計算機アーキテクチャ 119−21
（1996. 8. 28）

分散共有メモリ型計算機クラスタにおける遅延キャッシュ・コヒーレンシ・プロトコルの性能評価

中村秀一[†,††] 数藤義明[†] 福井俊之[†,††]
濱口一正[†,††] 柴山茂樹[†,††]

分散共有メモリ型計算機クラスタのシミュレータを利用して，遅延キャッシュ・コヒーレンシ・プロトコルの性能評価を行った．シミュレーションにより，本プロトコルが分散共有メモリ型計算機クラスタのようなメモリ・アクセス・レイテンシの大きなシステムにおいてキャッシュ・ライン・サイズを大きくした場合には有効性を示すことがわかった．

Performance Evaluation with Deferred Cache Coherence Protocol in Distributed Shared Memory Computer Cluster

Shuichi Nakamura,[†,††] Yoshiaki Sudo,[†] Toshiyuki Fukui,[†,††]
Kazumasa Hamaguchi[†,††] and Shigeki Shibayama [†,††]

In this paper, we give some simulation results of a Deferred Cache Coherence Protocol. Our protocol is aimed at enhancing performance of a class of shared memory computer cluster systems having relatively large remote memory access latencies. Our protocol assumes weak memory consistency model, which is utilized to combine plural deferred cache coherence maintenance transactions. The simulation disclosed that proposed protocol consistently outperforms MESI protocols in all simulation runs with cache line sizes of 64 bytes or greater.

1. はじめに

筆者らは広帯域な通信路を用いて複数の計算機を接続し，分散共有メモリを実現する分散共有メモリ型計算機クラスタ☆について研究を行っている[1)2)]．分散共有メモリ型計算機クラスタは，密に結合された分散共有メモリ型マルチプロセッサシステムと比較して，ノード間メモリ・アクセス(リモート・メモリ・アクセス)のレイテンシが大きいという特徴を持つ．この分散共有メモリ型計算機クラスタにおいてリモート・メモリを有効に活用するために，筆者らは必要な一貫性保持トランザクションのみを実行する遅延キャッシュ・コヒーレンシ・プロトコルを提案している[3)4)]．

本稿では，遅延キャッシュ・コヒーレンシ・プロトコルを採用した分散共有メモリ型計算機クラスタのシミュレータを利用して，遅延キャッシュ・コヒーレンシ・プロトコルの性能評価を行ったので報告する．

† キヤノン株式会社 情報メディア研究所
Media Technology Laboratory, Canon Inc.
†† 現在，キヤノン株式会社 CyberMedia プロジェクト
CyberMedia Project, Canon Inc.
☆ 通信路に光ファイバを利用したものを光バス計算機クラスタ(Optical Bus Computer Cluster:OBCC)と呼んでいる

2. 分散共有メモリ型計算機クラスタ

効率のよい負荷分散，及び低コストな並列・協調処理を目的として，分散共有メモリ型計算機クラスタが考えられている．分散共有メモリ型計算機クラスタは，低コストで並列/分散協調処理システム構築が可能というワークステーション・クラスタの特長と，低オーバヘッド通信，プログラミングが容易であるという分散共有メモリの特長を併せ持つが，反面リモート・メモリ・アクセス・レイテンシがローカル・メモリ・アクセスと比較して大きいという欠点がある．リモート・メモリに対するアクセス・レイテンシは，分散共有メモリ型計算機クラスタの試作機[1)]の場合では，約4.5μsec とローカルメモリに対するアクセス・レイテンシ 500nsec と比較して，10 倍程度の大きさである．試作機であることを考慮しても，適切な設計，及び最新の技術を利用すればリモート・メモリ・アクセスはローカル・メモリ・アクセスの 10〜100 倍程度で実現可能である．

しかしながら，このようにアクセス・レイテンシの大きいリモート・メモリを有効に活用するためには，リモート・メモリ・アクセス時にコンテキスト・スイッ

チしてレイテンシを隠蔽する手法[7]や，キャッシュ・メモリを利用して平均アクセス・レイテンシを削減する手法[5][6]等の工夫が必要となる．

コンテキスト・スイッチによるレイテンシ隠蔽手法は多数のコンテキストが存在する場合に，システム性能の向上という面に関しては有効であるが，コンテキストが少数である場合や1つのプログラムの速度向上を図りたい場合には有効でない．

