A Grid Portal Implementation for Genetic Mapping of Multiple QTL

Salman Toor[1], Mahen Jayawardena[1,2], Jonas Lindemann[3] and Sverker Holmgren[1]

[1] Division of Scientific Computing, Department of Information Technology, Uppsala University, Sweden
(salman.toor,sverker.holmgren)@it.uu.se
[2] University of Colombo School of Computing, Colombo, Sri Lanka
mahen@cmb.ac.lk
[3] Division of Structural Mechanics, Department of Building Sciences, Lund University, Sweden

Abstract. We present a grid portal implementation of a parallel algorithm for multidimensional QTL mapping problems. The implementation uses the LUNARC application portal framework which is modified to manage the multiple job submission used in the QTL mapping algorithm.

key words: QTL analysis, grid computing

1 Genetic mapping of QTL

Most traits of medical or economic importance are quantitative. Examples are agricultural crop yield, growth rate in farm animals and blood pressure and cholesterol levels in humans. These traits are generally believed to be governed by a complex interplay between multiple genetic factors and the environment. One method to locate the genetic regions underlying a quantitative trait is known as Quantitative Trait Locus (QTL) mapping. A QTL is a DNA region (*locus*, pl. *loci*), harboring a gene or a regulatory element affecting a quantitative trait. In a standard QTL mapping study, genetic data (*genotype* data) from an experimental population is used as input to a statistical model of the measured trait (*phenotype* data). The model fit and significance tests are performed using numerical algorithms implemented in a QTL mapping software. A review of QTL mapping methods is given in [6].

Finding the most likely positions of d QTL influencing a trait corresponds to minimization of a d-dimensional non-convex objective function (the *outer problem*) which is defined by the QTL model fit (the *inner problem*).

A popular approach for computing the model fit is the linear regression method [7,10,14], where a single least-squares problem is solved for each objective function evaluation. Efficient numerical algorithms for solving these least-squares problems in the QTL mapping setting are considered in [12, 13]. In standard QTL mapping software [2,3,11,15], the outer problem is solved using an exhaustive grid search. The computational requirement for this type of algorithm is $\mathcal{O}(d^2G^d)$, where the number of grid points G is of the order 10^3. This type of scheme is reliable but prohibitively slow for $d > 2$, which has resulted in that high-dimensional searches have so far not been used in practice. In this paper, we combine the efficient optimization scheme presented in [12] with the parallelization techniques presented in [9] and [8] and implement the resulting algorithm using the LUNARC grid portal framework.

It should be noted that already today, geneticists routinely fit models with multiple QTL. This is performed using a forward selection procedure where an identified QTL is included as a known quantity when searching for an additional QTL. In this way it is possible to search for d QTL by a sequence of d one-dimensional exhaustive grid searches. For general QTL models, it is not clear how accurate this technique is. It could be anticipated that the forward selection scheme can fail to detect QTL that only affect the phenotype through interactions with other QTL. Several analysis of real data sets have revealed such interactions between pairs of QTL, some of which were only detectable by solving the full two-dimensional optimization problem [5, 16, 17]. Such results motivate our interest for developing efficient algorithms also for high-dimensional QTL mapping problems.

In the experiments presented in this paper, we use the parallel code to search for potential QTL positions in data from an experimental intercross between European wild boars and white domestic pigs consisting of 191 individuals [1]. The pig genome has 18 chromosomes, and its total length is ~ 2300 cM[4]. In the computations we use models with d QTL, including both marginal and epistatic effects[5]. In this paper, we use this set of data and models as representative examples. We do not consider the relevance of the models used, nor do we consider the problem of which statistical model to use. Also, we do not attempt to establish the statistical significance of the results, and we do not draw any form of genetic implications from the computations. However, the code described in this paper provides a basis for future studies of all these issues, and for performing complete QTL mapping analysis using models including many QTL.

