

ピアノシミュレーターの FPGA 並列化に向けた処理の削減

岡崎 雅嗣[†] 富永 英嗣^{††}

[†]ヤマハ(株) 電子楽器事業部, ^{††}研究開発統括部 〒430-8650 静岡県浜松市中央区中沢町 10-1

E-mail: {[†]masatsugu.okazaki, ^{††}eiji.tominaga}@music.yamaha.com

あらまし ピアノの 3 次元有限要素法に基づくシミュレーションをリアルタイム動作可能なハードウェアへ実装するために、演算構造・メモリ構造・並列化特性を分析した。ピアノの有限要素モデルはモード合成法を用いたうえで可聴域へ縮退した後も数万規模の自由度を持つため、弦・ハンマー・本体の相互作用を可聴域の時間分解能で更新するには多くの演算量が必要となる。その結果、汎用 CPU では実時間の数百倍、GPU でも数十倍の処理時間を要した。

並列化を想定した分析の結果、最大のボトルネックは弦と本体の結合部における大規模な座標変換行列とベクトルの積であることと、さらに弦の部分振動の処理構造が直方体にならず、ハンマー・ダンパー・弦の縦振動モデルの構造が並列化の効率を上げられない要因であることを確認した。

そこで、音質劣化を極力抑えながら、弦と本体との方向別結合点数および本体部分振動の数の削減を行うことで、座標変換行列の削減し、計算量とメモリ量を圧縮した。その結果、当初は数十個の FPGA を要すると見積もった処理規模を数デバイスで実装可能な水準に削減し、物理モデル音源のリアルタイム実装の道筋を示した。

本成果は後に浜松市楽器博物館に寄贈した電子ピアノに搭載されているピアノ物理モデル音源の実装に寄与している。

キーワード ピアノ物理モデル、有限要素法、モード合成法、時間領域シミュレーション、リアルタイム処理、並列処理、FPGA 実装

Reduction of Processing for FPGA-Parallelization of a Piano Simulator

Masatsugu OKAZAKI[†] and Eiji TOMINAGA^{††}

{[†]Digital Musical Instrument Division, ^{††}Research and Develop Division} YAMAHA Corporation

10-1 Nakazawa, Chuou-ku, Hamamatsu, 430-8650 Japan

E-mail: {[†]masatsugu.okazaki, ^{††}eiji.tominaga}@music.yamaha.com

Abstract We analyzed the computational, memory, and parallelization characteristics of a three-dimensional finite-element-based piano simulation for real-time hardware implementation. Even after modal synthesis and reduction to the audible frequency range, the model retains tens of thousands of degrees of freedom, requiring heavy computation to update string-hammer-body interactions in real time, far exceeding the capabilities of CPUs and GPUs.

The analysis identified large matrix-vector multiplications at string-body junctions and limited parallelism in the string, hammer, damper, and longitudinal vibration models as the main bottlenecks.

By reducing direction-dependent string-body coupling points and body vibration modes while preserving sound quality, we significantly reduced computational and memory requirements, enabling a processing scale initially estimated to require dozens of FPGAs to be implemented using only a few devices, and demonstrating a practical path to real-time physical-model piano sound synthesis.

Key words Piano Physical Model, FEM, Modal synthesis, Time-domain Simulation, Real-Time Processing, Parallel Processing, FPGA Implementation

1. はじめに

1.1. ピアノの有限要素固有値解析

ピアノは、鍵盤、ハンマー、ダンパー、弦および本体（響板、フレーム、支柱など）など、全体で 1 万点弱ほどの部品から構成される楽器である。

ピアノを構成する部品には、木材、鉄、羊毛などの粘弾性体を用いられる。このような粘弾性体から成る構造は、有限要素法(FEM)を用いることで近似でき、振動特性のシミュレーションを行うことができる[2]。

我々がピアノ物理モデルの構築を推進している当初の目的は、「設計仕様と音との因果関係を明らかにし、楽器の改良に活かすこと」である。シミュレーション結果を可聴化して音として評価するための有限要素メッシュサイズは、波長の 1/6 に相当する 2.5mm 程度となる。図 1 にピアノの 3 次元モデルおよびメッシュの例を示す[5]。

