クラスタ型プロセッサのための分散投機メモリフォワーディング

入江英嗣[†1,†2] 服部直也[†1,†3] 高田正法[†1]
坂井修一[†1] 田中英彦[†1,†4]

クラスタ型アーキテクチャでは, 実行コアを複数のクラスタに分散し, 広い実行幅と高クロック動作の両立を目指している. クラスタ型アーキテクチャを前提とした様々な要素技術の研究が行われる一方で, 分散処理の難しいメモリ参照命令がボトルネックとなることが指摘されている. 本研究では, クラスタ型スーパスカラ・プロセッサのメモリ参照処理に着目し, 依存予測を用いたオーバヘッド隠蔽手法を提案する.

Distributed Speculative Memory Forwarding for Clustered Superscalar Processors

Hidetsugu Irie,[†1,†2] Naoya Hattori,[†1,†3] Masanori Takada,[†1]
Shuichi Sakai[†1] and Hidehiko Tanaka[†1,†4]

Clustered microarchitecture aims at coexistence of wider execution width and higher clock rate by distributing execution core to clusters. While research of various techniques on the clustered microarchitecture is performed, it is pointed out that memory instructions are hard to distribute and become a bottleneck. In this paper, memory instructions processing of a clustered superscalar processor are focused and the technique that conceals memory reference overhead by using dependence prediction is proposed.

1. 序　　論

情報処理の中核となるマイクロプロセッサには常に性能向上が期待されている. マイクロプロセッサは, デバイス微細化, スーパパイプライニング技術, 並列実行技術によって性能を向上させてきた. しかし, 現行のスーパスカラ方式は, 肥大したデータパスに配線遅延が大きく影響することが予想され, 性能向上の限界が指摘されている[1].

この問題に対し, 次世代プロセッサの選択肢として注目されている方式がプロセッサのクラスタ化である[3),11)]. クラスタ型プロセッサでは実行コアを複数のクラスタに分割し, 発行キュー, レジスタ, データパス等を局所化する（**図 1**）. 処理が複数クラスタに分散することによってオーバヘッドが生ずるが, タイミング・クリティカルパスとなっていたユニットが小型化し, 更なるスーパパイプライニング効果が期待できる.

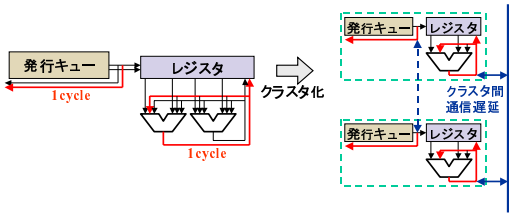


図 1　クラスタ型実行コア
Fig. 1　Clustered Execution Core

このような利点を背景にスーパスカラ・プロセッサをクラスタ化した時, クラスタ化による恩恵を受けないメモリ参照処理の遅延は相対的に大きくなることが

†1 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, University of Tokyo
†2 現在, 科学技術振興機構
Presently with Japan Science and Technology Agency
†3 現在, 日立製作所中央研究所
Presently with Hitachi, Ltd., Central Research Laboratory
†4 現在, 情報セキュリティ大学院大学
Presently with Institute of Information Security

予想される．メモリ参照遅延は現行のスーパスカラ・プロセッサでも主要なオーバヘッドとなっており，この影響を無視してクラスタ化の有効性は議論できない．

本研究では高クロック指向のクラスタ型スーパスカラ・プロセッサを想定し，メモリ参照処理オーバヘッドの増加について議論する． また，このオーバヘッドの隠蔽手法として，投機メモリフォワーディング技術[9),10),15)] をクラスタ型プロセッサに適用した “分散投機メモリフォワーディング” を提案する．提案手法では，メモリ依存予測と動的ステアリングを利用して，参照をクラスタ内に局所化する．

以下，本論文の構成は次のようになっている．第 2 節では関連研究についてクラスタ型アーキテクチャとメモリ参照処理の二つの点から紹介する．第 3 節ではクラスタ型スーパスカラ・プロセッサにおけるメモリ参照処理のオーバヘッドについて議論する．第 4 節では，“分散投機メモリフォワーディング” を提案する．ここでは，メモリ参照を局所化するための戦略と，高速性を損なわない実装方法について述べる．第 5 節では提案手法の適用率と，実行性能への影響を評価する．第 6 節でまとめを述べる．

2. 関 連 研 究

2.1 クラスタ型スーパスカラ・プロセッサ

一般に実行並列幅を増加させると，回路の複雑さが増し，動作クロックは低下する．プロセッサ設計時には，双方の最適なバランスを考慮しなければならない．Palacharla ら[11)] は，演算器のフォワーディングデータパスや発行キューについて回路レベルの解析を行い，デバイス技術が進むほど，実行幅を増やすことによるタイミング・クリティカルパスの増加が大きくなることを示している．

このため，小さな発行幅の実行クラスタを複数接続して構成された，クラスタ型のプロセッサ構成が注目されている．現行のプロセッサにおける例として，DEC Alpha 21264[8)] では，実行コアを 2 つのクラスタに分割し，高クロック化を図っている．各クラスタには物理レジスタと 2 つの演算器，局所化されたデータパスが備わっている．近年では，データパスだけでなく発行キューも局所化されたクラスタ型モデルが，要素技術のベースライン・プロセッサとして利用されるようになってきた[2),12)] ．

クラスタ型プロセッサでは，各クラスタは高速に動作するが，依存のある命令が異なるクラスタに割り当てられた場合，クラスタ間の通信遅延がオーバヘッドとなる．また，クラスタの負荷に偏りが生じると実行幅の活用が妨げられる可能性がある．分散された発行キューを持つ方式では，発行前に実行クラスタを選択する必要があり，この選択を行うステアリングロジックが性能を大きく左右する．プロセッサのポテンシャルを活用するためのステアリング手法の研究は様々に行われている[4),7),13),17),18)] ．

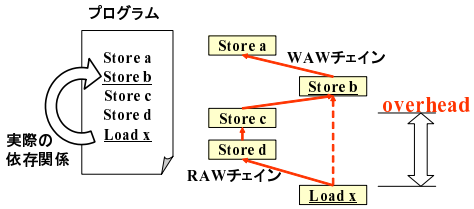


図 2 曖昧なメモリ依存のチェイン
Fig. 2 Ambiguous Dependence Chain of Memory Instruction

2.2 メモリ参照処理の高速化

ロード命令は通常のレジスタ演算命令に比べてステージ数が長く，更に，曖昧なメモリ依存関係の影響を受け，先行するストア命令が発行するまで発行できないため大きなボトルネックとなっている．このため，依存関係に注目してメモリ参照の高速化を目指す手法が提案されている．

