計算機アーキテクチャ 134−30
（1999. 8. 4）

ポインタ指向アプリケーションにおけるSCALTの評価

三竹 大輔* 清水 尚彦** 松原ルシア***
*東海大学大学院工学研究科 ** ***東海大学工学部

概要

近年、急激にプロセッサ性能が向上する中で、メモリとの性能格差が問題になっており、プロセッサの性能を十分に引き出すためにメモリレンテンシを隠蔽する技術が重要となっている。科学技術計算分野における配列指向アプリケーションでは、ベクトルコンピュータによる対応が主である。しかし、データベース分野等のポインタ指向アプリケーションでは、構成データの特殊性からベクトルコンピュータによる対応は望めない。我々の提案するレイテンシ隠蔽アーキテクチャSCALTは、配列指向アプリケーションにおいて、ソフトウェア制御プリフェッチを用い、シミュレーションによる有効性が確認されつつある。そこで、本報告では、ポインタ指向アプリケーションに対しても有効性を評価する。

Evaluation of SCALT on Pointer-Based Applications

Daisuke MITAKE*, Naohiko SHIMIZU**, Lucia Matuhara***
*Graduate School of Eng.,Toukai Univ.,** ***Faculty of Eng.,Toukai Univ.

Abstract

In recent years, processor performance is increasing with rapidity. Therefore, the gap between processor and memory speeds have appear, and latency tolerate techniques have the importance. Vector computer has overcome memory latency in array-based numeric applications. But, it can't overcome memory latency in pointer-based applications because of particular processing data structure. Our proposal architecture SCALT has overcome memory latency using software-controlled data prefetching in simulations of array-based numeric code. In this paper, we evaluate effect of the SCALT on pointer-based applications.

Keywords: Latency Tolerant, Software Prefetch, Decoupled Architecture, Pointer-Based Applications, Recurcive Data Structure

1 はじめに

近年の急激なプロセッサ性能の向上により、プロセッサとメモリ間の性能格差が拡がっている。このため、プロセッサの性能を十分に活かすために、メモリレイテンシを隠蔽するための技術がますます重要視されている。

科学技術計算分野における配列指向アプリケーションに対しては、連続的なメモリアクセスと演算によるベクトルコンピュータなどのアプローチによりレンテンシを隠蔽することが試みられている。しかし、より一般性のあるデータベースやグラフィックスアプリケーション分野などのポインタ指向アプリケーションにおけるメモリレイテンシ隠蔽[1]についてはあまり研究されていない。ポインタ指向アプリケーションでは、RDS(Recursive Data Structure) と呼ばれる再帰的なデータ構造がよく用いられる。RDSは、リンクリスト、木、グラフなどから成り、例として、図1のような木構造のものが上げられる。図において、構造体 tree は最下部の葉に至るまで再帰的に用いられており、そのため、より下の枝にアクセスするためには、順番にポインタの指し示すデータをアクセスして行かなければならず、アクセスするまで次の枝がどこにあるのかわからないという問題がある。

つまり、ポインタ指向アプリケーションでは、明

示的なデータ配置に対して連続的に処理を行なうベクトルコンピュータとは根本的にデータへのアクセス方法が異る。そのため、ベクトルコンピュータによるアプローチは望めない。

我々は、メモリレイテンシの隠蔽を目的にしたアーキテクチャSCALT[2),3),4),5)]を提案し、科学技術計算分野におけるシミュレーションによりその有効性を確認しつつある。SCALTは、専用のバッファに対してソフトウェア制御によってデータを適宜プリフェッチする事でメモリレンテンシを隠蔽する非常に単純で実装の容易なアーキテクチャである。そのため、ポインタ指向アプリケーションにおけるRDSに対しても柔軟に対応が可能であり有効性を見込めると考えられる。