キャッシュを利用する手法では，平均アクセスレイテンシが削減されるため，コンテキストが少数である場合や1つのプログラムの速度向上を図りたい場合にも有効である．しかしながら，キャッシュ・メモリによる手法は，マルチプロセッサ・システムのコヒーレント・キャッシュにおいては，コピーの一貫性保持処理のために付加されるオーバヘッドを解決しなければならない．

一般に，キャッシュ・メモリはアクセス・レイテンシの大きいメモリに格納されている複数のデータ・ブロックを1回のメモリ・アクセスで転送し，アクセス・レイテンシの小さいプロセッサ近傍のキャッシュ内に配置して，2回目以降のメモリ・アクセス時に利用することで平均アクセス・レイテンシの削減を図る．したがって，1回のメモリ・アクセスで転送されるデータ・ブロック・サイズ(一般にはキャッシュ・ライン・サイズ)が大きいほど，シーケンシャルにメモリ・アクセスを行う際のヒット率の向上が期待できる．1回のメモリ・アクセスのレイテンシが大きいシステムにおいては，この効果がさらに大きくなる．しかしながら，キャッシュ・ライン・サイズを大きくすると，

- 1回の転送時間が長くなり，プロセッサがメモリをランダムアクセスするような場合には，不要なデータ・ブロックの転送により，かえって性能が低下する．
- キャッシュ・メモリの容量を一定とした場合には容量性のキャッシュ・ミスが誘発される．
- マルチプロセッサ・システムにおいてはデータの一貫性保持は一般にキャッシュ・ライン・サイズと同一(もしくはその1/2)の大きさのデータ・ブロック☆で管理されるため，キャッシュ・ライン・サイズ☆☆を大きくし過ぎると，false sharing に起因する一貫性保持トランザクションの頻発により性能を低下させる場合がある[8]．

などの問題が生じる．

本稿では，アクセス・レイテンシの大きいシステムにおいて，キャッシュ・ライン・サイズを大きくした場合において生じる false sharing 問題に焦点をあてて，以降の議論をすすめることとする．

☆ 本稿では，データの一貫性が保持される単位ブロックをコヒーレンス・ブロックと呼ぶこととする

☆☆ この場合，キャッシュ・ライン・サイズはコヒーレンス・ブロック・サイズに等しい

3. 関連研究

false sharing 問題を解決するために，主に以下の2つの手法がとられている．

(1) データが false sharing されないように共有変数を別々のコヒーレンス・ブロックにソフトウェアで割り当てる手法

(2) 共有変数のサイズに応じてコヒーレンス・ブロック・サイズをハードウェアが変更する手法

(1) の手法については，Torrellas らがコンパイラによって共有変数を別々のコヒーレンス・ブロックに割り当てる手法[9]を提案しており，また，Woo らによる SPLASH2 ベンチマーク[8]では，キャッシュ・メモリのパラメータを意識して，アプリケーション内で明示的にコヒーレンス・ブロック毎に共有変数を割り当てる手法をとっている．

しかしながら，これらソフトウェア的にコヒーレンス・ブロックを共有変数の割り当てる手法では，コヒーレンス・ブロック・サイズが大きく，共有変数のサイズが小さい場合に利用できない領域がキャッシュ・ライン中に発生(フラグメンテーション)し，キャッシュ・メモリの容量性ミスが発生することが Woo らによって報告されている[8]．

(2) の手法については，Dubnicki らが Adjustable Block Size Coherent Cache[11] を提案している．これは，キャッシュ・ラインを2つのブロックに分割し，データ参照パターンのフィードバックによってコヒーレンス・ブロックのスプリット/マージを行い，コヒーレンス・ブロック・サイズをハードウェアで動的に変更する手法である．

