2Y-02

WebAssembly を x86-64 に変換可能なトランスレータの試作と評価

松本 怜也† 平山 勇太‡ 水村 総之介§ 早川 智一¶
明治大学理工学部† 明治大学理工学研究科‡ 明治大学理工学研究科§ 明治大学理工学部¶

1. はじめに

Web ブラウザ（以下，ブラウザ）上で高速な処理を行う技術として WebAssembly（以下，Wasm）が存在するが，実行環境によっては Wasm が JavaScript よりも低速に動作するという課題が存在する．複数のブラウザで JavaScript と Wasm の比較を行った研究 [1] では，ブラウザによっては JavaScript の方が速い場合と Wasm の方が速い場合があることが報告されている．

この課題を元に我々は以前の研究 [2] で，Wasm の高速化を支援する実行環境の切り替えが容易なフレームワークの提案を行った．提案フレームワークでは，サーバサイドにおける実行環境として，Java Virtual Machine（以下，JVM）を採用することで高速化を実現した．

しかし，より直接的な変換先である x86-64 アセンブリを対象とした変換・実行については検討が不足しており，JVM が実行環境として最適であるとは断言できない．

そこで我々は，Wasm を x86-64 アセンブリに変換可能なトランスレータの試作を行い，提案フレームワークへの組込みを行った．そして，x86-64 アセンブリに変換した場合のフレームワークの実行性能を中心に様々な再評価を行い，フレームワークにおける実行環境の再検討を行った．

2. 関連研究

Titzer ら [3] は，コンパイル時間とメモリ消費に着目した Wasm の高速化のためのインタプリタを提案している．本研究とは Wasm の高速化を目的としているという点で類似しているが，サーバサイド実行を高速化の手段としている点で提案手法が異なる．

3. 提案手法

本論文では，Wasm を x86-64 アセンブリに変換した場合の実行性能について言及するために，Wasm を x86-64 アセンブリに変換可能なトランスレータを提案する．Wasm の変換対象として x86-64 アセンブリを選択した理由としては，(1) Intel 製の CPU のシェア率が高く，(2) 先行研究で採用した JVM と比較して，実行に必要な工程が少ないため，高速な処理が期待できる——と考えたためである．

図 1 に提案トランスレータを組み込んだフレームワークの概要を示す．我々の以前の研究では，Wasm のサーバサイドの実行環境としてフレームワークの中で Node.js と JVM が選択可能であった．提案トランスレータをフレームワークに組み込むことで，Wasm のサーバサイドの実行環境のさらなる選択肢として x86-64 アセンブリに変換した実行が加わり，実行性能の比較が可能となる．

4. 設計

提案トランスレータの設計の概要を図 2 に示す．このシステムは次の流れで動作する：(1) バイナリ形式の Wasm をテキスト形式の Wasm である WebAssembly Text Format（以下，WAT）に変換する，(2) パーサが WAT から抽象構文木を生成する，(3) 抽象構文木から x86-64 アセンブリを生成する．

表 1 に WAT と x86-64 アセンブリの対応を示す．提案トランスレータは，WAT を 1 命令ごとに x86-64 アセンブリへと変換する．しかし，WAT

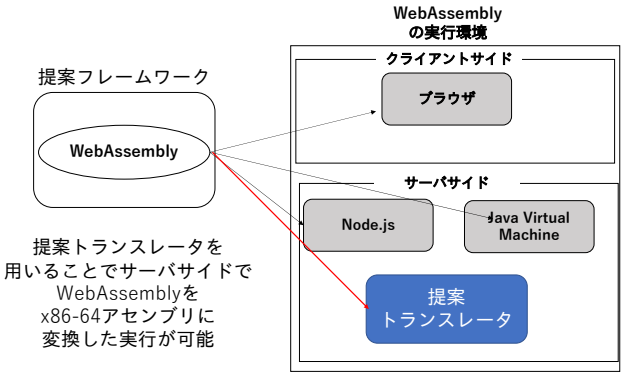

図 1 組み込みを行ったフレームワークの概要

An evaluation of translator that enables convert WebAssembly into x86-64

†Reiya Matsumoto, School of Science and Technology, Meiji University, ee197060@meiji.ac.jp
‡Yuta Hirayama, Graduate School of Science and Technology, Meiji University, y_uta@cs.meiji.ac.jp
§Sonosuke Mizumura, Graduate School of Science and Technology, Meiji University, m_sonosuke@cs.meiji.ac.jp
¶Tomokazu Hayakawa, School of Science and Technology, Meiji University, t_haya@cs.meiji.ac.jp

Copyright ©2023 Information Processing Society of Japan.
All Rights Reserved.

