ProtoX: A First Look

Het Mankad*, Sanil Rao*, Phillip Colella†, Brian Van Straalen†, Franz Franchetti*
*Electrical and Computer Engineering Department Carnegie Mellon University, Pittsburgh, PA, USA
†Lawrence Berekely National Laboratory, Berkeley, CA, USA
*{hmankad, sanilr, franzf}@andrew.cmu.edu, †{bvstraalen, pcolella}@lbl.gov

***Abstract*—We present a first look at ProtoX, a code generation framework for stencil operation that occurs in the numerical solution of partial differential equations. ProtoX is derived from Proto, a C++ based domain specific library which optimizes the algorithms used to compute the numerical solution of partial differential equations and SPIRAL, a code generation system that focuses on generating highly optimized target code. We demonstrate the construction of ProtoX by considering the 2D Poisson equation as a model problem and using the Jacobi method to solve it. Some of the code generated for this problem specification is shown along with initial speedup result.**

***Index Terms*—partial differential equations, stencil operations, code generation, Proto, SPIRAL**

I. Introduction

There is a myriad of application areas in the field of scientific computing and engineering where numerical solutions to partial differential equations (PDEs) are required to be computed. Numerical methods like the finite difference method (FDM), finite element method (FEM), finite volume method (FVM) and multigrid method are used to approximate the solutions to these PDEs. A key component of these algorithms is the stencil operation. Typically these numerical algorithms are iterative in nature resulting in performing the stencil operations multiple times. As such developers turn to libraries that provide stencil operations for them. One such library is Proto. It is a domain specific library written in C++ that provides a high level of abstraction for solving various PDEs using some of the aforementioned numerical methods. Proto's abstraction enables ease of programmability, but has drawbacks when it comes to performance. Many of Protos' abstractions can be fused and optimized together, resulting in better performance. However, abstraction fusion is something no compiler can easily perform. This results in additional burden on the library developers to manually introduce these optimizations. To enable abstraction fusion in Proto, we propose ProtoX, which is a C++ library based on Proto and runs a code generation system SPIRAL [1], [2] in the backend. The concept of using SPIRAL in the backend and a C/C++ based library in the front has shown positive results in the past [3], [4]. Some of the related works in the area of optimizing stencil computation with either automatic code generation or by optimizing data movement involved while performing stencil operation can be found in [5]–[8].

This project is funded by the DOE Office of Advanced Scientific Computing Research the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Contribution. Some of the main contributions of this work are: 1) A proof of concept of hooking SPIRAL code generation as backend into Proto is presented. 2) A Proto example is considered as a SPIRAL specification and a full program optimization including single kernel code gen and merged kernels across C++ functions are discussed for that example. 3) This work is a first look of ProtoX, so just basic block code generation is presented at this point.

II. Background

Proto. Proto is a C++ library designed to provide an intuitive interface that optimizes the designing and scheduling of an algorithm aimed at solving various PDEs numerically. In order to approximate the solution of a PDE, the domain is usually divided into either structured or non-structured grid of rectangles and the equations are discretized using methods like FDM, FEM or FVM. Currently, Proto takes into account a multidimensional rectangular, structured grid with periodic boundary condition for all model problems. There are four main C++ classes that implement a representation of the spatial rectangular grids. The `Point` class represents all the points in $\mathbb{Z}^D$. `Box` denotes the rectangular subsets of $\mathbb{Z}^D$ while `BoxData` class represents the multidimensional data arrays. Another important class in Proto is the `Stencil` class which is where stencils are defined as self-contained objects. In order to apply the stencils or any other function in a pointwise manner, Proto uses the `forall` function. It takes in as an argument the function which is to be applied in a pointwise manner and the corresponding `BoxData` used for it.

