音楽情報科学 9－4
（1995．2．24）

MIDI音源を用いた並列プログラムの動作理解

風間 一洋　　佐藤 孝治
NTT基礎研究所

マルチスレッドシステムやマルチエージェントシステムなどの並列プログラムの動作を認識することは難しい。その場合に、並列プログラムの動作を聴覚化することは、視覚化と同様に有効である。この論文では、並列プログラムの動作を認識する方法と、並列プログラムの情報を音へマッピングする方法を考察する。

Recognition of Parallel Program's Behavior using MIDI instruments

Kazuhiro Kazama, Koji Sato
NTT Basic Research Laboratories

It is difficult to recognize the behavior of parallel programs such as multi-thread systems or multi-agent systems. In such a case, auralization of parallel program's behavior is as useful as visualization. In this paper, we consider how to recognize parallel program's behavior, and mapping methods of parallel program's infomation to sound.

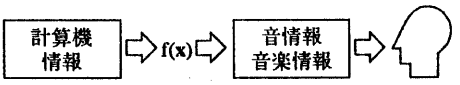

図 1: 聴覚化の概念

1 はじめに

マルチスレッドやマルチエージェントなどの並列プログラムは、複数の動作主体が並列に動作し、互いに相互作用するので、プログラムがどのように動作するのかが理解しにくい。

本論文では、そのような場合に聴覚化が有効であることを示す。そして、並列プログラムのデバッグや性能評価を対象に、並列プログラムのさまざまな情報を音に変換して、並列プログラムの動作を理解する方法について考察する。

2 音を用いた動作表現と動作理解

2.1 視覚化と聴覚化

データをユーザにわかりやすく提示する方法として、さまざまな**視覚化** (visualization) 手法が今まで提案されているが、本論文では**聴覚化** (auralization) 手法に注目する。

図1に示すように、聴覚化は $\boldsymbol{x} = (x_1, x_2, \cdots, x_m)$ で表される m 次元の計算機情報を、関数 $f(\boldsymbol{x})$ によって n 次元の $\boldsymbol{y} = (y_1, y_2, \cdots, y_n)$ で表される音情報や音楽情報に変換して、再生した音を人間が聞くことによって情報を獲得するアプローチである。

視覚化も計算機情報を別種類の色情報や図形情報に変換する点では同一である。視覚化と聴覚化の差は、情報伝達メディアとしての特性の違いによる。

たとえば、聴覚化では音楽情報から数値を認識するのは難しいが、視覚化では数値を表示したり、グラフに目盛を付ければ、容易に値を読みとることができる。

反対に、人間の視覚は細部の認識に時間が掛かるので、提示情報の一部を注視するのが普通で、並列イベントの認識は難しい。聴覚化では並列イベントをより高速に認識でき、さらに同時発生音をハーモニとして認識できる点が、並列プログラミングの分野では特に有効であると考えられる。

ただし、視覚化と聴覚化は対立する概念ではない。たとえば両方の機能をあわせ持つデバッガやモニタプログラムを作成し、互いの利点を生かしたマルチモーダルな情報提示が実現すれば、必要な情報が容易に認識できるようになる点が有効であると考えられる。

2.2 プログラムと音

レースの分野では、優秀なメカニックは車のエンジンの音を聞いただけで調子を知ることができる。この時は、バルブの音やタイミングチェーンの音をひとつひとつ明確に聞き分けているというより、音全体の微妙な違いから判断していると考えられる。同様にプログラマがプログラムの「音」を聞くことができるようにすることで、プログラムの構造や動作を理解したり、デバッグや性能評価に役立てることが可能にならないだろうか。

今まで文章として明文化されることはほとんどなかったが、たとえばプログラマやシステム管理者は、計算機の信号をアンプで拡大してスピーカーで聞いたり、電源、紙テープリーダー、フロッピーディスク、ハードディスクなどの各種の装置が出すノイズに耳を傾けることで、プログラムの実行、エラーの発生、無限ループなどを発見することをおこなってきた。

2.3 並列プログラムと音

並列プログラムでは、複数の動作が並列に実行され、互いにデータの交換、同期、排他制御などをおこなう。そのために、どのような動作がおこなわれるか必ずしも実行前に予測できない非決定性を持ち、ソースコードを見ただけで動作が理解するのは困難である。

