様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成　３０　年　　６　月　１５　日現在

機関番号：　３２６８９
研究種目：　基盤研究(C)（一般）
研究期間：　2015～2017
課題番号：　１５Ｋ００００５

研究課題名（和文）フラグによりCPUとアクセラレータが連携するヘテロジニアスマルチコアに関する研究

研究課題名（英文）A research on a heterogeneous multicore that enables flexible cooperation among CPUs, accelerators and data transfer units on a chip

研究代表者

木村　啓二（Kimura, Keiji）

早稲田大学・理工学術院・教授

研究者番号：５０３１８７７１

交付決定額（研究期間全体）：（直接経費）　　3,600,000円

研究成果の概要（和文）：本研究では、CPU、アクセラレータ、及びデータ転送ユニットの柔軟な連携を可能とするヘテロジニアスマルチコアのコンパイラ及びアーキテクチャを開発した。本研究による主な成果の一つとして、アクセラレータ用LLVMバックエンドコンパイラを含むコンパイルフローを開発し、コンパイルしたプログラムを開発したFPGAテストベッドで評価したところ、1CPU実行に対して24.91倍の性能向上が得られたことが挙げられる。

研究成果の概要（英文）：We developed a heterogeneous multicore architecture and its compiler flow, which enable flexible cooperation among CPUs, accelerator cores, and data transfer units, which is a kind of extended DMA controller, in a multicore chip. One of the main achievements in this research project is that a program parallelized by the developed compiler flow including LLVM backend for the accelerator core obtains 24.91x speedup on the heterogeneous multicore on an FPGA test bed, which is also developed in this research.

研究分野：　計算機システム

キーワード：　ヘテロジニアスマルチコア　自動並列化コンパイラ　アクセラレータ

１．研究開始当初の背景

組込みシステムから高性能計算システムに至るまで、特定の計算を高速に行うことができる GPU 等のアクセラレータが広く使われている。これらアクセラレータは、近年では電力当たりの性能向上のための、コンピュータシステムの重要な構成要素となっている。

しかしながら、これらヘテロジニアスシステムに対するプログラミング開発では、アクセラレータ駆動等の制御に必要なオーバーヘッド、及び CPU とアクセラレータ間のデータ転送オーバーヘッドに留意する必要が有り、多くのシステムではこれら制御とデータ転送のオーバーヘッドが大きく、CPU とアクセラレータを柔軟に連携しながら効率よく運用することが非常に困難となっている。

このような問題を解決するため、従来から上記オーバーヘッドを低減するための手法が提案及び利用されている。例えば NVIDIA 社の CUDA や、AMD 社等が策定している HSA（Heterogeneous System Architecture）では、CPU とアクセラレータ間の非同期データ転送や、アクセラレータ用タスクキュー等により、CPU とアクセラレータをなるべく同時に動作させ、上記の制御及びデータ転送オーバーヘッドを隠蔽可能とするような仕組みが用意されている[1,2]。しかしながら、アクセラレータと CPU の非同期動作を意識したプログラミングは困難であり、コンパイラによる自動最適化が望まれる。

２．研究の目的

上記のような背景を踏まえ，本研究ではヘテロジニアスマルチコア用コンパイラと、このコンパイラにより最適化されたコードを効率よく実行可能なヘテロジニアスマルチコアアーキテクチャに関する研究を行った。

図 1 に、本研究が対象とする同一プロセッサコア内部に CPU、データ転送ユニット（DTU）、及びアクセラレータ（ACC）を持つヘテロジニアスマルチコアアーキテクチャの 1 コア分のブロック図を示す。本アーキテクチャの特徴は CPU、DTU、及び ACC がそれぞれ非同期で動作可能であり、これらがローカルメモリ上のフラグ授受を通して同期し連携することである。本図では、CPU、DTU、ACC が同一プロセッサコア内でローカルメモリを共有し、そのローカルメモリにはフラグ変数 A と B が配置されている。そして、CPU がフラグ変数 A を更新し(1)、DTU がそれを検知すると(2)、DTU はデータ転送を開始して ACC が利用するデータをローカルメモリにロードする(3)。データ転送終了後、DTU はフラグ変数 B を更新し(4)、ACC がそれを検知すると(5)、アクセラレータはローカルメモリにロードされたデータを用いてその処理を開始する(6)。各要素ではフラグ送信後は各々別の処理を行うことができ、各処理のオーバーラップ実行が可能となる。すなわち、各要素での処理を積極的にオーバーラップさせることでオーバーヘッドを隠蔽し、効率的なプログラム実行が可能となる。

