# Model-Driven Search-Based Loop Fusion Optimization for Handwritten Code

# Extended Abstract

Ajay Panyala[1], Pamela Bhattacharya[2], Gerald Baumgartner[1], and J. Ramanujam[3]

[1] Division of Computer Science and Engineering (School of EECS),
Louisiana State University, Baton Rouge, LA 70803, USA
{ajay,gb}@csc.lsu.edu

[2] Department of Computer Science and Engineering,
University of California, Riverside, Riverside, CA 92521, USA
pamelabhattacharya@gmail.com

[3] Div. of Electrical and Computer Engineering (School of EECS),
and Center for Computation and Technology,
Louisiana State Univ., Baton Rouge, LA 70803, USA
jxr@ece.lsu.edu

## 1 Introduction

A large number of scientific and engineering applications are highly data intensive, operating on data sets that range from gigabytes to terabytes, thus exceeding the physical memory of the machine. Even though the math and logic behind the application code may be fairly straightforward, the orchestration of data movement can be very complex.

For example, electronic structure codes, which are widely employed in quantum chemistry, [13, 9], computational physics, and material science, require elaborate interactions between subsets of data. Instead of simply bringing data into the physical memory once, processing it, and then overwriting it by new data, subsets of data are repeatedly moved back and forth between a small memory pool, limited physical memory, and a large memory pool, the unlimited disk. The cost introduced by these data movements can have a large impact on the overall execution time of the computation. *Out-of-core* algorithms that explicitly orchestrate the movement of subsets of data within the memory-disk hierarchy must ensure that data is processed in subsets small enough to fit in the machine's main memory, but large enough to minimize the cost of moving data between disk and memory.

In previous research, we presented an approach to the automated synthesis of *out-of-core* programs [2, 10] in the context of the Tensor Contraction Engine (TCE) program synthesis system. [1, 6, 7, 5, 4]. The TCE targets a class of electronic structure calculations, which involve many computationally intensive components expressed as tensor contractions (essentially generalized matrix products involving higher-dimensional arrays). It generates efficient parallel and/or out-of-core code from tensor contraction formulas.

While the implementation in the TCE addresses tensor contraction expressions arising in quantum chemistry, the approach developed there has broader applicability. Our search-based loop fusion optimization has been successful in the Tensor Contraction Engine for minimizing storage or, together with a search-based tiling optimization, for automatically generating efficient out-of-core code from tensor contraction expressions. In this paper, we demonstrate how this approach can be generalized to handwritten code in the form of imperfectly nested loop structures operating on arrays potentially larger than the physical memory size. This would allow translating, for example, handwritten in-core tensor contraction code into out-of-core code. Such code structures can be found in computational chemistry packages, such as GAMESS, [14], Gaussian, [8], and NWChem, [15, 16], or in computational physics codes modeling electronic properties of semiconductors and metals. We present an adaptation of the loop fusion algorithm to generate GPU or out-of-core code from handwritten code. The loop fusion algorithm optimizes dense array computations by fusing producer and consumer loops to minimize the storage requirements for intermediate arrays. It undoes any fusion provided by the programmer and, using a cost model for memory minimization, searches all loop structures for the memory-minimal code. Then the fused loops are expanded to tile size and these tiles are used as units of data transfers between different levels in the memory hierarchy. We have implemented the fusion algorithm as a source-to-source translation using the ROSE compiler framework. We demonstrate the effectiveness of this approach for multicore CPUs and GPUs using an example.

## 2 Problem

The combination of fusion and tiling optimizations has proven successful in the TCE for minimizing disk to memory traffic for dense tensor computations. While other optimizations are specific to tensor contraction expressions, these two optimizations can be useful for optimizing handwritten dense array computations. The fusion and tiling optimizations can be used for different purposes. For example, the fusion optimization by itself can be used for minimizing memory requirements for intermediate results [12, 11]. Together with the tiling optimization and with different cost models, these optimizations can be used for minimizing disk to memory traffic [2, 10] or for space-time tradeoffs [4]. Since the problem of making the fusion algorithm work on an abstract syntax tree is independent of the cost model, we limit the discussion to the use of the fusion algorithm as a stand-alone optimization for memory minimization.

