# Final Report for Award # DE-SC3956

Separating Algorithm and Implementation via programming Model Injection (SAIMI)

PI: Michelle Mills Strout

Address: 1873 Campus Delivery, Fort Collins, CO 80523

Colorado State University

Dates for Award: April 15, 2010 through April 14, 2015
with no-cost extension through August 15, 2015

**Abstract**

Programming parallel machines is fraught with difficulties: the obfuscation of algorithms due to implementation details such as communication and synchronization, the need for transparency between language constructs and performance, the difficulty of performing program analysis to enable automatic parallelization techniques, and the existence of important "dusty deck" codes. The SAIMI project developed abstractions that enable the orthogonal specification of algorithms and implementation details within the context of existing DOE applications. The main idea is to enable the injection of small programming models such as expressions involving transcendental functions, polyhedral iteration spaces with sparse constraints, and task graphs into full programs through the use of pragmas. These smaller, more restricted programming models enable orthogonal specification of many implementation details such as how to map the computation on to parallel processors, how to schedule the computation, and how to allocation storage for the computation. At the same time, these small programming models enable the expression of the most computationally intense and communication heavy portions in many scientific simulations. The ability to orthogonally manipulate the implementation for such computations will significantly ease performance programming efforts and expose transformation possibilities and parameter to automated approaches such as autotuning.

At Colorado State University, the SAIMI project was supported through DOE grant DE-SC3956 from April 2010 through August 2015. The SAIMI project has contributed a number of important results to programming abstractions that enable the orthogonal specification of implementation details in scientific codes. This final report summarizes the research that was funded by the SAIMI project.

## 1 Overview

Solving important scientific problems through computational modeling requires high performance computing. High performance computing requires performance on individual processors and parallel programming to take advantage of the multitude of parallel computing architectures. Using general-purpose parallel programming models is error prone and results in code where implementation details obfuscate the original algorithm. Additionally implementation decisions necessary for one target platform are not necessarily applicable to future platforms, therefore requiring significant development costs when porting applications to new systems.

There are a range of ways in which implementation details are exposed in programming models. Declarative programming languages seek to avoid the exposure of implementation details. This approach avoids obfuscation due to parallelization implementation details, but does not provide a mechanism for the application programmer or performance programmer to take control of the implementation. Explicit parallel programming in general purpose programming models like C or Fortran with MPI provide almost complete control over implementation details such as the distribution and scheduling of data and computation, but the implementation is strongly coupled with the algorithm. This control and the use of the MPI interface results in good performance portability, but the entangling of the algorithmic and implementation specifications can lead to significant code obfuscation.

Recent programming model research and development includes programming languages and libraries with constructs that enable the orthogonal specification of algorithms and implementation details. Examples include OpenMP [8], which provides pragmas for labeling a loop/algorithm as having `forall` parallelism in and indicating implementation preferences; Parallel Global Address Space (PGAS) languages such as Chapel [6, 7], which provide user-defined distributions; and Standard Templates Adaptive Parallel Library (STAPL) [23, 3], which provide orthogonal scheduling of data structure iterators. There are also more restrictive programming models such as the polyhedral programming model [43, 11, 25, 15, 12, 14, 4], which enables orthogonal specification of scheduling and storage mapping within the compiler, and MapReduce [9], which enables implementation details, such as the runtime environment, to be specified and to evolve at runtime. Wholesale conversion of existing codes to new models is problematic due to the magnitude of the work involved and to an initial lack of tool support. Conversion to programming models that are minimally intrusive like OpenMP is more feasible, but the interface between algorithm and implementation specification in such models lacks power in terms of both the algorithmic abstractions and the implementation abstractions.

*In the SAIMI project, we use injectable programming models to orthogonalize programming implementation decisions such as performance transformations and raise the abstraction level for implementation specifications.* An injectable programming model is a more restricted, declarative programming model made available within the context of a more general programming model such as C/C++ or Fortran. We are initially making the more restricted programming models available through pragmas and/or library interfaces followed by source-to-source transformation. We are developing injectable programming models for specifying performance critical algorithms and optimizations such as lookup tables, stencil computations, dense and sparse matrix computations, and task graph computations. We envision high-level, orthogonal implementation abstractions that will enable application and performance programmers to specify the implementation for each algorithm differently for each target architecture.

