修士論文

題目

共役勾配法を用いたヤコビアン計算のパイプライン処理による高速化

指導教員

高木一義 教授

2024年

三重大学大学院 工学研究科 情報工学専攻
コンピュータアーキテクチャ研究室

今井智也 (422M506)

共役勾配法を用いたヤコビアン計算のパイプライン処理による高速化

今井智也

内容梗概

近年，機械学習や最適化アルゴリズムの進展が著しい中で，特に高次元の多変数関数におけるヤコビアンの計算が重要な役割を果たす．ヤコビアンは，多変数のベクトル値関数が各変数に対してどのように変化するかを表す行列であり，その情報は数学や物理学の様々な分野で幅広く利用されている．しかし，高次元でのヤコビアン行列の計算は，計算コストが高いという課題がある．

この課題に対処するため，本研究ではFPGA（Field Programmable Gate Array）を活用してヤコビアン計算のハードウェア化を行い，処理の高速化と低消費電力化を目指す．本研究では，CPU と FPGA を内蔵した Programmable SoC （System on Chip）を用いてヤコビアン計算の設計を行う. ヤコビアン計算を FPGA に実装し，データの転送には DMA（Direct Memory Access）通信を利用した．FPGA に実装する際には，ハードウェア資源と処理速度の最適化が必要であり，ビット削減や配列の再利用などを用いてハードウェア資源の使用効率を向上させた．同時に，パイプライン化を行いながらヤコビアン計算の処理速度を向上させることに焦点を当てた．

実験には Xilinx 社の PYNQ-Z2 ボードを導入し，バンドル調整手法を適用してヤコビアン計算の性能評価を行った．本研究では異なる 4 つのアーキテクチャを設計し，それらのアーキテクチャのハードウェア資源の使用量を比較した．実験の結果，設計したアーキテクチャと CPU の実行時間を比較すると，最大で約 256 倍の高速化が得られたことが示された．

Accelerating Jacobian Calculation Using Conjugate Gradient Method with Pipelining

Tomoya Imai

Abstract

In recent years, the computation of the Jacobian, especially in high-dimensional multivariate functions, has played an important role in the rapid progress of machine learning and optimization algorithms. The Jacobian is a matrix that describes how a multivariate vector-valued function varies with respect to each variable, and this information is widely used in various fields of mathematics and physics. However, the computation of the Jacobian matrix in high dimensions is computationally expensive.

To address this issue, this study aims to hardwareize the Jacobian computation by utilizing field programmable gate arrays (FPGAs) to achieve faster processing and lower power consumption. In this research, a Programmable SoC (System on Chip) with a built-in CPU and FPGA is used. (System on Chip) with a built-in CPU and FPGA. The Jacobian computation is implemented in FPGA, and direct memory access (DMA) communication is used for data transfer. The FPGA implementation required optimization of hardware resources and processing speed. At the same time, we focused on improving the processing speed of the Jacobian computation while pipelining.

The experiments were conducted on a Xilinx PYNQ-Z2 board, and bundle adjustment techniques were applied to evaluate the performance of the Jacobian calculations. We designed four different architectures and compared the hardware resource usage of these architectures. Experimental results showed that the designed architectures were up to 256 times faster than the CPU execution times.

目 次

図 目 次

表 目 次

1　はじめに

近年，機械学習や最適化アルゴリズムの進展が急速に進み，これらの技術は広範な応用分野で注目を集めている．これらの手法の中で，多変数関数の微分情報を含むヤコビアン計算は，勾配降下法やニュートン法などの最適化手法から，逆伝播アルゴリズムなどの機械学習モデルの訓練に至るまで，様々なタスクにおいて中心的な役割を果たしている．ヤコビアンは，関数の各変数における偏微分係数を要素とする行列であり，これを計算することで関数の局所的な変動や相互の依存関係を把握できる．

しかし，実際の問題において多くの機械学習モデルや最適化アルゴリズムは，非常に大規模かつ高次元のデータセットやパラメータ空間を扱う必要がある．このような複雑性が増す状況では，ヤコビアン計算では非常に計算コストが高くなり，従来のソフトウェアによる実装では十分な効率が得られないという課題がある．

ヤコビアン計算において，数値微分や解析的微分が用いられてきたが，これらの手法は計算コストが高いうえに，数値的な不安定性や誤差の蓄積などの問題がある．このため，新たな手法やアーキテクチャの検討が求められる．

