大規模分散共有メモリシステムの
一貫性制御に関する研究

提出先　　大阪大学大学院情報科学研究科

提出年月　2022年1月

細見　岳生

研究業績

1. 学術論文誌発表論文

(1) 細見 岳生, 安戸 僚汰, 鯉渕 道紘, 下條 真司, “ 二重同型 Hypercube ネットワーク, ” 情報処理学会論文誌コンピューティングシステム, vol. 14, no. 3, pp. 1–13, 2021.

(2) S. Sombatsiri, S. Shibata, Y. Kobayashi, H. Inoue, T. Takenaka, T. Hosomi, J. Yu, Y. Takeuchi, “ Parallelism-flexible Convolution Core for Sparse Convolutional Neural Networks on FPGA, ” IPSJ Transactions on System LSI Design Methodology, vol. 12, pp. 22–37, 2019.

(3) 細見 岳生, 加納 健, 中村 真章, 広瀬 哲也, 中田 登志之, “ 並列計算機 Cenju-4 の分散共有メモリ機構,” 情報処理学会論文誌, vol. 41, no. 5, pp. 1400–1409, 2000.

(4) 加納健 , 中村 真章 , 広瀬 哲也, 細見 岳生 , 中田 登志之, “ 並列計算機 Cenju-4 のユーザレベルネットワークインタフェース,” 情報処理学会論文誌, vol. 41, no. 5, pp. 1379–1389, 2000.

(5) 広瀬 哲也, 細見 岳生, 丸山 勉, 加納健, “ 並列コンピュータ Cenju-3 のプロセッサ間通信方式とその評価,” 情報処理学会論文誌, vol. 37, no. 7, pp. 1378–1387, 1996.

(6) 細見 岳生, 森 眞一郎, 中島 浩, 富田 眞治, “ ディレクトリ型キャッシュコヒーレンスプロトコルの性能評価,” 情報処理学会論文誌, vol. 37, no. 2, pp. 1265–1275, 1996.

2. 国際会議発表論文（査読付）

(1) T. Otoshi, S. Arakawa, M. Murata, K. Wang, T. Hosomi and T. Kanoh, “ Flexible Updating of Attractors in Virtual Network Topology Control with Bayesian Attractor Model,” IEEE International Conference on Communi-

cations, pp. 1–6, 2021.

(2) T. Otoshi, S. Arakawa, M. Murata, K. Wang, T. Hosomi, T. Kanoh, “ Acquiring New Categories by Self Data Gathering with Bayesian Attractor Model, ” IEEE International Conference on Human-Machine Systems, pp. 1–4, 2020.

(3) T. Hosomi, R. Yasudo, M. Koibuchi, S. Shimojo, “ Dual-Plane Isomorphic Hypercube Network, ” International Conference on High Performance Computing in Asia-Pacific Region, pp. 73–80, 2020.

(4) S. Sombatsiri, S. Shibata, Y. Kobayashi, H. Inoue, T. Takenaka, T. Hosomi, “ Parallelism-flexible Convolution Core for Sparse Convolutional Neural Networks, ” Workshop on Synthesis And System Integration of Mixed Information Technologies, pp. 188–193, 2018.

(5) Y. Watanabe, Y. Kobayashi, T. Takenaka, T. Hosomi and Y. Nakamura, “ Accelerating NFV application using CPU-FPGA tightly coupled architecture,” International Conference on Field Programmable Technology, pp. 136–143, 2017.

(6) Y. Kobayashi, T. Takenaka, T. Hosomi, Y. Nakamura, “ Accelerating an IoT Application by using FPGA tightly coupled with CPU,” Annual Design Automation Conference, 2016.

(7) T. Takenaka, H. Inoue, T. Hosomi, Y. Nakamura, “ FPGA-accelerated complex event processing,” Symposium on VLSI Circuits, pp. C126–C127, 2015.

(8) E. Kobayashi, S. Shibata, N. Suzuki, A. Shibayama, T. Hosomi, “ FPGA Oriented Intra Angular Prediction Image Generation Hardware for HEVC Video Coding,” Workshop on Synthesis And System Integration of Mixed Information Technologies, pp. 391–396, 2015.

(9) S. Shibata, E. Kobayashi, N. Suzuki, A. Shibayama, T. Hosomi, “ DMATP: A Design Method and Architecture of TU Parallel Processing for 4K HEVC Hardware Encoder,” Workshop on Synthesis And System Integration of Mixed Information Technologies, pp. 403–408, 2015.

(10) T. Miyamoto, K. Ishizaka, T. Hosomi, “ A Dynamic Offload Scheduler for Spatial Multitasking on Intel Xeon Phi Coprocessor,” Workshop on Synthesis And System Integration of Mixed Information Technologies, pp. 261–266, 2013.

(11) E. Korpeoglu, C. Sahin, D. Agrawal, A. El Abbadi, T. Hosomi and Y.

Seo, “ Dragonfly: cloud assisted peer-to-peer architecture for multipoint media streaming applications, ” IEEE International Conference on Cloud Computing, pp. 269–276, 2013.

(12) K. Ishizaka, T. Miyamoto, S. Akimoto, A. Iketani, T. Hosomi and J. Sakai, “ Power Efficient Realtime Super Resolution by Virtual Pipeline Technique on a Server with Manycore Coprocessors,” IEEE Symposium on Low-Power and High-Speed Chips, pp. 1–3, 2013.

(13) K. Kusano, M. Sato, T. Hosomi and Y. Seo, “ The Omni OpenMP Compiler on the Distributed Shared Memory of Cenju-4,” International Workshop on OpenMP Applications and Tools, pp. 20–30, 2001.

(14) T. Hosomi, Y. Kanoh, M. Nakamura and T. Hirose, “ A DSM architecture for a parallel computer Cenju-4, High-Performance Computer Architecture,” International Symposium on High-Performance Computer Architecture, pp. 287–298, 2000.

(15) Y. Kanoh, M. Nakamura, T. Hirose, T. Hosomi, H. Takayama and T. Nakata, “ Message Passing Communication in a parallel computer Cenju-4,” International Symposium on High Performance Computing, pp. 26-28, 1999.

(16) Y. Kanoh, T. Hirose, M. Nakamura, T. Hosomi, K. Tatsukawa, H. Araki, T. Sugawara and T. Nakata,“ Architecture of a parallel computer Cenju-4,” International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, pp. 105–113, 1998.

内容梗概

高性能な計算機システムの構築において，キャッシュを持ったプロセッサとメモリからなるノードをネットワークで接続した大規模分散共有メモリシステムが注目を集めている．大規模分散共有メモリシステムにおいては，ディレクトリ方式の一貫性制御が用いられる．ディレクトリ方式の一貫性制御の特徴は，メモリにおいてどのノードがデータのコピーを保持しているかを管理するディレクトリをメモリに保持することにある．キャッシュミスが発生すると当該データをメモリに保持しているノードに要求を送り，その要求を受けたノードのメモリはディレクトリを基にそのデータのコピーをキャッシュに保持するノードに要求を転送する処理を行う．

大規模分散共有メモリシステムの性能を左右するメモリアクセスレイテンシは次の三点に影響を受ける．一つ目は，あるノードのプロセッサが行った書込みを他のプロセッサに反映するプロトコルとして，Invalidate 型や Update 型等どのようなプロトコルを採用するかである．二つ目は，書込み時にはデータを共有しているノード全てに対して要求を伝える必要があるため，その場合の一貫性制御方式である．三つ目は，ノード間の要求やその応答をネットワークを介して送信する必要があるため，ネットワークのレイテンシである．本論文では，一つ目の課題に対して性能評価を行い，どのプロトコルが適切かを示す．また二つ目と三つ目の課題に対してそれらを改善する手法を提案する．また，このディレクトリ方式の一貫性制御の実現においては，安定動作のためにメモリアクセスが有限時間内に完了することを保証することも重要である．本論文では，この課題を実現する手法も提案する．

2 章では，ディレクトリ方式でのプロトコルの性能評価とその解析を行った．ディレクトリ方式の一貫性制御は，従来評価されてきたスヌープ方式とシステム構成もプロトコルも異なるため，ディレクトリ方式の特性を踏まえて最適なプロトコルを選択することが重要となる．本章では，Invalidate 型，Update 型，Competitive 型の代表的な 3 種のプロトコルの性能評価を行った．その結果，スヌープ方式での評価結果と異なり，Competitive 型がどのワークロードでも良好な性能を発揮することが明らかとなった．また，ディレクトリ方式特有の振舞いである，キャッシュからの読出しあるいは書込み要求を受けたメモ

リが他のノードに転送する要求の割合や数に強い影響を受けて，プロトコル間の性能差が発生していることを確認した．

3 章では，共有しているノード数の増加によるメモリアクセスレイテンシの悪化を抑えるディレクトリおよび一貫性制御手法を提案した．ディレクトリはデータのコピーを保持しているノードを管理する．このディレクトリには，システムを構成するノード数よりも少ないビット数で構成して，システムコストを抑える方式が広く用いられている．しかし，不正確な管理となるため，一貫性制御の上では共有しているノード数が実際よりも大幅に増加する問題がある．また，共有しているノードへの要求の送信とそれらのノードからの応答の受信を，ノード数に比例せず一定の時間で実現する一貫制御が不可欠である．本章では，ビットパターン方式のディレクトリを提案する．このビットパターン方式は，共有しているノード数が実際よりも大幅に増加する問題に対して，特に大規模システムの一部のノードを利用してアプリケーションを動作させるケースで，保持している可能性のあるノード数の増加を抑えることができることを目的とする．また，提案するマルチキャスト・ギャザー機構は，このビットパターン方式やその他の Coarse Vector 等のディレクトリをマルチキャストの宛先指定や，ギャザー時の待合せパターンに利用可能で，様々なネットワークのトポロジで実現可能な方式である．提案したビットパターン方式とマルチキャスト・ギャザー方式を実装した並列計算機 Cenju-4 においてメモリアクセスレイテンシの評価を行い，128 ノードのシステムにおいても書込みのアクセスレイテンシを一定に抑えることができることを確認した．

4 章では，近年多数のシステムで採用されている，二つのネットワーク・プレーンを持つシステムを対象に，二重同型ネットワークを提案した．この二重同型ネットワークは，各プレーンをグラフ同型のネットワークで接続することを特徴とする．Hypercube および Foleded-Hypercube を題材にして二重同型ネットワークの構成法を示し，またそれらの性能評価を実施した．その結果，適切な同型ネットワークの選択により，同一のネットワークを二重化した場合と比較して．ネットワークコストを増加させることなくレイテンシおよびスループットを改善できることを確認した．

最後に，5 章では，メモリアクセスが有限時間内に完了することの保障について議論を行った．ディレクトリ方式においては，一貫性制御はノード間で複数回メッセージをやり取りすることによって実現される．この一貫性制御によってデッドロックが発生するため，それを防止することが求められる．また，この一貫制御においては否定応答によって要求を再試行する仕組みが広く採用されており，他の要求に割り込まれて無限に要求の再試行を繰り返し，スタベイションを引き起こす．本章では，メッセージをメモリに退避することで資源の依存関係をなくし，単一チャネルからなるネットワークにも適用可能なデッドロック防止方式を提案する．また，否定応答による一貫制御の再試行をなくし，メモリが一貫性要求を受け付けた順で処理すること，そのために要求をメモリに一時退避す

る機構を持つことを特徴とする，スタベイション防止方式を提案する．これらの機構を並列計算機 Cenju-4 に実装し，アプリケーションを実行した評価で，メモリアクセスがスタベイションやデッドロックによってタイムアウトを起こすことなく，安定して動作することを確認した．

目次

第 1 章

序論

1.1 研究の背景

計算機システムを高性能化する取り組みとして，複数のプロセッサをネットワークで接続したマルチプロセッサシステムの研究が行われている．このマルチプロセッサシステムにおいては，複数のプロセッサを用いてネットワークを介して通信を行いながら処理が実行される．そのため，ネットワークを介した通信を高性能化することが重要な課題となっている．

このマルチプロセッサシステムにおけるプロセッサ間通信方式には，プログラム中にプロセッサ間の通信を明示するメッセージパッシング方式と，複数のプロセッサでメモリを共有しメモリへの読出しや書込みを通して通信する共有メモリ方式の二つが存在する．後者の共有メモリ方式は，並列プログラミング・モデルとして単純であることから注目され，近年においても複数のプロセッサを接続した小規模なシステムから中規模なシステムで用いられている [1–3].

この共有メモリシステムでは，各プロセッサがアクセスしたデータの一貫性を維持するためのキャッシュ一貫性制御が必要となる [4]．この一貫性制御によって，プロセッサが読出しアクセスを行った場合にシステム内の最新のデータへのアクセスを可能とする．また，書込みアクセスを行った場合には，更新前のデータへのアクセスが行われなくなるように制御を行う．これらの制御はプロセッサを跨る通信によって実現され，メモリアクセスレイテンシとして処理性能に影響を与える．読出しおよび書込みに伴うメモリアクセスレイテンシを小さく抑えることがシステムの高性能化において重要となる．

初期には，共有メモリシステムを接続するネットワークにはバスが用いられ (図 1.1)，そのバスにプロセッサとメモリを接続する構成を取っていた．このようなシステムにおけるキャッシュ一貫性制御には，プロセッサからバスにメモリアクセス要求を流し，その要求を他のプロセッサやメモリがスヌープして応答を返す，スヌープ方式が用いられてき

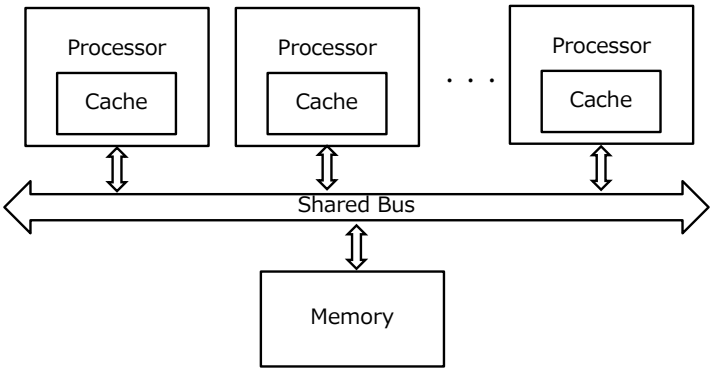


図 1.1　バス接続の共有メモリシステム

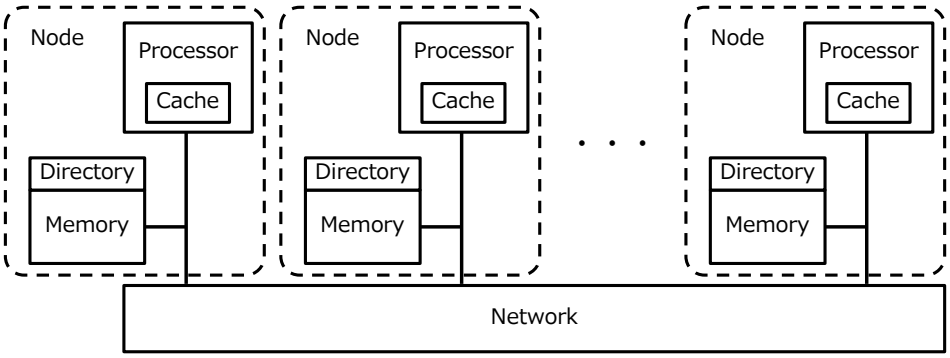


図 1.2　分散共有メモリシステム

た [5]．しかし，システムの大規模化において，プロセッサを接続するネットワークとしてバスが性能ボトルネックとなるようになり，Torus，Hypercube，Fat-tree 等の 1 対 1 接続のリンクとスイッチで構成される性能をスケールさせやすいネットワークが用いられるようになった．そのため，キャッシュ一貫性制御においても，バスを前提としたスヌープ方式に替わるものとして，様々なネットワークに適用可能なディレクトリ方式が提案されてきた [6]．

ディレクトリ方式は，キャッシュを持ったプロセッサとメモリからなるノードがネットワークで接続される分散共有メモリシステム (図 1.2) を対象とするキャッシュ一貫性制御方式である．データに加えてデータの状態管理のためのディレクトリ情報をメモリに，保持することを特徴とする．このディレクトリ情報は，キャッシュ一貫性制御の単位であるキャッシュラインサイズ毎に，メモリのデータがどのプロセッサのキャッシュに保持されているかを管理するものとなる．プロセッサはメモリアクセス要求を，ネットワークを介して対象とするデータを保持するメモリに対して送信する．要求を受けたメモリは，ディ

レクトリ情報を参照して，必要に応じてコピーを保持する全てのプロセッサに対してその要求を転送する処理を行い，またディレクトリ情報を更新する．転送された要求を受けたプロセッサは，要求に応じてキャッシュの状態を更新し応答を返す．このように，スヌープ方式がバスへのブロードキャストをベースとするプロトコルであるのに対して，ネットワークを介した複数の一対一通信で構築するプロトコルとなる．

1.2 研究の動機

大規模分散共有メモリシステムを対象とした，ディレクトリ方式の一貫性制御は，バスベースの共有メモリシステムを対象としたスヌープ方式とは以下の点が異なる [37].

- 一貫性制御プロトコルは，要求を一回バスにブロードキャストすることで実現されるスヌープ方式と異なり，ノード間で複数回メッセージをやり取りすることによって実現される．
- どのノードのキャッシュがデータのコピーを保持ししているかをディレクトリ情報としてメモリで管理する．
- バスではないトポロジのネットワークを用いる．

大規模分散共有メモリシステムの性能を左右するメモリアクセスレイテンシは次の三点に影響を受ける．一つ目は，あるノードのプロセッサが行った書込みを他のプロセッサに反映するプロトコルとして，Invalidate 型や Update 型等どのようなプロトコルを採用するかである．二つ目は，書込み時にはデータを共有しているノード全てに対して要求を伝える必要があるため，その場合の一貫性制御方式である．三つ目は，ノード間の要求やその応答をネットワークを介して送信する必要があるため，ネットワークのレイテンシである．また，上記の三つの課題に加えて，このディレクトリ方式の一貫性制御の実現においては安定動作のためにメモリアクセスが有限時間内に完了することを保証することも重要である．以降それぞれの研究課題について述べる．

一つ目は，メモリアクセスレイテンシの削減を目的とした，ディレクトリ方式でのプロトコルの性能特性の解析である．ディレクトリ方式の一貫性制御は，スヌープ方式とシステム構成もプロトコルも異なるため，ディレクトリ方式の特性を踏まえたプロトコルの選択が重要となる．スヌープ方式においてはプロトコルによるメモリアクセスレイテンシの違いが評価されてきたが [13]，ディレクトリ方式においては評価がされておらず，各種プロトコルがどのような性能特性を示すかを明らかにすることが不可欠である．

二つ目は，メモリアクセスレイテンシの削減を目的とした，共有しているノード数の増加によるメモリアクセスレイテンシの悪化を抑えるディレクトリおよび一貫性制御手法である．ディレクトリ方式においては，スヌープ方式と異なり，データの状態管理のための

ディレクトリ情報を保持する必要が有る．このディレクトリ情報は，キャッシュ一貫性制御の単位であるキャッシュラインサイズ毎にメモリにおいて保持され，当該データのコピーをどのプロセッサのキャッシュが保持しているかを管理する [37]．保持しているノードを正確に管理しようとするとプロセッサ数と同じビット数が必要となる．ノード数を N とするとノード当たり N に比例したサイズのディレクトリが必要となるため，ディレクトリ保持コストはシステムとして $O(N^2)$ となる．従来，この保持コストを抑えるために少ないビット数で管理する Coarse Vector 方式 [29] や階層ビットマップ方式 [31] 等が提案されている．しかし，これらの方式では，実際にデータを保持しているノードに対して，非常に多数のノードを保持している可能性のあるノードとして一貫性制御の対象とし [31]，ネットワークの負荷や一貫性制御を受けるノードの負荷が増大し，メモリアクセスレイテンシを悪化させる．また，書込みアクセス時には，これらの保持している可能性のあるノードに対して書込み要求を送信しまたそれらのノードからの応答を受信する必要があるため，一つ一つメッセージを送信また受信すると保持している可能性のあるノード数に比例した時間を要する [22]．そのため，保持している可能性のあるノードの増加を抑えるディレクトリ情報の保持方式と，またノード数が増加してもメモリアクセスレイテンシの悪化を防ぐ一貫性制御手法が重要となる．

三つ目は，メモリアクセスレイテンシの削減を目的とした，ネットワークの低レイテンシ化と高スループット化である．ディレクトリ方式においては，一貫性制御はノード間で複数回メッセージをやり取りすることによって実現される．それらを積算した遅延がメモリアクセスレイテンシとなるため，メモリアクセスレイテンシに占めるネットワーク通信遅延の割合が高く [37]，ネットワークの低レイテンシ化が重要となる．また，ネットワークの高スループット化によって，ネットワークに対して負荷がかかった状態でもレイテンシを抑える必要がある．

四つ目は，安定動作を達成するための一貫性制御においてメモリアクセスが有限時間内に完了することの保証である [36, 37]．一回のブロードキャストで実現されるバスベースの共有メモリシステムを対象としたスヌープ方式と異なり，ディレクトリ方式の一貫性制御は複数のノード間でメッセージをやり取りすることによって実現される．一連のメッセージのやり取りにおいて，要求を受けたノードは別のノードに要求あるいは応答を出力する．ネットワークの混雑により出力先のバッファがフルで要求や応答を出力できない状態になると，そのノードは受けた要求の処理を完了できず次の要求を受け付けられない状態となる．このようなメッセージを受ける資源とメッセージの出力先の資源の依存関係によってデッドロックが発生する．この一貫制御によるデッドロックは，ノード間のメッセージ配信をデッドロックなしで行うネットワークのみでは防止できない．そのため，一貫性制御としてデッドロックの発生を防止する機構が求められる．また，デッドロック防止に加えて，スタベイションの防止により一貫性制御の完了を保証することが重要とな

る．他の一貫性制御との衝突により特定の一貫性制御が再試行を無限に繰り返すことがないようにする必要がある．

1.3 論文構成

本論文では，前節で述べた，高性能なディレクトリ方式の一貫性制御実現におけるメモリアクセスレイテンシ削減を目的とした三つの課題に対応して，2 章でディレクトリ方式でのプロトコルの性能特性の解析を行い，3 章で共有しているノード数の増加によるメモリアクセスレイテンシの悪化を抑えるディレクトリおよび一貫性制御手法と，4 章でネットワークの低レイテンシ化と高スループット化を提案する．また，前節で述べた四つ目の課題である，安定動作するディレクトリ方式の一貫性制御を実現するために，5 章でメモリアクセスが有限時間内に完了することの保証手法を提案する．以下各章の概要を述べる．

2 章では，ディレクトリ方式でのプロトコルの性能特性の解析を実施する．分散共有メモリプロトコルは，一貫性制御の手順を定めたものであり，プロセッサがキャッシュに対して読出しアクセスを行った時，また書込みアクセスを行った時の手順を規程するものである．評価は，書込み時に他のプロセッサのキャッシュを無効化する Invalidate 型，他のプロセッサのキャッシュにも更新を反映する Update 型，またその中間的な動作をする Competitive 型 [12] の代表的な 3 種のプロトコルを対象とする．従来の研究では，スヌープ方式の共有メモリシステムを対象とした評価でこれらのプロトコルによる性能差がないとされていた [13]．ディレクトリ方式を対象としたこれらプロトコルの評価は存在せず，スヌープ方式とディレクトリ方式でのキャッシュ一貫性制御の振舞いの違いが，どのような影響を与えるかが明らかでなかった．そこで，ディレクトリ方式における 3 種のプロトコルの性能差を明らかとすることを目的として評価を行う．評価を通して，Competitive 型が有効に機能することを確認する．またそれが，ディレクトリ方式の特徴である，要求を受けたメモリがキャッシュにコピーを保持するノードに対してその要求を転送する処理に起因することを確認する．

3 章では，共有しているノード数の増加によるメモリアクセスレイテンシの悪化を抑えるディレクトリ情報の構成法および一貫性制御手法を提案する．一つ目の提案は，ビットパターン方式のディレクトリである．ビットパターン方式は，特に多数のノードからなる大規模システムの一部のノードを利用してアプリケーションを動作させるケースで，保持している可能性のあるノード数の増加を抑えることを目的に設計した．そのために，ノードを階層的に分割して管理し各階層による分割サイズが異なることを特徴とする．従来提案されてきたディレクトリ情報の実現方式 [29, 31] との比較評価を行い，大規模システムの全てのノードを利用するケースでは Coarse Vector 方式よりやや多くのノードを一貫性

制御の対象とする問題があるものの，想定した一部のノードを利用してアプリケーションを動作させるケースでは大幅に少ないノードで済むことを確認する．二つ目の提案は，書込み要求のマルチキャストとその応答の集約を，様々なディレクトリ，様々なネットワークトポロジで実装可能な，汎用的なマルチキャスト・ギャザー機能である．従来もマルチキャストやギャザーを用いて書込み要求の送信と応答の受信を効率化する手法は提案されていた [31]．しかし，そのマルチキャスト方式はネットワーク構造に合わせたマルチキャスト宛先指定によって実現され，その宛先指定に合わせたディレクトリ情報を採用する必要があった．そのため，提案するビットパターン方式や Coarse Vector [29] 方式等を採用することができなかった．また従来のギャザー方式はマルチキャスト時に経由するスイッチを逆に辿って集約する方式で，適用できるトポロジに制約があった．本章で提案する手法は，これらの制約をなくした汎用的なマルチキャスト・ギャザー方式である．マルチキャスト方式は，ネットワークの各スイッチがディレクトリ方式から宛先を算出してマルチキャストを行うことを特徴とする．これにより，ネットワーク構造に制約を受けることなくディレクトリ情報の構成法を選択することができる．また提案するギャザー方式は，送出先ノードがマルチキャストの宛先情報からネットワーク内での待ち合わせパターンを算出し，そのパターンに応じて各スイッチがメッセージを集約することを特徴とする．この特徴により，様々なネットワークトポロジに適用可能となる．提案するビットパターン方式のディレクトリとマルチキャスト・ギャザー機構を実装した並列計算機 Cenju-4 においてメモリアクセスレイテンシの評価を行い，128 ノードのシステムにおいても書込みのアクセスレイテンシを一定に抑えることができることを確認する．

4 章では，一貫性制御の遅延に大きな影響を及ぼすネットワークの高性能化方式の提案を行う．近年多数のシステムで採用されている，複数のネットワーク・プレーンを持つ Multi-Plane 方式 [68] を対象に，二重同型ネットワークを提案する．この二重同型ネットワークは，各プレーンをスイッチ間の接続が異なるグラフ同型のネットワークで接続することを特徴とする．Hypercube および Folded-Hypercube を題材にして二重同型ネットワークの構成法を示す．またそれらの性能評価を実施して，適切な同型ネットワークの選択により，ネットワークの経済的コストを増加させることなく遅延およびスループットを改善できることを示す．

5 章では，メモリアクセスが有限時間内に完了することを保証するために，メッセージをメモリに退避することによってデッドロックとスタベイションの両方を防止する方式を提案する．メモリアクセスが有限時間内に完了することを保証するためには，一貫性制御のプロトコルによってデッドロックが発生しないことに加えて，ある要求に起因する一貫性制御が無限に繰り返されるスタベイションが発生しないことの二点を満たす必要が有る．従来提案されていた方式は，デッドロックの発生を防ぐためにネットワークが複数のチャネルを持つことを要求し，ハードウェアコストが増大する問題があった．また，デッ

ドロックの発生を防ぐためにスタベイションを引き起こす可能性を持つ手法が選択されていた [36, 37]．スタベイションの発生を防ぐプロトコルも提案されていたが [32]，他のプロトコルと比べてメモリアクセスレイテンシが大幅に悪化する問題があった．提案方式は，従来方式のこれらの問題を解決し，単一のチャネルしか持たないネットワークにも適用可能なデッドロック防止方式であり，メモリアクセスレイテンシを悪化させることなくスタベイションを防止する方式である．提案するデッドロック防止方式は，メッセージをメモリに退避することで資源の依存関係をなくすことを特徴とする．提案するスタベイション防止方式は，否定応答による一貫制御の再試行をなくし，メモリが一貫性要求を受け付けた順で処理すること，そのために要求をメモリに一時退避する機構を持つことを特徴とする．提案する機構を実装した並列計算機 Cenju-4 においてアプリケーションを動作させ，メモリアクセスがスタベイションやデッドロックによってタイムアウトを起こすことなく，安定して動作することを確認する．また，本機構がメモリアクセスレイテンシに与える影響が小さいことを確認する．

6 章では，分散共有メモリシステムの関連研究についてまとめる．

最後に，7 章では，本論文の成果を要約し，最近の分散共有メモリシステムやキャッシュ一貫性制御の動向，またそれらの動向に対応した今後の研究課題について述べ，本論文のまとめとする．

第 2 章

分散共有メモリプロトコルの性能評価

2.1 はじめに

本章では，高性能かつ安定動作するディレクトリ方式の一貫性制御の実現における四つの課題のうち，一つ目のメモリアクセスレイテンシの削減を目的としたディレクトリ方式でのプロトコルの性能特性の解析を行う．

キャッシュ一貫性制御のプロトコルは Invalidate 型と Update 型に大別される [5]．この両プロトコルの違いは，共有しているデータに対してプロセッサが書込みを行った時の処理にある．Invalidate 型は，プロセッサ書込み時に，他プロセッサのキャッシュコピーを無効化することで一貫性を維持する．一方 Update 型は，プロセッサの書き込んだデータを他プロセッサのキャッシュコピーにも反映することで一貫性を維持する．

従来のバス接続の共有メモリシステムでは，スヌープ方式のキャッシュ一貫性制御プロトコルが用いられていた．そのプロトコルは Invalidate 型が主に採用され Update 型が用いられることは少なかった．それは，Update 型における冗長な更新，つまりプロセッサが必要としなくなったキャッシュコピーに対する更新が，ボトルネックとなりやすいバスのトラフィックの増加を招くことに原因がある [5] [11]．この Update 型における冗長な更新を減少させ，バスのトラフィック増加の問題を解消する Competitive 型 [12] も提案されているが，スヌープ方式では有効に機能しないという評価結果が文献 [13] で報告されている．

ディレクトリ方式の分散共有メモリシステムにおいても，スヌープ方式での評価結果を受けて Invalidate 型のプロトコルが用いられてきた [37] [35]．ディレクトリ方式を対象としたプロトコルの性能評価は Invalidate 型に限られ [7] [8]，Update 型や Competitive 型の性能評価は報告されていない．しかし，ディレクトリ方式のシステムは，スヌープ方

式のシステムと異なる性能特性を持ち，スヌープ方式での評価結果がそのまま当てはまらない．

そこで，本章ではディレクトリ方式を用いた分散共有メモリシステムを対象に，Invalidate，Update，Competitive の代表的な 3 つの一貫性制御プロトコルについて比較評価を行う [90]．実行駆動型のシミュレータを用い，4 つのワークロードを動作させて，読出しや書込みにともなうメモリアクセスレイテンシが性能に与える影響を評価する．また，プロトコル間で性能差が発生する要因の分析を通じて，ディレクトリ方式のどの特性が影響を持っているのかを明らかにする．

以降，2.2 節では，評価対象の 3 つの一貫性制御プロトコルについて説明する．2.3 節ではシミュレーション対象の計算機モデル，ワークロードについて説明する．2.4 節では評価結果を示し，最後に 2.5 節でまとめる．

2.2　評価するプロトコル

評価対象である以下の 3 種の一貫性制御プロトコルについて述べる．

1. Invalidate (Illinois 方式 [14])
2. Update (Fierfly 方式 [15])
3. Competitive (Fierfly 方式を改良)

いずれのプロトコルにおいても，キャッシュに存在するメモリブロックのコピーには以下の 2 種類の状態し，キャッシュのタグで管理される．

- システム内で唯一存在するコピーであり，プロセッサがブロックを占有している状態 (Exclusive).
- システム内で複数存在するコピーの一つであり，複数のプロセッサでブロックを共有している状態 (Shared).

