リアルタイムビートトラッキングシステムの並列計算機への実装
— AP1000 によるリアルタイム音楽情報処理 —

後藤 真孝　　村岡 洋一
早稲田大学 理工学部

あらまし　本稿では，並列計算機の音楽情報処理への応用例として，音楽音響信号に対してリアルタイムにビートを認識するシステムの，並列計算機への実装について述べる．本システムでは質が異なる複数の処理を同時に実行する必要があるため，計算機全体でデータパラレル処理をおこなう実装は適切でない．本実装では，並列計算機を構成するプロセッサをグループに分け，各グループに対し，処理の質に応じた 4 種類の並列処理方式を適用する．そしてこれらのグループ間でデータを流し，計算機全体ではパイプライン状にデータを処理する．本システムは実際に並列計算機 AP1000 上でリアルタイムに動作し，30 曲中 27 曲に対してビートトラッキング可能であった．

Abstract　This paper presents parallel implementation of a real-time beat tracking system that processes musical acoustic signals and recognizes temporal positions of musical-beats. Parallel processing is necessary to perform this computationally-intensive task in real time. We apply four kinds of parallelizing techniques to execute heterogeneous processes simultaneously. The processes are first pipelined, and then each stage of the pipeline is implemented with data/control parallel processing, pipeline processing and distributed cooperative processing. Our system has been implemented on the Fujitsu AP1000. In our experiment, it correctly tracked beats in 27 out of 30 popular songs.

1　はじめに

ビートトラッキングとは，人間が音楽に合わせて手拍子を打つように，曲のビート (4 分音符) の位置を認識する技術である．ビートは西洋音楽を理解する上で基本的な概念の一つであり，ビートトラッキングの実現は，音楽聴取過程を計算機上で実現する第一段階として重要である．さらに，音楽音響信号からリアルタイムにビートトラッキングする技術は，多くのマルチメディアアプリケーションにおいても望まれている [1]．しかし，従来のビートトラッキングに関する研究の多くは，MIDI 信号か少数の楽器で演奏された音響信号を対象にしており [2]–[7]，ドラムスを含む複数の楽器で演奏された音響信号を扱うことはできなかった．

本研究で実現するシステムは，ロック・ポップスの主にドラムスがビートを刻む曲を対象とし，複数の楽器で演奏された音響信号に対してリアルタイムにビートを認識・予測する．主要な処理は，複数の手がかりを抽出する周波数解析と，ビートの様々な解釈の可能性を同時に追跡するビート予測である．これらの処理は計算量が多いため，リアルタイムに実行するには並列処理するのが適している．

従来の並列計算機の応用例の多くでは，計算機全体でデータパラレル処理をおこなっていた．しかし，本研究が対象とするビートトラッキングのように，質が異なる複数の処理を同時に実行する必要がある場合には，この実装法は適切でない．

本実装では多様な処理を並列に実行するために，まず並列計算機を構成するプロセッサを複数のグループに分け，各グループに対してシステムを構成する処理単位を直接割り当てる．割り当てられた処理の質はそれぞれ異なるため，その質に応じた異なる並列処理方式 (パイプライン処理，データパラレル処理，コントロールパラレル処理，分散協調処理) を各グループ内で適用する．そして，これらのグループ間でデータを送受信することで，計算機全体としてはパイプライン状にデータを処理する．

我々は，本システムを富士通の分散メモリ型並列計算機 AP1000 (64 プロセッサ構成) 上に実装した．本実装では，あるプロセッサグループが同期をとっているときに，別のグループが処理を続ける必要がある．しかし，AP1000 の OS が提供する S-Net によるバリア同期は，すべてのプロセッサで明示的に同期をとる必要があるため利用できない．そこで，同期が不要なプロセッサは処理を続けることができるように，ステータスハードウェアを利用して時間確認可能な同期機構を実現した．

実装したシステムに対し，市販の Compact Disc (CD) の音響信号を入力して実験した結果，30 曲中 27 曲に対して正しいビートをリアルタイムにトラッキングできた．実装したシステムのターンアラウンドタイムは 127.71msec であったが，常に予測結果 (次のビートの時刻) を出力するというビートトラッキングの処理の性質から，この遅延時間は問題にならなかった．

Parallel Implementation of A Real-time Beat Tracking System — Real-time Musical Information Processing on AP1000 —　Masataka Goto and Yoichi Muraoka
School of Science and Engineering, Waseda University

2　処理モデル

本システムは，市販の CD などから得た複数の楽器音を含む音響信号を入力とし，4 分音符に相当するビートを認識する．その際，ビートの存在する時刻 (ビート時刻) を認識するだけでなく，そのビートが強拍か弱拍か (ビートタイプ) も判断する．そして，ビート時刻，ビートタイプ，現在のテンポの三つから成るビート情報を，入力された音楽に合わせて出力する．

本システムは，入力に対して以下の仮定をおこなう．

- 入力曲は 4/4 拍子であり，テンポは 70〜180 M.M. (4 分音符/分) の間で，曲中を通じてほぼ一定である．
- バスドラム (BD) が強拍 (1，3 拍目)，スネアドラム (SD) が弱拍 (2，4 拍目) で鳴る確率が高い．