本研究では，計算コストの削減と高い効率性を実現するために，Field-

Programmable Gate Array（FPGA）を活用したハードウェアアクセラレーションの手法を提案する．FPGA はその柔軟性と並列処理能力から，特に特定のアプリケーションに最適化されたハードウェアを構築するのに適している．これにより，ヤコビアン計算の並列性を最大限に引き出し，高速で効率的な計算を可能にすることが期待される．

これまでの研究では，数値計算の高速化や機械学習アルゴリズムの最適化において FPGA の活用例がいくつか論文はある．論文 [1] においては，SLAM（Simultaneous Localization and Mapping）技術の発展に焦点が当てられている．この技術は，自己位置推定と環境地図の同時生成を可能にするものであり，論文ではその中でも特定の処理に注目し，それを FPGA を駆使して最大 31 倍の高速化を達成している．この驚異的な速度向上は，現実世界のロボティクスや自動運転などのアプリケーションにおいて，リアルタイム性や効率性の向上に大きく寄与している．同様に，論文 [2] では，畳み込みニューラルネットワーク（CNN）に焦点が当てられている．CNN は画像処理やパターン認識などのタスクで幅広く使用されているが，その中でも特定の処理において FPGA を活用し，最大 5 倍の高速化が達成された．この高速化により，リアルタイム性の要求が厳しい画像処理タスクにおいて，効率的な処理が可能となる．しかし，ヤ

コビアン計算のハードウェアアクセラレーションに関する研究はほとんどない．そのため，本研究では，ヤコビアン計算に焦点を当てたハードウェアアクセラレーションを行い，高速化することを目標とする．

2　準備

2.1　FPGA を用いたハードウェアアクセラレーション

Field-Programmable Gate Array（FPGA）とは，論理回路を自由に書き換え可能な半導体デバイスであり，柔軟性や並列処理能力などの利点がある．

FPGA はハードウェアの機能をソフトウェアでプログラム可能な特性を備えており，これにより特定のアプリケーションやタスクに最適化されたハードウェアを構築することができる．通常の ASIC（Application-Specific Integrated Circuit）とは異なり，FPGA は再プログラム可能であり，設計の変更が必要な場合でもハードウェアの再設計や製造が不要である．これにより，開発サイクルが短縮され，柔軟性が向上する．

また，FPGA は数千から数百万の論理要素を含んでおり，これらの要素は同時に動作することができる．これにより，機械学習，信号処理，暗号解読などのアプリケーションにおいて大量のデータを同時に処理する際に，FPGA の並列処理能力が非常に有益である．

これらの利点から，FPGA は柔軟性，並列処理能力など特長を持ち，様々なアプリケーションで幅広く利用されている．

2.2 ヤコビアン

ヤコビアン（Jacobian）は，多変数のベクトル値関数における微分に関する行列であり，数学や工学，物理学などさまざまな分野で使用される概念である．ヤコビアンは，多変数関数の微小な変化に対する出力の変化率を表す行列として定義される．これにより，多変数関数の微分に関する情報をコンパクトにまとめることができる．

一変数関数の微分の概念を多変数関数に拡張すると，各入力変数に対する出力の変化率を表すヤコビアンが得られる．多変数関数 $F(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_m(\mathbf{x})]$ がベクトル値を返す場合，ヤコビアンは以下のように表される．

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

ここで，n は入力変数ベクトル $\mathbf{x} = [x_1, x_2, \cdots, x_n]$ の次元であり，各要素 $\frac{\partial f_i}{\partial x_j}$ は関数 f_i を変数 x_j で偏微分した値である．

ヤコビアンの応用として，最適化アルゴリズムでの勾配降下法，制御理論におけるシステムのモデリング，機械学習における誤差逆伝播法，数

値解析における微分方程式や連立方程式の解法などが挙げられる．ヤコビアンはこれらの分野で，多変数関数の微分に関する情報を抽出し，問題の解析や最適化に役立つ重要なツールである．

2.3　3 次元のためのバンドル調整

本研究は画像からの 3 次元復元に焦点を当て，その中でヤコビアン計算を効果的に行うためにバンドル調整手法を用いる．バンドル調整は，3 次元点群とカメラの位置と姿勢を最適化する手法であり，具体的には点群を画像上に再投影した際に，観測データに最も適合するような 3 次元点群とカメラ姿勢を推定する．

