2005－ARC－164（19）
2005／8／4

Sparse Directory を利用したキャッシュ制御手法の CMP への適用

田 辺 靖 貴† 住 吉 正 人† 天 野 英 晴†

Chip MultiProcessors(CMP) において，バスのボトルネックを避け，メモリアクセスのスループットを確保するためには，クロスバなどのスイッチ網を利用する必要がある．しかし，スイッチ網では，スヌープ方式を用いてキャッシュの一致制御ができなくなるため，ディレクトリ方式を用いられるが，利用メモリ量が増大して、レイテンシが増加する問題点がある．そこで，CC-NUMA 用に提案された Sparse Directory 方式を導入し，限られたメモリ容量でディレクトリ情報を管理し，オンチップにディレクトリを配置することにより，効率的なキャッシュ制御を可能とする方法を提案し、命令レベルシミュレーションにより評価した．結果として，オンチップに実装可能な限られた容量で，一般的なディレクトリベースのキャッシュ制御を行った場合と比較しても遜色ない性能を実現できることを示す．

Using a Sparse Directory as a Cache Coherence Maintenance scheme of CMP

Yasuki Tanabe,† Sumiyosi Masato† and Hideharu Amano †

In order to avoid the bus bottleneck and enhance the throughput, switch connected networks including crossbars have been introduced as a high performance interconnection method for Chip-MultiProcessors(CMP). However, snooping cannot be used in such switch based networks, directory based management that requires a large amount of memory must be used for consistency control of a cache system. Here, the Sparse Directory scheme which was proposed for an efficient directory management method in CC-NUMA is applied to switch based networks in CMP. Through the evaluation with an instruction level simulation, the performance is almost the same as traditional directory management method with much smaller memory requirement for the sparse directory.

1. はじめに

比較的シンプルなプロセッサコアを複数搭載する Chip MultiProcessors(CMP) は，チップの回路規模に応じたスケーラブルな性能向上が得やすく，また開発，検証が容易であるといった特徴を持ち，Power4[1] などのサーバ向けや, MPCore[2], M32R[3] など組込み機器向けなど，広い分野において利用されるようになってきている．

1 チップに搭載されるコア数は，現時点では 2〜4 程度が主流であるため、CMP におけるコア間およびメモリインターフェース等の内部接続網には，チップ内バスが広く利用されている．しかし，集積度の向上にオンチップ上に搭載されるコア数が増加し，コア自体の高性能化により通信量が増大すると，オンボード上同様に，バスのボトルネックが問題となることが予想される。この場合，大容量の転送能力を持つクロスバスイッチが内部の接続網として利用されることが想定される．実際，ヘテロジーニアスなマルチコア LSI では，このようなスイッチを持つチップが登場している．

ホモジーニアスなチップマルチプロセッサでは，オンボード上の SMP(Symmetric MultiProcessor) 同様，プライベートキャッシュを共有メモリに対して持つ構成が一般的となる．しかし，コア間の接続にクロスバスイッチを用いた場合，バスを利用した場合のように snoop を利用したシンプルなキャッシュ制御が行えなくなる．そのため，スイッチ結合されたシステムでは，一般的にはキャッシュの共有情報を一元的に管理する，ディレクトリを利用した方式が利用される．

しかし，ディレクトリを利用したキャッシュ制御では，一般的には共有メモリの全キャッシュライン毎にディレクトリエントリが必要となるため，ディレクトリ管理用の領域にも比較的大きなメモリ領域を必要とする．このため，CC-NUMA などディレクトリで管理するシステムでは、メインメモリ同様 DRAM 上に保持する場合が多い．しかし，CMP において，外部に設置される DRAM へキャッシュ制御のたびにアクセスを行うことは，チップ外部へのトラフィックを増やし，共有メモリアクセスレイテンシの増加の原因となる．

このため，スイッチ結合によりコア間を接続する CMP では，ディレクトリをチップ外に保持するのは理想的ではなく，ディレクトリをチップ上に配置し，高速にアクセス可能とすることが望ましい．しかし，ディレクトリをチップ上で保持するには，限られたメモリ容量のみでキャッシュラインの共有情報を管理する方法を用いなければならない．