The SAIMI project research goals include (1) evaluating existing mechanisms for orthogonally specifying implementation details, (2) raising the level of abstraction for implementation specification in existing mechanisms for a set of injectable programming models, and (3) developing libraries and preprocessors capable of composing orthogonal algorithm and implementation specifications. We evaluate the programmability and performance of our approach on performance-critical computations in various small, medium, and large scientific computing benchmarks written in C++ and Fortran.

Some defining characteristics of the SAIMI project are the incremental approach embodied by injectable programming models and the concept of providing abstractions for the high-level, orthogonal specification of implementation details. A more detailed overview of the SAIMI project was published in the 2011 SciDAC proceedings [34].

## 2 Project Contributions

The development of evaluation criteria and evaluation of existing programming model mechanisms for the separation of algorithms and implementation details provided us ideas for injectable programming models as well as mechanisms to evaluate our own contributions.

- In 2010, we proposed and published some qualitative criteria for evaluating the programmability of programming model features with respect to the extent to which they enable the orthogonal specification of implementation details [16, 17].
- In 2011 in collaboration with John Dennis at NCAR in Boulder we developed and published a tech report and conference paper about the CGPOP mini application [28, 30]. CGPOP [27] is a proxy application for the POP application.

- In 2012, we argued that the CGPOP mini app was a programmability proxy for the POP application as well as a performance proxy and in general discussed how programmability proxies should be determined [26].

The main approach we used to orthogonally specifying implementation details was to inject smaller programming models into full applications with programs and specify implementation details as transformations on the injected programs. Our various efforts all worked toward this goal.

One such example of an injected programming model was the Mesa tool which was developed to semi-automate the application of look-up table optimizations. This tool used a pragma mechanism and realized the expression programming model for dealing with expensive function calls such as `exp(), sin(), cos()`, etc.

- In 2010, we prototyped a tool called Mesa [1] that enables programmers to specify lookup table (LUT) optimizations orthogonally from the C++ specification of a computation. Submitted a paper [39] with initial results on using Mesa to apply LUT optimization to a Small Angle X-Ray Scattering (SAXS) application [2].
- In 2011, we released the Mesa tool [1, 39, 40], which was built on the ROSE compiler infrastructure. that enables programmers to specify lookup table (LUT) optimizations orthogonally from the C++ specification of a computation.
- In 2012, we published a paper about the Mesa [1]. Mesa enabled programmers to specify lookup table (LUT) optimizations orthogonally from the C++ specification of a computation [41]. The main contribution was the formulation of the look-up table programming model injection problem as an integer programming/optimization problem.
- In 2013, we published the final paper about the lookup table programming model [42]. The main contribution was the testing of the ROSE-based, source-to-source translator that automatically determined a pareto curve of possible LUT optimizations. The programmer could then choose the accuracy vs. performance tradeoff as appropriate for his or her application.

We also developed injectable programming models and performance optimizations for partial differential equations solvers, whose main performance bottlenecks are stencil computations.

- In 2012, we refined parameterized tiling in the polyhedral model [24]. Also investigated how the diamond tiling approach should theoretically scale better than pipelined tiles, which is applicable to stencil computations [44].
- In 2013, we published the first GridWeaver paper about abstractions to orthogonally specify the connectivity of semi-regular grids and stencil computations in atmospheric science applications like CG-POP [29].
- In 2014, we started development of the Loop Chain abstraction [19], where the data access patterns for loops are injected into existing data parallel programs. This led to the the use of the loop chain concept to optimize across loops in a Chombo Computational Fluid Dynamics benchmark [22] and in finite element solver benchmark used to model airplane engines [36]. Additionally, we are further developing the Loop Chain abstraction in the context of combustion simulations in the funded NSF grant CCF-1422725, "SHF: Small: The Loop Chain Abstraction for Balancing Locality and Parallelism".
- Over the past year (2014-2015), we have been incorporating advanced tiling techniques into stencil computations using a programming construct called an iterator from the Chapel programming language. This work is especially exciting, because it can lead to advanced tiling techniques being made available via libraries versus the need to incorporate them into compilers. For this work, Ian Bertolacci,

an undergraduate research assistant for SAIMI, won 1st place in the Undergraduate Poster Competition at Rocky Mountain Celebration of Women in Computing (RMCWiC) and won 3rd place at the Student Research Competition at Supercomputing. Additionally we published and presented a paper at the International Conference of Supercomputing (ICS) [5].

