1J-6

キャッシュの利用効率の向上に関する研究

浅見 公輔[†] 倉田 成己[††] 塩谷 亮太[†††] 三輪 忍[††] 五島 正裕[††] 坂井 修一[††]

[†] 東京大学 工学部
[††] 東京大学大学院 情報理工学系研究科
[†††] 名古屋大学大学院 工学研究科

1 背景

プロセッサの性能向上を妨げる原因として，キャッシュで発生する**競合**がある．近年マルチスレッド・プロセッサが普及し，一つのチップ上で同時に動作するスレッド数が増加して高いスループットを実現している一方で，キャッシュ上のスレッド間の競合が性能に大きな影響を及ぼしている．そのため，競合に対処する必要性がますます高くなってきている．

競合の原因として，**ストリーミング・アクセス命令**や**delinquent 命令**が挙げられる．ストリーミング・アクセス命令とは，配列の要素に順番にアクセスを続けるような命令のことであり，delinquent 命令とは，キャッシュ・ミスを頻繁に引き起こす静的な命令のことである．ストリーミング・アクセス命令や delinquent 命令は他の命令よりもワークロードが非常に大きく，キャッシュ・ミスを次々と引き起こしてキャッシュの他の命令のデータを追い出して競合を発生させる．そのため，キャッシュ・ラインのリプレースメント・ポリシを工夫してもストリーミング・アクセス命令や delinquent 命令が引き起こす競合に対処することはできない．加えて，このような命令は次から次へと新しいデータへのアクセスを繰り返すため，フェッチ後のデータの再利用性が低く，キャッシュの利用効率を下げている．

2 既存手法

キャッシュの競合を防ぐ手法として，代表的なものに**キャッシュ・パーティショニング**[1] がある．これは共有キャッシュをコア毎やスレッド毎の占有領域であるパーティションに分割することで競合を防ぐ手法である．従来のキャッシュ・パーティショニングでは，共有キャッシュ上で実行されるスレッド同士のワークロードの違いに着目し，その利害関係に対して最適なキャッシュ・サイズを見つける方向性で研究が行われてきた．しかしすでに述べたように，ストリーミング・アクセス命令や delinquent 命令と，それ以外の命令では，ワークロードに大きな違いがある．そのためスレッド間の競合は解決できても，命令間での競合問題が依然存在している．

また，キャッシュの利用効率を上げる技術として，再利用性がないデータを予測（**デッド・ブロック予測**）して優先的にキャッシュから排除する手法 [2] がある．この手法では，再利用性がないデータのフェッチ時のバイパシングと，すでにフェッチされたデータの排除を行うことができる．しかし，予測が外れた時の性能に対するペナルティが大きく，データに対するアクセス・パターンが変化した時の予測は難しい．

3 提案手法

我々はキャッシュ上における命令間の競合を防ぎ，キャッシュの利用効率を向上させる新しいキャッシュ・マネジメント手法を提案する．命令をグループごとに分け，それぞれに最適なキャッシュ・サイズを割り当てるキャッシュ・パーティショニングを行う．

コード 1 と図 1 を用いて，提案手法を説明する．コード 1 は二次元配列 a に対して 2×2 のアクセスを次々と行う，ストリーミング・アクセス命令群を含んだプログラムである．図 1 は，配列 a の領域と，コード 1 が 1 回のイタレーションでアクセスする領域を表した図である．

コード 1: 配列 a へのストリーミング・アクセス

```
for(y) {
    for(x) {
        load a[y][x]; // 命令 A
        load a[y][x+1]; // 命令 B
        load a[y+1][x]; // 命令 C
        load a[y+1][x+1]; // 命令 D
    }
}
```

A Study on Efficient Cache Utilization
Kousuke Asami[†], Naruki Kurata[††], Ryota Shioya[†††], Shinobu Miwa[††], Masahiro Goshima[††], Syuichi Sakai[††]
[†]Faculty of Engineering, The University of Tokyo
[††]Graduate School of Information Science and Technology, The University of Tokyo
[†††]Graduate School of Engineering, Nagoya University

