High Performance Parallel Disease Detection: an Artificial Immune System for Graphics Processing Units

Nicholas A. Sinnott-Armstrong and Delaney Granizo-Mackenzie and Jason H. Moore
Computational Genetics Laboratory
Dartmouth Medical School
Lebanon, NH 03756
jason.h.moore@dartmouth.edu

1. INTRODUCTION

Disease association studies aim to characterize an association between an individual's genotype and their disease status. For the past decade, the reduced costs of genotyping have led to massive case-control studies which examine hundreds of thousands of genomic variations across thousands of individuals, but these studies have failed to produce many of the significant associations which they were heralded to bring to light. Even techniques which can evaluate relevancy through pathway analysis or other annotation still suffer from an inability to detect large numbers of simultaneously interacting loci, as most studies only evaluate two way interactions.

It is computationally unfeasible to exhaustively search through every possible combination of loci in order to identify potential interactions. An alternative to exhaustive searches which does not limit the potential number of detectable interactions — only the distribution of likelihoods — is to use an evolutionary computing technique to refine explanatory models and successfully identify highly complex associations. Here, we evaluate an Artificial Immune System (AIS) as an evolutionary heuristic. An AIS maintains a population of antibodies that simulate genetic models, and then evolves these antibodies over a number of generations [1]. Penrod et al. [5] have shown that an AIS has promise in successfully finding epistatic models. One major limitation of the AIS algorithm is that it is relatively low performance; by comparison, exhaustive techniques are still potentially efficient when low-order solutions are desired. By using graphics cards as an execution platform for the AIS, high performance can be coupled with a minimum of assumptions and an efficient, fast, and powerful analysis solution emerges.

2. IMPLEMENTATION

GPUAIS is based on the PyCUDA framework [3], which integrates an easy-to-use Python interface for metaprogramming with a powerful CUDA-based backend for GPU execution. Using PyCUDA, high performance applications are easy to develop and maintain. Here we give an overview of the general AIS algorithm which runs on the PyCUDA platform.

An antibody consists of a sequence of rules (matching tests, such as `=2`, `!=0`, or `*` (don't care)), corresponding with attributes in the test data. Each antibody maps a genotype (a sequence of $\{0, 1, 2\}$) to a classification (sick or well). The core AIS algorithm depends on forming a subset of the antibodies, called the *elite*, which maintains the best classifiers at every generation and allows for variation on these existing antibodies through cloning and mutation. This elite is selected from the population of antibodies at every generation by choosing the antibodies which have the highest affinity ratio between cases (sick/affected individuals) and controls. The affinity metric is defined as the sum of the attribute-wise matching tests. Here is a step-by-step overview of the algorithm, along with the kernels associated with each step:

1. Initialize A vectors of n rules, termed antibodies. Each rule corresponds to single point of variation in the dataset, and can be either a wild card or prediction of the locus's genotype. (`genAntibodies`)
2. Repeat G times:
   (a) Compare each antibody with each individual, assigning 1 to correct rules and -1 to incorrect. Wildcards get 0. Sum along antibodies. (`calculateAffinity`, `sumSigns`)
   (b) Probabilistically choose the antibody such that for the most sick individuals the rule outputs summed to > 0. (`pickBest`)
   (c) Remove a certain percentage of the worst antibodies and replace them with mutated copies of the selected best antibody. Add some random antibodies to the population. (`cloneExpand`)
3. Return population of antibodies.

The CUDA implementation is based on a number of standard templates, including Langdon's linear congruent generator [4] and optimized parallel reduction structures [2]. It separates kernels into operations based on which orientation the threads should process data so that memory accesses are coherent whenever possible. There are options of a python or C dataset loader and step 2(b) is implemented both traditionally (using a sort-by-ratio in Python) and using the novel probabilistic picker in CUDA.

3. ALGORITHM

The GPU uses a probabilistic elite antibody selection algorithm which appears to work very well and is ideally suited to the parallel architecture of the graphics card. It cuts down the procedure from a the reference sorting-based $O(A^2)$ procedure to a $O(k)$ time procedure, where k is the number of parallel threads.

`calculateAffinity`, GPUAIS's most time intensive kernel, has an occupancy of 100% on all tested architectures and accounts for 90% of runtime. At least 25% of the GPU is utilized for all other kernels run each generation. These figures are based on tested thread counts, which were chosen to minimize runtime. Greater occupancy can be achieved with higher thread counts.

Each generation, the GPU runs a number of kernels in order to perform the steps of the AIS algorithm. Other than the random clone selection, all computation is performed device-side on the graphics processor. It is inefficient to perform this step on the GPU; the value is broadcast to a large number of threads and is well suited to storage in constant memory and a CPU-side computation.

The structure of the code is simple and understandable. The affinities are calculated for cases, then controls, and the

GECCO GPU Competition '10 Portland, Oregon USA

Atbs	Gens	1 GPU	2 GPU	3 GPU	4 GPU	6 GPU
ALL	ALL	18.96X	28.70X	33.67X	36.06X	35.90X
ALL	1000	21.17X	35.83X	44.67X	50.43X	55.33X
2000	ALL	23.12X	42.09X	55.61X	65.68X	74.20X
2000	1000	23.77X	45.91X	63.77X	79.48X	101.29X

Table 1: The speedup of the GPU solution across various constraints on the number of antibodies and generations. "ALL" indicates that every tested configuration's median speedup was averaged.

recognition ratio leads to selection of the best. Those which performed best form an "elite" and, in the `cloneExpand` kernel, are clonally expanded into the rest of the antibody slots.

