2P-9

メモリ上の全文検索システムのためのデータ構造と処理の効率化

渡辺 健太郎 † 高須 淳宏 †† 安達 淳 ††
† 東京大学 大学院 †† 国立情報学研究所

1 はじめに

メモリサイズが増大し低価格化が進んだことから，メモリ上でのデータベースの運用は珍しいものではなくなった．そのようなシステムではかつてのI/Oではなく，プロセッサやメインメモリのリソースがボトルネックとなりつつある．本研究では，転置索引を用いた全文検索システムにおいて，特にメモリ上で運用することを念頭に，圧縮データ構造の処理効率を改善することを目指す．本稿では，過去の研究 [4, 6] をベースに，現代のプロセッサで実装されている命令レベル並列処理技術を効果的に使用する手法を提案する．メモリ上の全文検索システムにおいて，クエリ処理のスループットの向上という観点から，圧縮された整数列の展開性能を大きく改善する可能性を示す．

2 前提

2.1 転置索引

ここでは以下のような環境を設定する．各文書には単一の文書IDが割り当てられ，各単語には単語IDが割り当てられる．転置索引が保持するのはそれらのIDである．各単語IDに対し，文書IDのリストである転置リスト (*inverted list*) を保持する．*Boolean retrieval query* のみならず，さらに *Phrase query* への対応には別に文書中で単語の出現する位置 (*position*) を保持しておく仕組みが必要となる．その際各転置リストの文書IDの情報の後に位置情報を保持するか，位置情報のみを別に保持するのが一般的である．

転置索引を用いた問い合わせ処理は概ね次のようになる．クエリの単語に対応する転置リストをメモリ上あるいはディスク上の索引から取得し，必要であれば圧縮データを展開する．そしてリスト同士のintersectionあるいはその他のBoolean filter処理を行うことで，クエリ中の単語の条件を満たす文書IDを決定する．文書IDに加えて索引に保持しておく付加的な情報 (単語の出現頻度等) を用いて候補の文書に関するスコアを計算し，top-k 件の文書を出力する．

そのため，転置索引を用いた問い合わせ処理において主な処理は次の3つになる．

- 圧縮転置リストの展開
- リスト同士のintersection
- top-k スコア計算

転置索引の圧縮の手法は過去に様々提案されてきた [5]．索引構造を圧縮して保持することを考えるとき，圧縮による空間効率と問い合わせ処理の効率はしばしばトレードオフの関係となる．

一方，文書IDを記憶する整数列自体についても圧縮手法が研究されてきた．文書IDを表現する整数列においてソートされていることを前提すると，整数列の内容を隣接する2項間の差分とする符号化 (Δ-*encode*) と，Null-Suppressionを組み合わせることで高い圧縮率を実現できる [6] のみならず，クエリ処理の効率も改善できることが示されている [4]．

Δ-*encode* された整数列はPrefix Sum処理を施すことで元の整数列の内容に戻る．一般にPrefix Sum処理とは以下のように，ある演算子 $\oplus$ と n 個の要素からなる整数列 A から整数列 A' を得る処理のことである [1]．

$$A = [a_0, a_1, \cdots, a_{n-1}] \quad (1)$$

$$A' = [a_0, (a_0 \oplus a_1), \cdots, (a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1})] \quad (2)$$

Δ-*encode* されたデータの復元には演算子として加算 $+$ を用いる．

2.2 On-chip SIMD 命令

現代のプロセッサは汎用命令，FPU命令の他に，単一の命令で複数のデータ単位を処理するSIMD命令を実装している．例として，IntelのXeonやAMDのPhenomといったプロセッサは *Streaming SIMD Extensions*(SSE) 命令セットをサポートしている．SSE命令セットでは128ビットのレジスタと対応する操作が実装されており，特に3Dグラフィクス，音声認識，画像処理，科学計算といった用途でパフォーマンスを向上させることを念頭に開発された [7]．SIMD命令の典型的な利用例は浮動小数点数演算であるが同時に，整数演算の用途においてもSIMD命令の適用によりパフォーマンスが向上することが示されている [2, 3]．SIMD命令の利用による性能の改善は，プロセッサがSIMD命令を実装していれば他に何も追加コストがかからないという点が大きな特色である．一方で，SIMD命令を使用しない通常のソースコードから，コンパイラが自動で必要なSIMD命令を検知，使用するということは現状，十分には実現されていない．