プロセッサは，キャッシュアクセス時に上記の状態を確認し，その状態に応じた処理を行う．以降プロセッサが読出し/書込みを行った時の各プロトコルの動作を概説する．

2.2.1　読出し時の動作

3 プロトコルともプロセッサが読出しを行った時の動作は同一である．キャッシュにヒットした場合キャッシュからデータを読出し処理は完了する．

キャッシュミスした場合の動作を図 2.1 と図 2.2 に示す．図 2.1 はメモリに最新のデータが存在するケースであり，図 2.2 は他のキャッシュが最新のデータを占有で保持する

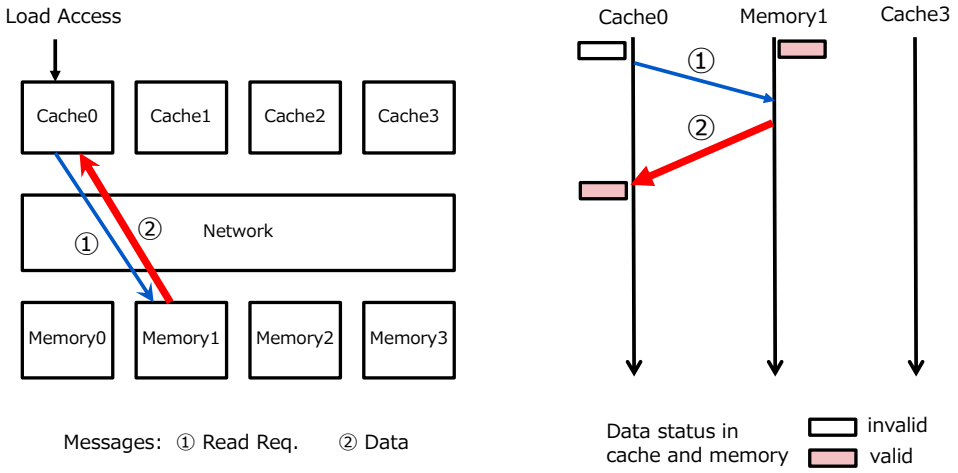


図 2.1　読出しキャッシュミス時の動作 (メモリに最新のデータが存在する場合の動作)

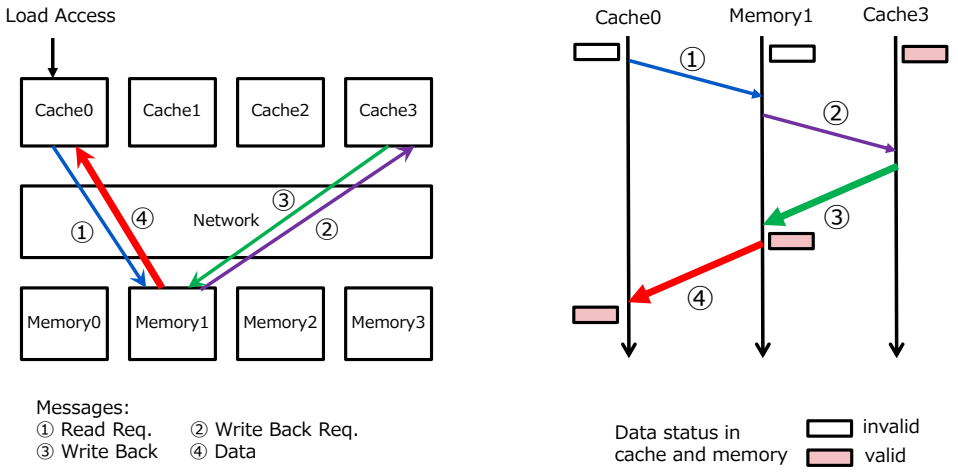


図 2.2　読出しキャッシュミス時の動作 (他のキャッシュに最新のデータが存在する場合の動作)

ケースである．どちらのケースも，まずメモリに対して読出し要求が発行される (① Read Req. メッセージ)．読出し要求を受けたメモリは最新のデータがメモリに存在するかどうかを確認し，存在すれば図 2.1 に示すようにデータをリプライする (② Data メッセージ)．メモリに最新のデータが存在しない場合は，図 2.2 に示すように最新のデータを保持するキャッシュに対して書き戻し要求を発行し (② Write Back Req. メッセージ)，キャッシュにデータの書き戻しを行わせる (③ Write Back メッセージ)．そして，データをリプライする (④ Data メッセージ)．

今，プロセッサの読出しに対して，Read Req. メッセージが発行される割合，即ち読出し時のキャッシュミスヒット率を**読出し要求率**と定義する．また，メモリが読出し要求を

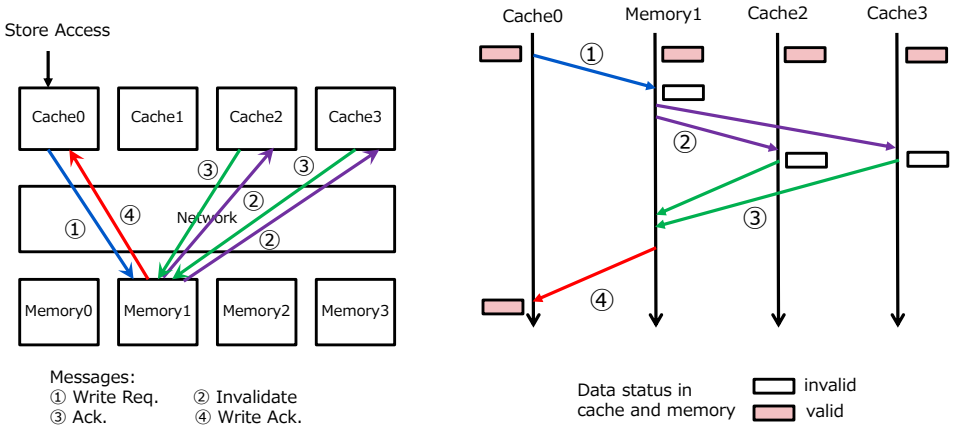


図 2.3　共有データに対する書込み時の動作 (Invalidate 型)

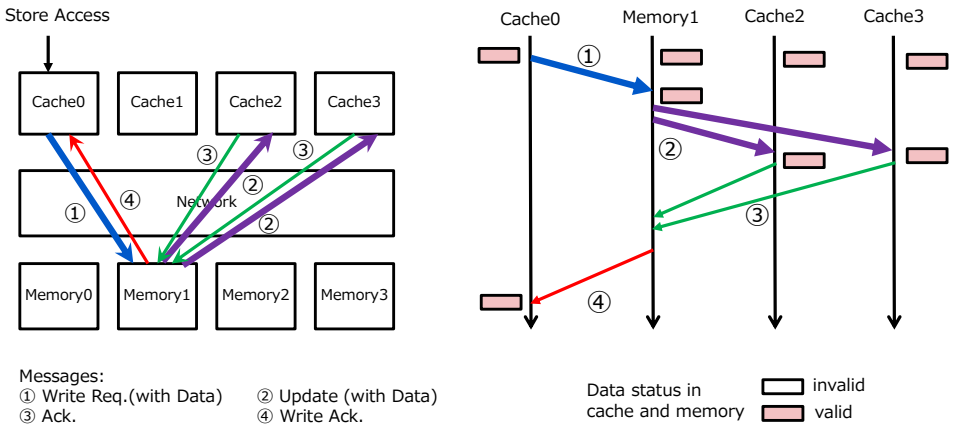


図 2.4　共有データに対する書込み時の動作 (Update 型)

受けた時に，最新のデータがメモリに存在せずキャッシュに対して書き戻し要求 (Write Back Req.) を発行しなければいけない割合を**書き戻し要求率**と定義する．

ディレクトリ方式において，プロセッサの読出しに起因するレイテンシは，以下の 2 項目に大きく依存する．

1. 読出し要求率 (RReqRatio)
2. 書き戻し要求率 (WBReqRatio)

スヌープ方式でのレイテンシが読出し要求率 (RReqRatio) にのみ依存するの対し，ディレクトリ方式では書き戻し要求率 (WBReqRatio) にも影響を受ける．

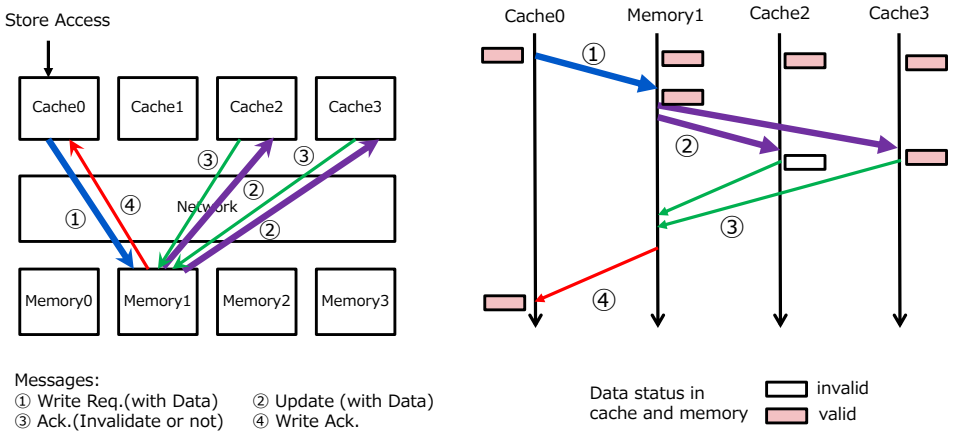


図 2.5　共有データに対する書込み時の動作 (Competitive 型)

2.2.2　書込み時の動作

占有しているブロックに対してプロセッサが書込みを行った場合の処理も 3 プロトコル共通しており，キャッシュにデータを書き込むことで処理は完了する．3 プロトコルで動作を異にするのは，キャッシュミス時，および共有しているブロックにヒットした場合である．

図 2.3、図 2.4、図 2.5 に共有しているブロックにヒットした場合の各プロトコルの動作を示す．3 プロトコルともメモリに対して書込み要求を発行し (① Write Req. メッセージ)，Update/Competitive 型の場合は書き込んだデータも同時に送られる．ただし，Update/Competitive 型については，一般にキャッシュに数エントリのライトバッファが設けられ，同一ブロックに対する書込みのマージが行われる．そのため，書込み要求はプロセッサの書込み毎ではなく，マージされた結果出てくる．書込み要求を受けたメモリはコピーを保持する全てのキャッシュに対して，Invalidate 型の場合は無効化要求を発行し (② Invalidate メッセージ)，Update，Competitive 型の場合は更新要求を発行する (② Update メッセージ)．この時，Invalidate 型ではコピーは無効化され，Update 型ではデータの更新が行われる．また Competitive 型では，ある条件に基づきデータの更新を行うか無効化するかが決定される．この条件については次節で詳しく述べる．メモリはキャッシュコピーに対する処理が終了するのを待ち (③ Ack メッセージ)，全コピーに対する処理が完了すると，キャッシュに対して処理完了を通知する (④ Write Ack メッセージ)．

キャッシュミスした場合の動作は共有ブロックに対して書込みを行った時の動作とほぼ同じであり，処理の最後にメモリからキャッシュに対してブロックデータがリプライさ

れる．

今，プロセッサの書込みに対して書込み要求が発行される割合を**書込み要求率**と定義する．また，書込み要求を受けたメモリが無効化/更新要求を伝えるキャッシュコピーの平均数を**平均ライト分配数**と定義する．

プロセッサの書込みに起因するレイテンシは，以下の 2 項目に依存する．

1. 書込み要求率 (WReqRatio)
2. 平均ライト分配数 (WDistAveNum)

スヌープ方式でのレイテンシが書込み要求率 (WReqRatio) にのみ依存するの対し，ディレクトリ方式では平均ライト分配数 (WDistAveNum) にも影響を受ける．

2.2.3　Competitive 型の動作

Competitive 型は，冗長な更新を削減するために Update 型を改良したものであり，ブロックに対する自プロセッサのアクセス頻度と他プロセッサの更新頻度から，キャッシュコピーを無効化するか更新するかを決定する．キャッシュの各ブロックにおいて，プロセッサが最後にアクセスしてから受け取った Update メッセージの数をカウントし，この値が一定値（更新回数の上限値）以上になると無効化を行う [12]．

具体的にはキャッシュブロック毎に数ビットのカウンタを設け，以下のような制御を行う．

- 読出しあるいは書込みよりデータをフェッチした時のカウンタの初期値は 0 とする．
- キャッシュを所有するプロセッサからのアクセスが発生するとカウンタを 0 にする．
- Update メッセージを受けるとカウンタに 1 を加える．この時，カウンタの値が更新回数の上限値未満であれば更新を行い，上限値以上であれば無効化を行う．

また，Ack メッセージに無効化したか否かの情報を付加し，その情報をもとにメモリでディレクトリを更新する．

2.3　プロトコルの評価方法

評価に用いたシミュレータおよびワークロードについて述べる．

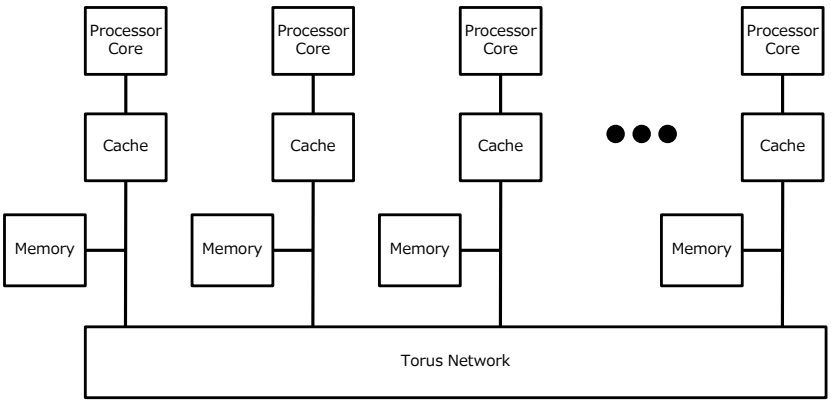


図 2.6 計算機モデル

表 2.1 シミュレーションのパラメータ

プロセッサ	1 命令の実行	1(cycle)
キャッシュ	ラインアクセス	2(cycle)
	ワードアクセス	1(cycle)
	タグアクセス	1(cycle)
ネットワーク	ラインデータなし (1hop)	8(cycle)
	ラインデータ付き (1hop)	24(cycle)
メモリ	ラインアクセス	14(cycle)
	ワードアクセス	10(cycle)
	ディレクトリアクセス	10(cycle)

2.3.1 シミュレータ概要

シミュレーションには実行駆動型のシミュレータを用い，プロセッサ部で命令レベルのシミュレーションを行った．また，メモリコンシステンシモデルは Weak [17] とした．Weak モデルにおいては，プロセッサは書込みアクセスを発行したときにそれによる一貫性制御の完了を待たず，メモリ同期命令が発行されたタイミングで先行する書込みアクセスによる一貫性制御の完了を保障するものである．これにより，書込みによるメモリアクセスレイテンシの隠蔽が可能なモデルとなっている．

評価の対象とする計算機のモデルを図 2.6 に示す．以下にこの計算機モデルの構成を示す．また，各部でのレイテンシを表 2.1 にまとめる．

[プロセッサ] 全命令 1 クロックで動作.

[キャッシュ] 1 階層のみとし，構成はダイレクトマップ．SuperSPARC プロセッサ [19] にあわせて， サイズは 1M バイト，ブロックサイズは 32 バイトとした.

Update/Competitive 型には，同一ラインに対する書込みのマージを行うライト・バッファを 2 エントリ用意．また，プロトコルによる性能差は書込みアクセスによって発生するので，読出しアクセスのみの命令フェッチはシミュレーションせず，データアクセスのみを評価．

[メモリ] フル・マップのディレクトリをブロック (32 バイト) 単位に保持．

[ネットワーク] 双方向 (リンク共有) のトーラスネットワークで，フローコントロール方式は store-and-forward [16] を採用．

2.3.2 ワークロード

共有メモリ型のマルチプロセッサシステムを評価するために様々な科学技術計算の分野から選ばれた実用的なアプリケーション群であり，多くの並列システムの評価に用いられている SPLASH [18] から以下に示す四つを選択してワークロードとした．WATER, Barnes-Hut，MP3D の三つは，SPLASH [18] のプログラムをそのままワークロードとして用いた．SOR は，SPLASH に属するプログラム OCEAN で用いられているアルゴリズムの一部を抜き出してワークロードに用いた．

- WATER — 液体状態の水分子の挙動をニュートン方程式を用いてシミュレーションするプログラム．シミュレーションは 125 分子，2 タイムステップ行い，2 タイムステップ目の実行のデータを収集した．
- Barnes-Hut — Barnes-Hut 法を用いて銀河の動きをシミュレーションするプログラム．シミュレーションは 512 粒子，2 タイムステップ行い，2 タイムステップ目の実行のデータを収集した．
- MP3D — モンテカルロ法を用いた流体力学シミュレーションプログラム．シミュレーションは 16384 粒子，平らな板が存在する $16 \times 16 \times 8$ の空間で，2 タイムステップ行い，2 タイムステップ目の実行のデータを収集した．
- SOR — Even-Odd 法による差分方程式の求解プログラム．シミュレーションは 512×512 のグリッドで，2 タイムステップ行い，2 タイムステップ目の実行のデータを収集した．

2.4 シミュレーションによるプロトコルの評価結果

前述の四つのワークロードを 32 プロセッサで動作させ，Invalidate，Update，Competitive の各プロトコルの性能に関わる様々なデータをシミュレーションによって収集した．なお，Competitive 型については，更新回数の上限値を 2 ～ 5 の範囲で変化させ，そ

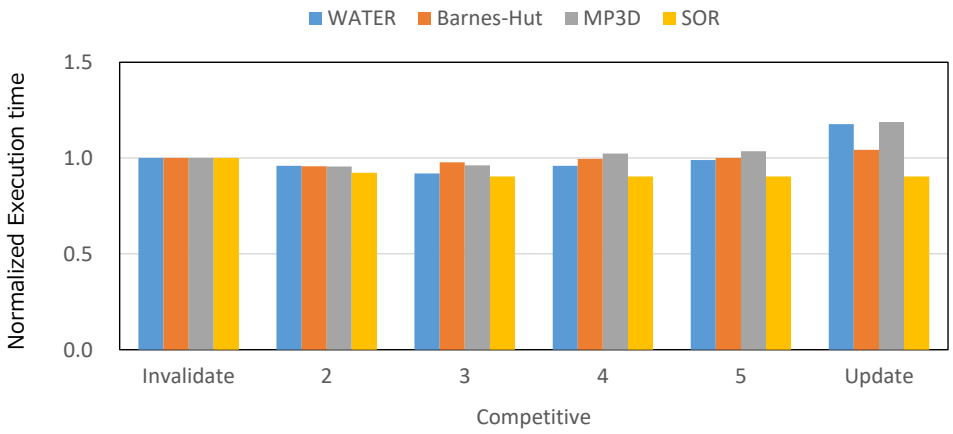


図 2.7　実行時間

れぞれについてのデータを収集した．図表中の 2 ～ 5 の値は，この上限値を表している．

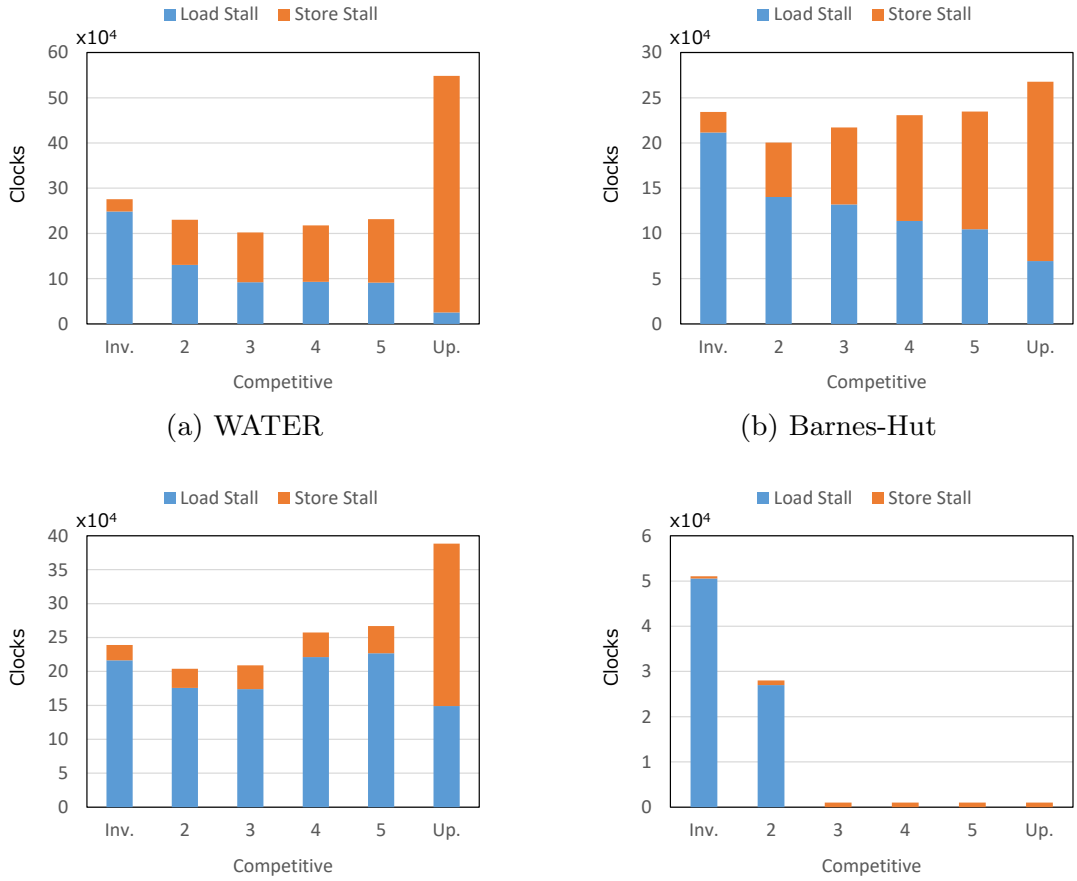
2.4.1　ワークロードの実行時間

プロトコルの違いが性能に与える影響を比較するために，ワークロードの実行時間を測定した結果を図 2.7 に示す．Update 型の実行時間を見ると，SOR では Invalidate 型よりも 10% 改善しているものの，他の 3 つのワークロードでは 4 ～ 19% 悪化している．Competitive 型は，更新回数の上限値を 2，3 程度に押さえた場合どのワークロードでも安定して良い性能を示しており，Invalidate 型を基準にして，2 ～ 10% の性能向上が得られている．更新回数の上限値を 4,5 にすると，Update 型で性能改善が見られた SOR を除き，徐々に性能改善効果が小さくなり，MP3D では悪化に転じている．

この結果より，Invalidate 型と比較して，Update 型はワークロードによって性能が大きく変わり多くの場合で性能が悪化するものの，Competitive 型は更新回数の上限値を 2 あるいは 3 に設定した場合に有効に機能し，性能を改善する効果を持つことが確認できた．

2.4.2　プロセッサストール時間

前節で示したプロトコルによる実行時間の差が発生する要因を解析するために、読出しおよび書込みによって発生するプロセッサストール時間を測定した結果を図 2.8 に示す．プロセッサストール時間は全プロセッサの平均を取った結果を示している．読出しアクセスによるプロセッサストールは，プロセッサが読出し要求を発行してからデータがキャッシュにフェッチされてアクセス可能になるまでの間発生する．そのため，キャッシュミス時のメモリアクセスレイテンシがそのままプロセッサストール時間として積算される．一



(a) WATER　(b) Barnes-Hut

(c) MP3D　(d) SOR

図 2.8　プロセッサストール時間

方，書込みアクセスによるプロセッサストールは，メモリコンシステンシモデルとして Weak [17] モデルを採用したことにより，プロセッサがメモリ同期命令を発行してから先行する書込みアクセスの一貫性制御が完了するまでの間となる．そのため，書込みアクセス時のメモリアクセスレイテンシはある程度隠蔽され処理性能に与える影響は抑えられている．

ストール時間全体を見ると，実行時間とおおむね同様の傾向を示し，Update 型は，SOR では Invalidate 型よりも 98% 削減できているものの，他の 3 つのワークロードでは 14 ～ 99% 増加している．Competitive 型は，SOR を除き，更新回数の上限値を 2 あるいは 3 にした場合どのワークロードでも安定して削減できており，Invalidate 型を基準にして 7 ～ 23% 改善している．また，更新回数の上限値を増やすと悪化する傾向も確認できる．一方，SOR では更新回数の上限値を 3 以上に設定した場合に Update 型と同様の大きな削減効果が得られている．

読出しに起因するストール時間を見ると，Update 型では 31 ～ 100% と大きく削減さ

表 2.2 読出し要求率 (RReqRatio)

	Inv.	Competitive				Up.
		2	3	4	5	
WATER	0.18	0.13	0.09	0.09	0.09	0.003
Barnes-Hut	0.23	0.17	0.16	0.15	0.13	0.09
MP3D	5.28	5.02	4.87	4.72	4.57	0.61
SOR	0.35	0.18	0.03	0.03	0.03	0.03

れていいる．Update 型よりも削減効果は小さいものの，Competitive 型はおおむね更新回数の上限値を増やすほど減少する傾向を示し，更新回数の上限値を 2 に設定した場合に 19 ～ 48% 削減され，5 に設定した場合は 51 ～ 100% 削減されている．

書込みに起因するストール時間を見ると，Update 型は SOR では 2 倍に留まるが，他のワークロードでは 8.8 ～ 19 倍と大幅に増大している．一方，Competitive 型も増加しているものの，その割合は Update 型と比較して小さい．Update 型で増加の割合が小さい SOR では，更新回数の上限値にかかわらず 2 倍となっているが，他のワークロードでは更新回数の上限値を 2 にすると 1.6 ～ 4.0 倍に増加し，5 にすると 1.8 ～ 5.1 倍に増加している．更新回数の上限値を大きくするほど増加の割合も大きくなる傾向を示している．

以上より，Invalidate 型と比較して，Update 型では読出しに起因するストール時間は大幅に削減できているものの，多くのワークロードではその削減効果を超えて書込みに起因するストール時間が増加し，プロセッサストール時間全体の増加に繋がっていることが確認できた．Competitive 型は，更新回数の上限値を 2 あるいは 3 に設定した場合に，Update 型よりは効果は低いものの読出しに起因するストール時間を削減し，Update 型で問題となっている書込みに起因するストール時間の増加を抑え，いずれのワークロードでも全体のストール時間の削減が達成できた．一方，更新回数の上限値を 4,5 と増やしていくと，多くのワークロードで書込みに起因するストール時間の増加の影響が大きくなり，全体のストール時間が増加する傾向を示した．

2.4.3 読出しストール時間の解析

前節で示した読出しに起因するプロセッサストール時間が，各プロトコルのどのような振舞いによって生じているかを分析するために，読出し要求率 (RReqRatio)，書き戻し要求率 (WBReqRatio) を測定した結果を，それぞれ表 2.2，2.3 に示す．

Invalidate 型を基準として，Update 型ではどれだけ読出し要求率 (RReqRatio) が削減できたかを見ると，最大では WATER の 98%，最小でも Barnes-Hut の 61% となった．また書き戻し要求率 (WBReqRatio) で見ると，Invalidate 型では最低の Barnes-Hut も

表 2.3　書き戻し要求率 (WBReqRatio)

	Inv.	Competitive				Up.
		2	3	4	5	
WATER	57.2	15.5	17.0	13.3	9.3	0
Barnes-Hut	19.9	14.1	9.9	8.3	4.6	0.6
MP3D	91.6	30.5	2.4	1.6	0.4	1.1
SOR	69.3	50.4	47.3	47.3	47.3	47.3

20% 程度，その他は 50% 以上の非常に大きな値を取っているのに対して，Update 型では SOR 以外のワークロードで数 % 以下の非常に低い値となっている．

Competitive 型でも，Update 型と同様に，読出し要求率 (RReqRatio) と書き戻し要求率 (WBReqRatio) を削減することができており，更新回数の上限値を増やすほど大きくなっていることが確認できる．ワークロードによって要求率の削減効果の傾向は異なり，WATER，Barnes-Hut，SOR では，読出し要求率 (RReqRatio) が 26 ～ 91% と削減され，書き戻し要求率 (WBReqRatio) も 28 ～ 84% と削減されている．一方，MP3D では，読出し要求率 (RReqRatio) が 5 ～ 19% の削減に留まっているが，書き戻し要求率 (WBReqRatio) は 67 ～ 100% と大きく削減されている．

以上より，Update や Competitive 型を用いることによる，プロセッサの読出しに起因するストール時間の削減効果は，以下の 2 点から得られていることが明らかとなった．

(a) 読出し要求率 (RReqRatio) の減少
(b) 書き戻し要求率 (WBReqRatio) の減少

また，Competitive 型では，(a) の効果が少ないワークロードでも，(b) の効果が大きく現れ，ストール時間が削減されることが明らかとなった．スヌープ方式でのストール時間削減が (a) のみに依存するのに対し，ディレクトリ方式では (b) の効果が更に加わる．従って，ディレクトリ方式で Update や Competitive 型による読出しに起因するストール時間の削減効果が，スヌープ方式よりも増している．

2.4.4　書込みストール時間の解析

前々節で示した書込みに起因するプロセッサストール時間が，各プロトコルのどのような振舞いによって生じているかを分析するために，書込み要求率 (WReqRatio)，平均ライト分配数 (WDistAveNum) を測定した結果を，それぞれ表 2.4，2.5 に示す．

Invalidate 型を基準として，Update 型の書込み要求率 (WReqRatio) を見ると，1.2 ～ 2.7 倍の増加となった．ただし，書込みに起因するストール時間は SOR を除き書込み要

表 2.4 書込み要求率 (WReqRatio)

	Inv.	Competitive				Up.
		2	3	4	5	
WATER	0.23	0.27	0.31	0.34	0.37	0.41
Barnes-Hut	0.09	0.16	0.20	0.21	0.23	0.24
MP3D	9.1	12.5	17.1	20.0	20.4	20.6
SOR	1.26	1.55	1.55	1.55	1.55	1.55

表 2.5 平均ライト分配数 (WDistAveNum)

	Inv.	Competitive				Up.
		2	3	4	5	
WATER	1.7	2.5	2.8	3.2	3.4	13.2
Barnes-Hut	4.8	4.8	5.4	5.9	6.8	10.0
MP3D	1.0	1.5	1.8	2.1	2.5	10.3
SOR	1.0	1.0	1.0	1.0	1.0	1.0

求率の増加以上に悪化し，図 2.8 より 8.8 ～ 19 倍となっている．この結果は，平均ライト分配数 (WDistAveNum) の増加によると考えられる．表 2.5 を見ると，平均ライト分配数 (WDistAveNum) は，SOR を除き 2.1 ～ 10 倍の増加となっており，絶対値では 10 を超えている．平均ライト分配数 (WDistAveNum) が増加すると，複数のノードに書込み要求を転送する必要が生じ，メモリにおける一貫性制御の処理負荷が増大し，またネットワークトラフィックも増加する．そのため，書込み要求率 (WReqRatio) を超える悪化に繋がっている．

以上より，Update 型でのプロセッサのストール時間の増加の原因は

(a) 書込み要求率 (WReqRatio) の増加
(b) 平均ライト分配数 (WDistAveNum) の増加

の 2 点にあることが明らかとなった．スヌープ方式でのストール時間の増加が (a) のみに依存するのに対し，ディレクトリ方式では (b) の影響が加わり，大きな悪化に繋がっている．

次に，Competitive 型の書込み要求率 (WReqRatio) を見ると，どのワークロードでも，Invalidate 型と Update 型の中間的な値を取っており，更新回数の上限値が小さいほど Invalidate 型に近く，更新回数の上限値が大きい場合は Update 型に近い値になっている．更新回数の上限値が 2 の場合で，Invalidate 型に対して 1.2 ～ 1.8 倍の増加となり，5 の場合では 1.2 ～ 2.6 倍の増加となっている．一方，平均ライト分配数 (WDistAveNum)

を見ると，その値は増加しているものの Invalidate 型に近い値に留まっており，更新回数の上限値を 5 に設定した場合でも，1.0 ～ 2.5 倍の増加に抑えられ，増加の割合が大きい WATER や MP3D でも，絶対値では 4 以下になっている．

以上より，Competitive 型では，Update 型の書込みに起因するストール時間の増加を，以下の 2 点で抑制していることが明らかとなった．

(a) 書込み要求率 (WReqRatio) の抑制
(b) 平均ライト分配数 (WDistAveNum) の抑制

スヌープ方式でのストール時間が (a) のみに依存するのに対し，ディレクトリ方式では (b) の影響が加わり，大きな抑制に繋がっている．

2.5 結論

分散共有メモリシステムにおいては，メモリアクセス時にデータの一貫性を維持するためのキャッシュ一貫制御が必要となる．キャッシュ一貫性制御では，プロセッサを跨った通信が発生しその遅延がメモリアクセスレイテンシとなるため，一貫性制御の遅延を小さくすることがプロセッサの処理性能を向上させるために重要となる．

本章では，メモリアクセスレイテンシの改善を目的として，ディレクトリ方式を対象に，Invalidate，Update および Competitive 型の 3 種の分散共有メモリ型の性能評価を実施した．メモリアクセスレイテンシが処理性能に与える影響を明らかにするために，メモリアクセスに起因してプロセッサがストールした時間を計測した．

その結果，Update 型はワークロードの性質によりその振舞を大きく変化させることが明らかとなった．一方，Competitive 型は更新回数の上限値を 2，3 に設定した場合に，どのワークロードでも良好な性能を発揮することが明らかとなった．

Invalidate 型と比較して，Update 型では読出しに起因するストール時間は大幅に削減できているものの，多くのワークロードではその削減効果を超えて書込みに起因するストール時間が増加し，プロセッサストール時間全体の増加に繋がることが確認できた．Competitive 型は，更新回数の上限値を 2 あるいは 3 に設定した場合に，Update 型よりは効果は低いものの読出しに起因するストール時間を削減し，Update 型で問題となっている書込みに起因するストール時間の増加を抑え，いずれのワークロードでも全体のストール時間の削減が達成できた．

