A Dependability Selection Method for Multicore Processors Considering Power-performance Trade-off

佐藤, 寿倫
九州大学 | 独立行政法人科学技術振興機構, CREST

舟木, 敏正
九州工業大学

出版情報：情報処理学会論文誌. 49 (6), pp.2005-2015, 2008-06-15. Information Processing Society of Japan
バージョン：

マルチコアプロセッサのための電力・性能間トレードオフを考慮したディペンダビリティ選択法

佐藤寿倫[†1,†2] 舟木敏正[†3]

半導体製造技術における微細化の進展により，ソフトエラーの増加が問題となっている．マルチコアプロセッサを利用しスレッドを冗長実行する方式では，消費電力が大幅に増大してしまう．一方，シングルプロセッサ内で命令を冗長実行する方式では，性能低下が深刻である．本稿では，マルチ・クラスタ型コア・プロセッサ（MCCP: Multiple Clustered Core Processor）と呼ばれるディペンダブルなマルチコアプロセッサ上での，消費電力と性能とのトレードオフを考察する．高い電力効率と高性能とを両立するために，スレッド冗長実行と命令冗長実行の組み合わせを提案する．シミュレーションにより，MCCP 上で提案方式を利用すると，スレッド冗長実行方式と比較して，エネルギー遅延積を 13%改善出来ることを確認している．

A Dependability Selection Method for Multicore Processors Considering Power-Performance Trade-off

Toshinori Sato[†1,†2] and Toshimasa Funaki[†3]

As deep submicron technologies are advanced, we face new challenges, such as power consumption and soft errors. A naive technique, which utilizes emerging multicore processors and relies upon thread-level redundancy to detect soft errors, is power hungry. Another technique, which relies upon instruction-level redundancy, seriously diminishes computing performance. This paper investigates a trade-off between power and performance of a dependable multicore processor, which is named multiple clustered core processor (MCCP). It is proposed to hybrid thread- and instruction-level redundancy in order to achieve both high power efficiency and small performance loss. Detailed simulations show that the MCCP exploiting the hybrid technique improves power efficiency in energy-delay product by 13% when it compares with the one exploiting the naive thread-level technique.

1. はじめに

マイクロプロセッサにおける近年の消費電力増大の問題を解決するために，マルチコアプロセッサが登場している．プロセッサの性能はその面積の平方根に比例し消費電力は面積に比例するという経験則を考慮すると，マルチコアプロセッサは電力利用効率の面で非常に優れた選択肢であると言える．

一方で，半導体製造技術における微細化の進展により，ソフトエラー率（SER: soft error rate）が増大しつつある[1),2)]．トランジスタサイズの縮小に伴い，ビットあたりの面積が小さくなっている．また高電界による破壊を防ぐために，電源電圧も縮小している．これらの結果ノードの電荷が小さくなり，宇宙線や α 粒子などにより容易にビットセルが反転する．一方でビットセルの面積が小さくなれば，中性子などの粒子がそこに衝突する確率も小さくなる．したがって両者の影響を考慮すると，ビットあたりの SER は世代間でほぼ一定であると言われている．しかしチップあたりのトランジスタ数は指数的に増大しているのであるから，チップあたりの SER も爆発的に増大することがわかる．

シングルイベントアップセット（SEU: single event upset）による故障を検出するために，またもし可能であるなら故障から回復するために，プログラムを冗長に実行することが提案されている[3)–9)]．近年登場しているマルチコアプロセッサは，この冗長実行に向いている．プログラムを複製し，同一チップ上の異なるコ

†1 九州大学
Kyushu University
†2 独立行政法人科学技術振興機構, CREST
JST, CREST
†3 九州工業大学
Kyushu Institute of Technology

アで冗長に実行する．一つのプログラムから得られる二つの冗長な結果が一致しなければ，SEU が検出されたことになる[3),4),7),9)]．本稿では，この方式をスレッド冗長方式（DTR: dependability with thread-level redundancy）と呼ぶ．残念なことに，DTR は大きな電力を消費する．二つのコアが動作すること，またそれらの結果を比較するための電力も考慮すると，少なく見積もっても一つのコアの消費する電力の二倍が必要である．

単一プロセッサ内での冗長実行を利用して SEU を検出することも可能である．プログラム自体ではなくプログラム中の命令を複製し，同一のプロセッサコア上で冗長に実行する．一つの命令から得られる二つの冗長な結果を比較し，もし一致しなければ SEU が検出されたことになる[5),6),8)]．本稿では，この方式を命令冗長方式（DIR: dependability with instruction-level redundancy）と呼ぶ．DIR は DTR に比べると低電力である利点を持つが，深刻な性能低下を引き起こす[5),6)]．

DTR における消費電力増大の問題と DIR における性能低下の問題を解決するために，これらの方式を組み合わせることを提案する．現在，プログラムに適応可能なマルチコアプロセッサを検討中である[10)]．マルチ・クラスタ型コア・プロセッサ（MCCP）と呼んでいるそれは，クラスタ型マイクロアーキテクチャ[11)]を採用している．MCCP 上で DTR と DIR を組み合わせることで，電力効率と性能の改善を目指す．我々の知る限り，ディペンダブルなマルチコアプロセッサの消費電力に関する研究はまだ行われていない．

本稿の構成は以下のとおりである．2 節で研究背景をまとめる．3 節で MCCP を紹介し，電力・性能間のトレードオフを考慮出来るディペンダブル方式を提案する．4 節でシミュレーション結果を紹介する．5 節でまとめとする．

2. 研究背景

本節では研究の背景を紹介し，MCCP で提供されるディペンダビリティの仮定を述べる．まずマイクロプロセッサでディペンダビリティを提供する方式を説明し，つづいてクラスタ型マイクロアーキテクチャついて述べる．