2.2.1 依 存 予 測

メモリ参照命令の発行は，依存のあるストア命令を追い越しては行われない．しかし，メモリ参照命令同士の依存関係は実行ステージまで判明しないため，フロントエンド処理における発行条件解析では，予防的に全てのメモリ参照命令同士に依存関係を仮定しなければならない．この結果，**図 2** のような発行条件のチェインが形成される． 図 2 では，ロード命令は実際には Store b にのみ依存があり，Store b が発行された後であれば，正しい処理を行うことができる．しかし，その情報はフロントエンド時点では判明しないため，ロード命令の発行は Store a から始まるチェインが順番に解決する事を待たなければならない．実際には，多くのロード命令は実行中のストア命令に依存関係がないことが知られており，不必要な待機によってロード命令処理の遅延が増やされている．

DEC Alpha 21264[8)] では，ロード命令は基本的に実行中のストア命令と依存がないとして，先行するストア命令の有無に関わらず，オペランドが揃い次第投機的に発行される．依存関係のあるストア命令を不当に追い越していたことが判明した場合はパイプライン・フラッシュにより実行をやり直す．このとき，このロード命令の PC について 1 ビットのフラグを立て，次回

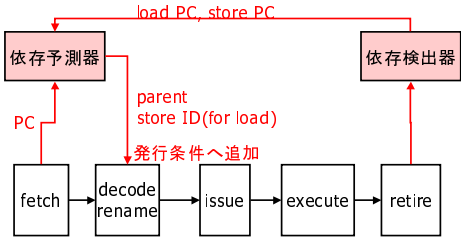


図 3 メモリ依存予測
Fig. 3 Memory Dependence Predicton

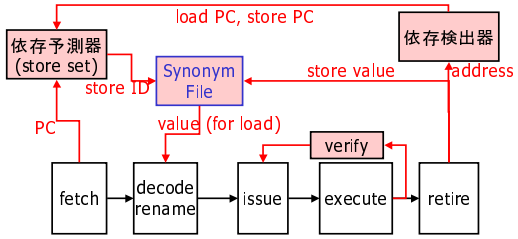


図 4 Moshovos らの手法
Fig. 4 Speculative Memory Cloaking

実行時からは投機を行わず, 先行するストア命令の発行を待つように制御する. このフラグは Wait Table と呼ばれるテーブルに保持され, デコード時に参照される. 一度のミスを元に投機が慎重になりすぎる事を防ぐために, Wait Table は一定サイクル期間で全てのビットが初期化される.

更に精度の高いアプローチとして, ロード命令と依存関係にある親ストア命令を予測する手法が提案されている. この方式では, ロード命令の発行条件を, 予測された親ストア命令に設定することで, 発行のタイミングを最速化することができる.

一度依存関係のあったストア命令とロード命令の PC 対は再び依存関係にある可能性が高く, 親ストア予測はこの性質を利用する. 親ストア命令とロード命令の PC 対の学習はリタイア時に行われ, 予測テーブルが更新される（**図 3**）. Chrysos らのストア・セット予測[5] では, ロード命令に過去依存のあった複数のストア命令 (ストア・セット) を保持することによって予測の適用範囲を広げている.

2.2.2 投機メモリフォワーディング

Moshovos ら[9),10)] の提案した “Speculative Memory Cloaking” は, 依存予測結果を利用し, 予測された親ストア命令のストア値をロード命令の実行結果として使用することにより, ロード命令の発行遅延とキャッシュ参照遅延の双方を短縮する. また同時期に Tyson ら[15)] が同様の手法 “Memory Renaming” を提案している. **図 4** に Moshovos らのモデルを示す. ストア命令実行後ストア値は Synonym File と呼ばれるバッファに書き込まれる. ロード命令はメモリ依存予測を利用して親ストア命令の ID を得, 更にその ID をキーとして Synonym File から予測値を得る.

この手法ではストア命令のバックエンドで書き込んだ値を, ロード命令のフロントエンドで読み出すため, ストア命令とロード命令の実行タイミングが離れていなければフォワードできない. Moshovos らは実装のオプションとして, 実行中のストア命令からフォワードするために, Synonym File にストア命令のリザベーションステーションへのポインタを保持するアイデアを示している. また, 更に積極的な手法として, ストア値を生成する “プロデューサ命令” からロード値を使用する “コンシューマ命令” へのフォワーディングを行う, “Speculative Memory Bypassing” という手法を提案している.

投機メモリフォワーディングを集中型のスーパスカラプロセッサに適用した評価では, 全ロード命令の約 40%がフォワーディングの対象となる一方, IPC はミス・フォワーディングの影響により逆に 5.63%低下したことが示されている.

3. メモリ参照処理によるオーバヘッド

3.1 クラスタ化によるオーバヘッドの増加

クラスタ型スーパスカラ・プロセッサでは, キャッシュ, ストアキューなどのメモリ参照ユニットも分散局所化されることが自然であるが, メモリ参照命令は, 依存関係が曖昧なため局所化が難しい. クラスタ型プロセッサの先行研究では, クラスタ外にある集中型のキャッシュとストアキューが全てのメモリ参照を処理する方式で近似されている事が多い.

本節では, 集中型の D1 キャッシュ構成についてオーバヘッド評価を行い, どの部分の高速化が求められているか解析する. 特に, 実行コアがクラスタ化されることにより, 傾向にどのような変化が現れるかに注目する.

メモリ参照処理のオーバヘッド要素は, 命令発行が遅れることによる発行遅延と, アドレス計算後の参照遅延に分けて考えることができる.

発行遅延は, ストア命令とロード命令の依存関係が完全には予測できないことから, ロード命令が不必要に発行を遅らされることによって生ずる. 曖昧なメモリ依存チェイン（図 2）の解決は, クラスタ化によって更に遅れることが予想される. 依存があると予測されたメモリ参照命令同士が異なるクラスタに割り当て

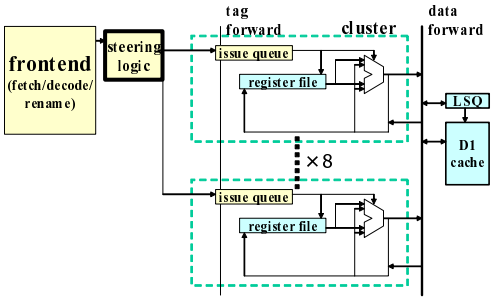


図 5 シミュレーション・モデル
Fig. 5 Simulation Model

られた場合, メモリ発行条件の伝播にクラスタ間通信遅延が伴うためである.