SPIRAL. SPIRAL is a GAP based code generation system that was initially developed to automatically generate optimized C/C++ programs for linear transforms like the discrete Fourier Transform (DFT), discrete cosine transform and many more [1], [2]. SPIRAL uses the Signal Processing Language (SPL) to develop algorithms for these signal processing transforms. SPL is a declarative mathematical language that expresses linear transforms as matrix-vector product. Here, the matrix is considered to be an operator with one input and one output vector. Operator Language (OL) is a superset of SPL and is the result of the ongoing efforts to expand the scope of SPIRAL beyond signal processing transforms [9]. One of the main difference between SPL and OL is that in OL the operators can have multiple input and output vectors. The mathematical operations described in OL are then placed into a rewrite system which in the end generates an optimized code for the problem specification.

```
// Defining the 5-pt Laplacian stencil
Stencil<double> laplace = Stencil<double>::Laplacian();
for (int iter = 0; iter < maxiter; iter++){
...
  // Solve for all boxes with each Box of size 64x64
  for (auto dit=phi.begin();*dit != dit.end();++dit){
    BoxData<double>& phiPatch = phi[*dit];
    BoxData<double>& rhoPatch = rho[*dit];

    // Compute the Laplacian
    BoxData<double> temp = laplace(phiPatch,wgt);

    // Jacobi iteration
    forallInPlace(jacobiUpdate,phiPatch,temp,
    rhoPatch,lambda);
 }
  // Computing || ||_{inf}
  double resmax=computeMaxResidualAcrossProcs(phi,
  rho,dx);
}
```

Fig. 1: Sample Proto code for the 2D Poisson problem

III. ProtoX

In this section we will describe the structure of ProtoX. The idea is to interpret Proto as a Domain Specific Language (DSL) with the help of SPIRAL. This is done by first interpreting a Proto program as a mathematical specification and then map the Proto program specification to an OL expression. This will help generate a highly optimized C++ code. We will explain these ideas with respect to a 2D Poisson equation.

2D Poisson equation. The Poisson equation is given as,

$$\Delta\phi(x,y) = \rho(x,y), \quad x,y \in \Omega := [0,1]\times[0,1], \tag{1}$$

where ρ is a given function and ϕ is what we are solving for. Δ is the Laplace operator. We use the 5-pt stencil as a second order finite difference approximation of the Laplacian. The Jacobi iteration method is implemented here to find the solution of (1). The three main steps involved in this algorithm are *Step 1*: Applying the 5-pt stencil to the given initial guess for ϕ. *Step 2*: Approximate the new value for ϕ using the Jacobi iteration method. *Step 3*: Check the convergence criterion. In Proto each of these steps correspond to a C++ function call.

In order to test the initial prototype of ProtoX, two approaches are taken to generate the code for the algorithm mentioned above. The first approach is to generate the code in SPIRAL corresponding to each of the function calls in Proto separately. The second approach involves merging all three steps of the algorithm into a single step computation. This is one of the main advantages of using SPIRAL as the backend for ProtoX.

The SPL breakdown rules in SPIRAL for computing $n\times n$ 2D Poisson equation with $m\times m$ interior elements are

$$\text{Poisson2D}_{n,m,t}^{\ell,w,a} \rightarrow \begin{bmatrix}\text{Jacobi}_{n,m,w,l}\\ \|.\|_{\infty}^{n,m,a}\end{bmatrix} \circ \left(\begin{bmatrix}\text{I}_{n^2}\\ \text{Laplace2D}\end{bmatrix} \oplus \text{I}_{n^2}\right), \tag{2}$$

$$\text{Laplace2D}_{n,m,t} \rightarrow \text{Scatter}_{n^2\times m^2} \circ [\text{Filt}(t)]_{i=0}^{m^2}, \tag{3}$$

$$\text{Jacobi}_{n,m,w,l} \rightarrow (1,w,-\lambda) \otimes \text{I}_{n^2}, \tag{4}$$

$$\|.\|_{\infty}^{n,m,a} \rightarrow (0, 1/(a^2), -1) \otimes \text{I}_{n^2}. \tag{5}$$

Here t denotes the filter taps corresponding to the 5-pt stencil. ℓ, w and $, a$ are scalar parameters used for the Jacobi iteration and the Laplacian. The symbol [] denotes *vertical stacking*, $[]_{i=0}^{m^2}$ is *iterative vertical stacking* and $\text{Scatter}_{n^2\times m^2}$ denotes the *scatter matrix* [7] in SPIRAL. A sample of the code gen is shown in Fig. 2.