Representing sequences of loops with the loop chain abstraction and/or performing diamond tiling results in tiles or tasks of computation that exhibit good data locality with partial parallelism between the tasks. Representing the dependencies between these tasks with a task graph is quite typical, thus our research in developing technology to support injectible programming models that depend on such a task graph abstraction.

- To sparse tile across loops in irregular applications, it is necessary to do an initial partitioning of one of the loops based on how iterations in the loops interact with each other. We developed a shared memory parallel graph partitioner for use with inspector-executor strategies [18].
- Experimentation with the execution of arbitrary generated task graphs [20]. We investigated Intel's Concurrent Collections, Cilk++ from Cilk Arts, TBB, and the OpenMP 3.0 Task model.

Stencil computations that operate on a structured or semi-structured grid can be represented at compile time with polyhedral or presburger sets. However, many important applications operate on unstructured or irregular meshes and thus use some sparse data structures where the memory access patterns are not known until runtime. Performance optimizations for such computations require an inspector phase at runtime to determine new data orderings and schedules. Previously, we had developed the sparse polyhedral framework for representing and manipulating inspector/executor transformations at compile time [32]. As part of the SAIMI project, we incorporated some of the SPF technology into exisiting source-to-source compilation tools (Chill [13]) and innovated new ways to represent inspectors to enable the determination of correct and efficient inspectors.

- In 2010, we used ideas from the sparse polyhedral framework (originally created by the PI) to simplify the specification of sparse matrix computations and performance optimizations thereof [10].
- In 2012, we determined the set and relation operations that are needed to expression inspector/executor transformations in a framework with polyhedral transformations [33].
- We prototyped a compiler for generating composed inspector code while using the sparse polyhedral framework to specify the composed transformations [35].
- In 2014, we published and presented "An Approach for Proving the Correctness of Inspector/Executor Transformations" with our collaborator Michael Norrish [21]. This introduces a way to handle correctness and performance in inspector/executor codes.
- In collaboration with Mary Hall's group at the University of Utah, we incorporated inspector/executor transformations into the Chill source-to-source compiler. These transformations included transformations that when composed in various ways with loop transformations such as tiling enable a compiler to generate the inspector that can translate between various sparse matrix formats [38, 37].

## 3 Personnel

In addition to the above research and publications, the SAIMI project supported the Ph.D.s of Christopher Wilcox, Andrew Stone, and Christopher Krieger. Christopher Wilcox graduated in 2012 and is currently a lecturer in the Computer Science Department at Colorado State University where he has recently won a teaching award. Dr. Stone and Dr. Krieger were especially impacted by the SAIMI project because they participated in the development of the proposal. Dr. Stone now works at Mathworks on the Matlab compiler. Dr. Krieger is a researcher at the University of Maryland.

This work has also supported the masters of Stephanie Dinkins, and Ian Bertolacci as an undergraduate research assistant. In addition, a research faculty member at CSU, Dr. Catherine Olschanowsky, was partially supported through SAIMI.

In the Summer of 2013 both Andrew Stone and Christopher Krieger finished their Ph.D.s. The money that was funding them was then used to support part of PI Strout?s sabbatical salary with permission from DOE program managers.

## 4 Presentations

PI Strout and others presented aspects of the SAIMI project on numerous occasions and indicated the work being presented was supported by the Department of Energy.