Gupta らは，CC-NUMA である DASH に用いることを想定し，限られたサイズのメモリにて，ディレクトリを保持し，これを用いてキャッシュ制御を実現する方法として，Sparse Directory を提案した[4]．この手法は，ディレクトリをキャッシュとして構成し, 限られたメモリのみをディレクトリ管理に利用し，保持しきれなくなったディレクトリのエントリについては，該当のラインを無効化する事により，ディレクトリ管理に必要なメモリ量を劇的

† 慶應義塾大学
Keio University

に削減する方法である．

本稿では、Sparse Directory 法を CC-NUMA ではなく，クロスバスイッチで接続された CMP に適用する手法を提案する．この手法により，チップ上におけるディレクトリの保持が可能となり，高速なディレクトリ情報の参照を可能とすることができる．

2. CMP-Sparse Directroy

2.1 Sparse Directory

Sparse Directory は，CC-NUMA である Starnford DASH アーキテクチャのキャッシュ制御方式の一つとして利用する事を想定し，考案されたディレクトリ情報の管理方式である．

共有データキャッシュの総容量は，総共有メモリのサイズに比較すると僅かであるため，殆どのラインに対応するディレクトリの状態は，カラの状態 である．このため，Weber らは，ディレクトリを保持するメモリ自体をキャッシュのように構成し，ディレクトリ保持に必要なメモリ容量を削減する方法を取った．

ただし，ディレクトリをキャッシュのように保持しているため，新規にエントリを登録する場合に，他のキャッシュラインのディレクトリと衝突が発生する場合がある．通常のキャッシュであれば，書き戻しが必要となるが，ディレクトリ情報も書き戻しを行うと，書き戻されたディレクトリ情報の保持用のメモリが必要となってしまう．そこで，Sparse Directory は，競合したラインのコピーをシステムから無効化した後，新規にエントリを登録する方法を取る．この方法では，Sparse Directory は，ディレクトリ情報の書き戻しを必要としないため，メモリライン全てに対しディレクトリを保持しておく必要がない．これにより，限られた容量のディレクトリ保持用のメモリでキャッシュ制御を実現する．

2.2 CMP への適用

Sparse Directory は，元来それぞれのノードが DRAM で構成されたホームメモリを持つことを想定して提案された手法である．これをクロスバスイッチで接続された CMP に適用するため，図 1 に示すアーキテクチャを提案する．

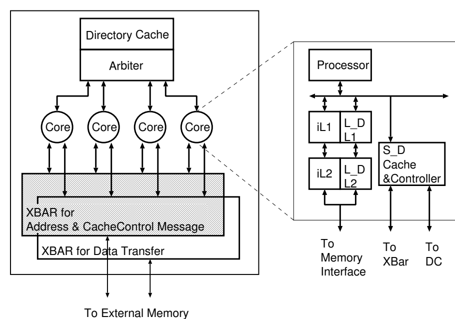

図 1 システム概観

ここで，各コアは，プロセッサコア，命令，ローカルデータのキャッシュ，共有データキャッシュ，共有メモリアクセスおよび共有データキャッシュ制御用のコントローラにより構成される．このコアを複数個が外部メモリインタフェースを介して共有メモリにクロスバスイッチで接続される。ここで、スイッチは、メモリアクセス要求およびキャッシュ制御メッセージの転送用のクロスバ・スイッチと，連続データ転送用のクロスバ・スイッチを別々に持たせることにより、キャッシュ制御とデータ転送のオーバラップを実現する．ここで、外部メモリインタフェースに代ってチップ内に共有 L2 キャッシュを持たせる構成も考えられる．しかし，本稿では、各コアのプライベートキャッシュが十分な大きさであることを前提として、直接外部メモリを接続する構成を取り，今後，L2 キャッシュをチップ内に保有する構成も検討していく事とした．

プロセッサコアには，Out-of-order 実行，2 命令同時発行，分岐予測をサポートした比較的簡単なプロセッサを想定した。プロセッサからは同時に複数の共有メモリアクセス命令が発行されうる．これを，適切に処理するために共有メモリアクセス制御用のコントローラ内にテーブルを設け，依存関係がない場合には並行に処理できるよう管理する事とした．

