Robust Asynchronous Optimization for Volunteer Computing Grids

Travis Desell*, Malik Magdon-Ismail*, Boleslaw Szymanski*, Carlos Varela*, Heidi Newberg[†] and Nathan Cole[†]

**Department of Computer Science*
Rensselaer Polytechnic Institute, Troy, New York, USA
Email: (deselt,magdon,szymansk,cvarela)@cs.rpi.edu
[†]*Department of Physics, Applied Physics and Astronomy*
Rensselaer Polytechnic Institute, Troy, New York, USA
Email: (heidi, colen2)@rpi.edu

Abstract—Volunteer computing grids offer significant computing power at relatively low cost to researchers, while at the same time generating public interest in different scientific projects. However, in order to be used effectively, their heterogeneity, volatility and restrictive computing models must be overcome. As these computing grids are open, incorrect or malicious results must also be handled. This paper examines extending the BOINC volunteer computing framework to allow for asynchronous global optimization as applied to scientific computing problems. The asynchronous optimization method used is resilient to faults and the heterogeneous nature of volunteer computing grids, while allowing scalability to tens of thousands of hosts. A work verification strategy that does not require the validation of every result is presented. This is shown to be able to effectively reduce the need for verification done to less than 30% of the reported results, without degrading the performance of the asynchronous search methods. An asynchronous version of particle swarm optimization (APSO) is presented and compared to previously used asynchronous genetic search (AGS) using the MilkyWay@Home BOINC computing project. Both search methods are shown to scale to MilkyWay@Home's current user base, over 75,000 heterogeneous and volatile hosts, something not possible for traditional optimization methods. APSO is shown to provide faster convergence to optimal results while being less sensitive to its search parameters. The verification strategy presented is shown to be effective for both AGS and APSO.

Keywords-astroinformatics; volunteer computing; asynchronous optimziation; particle swarm; genetic search;

I. INTRODUCTION

Volunteer computing grids (VCGs) can provide a massive amount of computing power at low cost to researchers. They also stimulate public interest in different scientific computing projects. They provide a democratic way for the public to participate in science, allowing them to choose what scientific projects they want to volunteer their computing resources towards. However, the public nature of this type of computing environment leads to many challenges that must be overcome in order to use it effectively. VCGs are highly heterogeneous, as many different computing architectures must be supported for wide public use, so computation and communication times vary widely. As the resources used are public and volunteered, they are highly volatile. There are no guarantees on when work may be completed and reported to the server, as work is computed at a volunteer's leisure and may never be returned if a volunteer decides to stop participating.

Additionally, malicious users may send incorrect results. VCGs also typically limit communication to client-server only. In order to overcome these issues, highly robust and asynchronous computing methods are required.

Milkyway@Home[1] uses the Berkeley Open Infrastructure for Network Computing (BOINC) to harness volunteered computing resources in creating a highly accurate three dimensional model of the Milky Way galaxy using data gathered by the Sloan Digital Sky Survey [1]. Calculating a single fit of the Milky Way model to a small wedge of observed stars can take over an hour on a high end processor. Finding the best fit of the model to one of the hundred wedges in the Milky Way can involve 50,000 or more model evaluations. Using the power of volunteer computing grids allows this optimization to be done in a reasonable amount of time.

This work examines how BOINC was modified to perform asynchronous optimization for MilkyWay@Home. Two different asynchronous global optimization methods, based on genetic search (AGS) and particle swarm optimization (APSO) are examined and tested with a challenging optimization problem. BOINCs verification scheme is improved upon for asynchronous optimization by only requiring verification of a limited subset of important results, instead of every result. APSO is shown to be more reliable than AGS, finding better models quicker while being less dependant on the input parameters to the search. Additionally, APSO is shown to perform best with a lesser amount of results requiring verification (less than 30%), while AGS requires more (less than 60%).

