社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－HPC－99 (45)
2004／8／1

性能安定化を実現するデータプレロードに関する一考察

城田祐介† 前田誠司†

Gridコンピューティング環境で利用される数値計算ライブラリでは，課金や計算機資源スケジューリングのため，性能安定性を保障する必要性が高まってきている．本稿では，性能安定化阻害要因であるキャッシュミスを安定的に防止し，速度性能の安定的な向上を実現する一手段として，プリフェッチ命令を利用したデータプレロードを利用する方法を検討する．プリフェッチ命令の挿入は，ライブラリの実行環境等を考慮して適切に行なわないと速度向上を得られない．しかし，この作業をアプリケーション作成者が毎回行なうのは困難であるため，結果的に特定の実行環境等に対して依存性の高いコードとなり，性能安定性が損なわれてしまう．そこで，本稿では，実行環境パラメタとデータパラメタセットに応じて，プリフェッチ命令の実行を自動調整する性能安定化型ライブラリの構築を検討する．そして，プリフェッチ命令を安定的に実行する上で課題となるアドレス変換処理の影響と，自動化の実装のシンプルさを考慮した際に生じる余計なプリフェッチ命令の実行オーバヘッドに関して一考察を与え，その予備評価を行なった．

On Performance Stabilization Using Data Preloading

Yusuke Shirota † and Seiji Maeda †

Now that commodity access to computing resources is realized with Grid Computing, performance assurance of the computing libraries on the Grid environment is increasingly becoming an important issue.

In this paper, data preloading using processor data prefetch instructions is introduced as a way to hide latency of data accesses and stabilizing performance. However, appropriate scheduling of the prefetch instructions depends on many architectural parameters, and so achieving high performance requires extensive platform-specific tuning, and the code ends up being highly tuned for a specific system and the performance stabilization on wide range of systems is lost.

In order to achieve high performance on many systems, we target at computing libraries with facilities that auto-tune prefetch instructions with the architectural parameters and application parameters. We look into the impact of the address translations and the execution overhead of unnecessary prefetches induced by our implementation. We have done some preliminary evaluations on the Pentium 4 processor.

1. はじめに

近年，Gridを用いたコンピューティング環境が整備されつつある．Gridコンピューティング環境を用いた数値計算ポータルにおける数値計算ライブラリにおいては，従来通りに，単に高い速度性能を保障するだけでは十分でない．Grid上の多種多様な計算機上において，高い速度性能を安定的に発揮できる性能安定性も保障する必要がある．

これまで，数値計算ライブラリでは，限定されたデータパラメタセットに対するピーク性能が評価指標となってきた．しかし，数値計算ポータル等では，投入されるGridアプリケーションのスケジューリングや課金を行なう必要がある．このために，実行環境パラメタとデータパラメタセットに依存せずに，ライブラリの実行時間を安定化し，正確に予測可能にする必要性が高まってきている．ここで，前者は，ライブラリで処理するデータセットのサイズやデータ配置等である．後者は，メモリバンド幅，及び，TLB(Translation Look-aside Buffer)やキャッシュのエントリ数，ウェイ数，置換アルゴリズム等である．

また，昨今の汎用プロセッサにおいては，性能向上にキャッシュを利用しているが，キャッシュヒット時とキャッシュミス時では，データアクセス時間が大きく変化してしまう．このため，キャッシュミスをいかに実行環境パラメタやデータパラメタセットに依らずに防止するかが，上記の性能保障という点で，最重要課題となる．

そこで，キャッシュミスを防止するために，データプレロードの活用を検討する．データプレロードの手段には，プリフェッチ命令を利用する．キャッシュミスの防止を安定的に行なうためには，プリフェッチ命令の挿入を，実行環境パラメタやデータパラメタセットに応じて適切に行なう必要があるが，これをアプリケーション作成者が毎回行なうのは，現実的ではない．このため，プリフェッチ命令の実行を自動調整するライブラリの検討を行なっている．そして，プリフェッチ命令を安定的に実行する上での課題となるアドレス変換処理の影響と，自動化の実装のシンプルさを考慮した場合に生じる余計なプリフェッチ命令の実行オーバヘッドに関して一考察を与え，その予備評価を行なった．