2.1 ディペンダビリティ

ディペンダビリティを提供するための最も単純な方式は，冗長実行である．時間的冗長性と空間的冗長性を利用出来る．Ray ら[5)]や Sohi ら[8)]は，各命令を複製し冗長に発行することを提案している．全ての命令はデコード時に複製され，それぞれ命令キューにディスパッチされる．二つの同じ命令は別々に発行され演算器で実行される．二つの冗長な結果はリオーダバッファで比較される．もし比較が一致しなければ SEU の発生を検出できている．同様に 6) では，動的命令スケジューラを利用して各命令を二度実行することを提案している．各命令は発行時に命令キューを開放せず，一度演算を実行した後も命令キューに滞在する．演算結果はリオーダバッファに格納される．冗長な演算結果はリオーダバッファ内にある一度目の結果と比較され，もし両者が一致しなければ SEU の発生を検出できている．これらの方式はいずれも，同じ命令から得られる二つの結果を比較することで SEU を検出している．つまり命令レベルでの冗長性を利用しているわけである．SEU 検出後のプロセッサ状態の回復には，投機実行失敗からの回復機構が使用される．分岐予測失敗からの回復機構[5),6)]や，データ投機失敗からの回復機構[6)]が使用される．つまり SEU に遭遇した命令を，投機に失敗した命令を読み替えているわけである．これにより，大きなハードウエアコストを追加することなく，故障からの回復を実現できている．

一方，マルチコアプロセッサは空間的冗長性の利用に適している[3),4),7),9)]．SEU を検出するために，プログラムを複製し同一プロセッサ上の異なるコアで冗長実行する．二つの結果が一致しなければ，SEU が検出されたことになる．Slipstream プロセッサ[9)]はマルチコアプロセッサであり，二つの冗長なスレッドを独立したコア上で実行する．しかし全命令を複製するわけではないので，全ての SEU を検出出来るわけではない．CRT[4)]もマルチコアプロセッサであるが，ステップ実行を可能にしているために上記の問題を解決できている．CRTR[3)]は CRT を改良したもので，SEU の検出後にプロセッサ状態を回復可能である．Shimamura ら[7)]は，メモリ上のデータ比較を行うマルチコアプロセッサを開発している．マルチコアプロセッサで SEU に対処するためには，同期，複製，検出，そして回復のそれぞれで考慮しなければならない点がある[12)]．つづいて，それらを説明する．

冗長に実行されるプログラムの結果を比較するためには，両者の同期を取る必要がある．CRTR[3)]では二つのプロセッサコア間にキューを配置し，先行コアがキューに書き込んだ演算結果を後続コアが読み出し，二つの演算結果が比較される．このようにキューを利用して同期が取られる．Reunion[13)]と Paceline[14)]ではプロセッサ状態をシグネチャに圧縮し，Reunion ではコア間でシグネチャを交換することで，Paceline で

は二つのシグネチャを比較用のキューに書き込むことで，それぞれ同期を取っている．

複製された命令が別々のコアで独立に実行されるので，同じロード命令が読み出す値が一貫しないという問題が生じうる[12),13)]．冗長な二つのロード命令に挟まれたストア命令が値を更新する可能性がある．CRT[4)] や CRTR[3)] では一貫性を保つための特別なキューを利用している．Slipstream[9)]，Reunion[13)]，そして Paceline[14)] では一貫性の保持を緩和している．一貫性の乱れを SEU の発生と同様にとらえプロセッサ状態を回復させることで，複雑な一貫性保持機構を排除している．

上記の同期の説明で述べたとおり，CRTR[3)] ではキューを利用して二つの演算結果を比較し，Reunion[13)] と Paceline[14)] では二つのシグネチャを比較している．

比較により SEU が検出されるとプロセッサ状態を回復しなければならない．CRTR[3)] では後続コアの状態を先行コアにコピーすることで SEU 検出前の状態に回復する．SEU 発生時には後続コアの状態は更新されないので，その状態は常に正しいことに注意されたい．Reunion[13)] と Paceline[14)] では定期的にチェックポイントを作成し，SEU 検出時には直前のチェックポイントにロールバックすることでプロセッサ状態を回復する．

MCCP では上述した，命令レベルでの SEU の検出および回復方式，そしてマルチコアプロセッサを利用する SEU の検出および回復方式を利用できると仮定している．以上が，3 節以降での検討の前提である．

マルチコア上の空間的冗長性を利用してディペンダビリティを提供すると，コア数が増える分だけ消費電力が増加する．これを抑えるために，SEU 検出を並列に実行する方式が提案されている[15)]．後続コアは先行コアの演算結果を利用できるので，本来なら不可能な並列化が可能になる．並列化できれば性能に悪影響を及ぼすことなく動作速度を低下でき，加えて電源電圧を下げることができる．これらにより消費電力の増加を抑制している．つまり 15) の方式は，(1)DVFS（Dynamic Voltage Frequency Scaling）を利用可能であることと，(2) 多数のコアを持つこととを必要とする．近い将来電源電圧を現在以上に下げることが困難になるという予想があること，また電源電圧を下げると SER が増加することが 15) の方式では問題となる可能性がある．加えて，コア数の増加も SER の増加につながることにも注意が必要である．対照的に，後述する提案方式では DVFS と多数コアの両方とも利用出来ない状況で，消費電力の増加を抑制することを目的としている．

2.2 クラスタ型マイクロアーキテクチャ

クラスタ型マイクロアーキテクチャは，配線遅延の問題に対するひとつの解決策を提供する[11)]．大きなプロセッサコアを複数のクラスタに分割する．各クラスタは十分小さいので，配線遅延の問題を回避出来る．汎用プロセッサはあらゆるアプリケーションプログラムで満足のいく性能を提供出来なければならないため，たいていのプログラムに要求されるより多くの演算資源を持っているのが普通である．したがって，電力の観点からは，アプリケーションの要求にしたがって演算資源をオン・オフ出来るのが好ましい．パイプラインバランシング[16)] はそのような方式であり，プログラムが大きな命令発行幅を要求しないときには，命令発行幅を縮小する．クラスタ型マイクロアーキテクチャで一部のクラスタをオフにすることで，この方式を実現可能である．

3. マルチ・クラスタ型コア・プロセッサ

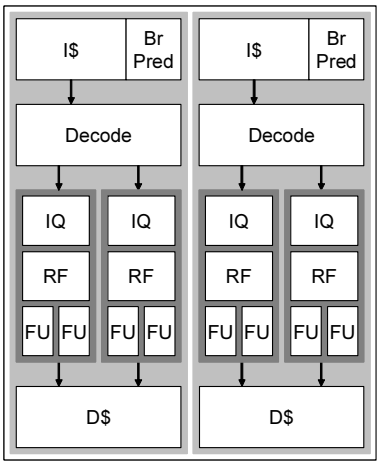


図 1 マルチ・クラスタ型コア・プロセッサ
Fig. 1 Multiple Clustered Core Processor

MCCP[10)] の例を**図 1** に示す．均質型マルチコアプロセッサである．従来の均質型マルチコアプロセッサとの違いは，各コアが単一的なコアではなくクラスタ型マイクロアーキテクチャ[11)] を採用している点である．図 1 の例では，二つのクラスタから構成されるコアを二つ持っており，各コアには命令キュー（IQ），レジスタファイル（RF），そして機能ユニット（FU）が備わっている．命令キャッシュ（I$），データキャッシュ（D$），分岐予測器（BrPred），そしてデコーダ