The search for the best QTL model fit should in principle be solved by optimizing over all positions x in a d-dimensional hypercube where the side is given by the size of the genome. The genome is divided into C chromosomes, re-

[4] A standard unit of genetic distance is *Morgan* [M]. However, distances are often reported in centi-Morgan [cM]

[5] Marginal effects are additive, i.e. the combined effect from two loci equals the sum of the individual effects. For epistatic effects, the relationship is nonlinear

sulting in that the search space hypercube consists of a set of C^d d-dimensional unequally sized *chromosome combination boxes*, cc-boxes. A cc-box can be identified by a vector of chromosome numbers $c = [c_1 \quad c_2 \ldots c_d]$, and consists of all x for which x_j is a point on chromosome c_j. The ordering of the loci does not affect the model fit, and this symmetry can be used to reduce the search space. We can restrict the search to cc-boxes identified by non-decreasing sequences of chromosomes. In addition, in cc-boxes where two or more edges span the same chromosome, for example $c = [1 \quad 8 \quad 8]$, we need only consider a part of the box.

Since genes on different chromosomes are unlinked, the objective function is normally discontinuous at the cc-box boundaries. This means that the QTL search could be viewed as essentially consisting of $n \approx C^d/d!$ independent global optimization problems, one for each cc-box included in the search space. This partitioning of the problem is a natural basis for a straight-forward parallelization of multi-dimensional QTL searches:

```
do (in parallel) i=1:n
  l_sol(i) = global_optimization(cc-box(i));
end
Find the global solution among l_sol(:);
```

The final (serial) operation only consists of comparing n objective function values, and the work is negligible compared to the work performed within the parallel loop. This type of parallelization was also used in [4] for mapping of single QTL.

In the grid portal implementation we use the modified version of the global optimization scheme DIRECT presented in [12]. To increase the efficiency of the parallelization technique presented above, we use the block-cyclic partitioning of the cc-boxes presented in [8]. In this case, the n independent tasks corresponding to global optimization in single cc-boxes are lumped together to m independent tasks, $m < n$, where m is a parameter that is chosen by the user.

2 Grid computing and grid portals

Grid technology promises to change the way we tackle computational problems and use data and other resources across institutional boundaries. In the last decade, a number of research projects have been started with the goal of implementing grid computing. These include Globus (one of the underlying environments for many grid software packagers), GridPP, gLite and NorduGrid/ARC. The main concept behind the grid framework is the use of in-homogenous networks, commodity class computers and clusters for performing large scale computational tasks. This is especially valuable for institutions where funding for dedicated HPC hardware cannot be obtained easily.

2.1 Application portals for grid systems

The grid middleware currently available are reliable and often used in many large projects like the LHC (Large Hadron Collider) at CERN. Still, more effort is required to make grid systems practical for the general user community. One of the major areas that limit the use of grid systems is that, infrastructure for application handling is missing. This problem becomes worse when the user needs to handle many jobs for a single application.

So far, the general approach is that researchers write shell-scripts for job submission. This provides a solution but also diverts attention from the original task. Portals such as gridsphere, CrossGrids, and GridBlocks often provide almost all the functionality of the middleware. Since these portals are not specially designed for hosting grid applications, users need good technical knowledge of object-oriented concepts and web development in order to extend the portal for their own applications.

For the QTL application presented in this paper, we use the Lunarc Application Portal (LAP). The LAP project is an effort to provide an application oriented web based environment, providing targeted user interfaces for commonly available applications. The portal currently provides user interfaces for applications, such as MATLAB, OCTAVE, ABAQUS (Structural analysis) and MOLCAS (Computational chemistry) (under development).

The portal can also be viewed as a python-based framework for implementing web interfaces to user applications. Additional user interfaces are added as plugins to an installed portal instance.

The implementation goals of the LAP framework have been:

- **Lightweight** – Easy to understand without large dependencies on other libraries. Easy to deploy and maintain.
- **Extendible** – It should be easy to extend the portal using a built in plugin-architecture.
- **Customizable** – The graphical design should be customizable to adapt to existing web designs.
- **Available** – The next release will be available under an open source license (GPL).

