様　式　Ｃ－１９、Ｆ－１９、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成 ２８ 年　 ６ 月　 ２ 日現在

機関番号：１２６０１
研究種目：若手研究(A)
研究期間：2012～2015
課題番号：２４６８０００４

研究課題名（和文）補助スレッドによるメニーコアプロセッサの動的アーキテクチャ最適化の研究

研究課題名（英文）A study on helper-thread based runtime architectural state optimization methodology

研究代表者
近藤　正章（Kondo, Masaaki）

東京大学・情報理工学（系）研究科・准教授

研究者番号：３０３７６６６０

交付決定額（研究期間全体）：（直接経費）　　9,400,000円

研究成果の概要（和文）：メニーコアプロセッサ向けにソフトウェアからプロセッサ内部構成を最適化する手法の研究を行った。キャッシュ制御への適用では、キャッシュラインの置換制御をソフトウェアから制御する手法を開発した。キャッシュ置換の際のデータ挿入位置をページの粒度でハードウェアに通知し、ハードウェアがそれに基づいて制御を行う手法を用いることで、約11%程度の性能向上効果があることがわかった。
また、ソフトウェアからハードウェア構成を最適化する本研究の発展形として、OpenCLプログラミングによるFPGAアクセラレーションでのコード最適化の研究も行い、コード変化によりいくつかの場合で高速化が可能であることを示した。

研究成果の概要（英文）：In this research, we developed a software-based runtime architectural state optimization methodology for manycore processors. We applied proposed method to the optimized control of cache line replacement with helper-threading in order to mitigate the cache contention. In this technique, the helper-thread optimizes and indicates the policy of date insertion position within a cache set with a page granularity. The hardware performs data replacement accordingly. Based on the cycle accurate simulation, it is revealed that the proposed method achieves about 11% performance improvement over the conventional hardware-based cache replacement.
As an extension of this research, we studied and analyzed code optimization for FPGA accelerators. We proposed some code modification strategies for achieving higher performance in OpenCL based FPGA accelerations and showed that they are effective in some cases.

研究分野：計算機アーキテクチャ

キーワード：計算機アーキテクチャ　並列・分散処理　ハイパフォーマンスコンピューティング　メニーコア　マルチスレッド

１．研究開始当初の背景

近年のマイクロプロセッサは、消費電力や設計コスト増大といった問題を背景に、クロック周波数の向上に頼らずに、複数コアをチップ内に搭載して並列・並行処理により高い性能を得るマルチコア・プロセッサが主流となっている。今後は数十から数百といったより多数のプロセッサコアを搭載するメニーコアの時代になると予想され、より高い性能を得ることができると期待されている。しかしながら、アプリケーションプログラムから抽出できる並列性には限界があり、逐次実行部が性能向上を律速してしまうため、メニーコアプロセッサの性能向上には自ずと限界がある。

従来研究では、コア数が増加するとそれらを有効に活用できるだけの並列性を抽出できず、本問題が深刻化することが指摘されている。また、メニーコア化によりチップあたりの演算性能が大きく向上する反面、チップ外にある主記憶とのデータ転送性能が相対的に低下し、メモリアクセスがシステム全体の性能向上を阻害してしまうというメモリウォール問題も深刻化の一途を辿ると考えられている。以上より、たとえアプリケーションを並列化しても、メニーコアプロセッサが本来持つ演算能力を最大限活用した性能向上を得ることが今後は難しくなると予想される。さらには、逐次プログラムで記述された既存アプリケーションも多数存在し、それらを高速に実行したいという要求も依然として多い。

研究代表者は、これまで高性能・低電力プロセッサアーキテクチャやメニーコアプロセッサの設計に関する研究を行ってきた。その中で、上記理由により全コアが性能向上に寄与する機会が多くないことを経験し、将来的に半導体技術の進歩の恩恵を享受しつつ、恒久的にプロセッサの性能を向上させるためには、メニーコアプロセッサの演算処理能力を最大限に活用できる新しいアーキテクチャ技術・実行モデルの開発が必要であると考えるに至った。

