様　式　Ｆ－１９－２

科学研究費助成事業　　研究成果報告書

令和　　６ 年　　６ 月 ２０ 日現在

機関番号：１２６０１
研究種目：国際共同研究加速基金（国際共同研究強化(A)）
研究期間：2019～2023
課題番号：１８ＫＫ０３１５

研究課題名（和文）個々のアプリケーションに特化した組込みシステム向けメモリ管理技術

研究課題名（英文）Application-Specialised Memory Management for Embedded Systems

研究代表者
鵜川　始陽（Tomoharu, Ugawa）

東京大学・大学院情報理工学系研究科・准教授

研究者番号：５０４２３０１７

交付決定額（研究期間全体）：（直接経費）　　8,800,000円
渡航期間：　10ヶ月

研究成果の概要（和文）：本研究では，IoT機器などの組込みシステム向けのJavaScript VMを構成する技法として，アプリケーションに特化させることでメモリフットプリントを抑える技法を開発した．特に，オブジェクトの動的型を表現するデータ構造であるhidden class treeを，事前にアプリケーションを実行して得られたプロファイリングをもとに，オフラインで最適化する技法を開発した．また，hidden classのようなメタオブジェクトを含むヒープをコンパクションするごみ集め(GC)や，遺伝的アルゴリズムでインタプリタの命令ハンドラを並びかえて高速化する技法も開発した．

研究成果の学術的意義や社会的意義
本研究で開発した技法は，JavaScriptプログラムのメモリフットプリントを削減するものであり，IoT機器のようなメモリが限られた組込みシステムで，従来より大きなJavaScriptプログラムを実行できる．また，本研究で開発したJavaScript VMであるeJSVMは簡素で変更が容易なため，今後のプログラミング言語の研究のベースに利用できる．
本課題は海外の研究者と交流を深めることも目的としており，本課題を通して研究代表者のみならず，本研究に携った学生（研究代表者の指導学生でない学生も含む）も海外の共同研究者と交流した．

研究成果の概要（英文）：This research has developed techniques for constructing JavaScript VM for embedded systems such as IoT devices. These techniques reduce the memory footprint by specializing VM to individual applications. One of the techniques optimizes the hidden class trees, which represent dynamic types of objects, offline based on profiling obtained by running the target application in advance. We also developed garbage collection (GC) that compacts the heap containing meta-objects like hidden classes and technique to improve performance by reordering instruction handers of interpreters by using the genetic algorithm.

研究分野：プログラミング言語

キーワード：メモリ管理　JavaScript　フラッシュメモリ　ストレージストラテジ　遺伝的アルゴリズム　組込みシステム　ガーベージコレクション　hidden class

様　式　Ｆ－１９－２

１．研究開始当初の背景

組込みシステムのプログラムはC言語などの低級言語で記述されることが多い．低級言語のプログラムは手間がかかり，専門知識を要するため，IoT応用のアイデアを試作する妨げになっていた．JavaScriptはスクリプト言語であり，プログラムの試作に適している．また，JavaScriptはWebアプリケーションの記述言語として広く受け入れられており，プログラマが多い状況にあった．しかし，JavaScriptの言語仕様は巨大で，JavaScriptプログラムを実行するために必要なJavaScript VMは大きくなりがちで，一般的なJavaScript VMは，メモリが100KB程度のメモリしか持たない組込みシステムでの実行は難しかった．組込みシステムのプログラムをJavaScriptで記述できるようにするためには，JavaScript VMを小型化する必要があった．組み込みシステム向けに開発されていたJavaScript VMであるJerryScriptは実行ファイルが600KB程度，V7は300KB程度だった．また，実行中に必要になるワーキングメモリであるヒープ領域も小型化が必要だった．

２．研究の目的

IoT機器などの組込みシステムに適した小型のJavaScript VMを開発することと，それを通してVMの新しい構成法を開発することが本課題と基課題で共通する目的である．本課題では，特にヒープ領域の小型化に注目し，組込みシステムに適したオブジェクトモデル（メモリ上でオブジェクトを表現するためのデータ構造）とメモリ効率の良いコンパクションを行うガーベージコレクション（GC）を開発することを目的とした．

３．研究の方法

組込みシステムで実行するプログラムはシステムの開発時に決まり，そのプログラムだけがシステム上で実行される．これは，デスクトップコンピュータやブラウザなどの一般的なプログラム実行環境と大きく異なる点である．本研究では，この点に注目し，VM自身に対して特定のアプリケーションに特化させる，profiling guided optimization (PGO)を行う．場合によっては，対象としたアプリケーション以外は実行できないような最適化も許容することで，通常の最適化を超える性能向上が得られる手法を研究した．