本報告では、ポインタ指向アプリケーションでSCALTを用いた場合の性能を見ていく。

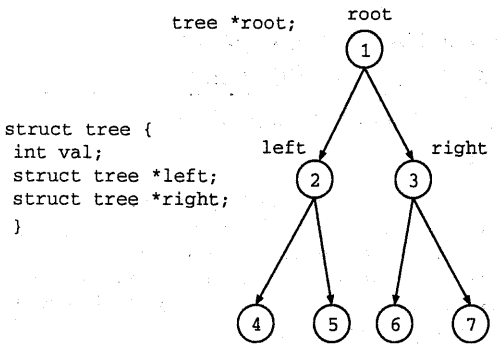

図 1: Pointer Tree

2 SCALTアーキテクチャ

提案するアーキテクチャの構成は、図2となっている。

以下、簡単にアーキテクチャについて述べる。

1. プロセッサ固有の物理アドレスにマッピングされたバッファメモリを持つ
2. バッファメモリは、エントリと呼ぶ小空間に区分し、各々番号付けする
3. バッファーメモリ間のデータ転送は、エントリを指定して行う
4. バッファチェック命令によりデータ到着を判別する

1に関して、バッファはキャッシュと同階層に配置する。バッファの構成素子は、既存技術にあるキャッシュと同様のものを用いる。
2に関して、各エントリにはそれぞれ有効ビット(V)を設ける。エントリへのアクセスは、通常のロード/ストア命令によって行う。
3に関して、バッファへのデータ転送は、バッファエントリ数分発行できる。バッファへのデータ転送が指示されると有効ビットは解除される。データ転送が終了すると有効ビットはセットされる。有効ビットが解除されているエントリへのload命令は、プロセッサをストールさせる。

アーキテクチャ決定における考慮点を以下に述べる。

load/storeアキテクチャとの親和性 SCALTは、従来型のload/storeアーキテクチャに対して数命令の追加で実現できる。

1回のアドレス変換でエントリ単位のデータ移動するため、従来型RISCのアドレス計算のパイプラインをそのまま利用できる。

キャッシュリフィル・パージ論理とデータパスを共有できる。つまり、キャッシュとバッファは、両方共load/store命令によってアクセスする。双方のアクセスの違いは、対象となるアドレスが違うだけである。そのため、TLBからキャッシュのクリティカルパスには余分な論理は入らない。

アーキテクチャとバッファ容量を独立 アウトオブオーダー実行や明示的なレジスタプリフェッチでは、相対レイテンシの増大に比例してレジスタ数が必要になる。(アウトオブオーダーでは、複雑度が増す。レジスタプリフェッチでは、アーキテクチャの変更が必要。)

SCALTでは、アーキテクチャに独立なバッファエントリ数を増やす事で対応するため、実装が困難になる事は無い。また、エントリ番号は、レジスタにより指定するので、アーキテクチャの変更なしにエントリ数の増加を可能とする。

バッファのソフトウェアコンテキスト化 ソフトウェアによる読み出しデータの保有を前提とする事で、ソフトの制御 (排他制御等) をやりやすくする。

SMP 時の他プロセッサからのパージが減る事で性能が向上する。

同一世代のアーキテクチャで幅広い性能レンジを実現 エントリ数とエントリの大きさを変更する事で、アーキテクチャを変更せずに、「1 命令当たりの転送データ長を増やす事」と「増大する相対レイテンシに対応する事」が可能。

$\Longrightarrow$ デスクトップ WS からメモリインタリーブの多いスーパーコンピュータクラスのマシンまで、同一アーキテクチャで対応可能。さらに、命令当たりのデータ転送スループットの増大もエントリのサイズ変更により同一アーキテクチャで実現できる。

NUMA との親和性 バッファは、コヒーレンシ制御不要であり、NUMA を実現しやすい。また、NUMA で問題となるレイテンシの変動にもソフトウェアでの対応が容易である。

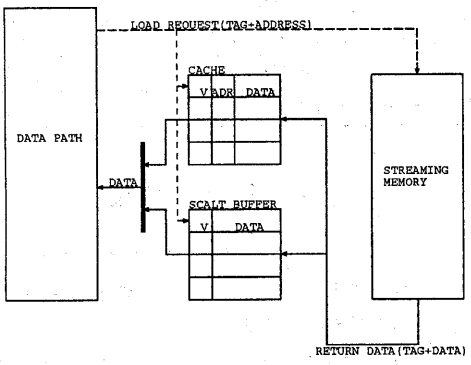

図 2: An example load path configuration with SCALT

3 RDS アクセス

前述の図 1 のポインタツリーは、各構造体が値 (val) と次の枝へのポインタ (*left,*right) を持つ (図 3)。これは、treeadd と呼ばれる性能評価プログラムで用いられるポインタツリーであり、treeadd では、それぞれの枝の値 (val) の総和を求める。ポインタを通して次のデータ構造にアクセスするため、枝の先のデータにアクセスするためには、順番にポインタの値を追って行く必要がある。