これらの仮定は，ロック・ポップスの多くの曲に当てはまる．

このような音響信号に対するビートトラッキングを実現するには，主に次のような課題がある．(1) 音響信号波形にはビートに直接関係のないエネルギーピークが多数あるため，ピーク検出した後に閾値処理をする単純な手法では，ビートをとらえることはできない．(2) 対象とする音響信号には様々な楽器音が混在しており，MIDI の場合には容易に得られる各音の発音時刻を，正確に求めることは困難である．(3) ビートは音楽に対して人間が知覚する概念であり，ビートが実際の信号に直接対応するとは限らない．音響信号がないところにビートが存在する場合もある．このような曖昧さがあるためにビートを一意に解釈することはできず，常に複数の解釈の可能性がある．(4) 4 分音符の長さやビートタイプの判断は音楽的知識を必要とし，一般には難しい．

以上を解決するために，複数の手がかりを抽出し，複数の解釈の可能性を同時に追跡する．まず，ビートに直接対応する特定の手がかりがないため，周波数解析で複数の手がかりを抽出する．具体的には，各周波数帯域ごとの発音時刻と，主要なドラム音 (BD と SD) の発音時刻を検出する．次に，ビート予測において，複数のエージェントがビートの様々な解釈の可能性を並列に調べる．各エージェントは，発音時刻をそれぞれ異なった戦略により解釈し，次のビートの時刻を予測した後に，その解釈の確信度を自己評価する．最終的に出力するビートの時刻は，最も確信度の高いエージェントの解釈に基づいて決定する．これにより，あるエージェントの解釈がはずれても，他のエージェントが正しく解釈している限り，ビートを見失わずにトラッキングできる．また，BD と SD の出現位置から，4 分音符の長さを推定しビートタイプを判断する．

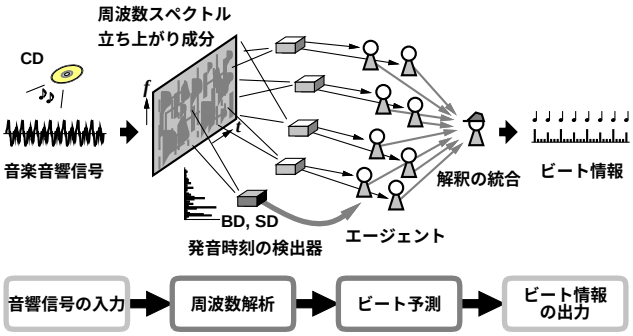


図 1: 処理モデル

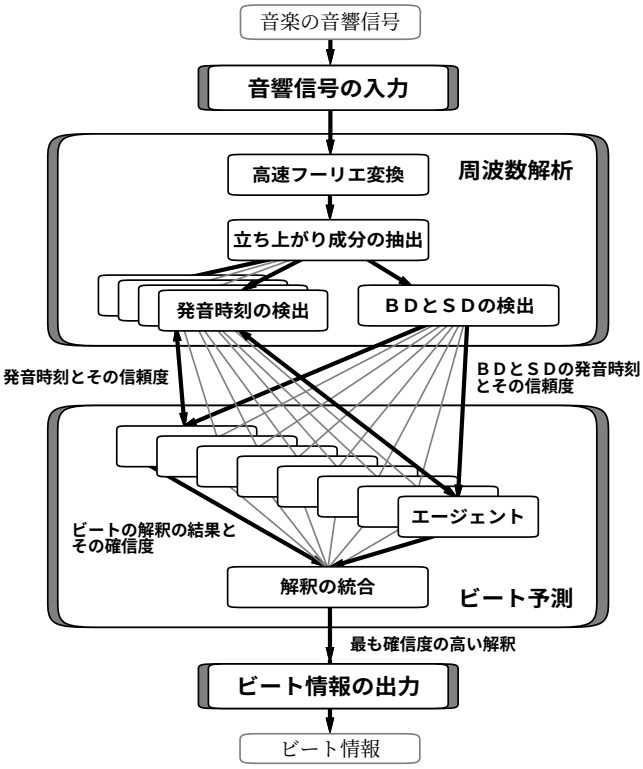


図 2: 処理の流れ

本システムの処理モデルの概念図を図 1 に示し，処理の流れを図 2 に示す．まず，**音響信号の入力**で A/D 変換された音響信号に対して，**周波数解析**をおこない，各周波数帯域ごとの発音時刻と，BD と SD の発音時刻を検出する．これは，様々な視点から検出する複数の発音時刻の検出器がおこなう．次に，**ビート予測**の各エージェントが，過去に得られた発音時刻からビートの間隔を求め，次のビート時刻の予測とビートタイプの判断をおこなう．そして，全エージェントの解釈を統合してビート情報を生成する．最後に**ビート情報の出力**が，ネットワークを通じて他のアプリケーションプログラムへとビート情報を送信する．

以下では，主要な処理である周波数解析とビート予測について述べる．

2.1 周波数解析

複数の発音時刻の検出器が様々な視点から手がかりを抽出する．まず，高速フーリエ変換で得られた周波数スペクトルに対し，立ち上がり成分の抽出をおこなう．この結果に対し，発音時刻の検出器[1]が，様々な感度・周波数帯域において発音時刻を検出する．また，これ以外にBDとSD専用の発音時刻の検出器もある．