具体的には，まず，ベンチマークプログラムの整備を行った．このために，渡航先の共同研究者が開発していたベンチマークプログラム集を我々が開発している組込みシステム向けのJavaScript VMであるeJSVMに移植した．以降の研究では，このベンチマークプログラム集を用いて評価した．

次に，渡航先でオブジェクトモデルの最適化について研究を始めた．まず，実行中にアプリケーションのプロファイリングを行い，それに基づいて実行中に最適化する通常の動的最適化を実装し，その上で，プロファイリング結果を使ってオフラインでアプリケーションにより特化した最適化を行う方法を考案した．途中，通常の動的最適化の実装を終えたところで新型コロナウィルス感染症の流行により帰国を余儀なくされたため，続きの研究は海外の共同研究者とオンラインの打ち合わせを行いながら進めた．

帰国後，並行してコンパクションを行うGCの研究や，アプリケーションに特化したVMを開発するためのドメイン特化言語(DSL)を国内の研究者とも協力して行なった．DSLの研究は，本課題の目的であるメモリの小型化とは直接は関係ないが，我々が開発しているeJSVMのメンテナンス性を高めるために必要であった．オブジェクトモデルの最適化の研究の後，引き続きオンラインで海外の共同研究者と，インタプリタの実行速度の高速化にPGOを利用する研究を行なった．

４．研究成果

(1) Hidden classのインラインキャッシングを用いた最適オブジェクトサイズの決定

JavaScriptのような動的なプログラミング言語では，プログラムの実行中にオブジェクトのフィールド（JavaScriptではプロパティと呼ばれる）が追加される．そのため，newキーワードによってオブジェクトを生成する時点では，そのオブジェクトが将来どれだけフィールドを持つかが分からず，オブジェクトにどれだけのメモリを割り当てればよいかが，一般には分からない．そこで，動的言語では一般に，オブジェクトを固定長のヘッダ部，そこからポインタで指されるフィールドの配列の二つのデータ構造で実現し，フィールドの追加に応じてフィールドの配列をより大きな配列に置きかえられるようにする．この場合，フィールドの配列を実際のフィールド数よりも少し余裕を持った大きさにしておかなければ，フィールド配列の再割り当てが頻繁に起こって性能が大幅に低下してしまう．一方で，余裕を持った大きさにすると余分にメモリが必要になるため，組込みシステムでは避けたい．さらに，ヒープ中のデータ構造にはGCのためのヘッダが必要で，オブジェクトを二つのデータ構造で表すこと自体が空間的なオーバヘッドになる．このオーバヘッドは，オブジェクトが小さいと相対的に大きくなる．本研究では，実行中に過去の実行の経歴からオブジェクトが将来持つ可能性があるフィールドの数を予測して，それがちょうど収まる大きさのメモリを割り当てる方法を提案した．この予測が当たっている限り，フィールド配列の再割り当ては必要ないので，フィールド配列をヘッダ部に埋め込むことで，オブジェクトを二つのデータ構造で表すことによるオーバヘッドも削減

した.
フィールド数の予測には，同じアロケーションサイト（プログラム中のオブジェクトを生成する位置であり，プログラムの new の出現位置に対応する）で作られたオブジェクトは同じフィールドの集合を持つ可能性が高く，そうでなくてもフィールドの集合のバリエーションは少ないことが多いという経験則を利用した. 高性能な JavaScript VM では, 各オブジェクトが hidden class と呼ばれる，オブジェクトの動的な型（オブジェクトが持つフィールドの集合）を表すデータ構造へのポインタを持つことで，実行を効率化している．オブジェクトにフィールドが追加されると，そのオブジェクトは追加されたフィールドを持つ hidden class を指すようになる．提案手法では，オブジェクト毎に，そのオブジェクトが作られたアロケーションサイトを記録しておき，GC のタイミングで各オブジェクトが持つ hidden class を，そのオブジェクトを作ったアロケーションサイトにキャッシュする．これにより，以降，そのアロケーションサイトでオブジェクトを作るときは，キャッシュされた hidden class の型が持つフィールドをちょうど全て格納できる大きさのメモリをオブジェクトに割り当てることができる．この基本的な手法に加え，最終的に追加されるフィールドのうち一部のみしか持たない期間が長いオブジェクトでは，将来追加されるフィールドのために割当てておくメモリが無駄になる問題や，最終的に追加されるフィールドの集合に複数のバリエーションがある場合に対処した.
この成果は，仮想機械に関する国際会議 MoreVMs 2021 で発表した.

