Cell 用の Fine-grain Task Manager の実装

宮 國渡[†1] 河 野 真 治[†2]
神 里 晃[†3] 杉 山 千 秋[†4]

PlayStation 3 では、搭載されている Linux を用いてゲームを開発することができるが、GPU の詳細が公開されていないため、Frame Buffer 上に描画する必要がある。Frame Buffer 上の描画は非常に低速である。本研究では Cell 用の Task Manager を実装し、Frame Buffer 上で高速な 3D Graphics Renderer を開発する。

Wataru MIYAGUNI,[†1] Shinji KONO,[†2] Akira KAMIZATO[†3]
and Chiaki SUGIYAMA [†4]

In PS3, the game can be developed by using installed Linux. However, because details of GPU are unpublished, it is necessary to draw on Frame Buffer. Drawing on Frame Buffer is very low-speed. In this research, we implement Task Manager for Cell and develop 3D Graphics Renderer high-speed on Frame Buffer.

1. 研究の目的

近年、CPU の性能向上は、クロックサイクルを上げることよりも、複数の CPU コア (Many Core Architecture) を導入することにより得られるようになってきている。しかし、Many Core Architecture のプログラムは複雑であり、その信頼性を確保することは難しい。

本研究では、Many Core Architecture 向けの Fine Grain Task OS を設計する。この OS では、Amdahl 則を考慮して、定常的な並列性を細粒度タスクを使って実現する。そのために、シーングラフレベルから並列実行を考慮したプログラムとする。

細粒度タスク自体や、タスク全体のデバッグを容易にするために、同じタスクが Mac OS X や Linux、PS3 上など複数の環境で動くようにする。また、Thread を多用せず、細粒度タスク内での同期は行わない。これにより、並列プログラミングの経験の低いプログラマでも容易に使用できる。

例題として、本研究室で作成した、Rendering を含む PS3 上のゲームプログラム用 OS である Cerium[1)] を用いる。

Cerium は、次の 3 つから構成される。

- Scene Graph
- Rendering Engine
- Fine Grain Task Manager

Cerium では、Cell の性能を十分に引き出し、今まで作成してきた PS、PS2 のゲームプログラムを PS3 へ容易に移植できることを目的とする。

2. Cell

Cell Broadband Engine[2)] は、SCEI と IBM 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 (Synergetic Processor Element) を持ち、EIB と呼ばれる高速リングバスで構成されている。本研究で用いた PS3Linux (Fedora 8) では、6 個の SPE を使うことができる (**図 1**) 。

SPE には 256KB の Local Store (LS) と呼ばれる、SPE から唯一直接参照できるメモリ領域があり、バスに負担をかけること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

†1 琉球大学理工学研究科情報工学専攻
Interdisciplinary Infomation Engineering, Graduate School of Engineering and Science, University of the Ryukyus.
†2 琉球大学工学部情報工学科
Infomation Engineering, University of the Ryukyus.
†3 琉球大学理工学研究科情報工学専攻
Interdisciplinary Infomation Engineering, Graduate School of Engineering and Science, University of the Ryukyus.
†4 琉球大学工学部情報工学科
Infomation Engineering, University of the Ryukyus.

