Vol.2012-ARC-200 No.2
Vol.2012-OS-121 No.2
2012/5/7

組込み向けメニーコアアクセラレータ用 OpenCLの設計と組込みOSの実装

坂本 龍一[1]　望月 秋人[1]　小林 弘明[1]　高橋 昭宏[1]　佐藤 未来子[1]　天野 英晴[2]　中村 宏[3]
並木 美太郎[1]

概要：
計算機システムにおける性能向上の一つの方法として，アクセラレータを搭載し，高性能化を実現することは重要である．筆者らの研究においては，Cube-1 と呼ぶホスト CPU(Geysre-Cube) と複数のアクセラレータ (CMA-Cube) から構成されるシステムを実チップとして実装し，高性能かつ省電力な計算機の構築を行っている．しかし，CMA-Cube アクセラレータのプログラミングでは，アクセラレータ間との同期やデータ転送，入出力と全体の制御を行う Geyser-Cube ホスト CPU との連携など，煩雑な制御を効率よく行う必要がある．
本研究では，CMA-Cube アクセラレータの実行制御として OpenCL を用い，ユーザプログラムから簡便な実行環境を提供する．OpenCL を CMA 向けに実現し，OpenCL の API の枠組みの中で CMA をユーザプログラムに提供する．同期のタイミングなどを OpenCL に隠蔽し，低レベルの煩雑な制御をプログラマが書かなくて良いシステムとした．同時に，CMA 用 OpenCL を実現するために，組込み OS に CMA の資源管理機構を導入し，複数のアクセラレータで同期を取りながら計算を実行するためのスケジューラを実現した．この結果，CMA 資源管理機構の実装を行い，3% 程度の実行時間のオーバヘッドのみで CMA-Cube を効率良く実行できる枠組みを提供できた．

キーワード：省電力，組込み，OS，マルチコアプロセッサ，OpenCL

Design and Implementation of OpenCL Library and its Embedded OS for Embedded Many-Core accelerator

Ryuichi Sakamoto[1]　Akito Motiduki[1]　Hiroaki Kobayashi[1]　Akihiro Takahashi[1]　Mikiko Sato[1]
Hideharu Amano[2]　Hiroshi Nakamura[3]　Mitaro Namiki[1]

Abstract: It is important to achieve high performance using accelerators in a computer system. In this study, Cube-1 processor is designed and implemented to realize a high-performance and low-power computer system, which consist of a CPU (Geyser-Cube) and three accelerators (CMA-Cubes). The programming for a CMA-Cube accelerator needs to perform complicated controls efficiently, that are the synchronization between accelerators, the data transfer to an accelerator, and the cooperation with the host CPU Geyser-Cube which performs input /output and control the whole system. This paper describes execution environment in which OpenCL is used to control the CMA-Cube accelerator. OpenCL is implemented for CMA-Cube and the framework of OpenCL API provides CMA-Cube to a user program. Since timing control of synchronization is covered with OpenCL, the programmer does not need to write the execution control at a low level. Furthermore, in order to realize OpenCL for CMA-Cube, the resource management mechanism of CMA-Cube is added to the embedded OS and the task scheduler is realized in order to calculate taking the synchronization between accelerators. The framework for efficiently executing CMA-Cube has been provided only by the increase in about 3% of execution time.

Keywords: Low-Power, Embedded, OS, Multi-Core Processor, OpenCL

1. はじめに

近年スマートフォン等の普及により組込み機器の高性能化が進んでいる．それらはマルチコアのプロセッサを搭載したり 1GHz を超えるような動作周波数で動作するものもある．このような高性能な組込み機器では，消費電力の増大が問題となっている．本研究では CREST 省電力プロジェクト [1] において，省電力な汎用プロセッサ [2] と電力効率の高いアクセラレータの Cool Mega-Array(CMA-Cube)[3] を複数組み合わせて，三次元方向に積層した Cube-1[4] という高性能な省電力組込みプロセッサの研究開発に取り組んでいる．現在 Cube-1 のテープアウトを終え，チップのテストを行っている段階である．

本 Cube-1 で積層した複数の CMA-Cube を有効に並列動作させて利用するためには，省電力汎用プロセッサ上から CMA-Cube の実行を適切に制御することが必要である．Cube-1 のような，ヘテロジニアス構成の計算資源を有効活用するための並列計算用フレームワークとして，従来より OpenCL が提案されている．OpenCL では，GPGPU やマルチコアといった並列計算用の演算プロセッサを抽象化したモデルに対して，データ転送，演算プロセッサの実行などを制御するための共通 API が用意され，C/C++から利用できる．

OpenCL では，GPGPU 等の演算プロセッサ上で実行される「カーネルコード」と，汎用プロセッサ等の制御プロセッサ上で実行される「ホストコード」を用いる．ホストコードには，カーネルコードの実行制御や演算データの転送制御を記述することができ，イベントオブジェクトと呼ばれるオブジェクトを用いて，イベントの実行順序も制御できる．すなわち，演算を行うカーネルコードの実行順序や，データ転送制御などをホストコードでイベントオブジェクトを用いて適切に記述できるという特徴がある．

そこで，本論文では，OpenCL による CMA-Cube の抽象化を行い，これをサポートする OS による実行機構を提案する．Cube-1 プログラマは，データ制御，演算プロセッサの実行を指示し OpenCL ライブラリ内では，CMA-Cube を操作するタスクを自動生成してデータ転送と同期処理を行う．OS では，OpenCL ライブラリから渡された実行制約を，受け取り，実行順序リストを生成する．生成した実行順序リストをもとに，タスクのスケジューリング制御を行う．