しかしながら，Adjustable Block Size Coherent Cache を実際にハードウェアに実装した場合，コヒーレンス・ブロック・サイズはハードウェアに実装したサイズに静的に決まっており，種々のアプリケーションに対して適切なコヒーレンス・ブロック・サイズの初期値設定を行うことはできない．また，ラインを2分割するだけであるので，キャッシュ・ライン・サイズの1/2(すなわち最小のコヒーレンス・ブロック・サイズ)がプログラムのワーキング・セットより大きくなってしまい，false sharing が発生してしまうといった問題がある．したがって，キャッシュ・ライン・サイズを大きくした場合にはコヒーレンス・ブロック・サイズがプログラムのワーキング・セットより大きくなる可能性が高く，問題となる．

そこで，筆者らはこれらの問題点を解決し，キャッシュ・ライン・サイズを大きくした場合でも false sharing の影響が少なくなる遅延キャッシュ・コヒーレンシ・プロトコルを提案している[3][4]．

4. 遅延キャッシュ・コヒーレンシ・プロトコル

遅延キャッシュ・コヒーレンシ・プロトコルでは，緩いメモリ・コンシステンシ・モデルに対応した一貫性保持を行うことを特徴とする[3)4)]．具体的には，

(1) プロセッサがメモリ・アクセスを発行した時点では，即時に一貫性保持処理トランザクションを発行せず，緩いメモリ・コンシステンシ・モデルで定義された同期点までを最大限として発行を遅延し，ある契機により一貫性保持処理トランザクションを発行する．

(2) 遅延された複数の一貫性保持処理トランザクションをまとめて発行する．

ことにより，不要な一貫性保持処理トランザクション数を削減することが可能であることが特徴である．

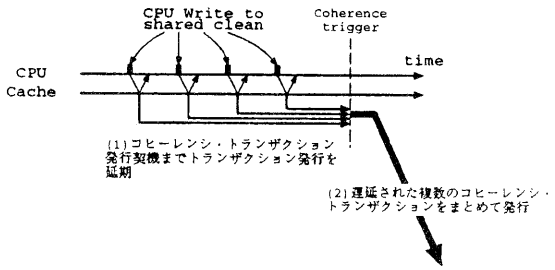

図1　遅延キャッシュ・コヒーレンシ・プロトコルの特徴

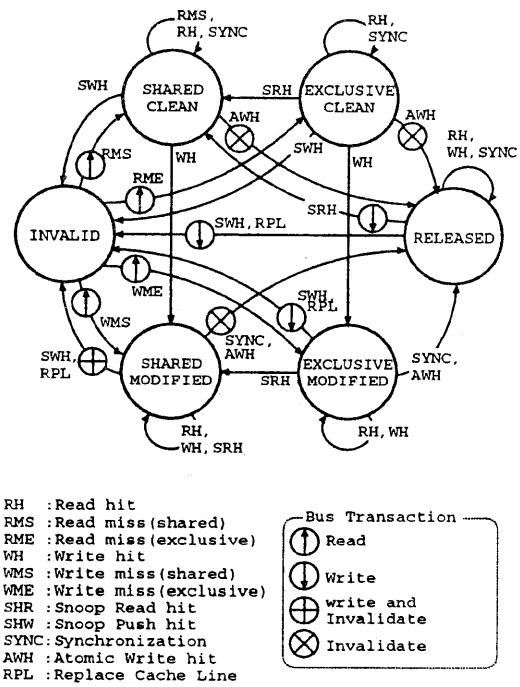

図2　遅延キャッシュ・コヒーレンシ・プロトコルの状態遷移

4.1 プロトコルの実装例

図2に遅延キャッシュ・コヒーレンシ・プロトコルの一実装例の状態遷移を示す．データの一貫性はコヒーレンス・ブロック単位で管理され，キャッシュ・メモリに対して，CPU からのアクセスやバスからのスヌープの結果によって状態が遷移する．なお，ロック等で同一アドレスに対するライト・アクセスのデータの一貫性を保持したい場合は，特別なライト・アクセス(アトミック・ライト)を使用しなければならない．