II. VOLUNTEER COMPUTING

Volunteer computing enables people across the world to volunteer their computing resources, such as processors, graphics processing units and hard drive space. Popular examples include SETI@Home [2], which was generalized to the Berkeley Open Infrastructure for Network Computing (BOINC) [3], IBM's World Community Grid[2] and Stanford's Folding@Home [4]. Users can even volunteer non-standard computing units such as gaming consoles like the XBOX 360 and Playstation 3. These computing frameworks not only provide thousands (even millions)

---

[1]MilkyWay@Home can be visited at http://milkyway.cs.rpi.edu
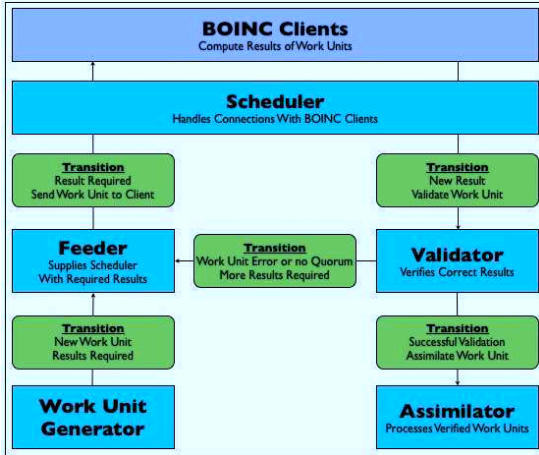[2]http://www.worldcommunitygrid.org

Figure 1. Handling of work units by the BOINC server software.

of personal computers as very powerful distributed computing networks at the low price of running a server, but generate public interest in different scientific computing projects.

However these benefits do come at a price. The computing architectures involved can be extremely different, located world-wide with dramatically different latencies and are sporadically available at the volunteers' whim. The result of this is that the computing network is highly heterogeneous in terms of computing power, architecture and latency, as well as extremely volatile as there is no guarantee that results reported by a volunteered host are correct, or when they will be returned, if ever. In addition, only client-server communication is allowed which even further limits synchronization and work sharing between volunteers.

The Milkyway@Home project uses BOINC for volunteer computing for a variety of reasons. First, BOINC currently has a large existing user base and it is easy for current users to join new projects. All a user who has already installed the BOINC client needs to do to join a new project is enter the projects website to join and start participating. Second, the code is open source and extensible, which allows easy modification to develop asynchronous optimization methods.

A. BOINC

The BOINC server side software consists of six services that handle work generation, communication with clients and result verification (see Figure 1):

- The **Transitioner** determines which work units are ready to be validated or need more results to be calculated. It will resend work to clients if a timeout has elapsed or an error occured in calculating a result. If a workunit has been assimilated, it flags that workunit as ready to have its associated files deleted.
- The **Feeder** maintains a queue of workunits that are ready to be sent to clients.

- The **Scheduler** handles incoming and outgoing communication with clients. New incoming results flag that workunit to be processed by the transitioner. This spawns CGI scripts which handle connections with clients that send work units from the ready-to-be-sent queue.
- The **Validator** is a daemon that verifies the reported results for workunits. When it determines that a certain quorum of results are the same, that workunit is ready to be handled by the assimilator.
- The **Assimilator** handles workunits that have been verified to be correct. After a workunit has been assimilated, it will not be sent to clients again.
- The **Work Generator** is either a daemon or a script which generates new workunits to be sent to clients. After work units are generated they will eventually place them in the ready to send queue.

B. Using BOINC for Asynchronous Optimization

While typical BOINC projects, such as SETI@Home, generate large amounts of workunits at a time and then later examine those results, asynchronous optimization requires new workunits to be generated in response to results reported from previous workunits. Additionally, using the limited redundancy strategy discussed in Section IV, not all workunits need to be verified, only those that could potentially improve the population.

Because validation, assimilation and work generation all depend on the population of an asynchronous search, these three services have been combined into a single `search manager`. The search manager can handle multiple simultaneous asynchronous searches. Searches run as follows:

Initialization: Each search has a maximum population size (or number of particles). While the population is not complete, random individuals are generated and their results verified before being inserted into the population.

Work Generation: After the initial population has been generated and verified, the search manager tracks the number of workunits that have been generated but are not yet being processed by a client. When the number of workunits available for update falls beneath a certain threshold, more individuals are generated either from the verification queue or by the search method (as described in Section III-C). If there are any individuals in the verification queue, the verification rate is used to determine how many workunits are generated from the verification queue.

Work Insertion: When a result is reported it is compared to the workunits in the verification queue, if it verifies one of these, that individual is removed from the queue and inserted into the population. If the individual does not verify an individual in the verification queue and could improve the population it is added to the verification queue, otherwise it is discarded.

