Vol.2011-ARC-197 No.10
Vol.2011-HPC-132 No.10
2011/11/28

OpenCL を用いたパイプライン並列プログラミング API の初期検討

薦田 登志矢[†1] 三輪 忍[†1] 中村 宏[†1]

シングルスレッド性能向上の限界，電力制約の問題から特定アプリケーションに特化したアクセラレータを利用することの重要性が高まっている．これまでのアクセラレータを利用する事例はデータ並列性を利用するアプリケーションを主たるターゲットとしてきた．しかし，特に組み込みシステムにおいてパイプライン並列性を利用することがアプリケーションの性能向上を，与えられた電力制約のもと達成するために重要となる．本稿では組み込みシステムにおいてアクセラレータを利用する場面を想定し，アクセラレータを含むシステム上でパイプライン並列性を利用するアプリケーションを容易にかつ柔軟に実現するためのライブラリを提案する．提案ライブラリではアクセラレータプログラミングの標準として策定された OpenCL を用い，ソフトウェアパイプライニング技術を応用することで，アクセラレータ上におけるパイプライン並列処理を実現すると同時に，パイプライン並列アプリケーションを開発するための簡潔なユーザーインタフェースを提供する．プロトタイプシステムの評価により，パイプライン並列処理におけるタスクスケジューリングや通信バッファの管理といったシステムの複雑さをプログラマから隠蔽しつつ，アクセラレータデバイス上においてパイプライン化による性能向上を達成できることが分かった．

1. はじめに

近年，GPU に代表される特定アプリケーションの持つ並列性やメモリ特性に特化したアーキテクチャを持つプロセッサ，いわゆるアクセラレータを利用する研究が盛んに行われている．今後のマイクロプロセッサでは，消費電力が実質的なシステム性能向上のボトルネックになるため，特定のアプリケーションに特化することで汎用 CPU に比べて格段に高い電力効率を実現する専用ハードウェアユニットを，汎用コンピューティングの中で利用する技術の重要性はますます高まっている[3)]．

こうしたプログラマブルなアクセラレータによる性能向上の恩恵を受けるためには，アプリケーションが並列化されている必要があり，並列化アプリケーション開発の重要性がますます高まっている．複数のコアを利用できるプロセッサはアクセラレータを含めて数多く存在し，プラットフォームに依存したコードの記述が要求される状況であった．こうした中，並列化プログラムのポータビリティを確立する目的で OpenCL が策定された[7)]．OpenCL を用いた並列化アプリケーション開発を行うことで，クロスプラットフォームに動作する並列化アプリケーションの開発が可能になった．多くの CPU ベンダ，GPU ベンダ，SoC ベンダが自社デバイスにおける OpenCL のサポートを開始・検討しており，様々なメモリアーキテクチャを持つデバイス上に効率的な OpenCL ランタイムシステムを構築する試みもなされている 4)11)．OpenCL の策定を機に多様な分野において，アクセラレータデバイスを活用することによるシステム性能の向上が期待されている．

これまでのアクセラレータ利用に関する研究で加速の対象としているアプリケーションはデータ並列性を利用するものが中心であった．しかし，特に性能制約・電力制約ともに厳しい組み込み向けのマルチメディア処理システムでは，タスクの処理特性に合わせて特性の異なるコア群を用いて，パイプライン並列性を利用したアプリケーションを構築することが重要である[13)]．本稿では，特に組み込みシステムにおけるアクセラレータデバイス利用を念頭に，アクセラレータデバイスを用いつつ，パイプライン並列性を利用するためのプログラミング技術について検討する．アクセラレータを用いるためのプログラミング環境として，前述の OpenCL を用いてパイプライン並列性を利用するアプリケーションを構築する場合，タスクのスケジューリングやバッファの管理といったパイプライン並列処理に伴う煩雑な処理をプログラマが手動で記述しなければならない，という問題が生じる．従来のシステム，例えば Linux において並列タスク間の通信 API を用いてパイプライン並列処理を実現する場合には，このようなタスクのスケジューリングやバッファの管理はシステム側が提供するサービスであり，プログラマから隠蔽されている作業である．