遅延キャッシュ・コヒーレンシ・プロトコルが他のプロトコルと異なり，特徴となっている点は，共有状態(SHARED)にあるコヒーレンス・サブ・ブロックに対してライト・ヒットした際に，一貫性保持トランザクションが即時に発行されずに，遅延される点である．遅延された一貫性保持トランザクションは，ある一貫性保持契機に従って発行される．図2の場合の一貫性保持契機は，同期(SYNC)アクセス，又はアトミック・ライト・ヒットである．

4.1.1 キャッシュ・メモリの設計例

図3に遅延キャッシュ・コヒーレンシ・プロトコルを採用したキャッシュ・メモリの構成を示す．通常のキャッシュ・メモリと異なる点は，一貫性保持トランザクションが遅延されているという情報を状態フラグ中に持つ点である．さらに，本設計例では1キャッシュ・ライン中に複数のコヒーレンス・ブロック(コヒーレンス・サブ・ブロック)を設け，CPU の同期アクセスを一貫性保持契機として，当該キャッシュ・ライン中の一貫性保持が未解決のコヒーレンス・サブ・ブロックの一貫性保持を行う．トランザクション・ビットマップは，どのコヒーレンス・サブ・ブロックに対して一貫性保持を行うかを指定する．したがって，一貫性保持トランザクションは，従来のアドレス，データ，トランザクションの種別の情報にトランザクション・ビットマップが付加される形となる．

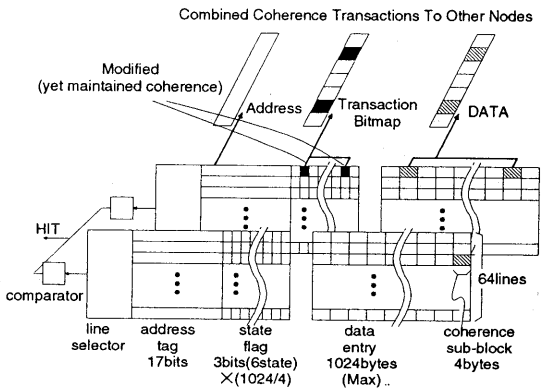

図3　キャッシュ・メモリの構成

5. 性能評価

5.1 シミュレータ

分散共有メモリ型計算機クラスタをモデル化したシミュレータ上で2つのベンチマーク・プログラムを実行させて性能評価を行った．シミュレータは実行駆動型(execution driven)である．シミュレータ上の仮想分散共有メモリ型計算機クラスタは8ノードのワークステーションがバスによって結合された構成であり，各ノードのCPUはsparc version8の命令，及びいくつかの独自トラップ命令を実行する．メモリは各ノード内に分散して配置され，ディレクトリによってキャッシュの一貫性が保持される．

シミュレータ上の仮想分散共有メモリ型計算機クラスタを図4に，そのシステム・パラメータを表1に示す．

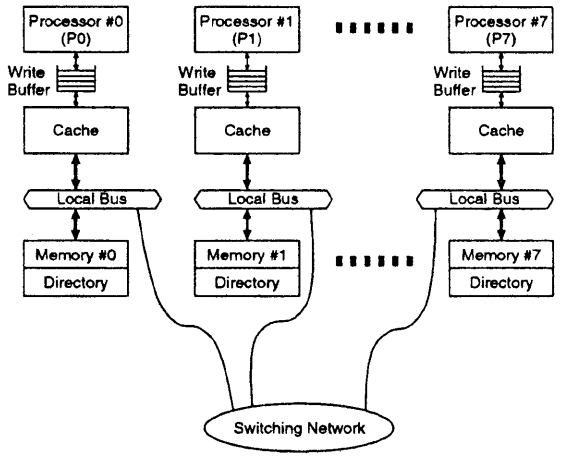

図4　仮想分散共有メモリ型計算機クラスタ

表1　システム・パラメータ

ライト・バッファ		
エントリ数	4	
キャッシュ・メモリ		
ウェイ数	8(LRU)	
ライン数	1~128	
キャッシュ・ライン・サイズ	16~2048	byte
キャッシュ・サイズ	512	Kbyte
コヒーレンス・ブロック・サイズ	8~2048	byte
アクセス・レイテンシ		
ライト・バッファ	1	cycle
キャッシュ・メモリ	1	cycle
DRAMメモリ	10	cycle
グローバル・バス	100	cycle/1way
転送帯域幅		
ローカル・バス	8	byte/cycle
グローバル・バス	8	byte/cycle