参照遅延は, 実行ステージにおけるアドレス計算終了後のメモリステージの長さである. 大きなキャッシュアレイの参照遅延に加えて, 実行ユニットとキャッシュの間の通信遅延等がオーバヘッドとなる. 配線遅延に耐性のない要素であるため, 高クロック指向に設計された実行コア部分に比べて相対的に長い遅延を生ずることが予想される.

3.2 評価環境

評価は, **図 5** に示す構成の高クロック指向クラスタ型スーパスカラ・プロセッサをトレースシミュレータに実装して行った. 議論で用いるシミュレーション・モデルの設定を**表 1** に示す. 特に表記がない場合は, 次節以降もこの設定を用いる.

シミュレーション・モデルは 1 命令実行幅のクラスタ 8 個による構成となっている. それぞれのクラスタは発行キュー, 複製されたレジスタファイル, ALU, フォワーディングデータパスを持つ. 各クラスタは高速な動作が可能であるが, 異なるクラスタ間での実行結果の反映にはクラスタ間通信遅延の 2 サイクルを要する. メモリ参照に焦点を当てるため, クラスタのトポロジについては簡略化し, 任意のクラスタ間で一律の通信遅延を仮定している. クラスタの位置に応じて通信遅延が異なる場合のステアリング最適化は文献 13) で議論されている.

命令をどのクラスタで実行するかは, ディスパッチ処理に加えられるステアリングロジックによって動的に決定される. ステアリングロジックには, 命令間の依存関係とクラスタ負荷分散の 2 つの指標を基に実行クラスタを選択する Parcerisa 方式[12] を用いた.

D1 キャッシュ及びストア・キューはクラスタ外に存在する集中型ユニットとなっており, 全てのクラスタにおけるメモリ参照処理を処理する. メモリ参照命令

表 1 ベースライン・パラメタ
Table 1 Baseline Parameters

number of clusters	8
components of each cluster	64entry IQ, 256physical registers execute 1 instruction/cycle
fetch/retire width	up to16insts/cycle
I-cache	perfect
branch predictor	gshare predictor 16k-entry, 10bit history
memory disambiguation	16k entry wait table, 100k cycle refresh interval
D-cache	64kB, 2-waySA, 8cycle hit latency 64byte lines 3 read ports 1read/write port
total pipeline depth	16stages
frontend latency	11cycles
issue to issue latency	1cycle
execute(int)	1cycle
execute(int MULT)	15cycles
execute(float)	4cycles
inter cluster forwarding	2cycles

は, 実行ステージにおけるアドレス計算までは他の命令と同様にクラスタで処理される. その後, アドレスやストア値が D1 キャッシュ及びストアキューへ送られ参照処理を行う. ロード命令の場合, アドレスを計算してからロード値を得るまでに, 往復の通信遅延やキャッシュアレイの参照遅延等の影響があり, 参照遅延を 8 サイクルに設定している.

シミュレーション・モデルの命令セットは DEC Alpha21264 に準じ, 実行トレースを入力とする. 評価には SPEC95int(train) から compress, gcc, go, ijpeg, li, m88ksim, perl(jumble, primes, scrrable), vortex の 10 種類を用い, 先頭から最大 256M 命令について動作を計測した.

3.3 ロード命令の発行遅延によるオーバヘッド

クラスタ型アーキテクチャにおけるロード命令の発行遅延オーバヘッドを評価するため, 現在用いられている Wait Table を用いたモデルと, フロントエンド時に全てのメモリ依存が判明しているモデルとの性能比較を行った. Wait Table を用いたモデルは, 小容量の追加ハードウェアで実現可能だが, ロード命令の不正な追い越しによるパイプラインフラッシュが発生する, wait と判断されたロード命令が不必要に発行を遅らされる等のオーバヘッドが生ずる. 一方, フロントエンドで全てのメモリ依存が判明するモデルは理想モデルであり, ロード命令は, 親ストア命令の発行を的確に待ち, 最速のタイミングで発行される.

この比較を, 8 命令実行幅のユニット 1 つによる集

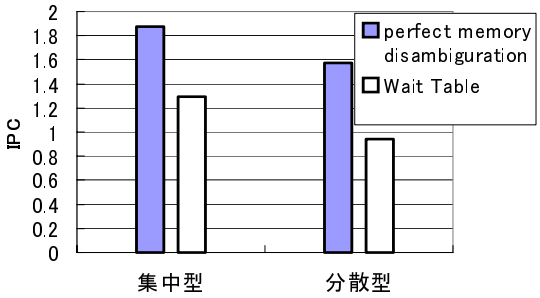


図 6 メモリ命令発行遅延を理想化したときの性能向上
Fig. 6 Performance Improvement of Perfect Memory Disambiguration

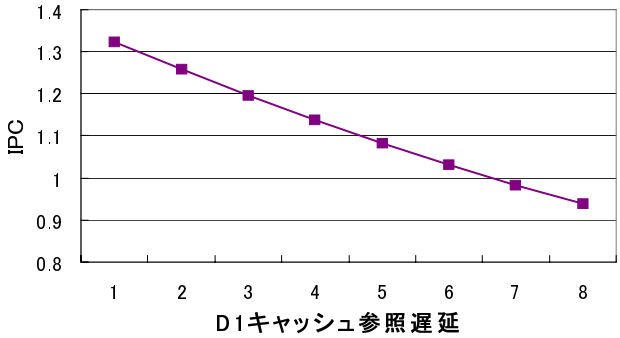


図 7 D1 参照遅延を変化させたときの IPC
Fig. 7 D1 Access Latency vs IPC

中型の場合と, 1 命令実行幅のクラスタ 8 つによるクラスタ型の場合でそれぞれ行った. 性能の指標には総リタイア命令数を総実行サイクル数で割った, IPC (Instructions Per Cycle) を用い, 10 種類のベンチマークについて調和平均を求めた.

