Proc. 7th Int. Conference Parallel Processing and Applied Mathematics, PPAM07, Gdansk, Poland, September 2007, Lecture Notes in Computer Science, Springer, vol. 4967, Berlin, 2008, pp. 457-468.

The Effects of Heterogeneity on Asynchronous Panmictic Genetic Search

Boleslaw K. Szymanski, Travis Desell, and Carlos Varela

Department of Computer Science,
Rensselaer Polytechnic Institute, Troy NY 12180, USA,
{szymansk,deselt,cvarela}@cs.rpi.edu, http://wcl.cs.rpi.edu/

Abstract. Research scientists increasingly turn to large-scale heterogeneous environments such as computational grids and the Internet based facilities to satisfy their rapidly growing computational needs. The increasing complexity of the scientific models and rapid collection of new data are drastically outpacing the advances in processor speed while the cost of supercomputing environments remains relatively high. However, the heterogeneity and unreliability of these environments, especially the Internet, make scalable and fault tolerant search methods indispensable to effective scientific model verification. An effective search method for these types of environments is asynchronous genetic search, where a population continuously evolves based on asynchronously generated and received results. However, it is unclear what effect heterogeneity has on this type of search. For example, results received from slower workers may turn out to be obsolete or less beneficial than results calculated by faster workers. This paper examines the effect of heterogeneity on asynchronous panmictic (single population) genetic search for two different scientific applications, one used by astronomers to model the Milky Way galaxy and another by particle physicists to determine the existence of theory predicted, yet unobserved particles such as missing baryons. Results show that for both applications results received from slower workers while overall less beneficial are still useful. Additionally, a modification of asynchronous genetic search shows that different parameter generation strategies change their effectiveness over the course of the search [1].

1 Introduction

The rate of increase in CPU performance does not nearly match the rapidly increasing rates of data acquisition in all scientific disciplines. This is leading to significantly long, if not intractable, turn around times between the development of a scientific model and its verification using traditional computing environments. Testing current scientific models can involve processing terabytes of data

[1] This work has been partially supported by the following grants: NSF CAREER Award No. CNS-0448407 and NSF OISE Grant No. 0334667. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

using computationally intense modeling techniques, which results in program execution times of weeks to months on a single high end computer to calculate only a single parameter set for a scientific model. Even the best scientific model verification search methods require the evaluation of thousands of parameter sets of a scientific model over the data set. This makes using large-scale computing environments, such as computational grids and the Internet, highly desirable platforms for performing scientific model verification. The processing power these environments provide enables them to compute these models in a short amount of time, which in turn allows scientists to more quickly gather results and improve their models and understanding.

Grids and the Internet introduce additional challenges in comparison to homogeneous large-scale computing environments such as supercomputers. In addition to scalability and heterogeneity concerns, the reliability of the host nodes comes into question, especially in the case of internet computing architectures such as BOINC [4] where computing nodes can disconnect at random and for computationally significant amounts of time. Most search methods used in scientific model verification are iterative (or synchronous) in nature [7], and therefore not well suited to heterogeneous and unreliable computing environments. Additionally, it is uncertain what the effect of heterogeneity will have on asynchronous search.

A software framework for distributed scientific model evaluation and search (GMLE [8]) was extended with an asynchronous distributed evaluation framework. GMLE was used to implement an asynchronous panmictic (single population) genetic search, and a modification to it which improves the rate of convergence to a solution. Two different scientific applications, one used for astronomical modeling and the other for particle physics modeling, were used to evaluate the convergence rates of the asynchonous genetic searches. To compare the effect of heterogeneity on the search, the applications were run using Rensselaer's CCNI BlueGene as a high-performance homogeneous testing environment and the Rensselaer Grid as a heterogeneous testing environment.

Results show that the asynchronous genetic search (AGS) with continuous update converges in a half to a third of the evaluations compared to traditional iterative genetic search (IGS) on the BlueGene. Additionally, AGS on the Rensselaer Grid still converges within half the number of evalautions as needed by IGS. For a heterogeneous environment, it is shown that while fitness evaluations from slower workers are not as effective in improving the population, they still do provide benefit. This means that over large-scale heterogeneous environments, any additional processors, even slow ones, can still improve the performance of an AGS. The benefit of different types of parameter generation, in addition to the benefit of results with different calculation times, is shown to change over the course of the application. This means it may be possible to develop modifications to AGS which reduce the effect of heterogeneous calculation times. Such modification may be combined with improvements in convergence resulting from adaptively determining what parameter generation methods to use.

