日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

Cell Task Manager Cerium を用いたゲーム作成

小林 佑亮　　河野 真治

我々は PlayStation3 上でのゲームプログラミングをサポートする Cerium Engine を開発した。Cerium におけるゲーム開発ではオブジェクトの描画や動作を Task という単位で管理しており、この Task を動的に SPE に割り振ることによってプログラムの並列化を図っている。現在、Cerium を用いたゲーム開発を進めており、その過程で様々なバグが発生することがわかった。本稿では Cerium によるゲーム開発の例と Test の手法について提案する。

1 概要

当研究室ではこれまで家庭用ゲーム機上でのゲームプログラミングの開発を行ってきた。過去には PlayStation や PlayStation2、 Game Boy Advance を用いており、現在は PlayStation3(以下 PS3) で動作するゲーム開発を行っている。

PS3 では Fedora や Yellow Dog Linux といった Linux OS を動作させることができるので (現在の公式のサポートは終了している) C や C++ といったプログラム言語を用いて Linux 上でプログラミングすることが可能となっている。しかし PS3 の Architecture である Cell Broadband Engine は複数の SPE を使用する並列プログラミングが求められている。そこで我々は Cell のような Many Core Architecture を用いた、並列プログラムの開発をサポートするフレームワークとして Fine Grain Task Manager を開発した。この Task Manager を用いたゲーム開発フレームワークが Cerium である。

Cerium におけるゲーム開発ではオブジェクトの動作 (Move) と、相互作用 (collision) を Task として記述し、その計算に必要なパラメータを SPE に転送して処理させる。しかし、Task の依存関係 (dependency) や転送するパラメータの整合性が取れないと期待した動作が行われない。本研究では Cerium を使ったゲーム開発を通して生じたバグを考察し、効果的なテスト手法を提案することで、ゲームプログラムの動作を保証することを目的とする。

development of games on Cell Task Manager Cerium
Yusuke KOBAYASHI, Shinji 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 Engi- neering and Science, University of the Ryukyus..

2 Cell Broadband Engine

Cell Broadband Engine は SCEI と IBM、東芝によって開発された CPU である。2 thread の PPE(PowerPC Processor Element) と、8 個の SPE(Synergistic Processor Element) からなる非対称なマルチコアプロセッサであり、高速リングバスである EIB(Element Interface Bus) で構成されている。PS3 Linux では 6 個の SPE を使うことが出来る。(図 1)

2.1 PPE(PowerPC Processor Element)

PPE は Cell Broadband Engine のメインプロセッサで、複数の SPE をコアプロセッサとして使用することができる汎用プロセッサである。メインメモリや外部デバイスへの入出力、SPE を制御する役割を担っている。PPU(PowerPC Processor Unit) は、PPE の演算処理を行うユニットで、PowerPC アーキテクチャ

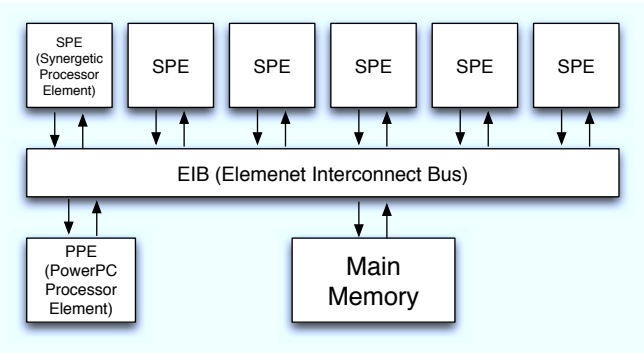

図 1 Cell Broadband Engine Architecture

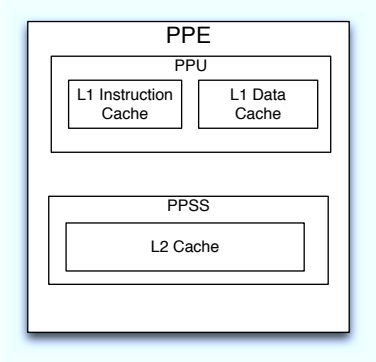

図 2 PPE(PowerPC Processor Element)

をベースとした命令セットを持つ。PPSS(PowerPC Processor Storage Subsystem) は PPU からメインメモリへのデータアクセスを制御するユニットである。(図 2)

2.2 SPE(Synergistic Processor Element)

SPE には 256KB の Local Store(LS) と呼ばれる、直接参照できるメモリ領域があり、バスに負担をかける事なく並列に計算を進めることが出来る。SPE からメインメモリへは、直接アクセスすることは出来ず、SPE を構成する一つである MFC (Memory Flow Controller) へ、チャネルを介して DMA(Direct Memory Access) 命令を送ることで行われる。(図 3)