そこで我々は，OpenCL を用いたパイプライン並列処理を行うアプリケーション開発に関する上述の煩雑さを隠蔽するためのパイプライン並列ライブラリを提案する．提案ライ

†1 東京大学
The University of Tokyo

Vol.2011-ARC-197 No.10
Vol.2011-HPC-132 No.10
2011/11/28

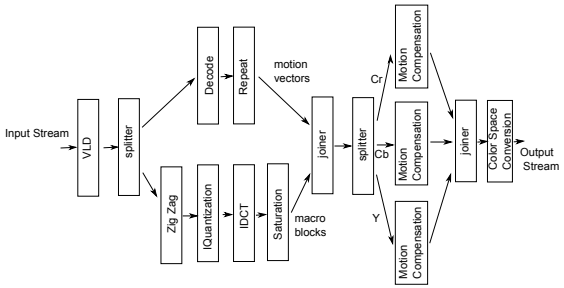

図 1 mpeg デコーダにおけるタスクグラフ
Fig. 1 A Task Graph of MPEG2 Decoder System

ブラリは，現状 (Version1.1) の OpenCL の仕様に手を加えることなく，OpenCL プラットフォーム上で動作するライブラリレイヤによって，パイプライン並列処理の記述に伴う煩雑さをプログラマから隠蔽しつつ，アクセラレータを含むコンピューティング環境におけるパイプライン並列処理を実現する．

本稿の構成は，以下のようになっている．

２章では，OpenCL プラットフォームおよびパイプライン並列処理について説明する．続く３章で，ソフトウェアパイプライン技術を応用したパイプライン並列処理を用いて OpenCL 上でパイプライン並列処理を実現する方法について述べる．４章では，タスクスケジューリングおよび通信バッファの管理をユーザーから隠蔽する OpenCL パイプライン並列ライブラリを提案し，システムの概要を述べる．５章で提案システムの初期評価結果を示し，６章において関連研究を述べ，７章でまとめと今後の課題を述べる．

2. 背　　景

2.1 OpenCL プラットフォーム

アクセラレータデバイスを利用するための最初の標準的なプラットフォームとして策定された OpenCL におけるアプリケーション開発について概要を述べる．OpenCL プラットフォームは，アプリケーション全体を中央制御するホストマシンと計算処理を担当する OpenCL デバイスの 2 種類のマシンからなる．ホストおよびデバイスごとにメモリ空間が異なっており，ホスト-デバイス間のデータ転送はプログラマによって明示的に行われる．また，デバイス側の処理を記述するために，OpenCL C と呼ばれる C 言語をベースとした言語が定義されている．OpenCL C で記述されたデバイス上の処理は，OpenCL カーネルと呼ばれる実行単位により管理される．

プログラマはホスト側で動作し OpenCL カーネルの実行やホスト-デバイス間のデータ通信を制御するためのホストマシン用のコードとともに，アクセラレータデバイス上で実行され，加速の対象となるカーネルコードを記述する．カーネル内部は，work-item と呼ばれる実行単位によってデータ並列実行される．カーネルコードを記述するための OpenCL C は，データ並列性を利用する仕組み，豊富なビルトイン関数，プログラマによる明示的なメモリ管理機構を提供しており，これらを用いてアクセラレータデバイスを効率的に利用できるようになっている．また，コマンドキューと呼ばれるタスク管理機構を用いてデバイス内およびデバイス間においてタスク並列性を利用した OpenCL カーネルの実行が可能になっている．OpenCL では，同期ポイント以外でのメモリ内容の一貫性を保証しないメモリモデルを採用しているため，タスク間でデータ通信を行うタイミングはカーネルの実行開始時および終了時に限定される⋆1．

2.2 パイプライン並列処理

マルチメディア処理に代表されるストリーム処理システムには，データ並列性・タスク並列性・パイプライン並列性が豊富に内在しており，これらの並列性を利用したアプリケーションの高速化が重要である[14]．特に，ハイエンドな組み込みシステムでは従来からパイプライン並列性を利用することが行われている[5]．

