2004－HPC－99　(26)
2004／7／31

マルチスレッドを用いてページ転送遅延を隠蔽するソフトウェアDSMシステム

立川　純[†]　小出　洋[†]

1ノード上でマルチスレッドを実行し，リモートページ転送時に，別のスレッドに切り替えることでページ転送遅延を隠蔽するソフトウェアDSMを実装した．計算を行うスレッドとは別にスケジューラスレッドを実装し，各計算スレッドはページフォルトハンドラから，TCPソケットを介してスケジューラスレッドにフォルトの発生を通知する．スケジューラスレッドはページ転送要求を送信すると共に，別の計算スレッドを起動する．16ノードのクラスタを用いた評価では，不規則な配列参照を含んだNPB CGベンチマークにおいて，4スレッド実行時に，1スレッド実行時の17%の性能向上を確認した．

Hiding remote page access latency with multithreading in Software Distributed Shared Memory system

Jun Tachikawa [†] and Hiroshi Koide[†]

We have designed and implemented a software distributed shared memory system, hiding remote page access latency with multithreading. This system consists of two types of thread, computational thread and scheduler thread, which are connected with local TCP socket each other. When the page fault occur by a computational thread, page fault notice is sent to scheduler thread via local socket from page fault handler. After receiving the page fault notice, scheduler thread send a remote page request to home node of requested page, and invoke a computational thread which is not executing currently. The experiment on 16 processors shows that NPB CG benchmark with 4 computational thread runs 17% faster than that with 1 computational thread.

1. はじめに

近年のマイクロプロセッサの高性能化とネットワーク技術の発展により，高性能な並列計算環境として，PCやEthernet等のコモディティパーツを利用したPCクラスタが利用されるようになっている．一般的にPCクラスタでは，MPI等のメッセージパッシング型ライブラリを用いて並列プログラムを記述する．一方，ソフトウェアDSMシステムを用いれば，クラスタのような分散メモリ上でも共有変数を用いたプログラミングが可能となり，並列プログラムの開発がより容易に行えると言われている．さらに，ソフトウェアDSMを下位のランタイムとしてOpenMPコンパイラを構築する研究にも注目されている．

ページベースと呼ばれる一般的なソフトウェアDSMでは，仮想記憶管理機構を利用してプロセッサによるメモリ参照命令でページフォルトを発生させ，フォルトハンドラ内で，参照されたページの所有ノードからそのページをローカルに転送し，その後，元のメモリ参照命令に復帰することで共有メモリ空間を実現する．このようなソフトウェアDSMでは，共有空間にアクセスされた時点で上述のページ転送通信が発生し，その間，プロセッサに割り当てられた計算は停止する．従って，ページ転送の遅延時間を効率よく削減ないしは隠蔽し，計算の停止時間をできるだけ削減することがソフトウェアDSMの性能向上にとって重要である．以降では，各プロセッサに割り当てられた計算を行うスレッドを計算スレッドと呼ぶ．

例えば各ページの管理ノードをホームノードとして固定し，ページ要求時/同期時に当該ページのホームノードとのみ通信を行う，ホームベースと呼ばれる実装では，ホームノードが自分の管理するページを参照した場合に

† 九州工業大学 情報工学部 知能情報工学科
Department of Artificial Intelligence, Kyushu Institute of Technology

はローカルメモリを直接参照するだけで良く，通信は発生しない．従って，ホームノードの割り当てやプログラミングによる参照パターンの調節を行って，各計算スレッドが実行しているノード自身がホームになっているページを多く参照するように最適化を施すことが上記の問題の対策として挙げられる．

しかしながらそのような最適化を施しても完全にリモートページ参照が削除できるわけではなく，配列データが複数スレッドからランダムに参照される場合のように，一つのページに対する複数ノードからの参照を回避することが困難なアプリケーションが存在する．従ってページ転送時の計算スレッドの停止を回避するための別の手法が必要である．