以上の発音時刻の検出結果は，ビート予測の各エージェントへと送られる．

2.1.1 高速フーリエ変換(FFT)

A/D変換された音響信号に対してFFTをおこない，周波数スペクトル(パワースペクトル)を得る．これにより，各周波数成分のパワーの時間変化が得られる．観測区間(WindowSIZE)を時間軸方向に単位時間(ShiftSIZE)ずつずらしながらFFTを適用する[2]．

2.1.2 立ち上がり成分の抽出

スペクトルのパワーが確実に増加し続けている周波数成分を，立ち上がり成分として抽出する．多くの場合，発音したときの鳴り始めの周波数成分がこの抽出で得られる．

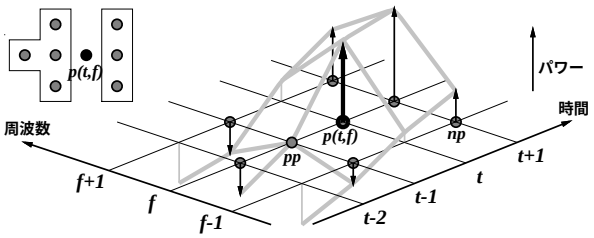


図 3: 立ち上がり成分の抽出

FFTの結果に対して，式(1)を満たす周波数成分$p(t,f)$を，立ち上がり成分とする(図3)．ただし，$p(t,f)$は時刻t，周波数f[3]におけるスペクトルのパワーとし，ppとnpは式(2)，式(3)で与えられるものとする．

$$\begin{cases} p(t,f) & > & pp \\ np & > & pp \end{cases} \quad (1)$$

$$pp = \max(p(t-1,f), p(t-1,f\pm1), p(t-2,f)) \quad (2)$$

$$np = \min(p(t+1,f), p(t+1,f\pm1)) \quad (3)$$

$p(t,f)$が立ち上がり成分であるとき，その立ち上がりの度合$d(t,f)$は，式(4)により求める．

$$d(t,f) = \max(p(t,f), p(t+1,f)) - pp \quad (4)$$

[1]現在の実装では，発音時刻の検出器の数は15個である．

[2]現在の実装では，WindowSIZEは1024点(46.44msec)，ShiftSIZEは256点(11.61msec)であり，周波数分解能は21.53Hz，時間分解能は11.61msecになる．

[3]ただし，t，fは整数値をとり，$1t$，$1f$はそれぞれ時間分解能，周波数分解能に相当する．

2.1.3 発音時刻の検出

複数の発音時刻の検出器が，それぞれ異なるパラメータにより発音時刻を求める．各検出器は，**2.2**で述べるエージェントの組に対応しており，検出結果を対応する組へ送信する(図1, 図5)．

発音時刻は以下のように得る．まず，各時刻における立ち上がり成分の合計値$D(t)$を式(5)により求め，その時間変化のピーク時刻とピーク値を，平滑化微分を用いたピーク検出により求める[4]．こうして得られたピーク時刻を発音時刻とする．

$$D(t) = \sum_f d(t,f) \quad (5)$$

それぞれの検出器は，検出感度，検出周波数帯域という二つのパラメータを持つ．検出感度は，平滑化微分における平滑化の時間幅であり，平滑化幅が小さいほど感度が高くなる．検出周波数帯域は，式(5)中の$\sum$における周波数帯域の指定で，これにより制限された帯域の発音時刻を得ることができる．

2.1.4 BDとSDの検出

立ち上がり成分の抽出結果から，現在の曲のBDとSDの音に対応する特徴周波数を自動的に獲得し[5]，これらの発音時刻を検出する．立ち上がり成分が検出された時刻において，その周波数軸方向のピーク周波数がBDかSDの特徴周波数に一致した場合，BDまたはSDがその時刻に発音したと判断する．

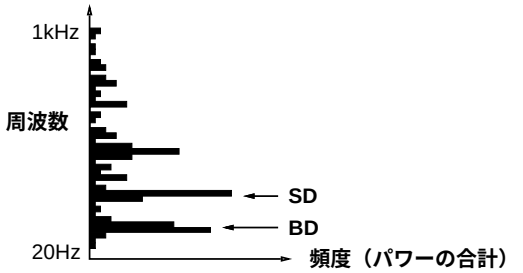


図 4: ヒストグラム中のBDとSDの特徴周波数

特徴周波数は以下のように獲得する．まず，立ち上がり成分が存在する時刻において，周波数軸方向のピークを求め，そのヒストグラムを作成する(図4)．その際，ヒストグラムには各ピークの立ち上がりの度合$d(t,f)$を加える．そして，ヒストグラム中で最も周波数の低いピークをBD，それより周波数が高く最もパワーの大きいピークをSDの特徴周波数とみなす．

[4]SavitzkyとGolayの2次多項式適合による平滑化微分を用いたピーク検出により，ノイズの影響を平滑化で減らした後に極大値を与える時刻を検出する．

[5]一般に，これらのドラム音の音色は曲ごとに異なるため，特徴周波数を事前に与えることはできない．

2.2 ビート予測

周波数解析の結果を，複数のエージェントがそれぞれ異なった戦略により解釈し，次のビート時刻を予測する．各エージェントが出力する解釈の結果は，次のビート時刻，そのビートタイプ，ビートの間隔の三つから成る (図 5)．この結果は，解釈の統合へと集められ，最も確信度の高い解釈が選択される．