III. GLOBAL OPTIMIZATION

Evolutionary search methods are the most common global optimization methods. Evolutionary search methods

maintain a population of parameters and their corresponding fitness. The population is used to generate new populations via different recombination heuristics to find a globally optimal fitness. These methods are typically iterative in nature, where the current population is used to generate the next population which then has fitness of its members evaluated. Unfortunately this traditional approach is not particularly suitable to volunteer computing networks due to its lack of scalability and sensitivity to faults. This work examines two popular global optimization methods, genetic search and particle swarm optimization and how they have been modified for use on MilkyWay@Home.

A. Distributed Genetic Search

In the simplest form, genetic search selects an initial population randomly in the search space and then generates subsequent populations by generating new individuals using *selection*, *crossover* and *mutation* operators on members of the previous population. Individuals represent a set of parameters within the search space. Selection simply involves taking a set of the most fit individuals from the previous population and copying them to the subsequent population. For continuous search spaces, the most common crossover operator is simply to take two parameter sets in the population and average them. Mutation takes a parameter set, select a single parameter at random from within that set and perturbs it. It is common to initially select a random point within the range of possible values, and then later decrease the range to points around the mutating parameter.

A wide range of parallel genetic algorithms (PGAs) have been examined for different distributed computing environments. Generally, parallel genetic algorithms can be classified in three different types: `cellular`, `island` or multi-population, and `panmictic` or single population [5].

Cellular GS parallelize by single individuals that communicate with a a subsection of the population (its neighbors). A processor evaluates its individual then performs crossover or mutation with thes neighbors [6], [7], [8]. This approach is well suited for homogeneous and tightly coupled (low latency) processors, such as a supercomputer, however it has also been shown to be effective in peer-to-peer environments [9].

Island GS parallelize by sets of individuals (sub-populations) which update independantly of each other, until they periodically propagate their most fit individuals to other islands. It has been shown that superlinear speedup can be attained using this method, as smaller populations can converge to minima quicker than larger populations [10], [11]. This approach is well suited to clusters and grids as inter-population communication can reduce the effect of high latency connections [12], [13].

Panmictic GS traditionally parallelizes by generating a new population and evaluating each individual simultaneously, which limits the scalability of this approach to the population size. Faults can also cause significant delays.

B. Distributed Particle Swarm Optimization

Particle swarm optimization (PSO) was initially introduced by Kennedy and Eberhart [14], [15] and is a population-based global optimization method based on biological swarm intelligence, such as bird flocking, fish schooling, etc. This approach consists of a population of particles, which `fly` through the search space based on their previous velocity, their individual best found position (cognitive intelligence) and the global best found position (social intelligence). The population of particles is updated iteratively as follows, where x is the position of the particle at iteration t, v is its velocity, p is the individual best for that particle, and g is the global best position:

$$
\begin{aligned}
v_i(t+1) &= \omega * v_i(t) \\
&\quad +c_1 * rand() * (p_i - x_i(t)) \\
&\quad +c_2 * rand() * (g_i - x_i(t)) \quad (1) \\
x_i(t+1) &= x_i(t) + v_i(t+1) \quad\quad\quad (2)
\end{aligned}
$$

Two user defined constants, c_1 and c_2, allow modification of the balance between local (cognitive) and global (social) search. Later, the inertia weight ω was added to the method by Shi and Eberhart to balance the velocity with the local and global search capability of PSO [16] and is generally used by most modern PSO implementations.

PSO has been widely modified taking inspiration from quantum mechanics [17], [18] and immune systems [19]. Adaptation of the PSO parameters [20] and population diversity [21] have also been examined.

Various strategies have been used for distributed PSO. Koh et al. and Venter et al. have examined asynchronous PSO using work stealing [22], [23]. In this strategy, the master keeps a queue of unevaluated particles that the workers retrieve and process. Xu and Zhong use an asynchronous approach where each processor is assigned a particle, and new global best positions are reported to a master processor which then broadcasts this position to the other processors [24].