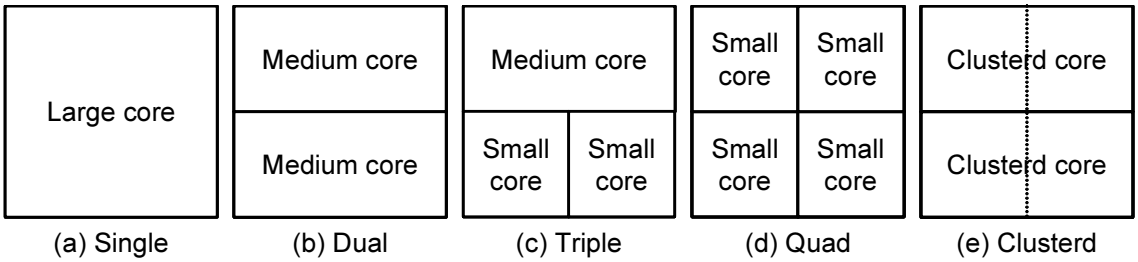


図 2 様々なマルチコアプロセッサ
Fig. 2 Different Types of Multicore Processors

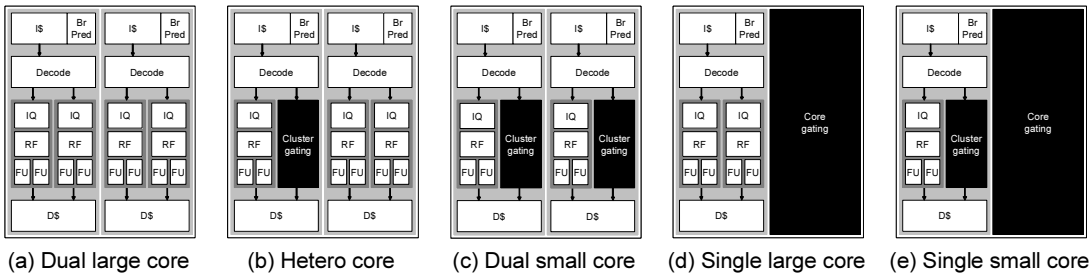


図 3 クラスタゲーティングとコアゲーティング
Fig. 3 Cluster Gating and Core Gating

(Decode) は，同一コア内の全てのクラスタで共有されている．クラスタ型マイクロアーキテクチャの特徴を利用し，これまでに電力と性能との間のトレードオフに関して検討してきた[10]．

3.1 電力と性能のトレードオフ

マルチコアプロセッサは小電力で高性能を達成出来る魅力的な選択肢であると言われている．**図 2** にマルチコアプロセッサの様々な構成を示す．図 2a は単一プロセッサである．図 2b と図 2d は均質型マルチコアプロセッサであり，図 2c は不均質型マルチコアプロセッサである．容易にわかるように，不均質型マルチコアプロセッサには，大きさと性能の異なる様々なコアが備わっている．高性能が要求されるが並列性の低いスレッドを実行する時には，大きなコアを利用する．同様に高性能が要求されるが並列性の高いスレッドを実行する時には，小さなコアを複数利用する方が電力効率の点で好ましい．一方，高性能が要求されない時には，小さなコアを選択する．不均質なコアから適切なものを選択することで，必要最小限の電力で要求される性能を満足できる．様々な大きさと性能のコアを備える不均質型マルチコアプロセッサは，電力効率の面で優れた選択肢である[17),18)]．

残念なことに不均質型マルチコアプロセッサは，均質型に比較すると設計やプログラミングが困難である．この問題を解決するために，MCCP は均質型マルチコアプロセッサ上に不均質性を実現する[10]．二つのクラスタを持つデュアルコア MCCP を例に用いて，クラスタゲーティングとコアゲーティングを利用して実現される不均質型マルチコアを**図 3** に示す．図 3a は大きなコアを二つ持つ均質型デュアルコアプロセッサである．性能が余っている時には，図 3b の様に一部のクラスタをオフにする．黒い四角はそのクラスタがオフの状態にあることを表している．クラスタゲーティングを利用することで，不要なクラスタをオフにし，アプリケーションが要求するに足るだけの性能を提供する．これは不均質型デュアルコアプロセッサである．図 3c では両コアでそれぞれひとつずつクラスタがオフになっており，小さなコアを二つ持つ均質型デュアルコアプロセッサとなっている．依然として性能に余裕がある時には，図 3d のように一部のコアをオフにする．黒い四角はそのコアがオフの状態にあることを表している．コアゲーティングを利用することで，不要なコアをオフにし，アプリケーションが要求するに足るだけの性能を提供する．クラスタゲーティングとコアゲーティングを同時に利用すると，図 3e のように小さな単一プロセッサも構成出来る．

上述のように均質型マルチコアプロセッサ上に不均質性を実現出来るので，均質型を意識して作成したプ

ログラムでも不均質性による恩恵を得ることが出来る．どのコアにどのスレッドを割り当てるべきかをプログラム時に考慮しなくて良くなるので，プログラミングが容易になると期待される．スレッドが要求する性能に足るコア構成を自在に用意できるので，不要な電力消費を削減出来る．MCCP のこの考え方は，組込み用途を指向している適応型マルチパフォーマンスプロセッサ[19] に生かされている．

クラスタゲーティングを実装するには様々な選択肢が考えられる．一つはハードウエアによる方法である．コア内の専用ハードウエアがスレッドの振る舞いを観測し，必要なクラスタ数を決定する．ソフトウエアによる方法も考えられる．命令セットにクラスタをオン・オフするための命令を追加する．プログラマやコンパイラがその命令をスレッド中に挿入する．プログラミングの容易さに配慮すると，クラスタ数を明示するよりも必要な性能を明示出来る方が望ましい．この方式は，アノテーションや API の利用により実現出来ると考えられる．例えば，プログラマが挿入したアノテーションをコンパイラが適切な命令に変換し，クラスタ数を提示する．異なるマルチコアプロセッサ間での互換性や透過性は，ソースレベルで提供出来る．以上のような選択肢が存在するが，クラスタゲーティングの具体的な実現方法は将来の検討課題とする．本稿では，ハードウエアによる方法を扱う．

