An Efficient and Scalable Implementation of SNP-Pair Interaction Testing for Genetic Association Studies

Lars Koesterke[*], Dan Stanzione, Matt Vaughn
Texas Advanced Computing Center
The University of Texas
Austin, Texas, USA
*e-mail: lars@tacc.utexas.edu

Stephen M. Welch[*], Waclaw Kusnierczyk
Department of Agronomy
Kansas State University
Manhattan, Kansas, USA
*e-mail: welchsm@ksu.edu

Jinliang Yang, Cheng-Ting Yeh, Dan Nettleton, Patrick S. Schnable[*]
Iowa State University
Ames, Iowa, USA
*email: Schnable@iastate.edu

Abstract— **This paper describes a scalable approach to one of the most computationally intensive problems in molecular plant breeding, that of associating quantitative traits with genetic markers. The fundamental problem is to build statistical correlations between particular loci in the genome of an individual plant and the expressed characteristics of that individual. While applied to plants in this paper, the problem generalizes to mapping genotypes to phenotypes across all biology. In this work, a formulation of a statistical approach for identifying pairwise interactions is presented. The implementation, optimization and parallelization of this approach are then presented, with scalability results.**

Keywords: parallel, trait, gene expression, eQTL

I. Introduction

Pressure on the world's food supply from increased population, dietary changes, loss of farmland, and anthropogenic climate change makes it essential that we radically improve the rate and efficiency at which we produce our staple food crops [1-2]. Over the course of the last century, the science of plant breeding has advanced to produce the annual increases in crop yield that have allowed the world population to burgeon while the average price of calories has declined (USDA-ERS). Yet, just as we begin to truly need even more spectacular increases in yield, the rate at which improvements are made has begun to reach a plateau - we've essentially reached the limits of traditional plant breeding [3]. If we're not successful in reversing this trend, then by sometime in the middle of this century we'll simply be unable to provide enough calories to feed the citizens of our world [4].

For the most part, plant breeding is an empirical process - it proceeds by selecting and breeding together parents that have the best versions of traits deemed desirable. This has been surprisingly successful, but we don't know (with a few exceptions) which biochemical pathways and genetic processes are being affected by the selection process. Faced with diminishing returns, we need to obtain a fine-grained understanding of the relationships between the genetic language of our crop plant species, the dynamic environment in which they grow, and their resulting complement of traits [5-6]. We need this knowledge so that we can predict the genetic codes that will, in specified environments, yield desirable traits such as increased drought tolerance, resistance to disease, or enhanced starch production. This knowledge provides invaluable guidance in selecting which parents to breed together. Conversely, we need to be able to predict the features that an individual will develop knowing little more about it than its set of genetic codes. This will enable us to predict the potential performance of the offspring that might arise from given parents.

One way of obtaining this knowledge comes in the form of 'association studies'. Broadly stated, this method exploits the natural variation that different individuals display for specific traits of interest. This information is coupled with experimental measurement of the genetic code at intervals across the genome for those same individuals and, if present, correlative relationships between the two data sets are identified. Depending on the experimental design, the density of genome locations surveyed, and the inheritance structure of the population being examined, these studies are known as Quantitative Trait Locus (QTL) mapping experiments or Genome Wide Association Studies (GWAS). A variety of statistical methods used in such studies – including maximum likelihood, Bayesian estimation, and general linear modeling [7-12] – all share a common problem in that as the analyses increase in complexity and produce more detailed data, the computational requirements to perform them rapidly exceed the resources readily available to the average breeder. Thus there is significant interest in using high-performance computing to accelerate this process [13-15, P. Bradbury pers. comm].

Even when a specific marker is found to associate with a trait of interest, it is a common situation that association only accounts for a fraction of the trait's variance. This is not surprising because many traits are 'polygenic', meaning that several genes contribute additively to the final outcome. However, even when polygenic relationships are identified, as much as 95% of the variance can remain unaccounted for by classical inheritance [16]. One popular hypothesis is that pairwise interactions between and among genes may account for much of this missing heritability [17-19].

However, searching for interactions is much more computationally intensive than single-gene analysis because all pair-wise relationships between genetic positions have to be computed and tested for statistical significance. Specifically, if one has a data set consisting of 10^6 genetic positions (which is a reasonable number given the state of the art in DNA sequence technology), he or she will need to perform not 10^6 computations but $5x10^{11}$ computations for every trait to be examined. The problem may be amplified by the need to use permutation-testing methods for identifying significance thresholds.

The nature of the algorithm and the amount of data puts the analysis well in the realm of High Performance Computing (HPC) on large clusters. Code based on scripting languages like Perl, Python, R, etc., which are widely used in Bioinformatics, are insufficient because they are very slow, and they cannot (easily) scale to thousands of compute cores. Resolving the latter problem alone would not even be sufficient. The competition for HPC compute resources is fierce and only projects that use efficient approaches and languages have a chance to be awarded substantial amounts of compute time.

