Vol.2011-ARC-195 No.32
Vol.2011-OS-117 No.32
2011/4/14

ゲームフレームワーク Cerium TaskManager の改良

金城　裕[†1]　河野真治[†1]
多賀野海人[†1]　小林佑亮[†1]

ゲームフレームワーク Cerium TaskManager を開発した。プログラムは、PPE から Mail 機能を用いて各 SPE に処理が割り振られ並列実行される。しかし、 PPE からの Mail 応 答が遅い場合、 SPE の待ち時間が発生し、処理性能が低下する。また、 SPE への データ転送を頻繁に行うと転送のオーバーヘッドがかかる。これらを改良するために、Mail のタイミングの変更、SPE 内のキャッシュの実装を行った。その結果、例題を 用いた計測では、約 7 倍の処理速度向上を確認した。

improvement of Game Framework Cerium TaskManager

Yutaka Kinjyo,[†1] Shinji Kono,[†1] Kaito Tagano[†1]
and Yusuke Kobayashi [†1]

We have developed Cerium of Game Framework. Program is alloted processing to each SPE using Mail function from PPE, and It's excuted in parallel. But when Mail reply from PPE is late, the waiting time of SPE occurs, and the treatment performance fails. In addition, When PPE is frequency execute data transfer to SPE, overhead for a data transfer can't be being ignored any more. To improve these, We implemented timing change in the Mail and Cash in the SPE. As a result, We confirmed the about 7 times of processing speed improvement in measurement using an example program.

1. はじめに

我々は並列プログラミング用のフレームーク Cerium TaskManager を開発している。

†1 琉球大学
Ryukyu University

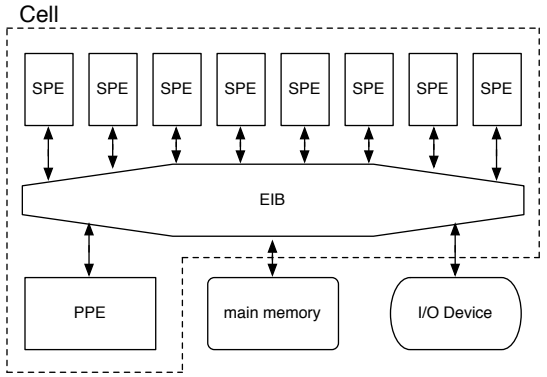

図 1 Cell Broadband Engine Architecture

Cerium は PS3/Cell, MacOSX, Linux 上で開発できる。それぞれのプラットフォームで同じプログラムで動作する。その中でも特に Cell に特化しているといえる。Cerium TaskManager では、関数やサブルーチンを Task として扱う。Task は Task 同士の依存関係に従って、実行される。Cell 上の場合、各 SPE に Task が割り当てられ、並列に実行される。Cerium は TaskManager に加え、SceneGraph, RenderingEngine で構成され、この 3 つでゲームフレームワークとして動作する。Task には input データ、output データ、依存関係を設定する。ゲームや、WordCount, Sort を例題として実装した。TaskManager と、TaskManager を使うユーザ側の両方の視点から、実装の際に直面した問題とその改良方法と効果について報告する。

2. Cell Broadband Engine

Cell Broadband Engine は、ソニー・コンピュータエンタテインメント、ソニー、IBM , 東芝によって開発されたマルチコア CPU である。Cell は、1 基の制御系プロセッサコア PPE (PowerPC Processor Element) と 8 基の演算系プロセッサコア SPE (Synergistic Processor Element) で構成される。各プロセッサコアは、EIB (Element Interconnect Bus) と呼ばれる高速なバスで接続されている。また、EIB はメインメモリや外部入出力デバイスとも接続されていて、各プロセッサコアは EIB を経由してデータアクセスをおこなう。PPE はメインメモリにアクセスすることができるが、SPE は、246KB の LS にのみ直