5.2 ベンチマーク・プログラム

ベンチマーク・プログラムに2048要素の整数Quicksortとsplash2[8]の1024点複素FFTを用いた．各プログラムはgcc 2.7.2の-mv8 -O4オプションでコンパイルしている．

5.3 性能評価結果

他のキャッシュ・コヒーレンシ・プロトコルの例としてMESIプロトコルでの結果と比較した．

図5はキャッシュ・ラインをコヒーレンス・サブ・ブロックに分割しない場合において，キャッシュ・ライン・サイズを変化させたときのベンチマーク実行時のMESIプロトコルと遅延キャッシュ・コヒーレンシ・プロトコルの一貫性保持トランザクション数を示している．図6はキャッシュ・ラインを複数のコヒーレンス・サブ・ブロックに分割した場合において，キャッシュ・ライン・サイズ=512bytesでコヒーレンス・サブ・ブロック・サイズを変化させたときのベンチマーク実行時のMESIプロトコルの一貫性保持トランザクション数，及びベンチマーク実行時間を示している．図7は図6と同一の条件で遅延キャッシュ・コヒーレンシ・プロトコルの場合の一貫性保持トランザクション数，及びベンチマーク実行時間を示している．

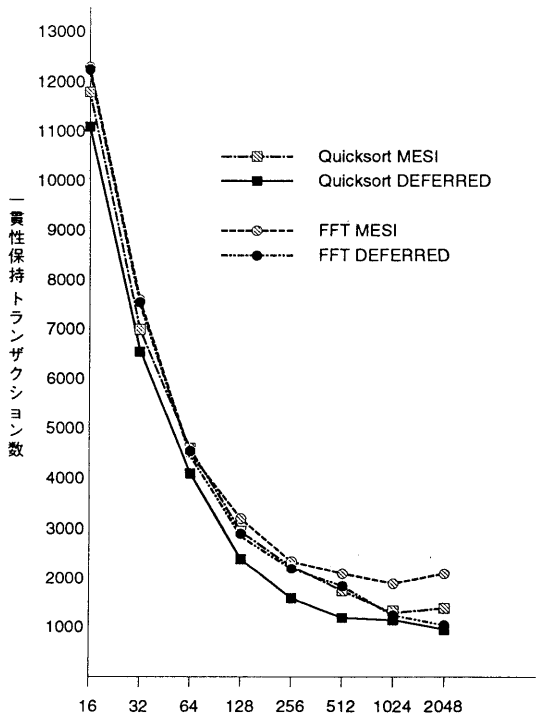

図5　一貫性保持トランザクション遅延効果

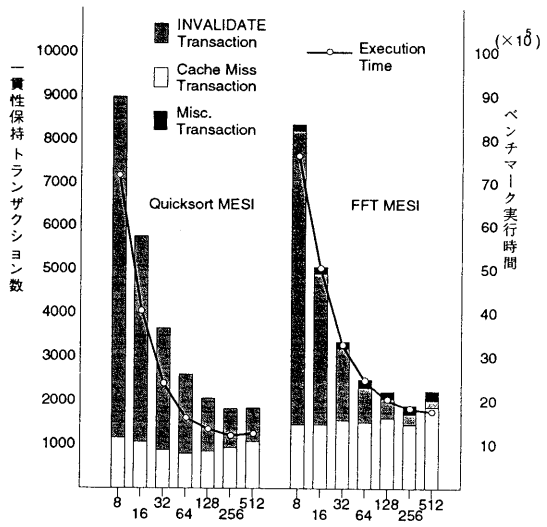

図6　コヒーレンス・ブロック分割効果

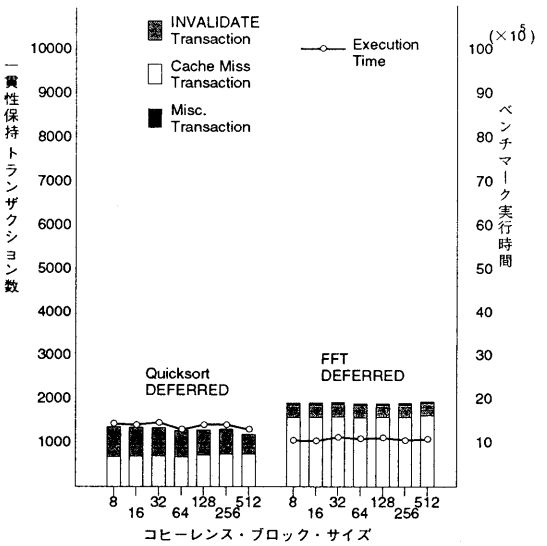

図7　遅延+コヒーレンス・ブロック分割効果

5.4 考　　察

図5はキャッシュ・ライン・サイズを変化させたときの一貫性保持トランザクションを遅延させることの効果をみるための図である．図5を見ると，どのキャッシュ・ライン・サイズにおいても，●や■の点で表されている遅延キャッシュ・コヒーレンシ・プロトコルの方が無効化トランザクション数，及びキャッシュ・ミス・トランザクション数ともに下回っており，特にキャッシュ・ライン・サイズが大きい領域では差が大きくなっている．理由としては，遅延キャッシュ・コヒーレンシ・プロトコルではfalse sharingされているデータ・ブロックがただちに無効化されないため，キャッシュ・ミスが減少して，一貫性保持トランザクションが頻発していないためと考えられる．このことから一貫性保持トランザクションを遅延することは，false sharingの影響を受けにくくする効果があるといえる．