3.2 ディペンダビリティの提供

MCCP はディペンダビリティを提供する点でも好ましい特徴を備えている．MCCP はマルチコアプロセッサであるから，DTR を利用出来る．プログラムを複製し，**図 4**a の様に複数のコアで冗長実行する．残念ながら，DTR は二つのコアで冗長実行するために，消費電力が増大する．

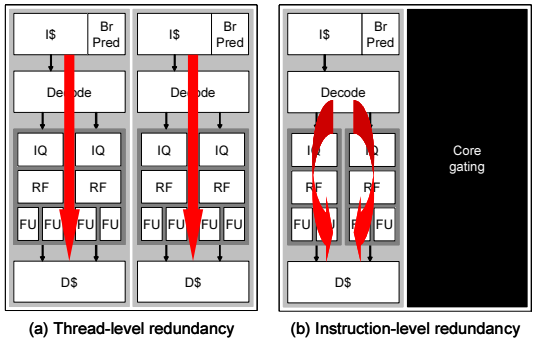


図 4 ディペンダビリティモード
Fig. 4 Dependability Modes

MCCP は DIR を利用することも可能である．プログラム内の各命令を複製し，図 4b のように同一コア内の異なるクラスタで冗長実行する．しかし，実行命令数が二倍に増加するために，性能低下が深刻である．

DTR における消費電力の問題と DIR における性能の問題を解決するために，MCCP 上で両方式を組み合わせて利用することを提案する．この方式をハイブリッド方式（DHR: dependability with hybrid thread- and instruction-level redundancy）と呼んでいる．MCCP は要求される性能に基づいてディペンダビリティモードを選択する．ある場合には dtr モードになり，また別の場合には dir モードを選択する．これ以降ディペンダブル方式とディペンダビリティモードを明確に区別するために，方式には大文字（DTL，DIL，DHL）を使用し，モードには小文字（dtr，dir）を使用する．ディペンダビリティモードの選択に MCCP の持つ適用性を利用することで，DTR と DIR の組み合わせが可能になる．本稿の焦点はディペンダビリティモードの選択方法である．考えられる戦略のひとつは，プログラマに頼る方法である．プログラマが各スレッドの重要度を予め決定し，アノテーション等でプロセッサに伝える．他には OS による方式が考えられる．デッドラインなどの指標に基づいて，OS がスレッドの重要度を決定する．本稿では，完全に自動なハードウエアによる方式を提案する．

3.3 IPC に基づくディペンダビリティモード選択

命令レベル並列性（ILP: instruction level parallelism）はプログラム毎に異なる．プロセッサが DTR だけに頼っていると，ILP が小さなプログラムでは電力を浪費してしまう．一方でプロセッサが DIR だけに頼っていると，ILP の大きなプログラムでは性能が大幅に低下してしまう．加えて，ILP はプログラム実行中でも大きく変化する[16]．**図 5** は SPEC2000 CINT ベンチマーク中の gcc をデュアルクラスタ・コアで実行した際の発行命令数の変化を示している．コアの構成の詳細は 4.1 節で述べる．X 軸は実行サイクル数を，Y 軸は 10,000 サイクル毎のサイクルあたりの平均発行命令数（発行 IPC）を現している．数百万命令に渡って発行 IPC が二倍以上の違いで変化していることがわかる．プロセッサが DTR だけに頼っていると，発行 IPC が小さな間には電力を浪費してしまう．一方でプロセッサが DIR だけに頼っていると，発行 IPC が大きな間には性能を著しく低下してしまう．この発行 IPC の変化を，ディペンダビリティモードの選択に利用することを検討する．

上述したように，DTR は発行 IPC が小さいと電力

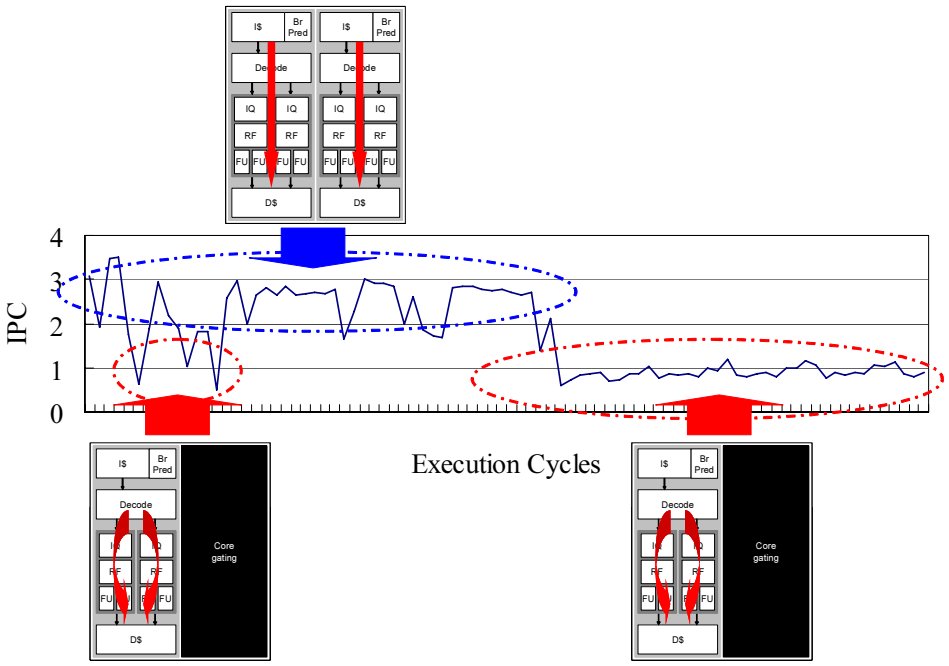


図 6 IPC に基づくモード選択
Fig. 6 IPC-directed Mode Selection

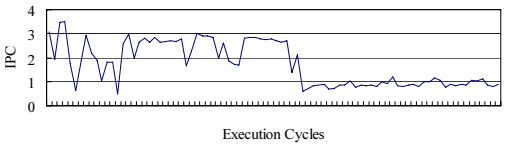


図 5 gcc における発行 IPC の変化
Fig. 5 Issue IPC Variation for gcc

を浪費し，DIR は発行 IPC が大きいと性能を低下する．この観察結果を DTR と DIR の組み合わせに生かすことを考える．すでに述べたように，この組み合わせ方式を DHR と呼ぶ．**図 6** に見られるように DHR は，発行 IPC が大きな時には dtr モードを，発行 IPC が小さな時には dir モードを利用する．発行 IPC が小さな時には演算器資源は余っているので，dir モードは深刻な性能低下を引き起こすことなくディペンダビリティを提供出来る．一方で，dtr モードは状況に応じてオフになるので，無駄な電力の消費が抑えられる．

