

# PARALLELIZING AN IMMUNE-INSPIRED ALGORITHM FOR EFFICIENT PATTERN RECOGNITION

ANDREW WATKINS<sup>§</sup>, XINTONG BI, AND AMIT PHADKE  
{[andrew.xb7.aap50@cse.msstate.edu](mailto:andrew.xb7.aap50@cse.msstate.edu)

Department of Computer Science and Engineering,  
Mississippi State University, USA

<sup>§</sup>Also: Computing Laboratory,  
University of Kent, Canterbury, UK

## *ABSTRACT*

In recent years, there has been a growth of investigation into the use of the mammalian immune system as a source of inspiration and metaphor for computational tasks. One avenue of this investigation has been the exploration of the learning capabilities demonstrated by these biological systems. A second appealing aspect of biological immune systems is their inherent distributedness. This paper examines these two strains by proposing modifications to an existing immune-inspired algorithm to provide a more efficient, parallel version. This investigation is offered by way of a case-study in the gains parallelization can provide to current Artificial Immune Systems and as a proof-of-concept in the field of parallel immune learning.

## **INTRODUCTION**

In recent years, there has been a growth of investigation into the use of the mammalian immune system as a source of inspiration and metaphor for computational tasks (deCastro and Timmis 2002). One avenue of this investigation has been the exploration of the learning capabilities demonstrated by these biological systems (Watkins, Timmis, and Boggess 2003; Timmis and Neal 2001; deCastro and Von Zuben 2002). A second appealing aspect of biological immune systems is their inherent distributedness. This paper examines these two strains by proposing modifications to an existing immune-inspired algorithm to provide a more efficient, parallel version. Since very little work has been undertaken, to date, in the parallelization of immune-inspired **learning** algorithms, this investigation is offered by way of a case-study in the gains parallelization can provide to current Artificial Immune Systems (AIS) and as a proof-of-concept in the field of parallel immune learning.

The algorithm under investigation, CLONALG, utilizes the immunological concepts of memory cells, free antibodies, clonal selection, and affinity maturation to provide solutions for pattern recognition and function optimization tasks. This algorithm was developed by Leandro de Castro and offers us an ideal test bed for our initial explorations of immune learning in parallel. In (deCastro and Von Zuben 2002), the authors introduce an artificial immune system algorithm inspired primarily by the clonal selection theory first popularized by Burnet (1959). For a complete specification of this algorithm, please see (deCastro and Von Zuben 2002).

CLONALG's primary goal is in the development of memory cells which provide some representation of the data set. Similar to many evolutionary algorithms, CLONALG proceeds through a number of generations in the development of this representation. What really intrigues us about CLONALG is the almost embarrassingly parallel nature of the algorithm. Since the algorithm is one of the more simplistic immune-inspired models, this seemed to be an excellent starting place for our exploration of parallel computing techniques and Artificial Immune Systems.

## DESCRIPTION OF CLONALG AND ITS PARALLELIZATION

CLONALG, as presented in (deCastro and Von Zuben 2002), has two incarnations: one for pattern recognition tasks and one for multi-modal function optimization. We have decided to explore the pattern recognition version more thoroughly. This version of the algorithm proceeds in a similar fashion to many evolutionary algorithms; however, the inspiration for the algorithm is clearly from the realm of immune systems. The input (or antigens) to the algorithm consists of a collection of  $S$  patterns to be recognized. The algorithm begins by generating a random population ( $P$ ) of  $N$  cells (or antibodies), where  $|P| \geq |S|$ . For the current application, these cells simply consist of a binary feature string that represents a given binary pattern. A pattern from  $S$  is presented to each cell in  $P$ , and the affinity of each cell to the pattern is calculated. Affinity, for the current application, is based on hamming distance and is measured simply as the number of incorrect bits in the antibody when compared to the input pattern. Therefore, a lower affinity number implies a closer match to the input pattern. Once the affinity is calculated for each antibody in  $P$  as compared to a given input pattern, the best  $n1$  antibodies are selected to undergo somatic hypermutation as a mechanism for providing affinity maturation. The authors of (deCastro and Von Zuben 2002) specify that the mutation should be dependent on the affinity value of the antibody in two ways: 1) the number of mutated offspring a given cell is allowed to produce increases as the hamming distance decreases and 2) lower the hamming distance of an antibody the less mutation of the features should occur. CLONALG encourages the exploitation of the good solutions it has found by allowing high affinity (low hamming distance) cells to produce more offspring. It encourages exploration by increasing the mutation rate for low-affinity cells. The input pattern is then presented to each of the newly created offspring, and their affinity levels are calculated. The single best antibody is then compared to the memory cell for the given input pattern. If this antibody's affinity is better than that of the memory cell, the newly generated antibody replaces the established memory cell. Finally,  $n2$  random antibodies are created and replace existing cells in the population  $P$  of antibodies. This process continues until all the input patterns have been presented to the population. One cycle through the input patterns is considered one generation. The algorithm can continue for a fixed number of generations or until a certain convergence criterion is reached. For the current experiments, the algorithm is only halted when absolute convergence occurs. By absolute convergence, we mean that the memory cell set which has evolved exactly matches the input pattern set. While in real-world learning situations this convergence criterion would not be as useful as some fuzziest threshold which

