B-024

x64プロセッサ向けメモリアクセス最適化技術

Memory Access Optimization Techniques for x64 Processor

高畠　志泰†　伊藤　信一‡　橋本　博幸†　本川　敬子†
Motoyasu Takabatake　Shinichi Itou　Hiroyuki Hashimoto　Keiko Motokawa

1. はじめに

科学技術計算の分野ではAMD社やIntel社のx86プロセッサを64bitに拡張したプロセッサ(x64)を使ったクラスタシステムが普及している。また、このような分野では大規模データを扱うため、プログラムの高速化においてはメモリアクセスの最適化が重要となる。

メモリアクセスでは、メモリレイテンシを隠蔽する手段としてデータプリフェッチが有効である。データプリフェッチとは、利用が見込まれているデータをあらかじめキャッシュメモリへ入れておく方法であり、ハードウェアプリフェッチとソフトウェアプリフェッチがある[3]。

大規模なデータをメモリへ書き込む場合は、データをキャッシュメモリへ入れないで、メインメモリへ書き込むストリームストアが有効である。ストリームストアとは、メインメモリへデータを書き込む際に、書き込み用のバッファにデータを貯め、バッファからメインメモリへの書き込みを一括処理することでストア処理を高速にする方法である。

本稿では、x64向けのプリフェッチやストリームストアを使った最適化技術について述べる。

2. x64のメモリアクセス種別

2.1 プリフェッチ

x64にはハードウェアが規則的なデータアクセスを検出し自動的にキャッシュへデータを移動させるハードウェアプリフェッチ機構が備わっている[1][2]。また、プリフェッチ命令には、命令仕様としてL1だけでなくL2やL3キャッシュへ入れるような命令や、再利用されないデータを扱う非テンポラルな命令がある[1][2]。しかし、実際には、仕様どおりに実装されていない命令があるので命令の利用には注意が必要である。

2.2 ストリームストア

ストリームストアは、連続したアドレスに大規模データを書き込むような場合に有効である。これは、非テンポラルストア命令により、近い将来に再利用されないようなデータを、図1のようにキャッシュに入れずに、書き込み用のバッファを介してメインメモリへ書き込む。また、ライトコンバイニング機能により、バッファで複数のデータを貯め、メインメモリへの書き込みを一括に処理することで高速化が可能になる。また、非テンポラルストア命令は、キャッシュを介さないので、キャッシュ汚染も回避できる。しかし、非テンポラルストア命令はSSE(Streaming SIMD Extensions)命令のため利用可能なレジスタが決まっており、プログラム上の配列変数の型によっては非テンポラルストア命令が利用できない場合もある[1][2]。

† (株) 日立製作所 システム開発研究所,
Hitachi, Ltd. Systems Development Laboratory.
‡ (株) 日立製作所 ソフトウェア事業部,
Hitachi, Ltd. Software Division.

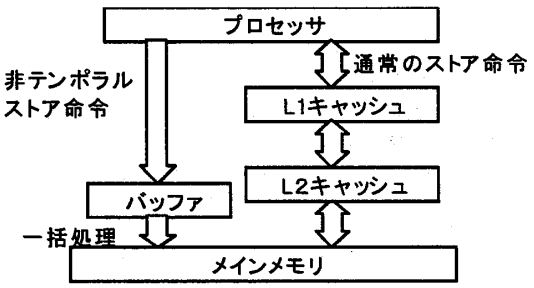

図１：ストリームストアの概要

3. メモリアクセス最適化

3.1 プリフェッチを使った最適化

x64では、ハードウェアプリフェッチにより自動的にプリフェッチが行えるが、詳細な振る舞いが不明なため、本稿ではコンパイラによるプリフェッチ命令を挿入する方式を選択する。コンパイラがデータアクセスパターンを解析することにより、プリフェッチ命令が実行されてから、実際にデータを利用するまでの時間(サイクル)の見積りができるため、効率的なプリフェッチを行えるようになる。また、プリフェッチ命令には、仕様どおり実装されていないものがあるので、L1キャッシュを対象とした命令のみを使うことした。

3.2 ストリームストアを使った最適化

ストリームストア化は、コンパイラがデータ書き込みのアクセスパターンを解析することにより、連続アドレスへのデータの書き込みに非テンポラルストア命令を使うように変換する。非連続アクセスやアクセスパターンが不明の場合は通常のストア命令を使用する。また、データアクセスが書き込みのみであるかも解析する。データアクセスが書き込みのみでなく、読み出しもある場合は、データをキャッシュに置いておく方がよいので、通常のストア命令を使用する。

ストリームストアは大規模データを扱う場合に効果が高いが、そのデータサイズの閾値は、L2キャッシュのサイズの半分とした。L2キャッシュのサイズの半分を超える場合に非テンポラルストア命令を使用する。L2キャッシュの半分以上のデータサイズのものをL2キャッシュに書き込まないようにし、他の必要となるデータがキャッシュから追い出されないようにするためである。コンパイラでは、ストアするデータサイズが静的に解析できれば、非テンポラルストア命令の使用の有無を決める。静的にデータサイズが分からない場合は、実行時に判定を行う。ストア処理の実行前にデータサイズをチェックし、L2キャッシュサイズの半分以上の場合は、非テンポラルストア命令を実行し、

