ハイパフォーマンス
コンピューティング 74−15
（1998. 12. 11）

変動するメモリレイテンシに対応するプロセッサ構成と評価

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

分散共有メモリマルチプロセッサなどの技術の普及により、ソフトウェアから予測する事が難しいメモリレイテンシの変動要因が浮き彫りになっている。ハードウェアによる対応方法として、マルチスレッドプロセッサによるスレッド切替えが考案されてきたが、単一プログラムの性能向上には一般的でない。著者らが検討を進めている SCALT は、ソフトウェアにキャッシュメモリ相当の資源としてバッファを開放し、バッファへのデータ到着の有無をチェックする専用命令により、レイテンシの変動に対応しようとするものである。本報告では、SCALT の構成を説明し、SMP 構成とした場合の性能をイベントドリブンシミュレータにより評価する。

Processor architecture and evaluation which correspond to deviation of the memory latency

Daisuke Mitake* Naohiko Shimizu**
*Graduate School of Eng., Toukai Univ. **Faculty of Eng.,Toukai Univ.

The deviation of the memory latency is hard to predict for software especially on the SMP or NUMA systems. As the correspondence method by hardware, the multi-thread processor has been devised. However, it is not general to improve a performance in a single program. We have been proposed SCALT which has a buffer as a software context. For the deviation of a latency problem,a instruction which checks existence of the data arrival to a buffer has been proposed. This report describes the SCALT and evaluatation results the performance of SCALT with buffer check instruction with an event-driven simulator for SMP system.

1 はじめに

近年、数値計算分野において分散共有メモリマルチプロセッサなどの技術が普及し、ソフトウェアから予測する事が難しいメモリレイテンシの変動要因が浮き彫りになっている。これにより、ますますメモリレイテンシを隠蔽する技術が重要となっている。

ハードウェアによる対応方法としては、マルチスレッドプロセッサにおいて、キャッシュミス時に別のスレッドへ切替える事によりメモリレイテンシを隠蔽する事が考案[1]されてきたが、異なるスレッド間でプログラムを共有する事は難しく、一般的な単一プログラムの性能向上を期待する向きには至っていない。

また、リクエストを発行するユニットと演算を行うユニットを分離するデカップルドアーキテクチャ[2]によりメモリレイテンシを隠蔽することが試みられてきた。デカップルドアーキテクチャからの派生に、キャッシュミスによるペナルティを後続命令の実行を先に行う事で隠蔽するアウトオブオーダー実行による対応があるが、後続命令をキューとなるレジスタに納めておく事により可能となるランダムアクセスは、大きなレイテンシに対応するためにはレジスタ資源の増大を意味し、プロセッサの性能向上に対して実装を困難にする。

データプリフェッチによる方法としては、ハードウェアにより動的に対応する方法がある。一例として上げる lookahead PC[3] は、一定のストライドアクセスパターンを発見するために履歴バッファを用い、分岐予測を使ってプログラムカウンタの先へ投機的にプログラムを進ませ、履歴内に一致したストライドエントリを見つけたとき、プリフェッチを発行するものである。

データプリフェッチには、ソフトウェアによる静的な解析により対応する方法もある。例えば、コンパイル時にプログラムの内部ループにおいてメモリレイテンシよりも先のループで使用するデータをプリフェッチするようにプリフェッチ命令を挿入するようなことである。これは、大量のレジスタセットをレジスタウィンドウにより切り替えて使う疑似ベクトル処理[4]でも見られる

これら、データプリフェッチに関する事は、mowry の論文[5]がより詳しい。

我々は、データプリフェッチの中でもソフトウェアによる制御に着目し、ソフトウェアに対してキャッシュメモリ相当の資源としてバッファを開放し、バッファへのデータ到着の有無をチェックする専用命令により、レイテンシの変動に対応する SCALT[6][7][8][9] を考案している。

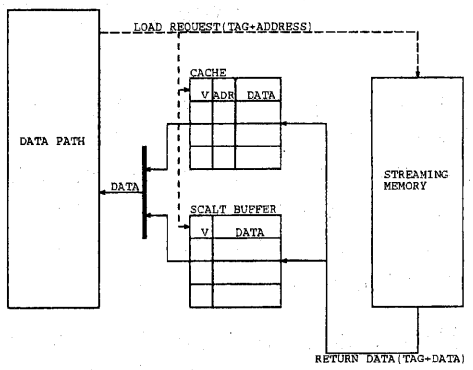

図 1: An example load path configuration with S-CALT

2 SCALT

SCALT は、メモリレイテンシの隠蔽のためにプロセッサ実装の容易性とシステムリソースにスケーラブルな性能を目的としたアーキテクチャである。その構成は、図 1となっている。