以下，本稿では，2章でプロセッサのキャッシュ機構

† (株) 東芝 研究開発センター
Toshiba Corporation, Corporate Research & Development Center

について説明し，3 章で性能安定化について説明する．4 章でキャッシュミス防止手法とデータプレロード方法について説明し，5 章でプリフェッチ命令の実行を自動調整するライブラリの構想を述べる．6 章で予備評価を Pentium4 上で行なった結果を示す．7 章で関連研究について述べ，8 章でまとめる．

2. 汎用プロセッサのキャッシュ機構

2.1 プロセッサによるデータアクセス

プロセッサのデータアクセスでは，プロセッサのアドレス変換処理とキャッシュ検索が順に行なわれる．

アドレス変換処理では，TLB ミスすると，PTE(Page Table Entry) が，ページテーブルより検索される．

キャッシュ検索は，上位から下位レベルのキャッシュに対して順々に行なわれ，キャッシュミスすると，メモリアクセスが行なわれる．

TLB やキャッシュは，アクセスレイテンシが数クロックサイクル程度と小さい．一方で，メモリアクセスのレイテンシは，数百クロックサイクルと非常に大きい．

2.2 基本的性質

TLB とキャッシュの構成は，ほぼ同一である．違いは TLB では PTE が格納され，キャッシュではキャッシュラインが格納される点である．

次に，これらのキャッシュ機構の基本的性質を 2 つ示す．

- 容量ミス:
  キャッシュ機構の置換アルゴリズムが LRU で，ワーキングセットのサイズがキャッシュ容量以下であれば，エントリが再利用される．しかし，少しでも超えた場合，一度ロードされたエントリが再利用されることなく置き換えられてしまう．
- 競合ミス:
  ストライドアクセスのアクセス間隔が，キャッシュ機構のセット数の倍数の場合，特定のセットに対して使用するエントリが集中し，ミスが発生してしまう．

このように，キャッシュミス/ヒットが変化する境界条件が存在する．境界条件は，実行環境パラメタとデータパラメタセットのそれぞれに対して存在する．前者は，TLB やキャッシュのエントリ数等である．後者は，ストライドアクセス間隔やデータ配置等である．更に，一つのアプリケーション内には複数のアクセスパタンが存在し，キャッシュや TLB を共有するため，境界条件は複雑化する．

3. 性能安定化

性能安定化とは，単に高い速度性能ではなく，実行環境パラメタとデータパラメタセットに依らずに，高い速度性能を実現することである．

一方で，TLB ミスとキャッシュミスを防止できないと，メモリアクセスを伴うためにデータアクセスが低速化し，アプリケーションの実行時間が大きく変化してしまう．更に，キャッシュ機構のミス/ヒットは，前章で述べた境界条件により変化してしまう．

このため，性能安定性を保障するためには，これらの境界条件に応じて，TLB ミス及びキャッシュミスを防止する手法が安定的に行なわれるようにする必要がある．

4. キャッシュミス防止手法とデータプレロード

性能安定化を実現するためには，TLB ミス防止手法とキャッシュミス防止手法を安定的に実行する必要がある．アプリケーション実行時間全体の性能安定化の実現に向けて，本稿では，まず，後者に注目する．

4.1 キャッシュミス防止手法

キャッシュミス防止手法は，大きく 2 つある．一つは，ループブロッキング等のデータアクセスの局所性を向上させる手法である．もう一つは，データプレロードを活用する方法である．データプレロードは，キャッシュミスを発生するデータアクセス要求を，キャッシュミスレイテンシ分だけ先行して発行しておくことで，データが実際に必要になる時点ではプロセッサのデータ到着待ちがなくなり，メモリ転送時間をアプリケーション実行時間から隠蔽できる．

後者は，前者が適用できない状況でも，事前にアクセスパタンが分れば適用できる可能性があるため，制御性が高い．また，多くのプロセッサが持つ，データプレロードをサポートする機構を利用できるメリットもある．また，アルゴリズムの変更が必要ない点では，本稿の後半で検討するプレロード処理の自動化との相性もよい．また，当然，両者を併用し，性能安定性の更なる強化も十分可能である．

本稿では，データプレロードを活用したキャッシュミスの防止を，実行環境パラメタやデータパラメタセットに依らずに安定的に行なうことで性能安定化を実現する方法を検討課題とする．

4.2 2 つのデータプレロード方法

データプレロードを行なう方法には 2 つある．一つは，擬似的なロード命令を挿入する方法である．もう一つは，プロセッサが持つプリフェッチ機構を利用する方法である．

本稿では，後者が有する以下の 3 つの特徴により，後者を利用することを選択した．

(1) 性能面で有利
(2) アプリケーションのセマンティックスを変更しないためプレロード処理の自動化向け
(3) プレロード先のキャッシュレベルやウェイを指定できる制御性

