

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ー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 よりオーバー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ー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 MIYOSHI^{†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 mechanisms, 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、プログラム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ー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ム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系研究科

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 東京工業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

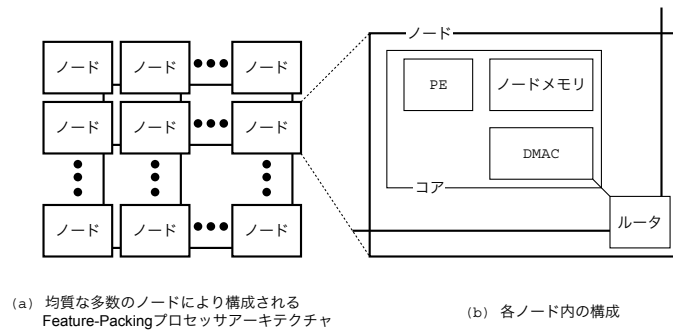


図1 Feature-Packing プロセッサアーキテクチャ概要

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。オーバー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。オーバーレイでは、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、オーバー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 スーパースカラ程度の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-Core⁴⁾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・アーキテク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、プログラム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、プログラムに意識せず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ーレイによって割当てて、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ラ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ムの開発コストを削減することができる。

オーバーレイは、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

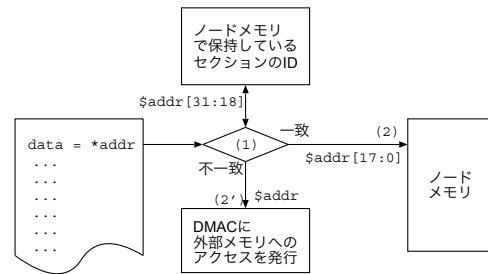


図2 ソフトウェアによる明示的な管理下でのノードメモリアクセス

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ふ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ー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、アクセスしようと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ー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、オーバー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、オーバー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