We describe here the outcome of a collaboration between researchers versed in implementing highly optimized and parallelized scientific algorithms, plant molecular geneticists, and statisticians, to produce an efficient and scalable code for performing permutation tests for detecting pairwise gene interactions that will facilitate rigorous analysis of large, complex molecular breeding data sets.

II. Background

The task of identifying stretches of DNA that may contain genes controlling particular traits is critical to solving the “Genotype-to-Phenotype Problem”, declared by the National Research Council to be a top-priority problem in applied biology [20]. To date there are two major approaches. The first is direct experimentation, which involves using various technologies to generate either "shotgun" or targeted changes to the DNA, and then tracking down the location of changes that seem to affect the trait of interest. Such approaches are time-consuming and can easily take several years. The second general class of methods involves looking for statistical associations between naturally-occurring DNA differences and variation in numerical measurements of the trait of interest. Commonly, the statistical methods involve heavy use of general linear models. The precise form of the model depends on the breeding pattern that generated the particular individuals whose trait values were measured. These patterns range from the very free-form family trees that characterize human populations to very structured arrangements within plant populations that have been specifically produced for use in association studies. The range of general linear models (GLMs) varies correspondingly.

From the computational standpoint, there are also various methods used for estimating parameters of these GLMs to extract the desired genetic information. These methods can range from simple least-squares regression to the iterative procedures used for "mixed models" - that is, models that explicitly represent both random and fixed effects. There are computational commonalities that exist whichever of these methods is used. The basic idea is to use the markers as independent variables and the trait scores as dependent variables. Then the models are fit to the data using model-specific matrix methods. If a particular marker, or set of markers, is near genes that, in fact, control the trait of interest, then the expectation is that the model, when fit, will explain a statistically significant percentage of the trait variation. For markers where this does not happen, one concludes that whatever effect they may have, if any at all, is too small to be detected.

Thus, the computational task of model fitting must be repeated for as many markers or combinations of markers as are deemed relevant to the problem. As described elsewhere, this can run into many billions of model fitting operations. Although there are some differences in nomenclature, this general approach is referred to as "QTL mapping" or "association mapping". The acronym "QTL" stands for "quantitative trait locus" - that is, a place in the genome that is associated with some quantitative trait. Studies are currently underway by our group and others to find ways to accelerate these calculations, including GPU based implementations.

The approach taken in this paper has the same general objective as these standard methods - to identify genomic regions exerting control on a trait or traits of interest beyond simple additive effects. There are, however, two differences. The first is that we are specifically looking for interacting controls, i.e. pairs of markers that jointly influence the traits of interest. The second difference is that our approach does not directly involve matrix operations, but only simple differences between the mean trait values of subgroups of individuals whose marker states vary in a systematic way (section III.B.1).

III. Data, Statistical Approach, and Algorithm

A. *Input Data*

Two datasets containing Marker Scores and Trait Values from two different files are used for the calculations. The Markers file contains the genotype information. For each individual, information about which parent provided the particular DNA character at a particular location is specified. This information is gathered from DNA sequencing, microarrays, or amplification-based methods. The Trait Value file contains the measured values for particular characteristics of those same individuals. For this type of analysis, the traits in question can all be expressed as a numerical value, which is why they are called “quantitative traits”. A real-world example of a quantitative trait is plant height at developmental maturity. In both files a line starts with a RIL (Recombinant Inbred Line) designator (1st column), which is a unique string used to identify the individual from which the data were derived. Both datasets have the same number of lines (*NRIL*) and the RIL designators match. Each RIL represents one specific genotype (or "line") that results from cross-breeding of two parents with known genetic characteristics, followed by several generations of self-breeding to eliminate heterozygosity across its genome.

The first file contains Trait Values for a number of traits. Each trait is represented by one column. The trait values are only known to a few significant digits and single precision is sufficient for storage and all numerical operations. An example of the Trait Values file is given below. The first column contains the line designators. Other columns hold the measured values for the traits. Missing information is denoted NA. Columns and rows are trimmed.

```
M0001 0.2859 2.3228 26.5275 163.401 29.3054 379.593 …
M0002 0.2535 3.0007 25.0224 174.592 NA      509.122 …
M0004 0.2617 1.8441 NA      142.614 16.7467 237.905 …
 …
```

The second file contains markers with chromosomal coordinates and genotypes for each of the RILs (lines). An example of the Markers file is given below. The first column contains the line designators. "NA" is missing data. Columns and rows are trimmed.

```
M0001  1  1  1  0 NA  0 …
M0002  0  0 NA  1  1  1 …
M0004  0  0  0  0 NA  1 …
 …
```

Note that both files contain additional information, i.e., descriptive information of the traits, trait names, chromosome on which the marker resides, and the location of the marker within the chromosome measured in relative units (centiMorgans). This information is used during subsequent steps, but is not relevant for the interaction testing described in this paper..

B. Statistical Approach

It is the goal to calculate interaction values (F_{inter}) and a distribution of maximum interaction values (F_{max}) for all traits provided in the Trait Values file. We describe the calculation of F_{inter} first (section. III.B.1) and progress to the calculation of F_{max} in section. III.B.2.