プリフェッチ命令はキャッシュラインの更新のみを行ない，実際のロードを行なわないため，特徴 1 と 2 の効果が期待できる．

4.3 プロセッサのデータプリフェッチ機構

プロセッサのデータプリフェッチ機構には，ハードウェアプリフェッチとプリフェッチ命令がある．

ハードウェアプリフェッチは，データアクセスパタンを予測し，プリフェッチを自動的に行なうため，アプリケーション側での変更が必要がない．その一方で，ハードウェアプリフェッチは，以下の欠点を有する．

(1) 予測に基づくため，スタートアップコストがありプログラマが期待する全データに対してプリフェッ

チが行なえない

(2) 単純なアクセスパタンのみに対応

このようにハードウェアプリフェッチは万能でなく，性能安定化の実現のためには十分ではない．

一方で，プリフェッチ命令は，その明示的な挿入が必要だが，事前にアクセスパタンが分れば，単純なアクセスパタン以外でも対応できる可能性がある．よって，プリフェッチ命令の利用は，性能安定化には必須である．

4.4 性能安定化のためのプリフェッチ条件

プリフェッチ命令を用いて性能安定化するためには，以下の2つの条件を任意の実行環境パラメタとデータパラメタセットで満たすことが必要である．

[条件1：プリフェッチのためのタイミング条件]

プリフェッチは，実際のデータアクセス時にはキャッシュにデータが入っているように，十分前に開始することが必要である．仮に，プリフェッチが十分前に行なわれなかった場合，メモリ転送のレイテンシが完璧に隠蔽できず，データアクセス時間が変化してしまう．

また，プリフェッチしたデータが，実際に利用される前にキャッシュから追い出されないよう，プリフェッチは実際のデータアクセスに対して手前過ぎない必要もある．

[条件2：アドレス変換処理に関する条件]

プリフェッチ命令の実行時に，プリフェッチ対象のデータが存在するページに対応するPTEがTLBに入っていない場合には，ページテーブルでの検索が行なわれず，プリフェッチは行なわれない．つまり，プリフェッチを確実に実行するためには，TLBに当該PTEが入っていることを保障することが必要となる．

4.5 プリフェッチ命令のための条件を満たす方法

以下に，上記2つのプリフェッチ条件を満たす方法について述べる．

[条件1を満たす方法]

実際にどの程度前に，プリフェッチ命令を挿入するかは，実行環境パラメタであるメモリバンド幅とキャッシュラインサイズに主に依存する．ループ中にプリフェッチ命令を挿入するような場合，各イタレーションでの処理時間に対するキャッシュラインの転送時間を考慮し，何イタレーション前にプリフェッチ命令を実行すればよいかを算出する[1]．

[条件2を満たす方法]

プリフェッチ命令の実行時に該当PTEがTLBに入っているようにする方法として，以下の2通りがある．

(1) 当該PTEがTLBに入っている期間をプリフェッチ条件1を満たす中から見つけ，その間にプリフェッチ命令を実行する．

(2) 事前に当該PTEをプレロードしてTLBに入れる．

方法1について考える．プリフェッチ条件1を考慮すると，プリフェッチはある期間中に行なう必要があるが，この期間中に行なわれるデータアクセスにより，TLB中のPTEは刻々と変化する．プリフェッチ命令は，刻々と変化するTLBの中に当該PTEが入っているタイミングを計算し，挿入しなければ，期待通りのプリフェッチは行なわれない．このタイミングは，TLBに関する実行環境パラメタとデータパラメタセットに影響するため，これらを考慮して総合的に決定する必要がある．

方法2は，当該PTEをプレロードしてTLBに入れる方法である．TLBプレロードは，当該ページの一部に対して事前にロードを行ない，TLBミスを発生させ，当該PTEをTLBに入れることで実現できる．TLBプレロードは，実際のデータアクセスの十分前に実行するプリフェッチ命令を実行する時点で，当該PTEがTLBに入るように，更に十分前に実行する必要がある．実際にどの程度前に実行するべきかは，プリフェッチ条件1のキャッシュラインをPTEに置き換えれば，同様の算出方法が利用できる．

