日本ソフトウェア科学会第 28 回大会 (2011 年度) 講演論文集

Cerium における DataSegment API の設計

金城 裕　　河野 真治

本研究室では、Cell [2] 用の並列 TaskManager Cerium [6] [5] を作成し、Rendering Engine を含むゲームや並列計算の例題の作成と評価を行ってきた。TaskManager と Rendering Engine はシューティングゲームやレーシングゲームを記述するのに十分な性能を持っており、台数効果も満足いくものとなっている。しかし、この開発により Ceirum の問題点も明らかになってきている。本論文では、今までで Cerium の構成と問題点を記述し、新しい TaskManager の設計方針を述べる。

1　Cell 用 Task Manager Cerium

Cerium は PS3 (Cell [2]) 用のゲームフレームワークであり、ソフトウェアレンダリングを含む並列処理を Task 単位で記述する。今は C++ で記述されており、基本的な例題や、シューティングなどの例題で妥当な性能がでている。

しかし、Task の種類などが増え、記述が繁雑であるなどの欠点も明らかになっている。この論文では Many Core 向けの改良を提案する。

2　Cerium での並列プログラミングの問題点

Cerium では、ゲームプログラミング及び、sort や word count などの例題を書いたが、いくつかの問題点が明らかになっている。

`Task` の取り扱うデータ型が示されない
`Task` 自体は簡単だが `Task` を構成する方法が繁雑
`Open CL\cite{opencl}` に比べても構文的に複雑
`Task` の種類が複雑
`Task` の依存関係の記述がデータの依存関係と無関係
`Task Scheduler` が大きくメモリを圧迫

Design of DataSegment API in Cerium

Yutaka Kinjyo, Shinij KONO, 琉球大学大学院理工学研究科情報工学専攻並列信頼研, Dept.Concurrency Reliance Laboratory, Information Engineering Course, Faculty of Engineering Graduate School of Engineering and Science, University of the Ryukyus.

などである。実装方法的にもいくつか問題がある。

`C++` と `Task` 記述の相性が良くない
`Task Manager` が複雑になりすぎ

Task Scheduler は Queue から Task を取り出して一つ一つパイプライン実行を行うインタプリタ的な構造を持っている。これが、Task Manager 自体を複雑にする原因になっている。

3　Continuation based C との相性

当研究室で開発している Continuation based C [7] は、並列処理の基本単位である Task に対応した code segment を持っている。これを Cerium に対応させようとすると以下のような問題がある。

`Inteface` の型が整合しないと `Task` 同士を接続できない
`Scheduler` への接続が特定の `Interface` を要求する

どちらも、Code segment の interface (入力と出力) は、決まった形であるべきだと言うことを示している。しかし、Task 自体は様々なデータを取り扱う必要がある。ここに矛盾がある。この矛盾を解決するためには、データ側も基本単位を導入するべきだいうことになる。

Data Segment は、Code segment の双対概念であり、C の構造体に相当する。CbC [4] の Code segment は、

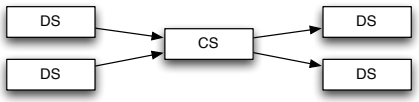

図 1 DS and CS

```
input interface (関数の引数の型)
output interface (goto 文の引数の型)
```

を持っているが、これらを

```
input datasegments
output datasegments
```

に置き換える。つまり、Code segment は、複数の動的に割り当てられた Data segment を持っている (図 3)。これらは、標準的な構造を持っているので、Interface の型の不整合を避けることができる。

Data segment は型を持っていて、その型は実行時に一致している必要がある。分散通信を考えて、Data segment の型は MessagePack [3] を用いる。

4 C++ との相性

Cerium の Task は、Cell の spu と ppu で共通であり、同じ Task ID で管理されている。これは C++ のオブジェクトとは関係ない。Cerium の開発でわかったのは、Cerium のデータは、Actor の become 的 [1] に書き換えられるということである。C++のようなポインタを使い合わし、オブジェクトの内部の書き換えで状態を作るようなオブジェクト指向プログラミングと、細分化した Task を並列に廻す Ceirum のようなシステムとの相性は良くない。Task の入力と出力は異なる場所に書かれる。処理は、常にダブルバッファを用いて行われているのでそのようになる。

5 階層的パイプライン

プログラム中の自明な並列性は、データ並列とループのパイプラインの二つであり、パイプラインはプログラムの中で、様々なレベルで行われる。Task そのものは入力データから出力データを計算するだけなので単純だが、その入出力データをダブルバッファリングとして切替えたり、適切な並列度を得られるように徐々に生成するのは非常に繁雑になる。

これらのデータの管理は、中心となるアルゴリズムとは別に並列実行を行うアーキテクチャに特化した処理が必要となる。例えば、分散環境で並列処理するのか、MPI なのか、Cell や Open CL なのかによって異なる。これらを、すべて Task という一括りで扱うと並列計算しない複雑な Task ができてしまう。

これらのデータ管理用の Task は、本質的には Data Segment に対する Iterator であり、ライブラリまたはコンパイラにより生成されるべきものだと考えられる。

6 Task 内部での Task 生成

Cerium では、複数の input と output を決めたパイプライン実行が通常であるが、Task の途中で Main Memory を参照したいことが良くある。

```
描画 Texture のデータ
SceneGraph の次のノード
```