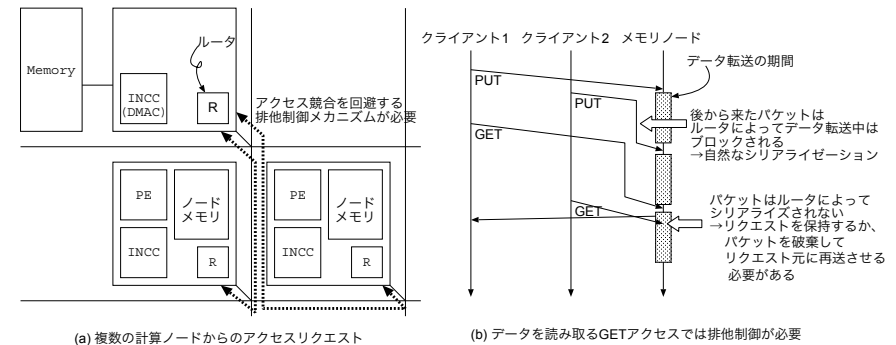


図3 メモリアccessの競合

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。

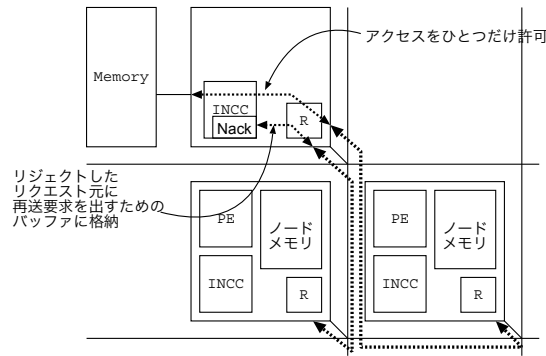


図4 ハードウェアによる排他制御の実現手法

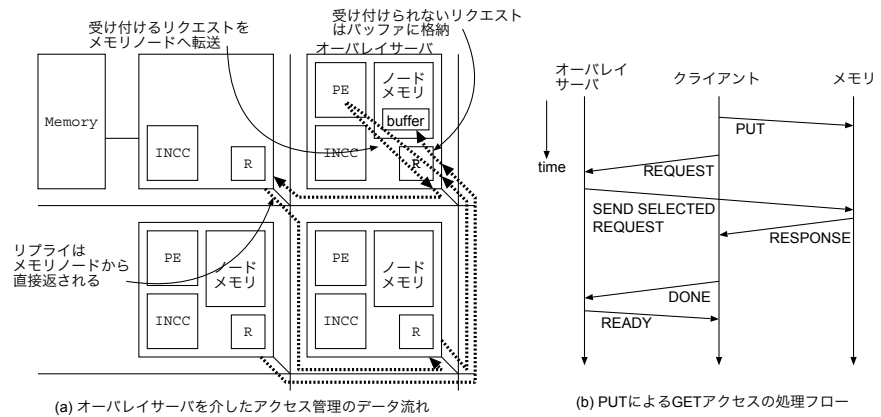


図5 ソフトウェアによる排他制御の実現手法

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、メイン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ーレイサーバと呼ぶ。

図5にオーバー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、オーバー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。オーバーレイサーバでは、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ーレイ手法をメインコアアーキテクチャのシミュレータである SimMc⁴⁾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、オーバー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2⁸Bと制限し、最大2¹²B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ー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ーレイの発行回数は等しい。そのため、この制約の下での評価は、ローカルストレージ2¹⁸Bに対して、2²²Bの実データをオーバーレイして実行するプログラムの評価としてみることが出来る。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。

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、オーバー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ー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、オーバー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、オーバー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ー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

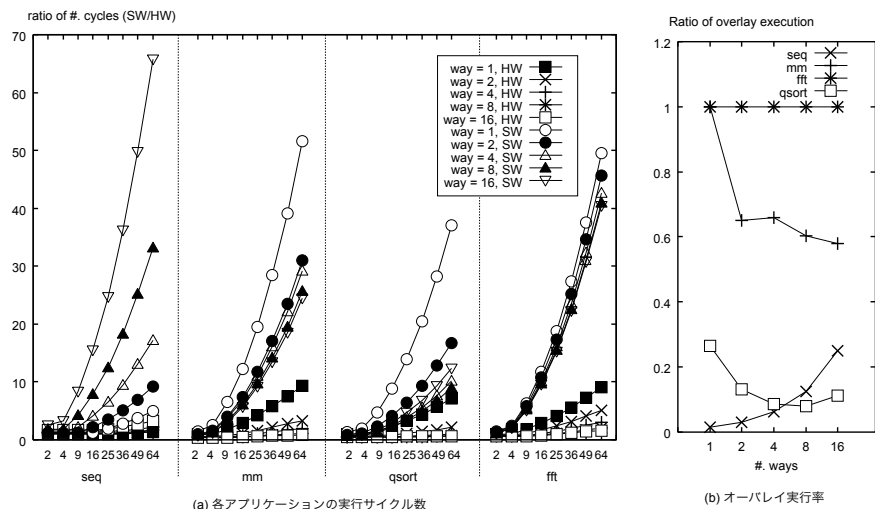


図 6 二つの実装方法による実行サイクル数の比とオーバーレイ実行率

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、プロセッサ全体が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、オーバー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、今行ったようなオーバーレイ領域の分割や、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ー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ー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も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 Machine⁸⁾ のメモリ管理手法として Maps⁹⁾ 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 Feature-Packing プロセッサアーキテクチャを対象としたオーバー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。実装に必要な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

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、オーバーレイ対象領域へのメモリアクセス中にオーバーレイを実行する必要がある回数を全アクセス数に対する割合で示す。オーバー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ー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・オーバー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 XC3S500E⁵⁾ を対象に ISE WebPack 11.3⁶⁾ 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

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、オーバー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ー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の実現に必要な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、オーバー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を考慮した静的、動的な最適化によるオーバー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、今回はオーバー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) Michael Bedford Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpfen, 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 WebPack Design Software. <http://www.xilinx.com/tools/webpack.htm>.
- 7) 高前田伸也, 渡邊伸平, 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) John H. Kelm, Daniel R. Johnson, Matthew R. Johnson, Neal C. Crago, William Tuohy, Aqeel Mahesri, Steven S. Lumetta, Matthew I. Frank, and Sanjay J. 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) John H. Kelm, Daniel R. Johnson, Steven S. Lumetta, Matthew I. Frank, and Sanjay J. 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.