C. Asynchronous Optimization with Genetic Search and Particle Swarm Optimization

For both distributed GS and PSO, the same problems remain in applying these strategies to large scale volunteer computing grids. The methods are either limited in scalability to the population size, require inter-client communication, or both. They are also highly vulnerable to faulty clients. Using asynchronous optimization as described in Section II-B results in search methods that are not dependant on any particular results, making them resilient to faults, and that can also generate work at any time, letting them scale to a very large number of clients, all while remaining within the restriction of only client-server communication. The following sections describe how genetic search and particle swarm optimization can be used within this asynchronous framework.
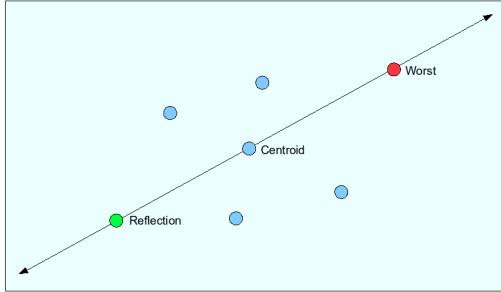
Figure 2. Simplex recombination generates a point randomly along the line created by the worst parent and the centroid (average) of the other parents.

1) Asynchronous Genetic Search: The approach used by asynchronous genetic search is similar to steady state genetic search, which instead of generating whole populations per iteration, generates single individuals using crossover and mutation and then inserts these into the population by replacing the least fit individual. Instead of generating a single individual at a time, multiple individuals are generated from the population in response to requests for work, and the population is updated when the results of these new individuals are reported. In previous work we have examined different operators for this type of asynchronous genetic search [25]. For the Milk-Way@Home application it was shown that recombination using an operator similar to the Nelder-Mead simplex algorithm provides the best results in comparison to other tested recombination operators.

The Nelder-Mead simplex search takes $N + 1$ sets of parameters, and performs *reflection*, *contraction* and *expansion* operators between the worst set of parameters and the centroid of the remaining N (see Figure 2). After calculating the centroid, a line search is performed by expanding or contracting the simplex along this line. Because in the asynchronous model it is not possible to iteratively perform expansions and contractions, a random point is selected on the line joining the worst point and its reflection. There are three parameters involved in this operator, N, the number of parents used to form the simplex (chosen randomly from the population), and two limits l_1 and l_2 which specify the range on the line over which points can be generated. For example, $l_1 = -1$ would set one limit to the reflection and $l_2 = 1$ would set the other limit to the worst point.

2) Asynchronous Particle Swarm: Asynchronous particle swarm works as follows. The search is initialized by having positions of particles generated at random with zero velocity until there has been a fitness reported for a possible position of each particle. The server keeps track of each particle's current position and current velocity, generating new positions for particles in a round-robin fashion when work is requested. As opposed to the previously discussed approaches that only process one position per particle at a time, this approach continues to generate new future positions for particles and send

them to workers. Using this approach, multiple positions for a single particle can be calculated concurrently, and the search does not need to wait for un-reported fitnesses. When a worker reports the fitness of a particle, it also reports the position and velocity of that particle. If the fitness of the reported particle is better than that particle's locally best found position, that position is updated, and the velocity of the particle is reverted to the reported velocity. If the fitness is the globally best found fitness, the position of the global best particle is updated as well.

In this way, asynchronous particle swarm performs nearly identical to traditional particle swarm when the number of processors used is less than the number of particles and there are no faults, however it can also scale to very large systems by letting workers evaluate possible future positions of a particle. For a large number of workers, the search is more exploratory, examining many possible future positions of a particle assuming the local and global best positions have not been updated. The approach is also resilient to unreported results, as more future positions of a particle are generated until one is found which improves that particle's locally best found position.

IV. LIMITED REDUNDANCY FOR ASYNCHRONOUS OPTIMIZATION

Due to the large scale, public and highly heterogeneous nature of volunteer computing systems, some kind of verification strategy is required to ensure that the results reported are correct. BOINC uses redundancy to ensure correct results. Each unit of work is sent to multiple hosts, and when a quorum reaches the same result it is verified for use by the validator. Unfortunately, even the smallest minimum quorum (two identical results) requires each work unit to be calculated at least twice, effectively reducing the amount of work that can be done by half.