1) Calculation of F_{inter}. For now we assume that no data are missing in either input file. Interaction values are defined as the interaction between a pair of markers, i.e. two columns in the marker file. All possible combinations will be examined; for N_{mark} markers, $N_{pair}=(N_{mark}\ (N_{mark}-1))/2$ pairs can be selected.

Since each potential interaction involves two markers, four different marker combinations are possible: 11, 00, 01, and 10. All summations are done with respect to these four marker combinations. The trait values are denoted by *y;* specifically y_{ij} is the score of the j^{th} individual with marker combination *i*=1..4, corresponding to 11, 00, 01, and 10, respectively.

n_i is defined as the number of individuals with the i^{th} marker combination (i.e., 11, 10, 01, or 00). If one or both markers are missing the individual is ignored. For a marker pair the interaction value (F_{inter}) is defined as follows:

$$\bar{y}_{i\bullet} = \frac{1}{n_i}\sum_{j=1}^{n_i} y_{ij}$$

$$MSE = \left(\sum_{i=1}^{4}\sum_{j=1}^{n_i} \left(y_{ij} - \bar{y}_{i\bullet} \right)^2 \right) \Bigg/ \left[\left(\sum_{i=1}^{4} n_i \right) - 4 \right]$$

$$F_{inter} = \frac{\left(\bar{y}_{1\bullet} + \bar{y}_{2\bullet} - \bar{y}_{3\bullet} - \bar{y}_{4\bullet} \right)^2}{\left(MSE \right)\left(\sum_{i=1}^{4} n_i^{-1} \right)}$$

The *MSE* numerator can be simplified as follows

$$\sum_{i=1}^{4}\sum_{j=1}^{n_i}\left(y_{ij} - \bar{y}_{i\bullet} \right)^2 = \sum_{i=1}^{4}\sum_{j=1}^{n_i}\left(y_{ij}^2 - 2y_{ij}\bar{y}_{i\bullet} + \bar{y}_{i\bullet}^2 \right)$$

$$= \sum_{i=1}^{4}\sum_{j=1}^{n_i} y_{ij}^2 - 2\sum_{i=1}^{4}\sum_{j=1}^{n_i} y_{ij}\bar{y}_{i\bullet} + \sum_{i=1}^{4}\sum_{j=1}^{n_i} \bar{y}_{i\bullet}^2$$

$$= \sum_{i=1}^{4}\sum_{j=1}^{n_i} y_{ij}^2 - 2\sum_{i=1}^{4} \bar{y}_{i\bullet}\sum_{j=1}^{n_i} y_{ij} + \sum_{i=1}^{4} n_i\bar{y}_{i\bullet}^2$$

$$= \sum_{i=1}^{4}\sum_{j=1}^{n_i} y_{ij}^2 - 2\sum_{i=1}^{4}\left(\frac{1}{n_i}\sum_{j=1}^{n_i} y_{ij} \right)\sum_{j=1}^{n_i} y_{ij} + \sum_{i=1}^{4} n_i\left(\frac{1}{n_i}\sum_{j=1}^{n_i} y_{ij} \right)^2$$

Setting $S_{1i} \equiv \sum_{j=1}^{n_i} y_{ij}$ and $S_{2i} \equiv \sum_{j=1}^{n_i} y_{ij}^2$, *MSE* simplifies to:

$$MSE = \left(\sum_{i=1}^{4} S_{2i} - 2\sum_{i=1}^{4} n_i^{-1}S_{1i}^2 + \sum_{i=1}^{4} n_i^{-1}S_{1i}^2 \right) \Bigg/ \left[\left(\sum_{i=1}^{4} n_i \right) - 4 \right]$$

$$= \left(\sum_{i=1}^{4} S_{2i} - \sum_{i=1}^{4} n_i^{-1}S_{1i}^2 \right) \Bigg/ \left[\left(\sum_{i=1}^{4} n_i \right) - 4 \right]$$

$$= \left(\sum_{i=1}^{4} S_{2i} - \sum_{i=1}^{4} S_{1i}\bar{y}_{i\bullet} \right) \Bigg/ \left[\left(\sum_{i=1}^{4} n_i \right) - 4 \right]$$

A total of N_{pair} interaction values are calculated for each trait. Once these calculations are performed for the data as observed, the y-values are randomly shuffled across the lines and the calculation is repeated. This is done many times (N_{perm}) in a Monte Carlo simulation to approximate the distribution of the F_{max} statistic (next section) under the null hypothesis of no association between the trait in the marker scores.

2) Calculation of F_{max}: F_{max} is defined as the maximum (across all marker pairs) of all interaction values (F_{inter}) for a given permutation of the data. Hence, the calculation described in sect. III.B.1 is executed 1+N_{perm} times for every trait, once with the original order of trait values for the calculation of F_{inter}, and N_{perm} times for the calculation of the F_{max} distribution. This distribution is then used to calculate the significance levels of the F_{inter} values computed for each marker pair in a manner adjusted for testing multiple dependent marker pairs within each trait [21].

