GPUによる MP3エンコードに関する研究

高性能コンピューティング学講座
M2
史　晨悦　（シ　シンエツ）
主任指導教員 本多弘樹

目次

- 研究概要
- 研究背景
- 関連研究
- 提案方法
  - 方法1
  - 方法2
- まとめ
- 参考文献

研究概要

- GPGPU開発環境CUDAを用い、
  非圧縮、リニアPCMのサンプリングデータからMP3（MPEG Audio Layer-3）規格までの静的エンコーディング並列高速化方法を研究する。

研究概要

- 並列化する対象は量子化繰り返し部分（iteration loop）

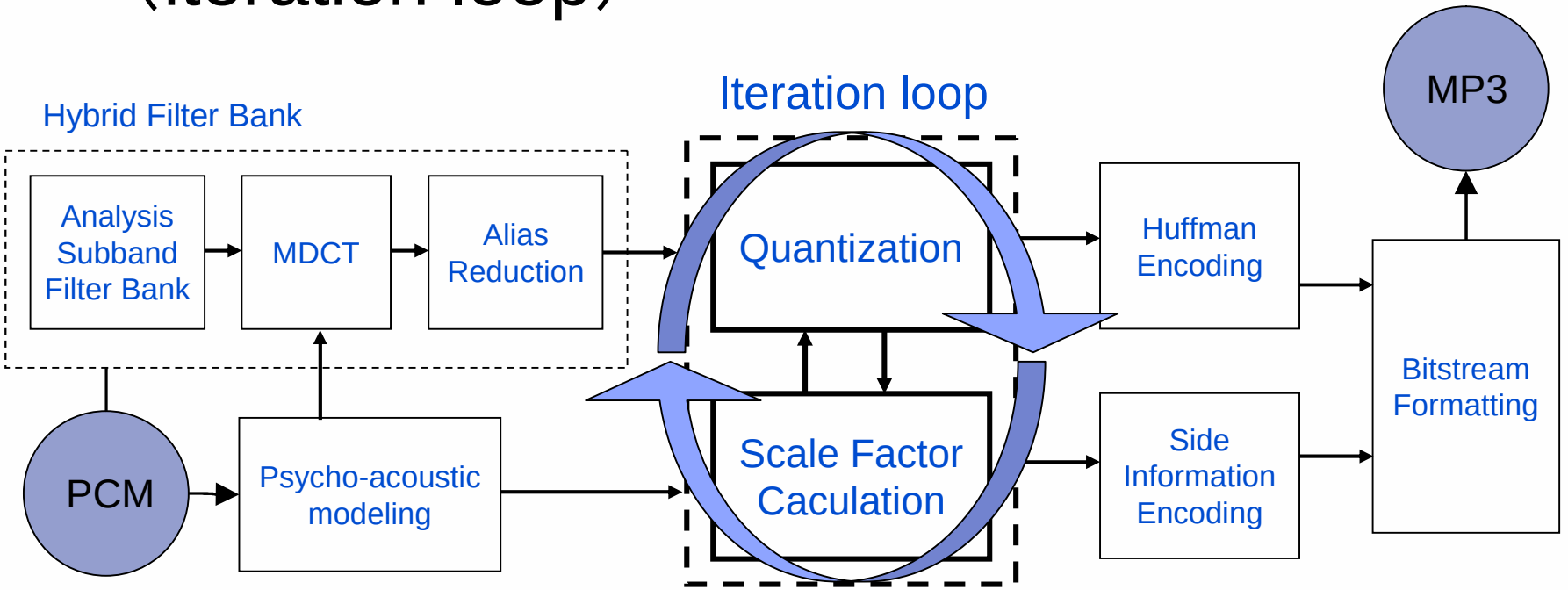


図　MP3エンコードのプロセス

研究背景

- CUDA
  - もっとも注目されているGPGPU開発環境
  - C言語に基づくプログラミング言語
- マルチメディアのエンコード
  - 映像、音声のエンコード処理は計算時間が掛かり、高速化の要求が高い