本稿が対象としているパイプライン並列処理が適しているアプリケーションの例として，図 1 に MPEG-2 デコーダのタスクグラフを示す[6]．MPEG-2 は，広く普及しているデジタル動画の圧縮用標準規格である．デコーダは複数の機能ブロックから構成される．データ並列性を有するブロック，シーケンシャルな処理が多いブロックといったように異なる特性を持つ機能ブロックがパイプライン式にデータを受け渡しながらストリーム処理を行う．

3. OpenCL を用いたパイプライン並列処理

この章では，OpenCL を用いてアクセラレータデバイス上でパイプライン並列処理を記述する方法について検討する．

3.1 ソフトウェアパイプライニングを用いたパイプライン並列処理の実現

本稿では，OpenCL のタスク並列機能を用いたパイプライン並列処理を行うためにソフ

⋆1 データ並列実行に特化したアーキテクチャにおけるメモリシステムでは，メモリレベル並列性を十分に利用するためにハードウェアによるキャッシュコヒーレンスを保障しない[8]．このため，OpenCL カーネルの実行モデルでは実行中における異種カーネル間でのデータ共有を禁止している．

Vol.2011-ARC-197 No.10
Vol.2011-HPC-132 No.10
2011/11/28

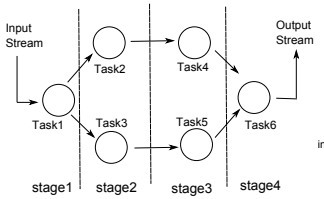

図 2 通信グラフのパイプライン分割
Fig. 2 Pipeline Splitting of a Communication Graph

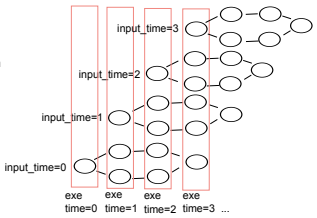

図 3 ソフトウェアパイプライニングを用いた並列実行
Fig. 3 Parallel Task in Pipeline Execution

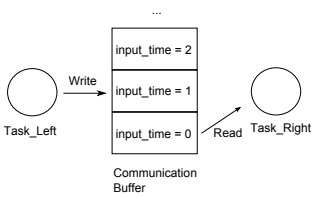

図 4 通信バッファの多重化
Fig. 4 Multiplication of the Communication Buffer

トウェアパイプライン技術を応用する．

図 2 に示すようなタスクグラフを持つアプリケーションを例にソフトウェアパイプラインを用いたパイプライン並列化について説明する．図では，丸でタスクを矢印でタスク間の通信を表現しており，矢印の方向にデータが受け渡される．また，ここでは仮にステージを 4 つに分割することにし，ステージ 1 はタスク 1，ステージ 2 はタスク 2，3，ステージ 3 はタスク 4，5，ステージ 4 はタスク 6 から構成されている．また，タスク 1 は入力ストリームからデータを受け取り，タスク 6 は出力ストリームにデータを送り出すとする．

パイプライン処理では，連続的に入力されるデータをステージ間で受け渡しながら，処理を行うことで異なるステージに存在するタスク群を並列に実行することができる．この様子を，図 3 に示す．図では，実行時において並列に実行されるタスクを縦方向に示しており，一度の並列実行単位 (図中，赤色の長方形) が実行されるごとに同期がとられる．縦方向に，横にずれながら重なっている通信グラフは異なる時刻（*input_ time*）に入力されたデータが処理される様子を表している．このように，時刻をずらしながら並列実行を行うことでタスク間のデータ依存性を満たしつつ，ステージ間におけるタスク並列性を利用できる．

OpenCL における実装では，各タスクの一回分の処理実行が OpenCL カーネルの一回の実行に相当する．カーネルの並列実行，および同期の制御はホスト側の制御コードに実現する．パイプライン並列実行部分ではループ処理によって入力データが終了するまで，連続的に OpenCL カーネルをコマンドキューに投入していく．