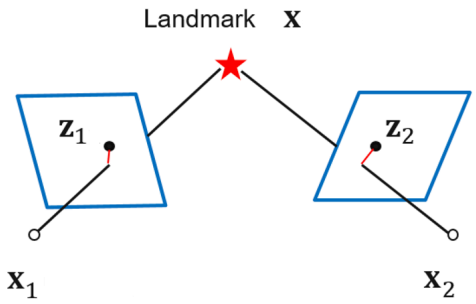

図 2.1: バンドル調整

図 2.1 において，3 次元空間に存在する点 (★) を画像上に再投影したときの，観測データ z_1, z_2 との誤差 (赤線の長さ) が最小となる x_1, x_2 を推定する．要するに，復元した三次元の点群データとカメラの位置をもとに，二次元の画像上に投影された点と観測点との誤差を最小化するようにパラメータを微調整するのがバンドル調整の目的である．

この手法は様々な実応用シーンで活躍しており，例えば自動運転技術，

掃除ロボットのナビゲーション，ドローンの姿勢制御などに利用されている．こうした実用例では，バンドル調整が多変数関数の微分情報を含むヤコビアン計算を中心的に必要とし，その結果として高度な精度と効率性が得られることが確認されている．

2.4　ガウス消去法によるヤコビアン計算

2.4.1　バンドル調整とヤコビアン計算

本研究では，カメラの位置パラメータと三次元点群の位置パラメータを最適化するための計算を行う．三次元点群の位置パラメータとカメラの位置パラメータの δ を更新するためには，以下の式を用いて計算する．

$$J^T J\delta = J^T E \tag{1}$$

式 (1) は残差ベクトル E を最小化するために最小二乗法を使用してパラメータ δ を更新するものである．具体的な手順は以下の通りである．

2.4.2　ヤコビアン行列の構築

画像上の点 $x_k = (x, y)$ がカメラ $c_1 ... c_n$ および各三次元点群 $p_1 ... p_m$ の微小な変化により，どのように移動するかを考える．

$$J = \begin{bmatrix} \frac{\partial \mathbf{x}_1}{\partial c_1} & \cdots & \frac{\partial \mathbf{x}_1}{\partial c_n} & \frac{\partial \mathbf{x}_1}{\partial p_1} & \cdots & \frac{\partial \mathbf{x}_1}{\partial p_m} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial \mathbf{x}_k}{\partial c_1} & \cdots & \frac{\partial \mathbf{x}_k}{\partial c_n} & \frac{\partial \mathbf{x}_k}{\partial p_1} & \cdots & \frac{\partial \mathbf{x}_k}{\partial p_m} \end{bmatrix} \tag{2}$$

ヤコビアン行列 J は式 (2) となる．これらの各項は連鎖律により計算する．

まず，式（2）の左側行列の画像上の点 $x = (x, y)$ のカメラパラメータ $c_1 ... c_n$ の微小変化による偏微分について考える．

$$\frac{\delta \boldsymbol{x}}{\delta c} = \frac{\delta \boldsymbol{x}}{\delta X_c} \frac{\delta X_c}{\delta c} \tag{3}$$

式 (3) は，画像上の点 x がカメラのパラメータ c に対してどれだけ敏感かを表すヤコビアン行列を計算する式である．以下にそれぞれの項を具体的に説明する．

・ $\frac{\delta x}{\delta X_c}$ ： これはカメラ座標系における三次元点 X_c が画像座標 x に対してどれだけ変化するかを示すヤコビアン行列である．

・ $\frac{\delta X_c}{\delta c}$ ： これは三次元点 X_c がカメラパラメータ c に対してどれだけ変化するかを示すヤコビアン行列である．

これらのヤコビアン行列を連鎖して計算することで，画像上の点 x がカメラのパラメータ c に対してどれだけ変化するかを示すヤコビアン行列が得られる．

次に，式（2）の右側行列の画像上の点 $x_k = (x, y)$ の三次元点群 $p_1 ... p_m$ の微小変化による偏微分について考える．

$$\frac{\delta \boldsymbol{x}}{\delta p} = \frac{\delta \boldsymbol{x}}{\delta X_c} \frac{\delta \boldsymbol{X_c}}{\delta \boldsymbol{X_w}} \frac{\delta \boldsymbol{X_w}}{\delta p} \tag{4}$$

式 (4) は，画像上の点 x が三次元点 p_m の位置に対してどれだけ敏感かを示すヤコビアン行列を計算するものである．以下にそれぞれの項を具体的に説明する．

$\frac{\delta \boldsymbol{x}}{\delta X_c}$: 上と同じである．

$\frac{\delta \boldsymbol{X_c}}{\delta \boldsymbol{X_w}}$: これは三次元点 X_c が世界座標系における三次元点 X_w に対してどれだけ変化するかを示すヤコビアン行列である．