Update や Competitive 型を用いることによる，プロセッサの読出しに起因するストール時間の削減は，読出し要求率 (RReqRatio) および書き戻し要求率 (WBReqRatio) の減少から得られていることも明らかとなった．Competitive 型では，読出し要求率 (RReqRatio) の減少効果が少ないワークロードでも，書き戻し要求率 (WBReqRatio) の

効果が大きく現れ，ストール時間が削減された．Update 型を用いることにより，プロセッサの書込みに起因するストール時間の増加は，書込み要求率 (WReqRatio) の増加と平均ライト分配数 (WDistAveNum) の増加に原因があることが確認された．Competitive 型は，それらの増加，特に後者の平均ライト分配数 (WDistAveNum) の増加を抑えてストール時間の悪化を抑制した．スヌープ方式では読出し要求率 (RReqRatio) や書込み要求率 (WReqRatio) に依存してストール時間が変化するのに対して，ディレクトリ方式では書き戻し要求率 (WBReqRatio) や平均ライト分配数 (WDistAveNum) にも影響を受けて変化するため，スヌープ方式では効果のなかった Competitive 型が有効に機能することが明らかとなった．

今後，ノード数やネットワーク性能を変更した評価，またワークロードを増やし評価を行う．また，これらの評価結果から，ネットワークのマルチキャスト・ギャザー機能を用いて平均ライト分配数が多くなっても書込みに起因するアクセスレイテンシの増大を防ぐことが，ディレクトリ方式の分散共有メモリシステムにおいては重要であることも再確認され，これについては 3 章で議論する．

第 3 章

ビットパターン・ディレクトリと汎用的なマルチキャスト・ギャザー機構

3.1 はじめに

本章では，高性能かつ安定動作するディレクトリ方式の一貫性制御の実現における四つの課題のうち，二つ目の一貫性制御の負荷低減とメモリアクセスレイテンシの削減を目的としたディレクトリと一貫性制御手法について述べる．

ディレクトリ方式では，システム規模が大きくなりデータを共有するノード数が増加すると，書込み時の一貫性制御に要する処理遅延が増大する問題があった．Stanford DASH [37]，MIT Alewife [35]，SGI Origin2000 [38] 等のシステムでは，メモリからコピーを保持するノードへの書込み要求の転送を順に行い，またそれらからの応答を一つずつ受け取る方式を取っていた．そのため，これらの処理に保持しているノード数に比例する時間を必要とする問題があった．

ディレクトリ情報の構成法がこの問題をさらに悪化させた．ディレクトリ方式においては，システムを構成するノード数にかかわらずディレクトリに必要なメモリのサイズがノードあたり一定であることが，ハードウェアコストの観点で不可欠となる．従来，ディレクトリにおいてデータを保持しているノードを管理する手法として，Full Map 方式 [27] が提案されていたが，ノード数に比例するビット数を必要とするため大規模システムにはコスト的に適用できない問題があった．この問題を，少ビットでディレクトリを構成することで解決する様々な手法 [29, 31] が提案されてきた．しかし，これらの方式では，実際にデータを保持しているノードに対して，非常に多数のノードを保持している可能性のあるノードとして一貫性制御の対象とする [31]．そのため，書込み時の一貫制御に

要する処理遅延をさらに悪化させる問題があった．

ノード数に比例する時間を必要とする問題を解決する手法として，書込み要求の送信をマルチキャストする手法が提案されていた [20, 21]．これらは，トーラス等のネットワークにおいて，デッドロックが起こらないマルチキャスト手法について議論していた．しかし，書込み要求に対する複数の応答については一つずつ受け取る方式となっており，[22] で示されているように，書込みの遅延は複数の応答の受信がボトルネックとなり共有するノード数に比例する時間を要する問題が残っていた．

[23] では，トーラスを拡張したネットワークでのマルチキャスト手法を提案し，そのマルチキャストパターンとしてネットワークの階層構造にあわせた形式を用いることを提案していた．また，同時に応答についてもギャザー機能によってネットワーク内で一つのメッセージに集約する手法を提案していた．この手法を用いることにより，書込みの遅延を一定に抑えることを実現できていた．しかし，この手法はネットワーク構造に依存したマルチキャストパターンを前提とする手法で，メモリで保持するディレクトリをそれに合わせる必要があった．そのため，Coarse Vector [29] 等の方式と比較しても，実際に保持しているノードに対してマルチキャスト時の宛先ノード数が大幅に増加し，メモリアクセスレイテンシが悪化する問題があった [31]．さらに，応答の集約についても，応答メッセージがマルチキャストメッセージと同じスイッチを逆にたどることを前提とした方式であったため，両メッセージで通るスイッチが異なるネットワークには適用できない問題があった．また，少ビットのディレクトリで非常に多数のノードを保持している可能性のあるノードとして一貫性制御の対象とするため，例えマルチキャストとギャザーを用いてもネットワークの負荷や要求を受けるノードの負荷が高く，メモリアクセスレイテンシが悪化する問題が残った．

本章では，これらの問題を解決するビットパターン方式のディレクトリと，汎用的なマルチキャスト・ギャザー機能を実現するネットワークを提案する [89]．提案するビットパターン方式は，ノードを階層的に分割して管理し各階層による分割サイズが異なることを特徴とする．この特徴により，特に多数のノードからなる大規模システムの一部のノードを利用してアプリケーションを動作させるケースで，保持している可能性のあるノード数の増加を抑えることができる．提案する汎用的なマルチキャスト・ギャザー機能は，本章で提案するビットパターン方式や，Coarse Vector [29]，Limited Pointer [8] 方式等のディレクトリ情報を活用可能で，また様々なトポロジのネットワークで実現可能であることを特徴とする．これにより，従来のマルチキャスト・ギャザー手法が適用なかった，ディレクトリやネットワークトポロジを採用したシステムにおいても，ノード数に依存しないメモリアクセスレイテンシの実現が可能となる．

以降，3.2 節でディレクトリ情報とマルチキャスト・ギャザー機構の従来方式とその課題を示す．3.3 節でビットパターン方式を提案し，他の方式と比較して保持している可

能性のあるノード数の増加を抑えられることを示す．3.4 節で提案するマルチキャスト・ギャザー方式とその効果を示す．3.5 節で，並列計算機 Cenju-4 におけるビットパターン方式およびマルチキャスト・ギャザー機能の実装と，実機で書込み遅延を測定した結果を示す．3.6 節でまとめる．

3.2 従来方式とその課題

3.2.1 ディレクトリ情報

従来，ディレクトリで保持ノードを管理する方式として様々な方式が提案されている [27, 29, 31, 35]．

フルマップ方式 [27] は，Stanford DASH [37] で採用された方式であり，ノード数分のビットを用いて，各ノードのキャッシュがコピーを保持しているかどうかを該当するビットで記録する方式である．しかし，この方式はノード数に比例したビット数を必要とするため小規模システムであれば良いが，例えば 1024 ノードなどの大規模システムであれば 1024 ビット必要となり，その保持コストが非常に大きくなる問題があった．

従来，フルマップ方式の問題を解決するものとして幾つかの方式が提案されており，以下に代表的な 3 つの方式について述べる．

Limited Pointer 方式は，MIT Alewife [35] で採用された方式であり，少数のノードへのポインタをディレクトリに保持する方式である．例えば，1024 ノードのシステムであれば，10 ビットのポインタとそのポインタが有効かを示す 1 ビットの計 11 ビットからなるフィールドを複数保持する方式となる．そのため，保持ノード数が設けたフィールド数以下の場合には，無駄な一貫性制御の発生を防ぐことができる．一方，保持ノード数がフィールド数を超えると，以降は全ノードが保持しているとみなすため，大幅に無駄な一貫性制御が増加する問題があった．

Coarse Vector 方式 [29] は，SGI Origin 2000 [38] で採用された方式である．1 ビットで複数のノードをまとめて管理する方式となる．例えば 1024 ノードを 64 ビットで管理する場合，1024 ノードを 16 ノードからなる 64 個のグループに分割し，グループ内にコピーを保持するノードが一つでも存在すれば，該当するビットをセットする方式となる．この方式では，グループ内に一つでも保持しているノードが存在すると，グループ内の全てのノードが保持している可能性があるノードとみなされ，一貫性制御においてグループ内の全ノードを対象とした処理が必要となる問題があった．

また，階層ビットマップ方式 [31] は，Jump-1 [42] で採用された方式である．深さ m の n 進木において末端の葉をノードとみなし，各階層においてどの枝の先にデータを保持するノードが存在するかをビットマップで保持する方式となる．各階層の全てのビット

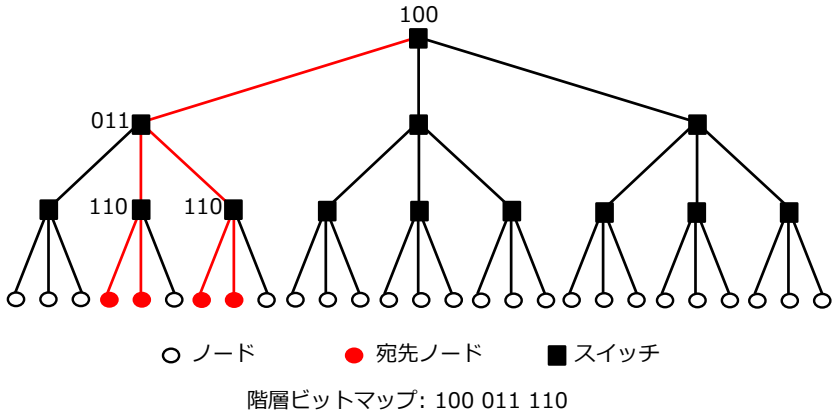


図 3.1　階層ビットマップによるマルチキャストとギャザー

マップを一つのビットマップに論理和して管理することで，必要なビット数の削減を実現している．例えば，1024 ノードは，深さ 5 の 4 進木で管理することができ，20 ビットに抑えることができる．この方式においても，各階層で一つのビットマップで管理するため，実際に保持しているノード数よりも多くのノードを保持している可能性のあるノードとみなすこととなる．

従来，提案されてきたこれらの方式は，ディレクトリの保持コストを抑えることには有効であった．また保持している可能性のあるノード数の増大もある程度抑えていた．しかし，これらの方式はいずれもシステムの全てのノードを用いて処理が動作することのみを想定し設計されていた．大規模なマルチプロセッサシステムの利用において頻繁に発生する，システムを分割し一部のノードを利用して処理を実行するケースが想定されてこなかった．例えば，1024 ノードのシステムで 16 ノードを用いて動作する場合に，64 ビットの Coarse Vector 方式では 64 ビット中 1 ビットのみが利用され，その 1 ビットで 16 ノードを管理する方式となる．そのため，1 ノードがデータをキャッシュするだけで全ての 16 ノードが保持するものと見なしてしまう問題があった．

3.2.2　マルチキャストとギャザー方式

[23] では，トーラスを拡張したネットワークでの書込み要求のマルチキャストとその応答のギャザー手法を提案していた．本節ではその方式と課題について述べる．

マルチキャスト方式は，ネットワークが n 進木の構造を内包することを前提とし，その階層構造に合わせた階層ビットマップ方式でマルチキャストの宛先を指定することを特徴とする．図 3.1 は，27 ノードからなるシステムで 3 層からなる 3 進木をネットワークが内包する場合を示している．あるノードから，複数の宛先ノードにマルチキャストする

場合は，ネットワーク内でまず送信元ノードからルートに相当するスイッチに要求が送られ，そのルートのスイッチから図示したルートで階層ビットマップで指定されたパターンに従ってメッセージがマルチキャストされる（図中の赤線で上から下）．図に示したケースでは指定した階層ビットマップによって 4 ノードに対してメッセージがマルチキャストされる．

次に，ギャザー方式は，マルチキャストと逆のルートでメッセージが配信されることを前提とする方式となる（図中の赤線で下から上）．マルチキャスト時にネットワークの各スイッチにて応答待合せ用のリソースを確保する．書込み要求に対する応答を送出する各ノードは，マルチキャスト時とは逆のルートでメッセージを送信する．各スイッチで確保されたリソースを用い，階層ビットマップで指定された形式に合わせて待ち合わせを行い，メッセージを一つに集約し，上の階層のスイッチに対してメッセージを送信する．ルートスイッチでの集約が終わると，応答の宛先ノードに対して一つのメッセージが送信される．これによって，図 3.1 のケースでは 4 ノードからの応答を一つのメッセージに集約することができる．

この方式により，書込み要求の送信とその応答の受信を保持ノード数にかかわらず一定の時間で処理することが可能となっている．また，ネットワークが内包している木構造と一致した階層ビットマップ方式を用いることで，スイッチにおけるマルチキャストの実装が容易となっている．

しかし，この方式には次の二つの課題が存在する．一つは，一貫性制御の対象となるノード数が増加し，ネットワークや一貫性制御のメッセージを受けるノードの負荷が増大する問題である．これは，ディレクトリ情報として，階層ビットマップ方式を用いる必要があり，他の方式を用いることができないことに起因する．3.3 節の評価で示すが，Coarse Vector 方式や提案するビットパターン方式と比較して，階層ビットマップ方式は一貫性制御の対象となるノード数が増加する．そのため，書込み要求をマルチキャストする時の宛先ノード数が増加し，ネットワークの負荷や一貫性制御の要求を受けるノードの負荷が増大する．

もう一つは，この実装方式はネットワークが n 進木のバランス木構造を内包していることを前提としている．そのため，Fat-tree や多段網等のネットワークであれば適用可能であるが，トーラスやハイパーキューブ等の他のトポロジになると適用困難となる．また，ギャザー機能については，マルチキャストと逆のルートでメッセージをルーティングすることを前提とする方式であるため，両者が異なるルートとなるネットワークには適用できない．例えば，トーラスネットワークにおいて，マルチキャストもギャザーも同じ次元オーダーのルーティングが行われる場合は，異なるルートとなるため適用できない．[23] では、この問題を解消しかつデッドロックを防止するために，マルチキャストとギャザーでそれぞれ異なるネットワークチャネルを用い，かつマルチキャストのルートとギャザー

のルートが同一になるように制御していた．しかし，この方式では複数のネットワークチャネルが必須になる問題がある．また，マルチキャストの送信元と，ギャザーの宛先を別とする場合 [37] には，やはり適用できない問題があった．

3.3　ビットパターン方式の提案と評価

本節で提案するビットパターン方式は，ディレクトリの保持コストを抑えつつ，特にシステムを分割して一部のノードを利用するケースにおいて，保持している可能性のあるノード数の増大を防ぐことを狙ったものである．

3.3.1　ビットパターン方式

ビットパターン方式は，ノードを階層的に分割して管理し各階層による分割サイズが異なることを特徴とする．この特徴を実現するために，以下のような手法によってノード番号集合からビットパターン表現を構築する．ノード番号 (m ビット) の i 個の集合が以下で表されるとする．

$$\{n_1, n_2, ..., n_i\}$$

この集合の各ノード番号 (n_x) について，m 個のビットを j 個のフィールドに分割し，j 個の要素を持つ i 個の集合を得る．

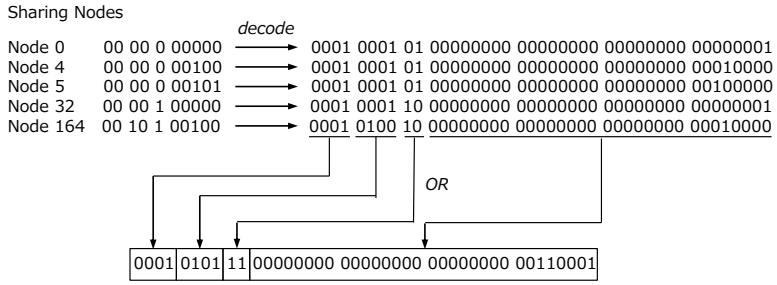
$$\{\{n_{11}, n_{12}, ..., n_{1j}\}, \{n_{21}, n_{22}, ..., n_{2j}\}, ..., \{n_{i1}, n_{i2}, ..., n_{ij}\}\}$$

なお，m ビットのノード番号の j 個のフィールドへのビット分割は，全ての i 個のノード番号について同一の規則で行う．最後に，j 個の全ての要素についてそれぞれの値をビットデコードしてビット論理和を取る．

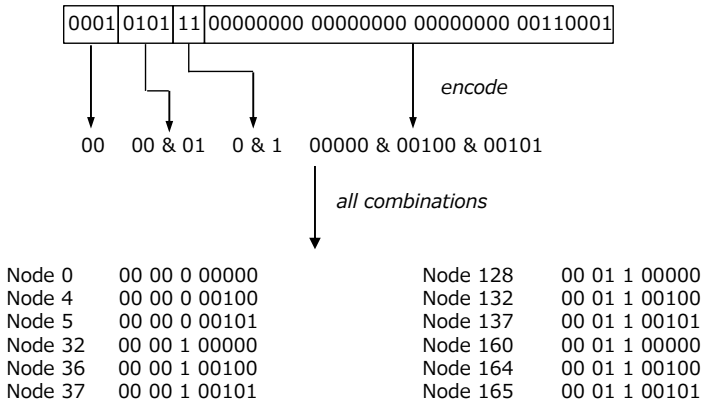
$$\{d_1, d_2, ..., d_j\}$$
$$d_x = bitdecode(n_{1x})|bitdecode(n_{2x})|...|bitdecode(n_{ix}) \qquad (x = 1..j)$$

このようにして得られた各要素の値 $(d_1, .., d_j)$ をビット連結した値がビットパターン表現となる．これは，ノードを葉とし，j の深さからなるバランス木構造で，$1 \sim j$ の各層の分岐数が $d_1 \sim d_j$ のビット数になることに相当する．x 層の全ての節において同一の d_x を用い，全ての層の対応する d_x の値が ‘1’ の分岐下の葉がデータを保持するノードとなる．以降ビットパターン表現を，総ビット数と，j 個の要素をビットデコードした後の bit 数で区別する．

図 3.2 に，ビットパターン方式の例を示す．図示した例は，1024 ノードを 42 ビットで管理する方式となっている．このビットマップ方式においては，保持するノードの番



(a) 保持ノード番号からビットパターン方式への変換



(b) ビットパターン方式から保持ノード番号への変換

図 3.2 ビットパターン方式の例

号（10 ビット）を 2-2-1-5 ビットの 4 つのフィールドに分割し，各フィールドをビットデコードして得られる 4-4-2-32 ビットのデータに変換し管理する．図のケースは，ノード 0，4，5，32，164 の 5 ノードがコピーを保持している場合である．図 3.2(a) は 5 ノード番号からビットパターン表現への変換例を示している．図 3.2(b) は，上記表現時にビットパターン表現でコピーを保持している可能性のあるノード番号への変換を示している．この例では，実際の保有しているノードは 5 ノードであるが，12 ノードが保持している可能性があることになる．

階層ビットマップ方式は，ビットパターンの一形態でノード番号を均等のビット数からなるフィールドに分割した方式となる．例えば深さ 5 の 4 進木で表現される階層ビットマップ方式は，ノード番号を 2-2-2-2-2 ビットの 5 つのフィールドに分割し 4-4-4-4-4 ビットの 20 ビットで表現されるビットパターン方式と同一のものとなる．一方，ビットパターン方式はフィールドの分割を自由に設計が可能で，非均等な分割により下位ビットを厚くする (図の例ではノード番号の下位 5 ビット) ことができる．この特徴により，例えば図示したビットパターン方式では，32 ノードを超えるシステムであっても，32 ノード以下しか使用しないプログラムであれば，ノード番号の上位 5bit が同一の物理ノード

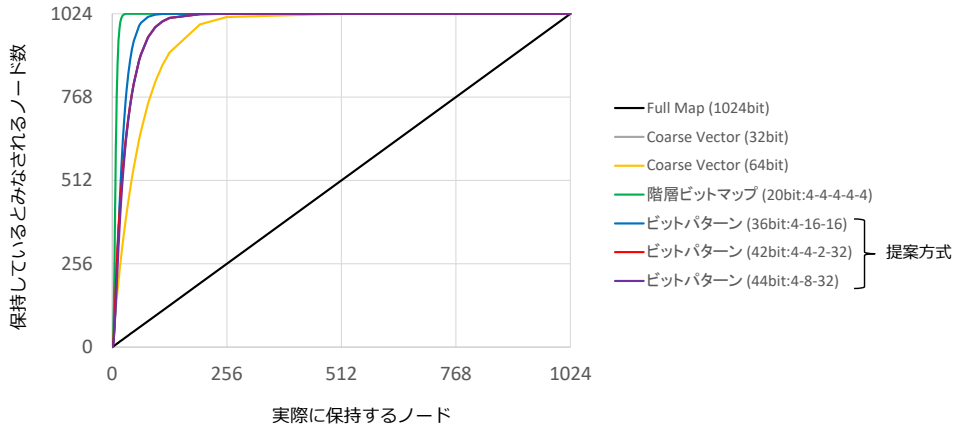
に処理を割り当てることで，厳密な保持ノード管理が実現できる．

3.3.2　保持ノード管理の評価

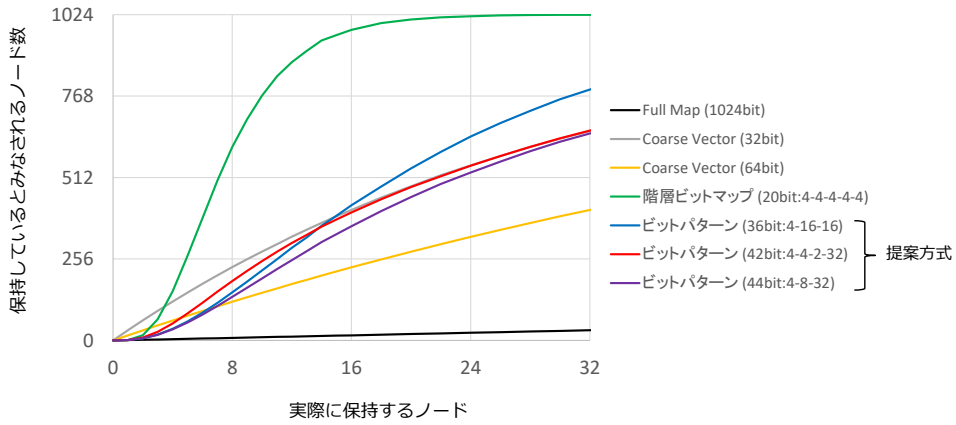
ノード数が 1024 からなるシステムを想定し，図 3.3 に Coarse Vector 方式 [29]，階層ビットマップ方式 [31]，提案するビットパターン方式を比較した結果を示す．横軸は実際に保持しているノード数（(a)(b) は 1024 ノードからランダムに選出し，(c) は 128 ノードからランダムに選出）を示し，縦軸はその場合に各保持ノード管理方式で保持している可能性があるとみなされるノード数を示している．Coarse Vector 方式は 32 ビットおよび 64 ビットの 2 ケースを示す．階層ビットマップ方式は 4-4-4-4-4 ビット (深さ 5 の 4 進木) の 20 ビットの場合を示す．ビットパターン方式は 4-16-16 ビットの 36 ビット (ノード番号を 2-4-4 ビットの 3 フィールドに分割)，4-4-2-32 ビットの 42 ビット (ノード番号を 2-2-1-5 ビットの 4 フィールドに分割)，4-8-32 ビットの 44 ビット (ノード番号を 2-3-5 ビットの 3 フィールドに分割) の場合を示している．また参考として，理想ケースの FullMap 方式も示している．

図 3.3(a) は，1024 ノードからなるシステムの全てのノードで処理が動作する場合を想定し，共有しているノード数を変化させた場合である．この結果より，保持するノード数が増加するといずれの方式も急速に保持している可能性のあるノード数が増加していくのが確認できる．本質的に 1024 ビット必要な情報をいずれも 64 ビット以下に圧縮しているため，それによって増加している．また，各方式を比較すると，Coarse Vector(64 ビット) が最も良く，次いで Coarse Vector(32 ビット)，ビットパターン (4-4-2-32 ビット)，ビットパターン (4-8-32 ビット) が同じ様相を示し，ビットパターン (4-16-16 ビット) と続き，階層ビットマップ (4-4-4-4-4 ビット) が最も悪化させている．この結果より，それぞれの方式で最もビット数が多いフィールドの数値に依存して保持している可能性のあるノード数が決まっていることが確認できる．そのため，最大のビット数をいかに大きくするかが重要であることが分かる．全てのビット数を一つのフィールドで占める Coarse Vector 方式が最も優れており，次いでビット数を自由に決定できるビットパターン方式，均等に分割する階層ビットマップ方式の順になっている．

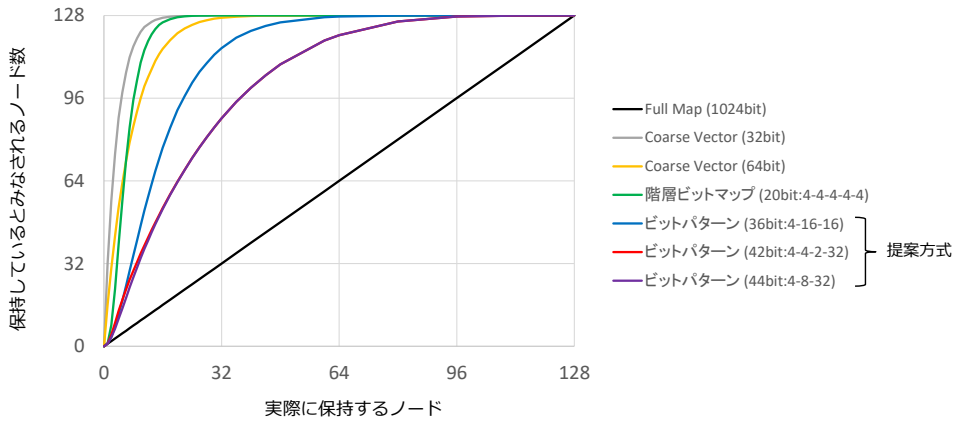
また，図 3.3(b) も，1024 ノードからなるシステムの全てのノードで処理が動作する場合を想定し，共有しているノード数を変化させた場合であり，保持ノード数が少ない場合に注目したグラフである．この図より図 (a) で重なって見えた Coarse Vector(32 ビット)，ビットパターン (4-4-2-32 ビット)，ビットパターン (4-8-32 ビット) を比較すると，その逆順で保持している可能性のあるノード数を抑えられていることが分かる．この結果より，ビットパターン方式における最大ビット数以外のフィールドが，保持ノード数が少ない場合に有効に機能して Coarse Vector 方式よりも良い数値を示すことが分かる．ま



(a) 1024 ノードで共有する場合 (0~1024 ノード)



(b) 1024 ノードで共有する場合 (0~32 ノード)



(c) 128 ノードで共有する場合 (0~128 ノード)

図 3.3　保持ノード管理の評価

た，同じビットパターン方式を比較すると，分割するフィールド数が少ない方が有効であることも確認できる．

次に，図 3.3(c) は，1024 ノードからなるシステムの一部の 128 ノードで処理が動作する場合を想定し，その 128 ノード内で共有しているノード数を変化させた場合である．この 128 ノードのノード番号の上位 3 ビットは同じ値を持つことを想定している．この図より，各方式を比較すると，図 3.3(a) の結果とは異なり，ビットパターン (4-8-32 ビット) とビットパターン (4-4-2-32 ビット) が良い性能を示し，次いで，ビットパターン (4-16-16 ビット)，Coarse Vector(64 ビット)，階層ビットマップ (4-4-4-4-4 ビット)，Coarse Vector(32 ビット) と続いている．この結果より，ビットパターン方式で想定したシステムを分割して利用するケースにおいて，本方式が有効に機能していることが確認できる．128 ノードの場合は下位 7 ビットのみが変化する場合となるため，その部分に影響を受けるフィールドのみを考慮した場合，ビットパターン (4-8-32 ビット) はビットパターン (4-32 ビット)，ビットパターン (4-4-2-32 ビット) はビットパターン (2-2-32 ビット)，ビットパターン (4-16-16 ビット) はビットパターン (8-16 ビット)，階層ビットマップ (4-4-4-4-4 ビット) は階層ビットマップ (2-4-4-4 ビット)，Coarse Vector(64 ビット) は Coarse Vector(8 ビット)，Coarse Vector(32 ビット) は Coarse Vector(4 ビット) に相当する．そのため，図 3.3(a) で解析した結果と同様で最大ビット数の大きな順で良い性能を示していることが確認できる．

以上の評価結果より，本節で提案したビットパターン方式が，様々な利用シーンにおいて保持ノードを効率よく管理することができる方式であることが確認できた．システムを構成する全てのノードを用いてアプリケーションが動作するケースでは，提案方式は Corase Vector 方式に次いで良い性能を示し，システムを分割して一部のノードを利用するケースでは，Corase Vector 方式や階層ビットマップ方式よりも，良い性能を示した．

3.4　マルチキャスト・ギャザー方式の提案

提案するマルチキャスト方式は，ビットパターン方式，Limited Pointer 方式，Coarse Vector 方式等のシステムが採用したディレクトリ形式で指定されるマルチキャスト宛先情報を基に各スイッチにおいて出力するポートを算出しマルチキャストを行うことを特徴とする．これにより，従来のマルチキャスト方式で課題であった，マルチキャストの宛先ノード数が大幅に増加する問題を解消する．また、提案するギャザー方式は，送信元の各ノードにおいて，送信元ノード集合（＝マルチキャストメッセージの宛先ノード集合）と宛先ノード情報から，ネットワークの経路上の各スイッチでどの入力ポートからのメッセージと待ち合わせを行うかを算出し，待合せ情報としてギャザーメッセージに付加することを特徴とする．これにより，様々なネットワークに適用可能な汎用的なギャザー機能

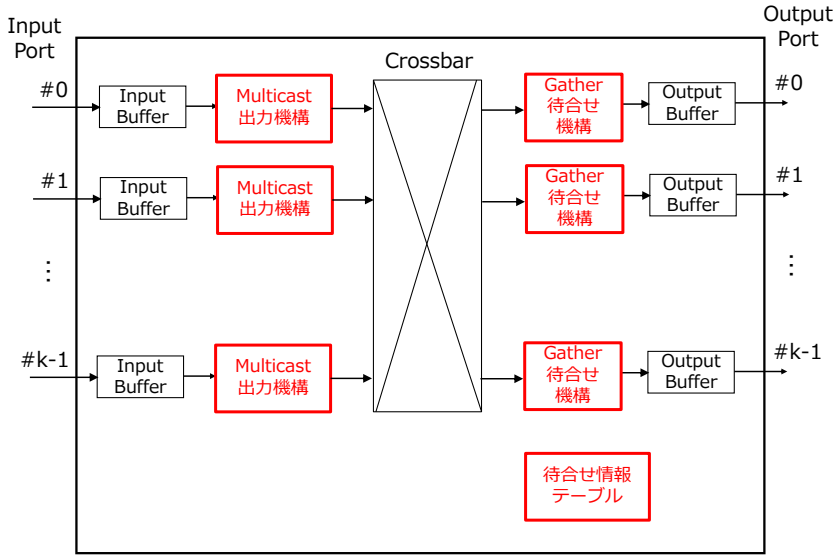


図 3.4　提案するマルチキャスト・ギャザー方式を実現するスイッチ構成

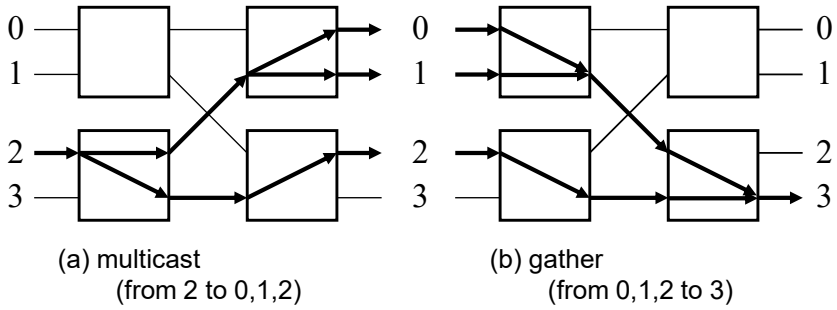


図 3.5　マルチキャストとギャザーの動作例

意を実現する．

提案するマルチキャスト・ギャザー機構を実現するネットワークスイッチの構成を図 3.4 に示す．図の赤字部分が提案機構によって追加で必要となる部分であり，入力ポート毎に Multicast 出力機構を持ち，出力ポート毎に Gather 待合せ機構を持つ．また，待合せ情報を記録する待ち合わせ情報テーブルを一つ持つ．

スイッチの Multicast 出力機構は，マルチキャストメッセージを受けて出力するポートを算出する．そのメッセージに付加されているマルチキャスト宛先情報（ディレクトリ形式）で示される宛先ノード集合と，出力ポートを経由して送信元ノードからメッセージが配送されるノード集合に重なりがある場合に，その出力ポートにメッセージを送出する必要があると判断する．算出された結果に従って複数の出力ポートにメッセージは送出されマルチキャストが実現される．