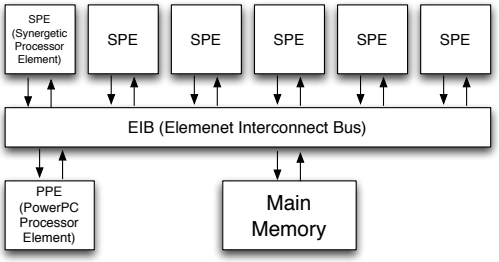

図 1 Cell Architecture

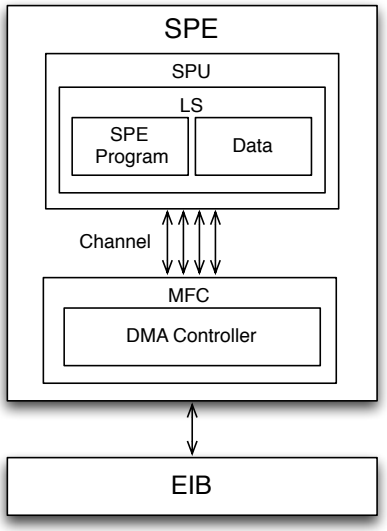

図 2 Synergetic Processor Element

して DMA (Direct Memory Access) 命令を送ることで行われる (**図 2**) 。

SPE はグラフィックスに適した、4 つの固定小数点、浮動小数点を同時に演算する命令などを持ち、PPE に比べて高速な演算が可能である。そのため、ほとんどの演算を SPE 上で行わせることが推奨されている。

2.1 SPURS

ここでは、現在発表されている Cell の開発環境である SPURS[3] について説明する。

SPURS は、閉じた並列分散と考えることができる Cell の環境で、いかに効率よく動作させるかということを考えたシステムである (**図 3**) (**図 4**) 。

Cell の性能を存分に生かすためには SPE を効率よく使い切ることである。SPE の動作を止めることなく、同期を最小限に行う必要がある。そこで SPURS では SPE を効率よく利用するために、PPE に依存せずに SPE コードを選択し、実行することと機能は効率重視で割り切ることを挙げている。現在 SPURS は一般には公開されていない。

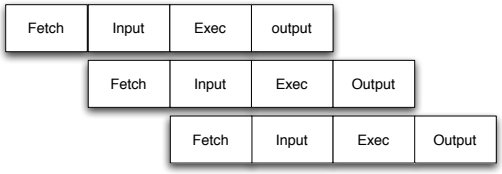

図 3 SPURS Pipeline

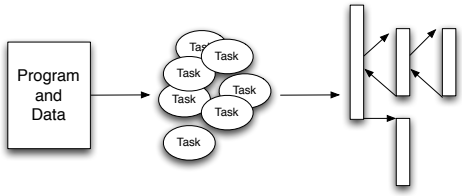

図 4 SPURS Task

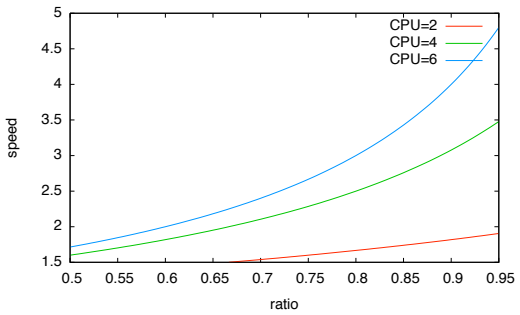

図 5 Amdahl 則

3. Many Core 上のプログムの特徴

従来の逐次型のプログラムでは、Cell といった Many Core の性能を十分に引き出すことは出来ない。

3.1 定常的な並列度の必要性

並列実行には Amdahl 則[4] があり、プログラムの並列化率が低ければ、その性能を生かすことは出来ない。0.8 程度の並列化では、6 CPU でも 3 倍程度の性能向上しか得られない (**図 5**) 。

高い並列度ではなくとも、恒常的に並列度を維持する必要がある。このため、逐次型のプログラムの一部を並列化するという手法では不十分である。LSI などのハードウェアの場合は、演算の対象がもともと多量の演算とデータパスを持つので、並列計算の効果を定常的に得られることが多い。しかし、C 等で記述されたプログラムでは、for 文や配列のアクセス等に並列性が隠されてしまい、それを引き出すことが難しい。

プログラム中の並列度は、主に二つの形で取り出すことが出来る。

- データ並列：配列や木の中の個々の要素に対して並列に実行する

- パイプライン処理 : 複数の逐次処理の隣同士を重ねて実行する

この二つを同時に用いることで、定常的な並列度を維持することが可能となることがある。パイプライン処理は、プログラム中で階層的に使われることが多い。

データ並列とパイプライン処理を可能にするためには、プログラムとデータの適切な分割を行う必要がある。for 文、あるいは、木をだとって処理する個々のステートメントがプログラムの分割の対象となる。

