B-030

キャッシュヒント自動付加を用いたソフトウェア高速化
Software Acceleration for Using Automated Cache Hint Annotation

稲垣 良一† 上田 和紀‡
Ryoichi INAGAKI Kazunori UEDA

1. はじめに

プロセッサとメモリの速度差は広がる一方で，メモリアクセスコストの相対的増加が続いている．プロセッサの性能を活かすためにはメモリアクセスコストを抑えることが重要で，キャッシュの有効活用やプリフェッチなどによるメモリアクセス遅延の隠蔽をソフトウェアレベルで考慮する必要がある．

Itanium2 プロセッサ [1] ではキャッシュヒントによるメモリアクセス命令単位でのキャッシュの制御が可能で，適切に使用することでキャッシュの利用効率向上が期待できる．適切なキャッシュヒントを生成する手法が文献 [2] などで提案されているが，GCC や Intel Compiler といった既存のコンパイラではキャッシュヒントを使用したオブジェクトコードを生成しておらず，コンパイラ利用者がその恩恵を受けることができないのが現状である．

そこで，本稿では既存のコンパイラを利用しながらも，コンパイル時に自動的にキャッシュヒントを付加する手法を提案する．本手法に基づくキャッシュヒント自動付加システム Cache Hint Supply Utility (**CHSU**) を設計し，性能評価を行った結果について報告する．

2. キャッシュヒント

キャッシュヒントはデータを配置するキャッシュ階層を制御するための機能である．Itanium2 プロセッサでは各メモリアクセス命令にヒントを直接付加する形で表現され，メモリアクセス命令単位でのキャッシュの詳細制御が可能である．

Itanium2 プロセッサには四種類のキャッシュヒントが存在するが，本研究ではその中から `.nta` ヒントに注目する．このヒントが付加されたメモリアクセス命令は L2 キャッシュにのみデータを配置し，キャッシュの LRU ビットも更新しない．他のヒントと比べて，データを最もキャッシュに残さないヒントである．通常のストア命令と `.nta` ヒントが付加されたストア命令の振る舞いをそれぞれ図 1，図 2 に示す．

再利用性の低いデータのメモリアクセスに `.nta` ヒントを使用することで，他の再利用性の高いデータがキャッシュから追い出されることを防ぎ，キャッシュの利用効率をあげることができる．

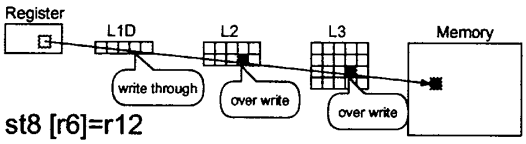

図 1: 通常のストア命令の振る舞い

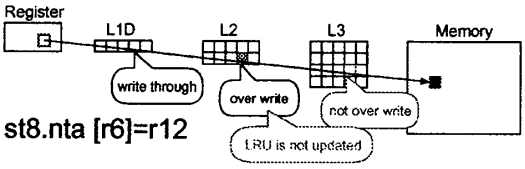

図 2: `.nta` ヒントを付加したストア命令の振る舞い

3. CHSU の設計

3.1 処理方式

本研究で提案するキャッシュヒント自動付加手法の処理の流れを図 3 に示す．まず最初に，対象となるソース

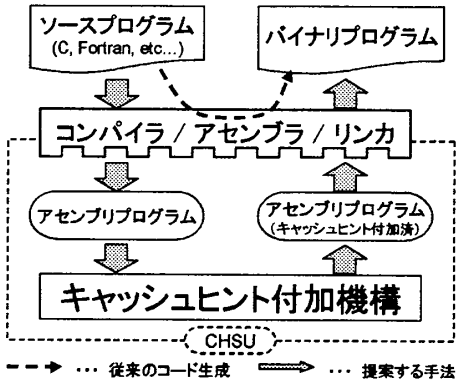

図 3: 提案するシステムの概要

プログラムを入力として，既存のコンパイラからアセンブリプログラムを生成する．次に，生成したアセンブリプログラムを解析し，然るべきメモリアクセス命令にキャッシュヒントを付加する．最後にキャッシュヒントが付加されたアセンブリプログラムをアセンブラで処理し，オブジェクトコードを生成する．コンパイラがアセンブリプログラムを生成できるという前提で，本手法はプログラミング言語の種類に依存しない手法である．