The paper proceeds as follows. Section 2 discusses related parallel genetic search methods and frameworks for large-scale scientific evaluation. Section 3 describes the GMLE architecturem, the asynchronous distributed evaluation framework and the genetic searches used. The different searches are evaluated in Section 4. Lastly, conclusions and future work are discussed in Section 5.

2 Related Work

A wide range of parallel genetic algorithms (PGAs) have been examined for different distributed computing environments. Generally, there are three types of parallel genetic algorithms: single population (panmictic, coarse-grained), multi-population (island, medium-grained), or cellular (fine-grained) [7]. Typically, these approaches are synchronous. Panmictic GAs create a population, evaluate it in parallel, and use the results to generate the next population. Island [3, 5] approaches evaluate local populations for a certain number of iterations, then exchange the best members with other islands. Cellular algorithms [2, 9] evaluate individual parameter sets, then update these individual sets based on the fitness of their neighbors. Hybrid approaches [14, 18] have also been examined.

P-CAGE [11] is a peer-to-peer (P2P) implementation of a hybrid multi-island genetic search built using the JXTA protocol [12] which is also designed for use over the Internet. Each individual processor (a member of the P2P network) acts as an island (a subpopulation of the whole) and evolves its subpopulation cellularly. Every few iterations, it will exchange exterior neighbors of its population with its neighbors.

There have also been different approaches taken in develping PGAs for computational grids. Imade et. al. have studied synchronous island genetic algorithms on grid computing environments for bioinformatics [13]. Lim et. al. provide a framework for distributed calculation of genetic algorithms and an extended API and meta-scheduler for resource discovery [15]. Both approaches use synchronous island-style GAs. Nimrod/O [16] is a tool that provides different optimization algorithms for use on grids and has been used to develop the EPSOC algorithm [14] which is is a mixture of a cellular and traditional GA. Populations are generated synchronously but the elimination of bad members and mutating good ones is done locally.

It has already been shown by Dorronsoro et. al. that asynchronous cellular GAs can perform competitively and discuss how update rate and different population shapes affect the convergence rate [10]. In this paper, we introduce a novel approach (to the best of our knowledge) that evaluates asynchronous panmictic GAs. This approach is well suited for both Internet and Grid computing infrastructures, because it easily facilities scalability, fault tolerance without redundancy, and does not require inter-worker communication.

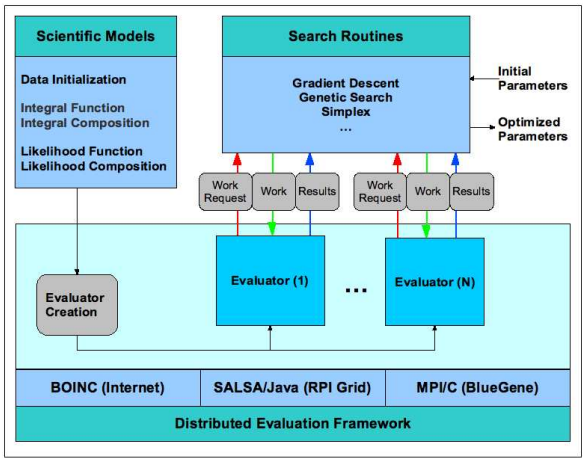

Fig. 1. GMLE with an asynchronous distributed evaluation framework.

3 Distributed Search on Heterogeneous Environments

The GMLE framework has been designed to facilitate collaboration between researchers in machine learning, distributed computing and experts with different scientific domain knowledge who are interested in distributed model verification or parameter optimization. GMLE has previously used a synchronous distributed evaluation framework for performing maximum likelihood evaluation with astronomical and particle physics applications on an IBM BlueGene supercomputer and the Rensselaer Grid [8]. The framework partitions data across a set of processors that perform partial evaluations of the model in parallel, after which the results are composed into the final result. This has been shown to be efficient for both supercomputing and grid environments, however it does not work well on highly heterogeneous and unstable environments like the BOINC infrastructure and some grids.