ここで電力利用効率について考える．その指標としてエネルギー遅延積（EDP: energy-delay product）を採用する．もし DIR と DTR とで実行時間が同じであれば，明らかに DIR に方が EDP は小さくなる．しかし上述したように，DIR では実行時間が大きくなるので，DTR よりも EDP が大きくなる可能性がある．最悪ケースを考えよう．冗長実行しない場合と比較して，DTR は電力が 2 倍で実行時間は同じとなるので，EDP も 2 倍となる．一方 DIR は，電力は同じで実行時間は最悪 2 倍に延びるので，EDP は 4 倍になる可能性がある．つまり，実行時間への影響が小さいときのみに dir モードを選択しないと，DHR は DTR よりも電力効率が悪くなり得る．言い換えると，実行時間への影響が小さい状況下で dir モードを選択できれば，DHR が DTR よりも電力効率を改善できる．さらに付け加えると，dtr モードを選択することで実行時間を削減できるなら，DHR は DIR よりも電力効率を改善できる．

過去のプログラムの振る舞いで将来の振る舞いを予想できると仮定している．つまり，過去の発行 IPC を利用して将来の発行 IPC を予測する．浮動小数点数除算を避けるために，固定のサンプリング幅に発行された命令数を計数する．つまり，過去の発行 IPC ではなく過去の発行命令数で，将来の発行 IPC を予測するわけである．この予測された発行 IPC に基づいてディペンダビリティモードを選択する．dtr モード時に予測 IPC がある閾値（T_{T2I}）より小さくなると，dir モードに切り替える．同様に．dir モード時に予測 IPC がある閾値（T_{I2T}）より大きくなると，dtr モードに切り替える．

4. 評　　価

4.1 方　　法

SimpleScalar/PISA ツールセット[20] を利用してシミュレーションを行う．各コアが二つのクラスタを持つデュアルコア MCCP で評価する．**表 1** にプロセッサコアの構成をまとめる．フロントエンドと一次キャッシュは同一コア内の二つのクラスタで共有されている．同様に，二次キャッシュは同一チップ上の二つのコアで共有されている．ベンチマークプログラムには，SPEC2000 CINT から選んだ 6 本の整数系プログラムを使用する．

表 1 プロセッサコア構成
Table 1 Processor Core Configurations

Fetch width	8 instructions
L1 instruction cache	16K, 2-way, 1 cycle
Branch predictor	1K-gshare + 512-BTB
Dispatch width	4 instructions
Instruction window size	16 entries / cluster
Issue width	2 instructions / cluster
Commit width	4 instructions / cluster
Integer ALUs	2 units / cluster
Integer multiplires	2 units / cluster
Floating ALUs	2 units / cluster
Floating multiplires	2 units / cluster
L1 data cache ports	1 port / cluster
L1 data cache	16K, 2-way, 1 cycle
Unified L2 cache	512K, 2-way, 10 cycles
Memory	Infinite, 100 cycles

dir モード時の結果比較は細粒度であり，そのオーバヘッドは非常に小さく無視できるとみなせる．一方 dtr モード時に二つの結果を比較するためにはコア間の同期が必要であり，2.1 節で述べた方式のどれを選択するかによってオーバヘッドに違いが現れると考えられる．シグネチャを用いる方式と比較すると，各命令の結果を比較する方式はオーバヘッドが大きく，DTR および DHR の dtr モード時の性能が悪化すると予想される．したがって，その時の電力利用効率にも悪影響がある．本評価では，比較に要するオーバヘッドを無視できる方式を採用出来ると仮定する．また回復については，本稿では SEU の起こらない通常動作時について評価することとし，回復方式の影響は考えない．DIR については回復に要するオーバヘッドの影響が調査されており[21]，現実的な SER の環境ではその延慶は小さいことが判っている．

二段階で評価する．まずディペンダビリティモードの選択に必要な閾値を決定する．DHR の潜在的な有効性を見出すために，この段階では細粒度なモード切り替えが可能であると仮定する．サンプリング幅として 256 サイクルを選択し，モード切り替えにはオーバヘッドは無いと仮定する．各プログラムで，先頭の 2 億命令をスキップし，その後の 1 億命令を詳細にシミュレーションする．閾値が決定出来ると，続いて DHR における電力と性能のトレードオフを検討する．この段階ではより現実的な仮定を追加する．dir モードから dtr モードへの切り替え時には 100 サイクルのオーバヘッドを被ると仮定する．後ほど，オーバヘッドのサイクル数が変化したときの影響も調査する．このような大きなオーバヘッドを考慮すると小さなサンプリング幅は無意味であるので，本段階では 10,000 サイクルのサンプリング幅を選択する．サンプリング幅の違いの影響についても，後ほど調査する．先頭の 10 億命令をスキップし，その後の 20 億命令を詳細にシミュレーションする．

消費電力については単純な仮定を置く．動作中のコア数に比例して電力を消費すると仮定する．つまり，dtr モード時には dir モード時の二倍の電力を消費すると仮定する．現実にはクロックゲーティングなどが採用されるため，コア内の演算器資源の利用率の違いによりコア単体の消費電力は変化する．例えば，IPC=1 の状況と IPC=2 の状況を比較すると，後者のときの消費電力の方が大きいと予想される．このため，演算器資源の利用率の高い DIR と DHR と比較すると，その低い DTR は消費電力を大きく見積もってしまう可能性があることに注意されたい．また，上述したモード切替時には，dtr モード時の電力を消費すると仮定している．

