Using Parallel Computing and Grid Systems for Genetic Mapping of Multifactorial Traits

Mahen Jayawardena[1,2], Kajsa Ljungberg[1], and Sverker Holmgren[1]

[1] Uppsala University, Department of Information Technology, Uppsala, Sweden
{mahen.jayawardena,kajsa.ljungberg,sverker.holmgren}@it.uu.se
[2] University of Colombo, School of Computing, Colombo, Sri Lanka
mahen@cmb.ac.lk

Abstract. We present a flexible parallel implementation of the exhaustive grid search algorithm for multidimensional QTL mapping problems. A generic, parallel algorithm is presented and a two-level scheme is introduced for partitioning the work corresponding to the independent computational tasks in the algorithm. At the outer level, a static block-cyclic partitioning is used, and at the inner level a dynamic pool-of-tasks model is used. The implementation of the parallelism at the outer level is performed using scripts, while MPI is used at the inner level. By comparing to results from the SweGrid system to those obtained using a shared memory server, we show that this type of application is highly suitable for execution in a grid framework.

1 Genetic Mapping of Quantitative Traits

Many important traits in animals and plants are quantitative in nature. Examples include body weight and growth rate, susceptibility to infections and other diseases, and agricultural crop yield. Hence, understanding the genetic factors behind quantitative traits is of great importance. The regions in the genome affecting a quantitative trait can be found by analysis of the genetic composition of individuals in experimental populations. The genetic regions are also known as Quantitative Trait Loci (QTL), and the procedure of finding these is called QTL mapping. A review of QTL mapping methods is given in [4].

So far, standard QTL mapping software has used an exhaustive grid search for solving the global optimization problem arising from the mapping procedure. This type of algorithm is robust, but the computational requirement grows exponentially with d, the number of QTL influensing a trait. This has resulted in that often only mapping of a single QTL ($d = 1$) can be easily performed. One of the objectives of the work presented here is to perform a set high-dimensional QTL mapping computations using the (costly) exhaustive grid search. It is only for this type of algorithm that it is possible to be completely sure of that the best model fit is found. The results obtained will in the future be used to evaluate the accuracy of faster optimization algorithms. Another objective is to implement a parallel computer code that will provide a basis also for the implementation of the more efficient optimization schemes in a variety of high performance computing environments.

2 Parallelization of the Exhaustive Search for Multiple QTL

Since genes on the C different chromosomes in the organisms are unlinked, the search space can be divided into $n \approx C^d/2$ independent *cc-boxes*, each representing an independent global optimization problem. This partitioning of the problem is a natural basis for a straight-forward parallelization of multi-dimensional QTL searches. This type of parallelization was also used in [3] for mapping of single QTL. Since the objective function evaluations (model fit computations) are expensive, the work for performing global minimization in a cc-box is almost exclusively determined by the number of calls to the objective function evaluation routine. For an exhaustive grid search algorithm, this number is known a priori. However, since the size of the different cc-boxes varies a lot (the chromosomes have very different lengths), the work will be very different for different boxes. Hence, the two main issues when implementing the parallel algorithm is load-balancing and granularity for the parallel loop over cc-boxes.

3 Implementation

Our implementation of the exhaustive grid search for multi-dimensional QTL problems is based on existing serial codes written in both C and Fortran 95. These codes use the efficient objective function evaluation algorithms described in [5].

The parallel implementation uses a hybrid, two-level scheme scheme for partitioning the tasks corresponding to global optimization in different cc-boxes over a set of parallel processors. On the outer level a static partitioning of tasks is used, and on the inner level dynamic partitioning is exploited. For the outer, static level a separate code partitions the cc-boxes over p_s different jobs, which are submitted using batch scripts (in the experiments we use XRSL-scripts which specifies jobs submitted using the Nordugrid ARC grid middleware [1]. The inner, dynamic level of parallelization is implemented using MPI. Each of the p_s instances of the computational code is executed as a $p_d + 1$-process MPI job, where the partitioning of the cc-boxes over the processes is performed using a master-slave scheme[3]. The master process maintains a queue of tasks, each consisting of global optimization in a single cc-box. These tasks are dynamically handed out to the p_d worker processes as soon as they have completed their previous task.