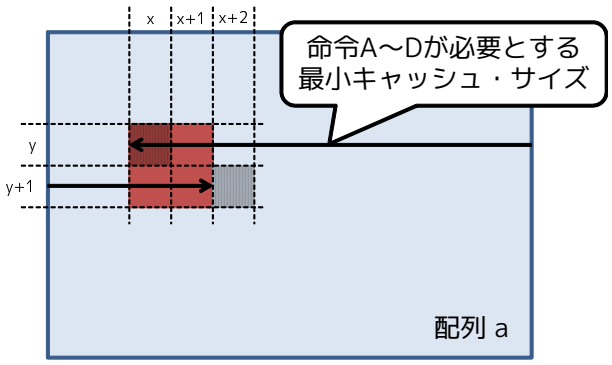

図 1: 配列 a とイタレーション 1 回のアクセス領域

キャッシュ・マネジメントを行わず，LRU なキャッシュ・ラインをリプレースメント対象とする従来のキャッシュでは，コード 1 を実行すると配列 a 全体がキャッシュにフェッチされて，配列 a のサイズだけ，他の命令が使うデータの追い出しが発生し，結果として競合を起こす．

ここで，コード 1 の命令 A ・命令 B ・命令 C ・命令 D を一つの命令群とみなし，この命令群がキャッシュに対してどうアクセスするかを考える．

命令 A がアクセスしたデータ（図 1 上，要素 a[y][x] に該当する網掛部のデータ）は次のイタレーションでは使わず，命令 D が次のイタレーションでアクセスするデータ（図 1 上，要素 a[y+1][x+2] に該当する網掛部のデータ）は次のイタレーションで初めて使われる．つまり命令群が必要とする最小キャッシュ・サイズは，配列 a の y 1 列分 + 配列 a の要素 2 つ分である．結局，命令群が使えるキャッシュ・サイズを y 1 列 + 要素 2 つに制限すれば，命令群が起こす競合を最小限に防ぐことができる．さらに，再利用性のないデータを将来利用するデータでリプレースするため，キャッシュの利用効率も向上させることができる．

このように，提案手法では命令 A ・命令 B ・命令 C ・命令 D のような複数の命令を一つの命令群と考え，命令群単位で使用できるキャッシュ・サイズを決めて制限することで，命令間の競合を防ぎ，キャッシュの利用効率を向上させる．我々はこの分割単位とする命令群のことを**命令グループ**と呼ぶ．

提案手法では，まず命令グループを決定する．そして命令グループが使用できるキャッシュ・サイズを決めて，命令グループが使えるキャッシュの領域を制限することでパーティショニングを行う．割り当てられたキャッシュ・サイズが命令グループが必要とする最小キャッシュ・サイズより小さい場合，命令グループは割り当てられたキャッシュ内で自ら競合を起こしてしまう．逆に大きく割り当てると，それだけ他命令の競合を招きやすくなるため，命令グループの必要最小キャッシュ・サイズに割り当てキャッシュ・サイズができるだけ近づくよう，命令グループのキャッシュ・ヒット/ミスを観測して動的にパーティション・サイズを変更する．命令グループが必要とする最小キャッシュ・サイズがキャッシュの全体サイズよりも大きい場合は，キャッシュに対してデータをバイパシングさせることで競合を防ぐ．

従来のキャッシュ・パーティショニングは分割単位がコア毎やスレッド毎であったため，分割領域数は 2・4 程度であり，キャッシュのウェイ方向に分割する手法が多かった．しかし，命令グループを分割単位とする提案手法では，分割領域数は従来のキャッシュ・パーティショニングより多い．ウェイ方向にキャッシュ・パーティショニングを行おうとすると，分割領域数に比べてキャッシュのウェイ数が少ない場合，分割された領域間で競合が発生してしまう．これを避けるため，提案手法では**セット方向へのキャッシュ・パーティショニング**を行う．セット方向にキャッシュを分割することで，分割された領域間での競合を防ぎ，かつウェイ方向への分割よりも高い精度でのパーティション・サイズの決定を可能とする．

4 まとめ

本稿では，命令間の競合を解決し，キャッシュの利用効率を向上させる手法として，命令グループごとのキャッシュ・パーティショニングを提案した．

今後は，提案手法を実現する構成を考えた後，その実装と評価を進める予定である．

参考文献

[1] G. E. Suh, S. Devadas, and L. Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, pp. 117–128, 2002.

[2] A-C. Lai, C. Fide, and B. Falsafi. Dead-block prediction & dead-block correlating prefetchers. In *Proceedings of the 28th annual international symposium on Computer architecture*, pp. 144–154, 2001.