3) Incomplete data: Collection of trait data is often a very labor intensive process, requiring months of field work. Sometimes, individual RILs may fail to grow or be lost to herbivory and disease, and in some cases, the trait measurement may simply be unsuccessful. Collection of genome data takes place in laboratory conditions and is generalbly reliable, but the large number of measurements may still result in missing data for specific markers in particular lines. Thus, data may be missing in either input dataset. If either the trait value (y_j) or any markers are missing, the line is omitted from all summations. If a trait value is missing the line is not considered in the permutations. Also, if any of the n_i's are zero or if the sum(n_i) <= 4, then no interaction value (F_{inter}) is calculated.

C. Algorithm

Four implementations are described: serial/naive, serial/optimized, parallel/OpenMP, and parallel/MPI. The description covers the actual calculation, but does not consider (binary) I/O.

1) Naive serial implementation: A straightforward implementation of the equations above was initially coded in serial fashion as a baseline. The pseudo-code for the serial implementation is provided below in listing 1. Note that the calculation of Finter and Fmax is independent of the trait and the itr index is dropped from the description. Hence the index itr does not appear in the pseudo-code except as the index of the outermost loop.

```
Loop over all Traits: $itr$
  Loop over 1+$N_{perm}$ permutations: $iperm$=0..$N_{perm}$
    If $iperm$ > 0: permute valid trait values ($y_j$ != "NA")
    $F_{max}$ = 0.
    Loop over $N_{pair}$ marker pairs: $imp$
      $n_i$ = 0    $i$=1..4
      $s1_i$ = 0.   $i$=1..4
      $s2$ = 0.
      Loop over all lines (RILs): $j$
        If $y_j$ = "NA": cycle
        If $marker1$ or $marker2$ = "NA": cycle
        Determine index $i$ from the two markers
        $n_i = n_i + 1$
        $s1_i = s1_i + y_j$
        $s2 = s2 + y_j * y_j$
      End loop ($j$)
      If any($n_i$) =  0 or
         sum($n_i$) <= 4:    $F_{inter}$ = "NA"; cycle
      $mean_i = s1_i / n_i$ (I=1..4)
      $contrast = mean_{i=1} + mean_{i=2} - mean_{i=3} - mean_{i=4}$
      $mse$ = s2 – sum($s1_i * mean_i$, $i$=1..4) / (sum($n_i$,i=1..4) – 4)
      $F_{inter}$ = $contrast$^2 / ($mse$ * sum(1 / $n_i$, $i$=1..4))
      If $iperm$ = 0: Output $F_{inter}$
      $F_{max}$ = max($F_{max}$, $F_{inter}$)
    End loop ($imp$)
    Output $F_{max}$
  End loop ($iperm$)
End loop ($itr$)
```

The naive serial version written in Fortran is about 1000 times faster than a similar implementation in Python.

2) Optimized serial implementation: The naive serial implementation suffers from two performance penalties. The number of floating-point operations is not minimized, but more importantly the innermost loop over the lines (RILs) cannot be vectorized because of the two if conditions that skip over invalid traits and/or marker pairs, and the indirect addressing of n_i and $s1_i$ through the index i. Without vectorization only one multiplication or addition can be executed in a SIMD instruction (Single Instruction Multiple Data) out of eight possible concurrent operations on a modern processor (single precision, four multiplications and four additions).

To enable vectorization and to increase speed, the loop order has to be changed and a vectorizable loop has to become the innermost loop. The trait loop (*itr*) and the permutations loop (*iperm*) are candidates but the loop over marker pairs is not, since it bears the indirect addressing. For the following two reasons we chose to move the permutation loop inside the RIL-loop. First, the number of loop iterations is large (at least 1000 to achieve meaningful statistics), while the number of traits may be very small or even one. Second, the exact number of permutations is not important, allowing us to choose a total number of loop iterations in the innermost loop (1+N_{perm}) that is divisible by 4, which supports effective loop vectorization.

The pseudo-code for the optimized serial implementation is given below. The core is a nested loop over RILs and permutations. The sum n_i is independent of the permutation and can be determined outside of the core. The permutation of the Trait Values is done in advance and the permuted Trait Values are stored in a separate 2D array $z_{j,iperm}$ to avoid indirect addressing. The inner loop of the core vectorizes and the number of loop iterations is divisible by 4 which is the SIMD width of x86 processors in single precision. Note that the calculation of F_{inter} and F_{max} is independent of the trait. Thus, the index *itr* does not appear in the pseudo-code.

```
Loop over all Traits: $itr$
  Loop over $N_{pair}$ marker pairs: $imp$
    Calculate $n_i$ for the marker pair under consideration
    Calculate $z_{j,iperm}$ from permutations of $y_j$

Comment – Core of nested loops: RILs and permutations
    Loop over all lines/RILs: $j$
      If $z_{j,iperm=0}$ = " NA": cycle
      If $marker1$ or $marker2$ = "NA": cycle
      Determine index $i$ from the two markers
      Loop (vectorized) over permutations: $iperm$=0..$N_{perm}$
        $s1_{i,iperm} = s1_{i,iperm} + z_{j,iperm}$
        $s2_{iperm} = s2_{iperm} + z_{j,iperm} * z_{j,iperm}$
      End loop (iperm)
    End loop (j)  Comment – End of nested core loops

    Calculate and output $F_{inter}$
    Update $F_{max}$ for all permutations
  End loop ($imp$)
  Output $F_{max}$
End loop (itr)
```

The optimized serial version is about 12 times faster than the naive serial version, which can be mainly attributed to the vectorization of the inner loop of the core.