GMLE was extended with an asynchronous distributed evaluation framework (see Figure 1). Evaluators request work from a master, process that work and return the result, repeating as necessary. Work requests and results are all processed asynchronously by the master which peforms the different search methods. The master does not need to wait or have any dependencies on the results of the different evaluators which makes evaluator failures easily ignored and reduces the need for redundant, wasted computations. The rest of this section details the different search methods and how they are parallelized.

Iterative Genetic Search Algorithm 1 shows pseudocode for the IGS algorithm. In this algorithm, an initial population of parameter sets is generated randomly and the fitness of the model for each of those parameter sets is calculated. The iterative genetic search repeatedly calculates a new population based on the pre-

Algorithm 1: Iterative Genetic Search (IGS)

```
Data: X /*Best to keep*/, Y /*Number of Reproductions*/, Z /*Number of
      Mutations*/
Result: Converged Population
for p ∈ P[1] ... P[X+Y+Z] do p.params = random_params()
evaluate(P)
while not converged(P) do
    for p ∈ P'[1] ... P'[X] do p = P.get_next_best()
    for p ∈ P'[X+1] ... P'[X+Y] do p = reproduce(P[random()], P[random()])
    for p ∈ P'[X+Y+1] ... P'[X+Y+Z] do p = mutate(P[random()])
    P = P'
    evaluate(P)
```

Algorithm 2: Asynchronous Report Work

```
Data: P /*Population*/, max /*Maximum Population Size*/, R /*Result*/
Result: Updated Population
if P.size < max then P.insert(R)
else if R.fitness > worst(P).fitness then
    P.insert(R)
    P.remove(worst(P))
```

vious one using selection, reproduction and mutation. Selection takes the best members of the previous population and moves them to the new population. Reproduction takes two randomly selected members of the previous population and generates a new parameter set that is their average. Mutation takes a randomly selected member of the previous population and creates a new parameter set which is equal to the selected member except that one value is mutated to a new randomly selected value. In this way, iterative genetic search will converge to minima using reproduction and use mutation to prevent being stuck in a local minimum. The population size, S, is typically kept constant, so $S = X + Y + Z$, where X is the number of selections, Y is the number of reproductions, and Z is the number of mutations.

Asynchronous Genetic Search AGS is similar to IGS in that it keeps a population of parameters and generates reproductions and mutations based on it. However, instead of using a parallel model of concurrency like IGS, it uses a master-worker approach. Instead of iteratively generating new populations, new members of the population are generated when a worker requests work, and the population is updated when a worker reports work to the master. The AGS algorithm consists of two phases and uses two asynchronous message handlers (see Algorithms 2 and 3). The server can either be processing a *request work* or a *report work* message and cannot process multiple messages at the same time.

In the first phase of the algorithm (while the population size is less than the maximum population size) the server is being initialized and a random popu-

Algorithm 3: Asynchronous Request Work

```
Data: P /*Population*/, C /*Reproduction Probability*/, max /*Maximum
      Population Size*/
Result: New Parameters to Evaluate
if P.size < max then return random_params()
else
    if random() < C then
        p1 = P[random()]
        p2 = P[random()], where p1 != p2
        return reproduce(p1, p2)
    else return mutate(P[random()])
```

Algorithm 4: Double Shot Reproduce

```
Data: Member m1, Member m2
Result: Reproduced parameters
Member[] result
result[0].params = (m1.params + m2.params)/2
diff = result[0].params - m1.params
result[1].params = diff - m1.params
result[2].params = diff + m2.params
return result
```

lation is generated. When a *request work* message is processed, a random parameter set is generated, and when a *report work* message is processed, the population is updated with the parameters and the fitness of that evaluation. When enough *report work* messages have been processed, the algorithm proceeds into the second phase which actually performs the genetic search.

In the second phase, *report work* will insert the new parameters and their fitness into the population but only if they are better than the worst current member, and remove the worst member to keep the population size the same, otherwise the parameters and the fitness is discarded. Processing a *request work* message will either return a mutation or reproduction from the population.