図 6 に, 評価結果を示す. ロード命令の発行遅延によって, 性能が制限されていることが分かる. 集中型とクラスタ型で同じ遅延パラメタを用いているため, 集中型の IPC が有利となっている.

クラスタ型では, メモリ発行条件の伝搬に通信遅延が加わるため, ロード命令発行遅延の影響が増大する. Wait Table の値と理想モデルの値との比率に注目すると, クラスタ型では集中型に較べて, オーバヘッドの影響が増大していることが確認できる.

3.4 参照遅延によるオーバヘッド

キャッシュの参照遅延は, フロアプランや回路最適化, デバイス世代等によって決定されるため, 正確に見積もることが難しい. そこで, ここでは参照遅延を 1 サイクルから 8 サイクルまで変化させ, センシティビティに注目する. 性能指標には発行遅延と同じく IPC を用いた.

図 7 に D1 参照遅延を変化させた時の IPC の変化を示す. グラフはほぼ線形であり, D1 参照遅延が 1 サイクル減る毎に IPC が約 4%向上している.

クラスタ化によって実行コアが高クロック指向になると集中型 D1 の参照遅延は大きく増加することが予想される. 例えば, シミュレーションモデルで設定している 8 サイクルは次のような概算から決定している. まず, クラスタと D1 との通信遅延が, クラスタ間通信遅延と同じ値であると仮定する. この遅延はロード命令の場合, アドレスの送信, ロード値の受信となり, 往復で 2×2 サイクルである. キャッシュアレイの参照遅延は配線遅延に耐性がないため, 現行の参照遅延より大きい 3 サイクルを仮定した. 更に, 複数クラスタから発せられる参照リクエストのポート調停に 1 サイクルを見込んでいる.

D1 参照遅延の大幅な増加が見込まれる上, 図 7 のように D1 参照遅延は IPC にリニアに影響する要素となっていることから, クラスタ化によりメモリ参照遅延のオーバヘッドが増大することが予想される.

このように, クラスタ型スーパスカラ・プロセッサにおけるメモリ参照処理は, 発行遅延, 参照遅延双方共に増大し, 性能を制限する事が予想される.

4. 分散投機メモリフォワーディング

クラスタ化によってメモリ参照のオーバヘッドが増大する主な理由は処理が分散することによる通信遅延である. クラスタ型スーパスカラ・プロセッサでは, D0 キャッシュにあたる小容量のバッファをクラスタ内に設け, メモリ処理を局所化する手法が適していると言える.

このようなバッファの構成法はいくつか選択肢が考えられ, それぞれに得失がある. 小容量の D0 キャッシュの複製を各クラスタが持つ方式の場合, 機構は単純になるが, 複製のため, 容量効果が低い. また各クラスタによる更新を反映させるためライトポートが多く必要となってしまう. 一方, アドレスでバンク分けされたキャッシュをクラスタ内に持つ方式の場合, ステアリングが問題となる. また, これらの方式では予測スケジューリングや発行遅延の軽減が難しい.

そこで本論文では小容量バッファを利用する別のアプローチとして, 参照遅延, 発行遅延双方を改善する

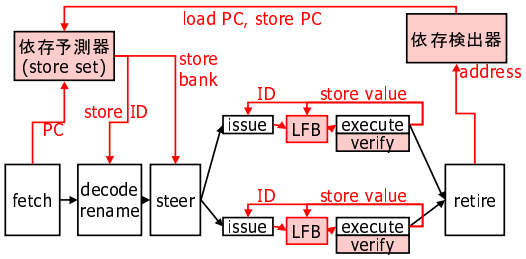


図 8 分散投機メモリフォワーディング
Fig. 8 Distributed Speculative Memory Forwarding

投機メモリフォワーディング技術に注目し, 予測スケジューリングを行うクラスタ型スーパスカラ・プロセッサへの適用を考える.

4.1 提案手法の概要

クラスタ内小容量バッファを投機メモリフォワーディングに用いるための, フォワーディング局所化手法を提案する. また, クラスタ型スーパスカラ・プロセッサでは実行中の命令が多いため, 実行中のストアからのフォワードは必須となる. 先行研究における実行中ストアからのフォワーディングのアイデアは, リザベーションステーションを想定したデータ駆動的なものであった. 本論文では, 高クロック指向のプロセッサにおける投機メモリフォワーディング制御を提案する. 提案手法である “分散投機メモリフォワーディング” の概念図を**図 8** に示す. 提案手法ではフォワードされるストア値は, 各クラスタ内のローカルフォワードバッファ（図 8 中の “LFB”）に保持され, 同じクラスタに割り当てられたコンシューマ命令からのみ参照される. また, 発行キューを拡張して, ストア命令の発行を監視し, 最速のタイミングのフォワーディング制御を行う. フォワーディング制御はクラスタ内に局所化され, 通信遅延やポート競合の影響を受けない.

更に, ステアリングロジックでメモリ依存予測情報を利用することにより, 局所化されたフォワーディング機構を支援する.

4.2 メモリ依存予測機構

依存のあるストア命令とロード命令を予測する予測機構には関連研究[5),10)] と同様の機構を用いる. 予測機構は大きく検出器と予測器に分けられ, それぞれリタイア処理とフロントエンド処理に追加される. フロントエンド処理やバックエンド処理の増加は, 実行のクリティカルパスとなりにくいため, 一般に実行性能にそれほど影響しない[11),14)].

検出器では, 参照アドレスを比較することで, メモリ依存の検出を行う. 検出結果は予測器に保持され,

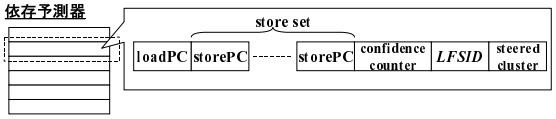


図 9 予測器 1 エントリの保持情報
Fig. 9 An Entry of Prediction Table

メモリ依存が学習される. 予測器の保持内容は**図 9** のようになっている. 依存予測器のエントリはロード命令の PC について 1 対 1 に作成される. 各エントリにはロード命令の PC, そのロード命令に過去依存のあったストア命令, 確信度カウンタ, 最後にフェッチされた該当ストア命令の ID(*LFSID*) とそのステア先が保持される. メモリ依存の履歴の中では, 一つのロード命令に複数のストア命令が依存関係を持つ可能性があり, 過去依存のあったストア命令は, ストア・セットとして保持される.