3) Parallel implementation with OpenMP: The second loop (marker pairs: *imp*) is parallelized with OpenMP. Summation variables (n_i, $s1_i$, $s2$, etc.) are made private and an OpenMP reduction is used for the calculation of F_{max}.

```
Loop over all Traits: itr
  OpenMP: parallel do with reduction (+) of $F_{max}$
    Summation variables ($n_i$, $s1_i$, $s2$, etc.) are made private
  Loop over $N_{pair}$ marker pairs: imp
    Calculate $n_i$ for the marker pair under consideration
    Calculate $z_{j,iperm}$ from permutations of $y_j$
      Loop over all lines/RILs: j
      If $z_{j,iperm=0}$ = "NA": cycle
      If marker1 or marker2 = "NA": cycle
      Determine index i from the two markers
      Loop over 1+$N_{perm}$ permutations: iperm=0..$N_{perm}$
        $s1_{i,iperm} = s1_{i,iperm} + z_{j,iperm}$
        $s2_{iperm} = s2_{iperm} + z_{j,iperm} * z_{j,iperm}$
      End loop (iperm)
    End loop (j)

    Calculate and output $F_{inter}$
    Update $F_{max}$ for all permutations
  End loop (imp)
  omp: end parallel
  Output $F_{max}$
End loop (itr)
```

4) Parallel implementation with MPI: The outermost loop (*itr*) is parallelized with MPI. Every MPI task is assigned a range of traits (*itr_srank* and *itr_erank*) to distribute the work as evenly as possible. Note that OpenMP may be used seamlessly within the MPI implementation.

```
MPI_Init
MPI_Comm_size(…, size, …)
MPI_Comm_rank(…, rank, …)

Determine non-overlapping and balanced $itr\_s_{rank}$ and
$itr\_e_{rank}$ for every MPI task.

Loop over all Traits: $itr = itr\_s_{rank}, itr\_e_{rank}$
  omp: parallel do with reduction (+) of $F_{max}$
     Summation variables ($n_i$, $s1_i$, $s2$, etc.) are made private
  Loop over $N_{pair}$ marker pairs: imp
    Calculate $n_i$ for the marker pair under consideration
    Calculate $z_{j,iperm}$ from permutations of $y_j$
      Loop over all lines/RILs: j
      If $z_{j,iperm=0}$ = "NA": cycle
      If marker1 or marker2 = "NA": cycle
      Determine index i from the two markers
      Loop over 1+$N_{perm}$ permutations: iperm=0..$N_{perm}$
        $s1_{i,iperm} = s1_{i,iperm} + z_{j,iperm}$
        $s2_{iperm} = s2_{iperm} + z_{j,iperm} * z_{j,iperm}$
      End loop (iperm)
    End loop (j)

    Calculate and output $F_{inter}$
    Update $F_{max}$ for all permutations
  End loop (imp)
  omp: end parallel
  Output $F_{max}$
End loop (itr)
```

5) MPI version limitations & alternatives

The trait-based MPI implementation limits the number of MPI tasks to the number of traits, and therefore the amount of resources that can be utilized. However, the dataset at hand contains over 13,000 traits, which would allow for using at least 13,000 cores, and up to ~100,000 cores if combined with OpenMP on eight-core blades. However, this exceeds the number of cores in most clusters and the single-core time to analyze one trait is only a few minutes.

The situation changes with datasets that contain just a few traits (or even one) but a greatly increased number of markers. The number of marker pairs and, therefore, the compute time per trait would increase quadratically while the number of cores would be dramatically limited. To enable the use of many compute cores, the MPI parallelization would have to move from the trait-loop to the marker-pair-loop, i.e., the same loop that is parallelized with OpenMP. Changes to the code would be minimal and would mostly be limited to the use of an MPI_Reduce call for the calculation of F_{max}. Note that these changes can be easily done while keeping the OpenMP parallelization intact.

IV. PERFORMANCE AND SCALING

We introduce the experimental setup and the hardware in section IV.A and discuss the performance and scaling aspects of the serial and parallel versions in sections III.B to III.D. Figure 1 (sections III.C-D) gives scaling diagrams.

A. Experimental Setup and Hardware

We have used 2 large clusters, Ranger and Longhorn, at the Texas Advanced Computing Center (TACC) for the evaluation of the performance and the scaling. Ranger is an AMD Opteron based system (2.3GHz) with a total of 62,976 compute cores arranged in 3,936 quad-socket, quad-core blades. Longhorn has 256 dual-socket, quad-core Intel Nehalem blades with a total of 2048 cores (2.53GHz). The dataset contains 13,824 traits, 56 lines (RILs) and 1127 markers (634,501 marker pairs). The number of permutations is set to 999. To keep the run-time relatively consistent in the strong and weak scaling experiments, only a few traits are selected. The total number of operations and the amount of output scale linearly with the number of traits.