In the global search methods discussed, only results which will improve the population of the genetic search, or update the local and global best particle positions need to be verified, as the other results are simply discarded. The amount of redundancy required can be significantly reduced by treating incorrect results with fitness worse than the population simply as those with bad fitness, which are discarded. By doing this only results with fitnesses that would be inserted into the population need to be verified, preventing the population from being corrupted by invalid parameter sets. For the optimization problem examined in this work, less than 30% of results were actually inserted into the population (see Section V-B). Because so few results actually need to be verified this can be a significant performance improvement.

Instead of evaluating every work unit at least twice using BOINC's built in redundancy, MilkyWay@Home uses the above strategy and a tunable redundancy rate. The search manager keeps track of results that need to be verified, and when work is generated a corresponding proportion of those workunits are copied from this list. For example, with a redundancy rate of 0.5, 50% of the

workunits generated would be for verification (assuming there are results that need to be verified). This verification rate allows the search to determine whether to devote resources to verifying potentially excellent individuals, as opposed generating more recombinations with the already known-to-be-correct population.

V. Search Analysis

Asynchronous genetic search and particle swarm were tested using MilkyWay@Home on Sagittarius Stripe 22 from the Sloan Digital Sky Survey [1]. This problem involves calculating how well a model of three streams of stars and a background function fit a 5 degree wedge of 100,789 observed stars collected along Sagittarian Longitude 55 degrees from Sagittarian Latitude 155 degrees to 230 degrees (for more information about the fitness function readers are referred to [26], [27]). In total there are 20 parameters to be optimized. This model is calculated by a wide variety of hosts. The fastest high end double precision GPUs can calculate the fitness in under two minutes. High end CPUs require around an hour, and the slowest CPUs can take days. At the time these results were gathered, MilkyWay@Home had approximately 75,000 volunteered hosts participating in the experiments.

Both asynchronous genetic search (AGS) and asynchronous particle swarm optimization (APSO) used a 50 member population for all searches. AGS was tested using 2, 4, 6, 8 and 10 parent individuals as input to the simplex recombination (see [25] for an in depth presentation of this method), while APSO was tested with inertia weights of 0.2, 0.4, 0.6, 0.8 and 1.0. As input to the simplex recombination, $l_1 = -1.5$ and $l_2 = 0.5$ were used as these have been shown to be good values for this problem [28], [25].

Verification rates of 0.3, 0.6 and 0.9 were tested across all these search parameters. For each search parameter and verification rate, five searches were performed simultaneously and exclusively by the search manager. This was done to keep the network that the searches were tested on as similar as possible.

A. Convergence

Tables I and II shows the best, average and worst fitness across these five searches after 25,000 and 50,000 reported individuals, for all combinations of verification rates and search parameters. Values in boldface represent the best individual found by the searches with the same verification rate after a certain amount of reported individuals, values in italics represent the best values found over all verification rates. Underlined values show the best fitness found over both AGS and APSO.

APSO performed better than AGS overall, finding the best fitness over all the searches and having the better average fitnesses among its searches. In generate, the APSO searches performed very well with low inertia weights (less than 0.6), and the lowest verification rate (0.3). However, APSO seemed to be more robust in that other values for the inertia weight and verification

Asynch. Particle Swarm Inserts			
v:	0.3	0.6	0.9
w = 0.2	746.6	854.2	608.8
w = 0.4	754.6	849.0	611.8
w = 0.6	691.8	773.0	484.2
w = 0.8	552.8	563.4	402.2
w = 1.0	432.6	382.6	323.0

Table III
THE NUMBER OF PARTICLES INSERTED INTO THE APSO POPULATION AFTER 50,000 RESULTS.

Asynch. Genetic Search Inserts			
v:	0.3	0.6	0.9
p = 2	1315.5	1120.2	598
p = 4	1259.6	1063.6	625.8
p = 6	1293.6	1161.2	642.4
p = 8	1269.4	1153.6	610.3
p = 10	1344.8	1178.0	611.5

Table IV
THE NUMBER OF INDIVIDUALS INSERTED INTO THE AGS POPULATION AFTER 50,000 RESULTS.

rate still provided competitive results. The various AGS searches seemed to perform competitively with the lowest verification rate for any number of parents, however higher verification rates caused significantly worse results.