並列プログラミングの分野における音の利用として、J. M. Francioni らは、並列プログラムの動作を MIDI インターフェースを用いて音として表現するシステムを作成しデバッグに用いる研究をおこなった。これでは評価するデータはあらかじめ収集された並列プログラム特有のデータに限定されていた [2]。本論文では、メッセージの交換、排他制御機構、同期機構などの並列プログラム特有の情報に限定しないで、一般的な計算機情報にまで対象を広げている。

3 プログラムの動作理解と情報獲得

3.1 プログラムの動作理解

プログラムの動作理解を、ソースコードがどのように実行されているかを知る**論理レベルの動作理解**と、そのプログラムが OS やハードウェア上で実際にどのように動作しているかを知る**物理レベルの動作理解**に分類できる。

たとえば、論理レベルの動作理解が必要な例は、デバッガでソースコード中の制御構造の実行や変数の値などを参照したりする場合が、物理レベルの動作理解

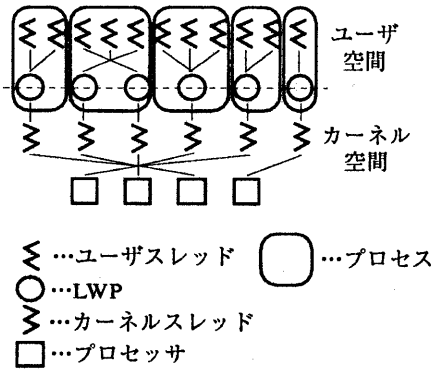

図 2: スレッドの概念図

が必要な例は、プログラム実行の性能評価やチューニングを目的とする場合が挙げられる。

3.2 並列プログラムの動作理解

本論文は、Solaris 2.3 のスレッドライブラリを用いたプログラムを対象とする。スレッドの概念図を、図2に示す。

ユーザは**ユーザスレッド**を用いてプログラムを記述する。スレッドライブラリは、ユーザスレッドを仮想 CPU と考えることができる **LWP** (Light Weight Process) に割り当てて、スケジューリングする。LWP はカーネル中の**カーネルスレッド**に結び付けられ、各プロセッサにスケジューリングされる。ユーザスレッドと LWP、LWP とプロセッサの対応関係は、ユーザが明示的に指定しない限り、自動的に最適化される。

つまり、アプリケーションの論理構造が持つ並行性と実際にプロセッサ上で実行される並列性は一般に一致しないことが多いので、Solaris 2.3 ではプログラムの論理レベルの並行性と物理レベルの並列性をユーザスレッドと LWP という形で分離して、ユーザが物理レベルを意識しないでプログラムを作成しても効率的に実行される。

しかし、たとえばデバッグやチューニングをおこなおうとする場合には、論理レベルの並行性と物理レベルの並列性を両方理解しなければならない。そこで、複数のレベルの情報を獲得して照合しながら動作を理解する方法を実現する必要がある。

4 情報のマッピング

4.1 計算機情報

並列プログラムの実行時に得られる情報は、並列タスク数、CPU 稼働率、処理時間などに代表される計算機の状態と、イベントを実行した並列タスク名、イベントの種類 / パラメータの値、イベント発生時刻などのイベントに関する情報の 2 つに大きく分類される。イベントとは、関数やシステムコールの実行やシグナルの発生などのプログラム実行中の発生した何らかの出来事を示す。

J. M. Francioni が音で表したイベントは、並列マシンの各プロセッサごとのメッセージの送受信や同期など、並列プログラム固有の情報に限られていたが、本論文では並列プログラム固有の情報だけではなく、より一般的な計算機情報も対象とする。つまり、並列プログラム固有の情報でなくても、各イベントが並列に発生するので、生成される音列全体から並列性を認識できる情報を得ることができる。

たとえば、セマフォやモニタなどの排他制御機構やコンディション変数などの同期機構に直接注目しなくても、プログラムの特徴的なイベントに注目すれば、排他制御や同期の様子や、さらにループやダブルバッファリングなどのプログラムの構造の認識や、音のテンポや時間的変化によるボトルネックの発見やバグの修正が可能になると考えられる。

4.2 音・音楽情報

本論文では、低レベルの音楽情報として音色、音量、音程音の開始時刻、終了時刻、持続時間、音の定位などの音の特性を考慮している。

聴覚化は、視覚化に比べて情報の提示が容易である。たとえば視覚化では、スレッドの数をバーグラフで示すために、バーグラフの大きさや色、表示する位置の決定など、真に必要な情報以外に多くの図形情報を決定する必要があり、表示する手続きが繁雑になるが、聴覚化は必要なパラメタが少なく容易である。