図 3.5(a) に本方式でのマルチキャストの動作例を示す．4 ノードからなるシステムが，2x2 のスイッチを用いた 2 段のネットワークで接続され，ノード 2 からノード 0,1,2 にマルチキャストする場合を示している．ノード 2 からのメッセージを受けた 1 段目のスイッチの Multicast 出力機構は，出力ポート 0 によって到達可能なノード集合が 0,1，出力ポート 1 によって到達可能なノード集合が 2,3 であり，宛先ノード集合が 0,1,2 であることから，出力ポート 0 と出力ポート 1 に対してメッセージを出力する．2 段目の上のスイッチの Multicast 出力機構は，出力ポート 0 によって到達可能なノード集合が 0，出力ポート 1 によって到達可能なノード集合が 1 であり，宛先ノード集合が 0,1,2 であることから，出力ポート 0 と出力ポート 1 に対してメッセージを出力する．また，2 段目の下のスイッチの Multicast 出力機構は，出力ポート 0 によって到達可能なノード集合が 2，出力ポート 1 によって到達可能なノード集合が 3 であり，宛先ノード集合が 0,1,2 であることから，出力ポート 0 に対してのみメッセージを出力する．これによって，ノード 2 からノード 0,1,2 に対してメッセージが送信される．

スイッチの Gather 待合せ機構は，ギャザーメッセージに付加された待合せ情報を基に複数の入力ポートからのギャザーメッセージの待ち合わせを行い最後のメッセージのみを出力する．なお，ネットワークは独立したギャザーを複数同時に行えるように，ギャザーメッセージにはどのマルチキャストに対する応答かを示す識別子が付与される．それぞれの識別子に応じて待ち合わせが行えるように，スイッチはその識別子数分のエントリからなる待合せ情報テーブルを保有し，そこに待ち合わせ情報を記録する．待合せ情報は入力ポート数分のビットからなり，各ビットは対応する入力ポートからのメッセージを未受信か受信済みかを記憶する．

図 3.5(b) に本方式でのギャザーの動作例を示す．4 ノードからなるシステムが，2x2 のスイッチを用いた 2 段のネットワークで接続され，ノード 0,1,2 からのメッセージをギャザーしてノード 3 に送信する場合を示している．ノード 0 とノード 1 は，1 段目のスイッチで入力ポート 0 および 1 からのメッセージを待合せ，2 段目のスイッチで入力ポート 0 および 1 からのメッセージで待合せる必要が有ると算出し，それを待合せ情報としてメッセージに付加する．ノード 2 は，1 段目のスイッチでは待ち合わせが不要であり，2 段目のスイッチで入力ポート 0 および 1 からのメッセージで待合せる必要が有ると算出し，それを待合せ情報としてメッセージに付加する．1 段目のスイッチが最初にノード 0 からのメッセージを受けると，Gather 待合せ機構において待合せの判定を行う．入力ポート 0 と入力ポート 1 からの待ち合わせが必要なことがメッセージに示されており，待合せ情報テーブルからまだどの入力ポートからもメッセージを受けていないことを確認し，待合せ情報に入力ポート 0 から受信済みであることを記録してメッセージを破棄する．次に，ノード 1 からのメッセージを受けた 1 段目のスイッチは，同様に Gather 待合せ機構において待合せの判定を行う．この場合，待合せ情報テーブルには入力ポート 0 からのメッ

図 3.6　並列計算機 Cenju-4

セージを受信済みであることが記録されており，受けたメッセージでこのスイッチでの待ち合わせが完了したと判断し，当該メッセージを出力する．また待合せ情報テーブルの値を未受信に初期化する．このように各スイッチにおいて待ち合わせを行い最後のメッセージのみを出力することで，ノード 0,1,2 からのメッセージが一つのメッセージに集約されてノード 3 に届く．

3.5　並列計算機 Cenju-4 での実装と評価

3.5.1　並列計算機 Cenju-4 の概要

Cenju-4 [33] は各ノードがメモリを持つ分散メモリシステムである．図 3.6 に Cenju-4 の外観図を，図 3.7 に Cenju-4 の構成概要を示す．1 ノードは MIPS 社のプロセッサ R10000 [34] と 1MByte の 2 次キャッシュ (128bit データ幅 ×133MHz の SSRAM)，512MB まで拡張可能なメモリ (64bit データ幅 ×80MHz の SDRAM)，PCI バス (SCSI, ethernet 等が接続)，それらをコントロールする制御チップ（80MHz で動作）で構成される．この制御チップは上記のコントロールの他，ネットワークとも接続し分散共有メモリ機構を実現している．図 3.6 に示す 1 ラックに 16 プロセッサを搭載可能で，128 プロセッサは 8 ラック，1024 プロセッサは 64 ラックで構成する．

ネットワークは，80MHz で動作する 4×4 のクロスバースイッチを 8 個用いた 16×16

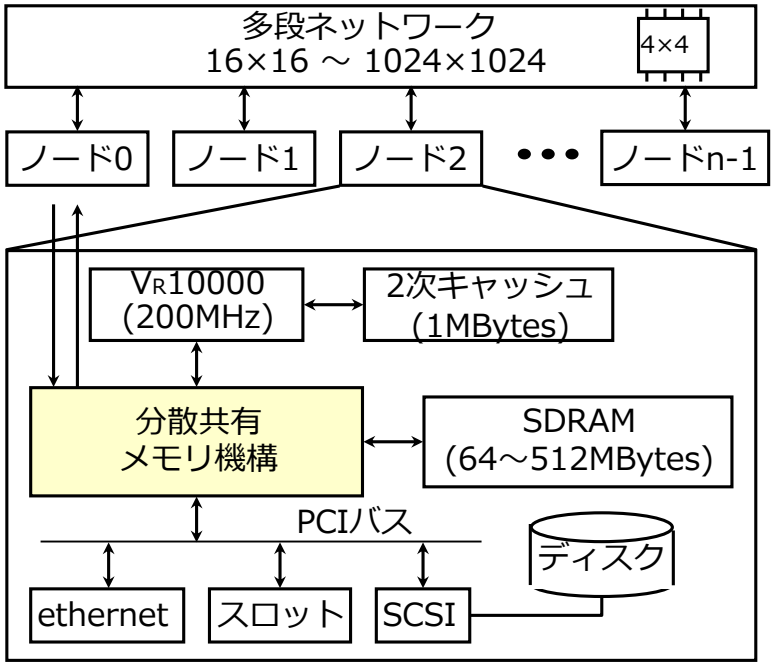


図 3.7　並列計算機 Cenju-4 の構成概要

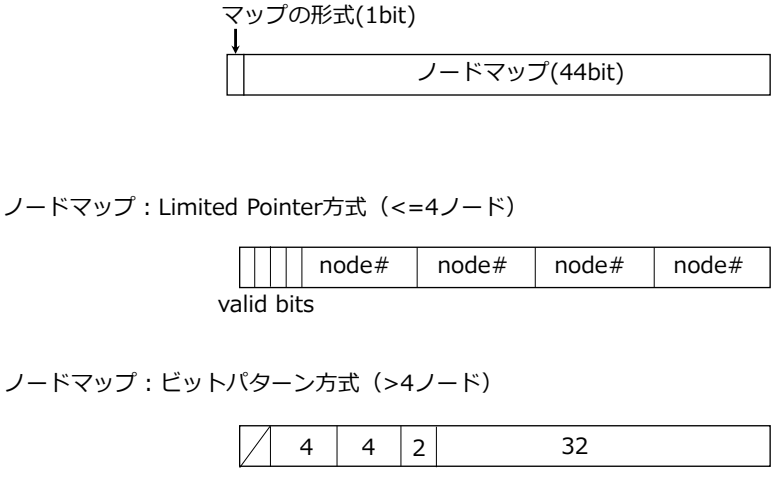


図 3.8　ディレクトリ構成

のネットワークボードを基本とする多段網で構成される．各ポートは 200MB/s のスループット（片方向）を持つ．このネットワークボードを複数接続することで，16 × 16 から 1024 × 1024 のネットワークを構成する．

3.5.2 ディレクトリの実装

Cenju-4 は，ハードウェアコスト削減のために，ディレクトリ専用のメモリを設けずに主メモリの一部をディレクトリ領域として利用している．これにより，主メモリとは別のメモリモジュールを必要とせず，ハードウェアコストを悪化させることなくディレクトリを保持することが可能となっている．また，システムを構成するノード数に関わらず，メモリブロック (128Byte) ごとに 64 ビットの一定サイズのディレクトリが用意され，メモリに占めるディレクトリのサイズは 1/16 となり，ディレクトリ保持コストの増加を防いでいる．また，一エントリを主メモリのビット幅と一致する 64 ビットとすることで，ディレクトリの読み出しは一メモリアクセスで済む設計とし，ディレクトリアクセスコストを一定に保っている．

図 3.8 にディレクトリの構成を示す．ディレクトリ情報は，当該メモリブロックのコピーを保持しているノードの数によりその形式が切り替わる．ノードの数が 4 以下の場合は Limited Pointer 形式となる．ポインタ形式では，ノード番号を保持する 4 つのフィールドと，各フィールドが有効かどうかを示す 4 ビットからなる．一方，ノードの数が 4 を超えた場合はビットパターン形式となる．このビットマップ形式は，総計 42 ビットで 4-4-2-32 ビットの 4 つのフィールドで構成される．

この構成を採用することにより，保持ノード数が 4 以下のケースではシステムの規模に依存せず厳密な管理が可能となり，32 ノードのシステム構成まではビットパターン形式によって厳密な管理が可能となる．32 ノードより大きいシステムにおいても，システムを分割して利用するケースにおいて，ノード番号の上位 5 ビットが同一の 32 ノードまでであれば厳密な管理が可能となる．

3.5.3 マルチキャストとギャザー機能の実装

Cenju-4 では，書込み要求の転送と応答の受信を効率化するために，本章で提案したマルチキャストおよびギャザー機能を実装している．対応するディレクトリ形式は，Limited Pointer 形式とビットパターン形式である．

ネットワークスイッチの設計において，マルチキャスト機能を実現するためには，2 つのスイッチが同じスイッチにマルチキャストメッセージを書き込む場合の調停で発生するデッドロックを回避する必要がある．Cenju-4 では，書き込み時に調停が不要なクロスポイントバッファ方式でスイッチを構成し，バーチャルカットスルー方式でフロー制御を行い，この問題を解決している．クロスポイントバッファ方式では入力と出力の全組合せ分のバッファがスイッチ内に用意される．Cenju-4 では，4 × 4 のスイッチであることか

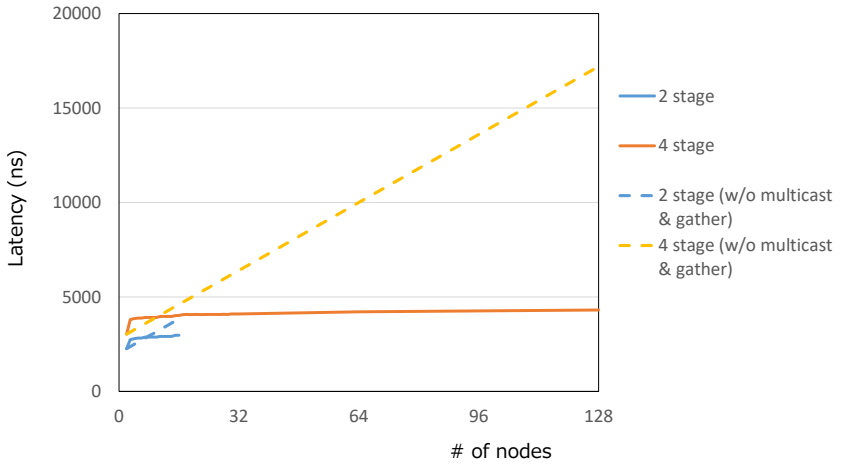


図 3.9　書込みアクセスのレイテンシ

ら，16 個のバッファがスイッチ内に用意される．各出力ポートにおいて入力ポート毎に異なるバッファを用意しているため，入力ポート間での調停が不要となっている．

Cenju-4 で採用したギャザー機能は，4 入力 4 出力のクロスバスイッチが 6 段接続される構成に対応した方式を実装している．ギャザーメッセージには，各段のスイッチに対応した 6 つの 4 ビットからギャザー情報が付加され，各スイッチではそのギャザー情報をもとに応答の待ち合わせを行う．また同時に実行可能な書込み要求処理数を 1024 とし，その数に対応する待合せ情報テーブルをスイッチに持つ．スイッチチップにこのテーブルが占める割合はゲート数で 3.6% であり，ハードウェアコストとして許容できるものとなっている．

3.5.4　書込みアクセスのレイテンシ評価

書込みアクセスのレイテンシを実機で測定した結果を図 3.9 に示す．複数のノードで共有している他ノードのメモリのデータに対して書き込みを行った場合のレイテンシであり，横軸が宛先ノード数を示している．

Cenju-4 は多段のネットワークで構成されており，その段数によりレイテンシが異なる．今回の評価では 2 段（〜16 ノード）の場合と 4 段（〜128 ノード）の場合のレイテンシを測定した．また，Cenju-4 の設計情報から，マルチキャスト・ギャザー機能を用いなかった場合のレイテンシを推定した結果も示している．

共有しているノード数が与える影響を見ると，マルチキャスト・ギャザー機能を用いない場合のレイテンシが宛先ノード数に比例して増加するのに対して，マルチキャスト・ギャザー機能を利用することによりノード数が与える影響を非常に小さいものとし，ほぼ

一定のレイテンシに出来ていることが分かる．レイテンシに若干の増加が見て取れるが，これはネットワーク内での待ち合わせを行うことにより，待ち合わせが不要なケースと比較して遅延が増加する分となっている．この結果より，マルチキャストおよびギャザー機能を利用することが，書込みアクセスのレイテンシをスケーラブルなものとするための有効な手段であることが実機でも確認できた．

一方，共有ノードが 2 か 3 以上かによりレイテンシが大きく変化する．共有ノードが 2 の場合は無効化要求を伝える相手が 1 ノードなのでマルチキャストおよびギャザーが必要ないのに対し，共有ノードが 3 以上の場合は相手が複数ノードとなりマルチキャストおよびギャザーが用いられるためである．この結果から，メモリアクセスレイテンシを最適化する観点では，共有しているノード数によってマルチキャストおよびギャザー機能を利用するかしないかを切り替えることも設計の選択肢になりえることが分かる．

またメモリアクセスレイテンシに占めるネットワークの通信遅延の割合は，2 段の場合と 4 段の場合でそれぞれ 50% と 65% となり大きな割合を占める．そのため，ネットワークの通信レイテレイテンシを削減することが重要であることも確認できた．

3.6 結論

分散共有メモリシステムにおいては，メモリアクセスレイテンシがその性能に大きな影響を与えるため，特にシステム規模の増大によらずそのレイテンシを一定に抑えることが重要となる．本章では，2 章の評価で明らかとなった，書込みに起因するメモリアクセスレイテンシが平均ライト分配数の増加によって悪化する問題を解消することを目的として，ビットパターン方式のディレクトによって平均ライト分配数が実際に保持しているノードよりも大幅に増加するのを抑制し，ネットワークのマルチキャストとギャザーを活用して書込み要求の送信と応答の受信を効率化する汎用的な方式を提案した．

ビットパターン方式は，ノードを階層的に分割して管理し各階層による分割サイズが異なることを特徴とする．評価の結果から，ビットパターン方式のこの特徴が有効に機能し，様々な利用シーンにおいて保持ノードを効率よく管理することができる方式であることが確認できた．システムを構成する全てのノードを利用するアプリケーションのケースでは，階層的に分割しない Coarse Vector 方式が最も優れているものの，提案するビットパターン方式が次いで保持している可能性のあるノード数が大幅に増加するのを抑制できることを確認した．一方，設計目標とした，大規模システムの一部のノードを利用してアプリケーションを動作させるケースでは，階層的な分割が有効に機能し，保持している可能性のあるノード数が大幅に増加するのを Coarse Vector 方式よりも抑制できることも確認した．

提案したマルチキャスト・ギャザー方式は，従来提案されていた手法の以下の二つの制

約を解消することを目的として設計した．一つ目の制約は，従来のマルチキャスト方式はネットワーク構造に合わせたマルチキャスト宛先指定によって実現され，その宛先指定に合わせたディレクトリ方式を採用する必要があった点である．この制約によって，保持している可能性のあるノード数が大幅に増加し一貫性制御のオーバーヘッドが増加してメモリアクセスレイテンシが悪化する．また，二つ目の制約は，適用可能なネットワークトポロジに制約がある点である．提案したマルチキャスト方式は，ネットワークの各スイッチがディレクトリ方式から宛先を算出してマルチキャストを行うことを特徴とする．これにより，ネットワーク構造に合わせることなく効率的なディレクトリ方式を採用することが可能となる．また提案したギャザー方式は，送出先ノードがマルチキャストの宛先情報からネットワーク内での待ち合わせパターンを算出し，そのパターンに応じて各スイッチがメッセージを集約することを特徴とする．この特徴により，様々なネットワークトポロジに適用可能となる．

本機構を実装した並列計算機 Cenju-4 でメモリアクセスレイテンシを評価した結果，大規模な分散共有メモリシステムにおいても書込みに起因するメモリアクセスレイテンシを，ノード数にかかわらず一定の値に抑える効果を持つことが確認された．また、このレイテンシの評価から，ネットワークの性能向上がメモリレイテンシを抑えるために重要であることも確認された．これについては 4 章で議論する。

第 4 章

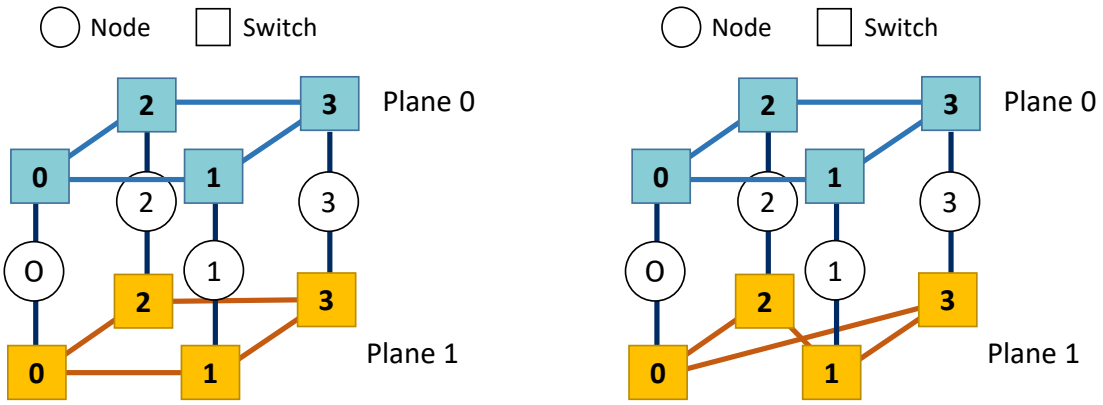
二重同型化によるネットワークの高性能化

4.1 はじめに

本章では，高性能かつ安定動作するディレクトリ方式の一貫性制御の実現における四つの課題のうち，三つ目のメモリアクセスレイテンシの削減を目的としたネットワークの低レイテンシ化と高スループット化について述べる．

分散共有メモリシステムの性能向上において，メモリアクセスレイテンシの削減は非常に重要な問題である．3 章における評価で，メモリアクセスレイテンシに占めるネットワークの通信遅延の割合は 50% から 65% になることが分かっている．システムの大規模化に従ってネットワークのレイテンシは増加しており，そのレイテンシを抑えることが重要となる．

近年，ネットワークの性能を向上させる方式として，1 ノードが複数のネットワーク・ポートを持つ Multi-Rail 方式を採用するシステムが増加している [68–70]．この Multi-Rail 方式には，二つの方式が存在した．一つは，ネットワークが一つのプレーンからなり，ノードが持つ複数のポートを一つのネットワーク・プレーンに接続してスループットを向上させる Port-Aggregate 方式である [69, 70]．この方式は，スループット向上が図れるものの，ネットワーク・プレーンのサイズがポート数にノード数をかけた大きさが必要となりトポロジや接続するノード数によっては平均最短距離が悪化する問題があった．もう一つは，ネットワークが複数のプレーンからなり，各ポートをそれぞれ異なるプレーンに接続する Multi-plane 方式である [68]．この方式では，各プレーンは同一であり，均等にプレーンを利用すればスループットをプレーン数倍にすることができる．また，どのプレーンを利用してもノード間の距離は同一であるため，平均最短距離は単一プレーンと同じとなる．



(a) 二重 2 次元 Hypcube ネットワーク　(b) 二重同型 2 次元 Hypcube ネットワーク

図 4.1　二重 Hypercube ネットワークと二重同型 Hypcube ネットワーク

本章では，この Multi-plane 方式においてネットワークをさらに高性能化する二重同型ネットワークを提案する [88]．二重同型ネットワークは，各プレーンに同一ではなく同型のネットワークを採用することで，遅延を削減しスループットを向上させるものである．ここで同型とは，グラフ同型のトポロジでありスイッチ間接続の組み合わせが異なるものである．図 4.1 に従来の Multi-plane 方式での二重 2 次元 Hypercube ネットワークと，提案している二重同型 2 次元 Hypercube ネットワークを例として示す．図に示すように，二重 2 次元 Hypercube ネットワークでは．プレーン 0 とプレーン 1 ともに，ノードとスイッチ間．およびスイッチのスイッチ間の接続が同一の構成を取る．一方，二重同型 2 次元 Hypercube ネットワークでは，プレーン 0 とプレーン 1 ともに同じ Hypercube トポロジではあるが，異なるスイッチ間の接続を取る．同一のネットワークを二重化する従来の Multi-plane 方式と比較して，このような二重同型ネットワークを採用することで，あるノードへの距離がそれぞれのプレーンで異なるものとなるため，パケットを送出する際に距離が短い方のプレーンを選択することで，遅延を改善することが期待できる．また，そのような選択を行うことで，パケットを送信する場合に利用するスイッチ間リンク数を削減することができるため，スループットの向上も同時に期待できる．

本章では，Hypercube ネットワーク [54] と Folded-Hypercube ネットワーク [55] の二つの良く知られたトポロジに対して，提案する二重同型ネットワークを適用し議論する．Hypercube ネットワークは HPC システムにおいて重要なトポロジの一つであり，現在の HPC システムでも広く用いられており [68, 71]，また分散共有メモリシステム SGI Origin2000 [38] でも採用されている．また HPC システムでの標準的なインターコネクトである InfiniBand [66] でもサポートされている．

本章の構成は以下である．4.2 節において二重同型 Hypercube ネットワークについてグラフ解析を行い，平均最短距離やスループットについて評価した結果を示す．また，4.3

節において実システムに適用した場合の評価として，ケーブル遅延を考慮した二重同型 Hypercube ネットワークの最適化方式を提案し，ネットワークの経済的コストの解析，サイクルレベルシミュレーションによる性能評価を行う．最後に 4.4 節において結論を述べる．

4.2 二重同型 Hypercube ネットワークのグラフ解析

この節では，二重 Hypercube(D-HC)，二重同型 Hypercube(DI-HC)，二重 Folded-Hypercube(D-FHC)，二重同型 Folded-Hypercube(DI-FHC) の 4 つのネットワークを比較する．まずは同型ネットワークの生成方法と二重同型ネットワークでのルーティングについて述べる．次に，多数の選択肢が存在する同型ネットワークの中から最適なものを選択するために二つの性能指標を導入する．一つは，平均最短距離であり遅延を表すものとして用いる．もう一つはスループットを表す指標として全対全最大トラフィックを新たに定義しそれを用いる．最後に，様々なネットワーク・サイズでの解析結果を示す．なお，本節では，簡単のためにスイッチ当たりのノード数を 1 として検討を行った．

4.2.1 二重同型 Hypercube および二重同型 Folded-Hypercube の生成

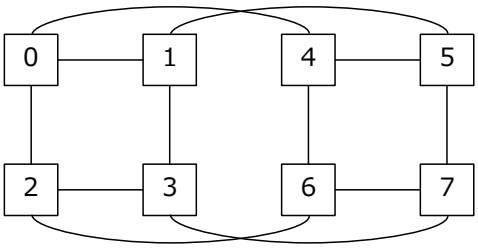
まず，DI-HC の生成について述べ，次に DI-FHC の生成について述べる．二重同型では，両プレーンにおいて異なるスイッチ間接続をとることを特徴とする．ここで，各ノードには 0〜$2^n - 1$ の ID が割り当てられ，両プレーンのスイッチにも 0〜$2^n - 1$ の ID を割り当てられる．ノードと接続する両プレーンのスイッチの関係はどの同型においても同一で，同じ ID を持つ者同士が接続するものとする．DI-HC のプレーン 0 には一般的な Hypercube 接続を用い，スイッチ ID のハミング距離が 1 のスイッチ同士を接続することによって構成する．一方プレーン 1 には，前記プレーン 0 とは異なるスイッチ同士を接続して同型 Hypercube を構築する．

ここで，Hypercube のスイッチ間の接続関係に着目して，それぞれのプレーンの接続を表す $H(0,n)$ と $H(1,n)$ を定義する．プレーン 0 の n-Hypercube の接続は集合 $H(0,n)$ は以下の式で定義できる．

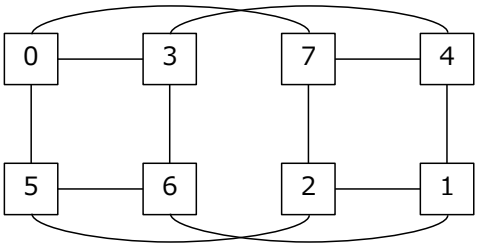
$$H(0,n) = \{1, 2, \ldots, 2^{n-1}\}$$

各スイッチは自 ID と $H(0,n)$ の各要素とビットごとの排他的論理和を取って得られる ID を持つ n 個のスイッチと接続する．即ち，ID が x のスイッチは，以下の式で求められるスイッチ集合 Y と接続する．

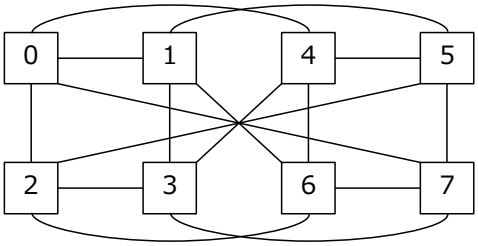
$$Y = \{x \oplus h \,|\, h \in H(0,n)\}$$



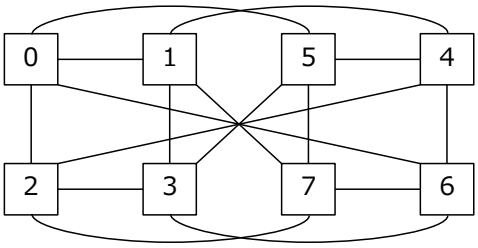
(a) 3-Hypercube $H(0,3) = \{1,2,4\}$



(b) Isomorphic 3-Hypercube $H(1,3) = \{3,5,7\}$



(c) 3-Folded-Hypercube $FH(0,3) = \{1,2,4,7\}$



(d) Isomorphic 3-Folded-Hypercube $FH(1,3) = \{1,2,5,6\}$

図 4.2　二重同型 Hypercube(DI-HC) および Folded-Hypercube(DI-FHC) の接続例 (n=3)

例えば，図 4.2(a) に示す 3 次元 Hypercube は $H(0,3) = \{1,2,4\}$ で定義され，ID が 1 のスイッチは，ID が $\{0,3,5\}$ のスイッチと接続する．$H(1,n)$ も要素数 n の集合で，各要素は 1 以上 2^n 未満の自然数となる．この $H(1,n)$ を用いて，プレーン 1 の ID が x のスイッチは，以下の式で求められるスイッチ集合 Y と接続する．

$$Y = \{x \oplus h \,|\, h \in H(1,n)\}$$

ここで，構成される Network が一つの連結グラフとなる $H(1,n)$ のみが同型 Hypercube となる．例えば，図 4.2(b) に同型 3 次元 Hypercube で $H(1,3) = \{3,5,7\}$ の例を示す．

表 4.1 ルーティング用のテーブル (図 4.2 (a)(b) に示す二重同型 3 次元 Hypercube の例)

Src.ID ⊕ Dst.ID	Dist.@ Plane 0	Dist.@ Plane 1	Routing Value @ Plane 1
0	0	0	0
1	1	3	7
2	1	2	6
3	2	1	1
4	1	2	5
5	2	1	2
6	2	2	3
7	3	1	4

この同型 Hypercube では例えば ID が 1 のスイッチは ID が $\{2, 4, 6\}$ のスイッチと接続する．このように $H(0, n)$ と $H(1, n)$ で定義されるグラフ同型の Hypercube を利用することで，DI-HC を構成する．

DI-FHC も同様に構成できる．Folded-Hypercube は，Hypercube の各頂点に対して最も遠い距離の頂点と接続するリンクを追加するものである．Hypercube と同様にプレーン 0 の Folded-Hypercube は頂点間の接続関係から以下の $FH(0, n) = \{1, 2, \ldots, 2^{n-1}, 2^n - 1\}$ で定義できる．またこの Folded-Hypercube において，ID が x のスイッチは，以下の式で求められるスイッチ集合 Y と接続する．

$$Y = \{x \oplus h \,|\, h \in FH(0, n)\}$$

図 4.2(c) に 3 次元 Folded-Hypercube の接続を示す．この例では，$FH(0, 3) = \{1, 2, 4, 7\}$ であり，ID が 1 のスイッチは ID が $\{0, 3, 5, 6\}$ のスイッチと接続する．DI-FHC の $FH(1, n)$ も，最終要素を除いた n 個の要素について DI-HC の構築と同様の規則で生成する．Folded で追加されるリンクについては，求めた n 個の値のビットごとの排他的論理和で求められる値となる．図 4.2(d) に 3 次元同型 Folded-Hypercube の接続を示す．この例では，$FH(0, 3) = \{1, 2, 5, 6\}$ であり，ID が 1 のスイッチは ID が $\{0, 3, 4, 7\}$ のスイッチと接続する．

ここで，$H(1, n)$ や $FH(1, n)$ には多数の選択肢が存在するが，以降で述べる評価指標を用いて効果の大きい集合を選択する．

4.2.2 ルーティング

DI-HC および DI-FHC の最短パスルーティングは，以下の二つのステップからなる．

1. ノードは距離が短いプレーンを選択し，そのプレーンにパケットを送出する．同一の場合はどちらかを選択する
2. 選択したプレーン内で各スイッチは最短パスルーティングを行う

ここで，各ノードは表 4.1 に示すルーティング用のテーブルを保有する．表はノード数分のエントリからなり，ソースとあて先ノード ID のビットごとの排他的論理和を取って得られる値をインデックスとして，プレーン 0 での距離，プレーン 1 での距離，プレーン 1 でのルーティング用の値を保持する．以降表 4.1 に示した DI-HC を例に説明するが，DI-FHC においても同様の考え方でルーティングが可能である．

ステップ (1) においては，表 4.1 を用いて，あて先ノードへの距離をそれぞれのプレーンについて読出し，値を比較することによって決定する．ステップ (2) におけるプレーン 0 のルーティングは，既存の次元オーダーの最短パスルーティングが利用可能である．次元オーダーのルーティングでは，ソースとあて先ノード ID のビットごとの排他的論理和を取って得られる値を用い，下位から 1 が立っているビットを順に選択し対応するリンクを辿ることで実現される．一方プレーン 1 では，前節で示したようにプレーン 0 とは異なる接続となるため上記のプレーン 0 と同じ方法でルーティングすることは出来ない．そのため，両プレーンがグラフ同型であることを利用し，ソースとあて先ノード ID のビットごとの排他的論理和を取って得られる値ではなく，それを変換した値を用いることでプレーン 0 と同様の次元オーダーのルーティングを可能となる．この変換は，プレーン 1 でのあるスイッチ ID が，プレーン 0 のどのスイッチ ID と同じ位置にあるかを求めるものである．これにより，プレーン 1 でのあるスイッチ ID へのルーティングパスが，プレーン 0 ではどのスイッチ ID へのルーティングパスと同一であるかを求めることができ，次元オーダーの最短パスルーティングが実現できる．表 4.1 のプレーン 1 でのルーティング用の値は，図 4.2(b) に示した 3 次元同型 Hypercube の例での変換となる．プレーン 1 におけるスイッチ ID 0,1,2,3,4,5,6,7 は，それぞれプレーン 0 におけるスイッチ ID 0,7,6,1,5,2,3,4 と同じ位置にあるため，前記プレーン 0 におけるスイッチ ID の値が表に保持される．

上記より，以下の様にノードからノードへルーティングが行われる．ノード 0 からノード 6 へのルーティングは，ステップ (1) で表 4.1 よりプレーン 0 とプレーン 1 の距離が同じなため，どちらかのプレーンが選択される．プレーン 0 が選択された場合，ステップ (2) において次元オーダーの最短パスルーティングが行われ，下位から 2 および 3 ビット目が 1 であることから $H(0,3) = \{1,2,4\}$ の 2,3 番目の接続が順に利用され，スイッチ ID 0-2-6 とルーティングされる．プレーン 1 が選択された場合，ステップ (2) において表 4.1 より 6 から 3 に変換され，下位から 1 および 2 ビット目が 1 であることから $H(1,3) = \{3,5,7\}$ の 1,2 番目の接続が順に利用され，スイッチ ID 0-3-6 とルーティング

