McLab: An extensible compiler toolkit for MATLAB and related languages

Andrew Casey, Jun Li, Jesse Doherty
Maxime Chevalier-Boisvert, Toheed Aslam, Anton Dubrau
Nurudeen Lameed, Amina Aslam, Rahul Garg
Soroush Radpour, Olivier Savary Belanger, Laurie Hendren
Clark Verbrugge

Sable Research Group,
School of Computer Science, McGill University
Email: [acasey, jli127, jdoher1, mcheva, taslam, adubra, nlamee,
aaslam1, garg, sradpo, osavary, hendren, clump]@cs.mcgill.ca
Website: `http://www.sable.mcgill.ca/mclab`

Abstract

MATLAB is a popular language for scientific computation. Effectively compiling MATLAB presents many challenges due to the dynamic nature of the language. We present McLab, an extensible compiler toolkit for the MATLAB and related languages. McLab aims to provide high performance execution of MATLAB on modern architectures while bringing modern programming concepts such as aspect-oriented programming and other extensions to MATLAB. McLab consists of several components. The first component is an extensible frontend to parse and analyze MATLAB as well as extensions to MATLAB. The second component, called McFor, is a compiler to translate a static subset of MATLAB to FORTRAN. The third component, McVM, is a virtual machine including a JIT compiler to execute MATLAB code. Finally we also provide language extensions such as AspectMatlab. We present the current state of the implementation of McLab and describe ongoing work and future directions of the project.

Keywords: Compiler, MATLAB*, programming languages, scientific computing, JIT compilation.*

1. INTRODUCTION

MATLAB®[1] is currently one of the most popular languages for scientific and numerical computing. MATLAB is known for its flexible handling of arrays, simplicity of learning and its dynamic features. However, there remain many challenges. First, new compilation and virtual machine techniques are required to get faster execution which can also take advantage of modern processors. Second, new language support is required for the effective expression of general purpose data structures and algorithms beyond arrays and numeric codes.

The McLab project aims to combine the ease and familiarity of the core MATLAB language with a more modern feature set based upon the work of programming language theory and compiler communities. McLab provides an extensible set of compilation, analysis and execution tools built around the core MATLAB language to experiment with new language features and compiler optimizations. The extensibility of the toolkit has already been demonstrated with an aspect-oriented programming extension to MATLAB language implemented within McLab.

2. COMPONENTS OF MCLAB

McLab is built as a set of reusable, extensible and mostly independent components and thus provides the compiler community an opportunity to use the components within their own projects.

Figure 1 shows the overall structure of McLab. The shading in the components indicates the maturity of each component. The darker shaded components have complete initial implementations and the unshaded components are our future plans. In the remain-

[1] MATLAB is a registered trademark of The Mathworks, Inc. (`www.mathworks.com/products/matlab/`).

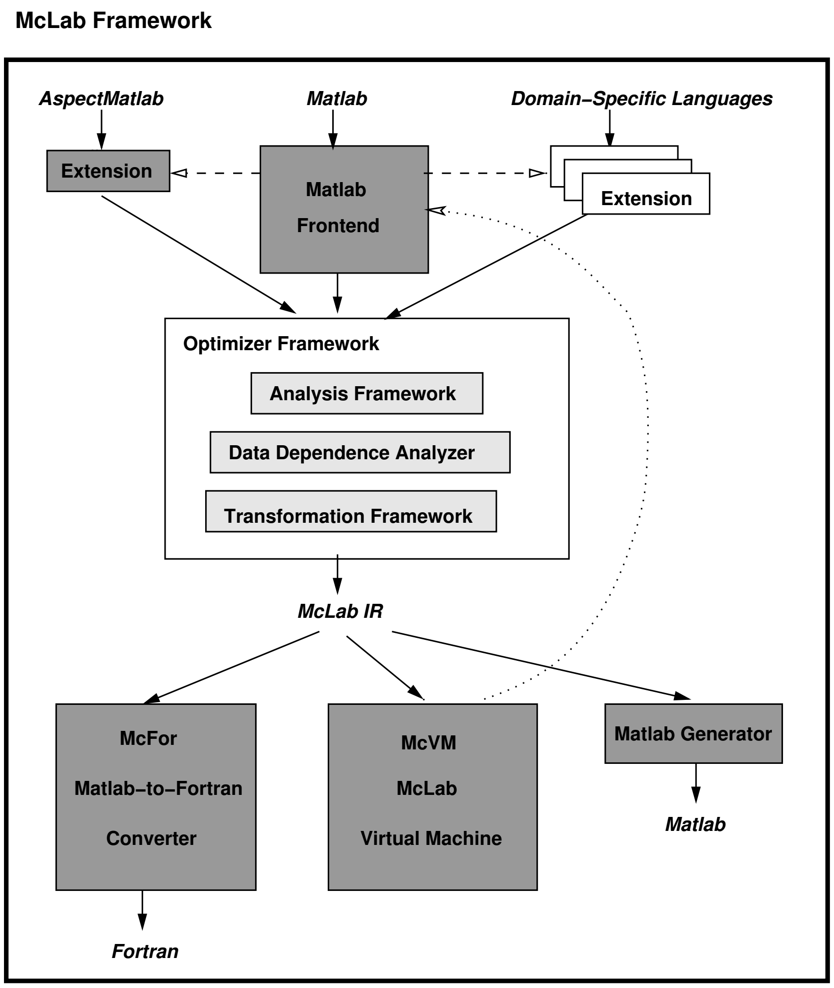

Figure 1: Structure of McLab

ing sections we discuss the components in more detail.