すべてのエージェント[6]は，二つずつの組に分けられる．同じ組の二つのエージェントは協調し合いながら，両者が同じビートの間隔で，互いにビートの間隔の 1/2 ずれた時刻を予測する．これにより，一方が 4 分音符の裏拍を予測し始めても他方が正しい表拍を予測できる．各エージェントの組は，**2.1.3** で述べたように，それぞれ対応する検出器から発音時刻を受けとる (図 5)．

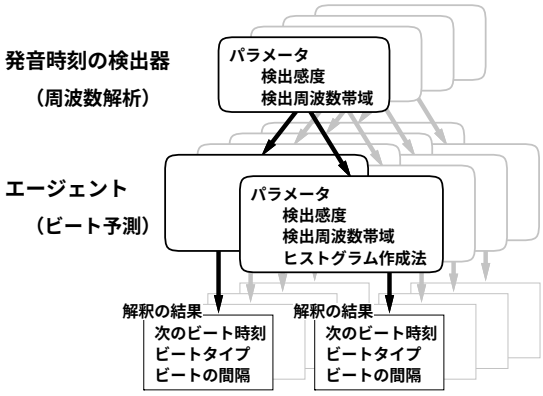


図 5: エージェントと発音時刻の検出器

エージェントは，ビートを解釈する際の戦略を変更する三つのパラメータ (検出感度，検出周波数帯域，ヒストグラム作成法) を持つ．パラメータの値の設定は，各エージェントの組ごとに異なり，同じ組の二つのエージェントだけが同じ値をとる．検出感度と検出周波数帯域は，発音時刻の検出器内の対応するパラメータを制御し，エージェントが受けとるビートの手がかりの質を調整する．ヒストグラム作成法は，発音時刻の間隔 (IOI: inter-onset-interval) のヒストグラムを作成する方法を変更する．またエージェントは，自己評価した確信度が長期間低い場合に，これらのパラメータを自己修正する機能も持つ．

2.2.1 エージェントの動作

各エージェントはビートの間隔を求めて次のビート時刻を予測し，解釈の確信度を自己評価する．そして，予測したビートに対してビートタイプを判断し，解釈の確信度を補正する．

[6]現在の実装では，エージェントの数は 30 個である．

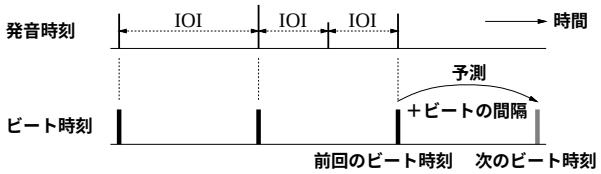


図 6: 次のビート時刻の予測

次のビート時刻の予測は，前回のビート時刻にビートの間隔を加えておこなう (図 6)．ビートの間隔は，IOI のヒストグラムの最大ピークから求める．もし前回のビート時刻とほぼ一致する発音時刻がある場合には，その発音時刻にビートの間隔を加える．

各エージェントは，以下のように解釈の確信度を自己評価する．過去に予測したビート時刻に発音時刻が一致していれば，確信度を上げる．また，過去に予測したビート時刻の間の 8 分音符や 16 分音符に相当する時刻に，発音時刻が一致している場合にも，確信度を少し上げる．これらに一致しない場合には，確信度を下げる．

次に，予測した各ビート時刻に対し，BD と SD の発音時刻の検出結果からビートタイプを判断する．これらがある程度交互に鳴るという性質を用いて，次のビートに BD と SD のどちらが予測されるかを判断する．もし次のビートが BD だと予測すれば，そのビートタイプを強拍とし，SD の場合には弱拍とする．そして，ビート時刻にこれらがある程度交互に検出されれば，その解釈の確信度を上げ，4 分音符に相当するビートの間隔を持つ解釈が選択されるようにする．

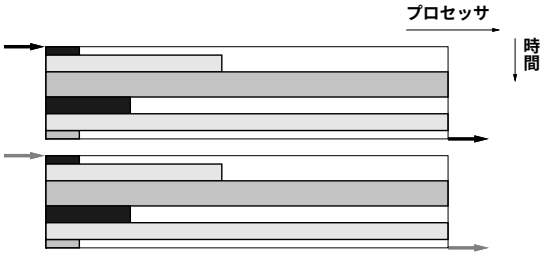
2.2.2 解釈の統合

すべてのエージェントから集められた解釈の結果は，ビート時刻とビートの間隔が同じもの同士，グループ分けされる．各グループにおいて，グループ内の全解釈の確信度を合計し，そのグループの確信度とする．そして，最も確信度の高いグループ内の最も確信度の高い解釈を選択する．

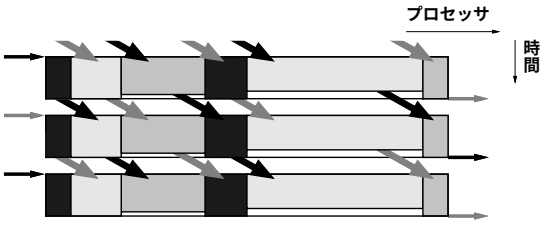
3 並列化手法

2 で述べた処理モデルから，質が異なる多様な処理を，並列計算機上でリアルタイムに実行する必要があることがわかる．このような処理を並列化するには，(a) 計算機全体で時分割処理する手法 と，(b) 計算機全体でパイプライン処理する手法 が考えられる (図 7)．ここでは，手法 (a) の問題点を指摘し，本実装で用いた手法 (b) について述べてゆく．