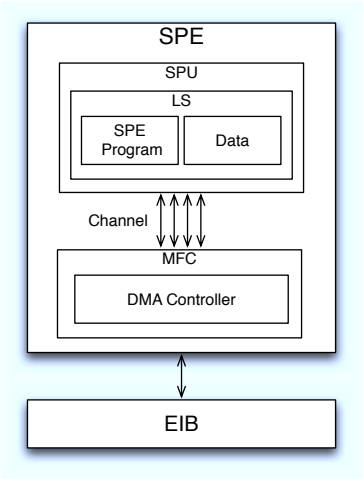

図 3 SPE(Synergistic Processor Element)

3 Cerium Engine

Cerium は 独自の Rendering Engine と Scene Graph、Task Manager の 3 つによって構成される。ゲーム中のオブジェクトの振る舞いやルールは Scene-Graph によって管理され、それらの動きや Rendering の処理を動的に SPE に割り振るカーネルとして Task Manager が用いられる。

Cerium は C++で実装されており、画像の読み込みや入力デバイスは SDL を用いて行っている。

3.1 SceneGraph

Cerium ではゲーム中の一つの場面 (Scene) を構成するオブジェクトやその振る舞い、ゲームのルールの集合を SceneGraph としている。SceneGraph のノードは親子関係を持つ tree で構成される。(図 4) 親子関係とは、親オブジェクトの回転や並列移動等の行列計算による頂点座標の変更が、子オブジェクトにも反映する関係のことである。これは子に対してスタックに積まれた親の変換行列を掛けることで実現できる。

3.2 Rendering Engine

Cerium の Rendering Engine では、以下の 3 つの Task を持つ。

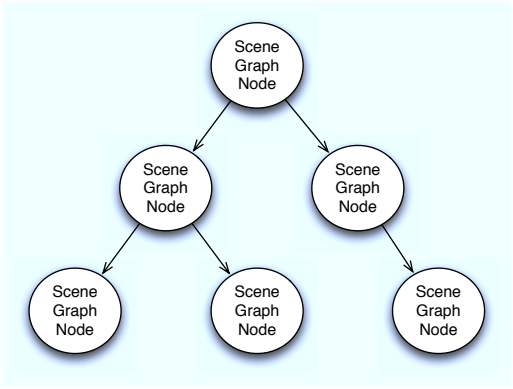

図 4 SceneGraph tree

- SceneGraph が持つ Polygon の座標から、実際に画面に表示する座標の計算を行い、PolyPack を生成する Task
- PolygonPack から同じ Y 座標を持つ線分の集合である SpanPack を生成する Task
- SpanPack を Texture を読み込みながら Z Buffer を用いて描画する Task

この 3 つの Task は表示画面毎にパイプライン的に実行される。そのため、Cerium では並列度を維持することが出来る。

3.3 Task Manager

Task Manager は Task と呼ばれる、分割された各プログラムを管理する。Task の単位はサブルーチンまたは関数とし、Task 同士の依存関係を考慮しながら実行していく。

3.3.1 Task の入出力

Task に渡す入力として、 add_inData がある。add_inData(addr, size) は、Task に渡すデータのアドレスと、そのデータのサイズを引数として入力する。このデータは DMA 転送されるため、addr は 16 バイトアライメントが取れており、size は 16 バイト倍数である必要がある。

Task の出力先は add_outData を使用する。使用方法は add_inData と同じで、アライメント、バイト数にも気をつける必要がある。

3.3.2 Task の依存関係

Task Manager は Task 依存を解決する機能を持っている。以下は記述例である。

```
task3->wait_for(task1);
task3->wait_for(task2);
```

wait_for は複数の Task を指定できる。上記の場合は task3 が task1、task2 の二つの Task 終了を待つ形となる。

3.3.3 Task 終了時に実行される関数

set_post 関数を使うことによって Task が終了した際、メインスレッドで実行される関数と、その引数を指定できる。

```
int data = 3;
task->set\_post(func1, (void*)data);


void
func1(void *data)
{
  printf("func1: data = %d\n", (int)data);
}


//実行結果
func1: data = 3
```

set_post により、ユーザ側でも Task が終了したということを検知できる

4 Cerium を用いたゲーム開発の手法

多くのゲームでは毎フレーム、オブジェクトのパラメータを計算し、その結果によって Rendering や collision の判定を行う必要がある。Cerium 環境では毎フレーム、オブジェクトの動作 (Move) を記述した Task を生成しオブジェクトのパラメータと一緒に SPE に送ることで並列性を持ったゲームプログラムを実現できる。しかし SPE の LS は 256KB しか無い為 (2.2 節)、Polygon や Texture などの余分な情報の入った SceneGraph そのものを送るのは望ましくない。そこで計算に必要なパラメータだけを持つ Property をユーザ側で定義し、Task と共に SPE に送る手法を取る。SPE に計算された値は post_func を用いて SceneGraph に反映され、Rendering Engine