B. Redundancy Overhead

Tables III and IV show the average number of individuals inserted into the populations for 50,000 reported results, for the varying verification rates and search parameters. Of all the results reported, 25% or less were inserted into the populations, meaning the verification rate of 0.3 was a good reflection of the amount of resources that should be devoted to verification. Increasing the verification rate tended to reduce the number of individuals inserted, which was to be expected as more resources were devoted to verifying results that were already reported. Interestingly, while APSO had better results than AGS, it did so inserting less individuals into its population, especially with lower verification rates. Because of this, it required less verification which could partially explain its better performance compared to AGS.

VI. Discussion

This paper examined extending the BOINC computing framework to perform asynchronous global optimization. Asynchronous optimization maintains a population of individuals representing points in the search space, and at any time uses different recombination operators to generate new individuals. Individuals are inserted into the population when their results are received from clients. In this way, asynchronous optimization is not dependent on any particular result, making it tolerant to faults, and as new work can be generated at any time it can scale to as many clients as are available.

Traditional particle swarm optimization and genetic search have been modified to work within this asynchronous optimization model. Asynchronous particle

Asynchronous Particle Swarm Convergence							
		25,000 results reported			50,000 results reported		
		best	average	worst	best	average	worst
v = 0.3	w = 0.2	***_-3.169272_***	**-3.170184**	-3.170872	-3.169037	-3.169355	**-3.169679**
	w = 0.4	-3.169807	-3.170349	-3.170816	***_-3.167358_***	***_-3.169100_***	-3.170081
	w = 0.6	-3.169887	-3.170272	**-3.170420**	-3.169128	-3.169579	-3.169930
	w = 0.8	-3.169913	-3.170747	-3.172100	-3.169243	-3.169965	-3.170665
	w = 1.0	-3.171184	-3.172528	-3.174188	-3.170732	-3.171247	-3.171778
v = 0.6	w = 0.2	-3.169545	***_-3.169812_***	-3.170252	**-3.168843**	-3.169197	-3.169487
	w = 0.4	-3.169694	-3.169936	***_-3.170110_***	-3.169073	-3.169405	-3.169650
	w = 0.6	**-3.169505**	-3.169815	-3.170221	-3.168870	**-3.169142**	***_-3.169374_***
	w = 0.8	-3.170305	-3.170817	-3.171458	-3.169647	-3.169900	-3.170217
	w = 1.0	-3.171372	-3.172507	-3.173307	-3.171011	-3.172153	-3.173089
v = 0.9	w = 0.2	-3.170161	-3.171611	-3.173086	-3.169592	-3.170143	-3.171253
	w = 0.4	**-3.169895**	**-3.170542**	**-3.171397**	**-3.169333**	**-3.169583**	**-3.170070**
	w = 0.6	-3.170182	-3.171695	-3.175647	-3.169506	-3.169807	-3.170136
	w = 0.8	-3.171230	-3.172986	-3.177449	-3.170344	-3.171061	-3.171553
	w = 1.0	-3.170701	-3.173401	-3.175725	-3.170329	-3.171741	-3.173602

Table I
FIVE ASYNCHRONOUS PARTICLE SWARMS WERE RUN FOR EACH VERIFICATION RATE (V) AND INERTIA WEIGHT (W). THIS TABLE SHOWS THE BEST, AVERAGE AND WORST OF THE BEST PARTICLES FOUND BY THESE SEARCHES AFTER 25,000 AND 50,000 RESULTS REPORTED. BEST RESULTS FOR EACH VERIFICATION RATE ARE IN BOLDFACE, BEST FOR ALL REUNDANCY RATES ARE IN ITALICS, AND THE BEST ACROSS APSO AND AGS ARE UNDERLINED.