以下，本論文では，CMA-Cube アクセラレータ向け OpenCL ライブラリ設計と，そのライブラリの実行環境となる組込み OS について述べる．

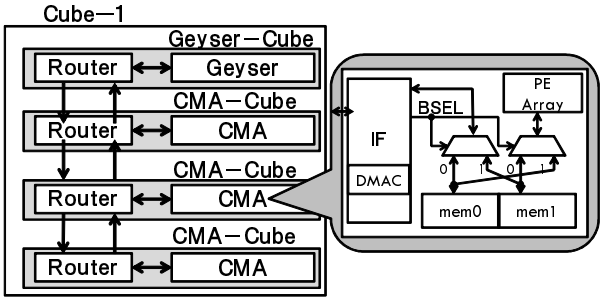

図 1 Cube-1 のアーキテクチャ概要

2. ハードウェアアーキテクチャの概要

2.1 Cube-1 のハードウェアアーキテクチャ

Cube-1 は汎用プロセッサの Geyser-Cube プロセッサと，アクセラレータの CMA-Cube 三つから構成される組込み向け省電力プロセッサである．図 1 に Cube-1 のハードウェア構成を示す．Cube-1 は Geyser-Cube チップ一枚と，三つの CMA-Cube チップを三次元方向へ積層した構成となっている．これらの四つのプロセッサはリングバス上にルータを介して接続されており，各 CMA-Cube に備える二面のローカルメモリによるダブルバッファリングにより，CMA-Cube 間で効率の良いデータ転送を行える．また CMA-Cube 間の同期制御を行うことで，Cube-1 におけるパイプライン処理を実現できるように設計されている．ハードウェアは主として慶應義塾大学で開発されている．

2.1.1 Geyser-Cube

Geyser-Cube は MIPS R3000 をベースとした細粒度パワーゲーティング技術を加えた省電力プロセッサ [2] である．Geyser-Cube はリングバス上ではマスターデバイスとして動作し，Geyser-Cube から三つの CMA-Cube に対する実行制御を行う．また，Geyser-Cube が CMA-Cube のルータを制御することで，CMA 間でのデータ転送も制御する．

2.1.2 CMA-Cube

CMA-Cube は演算器アレイ型の超省電力再構成可能デバイス [3] である．CMA-Cube は演算器アレイとマイクロコントローラとローカルメモリから構成される．マイクロコントローラは演算器アレイの設定を行い，ローカルメモリからのデータを読み出し，演算器アレイへ転送する．また，演算器アレイの計算結果はマイクロコントローラによって再度，ローカルメモリへ書き戻される．演算器アレイは組み合わせ回路から構成され 10 数 mW/CMA の電力消費で動作し，電力あたりの計算性能が非常に高いことが特徴である．

CMA-Cube のローカルメモリは，二つのバンクのメモリから構成されている．バンクの切換えは BSEL と呼ばれる信号線で制御する設計であり，一方のローカルメモリをルータ側へ接続し，もう一方をマイクロコントローラと演

[1] Tokyo University of Agriculture and Technology
[2] Keio University
[3] University of Tokyo

算器アレイ側へ接続する．

CMA-Cube のローカルメモリ，マイクロコントローラのプログラムメモリ，CMA 各種制御レジスタなどを，すべて Geyser-Cube から制御する．例えば，一方のローカルメモリで演算を行っている間に，もう一方のローカルメモリの演算データを入れ替えるように Geyser-Cube が制御を行えば，高効率に CMA-Cube を利用できる．

2.2 Cube-1 のプログラミングモデル

CMA-Cube はダブルバッファリングでデータ転送しながら演算を行うことで，CMA-Cube の演算を止めることなく，高効率な演算実行が可能である．本研究では，Cube-1 の複数の CMA-Cube を効率よく利用するためのプログラミングモデルとして，CMA-Cube ごとに処理内容を定め，複数の CMA-Cube 間で演算データを受け渡す，パイプラインモデルが有効である．

例えば，Cube-1 でマルチメディアに代表されるストリーミング処理をパイプラインモデルで実行する場合，ストリーミング処理の中でパイプライン的に実行できる部分を各 CMA-Cube の処理として割り当て，演算データを複数の CMA-Cube 間で受け渡しながら並列処理する．このようなパイプラインモデルで並列実行することで，Cube-1 の並列演算性能を生かすことができると考える．ただし，Cube-1 では各 CMA-Cube のダブルバッファリングによるデータ転送と並列処理に対する同期制御をすべて Geyser-Cube で実行するため，Geyser-Cube 上では CMA-Cube 制御用コードが必要となる．

Cube-1 のようなアクセラレータを効率よく利用するためには，転送するデータサイズ，データを転送するタイミング等の設計が重要である．例えば，ダブルバッファにおいては，演算が終了する前にデータを転送しなければ効率が低下する．また，並列制御においてはボトルネックとなる処理を分析し，並列化可能であれば複数の CMA-Cube に処理を分散することなども有効である．このような CMA-Cube のための制御コードを，OpenCL のような公開され，かつハードウェア独立の仕様で記述できれば，Cube-1 のハードウェア構成を知らないプログラマであっても CMA-Cube のパイプラインモデルを設計しやすくなる．同時にプログラマな同期指示，CMA の実行開始や停止などの I/O レジスタ群への読み書き，データ転送処理などの，CMA に対する低レベルな操作指示を OpenCL 内部で隠ぺいできると考えた．

3. 本研究の目標

前の章では，高効率なアクセラレータである CMA-Cube について，アクセラレータへのカーネルコード転送，ホスト CPU と複数のアクセラレータの実行制御，ホスト CPU とアクセラレータのデータ転送制御を効率良く実行する必要があると述べた．これらの制御をパイプライン的に実行し，高水準言語から効率良く，簡便に実行できる実行環境の実現が必要不可欠である．

そこで，本研究では，Cube-1 の制御プロセッサである Geyser-Cube から複数の CMA-Cube をパイプラインモデルで効率良く制御できるようにするために，Cube-1 用 OpenCL とそのための OS の実現を目指す．特に

(1) CMA-Cube を OpenCL で抽象化し，OpenCL の API の形式に適合するモデル化と実行機構を提案する．
(2) 同時に，OpenCL を実行するための OS サポートを行う．

