Compiler Techniques for Efficient MATLAB to OpenCL Code Generation

Luís Reis
Faculdade de Engenharia (FEUP)
Universidade do Porto, and
INESC-TEC
luis.cubal@fe.up.pt

João Bispo
Faculdade de Engenharia (FEUP)
Universidade do Porto, and
INESC-TEC
jbispo@fe.up.pt

João M. P. Cardoso
Faculdade de Engenharia (FEUP)
Universidade do Porto, and
INESC-TEC
jmpc@fe.up.pt

ABSTRACT

MATLAB is a high-level language used in various scientific and engineering fields. Deployment of well-tested MATLAB code to production would be highly desirable, but in practice a number of obstacles prevent this, notably performance and portability. Although MATLAB-to-C compilers exist, the performance of the generated C code may not be sufficient and thus it is important to research alternatives, such as CPU parallelism, GPGPU computing and FPGAs. OpenCL is an API and programming language that allows targeting these devices, hence the motivation for MATLAB-to-OpenCL compilation. In this paper, we describe our recent efforts on offloading code to OpenCL devices in the context of our MATLAB to C/OpenCL compiler.

CCS CONCEPTS

•**Computing methodologies** →**Parallel programming languages;** •**Software and its engineering** →*Retargetable compilers;*

KEYWORDS

Optimizing Compilers, MATLAB, OpenCL, GPGPU

ACM Reference format:
Luís Reis, João Bispo, and João M. P. Cardoso. 2017. Compiler Techniques for Efficient MATLAB to OpenCL Code Generation. In *Proceedings of IWOCL '17, Toronto, Canada, May 16-18, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3078155.3078186

1 INTRODUCTION

MATLAB [4] is a matrix-oriented high-level programming language used in various scientific and engineering fields. However, deployment of MATLAB programs is hampered by portability and performance concerns, as a compatible runtime is required. Multiple projects have attempted to solve this problem by compiling MATLAB code to lower-level languages, such as C, but in some cases the resulting code may still not achieve the required performance levels. When this happens, exploring alternatives, such as GPGPU computations, may be helpful.

We have extended MATISSE [1], a MATLAB compiler framework, to generate C and OpenCL code [2]. This paper briefly presents our most recent enhancements to the OpenCL generator. Specifically, we briefly show the programming model adopted, the architecture of the compiler, code generator and the most important optimizations.

There are other approaches to compile MATLAB to parallel languages, such as Mix10 [3] and MEGHA [5]. However, this is, to our knowledge, the first Matlab-to-OpenCL compiler, so we are able to target multiple devices with a single backend.

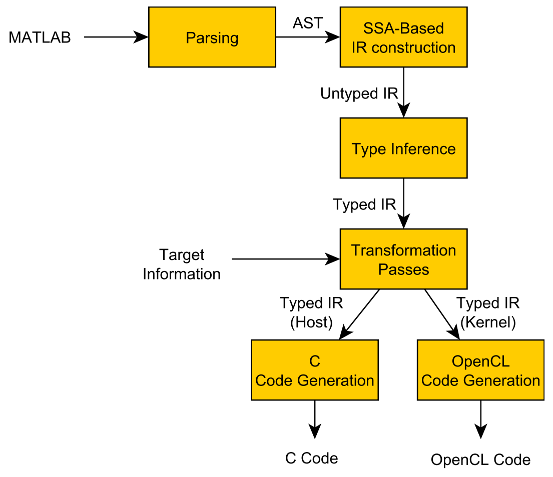

Figure 1: Overall architecture of the MATISSE OpenCL backend.

2 OUR APPROACH

The MATISSE C compiler is able to deal with a representative subset of MATLAB. The OpenCL backend processes sections identified with custom directives and aggressively attempts to parallelize code within those sections. Operations that can not be offloaded (such as memory allocations) are still executed sequentially.

Figure 1 presents the overall architecture of the compiler. MATISSE builds a custom IR for the MATLAB code, performs type inference, applies a set of transformation passes, and then generates code based on this IR.

Figure 2 shows a MATLAB function that computes the sum of two vectors. MATISSE identifies the `%!parallel` directive, embedded in a comment that is ignored by other MATLAB-compatible tools, and attempts to offload as many computations as possible to OpenCL.

We believe that our model is a good compromise between ease of use, flexibility and compatibility with multiple coding styles.

```
%!parallel
function y = vectoradd(A, B)
    y = A + B;
end
```

Figure 2: An annotated MATLAB function that performs vector addition.

```
__kernel void vectoradd_1_1(
    global double* A, global double* B,
    global double* y, uint N_1)
{
    ...
    global_id0_1 = get_global_id(0U);
    if(global_id0_1 < N_1){
        int i = global_id0_1 + 1;
        y[i - 1] = A[i - 1] + B[i - 1];
    }
}
```

Figure 3: Automatically generated OpenCL code to compute the sum of two vectors, abridged for brevity.

