画像認識技術を応用した新しい自動並列化技術の検討

研究代表者　　和田　康孝　　明星大学　情報学部　准教授

1 はじめに

プロセッサのマルチコア化があらゆるコンピュータシステムで進んでおり，アプリケーションの並列化に対する要求・期待が高まっている．一般に，アプリケーションを並列化する際には，コンパイラによる自動並列化機能を利用するか,プログラマによる手動並列化を適用することになるが,それぞれ一長一短があり，両者の恩恵を同時に受けることは難しい.

従来の並列化コンパイラでは，形式的な中間言語に変換されたアプリケーションに対して数学的なモデルをベースにした解析を行い，その結果に基づいて並列実行可能な箇所を抽出する．プログラマの技量や知識に依存せずにある程度の並列化性能を得ることが可能である一方,コンパイラによる自動並列化においては，通常，入力として得られる情報がアプリケーションのソースコードのみであり，アプリケーションの実行結果を正しく保つためには常に保守的な判断を下さざるを得ない．そのため，アプリケーションを短時間で簡単に並列化できるが，プログラマによって手動で並列化したものと比較して性能が低くなることが多い.

自動並列化と手動並列化，つまりコンパイラと人間によるアプリケーション並列化への取り組み方の違いについて比較してみると，１）アルゴリズム等，ソースコードの外にある情報を得られるかどうか，２）過去の他のアプリケーションに適用した並列化手法の効果に関する知見の有無，３）ソースコードを視覚的に捉えているかどうか，の３点が主な違いとしてあげられるものと考えられる．プログラマがアプリケーションを並列化する際には，アプリケーションのアルゴリズムを理解した上で視覚的にプログラムの構造を把握し，過去の他のアプリケーションへの適用結果なども用い経験的に判断・決定している．より高い性能が得られる反面，高いスキルと長い作業時間を要する.

以上の観点から，本研究では，機械学習，特に深層学習のように，知識や経験を数値化・モデル化して活用することができるような仕組みを並列化コンパイラにおいても利用し，自動並列化技術にも手動並列化の要素を取り入れることを目的として，その初期検討を行った．特に，プログラムを画像として捉え，近年精度の向上が著しい画像認識技術を取り入れる可能性について検討を進めた.

2 関連研究

近年，コンパイラの最適化技術にも機械学習が徐々に活用され始めている[1,2]．ここでは関連する研究・技術について述べ，本研究との差異について考察する.

2-1 プログラムの自動生成とコンパイラ最適化

ソフトウェアのコンパイル時に適用する最適化機能は通常，コンパイラを用いる際にオプションによって選択される．その際，通常は”-03”や”-fast”のように，いくつかの有効な最適化機能をまとめたオプションが利用される．しかし実際は，これらの代表的なオプションは経験的に有効と考えられている最適化をまとめただけのものであり，個別の最適化手法をどのような順番と組み合わせで適用するかによってプログラムの実行性能が大きく異なる場合がある．最適化手法の最適な組み合わせと適用順をあらかじめ知ることは難しく，個々の最適化手法においても，どのようなパラメータでそれぞれが実行されるべきかは状況によって異なる.

これらの問題を解決するため，各種最適化の適用順と組み合わせを機械学習によって決定する手法[1,2]や，最適化の効果を高めるため，既存のプログラムを学習データとして用い，その学習結果から多数のプログラムを合成して最適化効果の向上に役立てる仕組み[3,4]などが提案されている．また，グラフとして表現されるプログラムの特性や，パフォーマンスカウンタ等を用いて取得したプログラム実行時の特性を用いて機械学習を適用し，対象プログラムに対して適切なコンパイルオプションの指定を行う手法[15]も提案されている.

2-2 機械学習技術の並列プログラム最適化への適用

また近年では，CPU のみならず，GPU や FPGA などのアクセラレータを備えたコンピュータシステムが広く利用されるようになってきている．このようなシステムにおいては，複数あるいは多数の特性・性能の異なる計算資源が利用可能であるため，どのような処理をどの計算資源上で実行するべきかを考慮して処理割り当てを行う必要がある．しかしながら，一般的には，CUDA や OpenCL といった，これらのアクセラレータを活用するために必要なプログラミング言語への移植に大きなコストを払う必要があり，そういった細かいチューニングを行うことは難しい．そこで，アクセラレータで実行可能なプログラムの特性を数値化して SVM（サポートベクターマシン）によってモデル化し，これによって，CPU つまり汎用プロセッサ上で実行した方がより性能が得られるか，アクセラレータ上で実行すべきか，を判定する手法が提案されている[5].

