CEDEC2006 CRI・ミドルウェア　セッション資料

いよいよ1080pの時代が到来！
「CRI Sofdec」による高画質ムービーの実現
～ CellによるSofdecデコーダの並列処理 ～

2006年8月30日

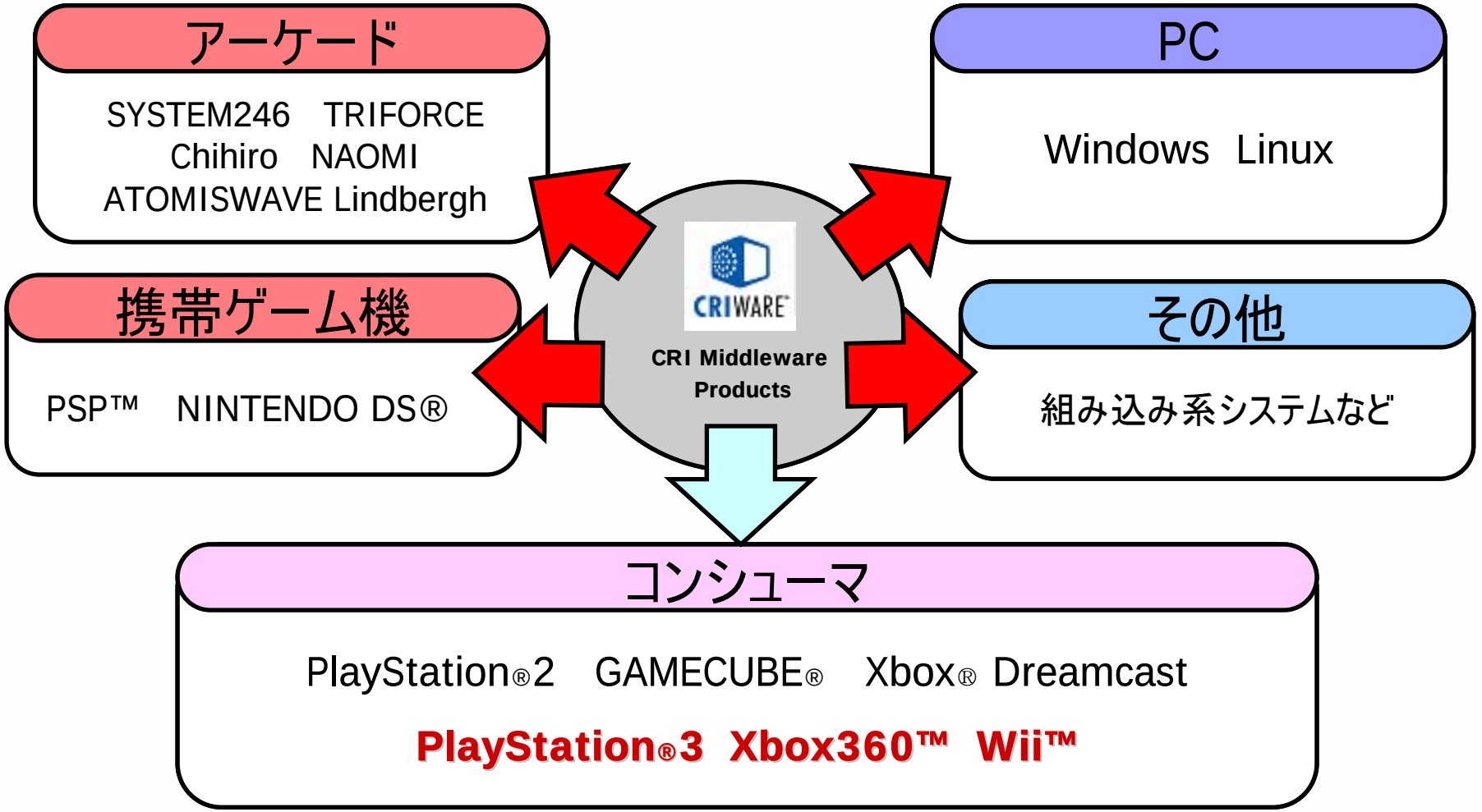
(株)CRI・ミドルウェア

Ver.1.02

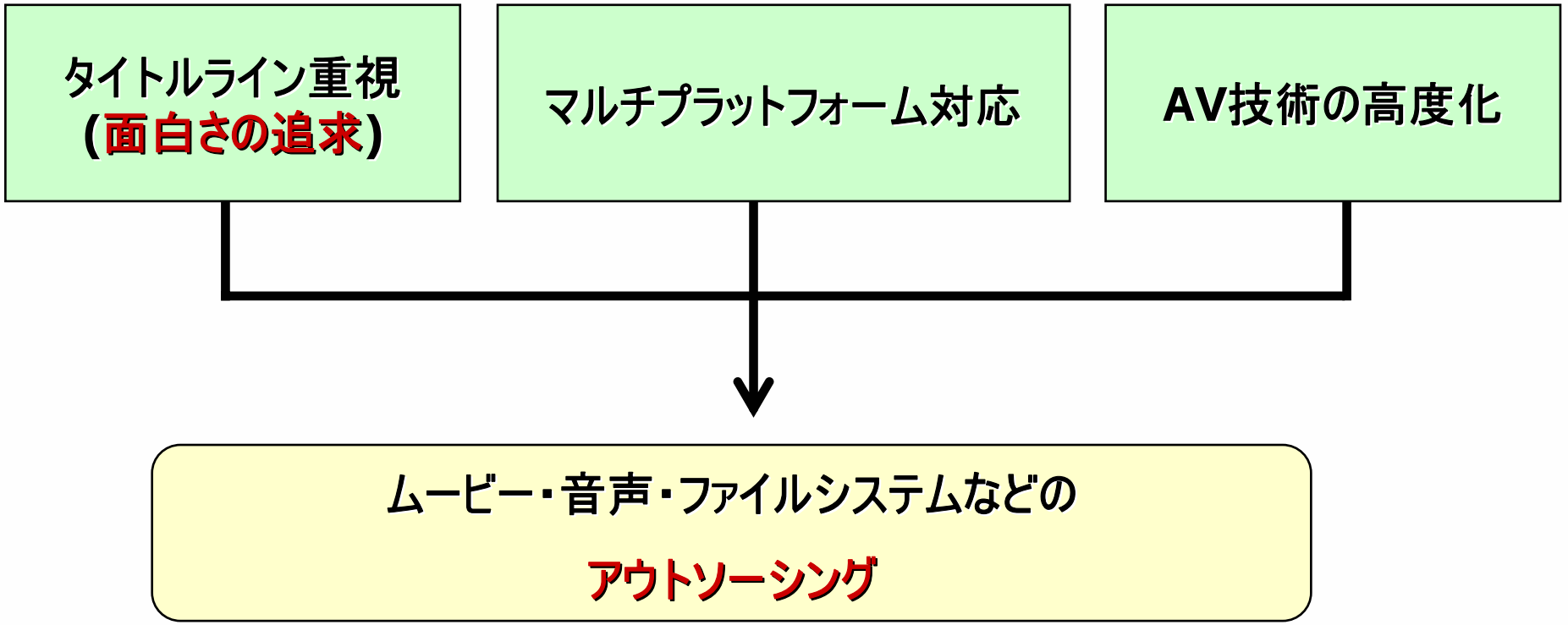
セッション概要

1. 高画質ムービーとHD向けサポート機能
   - HDムービーにおける高画質とは
   - Sofdecを用いた応用例
   - マルチリンガルムービー
2. CellによるSofdecデコーダの並列処理
   - Sofdecの並列処理設計
   - CPU時間の有効活用
   - 並列デコード・SPUコードの高速化

マルチプラットフォーム展開



なぜミドルウェアか？


タイトルライン重視
(面白さの追求)
マルチプラットフォーム対応
AV技術の高度化
ムービー・音声・ファイルシステムなどの
アウトソーシング

1. 高画質ムービーとHD向けサポート機能

- HDにおける高画質とは
- Sofdecによる高画質ムービーの実現
- Sofdecの応用例

  → オープニング/エンディングだけがSofdecではない!!
- HD向けサポート機能

  → Sofdecを使ってゲーム開発を楽にしよう!!

HDムービーとは

略称	日本規格名	解像度	フレームレート	表示方式
1080i	D3	1920x1080	29.97Hz 30Hz	インタレース
720p	D4	1280x720	59.94Hz 60Hz	プログレッシブ
1080p	D5	1920x1080	59.94Hz 60Hz	プログレッシブ

- **次世代ゲーム機では720pが主流**
- **プリレンダリングムービーは1280x720, 30Hz**

SofdecによるHDムービー再生

1. 実写	1280x720, 59.94fps
2. アニメーション	1280x720, 24fps
3. CGムービー(プリレンダ)	1280x720, 59.94fps

解像度とビットレート

規格名	フレームレート	ビットレート	データサイズ
480i	29.97 Hz	4～8Mbps	1倍
1080i	29.97 Hz	18～26 Mbps	6.75倍
720p	59.94 Hz	18～25 Mbps	6倍
720/30p	**29.97 Hz**	**8～20 Mbps**	**3倍**
1080p	59.94 Hz	22～30 Mbps	13.5倍

HDムービークオリティとは

1. **動きがなめらか / チラつかない**

   現行の放送は60Hzインターレース、ゲームのムービーは30Hz

   ➔ **フレームレート60Hz ＋ プログレッシブ**

2. **細線をシャープに再現 / ジャギーレス**

   ➔ **高解像度 1280x720/1980x1080**

3. **立体感・奥行き感を再現**

   ➔ **HDといえども平面のはず？？？**

立体感・奥行き感？

平面での立体表現 ⇒ **陰影法**

陰影法 ➔ 影 ➔ グラデーション ➔ **階調表現能力**

微妙な色の変化から立体感を感じ取っている

グラデーションとHD

■ 諧調表現が低いとマッハバンドが見える

- 階調表現を損なわない圧縮方法が必要

■ 高解像度のメリット

- 複数のピクセルの組み合わせで階調を増やす
- 単位面積当たりのピクセル数が増えるため、
  より滑らかなグラデーションが表現可能 (**擬似諧調表現**)

美しい映像とは・・・

綺麗な
元の画像

平面的な映像

諧調が
失われると・・・

拡散ディザ (RGB各3ビット)

高解像度であれば
擬似諧調で
クオリティUP!!

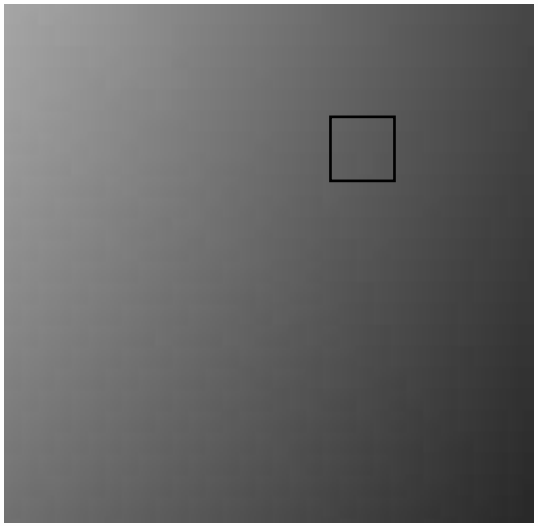
マッハバンドノイズ

高周波成分の削除

ディザの消滅

マッハバンドノイズ

ディザの周波数領域への変換



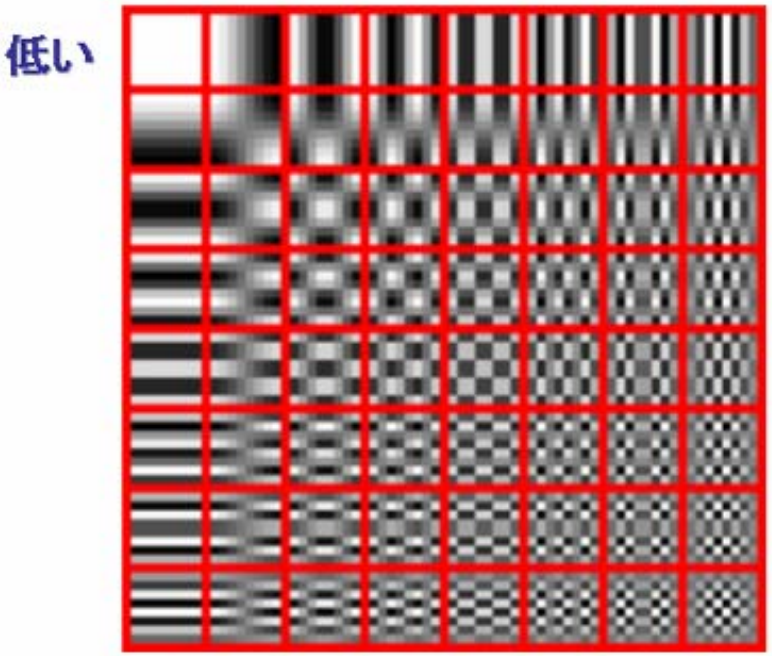
DCT変換 (8×8)