ことが本研究の目的である．なお，OpenCL のカーネルコードとなる CMA-Cube 本体のプログラミングは本研究の対象外としている．あくまで，ホスト CPU のコードと OpenCL のカーネルコードの実行制御とデータ転送管理を提案する OpenCL と組込み OS で実現する．

OpenCL については，CMA-Cube を OpenCL のデバイスモデルへの抽象化する．この OpenCL ライブラリを用いて，Geyser-Cube 上で実行する C/C++のコードから，Cube-1 の CMA-Cube のパイプラインモデルを制御できるようにする．本研究の Cube-1 プログラマは，各 CMA-Cube で実行する演算処理を定義し，CMA-Cube 用 OpenCL の仕様に従って作成した CMA-Cube 制御用コードを準備する．アクセラレータに依存した詳細かつ低レベルの実行制御，データ転送制御をプログラマは記述する必要はなく，可能な限り OpenCL の枠組みの中で処理を記述できる．低レベルの実行制御とデータ転送は，本 OpenCL ライブラリと OS で隠蔽されている．

本 OpenCL 用の OS では，OpenCL ライブラリが用いる実行制御の関数群を基に各処理のタスクとして生成しそのタスクが連携しあうことで CMA-Cube 間ないしは CMA-Cube と Geyser-Cube の同期を管理すると同時に，ホスト CPU とアクセラレータ間のデータ転送を管理する．同期制御は，OpenCL ライブラリが OS へ渡した実行制約を OS 側でまとめ，実行順序リストを作成し，OS が実行順序リストに基づいて実行を制御する．このような方式とすることで，プログラマは低レベルの煩雑な同期を記述する必要はなくなると同時にデバイスに応じた適切な性能を得ることができる．

4. システムの構成と機能

4.1 システム構成の概要

Cube-1 向けのプログラム実行環境を図 2 に示す．本システムは，主として次の二つから構成される．

- Cube-1 向け OpenCL ライブラリ
- Cube-1 OS

OpenCL ライブラリは，OpenCL と CMA の実行モデルの間を埋める．CMA の実行モデルを OpenCL の枠組みの中

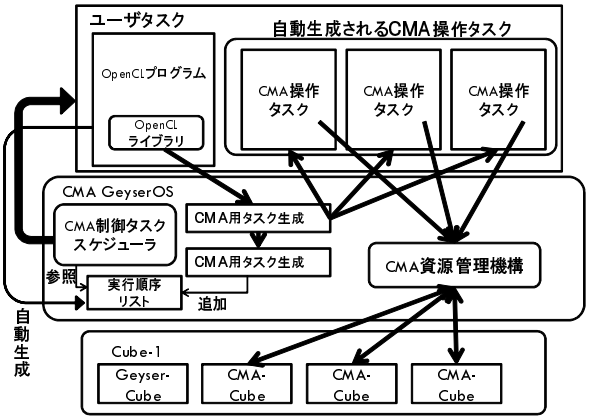

図 2 提案するシステム

に，性能を低下させることなく隠蔽する．このライブライは，ユーザプログラムから利用され，

- カーネルコードの転送や実行制御
- データ転送と制御

を行う．ただし，プログラマは CMA の低レベルな機構を用いるのではなく，CMA-Cube 向けに拡張された OpenCL の API に基づいてプログラミングを行う．OpenCL ライブラリでは，プログラマが明示的に記述したタスク以外に実行制御のために，図 2 の中の実行順序リストを自動的に生成することで，煩雑な実行制御のプログラミングをプログラマから解放する．

Cube-1 OS は，

- CMA 資源管理機構
- CMA 制御タスクスケジューラ

から構成される．

CMA 資源管理機構は，デバイスドライバ的な位置付けとなっており，CMA の I/O レジスタ群を OpenCL からデバイスアクセスするためのインタフェースを提供する．また，ユーザタスクからは保護されているアクセラレータの割込み，DMAC などを管理する．

CMA 制御タスクスケジューラについては, ホスト CPU と三つのアクセラレータの同期実行管理を実現する．OpenCL ライブラリは，データ制御，演算デバイスの実行を行う．また，OpenCL ライブラリが自動生成する CMA 制御タスクの生成依頼と同時に，実行順序の依存関係の情報が渡される，OS はこの情報に基づいて，同期管理とデータ転送管理を行う．

4.2 OpenCL ライブラリの機能概要

OpenCL ライブラリでは，OpenCL API を用いて記述した，データ制御，カーネルコードの実行という非同期な処理を，OS 上で非同期に実行可能なタスクへの変換を行う．マルチタスク環境では，タスクは非同期に実行される．そのためデータ転送，カーネルコード実行などの処理を，これら，複数のタスクに割り当てることで非同期に実行させる．

また，OpenCL で記述した，データ制御，カーネル実行はイベントオブジェクトで実行順序を制御できるようになっている．そこで，これら，イベントオブジェクトをタスクの実行制約として OS へ通知する．

4.3 Cube-1 OS の機能概要

OpenCL ライブラリによって，タスクへ分割されたデータ転送処理タスク，カーネルコード実行タスクらをマルチタスク環境として非同期に実行を行う．しかし，一般的に複数生成されたタスクへの実行順序を指定することはできない．そこで，本 Cube-1 OS では，これらの複数のタスクに対して OpenCL ライブラリから受け取ったタスクの実行順序の制約を守るようにマルチタスクを実行する．

4.4 CMA 実行モデルと OpenCL API と処理タスク

前の節で示した各構成要素で，それぞれの処理がどのように実行されるかを図 3 に示す．上段に Cube-1 でのパイプライニング処理のモデル，中段に OpenCL API との対応，下段にマルチタスクとの対応を示す．OpenCL ライブラリを用いて記述した中段のコードは，OpenCL ライブラリから，Cube-1 OS へタスク生成の要求が行われ，Cube-1 OS によって実際に CAM 操作に対応するタスクが下段に示すように生成される．また，上段，中段，下段の図中で同じ場所にある四角の枠は，同一の処理に対応している．