OpenCL では，カーネル実行中における同一メモリ領域の共有を禁止していることから，タスク間の通信に用いるバッファは，多重化しておかなければならない．処理対象のデータが入力された時刻から，多重化されたバッファのうちどのバッファを用いるかを決定し，並列実行されるカーネルが同時に同じ位置のメモリを参照しないようにするとともに，正しいデータが後段のタスクに受け渡されるよう，バッファのアクセスを管理する必要がある．図 4 に，多重化された通信バッファにおける通信を模式的に示す．図の中で，*Task_Left* は，次のサイクルにおける並列実行において *Task_Right* が使用するデータを図中下から二番目の領域に書き込み，*Task_Right* はひとつ前のサイクルで *Task_Left* が一番下の領域に書きこんだデータを読み込んで処理を行っている．必要な循環バッファの深さは，タスクのスケジューリングに依存する．図 3 におけるスケジューリングの例では，バッファに書き込まれたデータは最大でも次のサイクルにはデータの受け手によって読まれるため，この場合の循環バッファの深さは 2 で十分である．データの受け渡しが複数サイクルにまたがる遅延を伴う場合，バッファはその分だけ深くする必要がある．

OpenCL では，デバイスごとにメモリ空間が異なっていることから複数のデバイスをまたがってパイプライン並列処理を実現する場合，これらの通信バッファの多重化に加えてデバイス間でデータを明示的に転送する必要がある．単一のデバイス内におけるパイプライン並列処理では，このようなデータ通信の必要は生じない．

ソフトウェアパイプライニングによって実現できるパイプライン並列処理は，タスク間の通信が規則的な場合に限られる．これはタスク間の通信が規則的な場合のみ，タスクの実行順序が静的に定まるからである．静的にスケジューリング可能なパイプライン処理を行うシステムは *Synchronous Data Flow*[10] として知られており，信号処理分野を含む多くのアプリケーションがこのシステムモデルを用いて記述できることが知られている

3.2 非同期なパイプライン並列処理

ここで非同期なパイプライン処理とは，各タスクの実行順序があらかじめ決定できないようなパイプライン処理のことを指している（例えば，入力データや外部のイベントに依存して実行されるタスクが存在する場合や，通信パターンの中に分岐が含まれるような場合）．この場合，各タスクの実行順序をあらかじめ決定することができないため，共有データアクセスにおけるレースコンディションの防止，およびデータ書き込み・読み込み処理における同期をとるために，ロックや条件変数といった細粒度な同期プリミティブを利用する必要がある．細粒度な同期プリミティブをサポートしていない OpenCL では，非同期に実行されるパイプライン並列処理を効率的に実現することは難しい．

Vol.2011-ARC-197 No.10
Vol.2011-HPC-132 No.10
2011/11/28

プログラム 1　サンプルコード

```
OclPipeline pp;
// set I/O Stream
pp.addInputStream("input1", io_descripter1);
pp.addOutputStream("output1", io_descripter2);

// regist the kernels (tasks)
pp.addKernel("kernel1", size_param1, param1);
pp.addKernel("kernel2", size_param2  param2);
pp.addKernel("kernel3", size_param3, param3);
pp.addKernel("kernel4", size_param4, param4);
pp.addKernel("kernel5", size_param5, param5);
pp.addKernel("kernel6", size_param6, param6);

// regist the communication between kernels
pp.connect("input1","kernel1",  input_buffer_size);
pp.connect("kernel1","kernel2",buffer_size1);
pp.connect("kernel1","kernel3",buffer_size2);
pp.connect("kernel2","kernel4",buffer_size3);
pp.connect("kernel3","kernel5",buffer_size4);
pp.connect("kernel4","kernel6",buffer_size5);
pp.connect("kernel5","kernel6",buffer_size6);
pp.connect("kernel6","output1",output_buffer_size);

// pipeline parallel execution on OpenCL Devices
pp.execute();
```

図 5　提案ライブラリを用いた通信パターンの記述
Fig. 5　Description of the Communication Pattern using the Proposed API