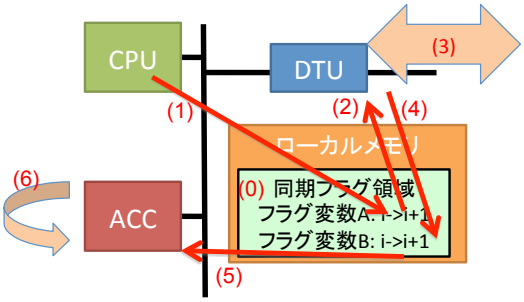

図 1: フラグセット・チェックにより連携する CPU，ADD，DTU のブロック図

以上の議論を前提とし、本研究ではソースプログラムからオーバーヘッド隠蔽に適した CPU、DTU 及び ACC のタスクを抽出する手法、さらに各々の処理要素を空き時間なく実行可能とするための CPU、DTU 及び ACC のスケジューリング手法を研究する。さらに、本スケジューリング手法を自動並列化コンパイラに実装し評価を行う。

また、本アーキテクチャでは CPU、DTU 及び ACC が共有できるメモリ上のフラグ変数を配置することにより各要素間で連携を行うため、フラグ送受信の仕組みと共に、メモリ構成方法も重要な検討項目となる。そのため、本研究では前述のソフトウェア技術の開発と共に、フラグ送受信を行う仕組み及びフラグ変数を配置するメモリの構成（メモリアーキテクチャ）に関しても検討を行い、これら検討したアーキテクチャを、アーキテクチャシミュレータを用いて評価する。

３．研究の方法

本研究では、フラグを用いた CPU、DTU（データ転送ユニット）、及び ACC（アクセラレータ）連携に関して、ソフトウェアとアーキテクチャの両面から以下のように進める。

まずソフトウェア・アーキテクチャ共に、一組の CPU、DTU、ACC を連携させるプロトタイプ版の開発を行う。より具体的には、ソフトウェアに関しては、タスク抽出及びスケジューリング手法の検討を行い、そのプロトタイプ版を自動並列化コンパイラに実装する。本手法では、ACC はローカルメモリ上のデータを使って計算を行うことを想定しているため、申請者等のグループが開発したローカルメモリ管理手法との融合もこの時点で検討する[3]。また本研究では、研究対象とするソフトウェア手法をシミュレータで実行するためのコード生成系の開発も行う。アーキテクチャに関しては、メモリ構成及びフラグ授受機構の基本検討を行い、アーキテクチャシミュレータ上に実装する。対象とするアクセラレータのモデルとしては、GPU 等各種アー

キテクチャを、コード生成系やシミュレータへの実装の難易度、及び評価に利用するアプリケーションの特性を合わせて比較検討し決定する。

その後プロトタイプの評価を行うと共に、本手法を複数の CPU、DTU、ACC を持つマルチコアシステムに対応するための拡張を行う。特にソフトウェアでは、マルチコアの各コアに割り当てるタスクの抽出、及び各コア内部の CPU、DTU、ACC に割り当てるタスクの抽出とそのスケジューリングを行うといった階層的な処理が必要になる。さらに、各コアが持つローカルメモリと主記憶との間のデータ転送に加えて、ローカルメモリや分散共有メモリを活用したコア間直接データ転送を行う。