LFSID はストア命令のフロントエンド処理時に更新される. メモリ依存予測器をストア命令の PC について参照し, 該当する PC を含んでいるエントリが存在した場合には, そのエントリの *LFSID* にこのストア命令の ID を保持する. また, このストア命令が割り当てられたクラスタ番号を保持する. 予測器エントリはロード PC に対して 1 対 1 に作られるため, 複数のロード命令の親にあたるストア命令では, 複数のエントリが該当する場合がある. このため, この操作は全エントリを対象として行われる.

予測器にエントリが存在したストア命令は, その後の処理でローカルフォワードバッファにストア値と命令 ID の保持を行うように制御される.

一方, ロード命令のフロントエンド処理時に, 予測テーブルをロード命令の PC について参照することにより, 親と予測されるストア命令の動的 ID とステア先クラスタを得る. 確信度カウンタにより投機可能と判断されれば, そのロード命令について投機メモリフォワーディングを行う.

投機メモリフォワーディングの適用となったロード命令の値は, ローカルフォワードバッファを親ストア ID について参照することにより, 高速に得ることができる. フォワーディング制御については後述する.

予測器エントリには確信度カウンタが設けられ, ミスフォワーディングを起こした場合インクリメントされる. この値が閾値を越えると, 予測は適用されない. 予測適用が慎重になりすぎる事を避けるために, 確信度カウンタは一定期間でリフレッシュされる.

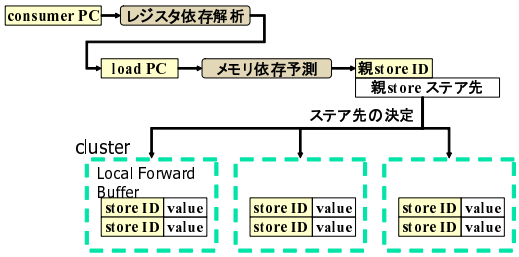


図 10 依存予測を利用したメモリ参照の局所化
Fig. 10 Memory Access Localization with Dependence Prediction

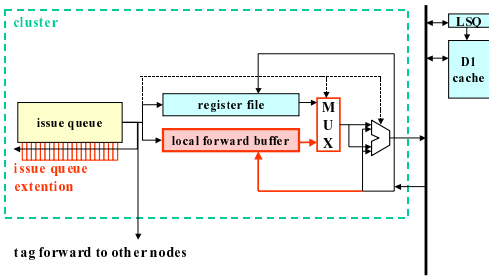


図 11 フォワーディング機構の構成
Fig. 11 Components of Forwarding Mechanism

4.3 フォワーディングの局所化

分散投機メモリフォワーディングでは, メモリ依存予測情報を利用して, メモリ依存関係にある命令を同じクラスタへ割り当てる（**図 10**）.

ステアリング時に, メモリ依存予測テーブル（図 9）を参照することで, 親ストア命令の割り当て先のクラスタ番号を得ることができ, コンシューマ命令は親ストア命令と同じクラスタへ割り当てられる. と同じクラスタを選択する. 他の命令は, 通常通りのステアリングロジックに従う.

このステアリングを利用することにより, クラスタ内のみから参照される高速なバッファを介して投機メモリフォワーディングを行うことが可能となり, 高速参照, 小ポート, 容量効果等, 分散局所化の利点を得る事ができる. また, メモリ発行条件の伝播にクラスタ間通信遅延が加わる影響を抑える事ができる.

一方, 欠点として, ステアリングの自由度を下げてしまう点が挙げられる. しかし, メモリ参照命令はクリティカルパスとなりやすく, メモリ参照に重点を置いたステアリングを優先する事が性能向上につながると考えられる.

なお, フォワーディング制御に関して, 本研究では, ロード命令のディスティネーションに値をフォワードするのではなく, コンシューマ命令のソースへ値をフォワードする実装を考える. この実装は, ローカルフォワードバッファの参照のタイミングをレジスタと同様の制御にすることができ, 発行スケジューラとの親和性が高い.

4.4 フォワーディング機構

図 11 に分散投機メモリフォワーディング機構を備えたクラスタのブロック図を示す. 追加されたハードウェアは, ローカルフォワードバッファとその参照のためのデータパス, 及び親ストア命令の発行を監視するための発行キュー拡張である. 提案手法を導入した場合, コンシューマ命令のオペランドについて, ready と

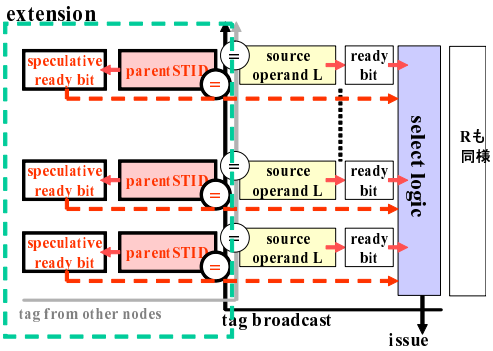


図 12 発行キューの拡張
Fig. 12 Extension of Issue Queue

なる条件は, i）ソースレジスタに書き込むロード命令が発行され, 予測される参照遅延分のサイクルが経った場合（正規 wake up）, ii）ソースレジスタに書き込むロード命令の親ストア命令が発行された場合（投機 wake up）, の二通りが可能となる. この ii）の条件を監視するために, 発行キューを**図 12** のように拡張し, コンシューマ命令の発行条件に親ストア命令を関連付ける.

ストア命令は発行されると, オペランドを読み出し, クラスタ内でアドレス計算を行い, ストア情報をクラスタ外部のキャッシュ階層へ転送する. また, フォワードのソースとなるストア命令の場合は, ストア値を命令 ID と関連付けてローカルフォワードバッファへ書き込む.

