マルチコアプロセッサのプログラミングに求められるもの

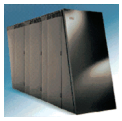
- スレッドプログラミング技術を中心として -

2007年2月23日
日本アイ・ビー・エム(株)東京基礎研究所
阪本　正治

講演の概略と目次

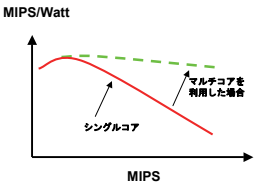
- 概略
  - IBM は長年データプロセッシングの領域で最先端の技術を研究してまいりました。 最新の動向としては、マルチコアプロセッサー、特にヘテロジーニアスなマルチコアに対するプログラミング技術が最先端として注目されています。IBMは、ソニー株式会社様、株式会社ソニー・コンピュータエンタテインメント様および東芝様と共に、Cell Broadband Engine™と呼ばれる次世代のマイクロプロセッサーを開発いたしました。それは、次世代のメディアプロセッシングで要求される膨大な計算処理を、複数の演算コアの同時並行動作によって可能にします。しかしながら、アプリケーション・プログラムからマルチコアプロセッサーの能力を活用するためには、複数のコアを有効活用した並列処理や、データの入出力を考慮に入れたプログラミングが必須となります。本講演では、マルチコアプロセッサの能力を遺憾なく発揮させるためのプログラミング戦略について説明いたします。
- Agenda
  - はじめに
  - Cell Broadband Engine HW概説
  - 並列処理の基礎
  - Cell Broadband Engineでのマルチスレッドプログラミング
  - まとめ

スケールアップからスケールアウトへ



Blue Gene

- **スケールアップ**
  - CPUやサーバー単体の性能を強化する
- **スケールアウト**
  - CPUのコア数やサーバーの台数を増やして性能を上げる

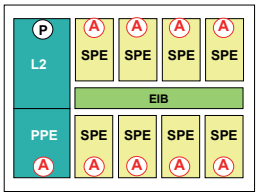


Cell Broadband Engine

Blue GeneはIBM Corporationの商標。

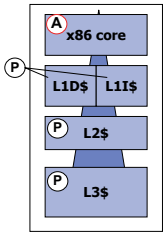
従来型プロセッサーとの比較

Cell Broadband Engine



Ⓐ Active　Ⓟ Passive

複数のアクティブな「頭」を持つ疎結合（Decoupled)アーキテクチャー



一つのアクティブな「頭」を持つ従来型アーキテクチャー

Cell Broadband Engine ハードウェア概説

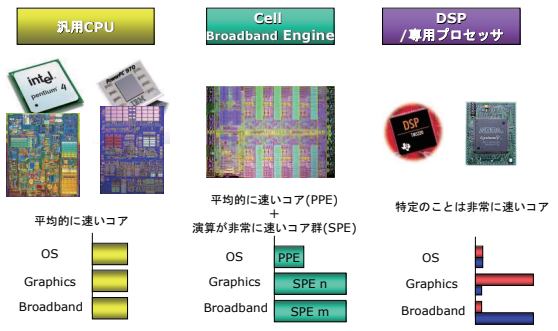
マルチコアプロセッサー

- **ホモジーニアス型**
  - 同じコアが複数搭載されたマルチコアプロセッサー
- **ヘテロジーニアス型**
  - 異なるコアが複数搭載されたマルチコアプロセッサー
    - Cell Broadband Engineは、1個の64ビットのPowerプロセッサー・コアと複数のSynergetic Processor Elementが搭載されている