Using a hybrid scheme of the type described above gives us the flexibility to use only static or dynamic load balancing, or a combination of the two. Also, since we have used different parallelization tools for the two levels, we can easily port the code to a variety of different computer systems and configurations.

[3] In the special case when $p_d = 1$, a single computational process is initiated and MPI is not used.

4 Results

Here, we present results for QTL searches where $d = 3$. In Table 1, we show the results for a shared memory server, using both static and dynamic partitioning of the work. In this case, the number of cc-boxes is 1140, and it is easy to balance

Table 1. Timings for $d = 3$ on the shared memory server. Static and dynamic partitioning of work.

p_s ($p_d = 1$)	Speedup	Runtime [s]	p_d ($p_s = 1$)	Speedup	Runtime [s]
1	1	535813	1	1	535730
2	1.99	269450	2	1.97	272242
4	3.94	135905	4	3.84	139439
8	7.78	68979	8	7.49	71561
16	14.97	35804	16	15.18	35287

the load for large numbers of processors. The table show that the performance of the two schemes is very similar, with a small advantage for the version using dynamic partitioning.

Next, we present results for the SweGrid system. On this systems, the experiments had to be performed under regular load conditions and using the regular scheduling policies. Some indication of the load of the SweGrid system can be given by the idle time, i.e., the time the job has to wait in queue before it is started. However, the idle time also depends on the scheduling policy of the queuing systems at the local clusters, especially if MPI-job are used. In Table 2, timings for the static and dynamic work partitioning schemes are presented. Note

Table 2. Timings for $d = 3$ on SweGrid. Static and dynamic partitioning of work.

p_s ($p_d = 1$)	Speedup	Average idle time [s]	Runtime [s]	p_d ($p_s = 1$)	Speedup	Idle time [s]	Runtime [s]
1	1	1020	955800	1	1	180	941400
2	2.03	240	471180	2	2.01	180	469140
4	3.93	360	243240	4	4.00	540	235200
8	7.56	28440	12480	8	8.02	240	117420
16	14.20	240	67320	16	8.99	181160	104760
30	24.36	600	39240	30	29.49	140760	31920
60	47.98	5460	19920	60	43.11	118440	21840

that when using the dynamic scheme, a $p_d + 1$-processor MPI-job is executed on one of the SweGrid clusters. Since the static work partitioning scheme exploits XRSL-scripts for executing multiple instances of a serial code, it is similar to schemes used in other major grid applications, e.g. by the LHC Cern project [2]. From the results in Table 2, it is also clear that the scheduling policies used in the queuing systems at the SweGrid clusters favors this type of usage of the grid. Parallel MPI-jobs often have to wait in queue a long time before they are started. To some degree, this time could be reduced by giving parallel jobs higher priorities. However, when the load on the system is high it is probably difficult to improve the situation very much.

A major aim of this work was to investigate if it possible to solve the multidimensional QTL search problem efficiently on a computational grid. The results presented above show that this is indeed the case. The speedup achieved when using the SweGrid system is very similar to when a shared memory server is used. For our implementation, the static partitioning scheme resulted in shorter turn-around times than the dynamic scheme. The reason for this is that the static version uses serial jobs which are efficiently scheduled on available processor nodes by the Nordugrid middleware, resulting in short queuing times for the jobs.

References

1. http://www.nordugrid.org.
2. http://lhc.web.cern.ch/lhc/.
3. Ö. Carlborg, L. Andersson-Eklund, and L. Andersson. Parallel computing in regression interval mapping. *Journal of Heredity*, 92(5):449–451, 2001.
4. R. Doerge. Mapping and analysis of quantitative trait loci in experimental populations. *Nature Reviews Genetics*, 3:43–52, 2002.
5. 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.