ディレクトリ情報用のキャッシュ(以下 DC) は，各コアから，arbiter を介して接続され，アクセスされる．DC は，2 way set-associative 構成でディレクトリ情報を保持し，競合が発生した場合は，無効化するラインのエントリを LRU で決定する．また，同一のラインに対する複数のプロセッサからの要求に対し，適切にディレクトリ操作を行うため，各ディレクトリは，エントリ毎に，Transient であるかどうかの状態を保持することとし，Tansient な状態に設定されたラインに対する要求には，NACK を返し再度 DC のアービタへ要求を転送しなおす事とした．

また，外部メモリインタフェースは，8 個までの要求をバッファ可能とし，登録順にメモリアクセスを行う事とした．

2.3 キャッシュ管理プロトコル

CMP では，キャッシュミス時に，レイテンシの大きな外部メモリへアクセスするよりも，オンチップの通信のためレイテンシの少ないキャッシュ間転送を利用し，外部メモリアクセスは極力避けるほうが効率的であると考え，MOESI プロトコルを用いる事とし，可能な限りキャッシュ間転送を行う方式を採用した．

2.4 各要求の処理手順

プロセッサから発行された，共有データへの Read，Write 要求は次のように，処理される．

(1) 共有データキャッシュへアクセスを行い，ヒットした場合は，Read 要求であればプロセッサへデータを返信し，Write 要求でキャッシュの状態が Exclusive であればキャッシュしているラインを更新しアクセスを終了する．

(2) キャッシュミスや，ヒットしたが，Write 要求でキャッシュの状態が Exclusive でない場合には，DC のアービタへ要求を登録し，アクセスの許可を待ち，DC へアクセスを行う．

(3) DC へアクセスし，以下のように処理を行う．

- DC にエントリが登録されていたが，エントリが Tran-

sient な状態に設定されていた場合は，アクセスに対し NACK が返され，再度，DC へアクセスを行う．

- DC にエントリが登録されており，Transient に設定されていなかった場合には，他の要求によりこのエントリの情報が更新されないように該当エントリを Transient に設定する. その後，各コアは，DC に登録されていた共有情報をもとにキャッシュラインの転送要求，無効化要求などを転送する．その完了後，再度 DC のアービタへ要求を転送し，アービトレーションの後，DC 上のエントリの更新を行い Transient 状態を解除する．
- DC にエントリが登録されていない場合は，DC に新規にエントリの登録を行おうとする．この時，既存の DC 上のエントリと競合していない (空いている Way が存在する) 場合は，エントリを登録し，Transient に設定した後，外部メモリへアクセスを行い，キャッシュラインのコピーを取得する．その後，再度 DC のアービタへ要求を転送し，アービトレーション後，DC 上のエントリの更新を行い Transient 状態を解除する．

  この時に，既存の DC 上のエントリと競合した場合は，競合するエントリを Transient 状態に設定した上で，そのラインを共有するコアにラインの無効化要求と，必要であれば書き戻し要求を転送する．その終了を待ち，再度 DC へアクセスを行い，アービトレーションの後，Transient に設定していた DC 上のエントリの無効化を行う．この無効化と同時に，新規にエントリを追加してもよいが，動作の簡略化のため，再度 DC へのアクセスを行い，現在の要求に対応するエントリを，無効化した Way に登録する事とした．エントリを登録した後の動作は，競合が発生しなかった場合と同様である．

3. 評　　価

3.1 評価方法

提案する CMP 用 Sparse Directory の評価を行うため，シミュレータ構築ライブラリ ISIS[5] を用いてシミュレータを構成した．プロセッサコアには，Out-of-order 実行，2 命令同時発行，分岐予測をサポートした SimpleScalar が提供する sim-outorder をマルチプロセッサシステムのシミュレーションに利用できるように改変し，本研究室にて開発した並列計算機シミュレータ構築用ライブラリ ISIS のライブラリの一つとして利用できるようにしたモデル[6] を利用している．キャッシュ、クロスバ、メインメモリについては ISIS のモジュールとして実装した．