ディペンダビリティの品質については，ディペンダブル方式やディペンダビリティモードに関係なく，均一な品質を提供出来ると仮定する．実際には，プロセッサ上で保護されている領域の違いによりディペンダビリティの品質は異なる[22] が，本稿は消費電力と性能との間のトレードオフに焦点を当てているため，このような単純な仮定を採用する．電力と性能に加えてディペンダビリティも考慮するトレードオフについての考慮は，将来の課題とする．

4.2 モード選択のための閾値

表 1 によると，発行 IPC はクラスタあたり最大 2 命令，コアあたり最大 4 命令である．したがってモード選択のための閾値は 2（×サンプリング幅，つまり 256 サイクル）近傍が妥当である．そこで，T_{T2I} については 1.6，1.8，2.0（× 256）を，T_{I2T} については 1.0，1.2，1.4，1.6，1.8（× 256）を検討する．**図 7** に T_{T2I} と T_{I2T} をそれぞれ変化させた時のプロセッ

サ性能を示す．サイクルあたりにコミットされた命令数の平均（コミット IPC）を指標として用いる．それぞれの値は，DTR を採用するプロセッサのコミット IPC で正規化している．各プログラムに対する 10 本の棒グラフのうち，最も左にあるものが DIR のコミット IPC であり，残りが閾値を変えた時の DHR のコミット IPC である．図中の X/Y は T_{T2I} と T_{I2T} の組み合わせを示している．例えば 1.6/1.0 は以下の意味である．直前のサンプリング幅で発行 IPC が 1.6 よりも小さいと dtr モードから dir モードに切り替わり，直前のサンプリング幅で発行 IPC が 1.0 よりも大きいと dir モードから dtr モードに切り替わる．

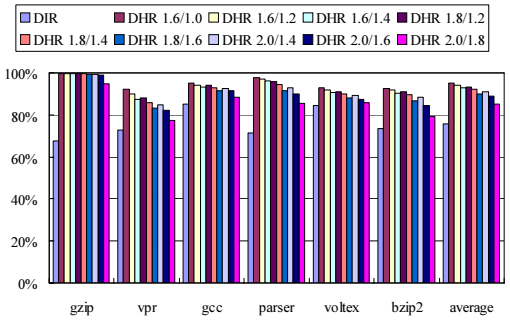


図 7 性能への影響（サンプリング幅=256，オーバヘッド=0）
Fig. 7 Impact on Performance
(window=256, overhead=0)

DIR だけに頼るとプロセッサ性能が著しく低下することが観察される．平均して 25%の性能低下である．DHR を採用することで性能低下を軽減出来る．例えば，1.6/1.0 の場合には性能低下は 5%にまで改善される．小さな T_{T2I} と小さな T_{I2T} は dtr モードを指向し，反対に，大きな T_{T2I} と大きな T_{I2T} は dir モードを指向する．より dtr モードを指向する閾値の組み合わせの方が小さな性能低下となることが観察される．しかし dtr モードは消費電力が大きいことに注意しなければならない．続いて電力効率を調査する．

図 8 には電力効率への影響がまとめられている．指標は EDP である．EDP が小さいほど電力効率が良好であることを示している．図 8 のレイアウトは図 7 のレイアウトに DTR を加えている．それぞれの値は，DTR を採用するプロセッサの EDP で正規化している．T_{T2I} と T_{I2T} が大きくなるにしたがい，電力効率が改善することがわかる．なぜなら大きな T_{T2I} と大きな T_{I2T} は，消費電力の小さい dir モードを指向しているからである．同じ理由で，DIR だけに頼るプロセッサがいくつかのプログラムで最も良い EDP を

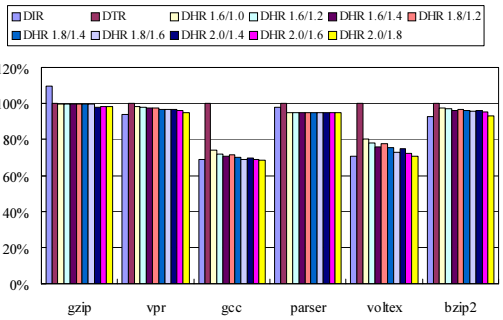


図 8 電力効率への影響（サンプリング幅=256，オーバヘッド=0）
Fig. 8 Impact on Energy Efficiency
(window=256, overhead=0)

示している．

コミット IPC と EDP を考慮し，T_{T2I} と T_{I2T} としてそれぞれ 2.0 と 1.6 を選択する．T_{I2T} として 1.8 を選択しないのは，15%の性能低下は許容出来ないと考えたためである．

4.3 電力と性能のトレードオフ

前節の考察から，T_{T2I} と T_{I2T} としてそれぞれ 2.0 と 1.6 を採用する．前節との違いは，ここでは，dir モードから dtr モードへの切り替え時に 100 サイクルのオーバヘッドを仮定していることと，サンプリング幅を 10,000 サイクルとしていることである．

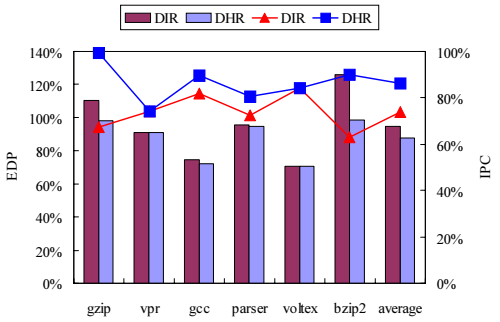


図 9 DHR と DIR の IPC と EDP
（サンプリング幅=10,000，オーバヘッド=100）
Fig. 9 Relative IPC and EDP of DHR and DIR
(window=10,000, overhead=100)

図 9 には，DHR と DIR をそれぞれ採用するプロセッサの発行 IPC と EDP がまとめられている．折れ線グラフは発行 IPC を，棒グラフは EDP を表している．折れ線グラフのうち，▲は DIR の結果を，■は DHR の結果を表している．二本の棒グラフのうち，左は DIR の結果を，右は DHR の結果を表している．いずれの値も，DTR の相当する値に対して正規化さ

れている．DTR と比較すると，DIR はコミット IPC を 27%も低下させているにも関わらず，EDP は 5%しか改善できていない．DHR はこの性能低下を軽減し，EDP を改善できている．DTR と比較すると，DHR はコミット IPC を 13%しか低下させず，その上 EDP を 13%も改善出来ている．DHR が全てのプログラムで EDP を改善出来ていることは強調するに値する．しかし DIR は二つのプログラムで EDP を悪化させている．以上の結果から，DHR は DTR における電力増大の問題と DIR における性能低下の問題を同時に軽減出来ていると結論付けられる．

