6A-01

# Sinkhorn アルゴリズムの FPGA 実装のための固定小数点化における必要ビット幅削減

片山 熙優 † 穐山 空道 †

† 立命館大学 情報理工学部

## 1. 研究背景

### 1.1 最適輸送問題

最適輸送問題とは、自然数の集合 $\boldsymbol{a}=\{a_1,a_2,...,a_n\}$ と $\boldsymbol{b}=\{b_1,b_2,...,b_m\}$ および行列 $\boldsymbol{C}\in R^n\times R^m$ が与えられたとき、以下の条件を満たしかつ $\sum(\boldsymbol{C}_{ij}\times\boldsymbol{P}_{ij})$ を最小化する行列 $\boldsymbol{P}\in R^n\times R^m$ を求める問題である。

$$\boldsymbol{P}_{ij}\geq 0 \tag{1}$$

$$\sum_{j=1}^{m}\boldsymbol{P}_{ij}=a_j \tag{2}$$

$$\sum_{i=1}^{n}\boldsymbol{P}_{ij}=b_i \tag{3}$$

最適輸送問題は輸送コストの最小化問題と捉えられる。すなわち $\boldsymbol{a},\boldsymbol{b}$ をある商品の各供給地での供給数および各消費地での消費数と、$\boldsymbol{P}_{ij},\boldsymbol{C}_{ij}$ を供給地 $i$ から消費地 $j$ への輸送数および輸送コストと見る。条件 (1) は消費地から供給地へ逆向きに輸送はできないことを、条件 (2) と条件 (3) は受給がちょうど満たされることを表す。

本研究では最適輸送問題のソートへの応用を考える。ソートしたい値の集合 $\boldsymbol{x}=\{x_1,x_2,...,x_n\}$ に対し、最適輸送問題への入力を $\boldsymbol{a}=\boldsymbol{x},\boldsymbol{b}=\{0,1,...,n-1\}$ とすると、$\boldsymbol{Px}$ は $\boldsymbol{x}$ の要素が昇順に並んだベクトルの近似値になることが知られている。以降では $\boldsymbol{b}$ を、入力の対応する要素のソート後順位という意味で「ランク」と呼ぶ。また行列 $\boldsymbol{C}$ は以下のように与えることが一般的である。

$$\boldsymbol{C}_{ij}=(a_i-b_j)^2 \tag{4}$$

### 1.2 Sinkhorn Algorithm (SA)

Sinkhorn Algorithm (SA) とは最適輸送問題を効率的に解くアルゴリズムである。SA の計算手順を以下に示す。

1. 値の初期化：$\boldsymbol{P}=\exp(-\lambda\boldsymbol{C})$ と初期化する。ただし $\lambda$ は正の定数で、exp は行列の要素ごとに計算する。また長さがそれぞれ $n,m$ のベクトル $\boldsymbol{u},\boldsymbol{v}$ を 1 で初期化する（i.e., $\boldsymbol{u}=\{1,1,...,1\}$）。
2. 反復計算：次の式で $\boldsymbol{u},\boldsymbol{v}$ を更新する。ただし $<\boldsymbol{x},\boldsymbol{y}>$ は二つのベクトル $\boldsymbol{x},\boldsymbol{y}$ の内積を表す。

$$v_j=\frac{b_j}{<\boldsymbol{P}_i,\boldsymbol{u}>} \tag{5}$$

$$u_i=\frac{a_i}{<\boldsymbol{P}_i,\boldsymbol{v}>} \tag{6}$$

これを一定の回数に達するまで反復する。

3. 解の計算：最適解 $\boldsymbol{P}^*$ は $\boldsymbol{P}^*=\mathrm{diag}(\boldsymbol{u})\boldsymbol{P}\mathrm{diag}(\boldsymbol{v})$ で与えられる。ただしベクトル $\boldsymbol{x}$ に対し $\mathrm{diag}(\boldsymbol{x})$ は $\boldsymbol{x}$ を対角成分とする行列である。

### 1.3 組み込みシステムにおける SA

SA は組み込みシステムでも利用される。文献 [1] では高効率な Brain-Computer Interface (BCI) の実現に SA を利用する。具体的には脳内の運動皮質から取得したデータ集合を実際の体の動きのデータ集合とマッチングする際に SA を利用し、SA をアクセラレータ上で実行することで全体を高速化する。

SA のような数値計算アルゴリズムの組み込みシステムでの効率的な実行には固定小数点化が有効である。固定小数点化とはプログラム内の浮動小数点演算を固定小数演算に置き換える操作である。FPGA や ASIC への回路実装では固定小数点化により浮動小数点演算回路を生成・利用する必要がなくなる利点がある。固定小数点化による組み込みシステムの効率化は AI [2] や産業用アプリケーション [3] などの応用がある。