4. パイプライン並列プログラミング API

4.1 提案 API の目的と概要

前章において，静的にスケジューリングが決定するパイプライン並列処理を，OpenCL を用いて実現する方式について説明した．

このようなパイプライン並列処理を実装する過程においてプログラマは，アクセラレータデバイスを利用する各タスク内の処理を記述することに加え，タスク間の通信パターンを考慮して以下の作業を行わなければならない．

(1)　与えられたタスク群と通信パターンから，ロードバランスを考慮しパイプラインステージ分割を決定する．
(2)　パイプライン化されたタスクの並列性を利用しながら，複数デバイス上において効率的なタスクマッピング・スケジューリングを決定する．
(3)　決定したタスクマッピング・タスクスケジューリングに従い，OpenCL デバイスごとに異なるメモリ空間の間で適切なデータ転送処理を行いつつ，適切な同期処理を行いながら各タスクを各デバイス上で実行するホストコードを記述する．

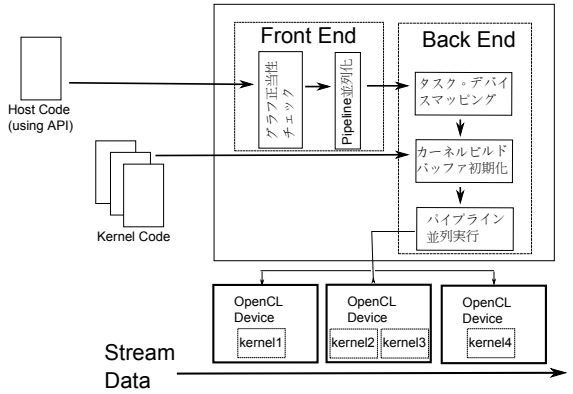

図 6　OpenCL パイプライン並列ライブラリの内部アーキテクチャ
Fig. 6　The OpenCL Pipeline Parallel Library Internal Architecture

これらの作業はアプリケーションが大規模化し，タスク数，通信パターンが複雑になるにつれて急速に複雑になっていく作業であり，またプログラム自体も複雑になっていく．これに伴いプログラム開発の生産性は低下し，バグも混入しやすくなる．従来のパイプライン並列処理をサポートするシステムでは，これらの煩雑な処理は，システム側がサービスとして提供することでプログラマからは隠蔽されてきたものである．

ここでは，OpenCL においてパイプライン並列処理を行うアプリケーションを記述する場合に生じるこのような問題を隠蔽するための OpenCL パイプライン並列ライブラリを提案する．提案する OpenCL パイプライン並列ライブラリは，現状の OpenCL に拡張を入れることなく実現できる．これにより，OpenCL を用いたアプリケーション開発を行う利点であるプログラムのポータビリティを損なうことがない．

4.2 並列ライブラリのアーキテクチャ

プログラマは，各タスク内における処理を OpenCL C を用いて記述する．これに加えて，提案ライブラリの API を用いて各タスク間の通信パターンと入力ストリームおよび出力ストリームを明示的に指定する．図 5 に，提案ライブラリを用いて図 2 に示した通信グラフを記述した場合の疑似コードを示す．提案ライブラリでは，*addKernel* 関数およ

Vol.2011-ARC-197 No.10
Vol.2011-HPC-132 No.10
2011/11/28

び *addInput(Output)Stream* 関数を用いてタスク群，および入出力ストリームを登録する．これは，通信グラフにおけるノードを指定することに相当する．通信パターンの記述には，*connect* 関数を用いる．これは，通信グラフにおいて辺を指定することに相当する．このとき，当該通信路を通る最大のデータサイズを指定する．

ライブラリ内では，ユーザーから入力された通信パターンからパイプライン並列性を抽出し，適切なタスク・デバイスマッピング，スケジューリングを行い複数の OpenCL デバイス上で各タスクを並列実行する．図 6 に，提案するパイプライン並列プログラミングライブラリのシステム概要図を示す．ライブラリはまず，ユーザーから与えられた通信グラフを解析し，これらが静的にスケジューリング可能なものであるかどうかをチェックする．その後，タスクをステージに分割し，(図中，Pipeline 並列化)，タスクの実行デバイスおよびスケジューリングを決定する (図中，タスク・デバイスマッピング)．このタスクスケジューリング結果に基づき，ライブラリ内のパイプライン並列実行エンジンが，デバイスメモリ，各タスクのカーネルビルド，等の初期設定を行った後，適切な同期・デバイス間のデータ通信を行いながらタスク群を複数の OpenCL デバイス上で並列実行する (図中，パイプライン並列実行)．図では，３つの OpenCL デバイス上で，４つのカーネルがストリームデータを受け渡しながら実行されている様子を示している．