TLBプレロードは，以下に示すクロック数分だけ事前に行なえばよい．

[プリフェッチのためのタイミング条件を満たすクロック数] + [TLBプレロードのためのタイミング条件を満たすクロック数]

また，TLBプレロードは，プレロード用にTLBエントリを消費するが，これさえ確保できれば，ある程度は他のアクセスによる影響を受けずに，プリフェッチ命令用のPTEをTLB内に確保できる．

性能安定化を実現するためには，このように，実行環境パラメタとデータパラメタセットに応じて，アドレス変換処理を考慮した上で，プリフェッチ命令の挿入位置を決定し，プリフェッチが確実に行なわれるようにする必要がある．このための方法論は，今後確立する必要がある．

5. プリフェッチ命令の実行を自動調整するライブラリ

本稿では，性能安定化を実現するために，アドレス変換処理を考慮した上で，プリフェッチ命令の実行を自動調整するライブラリを構築する．このライブラリを利用することで，アプリケーションの実行時間が予測可能になり，正確なスケジューリング等が実現可能になる．

本章では，このための方法論や課題を提案する．

5.1 プリフェッチ命令の実行の自動調整

プリフェッチ命令を利用したデータプレロードをアプリケーション作成者が行なうためには，利用する実行環境のアーキテクチャを充分に理解し，アプリケーションのデータパラメタセットを考慮した上で，2つのプリフェッチ条件を満たすように，プリフェッチ命令の挿入位置を適切に決定する必要がある．この作業を，実行環境の差位を考慮して行なうのは難しい．結果的に，特定の実行環境やデータパラメタセットに依存したアプリケーションになってしまう可能性が高い．

このため，プリフェッチ命令を利用した性能安定化処理は，数値計算ライブラリに内蔵される性能安定化機構で自動化する．

数値計算ライブラリでは，計算処理が定形化されている．定形化されている処理を性能安定化させるためには，実行環境パラメタとデータパラメタセットで決定されるキャッシュミスの境界条件に対して，これを防止するためのプリフェッチ命令の挙動を適宜変更する必要がある．

この方法として，プリフェッチ命令の自動挿入を行なう方法と，プリフェッチ命令の実行を自動調整する方法がある．前者は命令の挿入位置を計算する必要があるが，後者では，アドレス変換処理等を意識した方法をパラメタ化しておき，その機能を実現するよう事前にプリフェッチ命令のコードを入れておく．そして，パラメタの値を計算させて実行させればよい．数値計算ライブラリの開発容易性という観点から，本稿では後者を選択した．

5.2 プリフェッチ命令を利用した性能安定化機構内蔵ライブラリの特徴

プリフェッチ命令の実行を自動調整する数値計算ライブラリは，以下の 3 つの特徴を持つ．

- 実行環境パラメタを考慮する
- データパラメタセットを考慮する
- プリフェッチ命令実行の自動調整を行なう

ライブラリの基本的な枠組みは，外部入力として実行環境パラメタとデータパラメタセットを想定し，これに対して適切にプリフェッチ命令の実行を行なうことである．

5.3 ライブラリの構想

本ライブラリの処理手順は次の通りである．

(1) 処理用データ領域の作成
(2) 処理用データの再配置
(3) プリフェッチ命令実行の自動調整
(4) プリフェッチ命令の実行

手順 1 では，ライブラリで処理するデータを，ライブラリのユーザが使用しやすい形式でユーザのデータ領域に用意させ，ライブラリ API で通知させる．

手順 2 では，手順 1 で用意されたデータを，実行時のデータ局所性と処理の容易さを考慮したデータ構造の変換等を行ない，別途用意する処理用ワークエリアに再配置する．また，他のキャッシュミス防止手法と併用するために，例えば，行列を転置し，ハードウェアプリフェッチが効くようにすることも考えられる．再配置処理には，別途処理オーバヘッドが掛かるため，コストに見合うかの見極めも必要である．

手順 3 では，実行環境パラメタとデータパラメタセットを考慮し，プリフェッチ条件とアドレス変換処理に関する条件を求め，プリフェッチ命令と TLB プレロード用のコードのパラメタを計算する．

また，付加的処理として，アプリケーション実行時間の見積もりも行なう．TLB エントリ等は有限であるため，全ての外部入力の組に対して，性能安定性を保障できるとは限らない．数値計算ポータルでの課金という観点からは決定性が重要であるため，その判定手段も必要となる．例えば，TLB プレロードでは，その多くは，ループ内で実行されることが多い．この場合，少くともプレロードが使用するエントリ数の半分が，TLB のエントリ数よりも多ければ，PTE が意図した通りに入らず，プリフェッチ命令は安定的に行なわれずに，性能安定性が損なわれることが事前に判る．