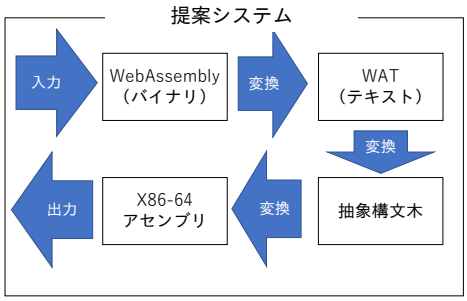

図２　提案トランスレータの設計の概要

表１　WAT と x86-64 アセンブリの対応

変換前の WAT	変換後の x86-64 アセンブリ
i32.const 1	push $1
if 処理 A end 処理 B	pop %rax cmp $1, %rax jne L1 処理 A L1: 処理 B

の命令セットは x86-64 アセンブリの命令セットよりも高級であり，x86-64 アセンブリの 1 命令で表現しきれない場合がある．x86-64 アセンブリの 1 命令で表現しきれない命令は，複数の命令を組み合わせることで等価な変換を行っている．

5.　評価

本論文では，提案トランスレータを先行研究のフレームワークへ組込み，その再評価を行った．

以前の研究では，以下の 3 つのアルゴリズムを用いて様々な環境でフレームワークの評価を行った：(1) 小銭の払い方，(2) ハノイの塔，(3) 分割数．提案トランスレータを組み込んだ先行研究のフレームワークでも同様の評価を行った．

表２に評価結果の一部を示す．この表はハノイの塔に関する提案フレームワークの実行時性能を示しているが，JVM に比べ，トランスレータが生成した x86-64 アセンブリは， 10 倍低速化していることがわかる．その他のアルゴリズムにおいては，小銭の払い方では 13 倍，分割数では 6 倍低速となり，提案トランスレータの生成する x86-64 アセンブリを利用したフレームワークの高速化が困難であることがわかった．

これは，提案トランスレータが出力する x86-64 アセンブリには最適化がかかっておらず，x86-64 アセンブリに変換した際のパフォーマンスを十分に発揮しているとは言えないためである．そのため表 2 の結果のみでは，フレームワークのサーバサイドの実行環境として，x86-64 アセンブリに変換した実行が不適切であるとは言い切れない．

表２　ハノイの塔の実行結果 1（ms）

実行環境	Google Chrome	Mozilla Firefox	Microsoft Edge	Opera
JavaScript	43,506	36,882	23,479	23,320
Wasm	14,561	11,632	16,600	17,173
クライアントサイド Wasm	13,594	11,664	16,714	16,727
サーバサイド NODE	13,091	11,901	11,975	11,805
サーバサイド JVM	2,307	2,569	2,361	2,554
サーバサイド x86-64	24,406	24,689	23,837	26,487

表３　ハノイの塔の実行結果 2（sec）

	変換時間	実行時間
x86-64（トランスレータ）	0.096	21.886
x86-64（gcc -O0）	0.126	21.087
x86-64（gcc -O1）	0.13	5.141
x86-64（gcc -O2）	0.133	2.962
x86-64（gcc -O3）	0.141	3.016
JVM	0.299	3.447

そこで，我々は Wasm を C に変換した後に，gcc を利用した最適化を行い，x86-64 アセンブリにコンパイルした場合の実行時性能を調査した．

表３に変換に gcc を利用した場合の実行結果を示す．この表は，トランスレータによる x86-64 アセンブリと gcc を利用した x86-64 アセンブリ・JVM でハノイの塔を実行した場合の実行時性能を示している．-O1 を除いた gcc の最適化を用いて生成された x86-64 アセンブリと JVM とでは，同等程度の速度であることがわかる．

この評価結果からサーバサイドの実行環境として，x86-64 アセンブリに変換した実行と JVM の両方が，Wasm の高速化を支援する選択肢として選択可能であると我々は考える．

6.　おわりに

本論文では，Wasm を x86-64 アセンブリに変換可能なトランスレータの試作と評価，先行研究の実行環境の再検討を行った．その結果，Wasm のサーバサイドの実行環境として x86-64 アセンブリに変換した実行と JVM の両方が妥当であるという結果を得ることができた．今後の課題として，トランスレータへの最適化の導入が挙げられる．

参考文献

[1] Yan, Y., Tu, T., Zhao, L., Zhou, Y. and Wang, W.: Understanding the Performance of Webassembly Applications, *Proceedings of the 21st ACM Internet Measurement Conference*, New York, NY, USA, Association for Computing Machinery, pp. 533–549 (2021).

[2] 水村総之介，早川智一：WebAssembly の高速化を支援する実行環境の切り替えが容易なフレームワークの提案と試作，第 30 回マルチメディア通信と分散処理ワークショップ論文集，pp. 141–148 (2022).

[3] Titzer, B. L.: A Fast In-Place Interpreter for WebAssembly, *Proc. ACM Program. Lang.*, Vol. 6, No. OOPSLA2 (2022).

Copyright ©2023 Information Processing Society of Japan.
All Rights Reserved.