PowerPCは、IBM Corporationの商標です。

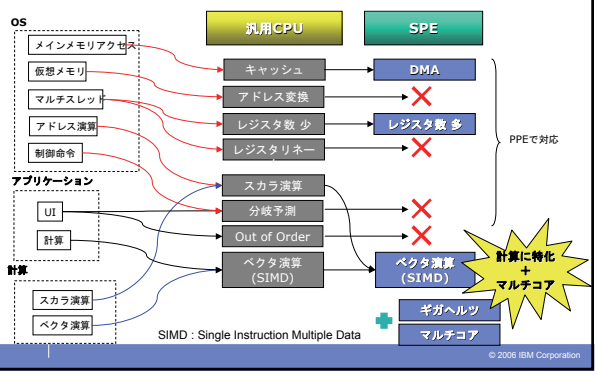
SPEのアーキテクチャー

Cell Broadband Engine の目指すところ

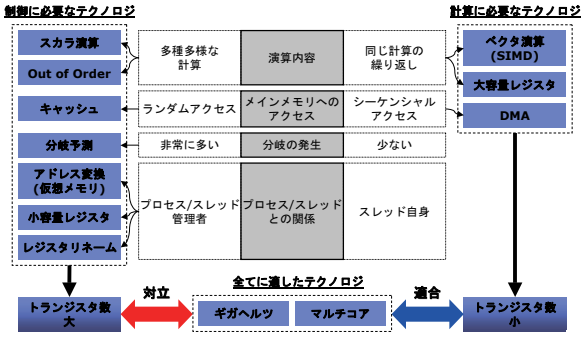


SPEのアーキテクチャー

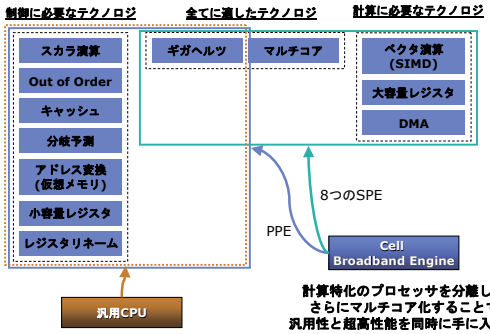
SPEのコンセプト



制御と計算に必要なテクノロジ

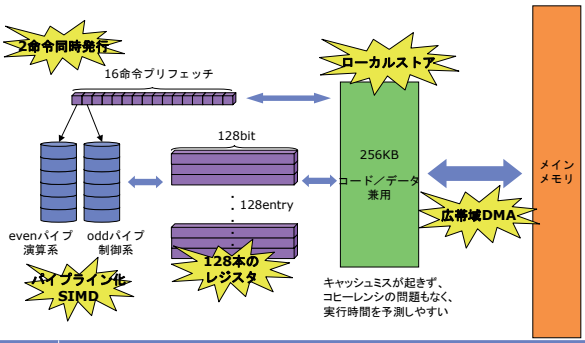


制御も計算も高速であるためには

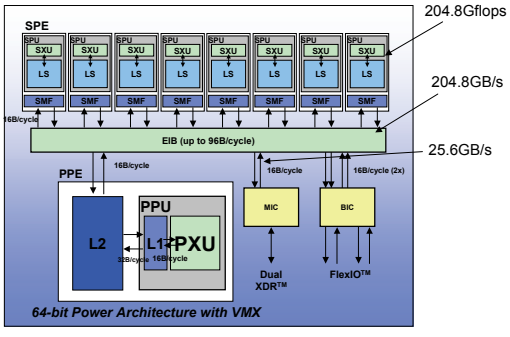


SPEのアーキテクチャー

SPEの特徴



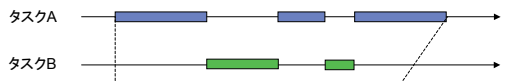
Cell Broadband Engineの内部構造



並列処理の基礎

並行(Concurrent) と並列(Parallel)

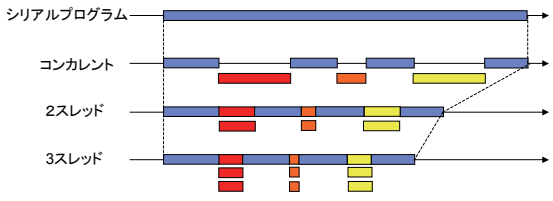
- **並行(Concurrent)**



- **並列(Parallel)**

マルチスレッドプログラミング

- **シリアルな処理を行う通常のプログラムからコンカレントな処理を見つけ出し、スレッド化し、各コアに割付ける。**

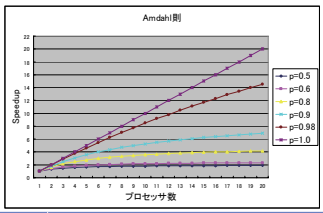


マルチスレッドプログラミングの利点と問題点

- **利点**
  - ターンアラウンド時間の短縮
    - 処理要求を出してから処理完了までの所要時間 -> 科学技術計算
  - スループットの向上
    - 単位時間当たりの処理能力の向上 -> 組み込みシステム
  - 複雑な機能の実現
    - 異なる機能を異なるスレッドで処理する
- **問題点**
  - 複雑さが増す
  - デバッグの困難さ(race condition or deadlock)