The code is written in CUDA and runs on all NVIDIA GPU's with compute capability 1.0 (G80 chipset) or higher.

The high performance parallel execution, as well as the shared memory, are exploited to ensure high speeds.

All antibody operations in an AIS are embarrassingly parallel; this allows for explicitly block-based execution and a high efficiency on all map and reduction steps.

4. SPEED

The speedup of the novel GPU solution was tested using a number of different datasets and across many parameters. In particular, datasets with 40, 60, and 80 loci and 400, 600, or 800 individuals, along with a dataset containing 20 loci and 400 individuals, were tested across a number of antibody counts (100, 200, 300, 500, 1000, and 2000) and generations (100, 300, 500, and 1000). For each GPU configuration (1, 2, 3, 4, or 6 GPUs) and the reference CPU implementation, each dataset was tested five times and the median of the results was used as a representative. The GPU implementation was tested on a graphics card workstation with a quad core Intel Core i7 920 processor, 12GB of RAM, and three NVIDIA GTX 295 graphics cards with 896MB of RAM per GPU. The CPU implementation was tested on an 8 core Xeon X5472 machine with 64GB of RAM. All reported speedups are from the real time reported by `time(1)`.

Table 1 gives the speedups obtained for various configurations of antibodies and generations. The speedup increase for larger datasets is due mostly to initialization overhead reduction and dataset loading speed differences.

The maximum median speedup obtained (using the largest dataset, 2000 antibodies, and 1000 generations) was 30.97X, 60.19X, 85.53X, 107.47X, and 143.16X.

The resource utilization of the kernels used in GPUAIS is high. The most time-critical kernel (approximately 90% of the kernel execution time), `calculateAffinity`, has an occupancy of 100%. Other kernels go up to approximately 50% occupancy; they use a large number of local variables in the random number generator and are limited by the total number of registers available on the device. A table of the resources used by each kernel is given in Table 5.

The program uses the Parallel Python library (available from `http://www.parallelpython.com/`) to enable seamless networked and local multiprocessor execution. Testing locally, significant speedups were possible by exploiting the multiple GPUs present in commodity cards such as the NVIDIA GTX 295. Initial tests used up to six GPUs on a single machine, but this can easily be extended to more graphics processors and/or to clusters of machines. GPUAIS has been tested on Linux and Mac OS X.

5. EVOLUTIONARY COMPUTATION

When developing AIS algorithms, a major issue is effective testing. One must run exhaustive parameter sweeps in order to verify the power of their algorithm and to help determine the next steps in development. Normally a super-computing cluster must be utilized for this testing, but our GPU solution enables a small desktop workstation to perform the same task in a very similar time-frame and facilitates faster algorithmic development. Additionally, the power of the algorithm increases with a larger search space [5]. The exact relationship will vary per algorithm and is not generally known; however, the search space is approximately $O(AG)$ where A is the number of antibodies and G is the number of generations. Thus, a speedup of N times enables a linear increase in search space and a corresponding linear power increase.

Name	% Time	Thread	Reg.	% Occupancy
`genAntibodies`	NA	8	28	25% (17%)
`calculateAffinity`	89.96%	128	10	100% (100%)
`sumSigns`	1.51%	16	17	25% (25%)
`pickBest`	1.97%	64	28	50% (33%)
`cloneExpand`	6.56%	128	29	50% (33%)

Table 2: The resource utilization of the GPU solution on various kernels. In all cases, shared memory was minimal. Percent of time is based on 500 antibodies in the population. Occupancy in parentheses is for devices with compute capability 1.0 or 1.1.

Providing more runtime to an AIS will increase the power and enable higher confidence in the results and by definition greater statistical significance. With ability to run with more antibodies we can explore more thoroughly interesting questions — such as how power relates to number of antibodies and generations, and whether it relates equally. Overall this will enable researches to quickly run tests and vastly speed up the development process.

6. CONCLUSIONS

The Artificial Immune System provides a powerful, directed search of a large combinatorial space and is able to efficiently detect signals which are hard to capture using traditional machine learning techniques. By improving the performance of AIS, larger datasets can be tackled by enabling searching of larger spaces. In addition, graphics card implementations are a useful way to improve the debug cycle while developing algorithms and tweaking their characteristics for optimal performance. Here, we show that GPU execution can increase the performance of an AIS by more than two orders of magnitude and still maintain a high degree of understandability and robustness.

7. ACKNOWLEDGMENTS

This work is funded by NIH grants LM009012, LM010098, AI59694, HD047447, and ES007373.

8. REFERENCES

[1] L. N. de Castro and J. Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer-Verlag, London, 2002.

[2] M. Harris. Optimizing Parallel Reduction in CUDA. NVIDIA Developer Technology Presentation, 2008.

[3] A. Kloeckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih. Pycuda: Gpu run-time code generation for high-performance computing. Technical Report 2009-40, Scientific Computing Group, Brown University, Providence, RI, USA, Nov. 2009.

[4] W. B. Langdon. A fast high quality pseudo random number generator for nvidia cuda. In *GECCO '09: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference*, pages 2511–2514, New York, NY, USA, 2009. ACM.

[5] N. M. Penrod, C. S. Greene, D. Granizo-MacKenzie, and J. H. Moore. Artificial immune systems for epistasis analysis in human genetics. In *EvoBIO*, pages 194–204, 2010.