シミュレータでは，各処理に必要なレイテンシを以下のように設定した．共有データキャッシュへのアクセスには 2 cycle を，DC へのアクセスはアービタへの要求の登録に 2 cycle，アービトレーション後のディレクトリ情報の参照と更新に 3 cycle を必要とすることにした．また，無効化やキャッシュ間転送の要求等のキャッシュ制御メッセージおよび，メモリインターフェースへの外部メモリアクセス要求の登録には 4 cycle のレイテンシを付加した．

表 1　最短共有メモリアクセスレイテンシ

Access	Latency(cycle)	non-sparse(cycle)
Read L1 Hit	4	4
Read L1 Miss(from Memroy)	67	77
Read L1 Miss(from L1)	24	34
Write L1 Hit(Exclusive)	3	3
Write L1 Hit(Shared)	14	24
Write L1 Miss(Shared)	22	32
Write L1 Miss(Invalid)	65	75

外部メモリアクセスにはさらに DRAM へのアクセスとして，40 cycle と，キャッシュラインの転送時のレイテンシが付加される．キャッシュラインの転送時には，64Bit 幅でラインが送信されることを想定した．例えば 64Byte のキャッシュラインサイズを用いている場合であれば，メモリ-キャッシュ間，および，キャッシュ-キャッシュ間のライン転送時にレイテンシとして 8cycle が付加される．

これをもとに，表 1 に共有メモリアクセスのレイテンシをまとめる．キャッシュミス時のレイテンシとしては，キャッシュ間転送を行う場合のレイテンシと，外部メモリへアクセスする場合のレインテンシを示している．ただし，DC が Transient に設定されている場合や，DC で他のエントリとの競合が発生した場合には，レイテンシがそれに応じて変化するが，ここでは，最短の場合のレイテンシを記載している．また，Write 要求では，プロセッサは要求を管理するテーブルに Write 要求を登録した後は，その完了を待たずに次の処理を継続できる．このため，Write 要求については，プロセッサが Write 要求を発行してから，共有データキャッシュ上でキャッシュしている，もしくはキャッシュしたラインを更新するまでのレイテンシを示している．

また，比較対象のモデルとして，全共有メモリのライン毎にディレクトリを DRAM に保持するモデル (non-sparse) も用意した．このモデルでは，Sparse Directory を用いた場合と比較して，Directory へのアクセスに余分にレイテンシがかかる事を考慮し，DC のアービタへの登録に 12 cycle, DC のアクセスに 3 cycle を設定した．

このレイテンシは，外部 DRAM にアクセスする事を考慮すると低めのレイテンシであるが，今回は，評価結果が，Sparse Directory に偏って良い結果とならないように，このような低めのレイテンシで評価を行なった．

3.2 ディレクトリ用メモリ容量の比較

フルマップ方式で共有するコアを管理する場合，コア数 (P) と，MOESI 状態の管理に 3bit，Transient 状態の管理で 1bit で，計 $P+4$bit 分がキャッシュライン毎に共有情報として必要となる．また，DC がキャッシュ構成であるため，エントリ毎にタグを保持する必要がある．

このため，M を全共有メモリのサイズ (byte)，C をキャッシュサイズ (byte)，L をキャッシュラインサイズ (byte)，E を DC の総エントリ数, W を DC の Way 数とすると，DC の 1 エントリ毎に必要な bit 数 (B) は，以下のように表せる．

$$B = \log_2 \frac{M/L}{E/W} + (P+4)$$

DC のエントリ数として，システムでキャッシュしうる総

ライン数の F 倍を用いる場合，DC に必要なメモリ容量 S(byte) は，

$$\begin{aligned} S &= EB/8 \\ &= FP\frac{C}{L} \cdot (\log_2 \frac{M/L}{FP\frac{C}{L}/W} + (P+4))/8 \\ &= FP\frac{C}{L} \cdot (\log_2 \frac{MW}{FPC} + (P+4))/8 \end{aligned}$$