Vol.2011-ARC-195 No.32
Vol.2011-OS-117 No.32
2011/4/14

接アクセスできる。DMA を用いると、SPE はメインメモリにアクセスできる。本研究で用いた PS3Linux (Yellow Dog Linux 6.2) では、6 個の SPE を使うことが出来る (**図 1**) この PPE と SPE の 2 種類の CPU を、プログラマ自身が用途に合わせて適切に使い分けるように考慮する必要がある。

2.1 Mailbox

Cell の機能に Mailbox がある。Mailbox は PPE と SPE との間を双方向で、32bit のデータの受け渡しが可能であり、FIFO キュー構造になっている。Mailbox のひとつに SPE から PPE へデータを渡すためのキュー、SPU Outbound Mailbox があり、最大 1 個までのデータを蓄積できる。もし、Outbound Mailbox がすでに 1 個のデータを保持している場合には、SPE プログラムは PPE プログラム側でキューからデータを読み出すまでデータの書き込みを待ち続ける。

2.2 DMA 転送

SPE は LS 以外のメモリに直接アクセスすることができず、PPE が利用するメインメモリ上のデータにアクセスするには DMA (Direct Memory Access) を用いる。DMA 転送とは、CPU を介さずに周辺装置とメモリとの間でデータ転送ことで、Cell の場合はメインメモリと LS 間でデータの転送を行う。DMA 転送するデータとアドレスにはいくつか制限がある。転送データが 16 バイト以上の場合、データサイズは 16 バイトの倍数で、転送元と転送先のアドレスが 16 バイト境界に揃えられている必要がある。転送データが 16 バイト未満の場合、データサイズは 1,2,4,8 バイトで、転送サイズに応じた自然なアライメントである (転送サイズのバイト境界に揃えられている) ことが条件となる。

3. Cerium の改良

Cerium TaskManager では PPE で Task を定義し、PPE から SPE に Task を割り振る。SPE は DMA 転送（2.2 節）によって、Task と、Task で用いるデータを受け取る。DMA 転送を用いる場合、待ち時間が発生し、その間 SPE の処理が止まる。そのため、転送をパイプライン化し、DMA 転送の待ちを隠す必要がある。Cerium では SPE にスケジューラを持ち Task とデータ の 読み込み、実行、書き出しをパイプライン化している。(**図 2**) Task は一つずづ転送されるのではく、ある程度の数を集めて、TaskList として転送される。

3.1 MailQueue

Task には依存関係が設定でき、PPE 側で解決する。実行完了した Task の情報を SPE 側から PPE 側に 通知するために Cell の Mailbox 機能を使用した（2.1)。SPE スケジューラは Task が処理完了になる毎に、Mail を Outbound Mailbox に書きこむので、PPE 側で Mail の読み込みが間に合わないと、待ちが入り、SPE の処理が止まってしまう。

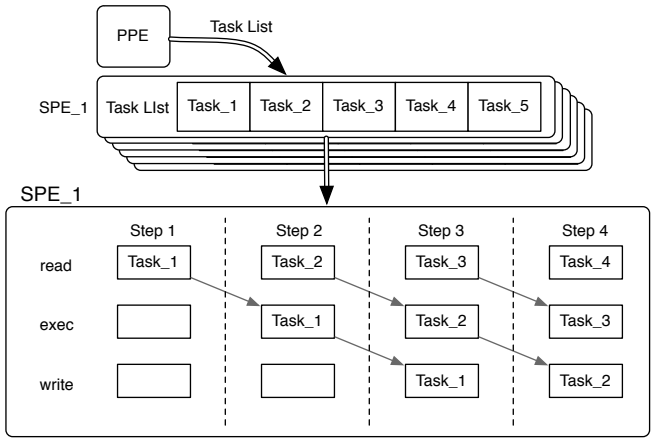

図 2 スケジューラ

これを解消するために MailQueue を導入した。MailQueue は、SPE から書き込みきれない Mail を一時的にキューに退避させるものである。TaskList を書きだす時に溜まった Queue の中身をすべて書き出す。Task 完了を知らせる Mail 書き出しの待ちは、Task 毎から、TaskList 毎になる。TaskList のサイズは 32。MailQueue を有効にしたときの実行速度は以下にようになる速度比較には、RenderingEngine を使ったボールが跳ねる例題 ball_bound を用いた。