$\frac{\delta \boldsymbol{X_w}}{\delta p}$: これは世界座標系における三次元点 X_w が点群 p に対してどれだけ変化するかを示すヤコビアン行列である．

これらのヤコビアン行列を連鎖して計算することで，画像上の点 x が点群のパラメータ p に対してどれだけ変化するかを示すヤコビアン行列が得られる．

2.4.3 ヘッセ行列の計算

ヘッセ行列 $J^T J$ は，ヤコビアン行列の転置 J^T とヤコビアン行列 J の積である．ヘッセ行列は最小二乗法において残差の二乗和を最小化するためである．

J^TJ を計算するときに一工夫する． $\frac{\delta \boldsymbol{x}}{\delta c}$ は２行６列，$\frac{\delta \boldsymbol{x}}{\delta p}$ は２行３列である．仮に b，c，d，e を用いて式 (5)，式 (6) と置く．

$$\frac{\delta \boldsymbol{x}}{\delta c} = \begin{pmatrix} b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \\ c_0 & c_1 & c_2 & c_3 & c_4 & c_5 \end{pmatrix} \tag{5}$$

$$\frac{\delta \boldsymbol{x}}{\delta p} = \begin{pmatrix} d_0 & d_1 & d_2 \\ e_0 & e_1 & e_2 \end{pmatrix} \tag{6}$$

また，式（５）,（６）を用いて以下の式を作る．

$$A_{ij} = b_i b_j + c_i c_j \tag{7}$$

$$B_{ij} = b_i d_j + c_i e_j \tag{8}$$

$$C_{ij} = d_i d_j + e_i e_j \tag{9}$$

式（７）,（８）,（９）を用いて J^TJ を計算し，式 (10) に示す．

$$J^TJ = \left(\begin{array}{ccc|ccc} A_1 & & & B_{1,1} & \dots & B_{1,m} \\ & \dots & & \dots & \dots & \dots \\ & & A_n & B_{n,1} & \dots & B_{n,m} \\ \hline B_{1,1} & \dots & B_{n,1} & C_1 & & \\ \dots & \dots & \dots & & \dots & \\ B_{1,m1} & \dots & B_{n,m} & & & C_m \end{array}\right) \tag{10}$$

2.4.4　勾配ベクトルの計算

勾配ベクトル $J^T E$ は，ヤコビアン行列の転置 J^T と残差ベクトル E の積である．残差ベクトル E は各データ点におけるモデルの予測データと実際の観測値データの差をまとめたベクトルである．

式 (5)，(6) を用いて $J^T E$ を計算し，式 (11) に示す．x_{err} と y_{err} はモデルの予測データと実際の観測値データの差である．

$$J^t E = \begin{pmatrix} b_{c_0 0} x_{err0} + c_{c_0 0} y_{err0} \\ \cdots \\ b_{c_n 5} x_{errm} + c_{c_n 5} y_{errm} \\ d_{p_0 0} x_{err0} + e_{p_0 0} y_{err0} \\ \cdots \\ d_{p_m 2} x_{errm} + c_{p_m 2} y_{errm} \end{pmatrix} \tag{11}$$

2.4.5　線形ソルバの計算

式 (1) は，ヘッセ行列 $J^T J$ と勾配ベクトル $J^T E$ からなる線形ソルバーを解き，パラメータ δ を求める．線形ソルバーはガウス消去を用いて計算している．ガウス消去法は，連立方程式の解を求める手法の一つである．連立方程式は複数の未知数から成る一連の等式で表され，ガウス消去法はこれらの方程式を単純な形に変換して解を求める．基本的な手順は以下の通りである．

- 前進消去： J^TJ 行列を上三角行列に変換するために，各段階で行基本変形を行う．これにより，下部にゼロが並んだ上三角形の行列が得られる．

- 後退代入：上三角行列から未知数の値を逆算していくことで，連立方程式の解を求める．

ガウス消去法は，連立方程式を解くための汎用的で安定した手法であり，数値計算や線形代数の基本的な手法としてよく知られている．

2.4.6 パラメータの更新

得られたパラメータ δ を用いて，カメラパラメータ $c_1 ... c_n$ および三次元点群のパラメータ $p_1 ... p_n$ を更新する．この更新により，モデルの予測データと実際の観測されたデータとの誤差を最小化する方向にパラメータが調整され，最終的には最適なモデルパラメータに収束する．