2-3 本研究の位置付け

しかしながら，上記で述べた手法はいずれも手動あるいは自動で並列化された結果に対して実行性能を高める，あるいは逐次実行における性能を高めるための最適化手法の選択を行う，というものである．本研究のようにプログラムの並列化可能箇所を抽出するためのものではない．また，従来の提案手法では対象のプログラムを一度実行した結果から得られる特徴量を用いることが一般的であり，本研究のように，事前にプログラムを実行することなく並列化・最適化を適用する手法は少ない．本研究では，人間がアプリケーションを最適化する際の手順や考え方を機械学習の手法を通じて導入することで，より効率の良い自動並列化を実現することを目的としている．

3 学習データの生成

機械学習，特に深層学習を有効に活用するためには，膨大な量の学習データ，教師データが必要であるとされる．例えば，画像認識技術の競技会として広く知られている ILSVRC（Large Scale Visual Recognition Challenge）[9]では，1000 のカテゴリに分けられた画像を計 100 万点以上用いて学習処理を行う．一般に画像分類の入門として広く扱われている MNIST でも，6 万点の学習データが用意されている[10]．一方，コンパイラの性能評価に用いられるベンチマークプログラムの数はせいぜい十～数十程度であり，機械学習に活用するには十分ではないということが考えられる．ここでは，本研究で用いた学習データの生成・準備について述べる．

3-1 ディープラーニングを用いた C プログラム生成の検討

上述の通り，一般に機械学習，特に深層学習には多くの学習データが必要となるが，通常のベンチマークプログラムを元に抽出をするだけでは，十分な数が得られないことが考えられた．そこで，本研究ではまず CLgen[3, 4]を改変し，深層学習によってループを主体としたプログラムを生成することで，十分な数の学習データを用意することを試みた．

CLgen は本来 OpenCL のカーネルプログラムを入力として深層学習を行い，学習結果を元に新たな OpenCL カーネルを合成するものである．多数生成される出力プログラムから，さらに OpenCL の文法やセマンティクスに合致しているもののみを抽出して利用する構造になっている．

本研究では，OpenCL のカーネルプログラムの代わりに C プログラムを学習データとして用いることで，C プログラムを生成することを目的として実験を行ったが，その結果，現実的な時間では十分な数の C プログラムを生成することができなかった．この原因としては，1)学習に用いた入力プログラムの不足，2)用いた LSTM ネットワーク形状のチューニング不足，3)OpenCL に対して C は記述の自由度が高く，「文法に適った記述」を学習することが難しいこと，などが考えられる．

3-2 ベンチマークを用いた学習データの生成

3-1 で述べた通り，CLgen を用いた学習プログラムの合成には多大な時間を要し，必要な数の学習データ，つまり C プログラムを現実的な時間で得ることが現時点では難しいことがわかった．そこで，改めてベンチマークプログラムから抽出する方法ととることとした．ただし，一般のベンチマークプログラムからループ部分のみを抽出して学習データとするのは数量および作業量の面から難しい．そこでここでは，多数のプロ

グラム片からなるカーネルベンチマークのうち，特にループ最適化，並列化，ベクトル化のために用意されているものを用いることとした．

（1）TSVC を元にしたプログラム片の抽出

本研究では，学習データを作成するにあたり，C で記述された”Test Suite for Vectorizing Compiler”（TSVC）[6]を用いた．TSVC はコンパイラのベクトル化の機能テスト・評価のために開発されたベンチマークセットであり，およそ 150 の関数から構成されている．それぞれの関数はループを含んでおり，コンパイラの最適化によってコードの削除等が行われないよう，ループ内にはダミーの関数呼び出しが存在する．

この TSVC の実装から，個別の関数を切り出してそれぞれ単一のファイルとすることで，本研究で用いる学習データを準備した．なお，オリジナルのコードには時間計測や結果確認，表示等のコードが含まれるが，これらは本研究の目的においては学習の際のノイズとなりうるため削除した．その結果，151 のプログラム片が抽出された．