表 1 MailQueue の効果 (ball_bound)
Table 1 Effect of MailQueue(ball_bound)

	改良前	改良後	性能
ball_bound	29 FPS	33.3 FPS	15%向上

MailQueue 導入によって、27%の処理性能の向上が見られた。Task 毎から TaskList 毎に Mail の通知回数が減ったので、待ち時間が入るタイミングが減った。それによって、SPE の稼働率が向上し、処理性能の向上につながったと考えられる。

Vol.2011-ARC-195 No.32
Vol.2011-OS-117 No.32
2011/4/14

3.2 TaskArray

Task の依存関係を解決するために、SPE から Mail によって PPE へと処理が完了した Task の情報が通知される。その際に、同じ種類の Task は一つの Mail でよい場合がある。そこで、我々は Task Array を提案、実装した。Task Array は、複数の Task をまとめて扱うことが出来る。Task Array に登録した順番で依存関係を設定しているので、PPE で解決する Task の数が減り、 SPE からの TaskList 要求に応答しやすくなる。また、一度に多くの Task を TaskList に登録できるため、 SPE 側からの TaskList 要求の回数が減り、待ち時間が生じる可能性が減る。

Rendering Engine の中で、最も数が多く生成される DrawSpanTask を Task Array 化した。ボールが跳ねる例題 (ball_bound)、地球と月を表示する例題を対象に効果を測った。FPS(Frame Per Second) は、一秒間に表示できる Frame 数のことである。TaskArray は MailQueue と同様に Mail 通知に関係している。それぞれの有無の場合を計測した。

表 2 TaskArray の効果 (ball_bound)
Table 2 Effect of TaskArray(ball_bound)

	改良前	改良後	性能
ball_bound	29 FPS	34 FPS	17%向上

表 3 Task Array と MailQueue の効果 (universe)
Table 3 Effect of TaskArray and MailQueue(universe)

TaskArray	MailQueue	FPS	dma wait	mail wait
あり	あり	20 FPS	1.78%	67&
あり	なし	18.5 FPS	1.88%	69%
なし	あり	18.5 FPS	1.4%	74%
なし	なし	16.4 FPS	3.3%	84%

結果から DrawSpanTask を Task Array 化すると、FPS が上昇し、mail の wait 時間が減ったことが分かる。Rendering Engine では、 PPE 側でも処理をするべき Task があり、常に PPE が稼動している状態になっている。そのため mail を処理する時間が遅れ、SPE の mail 待ちが発生していると考えられる。 Task Array 化で Task をまとめることで SPE が一つの TaskList で多くの Task を実行できるようになったため、 TaskList を要求する回数が減って、待ち時間が発生する回数も減少した。また、それは SPE からの mail の数が減ったということなので、PPE 側の mail 処理の時間短縮になったと考えられる。

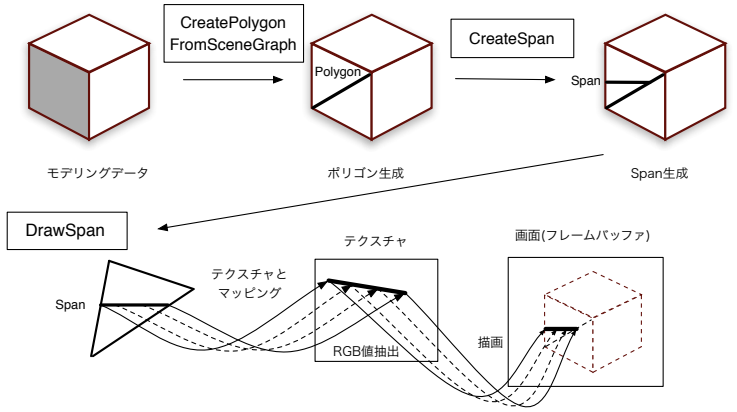

図 3 レンダリングエンジンの流れ

3.3 PipeLine 化

Cerium では RenderinEngine を実装している。RenderinEngine は大きく分けて、CreatePolygon, CreateSpan, DrawSpan, の 3 つの Task から構成されている。(**図 3**)