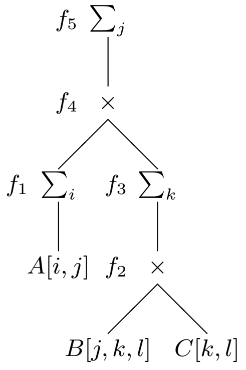
Consider the multi-dimensional summation shown in Figure 1(a). After algebraic transformations to minimize the operation count, we might arrive at the formula sequence shown in Figure 1(b) for computing the multi-dimensional summation. This formula sequence is represented by the expression tree in Figure 1(c).

$$W[k] = \sum_i \sum_j \sum_l (A[i,j] \times B[j,k,l] \times C[k,l])$$

(a) A multi-dimensional summation

$$\begin{aligned}
f_1[j] &= \textstyle\sum_i A[i,j] \\
f_2[j,k,l] &= B[j,k,l] \times C[k,l] \\
f_3[j,k] &= \textstyle\sum_l f_2[j,k,l] \\
f_4[j,k] &= f_1[j] \times f_3[j,k] \\
W[k] = f_5[k] &= \textstyle\sum_j f_4[j,k]
\end{aligned}$$

(b) A formula sequence computing (a)



(c) An expression tree for (b)

**Fig. 1.** An example multi-dimensional summation and two representations of a computation.

A naïve way to implement the computation is to have a set of perfectly-nested loops for each node in the tree. But the size of the arrays in this arrangement could be too high for them to fit in the available memory. Hence, there is a need to fuse the loops as a means of reducing memory usage. By fusing loops between the producer loop and the consumer loop of an in-memory array, intermediate results are formed and used in a pipelined fashion, and they reuse the same reduced array space. There are many different ways to fuse the loops, which could result in different memory usage. In this paper, we focus on the non-trivial problem of finding a loop fusion configuration for well behaved handwritten code that minimizes memory usage without increasing the operation count.

Our fusion algorithm [12] is a bottom-up dynamic programming algorithm that operates on expression trees. An expression tree has the advantage that the consumer of an intermediate array is the parent node of the producer of that array. Furthermore, all loops are implicit, since the interior nodes in an expression tree represent array operations. For generalizing the fusion algorithm to abstract syntax trees with explicit loops, it is necessary to reconstruct the producer-consumer relationship and recognize any language constructs or code structures that cannot be optimized with the loop fusion algorithm.

## 3 Algorithm

Our fusion algorithm searches through possible loop fusion configurations between the producers and the consumers of intermediates. When operating on an expression tree, the producer and the consumer of an intermediate are conveniently in a parent-child relationship. In handwritten code, they may be in different loop nests. Furthermore, for handwritten code it is necessary to identify the code fragments on which a loop fusion optimization can safely be performed.

Our memory minimization algorithm for handwritten code runs the loop fusion algorithm on the dependency graph after a series of traversals of the abstract syntax tree that prepare the abstract syntax tree for the fusion algorithm and identify where loop fusion optimizations can safely be performed. After completion of the loop fusion algorithm, we generate a new abstract syntax tree that represents the memory-minimal fused code. Our algorithm consists of the following steps:

**Canonicalization** moves code containing side effects out of expressions.

**Region identification** identifies code fragments in which loop fusion optimizations are safe.

**Subscript inference** computes the indices needed for intermediates in unfused code.

**Reaching definitions analysis** identifies producers and consumers of intermediate arrays.

**Loop fusion optimization** identifies the memory-minimal loop fusion configuration.

**Code generation** constructs an abstract syntax tree for the optimal loop fusion configuration.

## 4 Experimental Evaluation

For demonstrating the effectiveness of our fusion algorithm and compilation framework, we used handwritten code for the 4-index transformation equation:

$$B[a,b,c,d] = \sum_s (C1[s,d] \times \sum_r (C2[r,c] \times \sum_q (C3[q,b] \times \sum_p (C4[p,a] \times A[p,q,r,s]))))$$

In the handwritten code, all loop bounds are compile-time constants and loops may or may not have been fused. The experiments were performed on a quad-core Intel Xeon machine with 12GiB of memory and with the tensor dimensions chosen, such that the unfused code required up to 8.5GiB.