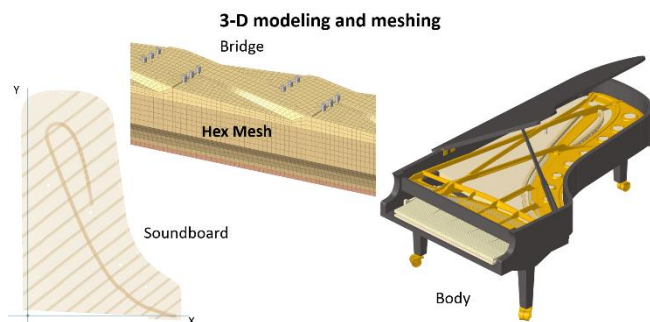


図 1 ピアノの 3 次元モデルおよびメッシュの例

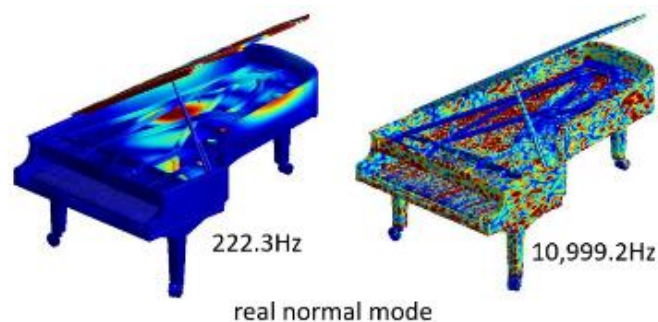


図 2 ピアノ本体の有限要素固有値解析結果の例

この有限要素モデルの自由度は、奥行きが 2.7m のコンサート用グランドピアノの場合、数千万規模となる。我々はこの規模の自由度をもつモデルの固有値解析を、2TB のメモリを搭載した解析サーバー上で商用ソフトウェアを用いて行っている。ピアノ本体の有限要素固有値解析の結果の例を図 2 に示す。

固有値解析によって、本体の部分振動がもつ周波数および形状を得ることができる[2]。全ての部分振動の数は、モデルの自由度と同じ数千万規模となるが、この中から聴覚的に有意な 1 万数千個程度の部分振動のみを採用している。

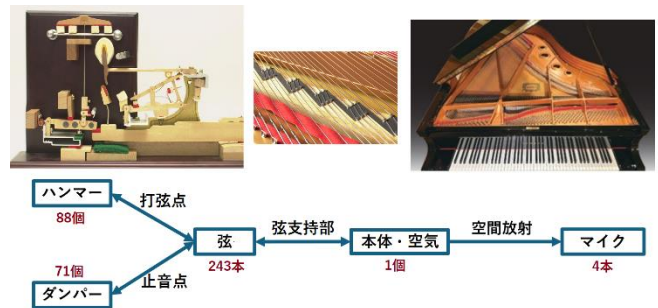


図 3 ピアノ部品の連結と相互作用

1.2. ピアノの時間領域シミュレーション

ピアノでは、本体に結合された弦をハンマーで打つことによって、ピアノ全体に振動が励起される。

仮想ピアノは実際のピアノと同様に、図 3 に示すハンマー・ダンパー・弦・本体の 4 種類の部品群に分類され、打弦点・止音点・駒（響板上の弦支持部）・アグラフ（フレーム上の弦支持部）（以下、ジャンクションと呼ぶ）を介して、それぞれの部品が接している。ジャンクションを介して相互に作用/反作用の力を受け渡し、それぞれの部品の変位が時間軸上で変化する。

なお、この変位更新の計算に際して、弦および本体の形状は、部分振動の重ね合わせ（モード合成法）で表現している[2-5]