Span とは、ポリゴンに対するある特定 Y 座に関するデータを抜き出したものである。この 3 つの Task はそれぞれバリア同期を行いながら、順に実行される。Cerium において、バリア同期を行う場合には二つの待ち時間がある。

- SPE が他の SPE を待つ時間
- バリア同期が完了し、PPE 側で次の Task が作られる時間

この二つの時間の間 SPE の処理が止まり、処理性能の低下につながる。この待ち時間を回避するには、Task の粒度を下げる、他の SPE の処理完了を待っている SPE に、別の Task を割り当てる、等の方法がある。別の Task を割り当てるには Task の実行をパイプライン化する方法がある。

そこで、特に処理の重い DrawSpanTask と、CreatePolygon,CreateSpan の Task でパイプライン化を行った。(**図 4**)

速度比較の対象として、SuperDandy と呼ばれる、学生実験で制作されたシューティングゲームを用いた。FPS は一秒あたりの Rendering Engine 全体の処理回数 (Frame per Scecond)、busy ration は SPE の稼働率。

Vol.2011-ARC-195 No.32
Vol.2011-OS-117 No.32
2011/4/14

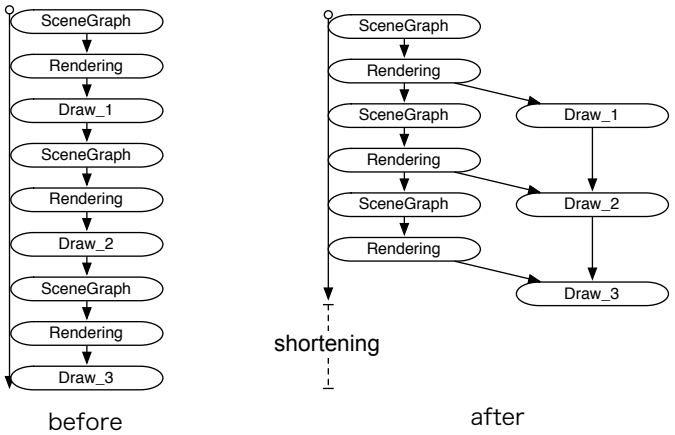

図 4 RenderingEngine のパイプライン化の様子

表 4 PipeLine 化の結果
Table 4 result of use Pipeline

	改良前	改良後	性能
FPS	29.4 FPS	49.5 FPS	68%向上
busy_ration	47.2%	78.1%	30.9&向上

パイプライン化した結果（**表 4**）、SPE の稼働率が向上し、FPS も向上した。処理性能を維持するには、SPE はなるべく稼働させ続けなければならない。その為には処理を Task に分割し、並列実行するだけでなく、バリア同期などで、SPE の待ち時間が発生することへ対処しないといけない。その対処の一つとしてそれにパイプライン化は有効である。

3.4 SPE でのキャッシュ効果

3.4.1 テクスチャの管理

Cerium ではソフトウェアレンダリングを、Task で定義し、処理している。描画の際には、SPE の LS へ必要なテクスチャの情報を読み込む。SPE のメモリ領域は 256KB しかないため、テクスチャのデータを分割し転送する必要がある。そこで、Cerium ではテクスチャを 8x8 のブロックに分割し、必要な時に沿って、ブロックを転送する方法を取った。

3.4.2 テクスチャの縮小

遠くにあるオブジェクトは小さく描画される。この場合、使用されるテクスチャは原寸大である必要がない。そこで、オリジナルのテクスチャの他に縮小したテクスチャを用意し、描画されるオブジェクトの大きさによって、使用するテクスチャを変更することにした。