と表わせる．

この式をもとに，L を 64Byte，M を 128MByte，C を 32KByte, W を 2-way とし，P, F を変化させ，DC に必要なメモリ容量を示したのが，図 2 である．

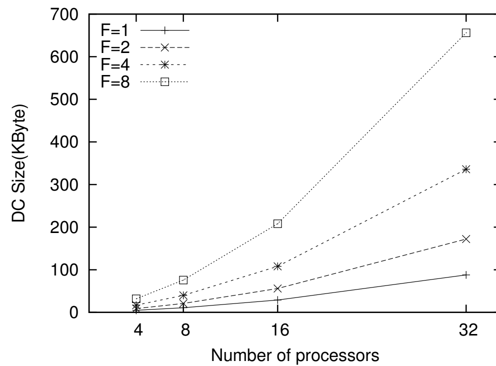

図 2 DC に必要なメモリ容量

これをみると，$F = 4$ で，16PU 構成時に 108KByte 程度と，チップ規模などにもよるが，オンチップ上の SRAM として現実的に実装可能な容量でディレクトリを管理できることがわかる．また，ディレクトリ用のメモリ容量はラインサイズ L をより大きくしたり，Coarse Vector[4] 等を利用する事などによりさらに削減する事も可能である．

また，全メモリライン毎に共有情報を保持する場合と比較すると，Sparse Directory 方式で必要なディレクトリ容量は，32 プロセッサ構成で，F を 8 とした場合でも，1/10 以下の容量となる．

Gupta らの論文にも示されているが，このように Sparse Directory 方式を利用する事によりディレクトリ用のメモリ容量は大幅に削減可能であり，オンチップのメモリにて実装可能なサイズまでに削減できると言える．

3.3 シミュレーションによる評価

次に，今回実装を行ったシミュレータを利用し，シミュレータ上でアプリケーションを実行して評価を行なった．

評価には，Splash-2 ベンチマーク集[7] から，fft, lu, radix を用いた．評価に使用している ISIS を利用したシミュレータは，詳細な評価が可能となっているが，動作速度があまり高速ではないため評価に用いているデータセットのサイズは，やや小さめのサイズではあるが，それぞれ，2^{14}, 128×128, 65536 keys に設定し実行した．

3.3.1 実行時間の比較

Sparse Directory を利用する事により，ディレクトリ用メモリ容量の大幅な削減が可能であり，オンチップにディレクトリを保持し高速なディレクトリの参照が可能となる一方，DC 上でのディレクトリの競合により，キャッシュラインの不要な無効化を招き，性能低下の原因となる恐れがある．そこで，16 コア構成時に，F を 0.5〜8 まで変化させてアプケーションを実行し，Sparse Directory を使用した場合 (w. sparse) と，使用しないで全メモリライン毎にディレクトリを外部 DRAM に保持した場合 (non-sparse) とで実行時間を比較した．その結果を，図 3 および，図 4 に示す．図 3 では，各コアの共有データキャッシュのサイズ C を 32KByte にしているが，図 4 では，キャッシュヒット率があまり高くない場合の挙動を見るために，共有データキャッシュサイズ C を 2KByte に設定している．また，図では，w. sparse での実行時間を，non-sparse の場合での実行時間で正規化している．

図 5 と，図 6 はこの時の，DC を参照した結果の内訳を，ディレクトリ情報がすでに登録済みであったか (Hit), エントリが登録されおらず，空いている Way にエントリを登録したか (Miss), エントリが登録されておらず，そのエントリの登録のために既存のエントリの無効化が必要としたか (Miss w Replace) の 3 つに分けて，その割合を示している．

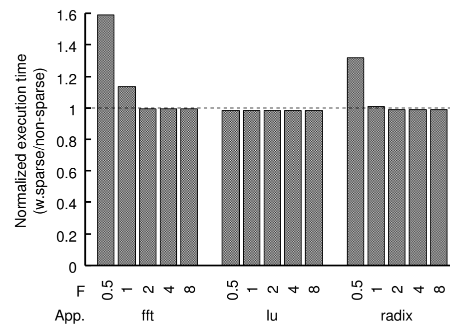

図 3 実行時間の比較 (C = 32Kbyte)