1.3. 部品・ジャンクションの変位更新関数

ピアノの各部品の変位を更新するプログラムは、部品およびジャンクションごとに分かれている。各関数について以下に概要を述べる。

1. ハンマー入力 Hammer_init()
ハンマーの操作情報に応じてハンマーの速度を更新する。
2. ダンパー入力 Damper_KB()/ Damper_DP()
鍵盤/ペダルの操作情報に応じてダンパーの位置を更新する。
3. シフトペダル入力 Shift_SP()
シフトペダルの操作情報に応じてハンマーの特性を更新する。
4. ハンマー→弦 Hammer_to_String()
ハンマーと弦の現在位置からそれらの相対位置を更新する。
5. ハンマー Hammer()
ハンマーの変位を更新する。ハンマーが弦に接触すると、フェルトが圧縮しながら[1]弦に力を与え、同時にハンマーは弦から反力を受けて減速し、やがて弦から離れる。
6. 弦 String()
ハンマー・ダンパーにより生じる弦の変位を弦支持部の変位を考慮しながら更新する。ここで、弦曲げ振動の非線形成分が弦の縦振動を励起する。
7. 打弦点 Striking_Point()
打弦点における弦の位置を更新する。
8. 弦→本体(弦支持部) String_to_Body()
弦が本体上の弦支持部に及ぼす力を更新する。

9. 本体

Body()

弦から作用する力によって生じる本体の変位を更新する。
木材が主体であるため、高次の振動成分ほど速く減衰する。

10. 本体→弦(弦支持部) Support_Point()

本体の変位に基づいて弦支持部の変位を更新する。これにより、ハンマーに打たれていない弦の共鳴が生じる。

11. ダンパー

Damper()

ダンパーが弦に接触すると、弦の運動エネルギーを吸収し、弦振動はより速く減衰する。

12. ディレイ

Next_Condition()

全ての変位を 1 タイムステップだけ過去のものにシフトする。次の時刻における変位の初期値を過去の値から予測した値に更新する。

13. 空間放射

Radiation()

ピアノの表面形状の変化に伴い、仮想周辺空間に音圧を放射する。音圧は直接波・反射波・回折波を含む複雑な伝播を行う。

※本稿では、この空間放射については以降割愛する

ハンマーが弦に衝突する様子と、音圧が空間へ放射される様子を図 4 に示す[6]。これらのシミュレーション結果を可視化したものについては、関連文献および Youtube 上で公開しているアニメーション動画を参照されたい。我々は、これらの可視化シミュレーションの結果に加え、可聴化シミュレーションの結果を分析・比較することにより、ピアノの物理機構の解明を進めている。



図 4 ピアノの物理シミュレーションの様子

2. リアルタイム処理の可能性

本シミュレーションは、CPU による並列処理で行われていたが、その実行効率はリアルタイム処理には遠く及ばず、パラメータにも依存するものの、仮想時間 1 秒の解析に約 300 秒を要した (2009 年、Core-i7 950 でソフトウェア最適化済)。つまりこのシミュレーションをリアルタイム化するには約 300 倍の高速化が必要となる。しかし、もしリアルタイム化が実現できれば、物理音源に転用できる可能性があると考え高速化を検討した[7]。

最初に、CPU 版プログラムを 2010 年当時利用可能であった黎明期の GPGPU (Geforce GTX480) [10]に移植し約 10 倍の高速化を達成した。しかし予想していた通り、GPU を用いてもリアルタイムには遠く至らなかった。さらなる高

速化を図るために、デジタル回路設計を前提とした検討を行った。

2.1. 事前計算とリアルタイム解析の分離

2.1.1. 音源 LSI に実装すべき処理の抽出

シミュレーションプログラムは、ピアノの研究用途に開発した経緯から、係数を事前に準備する処理とシミュレーションを行う処理が混在していた。音源にするのは後者のシミュレーション処理であり、まず以下の 3 種類に分離した。

- ・事前にピアノの物理情報から係数を作成する部分
- ・センサや LSI を制御するドライバに入る部分
- ・音源 LSI 上で高速に実行するシミュレーション部分

このうち LSI 化の対象となるシミュレーション部の関数の呼び出し順を図 5 に示す。なお、図中右側の関数が複数回呼ばれているのはピアノ全体の運動方程式を分離型反復解法により解いているためである。