並列計算機全体で時分割処理する場合には，図 2 の各処理を順番に切替えて実行する．しかし，並列度の低い処理によりアイドル状態になるプロセッサが増



(a) 計算機全体で時分割処理



(b) 計算機全体でパイプライン処理

図 7: 並列化手法の比較

えると共に，処理の切替えのオーバヘッドも大きくなる．具体的には，図 2 において，BD と SD の検出，解釈の統合などの処理は並列度が低く，これらを実行している間は，多くのプロセッサがアイドル状態となる (図 7 (a))．また，各処理を切替える際に，処理間の通信が終了するのを毎回待つ必要があるため，処理間の通信時間のすべてが常に全体の実行時間に影響する．

一方，並列計算機全体でパイプライン処理する場合には，プロセッサを複数のグループに分け，各グループに対して図 2 の各処理を割り当てる．そして，図 7 (b) のように，パイプラインの各ステージにグループを対応させ，グループ間でデータを流すことでパイプラインを形成する．各グループへその処理の並列度に応じてプロセッサを割り当てることで，アイドル状態となるプロセッサを削減できる．各プロセッサは，常に割り当てられた同種の処理を実行するので，処理の切替えのオーバヘッドも小さい．また，手法 (a) では別々におこなわれていた処理間の通信が一度におこなわれるため，最も時間のかかる通信だけが全体の実行時間に影響し，他の通信時間は隠蔽できる[7]．時間のかかる通信をおこなう処理が，より早い時刻から送信することができれば，全体の実行時間への影響はさらに小さくなる．

本システムの並列化では，手法 (b) により計算機全体でパイプライン処理するだけでなく，さらに異なる並列処理方式を組み合わせる．パイプライン中の 4 つのステージにおいて，それぞれの処理の質に応じてパイプライン処理，データパラレル処理，コントロールパラレル処理，分散協調処理を適用する (図 8)．

[7] この場合，通信ネットワークへの負荷は手法 (a) よりも大きくなるが，送信開始時刻をずらしたり，近接通信を多くすることで，通信時間の増加を抑えることができる．

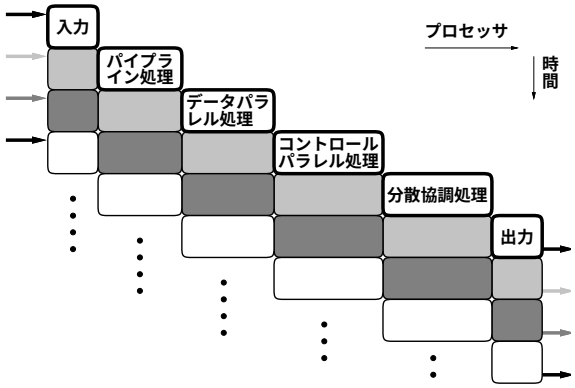


図 8: 異なる並列処理方式を組み合わせたパイプライン処理

以下では，これら 4 つの並列処理方式について順番に述べる．なお，現在の実装は並列計算機 AP1000 に対しておこなっており，各ステージのプロセッサ数は，総プロセッサ数 (64 台) を考慮して決められている．

3.1 パイプライン処理 (FFT)

FFT(**2.1.1**) を 5 台のプロセッサがパイプライン処理する．FFT 自体は逐次処理でおこなうが，各プロセッサが単位時間 (ft: フレーム時間) ずつずれて FFT を実行することで，パイプライン状に処理する．現在の実装では 1ft は 11.61msec であり，1 回の FFT(1024 点) の計算はおよそ 4.2ft かかる．そこで，5 台のプロセッサが FFT をおこなうことにより，毎 ft ごとに結果を得ることができる (図 9)．

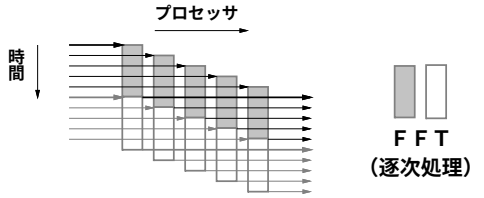


図 9: パイプライン処理 (5 プロセッサ)

3.2 データパラレル処理 (立ち上がり成分の抽出)

立ち上がり成分の抽出 (**2.1.2**) を 8 台のプロセッサがデータパラレル処理する．周波数スペクトルの全周波数帯域を 8 等分してプロセッサへ割り当てる．そして，各プロセッサは自分の担当する帯域の立ち上がり成分を抽出する (図 10)．

この処理は DOALL 型ループで実現でき，全プロセッサでデータパラレル処理することも可能である．今回は 1ft 以内に処理が終了すれば十分であるので，8 台のプロセッサで並列化した．

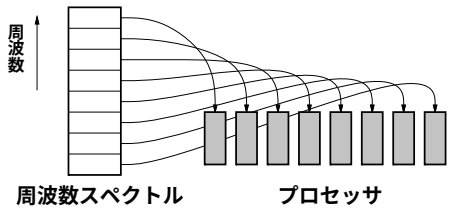


図 10: データパラレル処理 (8 プロセッサ)