(2) 事前プロファイリングに基づく hidden class tree の最適化

この研究では(1)で行なった動的な最適化をオフラインで行なった. オフラインで行うことにより，より強力な最適化を行うことができる．この研究では(1)で提案したアロケーションサイトへの hidden class のキャッシュに加え，hidden class を管理するデータ構造の最適化を行なった.
フィールドが追加された時, 追加されたフィールドを持つ hidden class を高速に検索できるように，hidden class は，それにフィールドを追加して作られる hidden class をポインタで指している. 一般には, ある hidden class に追加されるフィールドには複数の可能性があるので, hidden class は hidden class tree と呼ばれる木構造を構成することになる．実際に hidden class tree が分岐するときには(1)の方法は完全には働かず，分岐点の hidden class をキャッシュすることになる．そうすると，オブジェクトは分岐点までのフィールドしか保持できない大きさで作られ，以降に追加されるフィールドは，外部のフィールド配列に割り当てられる．このフィールド配列はフィールドの追加の度に再割り当てされ，オーバヘッドとなる．また，(1)の研究では考慮していなかったが，hidden class 自身が占めるメモリも無視できないオーバヘッドになる.
提案手法では, 事前に行なったプロファイリングに基づき，オフラインで hidden class tree を変形することで, 分岐の数や hidden class の数を削減した．図 1 の①を最適化前の hidden class tree とする．ここで，円が hidden class，矢印がフィールドの追加に対応する辺であり, 太さはそのフィールドが追加される頻度を表す. まず，hidden class tree が分岐する箇所については, その分岐先の hidden class の使用頻度を参照する. 使用頻度が極端に低い分岐があれば, それを使用頻度が高い分岐の先端に移動させ，分岐のない直線的な hidden class tree にする．これにより②の hidden class tree になる. 次に, 一時的にしか使われない hidden class を取り除き, hidden class tree を短くする．この背景には，実際のプログラムでは複数のフィールドが連続して追加されることが多いという観察がある．例えば，３次元空間の点の座標を表すオブジェクトでは，三つの軸座標の値 x，y，z は連続して設定される．このような場合，一つや二つの座標軸の値だけを持つ hidden class は過渡的な状況でしか使われない．そこで，過渡的な状況でしか使われない hidden class を取り除いて，プログラム上で x が追加された時，同時に y，z も追加されたこ

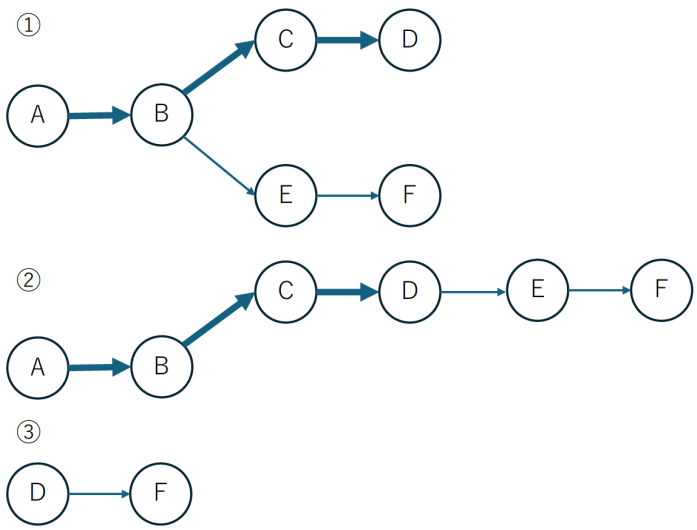

図 1 hidden class tree の最適化

とにする．一時的にしか使われない hidden class かどうかは，その hidden class を同時に参照するオブジェクトの数によって決定する．これにより，③のような hidden class tree が得られる．
このような最適化をアプリケーションに合わせて実行前に構築することで，消費するメモリを平均で 62%削減した．また，この手法で hidden class のバリエーションが減ることにより，インラインキャッシュのヒット率が向上するという利点があることも確認できた．
この成果は，仮想機械に関する国際会議 VMIL2022 で発表した．