237	0	0	0	0	1	0	0
0	0	0	0	0	0	0	-1
0	0	0	1	0	0	0	0
0	0	0	0	0	-1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	-1	0	0	0
0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0

ディザパターンは小さい高周波成分を持つ

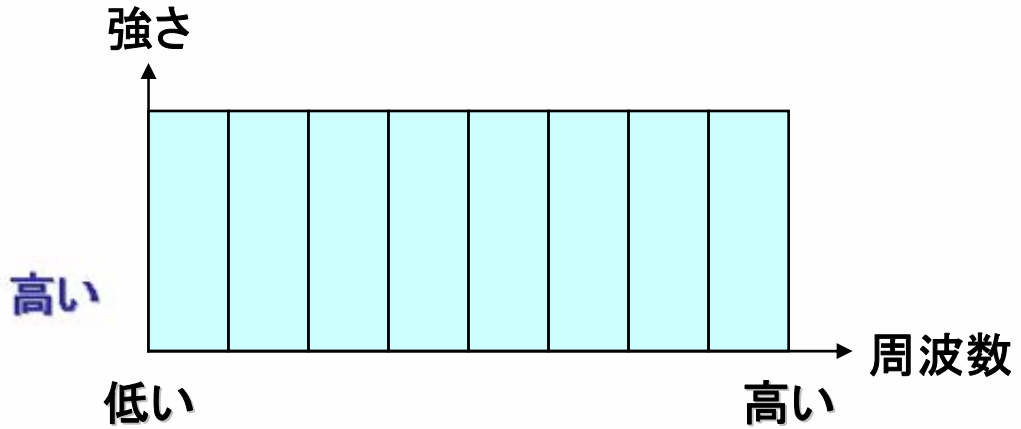
➔ 高周波成分を削ると細かいディザが消える。

DCT変換



DCT変換の性質

等間隔の周波数成分に分解



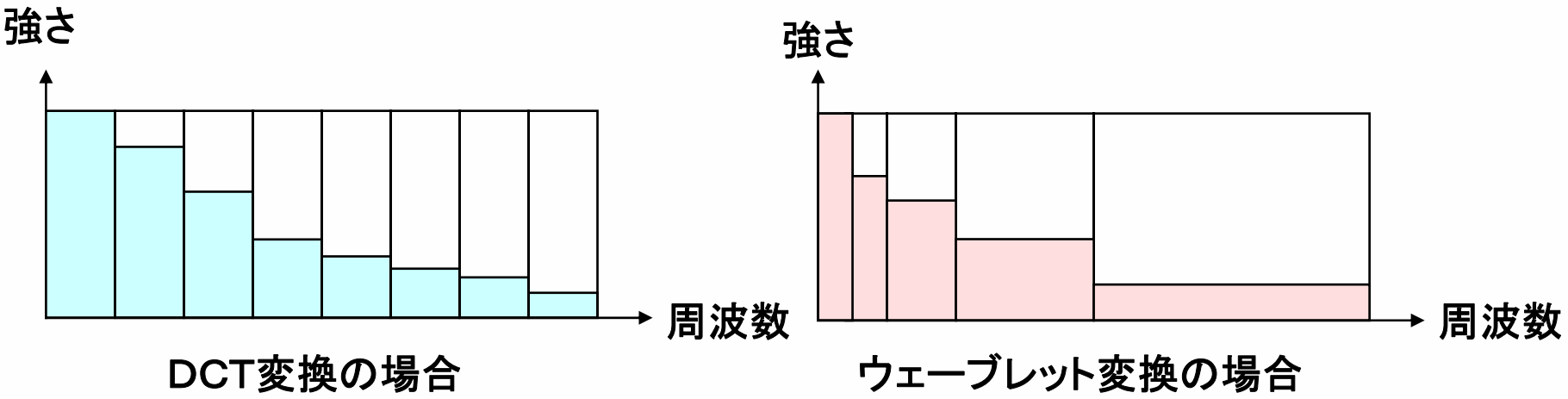
周波数領域での圧縮

一般的な画像圧縮の手法

高周波成分の情報を削り圧縮

➔ ローパスフィルタをかけたことと等価

➔ マッハバンドノイズ

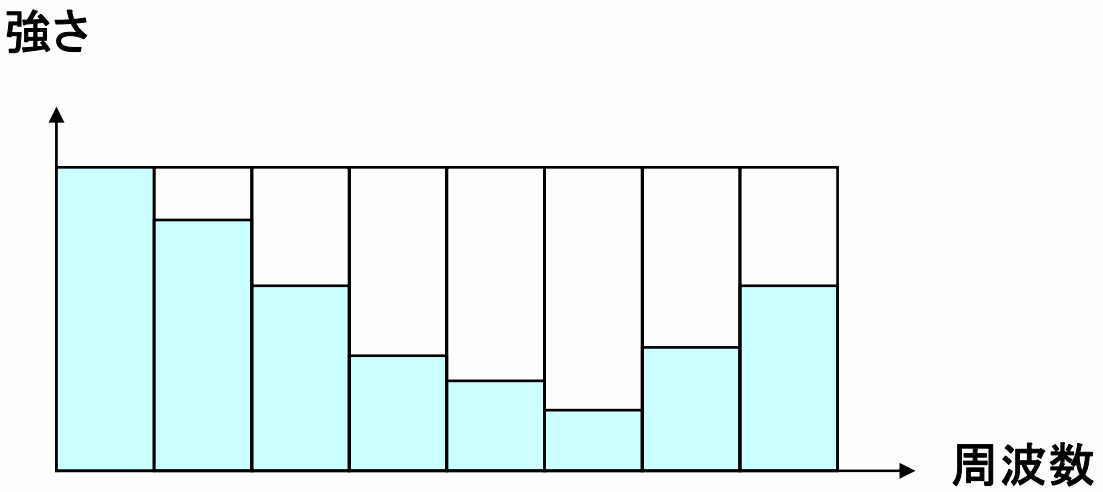


Sofdecの圧縮方法

Sofdecの圧縮方法

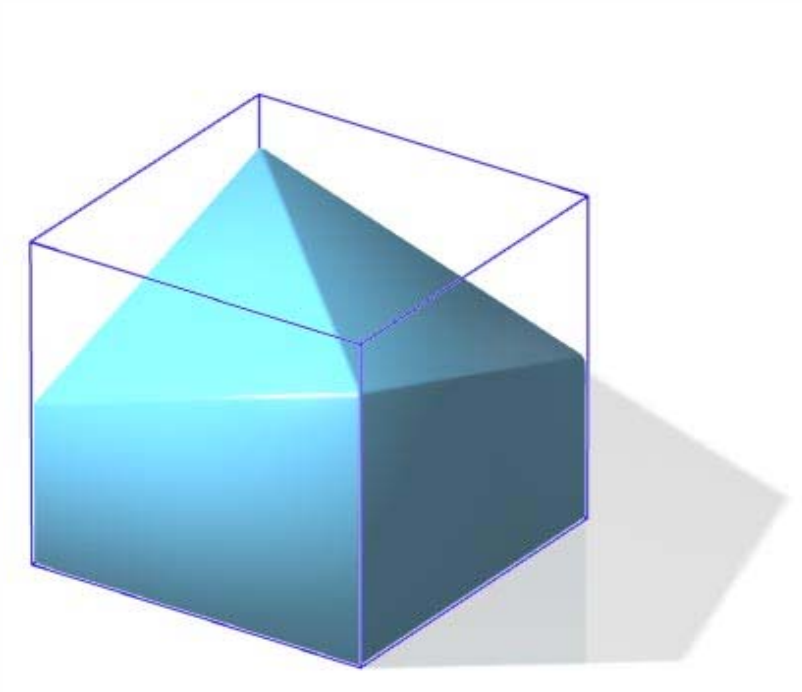
下図のように高周波成分を残す

➔ **マッハバンドノイズに強い / シャープで質感が保持される**



Sofdecの量子化マトリックス

8	1	1	2	4	8	16	24
1	1	1	4	8	16	32	16
1	1	1	8	16	32	16	8
2	4	8	16	32	16	8	8
4	8	16	32	24	8	8	4
8	16	32	16	8	8	4	2
16	32	16	8	8	4	2	1
24	16	8	8	4	2	1	4

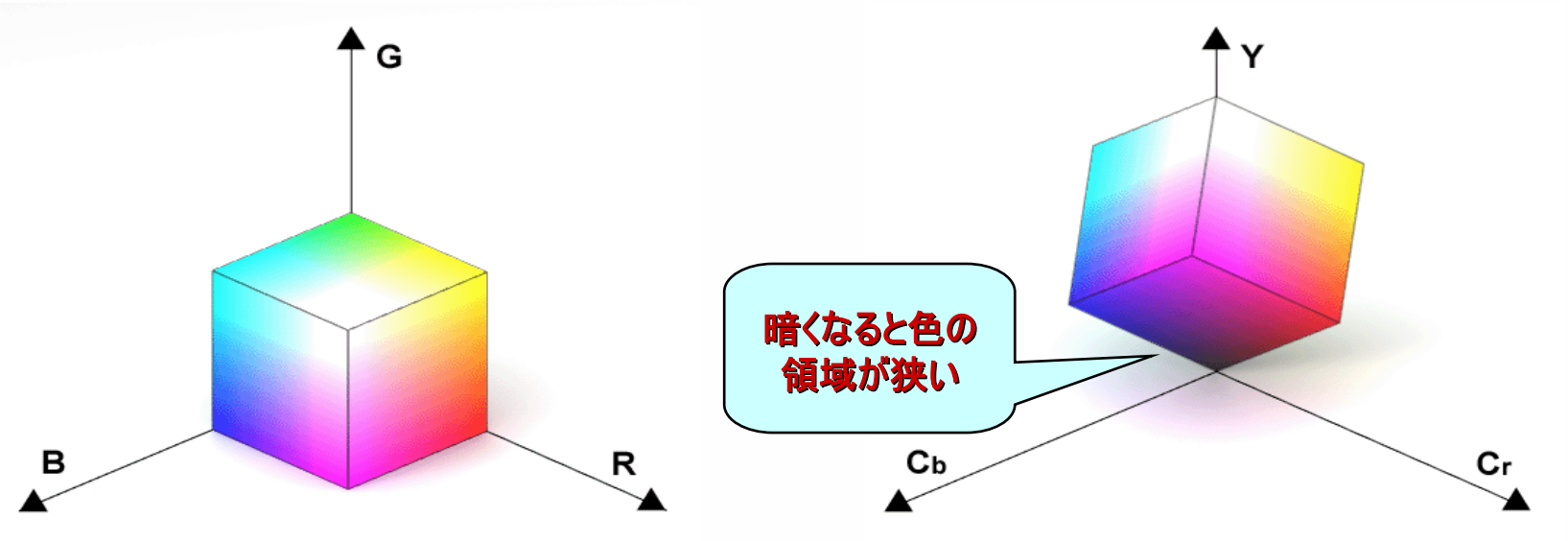


高周波成分を保存 ➔ シャープネス & 質感を保持

色空間補正

RGB空間とYUV空間

Y:輝度, U(Cb):赤の色差, V(Cr):青の色差



RGB空間　　YUV空間

HDムービーのまとめ

■ 圧縮のポイント ➔ Sofdecは高域を重視して圧縮

- CGムービーにおいて特に高周波成分は重要
- 高域を削ってしまうと、シャープさや質感が損なわれる

■ HD画質

- 解像度が上がる ➔ ・よりシャープに
  ・より滑らかに

} 質感・立体感

■ ディザとマッハバンド

- ディザによって階調表現が豊かになる
  ➔ ディザがなくなるとマッハバンドノイズに・・・

Sofdecを応用した表現手法(1)

- テクスチャ再生
  → ポリゴン上のテクスチャとして扱える
- アルファムービー
  →ムービーとポリゴンとの重ね合わせ
- Zムービー
  → 奥行きを加味したムービーとポリゴンとの重ね合わせ
- シームレス再生
  → 複数のファイルを連続した一本のファイルのように再生/分岐

Sofdecを応用した表現手法(2)

- マルチストリーム再生
  → Xbox360のCPU 1コアで64本のムービーを再生
- マルチリンガルムービー
  → 複数言語の音声、字幕テキストを簡単に切り替え
  もちろん、5.1chもサポート