図において、データバッファ(SCALT バッファ)としてキャッシュと同一のメモリ素子をキャッシュと並行に配置する。バッファは、ソフトウェアコンテキストであり、キャッシュにあるメモリとのコンシステンシは要求されない。また、特別な仮想アドレスにマッピングし、通常のロード／ストア命令によってアクセスする。この時、バッファ領域はエントリ番号をタグとして付けることにより管理する。プロセッサは、主記憶からの戻りデータを区別するためにタグをメモリ要求に付加して発行し、主記憶からの戻りデータは、タグの値によってバッファの対応エントリに入る。バッファへの転送要求 (ロード) またはバッファからの転送要求 (ストア) は、主記憶のアドレス、バッファエントリ番号を指定して発行する。バッファへの転送要求が発行されると、バッファの対応するエントリの有効ビットが解除され、転送

が終了すると有効ビットがセットされる。有効ビットが解除されているエントリへのロード命令は、プロセッサをストールさせる。バッファへのデータの到着の有無をチェックするため、有効ビットの値を返す命令としてSCALTバッファチェック命令を導入した。バッファを使用する前にバッファの有効ビットを調べる事でプログラムレベルでレイテンシの変動に対応できる。

3 SCALTバッファチェック命令

SCALTバッファチェック命令により、どのように変動するレイテンシに対応するかを図2にフローチャートとして示す。

レイテンシの変動によりデータが未到着であるバッファエントリが生じる。バッファチェック命令によりデータが未到着と判別されるエントリに対しては、演算処理を後回しに、演算に必要なデータが揃ったものから先に演算処理を行う。未到着のエントリは、未到着リストに入れて管理しデータ到着を再度チェックする。プロセッサが、未到着リストに載ったエントリに対して処理を行うころにはデータは到着していると考えられ、データの待ち合わせの時間を隠蔽する事になる。

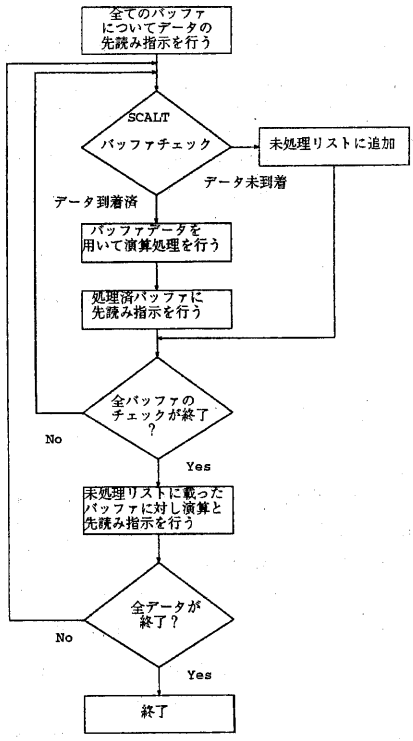

図 2: バッファチェック命令を用いたフローチャート

4 SCALTによるデータの流れ

図3は、$a[i]+b[i]$ という簡単な演算の一般的なRISCアーキテクチャのアセンブルコードとSCALTによるアセンブルコードの例である。また、SCALTでのアセンブルコードに対応するデータの流れを図4に示す。

B.FETCHは、SCALT BUFFERの対応するエントリへ主記憶からデータをフェッチする命令であり、一回のフェッチで、キャッシュライン長と同等のデータ量をフェッチする。B.CHECKは、指定されたバッファエントリの有効ビットをチェックし、その値を返す命令である。B.STOREは、バッファエントリのデータを主記憶へストアする命令である。図3では、バッファエントリ0と1に対するフェッチのみを記述しているが、実際には、もっと多くのエントリに出す事になる。バッファに到着したデータは、仮想的にマッピングされたアドレスから通常のload命令によりレジスタに読み出され、演算に用いられる。演算結果は、通常のstore命令により、仮想的なアドレスにマッピングされたバッファエントリへストアされる。store命令により、バッファエントリが満たされると、B.STORE命令によりエントリのデータを主記憶へストアする。

エントリに対してデータが到着していない時は、データ到着を待つ事になり、処理が滞る。ここで、B.CHECKによりデータ到着を調べ、到着していない時は、次のエントリ、例えば、エントリ2、3、…とデータ到着を調べていき、到着済のエントリに対してだけ演算を行い、未到着のデータを後に回し処理を進める。