- Cube-1 の実行モデル
  CMA-Cube は二つのバンクのローカルメモリを切換えながらダブルバッファリングを行いながら演算を行うことで CMA-Cube の演算を止めることなく，効率よく演算を行うことができる．また，ストリーミング処理等をパイプライニングを用いて効率よく演算することを想定している．各 CMA-Cube ごとにストリーム処理の内部のパイプライン関係になっている処理を割り当て，演算データを複数の CMA-Cube 間で受け渡すことでストリーミング処理を行うことができる．ストリーミング処理を行うことで，Cube-1 の主記憶へのアクセスを減らすことができる．これらのダブルバッファリングと，処理のストリーミングを同時に行う．
  モデルでは A，B，C の 3 つの処理があり，A,B はともに主記憶にあるデータを使用して演算を行い，その，A,B の両方の計算結果を使用して C の演算を行う例を用いている．CMA0 に処理 A を，CMA1 に処理 B を，CMA2 に処理 C を割り当てる．A,B の処理結果を CMA 間 DMA を持ちいている．あわせて，バンク切換え機能を操作し，ダブルバッファリングを行う．図 3 の上段にモデルを示す．
- CMA 用 OpenCL API との対応
  次に，OpenCL API でのパイプライニングモデルの対

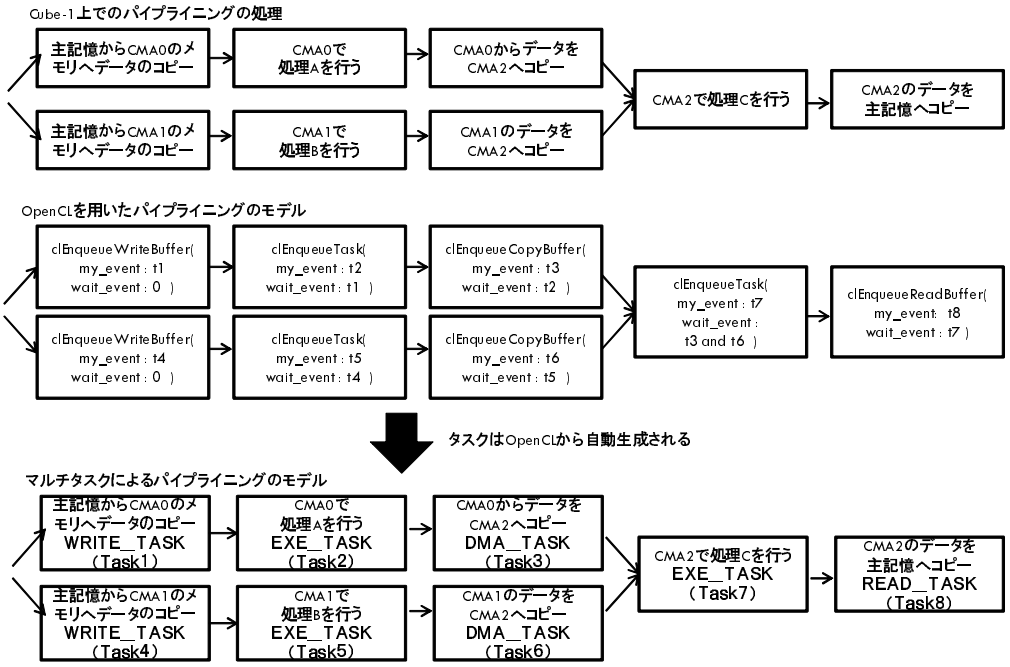

図 3 パイプラインのモデル

応を示す．主記憶から CMA へのデータの転送をデータの書き込む関数である clEnqueueWriteBuffer に対応させる．CMA での処理の実行の関数は clEnqueue-Task に対応させる．CMA 間でのデータの転送の関数は clEnqueueCopyBuffer に対応させる．CMA から主記憶へのデータの転送の関数は clEnqueueReadBuffer に対応させる．また，これら関数の引数に，関数実行順序の依存関係をイベントとして記述し，実行順序を制御する．CMA 用 OpenCL API との対応を図 3 の中段に示す．

- タスクとの対応
  マルチタスクを用いたパイプライニングのモデルを述べる．OpenCL ライブラリから CMA 操作タスクの生成が要求され，これらのタスクは Cube-1 OS から生成される．また，OpenCL の関数ごとに受け付けた，実行制約リストをもとに，CMA 制御タスクスケジューラは，処理の単位ごとにタスクを実行し，パイプライン処理を行う．これを，図 3 の下段に示す．Geyser-Cube の主記憶からの CMA-Cube のローカルメモリの転送を WRITE_TASK が行う．CMA-Cube でのカーネルの実行を EXE_TASK が行う．CMA 間でのデータ転送を DMA_TASK が行う．CMA のローカルメモリから，Geyser-Cube の主記憶へのデータの読込みを READ_TASK が行う．これらのタスクは OpenCL ライブラリ内部で，自動的に生成される．

5. Cube-1 用 OpenCL の設計

5.1 Cube-1 の OpenCL カーネルコード実行

従来の OpenCL では演算デバイスには複数の演算ユニットを搭載していて，どの演算ユニットからも読み出せるグローバルメモリがあることを想定している．そのため，従来の OpenCL では，メモリの配置を考慮してプログラミングは行わない．初めに，グローバルメモリに対してメモリの確保を行う．確保される演算デバイスのメモリは，OpenCL 側が自動で領域を確保し，プログラマに対して，どこのメモリを使用しているかを隠蔽していた．

OpenCL では初めに，演算デバイスのメモリの確保を行う．そこで，CMA 用 OpenCL では，CMA 用のメモリ確保関数を用意し，Cube-1 の三つあるうちのどの CMA を使うかを直接指示できるようにする．同時に，指定した CMA を予約する．