本研究ではこの方法として，1 ノードに登載されたプロセッサの数に関わらず複数の計算スレッドを効率的に実行させ，ある計算スレッドの停止時に別の計算スレッドに実行を切り替えることでページ転送の遅延を隠蔽する手法について着目する．本稿では，今回実装したマルチスレッドを用いたソフトウェア DSM についてその実装方法とその初期評価について述べる．

2. 1 ノード上のマルチスレッド実行に対する検討

2.1 ページ転送遅延の隠蔽

ページ転送中に計算が停止する以上，同時に一つのページ要求しか発生しない単一スレッド実行の場合では，リモートページ参照が発生した分だけ計算が停止し，全体性能が低下する．

計算スレッドを複数実行させ，ページ転送中にも別のスレッドによる計算を行うことで，ページ転送の遅延を別の計算で隠蔽し性能劣化を押さえる事が可能となる．ただし，どの程度劣化を押さえ得るかはアプリケーションのメモリ参照パターンに依存し，各計算スレッドの参照する領域が，スレッド毎に異なるような場合に効果が大きいと考えられる．

2.2 複数ページ転送の可能性

単一スレッド実行では，共有メモリへの参照がが必ず一つずつ逐次的に処理される．即ち，ページ転送の要求通知と，その返答としてのページデータの送信が繰り返される．複数の計算スレッドによってページ転送中に別の計算を実行させることは，ページ転送が完了しないうちに，割り当てられた別の計算自身もページ転送を要求する可能性を有している．このように複数のページ要求が発生する可能性があり，逐次的にページ転送を行う場合と比べて擬似的なプリフェッチ効果を得ることができる．

2.3 マルチスレッドによる影響

マルチスレッドを用いることの欠点として，コンテキストスイッチのオーバヘッドや，プロセッサ内部のキャッシュや TLB のミスヒットの上昇を促す等が挙げられる．従ってこれらの事項を考慮した計算スレッドのスケジューリングが本質的には重要である．

2.4 実装の指針

複数のスレッドによる CPU やキャッシュミスヒット増加による性能への影響を極力回避するため，実行するスレッドは同時に 1 つと限定し，そのスレッドがページフォルトによって停止された段階で別のスレッドに切り替えるようなスケジューリングを行う．

3. 実　　装

3.1 ページフォルト処理の実現

1 ノード上でのマルチスレッド実行に基づくソフトウェア DSM の設計にあたって，一般的なスレッドライブラリである pthread を用いた実装を検討する．

前節で述べたように，極力 CPU 資源の奪い合いによる性能劣化を避けるためには，同時に実行するスレッド数を制御する必要がある．つまり，ページフォルトを契機としたスレッドスケジューリングを行うことになるが，そのためにはページフォルトハンドラから他の計算スレッドに対する通信や同期を行う必要がある．しかしながら，pthread が提供する関数は，ページフォルトハンドラ等の非同期シグナルハンドラから呼び出した場合の動作が未定義である[1] ため，排他制御 (mutex) や条件変数を用いたスケジューリングは行えない．スレッド間の同期や通信を行うために，pthread 関数以外の手段を用いる必要がある．

そこで，図 1 に示すように計算スレッド以外にスケジューラスレッドを別途実装し，計算スレッドとスケジューラスレッドとの通信に read/write システムコールによるソケット通信を用いる．即ち各計算スレッドは各々フォルトハンドラを起動し，ハンドラ内でフォルトを発生したアドレスをローカルソケットを介してスケジューラスレッドに通知する．その後，read システムコールでスケジューラスレッドからの応答パケットをブロック受信す