４．研究成果

本研究では上記の通り、アクセラレータやデータ転送オーバーヘッドを隠蔽可能な並列化コンパイル手法と、そのコンパイラによる最適化コードを効率的に動作させることができるヘテロジニアスマルチコアアーキテクチャの両面から研究を行った。研究成果は、「FPGA によるアクセラレータを持つヘテロジニアスマルチコアの評価テストベッドの開発」、「フラグセット・チェックによるアクセラレータ駆動を可能とするヘテロジニアス自動並列化コンパイラフレームワークの開発」、及び「アクセラレータにデータを供給するメモリ管理技術の開発」の 3 項目にまとめることができる。以下にそれぞれについて説明する。

(1) FPGA によるアクセラレータを持つヘテロジニアスマルチコアの評価テストベッドの開発

当初計画では、本研究によるコンパイラ評価のプラットホームとしてアーキテクチャシミュレータの使用を予定していたが、FPGA 上に評価対象ヘテロジニアスマルチコアを構築し、この上でも評価することとした。この変更の主な理由は、研究期間中に FPGA 評価ボードやその開発環境が容易に利用可能となったこと、及びソフトウェアシミュレータよりも FPGA 上に評価対象アーキテクチャを構築した方が高速動作を期待でき結果として評価可能なアプリケーションの幅が広がること、の 2 点である。

FPGA に評価環境を実装するため、利用するアクセラレータの仕様も決める必要がある。本研究では、長年にわたるコンパイラ最適化の蓄積があること、及び LLVM によるアクセラレータ用バックエンドコンパイラ開発の目処が立ったことを理由に、ベクトルプロセッサをアクセラレータとして採用した。ベクトルプロセッサの構成は富士通 VPP を参考にした[4]。特徴としてはベクトルレジスタを持ち、演算パイプ間、ロードストア-演算パイプ間でのチェイニングが可能である点、ベクトル長が可変である点が挙げられる。ベクトル部分

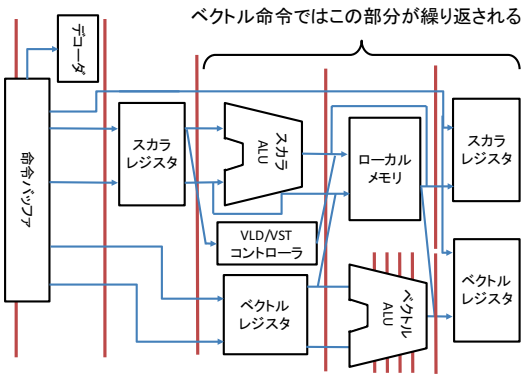

図 2: FPGA 上に実装したベクトルアクセラレータのパイプライン

のデータパスは 256bit であり、8 レーンの単精度浮動小数点演算が可能である。本ベクトルプロセッサのパイプラインの構成を図 2 に示す。

本ベクトルプロセッサをアクセラレータとして搭載するヘテロジニアスマルチコアを FPGA 上に構築した。実装には Arria10 SoC 評価ボードを利用した。また、CPU コアとしては NIOS II/fast を使用した。FPGA の詳細とリソース使用量を表 1 に示す。動作周波数は 100MHz となった。

表 1: FPGA 使用リソース量

	FPGA スペック	本アクセラレータ使用量	NIOS 使用量
LE 数	660M	-	-
ALM 数	251,680	19,274	1,157
DSP 数	1,687	22	3
M20K 数	2,131	263	74

256x256 の行列積と 256x256 の範囲を 9x9 のフィルタを用いる 2 次元コンボリューションにより、FPGA 上に実装したベクトルアクセラレータの評価を行った。評価の結果を表 2 に示す。アクセラレータは行列積において 1467MFLOPS の性能を確認した。これは FPU 付きの NIOS と比較し 117 倍の性能である。コンボリューションにおいて 1340MFLOPS の性能を確認した. これは FPU 付きの NIOS と比較し 77 倍の性能である。

表 2: 演算カーネルを用いた性能評価

カーネル	プロセッサ	MFLOPS
行列積	NIOS FPU 有	12.5
	NIOS FPU 無	0.964
	アクセラレータ	1467
コンボリューション	NIOS FPU 有	17.4
	NIOS FPU 無	0.832
	アクセラレータ	1340