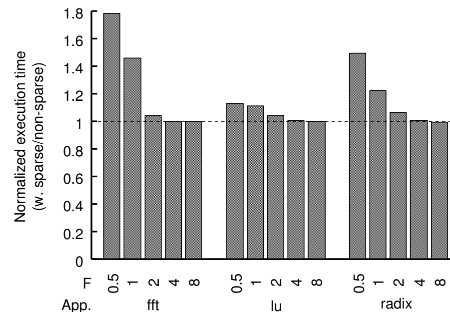

図 4 実行時間の比較 (C = 2Kbyte)

この結果，Gupta らの評価と同じように，どの場合でも，F を 4 に設定することにより，DC での競合が抑えられている．

また実行速度についても，F を 4 とする事により，F を 1 とした場合などと比較し，実行速度が向上しており，non-sparse と比較しても，対等か，僅かに高速となっている．

Sparse Directory を用いた場合に，ディレクトリ情報が

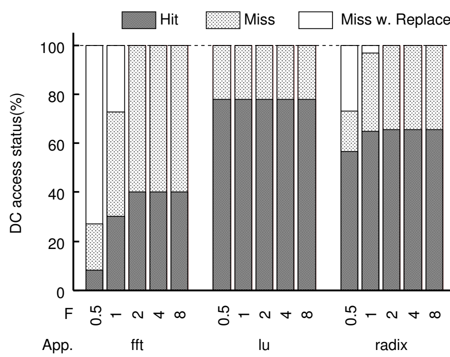

図 5　DC アクセスの内訳 (C = 32Kbyte)

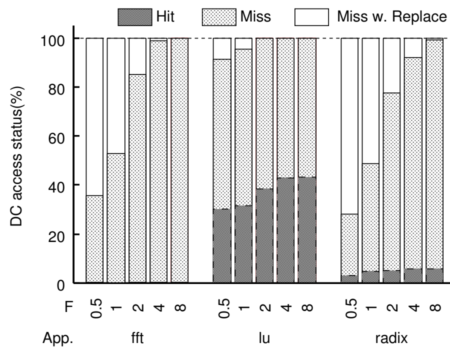

図 6　DC アクセスの内訳 (C = 2Kbyte)

高速にアクセスされるにもかかわらず，共有データキャッシュが 32KByte の場合に，non-sparse と比較してごく僅かな高速化に留まっているのは，non-sparse のディレクトリアクセスのレイテンシを，w.sparse よりも僅かに大きくなるように設定しためである．

キャッシュサイズが 2KByte の場合には，F を 4 に設定しても，実行速度は，non-sparse と変わらない．これは，ほとんどのラインが共有される事なく共有データキャッシュから追い出されてしまうため，図 6 に示されているように，lu を除いて，DC でのヒット率が少なく，ほとんどのメモリアクセスが外部のメモリを参照しており，DC のアクセスレイテンシの長短が実行時間に与える影響が少ないためである．

また，F を，0.5 や 1 とした場合などのように，DC において，Replace が発生する割合がある程度高い場合には，やはり non-sparse と比較して実行時間の低下を招いていることがわかる．キャッシュサイズに 32KByte を用い，DC のサイズを $F = 1$ として設定し，fft を実行した場合では，28%程度の Replace を発生させており，non-sparse と比較して，14%程度の速度低下となっている．

このように，DC のエントリ数を，チップ内で共有されうるキャッシュライン数の 4 倍程度設ければ，non-sparse と比較して，劣らないか，若干上回る性能を示すことがわかった．このときの DC に必要なサイズは，108KByte であるため，チップ規模にもよるがオンチップでの実装が現実的なサイズで，non-sparse と比較して劣らない性能が実現できることがわかる．また，LU などのように，アプリケーションによっては，DC のサイズをそれほど必要としないため，アプリケーションが限定できるならば，さらにディレクトリ容量を削減可能であると考えられる．