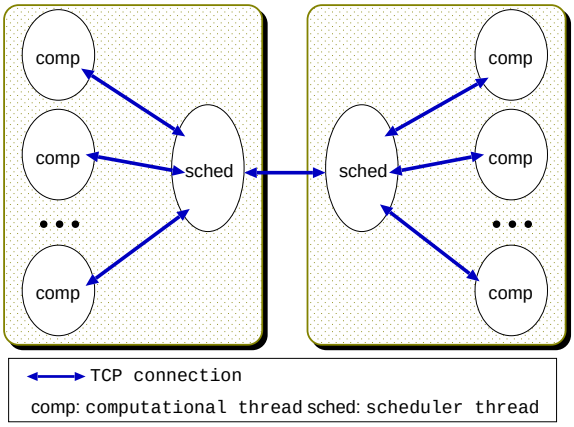

図 1 マルチスレッドに基づくソフトウェア DSM

ることでハンドラ処理の終了を認識し，かつハンドラ処理の間 CPU 資源を解放する．ただし，この方法では実行中の計算スレッドをスケジューラスレッドの側から停止させるようなプリエンプティブな制御は実現できない．

3.2 スケジューラスレッドの実装

スケジューラスレッドは select システムコールで複数の計算スレッドからの通知を非同期的に待ち，通知を受けたアドレスを含むページの状態に応じた処理を行う．またスケジューラスレッド同士も同様にソケット通信を用い，計算スレッドからの非同期通知の監視と同様 select システムコールによって非同期 I/O を実現する．こうすることでスケジューラスレッドが一括して非同期イベントを監視することになり，実装が単純化される．

スケジューラスレッドは，計算スレッドから通知されるページフォルト処理だけでなく，後述するバリア同期や共有メモリの一貫性管理等の処理を行う．

3.3 共有メモリの管理

共有メモリ領域は，スケジューラスレッドによってページ単位に管理され，各ページ毎にホームノードが割り当てられる．自ノードがホームノードであるページは初期状態を RO(Read Only) に設定され，他ノードがホームであるページは無効として設定される．計算スレッドが無効ページに参照すると，その計算スレッドからスケジューラスレッドに対してフォルトが通知され，スケジューラスレッドはホームノード上のスケジューラスレッドに対してページ転送要求を送信する．転送完了直後，そのページは RO として設定され，計算スレッドにフォルト処理の終了を通知する．そしてその後の書き込みによるフォルト処理で RW(Read Write) に変更され，スケジューラスレッドはページの複製 (twin) を作る．ホームノードが自らが管理するページに書き込んだ場合は，twin は作成せず書き込んだ事実だけを記録する．

バリア同期時，非ホームノードはキャッシュしたページ全てに対して，各ページのホームに当該ページとそれに対応する twin との差分 (diff) を送信する．また，読み込みしか行っていないノードは diff を含まないパケットだけを送信する．

ホームノードは自らが管理するページの diff を，キャッシュした全てのノードから回収し，当該ページを最新の状態に更新する．ページが diff によって更新された場合は，続いてそのページをキャッシュした全てのノードに対してページの無効化を通知する．無効化通知を受けたノードはそのページを無効状態に設定し，それ以後のアクセスでは再びページフォルトが発生する．

3.4 提供される API

本システムが提供する主な API を表 1 にまとめる．API の動作は pthread をベースに設計した．1 ノード上で複数の計算スレッドを実行するプログラムを記述する場合には，計算スレッドの生成関数を階層的に呼び出すことで行う．

各計算スレッドには，グループ属性があり，同じグループ変数が生成関数に渡されることによって，そのグループに属する計算スレッドが生成される．また，バリア同期は同じグループに属するスレッド間でのみとられる．従って全ノードに跨る全ての計算スレッド間のバリア同期を行う場合は，まずローカルな計算スレッド間でのバリア同期を取り，次にノード上の代表計算スレッド間でバリア同期をとることで行う．また，メモリ一貫性制御については，API 上は現在別になっており，ローカルスレッド間のバリア同期をとってから，各ノードの代表となる計算スレッドがメモリ一貫性制御付きバリア同期関数を呼び出すことで実現する．

また，`mdsmCompWait` 関数によって明示的に計算スレッドを停止することができ，実行を継続する計算スレッドをプログラミングレベルで制御できる．この関数が呼び出された計算スレッドは次節で述べる ready 状態に遷移し，他の計算スレッドが計算を継続できなくなった時に `mdsmCompWait` から復帰し，実行を開始する．