2.1 Frontend and static analysis

The first component of McLab is an extensible lexer and parser framework for MATLAB language. The framework is built around the JastAdd[6, 1] compiler compiler and our own extensible lexer called Metalexer[3, 2]. The second component is a static analysis framework which provides a framework for writing static flow analysis passes. The lexer, parser and static analysis system together constitute the McLab frontend. The frontend first translates MATLAB to a clean subset of MATLAB we call Natlab. The subset is then converted into a tree-based intermediate representation we call McIR. Our static analysis and transformation framework operates on McIR. Apart from traditional compiler optimizations, we have also utilized the static analysis framework in implementation of our AspectMatlab[8, **?**] extension and continue to look into opportunities for usage in other tools.

A new component of the frontend is a loop analysis and transformation framework. Typically loop dependence analysis requires the knowledge of loop bounds but these are typically not available statically in a language such as MATLAB. The basic aim of loop analysis framework is to avoid expensive run time dependence analysis tests by predicting the loop bounds which are required for data dependence testing based on the profiled information and do most of the computations at compile time. It consists of a profiler to record loop bounds from program runs, a heuristic engine to predict loop bound ranges based on profiling information as well as a dependence analyzer and loop transformer.

2.2 McFor

McLab provides two different backends: McFor[7] and McVM[4, 5]. McFor is a backend of McLab compiling statically to Fortran. It supports a subset of MATLAB with fewer dynamic features, which allows more aggressive static analyses like type inference and can thus produce highly-optimized Fortran code. This allows tapping into the high performance of existing advanced Fortran compilers.

Although MATLAB has too many dynamic features to allow for the full MATLAB language to be compiled efficiently to a static language like Fortran, we are pushing the envelope and attempting to handle as much of the language as possible. This effort is leading us to understand more about the tradeoffs between static and dynamic languages and is leading to an interesting subset of MATLAB that can be effectively automatically translated to Fortran. Our previous work has focused on compiling to Fortran 95. We have now started exploring the possibility of utilizing the polymorphic capabilities provided in newer versions of Fortran. These capabilities have only recently been implemented in Fortran compilers.

2.3 McVM

McVM is an high-performance virtual machine for the MATLAB programming language. It includes a generic interpreter and a type-specialization based JIT compiler. The VM performs a range of classical optimizations and some more advanced dynamic optimizations, including advanced type-inference optimization to achieve high performance.

MATLAB poses many new challenges in VM implementation and we continue to develop new sets of analysis and optimization techniques to increase performance. One such problem posed by MATLAB is that MATLAB has copy semantics for array assignment operator. However many such copies are not necessary. For example, if array A has been assigned to an array B, and if neither array is modified after the assignment, then there is no need to create a copy of B and all references to array A can then be replaced by references to array B. We are implementing a new analysis algorithm in McVM to identify the avoidable copies thus eliminating the overhead of unnecessary copies. Another optimization under development is inlining and polymorphic inlining caching implementation for McVM. Polymorphic inline caching will cache the last results of call dispatch for each call site. These cached dispatch results will then be inlined. Inlining will make function calls faster by removing the overhead associated with it (i.e. creating stack-frame, copying parameters) and will allow us to do use intra-procedural analysis and optimizations on the program instead of having to write more complex inter-procedural ones. Apart from the optimizations mentioned, we are also exploring the adaptation of techniques such as on-stack replacement for application in McVM.

We have also started a research project to extend McVM to generate code for multi-core CPUs and GPUs. GPUs have recently emerged as a high-performance alternative to CPUs for certain classes of highly parallel numerical codes such as dense matrix operations, n-body codes and many DSP algorithms. However, GPUs currently are typically situated on a PCI-express bus and have their own separate on-board memory and thus the programming model becomes heterogeneous instead of a homogeneous parallel programming model utilized for typical multi-core CPUs. Further, current GPU computation APIs such as OpenCL are low-level and require the programmer to write a different optimized version for each target architecture.

We aim to provide a much simpler programming paradigm providing a uniform parallel programming model for a wide class of homogeneous and heterogeneous parallel architectures. The compiler will be responsible for generating the optimized code for each architecture and will provide a simpler virtualized view of memory. We are currently investigating loop analysis and transformations and the runtime support required to support such a virtualized view of resources. We are also investigating loop transformations to generate efficient code for multiple architectures by taking memory hierarchy, parallel execution resources and SIMD instruction sets of each architecture into account.

