Acceleration Opportunities in Linear Algebra Applications via Idiom Recognition

João P. L. de Carvalho
IC-UNICAMP
University of Campinas
Campinas, Brazil
joao.carvalho@ic.unicamp.br

Braedy Kuzma
Dept. of Computing Science
University of Alberta
Edmonton, Canada
braedy@ualberta.ca

Guido Araujo
IC-UNICAMP
University of Campinas
Campinas, Brazil
guido@ic.unicamp.br

ABSTRACT

General matrix-matrix multiplication (GEMM) is a critical operation in many application domains [1]. It is a central building block of deep learning and computer graphics algorithms and is also a core operation for most scientific applications based on the discretization of systems of differential equations. Due to this, GEMM has been extensively studied and optimized, resulting in libraries of exceptional quality such as BLAS, Eigen, and other platform specific implementations such as MKL (Intel's x86) and ESSL (IBM's PowerPC) [3, 5]. Despite these successes, the GEMM idiom continues to be reimplemented by programmers without consideration for the intricacies already accounted for by the aforementioned libraries. To this end, this project aims to provide transparent adoption of high-performance implementations of GEMM through a novel optimization pass implemented within the LLVM framework using idiom recognition techniques.

CCS CONCEPTS

• **Software and its engineering → Compilers**.

KEYWORDS

GEMM, idiom recognition, LLVM

ACM Reference Format:
João P. L. de Carvalho, Braedy Kuzma, and Guido Araujo. 2020. Acceleration Opportunities in Linear Algebra Applications via Idiom Recognition. In *ACM/SPEC International Conference on Performance Engineering Companion (ICPE '20 Companion), April 20–24, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3375555.3383586

1 APPROACH

The approach makes use of idiom recognition to find opportunities for optimization. Idioms are recurrent constructs in programming languages that express a specific computation, can be easily recognized (by humans), and are simple to compose [4]. For example, `a > b ? a : b` is a common idiom in the C language to express a binary maximum operation.

This concept is applied to create an LLVM pass capable of finding code patterns representing naïve implementations of the GEMM idiom and replacing them with calls to high performance implementations. The process proceeds as shown in Figure 1. First, the clang frontend is executed to produce LLVM's intermediate representation (IR). This IR is put through our pass, using an idiom recognizer to determine if and where a GEMM idiom exists. If the GEMM does not exist, compilation proceeds as normal with no chance of harming performance. When an opportunity is discovered, the IR is passed to a rewriter pass which will isolate the idiom if possible and then replace the code with a library call.

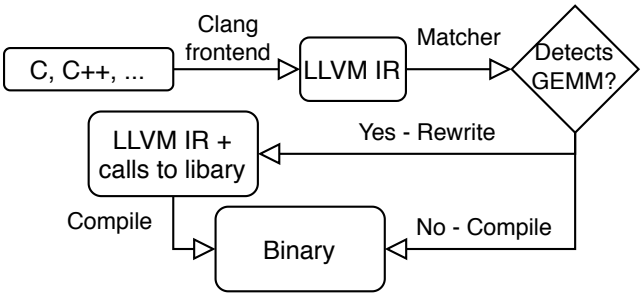


Figure 1

The presented matcher is a tree-based pattern matcher capable of finding complex patterns of IR code. A simplified example given in Figure 3b shows the basis of a matcher for the GEMM reduction equation. When applied to the IR code in Figure 3a, the tree matcher would discover two matching instances, one rooted at `%44` and a second rooted at `%48`.

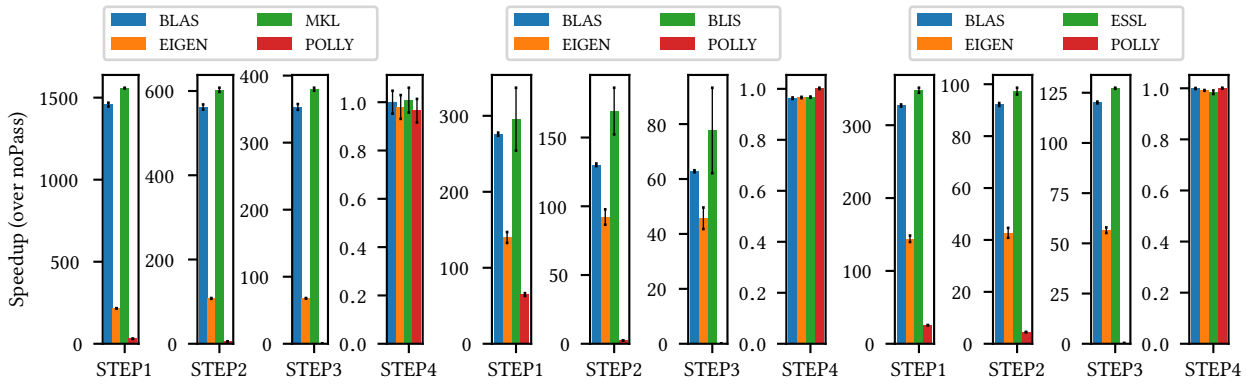
It is possible for more complicated matchers to be polymorphic in design through the use of disjunction nodes in order to match varying code patterns. For example, accesses to a two dimensional array or a one dimensional, linearized array are possible when implementing GEMM and a pattern matcher must account for this. Without disjunction nodes, a variation like this would require a new pattern to be written, creating a greater burden on developers.