5 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・

```
                    a[i]+b[i]

RISC                SCALT                SCALT(B.CHECK)

load  r1  #1000     B.FETCH  0 #1000     B.FETCH  0 #1000
load  r2  #2000     B.FETCH  1 #2000     B.FETCH  1 #2000
add   r3 r1 r2
store r3  #3000       load  r1  #4000      B.CHECK  0
                      load  r2  #4032      B.CHECK  1
                      add   r3 r1 r2
                      store  r3  #4320       load  r1  #4000
                            ...              load  r2  #4032
                                             add   r3 r1 r2
                      B.STORE  10 #3000      store  r3  #4320
                                                ...
                                             B.STORE  10 #3000
```

図 3: SCALT 命令によるアセンブルコード例

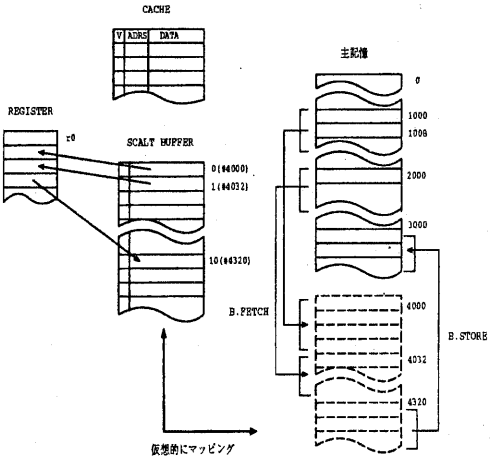

図 4: データの流れ

シミュレータを用いる。評価の対象としては、リバモアループを用い、アドレス生成パターンはハンドコンパイルにより決定する。

シミュレーションモデルは、図5を用いる。プロセッサは、1サイクル1命令実行を行い、プロセッサ台数は、パラメータとして1台から16台までを用いる。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,…,256 と増加させ、メモリバンク数を変化したときのループ実行回数と平均バンク利用率の変化を見る。シミュレーションに用いる代表的なパラメータは、表1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、リバモアループを一定時間繰り返し実行させる。また、SMP の各プロセッサは確率分布に従った時間遅れを伴ってループを実行する。SMP のメモリのバンクコンフリクトは頻繁に起こり、SCALT フェッチのレイテンシは動的に変更される。

表 1: Simulation Parameter

CPU サイクルタイム	5nS
メモリバンク数	M
LSI 間データ転送時間	5nS
SC リクエストピッチ	10nS
メモリバンク幅	16B
バッファエントリ長	32B
メモリサイクルタイム	250nS
メモリアクセス時間	N nS
バッファエントリ数	32

6 シミュレーション結果

リバモアカーネル1に対してシミュレーションを行った。シミュレーション結果は、図6から図9となっており、SCALT において B.CHCEK を用いるケースと用いないケースを比較した。

CPU 台数1のとき、バンク数64において、バッファチェック命令を用いたケースの方が用いないケースよりも性能が悪くなり、バンク数を増加させても性能の向上はみられなくなっている。同時に、バンク利用率のグラフにおいても利用率の低下が見られる。また、CPU 台数4においてもバッファチェック命令の性能向上を阻害する要因が出ている。これは、バンク数の増加に伴うバンクコンフリクトの減少により、バッファチェック命令の効果が現われなくなり、また、バッファチェック命令を用いた場合には、ループ内で実行する命令列が約2倍になるのだが、シミュレーションにおける命令列では、実際の演算部分の時間を考慮せずメモリ部分の実行時間のみ考慮している事により、シミュレーション実行時間内でバッファチェック命令に関する命令数の増加による余分な実行時間が際だってしまい、1サイクル1命令の CPU では、性能限界が来ていることを表している。しかし、実際のアプリケーションでは、演算の時間

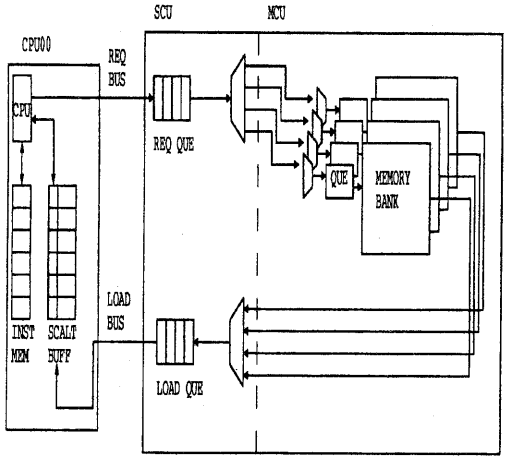

図 5: シミュレーションモデル

が必要となるので、バッファチェック命令による余分な時間の増加は数パーセント以内であり、バッファチェック命令の追加により性能を束縛するような事にはならない。

CPU台数の増加によりバンクコンフリクトが増加することで、8台以降では、メモリバンク数の増加に対しても、バッファチェック命令無しのケースに対して、約20%程の性能向上が見られる。