3.3 コントロールパラレル処理 (発音時刻の検出・BD と SD の検出)

発音時刻の検出 (**2.1.3**) 及び BD と SD の検出 (**2.1.4**) を，16 台のプロセッサがコントロールパラレル処理する．実装した処理モデルでは，15 個の通常の発音時刻の検出器と 1 個の BD と SD 専用の検出器があり，これらを各プロセッサに一つずつ割り当てる (図 11)．そして，立ち上がり成分の抽出結果という同一のデータに対し，16 台のプロセッサが，異なるアルゴリズムやパラメータで別々に発音時刻を検出する．この各プロセッサの処理は 1ft 以内に終了する．

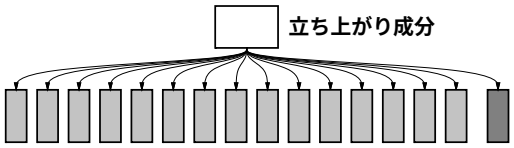


図 11: コントロールパラレル処理 (16 プロセッサ)

3.4 分散協調処理 (エージェント・解釈の統合)

ビート予測のエージェント (**2.2.1**) 及び解釈の統合 (**2.2.2**) を，31 台のプロセッサが分散協調処理する．

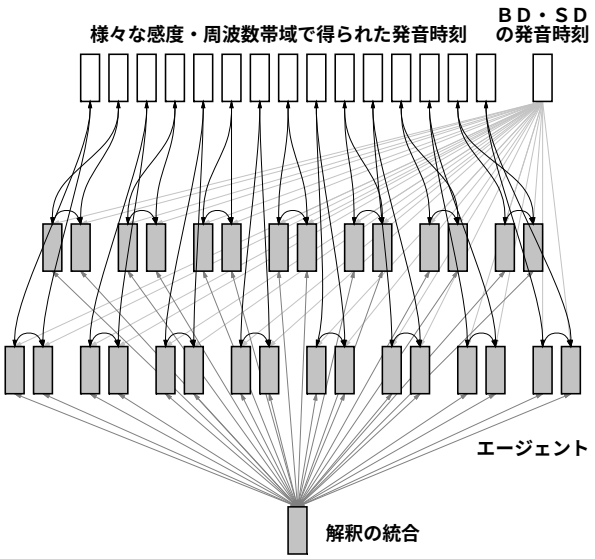


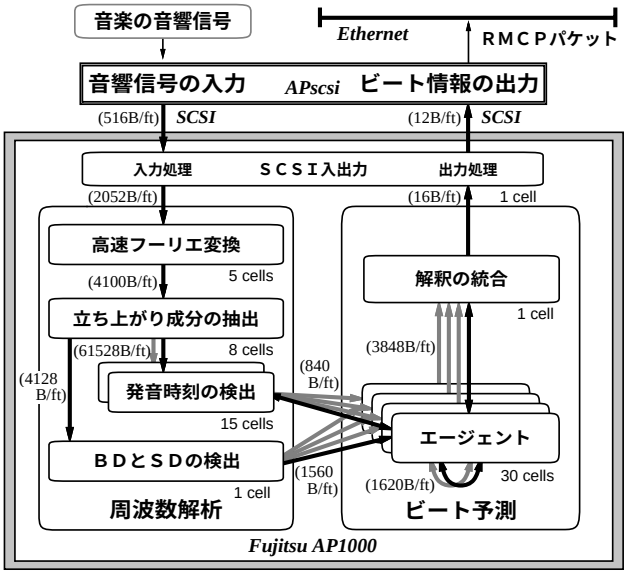
図 12: 分散協調処理 (31 プロセッサ)

実装した処理モデルでは，30 個のエージェントと 1 個の解釈の統合処理があり，これらを各プロセッサに一つずつ割り当てる (図 12)．

2.2 で述べたように，同じ組の二つのエージェント (プロセッサ) は，互いの解釈を通信し合い，協調して次のビート時刻を予測する．また各エージェントは，確信度が長期間低い場合にパラメータを自己修正する機能を持つが，これは他のエージェントの現在のパラメータを受信し，その状態を考慮しておこなう．具体的には，他のエージェントのパラメータとは異なる値で，最も確信度の高いエージェントのパラメータに近付くように，パラメータの値を調整する．

4 AP1000 への実装

3 で述べた並列化手法により，本システムを富士通の分散メモリ型並列計算機 AP1000 上に実装した (図 13)．実装した AP1000 は，64 台のセルと呼ばれる要素プロセッサで構成される．本実装では，これらのセルを図 13 のように 7 つのグループに分けた．これらのグループが，計算機全体のパイプラインを構成する．グループを表す長方形の右下に，そのグループに割り当てられたセルの数を示す．矢印はグループ間の大局的なデータの通信を表しており，主に入力から出力へと U 字型にデータが流れる．実際の通信量を，矢印の横に示す．



B/ft = bytes / frame-time　1 ft = 11.61 msec　1 B/ft = 86.13 bytes/sec

図 13: AP1000 への実装

4.1 入出力

音響信号の入力とビート情報の出力は，セルの 1 つと SCSI によって接続された入出力用ワークステーショ

ン (Sun SPARC Station 10: 以下 APscsi と呼ぶ) によりおこなう (図 13). この SCSI 接続をおこなうために, セルの一つに SCSI インタフェースを持つオプションハードウェア (富士通 DDV ボード) を追加した.

