

SIMD プログラミングのための共通インタフェースの 提案および実装

中西 悠[†] 渡邊 啓正[†] 本多 弘樹[†]

電気通信大学 大学院情報システム学研究科[†]

1. はじめに

近年のプロセッサにはデータ処理速度を飛躍的に向上させる SIMD 命令が用意されている。しかし、SIMD 命令セットはアーキテクチャによって異なるため汎用的なプログラミングを行うことが困難である場合が多い。本研究では SIMD プログラミングにおける共通の API を提案し、実装した。この API によってコードの可搬性を損なうことなく明示的に SIMD を用いることが可能となる。

2. SIMD プログラミング

SIMD 命令はレジスタを 1 つのベクトルとみなし、ベクトルの各要素に対して同じ演算を行う。このような演算をベクトル演算と言い、多量のデータに対して同種の演算を繰り返し行う必要がある場合に効果的である。

現在、SIMD 命令を利用する方法は以下の 3 つに大別することができる。

- コンパイラによる自動生成
- ライブラリ関数の利用
- アセンブリ言語、組み込み関数の利用

それぞれの特徴と問題点を以下に述べる。

2.1 コンパイラによる自動生成

この方法はプログラマの負担が少ない。しかし、SIMD 命令が利用されるのが限定的である場合が多く、これら命令の十分な活用は難しい¹⁾。また、コンパイラそのものの開発コストが大きい²⁾ため多数のアーキテクチャについてコンパイラを用意するのが困難である。

2.2 ライブラリ関数の利用

この方法は SIMD 命令が利用されているライブラリ関数を用いる方法である。コンパイラによる自動生成と同様にプログラマの負担は少ない。また、ライブラリ内部で効果的に SIMD が利用されている。代表的なものとして ATLAS²⁾のようなものがある。しかし、多数のアーキテクチャについてこれらライブラリが用意されているとは限らない。機能もある程度高いレベルで抽象化されていることが多

く、柔軟性に乏しい。

2.3 アセンブリ言語、組み込み関数の利用

この方法は各アーキテクチャに密接したプログラミング手法である。よって細やかな最適化を施すことが可能である。しかし、コードの可搬性は著しく低下し、アーキテクチャやコンパイル環境が限定されてしまう。

2.4 SIMD プログラミングの問題点

上記のように、それぞれの方法は問題を抱えており、コードの可搬性を落とさずに SIMD 命令を効果的に利用するのは困難である。

3. 共通 API の提案とツールの設計

本研究ではコードの可搬性を低下させることなく SIMD プログラミングを行うための共通の API 及びこの API を解釈するツールを提案する。この API は比較的低いレベルで機能の抽象化を行い、組み込み関数と同様に高い柔軟性を持たせる。

API は C/C++ 言語で記述されたソースコード内にプログラマによって記述される。また、対象のアーキテクチャの情報は外部ファイルに記述する。ツールは記述されたプログラムを認識し、中間表現に変換する。その中間表現に対して最適化などの処理を行い、その後に通常のコンパイラが読み取れる形に変換する(図 1)。現在、本ツールは対象のコンパイラを gcc/g++ としている。アセンブリ言語の出力はこれらコンパイラで用いられている拡張インラインアセンブラを利用している。

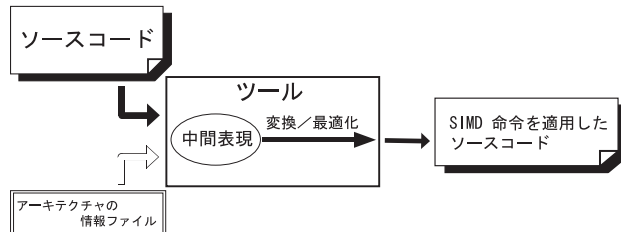


図 1 ツールの概要

4. ツールの実装

SIMD 命令列を生成するにあたり、問題となるのは以下の点である。

- 命令の選択
- ループのベクトル化
- コード最適化

Design and implementation of a common interface for SIMD programming

[†] Yu Nakanishi, Hiromasa Watanabe, Hiroki Honda

Graduate School of Information Systems, The University of Electro-Communications

● 制約の回避

これらの問題の解決方法を以下に述べる。

4.1 命令の選択

ツールは入力された API から対象となるアーキテクチャに従った適切な命令を選択する必要がある。まず、ツールは SIMD 命令の生成に必要な変換規則を記述した外部ファイルを読み込む。次に入力された API から対応した変換規則を選択し、中間表現に変換する。対象のアーキテクチャに対応する命令が存在しない場合、ツールはこの演算をスカラー演算と見なし、同義の C コードを出力する。その後の最適化はコンパイラに委ねられる。