3.2 デバッグ

並列プログラムの特徴として、デバッグが難しいことも挙げられる。実行が非決定的 (同じ状態で実行しても同じ結果が異なる) 場合があり、バグの状態を再現することが難しい。また、個々の Core 上のデータを調べる必要もある。

4. OSMesa/SDL を用いた Cell 上でのゲームプログラミング

PS3 では、搭載された Linux を用いて、PS3 上で動くゲーム開発することができる。しかし、現在 GPU (Graphics Processing Unit) の API は一般には公開されていないため、GPU を使った描画はできない。Frame Buffer 上には直接描画することはできるため、Cerium の先行研究[5] では、Frame Buffer Engine 用のレンダリングエンジンである OSMesa[6] と、マルチメディアライブラリの SDL[7] の二つを用いて開発を行っている。例題として、テクスチャを貼っていない一つの立方体を回転させるというプログラムを用いた。

PPE のみを使用した場合の実行速度は約 18 FPS (Frame Per Second) という結果になった。これは非常に遅い結果で、その理由は、SPE を使用していないため、Cell の能力を十分発揮できていないからだと考えられる。

次に、OSMesa の一部の機能を SPE に演算させることにより高速化を図っている。

その際の実行結果を **表 1** に示す。

表 1 実行速度 (描画領域:1920x1080)

実行環境	実行速度
SDL(1.2)+OSMesa(6.5.2)	18 FPS
SDL(1.2)+OSMesa(6.5.2) with SPE	24 FPS
SDL(1.2.13)+OSMesa(7.0.2) with SPE	43 FPS

表 1 より、SPE に処理を任せれば実行速度は速くなるということを示している。同時に、OSMesa を細分化し、できるだけ SPE を使うような設計ができればより速くなる可能性を示すものである。

しかし、OSMesa は巨大なマクロによるプログラム記述やコピーの多用、巨大な構造体等がある。テクスチャに関しても、使用するテクスチャモードによっていくつものコードが呼び出される。それら全てのコードに対しての細分化や、後に拡張をすることは難しい。

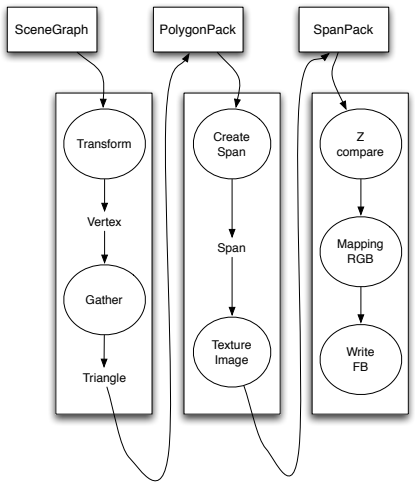

図 6 Cerium の 要素

5. Cerium

Cerium は Scene Graph, Rendering Engine, Task Manager から構成され、

(1) Scene Graph が持つ Polygon の座標から、実際に表示する座標の計算を行い、PolygonPack を生成する
(2) PolygonPack から、同じ Y 座標を持つ線分の集合 SpanPack を生成する
(3) SpanPack を (Texture を読み込みながら) Z Buffer を用いて描画する

という 3 つのタスクを持つ (**図 6**)。

Ceritum は、Scene Graph、PolygonPack、SpanPack に対してデータ並列実行を行う。さらに、この 3 つのタスクは表示画面毎にパイプライン的に実行される。そのため、Ceritum では並列度を維持することができる。

ここからは、Cerium を構成する 3 つのシステムについて述べる。

5.1 Scene Graph

本研究では、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 Scene Graph とする[8] 。Scene Graph のノードは親子関係を持つ木構造で構成される (**図 7**) 。そのため、Scene Graph の各ノードに対してデータ並列実行することが可能と言える。