The portal framework is implemented using the python web application framework Webware. This is a lightweight framework for developing object-oriented web applications. The framework contains design patterns for applications servers, server pages, servlets, session management and many other features. The framework is modular and easily extendable.

The portal application server is integrated with the Apache webserver using a special Apache module, mod_webkit provided with Webware. For security reasons the Apache webserver serves the web pages using the HTTPS protocol.

Access to grid resources is implemented using the ARC middleware. Currently the interface is implemented using the client command line tools in ARC,

this approach is gradually being replaced by a direct python binding to the ARCLib library. The general architecture of the portal is illustrated in figure 1. Currently work has been initiated to separate the job management to a separate

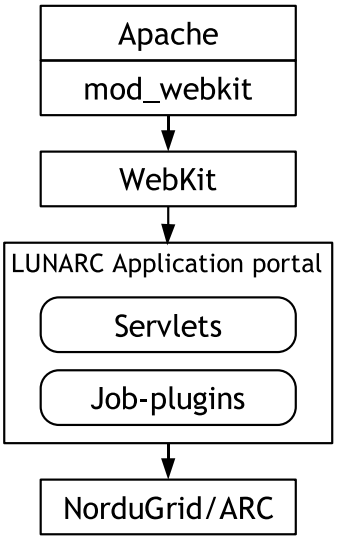

Fig. 1. General implementation architecture of the portal.

service. This service will handle job submission, monitoring and resubmission. The job service will also be able to submit jobs to other non-ARC-based grids.

3 Implementation

In the presentation below, we focus on the QTL mapping application presented above. However, it should be noted that the approach used can be generalized for many applications that have high computational requirements but can be easily subdivided into m independent tasks. The main idea of the implementation scheme is to exploit that the QTL mapping problem is subdivided into sub-tasks, and submit each sub-task as a separate grid job. Once all the sub-results have been gathered, the final search for the optimal sub-result will also be done on the portal.

During the study of building the grid portal for the QTL application we found the LUNARC grid application portal to be a very practical infrastructure for handling grid applications. The LUNARC portal provides all general features

that are the requirements of an application-specific grid portal. However, the portal was still lacking some functionality for task submission, i.e. multiple job submission and handling. The following modules in the portal needed to be extended:

1. Job submission
2. Monitoring of submitted jobs
3. Download application data
4. Kill or Clean the jobs

Job submission Submission of multiple jobs for a multi-job application is the most extensive modification. The main goal here is to be able to easily submit more than 100 jobs for one application. For this, a number of possibilities are available. **Serial approach**

1. In the simplest approach, all the job descriptions are written into one job description file. The user then attempts to submit all the jobs serially by using that single file. This approach is good for handling 3-5 jobs, but not more than that. The major drawback in this approach is that the user has to wait until all the jobs are not submitted or canceled by the system and the job submission time varies with the load on the system.
2. Divide the job descriptions into several files and try to submit each file, one at a time. In this approach, the user still cannot do anything until all the jobs have been submitted or canceled.

Parallel approach

1. An attractive approach is to use multi-threading and let the submission process be handled by a separate thread in the application. For this approach we create a job description file, put all the job descriptions into it and start the submission process by invoking a separate thread. Once all the jobs are submitted, the thread writes all the submitted job-IDs into a file named submittedJobs in the job directory.
2. The previous approach solves the problem of submitting a large number of jobs, but the whole process of job submission is still slow and gets worse as the number of jobs increases. The final approach is the combination of the two methods where we divide the multiple job descriptions into a number of description files, and the submission of each file is started by a separate thread. Once all the jobs are submitted, all the threads serially write the job-IDs into the file submittedJobs. This way we can decrease the time required for job submission and even increase the number of jobs per application. The number of threads is based on the number of job description files that are created for the application, i.e. the user-determined value m.

The last approach is good for multi-job applications but not very effective for single-job applications because the switching between threads and reading/writing into the file takes almost the same time as a serial submission. As a result, both approaches were made available in the portal. All the single jobs are handled serially, and multi-job applications are handled with threads. We tested this approach and successfully submitted more then 100 jobs of the QTL partitioning application.