バンドル調整は，最初に仮定されたモデルのパラメータを元に観測データを修正していく反復プロセスである．このステップを繰り返し実行し，各繰り返しでバンドル調整がカメラの姿勢と三次元点群の座標を調整して最適解に収束する．

3　ヤコビアン計算のハードウェア化

3.1　基本構成

FPGA を用いたヤコビアン計算の基本的な構成を図 3.2 に示す．本研究では，CPU と FPGA を内蔵した Programmable SoC（System on Chip）を利用してヤコビアン計算を行う．このデバイスは，FPGA と CPU が一体となったチップであり，ハードウェアとソフトウェアの統合を有機的に行うことが可能である．

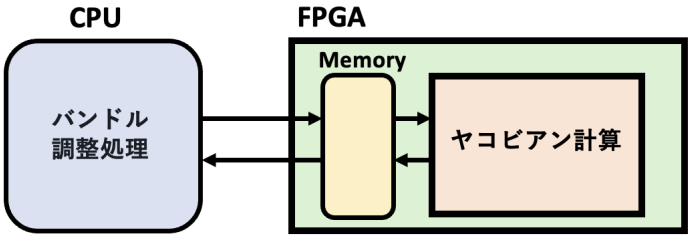

図 3.2: 基本構成

データ転送の側面では，CPU と FPGA 間での迅速で効率的な通信を実現するために，DMA（Direct Memory Access）通信を活用する．この手法により，入力データとしてカメラの位置パラメータと三次元点群の位置パラメータを送信し，出力データはそれぞれの最適化が行われた結果のパラメータデータを受信する．

3.2 共役勾配法

ヤコビアン計算の中で最も処理時間のかかる部分は，ガウスの消去法を用いた線形ソルバーである．ガウスの消去法は高い精度を誇りつつも，計算処理には時間がかかり，並列化が難しいという特徴があり，そのため FPGA には適してない．そのため，新たに線形ソルバーとして共役勾配法を採用する．

共役勾配法（Conjugate Gradient Method）は，数値最適化および連立一次方程式の解法に広く使用される効率的な反復法の一つであり，線形問題における収束速度が速く，メモリ効率が高い特徴を有している. 以下に，共役勾配法の基本的なアプローチおよび計算手順を説明する.

共役勾配法は，次のような連立一次方程式 $Ax = b$ を考える. ここで，A は正定値対称行列，x は未知のベクトル，b は既知のベクトルである．アルゴリズムの初期段階では，解の初期推定値 x_0 および残差ベクトル $r_0 = b - Ax_0$ を計算する．初期残差ベクトル r_0 を用いて，共役勾配ベクトル $p_0 = r_0$ を計算する．これが最初の探索方向となる．反復ステップ k では，以下の手順を繰り返す.

1. スケールド共役勾配:

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

を計算し，スケールド共役勾配 $x_{k+1} = x_k + \alpha_k p_k$ を更新する．

2. 残差更新: $r_{k+1} = r_k - \alpha_k A p_k$ として残差を更新する．

3. ベータ係数の計算: $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$ を計算し，新たな共役勾配ベクトル $p_{k+1} = r_{k+1} + \beta_k p_k$ を得る．

反復が収束条件を満たすか，あるいは一定の反復回数に達するまで，ステップ 1 から 3 を繰り返す．収束判定には，残差のノルムや目的関数の変化量などが用いられる．

共役勾配法は非線形最適化問題を解くための反復法であり，各反復ステップで残差と勾配の共役性を活用して収束を迅速に行う．この手法は計算処理が迅速で並列化が容易な利点があるが，ガウスの消去法と比較して精度がやや低いというデメリットがある．これは利用する際に慎重に考慮する必要がある．

新しい手法の導入により，線形ソルバーの処理時間を短縮し，ハードウェア資源をより効果的に活用できることが期待される．ただし，精度に関するトレードオフがあるため，具体的なアプリケーションにおいて

どれだけの影響があるかを検証することが重要である.