一方, ストア命令の発行はコンシューマ命令から監視されており, 投機 wake up の条件となる. 投機 wake up 状態で発行された命令は, オペランドをローカルフォワードバッファまたはフォワーディングデータパスから読み出し, 演算を行う. 演算結果はレジスタに書き込まれ, 後続の命令から参照される. なお, 投機 wake up 状態は, ソースレジスタ値が利用可能となっ

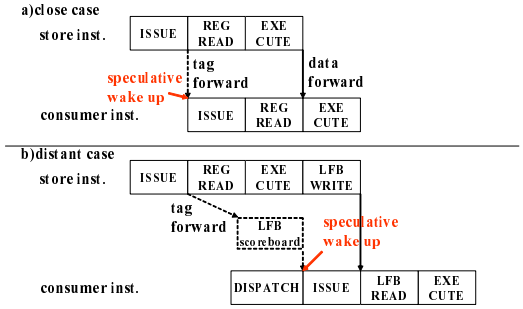


図 13 提案手法によるパイプラインタイミング
Fig. 13 Pipeline Timing with Forwarding

表 2 予測機構パラメタ
Table 2 Predictor Parameters

依存検出器エントリ数	128
依存予測器エントリ数	4k

た後は通常の wake up 状態に上書きされる．

ロード命令は通常のタイミングで発行され，キャッシュ参照処理を行い，投機フォワーディング結果が正しかったかをチェックする．間違った値を後続命令に渡していた場合は，コンシューマ命令以降をパイプラインフラッシュする．

4.5 提案手法によるゲイン

メモリ依存予測に基づく投機フォワーディングが成功した場合のパイプラインを**図 13** に示す． 提案手法では，メモリ依存予測に基づいてコンシューマ命令が発行され，ロード命令発行遅延の影響を軽減する．メモリデータはクラスタ内のフォワーディングデータパス（図 13（a））或いはローカルフォワードバッファ（図 13（b））から取得される．ローカルフォワードバッファの参照遅延は小さく，レジスタ参照が並行して行われるため隠蔽される．また，メモリ依存予測に従って，親ストア命令とコンシューマ命令が同じクラスタへステアされ，クラスタ間通信遅延のオーバヘッドを軽減する．これは，通常のレジスタ依存関係に注目したステアリングでは得られない効果である．

5. 提案手法の評価

3 章で用いたクラスタ型スーパスカラ・プロセッサに，分散投機メモリフォワーディングを実装し，シミュレータによる評価を行った．依存予測機構パラメタは，シミュレーションによって調べた最適なものを用いた．**表 2** にパラメタを示す．また，確信度カウンタの動作は，シミュレーション結果による知見から，以下のような動作を採用した．まず，閾値を 1 に設定し，一度で

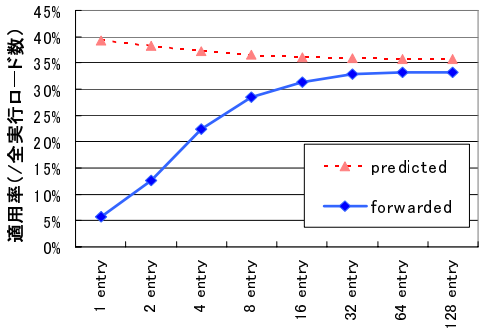


図 14 ローカルフォワードバッファのサイズとフォワード率
Fig. 14 Forwarding Coverage versus Local Forwad Buffer Size

もミスを起こしたロード命令には以降フォワーディングを適用しない．ただし，10 万サイクル毎に全てのエントリについて一斉に確信度カウンタのリフレッシュを行い，値を 0 に戻す．

5.1 提案手法の適用率

参照遅延やフロアプランの面から，ローカルフォワードバッファのエントリ数を少なく抑えられることが望ましい．一方，ローカルフォワードバッファのエントリ数が少なすぎると，ストアの書き込んだ値がコンシューマによって取得される前に上書きされてしまう．**図 14** にローカルフォワードバッファのエントリ数とフォワード適用率の関係を示す．グラフの点線は予測の適用されたロード命令の数を全実行ロード命令で割った値であり，35%程度のロード命令について親ストア命令予測が適用されていることが分かる．この値がフロントエンドにおける予測適用率である．一方，菱形でプロットした線が，ローカルフォワードバッファから値を得て正しく実行されたコンシューマ命令の数であり，同じく全ロード命令に対する割合で表している．この値に含まれているフォワードは性能向上に直接寄与する適用率を示している．依存予測が成立していても，ロード命令とコンシューマ命令の距離が離れていると，コンシューマ命令がディスパッチされた時点で既に先行のロード処理が完了している可能性があるが，この場合は投機フォワーディングよりも通常のレジスタリードが優先され，フォワーディングが成立しないためである． このグラフでは，ローカルフォワードバッファエントリ数が増加するとフォワード率も上昇し，逆に予測適用率は下がっている．フォワード率の上昇は容量効果であり，32 エントリでほぼ飽和，64 エントリで最大値となっている．これは D1 キャッシュと比較し

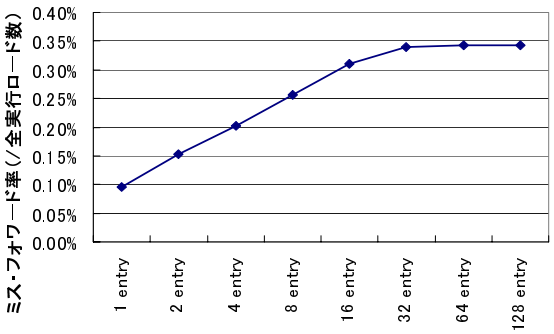


図 15 LFB サイズとミス・フォワード率
Fig. 15 Miss Forwarded Ratio versus Local Forward Buffer Size

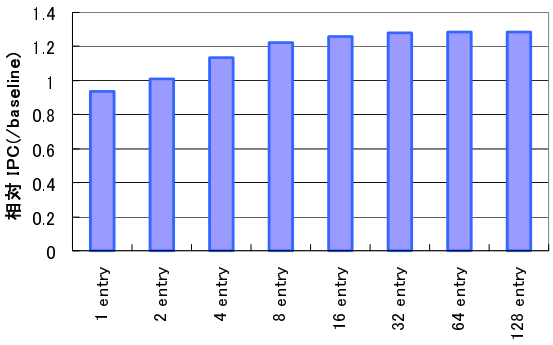


図 16 LFB サイズと提案手法の効果
Fig. 16 Effect of Forwarding versus Local Forward Buffer Size

て十分に小規模であり, また 4 エントリ, 8 エントリといった非常に小容量の構成でも良好な適用率を示している. 一方, 予測適用率が下がる原因は, フォワード率の上昇に伴ってミス・フォワード率も増えるため, 確信度カウンタによる判定が慎重になり, 不必要に予測適用を止めてしまうケースが増えるためである. 全ロード命令に対するミス・フォワード回数の割合を**図 15** に示す. 間違った予測に基づいてフォワーディングしてしまい, パイプラインフラッシュを発生させてしまう確率は全実行ロード命令に対し 0.35%程度であった.