アウトスタンディングロード数を 2 とする一般的な RISC プロセッサの場合、アーキテクチャ上でアクセスできるのは、同時に 2 つの枝だけである。このため、より深い階層に行くにつれて、全ての枝を追う事は、時間的なロスが大きくなる。SCALT では、図 1 において、深さが同じ階層の枝全てをプリフェッチするために幅優先でアクセスを行う。プリフェッチのリクエストは、SCALT バッファのエントリ数分発行できるので、より深い階層に行くことで、アクセスできる枝が増え、プリフェッチできる枝の数が増す。図 3 において、初めは、枝 1 に対してプリフェッチを行い到着したデータにより次の枝である 2、3 のアドレスがわかる。そして、そのアドレスにより枝 4、5、6、7 をプリフェッチでくる。このように、プリフェッチする枝の数が 2 倍で増えていく。プリフェッチしたデータ、つまり、アドレスがまた、SCALT は、エントリ数を増す事でより大きな木構造にも対応できる。

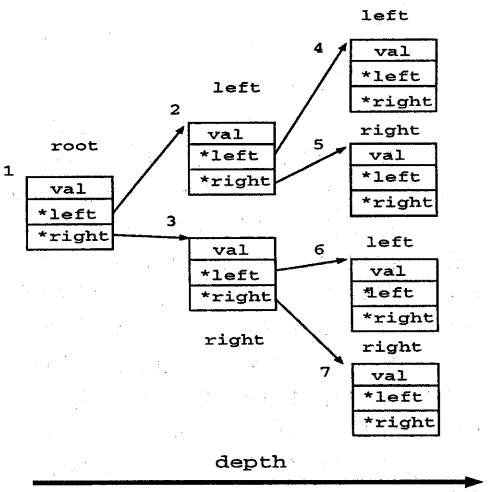

図 3: Structure

4 性能検証

ハードウェアの性能検証におけるシミュレーションは、C言語により記述されたイベント・ドリブン・シミュレータを用いる。評価の対象として、treeaddと呼ばれるプログラムを用いる。treeaddは、再帰的に用いられる構造体がもつ個々の値の総和を求めるものである。つまり、全ての枝、葉に至るまでアクセスする必要があり、今回、その完了時間を評価指標にする。
比較対象としては、一般的なRISCプロセッサでアウトスタンディングロード数が各々2、4のものを想定した。

シミュレーションモデルは、図4を用いる。プロセッサ台数は、1台とし、主記憶装置とプロセッサはクロスバ接続とする。

メモリバンク数は、1,2,4,8,16,32と増加させ、メモリバンク数を変化した時における木の深さ7(葉の数は、64)でのプログラム完了時間を測定する。また、メモリバンク数を32バンク固定とし、木の深さ(level)を2,3,4,5,6,7,8と変化した時のプログラム完了時間を測定する。シミュレーションに用いる代表的なパラメータは、Table1のようになっている。

シミュレーションではプロセッサの動作をイベントドリブンに記述した。

シミュレーションモデルでメモリからSCALTバッファへのデータ転送命令の流れを追ってみる。まず、INST MEMより、メモリからSCALTバッファへのデータ転送命令がフェッチされるとする。REQ QUEが空いているかチェックされ空いていれば確保され、空いていない場合は、空くまで待つ事になる。次に、LOAD QUEが空いているか同様にチェックされる。空いていれば、確保し、REQ QUEを解放する。同様に、QUEをチェックし確保する。MEMORY BANKが空いているかチェックし確保できたならば、QUEを解放する。MEMORY BANKを解放し、LOAD QUEに入る。LOAD BUSをチェックし、確保できたならば、LOAD QUEを解放する。そして、SCALT BUFFERのエントリにデータが転送される。