## 2. 課題とアプローチ

### 2.1 SA の固定小数点化における課題

一般にプログラムの固定小数点化では、計算結果に生じる誤差を許容できる範囲に収める必要がある。例えば地球上の天気予報を行うプログラムでは、0.1 °C の計算誤差は許容できる可能性があるが 100 °C の誤差は許容できない可能性が高い。

SA の固定小数点化で許容できない誤差を生じる原因の一つは、除算の分母が 0 になりうることである。式 (5) および式 (6) において、分母の真値がごく小さな値のときその値の固定小数点化による近似値は桁落ちで 0 になる。分母が 0 の除算は意味をなさないため、それ以降の結果が得られずこれは許容できない。

### 2.2 課題に対するアプローチ

本研究ではランクの各値の正規化により SA の固定小数点化の必要ビット幅を削減する手法を提案する。ここで「必要ビット幅」とは、計算途中で除算の分母が桁落ちにより 0 にならない最小のビット幅と定義する。例えば通常の計算方法で分母に現れる最小値が $2^{-30}$、提案手法で分母に現れる最小値が $2^{-20}$ のとき、必要ビット幅を 30 ビットから 20 ビットに削減したと考える。

必要ビット幅削減のための基本アイディアは以下の 2 点である。

1. ランクの各値を正規化すると、除算の分母が大きく変化する。
2. ランクの各値を正規化しても、それらが昇順に並んでいる限り正しいソート結果を得る。

Towards Reducing Required Bit-width in Fixed-point Sinkhorn Algorithm for FPGAs. *Hiromasa Katayama and Soramichi Akiyama (Ritsumeikan University).*

**本研究は、JSPS 科研費 JP23K28078 の支援を受けたものである。**

例えばソートしたい値の集合が $\boldsymbol{x}=\{103,101,105\}$ の場合を考える。このとき通常ではランクは $\boldsymbol{b}=\{0,1,2\}$ であるから、$\boldsymbol{P}_{ij}=\exp(-\lambda(a_i-b_j)^2)$ の $(a_i-b_j)^2$ の項は $10^4$ 程度の値になり結果として $\boldsymbol{P}_{ij}$ は非常に小さな値になる。一方で $\boldsymbol{b}$ の各要素に 100 を加え $\boldsymbol{b}'=\{101,102,103\}$ とすると、$(a_i-b_j)^2$ は $10^1$ 程度に収まり、ソート結果も正しく $\{100,101,105\}$ の近似値になる。

### 2.3 提案手法：正規化によるビット幅削減

基本アイディアに基づき、ランクの平均値を入力の平均値に揃える手法を提案する。式 (5) および (6) の分母が桁落ちにより 0 にならないためには、少なくとも一周目（すなわち $\boldsymbol{P}=\exp(-\lambda\boldsymbol{C})$ が成り立つ時点）で分母が 0 でないことが必要である。このためには 式 (4) において $(a_i-b_j)^2$ を平均的に小さく抑えればよい。提案手法によりランクと入力の各値が平均的に近くなるため $\boldsymbol{C}_{ij}$ の値を小さく抑えられる。

具体的な計算方法は以下の通りである。ランクの各値の平均値が $\bar{b}$、入力の各値の平均が $\bar{a}$ のとき、ランクの各値 $\boldsymbol{b}_i$ を $\boldsymbol{b}'_i=\boldsymbol{b}_i-\bar{b}+\bar{a}$ と正規化する。例えば $\boldsymbol{a}=\{103,101,105\}, \boldsymbol{b}=\{0,1,2\}$ のとき、$\bar{a}=103, \bar{b}=1$ であるから $\boldsymbol{b}'=\{102,103,104\}$ となる。このとき一周目の $\boldsymbol{P}$ の正規化前の値 $\boldsymbol{P}_{\text{before}}$ と正規化後の値 $\boldsymbol{P}_{\text{after}}$ は以下のようになり、正規化により $\boldsymbol{C}_{ij}$ の値が大きく減少している。

$$\boldsymbol{C}_{\text{before}}=\begin{pmatrix}103^2 & 102^2 & 101^2\\ 101^2 & 102^2 & 99^2\\ 105^2 & 104^2 & 103^2\end{pmatrix} \tag{7}$$

$$\boldsymbol{C}_{\text{after}}=\begin{pmatrix}1 & 0 & 1\\ 1 & 4 & 9\\ 9 & 4 & 1\end{pmatrix} \tag{8}$$