(2) フラグセット・チェックによるアクセラレータ駆動を可能とするヘテロジニアス自動並列化コンパイラフレームワークの開発

フラグセット・チェックによるアクセラレータ制御手法を開発し、早稲田大学で開発している OSCAR 自動並列化コンパイラに実装した。OSCAR 自動並列化コンパイラは従来の並列化コンパイラが扱っているループイテレーションレベルの並列化に加えて、ループや関数呼び出し間の並列性を利用する粗粒度並列性とプログラム実行文単位の並列性を利用する近細粒度並列性を階層的に組み合わせてプログラム全域から並列性を抽出するコンパイラである。本研究ではこれら並列性のうち粗粒度タスク並列性を利用する。そしてコンパイラが、アクセラレータで実行する方が有利であると判断した粗粒度タスクをアクセラレータが CPU と非同期に実行できるようにスケジューリングする。

上記の方式を OSCAR コンパイラに実装し本研究で構築した対象ヘテロジニアスマルチコア用コンパイルフローを図 3 に示す。本コンパイルフローでは逐次の C ソースコードを OSCAR コンパイラに入力し、OSCAR コンパイラで前述の並列化を行うと共にホスト側と汎用 CPU とアクセラレータにタスクをスケジューリングする。スケジューリング後、ホスト CPU コードとアクセラレータ用のコードを各々生成する。生成されたホスト CPU コードから gcc 等のホスト CPU 用コンパイラによりオブジェクトファイルを生成する。同様にアクセラレータ用コードから、図 2 の本研究で開発したベクトルアクセラレータ（VA）用のコード生成を可能とする拡張を施した Clang/LLVM コンパイラでオブジェクトファイルを生成する。最後にこれらオブジェクトコードをリンクしてターゲットのヘテロジニアスマルチコア用の実行バイナリを生成する。

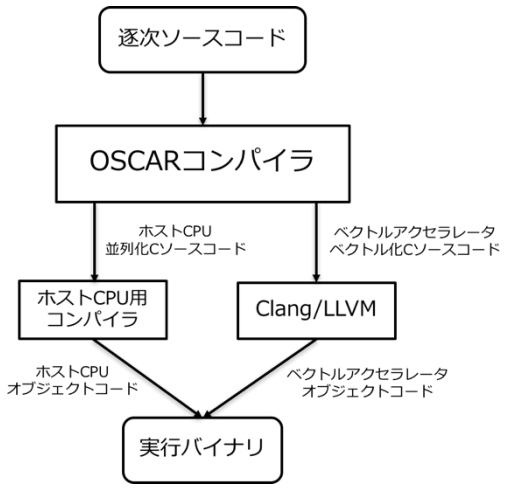

図 3: 開発ヘテロジニアスマルチコアコンパイラのコンパイルフロー

64x64 行列積カーネルプログラムを用いて評価を行った。まず、本コンパイルフローにより生成した実行バイナリを、対象ヘテロジニアスマルチコアを再現するアーキテクチャシミュレータ上で評価した結果を図 4 に示す。図中、グラフの左縦軸は実行クロックサイクル数を示しており、また右縦軸は 1CPU のみの実行での実行クロックサイクル数を 1 とした場合の速度向上率である。評価の結果 1CPU 単体の実行に比べ、1CPU+1VA+CPU 転送では 13.99 倍、1CPU+1VA+DTU 転送では 17.53 倍、2CPU+2VA+CPU 転送では 23.03 倍、2CPU+2VA+DTU 転送では 28.21 倍の速度向上率が得られた。また、1CPU+1VA+CPU 転送の実行に比べて、1CPU+1VA+DTU 転送は 1.25 倍、2CPU+2VA+CPU 転送の実行に比べて、2CPU+2VA+DTU 転送は 1.22 倍の速度向上率が得られた。