図6はキャッシュ・ラインを複数のコヒーレンス・サブ・ブロックに分割することの効果，及び，一貫性保持トランザクションの内訳をみるための図である．図6を見ると，キャッシュ・ライン・サイズが大きく，コヒーレンス・サブ・ブロック・サイズが大きい場合に，FFTベンチマークの方ではMESIプロトコルはfalse sharingの影響を受けていることがわかる．shared blockの変数にライト・ヒットした場合の無効化トランザクション数が大幅に増加し，かえって性能が低下している．

図7は一貫性保持トランザクションを遅延したことと複数のコヒーレンス・サブ・ブロックを混合したことの効果，及び，一貫性保持トランザクションの内訳をみるための図である．図7を見ると，キャッシュ・ライン・サイズが大きく，コヒーレンス・サブ・ブロック・サイズが小さい場合には，遅延キャッシュ・コヒーレンシ・プロトコルはfalse sharingの影響を受けなくなり，かつ，shared cleanの変数にライト・ヒットした場合においても無効化トランザクションが遅延・統合されるため，無効化トランザクション数の増加は見られない．コヒーレンス・サブ・ブロック・サイズの変化に対して，一貫性保持トランザクション数の変化があまり見られないことから，一貫性保持トランザクションの遅延効果の影響がより大きく，コヒーレンス・ブロックの分割効果が隠蔽されてしまっているものと考えられる．

また，図には示していないが，キャッシュ・ライン・サイズを必要以上に大きくし過ぎると，一貫性保持トランザクション数にはほとんど変化がないが，不要なデータ転送が行われて実行時間が増加することもわかっている．

以上のことより，分散共有メモリ型計算機クラスタのようなリモート・メモリ・アクセスの遅延が大きいシステムにおいて，キャッシュ・ライン・サイズを大きくした場合には，コヒーレンシ保持を遅延することの効果が大きく，コヒーレンス・ブロックを分割することは，悪影響が大きいことがわかった．

6. おわりに

分散共有メモリ型計算機クラスタのシミュレータを利用して，遅延キャッシュ・コヒーレンシ・プロトコルの性能評価を行い，シミュレーション結果から得られたデータについて考察した．

シミュレーション結果から，遅延キャッシュ・コヒーレンシ・プロトコルでは，メモリ・アクセス・レイテンシが比較的大きい場合において，キャッシュ・ライン・サイズを大きくした場合においても，false sharingの悪影響を受けず，キャッシュ・ライン・サイズを大きくしたことによるプリフェッチ効果の好影響を享受することができることがわかった．