3.4　Sparse Directory の適用による実行速度の変化

今回は，non-sparse でのディレクトリへのアクセスレイテンシを，w. sparse と比較して，10cycle 多く設定しているのみであることもあり，F を 4 としても，実行速度の改善は，わずかであった．しかし，オフチップにディレクトリを設置した場合，ディレクトリメモリ I/O の速度や，チップ内で利用する周波数次第で，より多くの cycle がディレクトリアクセスに必要となりうる．

そこで，non-sparse を，ディレクトリアクセスのレイテンシを，w.sparse よりも 10〜50cycle 多い場合について，それぞれ各アプリを実行し，その結果をもとに w.sparse と実行時間を比較した．その結果を，w.sparse の各アプリの実行時間を，レイテンシ毎に non-sparse で正規化して図 7 にて示した．図では，w.sparse の場合を基準とした non-sparse の場合のディレクトリアクセスのレイテンシの差を，x 軸に示している．

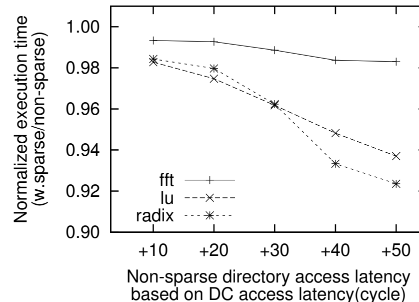

図 7　各ディレクトリアクセスレイテンシの non-sparse との比較

この結果，やはり，外部にディレクトリを置いた場合のレイテンシが大きいほど，sparse directory を適用することがより効果的であることがわかる．ただし，その差は，non-sparse のレイテンシが 10cycle 多い場合で，最大 1.7%程度，50cycle 多い場合でも，最大で，7.6%程度と，やはり僅かな高速化を実現するのみであった．

4. 関 連 研 究

本研究と同じように，ディレクトリ容量を削減し，オンチップにディレクトリを配置することを前提とした関連研究として，DRESAR と，MINDIC が挙げられる．

DRESAR[8] はネットワーク上のスイッチにディレクトリをキャッシュのように構成し，ディレクトリのアクセス時間を短縮している．しかし，DRESAR では，Sparse Directory や本研究と異なり，共有メモリ側にフルマップ方式のディレクトリが必要であり，ディレクトリ管理に必要なメモリ容量は大きい．

本研究室で提案した MINDIC[9] は，Multistage Interconnection Network(MIN) のスイッチ内にディレクトリを配置し，スイッチとともにディレクトリもオンチップに配置する事を考慮している．ただし，MINDIC では，MIN 接

続されたシステムでの利用を前提としており，また Write-through を行う無効化ベースのキャッシュ管理プロトコルに適用できるキャッシュ制御手法である．

また，Duato らは，ディレクトリを階層的に管理する方法で，ディレクトリ容量を削減する方法を提案した[10]．具体的には，1 層目に小容量のメモリを用い，最近アクセスされたキャッシュラインのディレクトリを Full-map 方式で保持し，2 層目で，ディレクトリを圧縮して保持することにより，大規模な CC-NUMA マルチプロセッサシステムにおいて，性能を低下させることなくディレクトリに必要なメモリ量を劇的に削減できること示している．この手法は，特に大規模なシステムにおいて効果的であるが，今回検討を行った CMP に Sparse Directory を適用する場合においても，構成が複雑にはなるが，オンチップに保持すべきディレクトリのエントリ数がかなり多くなる場合には，この手法によりさらにディレクトリ容量を削減することが可能となると考えられる．

5. まとめ

内部接続にクロスバなどのスイッチ結合を用いる CMP において，ディレクトリ方式による効率的なキャッシュ制御を実現する方式として，Sparse Directory を CMP に用いることを想定し評価を行なった．

Sparse Directory を CMP に適用することにより，各プロセッサが 32 KByte 共有データキャッシュを持ち，キャッシュラインサイズが 64byte の 16PU プロセッサ構成のシステム構成において，システムがキャッシュしうるライン数の 4 倍のエントリ数のディレクトリ情報を保持する事とした場合，ディレクトリ保持のために必要なメモリ容量は，108KByte と，オフチップに全てのキャッシュラインのディレクトリ情報を保持するシステムと比較して，必要とするディレクトリ用メモリ容量は大幅に少なくて済む事を示した．