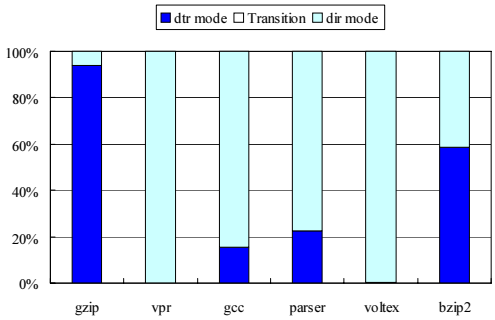


図 10 ディペンダビリティモードの内訳
（サンプリング幅=10,000，オーバヘッド=100）
Fig. 10 Breakdown of Dependability Modes
(window=10,000, overhead=100)

表 2 モード切替回数
（サンプリング幅=10,000，オーバヘッド=100）
Table 2 Mode Switching Count
(window=10,000, overhead=100)

gzip	vpr	gcc	parser	vortex	bzip2
8076	1	9085	6465	1857	3652

図 10 はディペンダビリティモードの内訳である．サイクル数で評価している．各棒グラフは三つの部分に分けられ，下から順に，dtr モードサイクルの割合，モード切替えサイクルの割合，そして dir モードサイクルの割合である．各プログラムでのモード切替回数は**表 2** にまとめられている．容易に判るように，モード切替サイクルは非常に小さい．興味深いことに，ひとつのプログラムはほぼ全てを dtr モードで実行し，対照的に別の二つのプログラムはほぼ全てを dir モードで実行している．残りの三つのプログラムはモードを使い分けている．以上から，IPC に基づくモード選択はプログラムの特性を掌握出来ており，プログラムの特性に応じて演算資源を適応出来ていることがわかる．それゆえに，DHR は DTR の電力増大の問題と DIR の性能低下の問題を軽減出来たと考えられる．

図 9 と図 10 をもとに，何故 DHR が電力効率を改善できるのかについて考える．まず dtr モードを指向する gzip について考える．図 9 から，DHR の dtr モード部分を dir モードで実行すると，実行時間が 1.47 倍になってしまう．コア数をひとつにして消費電力を半分にしても，EDP は 9%増加する．この値は図 9 から読み取れる値にほぼ等しい．つづいて dir モードを指向する vpr を考える．先ほどとは逆に DHR の dir モード部分を dtr モードで実行すると，実行時間が 0.74 倍に短縮できる．しかしコア数が二つとなって消費電力も二倍になるので，EDP は 10%増加する．この値も図 9 から読み取れる値にほぼ等しい．つまり，dtr モードを選択することによる実行時間での優位さと，dir モードを選択することによる消費電力での優位さとの両方を利用することで，DHR は DIR と DTR の両者に対して電力効率を改善できている．

4.4 オーバヘッドとサンプリング幅の影響

最後に，モード変更のためのオーバヘッドサイクル数と，サンプリング幅の影響を調査する．

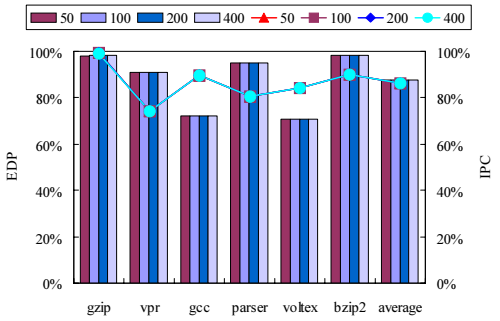


図 11 オーバヘッドの影響（サンプリング幅=10,000）
Fig. 11 Effect of Overhead (window=10,000)

図 11 には，オーバヘッドがプロセッサ性能と EDP に与える影響がまとめられている．サンプリング幅は 10,000 サイクルとしている．レイアウトおよび内容は図 9 と同様である．オーバヘッドのサイクル数は 50 から 400 サイクルの間で変化させている．明らかに，オーバヘッドサイクル数の違いは性能にも EDP にも影響していないことがわかる．この結果は図 10 からも容易に想像出来る．全てのプログラムで，モード切替えサイクルの割合は非常に小さい．

図 12 にはサンプリング幅がプロセッサ性能と EDP に与える影響がまとめられている．すでにオーバヘッドの影響は小さいことがわかっているので，評価した

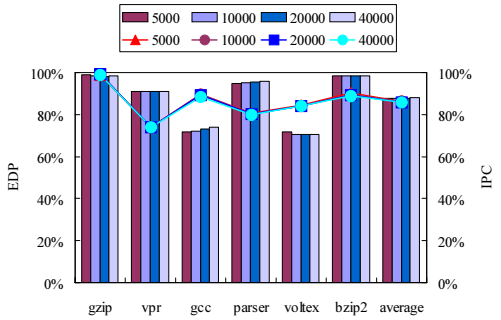


図 12 サンプリング幅の影響（オーバヘッド=400）
Fig. 12 Effect of Sampling Window (overhead=400)

中で最も大きな 400 サイクルを採用している．レイアウトおよび内容は図 9 と同様である．サンプリング幅を 5,000 から 40,000 サイクルの間で変化させた．オーバヘッドの場合と同様に，サンプリング幅の違いも性能と EDP にほとんど影響しないことがわかる．

5. おわりに

本稿では MCCP における電力と性能との間のトレードオフを検討した．マルチコアプロセッサは，スレッド冗長性を利用して SEU を検出可能なので，ソフトエラーの問題解決に向いている．しかし，残念なことに，冗長実行は多大な電力を消費する．本稿でスレッド冗長性と命令冗長性を組み合わせて利用することを提案し，スレッドの要求する性能に応じてディペンダビリティのモードを選択する方法について検討した．ILP が大きい時にはスレッド冗長性を利用し，逆に ILP が小さな時には命令冗長性を利用する．スレッド冗長性のみを利用する場合と比較して，提案方式はプロセッサ性能を 13%低下させるものの，EDP を 13%改善出来ることがわかった．一方で命令冗長性だけに頼ると，性能が 27%悪化する上に EDP は 5%しか改善出来ないこともわかった．これらのことから，MCCP 上で提案方式を利用する有効性が確認出来たと言える．