２．研究の目的

メニーコアプロセッサが持つ多数のコアを有効に活用し、並列プログラムだけでなく逐次プログラムの性能を向上させることを目的に、アプリケーションを実行するコアの内部動作を最適化する専用の補助スレッドを性能に寄与しない遊休コアで実行することで性能向上を狙う「補助スレッドによるメニーコアプロセッサの動的アーキテクチャ最適化の研究」を行う（図１）。補助スレッドは、キャッシュ置換制御や分岐予測制御など、メインスレッド実行中のコアのアーキテクチャを直接制御する。各アプリケーションの特徴に合わせて動的にアーキテクチャ動作を最適化できる他、ソフトウェアによる制御のため、単純なハードウェアで複雑な制御

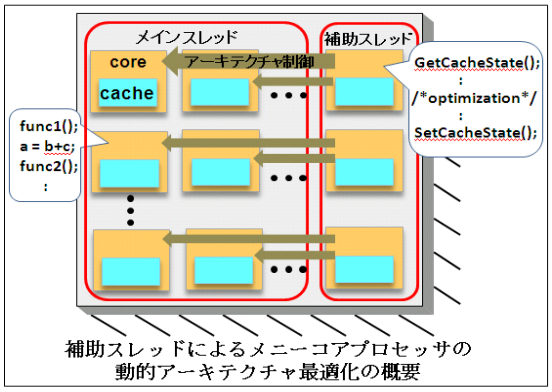

図１：提案方式の概要

を実現できるなど多くの利点を持ち、さらにはキャッシュ動作を最適化することによりメモリウォール問題への解決策としても期待できる。

３．研究の方法

補助スレッドを用いてコア内部のアーキテクチャを動的に最適化する本提案研究課題の有効性を明らかにするために、1) プログラムの特徴に応じてアーキテクチャ要素を制御した際の性能への影響調査、2)補助スレッドによるプロセッサコア内部状態の制御が可能なアーキテクチャ、3)補助スレッドによるコア内部状態の情報取得・制御・パラメータ変更が行えるメニーコアシミュレーション環境、4)アーキテクチャ要素を制御して性能を向上させるための最適化戦略、4)補助スレッドの作成アルゴリズム、といった要素技術を開発した上で、最終的に本手法の有効性を示すために、5)各要素技術を統合し、アプリケーションプログラムと補助スレッドをメニーコアプロセッサ上で実行させた際の性能を種々のベンチマークプログラムにより評価する。

４．研究成果

本報告では、メニーコアプロセッサ向けにソフトウェアからのプロセッサ内部構成最適化手法に関する研究として、特に効果の高かったキャッシュ制御への適用について述べる。また、近年回路構成をソフトウェアから変更可能な FPGA をアクセラレータとして用いる技術が注目され、さらに OpenCL 汎用プログラミング言語から直接 FPGA を構成可能な環境が研究期間内にリリースされたこともあり、ソフトウェアからハードウェア内部構成を最適化する本研究の発展形として、汎用プログラミングによる FPGA アクセラレーション最適化の研究も行った。その結果についても報告する。

４．１ ヘルパースレッドによるキャッシュ制御支援手法

(1) 背景

半導体プロセスの微細化に伴いプロセッサの消費電力，および発熱量は年々増大して

おり，従来のクロック周波数向上によるシングルコア・プロセッサの高性能化は困難となっている．そのため，近年では１つのチップ上に複数のコアを搭載するマルチコア・プロセッサが主流となっている．今後も，マルチコア・プロセッサのコア数は増加すると予想されるが，現在ではまだ多くのコアを活用できるような並列プログラムは限られており，増加するマルチコア・プロセッサのコアを有効利用することは重要な課題である．

マルチコア・プロセッサでは，キャッシュメモリの有効利用の観点から，複数のコアでキャッシュ(一般的にはラストレベルキャッシュ)を共有する場合が多い．しかし，共有キャッシュにおいてキャッシュ競合が発生すると，性能低下を引き起こすという問題がある．