Scene Graph によりプログラムが個々のゲーム場面に分割され、ゲーム場面の遷移を StatePattern を用いて記述すると、if,case 文が排除できるので、プログラムの可読性と独立性が向上する。これにより、過去の PS、PS2 によるゲームプログラムの移植や改良

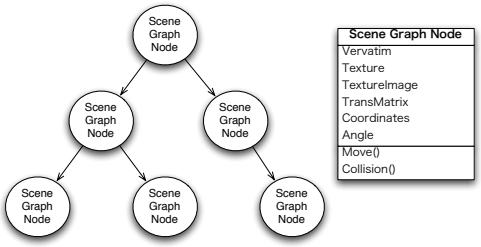

図 7 Scene Graph Structure

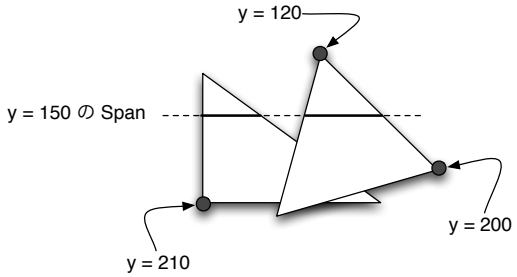

図 8 Span

にかかる作業時間短縮が見込める。

Scene Graph Node は以下のようなデータと動作を持つ。

- データ
  - Vervatim : ポリゴンオブジェクトの頂点座標
  - Texture : ポリゴンオブジェクトのテクスチャ座標
  - TextureImage : テクスチャイメージ
  - TransMatrix : ポリゴンオブジェクトの変換行列
  - Coordinates : オブジェクトの座標
  - Angle : オブジェクトの角度
- 動作
  - Move : 自律的なオブジェクトの動き
  - Collision : 他ノードと衝突したときの動き

今回は Scene Graph の作成に、オープンソースの 3D モデリングツールである Blender[9] を用いる。Blender でオブジェクトを作成し、ポリゴン情報やテクスチャ情報が記述された xml ファイルを出力する。その xml ファイルを Rendering Engine が受け取って Polygon を生成する。

5.2 Rendering Engine

Rendering Engine は Polygon から Span を生成する部分と Span に RGB をマッピングし描画する部分と二つに分けることが出来る。Span とは、Polygon に対するある特定の Y 座標に関するデータを抜き出したものである (**図 8**) 。

生成された Span を SPE が DMA で受け取り、Z

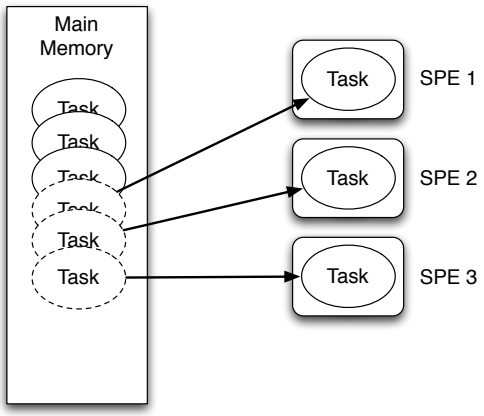

図 9 Task Manager

Buffer を用いて Frame Buffer に書き込むピクセルの座標に対応する RGB 値を、テクスチャデータから取り出して書き込む。テクスチャデータは、プログラム起動時に一度だけ DMA で全て取得する。

Frame Buffer は mmap されているので、ピクセルデータを DMA 転送することで描画している。

5.3 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ライブラリである。Task 同士の依存関係を考慮しながらメモリ上にマッピングし、SPE 上ではそのプログラムを DMA 転送によりロードする (**図 9**) 。SPE は 256KB という小さなデータ量しか持てず、大量のプログラムを全て SPE 上に置いておくことはできない可能性がある。そのため、必要な時に必要な実行プログラムだけが SPE 上にロードされていることが望ましい。