そこで，本稿では加算演算のPrefix Sum処理をSIMD命令を用いて処理する手法を提案する．必要となるSIMD命令は以下である．

- 128ビット単位のレジスタ間，レジスタ-メモリ間の転送命令
- 128ビットレジスタにおける32ビット整数要素毎の加算命令
- 128ビットレジスタにおけるバイト単位シフト命令

3 In-register Prefix Sum

CPU実装の標準的なSIMD命令をもちいた，Δ-*encode* されたデータの展開手法について述べる．この方法では1回のiterationでSIMD演算用のレジスタ1本分のデータを展開する．図1にレジスタのサイズが128ビットの場合の，レジスタ1本のデータの展開の概要を示す．

ワード数 n のレジスタに対して，$\log n$ 回のシフトと加算でデータを展開する．最初に n 個のデータをレジスタにロードする．最初の加算では，ロードしたデータを1ワード分シフトさせた別のレジスタを用意し，これらの間で加算を行う．2回目の加算では，最初の加算の結果を2ワード分シフトさせ同様に加算を行う．

このようにして，1回のiteration内の k 回目の加算では $k-1$ 回目の加算の結果を 2^{k-1} ワードシフトしたものとの加算を行うことになる．

各iteration間では前のiterationのレジスタの最上位ワードの内容を次のiterationに渡していく．対象となる整数列の要素数がレジスタのワード数の倍数ではない場合，最後に余った要素については別途処理を行わなければならない．

Data Structure and Efficient Processing for High Throughput IR System on Main-Memory
† Kentaro Watanabe, The University of Tokyo
†† Atsuhiro Takasu, Jun Adachi, National Institute of Informatics

Copyright ©2011 Information Processing Society of Japan.
All Rights Reserved.

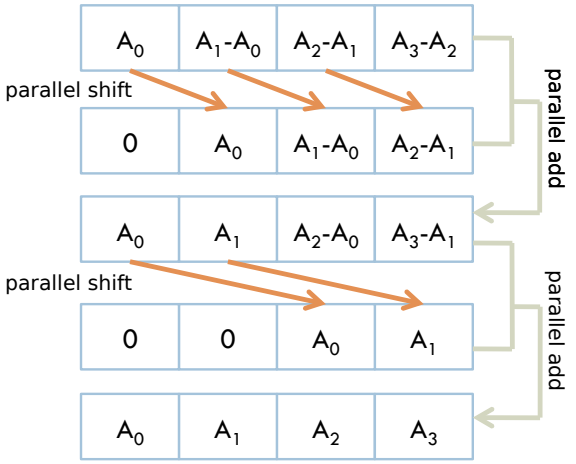

図 1: 128 ビットレジスタ・32 ビット整数列の Prefix Sum

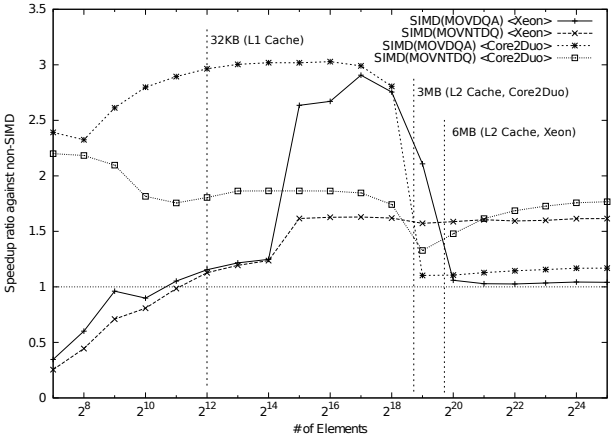

図 2: Xeon・Core2Duo プロセッサでレジスタ-メモリ間の転送に MOVDQA・MOVNTDQ 命令を用いた場合の In-register Prefix Sum の速度向上比

3.1 キャッシュの効率

SIMD 命令の利用による高速化は処理における命令レベルの並列性 (ILP) を向上させるものとして捉えられる [2]．メモリアクセス時のキャッシュミスによる遅延は数百 CPU サイクルにのぼるため，処理の対象となるデータを含めてアプリケーションが使用する領域の大きさがラストレベルキャッシュ(LLC) のサイズと同じオーダに達する場合，メモリアクセスについて考慮することは有益である．