## 3. 評価

提案手法によるビット幅削減を評価する実験を行う。入力データは 0 から 9 までの数値 3 つからなるベクトルであり、$\lambda$ の値は 1.5、反復回数は 100 とする。固定小数点化には Xilinx 社の提供する `ap_fixed` ライブラリ [4] を通常の C++ から使用する。これは高位合成ではなく CPU 上で動作するという意味である。また以下では提案手法を使わない計算方法を「通常手法」と呼ぶ。

図 1 に提案手法による分母最小値と必要ビット幅の変化を示す。分母最小値とは SA の計算途中で現れた除算の分母のうち最も小さい値である。提案手法によるビット幅削減は、通常手法および提案手法での分母最小値を $m_1, m_2$ として $\log_2(m_1)-\log_2(m_2)$ で計算する。横軸は入力データ、縦軸は左側が通常手法での分母最小値と提案手法での分母最小値、右側が提案手法による削減ビット幅である。横軸は提案手法による削減ビット幅削減の小さい順に並んでいる。なお入力データが全て昇順に並んでいるが、これは提案手法の有効性が入力データの順序に依存しないためである。

実験結果から以下の知見が得られる。

1. 入力データの最大値が大きいと通常手法での分母最小値が小さい傾向にあり、このとき提案手法によるビット幅削減効果が大きい。

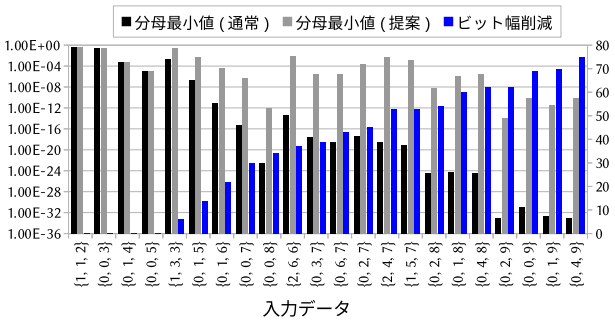


図 1: 提案手法による分母最小値と必要ビット幅の変化

2. 類似する入力データでも通常手法での分母最小値が大きく異なることがある。例えば $\{0,0,5\}$ と $\{0,0,9\}$ では通常手法の分母最小値が $10^{32}$ 程度異なる。
3. 提案手法では全ての入力データにおいて分母最小値を $10^{-12}$ 以上に抑え込めている。

## 4. 結論と今後の課題

本研究では SA を固定小数点化した際に除算の分母に 0 が現れない最大のビット幅を大幅に削減する手法を提案した。平均値ベースの正規化により必要ビット幅を最大で 70 ビット以上改善できることが分かった。

今後の課題の一点目はランクの**分散**の調整によるさらなる必要ビット幅の改善である。これはランクの各値は昇順に並んでいれば 1 刻みな必要はないことに基づく。例えば $\boldsymbol{b}=\{1,5,9\}$ でも、出力は入力の昇順ソート結果の近似値になる。この性質を利用し、入力値の分散が大きいときにはランクの分散も大きくすることが有効である。例えば $\boldsymbol{a}=\{30,0,777\}$ のとき、ランクの平均値のみを調整する場合に比べランクの分散も大きくすれば $(a_i-b_j)^2$ の最大値を小さく抑えられる可能性がある。このアイディアを用いた具体的な正規化手法および平均値ベース手法との両立が必要である。

今後の課題の二点目は、既存技術との比較である。SA のような数値計算アルゴリズムは数値的安定性のために計算方法を工夫する手法（例：小さな値での除算による誤差を防ぐために対数領域で計算する）が多数存在する。それらの手法の固定小数点化の必要ビット幅削減の有効性や本研究との関連性の議論が必要である。

## 参考文献

[1] Guy Eichler, Luca Piccolboni, Davide Giri, and Luca P. Carloni. Mastermind: Many-accelerator SoC architecture for real-time brain-computer interfaces. In *International Conference on Computer Design (ICCD)*, pp. 101–108, 2021.

[2] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning (ICML)*, pp. 1–10, 2015.

[3] Shuvangkar Shuvo, Eklas Hossain, and Ziaur Rahman Khan. Fixed point implementation of grid tied inverter in digital signal processing controller. *IEEE Access*, Vol. 8, pp. 89215–89227, 2020.

[4] AMD. Overview of arbitrary precision fixed-point data types. `https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Overview-of-Arbitrary-Precision-Fixed-Point-Data-Types`, 2022.