される．

表 4.1 において，プレーン 0 での距離やプレーン 1 での距離は，それぞれ src⊕dst の値および表 4.1 の Routing Value@plane 1 から popcount により算出可能であり，テーブルで保有するのではなくルーティング時に算出しても良い．また，各パケットは 1 つのプレーンのみを用いて転送され，プレーン内では既存の次元オーダーの最短経路を用いる．そのため本ルーティングはデッドロックフリーかつライブロックフリーであることは自明である．なお，1 つのパケットを他ノードを経由して 2 つのプレーンを用いて転送することで，単一のプレーン内でルーティングするよりも距離を短くできる可能性はあるが，本章では扱わないものとする．

4.2.3 評価指標

二重同型ネットワークの評価指標として，遅延に相当する平均最短距離に加えて，スループットに相当する全対全最大トラフィックを導入する．この全対全最大トラフィックは，各スイッチに 1 ノードが無限のスループットを持ったリンクで接続して Uniform Random 通信を行った時に，スイッチ間リンクのスループットを 1 として得られるノード当たりの最大スループットを求めるものである．以下に，その算出方法を示す．

1. 全ノードが全ノードに対してそれぞれ 2(=プレーン数) パケットを送付する．
2. 前節のルーティングのステップ (1) において，距離が短いプレーンがあれば，両パケットをそのプレーンに流す．一方，距離が同一の場合はそれぞれのプレーンに 1 パケットずつ流す．
3. 選択したプレーンで固定の最短パスルーティングを行う
4. 各スイッチのリンクにおいて通過するパケット数をカウントする (リンクは有向グラフとして方向別にカウントする)
5. ノードとスイッチ間のリンクを除く，全てのスイッチ間リンクにおいて最大の通過パケット数を求める
6. 1 ノードが送出するパケット数（ノード数× 2）を上記最大値で割った値を全対全最大トラフィックと定義する．

例えば，単一の Hypercube はこの値が 2 となり，D-HC では 2 倍の 4 となる．この指標がシミュレーション結果と一致するかどうかは 4.3.5 節で検証される．

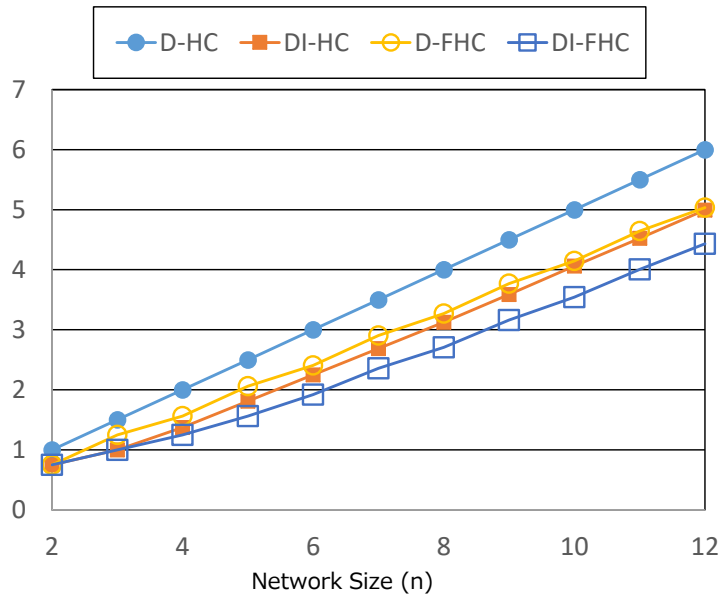
表 4.2　二重同型 Hypercube(DI-HC) および Folded-Hypercube(DI-FHC) の最適な Plane 1 接続

n	DI-HC Plane 1 $H(1,n)$	DI-FHC Plane 1 $FH(1,n)$
2	1,3	1,2,3
3	3,5,7	1,2,5,6
4	3,7,12,13	3,5,7,10,11
5	1,3,14,22,29	3,5,7,14,22,25
6	1,3,14,28,50,53	3,13,21,26,30,38,57
7	1,3,5,14,56,88,118	7,11,28,44,76,85,90,127
8	7,11,28,44,112,176,193, 194	1,14,112,146,164,187,201, 220,231
9	7,11,49,194,274,285,355, 419,463	1,15,113,183,219,237,315, 349,359,511
10	7,11,28,44,112,176,256, 719,853,854	1,14,112,146,149,225,263, 297,328,602,933
11	7,11,28,44,112,176,448, 469,704,1748,1751	147,196,461,669,903,1020, 1092,1413,1444,1559,1765, 2027
12	7,11,28,44,112,176,448, 704,717,1216,3341,3534	67,177,388,489,555,1365, 1602,1719,2020,2161,2498, 3162,3353

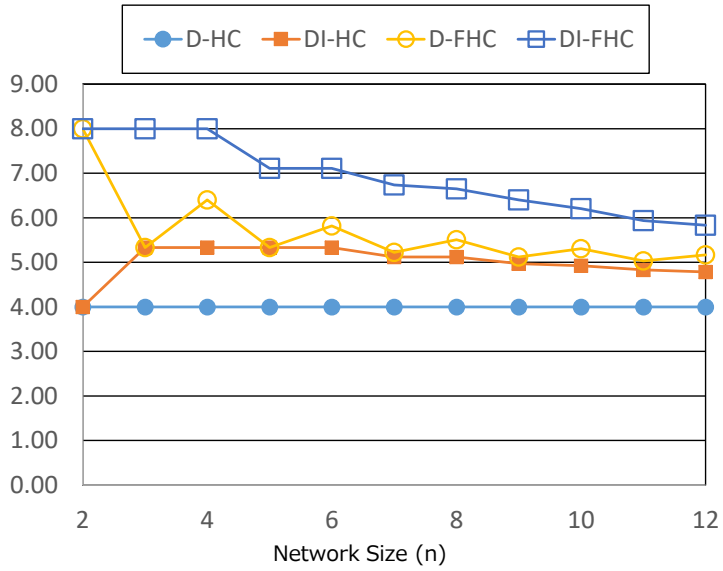
4.2.4　解析結果

表 4.2 に，ネットワークサイズとして n が 2 から 12 までについて，多数の選択肢の中から全対全最大トラフィックが最大で，平均最短距離が最小の値をとる $H(1,n)$ と $FH(1,n)$ を求めた結果を示す．なお，n が 8 以下については，全解探索によって最適な値を求め，9 以上については部分探索結果による準最適解となる．また，図 4.3 に，D-HC, DI-HC, D-FHC, DI-FHC の 4 つの構成について，平均最短距離と全対全最大トラフィックを求めた結果を示す．

図より，二重同型化によって平均最短距離を削減し，全対全最大トラフィックを向上させる効果があることが確認できる．$n = 8$ の時，平均最短距離は二重同型 Hypercube(DI-



(a) 平均最短距離



(b) 全対全最大トラフィック

図 4.3　二重同型 Hypercube(DI-HC) および Folded-Hypercube(DI-FHC) のグラフ解析結果

HC) で 22%(D-HC で 4, DI-HC で 3.13), 二重同型 Folded-Hypercube(D-FHC) で 17%(D-FHC で 3.27, DI-FHC で 2.71) 削減されている．平均最短距離の削減量はほぼ一定であり，n の増加にしたって削減率は徐々に減少している．$n = 12$ の場合，DI-HC で 17%, DI-FHC で 12% の削減となっている．全対全最大トラフィックも $n = 8$ の時に，DI-HC で 28%(D-HC で 4, DI-HC で 5.12), DI-FHC で 21%(D-FHC で 5.51, DI-FHC で 6.65) 改善している．こちらも平均最短距離同様に n が大きくなるにつれて向上率は減

少し，$n = 12$ の時に DI-HC で 20%，DI-FHC で 13% の改善となっている．

このように，二重同型化が Hypercube においても Folded-Hypercube において，平均最短距離および全対全最大トラフィックを改善する効果を持つことがグラフ解析結果から確認できる．

4.3 実システムを想定した二重同型 Hypercube ネットワークの最適化

実システム，特に大規模なシステムでは，ネットワークの遅延としてスイッチの遅延とケーブルの遅延の両方を考慮することが重要になってくると考えられる．4.2 節で同型ネットワークの評価指標として用いた平均最短距離は，これら二つのうちスイッチ遅延のみを反映したものであり，ケーブルの遅延は考慮していないものとなる．そこで，この節では平均最短距離に替わる新たな指標として，ケーブル遅延も考慮した平均最短遅延を導入し，その指標をもとに最適な二重同型ネットワークの接続を求める．

本節では，まず想定するシステム構成について示したあと，平均最短遅延を指標とした最適な二重同型ネットワークのグラフ解析結果を示す．次に，ネットワークの経済的コストを試算し，最後にサイクルレベルのシミュレータによるネットワーク性能評価結果を示す．

4.3.1 実システムの想定

システム構成

HPC システムで標準的に用いられている 19 インチラックを複数利用し，それらがなるべく正方形に近い形で並べられるシステム構成を想定する．1 ラックのサイズは幅 80cm，奥行き 150cm，高さ 200cm からなるとし，幅方向には隣接して並べ，奥行方向には保守スペースとして 100cm のスペースを取り並べることとした．また，正方形に近い形となるラック配置として，幅および奥行のラックの比率を 2:1 あるいは 4:1 のどちらかを選択して並べた．

1 ラックには 16 ノードとそれらのノードが接続するスイッチを収容するものとした．また，スイッチ当たりに接続するノード数は 4 とした．この 4 という数値は，ノードとスイッチ間．およびスイッチとスイッチ間には全て同じ Link 速度で接続する想定で，ノード-スイッチ間リンクではなく，スイッチ間のリンクがボトルネックとなるように選択した値である．なお，二重同型ネットワークを構成するうえで，接続を変更するのはスイッチ間のケーブルのみとし，ノードとスイッチ間の接続についてはどちらのプレーンも同一とした．

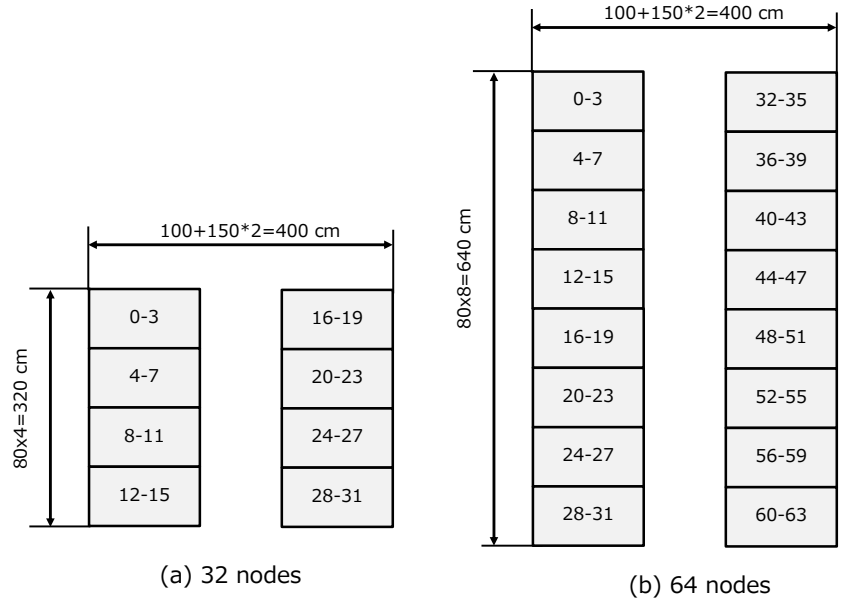


図 4.4 ラック配置

表 4.3 InfiniBand EDR ケーブルの種類

Cable types	Length (m)
Copper	2.0, 2.5, 3.0, 4.0, 5.0
Optical	10, 15, 20, 30, 50, 100

図 4.4 に 128 ノードおよび 256 ノード時のラック配置例を示す．128 ノード時は，8 ラックを幅方向に 4，奥行方向に 2 の 2:1 の比率で並べる構成となる．256 ノード時は，16 ラックを幅方向に 8，奥行方向に 2 の 4:1 の比率で並べる構成となる．

ケーブル長

市販されているケーブルを利用して配線を行うことを想定してケーブル長を算出した．現在 Mellanox 社が販売している InfiniBand EDR 用のケーブルの種類を表 4.3 に示す [72]．InfiniBand のケーブルには銅ケーブルと光ケーブルが存在し，銅ケーブルは 2m から 5m，光ケーブルは 10m から 100m が提供されている．

ラック内およびラック間の配線長は以下の様にして求めた．ラック内の，ノードとスイッチ間の配線長はノードの位置によって異なり，最短でほぼ 0，最長でラック高の 200cm となる．ここでは平均を取り 100cm とし，配線猶予の 100cm を加えた 200cm とした．ラック内の，スイッチとスイッチ間の配線長は，スイッチが隣接していることから，100cm とした．最後に，ラック間配線長は，マンハッタン距離でカウントし，200cm のケーブル余裕を加えることとした [57]．上記により算出した配線長に対して，表 4.3 に示されているケーブル候補から満たすものを選択し，今回試算するケーブル長とした．

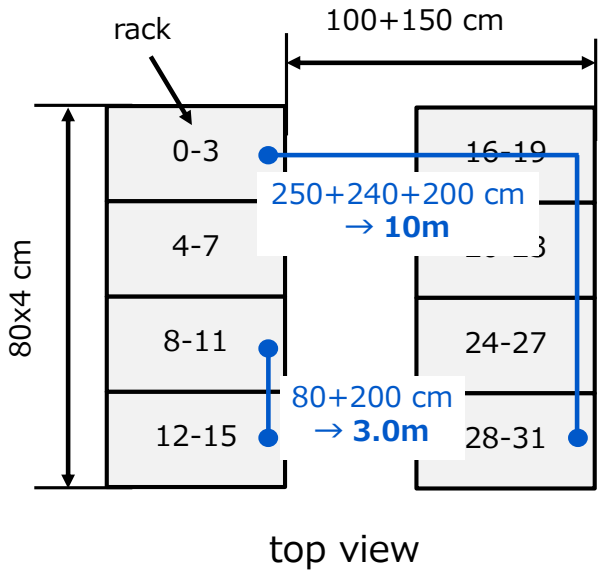


図 4.5　ケーブル配線長の計算

図 4.5 に 8 ラック構成時 (n=5) のラック間配線の配線長と選択されたケーブル長の例を示す．8 ラック構成時は幅方向に 4 ラック，奥行方向に 2 ラック並べる構成となる．ラック内の数値はそのラックに配置されるスイッチの ID を示す．例えば，スイッチ ID8 と 12 を接続する場合，幅方向で隣接するラック間の配線となり，280cm が配線長となる．この配線長を満たすケーブルは 3.0m のものとなるためそれがケーブル長となる．また，スイッチ ID0 と 31 を接続する場合，最も遠いラック間の接続となり，690cm が配線長でケーブル長は 10m となる．

4.3.2　実システムを想定した同型ネットワークの選択

遅延指標

実システムを反映した遅延指標として新たに下記の式で表される平均最短遅延を導入する．下記の平均最短遅延は，スイッチ遅延とケーブル遅延の両方を考慮したモデルであり，Uniform Random 通信を行った時の平均遅延の最小値を表すものである．

$$\text{平均最短遅延} = (\text{平均ケーブル長} \times \text{ケーブル遅延} + \text{スイッチ遅延}) \times \text{平均最短距離} + \alpha$$

このモデルでは，スイッチ間を 1 ホップするのに要する遅延を平均ケーブル長とスイッチの遅延から計算し，それに平均最短距離をかけることでスイッチ間の伝送遅延を求めている．α は定数分で，ノードからスイッチ，スイッチからノードに要する時間などに相当する．本節では後述するシミュレーション評価と同一のパラメータとして，スイッチ遅延

表 4.4 二重同型 Hypercube(DI-HC-r) および Folded-Hypercube(DI-FHC-r) の最適な Plane 1 接続

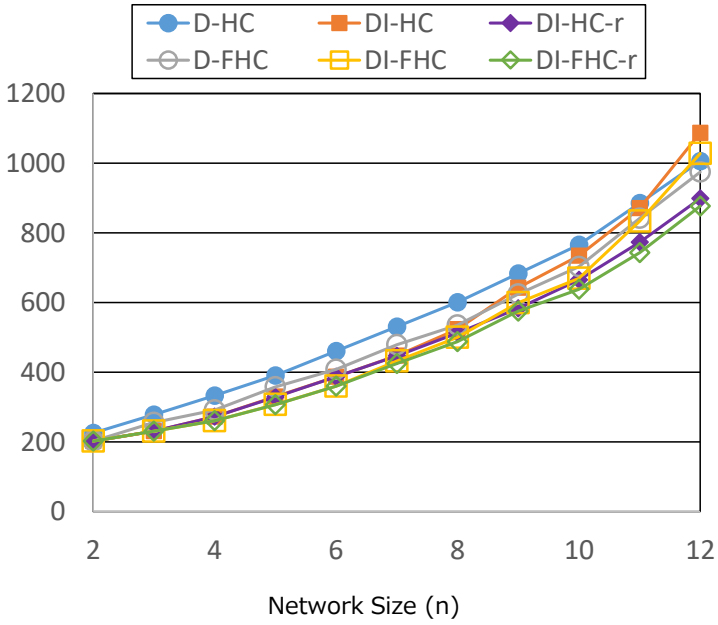
n	DI-HC-r Plane 1 $H(1,n)$	DI-FHC-r Plane 1 $FH(1,n)$
2	1,3	1,2,3
3	3,5,7	1,2,5,6
4	3,7,12,13	3,5,7,10,11
5	*3,13,15,19,20*	*7,9,10,11,19,29*
6	*1,2,15,28,51,52*	3,13,21,26,30,38,57
7	*7,9,11,28,53,97,98*	*1,14,22,38,59,61,90,99*
8	*3,5,7,24,63,105,193,209*	*1,14,50,88,119,159,164,195,234*
9	*1,2,5,28,44,57,194,322,449*	*1,2,4,57,79,143,208,246,280,448*
10	*1,2,4,15,56,88,119,385,641,899*	*1,3,9,14,122,128,256,436,468,543,896,*
11	*2,3,6,12,56,88,116,129,896,1408,1793*	*1,2,5,15,21,35,67,152,493,896,1411,1802*
12	*1,2,7,12,28,122,208,235,256,1792,2816,3595*	*2,3,5,30,42,56,118,182 968,1281,1792,2561,2816*

を 90ns，ケーブル遅延を 5ns/m，α は 131ns とした．なお，この指標がシミュレーション結果と一致するかどうかは 4.3.5 節で検証される．

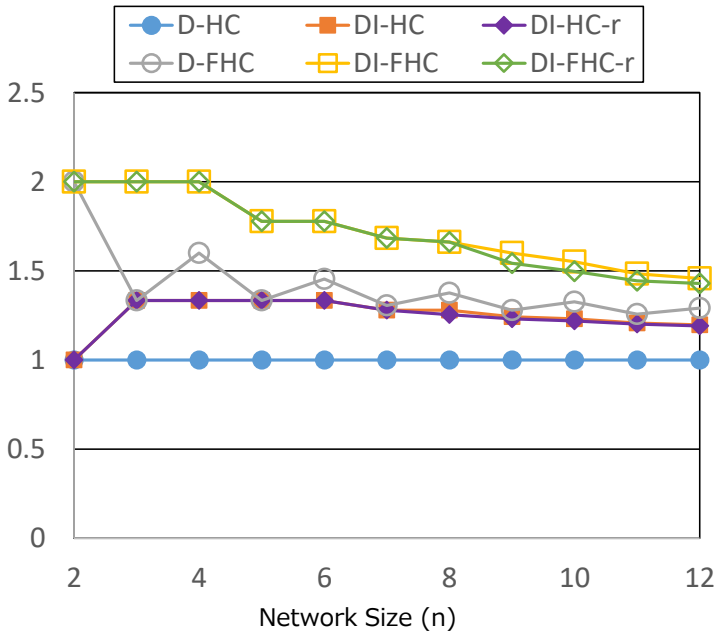
二重同型ネットワークの効果

平均最短距離を指標としたものと区別するために，平均最短遅延を指標とした二重同型ネットワークをそれぞれ DI-HC-r，DI-FHC-r と記す．

表 4.4 に多数の選択肢の中から全対全最大トラフィックが最大で，新たに導入した指標である平均最短遅延が最小の値をとる $H(1,n)$ と $FH(1,n)$ を求めた結果を示す．なお，n が 8 以下については全解探索によって最適な値を求めたものであり，9 以上については部分探索結果による準最適解である．表中で斜体で示したものは，平均最短距離を指標として選定された表 4.2 の接続と異なるものを示している．この結果より，Network サイズが大きくなると異なる選択を取っていることが分かる．



(a) 平均最短遅延 (ns)



(b) 全対全最大トラフィック

図 4.6　実システムを想定した二重同型 Hypercube の解析結果

また，図 4.6(a)(b) に，D-HC，DI-HC，DI-HC-r，D-FHC，DI-FHC，DI-FHC-r の 6 つのネットワークについて，平均最短遅延，全対全最大トラフィックを求めた結果を示す．全対全トラフィックについては，図 4.3(b) に示した値の 1/4 になっている．これは，図 4.3(b) においては 1 スイッチに接続するノード数を 1 としていたが，この評価では 1

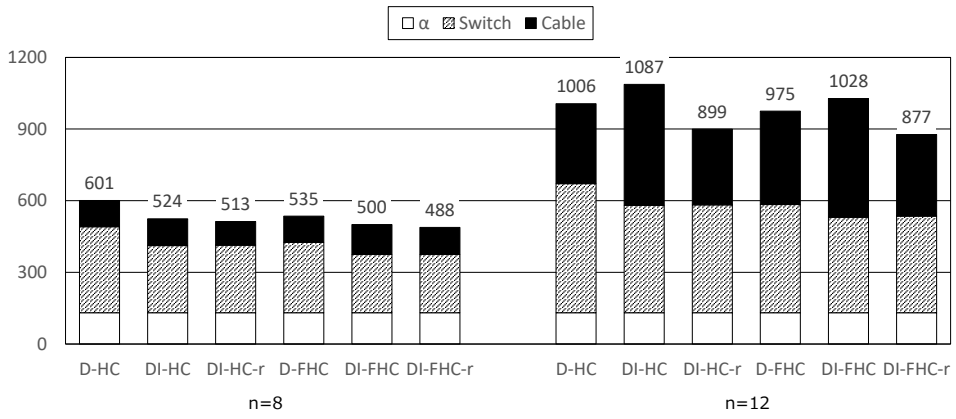


図 4.7　平均最短遅延 (ns) の内訳

スイッチに接続するノード数を 4 としているためである．

図より，$n \leqq 8$ においては，平均最短距離で最適化した DI-HC，DI-FHC，および平均最短遅延で最適化した DI-HC-r，DI-FHC-r ともにほぼ同様の傾向を示していることが確認できる．n=8 の場合に，DI-HC，DI-HC-r の平均最短遅延は，D-HC の 601ns からそれぞれ 524ns(13%) と 513ns(15%) に削減できており，DI-FHC，DI-FHC-r も，D-FHC の 535ns からそれぞれ 500ns(6.6%) と 488ns(8.9%) に削減できている．全対全最大トラフィックも，D-HC の 1.0 から DI-HC で 1.28(28%)，DI-HC-r で 1.25(25%)，D-FHC の 1.38 から DI-FHC，DI-FHC-r とも 1.66(21%) に向上している．

また，$n > 8$ の場合には違う傾向を示していることも確認できる．平均最短遅延において，平均最短距離で最適化した DI-HC，DI-FHC とも n が増加するにしたがって改善効果が減少し，n=12 においては D-HC や D-FHC よりも悪化する結果となっている．一方，平均最短遅延で最適化した DI-HC-r と DI-FHC-r は，悪化することなく改善を維持できていることが分かる．n=12 の場合に，平均最短遅延は D-HC の 1006ns から，DI-HC は 1087ns と 8.1% の悪化となっているが，DI-HC-r は 899ns と 11% の改善となっている．D-FHC の場合も，975ns から，DI-FHC は 1028ns と 5.5% の悪化となっているが，DI-FHC-r は 877ns と 10% の改善となっている．全体全トラフィックについては，$n > 8$ の場合も両者で大きな差は生じていない．n=12 の場合に，D-HC の 1.0 から DI-HC で 1.20(20%)，DI-HC-r で 1.19(19%)，D-FHC の 1.29 から DI-FHC で 1.46(13%)，DI-FHC-r で 1.43(11%) に向上している．

平均最短遅延の傾向の違いを確認するために，その内訳を図 4.7 に示す．図は，4.3.2 節の式で求められる値で，スイッチ遅延，ケーブル長遅延，およびその他遅延に分類して示したものである．

この図より，n=8 のケースにおいては，内訳に占めるスイッチ遅延の割合が高く，ケー

ブル遅延の割合が低いことが分かる．また，遅延の改善はおおむねスイッチ遅延の改善によることも分かる．一方，n=12 のケースにおいては，異なる様相を示し，内訳に占めるケーブル遅延の割合が増大している．n=8 のケースと同様に，スイッチ遅延についてはどの同型 Network も同様に改善している．しかし，ケーブル遅延については，平均最短距離で最適化した DI-HC および DI-FHC はケーブル遅延が大幅に増大し，それがスイッチ遅延の削減を打ち消し，全体として平均最短遅延の悪化に繋がっている．それに対して，平均最短遅延で最適化した DI-HC-r および DI-FHC-r はケーブル遅延を僅かながらも改善しており，スイッチ遅延の削減効果とあわせて平均最短遅延の削減に繋がっている．DI-HC での数値を見ると，スイッチ遅延は D-HC と比較して 90ns 減少しているが，ケーブル遅延は 172ns 増加し，全体として 81ns の悪化となっている．DI-FHC においても，D-FHC と比較してスイッチ遅延は 54ns 減少しているがケーブル遅延は 107ns 増加し，全体としては 53ns の悪化となっている．一方，DI-HC-r の数値を見ると，スイッチとケーブルの遅延は D-HC と比較してそれぞれ 89ns と 18ns 改善し，全体として 107ns 改善している．D-FHC-r においても，スイッチとケーブルの遅延は D-FHC と比較してそれぞれ 48ns と 49ns 改善し，全体として 98ns 改善している．

上記の結果より，ケーブルによる遅延を考慮した平均最短遅延で最適化した二重同型ネットワークが，平均最短遅延を削減し全対全最大トラフィックを向上させるという観点で有効であることが確認できる．

4.3.3　経済的コストの計算

経済的コストは，InfiniBand の市販価格をベースに試算する．今回のコスト算出の範囲は，NIC とスイッチ，およびノードとスイッチ間のケーブル，スイッチとスイッチ間のケーブルとし，ノード当たりのコストを様々なネットワークサイズで評価した．以降，コストを計算するにあたっての想定と算出方法を述べ，試算結果を示す．

コスト試算方法

Mellanox 社が販売している NIC とスイッチのポート数と価格の一覧を表 4.5(a)(b) に示す [72]．NIC のコストは各ノードが 2 ポートの NIC を 1 個利用するものとした．スイッチのコストについては，文献 [58, 61, 63, 64] を参考に，以下の方式で算出した．今回評価する構成のスイッチのポート数は 36 以下となる．スイッチ当たりのノード数は 4 としているので，n が 8 の時に Hypercube で 12 ポート，Folded-Hypercube で 13 ポート．n が 12 の時に Hypercube で 16 ポート，Folded-Hypercube で 17 ポートとなる．36 ポートを大きく下回るポート数からなるため，スイッチコストがポート数に比例するとして，36 ポートのスイッチコストから以下の方式で推定することとした．

表 4.5 InfiniBand EDR の市販価格

(a) NIC

Ports	Price (US$)
1	399
2	485

(b) Switch

ports	Price (US$)
36	11,170
216	50,500
324	61,995
648	93,000

(c) Copper cable

Length (m)	Price (US$)
2.0	134
2.5	140
3.0	148
4.0	242
5.0	278

(d) Optical cable

Length (m)	Price (US$)
10	545
15	575
20	610
30	685
50	975
100	1,665

$$\text{スイッチコスト} = 310.28 \times \text{ポート数} \quad (US\$)$$

ケーブル長の算出は 4.3.1 節で示した方法で行い，選択したケーブルのコストを表 4.5(c)(d) に示す市販価格 [72] を元にコストを算出した．

コスト試算結果

前節の試算方法に従って算出した，ノード当たりのネットワーク・コストを図 4.8 に示す．図から，ネットワークサイズの増加によってコストが悪化していることが確認できる．これは，ネットワークサイズが増えることによって，ノード当たりのスイッチポート数やケーブル数が増加すること，同時にラック数が増えシステム規模が大きくなりケーブル長が伸びることによる．また，平均最短距離で最適化した二重同型ネットワーク (DI-HC, DI-FHC) はネットワークサイズの増加によって二重ネットワーク (D-HC,D-FHC) よりもコストが悪化するが，平均最短遅延で最適化した二重同型ネットワーク (DI-HC-r, DI-FHC-r) ではそれが抑えられていることも確認できる．以降，二重同型化によるコストの変化に注目して議論する．

二重同型化によって変化するのはスイッチ間のケーブル長とそのコストである．図 4.9 にそれぞれ，平均ケーブル長とノード当たりのケーブルコストを示す．図 4.9(a) に示した平均ケーブル長の変化をみると，平均最短距離で最適化した場合は同型化によって大幅

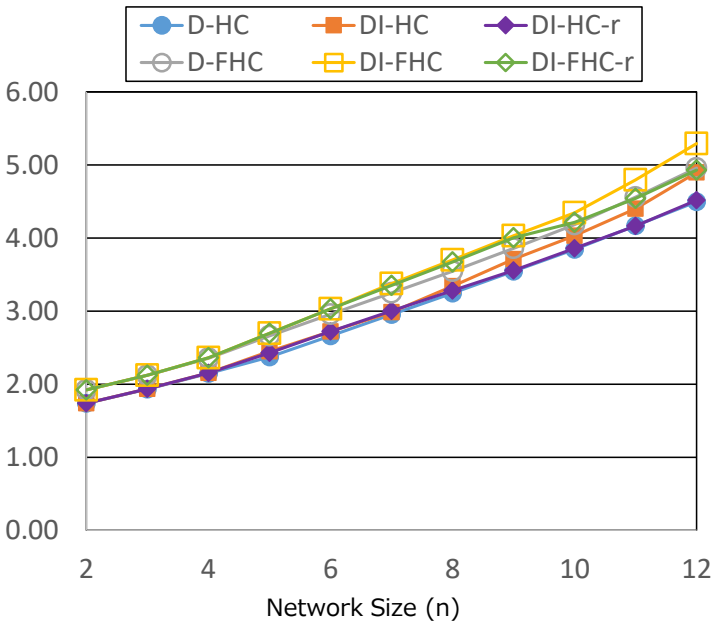
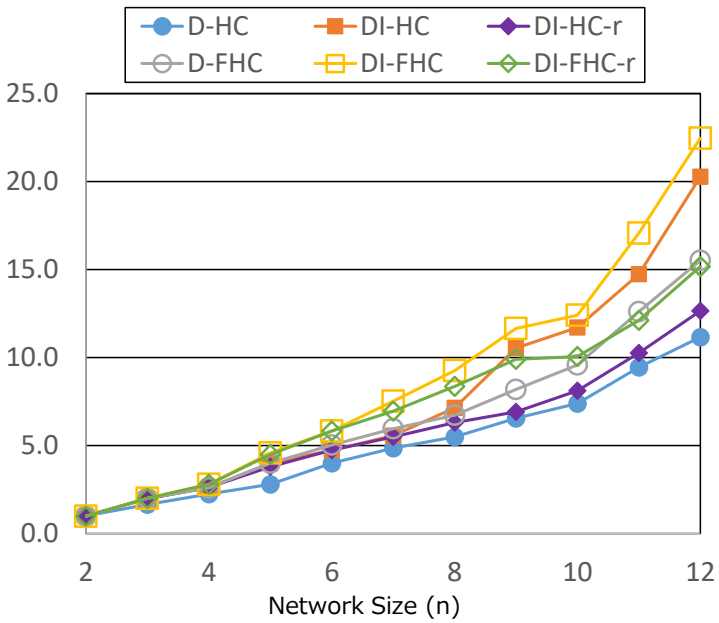
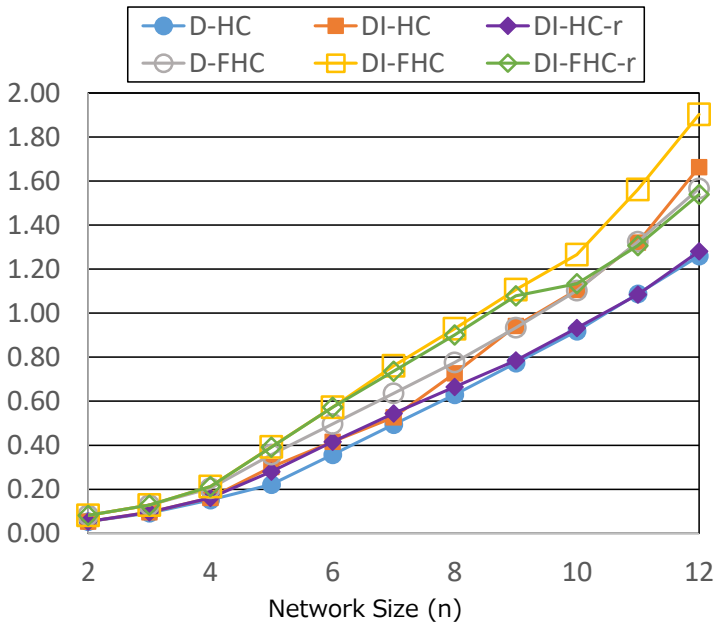