1. "SAIMI and SPF: Separating the Algorithm from the Implementation Details in Sparse Computations," presented at 2011 DOE Scientific Discovery through Advanced Computing (SciDAC) conference, July 12, 2011.
2. "Autotuning Needs for Run-Time Reordering Transformations," presented at the CScADS Autotuning Workshop, August 8, 2011.
3. "SAIMI: Separating the Algorithm from the Implementation Details," presented at the DOE ASCAC Meeting , August 24, 2011.
4. "The CGPOP Miniapp," by Andrew Stone, John Dennis, and Michelle Strout. Presented by Andrew Stone at HPC 2011 - First Annual Front Range High Performance Computing Symposium, September 23-24, 2011.
5. "The CGPOP Miniapp," by John Dennis (NCAR), Andrew Stone (Colorado State University), Michelle Mills Strout (Colorado State University). Presented at the DOE Scientific Discovery through Advanced Computing (SciDAC) conference, July 12, 2011.
6. "Automating Run-Time Reordering Transformations with the Sparse Polyhedral Framework (SPF) and Arbitrary Task Graphs," presented at Imperial College in London, November 21, 2011.
7. "Automating Run-Time Reordering Transformations with the Sparse Polyhedral Framework (SPF)," presented at Australian National University in Canberra, July 12, 2012.
8. Andrew I. Stone and Michelle Mills Strout. "Abstractions for Defining Semi-Regular Grids Orthogonally from Stencil Computations," poster with two page abstract to appear at LCPC, September 2012.
9. David Wonnacott and Michelle Mills Strout. "On the Scalability of Loop Tiling Techniques," talk at the Fourth Annual CnC Workshop, December 2012.
10. In 2013, our collaborator Dave Wonnacott presented the tiling scalability work at IMPACT (International Workshop on Polyhedral Compilation Techniques) [45].
11. In 2013, presented the SAIMI work at the Workshop on Optimizing Stencil Computations (WOSC) [31].
12. "Bulk Synchronous to Asynchronous Parallelism: Using Loop Chains and Full Sparse Tiling to Get There," presented at Australian National University, August 16, 2013.

13. Speaker: Ian J. Bertolacci Authors: Ian J. Bertolacci, Catherine Olschanowsky, Michelle Mills Strout, and David G. Wonnacott, In Collaboration With Bradford L. Chamberlain and Ben Harshbarger. "Chapel Iterators: Providing Tiling for the Rest of us," Presented at SuperComputing 2014 as part of SC14 Chapel Lightning Talks BoF, November 2014.
14. Ian Bertolacci, advisors: Michelle Strout and Catherine Olschanowsky. "Chapel Iterators: Providing Tiling for the Rest of Us," won 3rd place in Supercomputing SRC for undergraduates, November 2014.
15. Michelle Strout in collaboration with Ian Bertolacci, Catherine Olschanowsky, Brad Chamberlain, Ben Harshbarger, and David Wonnacott. "Practical Diamond Tiling for Stencil Computations Using Chapel Iterators," Presented at the ACM SIGPLAN 2nd Annual Chapel Implementers and Users Workshop (CHIUW), June 2015.

## 5 Conclusion

The DOE Early Career grant that supported the SAIMI project and my transition from an assistant professor to an associate professor has had a significant impact on my research and career and the careers of those who worked on the project. The most impactful work has been the incorporation of advanced tiling techniques within Chapel Iterator libraries and the Loop Chain abstraction. We are continuing to research and develop techniques for applying these technologies to full applications that are important to the DOE mission. Ground breaking work from the SAIMI project includes incorporating into the Chill program optimizer various inspector/executor transformations for sparse and irregular applications. Collaborations and research funded through the SAIMI project will continue to contribute to the DOE mission ASCR mission of enabling timely science and engineering progress by leveraging computational simulation. As the PI for SAIMI, I thank the DOE, my research collaborators, and the tax payers for funding such research.

## 6 Acknowledgements

The SAIMI project owes its success to numerous collaborators. We would like to especially thank Mary Hall at the University of Utah, Brad Chamberlain at Cray, Dave Wonnacott at Haverford, Paul Kelly at the Imperial College of London, Phil Colella and Brian van Straalen at LBNL, Stephen Guzin and Xinfeng Gao at Colorado State University, Michael Norrish at NICTA, and John Dennis at NCAR in Boulder. With these collaborators we were able to push forward and develop innovative approaches to orthogonally specify implementation details and more importantly evaluate them within the context of real applications. We would also like to thank Dan Quinlan and his research group at LLNL for providing the ROSE compiler infrastructure for other researchers.

## References

[1] Mesa software project at Colorado State University. `http://www.cs.colostate.edu/saxs/index.php?title=Mesa_Project`.

[2] Saxs: Data analysis for small angle x-ray scattering at Colorado State University. `http://www.cs.colostate.edu/hpc/SAXS/`.

[3] P. An, A. Jula, S. Rus, S. Saunders, T. Smith, G. Tanase, N. Thomas, N. Amato, and L. Rauchwerger. Stapl: An adaptive, generic parallel programming library for c++. In *Workshop on Languages and Compilers for Parallel Computing (LCPC)*, Cumberland Falls, Kentucky, August 2001.