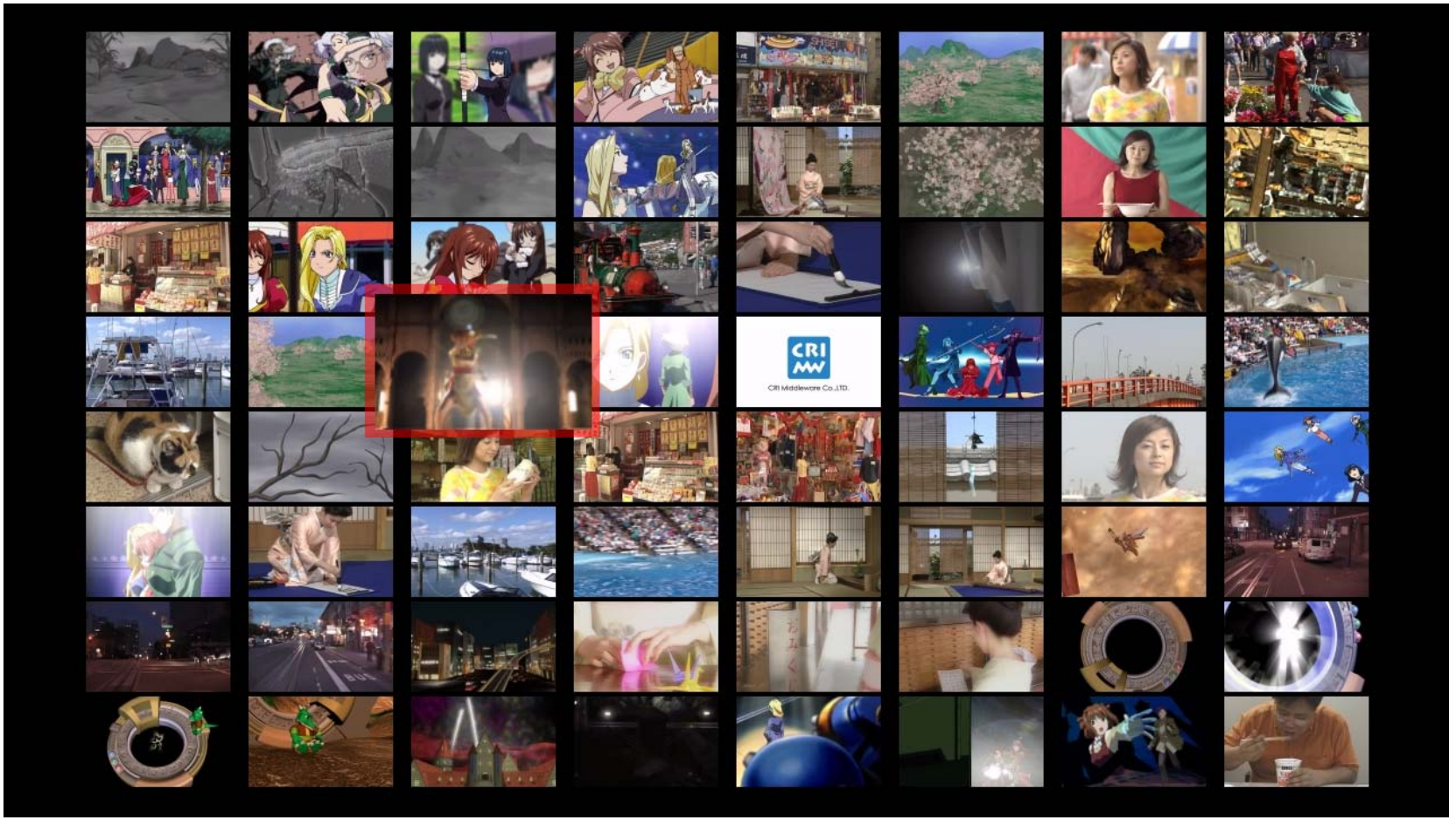
テクスチャ再生

アルファムービー

Zムービー

シームレス再生

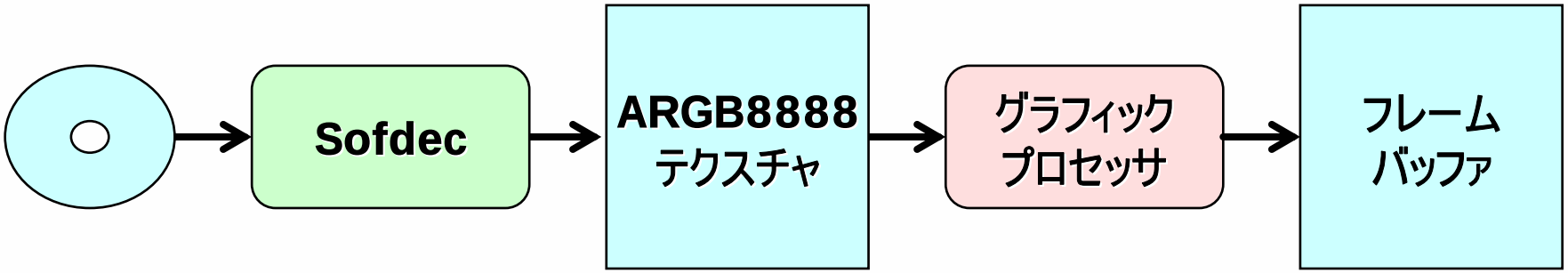
マルチストリーム再生



マルチリンガルムービー

アプリケーションとのインタフェース

■ デコーダと表示系との分離

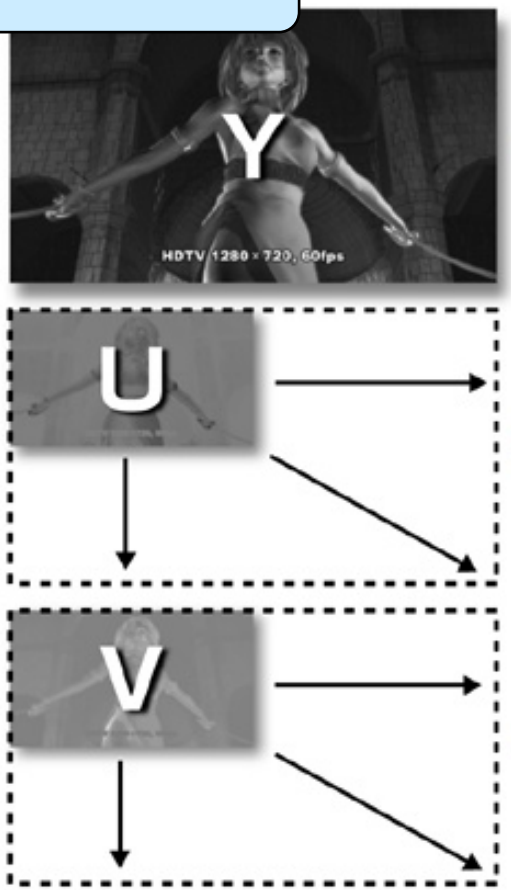


AV同期

- イメージデータを取得できたタイミングで表示するだけ
- **24fps素材**でもスムーズに再生

ピクセルシェーダによる色空間変換

Sofdecデコーダ出力



- 3枚のテクスチャとして各YUVプレーンを入力
- YUV-RGB変換 → フレームバッファ

CPU → デコード
GPU → 色空間変換
負荷を分散

アルファムービー(用途別機能)

- **加算・全画面α合成**
  - 通常のムービー再生でOK
  - 頂点のα値や演算方法をコントロール
- **ルミナンス合成**
  - 輝度情報からα値を計算
  - 演算が軽く、光や煙のエフェクト向き
- **3値α合成**
  - 透明、半透明、不透明 ➔ セル画のような合成向き
  - リソース(CPU負荷、メモリ)を消費しない
- **フルα合成**
  - 256段階のα値を再現
  - リソース: CPU負荷1.5倍、メモリ2倍

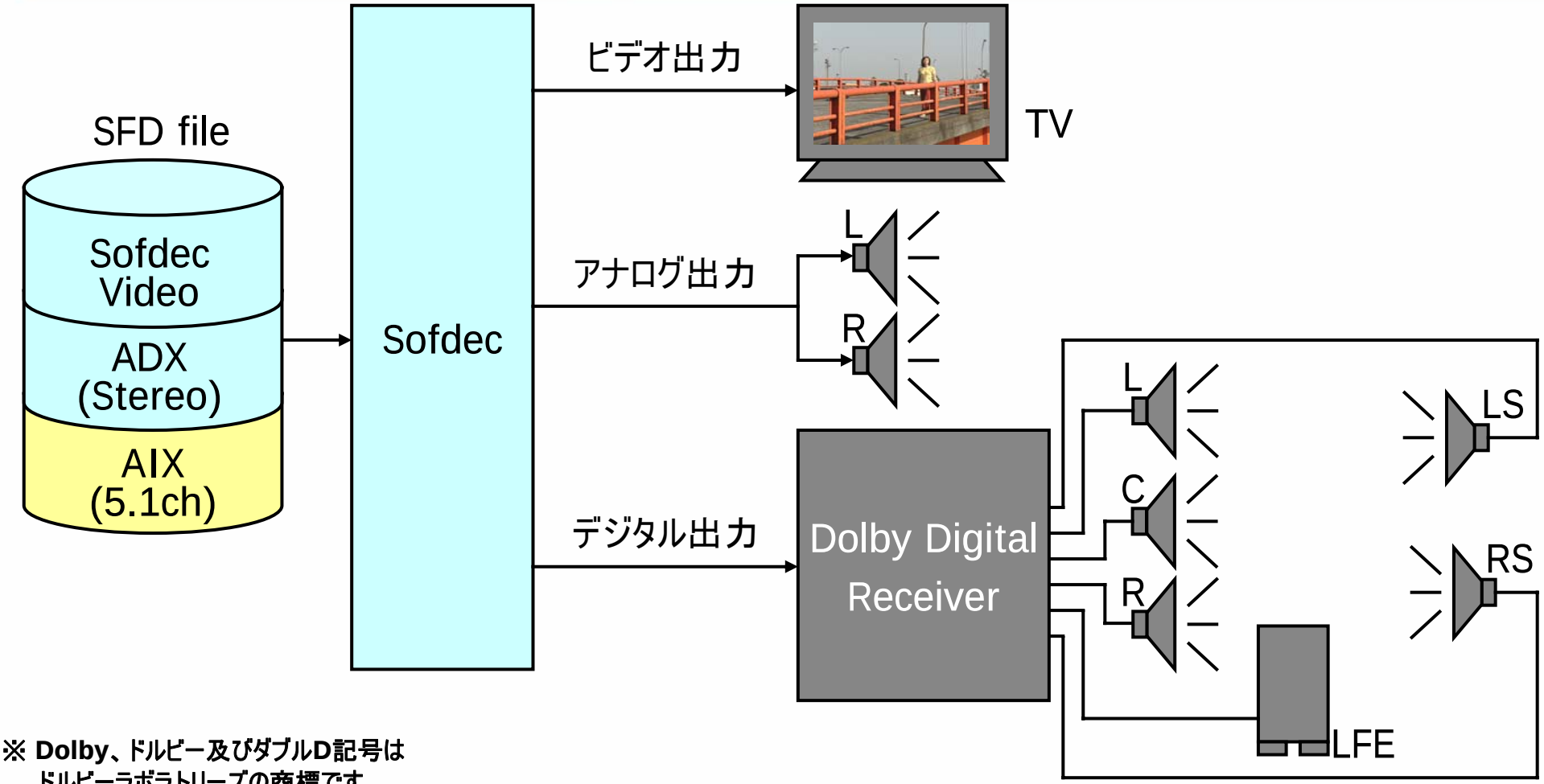
シームレス再生(補足)