これについて以下のコードを用いて説明する．4 ノードで計算を行い，各ノードでは 2 つの計算スレッド，全

表 1 提供される API
Table 1 Provided APIs

mdsmCreate	計算スレッドの生成
mdsmJoin	計算スレッドの終了待ち
mdsmMemAlloc	共有メモリの割り当て
mdsmMemFree	共有メモリの解放
mdsmBarrier	バリア同期
mdsmMemSync	メモリ一貫性制御付きバリア同期
mdsmCompWait	計算スレッドの停止

体で 8 つの計算スレッドを生成している状況を想定する．また，コード中の comp_id は各ノード上のローカルな計算スレッドの ID(0 または 1) で，計算スレッドはそれぞれグローバルな ID(0〜7) が別に振られているとする．

```
1: if(comp_id!=0) mdsmCompWait(comp_id);
2: lb = 1; ub = 400; step=1;
3: iter(&lb, &ub, &step);
4: for(i=lb; i<=ub; i+=step)
5:       A[i] = ...;
6: mdsmBarrier(comp_id);
7: if(comp_id==0) mdsmMemSync();
8: mdsmBarrier(comp_id);
```

- 1 行目で代表となる計算スレッド (comp_id==0) 以外が停止する．
- 2〜3 行目で，自分のグローバルなスレッド ID から計算領域を特定する．
- 4〜5 行目で，共有変数 A へのアクセスが発生する．このときページフォルトが発生すると，スケジューラスレッドは 1 行目で停止した comp_id==1 の計算スレッドの実行を開始する．
- 6 行目に先に到着した計算スレッドは，ローカルバリア同期待ち状態に入る．この時同時に，1 行目か 5 行目で停止していたスレッドの実行が再開される．
- 7〜8 行目．代表計算スレッド (comp_id==0) によるメモリ一貫性操作と，他のローカル計算スレッドとの待ち合わせ

3.5 計算スレッドのスケジューリング

この節では，スケジューラスレッドによるスケジューリングの詳細について説明する．まず，スケジューラスレッドが管理する計算スレッドの状態について述べ，次にスケジューリングの詳細について a) ページフォルトが発生した場合，b) 全計算スレッド間でバリア同期をとる場合についてそれぞれ説明する．

running 計算を実行している状態．スケジューラスレッドは running 状態にある計算スレッドの数を一つに制限して CPU の奪い合いを回避する．

ready 計算の割り当てが可能な状態．running 状態にある計算スレッドがページフォルトやローカルバリア同期に到達して計算の続行ができなくなった時，ready 状態にある計算スレッドの中から次に running になるスレッドが選択される．

wait ページフォルトやバリア同期が進行中の状態．各々の処理 (ページ転送やバリア同期) の終了後，ready 状態へ遷移する．

3.5.1 ページフォルト発生時のスケジューリング

フォルト通知で指示されたページのホームノードに対してページ要求を送信すると共に，フォルトを発生した計算スレッドの状態を wait に変更する．この時，running 状態の計算スレッドがすでに一つあればスケジューラスレッドは動作を終了するが，そうでなければ ready 状態のスレッドの一つを running 状態に遷移させ実行を開始する．ready 状態のスレッドがなければなにもせず終了する．

また要求したページが到着する前に，別の計算スレッドが同じページを参照してもフォルトが発生し，スケジューラスレッドはフォルトした計算スレッドを内部に記録し，同様に ready 状態のスレッドから一つを選択する．

要求したページが到着すると，そのページ参照によって wait 状態となっていたスレッドをすべて ready 状態に遷移させる．

3.5.2 バリア同期時のスケジューリング

基本的には 3.5.1 節での，ページフォルトをバリア同期と読み替えることで説明できる．バリア同期を通知した計算スレッドを wait 状態に遷移させ，その時の ready 状態のスレッドを選択し実行を開始する．

バリア同期の成立時には，同じグループに属した計算スレッドをすべて ready 状態に遷移させ，その時 running 状態である計算スレッドは基本的にないので，同時に ready から一つを選択し実行を開始する．