Asynchronous Genetic Search Convergence							
		25,000 results reported			50,000 results reported		
		best	average	worst	best	average	worst
v = 0.3	p = 2	-3.171410	-3.173042	-3.174443	-3.170022	-3.170320	-3.170653
	p = 4	-3.170200	-3.171203	***-3.171817***	-3.169413	**-3.169697**	***-3.170042***
	p = 6	-3.171318	-3.171887	-3.172633	-3.170146	-3.170519	-3.171506
	p = 8	-3.170104	-3.171009	-3.172346	-3.169340	-3.169903	-3.170603
	p = 10	***-3.169374***	***-3.170770***	-3.172604	**-3.169057**	-3.169980	-3.171557
v = 0.6	p = 2	-3.171732	-3.173018	-3.174762	-3.170036	-3.170656	-3.171376
	p = 4	-3.170204	-3.171466	-3.173090	***-3.168789***	***-3.169626***	**-3.170054**
	p = 6	-3.170994	-3.173666	-3.175648	-3.169242	-3.171014	-3.172707
	p = 8	-3.170366	-3.171647	**-3.172820**	-3.169572	-3.170106	-3.170700
	p = 10	**-3.169655**	**-3.171382**	-3.173527	-3.169431	-3.170073	-3.170988
v = 0.9	p = 2	**-3.171128**	**-3.175371**	-3.181739	**-3.169240**	**-3.172969**	**-3.177777**
	p = 4	-3.174878	-3.177981	-3.183386	-3.171966	-3.174694	-3.180511
	p = 6	-3.172288	-3.178447	-3.185784	-3.170294	-3.176296	-3.184854
	p = 8	-3.173696	-3.176495	**-3.179121**	-3.172342	-3.175242	-3.178753
	p = 10	-3.182238	-3.182855	-3.183472	-3.180842	-3.181347	-3.181852

Table II
FIVE ASYNCHRONOUS GENETIC SEARCHES WERE RUN FOR EACH VERIFICATION RATE (V) AND NUMBER OF SIMPLEX PARENTS (P). THIS TABLE SHOWS THE BEST, AVERAGE AND WORST OF THE BEST INDIVIDUALS FOUND BY THESE SEARCHES AFTER 25,000 AND 50,000 RESULTS REPORTED. BEST RESULTS FOR EACH VERIFICATION RATE ARE IN BOLDFACE, BEST FOR ALL VERIFICATION RATES ARE IN ITALICS, AND THE BEST ACROSS APSO AND AGS ARE UNDERLINED.

swarm optimization (APSO) is compared to the asynchronous opimization method found to be best in previous work, asynchronous genetic search (AGS) using simplex recombination [25]. APSO is shown to provide better results, while at the same time being less sensitive to its search parameters, making it more reliable.

Further, on noting that for asynchronous search, only results that will be inserted into the population need to be verified, it was shown that the amount of redundant calculation done to provide resilience against malicious and incorrect results can be reduced significantly. Where BOINC typically verifies each result at least once, this work has shown that less than 30% redundancy can provide the best optimization rates for asynchronous particle swarm and 60% redundancy for asynchronous genetic search.

The results have also shown that how fast the asynchronous optimization reaches good solutions is dependent on this redundancy rate. It may be possible that using an adaptive redundancy rate based on how many results are waiting to be verified could improve the optimization rate further, while eliminating one of the search parameters which would lead to more reliable results. Other future work involves testing asynchronous versions of other search methods, such as differential evolution, and testing these search methods on other scientific problems.

This research shows that volunteer computing grids can be effectively and efficiently utilized for global optimization of scientific problems. By using a selective verification strategy, asynchronous optimization can significantly reduce the amount of redundant computation required by a volunteer computing grid.

REFERENCES

[1] D. G. York et al., "The sloan digital sky survey: Technical summary," *Astrophysics Journal*, vol. 120, pp. 1579–1587, Sep. 2000.

[2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[3] D. P. Anderson, E. Korpela, and R. Walton, "High-performance task distribution for volunteer computing." in *e-Science*. IEEE Computer Society, 2005, pp. 196–203.

[4] V. Pande *et al.*, "Atomistic protein folding simulations on the submillisecond timescale using worldwide distributed computing," *Biopolymers*, vol. 68, no. 1, pp. 91–109, 2002, peter Kollman Memorial Issue.

[5] E. Cantu-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux et Systems Repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[6] B. Dorronsoro, E. Alba, M. Giacobini, and M. Tomassini, "The influence of grid shape and asynchronicity on cellular evolutionary algorithms," in *IEEE Congress on Evolutionary Computation (CEC2004)*, vol. 2, June 2004, pp. 2152–2158.

[7] E. Alba and B. Dorronsoro, "The exploration/exploitation tradeoff in dynamic cellular genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 9, pp. 126–142, April 2005.