5.2 OpenCL によるデバイスの抽象化

OpenCL で定義されているプラットフォームモデルと Cube-1 の対応について示す．本稿では今まで Cube-1 を利用したことのあるプログラマを対象として OpenCL の設計を行う．また，Cube-1 プログラマがカーネルの実行を行う CMA-Cube を指定してプログラミングが可能なものとする．そのために，プラットフォームモデルの対応も，三つの CMA を三つのアクセラレータがあるものと抽象化し，Cube-1 が三つの CMA を制御するようプログラミングする．図 4 に対応を示す．

OpenCL において，タスクの制御や，メモリの管理を行うホストに対応する部分を Cube-1 の Geyser-Cube に対応させた．また, 演算用のソフトウェアが動作する演算デバイスを Cube-1 の三つの CMA-Cube に対応させた．さらに，各 CMA-Cube をそれぞれ OpenCL の演算ユニットに対応させる．三つの CMA-Cube を三つの演算ユニットに

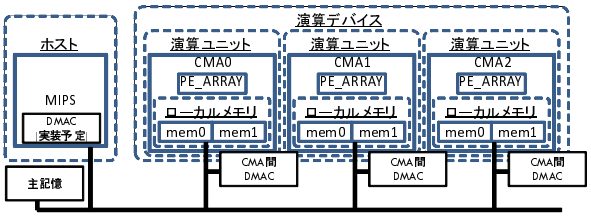

図 4 OpenCL によるデバイスの抽象化

対応させることで，プログラマが CMA-Cube の並列性を活用できるようにした．

5.3 OpenCL による Cube-1 制御

本研究で中心的な役割を果たす OpenCL の API を表 1 に示す．この中で，cl_CMA_CreateBuffer と cl_CMA_EnqueueChangeBSEL が Cube-1 用に追加拡張された API である．また，個々の API も内部で CMA の低レベルな処理を隠ぺいする．clEnqueueTask は，CMA 操作用タスクを自動生成すると同時に，実行順序などの情報を生成して OS カーネルに渡す．また，clEnqueueCopyBuffer は DMAC の処理と管理を内部で行う．OpenCL 本来の API に加え，CMA 用に拡張した OpenCL API を用いることで CMA へのデータの受け渡し，CMA 間 DMA，BSEL の切換え等を I/O レジスタやユニットなどのハードウェアの制御仕様を知らなくとも簡潔に CMA 制御を記述できる．また，イベントの指示を Cube-1 OS が管理，実行することで，プログラマが詳細な低レベルの処理を記述することなくパイプライン処理を実現する．簡単なプログラム例を図 6 に示す．

表 1 CMA 用 OpenCL API

CMA 用 OpenCL API	機能
cl_CMA_CreateBuffer	CMA のメモリを確保する（新規）
clSetKernelArg	CMA にカーネルをバインドする
cl_CMA_Enqueue ChangeBSEL	バンク切換え機能の操作（新規）
clEnqueueReadBuffer	CMA のメモリを読み出す
clEnqueueWriteBuffer	CMA へデータを書き込む
clEnqueueCopyBuffer	CMA 間 DMA を使う
clEnqueueTask	CMA の実行を行う

(1) メモリの確保 (cl_CMA_CreateBuffer)
三次元積層されたチップの温度によるリーク電力を考慮した省電力化，リングバスの利用効率向上のために，CMA 用 OpenCL では使用するメモリの場所を，すなわち，CMA の番号を指示してメモリの確保を行う．OpenCL では初めに，演算デバイスのメモリの確保を行う．そこで，CMA 用 OpenCL では，CMA 用のメモリ確保関数を用意し，Cube-1 の三つあるうちのどの CMA を使うかを直接指示できるようにする．同時に，指定した CMA を予約する．

```
typedef struct{
  cl_mem mem0;
  cl_mem mem1;
} cl_CMA_mem;\
```

図 5 cl_CMA_mem 構造体

また，CMA-Cube には二つのバンクのローカルメモリがあるため，これらに対応する構造体の新設を行った．また，OpenCL で二つのメモリを管理する cl_CMA_mem 構造体とする．図 5 に示す．cl_CMA_mem 構造体の mem0，mem1 をそれぞれ CMA-Cube のローカルメモリに対応させる．

(2) カーネルの割り当て (clSetKernelArg)
CMA-Cube 上で動作するカーネルコードの引数として場所を指定して確保した CMA-Cube のローカルメモリを指定する．このようにすることで，指定した CMA でカーネルを実行できる．

(3) 演算用データを CMA へ転送 (clEnqueueWriteBuffer)
Geyser-Cube の主記憶のデータを CMA へ転送する，タスクの生成を Cube-1 OS に依頼する．

(4) バンク切換え機能の設定 (cl_CMA_EnqueueChangeBSEL)
CMA-Cube ははダブルバッファリングを行い高い効率で実行するために，二つのローカルメモリを BSEL 信号を切換えるながら使用する必要がある．そのため，BSEL を操作するタスクの生成依頼を行う．バンクの切換えもデータ転送に依存するために，本 API も OS へタスクの生成を依頼し，実行制約を与える．また，引数にバンク切換えとして，“ mem0 ”又は“ mem1 ”を指定する．指定した値のメモリが CPU 側から読み書きできるようになり，指定していないもう一方が PE 側と接続される．

(5) 実行 (clEnqueueTask)
CMA カーネルの実行を行うタスクの生成を OS カーネルに依頼する．これらの，タスクには，BSEL_TASK，READ_TASK，WRITE_TASK，DMA_TASK，EXE_TASK がある．詳細は 5.4 節で述べる．また，このタスクに対して，OpenCL の実行制御情報を持つイベントの情報を基に，生成するタスクの実行順序の情報を持つ実行制御の情報を作成し，cl_task_create() のシステムコールにより OS カーネルにスケジューリング情報を渡す．