現在実装されている Task Manager の API を **表 2** に示す。

表 2 Task Manager API

create_task	Task を生成する
spawn	Task を実行 Task Queue に登録する
set_depend	Task の依存関係の考慮
set_cpu	Task を実行する CPU の設定
run	実行 Task Queue の実行

以下に Task Manager を使った記述例を示す。

```
int
main(void)
{
    TaskManager *manager = new TaskManager;
    Task *task1, *task2;

    /**
     * cmd  : 実行するタスク ID
     * size : in_addr で取得するデータのバイト数
```

```
     * in_addr  : 入力データ元アドレス
     * out_addr : 出力データ先アドレス
     */
    task1 = create_task(CMD_RUN1, size1,
                        in_addr1, out_addr2);
    task2 = create_task(CMD_RUN2, size2,
                        in_addr2, out_addr2);

    task1->spawn();
    task2->spawn();

    manager->run();

    return 0;
}
```

ここからは、Task Manager を構成する特徴について述べる。

5.3.1 Task の定義

タスクの定義を以下に示す。

```
class Task {
public:
    int command;
    int size;
    unsigned int in_addr;
    unsigned int out_addr;
    struct task *self;
};

class HTask : public Task {
public:
    // 自分を待っているタスクキュー
    TaskQueue wait_me;

    // 自分が待つタスクキュー
    TaskQueue wait_i;

    // 実行する CPU
    CPU_TYPE  cpu_type;
};
```

Task クラスは SPE が実行するタスクの単位オブジェクトである。SPE はメインメモリの `in_addr` から DMA で入力データを取得し、command に対応するコードを実行し、結果を メインメモリの `out_addr` に DMA で送信する。これらの処理はパイプラインに沿って動作する (5.3.2) 。

また、SPE が Task を取得する際、一つずつではなく、Task の集合である TaskList を DMA で取得する。DMA の回数を抑えることで実行速度を上げるためであるが、TaskList のサイズを大きくしすぎると SPE の LS に入らなくなる可能性がある。本研究で

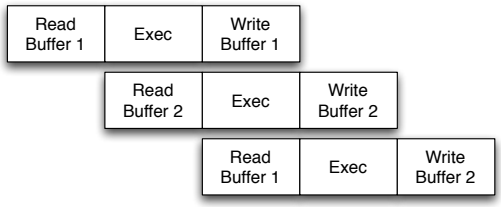

図 10 Pipeline

並列に実行するタスクの数は、図 6 の (SceneGraph → Polygonpack) タスクと (PolygonPack → SpanPack) タスク、そして SpanPack の数 × Rendering タスクとなる。そのため、SPE を 6 個使うと考えると TaskList に格納する Task の数は最大で 25 としている。

HTask クラスは TaskManager で管理する実行前のタスクオブジェクトである。`wait_me, wait_i` はタスク依存の条件 (5.3.3) に、`cpu_type` は PPE と SPE のタスクの切り替え (5.3.4) に用いる。

5.3.2 スケジューラ

Cell ではそれぞれのコアがメインメモリを直接参照することは出来ず、DMA 転送によりデータをやりとりする。DMA は CPU を介さず直接データ転送を行う方式である。SPE は DMA 完了を待たずに他の処理を行うことが出来るので、DMA のレイテンシを隠すことが出来る。また、ダブルバッファリングを行うことでパイプライン処理が可能となる (**図 10**) 。

パイプライン処理を入れたスケジューラを以下に示す。

```
SpeTaskBase *task1, *task2, *task3;

do {
    task3->write();
    task2->exec();
    task1->read();

    taskTmp = task3;
    task3 = task2;
    task2 = task1;
    task1 = task1->next();
} while (task1);
```