would allow for greater generalization by the system, for our current experiments that examine the behavior of the algorithm as various parameters are adjusted we felt this was a sufficient stopping criterion. The algorithm is presented below, in terms of the immune processes employed.

- 1) *Initialization*: Create a random population of  $N$  antibodies  $\mathbf{P}$  and memory cells  $\mathbf{M}$ .
- 2) *Antigenic Presentation*: for each antigenic pattern in  $\mathbf{S}$  do:
  - a) *Clonal Expansion*: Determine the affinity of each antibody in  $\mathbf{P}$  to the given antigenic pattern. Select the best  $n1$  antibodies to produce clones.
  - b) *Affinity Maturation*: Mutate clones based on affinities.
  - c) *Clonal Selection*: Choose the best clone and replace the memory cell in  $\mathbf{M}$  if the clone has a better affinity.
  - d) *Metadynamics*: Replace  $n2$  antibodies in  $\mathbf{P}$  with randomly generated cells.
- 3) *Cycle*: Repeat step 2 for a fixed number of generations or until the memory cells in  $\mathbf{M}$  have converged to within some threshold.

Since, essentially, there is no connection among the cells in the CLONALG algorithm (as there would be in a network model of the immune system (Timmis and Neal 2001)), there is no real need for all of this process to occur on a single processor. The goal of the algorithm is simply to discover a set of memory cells and the development of a single memory cell can occur independently of the development of the other cells. For the parallelization of CLONALG, the input set is simply divided by the number of processes involved in the parallel job. Each process then evolves that number of memory cells. Once this finishes, a root process gathers all of these memory cells together in order to present a final result to the user of the evolved pattern recognizing cells. There are three parameters that we examined in our experiments:  $N$ , the size of the population  $\mathbf{P}$ ;  $n1$ , the number of antibodies selected for mutation and cloning during each step of the antigenic presentation; and  $n2$ , the number of random cells generated to replace cells in  $\mathbf{P}$  after each antigenic presentation. With the parallelization we need to decide how to handle these user-defined parameters. Currently, we have investigated two options: we can divide the parameter by the number of processes and let that be the setting for each process, or we can let each parameter be the full amount on each process. That is, for example, if the user specified 80 as the parameter for  $N$  and there are 4 processes in the parallel job, we can let the value of  $N$  be 20 at each of the processes or 80. Either way should have an impact on the performance of the algorithm.

## EXPERIMENTAL DESIGN, RESULTS AND DISCUSSION

For the experiments described below, the binary character recognition data set discussed in (deCastro and Von Zuben 2002) were used. This data set consisted of 8 binary patterns representing a 10x12 graphical representation of the arabic digits 0-4, 6, 7, and 9. The length of each pattern to be recognized, therefore, was 120. We investigated the time to convergence as well as the number of generations the algorithm took to converge (where convergence is as described above). In order to parallelize the algorithm, we simply divided the input data set across the processes which each then proceeded to use this

divided data set as its input. Once all of the processes completed, the resulting memory cells were gathered back to a root process, and the time and number of generations to convergence (max across the processes) were recorded. With most of the results presented below, we divided these parameter values across the number of processes as well. All results presented represent an average of three runs.