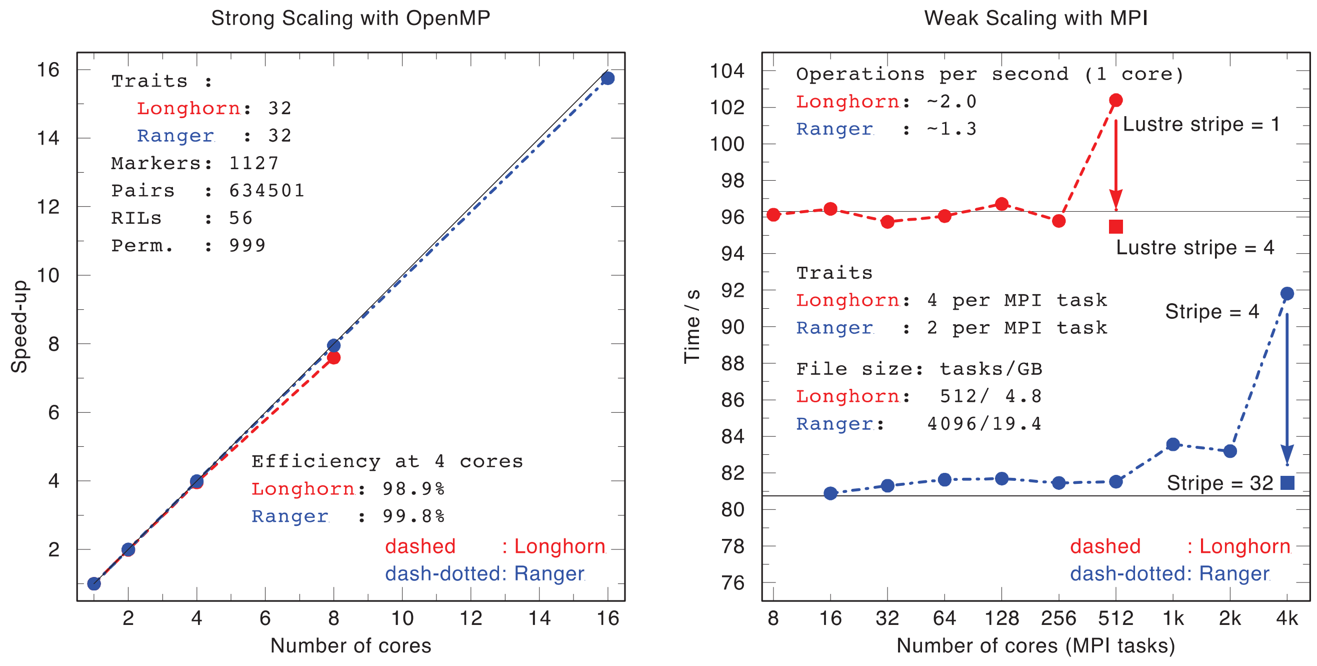

Figure 1: Strong and Weak Scaling on Longhorn and Ranger.

OpenMP achieves almost perfect scaling on one blade (left). The MPI version (right) scales perfectly to the highest core counts when the performance degradation due to I/O contention is addressed by increasing the number of Object Storage Targets (OST) used by the Lustre filesystem. Two timings are given at the highest core counts for 2 different stripe settings (default and an increase by 4 and 8, respectively). The timings with the higher stripe number are indicated by squares and the performance gain is indicated by arrows. The maximum amount of (binary) output is 4.8 and 19.4 GB, respectively, which translates to transfer rates of 248MB/s on Ranger and 51MB/s on Longhorn.

For the strong scaling (OpenMP) the number of traits is set constant, while for the weak scaling (MPI) the number of traits is selected proportional to the number of MPI tasks. Both parallelization techniques are implemented side-by-side, so that the code can be run in Hybrid mode (MPI with OpenMP, not shown).

B. *Serial Performance*

The code was written in Fortran2003 and compiled with the Intel compiler version 11 and was carefully hand-optimized for maximum performance. The optimized serial version of the code performs about 2 and 1.3 floating-point operations per clock cycle on Longhorn and Ranger, respectively. This translates to 25% and 16% of the peak performance, given that an x86 CPU has a nominal single precision peak performance of eight operations per clock cycle (four additions and four multiplications). The difference of performance on the two platforms can be traced back to the much improved memory bandwidth of the Intel Nehalem architecture. These are excellent performance numbers, particularly with an unbalanced innermost loop (two additions and one multiplication) and one division in the second innermost loop. Note that the timings include both the output and the random number generator calls.

The total number of operations needed to analyze the full dataset is ca. $N_{ops} \approx (N_{trait})(N_{marker\text{-}pairs})(21N_{perm}+3N_{perm}NRIL)$ or 1.6×10^{15} operations which translates to about 92 hours (Longhorn: 2.5GHz, 2ops/cycle of achieved performance). One trait can be analyzed in 25 seconds on Longhorn. The speedup gained by the optimization is about a factor of 12. Based on the naive serial implementation in C and Fortran and the Python implementation, the total run-time is estimated to 20-50 and 20,000 days, respectively. This demonstrates vividly that only highly optimized code is appropriate to perform the task.