将来の課題のひとつは，ディペンダビリティの品質を評価できる指標を改善することが挙げられる．本稿では，いずれの方式も均一の品質を提供出来ると仮定した．実際には，方式が異なれば提供出来る品質も異なるはずである．現在，アーキテクチャレベル脆弱性 (AVF: architectural vulnerability factor)[23] を考慮したディペンダビリティ品質の指標を検討中である[22]．さらに，電力と性能に加えてディペンダビリティも考慮するトレードオフの検討をしなければならない．

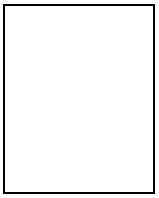
謝辞 本研究の一部は，文部科学省科学研究費補助金・基盤 A(No.19200004)，および科学技術振興機構・CREST プロジェクトの支援によるものである．

参 考 文 献

1) S. Borker: Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation, IEEE Micro, Vol. 25, No. 6 (2005)
2) T. Karnik, P. Hazucha, and J. Patel: Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes, IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 2 (2004)
3) M Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz: Transient-Fault Recovery for Chip Multiprocessors, 30th International Symposium on Computer Architecture (2003)
4) S. S. Mukherjee, M. Kontz, and S. K. Reinhardt: Detailed Design and Evaluation of Redundant Multithreading Alternatives, 29th International Symposium on Computer Architecture (2002)
5) J. Ray, J. C. Hoe, and B. Falsafi: Dual use of Superscalar Datapath for Transient-Fault Detection and Recovery, 34th International Symposium on Microarchitecture (2001)
6) 佐藤寿倫, 有田五次郎: 過渡故障に対するマイクロプロセッサ向けフォールトトレランス技術の提案, 並列処理シンポジウム JSPP (2001)
7) K. Shimamura, T. Takehara, Y. Shima, and K. Tsunedomi: A Single-Chip Fail-Safe Microprocessor with Memory Data Comparison Feature, 12th Pacific Rim International Symposium on Dependable Computing (2006)
8) G. S. Sohi, M. Franklin, and K. K. Saluja: A Study of Time-Redundant Fault Tolerance Techniques for High-Performance Pipelined Computers, 19th International Symposium on Fault-Tolerant Computing (1989)
9) K. Sundaramoorthy, Z. Purser, and E. Rotenberg: Slipstream Processors: Improving Both Performance and Fault Tolerance, 9th International Conference on Architectural Support for Programming Languages and Operating Systems (2000)
10) T. Sato and A. Chiyonobu: Multiple Clustered Core Processors, 13th Workshop on Synthesis and System Integration of Mixed Information Technologies (2006)
11) R. E. Kessler: The Alpha 21264 Microprocessor, IEEE Micro, Vol. 19, No. 2 (1999)
12) C. LaFrieda, E. Ipek, J. F. Martinez, and R. Manohar: Utilizing Dynamically Coupled

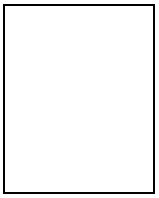
Cores to Form a Resilient Chip Multiprocessor, 37th International Conference on Dependable Systems and Networks (2007)
13) J. C. Smolens, B. T. Gold, B. Falsafi, and J. C. Hoe: Reunion: Complexity-Effective Multicore Redundancy, 39th International Symposium on Microarchitecture (2006)
14) B. Greskamp and J. Torrellas: Paceline: Improving Single-Thread Performance in Nanoscale CMPs through Core Overclocking, 16th International Conference on Parallel Architectures and Compilation Techniques (2007)
15) M. W. Rashid, E. J. Tan, M. C. Huang, and D. H. Albonesi: Exploiting Coarse-Grain Verification Parallelism for Power-Efficient Fault Tolerance, 14th International Conference on Parallel Architectures and Compilation Techniques (2005)
16) R. I. Bahar and S. Manne: Power and Energy Reduction via Pipeline Balancing, 28th International Symposium on Computer Architecture (2001)
17) M. Annavaram, E. Grochowski, and J. Shen: Mitigating Amdahl's Law through EPI Throttling, 32nd International Symposium on Computer Architecture (2005)
18) R. Kumar, K. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen: Single-ISA Heterogeneous Multi-core Architectures: the Potential for Processor Power Reduction, 36th International Symposium on Microarchitecture (2003)
19) 大山裕一郎, 室山真徳, 石原亨, 佐藤寿倫, 安浦寛人: 低消費エネルギーシステムのための適応型マルチパフォーマンスプロセッサ, 電子情報通信学会 2007 年総合大会 ISS 特別企画「学生ポスターセッション」(2007)
20) T. Austin, E. Larson, and D. Ernst: SimpleScalar: an Infrastructure for Computer System Modeling, IEEE Computer, Vol. 35, No. 2 (2002)
21) T. Sato and A. Chiyonobu: Evaluating the Impact of Fault Recovery on Superscalar Processor Performance, 12th Pacific Rim International Symposium on Dependable Computing (2006)
22) 舟木敏正, 佐藤寿倫: マルチ・クラスタ型コア・プロセッサにおける信頼性と性能のトレードオフ, 情報処理学会九州支部若手の会セミナー (2007)
23) C. T. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt: Reducing the Soft-Error Rate of a High-Performance Microprocessor, IEEE Micro, Vol. 24, No. 6 (2004)

(平成 年 月 日受付)
(平成 年 月 日採録)



佐藤 寿倫（正会員）

平成 1 年京都大学工学部電子工学科卒業．平成 3 年同大学大学院工学研究科電子工学専攻修士課程修了．同年，株式会社東芝入社．マルチプロセッサアーキテクチャ，消費電力見積り手法などの研究，および組み込み用途向けマイクロプロセッサの開発に従事．九州工業大学情報工学部助教授，九州大学システム LSI 研究センター教授を経て，平成 20 年 4 月より福岡大学工学部教授．マイクロプロセッサアーキテクチャ，LSI 設計手法に興味を持つ．博士 (工学)．情報処理学会平成 11 年度論文賞，同学会平成 15 年度山下記念研究賞を受賞．IEEE，電子情報通信学会，各会員．ACM シニア会員．



舟木 敏正（学生会員）

平成 18 年九州工業大学情報工学部知能情報工学科卒業．平成 20 年同大学大学院情報工学研究科情報科学専攻博士前期課程修了．同年 4 月よりパナソニックコミュニケーションズ株式会社．プロセッサアーキテクチャ研究に従事．