Discussion and Future Directions

Dexter Barrows
March 23, 2016

Right off the bat, we can see that the IF2 / parametric bootstrapping framework produces great results. This framework consistently out-performs both the HMCMC framework and S-mapping by itself or with Dewdrop Regression. This isn't to say that the results produced by the other methods are poor, but rather that the ones produced by IF2 are noticibly better. This is true in every scenario we have explored here, and is particularly pronounced in the SIRS and spatial forecasting set-ups.

A surprise has been how well S-mapping has performed. Given the almost ludicrously shorter running times exhibited by S-mapping, it is almost shocking how well it performs. In the SIRS scenario it produces results only slightly less accurate than the other two methods, and is even the most accurate at predicting the rise to the second outbreak peak. And in the spatial scenario it performs almost as well as IF2, and much better than HMCMC. The critical point here is that S-mapping, with its relative ease of implementation, efficiency, and accuracy would make a great "first-blush" forecasting tool that could be run and give a good prediction well before the other methods could even be set up.

1 Parallel and Distributed Computing

Whenever running times are discussed, we must consider the current computing landscape and hardware boundaries. In 1965, Intel co-founder Gordon E. Moore published a paper in which he observed that the number of transistors per unit area in integrated circuits double roughly every year. The consequence of this growth is the approximate year-over-year doubling of clock speeds (maximum number of sequential calculations performed per second), equivalent to raw performance of the chip. This forecast was updated in 1975 to double every 2 years and has held steady until the very recent past [*Nature ref*].

Recently, transistor growth has begin to falter. This is due to several factors. The size of the transistors themselves has become so small that the next generation of processors would need to use transistors only 10-15 atoms across, at which point their ability to transport electrons becomes unreliable, and their behaviours will start to be affected by quantum uncertainty. Second, denser transistor packing would require aggressive cooling strategies as the Thermal Design Power (TDP), or the heat generated by such chips would increase dramatically.

To compensate for these limitations, chip manufacturers have instead redesigned the internal chip structures to consists to smaller "cores" within a single CPU die. The resulting processing power per processor then stays on track with Moore's Law, but keeps the clock speeds of each individual core, and consequently the thermal dissipation requirement, under control.

Of course this raises many problems on the software and algorithm side of computing. Using several smaller cores instead of a single large has the distinct disadvantage of lack of cohesion – the cores must execute instructions completely decoupled from each other. This means algorithms have to be redesigned, or at least rewritten at the software level to consists of multiple independent pieces that can be run in parallel. This practice is known as parallelization.

Some compilers can actually detect areas in source code that contain obvious room for parallel execution (for example loop iterations with no dependence), and automatically generate machine code that can run on a multiprocessor with little to no performance overhead. This technology is still nascent and cannot be relied to operate successfully on anything but the most basic algorithms, and so usually we musts identify areas for parallelization and take advantage of them or risk not utilizing the full power of our machines. Further, high-performance computing essentially requires parallelization in its current form as large clusters and supercomputers rely on distributed computing "nodes".

When working with computationally intensive algorithms, particularly iterative methods such those used in this paper, the question of parallelism naturally arises. It may come as no surprise that the potential degrees of parallelism varies between methods.

Hamiltonian MCMC is cursed with high dependence between iterations. While HMCMC has an advantage over "vanilla" MCMC formulations in terms of efficiency of step acceptance and ease of exploration of the parameter per number of samples, each sample still depends entirely on the preceding one, and at a conceptual level the construction of a Markov Chain *requires* iterative dependence. We cannot simply take an accepted step, compute several proposed steps accept/reject them independently – doing so would break the chain construction and could potentially bias our posterior estimate to boot. We can, however, process multiple chains simultaneously and merge the resulting samples. If the required number of samples for a problem were large and the required burn-in time were low, this methods could prove effective. However, the parallel burn-in sampling is still inefficient as it is a duplication of effort with limited pay-off – in the sense that the saved sample to discarded burn-in sample ratio would not be as efficient as running a single long chain. Thus while parallelism via multiple independent chains would help with a reduction in wall clock running times, it would result in an *increase* in total computer time.

With regards to the bootstrapping process we used here, it should be clear that each bootstrap trajectory is completely independent, and thus this component of the forecasting framework can be considered "embarrassingly" parallel. Unfortunately, however, this is the least computationally demanding part of the process by several orders of magnitude, and so working to parallelize it would provide little advantage.