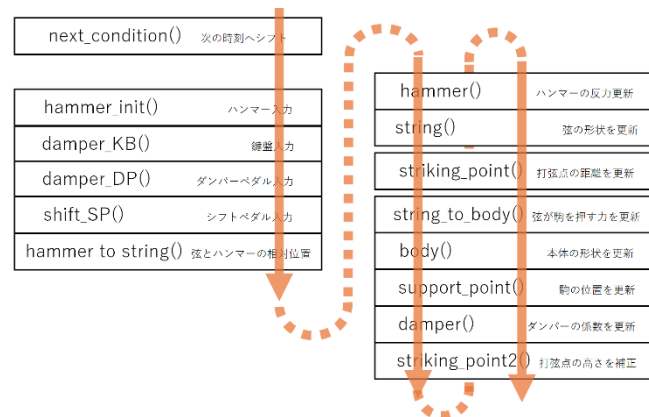


図 5 プログラムの各関数の実行順

2.1.2. 仮想時間刻みの調整

図 5 で示した更新処理を実行する時間刻み(仮想サンプリング周波数の逆数)と、図中で複数回実行しているイタレーションの回数は、リアルタイム実行時の ALU リソースに直接影響する。そこで、音質と計算安定性を保ったまま最低必要な時間刻みとイタレーション回数を検討した。シミュレーションプログラムの仮想サンプリング周波数は 88.2kHz としていたが 44.1kHz に落としても聴感上の影響は認められなかった。さらにそのうえで、イタレーションの反復回数を 2 回まで減らしても計算安定性に問題がないことを確認した。

2.2. 乗加算回数と除算・高等関数の計算量の見積り

2.2.1. 乗加算、制御操作の回数

乗算・加減算・制御操作などの演算回数を見積もった。時間刻み 1/44100 秒、すなわち、1 タイムステップの変位更新を行うために実行される「加算、半加算、乗算、ビット演算、読み出し、条件分岐」の回数を合算した結果を表 1 の中列に示す。1 回の変位更新あたり 69M 回の演算操作、仮想 1 秒あたりで 3.04T 回の演算や操作を行っている。つま

リアルタイムに動かすには3TFLOPSの実効値が必要であることが分かる。また、計算の大部分は、表中赤で示した弦と本体の計算が占めていることが分かる。

関数・機能名	変位更新毎の 演算操作回数	FPGA 実装用 簡略版
Hammer_init()	926	926
Damper_KB/DP()	707	707
Shift_SP	8	8
Hammer_to_String()	2,715	2,715
Hammer()	2× 38,696	2× 38,696
String()	2× 1,374,214	2× 1,374,214
Striking Point()	2× 24,032	2× 24,032
String to Body()	2× 48,424	2× 48,424
Body()	2× 16,289,008	2× 2,220,288
Support Point()	2× 16,135,680	2× 2,097,432
Damper()	2× 274,404	2× 274,404
Next Condition()	594,299	547,931
Total	68,967,571	12,706,795

表1 1タイムステップ毎の変位更新関数の計算回数

2.2.2. 除算と高等関数の回数

除算および高等関数が発生する回数についても評価した。まず、全ての除算は実装上係数化により回避可能であった。その一方で、ハンマーの計算[3,8]では、べき関数を用いており、仮想時間 1/44100 秒の 1 ステップ中に最大 1944 回(4 回×243 本×2 周)のべき関数を呼び出す可能性があった。実際のピアノ演奏では、同時にハンマーが弦に接触するのは両手で和音を弾いた指の本数に制限されると仮定できる。そのため、べき関数を求める回数の上限は 240 回(4 回×10 指×3 弦×2 周)に減らせそうである。しかし、LSI 回路上でこのような動的なモジュール化やベストエフォート型の制御を実装すると煩雑となるため、本検討では、88 鍵のハンマーが同時に弦に接すると想定した。

一方、ハンマーが弦に接触しているときには、アクション機構上ダンパーが弦に触れることはない。そのため、排他的に処理を行うことで、演算回路資源を共有できる可能性はあると考えられた。

関数・機能名	変数の数	係数の数
システム	0	665
入力操作	270	3
ハンマー	1,415	928
打弦点	1,303	486
ダンパー	176	34,268
弦	176,352	447,580
弦支持点	2,112	8,139,690
本体	45,840	45,840
出力	45,840	0
Total	273,308	8,669,460

表2 変数と係数の数 (word)

2.3. メモリ量の見積りと帯域

各々の重さや周波数などの係数群と、変位の変化を保持する状態変数群を数え上げ、必要となる記憶領域を算出した。その結果、係数メモリは約 8.67Mwords、変数メモリは約 273kwords を要することが分かった。各機能ブロックに