(6) CMA 間 DMA (clEnqueueCopyBuffer)
転送元と転送先を指定し CMA 間 DMA を用いデータの転送を行うタスクの生成を行う．

(7) CMA のデータをホストへ転送 (clEnqueueReadBuffer)
CMA のデータをホストへ転送するタスクの生成依頼を行う．

```
//q は command_queue    e は event
memA=cl_CMA_CreateBuffer(context,
    CL_MEM_READ_WRITE,mem_size,NULL,&err);
memB=cl_CMA_CreateBuffer(context,
    CL_MEM_READ_WRITE,mem_size,NULL,&err);
clSetKernelArg(kernelA,0,sizeof(int),memA);
clSetKernelArg(kernelB,0,sizeof(int),memB);
clEnqueueWriteBuffer(q,memA.mem0,CL_TRUE,
    offset,data_size,in_data,0,NULL,e1);
cl_CMA_EnqueueChangeBSEL(q,memA,mem1,1,e1,e2);
clEnqueueTask(q,kernelA,1,e2,e3);
cl_CMA_EnqueueChangeBSEL(q,memA,mem0,1,e3,e4);
clEnqueueCopyBuffer(q,memA.mem0,memB.mem0,
    offset,offset,data_size,1,e4,e5);
cl_CMA_EnqueueChangeBSEL(q,memB,mem1,1,e5,e6);
clEnqueueTask(q,kernelA,1,e6,e7);
cl_CMA_EnqueueChangeBSEL(q,memB,mem0,1,e7,e8);
clEnqueueReadBuffer(q,memB.mem0,CL_TRUE,
    offset,data_size,out_data,1,e8,NULL);
```

図 6 OpenCL 疑似コード

5.4 CMA 操作タスク

CMA 用 OpenCL では，データ転送，演算デバイスのル実行の際に CMA 操作タスクの生成の要求する．CMA 用 OpecCL API と生成する CMA 操作タスクとの対応を表 2 に示す．

表 2 OpenCL API と CMA 操作タスク

CMA 用 OpenCL API	CMA 操作タスク名
cl_CMA_Enqueue ChangeBSEL	BSEL_TASK
clEnqueueReadBuffer	READ_TASK
clEnqueueWriteBuffer	WRITE_TASK
clEnqueueCopyBuffer	DMA_TASK
clEnqueueTask	EXE_TASK

また，CMA 操作タスクは実行が終了した場合は，システムコールを発行し，CMA 操作が終了したことを通知し，Cube-1 の設計の章で述べる，実行制約リストが保持する，実行状態を終了へ更新する．

6. Cube-1 OS の設計

Cube-1 OS は CMA タスク操作の生成を行い，マルチタスクによってパイプライン処理を行ったり，CMA の制御レジスタを隠蔽を行う．Cube-1 OS の設計を述べる．

6.1 Cube-1 OS の目的と機能

Cube-1 OS では，OpenCL ライブラリからタスク生成の要求を受け，CMA 操作に対するタスクの生成を行い，条件にあった CMA 操作タスクの実行を行う．これにより，パイプライニング処理を実現することを目的としている．また，CMA を操作する際は，Geyser-Cube のアドレスマップ上にマップされた多数の制御レジスタを操作する必要がある．そこで，多数の制御レジスタ，入出力の抽象化を行う．本 OS の実現は，本研究室で作成した独自の組込み OS「開聞」を修正した．Cube-1 OS は，

(1) OpenCL ライブラリが生成したタスク
(2) そのライブラリから生成される実行制約リスト

に基づいて，スケジューリングを行うことで，パイプライン的実行を OpenCL を用いたユーザプログラムから隠蔽する．

Cube-1 には CMA 操作タスクを識別するタスク ID と，各タスクごとの，実行状態，実行まで待つタスク ID のを保持した，実行順序リストがある．実行の状態は，実行待ち，実行中，終了の状態を表す．OpenCL ライブラリからの実行順序制約を CMA 操作タスク生成のさいに，実行順序リストに追加する．CMA 制御タスクスケジューラは，実行順序リストから，実行待ちの CMA 操作タスクを探す．その中から，実行順序を守り実行できるタスクを探し，実行する．これにより，効率的に非同期実行とデータ転送を実行できる．

このスケジューリングは，組込み OS では簡便な修正でも実現できること，軽量なスケジューラが望ましいことから，「開聞」のスケジューラを修正し，スケジューラ内に埋込むこととした．今後は，Linux などの汎用 OS にも本方式を適用するため，別途スケジューラに対するシステムコールを用意し，ユーザプロセスからも実行制約に基づくスケジューリングを行う予定である．

6.2 CMA 資源管理機構

CMA 資源管理機構は，CMA-Cube 用のデバイスドライバであると同時に，DMAC や割込み制御を行う．CMA-Cube のハードウェアを抽象化する．CMA の制御やローカルメモリへのデータの入出力をシステムコールとして仮想化を行う．また，CMA からの実行終了，DMA の終了の割り込みへのハンドリング，CMA デバイスとの同期・非同期処理をサポートする．表 3 に CMA 資源管理機構が提供するシステムコールを示す．

表 3 の中で重要なシステムコールは次のとおりである．

(1) 同期/非同期 CMA 実行管理 (cma_exe()，cma_set_exe_sleep())
三つの CMA のうち，指定した CMA に対して実行命令を発行する．同期実行管理では，実行終了までポーリングして待つ．CMA での実行時間が短く，割込みで待つ場合に性能的に不利な場合に用いる．非同期実行管理は，実行命令の発行後タスクをスリープし，CMA-Cube の終了割り込みによりタスクを再開する．

(2) CMA のバッファ管理 (cma_set_bsel())
ダブルバッファを管理し，指定した CMA-Cube のバンクを切換える．

表 3 CMA 資源管理機構のシステムコール

ライブラリ関数名	機能
cma_init	初期化を行う
cma_reset	CMA をストップし初期化する
cma_stop	CMA をストップする
cma_set_pe	PE 構成情報の設定
cma_set_micro	マイクロコントローラの設定
cma_set_bsel	BSEL の切換え
cma_write	ローカルメモリへのデータ転送
cma_read	ローカルメモリからのデータ転送
cma_exe	実行しポーリングにて終了を待つ
cma_dma	DMA 転送を実行し ポーリングにて待つ
cma_set_exe	実行のレジスタのみ操作し CMA の終了を待たず戻る
cma_set_dma	転送のレジスタのみ操作し DMA の終了を待たず戻る
cma_check_exe_done	CMA の実行の終了を確認する
cma_check_dma_done	DMA の転送終了の確認をする
cma_set_exe_sleep	実行のレジスタを操作し終了する までタスクをスリープする.
cma_set_dma_sleep	転送のレジスタを操作し終了する までタスクをスリープする.