複数ムービーの連結再生

- ムービーを複数ファイルに分割し、映像・音声とも継ぎ目なく再生
- 一つのファイルをループ再生することも可能
- 任意の位置(ファイル)から再生することが可能

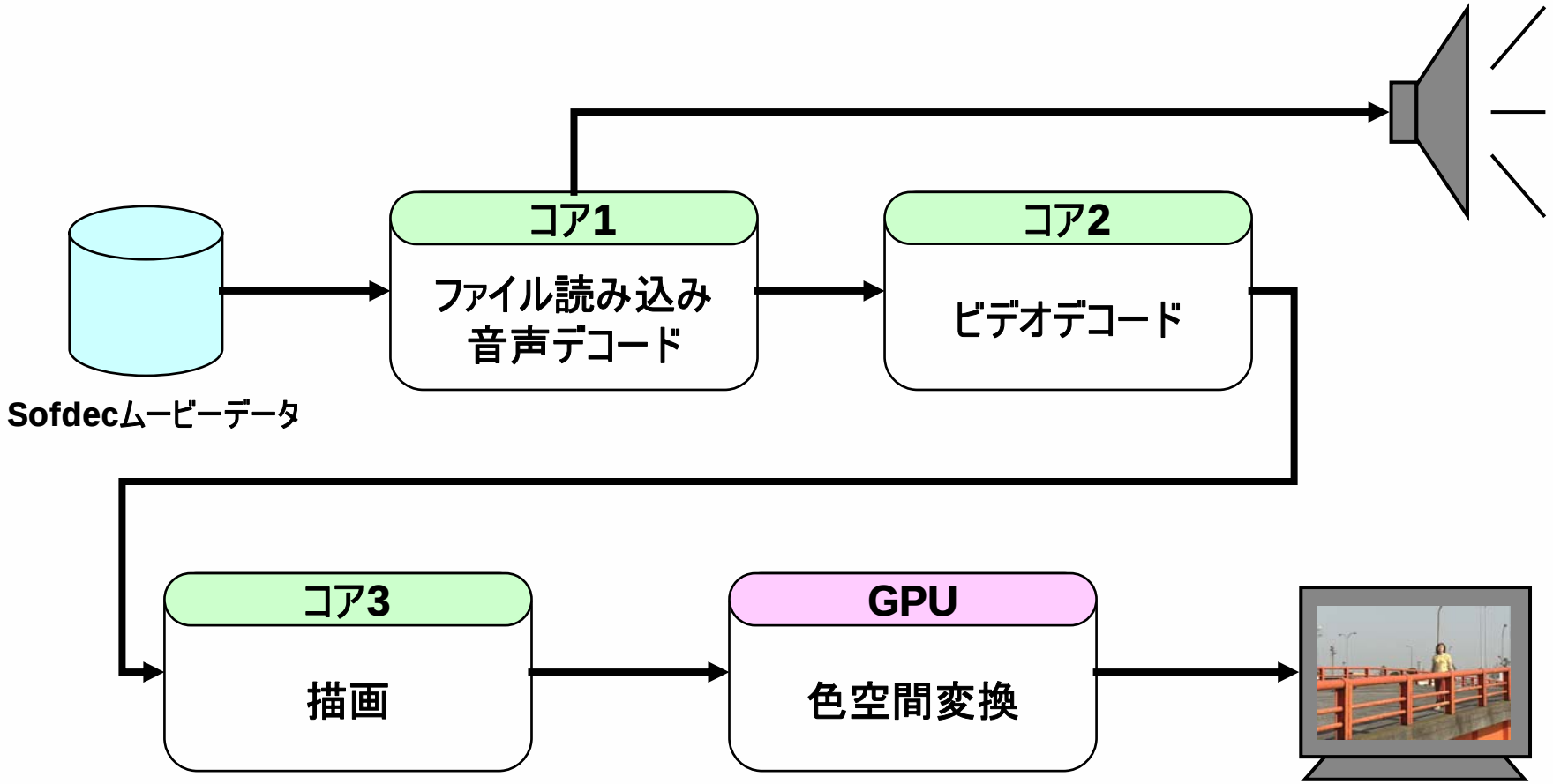
  → 擬似シーク機能

Sofdec with Dolby Digital



※ Dolby、ドルビー及びダブルD記号は
ドルビーラボラトリーズの商標です。

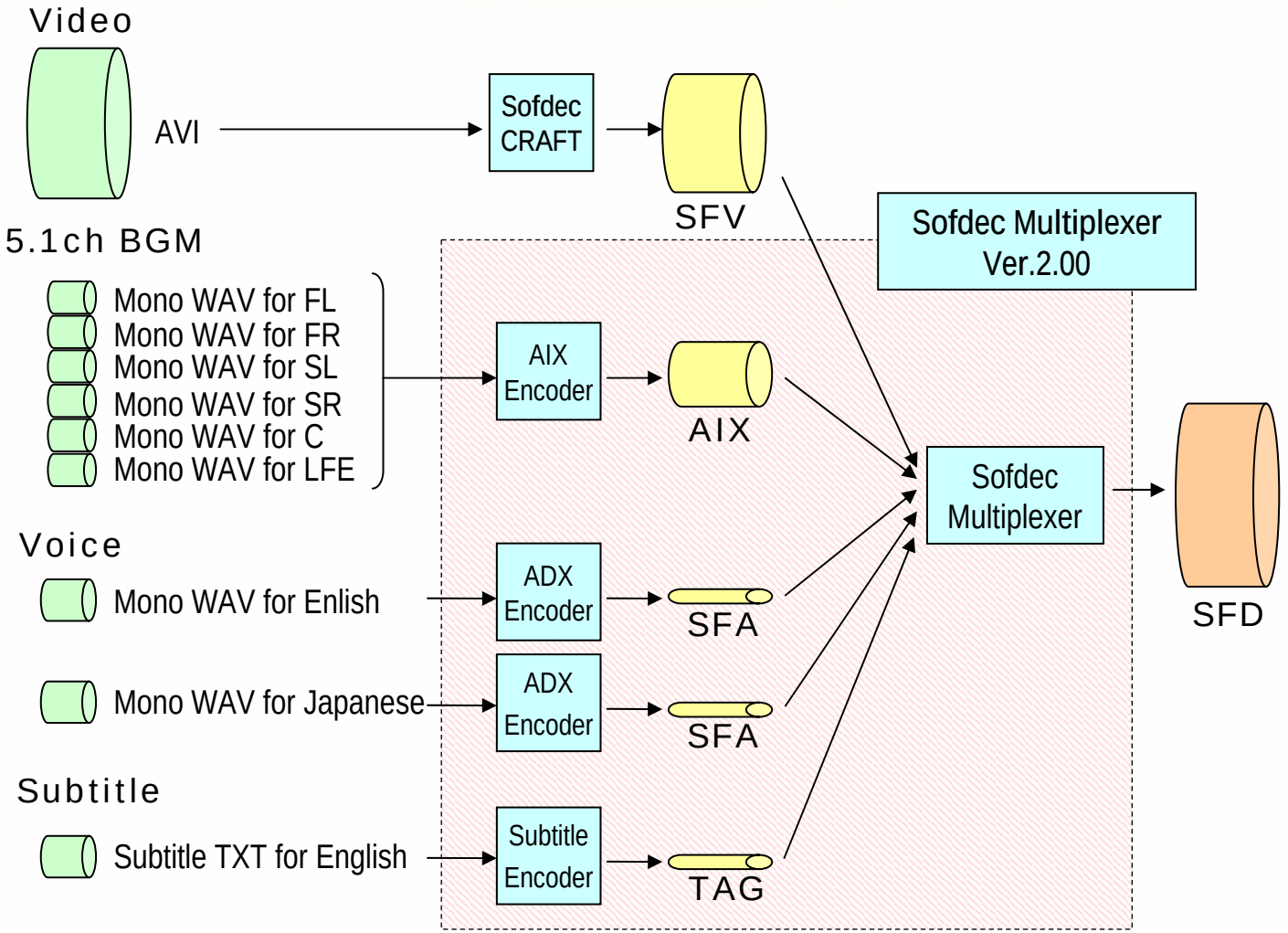
マルチコアによる動画再生



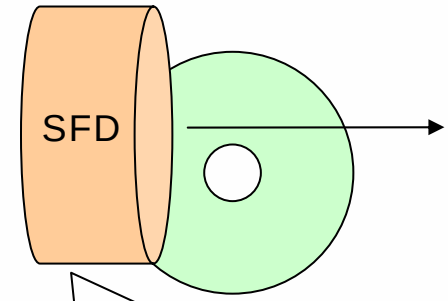
マルチリンガルムービー (必要な素材)

- ビデオ
  - 一般的なSofdecと同様にエンコード
- セリフ
  - モノラルWAVファイルを用意
  - 再生時にはセンターチャネルから出力
- 字幕テキスト
  - 時刻ユニット、IN/OUT点、文字列を記述
  - S-JIS, UTF-8, UTF-16(LE/BE)
- (必要に応じて)5.1ch用オーディオファイル
  - モノラルWAVファイル x 6本

マルチリンガルムービー(エンコード)



マルチリンガルムービー(再生処理)



Video Ch00 : Video
Audio Ch00 : n/a
Audio Ch01 : 5.1ch BGM
Audio Ch02 : English Voice
Audio Ch03 : German Voice
Audio Ch04 : Spanish Voice

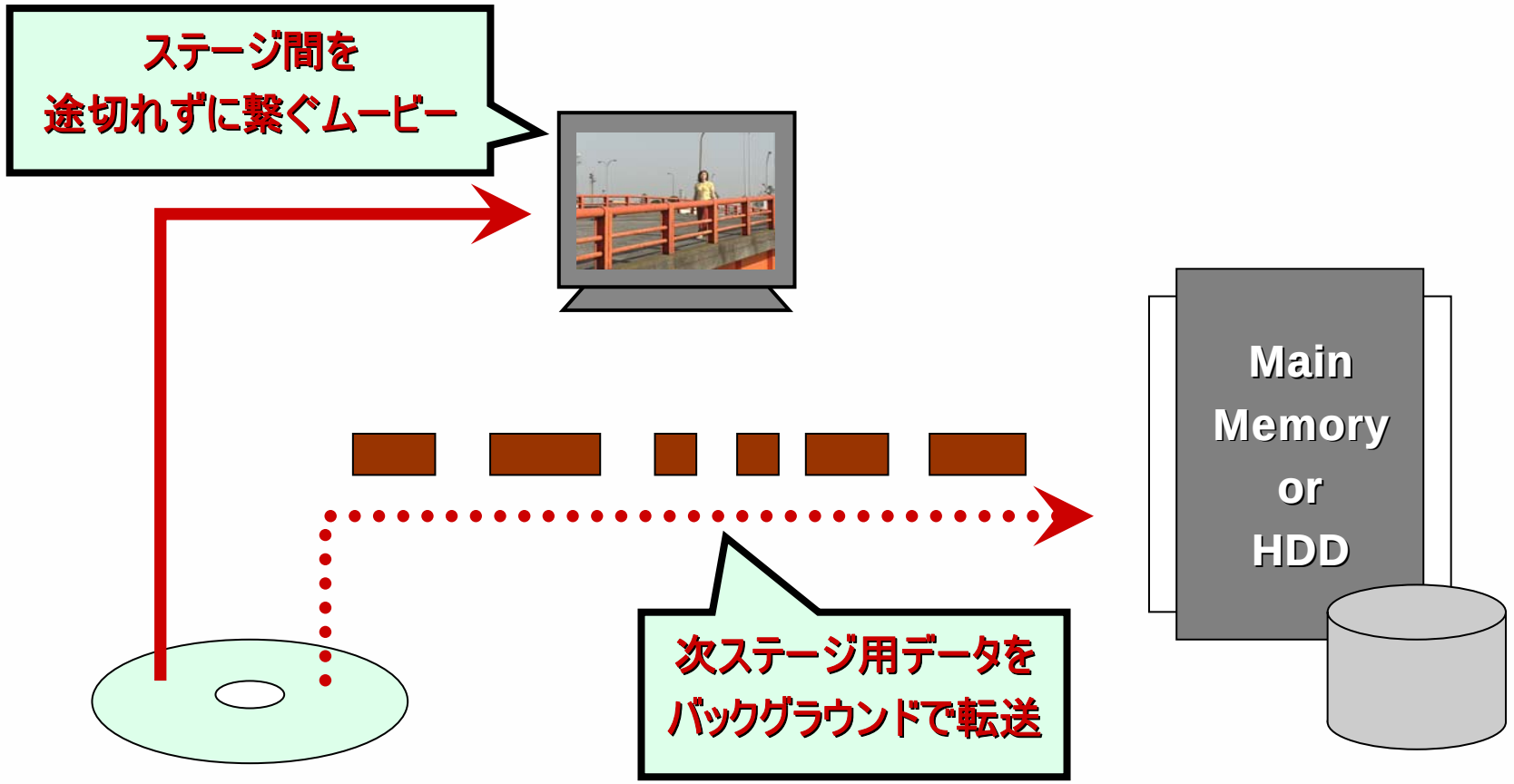
Game Program

```
mwPlyInitSfdFx(...)
ply = mwPlyCreateSofdec(...)

bgm_wksize = mwPlyGetMultiChannelWorkSize()
bgm_wkptr = malloc(bgm_wksize)
mwPlyAttachMultiChannel(ply, bgm_wkptr, bgm_wksize)
mwPlySetMultiChannelCh(ply, 1);

voice_wksize = mwPlyGetVoiceWorkSize()
voice_wkptr = malloc(voice_wksize)
mwPlyAttachVoice(ply, voice_wkptr, voice_wksize);
mwPlySetVoiceCh(ply, 16);

mwPlyStartFname(ply, ....)
```

5.1ch BGM Setting

Voice Setting

バックグラウンドデータリード



バックグラウンドデータリード

- Sofdec再生中もデータリードOK
  - リードしたデータをHDDに保存することも可能
  - 非同期に、必要なファイルだけをリード可能
- 大きなサイズのデータは圧縮して転送時間削減
  - アプリケーションにインパクトを与えない低負荷実装
- ステージ間を繋ぐムービー再生時に・・・

インタリーブ v.s. 非同期読み込み

- ムービーにデータをインタリーブしておくと便利？
  - ムービーのシステムデコード速度以上にリードできない
  - 部分的に必要なデータだけを取得することが困難
  - ムービースキップ時にもシステムデコードの継続が必要
- 非同期読み込みにしておけば・・・
  - 必要に応じて必要なファイルだけを取得可能
  - ムービーの再生スケジュールに依存しない
  - 細かいファイルが多い場合はシークの影響を考慮する

レンダリングプレビューとしてのムービー(Xbox360)

- ビットマップダンプツールを用意
  - ターゲット側の任意のタイミングでダンプ(プログラマブル)
  - PC側で連番BMPを出力
- Sofdec CRAFTでエンコード
  - 連番BMPはそのままエンコード可能
- 実機でプレビュー
  - Sofdecの高画質再生でリアルタイムレンダリングと同等の動画を実機上で確認可能

Sofdecはプレビューツールとしても使えます!!