5.2 性能への影響

図 16 に LFB サイズと IPC の関係を示す. IPC は, 同じプロセッサ構成で分散投機メモリフォワーディングを適用しなかった場合に対する相対 IPC で示している. 提案手法により, このプロセッサ構成では 28%程の IPC 向上を見込める事が分かる. また, エントリ数が少ない時でも IPC 向上を得ている.

次に 3 節と同様に, キャッシュ参照遅延を 1 サイクルから 8 サイクルまで変化させたときの IPC の様子を**図 17** に示す. 四角でプロットした線はベースラインプロセッサにおける IPC であり, 菱形でプロットした線は, 同じプロセッサモデルに分散投機メモリフォワーディングを適用した場合の IPC である.

フォワーディングにより参照遅延が改善するため, D1 参照遅延が大きくなるほど, フォワーディングを利用する効果が大きくなっている. 8 サイクルの D1 参照遅延では分散投機メモリフォワーディングを適用することにより 28%の性能向上を得ている. グラフの傾きに注目すると, 提案手法を適用した場合の方が傾きが緩く, D1 参照遅延が増加することによる性能低下に耐性があることが分かる.

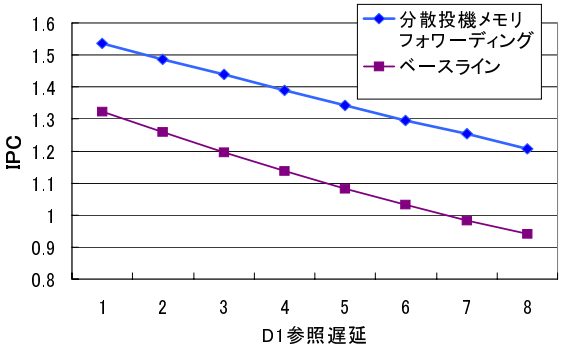


図 17 D1 キャッシュレイテンシと提案手法の効果
Fig. 17 Effect of Forwarding versus D1 Cache Latency

一方, D1 参照遅延が 1 サイクルの設定でも提案手法は 16%の性能向上を得ている, 依存予測に基づくステアリング及び発行によって, 発行遅延が改善していることが分かる.

提案手法は, メモリ参照命令の主要なオーバヘッドである発行遅延と参照遅延の双方を改善し, 性能向上に効果があると言える. また非常に少ないバッファ容量でも効果が高い特徴がある.

5.3 クラスタ台数効果の改善

図 18 に実装するクラスタ台数と IPC の関係を示す. 1 命令幅の実行クラスタ 1 個の構成から 16 個接続した構成まで変化させ, IPC を調べた. プロセッサの動作モデルや遅延パラメタは 3 節, 5 節で使用したものと同じ値になっている. この比較では, メモリ参照処理に焦点を当てるため, クラスタ間通信遅延を 2 サイクルに固定して単純化している. 実際はクラスタ台数が増えると一般にクラスタ間通信遅延が増加する

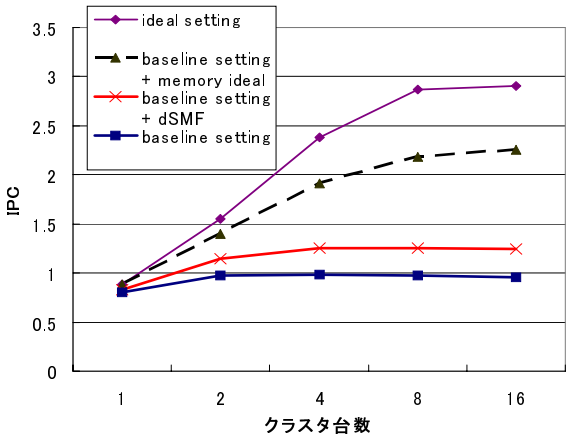


図 18 クラスタ台数と IPC
Fig. 18 IPC versus Number of Cluster

ため, 台数効果は, 図 18 よりも悪化する.

菱形でプロットした “ideal setting” の線はクラスタ化によるオーバヘッドとメモリ参照によるオーバヘッドを理想化した場合の IPC であり, シミュレーションしたパイプライン構成における並列性抽出のほぼ限界点を示している. 理想化設定は, クラスタ間通信遅延 0 サイクル, 集中型の発行キュー, メモリ依存関係をフロントエンド時点で取得, キャッシュ参照遅延 1 サイクル, となっている. 次に, 三角形でプロットした “baseline setting + memory ideal” の点線は, メモリ参照処理に関する, 発行遅延と参照遅延のオーバヘッドを理想化した場合の IPC である.

これらの値に比べて, 現実的な設定である “baseline” の線は性能が大きく乖離している. 台数効果は 2 クラスタ構成でほぼ飽和し, 8 クラスタ以上では逆に性能低下を招いている. 分散投機メモリフォワーディングを導入することにより, 台数効果は改善し, 4 クラスタから 8 クラスタまでの性能向上を達成している.

6. 結　　論

スーパパイプラインの限界を押し上げるクラスタ型構成が注目されているが, メモリ参照処理が高速化に追随できず, ボトルネックとなる可能性がある. 本論文では, 高速な実行コアと長い通信遅延を持つクラスタ型スーパスカラ・プロセッサを仮定し, メモリ参照処理のオーバヘッドを調べた. また, 投機メモリフォワーディングの考え方を利用し, ロード命令発行の早期化と参照の局所化を実現する, 分散投機メモリフォワーディングを提案した. 提案手法はストア依存予測を利用してステアリングを行い, 効果を高めている. シミュレータを用いて評価を行い, フォワード用のバッファは小エントリで構成できることを示した. また, 発行の早期化と参照の高速化双方が性能の向上に影響していることを示した.

一方, ステアリングの制約にメモリ依存を加えることによる自由度の低下がどのような悪影響を与えるか, 最適なステアリングロジックの観点から評価する必要がある. これらの提案, 評価は今後の課題である. また, クラスタ間遅延の厳密な評価や, 本手法で導入した発行キューの拡張による, タイミング・クリティカルパスの増加等, 回路的な解析も今後の課題である.

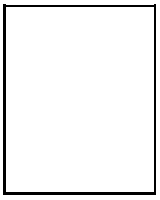
謝辞 本論文の研究は, 一部, 21 世紀 COE 「情報技術戦略コア」による.