手順 4 では，手順 3 で計算されたパラメタ値に従い，プリフェッチ命令を実行する．

5.4 自動調整の課題

性能安定化のために，プリフェッチ命令の実行を自動調整する際には，2 つの課題がある．

一つは，アドレス変換処理の影響である．このために，手順 2 のデータ再配置時には，将来的な TLB ハンドリングも考慮してページアラインを行なう．また，他のキャッシュミス防止手法と併用する際には，TLB プレロード用のエントリを確保した上で，他手法を適用する．そして，手順 3 で，実行環境パラメタやデータパラメタセットに応じて，PTE を TLB 内に確保する方法を選択し，TLB プレロード用のコードのパラメタを計算し，確実にプリフェッチ命令用の PTE が TLB に入っているようにする必要がある．

もう一つは，ライブラリの実装のシンプルさを追及した場合に生じる余計なプリフェッチ命令の実行オーバヘッドである．ライブラリの実装は，前述通り，事前にプリフェッチ命令のコードを埋めこむものとする．また，更なるシンプルさを追及するために，複雑な境界条件による条件分岐も極力少くし，そのオーバヘッドも削減する．これを考慮すると，複雑な境界条件による挙動変更に対応するために，コードのあらゆるところに各境界条件用のプリフェッチ命令が埋めてあることになる．この中には，当然，実際にプリフェッチが効果的に行なわれる命令に加えて，余計な命令も入っていることになる．

また，ハードウェア命令の弱点を補完する等の目的で，ハードウェアプリフェッチが実行される箇所にも，プリフェッチ命令を配置することも考えられる．この場合，ハードウェアプリフェッチとプリフェッチ命令が併用され，余計なプリフェッチ命令が実行されてしまう．これらのオーバヘッドの実行時間への影響が課題となる．

このようなライブラリを構築すれば，プリフェッチ命令が安定的に実行され，キャッシュミスが安定的に防止され，その結果，アプリケーション実行時間が予測可能になり，正確なスケジューリング等が実現できる．

6. 予備評価

プリフェッチ命令の実行を自動調整するライブラリを構築する上での課題について予備評価を行なう．

6.1 予備評価の目的

本予備評価は，以下の検証を目的に行なう．

(1) プリフェッチ命令の有効性
(2) 自動化の実装のシンプルさを考慮した際に生じる余計なプリフェッチ命令の実行オーバヘッド
(3) プリフェッチ命令に対するアドレス変換処理の影響

6.2 評価環境

予備評価は，2.66GHz の Pentium4(表 1) を搭載した PC(表 2) 上で行なった．

Pentium4 では，L2 キャッシュにデータをプリフェッチする機構が 2 つ用意されている．一つは，ハードウェアプリフェッチで，もう一つは，ストリーミング SIMD 拡張命令 (SSE2) で導入されたプリフェッチ命令である．

表 1 Pentium4 プロセッサのスペック

FSB クロック	533MHz
FSB 帯域	4.2 GB/s
D-L1 キャッシュサイズ	8KB
L2 キャッシュサイズ	512KB
キャッシュラインサイズ	64B
D-TLB エントリ数	64

表 2 評価用計算機

計算機	Toshiba Equium5090
CPU	Intel Pentium4 2.66GHz
メモリ	PC2100 対応 DDR SDRAM 1GB
OS	Linux kernel 2.4.20-8
コンパイラ	gcc 3.2.2
コンパイルオプション	-O2

```
for(i=0;i<N;i++){
  for(j=0;j<N;j++){
      for(k=0;k<N;k++)
        c[i][j] += a[i][k]*b[k][j];
  }
}
```

図 1 検証用行列積演算プログラム

前者は，シーケンシャルアクセスを検知し，自動的にプリフェッチを行なう．

6.3 検証用プログラム

予備評価には，倍精度浮動小数点の正方行列積 $C = AB$ を求めるプログラムを用いた．

一般に，正方行列積演算プログラムでは，キャッシュミスや TLB ミスを考慮し，3 つの行列のブロック化を行なうが，本予備評価では，正方行列積演算プログラムの高速化自体が目的でないため，ブロック化は行なっていない．