レンダリングプレビューとしてのムービー

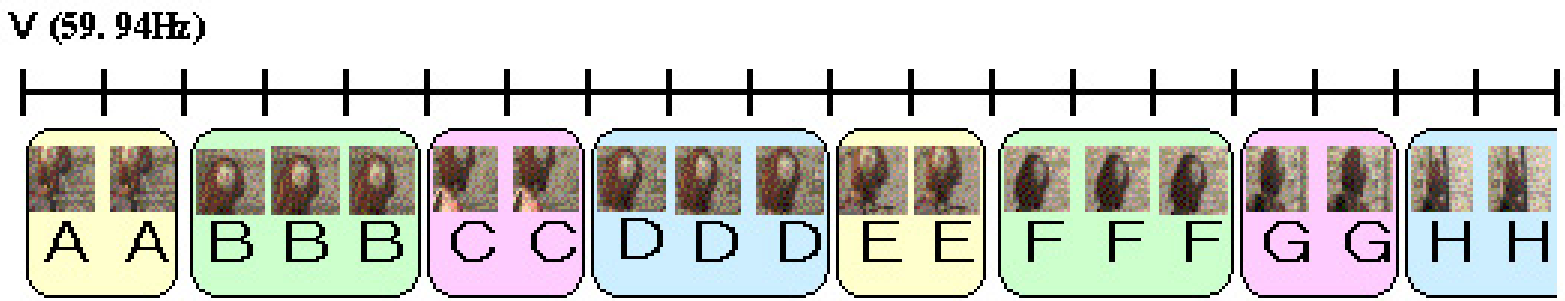
- チューニング後の性能を予測
  - 開発途中でも最終的なレンダリング結果を確認可能
- プロモーション用途にも
  - 実機クオリティそのままのムービーを作成可能
- ハードでキャプチャする場合は・・・
  - 300～1,000万円程度のシステムが必要
  - D5/HDMI映像をキャプチャする場合は機材が限られる

旧資産の活用(NTSC/PAL/映画素材)

- 旧資産を活用する際の注意点
  - フレームレートの違い
    - NTSC(29.97/59.94fps), PAL(25/50fps)
    - 映画/アニメ(24fps)
  - インタレース映像の扱い
  - ピクセルアスペクト比の違い
    - NTSCは1:0.91
    - PALは1:1.09
    - D3 ~ D5は1:1

映画・アニメを滑らかに

- 24fpsのムービーをスムーズに表示するには
  - 1Vsync間隔で画面更新がある
  - 表示間隔を均一に調整する(**Sofdecが管理**)



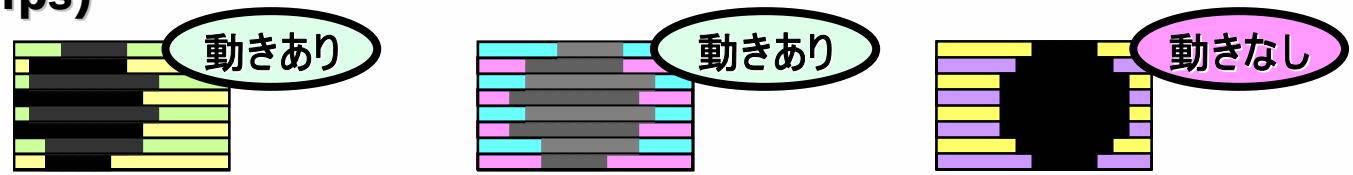
- ベストエフォート型のコーデックでは表示間隔の調整が難しい・・・

インタレース解除(I-P変換)

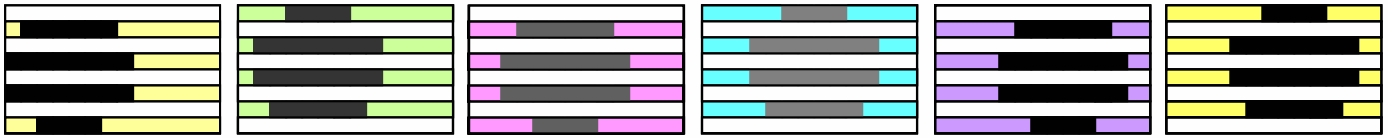
- NTSC用素材の多くは偶奇フィールドが混在
  - 1枚の静止画に時刻の異なる画像2枚分
  - そのままプログレッシブで表示すると
    → ダブルイメージが発生
- 次世代機では60p(59.94Hz)で再生しよう
  - **1枚の偶奇フィールド混在画像を2枚の静止画に**
  - 次世代機上では完全にプログレッシブで表示
    → フィールドオーダーを無視することができる

インタレース解除時の注意点

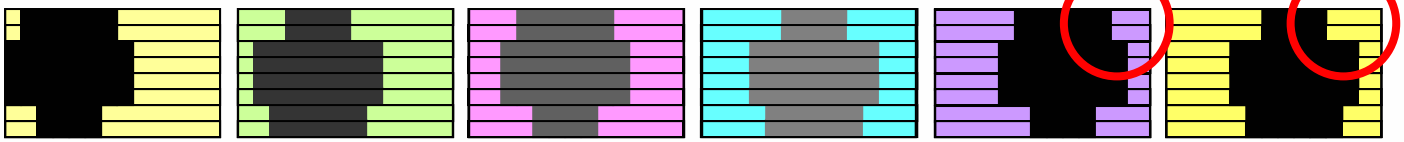
インタレース(30fps)



インタレースのフィールド分解後

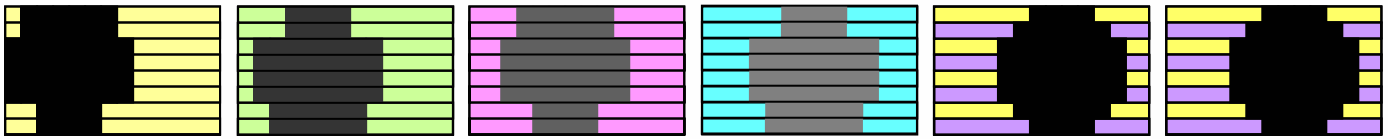


プログレッシブ(60fps):わるい例



解像度が落ちる

プログレッシブ(60fps):いい例



動きのあるところは
解像度より「動き」優先

動きの無いところは
フィールド分解しない

解像度優先

インタレース解除(D3素材)

■ 1080iから720pへ

- ゲームアプリのほとんどは**720p (プログレッシブ)**
- 実写のほとんどは**1080i(インターレース)**
  ➔ **ゲーム中で実写を使用するにはインタレース解除が必要**

■ インターレース解除は結構むずかしい・・・

- 解像度をできるだけ落とさない
- **1920x1080 ➔ 1920x1080** に縮小

  ➔ ダブルイメージ、コーミングノイズ、ボケを最小限に

エンコードサービス

- **プロフェッショナルスタッフによるエンコードサービス (有料)**
  分割エンコード, パラメータ調整

- **エンコーディングサポート (無料)**
  ノイズなどを解消できない場合は、お気軽にご相談ください。
  テストエンコードなども行います。

2. CellによるCRI Sofdecデコーダの並列処理

いよいよ1080pの時代が到来！

1080pの映像って、、、見たことありますか？

圧縮データの階層構造

Sofdecビデオデータは階層構造となっている。

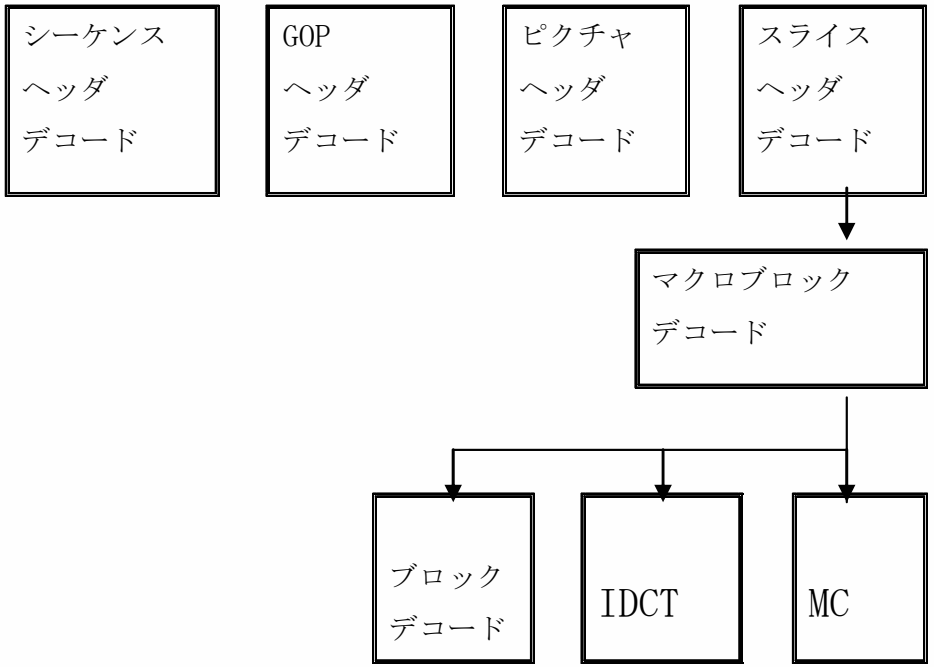
圧縮データの階層構造に着目し、
CellにおけるSofdecビデオデコーダの並列処理を
設計した。

圧縮データの階層構造

•シーケンス層	画像サイズ、フレームレート等
•グループオブピクチャ（GOP）層	タイムコード等
•ピクチャ層	ピクチャタイプ(I,P,B)等
•スライス層	スライス位置、Qスケール
•マクロブロック層	MBタイプ、動きベクトル等
•ブロック層	DCT DC/AC係数

※スライス層以上の上位階層はそれぞれの開始位置に
スタートコードが埋め込まれている。
スタートコードからデコード処理を開始できる。

Sofdecビデオデコーダのモジュール構造



- 各種ヘッダデコードデコード(シーケンス層、GOP層、ピクチャ層、スライス層)
- マクロブロックデコード(可変長符号デコード処理)
- ブロックデコード(可変長符号デコード処理、逆量子化)
- IDCT(逆離散コサイン変換)
- MC(フレーム参照、平均値計算、ブロック加算、飽和演算、フレーム出力)

マルチプロセッサによる並列処理の設計

設計の自由度は高い。

いろいろな並列処理が実装可能だが、
設計が悪ければ効果は低い。

マルチプロセッサの性能を最大限に引き出す
ために、並列処理の設計が重要。

マルチプロセッサによる並列処理の設計

【結論】

様々な並列処理方法を比較検討して、

Sofdecビデオデコーダは、

圧縮データのスライス構造

に着目した並列処理を採用。

マルチプロセッサによる並列処理の設計

（1） ループ処理の並列化
（2） 機能分割による並列化
（3） 圧縮データの階層構造による並列化
（4） 並列化の設計に応じた粒度の比較

（1） ループ処理の並列化

MB層やブロック層の処理ループは、処理を進めなければ
ビットストリームの区切りが分からないので順次処理するしかない。
（並列化不可能）

IDCTやMCの処理には単純なループ構造があり、
ループ処理を並列化できる。

しかし、、、

（1） ループ処理の並列化

【欠点】

- SIMD命令による命令レベルの並列処理があるので、
  プロセッサレベルで並列処理に割り当てる処理量はとても小さい。
- 並列処理の処理量が小さいので、相対的にオーバヘッドが大きくなる。
- 使用するSPUの数が固定的。

