組込みシステムを対象とした並列実装及びコードの生成と移植に関する研究

２０２２年度　理工学研究科

博士学位請求論文（要旨）

組込みシステムを対象とした並列実装及びコードの生成と移植に関する研究

情報科学専攻
津山　雅彦

１　問題意識と目的

近年，計算機のクロック周波数の向上によるプログラムの動作速度の向上は頭打ちになりつつあり，プログラムの高速化を行うための手段としてソフトウェア側の最適化が主流となっている．しかし並列に動作するプログラムの動作や最適化なプログラムの設計には，専用の計算装置や計算機のアーキテクチャに関する知識が必要である．そこで，著者は以下の 2 つを目的として研究を行っている．

- 並列実装のエッジコンピューティングへの応用
- マシンコードの自動生成

著者が所属する研究室では運動中の選手にセンサを取り付け，センサネットワークを動的に構築し，スポーツシーンにおける生体情報取得のためのシステムを実現しようと試みている．そこで，研究室では画像情報に基づいてセンサの位置関係を把握し，ネットワークを構築する Image-Assisted Routing(IAR) を提案しているが，そのためにはリアルタイムな人検出手法が必要であり，それには Informed-Filters とその並列実装が有望であるという研究がある．電力や場所の限られた屋外ではデスクトップ PC 用の強力な GPU を搭載したサーバマシンを用いて Informed-Filters の並列実装をリアルタイムに動作させるのは困難である．そのため，「並列実装のエッジコンピューティングへの応用」は，並列実装された人検出手法を屋外で動作させるために必要なプロセスである．これを行う上で問題となるのはハードウェアの制約や低消費電力，移植先のプラットフォームの制約であり，元の実装の処理性能を保ちつつこれらの問題に対処しなければならない．

また，計算機アーキテクチャに関する知識がなくても最適なプログラムを得られるようにするためにはコンパイラによるプログラムの最適化が重要である．今日の計算機においては，定数の畳み込み，ループの展開といった典型的なコード変換が行われている．しかし，それだけでは人手での最適化に比べると効果が薄い．そこで近年ではプログラムの最適化に機械学習を導入する研究が行われている．そのためのアプローチとして gcc や LLVM といった既存のコンパイラを用いる手法が一般的であるが，著者は「マシンコードの自動生成」というアプローチでこの問題に対処しようと考えている．「マシンコードの自動生成」を行うためのアプローチとして，近年盛んに研究が行われている自然言語処理を用いる方法が考えられるが，自然言語処理を行うためには強力な GPU と莫大な電力が必要であるといった問題がある．

本論文では，「並列実装のエッジコンピューティングへの応用」を目的として Jetson および C++ AMP を用いた移植方法の 2 つの手法を，「マシンコードの自動生成」を目的として模倣学習を用いてマシンコードの生成・移植を行う 2 つの手法について提案する．

２　構成及び各章の要約

1 章においては，始めに著者が取り組んでいるプログラムの並列実装のエッジコンピューティング向け移

植の目的となる，運動中の選手の生体情報をリアルタイムにモニタリングするためのシステムについて述べる．そして，著者がもうひとつ取り組んでいるマシンコードの自動生成の目的であるプログラムの最適化を行う意義，マシンコードの生成に一見有効そうである自然言語処理の問題点について述べる．本論文における研究目的は生体情報モニタリングシステムの実現に向けた検出プログラムのエッジコンピューティング向けの移植および，計算リソースの問題を解決したプログラムの自動生成であり，以降の章においてこれらの目的を達成するために行った取り組みについて述べる．

2 章においてはプログラムの並列実装のエッジコンピューティング向け移植の目的となる Image-Assisted Routing (IAR) の核となる技術である画像内の物体検出の既存研究および技術について述べる．まずは，より一般的な画像内から物体を検出しそれに対してクラス分類を行う一般物体検出と呼ばれるタスクとそれに対して提案されてきた手法について述べる．そして，IAR では対象を限定して検出を行う特定物体検出を行うため，それに用いる Informed-Filters とその関する特徴量設計および本論文でエッジコンピューティング向けに移植を行う CUDA を用いた Informed-Filters の並列実装について述べる．

3 章においてはプログラムの生成に用いた模倣学習の手法である Neural Programmer-Interpreters(NPI) について，モデルの構造，アーキテクチャ，モデルによる推論，訓練手法について述べる．

4 章においては Informed-Filters を NVIDIA Jetson Xavier 上への移植を行い，人検出を Xavier 上で動作させるためのシステムの構築を行う．IAR で用いられるカメラから得られる画像の解像度は 3840 × 2160 である．そこで，その解像度でもリアルタイムに Informed-Filters が実行可能かつ単体で運用可能な Jetson の選定を行い，さらにカメラから Jetson に 4K 画像を入力するための HDMI 出力を UVC に変換する UVC ビデオキャプチャの選定も行った．その結果，3840 × 2160 の画像 1 枚あたり約 22 fps で処理できることがわかり，リアルタイム処理に十分な速度が示された．