おける変数・係数の数を表2に示す。

全ての変数・係数は、仮想 1/44100 秒毎の変位更新処理において、少なくとも 1 回はアクセスされる。また表中赤字で示した弦支持部に関するメモリが、全体のメモリ量および帯域の大半を占めていた。

リアルタイム処理を想定すると、この赤字で示した係数 8.14Mwords を、Body() と Support Point() において 1/44100 秒の時間内に 4 回読み出す必要があり、約 1.44Twords/s の帯域が必要となる。そのため、外部メモリでの対応は難しく、データ量の削減とあわせ、オンチップ SRAM ヘデータを局所化することを前提として検討を進めた。また、この弦支持部の係数情報は削減対象の最有力と位置付けた。

2.4. 並列可能性の調査

並列実行する処理粒度について、大きく 2 つの階層を検討した。(1)関数単位で依存関係があるかという階層と、(2)関数内の for 文において、イタレーションなどの依存関係があるか、の 2 点である。

2.4.1. 各関数の並列可能性

図5で示した各関数の引数、すなわち入出力について調べ、依存関係を整理した結果を図6に示す。図上の「ディレイ・初期値推定」関数から開始し、図下のディレイ関数までが仮想 1/44100 秒の処理に相当する。図中の矢印は、矢印の元の関数を終了しなければ、矢印の先の関数を開始できないことを意味している。

この依存関係の整理により、回路（演算リソース）を独立に準備すれば、関数単位で並列実行が可能な区間があることが確認できる。

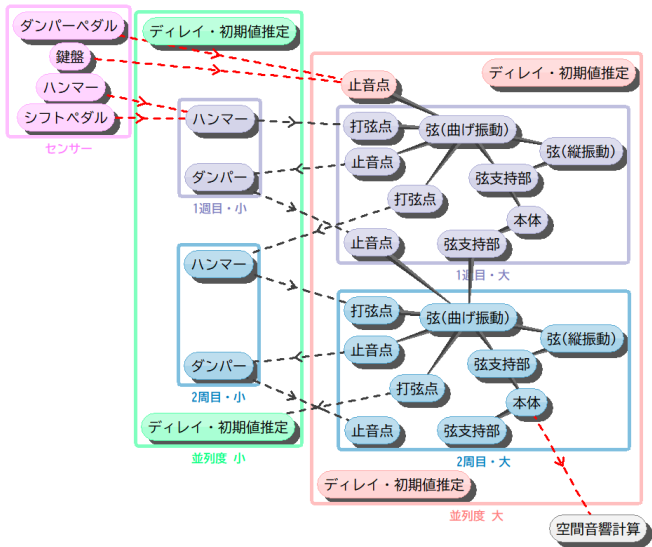


図6 ピアノ部品関数の依存関係

一方で、図6の「①弦⇒②弦支持部⇒③本体」の処理は経路が長く、表2と照らし合わせると計算量も多いため、クリティカルパスになると考えられた。これら3つの関数は、平均して 1/44100 秒の 1/6 以下の時

間で処理できるように、並列度を設計する必要があることが分かった。

2.4.2. for 文の並列可能性

各関数内の for 文にイタレーションなどの依存関係がある場合、そのループは逐次実行が避けられず、並列化は行えない。本シミュレーションの多くの処理は積和演算で構成されており、乗算は各要素を独立に処理できるため並列化できる。一方で、加算は依存性があった。2010 年当時の GPGPU での実装もリダクション加算に $\log_2 N$ のパイプライン段数を要していた。

本来、ディジタル回路はブロードキャスト線を増やすことが容易であり、1 クロックで多くの並列演算器に値を配ることができる。しかし、配布先の各 ALU が同時に生成した部分和を再度 ALU に戻し集めて段階的に集約すると、段数の時間が避けられず、これが ALU 全体のスループットを妨げる。

このボトルネックを解消するためには、毎クロック並列演算器に変数・係数を供給しつつ、同時に毎クロック結果を取り出せるように、並列度と同じ数のベクトルを同時に入力できる（例えば 16 入力や 32 入力など）多入力加算器が必要になる。

この多入力加算器を浮動小数点形式で実現すると回路規模が大きくなりパイプラインも長くなるため、採用は難しいと考えられた。固定小数であればコンパクトに多入力加算器を実現できるため、固定小数化が必要になるだろうと考えられた。