3.3 最適化

3.3.1 ハードウェア最適化のポイント

FPGA は制約のある有限なハードウェア資源しか活用できないため，通常のプログラムをそのままハードウェア化することは難しいという課題がある．ハードウェアリソースの最適な利用は，性能や効率に直結する重要な要素であり，そのためには設計段階での慎重な工夫が求められる．

通常のプログラムは，一般的には高水準の抽象度で表現され，柔軟性があるが，ハードウェア化においては固定されたリソース内で機能を実現する必要がある．そのため，どのようにして機能を分割し，ハードウェアリソースを最適に配置するかが重要な設計上のポイントとなる．

また，ハードウェア化が成功しても，CPU 上で同様の処理を行った場合と比較して処理速度が低下する可能性があるため，効果的な最適化が不可欠である．特に，ハードウェア化されたシステムではクロックサイクルに敏感であり，不適切な最適化は性能に悪影響を及ぼす可能性がある．

ハードウェア化の際には，FPGA 固有のアーキテクチャに合わせて最適化を行い，性能向上を図る必要がある．

本研究では，ヤコビアン計算の高速化とハードウェア資源の効果的な削減を目指して，以下の最適化手法を主に導入する．

3.3.2 配列の再利用

本研究では，配列の再利用をする手法を用いる．配列の再利用は，既存の配列を再利用することで新たなメモリ領域の確保を最小限に抑え，メモリ使用量を最適化する手法である．このアプローチでは，同じメモリ領域を異なる部分で再利用することで，新しいデータ構造や要素を格納するために新たなメモリを確保する必要がなくなる．

特に，この手法はリソース制約が厳しい環境や組み込みシステムにおいて有益であり，メモリ管理の効率を向上させる．プログラム内でのメモリの使い方を工夫することで，システムのパフォーマンス向上やメモリの効率的な利用が可能である．配列の再利用によって，既存のメモリ領域を再活用することで新たなメモリの確保が減少し，結果としてメモリ使用効率が向上する．

本研究では，線形ソルバの改良に焦点を当て，具体的には方程式 $Ax = b$ における係数行列 A の一部の要素に対して，ベクトル x の値を動的に代入するアプローチを取り入れた．この手法により，計算過程で生成される一時的な配列を再利用することが可能となり，その結果，メモリ使用量を有意に削減できる．

3.3.3 量子化

ヤコビアン計算において変数のビット数を削減する手法は，ハードウェア資源の削減が期待される．この手法は，変数の表現において冗長なビットを取り除き，それによってハードウェア上での計算に必要なリソースを効率的に削減する．ビット削減の主な利点は，ハードウェア資源の節約にある．変数のビット数を削減することで，メモリや演算回路などのハードウェア資源の使用量を減少させ，限られたリソース内で効率的な計算を可能にする．

これは，特に FPGA において，制約のある環境での実装において重要である．FPGA は有限なハードウェア資源しか利用できないため，ビット削減は制約内で最適な計算を実現するための重要な手法となる．

ただし，ビット削減は計算結果の精度に影響を与える可能性がある．変数のビット数を削減することで情報の損失が発生し，計算結果が元の精度から逸脱する可能性がある．そのため，ビット削減を適用する際には，計算結果の精度と性能のバランスを慎重に調整する必要がある．

本研究では，三次元点群の誤差計算において，計算プロセスにおける変数のビット数を削減した．この改善策として，具体的には，従来の浮動小数点数からよりコンパクトな半精度浮動小数点数への変更を行い，そ

の変更を基にした計算手法を導入した.

この変更により，計算におけるメモリ使用量や演算速度の向上が期待される．特に，三次元点群のような大容量かつ高次元のデータに対する誤差計算において，ビット数の削減がリソースの最適化に寄与し，計算プロセス全体の効率を向上させる.

3.3.4　並列化

ヤコビアン計算の処理をパイプライン化することによって，計算の効率や処理速度の向上が期待される．パイプライン化は，処理を複数の段階に分割し，これらの段階を同時に実行することで，処理のスループットを最大化する手法である．このアーキテクチャの特長は，個々のステージが独立して動作するため，一部のステージでの遅延が他のステージに影響を与えず，全体の性能を向上させることができる点である．

ヤコビアン計算においても，数多くの変数や微分項を考慮する必要があり，計算が複雑である．このような計算をパイプライン化することで，異なるステージでの演算を同時進行し，全体の計算時間を短縮できる．ボトルネックとなりやすい微分計算や行列演算などのステップを独立させ，同時に処理することで，ヤコビアン計算の効率向上が期待される．

本研究では，共役勾配法の反復ステップにおける特定の計算部分が，その並列処理が比較的容易でありながらも計算に時間がかかるという課題に着目し，その部分においてパイプライン処理を導入した．このアプローチは，処理を逐次的ではなく，同時に進行させることで並列性を引き出し，計算全体の効率を向上させることが期待される．

共役勾配法において反復ステップの最適化は，収束までの反復回数や

計算時間に大きな影響を与える重要な要素である．そこで，特に計算が複雑で時間がかかる部分においてパイプライン処理を採用することで，計算の高速化とリソースの効率的な利用を実現可能である．

