Research Statement - Matthew Rocklin

The goal of my research is to promote independent development of mathematical and algorithmic solutions in scientific computing. Problems in this domain require a depth of scientific, mathematical, and computational expertise which are hard to find in combination in an individual. My research endeavors to separate the scientific computing problem so that experts in each of these fields may contribute independently without close collaboration with the others.

This is hard. Problems in scientific computing often have high performance standards requiring tight integration between mathematical and algorithmic code. My work focuses around array programming, an effective abstraction layer between algorithmic and mathematical code. The following is a sequence of independent projects which may be used together to describe and solve standard numerical problems.

Compiling Matrix Expressions into BLAS/LAPACK Calls

Many scientific problems may be compactly represented as a sequence of matrix expressions. This motivated the development of BLAS and LAPACK, ubiquitous interfaces to libraries of dense linear algebra routines. They include thousands of well-tuned functions for special mathematical cases. While BLAS/LAPACK are powerful, their utility is limited by the ability of scientific programmers to select and call the correct sequence of subroutines to solve their problem.

We use logic programming, pattern matching, programmatic control strategies and a declarative definition of BLAS/LAPACK to compile a natural mathematical description of matrix expressions into a DAG of specific BLAS/LAPACK calls. We can then generate Fortran code that uses these libraries in a sensible way. We are able to gain substantial speedups not by implementing better algorithms but rather by selecting and composing the correct ones.

Static Scheduling onto Heterogeneous Architectures

We represent BLAS/LAPACK computations as a directed acyclic graph (DAG). If we have a parallel machine we may wish to schedule the individual jobs onto the different resources. We may also wish to use heterogeneous compute nodes (e.g. CPUs + GPUs) if available. Due to uncertainty of runtimes the scheduling task is traditionally done greedily and dynamically at runtime.

We analyze the feasibility of statically scheduling array operations onto heterogeneous architecture using integer programming. This problem is only feasible due to the following structure of array operations

1. The regularity of array computations makes them predictable and modelable.
2. The BLAS/LAPACK interface is implemented on a variety of hardware, exposing a common set of primitives on heterogeneous compute nodes
3. The low complexity of these DAGs enables the use of traditionally inaccessible, NP-hard algorithms and their approximations.

DAG Ordering

Finally, when we execute a single DAG on a sequential machine, we select the order in which the tasks are run. Several valid orderings may satisfy the data dependencies, giving us room to declare and satisfy additional policies. One such policy would be to maximize communication/computation overlap by starting asynchronous transfer jobs as early as possible and starting blocking waits as late as possible.

We represent each policy (e.g. data-dependence or communication-overlap) as a separate DAG. Because the policies may conflict we also enforce a meta-policy (e.g. data-dependence trumps communication-overlap) as a super-DAG. We then reduce this problem to the maximal union of a DAG of DAGs. We must find a conglomerate DAG which contains as many edges from each DAG as possible, respecting their relative priorities.

The method used to describe and conditionally merge DAGs is more general than asynchronous communication. We could express other policies (e.g. free memory as early as possible) with the same framework. This declarative description enables rapid prototyping of policy by non-expert users.

Conclusion

These projects are designed to be composed so that they can solve scientific problems in concert. They were also designed for independent use and development. Care was taken so that the interface of each project is accessible to mathematical programmers.

The desired result is that different experts can contribute to specific projects in a pipeline without deep knowledge of the other components. My hope is that by separating these components we

1. Enable reuse and give moderately high performance computing to a wider audience
2. Enable contributions from that wider audience back to high performance computing

This work is integrated into scientific and machine learning open source projects with large communities (SymPy, Theano). I have a small but growing base of users generating feedback.

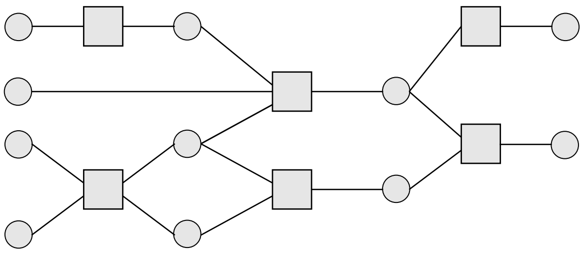
This visually represents how these projects might be used together to produce a code to compute the Kalman filter on a particular two-node network. (Images not a strict interpretation.)

Kalman Filter

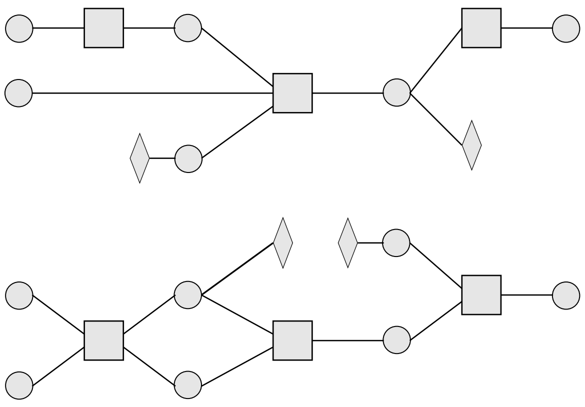
$$
\begin{aligned}
\mu' &= \mu + \Sigma H^T + \left(R + H\Sigma H^T\right)^{-1}(H\mu - data) \\
\Sigma' &= \left(\mathbb{I} - \Sigma H^T \left(R + H\Sigma H^T\right)^{-1} H\right)\Sigma
\end{aligned}
$$

Σ – symmetric, semi-positive-definite
R – symmetric, positive-definite
H – full rank

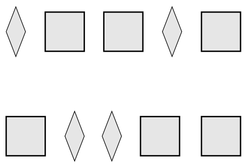
BLAS/LAPACK compilation



Static Scheduling



DAG Ordering



Legend

● - variable
■ - computation
♦ - communicaiton