TaskList にある Task が全て終了し、PPE からプログラム終了のメッセージを受け取ったら while 文を抜ける。

Task Manager を 図 6 に適用させると、**図 11** のようにパイプライン的に動作する。

SpeTaskBase クラスは、スケジューラによって実行されるオブジェクトである。スケジューラ自身のタ

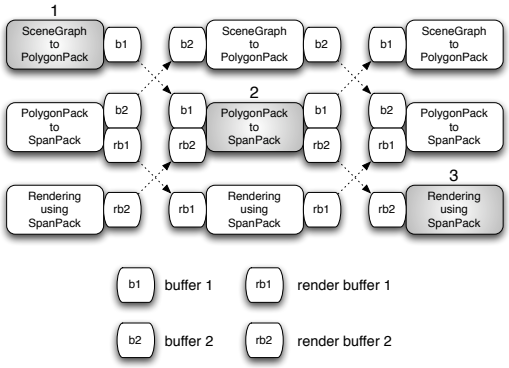

図 11 Task Manager が行う Pipeline

スクとして、以下のようなタスクがある。これらのオブジェクトは全て SpeTaskBase を継承している。

- SpeMail：PPE からのメッセージを待つ
- SpeTaskList：PPE から TaskList を取得する
- SpeTask：TaskList にある Task を実行する
- SpeExit：SPE を終了させる

5.3.3 タスク依存

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

```
// task2 は task1 が終了してから開始する
task2->set_depend(task1);
```

タスク依存が満たされたものをアクティブキューにいれ、SPE を起動する。SPE はアクティブキューから、処理するコードとデータを取得し、自律的に実行を行う。終了したタスクは PPE に対して終了のコマンドを発行し、PPE はそれを見てウェイトキューのタスクを調べ、タスク依存を満たしたものが出てきたらアクティブキューに入れ、SPE を再び起動する。

5.3.4 PPE コードと SPE コードの互換性

Cerium では、SPE を用いる Cell 上でのみ動作する SPE プログラムの他に、Linux や Mac OS X などで動く、アーキテクチャに依存しない FIFO プログラムを記述することが可能である。FIFO プログラムの場合、スケジューラはメインメモリ上で動作する。DMA 命令は、バッファのアドレス参照によりシミュレートする。パイプライン処理も、SPE プログラムと同様の動作を行う。

Task Manager には、タスクを PPE、SPE のどちらで動かすかを決める機能がある (`set_cpu`)。

```
task1->set_cpu(CPU_PPE); // PPE 上で実行される
task2->set_cpu(CPU_SPE); // SPE 上で実行される
```

これを用いることにより、環境依存によるプログラム変換は分割したタスクの部分だけとなり、全体の変換は必要ない。

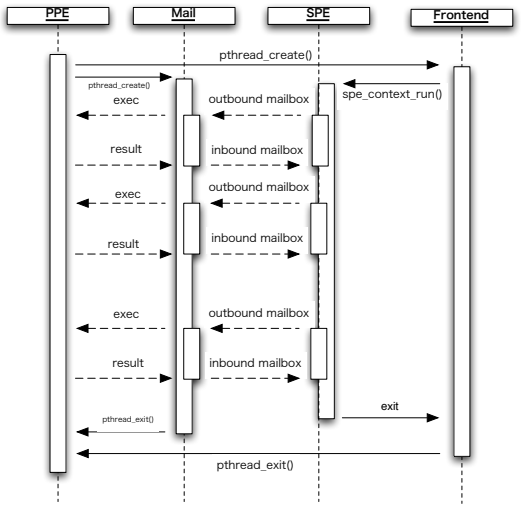

図 12 PPE, SPE threads

このことはデバッグにおいても有効である。デバッグが困難な並列プログラムである SPE プログラムの前に、まずは FIFO プログラムで開発を進める。FIFO プログラムはシーケンシャルな記述なため、二分法を用いたデバッグが可能となる。仕様やアルゴリズムの正しさを確認できたら、SPE プログラムに移行する。デバッグに関する詳細は第 6 節で述べる。