[4] C. Bastoul. Code generation in the polyhedral model is easier than you think. In *Proceedings of the 13th Interntional Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2004.

[5] I. J. Bertolacci, C. Olschanowsky, B. Harshbarger, B. L. Chamberlain, D. G. Wonnacott, and M. M. Strout. Parameterized diamond tiling for stencil computations with chapel parallel iterators. In *Proceedings of the 29th International Conference on Supercomputing (ICS)*, 2015.

[6] D. Callahan, B. Chamberlain, and H. Zima. The cascade high productivity language. In *Proceedings of the Ninth International Workshop on High-Level Parallel Programming Models and Supportive Environments*, pages 52–60, 2004.

[7] B. Chamberlain, D. Callahan, and H. Zima. Parallel programmability and the chapel language. *International Journal of High Performance Computing Applications*, 21(3):291–312, 2007.

[8] L. Dagum and R. Menon. Openmp: An industry-standard api for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1):46–55, 1998.

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Sixth Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[10] S. Dinkins, B. Kreaseck, and M. M. Strout. Steps toward simplifying sparse matrix data structures. In *Proceedings of the Colorado Celebration of Women in Computing (CCWIC)*, November 4-5, 2010.

[11] P. Feautrier. Some efficient solutions to the affine scheduling problem. I. one-dimensional time. *International Journal of Parallel Programming*, 21(5):313–347, October 1992.

[12] M. Griebl, C. Lengauer, and S. Wetzel. Code generation in the polytope model. In *In IEEE International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 106–111. IEEE Computer Society Press, 1998.

[13] M. Hall, J. Chame, J. Shin, C. Chen, G. Rudy, and M. M. Khan. Loop transformation recipes for code generation and auto-tuning. In *Proceedings of the Workshop on Languages and Compilers for Parallel Computing (LCPC)*, 2009.

[14] M. Kandemir, A. Choudhary, J. Ramanujam, and P. Banerjee. A framework for interprocedural locality optimization using both loop and data layout transformations. In *Proceedings of the 28th International Conference on Parallel Processing (28th ICPP'99)*, Aizu-Wakamatsu, Fukushima, Japan, 1999. University of Aizu.

[15] W. Kelly and W. Pugh. Finding legal reordering transformations using mappings. In K. Pingali, U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Proceedings of the 7th International Workshop on Languages and Compilers for Parallel Computing*, volume 892 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 8–10, 1994.

[16] C. D. Krieger, A. Stone, and M. M. Strout. Mechanisms that separate algorithms from implementations for parallel patterns. In *Workshop on Parallel Programming Patterns (ParaPLOP)*, March 2010.

[17] C. D. Krieger, A. I. Stone, and M. M. Strout. Qualitative evaluation criteria for parallel programming models. In *Proceedings of the Fun Ideas and Thoughts Session at PLDI*, June 8, 2010.

[18] C. D. Krieger and M. M. Strout. A fast parallel graph partitioner for shared-memory inspector/executor strategies. In *Proceedings of the 25th International Workshop on Languages and Compilers for Parallel Computing (LCPC), to appear*, September 2012.

[19] C. D. Krieger, M. M. Strout, C. Olschanowsky, A. Stone, S. Guzik, X. Gao, C. Bertolli, P. H. Kelly, G. Mudalige, B. V. Straalen, and S. Williams. Loop chaining: A programming abstraction for balancing locality and parallelism. In *Proceedings of the 18th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS)*, May 2013.

[20] C. D. Krieger, M. M. Strout, J. Roelofs, and A. Bajwa. Executing optimized irregular applications using task graphs within existing parallel models. In *Proceedings of the Second Workshop on Irregular Applications: Architectures and Algorithms ($IA^3$) held in conjunction with SC12*, November 11, 2012.

[21] M. Norrish and M. M. Strout. An approach for proving the correctness of inspector/executor transformations. In *Proceedings of the 27th International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, 2014.

[22] C. Olschanowsky, M. M. Strout, S. Guzik, J. Loffeld, and J. Hittinger. A study on balancing parallelism, data locality, and recomputation in existing pde solvers. In *In The IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, November 2014.

[23] L. Rauchwerger, F. Arzu, and K. Ouchi. Standard templates adaptive parallel library. In *Proceedings of the 4th International Workshop on Languages, Compilers and Run-Time Systems for Scalable Computers (LCR)*, Pittsburg, PA, May 1998.

