Extensions to Sparse Polyhedral Framework for Automatic Generation of Composable Inspectors/Executors

Payal Guha Nandy  
Mary Hall  
University of Utah  
payalgn@cs.utah.edu  
mhall@cs.utah.edu  

Eddie C Davis  
Catherine Olschanowsky  
Boise State University  
eddiedavis@u.boisestate.edu  
catherineloschan@boisestate.edu  

Mahdi Soltan Mohammadi  
Michelle Strout  
University of Arizona  
kingmohd@gmail.arizona.edu  
mstrout@cs.arizona.edu  

CCS CONCEPTS
• Software and its engineering → Compilers  
• Mathematics of computing → Computations on matrices;  

KEYWORDS
sparse computations; inspector/executor; parallelizing compilers

ICPP 2018, August 2018, Eugene, Oregon USA  
2018. This is the author’s version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in Proceedings of 47th International Conference on Parallel Processing (ICPP 2018). https://doi.org/10.1145/nmmmm.mmmmm.
Extensions to Sparse Polyhedral Framework for Automatic Generation of Composable Inspectors/Executors  

ACM Reference Format:  

ABSTRACT

Historically, compilers have been severely limited in their ability to optimize sparse computations found in irregular applications such as molecular dynamics simulations, finite element analysis, and big graph analysis due to the indirection that arises in indexing and looping over just the nonzero elements. In recognition of this property, inspector-executor strategies were developed to parallelize, simplify and/or improve the data locality of such computations. At runtime an inspector code traversed the index arrays to determine data access patterns and an executor code would use this information to derive transformed schedules and reordered data structures to execute the computation in a more efficient manner. Most research that uses inspectors relies on instantiating application specific inspector templates, invoking inspector library code, or manually writing inspectors. Our research is part of a project that aims to create a generalized framework for automatically generating Inspectors for a wide range of sparse matrix computations by extending the Sparse Polyhedral Framework (SPF) with composable loop and data transformations. SPF extends the polyhedral framework to represent runtime information with un-interpreted functions and inspector computations that explicitly realize such functions at runtime. Previously SPF was used to derive inspectors for data and iteration space reordering. Data transformations will now be introduced into SPF.  

We have started with defining the abstractions required for creating such a framework. Non-affine iteration and data reordering transformations are defined that derive run-time relations and perform data transformations in the inspector that are used by the executor. Uninterpreted functions capture the mapping between the original iteration and data space and the transformed iteration and data space. An Inspector Dependence Graph (IDG) is defined that represents the tasks the inspector must perform to generate at run time explicit versions of what are uninterpreted functions at compile time. Our compiler will generate the inspector code by walking the IDG and the original code is transformed into the executor code.  

Previous work has demonstrated the automatic generation of inspector/executor code, which orchestrates code and data transformations to derive high performance representations for the Sparse Matrix Vector Multiply, Sparse Matrix-Matrix Multiply and Stochastic Gradient Descent kernels. A feasibility analysis is currently underway to determine whether efficient elimination trees can be generated as a part of Cholesky factorization of sparse matrices leading to the structural representation of the lower triangular matrix.