関連研究

- CUDAによるマルチメディアのエンコード
  - Lawrence Chan, Jae W. Lee, Alex Rothberg, and Paul Weaver : Parallelizing H.264 Motion Estimation Algorithm using CUDA, Final report for 6.963: CUDA@MIT in IAP 2009

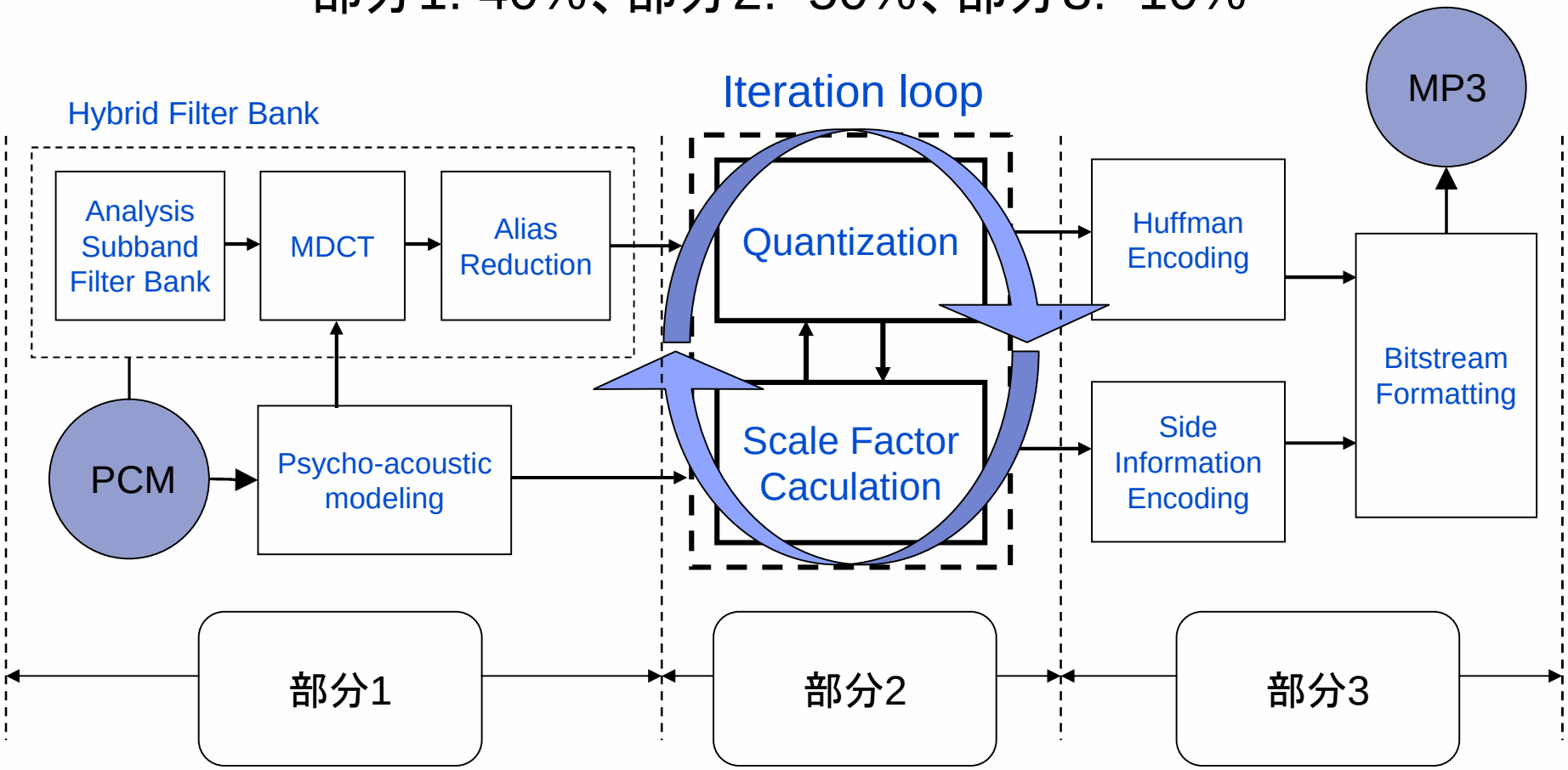
    H.264の移動予測部分の並列性を利用し、最大56倍の高速化が達成した。1フレーム内の並列性を利用しましたが、フレーム間並列性の利用は今後研究としてされている。
  - すでに製品化されたCUDAによるエンコーダ
    - Cyberlink PowerDirector7 Ultra
      - H.264エンコーダ　3.5倍高速化
    - Badaboom Media Converter