Amdahl則

- **複数のプロセッサを使ったときの理論上の性能向上の度合い 。**
- **1/(1-p)が並列化による速度向上の限界である。P=0.8なら5倍が限界。**



$$T_{total}(1) = T_{serial} + T_{parallel}$$

$$T_{total}(N) = T_{serial} + \frac{T_{parallel}}{N}$$

$$speedup = \frac{T_{total}(1)}{T_{total}(N)}$$

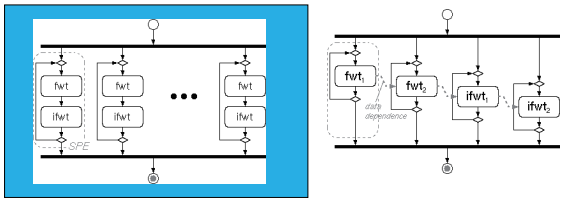
$$p = \frac{T_{parallel}}{T_{total}(1)}$$

$$speedup = \frac{T_{total}(1)}{T_{total}(N)} = \frac{1}{1 - p + \frac{p}{N}}$$

$$E(N) = \frac{speedup}{N}$$

データ並列とタスク並列

- **データ並列**
  - 独立したデータセットに対して並列に処理を実行する
- **タスク並列**
  - 独立したサブプログラムを並列に実行する



そのほかの検討事項

- 粒度
  - 並列化するタスクの粒度は、大きい方が有利。
    - ループでの並列化の場合には、最外側のループを並列化できると有利
    - 負荷の不均衡に注意
- 同期
  - 排他制御でデータにロックされている場合や、次の処理に移る前に、他のスレッドの終了を待つ必要が生じる。データを共有せず、同期を減らすのが得策。
- **Race condition**
  - 共有リソースであるデータへのアクセス順序によって、計算結果が変わることがある。このような状態をRace Condition (競合状態)と呼ぶ。
- Deadlock
  - 複数のスレッドが互いの処理終了を待ち、結果としてプログラムが停止してしまうこと。

デバッグが困難なことが多い

Cell Broadband Engineでのマルチスレッドプログラミング

性能を引き出すポイント

- **コアの演算特性を意識したタスク分割。**
  - 一般的には、PPEは処理全体を制御する役割を担い、SPEが計算処理を担う。
- **SIMDに適したデータ配置**
- **分岐ミスの回避**
- **演算とDMAのダブルバッファリング**
- **SPEのプログラミングモデル**

SPEプログラミング → 性能を引き出すポイント

PPEとSPEの役割分担

PPEに適したプログラム		SPEに適したプログラム		SPEを効果的に利用するためには
多種多様な計算	演算内容	同じ計算の繰り返し	➡	同じ計算を繰り返すプログラムをSIMDで実装する
ランダムアクセス	メインメモリへのアクセス	シーケンシャルアクセス	➡	連続領域のDMA転送
非常に多い	分岐の発生	少ない	➡	なるべく分岐をつくらない
プロセス/スレッド管理者 管理者は1つだけ	プロセス/スレッドとの関係	スレッド自身 複数スレッド可能	➡	1SPE 1スレッドとし、なるべくコードを再ロードしない

SPEプログラミング → 性能を引き出すポイント

SIMD演算とvector型

vector signed char
vector unsigned char

char 0	char 1	char 2	char 3	char 4	char 5	char 6	char 7	char 8	char 9	char 10	char 11	char 12	char 13	char 14	char 15

vector signed short
vector unsigned short

halfword 0	halfword 1	halfword 2	halfword 3	halfword 4	halfword 5	halfword 6	halfword 7