図 4.8　経済的ネットワークコスト評価結果 (k US$)

に悪化するが，ケーブル長を考慮した平均最短遅延で最適化した場合はその悪化を抑えることができていることが分かる．また，n が大きいほど平均最短遅延で最適化した場合にケーブル長の悪化を抑えられているのは，ケーブル長が遅延に与える影響が大きいためと考えられる．n=8 の時に，DI-HC は D-HC の 5.5m から 7.1m(30%) と悪化しているが，DI-HC-r は 6.3m(15%) に抑えられている．DI-FHC においても，D-FHC の 6.7m から DI-FHC は 9.3m(38%) に悪化しているが，DI-FHC-r は 8.4m(24%) に抑えられている．さらに，n=12 の時を見るとその差は顕著で，DI-HC は D-HC の 11.2m から 20.3m(82%) と大幅に悪化しているが，DI-HC-r では 12.7m(13%) 程度に抑えられている．DI-FHC の場合も同様に，D-FHC の 15.5m から 22.5m(45%) と悪化しているが，DI-FHC-r では 15.2m(-2.2%) と逆に減少している．図 4.9(b) に示したケーブルコストからも，ケーブル長と同様の傾向が見て取れる．平均最短距離で最適化した場合は同型化によって大幅に悪化するが，ケーブル長を考慮した平均最短遅延で最適化した場合は，特に n が大きくなると，その悪化を抑えることができていることが分かる．n=8 の時に，DI-HC は D-HC の 0.63k$から 0.73k$(15%) と悪化しているが，DI-HC-r は 0.67k$(5.6%) に抑えられている．DI-FHC においては，D-FHC の 0.78k$から DI-FHC は 0.93k$(20%) に悪化しており，DI-FHC-r では 0.90k$(20%) と若干改善している．さらに，n=12 の時を見るとその差は顕著で，DI-HC は D-HC の 1.26k$から 1.66k$(32%) と大幅に悪化しているが，DI-HC-r では 1.28k$(1.7%) に抑えられている．DI-FHC の場合も同様に，D-FHC の 1.57k$から 1.90k$(22%) と悪化しているが，DI-FHC-r では 1.54k$(-1.7%) と逆に減少している．ケーブル長の変化よりもケーブルコストの変化が小さくなっているが，これ



(a) Average Cable Length (m)



(b) Average Cable Costs per Node (k US$)

図 4.9　ケーブル長とケーブルコスト

は表 4.5(c)(d) より m 当たりのコストはケーブル長が伸びるほど低下するためと考えられる．

次に，図 4.10 に示したコストの内訳を見ると，n が 8 の時はケーブルコストが全体に占める割合が比較的低いことから，ケーブルコストの悪化がネットワークコストの悪化に

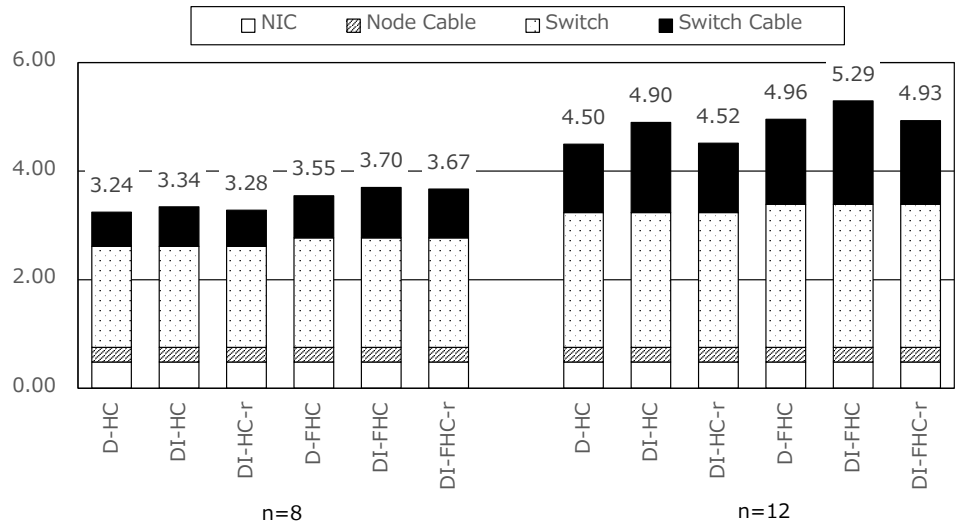


図 4.10　ネットワークコストの内訳 (k US$)

与える影響が低くなっていることが分かる．また，n が 12 の時はケーブルコストの影響が大きくなり，平均最短距離で最適化した場合はケーブルコストの悪化によって全体コストも悪化しているものの，平均最短遅延で最適化することでケーブルコストの悪化を抑え全体コストの悪化も抑えられていることが分かる．n=8 の時を見ると，スイッチケーブルのコスト割合は DI-HC の場合で 19%，DI-FHC の場合で 22% となっている．そのため，二重同型ネットワークの採用によるコストの増加割合は，最も大きい DI-FHC の場合でも D-FHC からの 4.3% となっている．n=12 の時では，スイッチケーブルのコスト割合が増加して DI-HC に占める割合は 28%，DI-FHC では 32% となっている．そのため平均最短遅延で最適化してケーブル長の増加を抑えた影響が見て取れ，DI-HC の場合では D-HC の 4.50k$から 4.90k$(9.0%) に増加しているが DI-HC-r では 4.52k$(0.47%) に抑えられている．また，DI-FHC の場合でも，D-FHC の 4.96k$から 5.29k$(%) に増加しているが，DI-FHC-r では 4.93k$と逆に 0.55% 改善している．

上記の結果より，ケーブル遅延を考慮した平均最短遅延で最適化した同型ネットワークを選択することが，ネットワークコストの観点でも有効であることが確認できる．

4.3.4　サイクルレベルシミュレーションによる評価

性能評価方法

ネットワークのスイッチの動作やケーブル遅延等をサイクルレベルで模擬したシミュレータを用いネットワークの遅延およびスループットの評価を行った．

ネットワークのサイズとしては，n=8 および n=12 の 2 構成で評価を行った．なお，スイッチ当たりのノード数は 4 としているため，総ノード数は 1024 および 16386 の構成と

なる．スイッチの構成は，入力バッファ，クロスバ，および小容量の出力バッファからなるものとした．ランダム通信においても内部構成がボトルネックにならずリンクと同等に近い性能が出せるように，スイッチ内のクロスバは倍速動作の 1 ポート当たり 200Gbps で動作するものとした．スイッチのポート to ポートの最短遅延は [67] を参考に 90ns とした．スイッチ間およびノード・スイッチ間のリンク速度は，InfiniBand EDR のリンク速度である 100Gbps とした．リンクの遅延は，システム構成で求めたケーブル長に 5ns をかけた値とした．ネットワーク内でのルーティングについては，静的な次元オーダーの最短ルーティングで評価を行った．

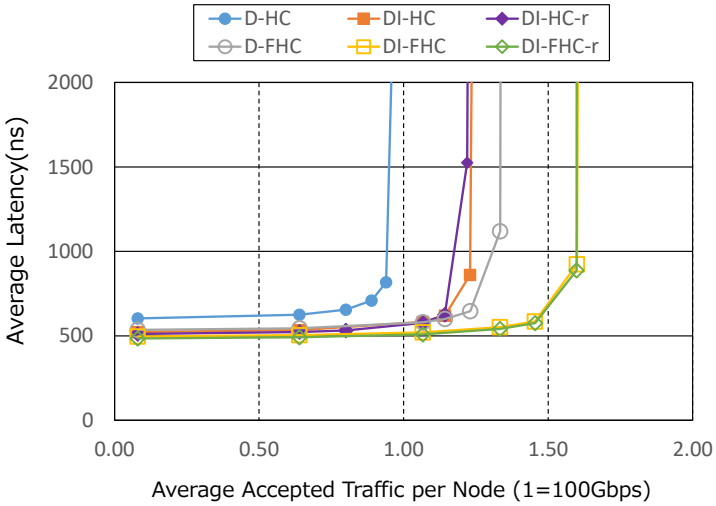
ノード間の通信パターンとしては，Uniform Random で評価を行った．各ノードは，上記通信パターンのパケットをある一定間隔で Network に送出し，シミュレーションを走らせて安定状態での受け取ったパケット量のノード当たりの平均 (Average Accepted Traffic) と，パケットの送出から受信までの平均遅延 (Average Latency) を測定した．ノードのパケット送出間隔を変更して複数回シミュレーションを行い，それぞれのシミュレーションで得られた複数の Average Accepted Traffic と Average Latency の組みでネットワークの特性を明らかにした．

性能評価結果

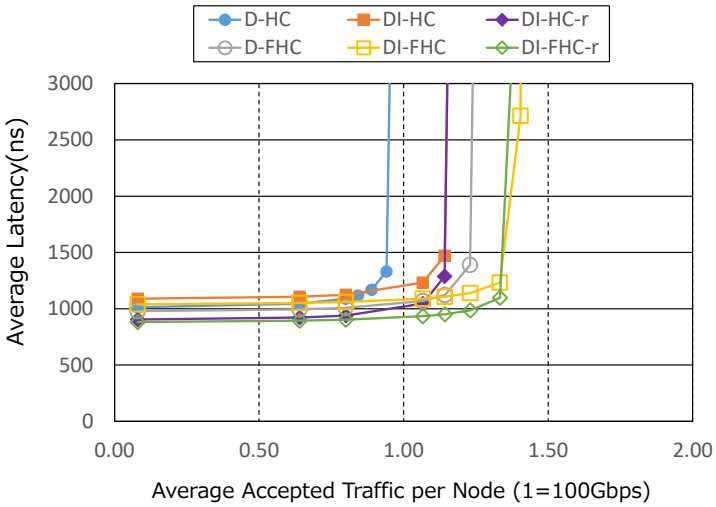
性能評価を行った結果を，図 4.11 に示す．以降の議論では，得られる最大の Average Accepted Traffic をネットワークのスループットとし，また最小の Average Latency をそのネットワークの遅延とした．これらは，[49] でも述べられている一般的な定義である．

図より, n=8 および n=12 の両方のケースで, 二重同型化（DI-HC, DI-HC-r, DI-FHC, DI-FHC-r）によってスループットが向上していることが確認できる．n=8 のケースでは，DI-HC と DI-HC-r のスループットはそれぞれ D-HC から 27% と 25% 向上している．また，DI-FHC と DI-FHC-r は，それぞれ D-FHC から 20% と 19% 向上している．n=12 のケースでは，DI-HC と DI-HC-r のスループットはそれぞれ D-HC から 19% と 18% 向上している．また，DI-FHC と DI-FHC-r は，それぞれ D-FHC から 12% と 10% 向上している．

遅延を見ると，n=8 のケースで二重同型化によって遅延が削減されていることが確認できる．平均最短遅延と同様に，n=12 のケースでは平均最短距離で最適化した DI-HC および DI-FHC は元の D-HC，D-FHC よりも悪化している．一方平均最短遅延で最適化した DI-HC-r および DI-FHC-r であれば遅延を削減できていることも確認できる．n=8 のケースでは，DI-HC と DI-HC-r の遅延はそれぞれ D-HC から 13% と 15% 減少し，DI-FHC と DI-FHC-r はそれぞれ D-FHC から 7.3% と 9.5% 減少している．n=12 のケースでは，DI-HC の遅延は D-HC から逆に 7.4% 増加し，DI-FHC の遅延は D-FHC から 6.2% 増加している．一方，DI-HC-r，DI-FHC-r であればそれぞれ 11% と 9.9% 減



(a) n=8



(b) n=12

図 4.11　性能評価結果

少している．

上記の結果より，これまでの解析結果と同様に，ケーブルによる遅延を考慮した平均最短遅延で最適化した同型ネットワークを選択することが，遅延を削減しスループットを向上させるという観点で有効であることが確認できる．

4.3.5　評価指標の検証

全対全最大トラフィックと平均最短遅延は，Uniform Random 通信時の最大スループットと最小遅延を表すものとして導入した．図 4.6 のグラフ解析結果と，図 4.11 のサイ

クルレベルシミュレーションによる評価結果比較すると，全対全最大トラフィックの誤差は平均して 2.75%(最小 1.32% から最大 3.82%) となっており，平均最短遅延は 0.48%(最小 0.14% から 1.1%) となっている．両者とも大きな差にはなっておらず高い精度で一致していると言える．

4.4 結論

分散共有メモリシステムにおいては，メモリアクセスレイテンシの削減は重要な課題であり，それにはネットワークの性能向上が重要となる．3 章における評価で，メモリアクセスレイテンシに占めるネットワークの通信遅延の割合は 50% から 65% になることが分かっている．システムの大規模化に従ってネットワークのレイテンシは増加しており，そのレイテンシを抑えることが重要となる．

本章では，Multi-plane ネットワークを採用したシステムにおいて，そのネットワーク性能を向上させる二重同型ネットワークの提案を行い，Hypecube ネットワークおよび Folded-Hypercube ネットワークを対象として，その遅延とスループットの評価を行った．この二重同型ネットワークは，二つのネットワーク・プレーンを保有するシステムにおいて，各プレーンをグラフ同型のネットワークで接続することを特徴とする．

同型ネットワークの候補は複数存在するため，二つのネットワーク選定法を示した．一つは，ネットワークの評価指標として良く用いられる平均最短距離と，スループットに相当する全対全最大トラフィックの両指標を改善する同型ネットワークの選定法である．もう一つは，遅延指標として平均最短距離だけでなくスイッチ間のケーブル遅延も考慮した平均最短遅延と，全体全最大トラフィックの両指標を改善する同型ネットワークの選定法である．

グラフ解析やサイクルレベルのシミュレーションの評価結果から，後者の選定法が遅延・スループット・経済的コストの観点から有効な選定法であることが確認された．両手法ともスループットについては改善効果を示し，両者に大きな差がないことが確認された．遅延についても 8 次元程度までの Hypercube であればどちらの方法でも同程度に改善できることも確認された．しかし，ネットワーク・サイズが大きくなりシステム規模が大きくなると，平均最短距離で最適化する手法ではケーブル長が伸びる影響を受けてケーブル遅延が増加し，遅延が逆に悪化することも明らかとなった．一方，平均最短遅延で最適化する手法であればケーブル遅延を指標に組み込んだことからそれを改善できることも確認された．経済的コストにおいても，遅延同様に，ネットワーク・サイズが大きくなると平均最短距離で最適化した場合はケーブル長が伸びることからコストが増加するが，平均最短遅延で最適化するとコストの増加を防ぐことができることも確認できた．

今後，様々なトラフィックパターンや，アプリケーションを用いた性能評価を行う必要

がある．Hypercube 以外の Network への適用，三重以上の Multi-plane ネットワークへの適用等の議論を進める．また，3 章で議論したマルチキャストとギャザー機能を二重同型ネットワークで実現する手法について十分に検討できておらず，今後その議論が必要である．一つの簡単な実装は，いずれか一つのプレーンを用いて実現する方式となる．しかし，その場合レイテンシの削減効果が得られないため，二つのプレーンを用いてマルチキャストとギャザーを実現する方式が有効かを議論する必要がある．

第 5 章

デッドロック・スタベイションフリーなプロトコル

5.1 はじめに

本章では，高性能かつ安定動作するディレクトリ方式の一貫性制御の実現における四つの課題のうち，四つ目の一貫性制御の安定動作を目的としたメモリアクセスが有限時間内に完了することの保証について述べる．

一貫性制御は，複数のノード間でメッセージをやり取りすることによって実現され，その手順はコヒーレンスプロトコルとして定義される．一連のメッセージのやり取りにおいて，要求を受けたノードは別のノードに要求あるいは応答を出力する．ネットワークの混雑により出力先のバッファがフルで要求や応答を出力できない状態になると，そのノードは受けた要求の処理を完了できず次の要求を受け付けられない状態となる．このようなメッセージを受ける資源とメッセージの出力先の資源の依存関係によってデッドロックが発生する．ノード間のメッセージ配信をデッドロックなしで行うネットワークのみでは，この一貫制御によるデッドロックは防止できない．

また，DASH [37] 等のコヒーレンスプロトコルでは，ある要求メッセージをメモリが受け付けて処理したが，他の要求を処理中で未完了の状態であった場合に，要求メッセージを発行したノードに対して否定応答を送信するケースが存在する．否定応答を受けたノードは要求メッセージの再送を行うが，その間にメモリがまた別の要求を受け付けて処理中の状態になっていると，また同じように否定応答が送信されることとなる．このような否定応答からの要求メッセージの再送が，他の要求に割り込まれて無限に繰り返された場合にスタベイションが発生する．

従来 DASH [37] では，要求と応答用に独立した 2 つのネットワークチャネルを持つことでデッドロックの問題に対応していた．そのため，ネットワークスイッチにおいてそれ

ぞれのチャネルに対応する独立したバッファを必要とし，スイッチのチップコストが増大する問題があった．さらに，これだけではデッドロックを防止することができないため，ある要求の処理を進めることができずデッドロックが発生する可能性を検知した場合は，その要求の処理を進めず要求を発行したノードに対して否定応答を返してデッドロックを回避していた．このようにデッドロックの発生を検知して回避する方式は否定応答を伴うため，デッドロックの問題を解決できたとしてもスタベイションの問題が発生することになり，メモリアクセスが有限時間内に完了することを保証するという点では問題の解決になっていなかった．

また，DASH [37] で採用されているコヒーレンスプロトコルは，あるメモリアクセス要求に関してその時点で一貫性を維持することができない状態に陥った場合，要求発行ノードに否定応答を返し再試行させていた．そのため，メモリアクセス要求が否定応答され続ける，すなわちスタベイションが発生する可能性があった．

Scalable Coherence Interface(以降 SCI) [32] で採用されたプロトコルは，このスタベイションの問題を解決している．データを保持するキャッシュにディレクトリを配置する分散ディレクトリ方式を採用し，メモリと共有しているノードの連結リストを構成する情報をそれぞれのディレクトリに分散保持する．複数のノードからの要求があった場合に，それらのノードを分散ディレクトリで構成する連結リストに要求を受け付けた順に結合し，その順に要求を処理していく．これにより，スタベイションを防止している．しかし，この方式は分散ディレクトリ方式で連結リストを構成する方式に強く依存したものであり，フルマップ方式 [27]，Coarse Vector 方式 [29]，また 5 章で提案したビットパターン方式等の連結リスト構造をもたないディレクトリ方式には適応できない．また，この SCI プロトコルでは，一貫性制御において，連結リストを辿るためにその情報を保持しているキャッシュを順に処理することになるため，データを共有しているノード数回の通信が必要となり，書き込み時の遅延が大幅に悪化する問題があった．

本章では，これらの問題を解消し，単一のチャネルしか持たないネットワークにも適用可能で，デッドロックとスタベイションの両方を防止可能な方式の提案を行う [89]．提案方式は以下の三つの特徴を持つ．一つ目の特徴は一貫性制御のプロトコルの設計であり，否定応答の発生をなくすことと，メモリアクセスレイテンシの増加を抑えるために，要求を出したプロセッサが応答を受け取るまでの通信回数を少なくすることの 2 点を実現する．これにより，メモリアクセスレイテンシを悪化させることなく次の特徴との組合せでスタベイションフリーを実現する．二つ目の特徴は，スタベイション防止のために，メモリにおいて同一ブロックに対する要求を受け付けた順番で処理する方式である．これを実現するために，メモリにおいて他の要求を処理中で別の要求を処理できない場合に，その要求メッセージをメモリに退避し後に処理可能になったタイミングで，受け付けた順番に処理を行う．この一つ目と二つ目の特徴によってスタベイションフリーな一貫性制御を実

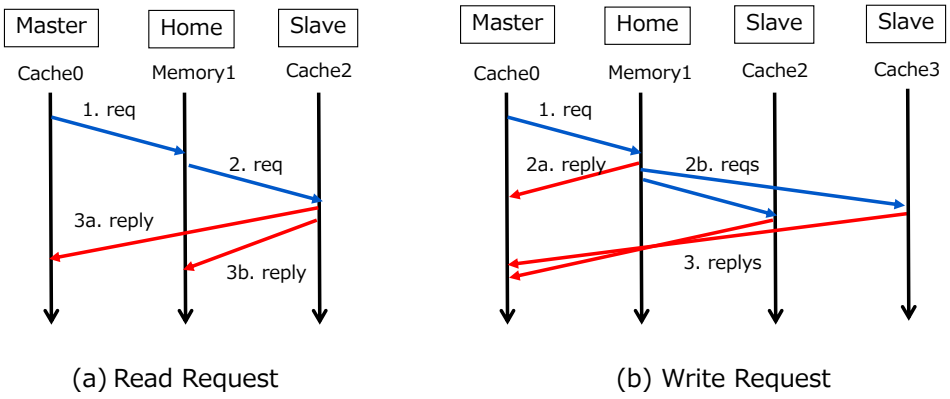


図 5.1 DASH プロトコル

現する．三つ目の特徴は，要求や応答を処理中に他のメッセージを受け付けることができない状況でデッドロックが発生するため，処理中であっても特定の要求や応答を全て受信可能とする方式である．これを実現するために，メモリに用意したバッファに特定の要求や応答メッセージを待避可能とする．これによってデッドロックの発生を防ぐ．

以降，5.2 節では，従来提案されてきたデッドロック・スタベイション防止方式の問題点を明らかにする．5.3 節では，提案するデッドロック・スタベイション防止方式について述べる．5.4 節では，この方式の並列計算機 Cenju-4 における実装について述べ，またアプリケーションを用いて評価した結果を示す．5.5 節でまとめる．

5.2 従来方式とその課題

一貫性制御は，複数のノード間でメッセージをやり取りすることによって実現され，その手順はコヒーレンスプロトコルとして定義される．このプロトコルがデッドロックやスタベイションにおいて重要な役割を占めるため，本節では従来方式の代表的なプロトコルとして DASH [37] で用いられているプロトコルと，スタベイションの解決を行っている SCI [32] プロトコルについて述べ，その方式と課題を示す．

5.2.1 DASH プロトコル

DASH のプロトコルの代表的な動作を図 5.1 に示す．図では，メモリアクセスを発行したプロセッサを有するノードをマスタ，アドレスが示すメモリを含むノードをホーム，キャッシュにコピーを保持しているノードをスレーブと記している．分散共有メモリシステムでは，これらマスタ，ホーム，スレーブ間でメッセージをやり取りすることで，キャッシュの一貫性制御が行われる．

図 5.1(a) は読出しアクセス要求におけるシーケンスで，メモリに最新のデータが存在せず他のキャッシュが保有しているケースとなる．マスタはホームに対して読出し要求 (1.req) を発行する．ホームは最新のデータを保有するスレーブに対してその要求を転送する (2.req)．要求を受けたスレーブは，要求を発行したマスタに対してデータを応答 (3a.reply) するとともに，データをホームに書き戻す (3b.reply)．マスタはデータを受け取り処理を完了する．このように動作することで，3 ステップの通信によってマスタは応答を受け取り処理を完了する．

次に，図 5.1(b) は書込みアクセス要求におけるシーケンスで，複数のキャッシュがデータを共有しているケースとなる．マスタはホームに対して書込み要求 (1.req) を発行する．ホームはデータをマスタに対して応答 (2a.reply) し，データを保有している複数のスレーブに対してそれぞれ書込み要求 (2b. req) を転送する．要求を受けたスレーブは，キャッシュのデータを無効化し，マスタに対して応答 (3.reply) を返す．マスタは，ホームからのデータと複数のスレーブからの応答を受け取り処理を完了する．このケースでも，3 ステップの通信によってマスタは全ての応答を受け取り処理を完了する．

デッドロックを防止するために，DASH においては要求を流すチャネルと応答を流すチャネルの二つの独立したチャネルをネットワークに有する．これにより，要求を受けたノードが応答を送信するケースについてはデッドロックを防止することができる．一方，ホームで発生する要求を受けて要求を転送するケースについては，デッドロックの発生が防止できない．そのため，要求を転送する処理が一定時間以上ブロックされる状態が続くと，要求を転送する処理を停止し否定応答をマスタに対して送信する．これによって，デッドロックの発生を回避している．否定応答を受けたマスタは，要求を再試行する処理を行うことで，一貫性処理を再試行する．

DASH においてはスタベイションの防止は行われていない．上記のデッドロックの回避方法で示したように，ホームはデッドロックの発生が懸念される場合は否定応答をマスタに返す．また，その他にも，読出し要求や書込み要求を処理している最中に同じデータに対して要求を受け取った場合にも否定応答が発生する．例えば，図 5.1(a) においてマスタがスレーブに読出し要求を転送している最中に，他のノードから読出し要求を受け取った場合である．この場合，ホームは後で受け取った要求もスレーブに対して転送する処理を行う．スレーブにおいては先行する要求が先に処理されデータがホームに返却されている．スレーブは後続の要求に対して応答を返却することができず，否定応答を後続する要求を出したマスタに返却することとなる．マスタは，否定応答を受けて要求を再発行するが，また同様の状態が発生するとこれを繰り返すことになりスタベイションが発生する．

以上，DASH においては，デッドロックの防止は要求用と応答用の二つの独立したチャネルを持ったネットワークと，要求に対して否定応答を返すことで実現されていた．そのため，単一のチャネルしか持たないネットワークには適用できない．また，スタベイショ

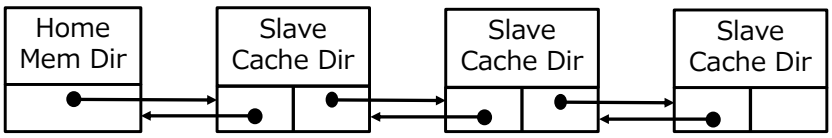


図 5.2　SCI の分散ディレクトリ

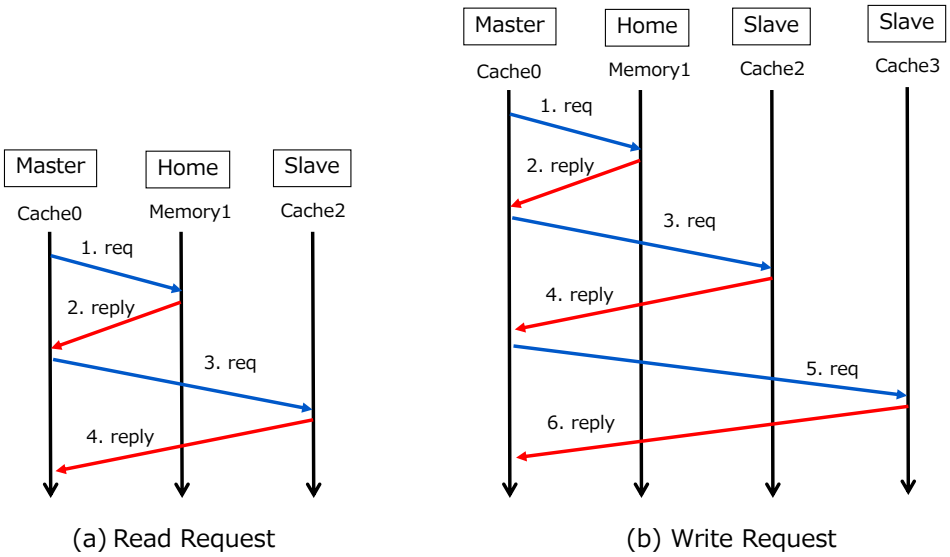


図 5.3　SCI プロトコル

ンは防止されていなかった．

5.2.2　SCI プロトコル

SCI プロトコルの特徴は，分散ディレクトリ方式であることである．分散ディレクトリ方式は，メモリでのディレクトリ情報に加えて，データを保持するキャッシュにおいてもディレクトリ情報を保持する方式である．図 5.2 に分散ディレクトリによる共有ノードの管理を示す．ホームのメモリを先頭として，データを保有するスレーブのキャッシュを双方向のリンクドリストで接続する方式となる．ホームは，先頭のスレーブノードへのポインタをディレクトリに保持する．各スレーブのキャッシュは，二つのポインタを持ち，それぞれが前後のノード（ホームあるいはスレーブ）を指す．

この分散ディレクトリによる SCI プロトコルのプロトコルの代表的な動作を図 5.3 に示す．図 5.3(a) は読出しアクセス要求におけるシーケンスで，メモリに最新のデータが存在せず他のキャッシュが保有しているケースとなる．マスタはホームに読出し要求 (1.req) を発行する．ホームは応答 (2.reply) をマスタに返す．この時，スレーブノードへのポイ

ンタも伝えられる．マスタは応答を受け取り，スレーブに対して要求 (3.req) を転送する．スレーブはデータをマスタに応答 (4.reply) し，マスタはデータを受け取り処理を完了する．このように動作することで，4 ステップの通信によってマスタは応答を受け取り処理を完了する．

次に，図 5.3(b) は書込みアクセス要求におけるシーケンスで，複数のキャッシュがデータを共有しているケースとなる．マスタはホームに書込み要求 (1.req) を発行する．ホームはデータをマスタに対して応答 (2.reply) し，先頭スレーブノードへのポインタ情報も伝える．マスタはこの応答を受け取り，先頭のスレーブノードに要求 (3.req) を送信する．スレーブは要求を受け取りキャッシュのデータを無効化し，マスタに対して応答 (4.reply) を返すとともに，次のスレーブノードへのポインタ情報も伝える．マスタは，この処理を全てのスレーブに対して順に行うため，図に示した 2 ノードで共有するケースでは処理の完了までに 6 ステップの通信を必要とする．

デッドロックの防止のために，DASH 同様に要求を流すチャネルと応答を流すチャネルの二つの独立したチャネル有するネットワークを用いる．また，上記で示したように，常にマスタからホームあるいはスレーブへの要求の送信と応答の回収というステップで行われ，マスタが要求発行時にそれに対する応答を受け取ることができるバッファを確保することで，デッドロックは防止される．

スタベイションの防止については，図 5.2 に示したリンク構造を活用して実現される．ホームは要求を受けると，その要求を発行したノード (以降ノード A とする) を先頭のスレーブノードとするディレクトリ構造へと変更する．ホームが次の要求を受け取ると，その要求を発行したマスタ (以降ノード B とする) に対して先行する要求を発行したノード A へのポインタを応答に付与して返す．ノード B のマスタは，ノード A のスレーブに対して要求を送信する処理を行う．ノード B からの要求を受け取ったノード A は，応答を返却できる状態，即ちノード A がマスタとして発行した要求の一貫性制御処理が完了するのを待ってから応答を行う．このようにホームが要求を受けた順にリストが構成され処理が行われる．これによってスタベイションが防止される．

以上，SCI においては，デッドロックの防止は二つの独立したチャネルを持ったネットワークが前提となっている．そのため，DASH 同様に単一のチャネルしか持たないネットワークには適用できない．また，スタベイションは防止されるものの，その実現が分散ディレクトリを前提としており，多数のノードがデータを共有する場合の一貫性制御の遅延が非常に大きくなる問題がある．

5.3 提案するデッドロック・スタベイション防止方式

本章で提案する方式は，単一のチャネルしか持たないネットワークに適用可能な方式であり，また DASH 同様に 3 ステップの通信によってマスタでの処理が完了し，一貫性制御の遅延を低く抑えることができることを特徴とする．

以降，提案するプロトコルについて述べ，次にデッドロックの防止方式とスタベイション防止方式を示す．

5.3.1 プロトコル

提案するプロトコルは，DASH プロトコルを改良し，3 ステップの通信によってマスタでの処理が完了するメリットを維持しつつ，否定応答の発生を回避するものである．

DASH プロトコルにおいて否定応答が発生するのは，あるマスタからの要求をホームが受け付けてスレーブに対して要求を発行し，スレーブに対する処理やそのスレーブからの応答をマスタが受け取る前に，同一データに対する他の要求をホームが受け付けて処理を進めてしまうためである．そこで，提案するプロトコルは，DASH プロトコルに対して以下の変更を行う．

- ホームは，要求を受けると該当ブロックのディレクトリに要求処理中のフラグを立てる
- スレーブおよびマスタにおいて受け付けた要求に伴う一貫性制御が完了するのを待ち，要求処理中のフラグを落とす
- 一貫性制御の完了をマスタで検知し，マスタがホームに対してその完了を通知する
- 要求処理中のフラグが立っているときに他の要求を受けた場合は，以降で説明するスタベイション防止機構を用いて，その要求を処理せず留めて置く

提案プロトコルの動作を図 5.4 に示す．図 5.4(a) は読出しアクセス要求におけるシーケンスで，メモリに最新のデータが存在せず他のキャッシュが保有しているケースとなる．マスタはホームに読出し要求 (1.req) を発行する．ホームでは最新のデータを保有するスレーブに対してその要求を転送 (2.req) する．同時に，ディレクトリの状態を要求処理中とする（要求処理中のフラグを立てる）．要求を受けたスレーブは，要求を発行したマスタに対してもデータを応答 (3.reply) する．マスタはデータを受け取り，読出しアクセスを完了させる．一方，一貫性制御は継続し，マスタはホームに対してデータを応答 (4.reply) する．データを受け取ったホームは，ディレクトリを要求処理中の状態を解除し，一貫性制御を終える．このように動作することで，マスタは 3 ステップの通信によっ