(3) Hidden class と通常のオブジェクトが混在するヒープに対するコンパクション

様々な大きさのメモリの割り当てと解放を繰り返すと，空き領域が細切れになり，連続領域が確保できなくなる．この症状をフラグメンテーションと呼ぶ．空き領域を詰めてオブジェクトをヒープの片方に寄せることによりフラグメンテーションを解消する処理をコンパクションと呼ぶ．コンパクションでは，個々のオブジェクトの大きさレイアウトを表す情報が必要になる．これらは hidden class に格納されている．Hidden class も通常のオブジェクトと同じヒープに格納することで，これらを別々のメモリ領域に格納する場合に比べて無駄が少なくなるが，コンパクションでは hidden class と通常のオブジェクトと同時にコンパクションする必要がある．しかし，従来のコンパクションアルゴリズムでは，コンパクション用に追加の領域を使わない限り，コンパクション中のオブジェクトの中身を参照することはできなかった．そのため，オブジェクトのレイアウトを表す hidden class を参照することができず，コンパクションができなかった．

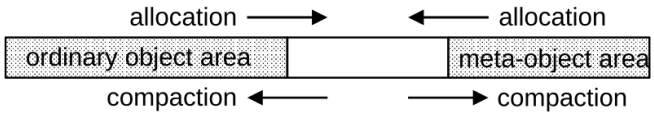

図 2 ヒープの両端に一般オブジェクトと hidden class (meta-object)を配置

本研究では，通常のオブジェクトと hidden class（meta-object）を同じヒープの右端と左端に詰めて配置することで，これらを同じヒープに配置しながら，別々にコンパクションを行うことを提案した．この時のヒープは図 2 のようになる．ヒープの左端から右に向かって通常のオブジェクトを割り当てる．Hidden class はヒープの右端から左に向かって割り当てる．両者の割り当てがぶつかると GC とコンパクションを行う．コンパクションでは，通常のオブジェクトは左に，hidden class は右に向かってスライドさせる．通常のオブジェクトのレイアウト情報が必要な処理が終わるまでは hidden class は移動させず，あとから hidden class を移動させる．
提案手法により，IoT アプリケーションを模したベンチマークプログラムで，従来 eJSVM に実装されていたマークスイープ GC では 48KB のヒープが必要だったところが 28KB で実行できるようになった．
この成果は，メモリ管理に関する国際会議 ISMM2021 で発表した．

(4) 遺伝的アルゴリズムを用いたバイトコードハンドラの並べ替え

バイトコードインタプリタは，バイトコード命令を解釈実行するインタプリタループを持つ．インタプリタループは，命令を解釈した後，その命令を処理する命令ハンドラに制御を移して，命令を処理する．命令ハンドラの実行が終わると，その場で次の命令を解釈し，次の命令ハンドラに制御を移す．この時，次の命令ハンドラがバイナリファイル中でどこにあるかで，実行時間が変化する．例えば，次の命令ハンドラが直後や近くにあると，CPU のキャッシュなどが効率よく働くと考えられ，実際，実行速度が向上する．プログラム中に頻繁に現れる命令列に対応するように命令ハンドラを並べ替えれば良いが，命令数が多く，解析的に最適な命令ハンドラの順序を求めるのは現実的ではない．さらに，キャッシュサイズや分岐予測器の仕組みなどは CPU によって異なり，それによって最適な命令ハンドラの順序は異なる．
本研究では，遺伝的アルゴリズムによって発見的に最適な命令ハンドラの順序を求めた．その結果，各アプリケーションに最適化された命令ハンドラの順序を用いることで，平均で 7.7%の高速化を達成した．さらに，あるアプリケーションで最適化した順序を別のアプリケーションに適用したところ，別のアプリケーションでも速度の向上が見られた．一方で，ある CPU に最

適化した順序を用いて別の CPU で実行したところ，有意な速度向上は見られなかった．
この成果は，国際会議 SAC2023 のプログラミング言語のトラックで発表した．

５．主な発表論文等

〔雑誌論文〕　計5件（うち査読付論文　5件／うち国際共著　1件／うちオープンアクセス　0件）

１．著者名 Huang Wanhong、Marr Stefan、Ugawa Tomoharu	４．巻 -
２．論文標題 Optimizing the Order of Bytecode Handlers in Interpreters using a Genetic Algorithm	５．発行年 2023年
３．雑誌名 Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing	６．最初と最後の頁 1384-1393
掲載論文のDOI（デジタルオブジェクト識別子） 10.1145/3555776.3577712	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 該当する

１．著者名 永谷 龍彦、鵜川 始陽	４．巻 40
２．論文標題 ストレージストラテジーによる組み込み向けJavaScript バーチャルマシンのメモリ使用量の削減	５．発行年 2023年
３．雑誌名 コンピュータ ソフトウェア	６．最初と最後の頁 4_54～4_66
掲載論文のDOI（デジタルオブジェクト識別子） 10.11309/jssst.40.4_54	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