4. 性 能 評 価

4.1 複数スレッドによるページ転送効率

まず，複数スレッドによって発生する複数のページ転送によってどの程度のスレッド数まで効果が得られるのかを調べる． 具体的には，2 つの PC を用いて，大きな配

列の全要素に片方が 0 を書き込む単純なループを実行する．ただし，その配列の全領域はもう一方の PC がホームになっており，全ページでフォルトが発生する．ホーム側の計算スレッドは一切動作せずスケジューラスレッドがページ転送要求に応じるだけである．

また，書き込みの最後にはメモリ一貫性制御を行い，ホームに対してデータを更新する時間を含む．さらに，純粋なページ転送時間の変遷との比較を行うために，読み込みだけの時間についても測定した．なお，評価環境には表 2 に示す PC が 16 台で構成されるクラスタを用いた．

1024 ページの領域について，書き込みを行うノード上で実行する計算スレッドの数を 1〜8 まで変化させた結果を図 2 に示す．読み込み時間 (図中，read) はスレッド数の上昇に応じて減少することを確認した．なお，図 2 の 1 スレッドでの読み込み時間から，1 ページ辺りの読み込みコストは，およそ 632usec である．

一方書き込み時間 (図中，write) は 4 スレッド以降では大きく変化しなかった．メモリ一貫性制御は自体はスケジューラスレッドによって実行されるため，計算スレッドの増加に伴ってメモリ一貫性制御のオーバヘッドが増加するとは考えにくい．メモリ一貫性制御付きバリア同期を実行する前に，すべての計算スレッドによって書き込みの終了を待つためにバリア同期をとっておりこれが測定時間に含まれているため，オーバヘッドの原因はローカルスレッド間のバリア同期であると考えられる．

4.2 ベンチマークによる評価

ベンチマークプログラムとして次の二つを用いる．いづれもベースとなるソースコードを本システム向けに変更したものを用いた．それぞれのベンチマークで 1 ノード上で実行するスレッド数を 1〜4 まで変化させたときの実行時間を測定する．

- laplace

  ヤコビ法による laplace 方程式の求解プログラムで，Omni OpenMP コンパイラ[2] に付属するものを用いた．行列サイズは 2000×2000，反復回数は 50 とした．このプログラムは共有変数への参照パターンがループ変数で決まっているため，比較的ローカリティが高い．

- NPB CG

  RWCP で開発された C 言語 OpenMP 版の NAS Parallel Benchmarks 2.3 をベースに，本システム用に変更したものを利用した．問題サイズは CLASS A を用いた．CG のカーネルループでは不規則な配列参照が行われるため，比較的ローカリティが低い．

表 2 評価環境

Table 2 Evaluation environment.