提案方法1

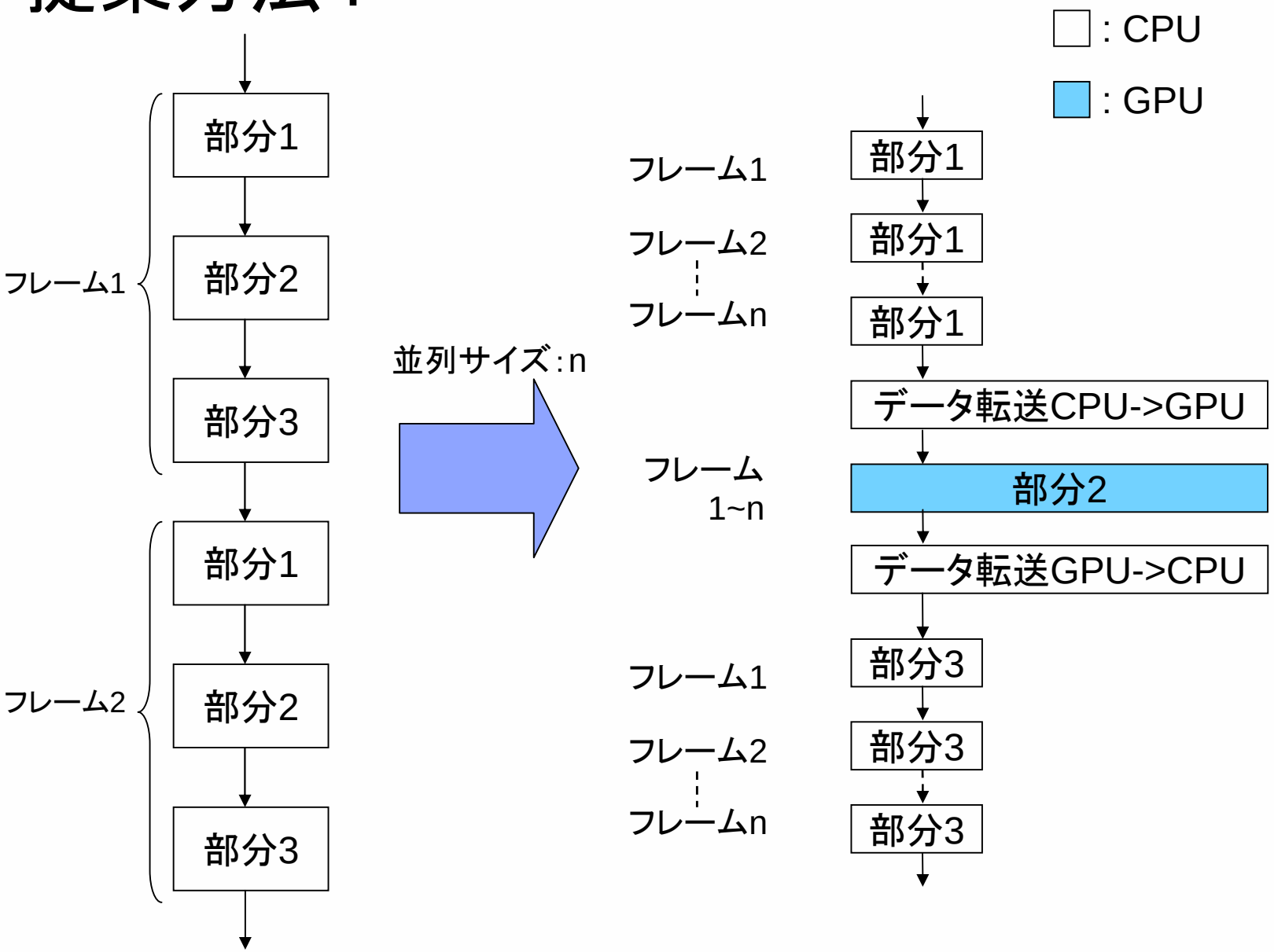
- フレーム内並列性をする上に、音声フレーム間の並列性を利用し、PCMからMP3までのエンコード処理の一部を高速化
  - 前後フレームの依存関係により、並列化できない部分を逐次で処理する
  - Iterationloop部分の各MP3フレームデータ部分の利用できるビット数の固定により、データ依存関係の解消
  - GPUの処理特性により、一部Double型変数をFloat型変数へ変更する
  - 使用するMP3エンコーダ：Bladeenc　0.94.2