また，システムがキャッシュしうるライン数の 0.5, 1, 2, 4，8 倍のエントリ数のディレクトリ情報を，キャッシュとして保持する場合についてそれぞれ評価を行い，どの場合でも，4 倍程度のディレクトリ情報をキャッシュすれば，オフチップに全てのキャッシュラインのディレクトリ情報を保持するシステムと比較しても性能が劣らないことがわかった．

この場合のディレクトリ容量は，108 KByte であるため，オンチップの SRAM にて実装し高速にアクセスできるとして，オフチップに全てのキャッシュラインのディレクトリ情報を保持するシステムと比較して評価を行なった．

その結果，sparse directory を適用し，オンチップにディレクトリを構成することにより，全共有メモリラインのディレクトリ情報をオフチップに保持するシステムと比較し，実行時間は，オフチップのディレクトリアクセスのレイテンシに応じ，1.7%〜7.6%程度の高速化を実現する事を示した．

このように，sparse directory を CMP に適用することにより，劇的な高速化を得ることはできなかったが，オンチップにディレクトリを配置することはディレクトリアクセス用の I/O を設ける必要がなくなる事や，オフチップにディレクトリ保持用に大きなメモリを設ける必要が無くなるといった他のメリットもある．

しかし，オンチップに実装する場合は，今回見積ったメモリ容量以外にも，アービタや制御ロジック等の回路によりハードウェア量が増大するというデメリットもあるため，これを今後評価していく予定である．

また，今回は評価用アプリケーションには,Splash-2 を用いているが，組み込み機器向けの CPU として，CMP が利用される事が多くなってきている現状を踏まえると，組み込み機器で利用されるアプリケーション実行時の実効性能の評価も必要であるだろう．これについても，今後，検討，評価を行っていく予定である．

さらに，今回は，共有 L2 データキャッシュを利用しない構成で検討を行ったが，これを利用する構成における性能評価も行う必要がある．

参 考 文 献

1) Keith Diefendorff. Power4 focuses on memory bandwidth. *Microprocessor Report*, Vol. 13, No. 3, 6 October 1999.
2) D Cormie. The arm11 microarchitecture. In *ARM Ltd. White Paper*, 2002.
3) S. Kaneko, K. Sawai, ··· , O. Tomisawa and T Shimizu. A 600mhz single-chip multiprocessor with 4.8 gb/s internal shared pipelined bus and 512kb internal memory. In *IEEE ISSCC Dig. Tech. Papers*, pp. 254–25, 2003.
4) Anoop Gupta, Wolf-Dietrich Weber, and Todd Mowry. Reducing memory and traffic requirements for scalable directory-based cache coherence schemes. In *1990 International Conference on Parallel Processing*, Vol. I, pp. 312–321, St. Charles, Ill., 1990.
5) 若林正樹, 天野英晴. 並列計算機シミュレータの構築支援環境. 電子情報通信学会論文誌, Vol. J84, No. 3, pp. 247–256, 2001.
6) 薬袋 俊也, 塙 敏博, 田辺靖貴, 天野英晴. ISIS-SimpleScalar の実装. 情報処理学会研究報告 2004-ARC-160, pp. 29–34, Dec 2004.
7) S.C.Woo, M.Ohara, E.Torrie, J.P.Singh, and A.Gupta. The splash-2 programs: Characterization and methodological considerations. *Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 24–36, 1995.
8) R. Iyer, L. N. Bhuyan, and A. Nanda. Using switch directories to speed up cache-to-cache transfers in CC-NUMA multiprocessors. In *Proc. of the 14th Int'l Parallel and Distributed Processing Symposium (IPDPS'00)*, pp. 721–728, May 2000.
9) Masato Sumiyoshi, Takashi Midorikawa, and Hideharu Amano. Design and implementation of switching fabrics for multistage interconnection network with directory cache. In *COOL Chips VII*, p. 75, April 2004.
10) Manuel E. Acacio, Jose Gonzalez, Jose M. Garcia, and Jose Duato. A two-level directory architecture for highly scalable cc-numa multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 16, No. 1, pp. 67–79, 2005 January.