7 まとめ

本報告では、バンクコンフリクトなどによりレイテンシが変動するケースに対し、我々の考案するアーキテクチャがどのように対応するかを述べ、また、シミュレーションによりその性能を評価した。我々のアーキテクチャは、レイテンシ変動に対し、幅広いメモリバンク数においても対応が期待できる事がわかった。

今後は、様々な対象に対してシュミレーションを行い、演算部分を含めたシミュレーション、コンパイラの開発に対しても取り組んでいく。

参考文献

[1] Richard J. Eickemeyer etc. "Evaluation of Multithreaded Processors and Thread-Switch Policies"ISHPC'97,1997

[2] Harris,Toham,"The Scalability of Decoupled Multiprocessors"SHPCC'94,1994

[3] J. -L. Baer etc. "An effective on-chip preloading scheme to reduce data access penalty."In Proceedings of Supercomputing'91 ,1991

[4] Nakazawa,Nakamura etc. "Pseudo Vector Processor based on Register-Windowed Supersalr Pipline",Supercomputing'92,1992

[5] Todd C. Mowry "Tolerating Latency Through Software-Controlled Data Prefetching"

[6] 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

[7] 清水"SCALT ／ SMP の性能評価"IPSJ Sig Notes,Vol.97,No.76,1997

[8] 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

[9] 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

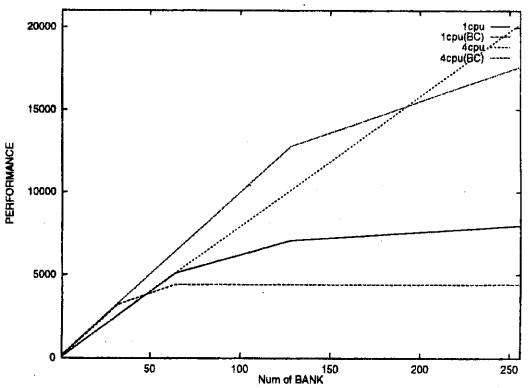

図 6: バンク数変化に対するループ実行回数 (1,4cpu)

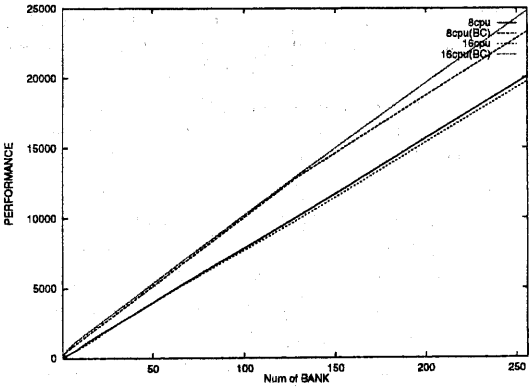

図 7: バンク数変化に対するループ実行回数(8,16cpu)

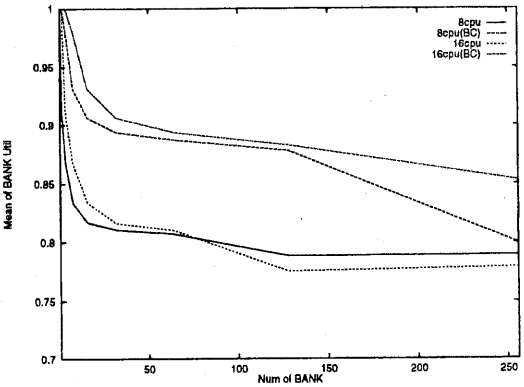

図 9: バンク数変化に対する平均バンク利用率(8,16cpu)

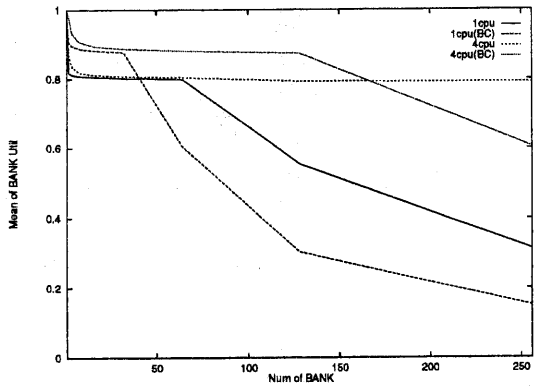

図 8: バンク数変化に対する平均バンク利用率(1,4cpu)