行列積演算プログラムは図 1 の通りである．

6.4 行列積演算プログラムの処理フロー

行列積演算プログラムの処理フローを考える．以後，配列サイズ N=1024 とする．配列 A へのアクセスでは，各 i ループにおいて，8KB の行へのアクセスが繰返し行なわれるため，各 i ループの処理中にデータが L2 キャッシュから追い出されることはない．また，配列 C へのアクセス頻度は低い．

一方で，配列 B へのストライドアクセスは，特定のセットに対する競合を引き起こす．このため，配列 B へのアクセスが性能上に大きな影響を与える．

そこで，ライブラリ手順 2 において，セット競合を回避するために，配列 B の各行について，ワークエリアをキャッシュライン 1 つ分だけ余分に確保し，データコピーを行なうとする．その結果，実行時間が 8.64 倍と大幅に向上した (表 3) .

6.5 プリフェッチ命令に対するアドレス変換処理の影響の検証

配列 B のサイズは L2 キャッシュの容量を超えるため，セット競合回避後は，配列 B の列へのアクセスが，各キャッシュラインへの最初のアクセスとなる場合には，その列に対してはキャッシュミスを起こす．そこで，配列 B の同じ行は，同じ PTE に対応するので，同じ行の d 個先のキャッシュラインをプリフェッチするようにした．ここでは，自動化のシンプルさを考慮し，本来は必要ないが全ての乗算に対してプリフェッチ命令 prefetcht0 をその前に挿入した．d を 1〜4 と変化させた結果，挿入前後の比較において，いずれも 2.3%の性能劣化を示した (表 3).

表 3 行列積演算の実行時間

プログラム	実行時間 [s]
セット競合あり	153
セット競合なし	17.7
d=1-4	18.0

表 4 方法 1 を使用した場合の実行時間

	実行時間 [s]
変更前	17.7
変更後	15.7

この性能劣化の原因は，プリフェッチ命令のために必要な PTE が TLB に入っていないことであると考えられる．配列 B の 1 列分のアクセスには PTE を 1024 エントリ使うため，TLB エントリ数が不足し，プリフェッチ命令の発行時に該当 PTE が TLB から追い出されている．このため，各乗算に対して TLB ミスが毎回発生し，プリフェッチが行なわれない．

この状況に対して，次の 3 通りの方法を試みた．

(1) プリフェッチ命令と配列 B の要素取得命令の位置の入れ換え
(2) TLB プレロード
(3) PTE が TLB に入っている位置を計算し，プリフェッチ命令を挿入

方法 1 と方法 2 は，プリフェッチ命令用の PTE を TLB に入れる方法である．前述通りにデータアクセス中に TLB ミスが毎回発生するが，その対象 PTE は，2 回目以降，全て L2 キャッシュに入っている．配列 B で使用する PTE 数は 2064 個で，各 PTE を 4Byte とすると，合計 8KB 強であり，更に，配列 B へのアクセスで常にこれがアクセスされるためである．このため，TLB プレロードを実行し易い条件が整っている．

方法 1 は，プリフェッチ命令の場所を入れ換えると，配列 B の要素の取得時に、TLB ミスが発生する．この時に TLB に入る PTE は、プリフェッチ命令に必要な PTE と同じであるため，そのあとにプリフェッチ命令を実行すれば、TLB ミスが発生しないので、プリフェッチが行なわれる．この結果，12%の性能向上を得た (表 4).

方法 2 は，TLB プレロードを行なう．プリフェッチ命令に間に合うように，TLB ミスを起こさせ，当該 PTE を事前に TLB に入れる．この時，プレロードを行なう当該ページ (N 要素先) をアクセスするようにした．N を 0〜64 と変化させた．表 5 に示す通り，4 つ先の要素の場合に，最も性能向上が得られた．

方法 3 では，配列 B の同じ行は，同じ PTE に対応するため，プリフェッチ開始点を，性能向上が得られなかっ

表 5 方法 2 を使用した場合の実行時間

TLB プレロード先 (N)	実行時間 [s]
0	18.1
2	16.1
4	12.2
16	13.0
32	16.5
48	25.7
64	26.3

表 6 方法 3 を利用した場合の実行時間

PTE 数	実行時間 [s]
64	13.8[s]
48	13.2[s]
36	12.8[s]
32-8	11.9-12.2[s]
4	12.5[s]
2	14.3[s]
0	18.0[s]

```
for(i=0;i<N;i++){
   for(j=0;j<N;j++){
      for(k=0;k<N;k++)
        c[i][j] += a[i][k]*b[j][k];
   }
}
```