4.3 情報のフィルタリング

聴覚化は、m 次元の計算機情報を、n 次元の音・音楽情報に変換するアプローチであると述べた。

ある手法で得られる計算機情報が m 次元であっても、必ずしも全部を音情報に反映するのではなく、ユーザが認識したい現象に関連した情報のみに限定するのが望ましい。

視覚化も本質的には同様であり、聴覚化のように全体を一瞬にして認識することが難しいが、特定の部分だけを注視することで、結果的に情報をフィルタリングすることができる。

しかし、聴覚化ではカクテルパーティ効果のように特定の音だけを認識できても、非常に多くの音の中か

ら複数の音を同時に区別して認識するのは難しい。そこで、視覚化以上に情報のフィルタリングが重要である。

4.4 音列情報の認識とマッピング

W. W. Gaver は SonicFinder で使われる情報の視覚と聴覚を用いた情報マッピングを Symbolic mapping、Metaphorical mapping、Iconic mapping の3種類に分類した [5]。

また M. M. Blattner は、Earcon の研究で情報を短い音節にマッピングした [6]。

これらの方法では、ある操作に直接音や短い音列を割り当てて人間に認識させていたが、本論文では、さらに計算機情報を割り当てた音のまとまり、つまり音列に注目して、それがより上位の情報を示すようなマッピングを追及している。

たとえば、音の高さが変化する音列、音の速さが変化する音列、和音 / 不協和音ある。

音の高さが変化する音列としては上昇音列、下降音列などが考えられる。これらは、プログラムの動作状況を示すために有効だと考えられる。この音列で示すのが有効なプログラムの構造としては、ループやダブルバッファリングなどが考えられる。

音の速さが変化する音列としては、つねに一定間隔の音列、テンポが速くなる音列、テンポが遅くなる音列などがあり、プログラムの動作速度や時間制約などを示すために有効であると考えられる。

さらに、和音を生成するような音の組み合わせに、不協和音が生じるような音を加えることは、例外処理などの特殊な状況を示すのに有効であると考えられる。

4.5 音色

音色の選択は、聴覚化において非常に重要な要素である。GM では 128 種類の音色が利用できるが、音色の特性に注目して分類して使い分けた。

音の立ち上がりが明確な音色 (ピアノ、打楽器など) は、あるイベントがおこった瞬間が認識しやすい。逆に音の大きさが一定している音色 (オルガンなど) は、瞬間の認識はしにくいが、ある時間間隔を認識しやすい。また、ある時間間隔再生する必要がある音色 (拳銃の音など) は、短い瞬間を示す用途には適さない。

本論文では、これらの分類と使い分けはプログラマが決定した。しかし、今後聴覚化の音色決定を容易にするために、望む特性を持ち識別可能な音色の集合を自動的に作り出す機能が望まれる。

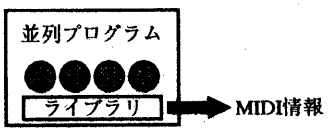

図 3: 直接的な情報獲得手法

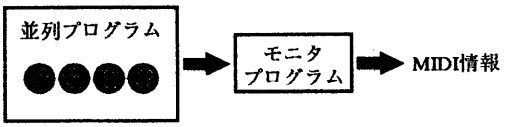

図 4: 間接的な情報獲得手法

なお本論文の実験では、スレッドや LWP などの並列タスクに対して音色を割り当てている。これは MIDI のパン機能を使って左右の音の大きさを調節し音場の異なる位置に定位することが簡単にできることと、MIDI インターフェースの各チャンネル (音色) の状態を表示するバーグラフに各並列タスクの状態が反映される特徴を利用して、視覚を合わせたマルチモーダルな理解できるためである。

4.6 情報獲得手法

並列プログラムの情報獲得のために、直接的な情報獲得手法と間接的な情報獲得手法を実現した。

4.6.1 直接的な情報獲得手法

直接的な情報獲得手法は、必要な情報を入手するためにプログラムに変更を加える手法である。プログラムに直接変更を加えなければならないが、任意の情報を自由に取り出すことができる。この手法はプログラマが直接編集・修正をおこなうソースコードに対しておこない、論理レベルの動作理解に適している