テクスチャは Tile に分割しなければならないため、縦横ともに 8 の倍数を保つようにする。これまでは、テクスチャの縦横どちらかのサイズが最小 (8 ドット) になったとき、縮小画像生成を終了していたが、テクスチャのサイズが縦横で違う場合、長い方に合わせて空のデータを埋め込み 8 の倍数の正方形にするように改良した。この改良によってテクスチャの最大縮小率を正確に求めることが出来るようになった。

3.4.3 ハッシュの管理

この時に、頻繁にテクスチャを読み込む場合にはその読み込みがボトルネックとなる。そこでキャッシュを実装した。キャッシュは MemorySegment と呼ばれるデータ構造単位でハッシュで管理する。MemorySegment はある一定の大きさに決め打った連続したメモリ領域のことである。キャッシュ実装の効果を示す。速度比較の対象には、使用するテクスチャ表示範囲が狭い ball_bound と、画面すべてにテクスチャを表示する panel を用いる。

表 5 1 秒辺りの Rendering Engine 全体の処理回数
Table 5 Frame per Second

	改良前	改良後	性能
ball_boud	4 FPS	30 FPS	7.5 倍向上
panel	0.2 FPS	2.6 FPS	13 倍向上

テクスチャような頻繁な転送が起こり得る場合には、キャッシュが非常に有効であることがわかった。Cell のような分散メモリでは、データの転送は慎重に管理しできるだけ転送回数を減らすことが性能向上につながる。

3.5 Memory Access

Cell において、SPE へのデータの割り振り方が性能に関わる場合がある。それはデータを得るために、DMA 転送が頻繁に起きるときである。Cerium を用いて実装した WordCount を例にとってみる。WordCount の Task は二つある。

(1) WordCountTask
(2) PrintTask

WordCountTask は、input で与えられた data を word count し、output data に書き出

Vol.2011-ARC-195 No.32
Vol.2011-OS-117 No.32
2011/4/14

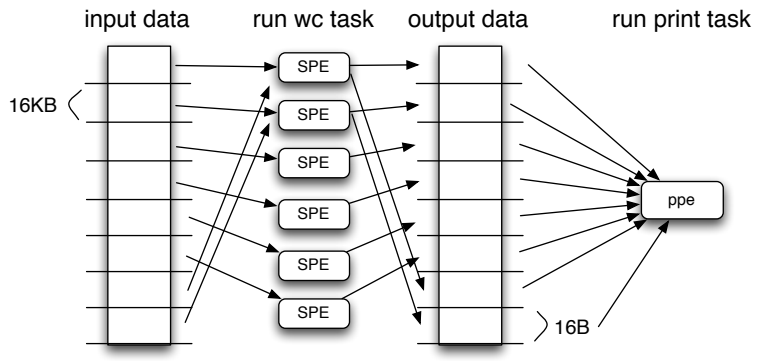

図 5 WordCount の Task の流れ

す Task である。PrintTask はすべての WordCountTask の実行完了を待ち、output へ書き出された値を集計し出力する Task である。一度に SPE に渡せるデータ容量は DMA の仕様上 16Kbyte までである。さらに転送する際には 16 の倍数 byte である必要がある。

wc する file をメモリへマッピングし、WordCountTask の input に、file data のアドレスを 16kbyte ごとに指定していく（**図 5**）。

PrintTask は WordCountTask を待ち Task と設定し、WordCount がすべて終わらないと、実行されない。

WordCount には TaskArray を用いた。TaskArray を用いる場合に注意すべき点がある。TaskArray を使わず、Task を用いる場合、Task は生成された順番にそって各 SPE に割り振られていく。ファイルを mmap し、その領域を上から順番に Task に設定しているので各 SPE は同時に近くのメモリ領域にアクセスすることになる。TaskArray を用いた場合に生成された順に TaskArray に設定するのはまずい。TaskArray 一つは、一つの SPE のみで実行され、分割されない。TaskArray に連続した領域を指す Task ばかりを格納すると、それによって、一つの SPE が連続領域を順番にアクセスして行く形になる。別の SPE は、16kB x TaskArray のサイズ分離れた領域からメモリアクセスを開始するので、離れたメモリ領域を各コアが同時にアクセスすることになる。なので、先に複数の TaskArray を用意し、Task が生成された順番にそって各 TaskArray に割り振るように改良した。