[24] L. Renganarayana, D. Kim, M. M. Strout, and S. Rajopadhye. Parameterized loop tiling. *CM Transactions on Programming Languages and Systems (TOPLAS)¡*, 34(1), May 2012.

[25] V. Sarkar and R. Thekkath. A general framework for iteration-reordering loop transformations. In *Proceedings of the ACM SIGPLAN 1992 Conference on Programming Language Design and Implementation (PLDI)*, pages 175–187, June 1992.

[26] A. Stone, J. Dennis, and M. Strout. Establishing a miniapp as a programmability proxy (poster). In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 333–334, New York, NY, USA, 2012. ACM.

[27] A. Stone, J. Dennis, and M. M. Strout. The cgpop miniapp. http://www.cs.colostate.edu/hpc/cgpop/, June 30 2011.

[28] A. Stone, J. Dennis, and M. M. Strout. The CGPOP miniapp, version 1.0. Technical Report Technical Report CS-11-103, Colorado State University, July 1 2011.

[29] A. Stone and M. M. Strout. Programming abstractions to separate concerns in semi-regular grids. In *In Proceedings of the 27th International Conference on Supercomputing (ICS)*, June 2013.

[30] A. I. Stone, J. M. Dennis, and M. M. Strout. Evaluating coarray fortran with the cgpop miniapp. In *Proceedings of the Fifth Conference on Partitioned Global Address Space Programming Models (PGAS)*, October 15, 2011.

[31] M. M. Strout. Compilers for regular and irregular stencils: Some shared problems and solutions. In *Proceedings of Workshop on Optimizing Stencil Computations (WOSC)*, October 27, 2013.

[32] M. M. Strout, L. Carter, and J. Ferrante. Compile-time composition of run-time data and iteration reorderings. In *Proceedings of the 2003 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, June 2003.

[33] M. M. Strout, G. George, and C. Olschanowsky. Set and relation manipulation for the sparse polyhedral framework. In *Proceedings of the 25th International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, September 2012.

[34] M. M. Strout, C. Krieger, A. Stone, C. Wilcox, J. Dennis, and J. Bieman. Evaluating the separation of algorithm and implementation within existing programming models. In *Proceedings of SciDAC*, 2011.

[35] M. M. Strout, A. LaMielle, L. Carter, J. Ferrante, B. Kreaseck, and C. Olschanowsky. An approach for code generation in the sparse polyhedral framework. Technical Report CS-13-109, Colorado State University, December 2013.

[36] M. M. Strout, F. Luporini, C. D. Krieger, C. Bertolli, G.-T. Bercea, C. Olschanowsky, J. . Ramanujam, and P. H. Kelly. Generalizing run-time tiling with the loop chain abstraction. In *In 28th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2014.

[37] A. Venkat, M. Hall, and M. M. Strout. Loop and data transformations for sparse matrix code. In *In Programming Languages Design and Implementation (PLDI)*, 2015.

[38] A. Venkat, M. Shantharam, M. Hall, and M. M. Strout. Non-affine extensions to polyhedral code generation. In *In International Symposium on Code Generation and Optimization (CGO)*, February 2014.

[39] C. Wilcox, M. Strout, and J. Bieman. Mesa: Automatic generation of lookup table optimizations. January 2011.

[40] C. Wilcox, M. Strout, and J. Bieman. Tool support for software lookup table optimization. *Sci. Program.*, 19:213–229, Dec. 2011.

[41] C. Wilcox, M. M. Strout, and J. Bieman. Optimizing expression selection for lookup table program transformation. In *Proceedings of the 12th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, September 2012.

[42] C. Wilcox, M. M. Strout, and J. M. Bieman. An optimization-based approach to lookup table program transformations. *Journal of Software: Evolution and Process*, pages 533–551, September 21, 2013.

[43] M. E. Wolf and M. S. Lam. Loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):452–471, October 1991.

[44] D. G. Wonnacott and M. M. Strout. On the scalability of loop tiling techniques. Technical Report 2012-01, Dept. of Computer Science, Haverford College, Haverford PA, Aug. 2012.

[45] D. G. Wonnacott and M. M. Strout. On the scalability of loop tiling techniques. In *Proceedings of the 3rd International Workshop on Polyhedral Compilation Techniques (IMPACT)*, January 2013.