謝辞 日頃より有益な御意見を頂くキヤノン(株)情報メディア研究所の諸氏に深く感謝致します．

参考文献

1) 福井 俊之，鈴木 茂夫，中村 秀一，下山 朋彦，数藤 義明，濱口 一正，柴山 茂樹，"光バスクラスタシステムの仕様と基本性能の評価，" 情処研報ARC-119，August 1996.
2) 鈴木 茂夫，福井 俊之，数藤 義明，柴山 茂樹，"光バスクラスタシステム対応の分散タスク/スレッド機構の実現，" 情処研報ARC-119，August 1996.
3) 中村 秀一，下山 朋彦，福井 俊之，濱口 一正，柴山 茂樹，"Weak Consistencyを利用した遅延キャッシュ・コヒーレンシ・プロトコル，" 情処研報 ARC113-20，pp.153-160，August 1995.
4) Shigeki Shibayama, Kazumasa Hamaguchi, Toshiyuki Fukui, Yoshiaki Sudo, Tomohiko Shimoyama and Shuichi Nakamura, "An Optical Bus Computer Cluster with A Deferred Cache Coherence Protocol," In Proceedings of the 1996 International Conference on Pararell and Distributed Systems, pp.175-182, Tokyo, June 1996.
5) Daniel Lenoski, et al. "The DASH Prototype: Implementation and Performance," *In Proceedings of the 19th International Symposium on Computer Architecture*, pp. 92-103, Gold Coast, Australia, May 1992.
6) Chris Holt, et al. "The Effects of Latency, Occupancy, and Bandwidth in Distributed Shared Memory Multiprocessors," *Stanford University-Computer Systems Laboratory, Technical Report*, CSL-TR-95-660, January 1995.
7) Anant Agarwal, et al. "The MIT Alewife Machine: Architecture and Performance," *In Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp. 2-13, Santa Margherita Ligure, Italy, June 1995.
8) Steven Cameron Woo, et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations," *In Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp. 24-36, June 1995.
9) Josep Torrellas, Monica S. Lam, and John L. Hennessy, "Shared Data Placement Optimization to Reduce Multiprocessor Cache Miss Rates," *In Proceedings of the 1990 International Conference on Parallel Processing*, pp. 266-270, August 1990.
10) Michel Dubois, Jonas Skeppstedt, Livio Ricciulli, Krishnan Ramamurthy, and Per Stenström, "The Detection and Elimination of Useless Misses in Multiprocessors," *In Proceedings of the 20th Annual International Symposium on Computer Architecture*, pp. 88-97, May 1993.
11) Czarek Dubnicki, Thomas J. LeBlanc, "Adjustable Block Size Coherent Caches," *In Proceedings of the 19th International Symposium on Computer Architecture*, pp. 170-180, Gold Coast, Australia, May 1992.
12) Leslie Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," IEEE Transactions on Computers, vol. C-28,No.9, pp. 690-691, September 1979.
13) Michel Dubois and Christoph Scheurich, "Synchronization, Coherence, and Event Ordering in Multiprocessors," IEEE Computer, vol. 21 pp. 9-21, February 1988.
14) Sarita V. Adve, Mark D. Hill, "Weak Ordering - A New Definition," In Proceedings of the 17th Annual International Symposium on Computer Archtecture, pp. 2-4, Seattle, Washington, May 1990.
15) Kourosh Gharachhorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy, "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," In Proceedings of the 17th Annual International Symposium on Computer Archtecture, pp. 15-26, Seattle, Washington, May 1990.
16) Pete Keleher, Alan L. Cox, and Willy Zwaenepoel, "Lazy Release Consistency for Software Distributed Shared Memory," In Proceedings of the 19th Annual International Symposium on Computer Archtecture, pp. 13-21, Gold Coast, Australia, May 1992.
17) Leonidsa I. Kontothanassis, Michael L. Scott, and Richardo Bianchini, "Lazy Release Consistency for Hardware-Coherent Multiprocessors," TR547, Competer Science Department, University of Rochester, December 1994.