（2）LLVM を用いた中間言語への変換

C によるプログラム表現そのままを学習データとして利用すると,C による記述の自由度の高さからプログラムごとの記述の揺らぎが大きくなり，これがノイズとなって学習がうまくいかないことが考えられる．例えば，C は構文によって括弧の有無が統一されない，必ずしも文ごとに改行することを要求しない，といった記述の曖昧さを含む．そこで本研究では， C プログラムにおける細かい記述の差異を隠蔽・削除し，プログラムの意味自体を抽出できるよう，コンパイラの中間言語に変換しこれを用いた．

様々なコンパイラがオープンソースソフトウェアとして利用可能であるが，本研究では LLVM[8]を用いることとした．LLVM は商用・非商用問わず広く利用されていること，中間言語（LLVM IR）に変換された入力プログラムを簡単にテキストファイルとして得られること，LLVM IR は特定の言語や環境によらない構造をしており，プログラム上の記述の揺らぎを削除できること，などが LLVM を選択した理由として挙げられる．

さらに，LLVM で生成した中間言語には，変数宣言やコンパイラのバージョン情報など不要な情報が記載されているため，それらを取り除き，対象とするループ構造を含む関数部分のみになるようにデータを整形した．これにより，対象ループを含む関数の呼び出しと復帰に関する部分は全ての入力データに共通の部分となり，対象ループ部分のみが個々の学習データ間の差異として残ることになる．対象とする関数部分のみ取り出す手順を図 1 に示す．

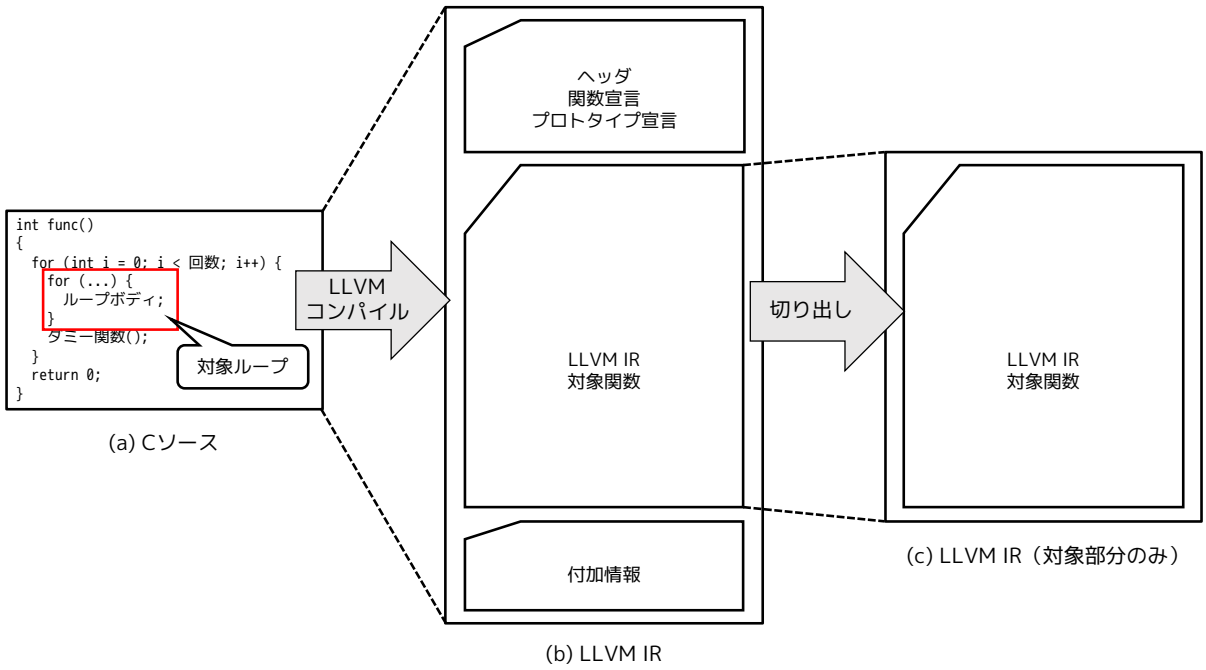

図 1．学習データの生成

（3）画像ファイルへの変換

3-2（2）で示した手順で生成された LLVM IR はあくまでもテキストファイルであり，このままでは，深層学習の分野でも特に技術的進展の著しい画像認識技術を適用することは難しい．そのため，テキストファイルを画像に変換し，学習データとしてこれを用いる．