提案方法1

- 説明を簡潔するため、MP3のエンコードプロセスを以下の図のように部分1、部分2、部分3に分ける
- BladeEnc0.94.2では各部分の実行時間の割合が約:
  部分1: 40%、部分2: 50%、部分3: 10%

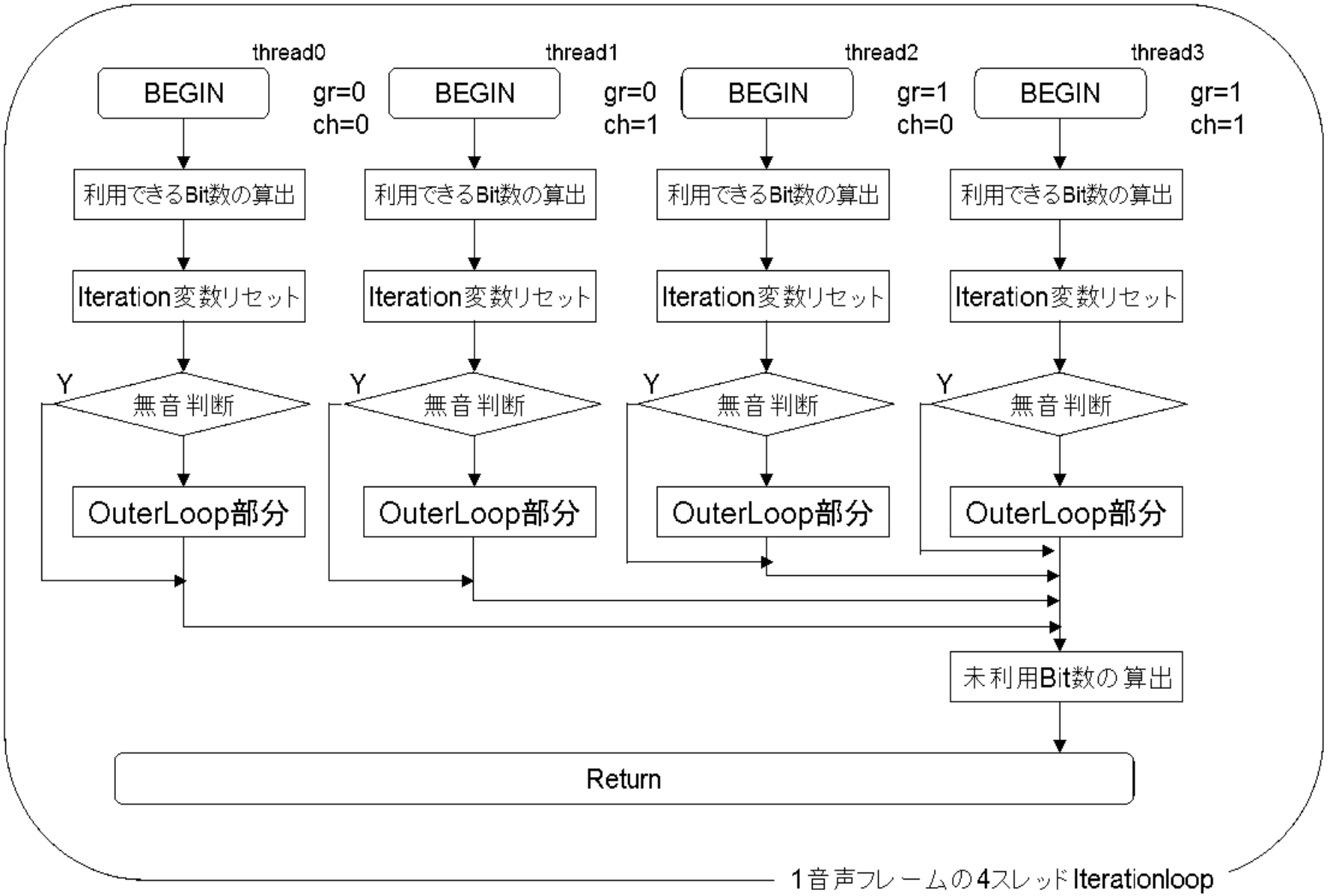


提案方法1


: CPU
: GPU
部分1
部分2
部分3
フレーム1
部分1
部分2
部分3
フレーム2
並列サイズ:n
フレーム1
フレーム2
フレームn
部分1
部分1
部分1
データ転送CPU->GPU
フレーム
1~n
部分2
データ転送GPU->CPU
フレーム1
フレーム2
フレームn
部分3
部分3
部分3

提案方法1

- フレーム内並列性の利用
  - 1フレームのMP3データは2 granuleから構成される。さらに、1 granuleは2channelに分けられ、1フレームは4単位データを持っている。それぞれが同じ処理をされるため、並列で計算することが可能。



1音声フレームの4スレッドIterationloop

提案方法1

- 実験環境

CPU	Intel Pentium4 3.0G
GPU	GTX260 240SP 896M Memory
OS	Windows XP
Compiler	VS2005+Cuda 2.2
同時エンコード フレーム数	1024
GPU並列サイズ	32Blocks 128Threads