参 考 文 献

1) V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger. Clock Rate versus IPC:The End of the Road for Conventional Microarchitectures. *27th Int.Symp.on Computer Architecture*, pp. 248–259, 2000.
2) R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi. Dynamically Managing the Communication-Parallelism Trade-off in Future Clustered Processors. *30th Int.Symp.on Computer Architecture*, pp. 275–286, 2003.
3) R. Canal, J. M. Parcerisa, and A. Gonzalez. A Cost-Effective Clustered Architecture. *Int.Conf.on Parallel Architectures and Compilation Techniques*, pp. 160–168, 1999.
4) R. Canal, J. M. Parcerisa, and A. Gonzalez. Dynamic Cluster Assignment Mechanisms. *6th Int.symp.on High-Performance Computer Architecture*, pp. 132–140, 2000.
5) G. Z. Chrysos and J. S. Emer. Memory Dependence Prediction using Store Sets. *25th Int.Symp.on Computer Architecture*, pp. 142–153, 1998.
6) B. Fields, S. Rubin, and R. Bodik. Forcusing Processor Policies via Critical-Path Prediction. *28th Int.Symp.on Computer Architecture*, pp. 74–85, 2001.
7) R. Kessler. The Alpha 21264 Microprocessor. *IEEE Micro*, Vol. 19, No. 2, pp. 24–36, 1999.
8) A. Moshovos and G. Sohi. Streamlining Inter-operation Memory Communication via Data Dependence Prediction. *30th Int.Symp.on Microarchitecture*, pp. 235–245, 1997.
9) A.Moshovos and G.Sohi. Speculative Memory Cloaking and Bypassing. *International Journal of Parallel Programming*, 1999.
10) S. Palacharla, N. P. Jouppi, and J. E. Smith.

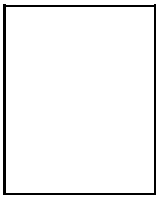
Complexity-Effective Superscalar Processors. *24th Int.Symp.on Computer Architecture*, pp. 1–13, 1997.
11) J. M. Parcerisa and A. Gonzalez. Reducing Wire Delay Penalty through Value Prediction. *33rd Int.Symp.on Microarchitecture*, pp. 317–326, 2000.
12) J. M. Parcerisa, J. Sahuquillo, A. Gonzalez, and J. Duato. Efficient Interconnects for Clustered Microarchitectures. *Int.Conf.on Parallel Architectures and Compilation Techniques*, pp. 291–390, 2002.
13) J. Stark, M. D. Brown, and Y. N. Patt. On pipelining dynamic instruction scheduling logic. *33rd Int.Symp.on Microarchitecture*, pp. 57–66, 2000.
14) G. Tyson and T. M. Austin. Improving the accuracy and performance of memory communication through renaming. *30th Int.Symp.on Microarchitecture*, pp. 218–227, 1997.
15) 服部直也, 高田正法, 岡部淳, 入江英嗣, 坂井修一, 田中英彦. クリティカルパス情報を用いた分散命令発行型マイクロプロセッサ向けステアリング方式. 情報処理学会論文誌コンピューティングシステム (ACS-6), pp. 12–22, 2004.
16) 服部直也, 高田正法, 岡部淳, 入江英嗣, 坂井修一, 田中英彦. 発行時間命令差に基づいた命令ステアリング方式. 先進的計算基盤システムシンポジウム 2004(SACSIS2004), pp. 167–176, 2004.

(平成？年？月？日受付)
(平成？年？月？日採録)



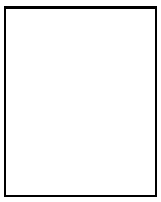
入江　英嗣（正会員）

1975 年生. 1999 年東京大学工学部電子情報工学科卒業. 2004 年同大学院情報理工学系研究科電子情報学専攻博士課程修了. 博士（情報理工学）. プロセッサアーキテクチャなどの研究に従事.



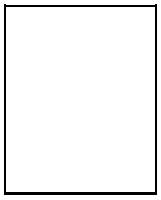
服部　直也

1976 年生. 1999 年東京大学工学部電子情報工学科卒業. 2004 年同大学院情報理工学系研究科電子情報学専攻博士課程修了. 博士（情報理工学）. プロセッサアーキテクチャなどの研究に従事.



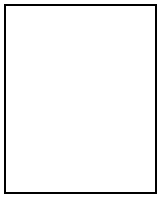
高田　正法（学生会員）

1979 年生. 2003 年東京大学工学部電子情報工学科卒業. 現在, 同大学院情報理工学系研究科電子情報学専攻修士課程在学中. プロセッサアーキテクチャなどの研究に従事.



坂井　修一（正会員）

昭和 56 年東京大学理学部情報科学科卒業. 昭和 61 年同大学院工学系研究科情報工学専門課程修了. 工学博士. 同年工業技術院電子技術総合研究所入所. 平成 3 〜4 年, 米国マサチューセッツ工科大学招聘研究員, 平成 5 〜8 年 RWC 超並列アーキテクチャ研究室室長. 平成 8 年筑波大学電子・情報工学系助教授. 平成 10 年東京大学大学院工学系研究科助教授, 平成 13 年同大学院情報理工学系研究科教授. 計算機システム一般, 特にアーキテクチャ, 並列処理, スケジューリング問題, マルチメディア応用などの研究に従事. 著書「論理回路入門」, 「図説コンピュータアーキテクチャ」. 電子情報通信学会, 人工知能学会, IEEE , ACM 各会員.



田中　英彦（正会員）

昭和 40 年東京大学工学部電子工学科卒業. 昭和 45 年同大学院工学系研究科博士課程修了. 工学博士. 同年同大学工学部講師. 昭和 46 年同助教授. 昭和 62 年同教授. 平成 13 〜16 年東京大学大学院情報理工学系研究科教授・研究科長. この間昭和 53 〜54 年米国ニューヨーク市立大学客員教授. 平成 16 年より情報セキュリティ大学院大学情報セキュリティ研究科教授・研究科長. 計算機アーキテクチャ, 並列処理, 自然言語処理, メディア処理, 分散処理, CAD 等の研究に興味を持っている. 著書「非ノイマンコンピュータ」, 「情報通信システム」, 「Parallel Inference Engine -PIE-」, 共著書「計算機アーキテクチャ」, 「VLSI コンピュータ I,II 」, 「ソフトウェア指向アーキテクチャ」. 本会フェロー. 電子情報通信学会, 人工知能学会, 日本ソフトウェア科学会, IEEE , ACM 各会員.