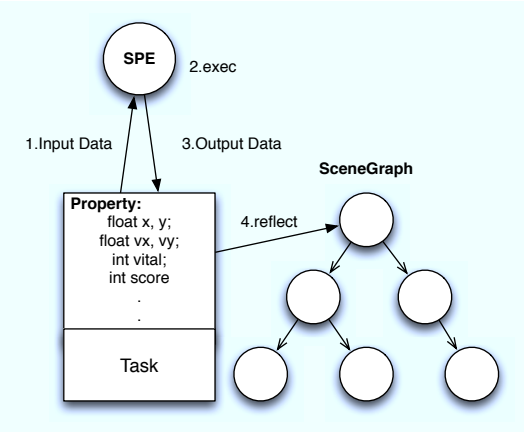

図 5 Property を用いた値の計算と SceneGraph への反映

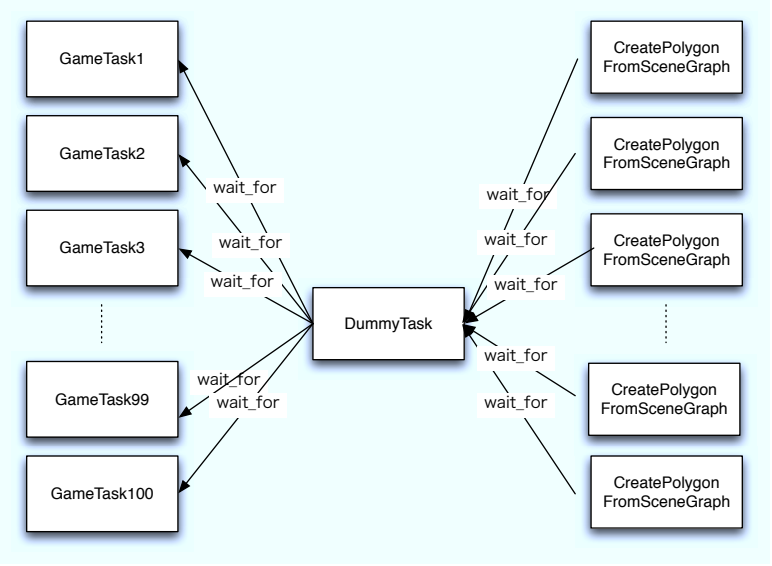

図 6 DummyTask による GameTask と Rendering の実行順序の調整

によって描画される (図 5)。

以下は Task を生成するルーチンの例である。

```
HTask *task = sgroot->tmanager->create_task(id);
  task->set_cpu(SPE_ANY);
  task->add_inData(property, size);
  task->add_outData(property, size);
  task->set_post(post_func, (void*)property, 0);
  task->spawn();
```

5 Cerium を用いたゲーム開発におけるデバッグ

Cerium を用いたゲーム開発を進める過程で発生したバグの例とそれを解決する為に用いた手法を示す。

5.1 Task の依存関係によるバグ

4 章の方法を用いてゲームの作成を行ったところ、Task による Property の計算が描画に反映されなかった。そこでプログラム上で動作している主要な Task の実行順序を調べてみると、以下のようになった。

1. GameTask(Game 本体の Task)
2. CreatePolygonFromSceneGraph(SceneGraph から PolygonPack を生成)
3. GameTask-¿post_func(GameTask によって計算された値を SceneGraph に反映)
4. CreateSpan(PolygonPack から SpanPack を生成)
5. DrawSpan(SpanPack から Texture を読み込みながら描画)

Cerium による Rendering は 3 つの Task によって実現されているが 3.2 その最初プロセスである CreatePolygonFromSceneGraph が SceneGraph への Property の値の反映より先に実行されているのがわかる。

これ対して、全ての GameTask の終了を wait_for で待つ DummyTask(何も動作しない Task) を Rendering Task との間に挟むことによって Task の実行順序を調整した (図 6)。

5.2 Task に送る Input Data と Output Data の比較

コントローラーの入力を見て、Property の値を変化させる Task を作ったところ、期待通りの動作をしなかった。この為、この Task で処理する Input Data と Output Data の比較を行ったところ、Output Data の Property にでたらめなパラメータが入っている事がわかった。この時点で Task 内になんらかの不当な処理がされていると予想した。

この Task では SPE に コントローラーの入力の値と Property を送っているが、Cerium の仕様により、これらのデータは void* 型で Input され、Task 内で必要な型に cast されて使用される。今回のバグは Task 内において、コントローラーの入力と Property の型を逆にして cast してしまったのが原因であることがわかった。

6 まとめと今後の課題

今回は Task の依存関係を列挙することにより、実行順序に関するバグを発見することが出来た。また、Task の Input, Output を調べることにより、Task 内での不正な処理を発見した。

今後はゲームの実装を進めるとともに、コードのメトリクス (数値データ) を取りながらテストの評価をしていく予定である。

参 考 文 献

[1] 宮國 渡. Cell 用の Fine-Grain Task Manager の実装. 琉球大学大学院 理工学研究科 情報工学専攻 平成 20 年度 学位論文 2008.

[2] 高橋 寿一. 知識ゼロから学ぶソフトウェアテスト. 翔泳社 2005.

[3] KENT BECK. テスト駆動開発入門. PEARSON Education Japan 2003.