(3) データ転送管理
(cma_read(), cma_write()) CMA-Cube のダブルバッファの中で, Geyser-Cube 側に接続された CMA-Cube のローカルメモリのデータを指定されたアドレスから指定されたバイト数読み書きする.

(4) 同期/非同期 DMA 管理
(cma_dma(), cma_set_dma_sleep())
CMA 間 DMA を実行する. 同期 DMA 管理では実行終了までポーリングにてまつ. 非同期 DMA 管理では命令を発行したらタスクをスリープし, CMA 間 DMAC から DMA 転送終了割込みによりタスクの実行を管理する. DMA 転送時間でオーバヘッドの少ない方を利用する.

6.3 CMA 操作タスクの生成とタスクの制御

本システムでは, OpenCL から渡された実行制御情報に基づいて, 各種操作タスクの実行制御を行う. 5 章で述べたライブラリにより Cube-1 のハードウェアを抽象化するが, 実行制御については OS で行うこととなるが, そのための情報は OpenCL ライブラリから渡される. 基本的には, BSEL を切換える BSEL_TASK, Geyser-Cube の主記憶からデータを CMA-Cube のローカルメモリへ転送する WRITE_TASK, CMA-Cube のローカルメモリのデータを Geyser-Cube の主記憶へ転送する, READ_TASK, CMA 間でのデータの転送を行う DMA_TASK, カーネルの実行を行う, EXE_TASK があるが, 今回の実装では, これらの順序制御をスケジューラで行っている. 同期制御については, ユーザ層で行うことも考えたが, 組込み OS でオーバヘッドを減らすために今回はスケジューラが実行順序リストに基づいて実行制御を行う方式を採用した. 予定である.

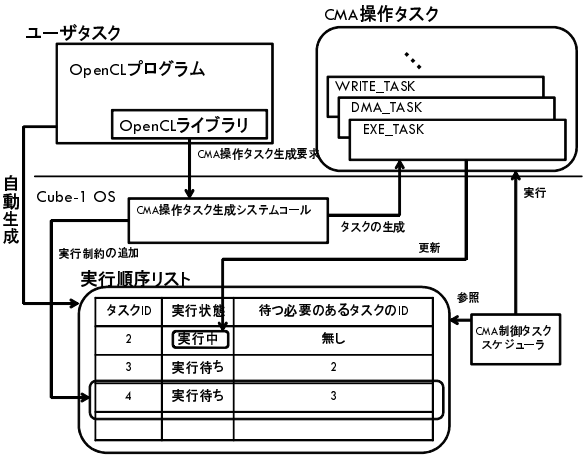

図 7 CMA 操作タスクの生成とタスクの制御

CMA 操作タスクの生成から, OS における CMA 操作タスクの実行を図 7 に示す.

(1) OpenCL ライブラリからの CMA 操作タスク生成要求発行
OpenCL プログラムから, データ転送, CMA-Cube の実行が指定されると OpenCL ライブラリは, CMA 操作タスクの生成を Cube-1 OS に要求する. CMA 操作タスクには BSEL_TASK, READ_TASK, WRITE_TASK, DMA_TASK, EXE_TASK がある. また, 図 8 の実行順序リストの情報がカーネルに渡される.
タスクの実行中, 終了などの状態を保持し, CMA 制御タスクスケジューラから, 実行の状態を確認できるように, タスクの実行状態を示す, 変数を持つ. CMA 制御タスクススケジューラは, タスクの実行状況も把握する必要があるために, OpenCL の実行順序を表す, イベントの値と, タスクの実行状態を示す 2 つの値を Cube-1 OS に通知する.

(2) タスク生成システムコール
Cube-1 OS では CMA 操作タスク生成要求を受け取る. 次に, 実行順序の制約を, 実行順序リストへ登録する. また, 該当する CMA 操作タスクの生成を行う. その後, CMA 制御タスクスケジューラを呼び出す.

(3) 実行する CMA 操作タスクの決定
CMA 制御タスクスケジューラは実行可能な CMA 操作タスクを検索し, 実行を行う. 実行順序リストを参照し, 実行状態が実行待ちのタスクを探し, 待つ必要があるタスクの ID の値を取り出す. 待つ必要があるタスク ID の状態を確認し, 実行終了の状態なら, 初めに選んだ, 実行状態が実行待ちのタスクを実行する.

(4) CMA 操作タスクの実行
CMA 操作タスクでは CMA 資源管理機構を操作し, CMA-Cube の制御を行う. また, すべての処理が終

了したら，終了したことを Cube-1 OS へ通知し，実行順序リストの実行状態を更新する．

7. 評価

提案を行った Cube-1 OS の実装を行い評価を行った．Cube-1 OS のオーバヘッドに関する評価を行う．Cube-1 OS の評価として，CMA のローカルメモリへの入出力の転送速度の評価と，計算データと，簡単な CMA のプログラムを転送し，計算し，結果を転送する簡単な実行時間の比較を行う．Cube-1 OS を介して CMA 操作をした場合と，直接 CMA を操作した場合の実行時間の比較を行う．