C. *OpenMP Performance*

The OpenMP performance was measured in strong scaling tests on a single Ranger and Longhorn blade. The scaling is excellent. Due to the non-uniform memory architecture (NUMA), applications parallelized with OpenMP perform best when the number of threads is equal to the number of cores per socket (4 on both platforms) which minimizes memory traffic between sockets. The parallel efficiency is about 99% for 4 threads on 1 socket. The graph is shown in Figure 1.

D. *MPI Performance*

Weak scaling with MPI is excellent as well. In this test, weak scaling was achieved by reading a fixed number of traits per processor from the input file and increasing the total number of traits as the number of MPI tasks is increased. The parallel efficiency at 128 cores is above 99% on both platforms. No degradation is observed at scale. The scatter is due to the utilization of the file system by other users. No performance test has yet been performed in dedicated mode. Based on 95% parallel efficiency the whole dataset can be analyzed within one hour on 96 Longhorn cores and 192 Ranger cores, which translates to 12 blades in both cases.

With OpenMP almost perfect scaling is achieved on one blade (left panel). The MPI version (right panel) scales perfectly to the highest core counts when the performance degradation due to the large amount of output is compensated for by increasing the number of Object Storage Targets (OST) in the Lustre file system. Two timings are given at the highest core counts for 2 different stripe settings (default and an increase by 4 and 8, respectively). The timings with the higher stripe number are indicated by squares and the performance gains are indicated by arrows. The maximum amount of (binary) output is 4.8 and 19.4 GB, respectively, which translates to transfer rates of 248MB/s on Ranger and 51MB/s on Longhorn.

E. *Analysis*

As mentioned previously, much larger datasets with a vastly increased number of markers will become available soon. Assuming that the number of markers may grow to the order of 10^6, the time that it takes to analyze one trait would grow to 5,500 hours on a single compute core (Longhorn), based on the estimate $(10^6 / 1127)^2$ x 25s. Even given this likely future problem size, the scalability results indicate that 125 blades (1,000 cores) could finish the job within a couple of hours. These core counts represent only a fraction of the size of modern leadership clusters. At such speeds the analysis of large datasets with a million markers and thousands of traits is within reach.

F. *Absolute Performance*

About two floating point operations per clock cycles are achieved on one core which translates to 25% of peak in single precision. This is high compared to average code written in HPC languages (Fortran, C/C++) that execute at about 1% to 5% of peak. The algorithm is of a conveniently parallel nature and no data has to be exchanged between MPI tasks. However, the assertion that this would lead naturally to a linear speed-up is wrong. Multiple OpenMP threads and MPI tasks are competing for resources, namely for higher-level caches and memory bandwidth on a node, and for I/O bandwidth. We have carefully analyzed the behavior of several trial implementations on multiple platforms to achieve the best scaling.

There is no competing software (apart from a Python version) to which the current implementation can be compared. However, from the high serial performance (25% of peak) and the linear scaling it can be asserted that a different implementation cannot be much faster.

We have conducted early production runs (not shown) with about 13,000 traits, 10,000 permutations, 1000 markers (0.5 million pairs) and 60 RIL's. A calculation of this size takes about 4.5 hours on 128 cores, which is equivalent to 600 hours on a single core (600 SUs). This means that similar calculations can be performed on any small resource (small cluster, single multi-core workstation). Calculations that take 1,000 to 10,000 times longer (0.6 – 6 million SUs) can be facilitated through the TeraGrid which is a nation-wide resource that has provided over 1.3 billion SUs in 2010 to researchers in the US and the world.

V. CONCLUSIONS

Marker association studies are one promising approach to solving the problem of relating genotype to phenotype. This challenge is fundamental to modern biology. While potentially a very powerful tool in tackling this challenge, association studies are computationally intense, and the availability of better sequencing technologies will make datasets available that can not be analyzed by conventional means. In this paper, we have shown that for one type of association study, detecting pairwise interactions, it is feasible to take advantage of both on-node and inter-node parallelism to effectively scale this analysis. Given the size and capability of modern clusters, this puts this technique within reach of biologists.

We live in a world wherein even transient food shortages can lead to severe price hikes resulting in rioting, as has been seen within the last few years. Climate changes that result in crop performance reductions can exacerbate this effect, unfortunately in parts of the world where the social infrastructures are least able to absorb the strains. Effective scaling and application of the code developed here could simplify the task of molecular plant breeders in selecting the most effective crosses, thus helping to accelerate crop yield increases.

This paper also represents the result of a very fast interdisciplinary collaboration; the team, working exclusively by email and conference calls, produced these results in 5 weeks of work (gathering the data initially took substantially longer!). Application of this code to larger datasets is coming soon; integration of visualization techniques, and much wider dissemination will all be pursued in the coming months.