4.2 ループのベクトル化

命令選択の結果、ループ内にベクトル演算とスカラー演算が混在してしまう可能性がある。この時、ツールは出力されるベクトル演算のベクトルサイズにあわせループの展開を行い、スカラー命令によって実行結果が変化しないようにスカラー命令を配置する。

4.3 コード最適化

記述された API をそのままアセンブリ言語で出力すると、多くの場合無駄な演算やロード、ストアが生じる。特にメモリアクセスの遅延は大きい。ため、こうした無駄なコードはできるだけ省く必要がある。本ツールでは読み込まれたソースコードを中間表現に変換し、通常のコンパイラと同様にコードの最適化を⁴⁾行う。最適化は冗長式の削除や共通部分式の削除など基本的なものを行う。また、レジスタ割付けも、できるだけ最適に近い割付けをするようにし、レジスタ不足が生じないように努める。

4.4 制約の回避

アーキテクチャによって SIMD 命令の実行条件が存在する。特に問題となるのはメモリアライメント⁵⁾で、一部の SIMD 命令セットではロードやストアの対象となるメモリのアライメントが特定の数で揃っている必要がある。この問題はツール及びコンパイラによるアライメントの調整と確認により解決を行う。

5. ツールの動作確認と評価

API を用いて行列積、高速フーリエ変換(以下 FFT)を記述し、ツールの動作確認を行った。また、ツールによって生成されたコードの実行速度を計測した。対象となる SIMD 命令セットは SSE, Emotion Engine(以下 EE) VU, 3DNow!を選択した。これらについて同一のソースコードから各種 SIMD 命令を適用したコードへの変換を行う。ただし、

環境に応じて必要なデバイスの制御などは別途記述している。また、実行速度の評価比較の対象としてインラインアセンブラによる人手での SIMD 命令の適用も行い、実行速度を計測した。

それぞれの実行環境を以下に示す。

- SSE Pentium4 2.8GHz
- EE(VU) Emotion Engine 300MHz
- 3DNow! Opteron 1.4GHz

行列積の実行結果を表 1, FFT の実行結果を表 2 に示す。

表 1 行列積の実行結果[単位:sec]

	SIMD 命令 未使用	ツールによる SIMD 命令の適用	人手による SIMD 命令の適用
SSE	0.64	0.36	0.23
EE(VU)	9.64	5.94	4.63
3DNow!	1.01	0.83	0.52

表 2 FFT の実行結果[単位:sec]

	SIMD 命令 未使用	ツールによる SIMD 命令の適用	人手による SIMD 命令の適用
SSE	1.01	0.42	0.28
EE(VU)	24.93	10.52	6.53
3DNow!	1.03	0.82	0.71

この結果から同一のソースコードから各種 SIMD 命令の適用に効果があることがわかる。しかし、人手で SIMD 命令を適用したコードに比べると遅いことから、最適化などに改善の余地があると思われる。

6. まとめと今後の課題

本研究では SIMD プログラミングを行うための共通の API 及びツールを提案、実装した。その結果、同一のソースコードから各種 SIMD 命令の適用にある程度効果があることがわかった。今回利用した命令は積和、差およびデータのロード、ストア命令がほとんどであったが、他の命令についても検討する必要がある。また、他の命令セットや実行環境に対しても検討する必要がある。

参考文献

- 1) Aart J. C. Bik, Milind Girkar, Paul M. Grey, Xinmin Tian: Automatic Intra-Register Vectorization for the Intel Architecture, International Journal of Parallel Programming, Volume 30, Issue 2, pp65-98 (2002).
- 2) R.Clint Whaley, Antoine Petit, Jack J.Dongarra : ATLAS Project <http://math-atlas.sourceforge.net/>
- 3) Aho, A. V., Sethi, R., Ullman, J. D. : Compilers - Principles, Techniques, and Tools, Addison-Wesley(1986).
- 4) Peng Wu, Alexandre E. Eichenberger, Amy Wang : Efficient SIMD Code Generation for Runtime Alignment and Length Conversion, CGO '05 pp153 - 164 (2005).