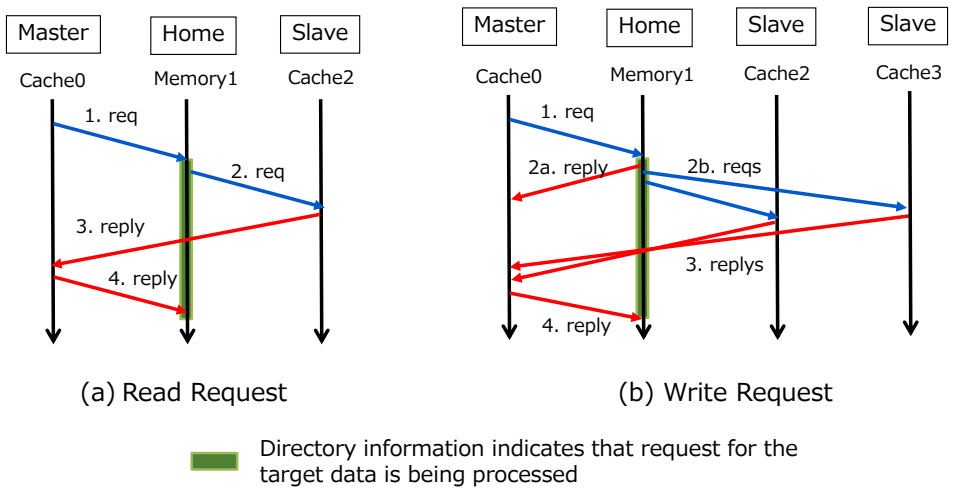


図 5.4 提案プロトコル

て応答を受け取り読出しアクセスを完了する．また，ホームはマスタからの応答を受け取ることで一貫性制御がスレーブとマスタにおいて完了したことを管理できるようになる．

次に，図 5.4(b) は書込みアクセス要求におけるシーケンスで，複数のキャッシュがデータを共有しているケースとなる．マスタはホームに書込み要求 (1.req) を発行する．ホームはデータをマスタに対して応答 (2a.reply) し，同時にデータを保有している複数のスレーブに対してそれぞれ書込み要求を転送 (2b.req) する．同時に，ホームのディレクトリの状態を要求処理中とする．要求を受けたスレーブは，キャッシュのデータを無効化し，マスタに対して応答 (3.reply) を返す．マスタは，ホームからのデータと複数のスレーブからの応答を受け取り，書込みアクセスを完了させる．同時にマスタからホームに対して応答 (4.reply) を送信する．このケースでも，3 ステップの通信によってマスタは全ての応答を受け取り書込みアクセスを完了する．また，ホームはマスタからの応答を受け取ることで一貫性制御がスレーブとマスタにおいて完了したことを管理できるようになる．

以上のように動作することで，DASH プロトコルと同様に，3 ステップの通信によってマスタは全ての応答を受け取り処理をメモリアクセスを完了できる．また，DASH プロトコルで発生していた否定応答が発生するケースを，要求処理中のフラグが立っているときに他の要求を受け付けて処理を進めないことによって回避することができる．

5.3.2 デッドロックの防止

デッドロック防止の議論のために，資源の依存関係を明らかにする．図 5.5 に一貫性制御のプロトコルによって発生する資源の依存関係を示す．図において各ノードの 3 モジュールとネットワークは資源であり，矢印はその資源間で要求や応答の送信が行われ依

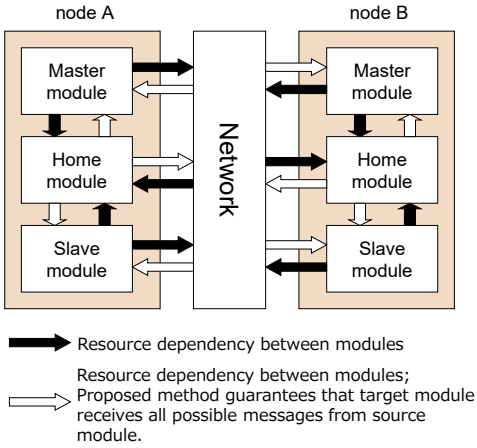


図 5.5 資源の依存関係

存関係があることを示している．今までのプロトコルの説明では一つのメモリアクセス要求に着目して各ノードをマスタ，ホーム，スレーブと分類していたが，各ノードはメモリアクセス要求によってマスタ，ホーム，スレーブと役割を変えて機能するため，それぞれの機能を持ったモジュールを有する．

提案する方式では，図 5.4 に示すプロトコルで動作する．要求や応答を受けて各モジュールが行う処理は中断されることがなく，一つの処理が終了するまで次の処理を開始できない．そのため，マスタとホームで相互に依存関係があり，ホームとスレーブについても相互に依存関係がある．一方，一つのメモリアクセス要求でマスタとスレーブが同一のケースは存在しないため，それらの間には直接的な依存関係は存在しない．また，それぞれのマスタ、ホーム、スレーブモジュールは，異なるノードに属するモジュール間とのやり取りのためにネットワークを用いる．ネットワークはそれ自身デッドロック・フリーであるが，2 ノード間のパスは一つしかなく追い越しも許していない．そのためネットワークとそれぞれのモジュールとの間で図 5.5 に示したような依存関係を持つ．この資源依存関係から明らかなように，様々なループが存在するためデッドロックが発生する可能性を持つ．

提案方式は，図 5.5 において白矢印で示した依存関係をなくすことでループをなくしデッドロックを防止するものである．メモリあるいはモジュール内にすべての要求や応答を受けきることが可能な大きさのバッファを設け，そのバッファに必要に応じて待避することで，依存関係をなくしている．必要とされるバッファは，スレーブが受ける要求用のバッファ，ホームがネットワークに出力する要求および応答用のバッファ，マスタが受ける応答用のバッファの 3 つのバッファである．他の矢印を選択してもループをなくすことは可能であるが，必要なバッファのサイズが最小化される選択を行っている．

マスタモジュールが受ける応答の数は最大でプロセッサの最大発行要求数（例えばプロセッサ R10000 [34] では 4）に抑えられる．マスタモジュールはそれをすべて受け取れるバッファをモジュール内に有している．

スレーブモジュールは要求のみを受け取り，それにはデータは含まれない．1 スレーブモジュールが受け取る要求の最大数は，プロセッサの最大発行要求数にノード数を掛けて得られる数となる．

ホームモジュールは要求とデータが付加される可能性のある応答を出力する．しかし，このデータは常にメモリブロックにあり，バッファに待避する必要はない．また，ホームは複数の書込み要求を送出するが (図 5.4(b))，バッファからネットワークに出力する際に複数の宛先に対して複製したメッセージを送信することで，あるいはマルチキャストメッセージの活用などで一つのメッセージを出力することで，バッファには一つのメッセージのみを保持するだけで済ますことができる．これにより，バッファに待避される要求や応答の最大数は，スレーブと同様にプロセッサの最大発行要求数にノード数を掛けて得られる数に抑えられる．

スレーブおよびホームモジュールともに複数個のメッセージを受け取るバッファをモジュール内に用意し，メモリに確保したバッファはこのモジュール内のバッファが溢れた場合にのみ用いることで，メモリアクセスレイテンシに極力悪影響を与えないようにすることが可能である．

この実装によって，単一のチャネルしか持たないネットワークにおいても，デッドロックの防止が可能となる．なお，本実装は，提案プロトコルに限定されることなく，DASH プロトコルにも適用可能な方式となっている．

5.3.3　スタベイションの防止

提案方式は，ホームがただちに処理できない要求を一時メモリに待避し後で処理するキューイング方式を採用している．図 5.6(a) に示すように，ホームにおいて一つの要求待避用バッファを管理する．ホームはある条件が成立した場合に受けた要求をメモリに確保したバッファに一次待避し，また応答を受けて待避した要求を読出して処理を開始する．この判断を行うために，ホームのディレクトリ情報において，他の要求を処理中かどうかを管理している．さらに，後述するキュー管理のために予約ビットをディレクトリ情報として持つ．

図 5.6(b) に動作を示す．ホームは，要求を処理して応答を待っている間に受けた要求 B と C をメモリに待避し，応答が戻ってきた時点でそれらの要求をメモリから取り出し処理する．要求を処理中の状態であることはディレクトリ情報から判別する．このように動作することで，要求に対して否定応答を返すことがなくなり，要求を受け取った順で処

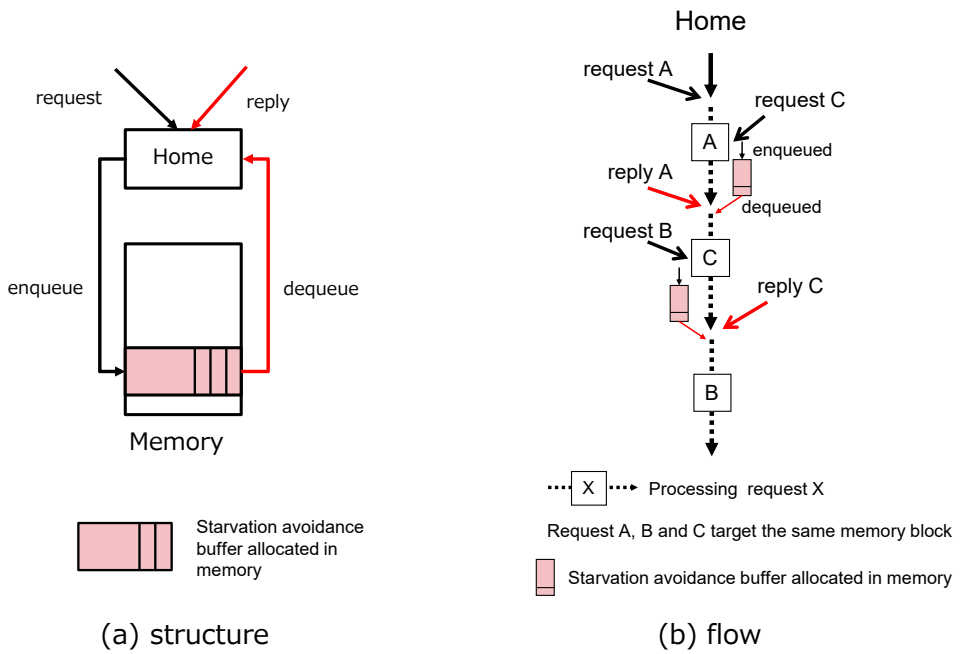


図 5.6　スタベイション防止方式

理することでスタベイションの防止が実現できる．

次に，メモリに要求を退避するためのバッファの実現方式を示す．異なるデータに対する要求も同一のバッファに退避し，バッファは FIFO キューとして管理する方式を取る．待避した要求がキューの先頭であればディレクトリの予約ビットをセットする．予約ビットがセットされたデータの応答を処理すると，予約ビットをアンセットし，キューの先頭から要求を一つ読み出し処理する．以降順にキューから要求を読み出し処理を進める．この処理は，キューが空になるまで繰り返される．ただし，まだ処理できない要求であれば，その要求に対応するディレクトリの予約ビットをセットし，そのデータに対する応答が届くのを再度待つ．これにより，データ毎にメモリにキューを持つ必要はなく一つのキューで実装可能となる．また，このバッファのサイズは，プロセッサの最大発行要求数にノード数をかけて得られる数の要求を退避するのに必要なサイズとなる．

5.4　並列計算機 Cenju-4 での実装と評価

5.4.1　実装方式

本章で提案した方式を図 3.7 で示した並列計算機 Cenju-4 で実装し評価した．図の分散共有メモリ機構において，デッドロック・スタベイション防止機構が実装され，また SDRAM の部分にバッファが確保される．また，プロセッサ R10000 が発行する最大要

表 5.1　ロードアクセスのレイテンシ (ns)

network stage (num. of nodes)	2 (∼16)	4 (∼128)
local(clean)	610	610
remote(clean)	1690	2210
local(dirty)	1900	2480
remote(dirty)	2420	3230

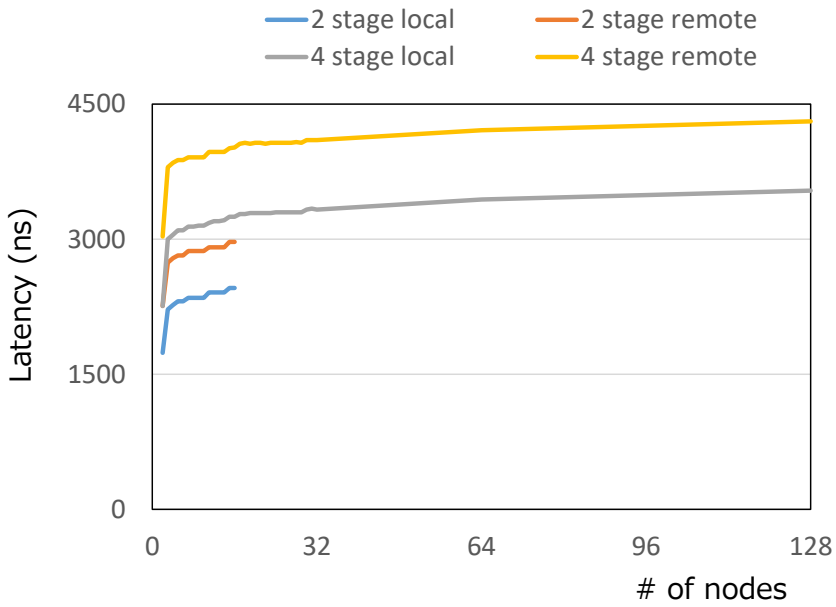


図 5.7　ストアアクセスのレイテンシ

求数は 4 となる．

デッドロック防止のために、スレーブおよびホームがメモリに確保するバッファのサイズは，それぞれ，当該バッファに退避する 1 メッセージの情報が 16 バイト，1 プロセッサの最大発行要求数が 4，最大ノード数が 1024 であることから，64K バイトとなる．またスタベイション防止のためのバッファは，当該バッファに退避する 1 メッセージの情報が 8 バイト，1 プロセッサの最大発行要求数が 4，最大ノード数が 1024 であることから，32K バイトとなる．

512M バイトのメモリに対してあわせて 192K バイトと非常に小さいサイズのバッファに抑えることができている．

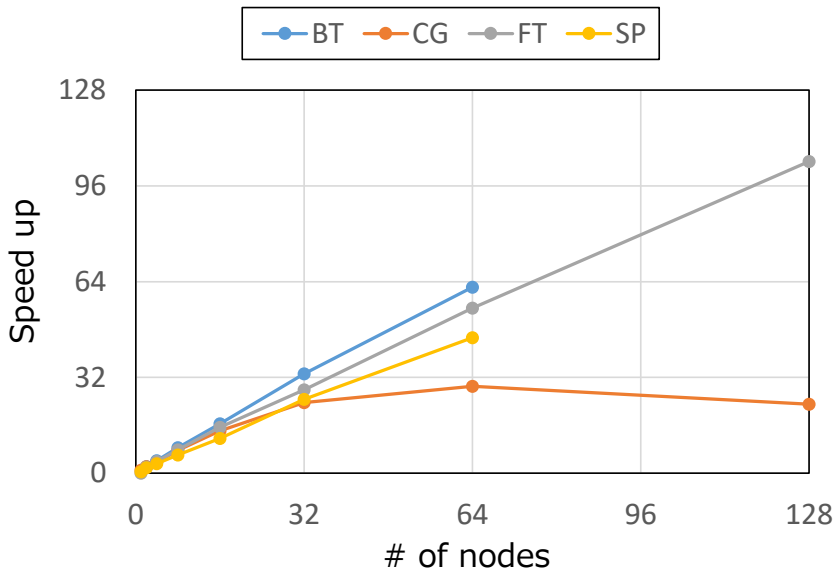


図 5.8　プログラムの実行性能

5.4.2　Cenju-4 のメモリアクセスレイテンシの評価

スタベイションやデッドロック防止機構がメモリアクセスレイテンシに与える影響を評価するために，まず Cenju-4 の基本的なメモリアクセス性能を測定した結果を示す．この結果と，後述するアプリケーションでのメモリに用意したバッファへのメッセージの退避率とその退避コストから影響を見積もる．

ロードおよびストアアクセスを行ったときのレイテンシを測定した結果を，表 5.1 と図 5.7 に示す．それぞれ，読み出しを行って 2 次キャッシュにミスした場合，複数のノードで共有しているデータに対して書き込みを行った場合のレイテンシである．Cenju-4 は多段のネットワークで構成されており，その段数によりレイテンシが異なる．今回の評価では 2 段（〜16 ノード）の場合と 4 段（〜128 ノード）の場合のレイテンシを測定した．

ロードアクセスのレイテンシは，表 5.1 を見ると，ネットワークを経由したアクセスを不要とする local(clean) のケースで 610ns となるものの，他のケースではネットワークの遅延が加わり，ネットワークが 2 段のケースで 1690〜2420ns、ネットワークが 4 段のケースで 2210〜3230ns となっている．

ストアアクセスレイテンシは，図 5.7 を見ると，共有しているノード数によって値が変わるものの，ネットワークが 2 段のケースで 1740〜2970ns、ネットワークが 4 段のケースで 2260〜4310ns となっている．

表 5.2　メッセージのバッファへの待避率

	Starvation	Deadlock	
	home	salve	home
BT	0.27%	2.9%	0%
CG	0.11%	0.92%	2.0%
FT	0%	0.07%	0%
SP	0.29%	0.49%	0.01%

5.4.3　スタベイションおよびデッドロック機構の評価

ワークロードとしては，NASA Parallel Benchmark (NPB) [43] の CLASS A を用いて評価を行った．使用したプログラムは BT，CG，FT，SP の 4 つである．提供されている逐次版のプログラムの最外ループの並列化を行った．また，共有配列の分割非共有化，共有データのノードへの配置指定を行い，メモリアクセスの最適化を施した．実行にはネットワーク段数が 4 段の 128 ノードシステムを利用した．

この 4 つのプログラムについて実行性能を測定した結果を図 5.8 に示す．CG において性能が飽和していものの，他の BT,FT,SP においては高い台数効果を示している．また，これらのアプリケーションの実行において，スタベイションやデッドロックが発生することはなく，プログラムの実行が停止することはなかった．

次に，BT と SP については 64 ノードで，CG と FT については 128 ノードのケースに，メモリに設けた三つのバッファへのメッセージの待避率を測定した．スタベイション防止用にホームが管理するバッファ，デッドロック防止用にスレーブが管理するバッファとホームが管理するバッファである．待避率は，メモリに待避されたメッセージ数をモジュールが受けたバッファを利用する可能性のあるメッセージの総数で割って求めている．

表 5.2 にその結果を示す．スタベイション防止用のバッファの待避率は，最も大きい SP でも 0.29% と非常に小さな値となっている．また，デッドロック防止用の両バッファの待避率も，最も大きい BT で 2.9% にとどまっている．

メモリアクセスレイテンシへの平均的な影響は待避率の和と、待避のオーバーヘッドを掛け合わせた値になると考えられる．待避率の和が 3% 程度であることと，メモリに設けたバッファへのメッセージの待避および読み出しのコストはあわせて 200 ns 程度であることから，その影響は 6ns 程度と見積もることができる．一方，メモリアクセスレイテンシは数 us となっており差は大きく，スタベイションおよびデッドロック防止機構が平均

メモリアクセスレイテンシに与える影響はほぼ無視できると考えられる．

5.5 結論

本章では，一貫性制御においてメモリアクセスが有限時間内に完了することを保証するために，一貫性制御によるメッセージのやり取りによるデッドロックを防止し，また一貫性制御を無限に繰り返すことがないようにスタベイションフリーな方式の提案を行った．

スタベイションの防止は，否定応答による一貫制御の再試行をなくし，メモリが一貫性要求を受け付けた順で処理すること，そのためにメモリに要求を退避する機能を持つことを特徴とする．デッドロックの防止は特定のメッセージをメモリに退避することを特徴とし，単一チャネルからなるネットワークにも適用可能なことを特徴とする．

本機能を並列計算機 Cenju-4 に実装したところ，退避に必要なバッファのサイズはあわせて 192K バイトと小さいものとなった．また，実機でアプリケーションを実行して評価を行った結果，設計した分散共有メモリ機構はスタベイションやデッドロックを起こすことなく安定して動作し，またメモリアクセスレイテンシに与える影響がほぼないことも確認できた．

今回提案した方式は，Invalidate 型のプロトコルを対象とした方式となっている．今回実装した方式を Update 型に適用する場合，待避する要求にデータが付与されている場合があり必要なサイズが増加する問題が残る．今後，Update 型にも対応したデッドロック・スタベイション防止方式の検討を行う．

第 6 章

関連研究

6.1 分散共有メモリシステムの関連研究

分散共有メモリシステムに関する研究として，様々なシステムの提案が行われている．本論文で示した，ディレクトリ方式の大規模分散共有メモリシステムの実現において生じる，メモリアクセスレイテンシの削減，ディレクトリ保持コストの低減，メモリアクセスが有限時間内に完了することの保障，の三つの課題について，以下のシステムでの取り組みを示す．

Stanford DASH [37] は，Stanford 大学にて構築されたシステムである．最大 64 プロセッサをメッシュトポロジのネットワークで接続する．ディレクりとして Full Map 方式を採用している．ディレクトリ保持コストの低減については議論されず，システムの大規模化においてディレクトリ構成がボトルネックとなる．メモリアクセスレイテンシについては，図 5.1 に示したように，3 ステップの通信で一貫性制御が終えられるように設計され，レイテンシを抑える方式となっている．ただし，書込み要求の転送は一対一通信によって，またその応答の回収も一つづつ行われる．そのため，書込みの遅延は共有するノード数に比例する時間を要する．デッドロックは 2 チャネルのネットワークによって防止し，スタベイションの発生可能性は残ったシステムとなっている．

MIT Alewife [35] は，MIT 大学にて構築されたシステムである．最大 512 プロセッサをメッシュトポロジのネットワークで接続する．DASH 同様に書込み要求の転送は一対一通信によって，またその応答の回収も一つづつ行われる．そのため，書込みの遅延は共有するノード数に比例する時間を要する．ディレクトリ保持コストの低減のために，ディレクトリとしては LimitLESS 方式を採用している．この方式は，Limited Pointer 方式 [8] を拡張したもので，ディレクトリ情報として有限個のポインタをハードウェアとして保有し，これがあふれた場合はソフトウェアで処理する方式である．ただし，書込みの遅延について，この方式では共有数が増加するとソフトウェア制御となり，ハードウェ

アのみで実装される場合と比較して大幅に悪化する問題がある．デッドロックは 2 チャネルのネットワークによって防止し，スタベイションの発生可能性は残ったシステムとなっている．

Stanford FLASH [41] は，Stanford 大学にて DASH の後継として開発されたシステムである．DASH の課題であったディレクトリ保持コストの低減のために，ディレクトリとして Dynamic Pointer 方式を採用している．こちらも Limited Pointer 方式を拡張したもので，元々アサインされた領域に格納できなくなった場合に，空きのディレクトリエントリをハードウェアが動的に確保して活用する方式である．一方，書込みの遅延の問題については議論されず，書込み要求の転送は一対一通信によって，またその応答の回収も一つづつ行われる方式となっていた．そのため，書込みの遅延は共有するノード数に比例する時間を要する．デッドロックは 2 チャネルのネットワークによって防止し，スタベイションの発生可能性は残ったシステムとなっている．

HP Exemplar [39] と Sequent NUMAQ [40] は，ともに Scalable Coherent Interface(SCI) [32] を利用してキャッシュの一貫性を取っている．SCI では集中ディレクトリ方式ではなく，分散ディレクトリ方式を採用している．そのため，ディレクトリのサイズは小さく押さえることが出来ている．一方，メモリアクセスのレイテンシはノード数に比例した通信回数を必要とする問題がある．デッドロックは 2 チャネルのネットワークによって防止されており，スタベイションも前記分散ディレクトリ方式によって防止される．

SGI Origin2000 [38] は，SGI 社が開発したシステムであり，最大 1024 プロセッサをハイパーキューブネットワークで接続する．書込み要求の転送は一対一通信によって，またその応答の回収も一つづつ行われる．そのため，書込みの遅延は共有するノード数に比例する時間を要する．ディレクりとしてシステム規模が小さい場合は Full Map 方式を採用し，システム規模が大きくなると Coarse Vector 方式を採用している．デッドロックは 2 チャネルのネットワークによって防止される．スタベイションも否定応答はあるが再試行回数に応じた優先度制御を行うことで解消している．

JUMP-1 [42] は，日本の大学の研究グループが開発したシステムであり，最大 1024 プロセッサを独自の RDT ネットワークで接続するシステムとなる．様々なプロトコルを実装し，無効化型に加えて更新型の両方のプロトコルを採用している．ディレクトリは，RDT に合わせた階層ビットマップ方式を採用し，キャッシュラインサイズではなくページ単位で一つのエントリで管理することで，必要なサイズを大幅に抑えている．また，ネットワークによるマルチキャストとギャザー機能を実装し，書込み要求の処理時間を一定に抑えている．デッドロックとスタベイションの防止については不明である．

以上のように，本論文で議論した，高性能かつ安定動作するディレクトリ方式の一貫性制御の実現において，上記で挙げた各システムにおいて部分的な取り組みはあるものの，

課題間でのトレードオフが発生していたり，全ての課題に対応した方式を示すことができていなかった．本論文では，3 章から 5 章でトレードオフを発生させることなく三つの課題を実現可能な方式を提案した．

6.2 メモリアクセスレイテンシの削減および隠蔽の関連研究

メモリアクセスレイテンシの削減について，幾つかの関連する研究を述べる．

ディレクトリ方式の一貫性制御のプロトコルには，以下のようなものが存在する．前節で述べたように，集中ディレクトリ方式においては DASH 等 [37,38] で採用されたプロトコルにより 3 ステップの通信で完了する．Alewife [35] で採用されたプロトコルでは 4 ステップの通信を必要とする．SCI [32] で採用された分散ディレクトリでは，共有するノード数の 2 倍のステップ数の通信を必要とし，さらに大幅に悪化する問題がある．本論文のスタベイション防止で提案し Cenju-4 に採用したプロトコルは，DASH 同様に 3 ステップの通信で完了し，プロトコルの設計としては低遅延なものとなっている．

また，メモリアクセスレイテンシの削減において，ディレクトリアクセスのオーバーヘッドの問題も存在する．また，LimitLESS 方式 [28] や Dynamic Pointer 方式 [30] のディレクトリでは，前者は共有ノードが増加した場合にソフトウェア処理に移行して大幅に遅延が悪化する問題，後者は共有ノード数に比例した回数のディレクトリアクセスを必要とし，やはり書込み時のレイテンシがノード数の増加に従って悪化する問題がある．

また，書込み時のレイテンシの削減については，本論文で議論したネットワークのマルチキャストやギャザー機能を用いた方式が提案されている [10,20,21,31]．[20,21] はネットワークにおいてデッドロックが発生しないマルチキャスト手法を提案している．[10,31] は，マルチキャストに加えてギャザーを行う手法も提案している．ただし，ギャザーについてはいずれもマルチキャスト同じパスを逆に辿ることを前提とした方式となっており，様々なネットワークに適用する場合，また DASH や本論文で提案したように，マルチキャストの送信元 (ホーム) と，それに対する応答の送信先 (マスタ) が異なるケースには適用できない問題がある．

次に，メモリアクセスレイテンシを隠蔽する従来研究として，メモリコンシステンシモデル，プロセッサのアウト・オブ・オーダー実行，プリフェッチ技術が存在する．

メモリコンシステンシモデル [24] には，Sequential や Strong など，書込みのレイテンシがそのままプロセッサの待ち時間に繋がるものと，Weak や Release など，書込みアクセスによって一貫性制御の処理を実施中であってもプロセッサは継続して処理を実行可能で，メモリ同期命令が発行された時に先行する書込みアクセスの完了を待つ方式が存在する．本論文の 2 章のシミュレータや 6 章の並列計算機 Cenju-4 によるアプリケーション評価では，Weak Consistency モデルで評価を行っている．そのため，書込みによるプロ

セッサのストールはメモリアクセスレイテンシをある程度隠蔽したものとなっている．

プロセッサのアウト・オブ・オーダー実行 [44] は，プロセッサが実行時にデータの依存関係を見ながら処理可能な命令を先行して実行するものである．基本的にはプロセッサ内の同時実行命令数を向上させることによって高性能化する技術ではあるが，複数のメモリアクセス命令が前の命令の完了を待たずに実行される．複数のメモリアクセス要求が同時に処理されることでメモリアクセスレイテンシの隠ぺいが可能となっている．本論文の 2 章の評価では本機能を用いていないが，6 章の並列計算機 Cenju-4 によるアプリケーション評価では，プロセッサが発行する最大メモリアクセス要求数は 4 となり，隠ぺいが行われている．

また，読出しアクセスのレイテンシを隠蔽するものとしてプリフェッチ技術がある [25, 26]．プリフェッチ技術は実際にデータをアクセスするロード命令に先行してソフトウェア命令によってあるいはハードウェアが自動的にキャッシュにデータをフェッチする技術となる．これにより，読出しアクセスのレイテンシを隠蔽することができる．本論文ではこの機構は用いていないが，提案したメモリアクセスレイテンシ削減技術と組み合わせて利用可能である．

6.3　ネットワークの関連研究

ネットワークの遅延削減やスループット向上に関する研究として，ネットワークトポロジの研究が多く行われている．近年の HPC システムに利用されているトポロジとしては，今回評価を行った Hypercube に加えて，Fat-tree [60], Dragonfly [58], Torus などが複数のシステムで用いられている．Fat-tree，Dragonfly，Hypercube は，InfiniBand を用いたシステムで採用されている．Torus は，スイッチのポート数を抑えられるトポロジとして，特に InfiniBand ではなく専用ネットワークを構築するシステムで採用されている．また，近年，直径が小さいトポロジとして，Random Topology [59], SlimFly [61], Flattened Butterfly [57] などが提案されている．いずれも，一つのネットワーク・プレーンにおける接続関係に着目したネットワークの低遅延化および高スループット化に関連する提案である．一方，本論文の提案は，多重プレーンのネットワークを有効活用することにより低遅延化や高スループット化を実現するものである．ノード間の距離が全て 1 の完全網を除いた他のトポロジにも適用可能な技術であり，ノードによる距離の差が大きい Torus や Hypercube ネットワークで特に効果が大きいと考えられる．

本論文で提案している Multi-plane の活用についていくつかの論文が存在する．性能向上に関する論文としては [45–48] などがある．[47, 48] は複数プレーンの Fat-tree におけるスループット向上の性能評価を行っている．[46] は複数プレーンのネットワークにおいて各プレーンへのパケット割当手法の提案と評価を行っている．[45] では，InfiniBand を

用いた MPI 通信において，複数プレーンへのパケット割当を実現する MPI の設計に関する議論を行っている．これらは，各プレーンに同一のトポロジを多重化する方式に関する提案である．一方，本論文の提案は，各プレーンに同型のトポロジを採用することを新規に提案し，同一のトポロジを採用した方式よりも低遅延化や高スループット化を実現するものである．

2 つの独立した，異なるネットワーク構成を用いる研究もある．代表例としては，ハイエンドなデータセンターやスーパーコンピュータを対象にして，大きなバルクデータ転送は光サーキットスイッチネットワーク，それ以外のデータ転送は電気スイッチで構成されるネットワークを用いる研究 [53] や，ステンシル通信用に 3 次元トーラス，バリアなどの集合通信用にツリートポロジの 2 つのネットワークを有するスーパーコンピュータである BlueGene/L が挙げられる．

また，ケーブル長を課題として議論している論文として Random Topology でのケーブル長とそのコスト削減手法を提案した [62] が存在する．本論文では, 二重同型 Hypercube ネットワークを対象に，ケーブル長やそのコスト削減に加えて，ケーブル長が遅延に与える影響を考慮した Network の選定方法やその影響の評価を行っている点に違いがある．

6.4 ディレクトリの関連研究

ディレクトリのサイズを抑える方式も様々提案されている．メモリにおいて保持ノードを管理する集中ディレクトリ方式の一エントリのサイズを抑える方式として，代表的な Coarse Vector 方式 [29] 以外にも様々な形式が提案されている [8, 28, 30, 31]．また，SCI [32] で採用されているような，メモリだけでなくデータを保持しているキャッシュにも情報を保持する分散ディレクトリもある．本論文で提案したビットパターン方式は，前者のもこの一エントリに必要なサイズを抑える方式となる．

それ以外にもディレクトリに必要なサイズを抑える幾つかの方式が提案されている．ディレクトリをキャッシュラインサイズ単位で保持するのではなく 4K バイト等のページサイズ単位で保持する方式 [42] がその一つである．キャッシュラインサイズ毎に必要な情報は残るものの，ノード数に比例して増加する部分はページ単位で持つことで，必要なサイズを抑えることが可能となる．また，Sparse Directory 方式 [83] は，ディレクトリをメモリの全データに対して保有するのではなく，キャッシュされたデータに対してのみ持つ方式である．キャッシュサイズはメモリのサイズと比較して非常に小さいことから，ディレクトリに必要なサイズを小さくすることができる．これらの方式は，本論文で提案したビットパターン方式においても組合せて適用可能な技術である．