ACKNOWLEDGMENT

This work was supported by the iPlant Collaborative, National Science Foundation Award #0735191, and by the National Science Foundation's TeraGrid.

REFERENCES

[1] Khush GS. What it will take to feed 5.0 billion rice consumers in 2030. Plant Mol Biol. 2005 Sep;59:1-6.

[2] Craufurd PQ, Wheeler TR. Climate change and the flowering time of annual crops. J Exp Bot. 2009;60(9):2529-39. Review.

[3] Cooper M. and Hammer G.L. (eds): Complex traits and plant breeding—can we understand the complexities of gene-to-phenotype relationships and use such knowledge to enhance plant breeding outcomes. Aust J Agric Res. 2005;56:869–960.

[4] Long SP, Ainsworth EA, Leakey AD, Morgan PB. Global food insecurity. treatment of major food crops with elevated carbon dioxide or ozone under large-scale fully open-air conditions suggests recent models may have overestimated future yields. Philos Trans R Soc Lond B Biol Sci. 2005 Nov 29;360(1463):2011-20. Review.

[5] Hammer G, Cooper M, Tardieu F, Welch S, Walsh B, van Eeuwijk F, Chapman S, Podlich D. Models for navigating biological complexity in breeding improved crop plants. Trends Plant Sci. 2006 Dec;11(12):587-93. Epub 2006 Nov 7. Review.

[6] Mittler R, Blumwald E. Genetic engineering for modern agriculture: challenges and perspectives. Annu Rev Plant Biol. 2010 Jun 2;61:443-62. Review.

[7] Bradbury PJ, Zhang Z, Kroon DE, Casstevens TM, Ramdoss Y, Buckler ES. TASSEL: software for association mapping of complex

traits in diverse samples. Bioinformatics. 2007 Oct 1;23(19):2633-5. Epub 2007 Jun 22.

[8] Cooper M, van Eeuwijk FA, Hammer GL, Podlich DW, Messina C. Modeling QTL for complex traits: detection and context for plant breeding. Curr Opin Plant Biol. 2009 Apr;12(2):231-40. Epub 2009 Mar 11.

[9] Sham PC, Cherny SS, Purcell S. Application of genome-wide SNP data for uncovering pairwise relationships and quantitative trait loci. Genetica. 2009 Jun;136(2):237-43. Epub 2009 Jan 7. Review

[10] van Eeuwijk FA, Bink MC, Chenu K, Chapman SC. Detection and use of QTL for complex traits in multiple environments. Curr Opin Plant Biol. 2010 Apr;13(2):193-205. Review.

[11] Arends D, Prins P, Jansen RC, Broman KW. R/qtl: high-throughput multiple QTL mapping. Bioinformatics. 2010 Dec 1;26(23):2990-2. Epub 2010 Oct 21.

[12] Da Costa e Silva L, Zeng ZB. Current progress on statistical methods for mapping quantitative trait loci from inbred line crosses. J Biopharm Stat. 2010 Mar;20(2):454-81. Review.

[13] Ma L, Runesha HB, Dvorkin D, Garbe JR, Da Y. Parallel and serial computing tools for testing single-locus and epistatic SNP effects of quantitative traits in genome-wide association studies. BMC Bioinformatics. 2008 Jul 21;9:315.

[14] Peng T, Du P, Li Y. PBEAM: a parallel implementation of BEAM for genome-wide inference of epistatic interactions. Bioinformation. 2009 Apr 21;3(8):349-51.

[15] Schüpbach T, Xenarios I, Bergmann S, Kapur K. FastEpistasis: a high performance computing solution for quantitative trait epistasis. Bioinformatics. 2010 Jun 1;26(11):1468-9. Epub 2010 Apr 7.

[16] Manolio TA, Collins FS, Cox NJ, Goldstein DB, Hindorff LA, Hunter DJ, McCarthy MI, Ramos EM, Cardon LR, Chakravarti A, Cho JH, Guttmacher AE, Kong A, Kruglyak L, Mardis E, Rotimi CN, Slatkin M, Valle D, Whittemore AS, Boehnke M, Clark AG, Eichler EE, Gibson G, Haines JL, Mackay TF, McCarroll SA, Visscher PM. Finding the missing heritability of complex diseases. Nature. 2009 Oct 8;461(7265):747-53. Review.

[17] Li Z, Pinson SR, Park WD, Paterson AH, Stansel JW. Epistasis for three grain yield components in rice (Oryza sativa L.). Genetics. 1997 Feb;145(2):453-65.

[18] Holland JB. Genetic architecture of complex traits in plants. Curr Opin Plant Biol. 2007 Apr;10(2):156-61. Epub 2007 Feb 8. Review.

[19] Maher B. Personal genomes: The case of the missing heritability. Nature. 2008 Nov 6;456(7218):18-21.

[20] National Research Council. Achievements of the National Plant Genome Initiative and New Horizons in Plant Biology. The National Academies Press, Washington, D.C. 2008.

[21] Churchill and Doerge. Empirical threshold values for quantitative trait mapping. Genetics 138:963-971. 1994.