本論文では、ソースコードに PRINT 文を挿入するように、MIDI コマンドを送信する MIDI 文を埋め込んでいる。

4.6.2 間接的な情報獲得手法

間接的な情報獲得手法は、プログラムの実行をモニタして必要な情報を入手する方法である。プログラムを変更しないでライブラリや OS、ハードウェアから情報を収集するので、物理レベルの動作理解に適している。しかし、あらかじめライブラリ、カーネルまたはハードウェアが情報を取り出せるようなしくみをサポートしなければならない。

本論文では、LWP の情報を獲得するために、Solaris 2.3 の /proc ファイルシステムを利用してプロ

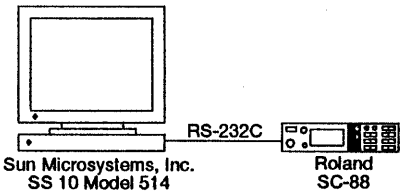

図 5: ハードウェア構成

セスの仮想メモリイメージをアクセスし、実行時間、LWP、スケジューリングクラス、システムコール、シグナル、ヒープ、スタック、レジスタなどのプロセスに関する情報を取得している。

ただし、この方法ではユーザスレッドの情報を獲得することができない。デバッガ作成のために提供されている`libthread_db`を用いれば、スレッドライブラリ内部の情報を取り出すことができるが、現時点ではまだインターフェースが公開されていない。

5 音の出力方法

5.1 ハードウェア構成

図5に本システムのハードウェア構成を示す。

Sun Microsystems, Inc. の SPARCstation 10 Model 514(4 プロセッサ) と、 Roland の SC-88 をシリアル接続して使用した。

5.2 ソフトウェア構成

次の 3 種類の MIDI ライブラリを実装した。

1. MIDI 音源へ MIDI メッセージを送信 (実行時)
2. MIDI 音源へ MIDI メッセージを送信 (指定時刻)
3. 標準 MIDI ファイルの作成

2は、指定時刻に MIDI メッセージを MIDI 音源に送るためにライブラリの内部でリアルタイム・スレッドを生成している。これを使用して、鷲坂・高田が作成した MIDI シーケンサプログラム [3],[4] を Solaris 2.3 上に移植した。3は、MIDI メッセージの送信時刻を指定して標準 MIDI ファイルを作成する。

なお並列プログラムの聴覚化手法は、オンライン出力とオフライン出力の 2 つに分類される。オンライン出力は 1 を使用し、オフライン出力は、標準 MIDI ファイルを 3 を使って作成し 2 を使用した MIDI シーケンサーで再生する。

5.3 処理速度と再生速度

昔は計算機の処理速度が遅かったのでオンライン出力でも人間が認識できたが、現在の計算機の処理速度は非常に高速なので聴覚化の方法によっては認識が不可能であり、 MIDI 音源で再生できないことがある。

そこで、本論文では計算機の処理速度と再生速度の関係に応じて、オフライン出力とオンライン出力を使い分けている。

オフライン出力は、プログラムから得た情報を標準 MIDI ファイルとして保存し MIDI シーケンサプログラムで聞きやすい速度で何度も繰り返し再生することができる。オンライン出力は、計算機情報を MIDI メッセージに変換して即座に MIDI インターフェースに送信するのでプログラムの動作と音が同期するので理解しやすい。

6 実験

6.1 哲学者の食事問題

E. W. Dijkstra が提示した哲学者の食事問題は、典型的な排他制御の問題である。 Solaris 2.3 のスレッドを用いてデッドロック回避を考えたプログラムを 2 種類作成した。

この例では、直接的な情報獲得手法を用いて、次のように情報マッピングをおこなって、オンライン出力をおこなった。

- 哲学者 → 音色、音の定位
- 食事開始 → 音の発生
- 食事終了 → 音の消去
- 食事をしている哲学者の数 → 音程

食事をしている間だけ哲学者に割り当てた音色で音が発生するが、同時に発生する音の数は食事中の哲学者の数を示すので、効率的な排他制御をおこなっているプログラムを判別できる。さらに食事中の哲学者の数に応じて音程を変化させると、違いがより明確に認識できた。同一の情報を複数の手段で提示することは有効だと思われる。