Asynchronous Double-Shot Genetic Search The AGS algorithm was extended with the double shot method, on the observation that for the astronomy model (along with many other scientific modeling applications), the parameter space is not well formed. In this case, when a reproduction is generated from two parameter sets, they often both lie on a slope, so using the average of two points will typically would not improve the fitness. The AGS double shot (AGS-DS) algorithm improves AGS by generating three children when doing a reproduction (see Algorithm 4). One child is the average of its parents, but the other two children lie outside the parent parameters. One child is equally distant from the average outside the first parent, and the other child is equally distant from the

average outside the second parent. This allows the population to travel down gradients much faster leading to improved convergence times.

Genetic Search Distributed Evaluation There are three ways that IGS can be parallelized: (1) the fitness of each member in the population can be evaluated in parallel, (2) the fitness calculation can be done in parallel, and (3) the fitness calculation can be done in parallel as well as the population being evaluated in parallel.

The first approach can scale to a number of processors equal to the population size, while the scalability of the second approach is dependent on how much of the fitness calculation can be done in parallel. The third approach can scale to a number of processors equal to the first times the second, however it is the most complex to implement. All three approaches suffer from the scalability limitation imposed either by the population size and/or the scalability of the fitness calculation. None perform well on heterogeneous environments without intelligent partitioning. In the first case, the algorithm will only progress as fast as the slowest fitness calculation, while in the second case, the algorithm will only progress as fast as the slowest calculation of part of the fitness. The third case suffers from both, making partitioning the most difficult.

AGS and AGS-DS can be distributed in two ways: (1) workers request and report work individually and asynchronously, and (2) all workers can calculate fitness collectively based on parameters generated by the current population which are then reported, and this process repeats iteratively.

The first approach has significant benefits in heterogeneous environments because the calculation of fitness can be done by each worker concurrently and independently of each other. The algorithm progresses as fast as work is received, and faster workers can processes multiple *request work* messages, in the style of CILK's work stealing [6], without waiting on slow workers. However, the second approach can be better on homogeneous environments due to the fact that new parameter sets are always generated from the newest (and best) population.

4 Results

Test Applications The physics application uses data from particle wave analysis (PWA) to determine the existence of theory predicted, but unobserved particles (*missing baryons*) [19]. PWA observes particle states and measures their quantum spin and parity using a beam of mesons formed in an accelerator. This beam strikes a liquid hydrogen target, causing some pions interact with a proton at the target which can result in a spray of particles. After this, some particles will live long enough to create trails in a particle detector and be observed. Missing baryons decay after an extremely short time (10^{23} seconds) and do not travel a measurable distance. A scientific model with 10 to 100 fit parameters is used to calculate the occurence of missing baryons based on the observed data. The genetic search finds values for these fit parameters that most closely match the data.

The astronomy application uses data from the SLOAN digital sky survey [1], which is measuring the positions and other data about all the stars in the sky. Currently, over 10TB of data has been collected. This data is used to calculate the accuracy of 3-dimensional models of the Milky Way galaxy [17]. Any given model consists of the background (stars uniformly dispersed in the galaxy) and different streams of stars formed when other galaxies have come close to the Milky Way, were ripped apart and spread around it. The genetic search finds values for parameters describing the background and different star streams which most closely match the observed sky survey data.

Test Environments Various test environments were used to evaluate the different types of asynchronous genetic search. Rensselaer's CCNI BlueGene was used as a homogeneous high-performance test environment. A 512 node partition was used in virtual mode for a total of 1024 processors, each a 700MHz PowerPC 440 with 1GB of RAM connected by a 3-dimensional torus with 175MBps in each direction and 1.5μsec latency. The Rensselaer Grid was used as a heterogeneous test environment, consisting of four different clusters. The Solaris cluster (SOL) consists of four single core, dual processor SunBlade 1000 Sun Solaris machines, running at 800MHz. The AIX cluster (AIX) consists of four quad-processor single-core Power-PC processors running at 1.7GHz. Two Opteron clusters were also used. The first (OP1) consists of 8 quad-processor, single-core machines, and the second (OP2) consists of 2 quad-processor, dual-core machines with each core running at 2.2MHz. Inter-cluster communication is over the Rensselaer's wide-area network (WAN).