TaskArray のサイズは 64, Task の input データの大きさは 16KB, wc 対象のデータの大きさは約 135MB。Task の数は約 8000。dma wait は処理全体にかかった時間の dma 転送待ちの割合。改良前は各 SPE が同時に離れた領域 (各 SPE 間は 16KB x 64 離れる) にアクセスする。改良後は近くのメモリ領域 (各 SPE 間はだいたい 16KB 離れる) にアクセスする。

表 6 メモリアクセスの局所性を保った改良

	改良前	改良後	性能
実行時間	30s	1.9s	15 倍向上
dma wait	14%	8%	1.7 倍向上

メモリアクセスの局所性を保った場合に処理性能の向上が見られた（**表 6**）。ページングや、スワッピングを抑えることができたと考えられる。それに伴って dma wait 時間も減少し、SPE の待ち時間が処理性能の向上に繋がっていると考える。

3.6 OpenGL との比較

OpenGL (Open Graphics Library) とは、Silicon Graphics 社が開発した、3D グラフィックス処理のためのプログラミングインターフェースである。上記で紹介した SuperDandy を Task を用いない OpneGL 上で動作するバージョンを用意して、Cerium と性能を比較してみた。OpenGL は PPE だけで動作している。Cerium は今までの改良をすべて加えたもの。

表 7 シューティングゲーム「dandy」の性能比較 (OpenGL, Cerium))

	OpenGL	Cerium	性能差
dandy	17.5 FPS	49.5 FPS	2.9 倍

コアを 1 つ用いている OpenGL 版に比べて、Cerium では 2.9 倍の性能向上が見られた。SPE を活用し、改良によって待ち時間の削減ができ、性能の向上ができた。

4. debug

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結果が異なる) な場合があり、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があり、デバッガが複数の Core を取り扱えることが必須である。Cell の場合、動作している複数 の SPE の一つに対して gdb で breakpoint を掛ければ、PPE や他の SPE も同時にストップするが、それら全ての CPU を手動で管理するのは厳しい。また、PPE と SPE ではメモリ空間が違うため、SPE から直接 PPE

Vol.2011-ARC-195 No.32
Vol.2011-OS-117 No.32
2011/4/14

のデータを見ることができない。Cerium での開発工程は、

(1) C によるシーケンシャルな実装
(2) 並列実行を考慮したデータ構造を持つ実装
(3) コードを分割し、シーケンシャルに実行する実装
(4) 分割したコードを並列実行する実装

となっている。逐次的な実行で正常な動作を確認した後、並列に実行した場合に正常な動作をしない場合がある。特に Cell 特有の問題としてデータ構造が合っていない。つまり、DMA 転送される際に、16 アライメントに設定されていないか、データのサイズが 16 の倍数になっていない場合に注意しなければならない。また Cerium では PPE 用と SPE 用が別個に存在するので、互いのコードが等価であるかもチェックしなければならない。一つのコードに統一しても良いが、別個で対応したい問題がでた時に対応してる。なるべく同一なコードにするのがよい。本来 SPE で動くべき Task が PPE で動作するケースもあるので、それもチェックするべき。