4　評価および結果

4.1　開発環境

本研究では，バンドル調整の実装において，図 5.3 の Xilinx 社の PYNQ-Z2 デバイス [3] を利用する．このデバイスは ARM Cortex-A9 CPU と Zynq 7020 FPGA が統合されており，高い柔軟性を有する．Zynq を用いれば，FPGA でデジタル回路を構成することによるハードウェア開発だけでなく，プロセッサとして使用している CPU 上で Linux などの OS を稼働することにより，ソフトウェア開発も行うことができる．また，Zynq は FPGA によるハードウェアと，プロセッサによるソフトウェアが，簡単にデータのやり取りを行えるという特徴を持つ．具体的な実装に際して，高位合成ツールである Vivado HLS とデザイン設計ツールの Vivado[4] を使用する．

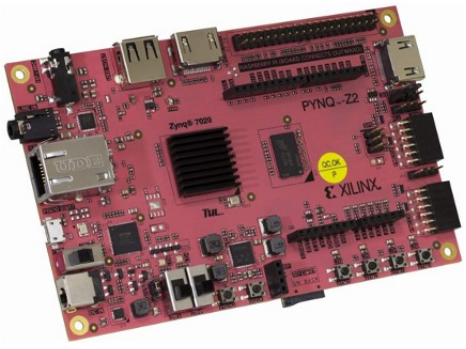

図 4.3: PYNQ-Z2

Vivado HLS は，高位合成を行うツールで，高水準言語で記述されたアルゴリズムをハードウェア記述言語に変換する．この過程で，豊富なプラグマが提供され，これらを適切に利用することで性能向上やリソースの最適化が可能である．特に，C++を用いて高位合成用のプログラムを記述し，それを Jupyter Notebook[5] を介して FPGA と接続することで，柔軟性と可読性が向上する．この構造により，FPGA のリソースを効果的に活用し，高い性能を発揮できることが期待される．

同時に，制御プログラムには Python が採用され，CPU 上でシームレスに実行される．これにより，高位合成されたハードウェアとの連携がスムーズに行え，全体のシステムが協調して動作する．

4.2 評価方法

本研究の主な目的は，バンドル調整を CPU および FPGA にそれぞれ実装し，両者を実行時間，ハードウェア資源の使用量，および精度の観点から比較・評価することである．ハードウェア資源の評価は，ブロック RAM (BRAM)，加算・乗算器 (DSP)，フリップフロップ (FF)，およびルックアップテーブル (LUT) の 4 つの側面に焦点を当てて行う．

BRAM は，FPGA 内でデータを格納するためのブロックメモリで，大容量で高速なメモリを提供する．DSP（Digital Signal Processor）は，FPGA 内で高度な信号処理や演算を行うための専用の回路で，乗算器や加算器を含む．FF（Flip-Flop）は，FPGA 内のフリップフロップ回路で，デジタルデータを格納して同期的な動作を実現する．LUT（Look-Up Table）は，FPGA 内で論理演算を行うためのテーブルで，真理値表に基づいて論理関数を実現する．

具体的に使用するデータセット [6] は，点群のマーカーが書かれている画像を複数の視点から撮影したもので，バンドル調整はサンプルコード [7] を使用する．

また，以下の 4 つの最適化手法のアーキテクチャを用意した．

1. 最適化なし

1 の段階では，ソフトウェアコードをそのままハードウェアに変換するアーキテクチャである．

2. 量子化や最適化

2 では，特定のヤコビアン計算の変数のビット数を 32 ビットの単精度浮動小数点数（float 型）から 16 ビットの半精度浮動小数点数に削減した．ただし，全ての変数に対してビット削減を行うことは困難である．ヤコビアン行列の計算において，連鎖律や微分の積の法則を利用し，ビット数を削減することで誤差の増加が起こる可能性がある．このため，適切な変数に対してのみビット数の削減した．同時に，配列の再利用も取り入れ，ハードウェア資源の削減した．

3. 並列化

3 では，2 で実施された量子化と最適化に加えて，パイプライン化を導入したアーキテクチャである．

4. 新しい手法

4 では，3 の段階での線形ソルバーの部分を共役勾配法に変更し，更に量子化や最適化，パイプライン化したアーキテクチャである．

4.3 結果

各アーキテクチャにおけるハードウェア資源の使用量を表 4.1 に示す．これらの値は Vivado HLS で高位合成した結果によるものである．

表 4.1 より，アーキテクチャ1 では BRAM の使用量が最大の上限を超え，この課題に対処するため，アーキテクチャ2 の段階で量子化や最適化の手法を駆使し，BRAM の使用量を低減させた．しかし，この対策の一環として，FF と LUT の利用量が著しく増大した．