vector float
vector signed int
vector unsigned int

word 0	word 1	word 2	word 3

vector double
vector signed long long
vector unsigned long long

doubleword 0	doubleword 1

qword

quadword

msb ←――――――――――――――――――→ lsb

SPEプログラミング → 性能を引き出すポイント

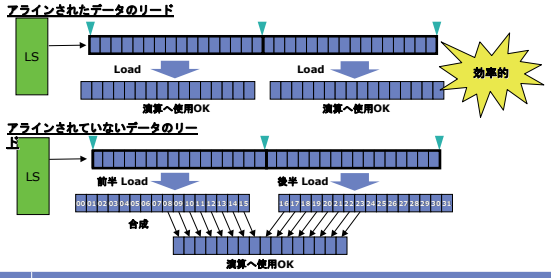
vector型

- **vector型に演算子による加減乗除(+,-,*,/,%など)は行えない**
- **使用できる演算子は以下の通り**
  - sizeof() ・・・ 常に16を返す
  - = (代入) ・・・ 同一のベクタ型である場合のみ有効
  - & (アドレス) ・・・ 有効
  - ポインタに対する + ・・・ 次のベクタを指す(16byte進む)
  - ポインタに対する * ・・・ ポインタの下位4bitをマスクしたアドレスからのロード/ストア。
    ※アラインされていないロード/ストアは組み込み関数で実現
  - キャスト ・・・ ポインタ同士はベクタ⇔ベクタ、ベクタ⇔スカラ、どちらも可能 ※16byteアラインに注意

SPEプログラミング → 性能を引き出すポイント

SIMDに適したデータ配置

- 16byteアラインメントを注意深く守る
  - 16byte単位のLoad/Store
  - 16byteアラインされたアドレス

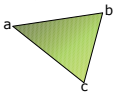


SPEプログラミング → 性能を引き出すポイント

SIMDに適したデータ配置

- データ配置もSIMDに最適化する
  - 16byte単位のLoad/Store
  - 16byteアラインされたアドレス

例) 三角形ポリゴンデータの表現と座標変換



SIMDの4つのデータに入れるのが自然だが. . .

a.x	a.y	a.z	1
b.x	b.y	b.z	1
c.x	c.y	c.z	1

Array of Structure (AOS)

計算を考えると. . .

$$\begin{pmatrix} ax+by+cz+d \\ ex+fy+gz+h \\ ix+jy+kz+l \\ mx+ny+oz+p \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

a[0].x	a[1].x	a[2].x	a[3].x
a[0].y	a[1].y	a[2].y	a[3].y
a[0].z	a[1].z	a[2].z	a[3].z
b[0].x	b[1].x	b[2].x	b[3].x
b[0].y	b[1].y	b[2].y	b[3].y

Structure of Array (SOA)

a	e	i	m
b	f	j	

効率的

SPEプログラミング → 性能を引き出すポイント

分岐ミスの回避

- 分岐そのものをなくす
  - 比較命令と選択命令で条件付代入を実現する

分岐を生成する通常の命令の代わりに...

```
if ( a > b ) x = a*2;
else         x = b*2;
```

1. 比較してマスクを生成

a0 a1
b0 b1
Compare
1111 0000

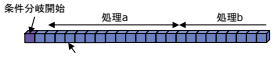
2. 選択命令で代入

a0 a1
b0 b1
1111 0000
Select
1の時はaを代入、
0の時はbを代入
x0 x1

分岐が消えた

- 分岐ヒント命令を使用する
  - gccの__builtin_expectを使用する
  - hbr, hbra命令

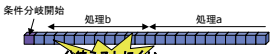
```
if ( cond )  処理a;
else         処理b;
```



分岐結果がわかるころには実行が進んでいる

__builtin_expect : 分岐しやすい方を指示する

```
if ( __builtin_expect(cond, false) )
   処理a;
else
   処理b; // 分岐のペナルティがない
```

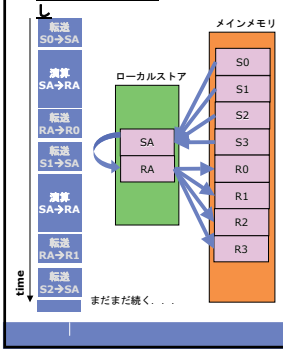


分岐ミスしにくいコードになる

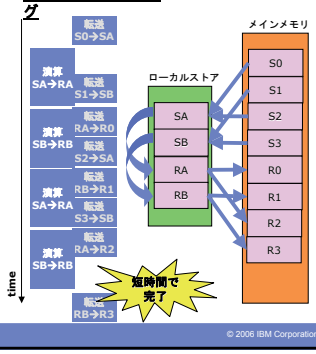
SPEプログラミング → 性能を引き出すポイント

演算とDMAのダブルバッファリング

バッファリングなし



ダブルバッファリング



短時間で完了

SPEプログラミング → 性能を引き出すポイント

提案されているプログラミングモデル

- **Cell Broadband Engineの柔軟なアーキテクチャは様々なアプリケーションの実現方法を可能にする**
  - 下記は公開資料中で提案されているモデル
  - SDKにサンプルがあるものもある