１．著者名 Yuta Hirasawa, Hideya Iwasaki, Tomoharu Ugawa, Hiro Onozawa	４．巻 30
２．論文標題 Generating Virtual Machine Code of JavaScript Engine for Embedded Systems	５．発行年 2022年
３．雑誌名 Journal of Information Processing	６．最初と最後の頁 679-693
掲載論文のDOI（デジタルオブジェクト識別子） 10.2197/ipsjjip.30.679	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

１．著者名 小野澤 拓, 岩崎 英哉, 鵜川 始陽	４．巻 38
２．論文標題 アプリケーションと実行環境に適応したカスタマイズが可能なJavaScript処理系	５．発行年 2021年
３．雑誌名 コンピュータソフトウェア	６．最初と最後の頁 3_23-3_40
掲載論文のDOI（デジタルオブジェクト識別子） 10.11309/jssst.38.3_23	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

１．著者名 Onozawa Hiro、Ugawa Tomoharu、Iwasaki Hideya	４．巻 -
２．論文標題 Fusuma: double-ended threaded compaction	５．発行年 2021年
３．雑誌名 Proceedings of the 2021 ACM SIGPLAN International Symposium on Memory Management	６．最初と最後の頁 94-106
掲載論文のDOI（デジタルオブジェクト識別子） 10.1145/3459898.3463903	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

〔学会発表〕　計7件（うち招待講演　0件／うち国際学会　2件）

１．発表者名 Tomoharu Ugawa, Stefan Marr, Richard E. Jones
２．発表標題 Profile Guided Offline Optimization of Hidden Class Graphs for JavaScript VMs in Embedded Systems
３．学会等名 The 14th ACM SIGPLAN International Workshop on Virtual Machines and Language Implementations（国際学会）
４．発表年 2022年

１．発表者名 Zihan Li, Tomoharu Ugawa, Ryota Shioya
２．発表標題 Cooperative Memory Management of a JavaScript Virtual Machine with Datatype based Hardware Memory Deduplication
３．学会等名 日本ソフトウェア科学会 第39回大会
４．発表年 2022年

１．発表者名 近森凪沙，高田喜朗，鵜川始陽
２．発表標題 フラッシュメモリを持つマイコン向けJavaScript仮想機械の文字列管理
３．学会等名 第24回プログラミングおよびプログラミング言語ワークショップ
４．発表年 2022年

１．発表者名
Tomoharu Ugawa, Stefan Marr, Ricahrd Jones

２．発表標題
Caching Hidden Classes for Pre-transitioning Object Memory Layout in JavaScript

３．学会等名
Workshop on Modern Language Runtimes, Ecosystems, and VMs 2021（国際学会）

４．発表年
2021年

１．発表者名
小野澤 拓，鵜川 始陽，岩崎 英哉

２．発表標題
オブジェクトレイアウトを表すメタオブジェクトを含むヒープに対するスレッド化コンパクション

３．学会等名
第23回プログラミングおよびプログラミング言語ワークショップ

４．発表年
2021年

１．発表者名
近森 凪沙，高田 喜朗，鵜川 始陽

２．発表標題
JavaScript処理系eJSのMbedへの移植

３．学会等名
令和２年度電気・電子・情報関係学会四国支部連合大会

４．発表年
2020年

１．発表者名
Tomoharu Ugawa, Richard E. Jones, Stefan Marr

２．発表標題
Pre-forming Object Shapes for In-Object Field Allocation in eJS JavaScript VM

３．学会等名
プログラミングおよびプログラミング言語ワークショップ

４．発表年
2020年

〔図書〕　計0件

〔産業財産権〕

〔その他〕

―

６．研究組織

	氏名 （ローマ字氏名） （研究者番号）	所属研究機関・部局・職 （機関番号）	備考
	Ｊｏｎｅｓ　Ｒｉｃｈａｒｄ (Jones Richard)	Ｔｈｅ　Ｕｎｉｖｅｒｓｉｔｙ　ｏｆ　Ｋｅｎｔ・School of Computing・Professor	共同研究期間中に退職、以降名誉教授
	Ｍａｒｒ　Ｓｔｅｆａｎ (Marr Stefan)	Ｔｈｅ　Ｕｎｉｖｅｒｓｉｔｙ　ｏｆ　Ｋｅｎｔ・School of Computing・Senior Lecturer	

７．科研費を使用して開催した国際研究集会

〔国際研究集会〕　計0件

８．本研究に関連して実施した国際共同研究の実施状況

共同研究相手国	相手方研究機関			
英国	University of Kent			
英国	University of Kent			