†早稲田大学大学院理工学研究科 情報・ネットワーク専攻
‡早稲田大学理工学部 コンピュータ・ネットワーク工学科

3.2 キャッシュヒントの付加方針

.nta ヒントを適切に付加するためには，再利用性の低いメモリアクセス命令をアセンブリプログラム中から発見する必要がある．本研究ではプログラムの局所性に注目する．アセンブリプログラムをプログラム内のラベルで区切られるブロック単位で解析し，局所性を推測する．その推測に基づいて .nta ヒントの付加を行う．

3.2.1 局所性が高いブロック

分岐命令の分岐先を調べることでループ構造を持つブロックであるかを判断できる．ループ構造を持つブロックは他のブロックに比べて局所性が高いと推測される．

ループ構造内で使用されるストア命令は配列要素へのアクセスのような連続的なアクセスであることが多く，ストアしたデータを再利用する可能性は低い．つまり，再利用性の低いメモリアクセスであると考えられるためストア命令に対しては .nta ヒントを付加する．

ロード命令については同じデータを再びロードすることはないものの，同じキャッシュライン上にある別のデータをロードする可能性があり，キャッシュラインが再利用される．つまり，再利用性の高いメモリアクセスであると考えられるためロード命令については .nta ヒントを付加しない．

3.2.2 局所性が低いブロック

プロシージャリターン命令を持つブロックはプロシージャ内で最後に実行されるブロックであるため，他のブロックに比べて局所性が低いと推測される．

ブロック内で使用されるメモリアクセス命令は他のブロックに比べ相対的に再利用性が低いと仮定し，ストア命令，ロード命令ともに .nta ヒントを付加する．

4. 実装・評価

CHSU は Java で実装し，SGI Altix 350 (Itanium2 1.4GHz – L1D:16KB, L2:256KB, L3:3MB) 上で評価を行った．

本稿では FFT ライブラリの FFTE のコンパイルについて **CHSU** を使用してキャッシュヒントを付加した場合と，付加しない場合との性能比較結果を報告する．アセンブリプログラムの生成およびコンパイルには Intel Compiler 8.1 を使用した．データ数 $N = 2^m$ を変化させながら一次元 FFT を計算した時の MFLOPS 値と性能向上比を表 1 に示す．

評価に用いたプロセッサでは $N \geq 2^{17}$ の場合にデータサイズが L3 キャッシュサイズより大きくなり性能が大きく低下するが，**CHSU** を使用した場合 $N \geq 2^{17}$ 以降で性能が向上する結果となった．特に $N = 2^{17}$ では

表 1: FFTE の性能

N	Original	**CHSU**	性能比
2^{15}	2762.62	2763.55	1.00
2^{16}	2463.82	2499.77	1.01
2^{17}	781.20	898.09	**1.15**
2^{18}	614.39	668.38	**1.04**
2^{19}	617.44	635.22	**1.04**
2^{20}	638.18	650.75	**1.02**

15% の性能向上を達成している．次に，$N = 2^{17}$ の場合についてプロセッサイベントを計測した結果を表 2 に示す．L3 キャッシュのヒット率 (L3D hit rate) が 15% 向上し，IPC，実行時間に占めるプロセッサストールの割合 (Total stalls) についてもそれぞれ改善された．キャッシュヒントの付加により L3 キャッシュヒット率が向上し，ソフトウェアの性能向上に貢献していると考えられる．

表 2: FFTE 実行時のプロセッサイベント

$N = 2^{17}$	Original	**CHSU**
L2D hit rate	0.982	0.982
L3D hit rate	0.401	**0.551**
Instructions/cycle (IPC)	1.703	**1.754**
Useful instructions/cycle	1.170	**1.363**
Total stalls	0.625	**0.605**

5. おわりに

既存のコンパイラを利用して，コンパイル時にキャッシュヒントを自動的に付加する手法を提案した．FFTE による性能評価ではキャッシュヒントを付加した場合，L3 キャッシュのヒット率が最大 15% 向上した．プロセッサに搭載されるキャッシュの容量は増加し続けており，ヒット率の向上を実現した本手法の有用性は高いと考えられる．

今後の課題としては，性能評価の充実，キャッシュヒント付加方針の検討，バイナリプログラムに対するキャッシュヒント自動付加への応用などが挙げられる．

参考文献

[1] インテル株式会社．インテル Itanium アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル 第一巻〜第三巻, Oct 2002.

[2] K.Beyls and E.D'Hollander. Compile-Time Cache Hint Generation for EPIC Architectures, In *Proc. EPIC2*, pp. 19–29, Nov 2002.