[8] B. Dorronsoro and E. Alba, "A simple cellular genetic algorithm for continuous optimization," *IEEE Congress on Evolutionary Computation (CEC2006)*, pp. 2838–2844, July 2006.

[9] G. Folino, A. Forestiero, and G. Spezzano, "A JXTA based asynchronous peer-to-peer implementation of genetic programming," *Journal of Software*, vol. 1, pp. 12–23, August 2006.

[10] E. Alba and J. M. Troya, "Analyzing synchronous and asynchronous parallel distributed genetic algorithms," *Future Generation Computer Systems*, vol. 17, pp. 451–465, January 2001.

[11] J. Berntsson and M. Tang, "A convergence model for asynchronous parallel genetic algorithms," in *IEEE Congress on Evolutionary Computation (CEC2003)*, vol. 4, December 2003, pp. 2627–2634.

[12] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, and B.-S. Lee, "Efficient hierarchical parallel genetic algorithms using grid computing," *Future Generation Computer Systems*, vol. 23, pp. 658–670, May 2007.

[13] H. Imade, R. Morishita, I. Ono, N. Ono, and M. Okamoto, "A grid-oriented genetic algorithm framework for bioinformatics," *New Generation Computing: Grid Systems for Life Sciences*, vol. 22, pp. 177–186, January 2004.

[14] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[15] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Sixth International Symposium on Micromachine and Human Science*, 1995, pp. 33–43.

[16] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *IEEE World Congress on Computational Intelligence*, May 1998, pp. 69–73.

[17] J. Sun, W. Xu, and B. Feng, "Particle swarm optimization with particles having quantum behavior," in *Congress on Evolutionary Computation*, vol. 1, June 2004, pp. 325–331.

[18] J. Liu, W. Xu, and J. Sun, "Quantum-behaved particle swarm optimization with mutation operator," in *International Conference on Tools with Artificial Intelligence*, November 2005.

[19] J. Liu and J. Sun, "Quantum-based particle swarm optimization based on immune memory and vaccination," in *IEEE International Conference on Granular Computing*, May 2006, pp. 453–456.

[20] Z. Dingxue, G. Zhihong, and L. Xinzhi, "An adaptive particle swarm optimization algorithm and simulation," in *IEEE International Conference on Automation and Logistics*, August 2007, pp. 2399–2042.

[21] J. J. Liang, A. K. Qin, P. M. Suganthan, and S. Baskar, "Particle swarm optimization with novel learning strategies," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, October 2004, pp. 3659–3664.

[22] B.-I. Koh, A. D. George, and R. T. Haftka, "Parallel asynchronous particle swarm optimization," *International Journal of Numerical Methods in Engineering*, vol. 67, no. 4, pp. 578–595, July 2006.

[23] G. Venter and J. Sobieszczanski-Sobieski, "A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations," in *Sixth World Congresses of Structural and Multidisciplinary Optimization*, May 2005, pp. 1–10.

[24] L. Xu and F. Zhang, "Parallel particle swarm optimization for attribute reduction," in *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, vol. 1, July 2007, pp. 770–775.

[25] T. Desell, B. Szymanski, and C. Varela, "An asynchronous hybrid genetic-simplex search for modeling the milky way galaxy using volunteer computing," in *Genetic and Evolutionary Computation Conference*, Atlanta, Georgia, July 2008.

[26] N. Cole, "Maximum likelihood fitting of tidal streams wit happlication to the sagittarius dwarf tidal tails," Ph.D. dissertation, Rensselaer Polytechnic Institute, 2009.

[27] N. Cole, H. Newberg, M. Magdon-Ismail, T. Desell, K. Dawsey, W. Hayashi, J. Purnell, B. Szymanski, C. A. Varela, B. Willett, and J. Wisniewski, "Maximum likelihood fitting of tidal streams with application to the sagittarius dwarf tidal tails," *Astrophysical Journal*, vol. 683, pp. 750–766, 2008.

[28] B. Szymanski, T. Desell, and C. Varela, "The effect of heterogeneity on asynchronous panmictic genetic search," in *Proc. of the Seventh International Conference on Parallel Processing and Applied Mathematics (PPAM'2007)*, ser. LNCS, Gdansk, Poland, September 2007.