2.4 Language extensions to Matlab

One of the central goals of McLab is to provide convenient programming abstractions to the scientific programmer. We aim to implement many new extensions to MATLAB providing both general purpose extensions as well as domain-specific extensions. We have implemented one such new extension called ASPECTMATLAB, an aspect-oriented extension to MATLAB [8, **?**].

ASPECTMATLAB brings aspect-oriented programming to the scientific programming community. It is designed to be simple to use and understand and it is also focused towards the needs of the numerical programming domain by providing loop-level extensions apart from more classical aspects provided in other languages. We have already demonstrated the utility of ASPECTMATLAB in writing aspects which provide better insight to the programmer about performance and correctness of the program.

3. RESULTS

We tested the performance of McVM against MATLAB on a number of benchmarks[5]. Testing was done on a Core 2 Quad Q6600 running Ubuntu 9.10 and MATLAB R2009a. Results are reported in Table 1. Column 2 presents absolute runtimes in seconds. Columns 3,4 and 5 present execution time relative to McVM JIT. In columns 3,4 and 5, a value of less than 1 indicates better performance than McVM JIT and a value of greater than 1 indicates worse performance than McVM JIT. McVM outperforms MATLAB in 8 out of 20 cases tested. JIT compilation in McVM provides over two orders of magnitude performance improvement over a pure interpreter. McFor was able to compile 13 of the benchmarks to Fortran and compiled Fortran code outperformed MATLAB in 12 out of 13 cases.

AspectMatlab system has been used for writing aspects such as counting the number of floating point operations and determining the sparsity of matrices used in a computation. Such aspects are useful for scientists and engineers attempting to understand the

Table 1: Results

Benchmark	McVM (JIT)	MATLAB	McVM (no JIT)	McFOR
	seconds	Relative to McVM JIT		
adpt	13.4	0.20	0.94	0.05
beul	3.07	1.01	0.51	N/A
capr	3.51	2.31	478	0.36
clos	6.84	0.11	1.99	1.15
crni	1321	0.01	1.35	0.0026
dich	2.80	1.68	410	0.67
diff	30.0	0.17	1.39	0.021
edit	54.9	0.20	1.48	0.0023
fdtd	20.1	0.17	0.43	0.014
fft	12.8	1.27	193	0.72
fiff	5.37	1.30	285	0.18
mbrt	34.6	0.13	2.84	0.03
nb1d	4.1	2.40	1.03	0.18
nb3d	3.88	0.40	0.65	0.23
nfrc	15.7	0.32	1.66	N/A
nnet	6.95	0.91	1.05	N/A
play	3.37	2.57	1.26	N/A
schr	2.48	0.84	1.22	N/A
sdku	1.23	7.93	13.1	N/A
svd	8.24	0.29	0.85	N/A

characteristics of a program.

4. CONCLUSIONS

McLab provides an extensible toolkit for compiler construction for MATLAB and related languages. One extension to MATLAB, AspectMatlab, has already been provided. A FORTRAN backend and a VM to execute MATLAB code has also been provided. We continue to improve upon the tools already built and are working on new optimizations as well as improving our compatibility to MATLAB. We are also working on taking the project to new directions such as compilation to GPUs and exploring new extensions to MATLAB.

It is our hope that our tools will provide a framework for other compiler teams to make progress on developing new language extensions and new compiler techniques for languages related to MATLAB. We also hope that the actual compilers and VMs produced using the toolkit will be useful for scientists by providing new language abstractions and better performance, especially for modern processors.

5. REFERENCES

[1] JastAdd. `http://jastadd.org/`.

[2] Metalexer. `http://www.sable.mcgill.ca/metalexer/`.

[3] T. Aslam. AspectMatlab: An aspect-oriented scientific programming language. Master's thesis, McGill University, 2010.

[4] A. Casey. The MetaLexer lexical specification language. Master's thesis, McGill University, September 2009.

[5] M. Chevalier-Boisvert. McVM: An optimizing virtual machine for the MATLAB programming language. Master's thesis, McGill University, August 2009.

[6] M. Chevalier-Boisvert, L. Hendren, and C. Verbrugge. Optimizing MATLAB through just-in-time specialization. In *International Conference on Compiler Construction*, March 2010. To Appear.

[7] T. Ekman and G. Hedin. The Jastadd extensible Java compiler. In *OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, pages 1–18, New York, NY, USA, 2007. ACM.

[8] J. Li. McFor: A MATLAB to FORTRAN 95 compiler. Master's thesis, McGill University, August 2009.

[9] A. D. Toheed Aslam, Jesse Doherty and L. Hendren. AspectMatlab: An aspect-oriented scientific programming language. In *Proceedings of 9th International Conference on Aspect-Oriented Software Development*, March 2010. To appear.