本研究では，遊休状態のコアの有効活用と，共有キャッシュの競合による性能低下の防止を図る手法を提案する．共有キャッシュ上で生じたキャッシュミス情報を基にどのデータをキャッシュに残すべきか予測を行う専用のスレッドをヘルパースレッドとして動作させる．再利用性の高いデータはデータ置換の対象になりにくいように Most Recently Used (MRU)の位置に，一方，再利用性の低いデータは再利用性のあるデータを追い出さ ないよう Least Recently Used (LRU)の位置にデータを挿入することによって，再利用性の高いデータがキャッシュに残りやすくなるよう制御する．これにより，再利用性の高いデータがキャッシュに残る確率が高くなり，共有キャッシュ上での競合の緩和による性能の向上を目指す．

これまでにも，キャッシュ競合への対処を目的として，キャッシュ分割などハードウェアによる対処手法が多く研究されてきた．しかしキャッシュ競合の状況はプログラムのメモリアクセスパターンや，同時に実行するスレッドの組み合わせに依存する．よって，ある特定の手法があらゆるキャッシュ競合下で効果的に動作するとは限らない．ヘルパースレッドを用い，ソフトウェアによりデータの置換制御を支援することで，実行しているプログラムに合わせて柔軟に制御が行えるため，従来手法のハードウェアベースの手法に比べて有用な手法になり得ると考えられる．

(2) ヘルパースレッドによるキャッシュ置換制御手法

ここでは，共有ラストレベルキャッシュ(LLC)を持つコア数 k+1 のプロセッサ環境を例として， ヘルパースレッドがデータの再利用性予測を行い，その予測結果を用いてキャッシュ置換制御を行うまでの一連の流れについて述べる．また，ヘルパースレッドによるキャッシュ置換制御に必要なハードウェアの拡張と，ヘルパースレッドによる再利用性予測アルゴリズムについて述べる．なお，キャッシュミス情報を格納する Miss State Holding Register (MSHR)を持つプロセッサを前提とする．

ヘルパースレッドの動作：まずヘルパースレッドは遊休状態であるコアで動作し，共有の LLC でロードミスが起きたとする．通常，キャッシュミスが生じた場合，キャッシュミス情報は MSHR へ書き込まれ，主記憶へデータアクセスが行われる．以下ヘルパースレッドの動作の流れを説明する．

i. キャッシュミス情報の読み出し: ヘルパースレッドは,定期的に MSHR にキャッシュミス情報があるかをチェックし，ある場合 MSHR からそれを読み出す．この情報読み出しはメモリマップド I/O 方式により行われる．

ii. 再利用性の予測: 取得したキャッシュミス情報を基に，ミスしたデータのアドレス，あるいはそのアドレスが含まれるページ全体の再利用性を予測する．ヘルパースレッドの再利用性予測アルゴリズムについては後述する．

iii. 置換制御情報の通知: データの再利用性予測を行った結果に従い，キャッシュ置換制御のための情報をキャッシュコントローラへ通知する．ヘルパースレッドにより通知される置換情報は，新たに追加する置換制御テーブルへ格納する．置換制御テーブルについても後述する．キャッシュコントローラは，予測した再利用性に基づき，再利用性が高いと判断されたデータは追い出されにくく，再利用性が低いと判断されたデータは追い出されやすくなるように，キャッシュ置換制御を行う．

ハードウェアの拡張：本研究の提案手法を用いるには，ソフトウェアからのキャッシュミス情報の取得やキャッシュデータの再利用性結果の通知，それに基づいた置換制御が行えるようハードウェア拡張が必要となる．

MSHR に保存するキャッシュミス情報は，ロード・ストア命令によってアクセスされた物理アドレスのみでなく，各コアがそれぞれ持つ識別用の番号であるコア ID と，ロード・ストア命令の仮想アドレスを保持するように拡張する．ヘルパースレッドはメモリマップド I/O 方式により，通常のロード命令により MSHR の情報を読み出せるようにする．そのため,MSHR をソフトウェアから読み込むための専用経路が必要になる．同じく，キャッシュコントローラへ情報を設定する専用経路を新たにハードウェアに追加する．