Monitoring of submitted jobs The aim of the modification in this module is to provide collective handling of all the jobs related to an application. For handling multi-job applications, we defined the status of the application at each step while a job is in the system. Remember that we are dealing with the status of the applications in the portal, not of the individual jobs.

1. Under-process: When a user presses the button to submit a multi-job application, the portal starts threads for submitting all the jobs. Until the status of an applications jobs are not returned from the middleware the portal declares the status of that application under-process.
2. Submitted: All jobs related to an application are submitted successfully.
3. Running: Each of the jobs related to an application has its own ARC status. Some of them are in the queue, some of them are finished before others, so as a whole we tag the application status as Running.
4. Finished: Once all the jobs related to an application are tagged as finished from the middleware, the status of the application is also change to Finished.

Previously only one page was used for the job status, but now the status is divided into two parts. First one is the front page that shows the status of all the single jobs and the overall status of the multi-job applications. The next page shows the detailed information about the jobs relevant to that multi-job application.

Download application data This module provides the facility for multi-job applications to download all the relevant jobs by just selecting the task. This option is available from the first page of the status handling module. Once all the jobs are downloaded into the server, the user can do some basic calculations on the portal to verify the results before starting to download everything to his or her local computer. Since downloading more then 50 to 100 jobs altogether is a big task, it sometimes happens that some of the jobs are not downloaded into the server. Then users have to explicitly select the jobs and download them into the server.

Kill or Clean the jobs Currently these two options are handled by the serial execution of the kill or clean commands of ARC middleware. This section can

also handle the threads, since in multi-job applications, users have to deal with hundreds of jobs. Threads are a good solution but writing job-IDs to the file and reading from the file each time make the portal response slower. These options are modified in such a way that the multi-job applications are processed on the application level.

4 Discussion

The modifications in the LUNARC portal provides the missing functionality of the efficient application handling for multiple jobs in e.g. the QTL mapping application. We define several test cases, see Table 1:

Case No.	Jobs per task, m	Threads per task	Jobs per thread
1	100	10	10
2	100	20	5
3	300	15	20
4	300	20	15
5	500	50	10

Table 1. Test cases for parallel job submission

All the test-runs that are presented below are using ARC middleware and mostly run on the SweGrid clusters. We submitted the jobs on different intervals in a week, so that we can have a better idea about how the submission process behaves when the system is heavily loaded or relatively free. The main emphasis of this project is to build a solution that can facility the grid user in there complex tasks, without having extensive knowledge of the overall system.

Case No.	Parallel submission time (minutes)		Serial submission time
	Minimum time taken by any thread	Maximum time taken by any thread	
1	3.98	14.70	28.69
1	40.49	126.54	130.83
2	2.61	11.68	19.77
3	11.10	49.63	55.77
4	1.24	4.26	23.76
5	1.45	5.12	126.54

Table 2. Comparison of submission times

In the experiments, we used different number of threads but equal number of jobs per thread for different QTL tasks. The number of threads per task varies

on the nature of applications. Just by increasing the number of threads for the tasks does not decrease the submission time. But it is preferable that the number of threads should be greater than the number of jobs per thread, otherwise the long serial submission of each thread increases the overall submission time. The results in Table 2 show that the parallel job submission approach provides an efficient mechanism compared to the serial submission of one large XRSL file through shell scripts. In serial submission, if a job takes a long time to submit then all the jobs in the queue have to wait until the status of that job will be cleared (submitted or failed by the system). In parallel submission, if this scenario occurs then only a group of jobs have to wait but all the other threads can continue the process of submission. The results in Table 2 also shows that the submission time is heavily dependent on the overall load on the system. The Timings shown in Table 2 for parallel submission is the shortest and longest time taken by any thread during the submission of the task.