提案方法1

- 実験データ

ソースファイル：PCM、44.1Khz、2Channel、1411kbps、長さ53分29秒

ターゲットファイル：MP3、 44.1Khz、2Channel、128kbps

	CPU部分2実行総時間（秒）
BladeEnc0.94.2	124.05

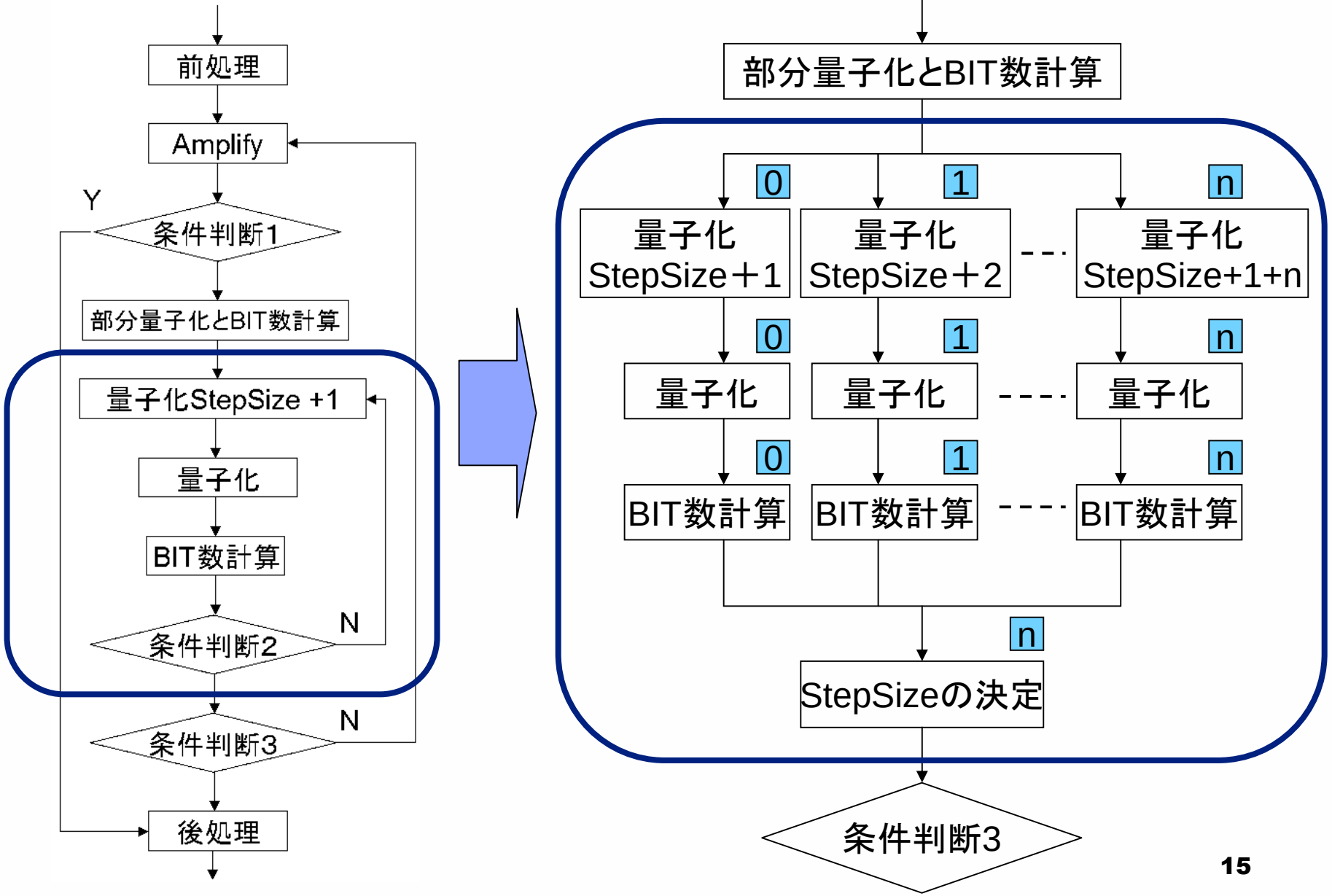
	GPU部分2総実行時間（秒）	総データ転送時間（秒）	総時間（秒）
提案方法1	28.85	6.88	35.73

提案方法1により、71.2%のiterationloop実行時間が節約したが、実行スレッド数が4096だと考えると、さらに高速化する余地がある。

提案方法2

- 方法1の上で、さらにフレーム内の並列性を利用する（現在実装中）
  - 量子化の繰り返し部分を複数のスレッドで実行し、並列性を向上させる。
  - GPUスレッドはできるだけ、条件分岐が少ないコードを実行させる。

提案方法2


前処理
Amplify
Y
条件判断1
部分量子化とBIT数計算
量子化StepSize +1
量子化
BIT数計算
N
条件判断2
N
条件判断3
後処理
n : Thread ID
部分量子化とBIT数計算
0
1
n
量子化
StepSize＋1
量子化
StepSize＋2
量子化
StepSize+1+n
量子化
BIT数計算
StepSizeの決定
条件判断3

まとめ

- MP3のiterationloop部分は、フレーム間の並列性を利用したことにより、高速化することが可能。
- しかし、条件分岐などCUDA　SIMD構造に不利な要素があるため、現在フレーム内の並列性とフレーム間の並列性を同時利用する方法を実装している。

主要参考文献

- 鹿野 裕明 , 鈴木 裕貴 , 和田 康孝 , 白子 準 , 木村 啓二 , 笠原 博徳 ," MP3エンコーダを用いたヘテロジニアスチップマルチプロセッサの性能評価",情報処理学会研究報告. 計算機アーキテクチャ研究会報告　2006
- Lawrence Chan, Jae W. Lee, Alex Rothberg, and Paul Weaver : Parallelizing H.264 Motion Estimation Algorithm using CUDA, Final report for 6.963: CUDA@MIT in IAP 2009
- 酒居 敬一, 光成 滋生, 成田 剛, 石田 計, 藤井 寛, 庄司 信利,"MP3エンコーダの高速化実装",情報処理学会論文誌, 第43巻, 第4号, pp.1028--1038, April 2002.