4.3 ライブラリ内における処理

ここでは，提案ライブラリ内で行われる通信グラフの正当性のチェック，パイプライン化，タスクマッピング，タスクスケジューリングの概要について述べる．

はじめにタスク群をパイプラインステージに分割する方法を述べる．まず，入力ストリームとつながったノードを入口ノード，出力ストリームとつながったノードを出口ノード，とした場合における通信グラフのフィードフォーワードカットセットを求める．カットセットとは，グラフの辺の集合でありこれらの辺がなくなった場合，入口ノードと出口ノードが分離されるような辺の集合である．フィードフォワードカットセットは，カットセットのうち，全ての辺が入口側から出口側へデータ通信を行う辺であるカットセットである．これらは一つの通信グラフに複数存在し，通信グラフにおけるステージ境界の候補となる．複数の OpenCL デバイスを用いた効率的なパイプライン並列処理のためには，ステージ間におけるロードバランスや，タスクと実行デバイスのマッピング，デバイス間のデータ通信により生じる通信オーバーヘッドといった事柄を考慮した性能モデルを用いて，タスク群のステージ分割・実行デバイスへのマッピングを協調的に決定する必要がある．５章で評価に用いたプロトタイプシステムでは，全てのタスクが同じ実行時間を持つと仮定してステージ分割を行ったのち，全てのデバイスにおいて均等にタスクが実行されるようにタスクマッピングを行っているが，効率的なタスクのステージ分割とマッピングを実現するための性能モデルの構築と，最適解の探索アルゴリズムの構築は今後の課題である．

OpenCL Host	Intel Core i7 Processor(4 コア)
OpenCL Device	Intel Core i7 Processor(4 コア)
Operating System	Linux 2.6

図 7 評価に用いた OpenCL プラットフォーム
Fig. 7 Evaluation Setup of OpenCL Platform

パイプライン化され，実行デバイスが決定したタスク群は，３章で説明したようにステージごとに処理するデータの時刻をずらしながら，複数の OpenCL デバイス上で並列に実行される．ただし，同一ステージ内に存在するタスク間ではデータ依存関係が存在する可能性があるため，データの依存関係を考慮したスケジューリングを行う必要がある．提案ライブラリ内では，コンパイラ内部における命令スケジューリングで用いられるリストスケジューリングを用いたステージ内タスクのスケジューリングを行う[1)]．

また，デバイス間におけるデータ通信は全て一度ホストのメモリを介して行われる．これは，OpenCL ではデバイス間でデータを直接やり取りする方法が提供されていないためである．異なるステージ間でのデバイス間データ通信は，スケジューリング時にデータ転送遅延を挿入する，すなわち疑似的にパイプラインステージ数を増加させることで，タスク実行とオーバーラップさせることが可能であり，デバイス間で生じるデータ通信コストを最小限に抑えることができる．

5. プロトタイプシステムの評価

5.1 評 価 環 境

提案する OpenCL パイプライン並列化ライブラリは OpenCL 実行環境がインストールされた Linux 上で動作する．一直線の形状を持つタスクグラフを持つアプリケーションを，単一の OpenCL デバイス上でパイプライン並列実行する，という機能に限定して提案ライブラリのプロトタイプを実装し，評価を行った．各タスクは，入力配列に対して一定量の演算処理を行い出力配列へ書き込み処理を行う，というシンセティックな処理を用いている．シーケンシャルな実行と比較した場合の性能向上率，手動でパイプライン並列処理を記述した場合と比較したコード行数と相対性能を評価する．