5.3.5 PPE と SPE 間の同期

Cerium では、以下の二種類のスレッドを扱う。

(1) SPE に対応した PPE 上の Thread
(2) Mailbox を待つ Thread

スレッド (1) では、各 SPE の起動と終了を見るだけである。スレッド (2) では、Outbound Mailbox (SPE → PPE メッセージ) を見て、その内容に対応する処理を PPE 上で行う。対処した結果を Inbound Mailbox (PPE → SPE メッセージ) で伝え、受け取った SPE はタスクを再実行する (**図 12**)。

Mailbox とは SPE の MFC 内の FIFO キューであり、PPE と SPE 間の 32 ビットメッセージの交換に用いられる[10]。通常、スレッド間で待ち合わせを行うと処理が止まってしまい、並列度が下がってしまうことがあるが、Mailbox はメッセージ交換なので待ち合わせを避けることが可能である。

6. Cerium の開発行程

Cerium 自身や Cerium を用いたプログラムの作成では、以下の段階にそれぞれ実装とテストを行う。

(1) C によるシーケンシャルな実装
(2) SPE を考慮したデータ構造 (PolygonPack, SpanPack) を持つ実装

(3) コードをタスクに分割し、FIFO キューでシーケンシャルに実行する実装
(4) タスクを SPE に割り当て並列実行する実装

段階 (1) の実装は Task Manager を用いず、プログラムのアルゴリズムの信頼性を確認するために用いる。

段階 (2) ではデータの変換が必要になり、段階 (1) と同じ結果を得られるかどうかを検証する。この段階までは、入力に対して出力が一意に決まる状況であり、テストは容易である。シーケンシャルな実装であるため、デバッグも二分法により容易に行うことが出来る。

段階 (3) の実装からは Task Manager を用いる。この段階まではアーキテクチャに依存しないので、ターゲットが開発途中の段階でも記述することが可能である。また、FIFO スケジューラを用いずに Random スケジューラを実装することにより、並列実行特有の非決定的な実行が導入される。非決定的な動作においても、段階 (2) までと同じ仕様を満たすことを検証する必要がある。これは、逐次型プログラムでは出て来ない問題である。

段階 (4) では、段階 (3) までが動いていれば問題なく動作すると期待される。また、SPE タスクを、SPE が持つ命令セットを用いて最適化することにより、更なる性能向上が期待できる。

段階 (3) から 段階 (4) へのプログラムの変換は非常に容易である。Task Manager の API である set_cpu() を用いることにより、Task を PPE で実行するか SPE で実行するかを明示的に書くことが出来るからである。

7. Cerium の評価と考察

Cerium Rendering Engine を用いて、OSMesa と同様に立方体を回転させる処理を評価する。今回、図 6 で示した Task に対する CPU の割り振りは**表 3** とする。

表 3 Task の実行 CPU

Task	CPU
SceneGraph to PolygonPack	PPE
Polygonpack to SpanPack	PPE
Rendering	SPE

PPE/SPE に対するコンパイルの最適化レベル及びその時の実行 FPS を**表 4** に示す。描画領域は 640x80 である。

表 4 Cerium 実行速度 (640x80)

PPE	SPE	FPS
最適化なし	最適化なし	263 FPS
-O2	最適化なし	317 FPS
最適化なし	-O2	812 FPS
-O2	-O2	1355 FPS

表 4 より、SPE に対する最適化が非常に有効であるといえる。

また、描画領域の大きさと実行速度は反比例すると考えると、1920x1080 の時、先行研究の結果と合わせると **表 5** のようになる。

表 5 実行速度 (1920x1080)

	実行環境	実行速度
1	SDL(1.2)+OSMesa(6.5.2)	18 FPS
2	SDL(1.2)+OSMesa(6.5.2) with SPE	24 FPS
3	SDL(1.2.13)+OSMesa(7.0.2) with SPE	43 FPS
4	Cerium	33 FPS