PPEとSPEの連携に関するモデル
- Function Offload モデル
- Asymmetric Thread Runtime モデル
- Streaming モデル

マルチSPEに関するモデル
- Computation-Acceleration モデル
- Asymmetric Thread Runtime モデル
- Pipeline モデル

SPEの起動に関するモデル
- SPU thread モデル
- SPUlet モデル

その他のモデル
- Device Extension モデル
- SPU Plugins モデル

SPE同士の連携に関するモデル
- Computation-Accelerationモデル
- Pipeline モデル
- Shared-Memory Multi-processorモデル

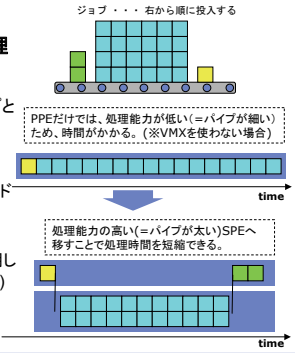
SPEでのマルチタスクに関するモデル
- Multi-thread モデル
- Run-to-completion モデル
- Job Queue モデル
- User-Mode Thread モデル

※分類は一例。複数のカテゴリーに当てはまるモデルも多い

SPEプログラミング → 性能を引き出すポイント

Function Offload モデル

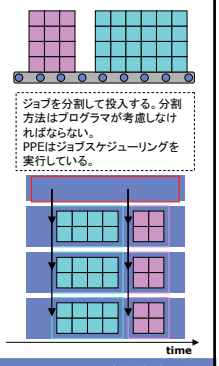
- **時間のかかる処理をSPEで処理**
  - 簡単な変更で効果を実感しやすい
  - 他のモデルへ展開する前のステップとするのもよい
- **プログラミング方法**
  - SDKライブラリを用いて、SPEスレッド作成、終了待機関数により実装する
  - SDKに付属するIDL (Interface Definition Language) ツールを使用して、RPC (Remote Procedure Call) 呼び出しとして実装する方法もある



SPEプログラミング → 性能を引き出すポイント

Computation-Acceleration モデル

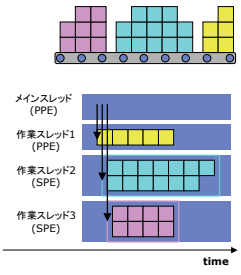
- **複数SPEにジョブを分割して投入して時間短縮をめざす**
  - データの相互関係がある場合はSPE間でメッセージ交換
- **配慮すべき点**
  - DMAをうまくスケジュールしないと、処理すべきデータが投入されず、待たされる。
- **プログラミング方法**
  - SDKのスレッド関数を利用する
  - パラメータのやりとりにはDMAやMailboxなどを利用する



SPEプログラミング → 性能を引き出すポイント

Asymmetric Thread Runtime モデル

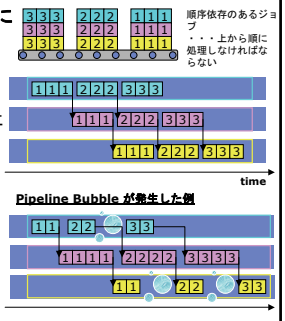
- **PPEとSPEを使ってマルチスレッド**
  - Function Offloadのように終了を待機せずにスレッドが並列動作する
- **配慮すべき点**
  - PPEとSPEの性格の違いに配慮した作業分担
  - 1SPE 1コンテキストの原則
- **プログラミング方法**
  - SDKのスレッド関数を用いて実装
  - 待機関数の代わりに、通信と割り込みでスレッドの終了を検知する



SPEプログラミング → 性能を引き出すポイント

Pipeline モデル

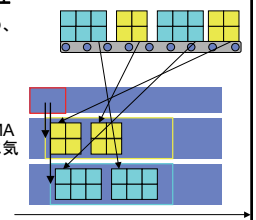
- **単一ジョブを複数の処理ステージに分割し、SPEに振り分ける**
  - データ分割で対応できない場合や瞬間スループットが重要なときに有効
  - 処理を同程度の処理時間のステージに分割できなければいけない
  - 何らかのSPE間通信は必須
- **配慮すべき点**
  - 各ステージの処理時間 を揃える
- **プログラミング方法**
  - SPEプログラムを適切な粒度に分割
  - MailboxとLS間DMA転送を用いる