当初は複数の状態で音を出していたが、生成される音が多過ぎて、プログラムの構造とマッピングを理解していない他人は認識できなかった。そこで、哲学者が食事している状態だけにフィルタリングすると、認識してもらうことができた。

この実験では実行状態を示すので、音の大きさが一定している音が適していた。

6.2 mtcopy

`mtcopy` はマルチスレッド化された`cp`コマンドであり、各ファイルの読み込み・書き込みごとにスレッド

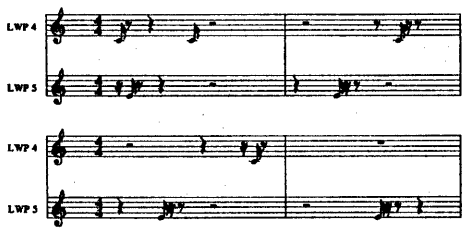

図 6: 採譜された楽譜

を割り当て、ファイル I/O 処理を多重化している [7]。

この実験例では、間接的な情報獲得手法を用いて得られた情報を、次のようにマッピングし、標準 MIDI ファイルとして保存してから聞き取りやすい速度で再生した。現在は LWP の情報しか獲得できないので、LWP とスレッドを 1 対 1 に対応させている。

- LWP ID → 音色、音の定位
- `read` システムコール → 音 (C4) の出力
- `write` システムコール → 音 (E4) の出力

`mtcopy` では、各スレッドは C4 または E4 の音しか発生しないので、各スレッドは`read` システムコールと`write` システムコールのどちらか片方だけしか実行していないと推測できた。また`read` システムコールを実行するスレッドと`write` システムコールを実行するスレッドの組がほぼ交互に動作するのが観測できた。またそれらの組が複数あることから、`mtcopy` は、ファイルごとに読み込み・書き込みスレッドの組を生成して、同期して動作していると推測できた。

ソースコード、読み込み・書き込みスレッドはセマフォを用いてダブルバッファリングしていた。このように、プログラマにある程度の知識があれば、プログラムのソースコードを詳しく読まなくても、聴覚化によってプログラムの構造を推測できる。

この実験では各イベントの発生した瞬間を示すので、音の立ち上がりがはっきりわかる音が適していた。

実験では、全部で 14 個の LWP を生成した。互いに協調していることを示すために、読み込み用の LWP 4 と書き込み用の LWP 5 の組の音だけを取り出して採譜した結果を図6 に示す。

7 まとめ

本論文では、並列プログラムの動作の聴覚化、プログラムの動作理解と情報獲得、計算機情報の音・音楽情報へのマッピングについて述べた。

今後は、スレッドライブラリ内部の情報の獲得を可能にすると共に、得られた複数の階層の情報を融合してユーザにより明確に望む情報が得られる手法と、視覚と聴覚を両方活用したデバッガやモニタプログラムを作成し、システム開発に使用し有効性を検討する。

謝辞

この研究に関して、議論や情報提供をして頂いた斉藤 康己グループリーダー、尾内 理紀夫グループリーダー、平田 圭二主任研究員、野島 久雄主任研究員、鷲坂 光一研究主任、高田 敏弘研究主任に感謝します。

参考文献

[1] Joseph Rothstein: MIDI A Comprehensive Introduction, The Computer Music and Digital Audio Series, A-R Editions, Inc., 1992.

[2] Joan M. Francioni, Larry Albright, Jay A. Jackson: Debugging Parallel Programs Using Sound, Proceedings of the Workshop on Parallel and Distributed Debugging, pp.68–75, May 1991.

[3] 高田 敏弘: UNIX 上の MIDI シーケンサーとそのインターフェース, 音楽情報科学, Vol.5, No.3, pp.13–16, 1994.

[4] 鷲坂 光一: 標準 MIDI ファイルからのメロディの自動抽出法, 音楽情報科学, Vol.5, No.3, pp.7–12, 1994.

[5] William W. Gaver: The SonicFinder: An Interface That Users Auditory Icons, Human-Computer Interaction, Vol4, pp.67–94, 1989.

[6] Meera M. Blattner, Denise A. Sumikawa, Robert M. Greenberg: Earcons and Icons: Their Structure and Common Design Principles, Human-Computer Interaction, Vol.2, pp.11–44, 1989.

[7] 日本サン・マイクロシステムズ株式会社: マルチスレッドを使用したプログラミング例, SunExpert, No.11, 1994.