図 2 検証用転置版行列積プログラム

た場合より，PTE N 個分前にすれば，PTE が TLB にまだ入っているはずである．プリフェッチ開始点を 0〜64 と変化させ，測定した結果は表 6 のようになった．

PTE が TLB に入っている確率が高い PTE の数が 8 から 32 の間では，高い性能を示した．

このように，アドレス変換処理を考慮しなければ，プリフェッチは意図通り行なえない．このため，これを自動化するための方法論を確立する必要がある．また，方法 3 はアクセスパタンに依存してしまうため，該当 PTE が存在しない場合等もあり得るが，方法 2 なら PTE を計算して入れることができる．このため，最高性能が得られるとは限らないが，性能安定化向きである．

6.6 余計なプリフェッチ命令の実行オーバヘッドの検証

ハードウェアプリフェッチが効く状況で，余計となるプリフェッチ命令も併用してしまったときの性能を評価する．今回は，ライブラリの手順 2 で，行列 B の転置とページアラインを行なったと仮定する．プログラムは，図 2 の通りである．この例では，配列 B へのアクセスは，シーケンシャルアクセスであるため，ハードウェアプリフェッチが効いていると考えられる．プリフェッチは，プリフェッチ先をキャッシュラインの 1 つ先，8 つ先と変化させ，毎キャッシュライン間隔に行なった．配列 B については，各 j ループで，2 ページをシーケンシャルアクセスしているので PTE が TLB から追い出されることはないと考えられる．実行時間を比較した (表 7)．

結果，余計なプリフェッチ命令の挿入において，性能差は 1%未満だった．この結果より，余計なプリフェッチ命令の実行は，性能上支障ないと考えられる．また，アドレス変換処理を考慮しなかった場合に，プリフェッチ命令を全乗算で行なった時に性能劣化が 2.3%だったことを考えると，自動化のシンプルさを追及した場合の余計なプリフェッチ命令の実行は，性能上問題ない範囲であると考えられる．これにより，自動調整のためにパラメタ化したプリフェッチ命令を事前に埋めこむ実装方法は有用であると考えられる．

表 7 転置版行列積演算の実行時間

	実行時間 [s]
プリフェッチなし	8.16
prefetcht0(キャッシュライン 1)	8.19
prefetcht0(キャッシュライン 8)	8.10

予備評価結果より，アドレス変換処理を考慮する必要性があり，また，実装上余計なプリフェチ命令の実行も生じるが，性能上問題ないと考えられるため，本ライブラリを利用した性能安定化が期待できる．

7. 関連研究

実行環境毎にキャッシュミス等を防止する自動チューニング機構を有するライブラリには，PC クラスタ向けの並列ライブラリの自動チューニングを行なう ABC-LIB[2] 等がある．また，配列の再構成と動的な再配置によって，キャッシュのセット競合を回避する性能安定化機構を自動チューニングライブラリに追加する方式については[3] が研究されている．[4] では，Xeon プロセッサ向けに，TLB エントリ数も考慮したブロックサイズの最適化等を行なっている．また，最適化において，ロード命令を使ったデータプレロードも行なっている．

8. おわりに

本稿では，性能安定化の一つの実現方法として，アドレス変換を考慮したプリフェッチ命令の実行をライブラリで自動調整する方法を検討した．また，その際の課題について，一考察を与え，予備評価を行なった．その結果，本ライブラリにより，性能安定化が期待できることが確認できた．今後は，予備評価結果を基に，アドレス変換処理を考慮してプリフェッチ命令の実行を自動調整するライブラリの設計及び実装を行なう．

参考文献

1) Intel: IA-32 Intel Architecture Optimization.
2) 自動ブロック化・通信最適化ライブラリ ABC-LIB: http://www.abc-lib.org/.
3) 今村俊幸, 直野健: キャッシュ競合を制御する性能安定化機構内蔵型数値計算ライブラリについて, ハイパフォーマンスコンピューティングと計算科学シンポジウム (HPCS2004), pp. 173-180 (2004).
4) 成瀬彰, 住元真司, 久門耕一: Xeon プロセッサ向け Linpack ベンチマーク最適化手法とその評価, 先進的計算基盤シンポジウム (SACSIS2004), pp. 287-294 (2004).