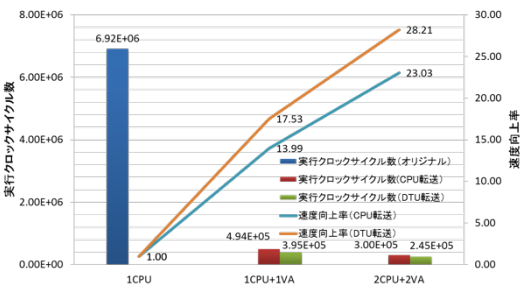

図 4: シミュレータによる行列積の評価結果

さらに、前項で述べた FPGA 上で行列積を評価した結果を図 5 に示す。図中、グラフの左縦軸は実行時間を示しており、また右縦軸は 1CPU のみの実行での実行時間を 1 とした場合の速度向上率である。評価の結果、1CPU 単体の実行に比べ、1CPU+1VA では 24.91 倍の速度向上率が得られた。

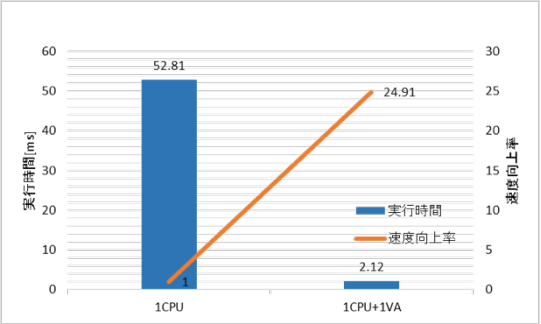

図 5: FPGA による行列積の評価結果

(3) アクセラレータにデータを供給するメモリ管理技術の開発

本研究では、さらにアクセラレータにコア内ローカルメモリからデータを供給するためのメモリ管理手法に関しても研究を行った。本メモリ管理手法は OSCAR コンパイラの粗粒度タスク並列処理を前提に、コア間共有メモリ（主記憶）上のデータをコア内ローカルメモリに収まるように分割し、粗粒度タスクスケジューリングによってコア内外のデータ転送を最小化する技術である。そのための基本

技術として、粗粒度タスク間のデータアクセス範囲を解析し、タスク間のデータ共有量を最大化するループ整合分割手法が従来から提案されている。また、タスク分割及びスケジューリング後に、ローカルメモリに分割されたデータを割り当て、さらにデータ転送オーバヘッドを最小化するようにデータ転送をスケジューリングする手法も提案されている。

しかしながら従来のループ整合分割手法では、例えば巨大な多次元配列を分割する場合であっても単一次元方向でしか分割せず、分割適用後であってもローカルメモリに収まりきらない可能性があった。本研究ではこのループ整合分割を複数ループ間の複数次元に渡るデータアクセス範囲解析により粗粒度タスク及びそのデータを多次元方向に分割可能にし、巨大な配列データをアクセスする場合であってもローカルメモリに格納可能なように分割する技術を開発し OSCAR 自動並列化コンパイラに実装した。

本研究では、まず開発したメモリ管理手法を組み込みマルチコア RP-2 で評価した。RP-2 は 8 コアを搭載のマルチコアで有り、チップ外の主記憶の他に各コアが 32KB のローカルメモリを持つ。評価プログラムとしては、NAS Parallel Benchmark の BT を利用した。BT を、本メモリ管理手法を実装した OSCAR コンパイラにより並列化し RP-2 上で評価した結果を図 6 に示す。図は横軸が使用コア (PE) 数、縦軸がメモリ管理無しの逐次実行時間を基準とした速度向上率を表す。本メモリ管理手法により、4 コア使用時でメモリ管理無し逐次実行に対し 5.76 倍、4 コアメモリ管理無しに対して 1.78 倍の性能向上を得ることができた。本メモリ管理手法を今後ヘテロジニアスマルチコアコンパイルフローに組み込んで評価を行う予定である。