通常の LRU 以外の置換制御を行えるようにするために，キャッシュコントローラ自体にも拡張が必要となる．キャッシュコントローラにはヘルパースレッドにより通知されたデータの再利用性予測情報を記録できるように，次の２つの置換制御テーブルを新たに追加する．

・Reusable-Data Address Table(RDAT):再利用性があると判断したアドレスを保持するテーブル．キャッシュミスが生じた際に RDAT 中に当該アドレスが属するページがある場

合，データは MRU 位置へ挿入される．
・Non-Reusable-Data Address Table (NDAT)：再利用性がないと判断したアドレスを保持するテーブル．キャッシュミスが生じた際に NDAT 中に当該アドレスが属するページがある場合，データは LRU 位置へ挿入される．

共有キャッシュでキャッシュミスが生じた際，キャッシュコントローラはこれらの置換制御テーブルへアクセスを行い，キャッシュミスアドレスと比較する．本提案手法では，基本的に LRU 方式に基づいてキャッシュ置換制御を行われるが，キャッシュミスしたデータを下位階層のメモリからリードし，キャッシュに書き込む際の位置が異なる．通常は MRU 位置にデータを挿入するが，提案手法では RDAT 中に当該アドレスが属するページがある場合,データは MRU 位置へ挿入され，NDAT 中に当該アドレスが属するページがある場合，データは LRU 位置へ挿入される．両者のテーブルともに当該アドレスが属するページが存在しない場合には，セット上の MRU と LRU の中間位置に挿入する．

再利用性予測アルゴリズム：ヘルパースレッドは，ハードウェアからキャッシュミス情報を取得し，それを基にキャッシュデータの再利用性予測を行い，その結果に応じて RDAT と NDAT へのアドレス情報の書き込みを行う．本研究では，ヘルパースレッドの再利用性予測アルゴリズムとして，プロファイルベース手法，ストリームベース手法，仮想拡張セット手法の 3 つを提案した.以下,仮想拡張セット手法について説明する．

・拡張仮想セット手法：再利用性が高いにも関わらず競合により共有キャッシュから追い出され，すぐにまたアクセスされるデータも多いことに着目し，追い出されたアドレス情報に基づいてデータの再利用性予測を行う手法である．本手法は，共有キャッシュから追い出されたデータのアドレスをヘルパースレッドが記録しておき，それが再びアクセスされるかをチェックする．そして，アクセスされたタイミングによってデータの再利用性の有無を決定する．

共有キャッシュから追い出されたデータをチェックする際，ヘルパースレッドで共有キャッシュ内の全セットをチェックするのは現実的に困難である．そこで共有キャッシュの一部セットをサンプリング対象とする．プロファイル手法やストリームベース手法とは異なり，この手法が収集する情報は共有キャッシュにおけるキャッシュミス情報ではなく，サンプリング対象となっているセットでキャッシュミスが発生した際に追い出されたデータのアドレスである．その情報を MSHR より収集することで，データの再利用性予測を行う．

(3) 評価

本技術の評価には，様々な評価環境を指向したが，ここでは PTLsim をベースに構築されたマルチコア・シミュレータ MARSSx86 を用いた場合の結果を述べる．本シミュレータに対し，提案手法のハードウェア拡張を実装した上で評価実験を行う．

ベンチマークプログラムには SPEC CPU2006 からいくつかを選択し，その組み合わせ毎に評価を行った．特に，はメモリアクセス回数の多いベンチマークの組み合わせ(MA-high)，はメモリアクセス回数の少ないベンチマークの組み合わせ(MA-low)，それらをミックスした組み合わせ(MA-mix)を使用した．