2.4.3. 計算処理構成の矩形性

各関数の特性を整理すると、弦曲げ振動および本体の処理は並列度を高めやすい一方、弦の縦振動、ハンマー、ダンパーの処理は大規模な並列化が難しい。例えば、弦の計算は以下の複数の要因を含むため、5～7 段の多重ループ（for 文）が必要となり、係数も多次元インデックスを持つ複雑な配列構造となる。

- ・ 鍵数：88 鍵
- ・ 鍵あたりの弦本数：1～3 本
- ・ 弦持点部：2 ヶ所
- ・ 空間次元：3 次元
- ・ 弦曲げ振動の部分振動の数：3～207 個
- ・ 時間ディレイ：3 サンプル

図 7 に示すように、弦曲げ振動の部分振動の数は周波数に概ね反比例して指数関数的に変化する。このため、各弦の並列可能性も指数的に変動し、計算処理の構成が鍵域間で矩形にならない。

具体的に、低域鍵では 200 並列以上の展開が可能である一方、高域鍵では最大でも 3 並列までしか展開できない。この一配列の添え字空間が直方体にならない一不整形な処理は、並列化時に負荷の偏りが生じ、高域での専用の並列化手法や補助的な設計上の工夫が

必要になるだろうと考えられた。

一方、本体の弦支持部で行う力および変位の計算は、 $15,280 \times 528$ （ $528=88$ 鍵 $\times 2$ 支持部 $\times 3$ 次元）の行列と、528 または 15,280 要素のベクトルとの積によって求められる。この弦支持部・本体処理は、並列化時に演算器が遊ぶ部分が少なく、高い並列効率を達成しやすいだろうと考えられた。

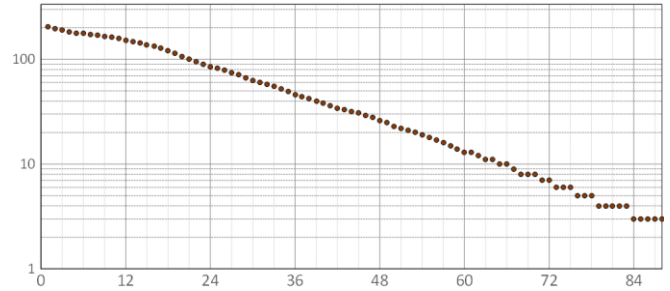


図 7 弦曲げ振動の部分振動の数

2.5. 複雑な分岐制御の確認

条件分岐はさほど多くはなく、ハンマーやダンパーの接触/非接触に関する分岐がみられた。

実装上、複雑だと感じられたのは以下の 3 点である。

- ・ 弦の縦振動の演算スキームが入り組んでいること
- ・ クロック/サンプル/センサスキャン/CPU タイマ/フレーム処理、の時間の整合がやや複雑であること
- ・ 空間放射は高速畳込みが 1000 並列規模で必要であり、タイミング設計が難しいこと[11]。

2.6. その他のボトルネックの特定

リアルタイム実装を想定し、FPGA クロック周波数を 200MHz、更新周期を 44.1kHz とすると、1 更新すなわち 4535 クロックあたり 68,967,571 回の演算操作が必要である。つまりこのままだと実効性能で 15,207 の並列化が必要であり、実際にはその数倍の並列演算回路が必要になる。

FPGA に実装する場合、DSP 数やロジックやメモリが足りなければ処理を複数のデバイスに分割する必要がある。しかしその場合は、デバイスをまたぐ配線や、帯域・レンテンシが新たな制約となる。当時の大規模な FPGA でも DSP 数は 2000～3600 であり[9]、この演算量をそのまま受けるには、少なくとも十数台の FPGA を繋ぐ構成となる。その場合、1 更新の期間内に転送できるデータ量も厳しくなるだろうと考えられた。

3. メモリと計算量の削減

計算に必要な演算器数やメモリ容量は、FPGA への実装可否および必要デバイス数に直結し、将来的な製品化時には音源 LSI のコストにも影響する。単に「動作させる」だけでなく、回路規模の最適化も求められる。そのため、本検討ではメモリ使用量と計算量の削減を行った。