SPEプログラミング → 性能を引き出すポイント

Streaming モデル

- **SPEが自立的にジョブを取り出して処理**
  - スレッド切り替えもPPEとの通信もないため、SPEがもっとも効率的に機能する
  - 動画や音声の再生などに適する
- **配慮すべきポイント**
  - SPUとLSの帯域は、LSとメインメモリのDMA転送よりも1桁大きいため、データの充填に気をつける
- **プログラミング方法**
  - SPEスレッド作成
  - メインメモリに共有ジョブキューや、メッセージ交換の仕組みなどを追加する
  - 起動後はSPEが能動的にジョブキューを調べて、自律的にデータを処理



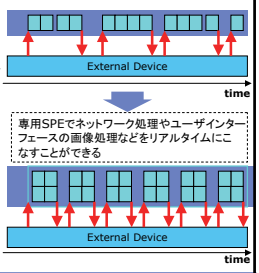
PPEはスレッドを作成した後は何もしなくてよい。SPEが自律的にジョブを取り出しては処理していく。PPEは別のことをしていてもよい

SPEプログラミング → 性能を引き出すポイント

Device Extension モデル

- **SPEが直接I/Oとやりとりして高速に応答**
  - ユーザインターフェースやネットワークの高速処理に適したモデル
  - SPEがデバイス処理に特化でき、リアルタイム性を要求されることが多いデバイスとのやりとりに有利
- **プログラミング方法**
  - 実装上はFunction Offloadモデルの特別の場合といえる
  - I/Oとは通常のDMA転送でやりとりする(メモリマップトI/O)

処理速度の点だけでなく、OSやアプリケーションなどを同時に扱わなければならないのでリアルタイム性の確保が難しい



専用SPEでネットワーク処理やユーザインターフェースの画像処理などをリアルタイムにこなすことができる

SPEプログラミング → 性能を引き出すポイント

プログラミングモデルの選択

処理時間を短縮するには

とにかく短縮できればよい	Function Offload モデル	• PPEを単一SPEへオフロード • SPEが終わるまでPPEは待機	易しい
平均スループットが重要	Computation -Acceleration モデル	• 複数のSPEへ処理をオフロードする • データまたは処理で分割する	
瞬間スループットが重要	Pipeline モデル	• 一連の処理を複数ステージに分割 • ステージの粒度を揃える	難しい
I/O処理を素早くしたい	Device Extension モデル	• SPEが直接I/Oデバイスからデータを取り込み処理して出力する	

大量のデータを処理するには

異なる種類のデータ生成済みの大量データ	Computation -Acceleration モデル	• SPEを複数使って処理 • SPEをグループに分けてもよい	一般的
同種のデータだ連続して生成される	Streaming モデル	• PPEを介することなく、SPEがデータを取り込んで処理する	効率的

Cell Broadband Engineアプリケーションの例

コーンビーム3次元X線CTの画像再構成

- ボクセル I(x,y,z) は、投影角度 β における投影ピクセル にフィルタ操作を施したP*(u,v, β)に、重み w(x,y,β)を掛け合わせたものの全角度にわたっての総和となる。

$$P'(u,v,\beta) = w * P(u,v,\beta) \quad .....(1)$$

$$w = \frac{D_{so}}{\sqrt{D_{si}^2 + u^2 + v^2}} \quad .....(2)$$

$$P^*(u,v,\beta) = P'(u,v,\beta) \otimes h(u) \quad .....(3)$$

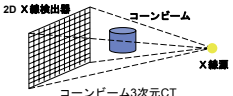
$$I[x,y,z] = \sum_{\beta} P^*[u(x,y,\beta), v(x,y,\beta), \beta] \cdot W(x,y,\beta) \quad ....(4)$$