評価の対象となるプロセッサは，各コアで専用の命令／データキャッシュを持ち，また共有ラストレベルキャッシュとして L2 キャッシュを持つ．評価には，プロファイルベース手法，ストリームベース手法，仮想拡張セット手法をそれぞれ用い，ヘルパースレッドを実行しない通常のキャッシュ置換制御をした場合との比較を行う．評価指標には Weighted SpeedUp を用いた．

(4) 評価結果

図 2 に評価結果を示す．横軸は，各ベンチマークの組み 合わせで，縦軸は提案手法による通常のマルチコアに対する性能向上比を示している．また，prof,strm，vset はそれぞれプロファイルベース手法，ストリームベース手法，仮想拡張セット手法の結果を示している．

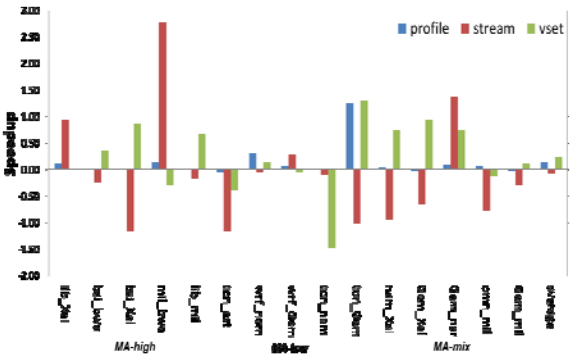

図 2：評価結果

図 2 より，通常の実行に比べてヘルパースレッドによりキャッシュ置換の補助を行うことで多くの場合で性能が向上していることがわかる．キャッシュミス情報をもとに，再利用性を予測して，キャッシュのデータ置換の際に，どこにラインを挿入するかを最適化することで，キャッシュミスが減少するためである．提案手法を用いることで実際に L2 キャッシュヒット率を向上させることができることもわかっている．

特に，プロファイルベース手法と仮想拡張セット手法を用いた場合に性能向上が大きい．このプロファイルベース手法により，最大で 11.5%の性能向上が得られることから，あらかじめプロファインリング実行ができるような状況では，効果的な手法であると考えられる．

仮想拡張セット手法は，場合によってはプ

ロファイルベース手法よりも良い性能を達成できることもある．また，平均で見た場合も，3.9%の性能向上を達成しており，プロファイルベース手法の平均 1.8%よりも良い結果が得られている．

上記のように，ある 1 つの手法が種々のパターンで効果的に動作するとは限らない．提案手法は，各コアで実行するアプリケーションの組み合わせや，状況に応じてヘルパースレッドのプログラムを変更することで，様々な場合にも効果的に動作するように最適化することができるため有用である．これは，ハードウェアベースの手法に比べても，本提案手法の大きな利点であると考えられる．

４．２ FPGA アクセラレータを用いた OpenCL プログラムの高速化の研究

(1) 背景

特定の計算に特化したアクセラレータが注目を浴びている．特に FPGA は現状多くの場面において GPU よりも計算が遅い分，消費電力が小さいという特徴がある．単位消費電力あたりの計算速度を GPU より改良することができれば，ある程度の計算速度かつ低消費電力が求められる用途では，GPU よりも FPGA のほうが適していると考えられる．

従来までは Verilog や VHDL などのハードウェア記述言語を用いて FPGA での計算処理を行うためのコードを記述する必要があり，一般のユーザがそれらの言語を用いてプログラムをするのは難しい．しかし，アクセラレータによる高速化をサポートした C 言語による並列コンピューティングのためのフレームワークの一つである OpenCL が，近年 FPGA にも適用できるようになっている．しかし，OpenCL を FPGA に適用した際の性能や性能最適化のコード変形の影響は未だ十分に解析されていない．そこで本研究では FPGA アクセラレータを用いた OpenCL プログラムの高速化を試みた．

本研究では OpenCL による FPGA の評価・解析をする上でまず有効と考えられる高速化手法として，「ループアンローリング」，「メモリ参照の一意性保証」，「演算ユニット数指定」，「ベクトル化」を考える．