In the case of IF2, we have a decidedly different picture. In IF2 we have 5 primary steps in each data point integration:

- Forward evolution of the particles' internal system state using their parameter state
- Weighting those state estimates against the data point using the observation function
- Particle weight normalizations
- Resampling from the particle weight distribution
- Particle parameter perturbations

Luckily, 4 of the 5 steps can be individually parallelized and run on a per-particle basis. The particle weight normalizations, however, cannot. Summation "reductions" are a well-known problem for parallel algorithms; they can be parallelized to a degree using binary reduction, but that only reduces the approximate running time from $\mathcal{O}(n)$ to $\mathcal{O}(\log(n))$. The normalization process requires the particles' weight sum to be determined, hence the unavoidable obstacle of summation reductions rears its head. However this is in practice a less-taxing step, and its more demanding siblings are more amenable to parallelization.

Further, the full parametric bootstrapping process is incredibly computationally demanding, and also completely parallelizable. Each trajectory requires a fair bit of time to generate, on the order of of the original fitting time, and can be computed completely independently. Hence, IF2 is a very good candidate for a good parallel implementation.

A future offshoot of this project would be a good parallel implementation of both the IF2 fitting process and the parametric bootstrapping framework. And ideal platform for this work would be NVIDIA's Compute Unified Device Architecture (CUDA) Graphics Processing Unit (GPU) computing framework. While a CUDA implementation of a spatial epidemic IF2 parameter fitting algorithm was implemented, it lacked a good front-end implementation, R integration, and a parametric bootstrapping framework and so was not included in the main results of this paper. The code, however, as well as some preliminary results, are included in the appendices.

S-mapping, like the other two methods, is parallelizable to a degree. However, the S-map is already a great deal faster than the other two methods, and in the worst case (paired with Dewdrop Regression and applied to a spatiotemporal data set) still only takes a few minutes to run. Setting this observation aside, if one were investing in developing a faster S-map implementation, this is certainly possible. By far the most computationally expensive component of the algorithm is the SVD decomposition, and algorithms exist to accelerate it via parallelization. Further, each point in the forecast can be computed separately; in the cases similar to the one here with application to spatiotemporal prediction, there can be a significant number of these points.

Further work developing parallel implementations of forecasting frameworks could be advantageous if the goal was to generate accurate forecasts under more stringent time limitations. IF2 seems to have emerged as a leader in forecast accuracy, if not in efficient running times, and demonstrates high potential for parallelism. Expansion of the CUDA IF2 (`cuIF2`) implementation to include a parallel bootstrapping layer and R integration could prove very promising.

2 IF2, Bootstrapping, and Forecasting Methodology

The parametric bootstrapping approach used to generate additional parameter posterior samples and produce forecasts has proven effective, but not necessarily computationally efficient.

A recent paper utilising IF2 for forecasting [*King reference*] generated trajectories using IF2, parameter likelihood profiles, weighted quantiles, and the basic particle filter. The parameter profiles were used to construct a bounding box to search for good parameter sets, within which combinations of parameters to generate forecasts were selected using a Sobol sequence. Finally the forecasts were combined using a weighted quantile, taking into account the likelihood of the parameter sets used. Whether this approach would result in higher quality forecasts or lower running times is of interest, and could serve as a future research direction.

Expanding on this, there are other bootstrapping approaches that could be used to produce forecasts. A paper focusing solely on using IF2 with varied bootstrapping approaches and determining a forecast accuracy versus computational time trade-off curve of sorts would be useful, and would be another step towards establishing which tools are best for which jobs.

3 Fin

The overarching theme in this paper, from the theoretical considerations to the results to the discussion, is that there still exists no “silver bullet” for forecasting problems. Largely you can decide, as the user, how accurate you need your results to be, how much computer time you have at your disposal, and how fast you need your results, and select the method that best satisfies your needs. If speed is the priority, then you can use S-mapping to get very quick and relatively accurate results. If you require accuracy above all else, you must turn to heavier results such as IF2, HMCMC, and parametric bootstrapping in order to produce the cleanest forecast possible. And this only represents three data points in a larger picture. There are a wide variety of methods that are similar but not identical to methods explored here, each with their own positive and negative attributes, their own advantages and disadvantages, and that are ultimately likely to fill out our spectrum of methods more completely. Thus future work should focus on attempting further direct comparison across a wider swath of techniques, and implementing those techniques in a parallel fashion to take advantage of the current and future landscape of high-performance computing.