5. ま と め

本研究では ゲームフレーム Cerium TaskManager の改良を行った。特に Cell 上での実行の場合、SPE の活用が処理性能に大きく関わる。SPE から PPE への Mail 通知には、PPE の Mail を確認するタイミングが関わってくるので、MailQueue, TaskArray を用いて SPE 側でなるべく通知にタイミングを減らし、待ち時間を減らした。SPE の稼働率を上げることで性能向上につながった。またキャッシュを用い、テクスチャなど頻繁に利用するデータを SPE の LS に常駐させることで、データ転送の回数を減らし待ち時間を削減した。数種の Task が混在し、バリア同期を行っている場合には、SPE の待ち時間が発生するので、Task のパイプライン化によって解決した。またデータアクセスの局所性を保つことでデータ転送の待ち時間を減少させることができる。Cerium は上記の改良を加え、改良前に比べ、約 7 倍の処理速度の向上が見られた。

Core の待ち時間を減らすことは、Core の稼働率の向上につながり処理性能が向上する。各 Core の待ち時間は並列プログラミングにおいて、特に注意しなければならない。それは、Amdahl の法則からも言える。

並列実行には Amdahl の法則 ([3]) があり、使用する CPU を増やしても、元のプログラムの並列化率が低ければ、その性能を活かすことができないとされている。例えば、プログラムの 8 割を並列化したとしても、6CPU で 3 倍程度の性能向上しか得られない (**図 6**)

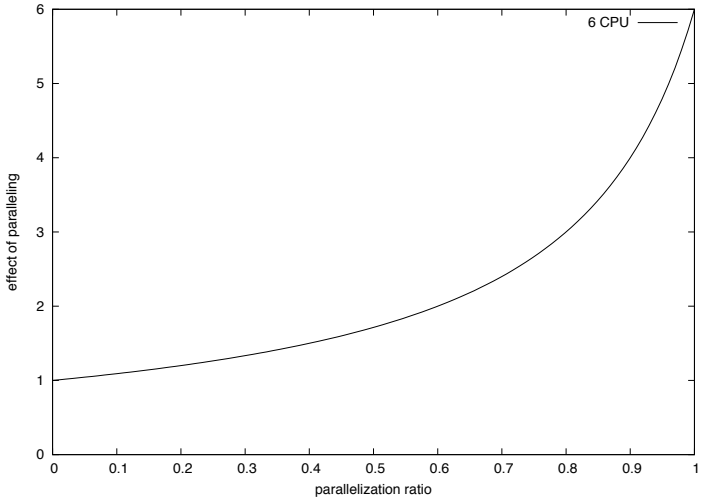

図 6 Amdahl の法則

このことから、恒常的に並列度を維持する必要があることが分かる。

6. 今後の課題

6.1 依存関係の設定

現在 Task のパイプラインは Task の依存関係をユーザが明示的に記述している。Task の数が増えるとプログラミングの難易度が格段に上がってしまうという問題がある。また、パイプライン化できる場所を特定することも難しくなるこの問題は Task の依存関係をユーザではなく、システム側が記述するようにすることで解決できる。 Task の依存関係が、処理するデータの依存関係と直結しているので、データの依存関係をシステム側で監視し、その依存関係を元に処理を行うことでシステム側からの依存関係の記述が実現できる。もしくは、Task の依存関係は別の言語で記述し、TaskManager がその記述に沿って、定義された Task の実行する方法も考えられる。また、TaskArray も Task とデータの依存関係から、自動で作成できると考える。

6.2 Code Load

SPE は LS 以外のメモリに直接アクセスすることができず、自身の LS の容量は 256KB

Vol.2011-ARC-195 No.32
Vol.2011-OS-117 No.32
2011/4/14

である。現在 Cerium では、プログラムを実行する前に、SPE で処理する Task のコードは事前に SPE に転送している。しかし、Cerium の開発を続けていく上で、ついにコンパイルの最適化を行っていない状態のコードを SPE 上に置いておくことができなくなってしまった。この問題から、現在 Cerium が SPE 上にデータを転送している様にして、SPE 上に必要のないコードの入れ替えを行う必要が出てきた。そこで、我々は SPE と PPE の間でやりとりするデータのすべてを Segment というデータ構造で行うことを提案する。その先駆けとして、現在は描画を行う Task に MemorySegment という形でデータの入れ替えのためのデータ構造を用いている。