（2） 機能分割による並列化

使用するSPUの数を決めて、それぞれのSPUの処理負荷ができるだけ均等になるように機能を割り当てる。

例えば、SPUを3個使用して、

- SPU_0 ： VLD（MB層デコード＋ブロック層デコード）
- SPU_1 ： IDCT
- SPU_2 ： MC

（2） 機能分割による並列化

VLDの出力がIDCTの入力となり、IDCTの出力がMCの入力となる。
並列動作させるために、次のようにパイプラインを組む。

SPU_0　VLD(0) → VLD(1) → VLD(2) → VLD(3) → VLD(4) → ・・・
SPU_1　IDCT(0) → IDCT(1) → IDCT(2) → IDCT(3) → ・・・
SPU_2　MC(0) → MC(1) → MC(2) → ・・・

※デコード処理の一部をハードウェアアクセラレータで高速化する場合と同じ処理の流れとなる。
SPU_1をIDCTアクセラレータ、SPU_2をMCアクセラレータと見なせる。

（2） 機能分割による並列化

【欠点】

- 同期のオーバヘッドが大きい。
- VLD,IDCT,MCの各処理負荷は変動しているし、
  均等にはならない。
- 各スロットで最大の処理時間のSPUにあわせて
  他のSPUはウエイトする。
- 使用するSPUの数が固定的。

（3） 圧縮データの階層構造による並列化

（3－1） シーケンス処理の並列化
（3－2） GOP処理の並列化
（3－3） ピクチャ処理の並列化
（3－4） スライス処理の並列化

（３－１） シーケンス処理の並列化

複数ムービを同時再生する場合に、
１つのムービのデコード処理を特定の
プロセッサに割り当てる。

【特徴】

○ オーバヘッド極小

△ 複数のムービを同時再生する場合にのみ適用可能。

（３－２） GOP処理の並列化

GOPデコード処理を複数のプロセッサで分担する。

【特徴】

○ オーバヘッド極小

△ ClosedGOPに適用可能。OpenGOPには適用困難。

× デコード結果を保持するための巨大なメモリが必要。

× 負荷バランスを調整し難い。

（３－３） ピクチャ処理の並列化

ピクチャデコード処理を複数のプロセッサで分担する。

【特徴】

○ オーバヘッド極小

× ピクチャタイプによる処理順序の制約を受ける。

Bピクチャなしストリームは並列処理できない。

IPピクチャ周期(M)の数以上の並列処理はできない。

× 負荷バランスを調整し難い。

（３－４） スライス処理の並列化

スライスデコード処理を複数のプロセッサで分担する。
（各スライスは依存性が無く、並列デコード可能）

【特徴】

- ○ オーバヘッド小
- ○ 柔軟なCPU割り当て(SPUn個＋PPUn個)
- ○ 動的な処理分散

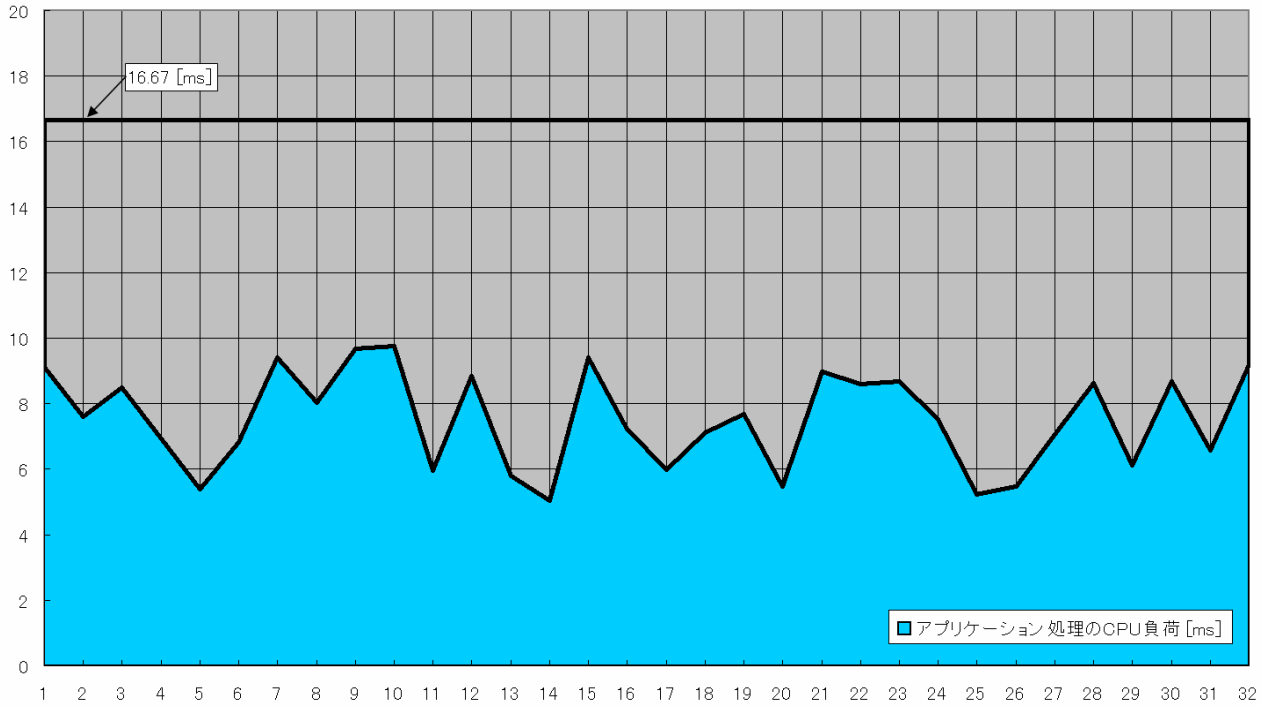
（4） 並列化の設計に応じた粒度の比較

並列化の設計		1 フレーム当たりの並列処理単位数 ※	粒度	オーバヘッド
ループ処理の並列化		ブロック処理の行数 =(1920/8)*1080 = 259200 ブロック数 = (1920/8)*(1080/8) = 32640	細	大きい
機能分割による並列化		マクロブロック数 = (1920/16)*(1080/16) = 8160		
圧縮データ階層構造による並列化	スライス処理	スライス数 = 1080/16 = 68		
	ピクチャ処理	ピクチャ数 = 1		
	GOP 処理	1/GOP ピクチャ数 ≦ 1	粗	小さい
	シーケンス処理	1/総ピクチャ数 ≒ 0		

※画像サイズ 1920×1080 で算出。

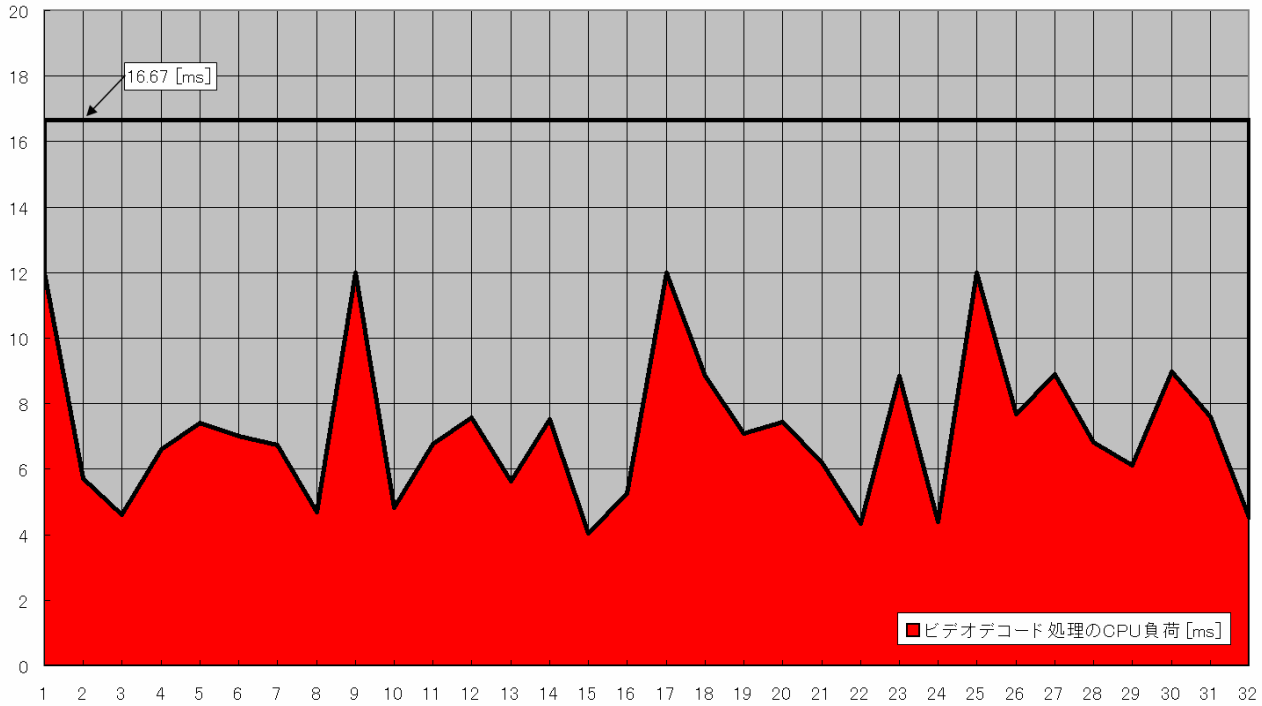
プロセッサ数は高々8個。これと比べて仕事を細分化しすぎるとバランス悪い。

CPU時間の有効活用(1)



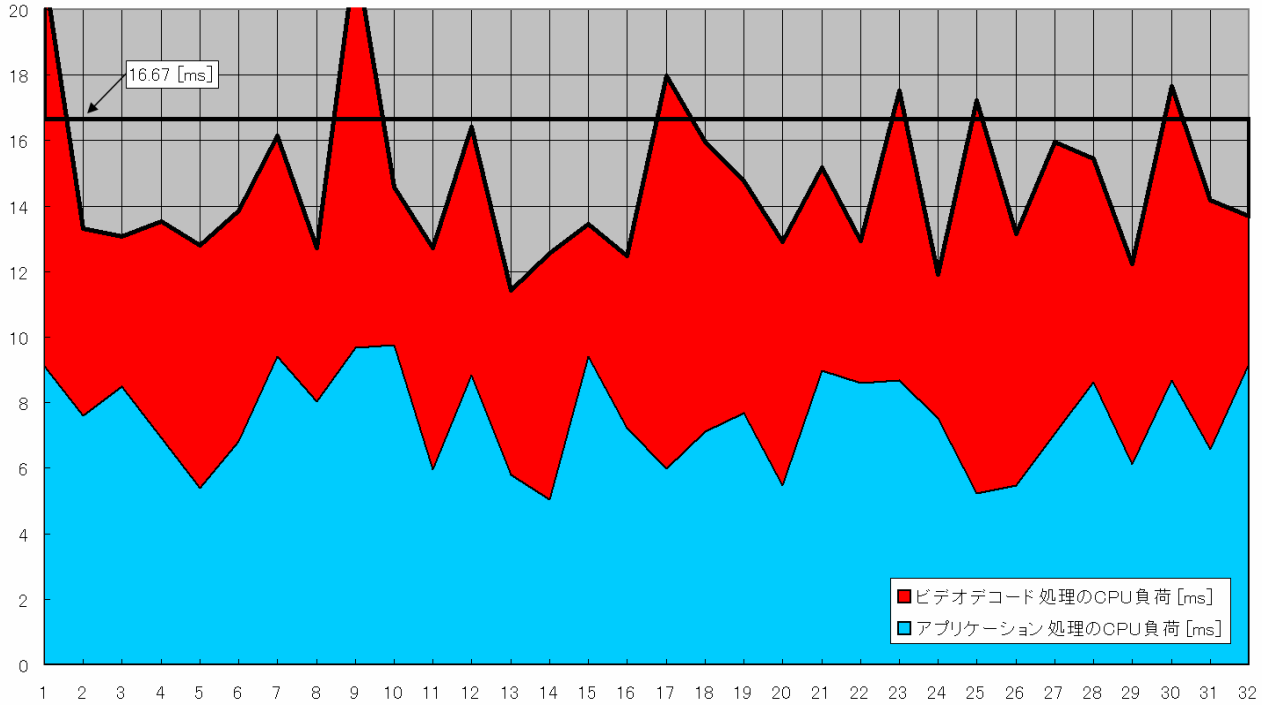
アプリケーションのCPU負荷は50%以下

CPU時間の有効活用(1)