Speedup. The current CPU runtime taken by the baseline Proto code (see Fig. 1) for 100 Jacobi iterations is about $\approx 2.5s$ while it takes $\approx 0.4s$ for the corresponding SPIRAL generated code for ProtoX. This gives ProtoX $6\times$ speedup over Proto.

```
void poisson2D(double *Y, double *X, double weight1,
double lambda1, double *rhs, double a_h1, double *retval1){
  static double T1[4357];
  static double T2[13068];
  static double T3[8712];
  ...
  // Computing the Laplacian
  for(int i13 = 0; i13 <= 4095; i13++) {
    int a691;
    a691 = ((66*(i13 / 64)) + (i13 % 64));
    T2[(a691 + 4356)] = ((T3[(a691 + 1)]
    - (4.0*T3[(a691 + 67)]))+ T3[(a691 + 66)]
    + T3[(a691 + 68)] + T3[(a691 + 133)]);
  }
  ...
  // Jacobi iteration
  for(int i6 = 0; i6 <= 4355; i6++) {
    T1[i6] = ((T8[i6] + (weight1*T8[(i6 + 4356)]))
    - (lambda1*T8[(i6 + 8712)]));
  }
  ...
  // Computing || ||_{inf}
  for(int i10 = 0; i10 <= 4355; i10++) {
    T14[i10] = (((1 / (a_h1*a_h1))*T15[(i10 + 4356)])
    - T15[(i10 + 8712)]);
  }
  ...
  for(int i2 = 0; i2 <= 4095; i2++) {
    t3 = ((((T13[i2] >= t3))) ? (T13[i2]) : (t3));
  }
  ...
}
```

Fig. 2: SPIRAL generated code for the merged 2D Poisson equation for a `Box` of size 64×64

IV. Conclusion

This work demonstrates a proof of concept of having SPIRAL code generation as a backend to Proto using the 2D Poisson problem. It shows that by writing a Proto program as a SPIRAL specification, we can interpret Proto as a DSL and powerful code generation is possible. This work is an early prototype, so only CPU results have been shown. The future goal is to add more targets and make ProtoX interoperable with FFTX [3] to do cross library optimization.

References

[1] M. Puschel, J. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. Johnson, and N. Rizzolo, "Spiral: Code generation for dsp transforms," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 232–275, 2005.

[2] F. Franchetti, Y. Voronenko, and M. Püschel, "Formal loop merging for signal transforms," *SIGPLAN Not.*, vol. 40, no. 6, p. 315–326, jun 2005.

[3] F. Franchetti, D. G. Spampinato, A. Kulkarni, D. Thom Popovici, T. M. Low, M. Franusich, A. Canning, P. McCorquodale, B. V. Straalen, and P. Colella, "Fftx and spectralpack: A first look," in *2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW)*, 2018, pp. 18–27.

[4] S. Rao, A. Kutuluru, P. Brouwer, S. McMillan, and F. Franchetti, “Gbtlx: A first look,” in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1–7.

[5] B. Hagedorn, L. Stoltzfus, M. Steuwer, S. Gorlatch, and C. Dubach, “High performance stencil code generation with lift,” ser. CGO 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 100–112.

[6] T. Brandvik and G. Pullan, “Sblock: A framework for efficient stencil-based pde solvers on multi-core platforms,” in *2010 10th IEEE International Conference on Computer and Information Technology*, 2010, pp. 1181–1188.

[7] M. Bolten, F. Franchetti, P. H. J. Kelly, C. Lengauer, and M. Mohr, “Algebraic description and automatic generation of multigrid methods in spiral,” *Concurrency and Computation: Practice and Experience*, vol. 29, no. 17, p. e4105, 2017, e4105 cpe.4105.

[8] P. Basu, M. Hall, S. Williams, B. Van Straalen, L. Oliker, and P. Colella, “Compiler-directed transformation for higher-order stencils,” in *2015 IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 313–323.

[9] F. Franchetti, F. de Mesmay, D. McFarlin, and M. Püschel, “Operator language: A program generation framework for fast kernels,” in *Domain-Specific Languages*, W. M. Taha, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 385–409.