Vol.2011-ARC-197 No.10
Vol.2011-HPC-132 No.10
2011/11/28

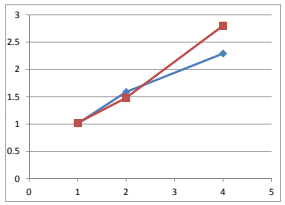

図 8 ステージ数とパイプライン並列化による性能向上
Fig. 8 Number of Pipeline Stage vs Speed Up

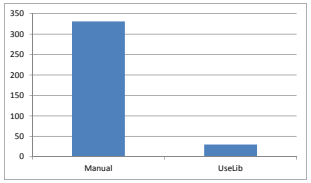

図 9 コード行数の比較
Fig. 9 Comparison of Line Numbers

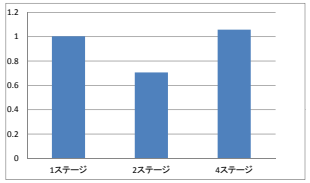

図 10 マニュアルコードとの性能比較
Fig. 10 Speed Up Compared to Manual Code

評価に用いた計算プラットフォームは表 7 に示したものである．

5.2 評価結果

図 8 にパイプラインステージ数を増加させたときの性能向上率の変化を示す．横軸は，パイプラインのステージ数，すなわち並列実行されるタスクの個数を表しており，縦軸は各ステージをシーケンシャルに実行した場合に対する相対的な性能である．赤色の線は一度に流れるデータのサイズが 4MByte の場合，青色の線はデータのサイズが 400KByte の場合である．パイプライン並列性を利用することで実行性能を向上できていることが分かる．

図 9 および図 10 に，手動でパイプライン並列処理を記述した場合と比較した場合のコード行数と相対性能を示す．ここで，コード行数はステージ数が 4 段の場合のコード行数を示している．提案ライブラリを用いることで，パイプライン並列処理に伴うプログラムの複雑化を防ぎつつ，マニュアルコードと同等もしくはそれ以上の性能を達成していることが分かる．

6. 関連研究

アクセラレータデバイスを用いたプログラミング環境の標準としての OpenCL の可能性を探る研究がこれまでにもなされている．OpenCL の性能評価[9] や，性能可搬性の向上技術[16]，特定のデバイス上におけるランタイムシステムの実装技術の研究がなされている[4)11)]．本稿では，これら先行研究が対象としてこなかったパイプライン並列性を利用するアプリケーションを OpenCL 上において実装する技術について検討しており，この点でこれらの研究とは異なる．パイプライン並列プログラミングモデルは，従来から組み込みシステムのシステム設計において用いられてきた，重要な並列プログラミングモデルである．こうしたパイプライン並列処理をもとにしたパイプライン並列プログラミングモデルを，組み込みシステムにとどまらず汎用コンピューティングにおいても利用しようとする研究が近年なされている．パイプライン並列性を利用するためのプログラミング言語[15]，アプリケーション開発におけるパイプライン並列プログラミングの利用事例[2]，モデルを用いたパイプライン並列プログラミングモデルの性能解析[12] が行われており，データ並列プログラミングモデルと並んでパイプライン並列プログラミングモデルは重要な並列プログラミングモデルと位置付けられている．

7. まとめと今後の課題

アクセラレータを含むヘテロジニアスなコンピューティング環境においてアプリケーション開発に伴うプログラマの負担を軽減することを目指し，OpenCL を用いたパイプライン並列プログラミング API を提案した．これまで，汎用的なアクセラレータデバイスを用いてデータ並列性を利用するアプリケーションを開発する技術に関して多くの研究がなされてきたが，パイプライン並列性を利用する研究はこれまでなされてこなかった．特に組み込みシステムにおいて，パイプライン並列性を利用することは重要である．本稿では，パイプライン並列性を利用したアプリケーションをクロスプラットフォームなアクセラレータプログラミング環境である OpenCL を用いて実装する方法を示すとともに，パイプライン並列処理に伴うアプリケーション開発作業の大部分をライブラリを用いて自動化できることを示した．評価の結果，手動でアプリケーション開発を行った場合に対して提案ライブラリを用いた場合では，大幅にプログラム記述量を削減しつつ，同程度の性能を達成することができることが分かった．今後は，アクセラレータデバイスを含むコンピューティング環境におけるパイプライン並列処理における最適なタスク分割・タスクマッピング・タスクスケジューリングアルゴリズムを検討するとともに，実アプリケーションを用いた性能評価を行う予定である．