4.1.1 音響信号の入力

APscsi は音響信号を 22.05kHz, 16bit, モノラル (1 channel) で A/D 変換する. この結果を 256 点ごとにブロック化して, SCSI 入出力をおこなうセルへ送信する. この 1 ブロックに相当する時間 (11.61msec) を 1 フレーム時間 (ft) と呼び, ft を AP1000 内のすべての処理の時間単位とする.

4.1.2 ビート情報の出力

APscsi は, SCSI 入出力をおこなうセルからビート情報を受信し, RMCP(Remote Music Control Protocol) パケットとして, 予測されたビート時刻に Ethernet 上へ送信 (ブロードキャスト) する. RMCP は, MIDI と LAN を融合した分散協調システム (RMCP システム) におけるサーバ・クライアント間の通信プロトコルであり, シンボル化された音楽情報をネットワークを通じて伝送するために設計された [8].

RMCP パケットとしてブロードキャストすることにより, アプリケーションの実装が容易になるだけでなく, 複数のアプリケーションでビート情報を同時に利用できる. 既に, 音楽に合わせてドラム音や手拍子の音を出力したり, 音楽に同期した CG(コンピュータグラフィックス)[8] を生成する数種類の RMCP サーバを実装した.

4.2 時間管理のための同期機構

計算機全体のパイプライン処理と主要なデータ通信のタイミングを図 14 に示す. この時間管理は, APscsi が A/D 変換する際に, 一定個数 (256 点) のデータが揃うまでブロックすることを利用しておこなう. AP1000 側は, APscsi から転送されてきたデータによるデータ駆動方式で動作し, 1 ブロックを受信する度に同期をとる (11.61msec 間隔). 図 14 の各横線が同期をとる時刻を表す. この同期は, APscsi からデータを受信する時刻に, 各セルの処理のタイミングを合わせるためにおこなう.

本実装では, あるプロセッサグループが同期をとっているときに, 同期が不要な別のグループは処理を続ける必要がある. しかし, AP1000 の OS が提供する S-Net によるバリア同期は, すべてのセルで明示的に

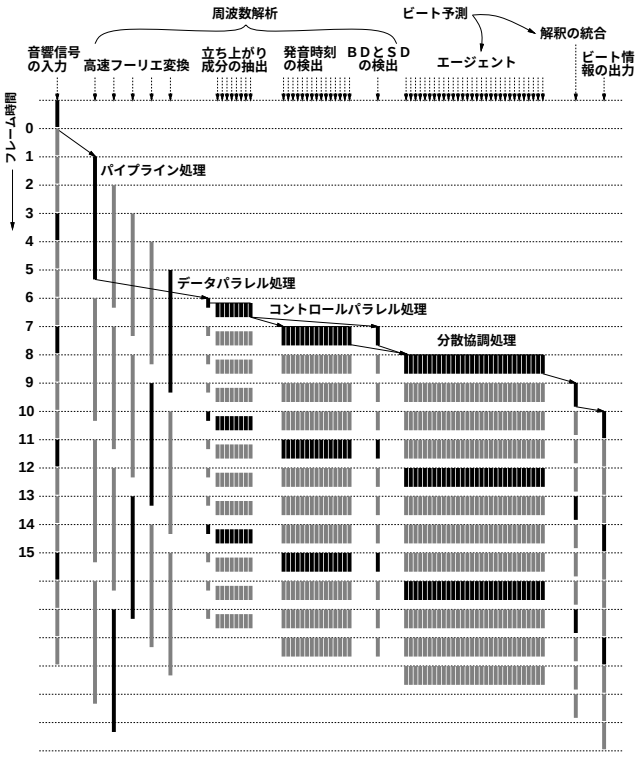


図 14: タイミングチャート

同期をとる必要があるため利用できない. そこで, ステータスハードウェアを利用して時間確認可能な同期機構を実現する.

S-Net は, バリア同期をおこなう同期ハードウェア以外にステータスハードウェアを備えている. これは, 全セルからの 8 ビットのステータスの論理和をハードウェアで求め, その結果を各セルが参照できる機構である[9]. そこで, この 8 ビットのうち上位 3 ビットを本来のステータス用 (0〜7), 下位 5 ビットを同期用ステータス (0〜31) として割り当てる.

同期は以下のようにおこなう. SCSI 入出力をおこなうセルがマスターとなり, APscsi からデータを受信する度に同期用ステータスをインクリメントする. 他のすべてのセルはスレーブとなり, 同期用ステータスの変化を検出して同期をおこなう. こうして同期をとる必要のあるセルだけが変化を監視して同期し, 必要のないセルはそのまま処理を続けることができる. 例えば FFT をおこなうセルは, 同期用ステータスを 5ft ずつ飛ばして確認する.

さらに本同期機構では, 同期用ステータスの値が予期していた値よりも進んでいると, 自分の処理が遅れていることがわかる. しかし, 現在の実装では, AP1000 の処理能力は本システムの計算負荷よりも十分に高く, 正常動作しているときに処理が遅れること

[8] 本システムを応用することで, 音楽のビートに合わせて踊る仮想の CG ダンサーの映像をリアルタイムに表示するシステムを実現した.