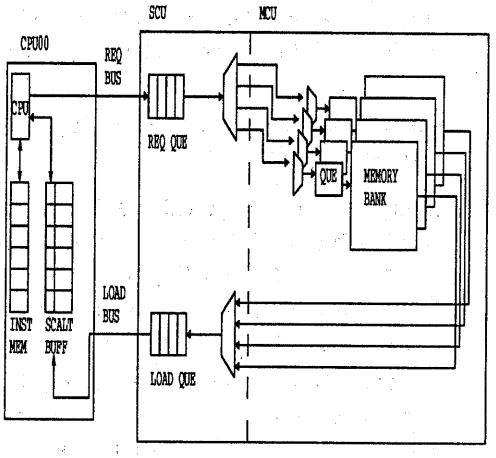

図 4: Simulation model

表 1: Simulation Parameter

CPU Cycle Time	5nS
Num of Memory Bank	L
Data transmission time between LSI	5nS
SC Request Pitch	10nS
Length of Memory Bank	16B
Length of Buffer Entry	32B
Access Cycle of Memory	250nS
Memory Access Time	250nS
Num of Buffer Entry	64

5 シミュレーション結果

図5は、メモリバンク数を変化したときのプログラム完了時間である。このときの木の深さは、level7(葉の数64)である。

バンク数が極端に少ないときは、バンクコンフリクトにより性能低下が表われている。このシミュレーションでは、バンク数は、全体的に8バンクで性能が十分に得られる事がわかる。SCALTは、アウトスタンディングロード数2のRISCプロセッサに対して約20%,4のRISCプロセッサに対して約60%完了時間を削減している。図6は、木の階層の深さを変化したときのプログラム完了時間である。SCALTは、一般的なRISCよりも高い性能を出すことに成功している。level7以降、一般的なRISC

では完了時間が急激に上がっているため、SCALTとRISC間の性能差はもっと開いていくと考えられる。

6 まとめ

本稿では、SCALTのアーキテクチャならびに、ポインタ指向アーキテクチャに対してSCALTがどのように対応するかを述べた。

我々のアーキテクチャは、RDSを含むtreeaddのシミュレーションにおいて従来のＲＩＳＣプロセッサを上まわる性能を得る事に成功した。

特に、木の深さが深くなるほど我々のアーキテクチャは有効性が高くなる事が予測される。

しかしながら、バンクコンフリクトなどの性能低下要因があるため性能を得るためにはある程度のメモリバンク数が必要とされる事がわかった。

今回は、最も単純なRDS横断パターンをもつtreeaddに対してシミュレーションを行なった。そのため、横断パターンが複雑になった場合についての更なる評価が必要と考えられる。

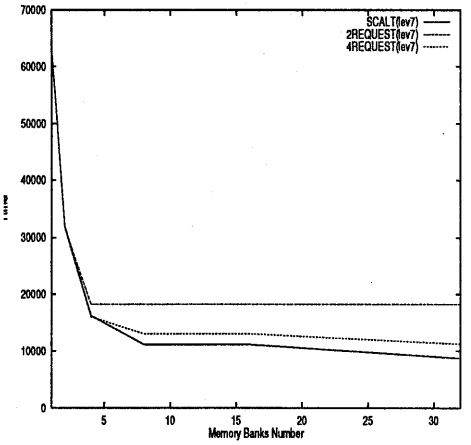

図 5:

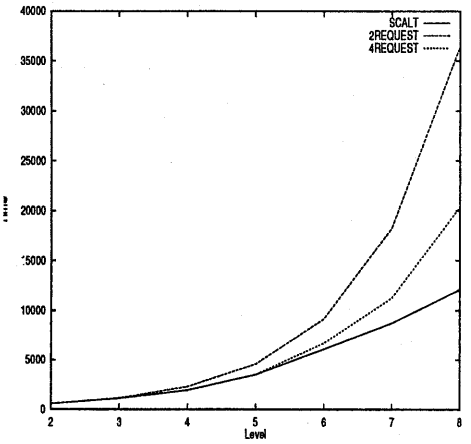

図 6:

参考文献

1) Chi-Keung Luk,Todd C.Mowry,"Automatic Compiler-Inserted Prefetching for Pointer-Based Applications"IEEE TRANSACTIONS ON COMPUTERS, VOL.48, NO.2,FEBRUARY 1999

2) 清水" スケーラブル・レイテンシ・トレラント・アーキテクチャ" IPSJ Sig Notes,Vol.97,No.21,1997

3) 清水"SCALT ／ SMP の性能評価"IPSJ Sig Notes,Vol.97,No.76,1997

4) 三竹、清水" 予測できないメモリレイテンシにソフトウェアで対応するためのハードウェア機構"1998 年 3 月信学会総合大会

5) 三竹、清水" バッファチェック命令により変動するレイテンシに対応するプロセッサの性能検討"1998 年 10 月情報処理学会全国大会