L2 キャッシュサイズの半分より小さい場合は、通常のストア命令を実行するようなコードを生成する。

また、非テンポラルストアはキャッシュにデータを書き込まないのでストリームストア化を行う場合は、ストアする配列に対してプリフェッチ命令を使わないようにする。

4. 評価と考察

メモリのロード/ストアの性能を測定する STREAM ベンチマークを用い、プリフェッチをよびストリームストアを適用した結果を図 2 に示す。プロセッサは、2.0GHz の Dual-Core の AMD Opteron*(2212)を 2 つ使って 4 並列で実行した。L1 キャッシュはデータ用 64KB と命令用 64KB であり、L2 キャッシュは 1MB である。OS は Linux CentOS 5.1 を使用した。図 2 の PF・noPF はプリフェッチ命令の有無で、SS・noSS はストリームストアの有無である。"noPF/noSS"と"PF/noSS"を比較するとプリフェッチ命令を使うことによりプリフェッチ命令を使わないハードウェアプリフェッチより 3%ほどの性能向上がある。しかし、"noPF/SS"と"PF/SS"を比較するとほとんど差が見られない。

ストリームストアでは、noSS と SS を比較するとプリフェッチ命令の有無に関係なく 50%以上と大きく性能が向上している。STREAM ベンチは大規模な配列データを書き込むプログラムのため高い効果が得られた。

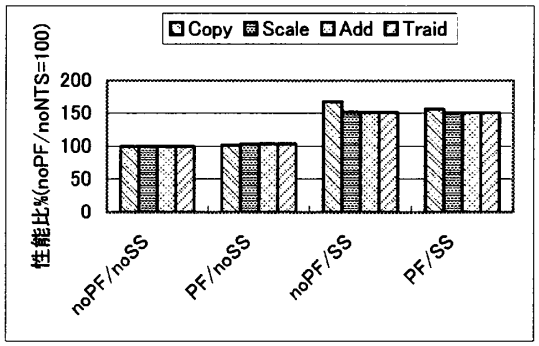

図 2 ：STREAM ベンチによる結果

非連続アクセスに対してストリームストアを適用した場合を図 3 に示す。評価に用いたプログラム例は図 4 であり、配列を 0 に初期化するループである。図 3 の横軸は図 4 のループのステップ数(step)であり、縦軸はステップ数を変えた場合の 16MB のデータを書き込む実行時間である。このプログラムではステップ数を単純に増やしているため、ステップ数が大きいほどストアするデータ量は少なくなっている。しかし、ストリームストアなしでは性能はほとんど変化しない。これは、ストアするデータ量が減っても、キャッシュからメインメモリへのデータ書き込みがキャッシュライン単位のため、メインメモリへの書き込むデータ量が変わらないためと思われる。

ストリームストアありは、ステップ数が 1 の場合は連続アクセスのためストリームストアなしの場合よりも良い結果になっている。しかし、ステップ数が 2 以上のストライドアクセスのような非連続の場合には性能が低下している。これは、連続アクセスであれば、データがライトコンバイニングのバッファに貯められ、メインメモリへの一括書き込みにより高速化されるが、非連続の場合はメインメモリへの一括書き込みが行えず、バッファ内のデータを個別にメインメモリへ書き込んでいるためと思われる。ステップ数が 16 でストリームストアありが良くなっているが、これは、ライトコンバイニングのバッファサイズが 64Byte のため、バッファへのデータの書き込みが 1 つのみになり、バッファからメインメモリへの書き込む回数も少なくなるためである。

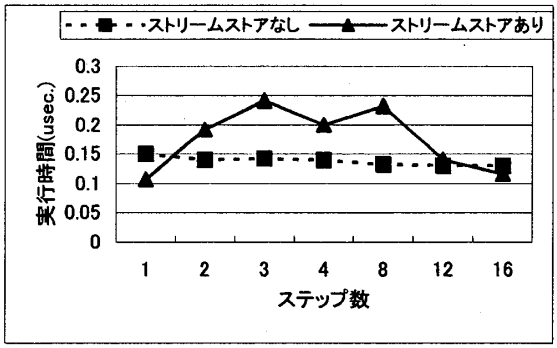

図 3 ：ストライドアクセスによる評価

```
do i=1, n-step-1, step
  a(i) = 0
enddo
```

図 4 ：非連続アクセスのプログラム例

5. おわりに

x64 向けメモリアクセス最適化として、プリフェッチ命令を使った方法と、非テンポラルストア命令を使った方法とを示した。プリフェッチ命令を使った場合は、今回の評価では、ハードウェアプリフェッチと同等の性能であった。非テンポラルストア命令を使ったストリームストアは、連続したアドレスへの書き込みを行う場合に高い効果が得られた。

今後の課題としては、他のプログラムにおいて、プリフェッチ命令の有効性を調べる必要がある。また、x64 でのプリフェッチ命令には非テンポラルなものがある。本稿で扱った L1 キャッシュを対象としたプリフェッチ命令との使い分けなどが今後の課題として挙げられる。

参考文献

[1] AMD, "Software Optimization Guide for AMD Family 10h Processors (Quad-Core)", http://www.amd.com/ us-en/assets/content_type/ white_papers_and_tech_docs/40546.pdf

[2 Intel, "IA-32 インテル アーキテクチャー最適化リファレンス・マニュアル", http://download.intel.com/jp/developer/jpdoc/IA32_Final_i.pdf

[3] 青木 秀貴 他, "SR11000 におけるソフトウェアプリフェッチ手法の評価", 情報処理学会研究報告 2004-ARC-159、pp.109-114, (2004).

*AMD Opteron は、AMD, Inc.の商標です。