This portal gives a better solution for efficient application handling but also open some new questions to work on. A major issue is the handling of failed jobs. Consider, if 100 jobs are going to be submitted and 5% of them are fail then what will be the mechanism to handle those failed jobs. This problem becomes more complicated for those application in which some jobs are dependent on the executional results of other jobs. Currently the portal can not handle this problem, but the work is in progress and will be dealt with in the next version.

References

1. L. Andersson, C. Haley, H. Ellegren, S. Knott, M. Johansson, K. Andersson, L. Andersson-Eklund, I. Edfors-Lilja, M. Fredholm, and I. Hansson. Genetic mapping of quantitative trait loci for growth and fatness in pigs. *Science*, 263:1771–1774, 1994.
2. C. Basten, B. Weir, and Z.-B. Zeng. *QTL Cartographer, Version 1.15*. Department of Statistics, North Carolina State University, Raleigh, NC, 2001.
3. K. Broman, H. Wu, S. Sen, and G. Churchill. R/qtl: QTL mapping in experimental crosses. *Bioinformatics*, 19:889–890, 2003.
4. Ö. Carlborg, L. Andersson-Eklund, and L. Andersson. Parallel computing in regression interval mapping. *Journal of Heredity*, 92(5):449–451, 2001.
5. Ö. Carlborg, S. Kerje, K. Schütz, L. Jacobsson, P. Jensen, and L. Andersson. A global search reveals epistatic interaction between QTL for early growth in the chicken. *Genome Research*, 13:413–421, 2003.
6. R. Doerge. Mapping and analysis of quantitative trait loci in experimental populations. *Nature Reviews Genetics*, 3:43–52, 2002.
7. C. Haley and S. Knott. A simple regression method for mapping quantitative trait loci in line crosses using flanking markers. *Heredity*, 69:315–324, 1992.
8. M. Jayawardena and S. Holmgren. Parallelization of the global optimization algorithm DIRECT for qtl mapping problems. Technical report, Uppsala University, Department of Information Technology, 2007. Manuscript.

9. M. Jayawardena, K. Ljungberg, and S. Holmgren. Using parallel computing and grid systems for genetic mapping of quantitaive traits. In *Proceedings of PARA 2006*, 2006.
10. S. Knapp, W. Bridges, and D. Birkes. Mapping quantitative trait loci using molecular marker linkage maps. *Theoretical and Applied Genetics*, 79:583–592, 1990.
11. S. Lincoln, M. Daly, and E. Lander. Mapping genes controlling quantitative traits with MAPMAKER/QTL 1.1. Technical report, Whitehead Institute, 1992. Technical report. 2nd edition.
12. K. Ljungberg. Efficient evaluation of the residual sum of squares for quantitative trait locus models in the case of complete marker genotype information. Technical Report 2005-033, Division of Scientific Computing, Department of Information Technology, Uppsala University, 2005.
13. K. Ljungberg, S. Holmgren, and Ö. Carlborg. Efficient algorithms for quantitative trait loci mapping problems. *Journal of Computational Biology*, 9(6):793–804, December 2002.
14. O. Martinez and R. Curnow. Estimating the locations and the sizes of effects of quantitative trait loci using flanking markers. *Theoretical and Applied Genetics*, 85:480–488, 1992.
15. G. Seaton, C. Haley, S. Knott, M. Kearsey, and P. Visscher. QTL express: mapping quantitative trait loci in simple and complex pedigrees. *Bioinformatics*, 18:339–340, 2002.
16. K. Shimomura, S. Low-Zeddies, D. King, T. Steeves, A. Whiteley, J. Kushla, P. Zemenides, A. Lin, M. Vitaterna, G. Churchill, and J. Takahashi. Genome-wide epistatic interaction analysis reveals complex genetic determinants of circadian behavior in mice. *Genome Research*, 11:959–980, 2001.
17. F. Sugiyama, G. Churchill, D. Higgins, C. Johns, K. Makaritsis, H. Gavras, and B. Paigen. Concordance of murine quantitative trait loci for salt-induced hypertension with rat and human loci. *Genomics*, 71:70–77, 2001.