Cerium は 環境 1, 2 は上回っているものの、環境 3 の速度を下回っている。この理由としては、SPE 上で動かしていないタスクがあるのが挙げられる。

現在、Cerium には DMA で SPE 上にプログラムをロードする機能を実装していないため、必要な時に必要なプログラムだけを SPE 上に置くということができない。SPE の LS の容量では、全てのタスクを SPE 上に乗せることはできないため、現在は Renderer のタスクだけを SPE 上にマッピングしてある。

また、プログラム開始直後にテクスチャデータを全て SPE 上にロードしている。そのため、必要無いデータまで SPE 上に置いている状態なので、メモリ領域を無駄に消費している。

Cerium の描画領域の横幅が 640 に抑えてあるのも、上記の問題が原因で 1920 pixel 分の Z Buffer を SPE 上に置くスペースが無いからである。オーバーレイ機能を導入するという手法もあるが、ロード中は SPE が止まってしまうため好ましくない。

また、パイプラインが正常に動いているかなど、アルゴリズムの検証を行う必要もある。

8. ま と め

本研究では、Many Core Architecture 向けの Fine Grain Task OS を提案し、例題として Cell 上で動作する、ゲームプログラム用 OS である Cerium を開発した。Cerium を用いることで、PS3 上という限られた環境だけでなく、Linux や Mac OS X でもテストやデバッグを行うことが出来るため、並列プログラミング経験の低い学生の実験にも使用できる。

9. 今後の課題

ここでは、Cerium に関する課題を示す。

9.1 Scene Graph

カメラや光源、コントローラ等からの入力に対応するノードが未実装である

9.2 Rendering

必要最低限の機能は実装しているが、ゲームとしては以下の機能が必須といえる。

- Shading
- Alpha blending

また、現在は描画時に Z Buffer を用いているが、Z sort を用いることにより、SPE 上に Z Buffer の領域は必要なくなる。Z sort とは、面に対してどちらが手前にあるかというのを判断し、おくにある面から描画していく手法である。

9.3 Task Manager

プログラムを SPE 上にロードする機能は必須である。

このライブラリを学生が使用する場合、並列化のためのデータとコードの分割は、並列プログラミングの経験がない学生には難しいため、何らかのひな形を示す必要がある。

図 10 では、Task Exec の中で DMA の処理を行うことを想定した実装にはなっていない。Exec の中でアドレスの計算をし、それを元に DMA 転送をする、というのもありえる。しかし、ユーザが勝手に DMA 命令を発効すると、DMA 完了待ちやデータ重複など、パイプラインの流れが崩れてしまう恐れがある。それを回避する実装が必要である。

参 考 文 献

1) : SourceForge.JP: Project Info - Cerium Rendering Engine, `https://sourceforge.jp/projects/cerium/`.
2) Sony Corporation: Cell broadband engine architecture (2005).
3) Keisuke Inoue: SPU Centric Execution Model (2006).
4) Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, and Doug Lea: *Java Concurrency in Prac-tice*, Addison-Wesley Professional (2005).
5) Akira KAMIZATO: Cell を用いたゲームフレームワークの提案，琉球大学理工学研究科情報工学専攻平成 19 年度学位論文 (2008).
6) : The Mesa 3D Graphics Library, `http://www.mesa3d.org/`.
7) : Simple DirectMedia Layer, `http://www.libsdl.org/`.
8) Chiaki SUGIYAMA: SceneGraph と StatePattern を用いたゲームフレームワークの設計と実装 (2008).
9) : Blender.jp - Blender Japanese Website, `http://blender.jp/`.
10) Sony Corporation: Cell BroadbandEngine アーキテクチャ (2006).
11) Shinji KONO: 検証を自身で表現できるハードウェア、ソフトウェア記述言語 Continuation based C と、その Cell への応用，電子情報通信学会 VLSI 設計技術研究会 (2008).
12) International Business Machines Corporation, Sony Computer Entertainment Incorporated, Toshiba Corporation: *SPE Runtime Management Library* (2006).