謝辞 本研究は日本学術振興会特別研究員奨励費（２３・８０６２）の助成を受けて行われたものである．

参考文献

1) AlfredV. Aho, Ravi Sethi, and JeffreyD. Ullman. *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.
2) Christian Bienia and Kai Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

Vol.2011-ARC-197 No.10
Vol.2011-HPC-132 No.10
2011/11/28

3) Shekhar Borkar and AndrewA. Chien. The future of microprocessors. *Commun. ACM*, 54:67–77, May 2011.
4) Konstantis Daloukas, ChristosD. Antonopoulos, and Nikolaos Bellas. Glopencl: Opencl support on hardware- and software-managed cache multicores. In *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, HiPEAC '11, pages 15–24, New York, NY, USA, 2011. ACM.
5) E. A. de Kock, W. J. M. Smits, P. van der Wolf, J.-Y. Brunel, W. M. Kruijtzer, P.Lieverse, K.A. Vissers, and G.Essink. Yapi: application modeling for signal processing systems. In *Proceedings of the 37th Annual Design Automation Conference*, DAC '00, pages 402–405, New York, NY, USA, 2000. ACM.
6) Matthew Drake, Hank Hoffmann, Rodric Rabbah, and Saman Amarasinghe. Mpeg-2 decoding in a stream programming language. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 108–108, Washington, DC, USA, 2006. IEEE Computer Society.
7) Khronous OpenCLWorking Group. Opencl specification verion 1.1, 2010.
8) Stephen W. Keckler, William J. Dally, Brucek Khailany, Michael Garland, and David Glasco. Gpus and the future of parallel computing. *IEEE Micro*, 31:7–17, 2011.
9) Kazuhiko Komatsu, Katsuto Sato, Yusuke Arai, Kentaro Koyama, Hiroyuki Takizawa, and Hiroaki Kobayashi. Evaluating performance and portability of opencl programs. In *The Fifth International Workshop on Automatic Performance Tuning*, June 2010.
10) EdwardAshford Lee and DavidG. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, 36:24–35, January 1987.
11) Jaejin Lee, Jungwon Kim, Sangmin Seo, Seungkyun Kim, Jungho Park, Honggyu Kim, ThanhTuan Dao, Yongjin Cho, SungJong Seo, SeungHak Lee, SeungMo Cho, HyoJung Song, Sang-Bum Suh, and Jong-Deok Choi. An opencl framework for heterogeneous multicores with local memory. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT '10, pages 193–204, New York, NY, USA, 2010. ACM.
12) Angeles Navarro, Rafael Asenjo, Siham Tabik, and Calin Cascaval. Analytical modeling of pipeline parallelism. *Parallel Architectures and Compilation Techniques, International Conference on*, 0:281–290, 2009.
13) H.-J. Stolberg, S.Moch, L.Friebe, A.Dehnhardt, M.B. Kulaczewski, M.Berekovic, and P. Pirsch. An soc with two multimedia dsps and a risc core for video compression applications. In *Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International*, pages 330 – 531 Vol.1, feb. 2004.
14) William Thies and Saman Amarasinghe. An empirical characterization of stream programs and its implications for language and compiler design. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT '10, pages 365–376, New York, NY, USA, 2010. ACM.
15) William Thies, Michal Karczmarek, and Saman Amarasinghe. Streamit: A language for streaming applications. In *International Conference on Compiler Construction*, Grenoble, France, Apr 2002.
16) 京昭倫 and 岡崎信一郎. Opencl の性能可搬性改善に向けた基本 api の提案. **情報処理学会研究報告. 計算機アーキテクチャ研究会報告**, 2011(12):1–8, 2011-07-20.