初めに，CMA 入出力の評価を行った．Cube-1 OS のシステムコールを利用して CMA のローカルメモリにデータを入出力したときの転送速度を測定した．CMA のローカルメモリに対する 2KB のデータの read/write を 100 回繰り返し，計 200KB のデータを読み書きしたときのそれぞれの転送速度を示す．また，本評価は RTL シミュレータ上にて処理にかかるクロック数を求め，Cube-1 の想定周波数の 200MHz にて動作した場合の数値としている．表 4 に CMA へ対する入出力を示す．

表 4 CMA0 への入出力の転送速度

	READ [MB/sec]	WRITE [MB/sec]
本システム (OS 有り)	6.62	11.10
OS 無し	6.68	11.26

本実装では，アクセス速度は読込みが約 6[MB/sec]，書込みが約 11[MB/sec] という結果を得た．また，提案した Cube-1 OS はデータ転送性能において 3% 以下のオーバヘッドにおいて実装可能であることが確認できた．

次に CMA で簡単なサンプルプログラムを動かした際の実行時間を計測した．サンプルプログラムの内容は 2KB のデータを転送し，その後構成情報を転送，実行し，2KB のデータを読み出す処理である．プログラムは三万回のループを回すものである．その実行速度を表 5 に示す．

表 5 サンプルプログラムの実行時間

処理内容	本システムの所要時間 [μsec]	OS なしの所要時間 [μsec]
CMA へのカーネルデータ転送	51.5	43.5
ローカルメモリへデータ書込み	214.9	214.9
実行	180.5	179.5
ローカルメモリのデータ読込み	311.2	311.7
合計	758.1	749.8

提案手法の Cube-1 OS はコンフィグレーションにおいて約 10[μsec] 程度遅い結果となった．しかし，コンフィグレーションははじめの 1 回のみ行いその後は，データ転送，実行が主な時間を閉める．そのため，大きなオーバーヘッドにはならないといえる．

8. 関連研究

OpenCL を用いたマルチコアプロセッサの利用や，アクセラレータの利用に関する研究はこれまでに多く行われている．NVIDIA や AMD による GPGPU への OpenCL 実装 [6][7] や，INTEL による OpenCL 実装がある．しかし，本研究では組込み機器を対象として OS 実装からライブラリ実装までを行っていることが異なる．また，デバイスモデルの抽象化においても，薦田氏らの研究 [8] では，演算器を隠蔽し，パイプライニングを自動抽出し，プログラマへパイプライニングの記述を意識させない抽象化もあるが，本稿ではより，デバイスを意識し，プログラマへパイプライニングの記述を行わせる設計とした．本稿では CMA の制御に OpenCL を用いているが，組込み向けのリコンフィギャラブルデバイスを対象に独自の API を用いて制御を行う研究 [9] もある．

9. まとめ

まとめを述べる．本稿では省電力なマルチコアプロセッサである Cube-1 を対象に OpenCL から制御するためのシステムソフトウェア環境の設計，実装を行った．OpenCL の API と，Cube-1 の操作の対応を設計し，OpenCL から Cube-1 が操作できることの確認を行った．実際に，CMA 資源管理機構，Cube-1 OS の実装を行い，今まで Cube-1 のプログラミングを行うにはアセンブリでのプログラムが必要であったが，C/C++から Cube-1 を小さなオーバーヘッドで利用可能とすることが確認できた．

今後の課題として，OpenCL ライブラリ部の実装がある．また，今回の環境では RTL シミュレーションを用いて評価を行っているが，実チップでの動作へ向けて環境の構築を行い，実チップ上で評価を行うことを課題とする．

謝辞 本研究は, 科学技術振興機構「JST」の戦略的創造研究推進事業「CREST」における研究領域「情報システムの超低消費電力化を目指した技術革新と統合化技術」の研究課題「革新的電源制御による次世代超低電力高性能システム LSI の研究」によるものである．

参考文献

[1] 中村宏，天野英晴，宇佐美公良，並木美太郎，今井雅，近藤正章：革新的電源制御による超低消費電力高性能システム LSI の構想，情報処理学会研究報告 ARC-173, pp.79-85,2007.
[2] 白井利明，香嶋俊裕，武田清大，中田光貴，宇佐美公良，長谷川揚平，関直臣，天野英晴: ランタイムパワーゲーティングを適用した MIPS R3000 プロセッサの実装設計と評価，電子情報通信学会技術研究報告 VLD, VLSI 設計技術，Vol.107, No.414(20080109) pp. 43-48.
[3] 小崎信明, 安田好宏, 斉藤貴樹, 池淵大輔, 木村優之, 天野

英晴, 中村宏, 宇佐美公良, 並木美太郎, 近藤正章: CMA: 超低電力再構成アクセラレータ. 組込みシステムシンポジウム 2011 論文集, 第 2011 巻, pp. 8-1-8-9, OCT 2011.
[4] 佐々木瑛一, 佐々木大輔, 松谷宏紀, 天野英晴, 竹康宏, 黒田忠広, 坂本龍一, 並木美太郎: チップ間ワイヤレス接続を利用した三次元積層アーキテクチャの研究. In IEICE-CPSY2011-10, 第 IEICE-111 巻, pp. 7-12, JULY 2011.
[5] OpenCL. http://www.khronos.org/opencl/
[6] NVIDIA OpenCL http://developer.nvidia.com/opencl
[7] AMD OpenCL http://developer.amd.com/zones/OpenCLZone/Pages/default.aspx
[8] 薦田登志矢，三輪忍，中村宏：OpenCL を用いたパイプライン並列プログラミング API の初期検討情報処理学会研究報告，2011-HPC-132(10), 1-7, 2011-11-21
[9] Clemente, Juan Antonio and González, Carlos and Resano, Javier and Mozos, Daniel:A task graph execution manager for reconfigurable multi-tasking systems. Microprocess. Microsyst., Vol. 34, No. 2-4, pp. 73-83, March 2010.
[10] V. Tunbunheng and H. Amano: Black-Diamond: a Retargetable Compiler Using Graph with Configuration Bits for Dynamically Reconfigurable Architectures", Proc. of The 14th SASIMI, pp. 412-419 (2007).