CPU	Pentium4-2.4GHz
Memory	1GBytes
Network	1000BASE-T(ns83820)
OS	RedHat7.3

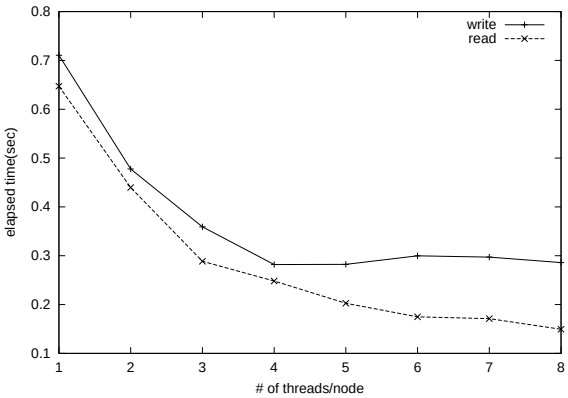

図 2 ページ転送時間の変遷

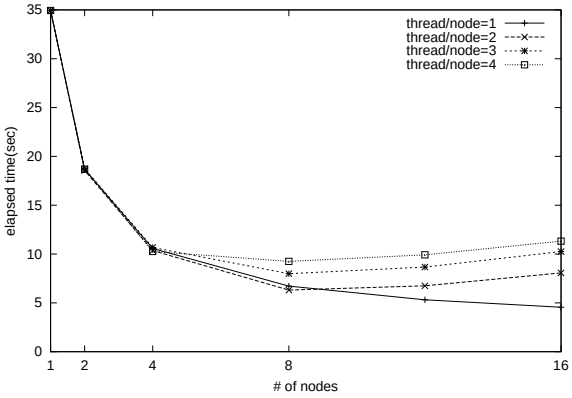

図 3 laplace の実行結果

Fig. 3 execution result of laplace

4.2.1 laplace の評価結果

laplace の実行結果を図 3 に示す．1 ノード上での実行スレッド数が 1 のときは 16 ノードでおよそ 7.7 倍の性能向上を達成した．それに対し，スレッド数を 1 より大きくした場合には，8 ノードで 2 スレッドの場合にはわずかに 1 スレッド時の性能に勝っているが，全体的に複数スレッドによる性能向上は達成できていない．

laplace では，各共有変数のホーム割り当てとループ分

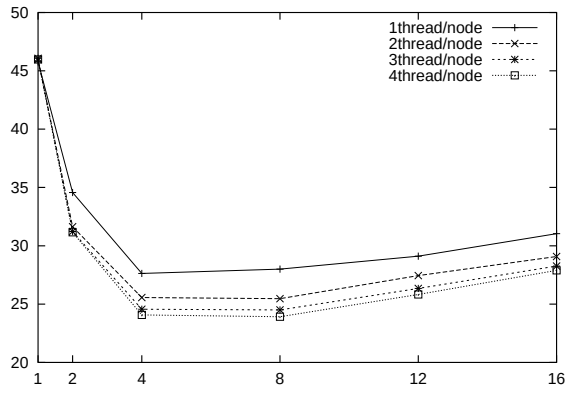

図 4 CG の実行結果

Fig. 4 execution result of CG

割の仕方を，ブロック分割することでローカリティを確保することが容易である．従って，ほとんどのメモリ参照でリモートページ転送を発生することがなく，ローカルな計算スレッド間の制御オーバーヘッドがそのまま全体の実行時間の増加を招いたものと考えられる．

4.2.2 NPB CG の評価結果

CG の実行結果を図 4 に示す．全体的に 8 ノード以降ではスケーラビリティを得られていないことが分かる．これは現在のメモリ一貫性同期時のオーバヘッドが大きいためと考えているが，その解明は今後の課題である．

スレッド数が 2 以上の場合で，スレッド数の増加に伴って実行時間が減少していることが確認できる．最大で，4 スレッド (8 ノード実行時) で 17%の高速化を達成することができた．

CG は間接参照による不規則な配列参照のため laplace と比べてローカリティが低く，1 スレッドでの実行時には頻繁にページフォルトが発生していたと考えられる．マルチスレッドを導入することで，複数のリモートページ参照がほぼ同時に発生し，全体的にページ転送遅延を隠蔽することができたものと思われる．

5. おわりに

本稿では，ページフォルトの発生時に別のスレッドに実行を切り替えることでページ参照遅延を隠蔽するソフトウェア DSM について，我々が行った実装方法とその初期評価について述べた．

ページフォルトを契機として，ページ転送中にスレッドスケジューリングを行うためにはフォルトハンドラからスレッド間の通信を行う必要がある．我々はローカルソケットを用いてフォルト処理の通知とスレッドスケジューリングを実現した．ローカルソケットを用いることによるオーバーヘッドがどの程度であるかの解析は今後の課題である．

ベンチマークによる評価では，NPB CG のような不規則な配列参照がある場合にはページフォルトの発生頻度が高く，マルチスレッドを導入することで計算の停止を回避することが可能であることが分かった．

謝辞 本研究の一部は，文部科学省科学研究費補助金 (課題番号 16016271，15700063)，及び総務省の援助により行なわれた．

参 考 文 献

1) ：Pthreads プログラミング, O'REILLY (1998).
2) ：Omni OpenMP Compiler Project (http://www.hpcc.jp/Omni).