SSE 命令ではレジスタからメモリへの書き込みにおいてキャッシュラインを経由するかどうかで異なる命令が存在する．たとえば，XMM レジスタからメモリへの書き込みにはアライメントが揃っている場合は通常，MOVDQA 命令が使用されるのに対し，後続の処理でしばらく使用しないテンポラルなデータの書き込みに対しては MOVNTDQ 命令の使用が推奨されている [7]．後者のようなストリーミング・ストア命令を使用することでキャッシュをフラッシュする影響を最小限に抑えることができる．

したがって入力データのサイズによって，メモリの書き戻しに適切な方法を選択することでキャッシュラインの利用を最適化することが可能であると考えられる．

4 実験と考察

Δ-*encode* された整数列の展開において，前節で説明した In-register Prefix Sum を用いて SIMD 命令を利用した場合と SIMD 命令を利用しない場合でそれぞれ処理時間を計測し，比較を行った．いずれも C 言語で実装し，SIMD 命令の使用には C-intrinsic 組み込み関数を用いた [7]．実験は，Intel の Xeon(X5492) および Core2Duo(T8300) プロセッサを用い，それぞれ Red Hat Linux 上，Mac OSX 上で行い，コンパイルにはいずれも GCC を用いた．Xeon は L1 データキャッシュ32KB，2 コア共通 L2 データキャッシュ6MB という構成であり，一方 Core2Duo は L1 データキャッシュ32KB，2 コア共通 L2 データキャッシュ3MB という構成である．SIMD 命令はそれぞれのプロセッサで，MMX，SSE，SSE2，SSE3 をサポートし，In-register Prefix Sum に必要な命令を実行できる．最適化のオプションはいずれのアルゴリズムも-O3 でコンパイルを行った．それぞれの実験は 150 回実行し，その結果のプロセッサの消費サイクル数の中央値を用いた．

図 2 に SIMD 命令を利用した場合としない場合の処理時間の比較結果を示す．グラフは横軸が処理した整数列の長さ，縦軸が速度向上比を表しており，値が大きい方がより速度が改善していることを示す．

整数列が短い場合，L1 キャッシュから L2 キャッシュに載る間の範囲では MOVDQA 命令を用いた場合，Xeon，Core2Duo プロセッサの双方で最大で 2.5〜3 倍弱の性能改善が見られる．一方，L1 キャッシュサイズに載る範囲では Core2Duo では改善が見られたものの，Xeon プロセッサでは改善が見られないか，あるいは逆に遅くなるということがわかった．

整数列が長く，キャッシュに載りきらない場合，アプリケーションの途中でキャッシュとメモリのやり取りが発生する．図 2 中で LLC に載るラインを示しているが，処理対象のデータが LLC に載らない場合は MOVDQA 命令を用いた In-register Prefix Sum による速度向上は限定的なものとなる．一方，MOVNTDQ 命令を用いた場合はメモリへの書き戻しでキャッシュラインに後続処理で使わないデータが残るという非効率性が除かれ，入力データが LLC に載らない大きさでも安定した速度の向上が見られる．

5 おわりに

本稿では SIMD 命令を用いた加算演算の Prefix Sum 処理の手法を提案した．入力データのサイズを考慮して命令を選択することで LLC に載らないサイズの整数列データでも安定して速度向上を実現できることを示した．

References

[1] G. Blelloch. Prefix sums and their applications. Synthesis of Parallel Algorithms, pp. 35-60, 1993.
[2] C. Kim et al. FAST: Fast Architecture Sensitive Tree Search on Modern CPUs and GPUs. In SIGMOD, 2010.
[3] B. Schlegel et al. k-Ary Search on Modern Processors. In DaMoN, 2009.
[4] F. Transier and P. Sanders. Compressed Inverted Indexes for In-Memory Search Engines. In ALENEX, 2008.
[5] J. Zobel and A. Moffat. Inverted files for text search engines. ACM Computing Surveys, 38(2), 2006.
[6] M. Zukowski et al. Super-Scalar RAM-CPU Cache Compression. In ICDE, 2006.
[7] IA-32 インテルアーキテクチャー・ソフトウェア・デベロッパーズ・マニュアル. http://www.intel.co.jp/jp/download/index.htm

Copyright ©2011 Information Processing Society of Japan.
All Rights Reserved.