6.2.1 Task の粒度

Task の粒度が大きい場合に、SPE 間で、Task の消化時間にバラつきがでやすい。それは特にバリア同期をしている場合に、待ち時間へ直結する。Task を粒度が小さい場合には PPE との通信時間がかさむ事になる。粒度が大きい場合には、他の Task を投入することで解決できる。その方法がパイプラインである。実は粒度が小さい場合にも次の Task を予想して dma 転送などパイプライン化し転送待ち時間を解消する対処法がある。しかし、Cerium のように PPE への Mail 通知で同期ポイントがある場合にパイプライン化では対処できなかった。そこで今回、MailQueue,TaskArray など同期するタイミングの回数を減らすことで、ある程度の改良を行った。TaskArray は Task の粒度が大きくなる影響があり、また巨大なデータを分割アクセスする場合にはメモリアクセスの局所性に注意しながらの Task の配分が必要である。キャッシュ、Task 配分によるメモリアクセス、Task スケジューリングのパイプライン化による必要なデータの先読みなど共通している項目はメモリの扱いに関係するものである。このさまざまなデータ構造を統一し管理しやすくするために、Task も含め、すべてのデータの扱いは MemorySegment で統一したほうが良いと考える。これらの一度に扱うデータのサイズ、Task の粒度の大きさなど、最適な大きさはわかっていない。なるべく非同期に Taks を設定し、パイプライン化を行うにあたって、それらの項目も考慮していく。

参 考 文 献

1) 多賀野海人. Cell Task Manager Cerium における Task を用いたパイプラインの改良. 琉球大学理工学研究科情報工学専攻 平成 22 年度学位論文, Feb 2011.
2) 宮國渡. Cell 用の Fine-Grain Task Manager の実装. 琉球大学理工学研究科情報工学専攻 平成 20 年度学位論文, Feb 2009.
3) Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, and Doug Lea. *Java Concurrency in Prac-tice*. Addison-Wesley Professional, 2005.
4) Sony Corporation. Cell BroadbandEngine ™ アーキテクチャ, 2006.
5) International Business Machines Corporation, Sony Computer Entertainment Incorporated, Toshiba Corporation. *SPE Runtime Management Library Version 2.3*, 2008.
6) International Business Machines Corporation, Sony Computer Entertainment Incorporated, Toshiba Corporation. *C/C++ Language Extensions for Cell Broadband Engine Architecture Version 2.6*, 2008.
7) Keisuke Inoue. SPU Centric Execution Model, 2006.
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. 琉球大学工学部情報工学科 平成 19 年度卒業論文, 2008.
9) blender.org. http://blender.org/.
10) 赤嶺一樹, 河野真治. Meta Engine を用いた Federated Linda の実験. 日本ソフトウェア科学会第 27 会大会 (2010 年度), Sep 2010.
11) 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.
12) International Business Machines Corporation, Sony Computer Entertainment Incorporated, Toshiba Corporation. *Cell Broadband Engine Linux Reference Implementation Application Binary Interface Specification*,
13) SourceForge.JP: Cerium Rendering Engine. https//sourceforge.jp/projects/cerium/. bibitemosmesa The Mesa 3D Graphics Library. http://www.mesa3d.org/.
14) Fedora Project. http://fedoraproject.org/.
15) Yellow Dog Linux for PowerPC Computers. http://us.fixstars.com/products/ydl/.
16) SourceForge.JP: Cerium Rendering Engine. https//sourceforge.jp/projects/cerium/.
17) International Business Machines Corporation. *Software Development Kit for Multicore Acceleration Version 3.1*, 2008.
18) Wataru MIYAGUNI. 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.
19) Simple DirectMedia Layer. http://www.libsdl.org/.
20) Aaftab Munshi, Khronos OpenCL Working Group. *The OpenCL Specification Version 1.0*, 2007.
21) Gallium3D. http://www.tungstengraphics.com/wiki/index.php/Gallium3D.