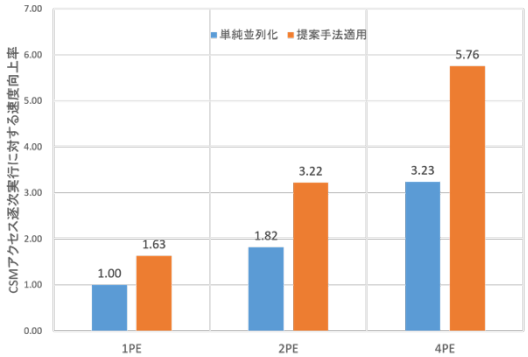

図 6: メモリ管理手法の性能評価結果

<引用文献>

(1) CUDA Zone, https://developer.nvidia.com/category/zone/cuda-zone/

(2) HSA Foundation, http://www.hsafoundation.com/

(3) 中野啓史, 桃園拓, 間瀬正啓, 木村啓二, 笠原博徳, “マルチコアプロセッサ上での粗粒度タスク並列処理のためのコンパイラによるローカルメモリ管理手法”, 情報処理学会論文誌コンピューティングシステム(ACS), Vol. 2, No. 2, pp.63-74, Jul. 2009.

(4) Miura, K., Takamura, M., Sakamoto, Y. and Okada, S.: Overview of the Fujitsu VPP500 supercomputer, Compcon Spring '93, pp.128-130 (1993).

5. 主な発表論文等

〔学会発表〕(計 8 件)

1. Boma A. Adhi, Masayoshi Mase, Yuhei Hosokawa, Yohei Kishimoto, Taisuke Onishi, Hiroki Mikami, Keiji Kimura, Hironori Kasahara, "Software Cache Coherent Control by Parallelizing Compiler", 30th International Workshop on Languages and Compilers for Parallel Computing (LCPC), Oct. 2017. (査読有り)
2. 白川智也, 阿部佑人, 大木吉健, 吉田明正, 木村啓二, 笠原博徳, "階層アジャスタブルブロックを用いた自動マルチコア・ローカルメモリ管理とその性能評価", 第 220 回システム・アーキテクチャ研究発表会 2017-ARC-220(デザインガイア 2017), Nov. 2017.
3. 高橋 健, 狩野哲史, 宮本一輝, 河田 巧, 柏俣智哉, 牧田哲也, 北村俊明, 木村啓二, 笠原博徳, "OSCAR ベクトルマルチコアアーキテクチャのコンパイルフロー構築及び評価", 情報処理学会第 80 回全国大会, 7H-02, Mar., 2018.
4. 柏俣智哉, Boma A. ADHI, 狩野哲史, 宮本一輝, 河田 巧, 高橋 健, 牧田哲也, 北村俊明, 木村啓二, 笠原博徳, "OSCAR ベクトルアクセラレータの FPGA 上での性能評価", 情報処理学会第 80 回全国大会, 7H-03, Mar., 2018.
5. Keiji Kimura, Gakuho Taguchi, Hironori Kasahara, "Accelerating Multicore Architecture Simulation Using Application Profile", 2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Sep. 2016. (査読有り)
6. Kouhei Yamamoto, Tomoya Shirakawa, Yoshitake Oki, Akimasa Yoshida, Keiji Kimura and Hironori Kasahara, "Automatic Local Memory Management for Multicores Having Global Address Space", The 29th International Workshop on Languages and Compilers for Parallel Computing (LCPC), Oct. 2016. (査読有り)
7. 丸岡晃, 無州祐也, 狩野哲史, 持山貴司, 北村俊明, 神谷幸男, 高村守幸, 木村啓二, 笠

原博徳, ″LLVM を用いたベクトルアクセラレータ用コードのコンパイル手法″, 情報処理学会 2016 年並列／分散／協調処理に関する『松本』サマー・ワークショップ (SWoPP 松本 2016) Vol.2016-ARC-221 No.4, Aug. 2016. (若手優秀賞受賞)
8. 影浦直人,和気珠実,韓吉新,木村啓二,笠原博徳, ″OSCAR 自動並列化コンパイラにおける解析時データ構造変換による並列性抽出手法″, 第 153 回ハイパフォーマンスコンピューティング研究発表会, Mar. 2016.

〔その他〕
ホームページ等
http://www.apal.cs.waseda.ac.jp/

６．研究組織
(1)研究代表者
木村 啓二 (Keiji Kimura)
早稲田大学・理工学術院・教授
研究者番号： 50318771