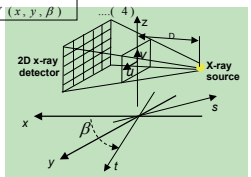
$$\begin{cases} s = x\cos\beta + y\sin\beta \\ t = -x\sin\beta + y\cos\beta \end{cases} \quad ....(5)$$

$$W = \left(\frac{D_{so}}{D_{so} - s}\right)^2 \quad ....(6)$$

$$u = \frac{t * D_{si}}{D_{so} - s} \quad ....(7)$$

$$v = \frac{z * D_{si}}{D_{so} - s} \quad ....(8)$$





M枚の投影画像から1辺の解像度がNの立方体を再構成する場合

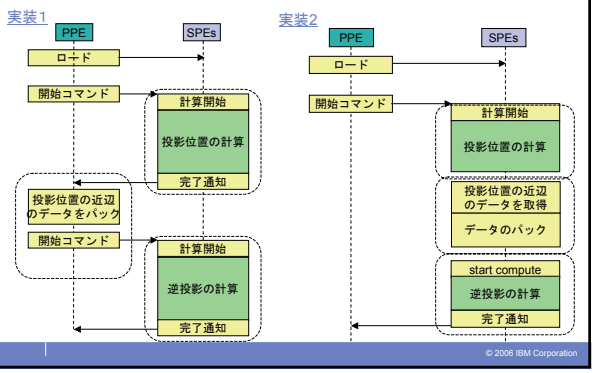
- 演算回数
  - 重み付け:MN^2
  - フィルタリング: $MN^2(2+ 10\log_2(2N))$
  - 逆投影:$17MN^3$
- 投影画像のデータ量
  - 各ピクセルを4バイトfloat型で表した場合、$4MN^2$バイト
- ボリューム画像のデータ量
  - 各ピクセルを4バイトfloat型で表した場合、$4N^3$バイト

画像再構成の実行結果 (Pentium® M 1.8GHz)

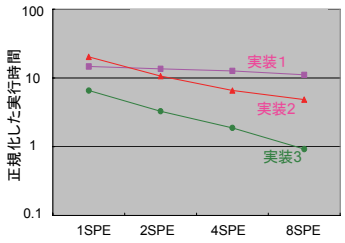
処理内容	処理全体に占める割合 [%]
重み付け	0.2
フィルタリング	1.0
逆投影	98.8

PentiumはIntel Corporationの商標。

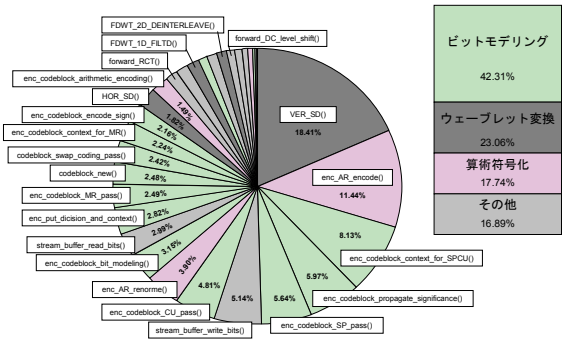
逆投影の実装



実行時間

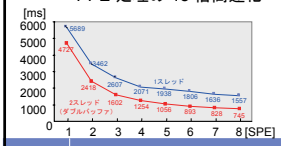


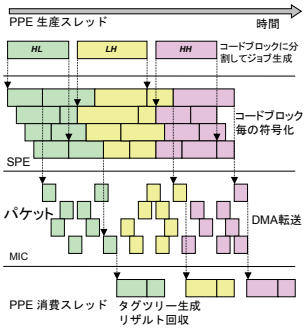
Motion JPEG 2000のプロファイリングの例



ビットモデリング・算術符号化 (EBCOT)への応用

- **実装**
  - コードブロック単位でジョブ生成
    - 8 SPEで並列符号化
- **最適化**
  - ダブルバッファリング
    - DMA転送隠蔽
  - PPE 2スレッド構成
    - タグツリー生成時間隠蔽
- **効果**
  - PPE 処理の 18 倍高速化

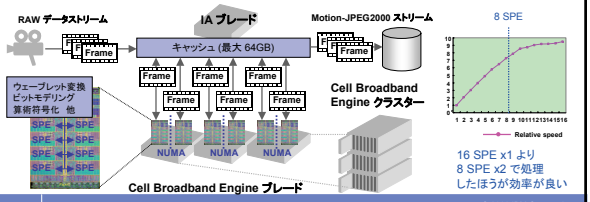




マルチエンコーダによる分散処理

- **1枚の IA ブレードと複数の Cell Broadband Engine ブレードで構成**
- **各 Cell Broadband Engine ブレード上で、NUMA により並列に2ノードの符号化サーバーを実行**

Cell broadband Engine を2個搭載したブレードサーバー



16 SPE x1 より
8 SPE x2 で処理
したほうが効率が良い

ありがとうございました。