Convergence Figure 2 shows the convergence rates of the different algorithms on the BlueGene and Rensselaer Grid. The convergence rates of the IGS, AGS and AGS-DS algorithms on a homogeneous environment were tested on the CCNI BlueGene in part because of the expensive fitness calculation of the astronomy application – 5 to 25 minutes on any of the processors in the Rensselaer Grid. Similar results were obtained by the physics application. The known optimal fitness for the sample astronomy data set used was approximately 3.026. IGS had not converged to the optimum even after 50,000 evaluations, while AGS took approxmiately 30,000 evaluations and AGS-DS took 18,000 evaluations. Both AGS and AGS-DS quickly converged to a local minimum in the data set (at a fitness of approximately 3.1). AGS-DS converged faster to both minima (the local and the optimal) due to the double shot technique allowing the algorithm to travel down gradients quicker. AGS-DS was also run on the Rensselaer Grid to evaluate the effect of heterogeneity on the search. The convergence rate was not as fast, but still better than IGS, converging at at around 30,000 evaluations. Compared to the homogeneous evaluation of AGS and AGS-DS, the population had more variation over the entire execution. Again, the physics application performed similarly, with heterogeneous AGS-DS converging faster than IGS, but not as fast as AGS and AGS-DS on a homogeneous environment.

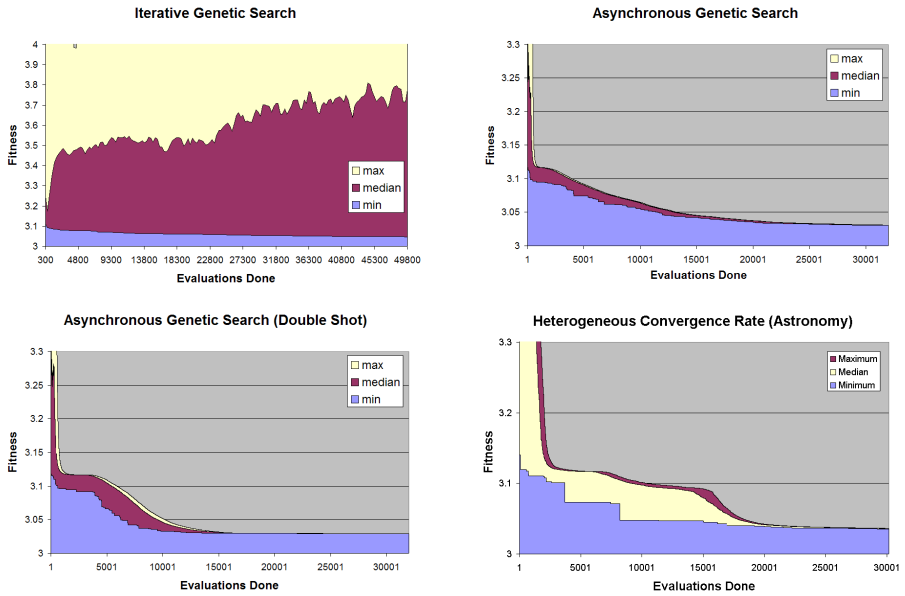

Fig. 2. Minimum, median and maximum population values for the astronomy application on the BlueGene with IGS (upper left), AGS (upper right), AGS-DS (lower left), and AGS-DS on the Rensselaer Grid (lower right).

Evaluation Utility The utility of the different evaluations performed by the search was also examined (see Figure 3). Utility is was calcualted as the number of results inserted into the population divided by the total number of results received for that speed. While results that were calculated faster had a higher chance of being inserted into the population, those with slower calculation rates still were useful. While initially, faster results tended to be much more useful than slow results, as the search began to converge, the utility of the results for all speeds decreased. Interestingly, after approximately 10,000 evaluations in the physics application, and 16,000 evaluations in the astronomy application, the utility rate started to increase again. For the physics application, the slower results gained the most benefit, while in the astronomy application the faster results improved the most. However, for both applications results of all speeds of improved their utilities. One possibility for this effect is that as the populations had both converged closely to a minima, the population was not changing as drastically, so the chance of a result being useful increased.