The matcher is able to capture the input and output matrices, their dimensions and their access orders. Moreover, polymorphic instances of the target idiom (e.g. with optional alpha or beta scaling) are also recognized. This information is used to construct calls to a variety of libraries from open-source projects and vendor-specific implementations. The replacement mechanism provided through the transformation pass can be easily extended to enable any libraries that implement GEMM.

Most importantly, the proposed pass can more robustly detect these patterns than other state-of-the-art idiom recognition tools.

ICPE '20 Companion, April 20–24, 2020, Edmonton, AB, Canada

ACM ISBN 978-1-4503-7109-4/20/04.
https://doi.org/10.1145/3375555.3383586

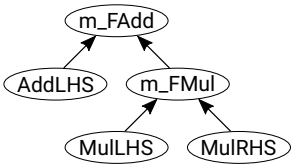


(a) Intel's x86 (b) AMD's x86 (c) IBM's PowerPC

Figure 2: Speedup over varying levels of hand optimization.

```
1  %41 = phi [ %53, %40 ], [ 0, %25 ]
2  %42 = phi [ %52, %40 ], [ 0.0, %25 ]
3  %43 = mul %41, %17
4  %44 = add %43, %23
5  %45 = getelementptr %4, %44
6  %46 = load %45
7  %47 = mul %41, %16
8  %48 = add %47, %26
9  %49 = getelementptr %6, %48
10 %50 = load %49
11 %51 = fmul %46, %50
12 %52 = fadd %42, %51
13 %53 = add %41, 1
14 %55 = icmp eq %53, %21
15 br %55, %30, %40
```

(a) Hello



(b) Hello

Figure 3

Special care is made to ensure that access order is respected, an aspect which is often ignored by other tools resulting in incorrect code. It is shown in Section 2 that the pass continues to recognize the idiom in various states of hand optimization, providing significant speedups.

2 PRELIMINARY RESULTS

Figure 2 compares speedups (y-axis) enabled through idiom recognition. The figure's x-axis shows results of four source-level optimizations: transposition of matrix A (STEP1); loop interchange (I, J, K) → (J, K, I) (STEP2); loop interchange as in STEP2 with tiling (STEP3); and interchange and tiling as in STEP3 with the addition of packing (STEP4). The first three bars, left to right, use our replacement strategy to transparently enable a different high-performance backend library: BLAS via OpenBLAS, Eigen, and a platform specific library (MKL on Intel x86, BLIS on AMD x86[1], ESSL on IBM PowerPC). The final bar (Polly) are the results of a state-of-the-art polyhedral tool in LLVM [2]. The speedup shown is against the source code with the indicated hand optimization and then compiled with `-O3` and native tuning (`march`, `mtune`, `mcpu`).

Preliminary results show that the platform specific library performs best among all evaluated alternatives. Moreover, the hand optimized code, even after significant optimization effort, is outperformed by all three libraries enabled via the proposed compiler transformation. In fact, even after rescheduling the loops and improving the matrices' access pattern (STEP3), platform specific libraries are still over 100 times faster. These results indicate that programmers should rely on the knowledge and expertise encoded into domain-specific libraries instead of rewriting known idioms.

The proposed approach recognizes the target idiom in each example despite increasingly complex code. However, the final stage (STEP4) is matched but not immediately transformable due to aliasing issues introduced by the packing optimization. Nevertheless, in such cases, a custom warning message can still inform programmers about potential opportunities to adopt domain-specific libraries.

[1]BLIS is not platform specific though it is officially promoted by AMD.

ACKNOWLEDGMENT

The authors would like to thank FAPESP (grants 2013/08293-7, 2016/15337-9, and 2019/01110-0) and Center for Computational Engineering and Sciences (CCES) for supporting this work.

REFERENCES

[1] Kazushige Goto and Robert Van De Geijn. 2008. High-Performance Implementation of the Level-3 BLAS. *ACM Trans. Math. Softw.* 35, 1, Article Article 4 (July 2008), 14 pages. https://doi.org/10.1145/1377603.1377607

[2] Tobias Christian Grosser. 2011. *Enabling polyhedral optimizations in llvm.* Ph.D. Dissertation.

[3] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

[4] Perlis, Alan J. and Rugaber, Spencer. 1979. Programming with Idioms in APL. *SIGAPL APL Quote Quad* 9, 4-P1 (May 1979), 232–235. https://doi.org/10.1145/390009.804466

[5] Field G. Van Zee, Tyler M. Smith, Bryan Marker, Tze Meng Low, Robert A. Van De Geijn, Francisco D. Igual, Mikhail Smelyanskiy, Xianyi Zhang, Michael Kistler, Vernon Austel, and et al. 2016. The BLIS Framework: Experiments in Portability. *ACM Trans. Math. Softw.* 42, 2, Article Article 12 (June 2016), 19 pages. https://doi.org/10.1145/2755561