Table 1 shows the effects of varying the parameter  $N$  on the behavior of the algorithm both in terms of time to convergence and number of generations to convergence. This data is shown for runs across 1, 2, 4, and 8 processors and employs the division of parameters technique. One thing this table shows is that the  $N$  parameter tends to have a fairly significant effect on the time and number of generations to convergence. What is interesting is that there seems to be a definite trade-off between time and number of generations. That is, the amount of time to convergence increased as the value of  $N$  increases; however, the number of generations it took to reach this convergence seems to decrease as  $N$  increases. Also of note is that while increasing the number of processes definitely decreases the overall running time of the algorithm, this increase in processes increases the number of generations needed to converge to a solution. Practically speaking, the decrease in run-time is probably more important to us than the actual number of generations the algorithm took to reach this convergence. Additionally, this increase in number of generations to convergence should not be entirely surprising. With our current scheme of dividing the resources across the various processors, we decrease the number of items the algorithm has to cycle through during a given generation. Naturally, this decrease means that we can then process more cycles in the a given amount of time. The rate of convergence for larger values of our parameters seems to indicate that convergence is encouraged through the diversity of cells being processed which is increased as the various parameters (well  $N$ , at least) are increased.

Table 2 shows the effects of varying the parameter  $n1$  on the behavior of the algorithm both in terms of time to convergence and number of generations to convergence. This data is shown for runs across 1, 2, 4, and 8 processors and employs the division of parameter technique. Again we see that an increase in the parameter's value leads to an increase in the running time of the algorithm, and again, we see that the number of processors employed decreases the running time. However, the  $n1$  parameter has little distinguishable affect on the number of generations needed for convergence. Table 3 shows the effects of varying the parameter  $n2$  on the behavior of the algorithm both in terms of time to convergence and number of generations to convergence. This data is shown for runs across 1, 2, 4, and 8 processors and employs the division of parameter technique. The  $n2$  parameter seems to have little overall affect on the behavior of the algorithm, whether in run time or number of generations to convergence. It is possible that larger values for this parameter might produce an effect or that in more complex data sets this parameter might have more significance.

One metric for determining what is gained through the parallelization of an algorithm is speed-up. Speed-up is simply a ratio of the serial run time over the parallel execution time. Ideally, speed-up exhibits linear behavior such that with an increase in the number of processors one sees a proportional decrease in the run time. Figure 1 shows the speed-up obtained from a given run ( $N=40, n1=8$ ,

$n2=2$ ) for both styles of parameter division in the parallelization. What is striking about this curve is that we see superlinear speed-up when we employ our parameter division method of parallelizing CLONALG. We also see the almost ideal curve when we do not employ this parameter division across the processes. This implies that the parameter division method employed for the parallelization of CLONALG has a significant impact on its overall performance. From Table 1 and Table 2, we suspect this has to do with the reduction of the  $N$  and  $n1$  parameters more than anything else. As evidenced from the descriptions of the algorithm in the previous section, these two parameters have the greatest effect on the size of the antibody populations being generated and the interaction of this population with the input patterns. A reduction in one or both of these would reduce the number of cells that need to be manipulated as the input patterns are learned.

### CONCLUSIONS AND SOME REMARKS CONCERNING PARALLEL AIS

This paper has shown that even simple parallelization techniques can have a role in the field of Artificial Immune Systems. Since often the goal of machine learning systems is to efficiently assist humans in the finding of interesting patterns in large amounts of data, any techniques that can speed up this process should have value. From our experiments with CLONALG, we were able to achieve a great deal of reduction in processing time through some basic parallelization. This seems encouraging.

Beyond just increased computational efficiency, parallelizing immune-inspired algorithms could also enable us to explore the immune metaphors employed more fully. Biological immune systems are inherently decentralized and distributed. Yet, there has been a lack of exploration of the decentralized nature of the immune system particularly in immune-inspired *learning* algorithms. One question that needs to be addressed is what, if anything, will be gained through the use of parallel/distributed computing techniques when applied to these algorithms. Perhaps it will be something as simple as the time-savings as seen with our experiments with CLONALG. However, perhaps it will allow us to build more complex models of the immune system that can solve more complex problems. There is still much left to be explored in this direction including some of the obvious quality of population discussions presented in (Cantú-Paz 1998).