5 章においては IAR に用いるドローン DJI Phantom 4 Pro V2.0 から画像を受け取り，Informed-Filters を用いてリアルタイムに処理をするための移植を行う．Phantom 4 に対し計算機から画像をリクエストするためには DJI Windows SDK を用いる必要があるが，DJI Windows SDK は Universal Windows Platform(UWP) 向けのライブラリであり，UWP のアプリケーションはセキュリティの関係でサンドボックス上実行されるため，CUDA を用いることができないという問題があった．そこで，本論文では UWP 上で GPU を用いて並列処理を行うための手段である C++ AMP を用いて Informed-Filters の並列実装を行い，実装したシステムの検出精度および実行速度の検証を行った．検出精度の評価では深層学習を用いた手法である EfficientDet-D0 および RetinaNet との比較を行った．EfficientDet-D0 および RetinaNet では通常通り訓練しても対象をほとんど検出できないため特別な方法で訓練を行ったが，その 2 つと比べても高い精度が示された．また速度の評価では約 42 fps で動作することが確認され，リアルタイム処理に十分かつシステムの他の部分で遅延が起きても問題がないことが示された．

6 章においては Neural Programmer-Interpreters を用いて RISC-V 命令セット向けのマシンコード生成手法の提案を行う．この手法はマシンコードの生成に向けた最初の手法である．この章では NPI の学習器の訓練方法を工夫することで，NPI を用いたソフト乗算を行うマシンコードの生成を実現する．本章では提案手法に先立って行った予備実験について述べその後に提案手法において行ったことを説明し評価を行う．予備実験では NPI の先行研究で実現されていた加算の筆算をベースとして乗算の筆算の実装を行ったが学習が安定せず，精度もあまりふるわなかった．著者らは予備実験の結果の原因を学習器への入力の多様化と考え，それをふまえて提案手法の実装を行った．提案手法では NPI でマシンコードの生成ができるようにするため，NPI において学習対象のタスクのステップとして与えられるサブプログラムのうち，即時サブプログラムを RISC-V 命令セットの命令に置き換えた．さらに置き換えた即時サブプログラムが実際の RISC-V 命令と同様に動作するように NPI において学習器の外部メモリとしての役割をする Scratch-Pad の改良を行った．提案手法における Scratch-Pad は CPU のレジスタ群としてふるまい，そこから出力される観測情報も RISC-V アーキテクチャの CPU のレジスタから自然に取得できるものとした．最後に提案手法を用いて訓練した NPI の学習器を用いて評価を行った．その結果学習に用いたデータが 5 bit 以内の乗算であったの

にも関わらず，32 bit の入力値からなる評価データセットを用いた評価において 100% の精度が得られた．

7 章においては 6 章で提案した訓練手法を拡張し,NPI を用いてコード移植器の構築を行う．これにより，Neural Programmer-Interpreters を用いたコード生成手法がより実用的なものになる．コード移植器は x86 64 から RISC-V へのマシンコードの変換を対象としている．そこで,本章では 6 章で実装した Scratch-Pad が RISC-V だけでなく x86 64 の命令も受けつけるように拡張し，さらに学習器出力するサブプログラムも両方のアーキテクチャ向けに出力できるように学習器訓練方法の変更を行っている．本章で提案する手法においては学習器の最初の訓練に加えて学習器が出力するプログラムのアーキテクチャを変更するための Fine-Tuning を行うが，その方法が問題となった．著者はこの問題に対して，訓練データにおいて即時プログラムを呼ぶ前に Wrapper サブプログラムを挿入することで解決することにした．ここで，最初の訓練は Wrapper サブプログラムを挿入した x86 64 向けのデータセットを用いて行い，Fine-Tuning は RISC-V 向けの Wrapper サブプログラムのデータセットを用いて行う．本章の提案手法に先立ち予備実験も行っている．予備実験では上記の工夫のみを施して訓練を行っているが，Wrapper サブサブプログラムを開始プログラムとして与えた際は RISC-V のマシンコードを出力するものの，本来学習器で行いたいソフト乗算の開始命令を開始プログラムとして与えた際は x86 64 のマシンコードが得られるという結果となった．著者はこの結果が学習器の内部状態によるものと考え，RISC-V 向けに作成した Wrapper サブプログラムのデータセットを x86 64 向けのデータセットの中に入れて訓練することにした．この手法を用いて訓練を行った学習器の評価を行ったところ，サブプログラムの置換はうまくいっていたがその引数がうまく学習できていないという結果になった．この結果から引数の問題を解決することで，NPI を用いてマシンコードの移植を行うためのプラットフォームの実現可能性が示された．