画像への変換には ImageMagick 7.0.8[14]を用い，テキストファイルを png フォーマットの画像に変換した．変換後の画像においては，フォントサイズ 12 ポイント，背景色は白，文字色は黒とした．また，LLVM IR の仕様上 1 行あたりの記述が長大になってしまう場合があるため，100 文字で強制的に改行を行った．その結果生成された画像の最大サイズは，縦およそ 1000 ピクセル，横およそ 700 ピクセルであった．

（4）データセットの構築

本研究では，ループの並列化可否を判定することを目的としている．そのため，教師データとして，3-2（3）で述べた手続きによって生成された学習データを分類してラベル付けする必要がある．そのためにここでは，PGI Compiler Community Edition [7]を用いた．PGI コンパイラは自動並列化およびベクトル化の機能を持ち，その適用の可否を確認することができるようになっている．ここでは，3-2（1）で抽出した対象プログラムそれぞれに対して PGI コンパイラの並列化およびベクトル化の機能を適用し，その結果に応じて”parallel”および”not_parallel”の２通りのラベルを付与して分類した．多くの場合，ベクトル化が可能なループはループ並列化も適用可能であると考えられるため，PGI コンパイラがベクトル化を適用すると判断したループを含む学習データを並列化可能と判断し，”parallel”とラベル付けを行った．それ以外のデータについては並列化不可，あるいは並列化の効果が得られないと判断し，”not_parallel”とラベル付けを行った．

4 評価実験

4-1 評価環境および評価条件

3-2 で述べた手順によって生成した学習データを 8:2 の割合でランダムに学習データと検証データに分けて入力データとし，学習と検証を行った．なお，本評価では Neural Network Console[11]上で，これに付属している LeNet[12]をベースに，上述の学習データが利用可能となるように入力を 3×1024×768，出力を 1 に改変したものを対象のネットワークとして用いた．また，より複雑な構造を持つ GoogLeNet[13]も同様に Neural Network Console に付属のものを改変して評価を行った．GoogLeNet については，入力は同様に 3×1024×768，出力は分類に合わせて 2 とした．また，2 値分類問題となることを考慮し，一部の活性化関数を Softmax から Sgimoid に置き換えた．深層学習における各種パラメータは，使用した学習データ数に合わせ，バッチサイズ 30，エポック数 8 とした．

4-2 評価結果と考察

4-1 で述べた評価環境および評価条件に基づいて実験を行なった結果を表 1 に示す．

表 1．評価結果

ネットワーク	LeNet	GoogLeNet
判定精度	0.52	0.52
最小学習エラー率	0.40	0.00
最小テストエラー率	0.39	0.47

今回は，問題設定が「並列化可能なループを含むか否か」という２値の問題であるため，表 1 から，特に LeNet においては，今回得られた学習精度は不十分なものである．また，GoogLeNet は過学習あるいはいずれかの層で値の発散が起こっているものと考えられ，正しく学習が行えているとは言えない．そこで，学習データを作成する際に，対象ループを含む関数ではなく，対象ループそのものが該当する部分のみを抽出したものを新たに用意し，これを用いて同様の評価実験を行った．その結果を表 2 に示す．表 2 から，特に大きな改善は見られず，学習の効果が上がらない原因は個々の学習データにおける表現の問題ではないことが

わかった.

表 2. 学習データ改善後の評価結果

ネットワーク	LeNet	GoogLeNet
判定精度	0.45	0.55
最小学習エラー率	0.47	0.00
最小テストエラー率	0.41	0.47

学習の効果および判定精度が向上しない主な原因としては，用意できた学習データの不足が主な原因として考えられるが，そのほかにも，いずれの場合も"parallel"あるいは"not_parallel"のどちらかに偏って判定されており,並列化の可否によるプログラムの差異が画像データ上ではそれほど大きな差異として現れていないことが考えられる．表１および表２において，判定精度に多少の差異はあるが，これは評価データに含まれる"parallel"・"not_parallel"それぞれのデータの数に多少の差異があるためであり，実質的に学習データの改善によって判定精度の向上があったとは言えない.