The utility of the different parameter generation strategies was also calculated for both applications for both applications. The benefit of mutation is initially very good and tapers off sharply, aftewards not adding much benefit. The average method of generating new parameters is the strongest of all three approaches, with the lower (the parameters generated outside the more fit parent) and higher (the parameters generated outside the less fit parent) methods

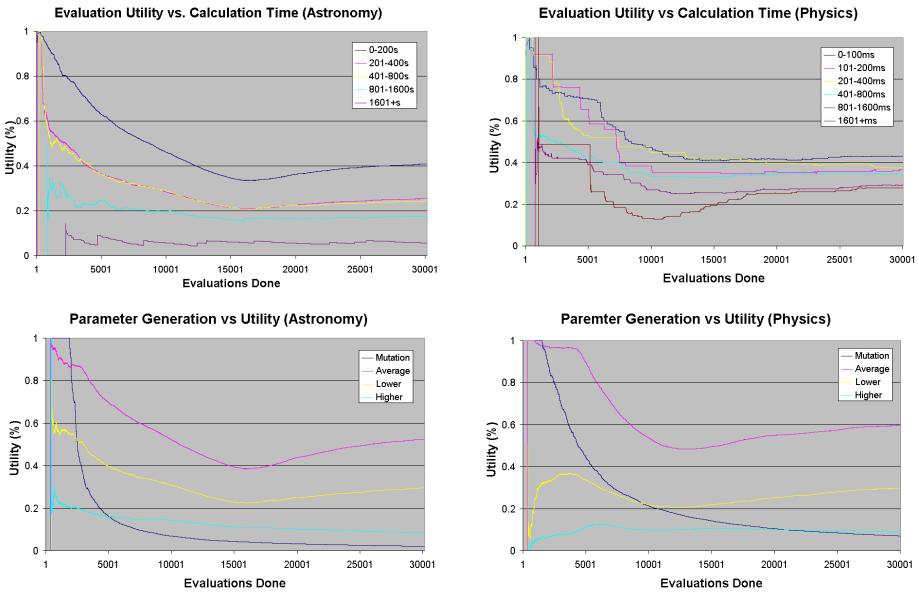

Fig. 3. Utility of results based on their calculation time for the astronomy application (upper left) and physics application (upper right), and utility of results based on how they were generated for the astronomy application (lower left) and physics application (lower right).

being less effective. For astronomy, as with calculation time, after 16,000 evaluations the average and lower methods started to improve in their ability to return beneficial results, however the improvement did level off as the search converged. Likewise, with the physics application after 10,000 evaluations average, lower and higher methods began to improve, but they also tapered off as the search converged.

5 Discussion

This paper examines two different types of asynchronous panmictic (single population) genetic search using the GMLE distributed modeling and search package. AGS is a desirable search technique for large scale and heterogeneous environments due to its inherent scalability and fault tolerance. Asynchronous genetic search (AGS) was evaluated with two different scientific applications, one used in astronomy and the other in physics. Traditional iterative genetic search (IGS) was compared to continuously updated asynchronous genetic search on a IBM BlueGene supercomputer and to asynchronous genetic search on a heterogeneous grid environment. AGS is shown to improve the convergence rate over IGS for all cases, with continuously updated AGS performing the best.

The effects of heterogeneity on AGS were also measured. Results have shown that the utility of a result, measured by its improvement of the population, is partially dependent upon how long it takes to be computed. Results which take longer to calculate are generated from older populations with less fitness, and thus have less chance to improve the current population. However, even results which are received from very slow workers still improve the population providing some benefit to the search. The utility of different types of parameter generation methods for the genetic search was also tested. Interestingly, it was shown that while the asynchronous double shot algorithm has the fastest convergence rate, the additional types of parameter generation used (higher and lower) were less likely to improve the population. For a faster convergence rate, even though the higher and lower methods of parameter generation are less likely to generate a result that will improve the population, the results generated must provide better benefit to the population when they are correct.