第 7 章

結論

7.1 本論文のまとめ

本論文では，前節で述べた，高性能かつ安定動作するディレクトリ方式の一貫性制御の実現における四つの課題に対応して，2 章でメモリアクセスレイテンシの削減を目的としたディレクトリ方式でのプロトコルの性能特性の解析，3 章でメモリアクセスレイテンシの削減を目的とした共有しているノード数の増加によるメモリアクセスレイテンシの悪化を抑えるディレクトリおよび一貫性制御手法，4 章でメモリアクセスレイテンシの削減を目的としたネットワークの低レイテンシ化と高スループット化，5 章で安定動作を達成するための一貫性制御においてメモリアクセスが有限時間内に完了することの保証，について議論した．

2 章では，Invalidate 型，Update 型および Competitive 型の分散共有メモリプロトコルの性能評価を行い，メモリアクセスレイテンシが処理性能に与える影響を明らかにするために，メモリアクセスに起因してプロセッサがストールした時間を測定しプロトコル間で比較を行った．その結果，Invalidate 型と比較して，Update 型はワークロードの性質によりその振舞を大きく変化させることが明らかとなった．一方，Competitive 型は更新回数の上限値を 2，3 に設定した場合に，どのワークロードでも良好な性能を発揮することが明らかとなった．Invalidate 型と比較して，Update 型では読出しに起因するストール時間は大幅に削減できているものの，多くのワークロードではその削減効果を超えて書込みに起因するストール時間が増加し，プロセッサストール時間全体の増加に繋がることが確認できた．Competitive 型は，更新回数の上限値を 2 あるいは 3 に設定した場合に，Update 型よりは効果は低いものの読出しに起因するストール時間を削減し，Update 型で問題となっている書込みに起因するストール時間の増加を抑え，いずれのワークロードでも全体のストール時間の削減が達成できた．Update や Competitive 型を用いることによる，プロセッサの読出しに起因するストール時間の削減は，読出し要求率および書き戻

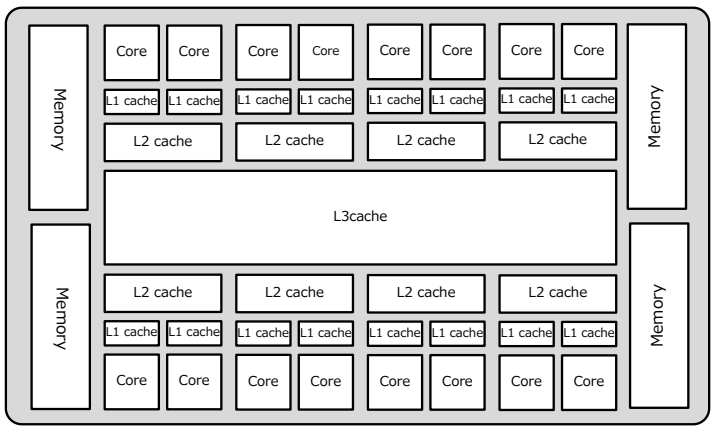
し要求率の減少から得られていることも明らかとなった．Competitive 型では，読出し要求率の減少効果が少ないワークロードでも，書き戻し要求率の効果が大きく現れ，ストール時間が削減された．Update 型を用いることにより，プロセッサの書込みに起因するストール時間の増加は，書込み要求率の増加と平均ライト分配数の増加に原因があることが確認された．Competitive 型は，それらの増加，特に後者の平均ライト分配数の増加を抑えてストール時間の悪化を抑制した．スヌープ方式では読出し要求率や書込み要求率に依存してストール時間が変化するのに対して，ディレクトリ方式では書き戻し要求率や平均ライト分配数にも影響を受けて変化するため，スヌープ方式では効果のなかった Competitive 型が有効に機能することが明らかとなった．

3 章では，書込み要求の一貫性制御による負荷増大とメモリアクセスレイテンシの悪化を抑えるビットパターン方式のディレクトリと，汎用的なマルチキャスト・ギャザー機能を実現するネットワークの提案をおこなった．ビットパターン方式は，ノードを階層的に分割して管理し各階層による分割サイズが異なることを特徴とする．評価の結果から，ビットパターン方式のこの特徴が有効に機能し，様々な利用シーンにおいて保持ノードを効率よく管理することができる方式であることが確認できた．システムを構成する全てのノードを利用するアプリケーションのケースでは，階層的に分割しない Coarse Vector 方式が最も優れているものの，提案するビットパターン方式が次いで保持している可能性のあるノード数が大幅に増加するのを抑制できることを確認した．一方，設計で目標として，大規模システムの一部のノードを利用してアプリケーションを動作させるケースでは，階層的な分割が有効に機能し，保持している可能性のあるノード数が大幅に増加するのを抑制できることを示した．提案した汎用的なマルチキャスト・ギャザー方式は，従来提案されていた手法の以下の二つの制約を解消することを目的として設計した．一つ目の制約は，従来のマルチキャスト方式はネットワーク構造に合わせたマルチキャスト宛先指定によって実現され，その宛先指定に合わせたディレクトリ方式を採用する必要があった点である．この制約によって，保持している可能性のあるノード数が大幅に増加し一貫性制御のオーバーヘッドが増加してメモリアクセスレイテンシが悪化する．また，二つ目の制約は，適用可能なネットワークトポロジに制約がある点である．提案したマルチキャスト方式は，ネットワークの各スイッチがディレクトリ方式から宛先を算出してマルチキャストを行うことを特徴とする．これにより，ネットワーク構造に合わせることなく効率的なディレクトリ方式を採用することが可能となる．また提案したギャザー方式は，送出先ノードがマルチキャストの宛先情報からネットワーク内での待ち合わせパターンを算出し，そのパターンに応じて各スイッチがメッセージを集約することを特徴とする．この特徴により，様々なネットワークトポロジに適用可能となる．本機構を実装した並列計算機 Cenju-4 でメモリアクセスレイテンシを評価した結果，128 ノードの分散共有メモリシステムにおいても書込みに起因するメモリアクセスレイテンシを，ノード数にかかわらず一

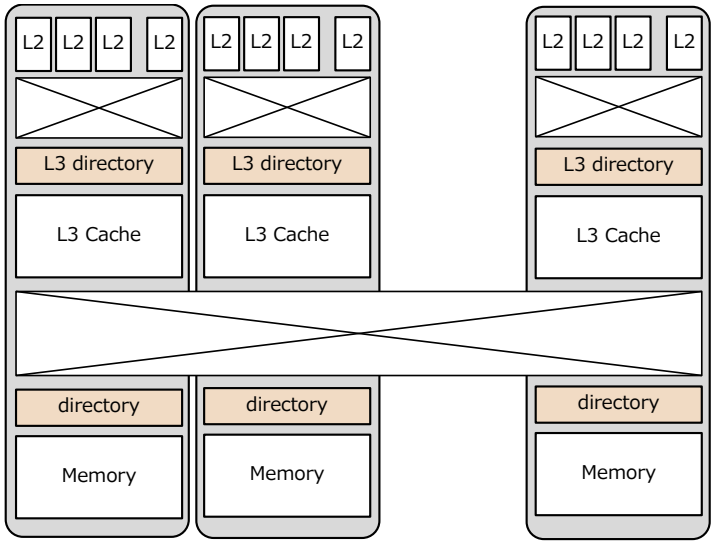
定の値に抑える効果を持つことが確認された．

4 章では，一貫性制御の遅延に大きな影響を及ぼすネットワークの高性能化方式の提案を行った．近年多数のシステムで採用されている，複数のネットワーク・プレーンを持つ Multi-Plane 方式を対象に，二重同型ネットワークを提案し，Hypecube ネットワークおよび Folded-Hypercube ネットワークを対象として，その遅延とスループットの評価を行った．この二重同型ネットワークは，二つのネットワーク・プレーンを保有するシステムにおいて，各プレーンをグラフ同型のネットワークで接続することを特徴とする．同型ネットワークの候補は複数存在するため，二つのネットワーク選定法を示した．一つは，ネットワークの評価指標として良く用いられる平均最短距離と，スループットに相当する全対全最大トラフィックの両指標を改善する同型ネットワークの選定法である．もう一つは，遅延指標として平均最短距離だけでなくスイッチ間のケーブル遅延も考慮した平均最短遅延と，全体全最大トラフィックの両指標を改善する同型ネットワークの選定法である．グラフ解析やサイクルレベルのシミュレーションの評価結果から，後者の選定法が遅延・スループット・経済的コストの観点から有効な選定法であることを確認した．両手法ともスループットについては改善効果を示し，両者に大きな差がないことを確認した．遅延についても 8 次元程度までの Hypercube であればどちらの方法でも同程度に改善できることも確認した．しかし，ネットワーク・サイズが大きくなりシステム規模が大きくなると，平均最短距離で最適化する手法ではケーブル長が伸びる影響を受けてケーブル遅延が増加し，遅延が逆に悪化することも明らかとなった．一方，平均最短遅延で最適化する手法であればケーブル遅延を指標に組み込んだことからそれを改善できることも確認した．経済的コストにおいても，遅延同様に，ネットワーク・サイズが大きくなると平均最短距離で最適化した場合はケーブル長が伸びることからコストが増加するが，平均最短遅延で最適化するとコストの増加を防ぐことができることも確認できた．

5 章では，一貫性制御によるメッセージのやり取りによるデッドロックを防止し，また一貫性制御を無限に繰り返すことがないスタベイションフリーな方式の提案を行った．スタベイションの防止は，否定応答による一貫制御の再試行をなくし，メモリが一貫性要求を受け付けた順で処理すること，そのためにメモリに要求を退避する機能を持つことを特徴とする．デッドロックの防止は特定のメッセージをメモリに退避することを特徴とし，単一チャネルからなるネットワークにも適用可能なことを特徴とする．これらの機能の実装に必要なバッファサイズは小さく抑えられることを示した．また，並列計算機 Cenju-4 に実装しアプリケーションを実行した評価で，設計した分散共有メモリ機構はスタベイションやデッドロックを起こすことなく安定して動作し，またメモリアクセスレイテンシに与える影響が小さいことを確認した．



(a) マルチコアプロセッサ



(b) 複数のマルチコアプロセッサを接続したマルチプロセッサシステム

図 7.1　マルチコアプロセッサによる分散共有メモリシステム

7.2　今後の課題

2 章や 5 章で行った研究後も，分散共有メモリシステムの研究や活用は幅広く行われている．その中で，まず大きなシステム構成の変化として，プロセッサのマルチコア化が挙げられる．以降，それに関連する今後の課題について述べる．

7.2.1 マルチコア化に対応した分散共有メモリシステム

従来は，単一のコアとそのコアに紐づいた 1 次および 2 次キャッシュからプロセッサは構成されていた．しかし，近年プロセッサの発熱の問題からマルチコア化が進展し，数コアから数十コアからなるプロセッサが一般化した [77] [75] [76] [78] [79]．図 7.1 にマルチコアプロセッサで構成される分散共有メモリシステムの模式図を示す．図 7.1(a) に示すように，マルチコアプロセッサは，コアとそれに紐づいた 1 次キャッシュ，数コアで共有される 2 次キャッシュ，さらに全コア共有される 3 次キャッシュからなる多階層のキャッシュで構成される．このようなプロセッサにおいては，3 次キャッシュと 2 次キャッシュの間はループやメッシュなどのトポロジでチップ内で接続される構成を取り，3 次キャッシュにどの 2 次キャッシュがデータを保持するかを管理するディレクトリを持つものが存在する [75] [78].

またそのようなプロセッサを複数接続するインターコネクトの規格として Intel Quick Path [73] [74] が存在する．このインターコネクトにおいては，メモリ管理モジュールにおいてディレクトリを保持する home snoop 方式が定義されている．即ち，図 7.1(b) に示すように，メモリ及び 3 次キャッシュにおいてそれぞれ上位階層のどのキャッシュがデータを保持しているかを管理するディレクトリを保有する構成を取っている．このような階層ディレクトリの研究が [81] [82] で行われている．またこのようなシステムにおいて，メモリのディレクトリ情報をプロセッサチップ上に保有するための技術として，キャッシュされているメモリのブロックのディレクトリのみを保持する Sparse Directory 方式 [83] が幅広く採用されている．このような階層的な管理によってディレクトリの保持コストは大幅に削減され，現時点では許容されるコストで実現可能となっている．

今後，このようなシステムを前提としてプロセッサ内のコア数が増加，また接続するプロセッサ数の増加に対して有効な共有メモリプロトコルの設計やディレクトリ形式の設計が重要になると考えられる．

例えば，メモリにおける Sparse Directory の最適な設計はその課題の一つである．Sparse Directory 方式ではその容量に制限があるため，溢れた場合に置き換えを行い置き換えられたブロックのキャッシュを全て無効化する必要が生じる．そのため，ディレクトリの総容量を一定とした場合に，どれだけの数のブロックを保持できるかと各エントリのディレクトリサイズがトレードオフの関係となる．そのため，最適な設計ポイントを明らかとする研究を今後行っていく必要があると考えられる．

また，階層的なネットワーク構成となるため，本論文で議論したメモリからキャッシュへ転送される要求も階層を辿りながら転送されていくこととなる．そのため，そのような動作によるメモリアクセスレイテンシの増大を抑えるプロトコルの設計が重要になってく

ると考えらえれる．この問題に対応する方式として，DiCo-CMP [84] ではメモリを経由せずに投機的にデータを保持している可能性のあるキャッシュに要求を送信する方式や，Intel Quick Path で採用されている source snoop 方式のように全てのキャッシュに要求を転送する方式などが提案されている．一方このような方式は大規模化に対してスケールしにくいという問題をもっているため，それらを解決するプロトコルの研究も必要があると考えられる．

さらに，このような各種方式の検討を進めるうえで，本論文で性能評価に用いた科学技術計算用途のワークロードに加えて，近年利用が進んでいるデータ分析や機械学習応用を用いた評価も重要になってくると考えられる．

謝辞

懇切なる御指導，御助言と格別なる御配慮を賜りました大阪大学サイバーメディアセンター下條真司教授に謹んで感謝の意を表します．

本論文をまとめるにあたり，貴重な時間を割いて頂き，懇切なる御指導と有益な御助言を賜りました，大阪大学大学院情報科学研究科マルチメディア工学専攻の藤原融教授，鬼塚真教授，原隆浩教授，松下康之教授，春本要教授，伊達進准教授に心より感謝の意を表します．

本論文第 2 章で扱った分散共有メモリプロトコルの性能評価に関する研究について，1992 年 1994 年，京都大学大学院情報科学研究科修士課程在学時に，多大なる御支援，御助言を頂きました富田眞治教授 (当時．現，京都情報大学院大学副学長)，中島浩助教授 (当時．後，京都大学学術情報メディアセンターコンピューティング研究部門教授，2021 年 10 月逝去)，森眞一郎助手 (当時．現，福井大学大学院工学研究科情報・メディア工学専攻教授)，またたびたび討論頂いた同研究室の諸氏に感謝いたします．

本論文第 3 章で扱ったビットパターン方式のディレクトリと汎用的なマルチキャスト・ギャザー機能による書込みレイテンシの削減，第 5 章で扱ったデッドロック・スタベイションフリーなプロトコルの二つの研究について，1994 年-2000 年，NEC C&C メディア研究所在籍時に，多大なる御支援，御助言を頂きました加納健氏，広瀬哲也氏，中村真章氏，中田登志之氏，ならびに同研究所の諸氏に感謝いたします．

本論文第 4 章で扱った二重同型化によるネットワークを高性能化に関する研究について，多大なる御支援，御助言を頂きました，国立情報学研究所アーキテクチャ科学研究系鯉渕道紘准教授，広島大学情報科学部安戸僚汰特任助教に感謝いたします．

本研究を推進するにあたり，多大なる御指導，御助言を頂きました，大阪大学サイバーメディアセンター応用情報システム研究部門木戸善之講師，また同研究部門の諸氏に深く感謝いたします．

本研究を推進するにあたり，多大なる御指導，御助言を頂いた NEC 研究所の諸氏に深く感謝いたします．

最後に，本論文執筆にあたり支援いただいた家族に感謝いたします．

参考文献

[1] HPE, “ HPE Superdome Flex280,” https://community.hpe.com/t5/Servers-Systems-The-Right/HPE-s-new-as-a-service-building-block-for-digital-transformation/ba-p/7111328.

[2] A. Biswas, “ Sapphire Rapids,” IEEE Hot Chips Symposium, pp. 1–22, 2021.

[3] IBM, “ IBM Power E1080,” https://www.ibm.com/jp-ja/products/power-e1080

[4] J. K. Archibald,“ The Cache Coherence Problem in Shared-Memory Multiprocessors,” PhD thesis, Department of Computer Science, University of Washington, 1987.

[5] J. Archibald and J.L. Baer, “ Cache Coherence Protocols: Evaluation Using Multiprocessor Simulation Mode,” ACM Transactions on Computer Systems, pp. 273–298, 1986.

[6] D. Chaiken, C. Fields, K. Kurihara and A. Agarwal, “ Directory-based cache coherence in large-scale multiprocessors,” Computer, vol. 23, no. 6, pp. 49–58, 1990.

[7] A.Gupta and W.D.Weber,“ Cache Invalidation Patterns in Shared-Memory Multiprocessors,” IEEE Transactions on Computers, vol.41, no.7, pp. 794–810, 1992.

[8] A.Agarwal, R.Simoni, J.Hennessy and M.Horowitz, “ An Evaluation of Directory Schemes for Cache Coherence,” International Symposium on Computer Architecture, pp. 373, 1988.

[9] C. Holt, M. Heinrich, J. Singh, E. Rothberg and J. Hennessy, “ The Effects of Latency and Occupancy in Distributed Shared Memory Multiprocessors,” CSLTR-95-660, Stanford University, 1995.

[10] D. Dai and D. K. Panda, “ Reducing cache invalidation overheads in wormhole routed DSMs using multidestination message passing,” ICPP Workshop on

Challenges for Parallel Processing, vol.1, pp. 138–145, 1996.

[11] S.J.Eggers and R.H.Katz, “ A Characterization of Sharing in Parallel Programs and its Aplication to Coherency Protocol Evaluation,” International Symposium on Computer Architecture, pp. 373–382, 1988.

[12] A.R.Karlin, M.S.Manasse, L.Rudolph and D.D.Sleator, “ Competitive Snoopy Caching,” Symposium on Foundations of Computer Science, pp. 244–254, 1986.

[13] S. J. Eggers and R. H. Katz, “ EVALUATING THE PERFORMANCE OF FOUR SNOOPING CACHE COHERENCY PROTOCOLS,” ACM Transactions on Computer Systems, pp. 2–15, 1989.

[14] M.S.Paramarcos and J.H.Patel, “ A low overhead coherence solution for multiprocessors with private cache memories,” International Symposium on Computer Architecture, pp. 348–354, 1984.

[15] C. Thacker, L. Stewart and E. Satterthwaite, Jr, “ Fierfly: A Multiprocessor Workstation,” IEEE Transactions on Computers, Vol. 37, No. 8, pp. 909–920, 1984.

[16] A.S. Tanenbaum, “ Computer Networks,” 2nd ed., Prentice-Hall, 1988.

[17] S.V.Adve and M.D.Hill, “ Weak Ordering - A New Definition,” International Symposium on Computer Architecture, pp. 2–14, 1990.

[18] J.P.Singh, W.D.Weber and A.Gupta, “ SPLASH:Stanford Parallel Applications for Shared-Memory,” Techhical Report CSL-TR-91-469, Stanford Univ, 1991.

[19] G. Blanck and S. Krueger, “ The SuperSPARC microprocessor,” Digest of Papers COMPCON Spring 1992, pp. 136–141, 1992.

[20] M. P. Malumbres, J. Duato and J. Torrellas, “ An efficient implementation of tree-based multicast routing for distributed shared-memory multiprocessors,” IEEE Symposium on Parallel and Distributed Processing, pp. 186–189, 1996.

[21] X. Lin and L. M. Ni, “ Multicast communication in multicomputer networks,” IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 10, pp. 1105–1117, 1993.

[22] H.C. Hsiao and C.T. King, “ Does multicast communication make sense in write invalidation traffic?,” International Conference on Parallel and Distributed Systems, pp. 221–228, 2000.

[23] H. Nishi, K. Nishimura, K. Anjo, T. Kudoh and H. Amano, “ The JUMP-1 router chip: a versatile router for supporting a distributed shared memory,” IEEE International Phoenix Conference on Computers and Communications, pp. 158–164, 1996.

[24] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta and J. Hennessy, " Memory consistency and event ordering in scalable shared-memory multiprocessors," International Symposium on Computer Architecture, pp. 15–26, 1990.

[25] T.F. Chen and J.L. Baer, " A performance study of software and hardware data prefetching schemes," 21th International Symposium on Computer Architecture, pp. 223–232, 1994.

[26] S. Byna, Y. Chen and X. Sun, " A Taxonomy of Data Prefetching Mechanisms," International Symposium on Parallel Architectures, Algorithms, and Networks, pp. 19–24, 2008.

[27] L. M. Censier and P. Feautrier, " A New Solution to Coherence Problems in Multicache Systems.," IEEE Transactions on Computers, vol. C27, pp. 1112–1118, 1978.

[28] D. Chaiken, J. Kubiatowicz and A. Agarwal, " LimitLESS Directories: A Scalable Cache Coherence Scheme.," International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 224–234, 1991.

[29] A. Gupta, W.D. Weber and T. Mowry: " Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes," International Conference on Parallel Processing, pp. 148–159, 1990.

[30] R. Simoni and M. Horowitz, " Dyanmic Pointer Allocation for Scalable Cache Coherence Directories.," International Symposium on Shared Memory Multiprocessing, pp. 72–81, 1991.

[31] T. Kudoh, H. Amano, T. Matsumoto, K. Hiraki, Y. Yang, K. Nishimura, K. Yoshimura, Y. Fukushima, " Hierarchical bit-map directory schemes on the RDT interconnection network for a massively parallel processor JUMP-1," International Conference on Parallel Processing, pp. I-186–I-193, 1995.

[32] " IEEE Std 1596-1992, IEEE Standard for Scalable Coherent Interface," Institute of Electrical and Electronics Engineers, Inc., 1993.

[33] Y. Kanoh, M. Nakamura, T. Hirose, T. Hosomi, H. Takayama and T. Nakata, " Message Passing Communication in a Parallel Computer Cenju-4," International Symposium on High Performance Computing, pp. 55–70, 1999.

[34] " MIPS R10000 Microprocessor User's Manual Version 2.0," MIPS Technologies, Inc., 1996.

[35] A. Agarwal, R. Bianchini, D. Chaiken, K. L. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie and D. Yeung," MIT Alewife Machine: Architecture and Performance.," International Symposium on Computer Architecture, pp. 1112–

1118, 1978.

[36] J. D. Kubiatowicz, “ Integrated Shared-Memory and Message-Passing Communication in the Alewife Multiprocessor,” PhD thesis, Massachusetts Institute of Technology, 1998.

[37] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta and J. Hennesy, “ The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor,” International Symposium on Computer Architecture, pp. 148–159, 1990.

[38] J. Laudon and D. Lenoski,“ The SGI Origin: A ccNUMA Highly Scalable Server,” International Symposium on Computer Architecture, pp. 241–251, 1997.

[39] T. Brewer and G. Astfalk, “ The evolution of the HP/Convex Exemplar,” IEEE COMPCON 97. Digest of Papers, pp. 81–86, 1997.

[40] T. Lovett and R. Clapp, “ STiNG: A CC-NUMA Computer System for the Commercial Marketplace,” International Symposium on Computer Architecture, pp. 308–317, 1996.

[41] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum and J. Hennessy, “ The Stanford FLASH Multiprocessor,” International Symposium on Computer Architecture, pp. 302–313, 1994.

[42] T. Matsumoto, T. Kudoh, E. Nishimura, K. Hiraki, H. Amano and H. Tanaka, “ Distributed Shared Memory Architecture for JUMP-1 a general-purpose MPP prototype,” International Symposium on Parallel Architectures, Algorithms and Networks, pp. 131–137, 1996.

[43] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan and S. K. Weeratunga, “ The NAS parallel benchmarks—summary and preliminary results,” ACM/IEEE conference on Supercomputing, pp. 158–165, 1991.

[44] R. M. Tomasulo, “ An Efficient Algorithm for Exploiting Multiple Arithmetic Units,” IBM Journal of Research and Development, vol. 11, no. 1, pp. 25–33, 1967.

[45] J. Liu, A. Vishnu and D.K. Panda, “ Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation,” ACM/IEEE Conference on Supercomputing, pp. 33–44, 2004.

[46] S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie and L. Gurvits, “ Using multirail networks in high-performance clusters,” IEEE International Conference on

Cluster Computing, pp. 15–24, 2001.

[47] N. Wolfe, M. Mubarak, N. Jain, J. Domke, A. Bhatele, C.D. Carothers and R.B. Ross, " Preliminary Performance Analysis of Multi-rail Fat-tree Networks," IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 258–261, 2017.

[48] N. Jain, A. Bhatele, L.H. Howell, D. Böhme, I. Karlin, E.A. León, M. Mubarak, N. Wolfe, T. Gamblin and M.L. Leininger, " Predicting the Performance Impact of Different Fat-tree Configurations," International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–13, 2017.

[49] J. Duato and S. Yalamanchili and L. Ni, " Interconnection Networks: an engineering approach," Morgan Kaufmann, 2002.

[50] W. Dally and B. Towles, " Principles and Practices of Interconnection Networks," Morgan Kaufmann Publishers Inc., 2003.

[51] W.J. Dally, " Performance Analysis of k-ary n-cube Interconnection Networks," IEEE Transactions on Computers, vol. 39, pp. 775–785, 1990.

[52] J. Duato, " A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks," IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 10, pp. 1055–1067, 1995.

[53] , K.J. Barker, A. Benner, R. Hoare, A. Hoisie, A.K. Jones, D.K. Kerbyson, D. Li, R. Melhem, R. Rajamony, E. Schenfeld, S. Shao, C. Stunkel and P. Walker, " On the Feasibility of Optical Circuit Switching for High Performance Computing Systems," ACM/IEEE Conference on Supercomputing, pp. 16–16, 2005.

[54] L.N. Bhuyan and D.P. Agrawal, " Generalized Hypercube and Hyperbus Structures for a Computer Network," IEEE Transactions on Computers, vol. 33, no. 4, pp. 323–333, 1984.

[55] A.El-Amawy and S.Latif, " Properties and performance of folded hypercubes," IEEE Transactions on Parallel and Distributed Systems, vol. 2, no. 1, pp. 31–42, 1991.

[56] M. Miller and J. Siran, " Moore graphs and beyond: A survey of the degree/diameter problem," Electronic Journal of Combinatorics, vol. 61, pp. 1–63, 2005.

[57] J. Kim, W. J. Dally and D. Abts," Flattened Butterfly: A Cost-efficient Topology for High-radix Networks," International Symposium on Computer Architecture, pp. 126–137, 2007.

[58] J. Kim, W. J. Dally, S. Scott and D. Abts, " Technology-Driven, Highly-Scalable Dragonfly Topology," International Symposium on Computer Architecture, pp.

77–88, 2008.

[59] M. Koibuchi, H. Matsutani, H. Amano, D. F. Hsu and H. Casanova, “ A case for random shortcut topologies for HPC interconnects,” International Symposium on Computer Architecture, pp. 177–188, 2012.

[60] C. E. Leiserson, “ Fat-trees: universal networks for hardware-efficient supercomputing,” IEEE Transactions on Computers, vol. 34, no. 10, pp. 892–901, 1985.

[61] M. Besta and T. Hoefler, “ Slim fly: A cost effective low-diameter network topology,” International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 348–359, 2014.

[62] M. Koibuchi, I. Fujiwara, H. Matsutani and H. Casanova, “ Layout-conscious Random Topologies for HPC Off-chip Interconnects,” International Symposium on High-Performance Computer Architecture, pp. 484–495, 2013.

[63] A.R.Curtis and T. Carpenter and M. Elsheikh and A.Lopez-Ortiz and S. Keshav, “ REWIRE: An optimization-based framework for unstructured data center network design,” International Conference on Computer Communications, pp. 1116–1124, 2012.

[64] J. Mudigonda and P. Yalagandula and J.C. Mogul, “ Taming the flying cable monster: a topology design and optimization framework for data-center networks,” USENIX conference on USENIX annual technical conference, p. 8, 2011.

[65] “ Top 500 Supercomputer Sites,” http://www.top500.org/.

[66] “ Deploying HPC Cluster with Mellanox InfiniBand Interconnect Solutions,” http://www.mellanox.com/related-docs/solutions/deploying-hpc-cluster-with-mellanox-infiniband-interconnect-solutions-archive.pdf.

[67] “ SB7700 InfiniBand EDR 100Gb/s Switch System,” http://www.mellanox.com/related-docs/prod_ib_switch_systems/pb_sb7700.pdf.

[68] “ NASA Peiades Supercomputer,” https://www.nas.nasa.gov/hecc/resources/pleiades.html.

[69] T. Papatheodore, “ Summit System Overview,” https://www.olcf.ornl.gov/wp-content/uploads/2018/05/Intro_Summit_System_Overview.pdf.

[70] S. Matsuoka, “ TSUBAME3 and ABCI: Supercomputer Architectures for HPC and AI/BD Convergence,” http://on-demand.gputechconf.com/gtc/2017/presentation/S7813-Matsuoka-scalable.pdf.pdf.

[71] “ HPE SGI 8600 System,” https://www.hpe.com/jp/ja/product-catalog/detail/pip.hpe-sgi-8600-system.1010032504.html.

[72] " COLFAX DIRECT," https://colfaxdirect.com/store/pc/home.asp.

[73] D. Ziakas, A. Baum, R. A. Maddox and R. J. Safranek, " Intel QuickPath Interconnect Architectural Features Supporting Scalable System Architectures," IEEE Symposium on High Performance Interconnects, pp. 1–6, 2010.

[74] " An Introduction to the Intel QuickPath Interconnect," https://www.intel.co.jp/content/www/jp/ja/io/quickpath-technology/quick-path-interconnect-introduction-paper.html.

[75] A. Sodani, " Knights landing (KNL): 2nd Generation Intel Xeon Phi processor," IEEE Hot Chips Symposium, pp. 1–24, 2015.

[76] I. Anati, D. Blythe, J. Doweck, H. Jiang, W. Kao, J. Mandelblat, L. Rappoport, E. Rotem and A. Yasin, " Inside 6th gen Intel Core: New microarchitecture code named skylake," IEEE Hot Chips Symposium, pp. 1–39, 2016.

[77] S. Undy, " Poulson: An 8 core 32 nm next generation Intel Itanium processor," IEEE Hot Chips Symposium, pp. 1–22, 2011.

[78] R. Singhal, " Inside Intel Core microarchitecture (Nehalem)," IEEE Hot Chips Symposium, pp. 1–25, 2008.

[79] D. Suggs, M. Subramony and D. Bouvier, " The AMD Zen 2 Processor," IEEE Micro, vol. 40, no. 2, pp. 45–52, 2020.

[80] L. Han, Jianfeng An, D. Gao, X. Fan, X. Ren and T. Yao, " A survey on cache coherence for tiled many-core processor," IEEE International Conference on Signal Processing, Communication and Computing, pp. 114-118, 2012.

[81] M. E. Acacio, J. Gonzalez, J. M. Garcia and J. 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.

[82] Milo M. K. Martin, Mark D. Hill and Daniel J. Sorin, " Why on-chip cache coherence is here to stay," Communications of the ACM, vol. 55, no. 7, pp. 78–89, 2012.

[83] A. Gupta, W.D. Weber and T. C. Mowry," Reducing memory traffic requirements for scalable directory-based cache coherence schemes," International Conference on Parallel Processing, pp. 312–321, 1990.

[84] A. Ros, M.E. Acacio and J.M. Garca, " DiCo-CMP: Efficient Cache Coherency in Tiled CMP Architectures, " IEEE International Symposium on Parallel and Distributed Processing, pp. 1–11, 2008.

[85] CXL Consortium, " Compute Express Link 1.1 Specification," http://www.computeexpresslink.org/.

[86] CXL Consortium, “ Compute Express Link 2.0 Specification,” http://www.computeexpresslink.org/.

[87] Gen Z Consortium, “ Gen Z overview,” https://genzconsortium.org/.

[88] 細見 岳生, 安戸 僚汰, 鯉渕 道紘, 下條 真司, “ 二重同型 Hypercube ネットワーク,” 情報処理学会論文誌コンピューティングシステム, vol. 14, no. 3, pp. 1–13, 2021.

[89] 細見 岳生, 加納 健, 中村 真章, 広瀬 哲也, 中田 登志之, “ 並列計算機 Cenju-4 の分散共有メモリ機構,” 情報処理学会論文誌, vol. 41, no. 5, pp. 1400–1409, 2000.

[90] 細見 岳生, 森 眞一郎, 中島 浩, 富田 眞治, “ ディレクトリ型キャッシュコヒーレンスプロトコルの性能評価,” 情報処理学会論文誌, vol. 37, no. 2, pp. 1265–1275, 1996.