This code is then optimized by our loop fusion algorithm, such that the storage requirements for intermediate tensors are minimized. This results in increased temporal locality between producer and consumer, but causes poor temporal locality within a contraction because of the elimination of entire tensor dimensions. Dimensions that were eliminated were then manually expanded to tiles to improve temporal locality within a contraction while staying within the memory limit. The inner-most loop nests, which now represent the tensor contraction on a tile, were then manually replaced by index permutation and matrix multiplication library calls. We generated versions of the code for single-core, multi-core, and GPU architectures.

We compared the performance of the code resulting from our optimization approach with code produced using polyhedral model optimization. For single-core and multi-core code, we compared against code generated by Pluto, Version 0.9.0-36-g5fb218a [3]. For GPUs, we compared against PPCG, Version c7179a0 [17]. As input to these tools we used both unfused and fused code. In all cases, the code generated by our optimization approach outperformed the code generated by the polyhedral model optimizers by at least a factor of two.

**Acknowledgments** This work was supported in part by the National Science Foundation under grants CNS-0509467, CCF-0541409, OCI-0926687, and CNS-1059417.

## References

1. G. Baumgartner, A. Auer, D. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. Harrison, S. Hirata, S. Krishnamoorthy, S. Krishnan, C. Lam, Q. Lu, M. Nooijen, R. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov. Synthesis of high-performance parallel programs for a class of ab initio quantum chemistry models. *Proceedings of the IEEE*, 93(2):276–292, February 2005.
2. A. Bibireata, S. Krishnan, D. Cociorva, G. Baumgartner, C. Lam, P. Sadayappan, J. Ramanujam, D. Bernholdt, and V. Choppella. Memory-constrained data locality optimization for tensor contractions. In *Proceedings of the 16th Workshop on Languages and Compilers for Parallel Computing*, College Station, Texas, October 2003.
3. U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In *Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '08, pages 101–113, New York, NY, USA, 2008. ACM.
4. D. Cociorva, G. Baumgartner, C. Lam, P. Sadayappan, J. Ramanujam, M. Nooijen, D. Bernholdt, and R. Harrison. Space-time trade-off optimization for a class of electronic structure calculations. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation (PLDI)*, pages 177–186, June 2002.
5. D. Cociorva, X. Gao, S. Krishnan, G. Baumgartner, C. Lam, P. Sadayappan, and J. Ramanujam. Global communication optimization for tensor contraction expressions under memory constraints. In *Proceedings of Seventeenth International Parallel and Distributed Processing Symposium (IPDPS '03)*, page 37b, Nice, France, Apr. 2003. IEEE Computer Society Press.
6. X. Gao, S. Krishnamoorthy, S. Sahoo, C. Lam, G. Baumgartner, J. Ramanujam, and P. Sadayappan. Efficient search-space pruning for integrated fusion and tiling transformations. *Concurrency and Computation: Practice and Experience*, 19(18):2425–2443, December 2007.
7. X. Gao, S. Sahoo, Q. Lu, G. Baumgartner, C. Lam, J. Ramanujam, and P. Sadayappan. Performance modeling and optimization of parallel out-of-core tensor contractions. In *Proceedings of the ACM SIGPLAN 2005 Symposium on Principles and Practice of Parallel Programming*, pages 266–276, Chicago, IL, June 2005.
8. J. Foresman and A. Frisch. *Exploring Chemistry with Electronic Structure Methods: A Guide to Using Gaussian.* Gaussian, Inc., 2 edition, 1996.
9. J. M. L. Martin. In P. v. R. Schleyer, P. R. Schreiner, N. L. Allinger, T. Clark, J. Gasteiger, P. Kollman, H. F. Schaefer III (Eds.). . *Encyclopedia of Computational Chemistry*, 1:115–128, 1998.