[9] 実際のハードウェアでは論理積を演算している. $x \vee y = \neg(\neg x \wedge \neg y)$

はない．そのため，処理が通常より遅れた場合には，異常が起きたと判断してシステムは停止する．

4.3 ターンアラウンドタイム

現在の実装では，AP1000 全体のパイプラインをデータが通過する時間 (ターンアラウンドタイム) は 11ft(127.71msec) である．つまり，ある時点の出力結果には，最短で 127.71msec 前の入力までが反映される．この遅延時間は，入力を直接加工して音響信号を出力する処理 (リバーブなどのエフェクト処理) では問題になるが，本システムのように過去のビートを認識して，常に予測結果 (次のビート時刻) を出力する場合には問題にならない．例えば，180 M.M. の最も短い 4 分音符の長さでも 333.33msec あるため，予測時に前回のビート時刻周辺の情報は利用できる．

ただし，これはターンアラウンドタイムが 333.33 msec でもよいことを意味するのではない．例えば，発音時刻を検出するには，その時刻の前後100〜200msec の情報を平滑化微分の処理が必要とする．また，ターンアラウンドタイムが短いほど，より新しい情報を利用して予測がおこなえる．

5 実験結果

市販の CD からサンプリングしたモノラルの音響信号を用いて実験した．BD が強拍 (1，3 拍目)，SD が弱拍 (2，4 拍目) で多く鳴る，テンポがほぼ一定なロック・ポップス 30 曲を入力した．これらの曲は，テンポが 78〜168 M.M. と広い範囲から選び，21 の異なるアーティストによって演奏されたものである．

実験の結果，30 曲中 27 曲に対して正しいビートをリアルタイムに出力できた．どの曲も最初は BD と SD の特徴周波数を獲得していないために，ビート時刻を認識できてもビートタイプは判定できない．しかし，BD と SD がほぼ定常的に鳴り出して数小節経つと，ビートタイプも正しく出力するようになった．

誤認識した 3 曲では，ビート時刻は正しく得られていたが，ビートタイプを誤っていた．これは，BD と SD の特徴周波数を正しく獲得できなかったためか，ドラムスの変則的なリズムが一時期続いていたのが原因であった．

6 おわりに

本稿では，音楽音響信号に対するビートトラッキングシステムの並列化手法と，並列計算機 AP1000 へのリアルタイム実装について述べた．本システムでは，並列処理することにより，複数の手がかりをリアルタイムに抽出し，ビートの様々な解釈の可能性を同時に追跡できるようになった．

本実装では，並列計算機全体でパイプライン処理をおこない，パイプラインの各ステージに，それぞれの処理の質に応じた 4 種類の並列処理方式を適用した．これにより，質の異なる様々な処理をリアルタイムに実行することが可能になった．また，AP1000 に実装する際に，ステータスハードウェアを利用して時間確認可能な同期機構を実現した．これにより，同期が不要なプロセッサは処理を続けることができ，各プロセッサは処理の遅れを知ることが可能になった．

今後は，ドラム音の検出手法や予測処理の改良により入力の制約を減らすとともに，エージェント間の協調・統合の方式も改良していく予定である．またビートトラッキングの技術は，ビデオ／オーディオ編集システムやリアルタイム CG など，多くのマルチメディアアプリケーションに有効である．今後，これらのマルチメディアシステムへの応用もおこなっていく．

謝辞

AP1000 の SCSI 入出力の実装について協力して頂いた富士通研究所の稲野 聡 氏に深く感謝いたします．また，AP1000 の実行環境を提供して頂いた富士通研究所 並列処理研究センターに感謝いたします．

参考文献

[1] Masataka Goto and Yoichi Muraoka: *A Beat Tracking System for Acoustic Signals of Music*, ACM Multimedia 94 Proceedings, pp.365-372 (1994).

[2] Roger B. Dannenberg and Bernard Mont-Reynaud: *Following an Improvisation in Real Time*, Proceedings of the 1987 International Computer Music Conference, pp.241-248 (1987).

[3] Peter Desain and Henkjan Honing: *The Quantization of Musical Time: A Connectionist Approach*, Computer Music Journal, Vol.13, No.3, pp.56-66 (1989).

[4] Paul E. Allen and Roger B. Dannenberg: *Tracking Musical Beats in Real Time*, Proceedings of the 1990 International Computer Music Conference, pp.140-143 (1990).

[5] David Rosenthal: *Machine Rhythm: Computer Emulation of Human Rhythm Perception*, Ph.D. Thesis, Massachusetts Institute of Technology (1992).

[6] W. Andrew Schloss: *On The Automatic Transcription of Percussive Music — From Acoustic Signal to High-Level Analysis*, Ph.D. Thesis, CCRMA, Stanford University (1985).

[7] 片寄 晴弘: 音楽感性情報処理に関する研究 , 大阪大学基礎工学部 博士論文, pp.45-48 (1991).

[8] 後藤 真孝, 橋本 裕司: MIDI 制御のための分散協調システム — 遠隔地間の合奏を目指して—, 情処研報, Vol.93, No.109, 音楽情報科学 93-MUS-4-1 (1993).