これらの結果と原因を考慮して，精度の向上に向けていくつかの方策が考えられる．例えば，プログラミング言語においては，全体の視覚的な形状とともに，各箇所がどのような予約語や構文，変数にと対応しているかが重要な情報となる．しかしながら，深層学習における画像認識において広く用いられている畳み込みニューラルネットワーク（CNN，Convolutional Neural Network）では，畳み込み処理によってそのような細かい情報が消えてしまったことが原因の１つと考えられる．特に本評価では，学習データを画像化する際に白黒の単純な画像としたため，実質的に色情報を活用できていないという問題がある．今後，予約語や変数の種類等によって色分けして表現するなどの試みが必要になるものと考えられる．また，今回は標準的な LeNet および GoogLeNet の構造を入力層のみ画像サイズに合わせて拡張して利用したため,ネットワーク全体のバランスに欠けていることも考えられる．今後，他の形状のネットワークを活用することも重要である.

5 まとめと今後の課題

本研究では，従来の並列化コンパイラで用いられていた数学的モデル，プログラミング言語の仕様に基づいたアプリケーションの解析結果による並列性の抽出方法に加え，知識・経験ベースの手法を取り入れることでより効率よくアプリケーションから並列性を抽出することを目的として，特に画像認識技術を並列化可否の判断に利用する方法について検討を進めた.

機械学習，特に深層学習を用いた手法においては膨大な量の学習データが必要となるが，ベクトル化コンパイラ向けのベンチマークをベースに用意した学習データをもとに CNN を用いた分類を行ったところ，今回利用した学習データおよび学習の手法では，十分な精度を得ることができなかった．現時点では実用に足る認識精度は得られていないものの，その原因や改善の方向性を明らかにすることができた.

今後の課題としては，より多くの学習データを用意し深層学習を適用するに十分な状態とすること，学習データ生成における作業の自動化，画像としてではなく文字情報として学習データを取り扱うことで自然言語処理の手法を取り入れること，などが挙げられ，特に自然言語処理技術を活用する手法について取り組みを進めている．また，学習データの拡充のためにそのデータベース化を行い，今後公開することも視野に入れて取り組んでいきたいと考えている．そのほか，LLVM などの実際のコンパイラへの機能組み込みも実用化のためには重要な課題である.

【参考文献】

[1] A. H. Ashouri, W. Killian, J. Cavazos, G. Palermo, and C. Silvano, "A survey on compiler autotuning using machine learning", CoRR, abs/1801.04405, 2018.

[2] A. H. Ashouri, G. Palermo, J. Cavazos, and C. Silvano, "Automatic Tuning of Compilers Using Machine Learning", Springer International Publishing, 2018.

[3] C. Cummins, P. Petoumenos, Z. Wang, and H. Leather, “Synthesizing benchmarks for predictive modeling”, In Proceedings of 2017 IEEE/ACM International Symposium on Code Generation and Optimization, pages 86–99, February 2017.
[4] CLgen: Deep Learning Program Generator, https://github.com/ChrisCummins/clgen
[5] A. Hayashi, K. Ishizaki, G. Koblents, and V. Sarkar, “Machine-learning-based performance heuristics for runtime CPU/GPU selection.”, In Proceedings of the Principles and Practices of Programming on The Java Platform, pages 27–36, September 2015.
[6] TSVC test suite, http://polaris.cs.uiuc.edu/~maleki1/TSVC.tar.gz
[7] PGI Community Edition, https://www.pgroup.com/products/community.htm
[8] The LLVM Compiler Infrastructure, https://llvm.org/
[9] Large Scale Visual Recognition Challenge (ILSVRC), http://image-net.org/challenges/LSVRC/
[10] The MNIST database, http://yann.lecun.com/exdb/mnist/
[11] Neural Network Console, https://dl.sony.com/ja/
[12] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-Based Learning Applied to Document Recognition”, In Proceedings of the IEEE, Vol. 86, No. 11, pages 2278-2324, November 1998.
[13] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, “Going Deeper with Convolutions”, CoRR, abs/1409.4842, 2014.
[14] ImageMagick, http://imagemagick.org/script/index.php
[15] Eun Jung Park, “Automatic selection of compiler optimizations using program characterization and machine learning”, PhD Thesis, University of Delaware, 2015.