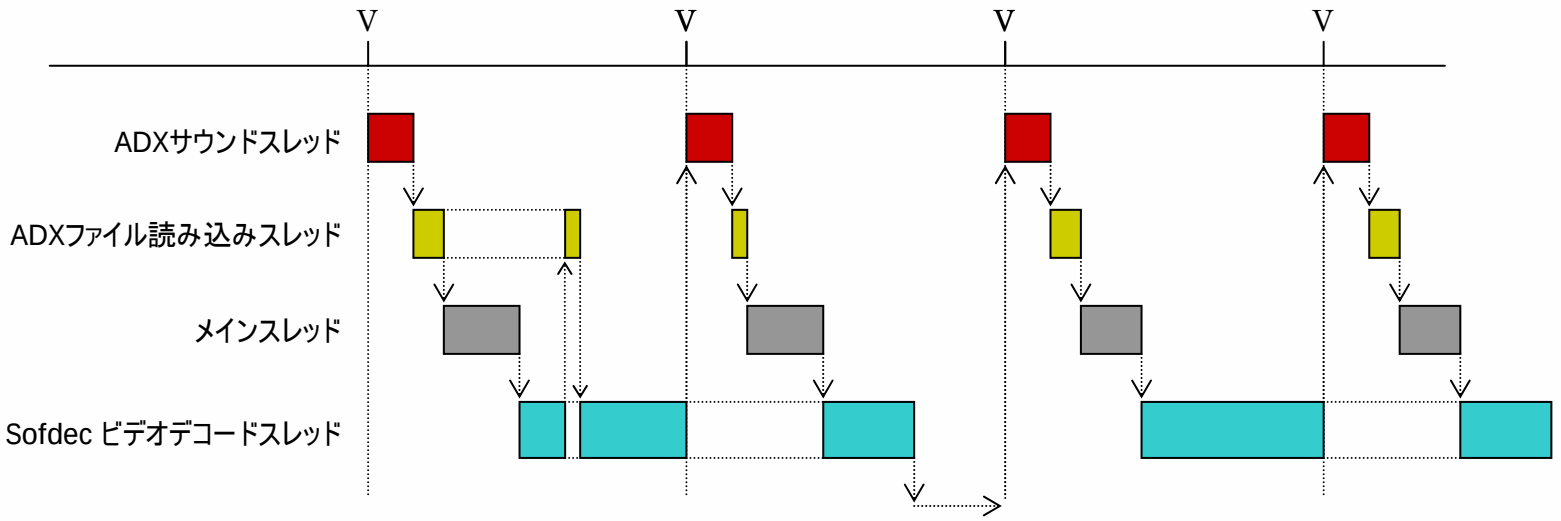
ビデオデコードのCPU負荷も50%以下

CPU時間の有効活用(1)



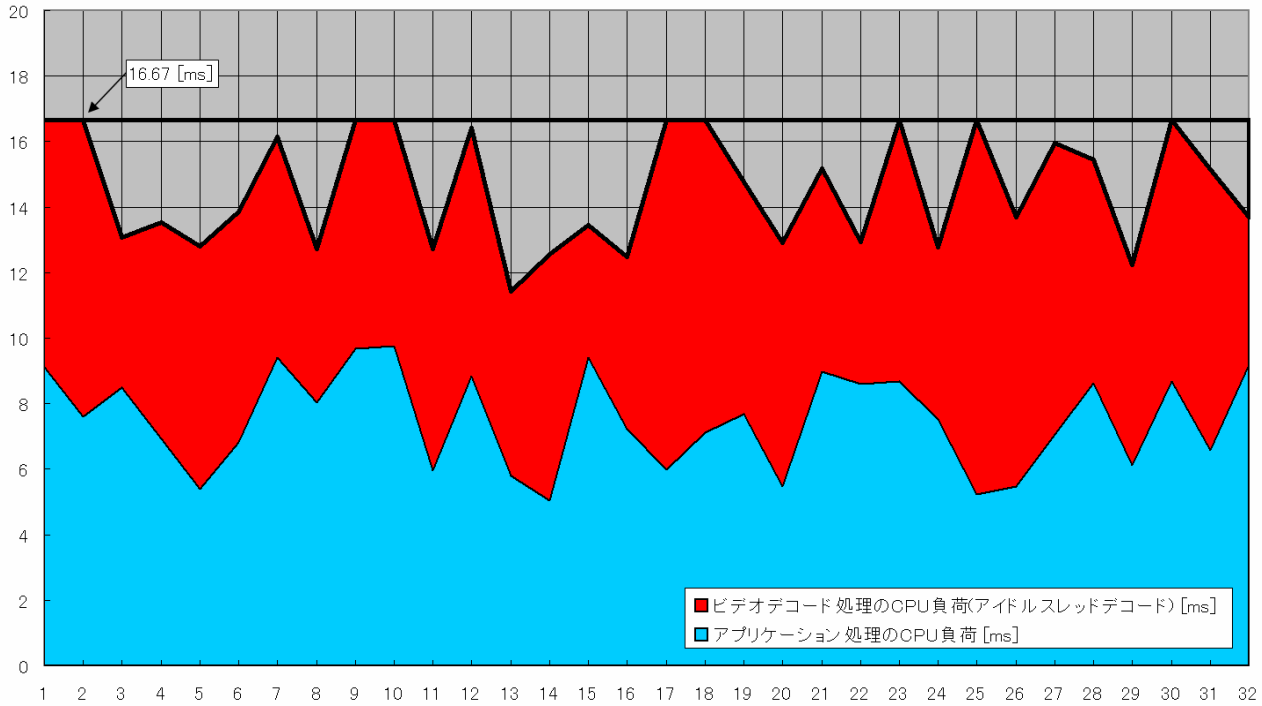
ところが、、、組み合わせるとコマ落ちが発生！

Sofdecのマルチスレッドアーキテクチャ



ADX サウンドスレッド	音声のデコード・出力、必ず1/60秒で動作
ADX ファイル読み込みスレッド	1/60間隔で動作開始、処理時間は不規則
メインスレッド	ほぼ1/60秒間隔、処理落ちも考慮
Sofdec ビデオデコードスレッド	CPUの空き時間でできる限りデコード

CPU時間の有効活用(1)



マルチスレッドアーキテクチャによって、コマ落ちなく、スムーズに再生する。

CPU時間の有効活用（1）

- 単純なシングルスレッドモデルは
  処理落ちを回避するためにピークの負荷を16msecに抑える。
  ピーク以外では空きCPU時間が無駄になる。
  →満足な動画性能が発揮できない。

- マルチスレッドモデルは
  アイドルスレッドでデコード。（低い優先度のスレッドで非同期デコード処理）
  フレームプールにデコード結果を貯金。（フレームの貯金 ≒ CPU時間の貯金）
  トータルのCPU時間が足りさえすればスムーズに再生する。
  トータルのCPU時間が足りないなら、デコードフレーム数を調整してAV同期を維持。
  → CPU時間を有効活用して最高の動画性能を引き出す。

CPU時間の有効活用（2）

- マルチプロセッサ環境では、アプリケーションや他のミドルウエアとうまく共存できるように、柔軟なCPU時間の割り当てに対応する。

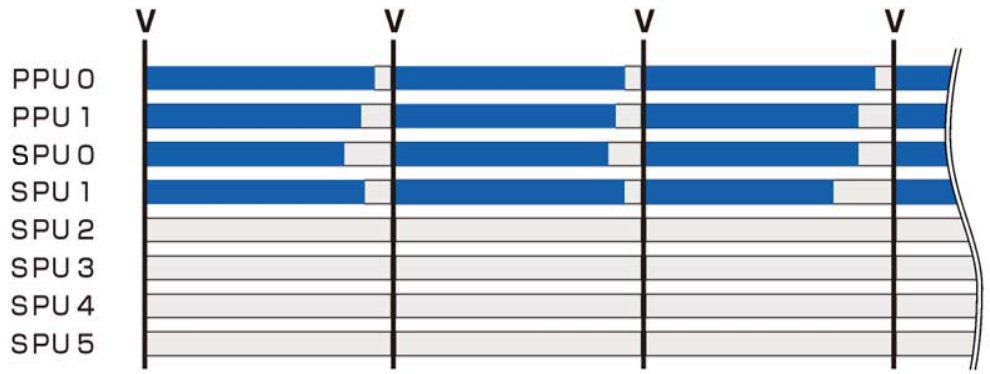
- PS3版Sofdecは、ビデオデコードに使用するプロセッサを次のように指定可能。

