
12) Manish Verma, Lars Wehmeyer, and Peter Marwedel. Dynamic overlay of scratchpad memory for energy minimization. In *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 104–109, New York, NY, USA, 2004. ACM.
13) S.Steinke, L.Wehmeyer, B.Lee, and P.Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, p. 409, Washington, DC, USA, 2002. IEEE Computer Society.
14) Manish Verma, Stefan Steinke, and Peter Marwedel. Data partitioning for maximal scratchpad usage. In *ASPDAC: Proceedings of the 2003 conference on Asia South Pacific design automation*, pp. 77–83, New York, NY, USA, 2003. ACM.
15) Muthu Manikandan Baskaran, Uday Bondhugula, Sriram Krishnamoorthy, J.Ramanujam, Atanas Rountev, and P.Sadayappan. Automatic data movement and computation mapping for multi-level parallel architectures with explicitly managed memories. In *PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pp. 1–10, New York, NY, USA, 2008. ACM.