その後のアーキテクチャ3 では，並列化手法を導入することで，全体的な資源の利用量が増加したという結果となった．一方で，アーキテクチャ4 においては，全体的なハードウェア資源の使用量がアーキテクチャ3 の構成とほぼ同等である．

表 4.1: ハードウェア資源量

	BRAM	DSP	FF	LUT
MAX	280	220	106,400	53,200
1	290	36	9,105	15,173
2	275	28	18,991	18,747
3	276	28	19,656	19,776
4	279	24	19,666	19,047

実行時間及び精度を表 4.2 に示す．表 4.2 より，アーキテクチャ4 の実行時間が最も速くなり，CPU と比較して約 256 倍高速化された．精度に関してはアーキテクチャ3 まではかなり優れているが，アーキテクチャ4 では約 24 %精度が低下した．

表 4.2: 実行時間と精度

	実行時間 (s)	精度 (%)
CPU	25.60	99.67
2	19.12	99.67
3	9.64	99.67
4	0.10	75.32

4.4 考察

本研究では，FPGA を用いてヤコビアン計算のハードウェア資源，実行時間，および精度を比較･評価した．ハードウェア資源の観点では，BRAM 以外の領域には十分な余裕があるが，BRAM の資源はかなり使用率が高い状態である．これは，多岐にわたる処理計算が行われているためである．通常，ビット削減を全ての計算に適用すると BRAM の使用量を大幅に減少させることができるが，微小な計算におけるヤコビアン計算ではビット削減を適用すると微分情報が失われ，その結果として精度が著しく低下する．このため，精度への影響が比較的少ない特定の計算にのみ量子化を適用することにした．

実行時間の観点から見ると，CPU と比較して，パイプライン化の導入により約 3 倍の高速化が達成され，さらに共役勾配法の採用によっては約 256 倍の高速化が実現した．これにより，パイプライン化が処理速度向上において重要な役割を果たしていることが明らかである．また，共役勾配法はガウスの消去法と比較して並列化が容易であり，高速な計算が可能であるが，その一方で精度の低下が約 24%まで発生した．このトレードオフを考慮し，精度と処理速度のバランスを検討する必要がある．ガウスの消去法は高い精度を確保できるが，共役勾配法はより高速な計

算が可能であるが，その代わりに精度を一部犠牲にする．具体的なアプリケーションや要件によって，計算時間や精度の要求が異なるため，どちらの手法を採用するかは慎重に検討する必要がある．

5　まとめ

本研究では，Programmable SoC を活用してヤコビアン計算をハードウェア実装した．コードの最適化や量子化を行うことで，ハードウェア資源の使用量を削減できることを示した．また，並列化を導入することで，CPU と比較して約 3 倍の高速化が達成した．さらに，ヤコビアン計算の一部の計算アルゴリズムを変更することで，変更前と比較して約 256 倍の高速化が実現したが，その代わりに一部の精度が犠牲になった．

今後の課題としては，コードの最適化や量子化以外での BRAM の使用量削減，残りのハードウェア資源の効果的な活用による更なる高速化，および精度の向上が挙げられる．

謝辞

本研究の進行に際し，高木一義教授からの多大なご指導に感謝の意を表します．また，大野和彦講師や深澤祐樹技術員からは研究に関する多くの有益なアドバイスをいただき，心から感謝しています．最後に，私の日常生活を支え続けてくれたコンピュータアーキテクチャ研究室の皆様にも，心からの感謝の意をお伝え申し上げます．

参考文献

[1] R.Liu , et al., "eSLAM: An Energy-Efficient Accelerator for Real-Time ORB-SLAM on FPGA Platform", Design Automation Conference 2019.

[2] Suhail Basalama, et al., "FlexCNN: An End-to-end Framework for Composing CNN Accelerators on FPGA", ACM Transactions on Reconfigurable Technology and Systems 2023.

[3] https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html Accessed:2024-01-7

[4] https://japan.xilinx.com/products/design-tools/vivado/vivado-whats-new.html Accessed:2024-01-7

[5] https://jupyter.org/ Accessed:2024-01-7

[6] https://github.com/tmako123/nextagejupyter/blob/master/BundleAdjustment.ipynb Accessed:2024-01-7

[7] https://github.com/tmako123/nextagejupyter/tree/master/data/pointmarker Accessed:2024-01-7