### REFERENCES

- Burnet, F. 1959. *The Clonal Selection Theory of Acquired Immunity*. Cambridge University Press.
- Cantú-Paz, E. 1998. A Survey of Parallel Genetic Algorithms. *Calculators Parallels*, 10 (2): 141-71.
- deCastro, L.N. and J. Timmis. 2002. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer-Verlag.
- de Castro, L. N. and Von Zuben, F. J. 2002. Learning and Optimization Using the Clonal Selection Principle. *IEEE Transactions on Evolutionary Computation*, 6(3): 239-51.
- Timmis, J. and Neal, M. 2001. A Resource Limited Artificial Immune System for Data Analysis. *Knowledge Based Systems*, 14(3-4):121-30.
- Watkins, A., J. Timmis, and L. Boggess, 2003. Artificial Immune Recognition System (AIRS): An Immune-Inspired Supervised Learning Algorithm. *Genetic Programming and Evolvable Machines*, In press.

**Table 1: Convergence Times and Number of Generations when Varying the  $N$  Parameter**

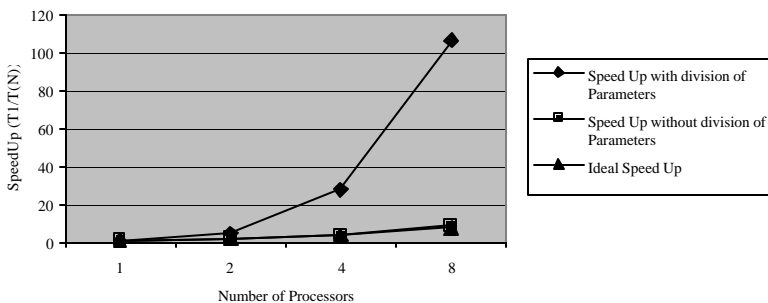
$N$	1 proc		2 procs		4 procs		8 procs	
	Conv. T(s)	Conv. Gen.	Conv. T(s)	Conv. Gen.	Conv. T(s)	Conv. Gen.	Conv. T(s)	Conv. Gen.
10	11.2	72	3.7	149	0.8	293	0.2	587
20	18.4	44	3.3	63	1.2	162	0.3	341
40	31.6	32	6.4	49	1.1	73	0.3	134
80	44.7	22	9.8	33	1.5	39	0.4	79
160	78.1	18	13.7	20	3.0	31	0.5	42

**Table 2: Convergence Times and Number of Generations when Varying the  $n1$  Parameter**

$n1$	1 proc		2 procs		4 procs		8 procs	
	Conv. T(s)	Conv. Gen.	Conv. T(s)	Conv. Gen.	Conv. T(s)	Conv. Gen.	Conv. T(s)	Conv. Gen.
2	11.1	31	1.8	44	0.7	75	0.3	134
4	19.3	33	3.4	48	0.7	75	0.3	134
8	31.6	32	6.4	49	1.1	73	0.3	134
16	48.3	31	9.2	42	2.4	91	0.5	133
32	89.3	33	16.7	46	2.9	69	0.8	134

**Table 3: Convergence Times and Number of Generations when Varying the  $n2$  Parameter**

$n2$	1 proc		2 procs		4 procs		8 procs	
	Conv. T(s)	Conv. Gen.	Conv. T(s)	Conv. Gen.	Conv. T(s)	Conv. Gen.	Conv. T(s)	Conv. Gen.
0	38.3	38	6.2	50	1.3	74	0.3	135
1	31.5	32	6.4	49	1.1	73	0.3	134
2	31.6	32	6.4	49	1.1	73	0.3	134
4	30.6	31	5.6	43	1.1	73	0.3	134
8	33.5	34	5.6	43	1.2	73	0.3	134

**Figure 1: SpeedUp of CLONALG provided through Parallelization**