(2) 基本評価

実験環境：本実験では Altera 社の Stratix V GXEA7 の FPGA を用い，開発環境は Altera SDK for OpenCL 15.0，および Quartus 15.0 である．評価では，FPGA と GPU(NVIDIA Tesla K20X)の実行時間を比較する．ベンチマークには、OpenCL のベンチマークである rodinia ベンチマークのうちの Backprop と Kmeans のプログラムを用いた．実際のアクセラレータでの実行時間は，計算時間(以下カーネル実行時間)とメモリ転送時間から成り立っている．カーネル実行時間とは，アクセラレータ上で計算に要した時間そのものを表しており，メモリ転送時間とは，ホスト CPU 側からアクセラレータ側へのデータ転送時間(変数やメモリなど）と，アクセラレータ側からホスト CPU 側へのデータ転送時間(計算結果など)である．本研究ではカーネル実行時間に着目し，メモリ転送時間は評価には含めない。

最適化なしの評価結果：以下に結果を示す．

・Backprop FPGA: 0.0556s, GPU:0.0013s
・Kmeans FPGA: 14.99s, 0.1134s

GPU と比較して，FPGA アクセラレータを用いた場合には遅い結果となっていることがわかる．

ループアンローリングの評価結果：以下にループアンローリングをした場合の結果を示す．

・Backprop: 0.034s
・Kmeans : 2.983s

両ベンチマークともにループアンローリングを行うことでカーネル実行時間に短縮に繋がっている．なお，これはカーネルコードにある for 文をループアンローリングした結果であり，Backprop ベンチマークの場合は 5 イテレーション分，Kmeans の場合では 34 イテレーション分のアンローリングを行った．

メモリ参照の一意性保証の評価結果：以下にメモリ参照の一意性保証をした場合の結果を示す．

・Backprop: 0.055s
・Kmeans : 15.05s

評価したベンチマークプログラムではカーネル実行時間の短縮には繋がっていない．

演算ユニット数指定の評価結果：以下にコンピュートユニット数を 3 に増やして実行した場合の結果を示す．

・Backprop: 0.028s
・Kmeans : 13.41s

コンピュートユニット数を増やすことで Backprop の場合，最大 0.49，Kmeans の場合最大 0.89 までカーネル実行時間が短縮されていることがわかる．

ベクトル化の評価結果: simd2. simd4. simd8, simd16 と設定した場合の結果を以下に示す．

・Backprop
  - SIMD2: 0.016s
  - SIMD4: 0.010s
  - SIMD8: 0.007s
  - SIMD16:0.004s
・Backprop
  - SIMD2: 14.96s
  - SIMD4: 14.95s
  - SIMD8: 15.25s
  - SIMD16:15.79s

Backprop の場合はカーネル実行時間の短縮に繋がっているが，Kmeans についてはわずかしかカーネル実行時間の短縮に繋がっていないことがわかる．

(3) コード変更による更なる高速化手法

CU 数の増加の際のメモリアクセス混雑緩和手法：Kmeans の結果より，コンピュートユニット数を増やすことにより，逆にグローバルメモリの混雑によりカーネル実行時間が長くなっている．そこで，ローカルメモリ活

用によりこれを回避することを考えた

Kmeans において 3.256s(コンピュートユニット数 1，アンローリング 17 回)だったものが，ローカルメモリを用いた結果，1.289s(コンピュートユニット数 2，アンローリング 17 回)に高速化した．ローカルメモリを使用したことで，グローバルメモリへのアクセス混雑が減少したため，実際にコンピュートユニット数が増加することによるカーネル実行時間の短縮が図れることがわかった．

ベクトル化不能のカーネルへの対処手法：Kmeans においてベクトル化（SIMD 化）の効果が見られなかった理由を考察した結果，グローバル ID によって処理に違いがあり，さらにその処理にループが含まれている場合，ループアンローリングができず、ベクトル化不能になることがわかった．グローバル ID による条件分岐があるため，ループアンローリングがうまく行えず，結果としてベクトル化が行えなかったと考えられる．