これらは実行時にしか次のデータのアドレスを決定することができない。これを読み出し前の Task と読み出し後の Task に分割して、さらにパイプライン実行してやると良いが、この記述は今までの Cerium では不可能で、明示的な DMA API を使う必要があった。

Task 内部で Task 生成をしてやると、これを記述することが可能だが、TaskManager の複雑度が上がってしまうとう問題点があった。

7 Data Segment を用いた Cerium の再設計

Cell 用の TaskManager Cerium の再設計の方針としては以下のようになる。

```
CbC の Code segment の導入
定型的な Data 単位である Data segment
Data segment の型の指定
Task Manager の Code segment による実装
Code segment (Task) の生成 API
```

7.1 Data Segment の型

Cerium では Task の入出力は単なる memory buffer だったので型が存在しなかった。今回は Message Pack を用いて、Json 的に型を指定する。

Data segment は様々なメモリ上に位置するので、それを識別する必要がある。Many Core 版の Cerium では、

```
Main Memory
SPU local memory
Cache
```

の三種類を用意する。これらは DMA や Cache 操作命令を通して移動する。移動したものは同一のものである。Cell SPU の Task や Many Core では、当然だが lcoal memory や Cache に乗らない限りアクセスできない。

Data segment は、Code Segment に input と output として接続される。

7.2 Data Segment の API

Data Segment は以下の API を持っている

```
create
read
update
delete
```

Create は allocate に相当する。型と位置を指定して create する。Main Memory 上の Data segment を読み書きする場合は、local memory または Cache を通してパイプライン的に実行される。

複数の Code Segment から update が起きる場合は、以下の操作を選択する。

```
Queuing
Update
Proority Queue
```

生成された Data segment は synchronized queue として使うことができる。

7.3 Task Dependendcy

今までの Cerium では、`task->wait_for(task1);` という形で明示的に Task の依存性を指定していた。この方法では、`task` の寿命 (既に終了してしまった task を待つような場合) などの問題がある。

しかし、Code Segment は input / outpu Data Segment によって自然な依存関係を持つので、明示的な `wait_for` は必要なくなる。

Code Segment と Data Segment は task を処理して行くうちに自然に消滅してしまう。Persistent なデータは明示的にデータベースに格納する必要がある。つまり、Data Segment に Persistent という位置が存在する。

7.4 Pipeline Execution

Cerium では、Task の read/exec/write は三段のパイプラインで実行されていた。Data Segment は Code Segment の実行の前に行われるが、他の Code Segment とオーバラップして実行して良い。Data Segment には、Data Segment の位置を変更するための Code Segment が存在している。

つまり、Data Segment は複数の Code Segment (この Data segment を待っている Code segment) と、Data Segment の位置などを変える Code Segment などが付随している。

一つの Core では、Data Segment に付属する Code segment を順次実行することにより、パイプラインを実行する。

Data Segment による依存関係を追い越さなければ並列実行は自由に行われる。これは、Task Scheduling を担当する Code Segment によってアーキテクチャに合わせて実行される。(図 7.4)

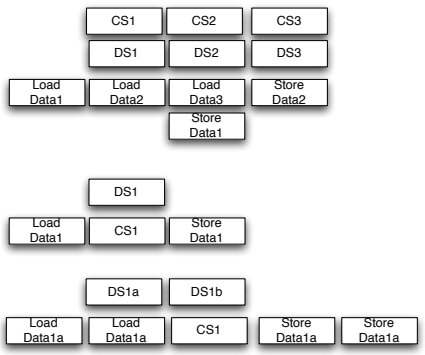

図 2 DS Pipeline

7.5 Data Segment Storage Type

Data Segment には位置と Identity を表す ID が付いている。Many Core 版ではメモリアドレス (64bit) を ID として使って良い。

SPU のような local memory では、hash を使って Data Segment の管理を行う。

Persistent な Data Segment では ID は使用する Database の table と key を表す。

7.6 Data Segment の処理の記述

Data Segment は Message Pack でもあり、Json 的な木構造を持っている。これが Cerium の Scene-Graph に相当する。

8 期待される効果

Data Segment API は、これから実装することになるが、

`Cerium` の既存の例題が動くこと

が一つの基準となる。PS3 が無事ならば PS3 でも動かしたい。Core i7 系、GPGPU 系、Open CL での共通のプログラミングフレームワークとして使用することができると期待している。

詳細な API は、これから決めることになると思うが、今の Cerium の API

参 考 文 献

[1] G. Agha and C. Hewitt. Concurrent programming using actors. In *Object-Oriented Concurrent Programming*. MIT Press, 1987.

[2] Sony Corporation. Cell broadband engine architecture, 2005.

[3] Sadayuki Furuhashi. MessagePack, March 2010.

[4] Shinji KONO. CbC, March 2008.

[5] 宮國 渡, 河野 真治, 神里 晃, 杉山 千秋 (琉球大学). Cell 用の Fine-grain Task Manager の実装 . 情報処理学会システムソフトウェアとオペレーティング・システム研究会, April 2008.

[6] 金城裕, 河野真治. Fine grain Task Manager Cerium のチューニング. 日本ソフトウェア科学会第 27 回大会 (2010 年度) 論文集, Sep 2010.

[7] 与儀 健人, 河野 真治 (琉球大学). Continuation based C コンパイラの GCC-4.2 による実装 . 情報処理学会システムソフトウェアとオペレーティング・システム研究会, April 2008.