3 OPENCL CODE GENERATION

The MATISSE OpenCL backend includes two code generators: one for C host code and another for offloaded OpenCL code. The host generator adds support for OpenCL APIs to the sequential C code generator, and the OpenCL generator extends the C generator in order to generate OpenCL kernel code. The OpenCL backend is built on top of the MATISSE C backend [6] and reuses nearly all of its infrastructure, while recognizing additional directives, extracting code within the parallel regions and generating OpenCL code.

The OpenCL backend first runs the same transformations/optimizations as the C backend. In addition to transforming inefficient MATLAB code patterns (such as matrix resizes in loops) into more efficient equivalents, this has the additional effect of normalizing the code. Because of this, the OpenCL backend has to recognize fewer patterns. We have also introduced several OpenCL-specific optimizations to deal with data transfers. MATISSE identifies several cases where these transfers are unnecessary and eliminates them. We can also generate code that takes advantage of Shared Virtual Memory (SVM), using heuristics to determine its profitability.

Figure 3 shows part of the the automatically generated kernel code. In this example, the number of elements to add was imported explicitly because it can differ from the global size (e.g., if the number of loop iterations is not a multiple of the desired local size).

4 EVALUATION

We are evaluating the compiler with MATLAB code from well-known benchmark repositories. We compared our generated parallel code with the sequential equivalent for the 6 benchmarks from [2] and a complex product computation using element-wise computations. We used a computer running Windows 10 64-bits with GCC 4.9.2, with an AMD A10-7850K APU (4.10 GHz) with 8 GB of RAM and a discrete AMD Radeon R9 280X GPU with 3GB of GDDR5 RAM. We were able to achieve a speedup 11× (geometric mean). 4 out of 7 benchmarks had speedups, with the Monte Carlo Option Pricing benchmark achieving a speedup of 1432×.

We also evaluated 2 larger benchmarks (Disparity and Tracking) from the San Diego Vision Benchmark Suite [7] running on the same computer. On these benchmarks, achieving speedups is more challenging, due to the lack of heuristics to determine when to offload the code. However, when the user manually indicates which sections to parallelize, we are able to achieve speedups of 31% and 10% for Disparity and Tracking, respectively.

5 CONCLUSIONS

This paper briefly presented our approach to generate OpenCL code from MATLAB. We implemented a prototype based on the MATISSE compiler framework, and developed a number of optimizations to improve performance. Ongoing work is focused on targeting additional OpenCL-compatible devices (such as FPGAs), develop device-specific optimizations, expand the supported MATLAB subset and research heuristics to determine which code sections to offload.

An online demonstration of the tool can be found at http://specs.fe.up.pt/tools/matisse/.

6 ACKNOWLEDGMENTS

This work has been partially supported by the TEC4Growth project, "NORTE-01-0145-FEDER-000020", financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF). Reis and Bispo acknowledge the support of the Portuguese Science Foundation (FCT), through the grants PD/BD/105804/2014 and SFRH/BPD/118211/2016, respectively.

REFERENCES

[1] João Bispo, Pedro Pinto, Ricardo Nobre, Tiago Carvalho, João M. P. Cardoso, and Pedro C. Diniz. 2013. The MATISSE MATLAB Compiler - A MATrix(MATLAB)-aware compiler InfraStructure for embedded computing SystEms. In *IEEE International Conference on Industrial Informatics (INDIN'2013)*. Bochum, Germany.

[2] João Bispo, Luís Reis, and João M. P. Cardoso. 2015. C and OpenCL Generation from MATLAB. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15)*. ACM, New York, NY, USA, 1315–1320.

[3] Vineet Kumar and Laurie Hendren. 2014. MIX10: Compiling MATLAB to X10 for High Performance. *SIGPLAN Not.* 49, 10 (Oct. 2014), 617–636.

[4] MathWorks. 2017. MATLAB - The Language of Technical Computing. http://www.mathworks.com/products/matlab/. (2017). Accessed: Feb 2nd, 2017.

[5] Ashwin Prasad, Jayvant Anantpur, and R. Govindarajan. 2011. Automatic Compilation of MATLAB Programs for Synergistic Execution on Heterogeneous Processors. *SIGPLAN Not.* 46, 6 (June 2011), 152–163.

[6] Luís Reis, João Bispo, and João M. P. Cardoso. 2016. SSA-based MATLAB-to-C Compilation and Optimization. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming (ARRAY 2016)*. ACM, New York, NY, USA, 55–62.

[7] Sravanthi Kota Venkata, Ikkjin Ahn, Donghwan Jeon, Anshuman Gupta, Christopher Louie, Saturnino Garcia, Serge Belongie, and Michael Bedford Taylor. 2009. SD-VBS: The San Diego Vision Benchmark Suite. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC) (IISWC '09)*. IEEE Computer Society, Washington, DC, USA, 55–64.