そこで，完全ループアンローリングと，For 文と if 文の入れ替えを行う方法が考えられるが，両者ともにリソース制約などの制限もあるが，今回はグローバル ID による条件分岐を行わずに，どのグローバル ID に対しても計算を実行し，比較・更新の際のみグローバル ID の条件判定を行う戦略を考案しコードの書き換えを行った．このコードの書き換えにより，ベクトル化可能となりコンパイルにも成功した．この結果，simd4 を指定した場合にもともと 2.732s の実行時間であったものが ベクトル化により 1.430s まで実行時間を短縮することに成功した．

５．主な発表論文等

〔雑誌論文〕（計　1 件）

1) Son T. Nguyen, Masaaki Kondo, Tomoya Hirao, and Koji Inoue, “A Prototype System for Many-core Architecture SMYLEref with FPGA Evaluation Boards”, IEICE Transactions on Information and Systems, Vol.E96-D, No.8, pp.1645-1653, Aug. 2013.

〔学会発表〕（計　7 件）

2) グェンチュオンソン，レイ ジャオ，近藤正章，平尾智也，井上弘士，“FPGA を用いたメニーコア・アーキテクチャ SMYLEref の評価環境の構築”，先進的計算基盤システムシンポジウム SACSIS2012，2012 年 5 月．

3) 松本洋平，近藤正章，和田康孝，本多弘樹，“GPU における細粒度パワーゲーティング向けスレッド発行制御手法の検討”，情報処理学会計算機アーキテクチャ研究会，2013 年 3 月．

4) 橋本 崇浩，近藤 正章，和田 康孝，本多弘樹，“マルチコア・プロセッサ向けのヘルパースレッドによるキャッシュ制御支援手法の提案”，先進的計算基盤システムシンポジウムSACSIS2013，2013年5月．

5) 宮部 創一，近藤 正章，和田 康孝，本多弘樹，“電力モード協調によるプロセッサと主記憶の省電力化の検討”，先進的計算基盤システムシンポジウムSACSIS2013，2013年5月．

6) 佐々木沢，近藤正章，和田康孝，本多弘樹，“次世代 3 次元実装メモリのメモリネットワーク構成に関する初期検討”，情報処理学会計算機アーキテクチャ研究会，2014 年 12 月．

7) Yuan He, Masaaki Kondo, Takashi. Nakada, Hiroshi. Sasaki, Shinobu. Miwa, Hiroshi Nakamura, “Runtime Multi-Optimizations for Energy Efficient On-chip Interconnections, The 33rd IEEE International Conference on Computer Design (ICCD'15), Oct. 2015.

〔招待講演〕（計　5 件）

1) 近藤正章，“メニーコア・アーキテクチャ SMYLEref とその評価環境”，第 4 回アクセラレーション技術発表検討会，2012 年 9 月．

2) Masaaki Kondo, “SMYLEref: A Reference Architecture for Manycore-Processor SoCs”, Asia and South Pacific Design Automation Conference (ASPDAC 2013), 2013年1月.

3) Masaaki Kondo, “SMYLEref: A Reference Architecture for Manycore-Processor SoCs”, 13th International Forum on Embedded MPSoC and Multicore (MPSoC2013), 2013年7月.

4) Masaaki Kondo, “Evaluating Power-Efficiency for an Embedded Microprocessor with Fine-Grained Power-Gating”, 14th International Forum on Embedded MPSoC and Multicore (MPSoC2014), 2014年7月.

5) Masaaki Kondo, “Assisting Cache Replacement by Helper-Threading for MPSoC”, 15th International Forum on MPSoC for Software-defined Hardware (MPSoC2015), 2015年7月.

〔図書〕（計　0 件）

〔産業財産権〕

○出願状況（計　0 件）

○取得状況（計　0 件）

〔その他〕

http://www.hal.ipc.i.u-tokyo.ac.jp/

６．研究組織

(1)研究代表者

近藤 正章（KONDO, Masaaki）

東京大学・大学院情報理工学系研究科・准教授

研究者番号：３０３７６６６０