PPU　：　0～2個　（PPUハードウエアスレッド）
SPU　：　0～6個　（SPUスレッド）

アプリケーションや他のミドルウエアのCPU使用状況に応じて、空いているCPU時間をうまく割り当てる。

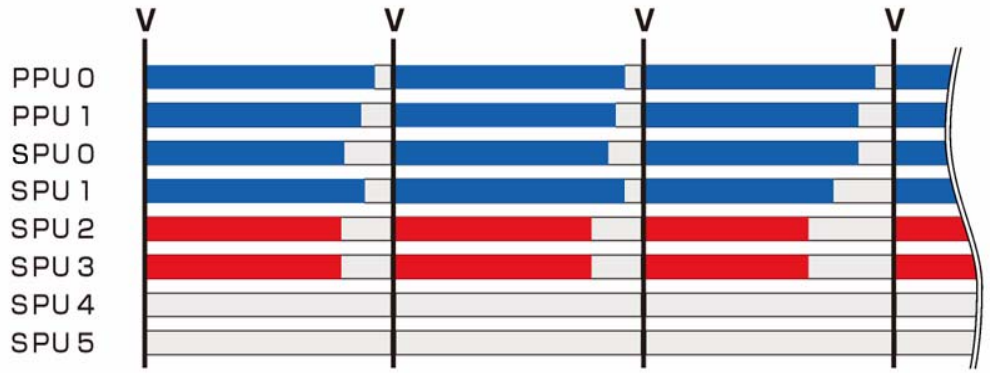
CellにおけるCPU割り当ての例（1）

- SPUが余っていれば、



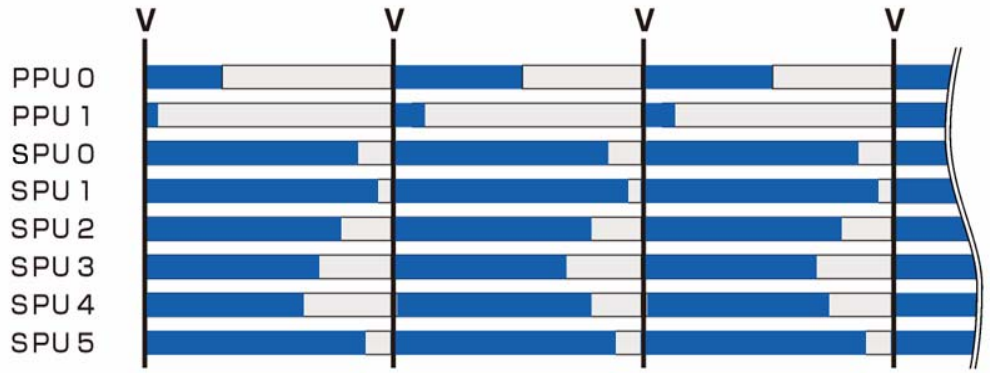
CellにおけるCPU割り当ての例（1）

- SPUが余っていれば、SPUだけでデコードさせる。



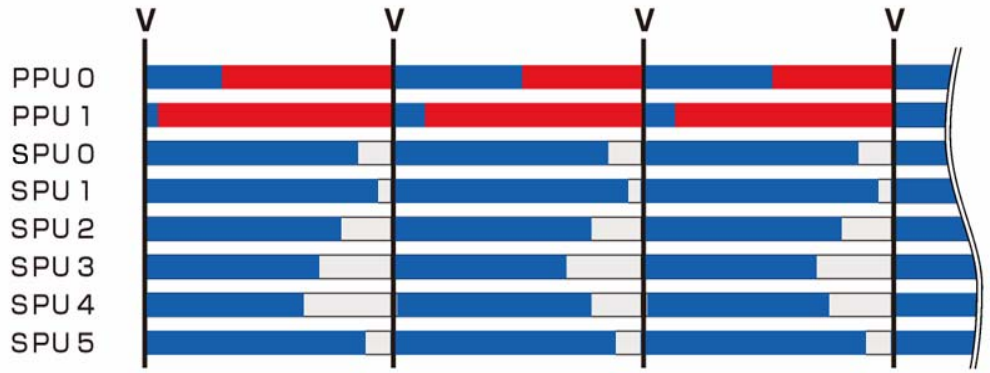
CellにおけるCPU割り当ての例（2）

- アプリケーションや他ミドルウエアがSPUを多用していて、



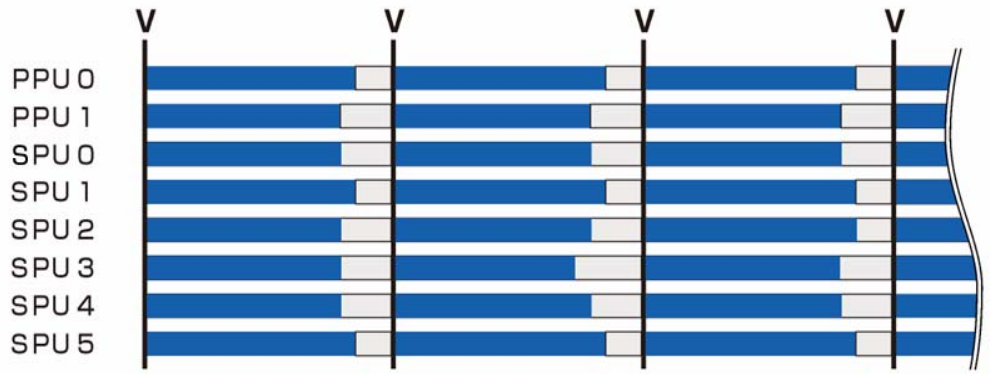
CellにおけるCPU割り当ての例（2）

- PPUに余裕があれば、PPUでデコードさせる。



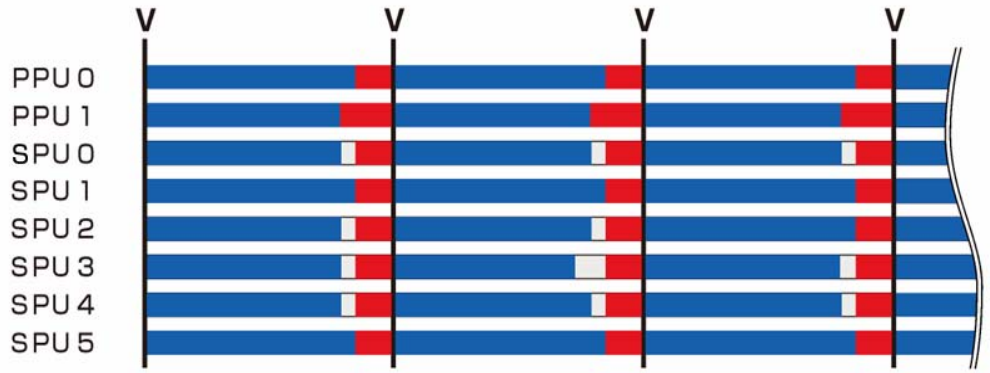
CellにおけるCPU割り当ての例（3）

- どのプロセッサも使用しているが少しずつ余り時間があるなら、



CellにおけるCPU割り当ての例（3）

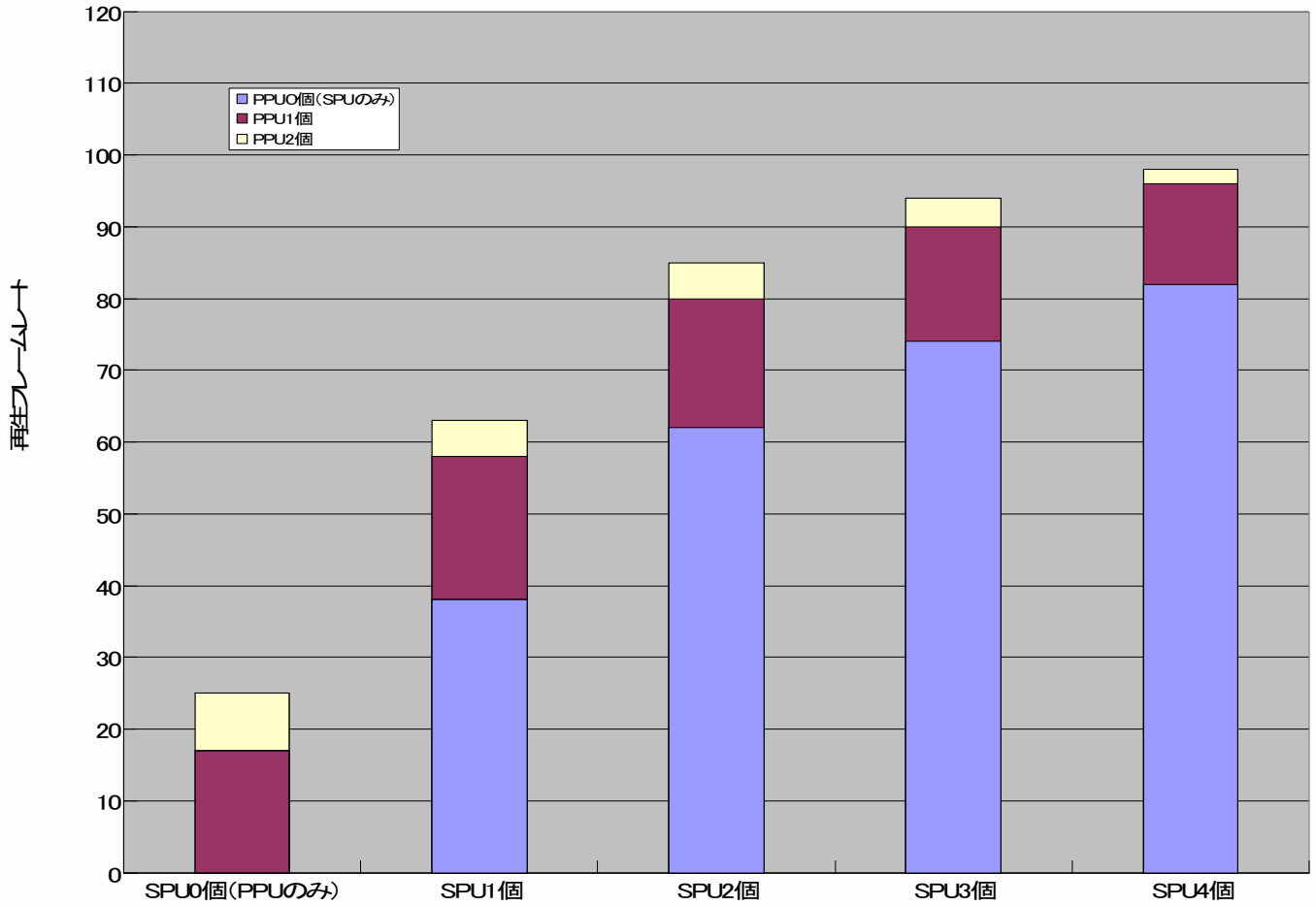
- SPUとPPUをフル稼働させ、極力短い時間で並列デコードさせる。



並列デコード処理のデモ

1920x1080, 60fps, Progressive, 36Mbps

並列処理の性能／効果



1920x1080, 60fps, Progressive, 36Mbps

並列処理の効果を妨げる要因

- 並列処理のために追加した処理
  - スライススタートコード検索
  - タスクリスト生成
  - SPUスレッド制御
  - 同期処理（終了待ち）
- 処理の並列化によって争奪の対象となるハードウエア資源
  - メモリ
  - バス
  - キャッシュ、演算器、パイプライン（PPUの場合）

SPU固有の高速化（1）

小さなメモリ

- ローカルストレージは僅か　256[Kbyte]
- フレームバッファは巨大

$$1920 \times 1080 \times 1.5 = 3[\text{Mbyte}]$$

SPU固有の高速化（2）

豊富な汎用レジスタ

16byte×128本

SPU固有の高速化（3）

DMAと演算処理の並列動作

- ビットストリームの読み込み
- フレームバッファの読み書き

SPU固有の高速化（4）

SIMD命令の適用

- IQ
- IDCTミスマッチ制御
- IDCT
- MC

SPU固有の高速化（5）

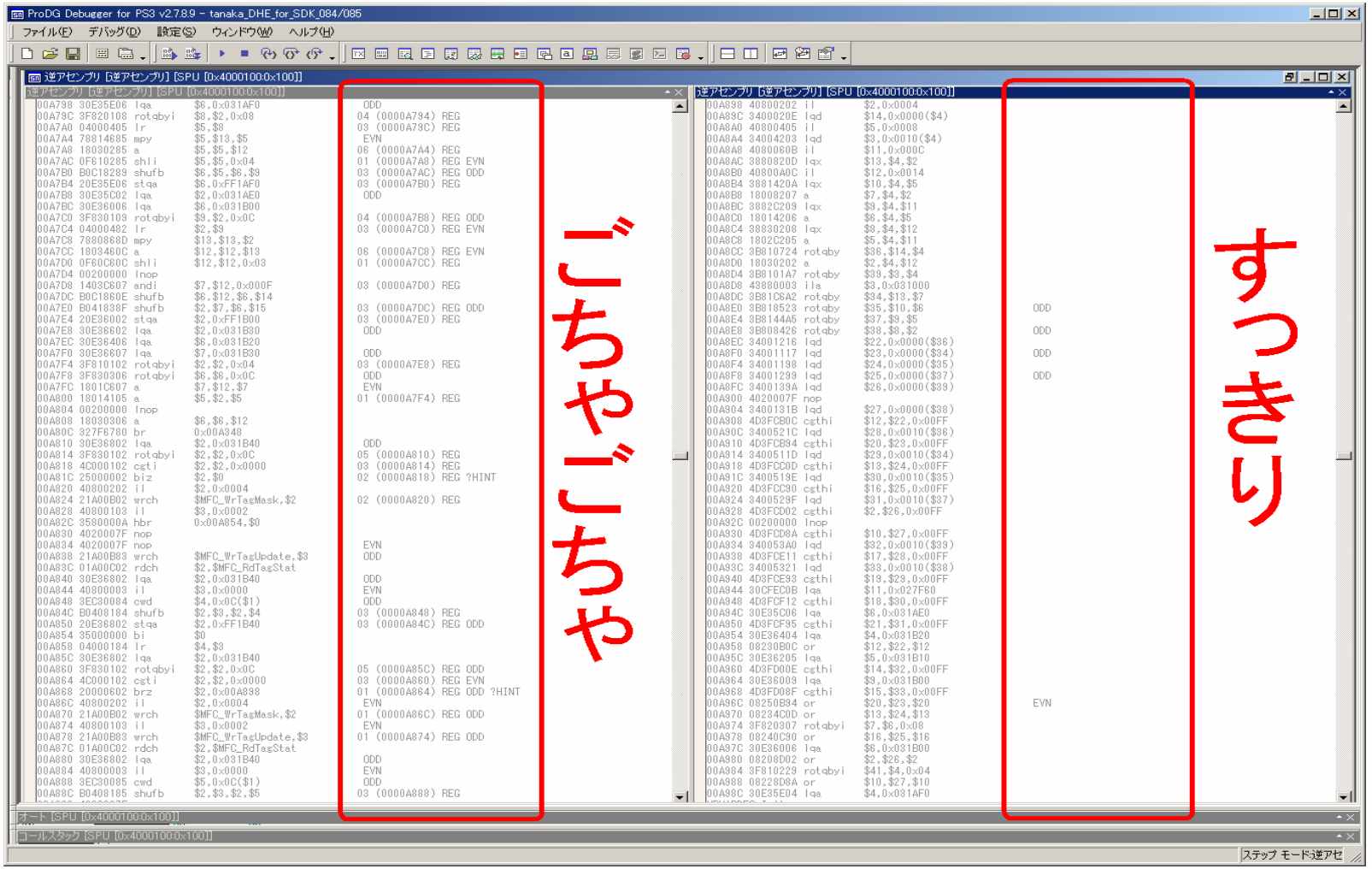
一般的な高速化手法も有効

- ループ展開
- インライン展開
- ソフトウエアパイプライン

SPU固有の高速化（6）

- 組み込み関数による、アセンブラ風コーディング
- 命令クラス、ODD命令／EVEN命令、
  レイテンシを意識して命令スケジュールする
- ODD／EVEN配置、NOP追加、分岐ヒント命令は
  コンパイラに任せる
- 逆アセンブル表示のパイプライン分析表示が参考になる

SPUパイプライン分析



SPU固有の高速化（7）

SPU命令セットの弱点

- ロード／ストア命令（128bitのみ）
- 飽和演算（4命令 ＝ cmpgt or cmpgt and）
- 32bit整数乗算（5命令 ＝ 16ビット乗算命令×3＋加算命令×2）
- レジスタ変数で配列データを扱いにくい

お問い合わせ先

- **Sofdec** : sofdec@cri-mw.co.jp
- **ADX** : adx@cri-mw.co.jp
- **ROFS** : rofs@cri-mw.co.jp
- **CRI Audio** : CriAudio@cri-mw.co.jp