10. S. Krishnan, S. Krishnamoorthy, G. Baumgartner, D. Cociorva, P. S. C. Lam, J. Ramanujam, D. Bernholdt, and V. Choppella. Data locality optimization for synthesis of efficient out-of-core algorithms. In *Proceedings of the the Intl. Conf. on High Performance Computing*, volume 2913 of *Lecture Notes in Computer Science*, pages 406–417, Hyderabad, India, December 2003. Springer-Verlag.
11. C. Lam. *Performance Optimization of a Class of Loops Implementing Multi-Dimensional Integrals*. PhD thesis, The Ohio State University, Columbus, OH, August 1999.
12. C. Lam, D. Cociorva, G. Baumgartner, and P. Sadayappan. Optimization of memory usage requirement for a class of loops implementing multi-dimensional integrals. In *Proceedings of the Twelfth Workshop on Languages and Compilers for Parallel Computing*, pages 350–364, San Diego, CA, 1999.
13. T. J. Lee and G. E. Scuseria. Achieving chemical accuracy with coupled cluster theory. In S. R. Langhoff, editor, *Quantum Mechanical Electronic Structure Calculations with Chemical Accuracy*, pages 47–109. Kluwer Academic, 1997.
14. M. Schmidt, K. Baldridge, J. Boatz, S. Elbert, M. Gordon, J. Jensen, S. Koseki, N. Matsunaga, K. Nguyen, S. Su, T. Windus, M. Dupuis, and J. Montgomery. General Atomic and Molecular Electronic Structure System (GAMESS). *Journal on Computational Chemistry*, 14:1347–1363, 1993.
15. T. P. Straatsma, E. Aprà, T. L. Windus, E. J. Bylaska, W. de Jong, S. Hirata, M. Valiev, M. Hackler, L. Pollack, R. Harrison, M. Dupuis, D. M. A. Smith, J. Nieplocha, V. Tipparaju, M. Krishnan, A. A. Auer, E. Brown, G. Cisneros, G. Fann, H. Früchtl, J. Garza, K. Hirao, R. Kendall, J. Nichols, K. Tsemekham, K. Wolinski, J. Anchell, D. Bernholdt, P. Borowski, T. Clark, D. Clerc, H. Dachsel, M. Deegan, K. Dyall, D. Elwood, E. Glendening, M. Gutowski, A. Hess, J. Jaffee, B. Johnson, J. Ju, R. Kobayashi, R. Kutteh, Z. Lin, R. Littlefield, X. Long, B. Meng, T. Nakajima, S. Niu, M. Rosing, G. Sandrone, M. Stave, H. Taylor, G. Thomas, J. van Lenthe, A. Wong, and Z. Zhang. *NWChem, A Computational Chemistry Package for Parallel Computers, Version 4.6*. Pacific Northwest National Laboratory, Richland, Washington 99352–0999, USA, 2004. `http://www.emsl.pnl.gov/docs/nwchem/`.
16. M. Valiev, E. Bylaska, N. Govind, K. Kowalski, T. Straatsma, H. V. Dam, D. Wang, J. Nieplocha, E. Apra, T. Windus, and W. de Jong. Nwchem: A comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications*, 181(9):1477 – 1489, 2010.
17. S. Verdoolaege, J. C. Juega, A. Cohen, J. I. Gómez, C. Tenllado, and F. Catthoor. Polyhedral Parallel Code Generation for CUDA. *ACM Transactions on Architecture and Code Optimization*, 9(4), 2013. Selected for presentation at the HiPEAC 2013 Conf.