It was also shown that the utility of results based on calculation time and parameter generation type changes over the course of program execution. In future work, more types of parameter generation could be developed to ameliorate the effect of slowly evaluated fitnesses, reducing the impact of heterogeneity. Additionally, an adaptive search could be developed which dynamically chooses which types of parameter generation to use based on the speed of the processor and past performance.

More future work will involve extending GMLE to work with the BOINC framework. This will allow AGS to be be evaluated on a very large-scale and heterogeneous environment. Additionally, this work evaluated single population, or panmictic versions of asynchronous genetic search. As the number of available workers inscreases, asynchronous island (multi-population) genetic search will be of interest, especially if multiple servers are required to handle the load from a large BOINC community. This work shows that while heterogeneity does have a negative effect on the convergence rate of AGS, it is not excessive, even when the evaluation time of workers differs by an order of magnitude. Additionally, with different types of parameter generation strategies and an adaptive search, it may be possible to reduce the impact of heterogeneity even more – allowing AGS to be done over very heterogeneous and large-scale environments.

References

1. J. e. a. Adelman-McCarthy. The 6th Sloan Digital Sky Survey Data Release, http://www.sdss.org/dr6/, July 2007. ApJS, in press, arXiv/0707.3413.
2. E. Alba and B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 9:126–142, April 2005.
3. E. Alba and J. M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems*, 17:451–465, January 2001.
4. D. P. Anderson, E. Korpela, and R. Walton. High-performance task distribution for volunteer computing. In *e-Science*, pages 196–203. IEEE Computer Society, 2005.

5. J. Berntsson and M. Tang. A convergence model for asynchronous parallel genetic algorithms. In *IEEE Congress on Evolutionary Computation (CEC2003)*, volume 4, pages 2627–2634, December 2003.
6. R. D. Blumofe and C. E. Leiserson. Scheduling Multithreaded Computations by Work Stealing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94)*, pages 356–368, Santa Fe, New Mexico, November 1994.
7. E. Cantu-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998.
8. T. Desell, N. Cole, M. Magdon-Ismail, H. Newberg, B. Szymanski, and C. Varela. Distributed and generic maximum likelihood evaluation. In *3rd IEEE International Conference on e-Science and Grid Computing (eScience2007)*, page 8pp, Bangalore, India, December 2007. to appear.
9. B. Dorronsoro and E. Alba. A simple cellular genetic algorithm for continuous optimization. *IEEE Congress on Evolutionary Computation (CEC2006)*, pages 2838–2844, July 2006.
10. 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)*, volume 2, pages 2152–2158, June 2004.
11. G. Folino, A. Forestiero, and G. Spezzano. A JXTA based asynchronous peer-to-peer implementation of genetic programming. *Journal of Software*, 1:12–23, August 2006.
12. L. Gong. Jxta: A network programming environment. *IEEE Internet Computing*, 5:88–95, May/June 2001.
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*, 22:177–186, January 2004.
14. A. Lewis and D. Abramson. An evolutionary programming algorithm for multi-objective optimisation. In *IEEE Congress on Evolutionary Computation (CEC2003)*, volume 3, pages 1926–1932, December 2003.
15. 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*, 23:658–670, May 2007.
16. T. Peachey, D. Abramson, and A. Lewis. Model optimization and parameter estimation with Nimrod/O. In *International Conference on Computational Science*, University of Reading, UK, May 2006.
17. J. Purnell, M. Magdon-Ismail, and H. J. Newberg. A probabilistic approach to finding geometric objects in spatial datasets of the Milky Way. In *Foundations of Intelligent Systems*, volume 3488/2005, pages 485–493. Springer Berlin / Heidelberg, 2005.
18. A. Sinha and D. E. Goldberg. A survey of hybrid genetic and evolutionary algorithms. Technical Report No. 2003004, Illinois Genetic Algorithms Laboratory (IlliGAL), 2003.
19. W. Wang, K. E. Maghraoui, J. Cummings, J. Napolitano, B. Szymanski, and C. Varela. A middleware framework for maximum likelihood evaluation over dynamic grids. In *Second IEEE International Conference on e-Science and Grid Computing*, page 8 pp, Amsterdam, Netherlands, December 2006.