SweeD: Likelihood-based detection of selective sweeps in thousands of genomes

Pavlos Pavlidis[1*], Daniel Živković[2], Alexandros Stamatakis[1], Nikolaos Alachiotis[1]
[1]The Exelixis Lab, Scientific Computing Group, Heidelberg Insitute for Theoretical Studies (HITS gGmbH), Schloss-Wolfsbrunnenweg 35, D-69118, Heidelberg, Germany
[2]Section of Evolutionary Biology, Biocenter, University of Munich, D-82152 Planegg-Martinsried, Germany

***Corresponding author:**

Pavlos Pavlidis
pavlidisp@gmail.com
The Exelixis Lab, Scientific Computing Group,
Heidelberg Insitute for Theoretical Studies (HITS gGmbH),
69118 Heidelberg
Germany

Keywords: selective sweep, positive selection, adaptive evolution, high-performance computing, site frequency spectrum

Nonstandard abbreviations: CLR, SFS, VCF, BFGS, DMTCP, MPFR

Running Title: SweeD: Sweep Detector

Abstract

The advent of modern DNA sequencing technology is the driving force in obtaining complete intra-specific genomes that can be used to detect loci that have been subject to positive selection in the recent past. Based on selective sweep theory, beneficial loci can be detected by examining the SNP patterns in intra-specific genome alignments. In the last decade, numerous algorithms have been developed to identify selective sweeps. However, the majority of these algorithms has not been designed for analyzing whole-genome data.

We present SweeD (Sweep Detector), an open-source tool for the rapid detection of selective sweeps in whole genomes. It analyzes site frequency spectra and represents an extension of the widely-used SweepFinder program.

The sequential version of SweeD is up to 22 times faster than SweepFinder and, more importantly, is able to analyze thousands of sequences. We also provide a parallel multi-core implementation of SweeD. Furthermore, we implemented a checkpointing mechanism that allows to also deploy SweeD on cluster systems with queue execution time restrictions, as well as to resume long-running analyses after processor failures. Finally, the user can specify a demographic model via the command-line to calculate the theoretically expected site frequency spectrum of a demographic model. Therefore, (in contrast to SweepFinder) the neutral site frequencies can optionally be directly estimated from a demographic model.

Introduction

The seminal paper by Maynard Smith and Haigh (1974) coined the term “genetic hitchhiking”, that is, the evolutionary process where a strongly beneficial mutation emerges and spreads in a population. As a consequence, the frequency of linked neutral or weakly selected variants will increase. The authors showed that, in sufficiently large populations, the hitchhiking effect drastically reduces genetic variation near the positively selected site, thereby inducing a so-called selective sweep. According to their deterministic model, diversity vanishes at the selected site immediately after the fixation of the beneficial allele. The model also predicts that with increasing distance (scaled by $\alpha = r/s \log(2N)$, where r is the recombination rate, s is the selection coefficient, and N is the effective population size) from the selected site i) diversity accumulates, ii) the distribution of the frequencies of segregating sites changes, and iii) linkage-disequilibrium patterns are generated around the target site of the beneficial mutation.

Neutral mutations are assumed to arise in a sufficiently large population at a rate of $\theta/2$, ($\theta = 4N\mu$, μ being the mutation probability per site and per generation). Initially, they are present as a single copy. Thus, according to the infinitely-many sites model (Kimura 1969), they occur at previously monomorphic sites. The site frequency spectrum (SFS) of a population denotes the distribution of the expected number of polymorphic sites, $\phi(x)dx$, at which the mutant allele has a frequency in $(x, x+dx)$, $0 < x < 1$. Kimura (1971) demonstrated that the SFS for the standard neutral model is given by $\phi(x)dx = \theta/x\, dx$. For the selective sweep model, Fay and Wu (2000) have shown that the frequency spectrum of neutral sites which are sufficiently close to the beneficial mutation shifts toward an excess of high- and low-frequency derived alleles in proportions $x\phi(x)dx = \theta\, dx$ and $(\theta/x - \theta)dx$, respectively. While the aforementioned neutral and selective models assume a constant population size, analytical results for the SFS have also been obtained for scenarios in which the population is subject to deterministic size changes (Griffiths 2003). However, deriving an analytical approximation of site frequency spectrum when sites are subject to genetic hitchhiking (in populations with varying size over time) still remains a challenge.

Regarding analyses of DNA sequence samples, the sample SFS (and not the population SFS) is of interest. The sample SFS, $f_{n,i}$, is the distribution of the expected number of sites at which there are *i* derived alleles, $1 \leq i \leq n-1$, in a sample of *n* sequences. The relative frequencies are obtained from these absolute frequencies via division by the total number of segregating sites. If the mutant allele can not be distinguished from the wild type, the folded version of the SFS is used. Kim and Stephan (2002) interpreted $f_{n,i}$ as the probability of observing a single site where i derived alleles are found in a sample of size *n*. The authors used the derivation of the SFS by Fay and Wu (2000) to develop the first composite likelihood ratio test (CLR) for detecting selective sweeps in typically small (up to a few hundred kilobases) genomic regions (henceforth called subgenomic regions). Nielsen et al. (2005) introduced two major modifications to the CLR method by Kim and Stephan (2002) for detecting selective sweeps in whole-genome data.

First, instead of using the model by Fay and Wu (2000), that relies on the population mutation parameter θ, Nielsen *et al.* (2005) proposed a model that quantifies the frequency of an allele at a distance *d* from the beneficial mutation independently of θ by conditioning on the observation of a SNP. Second, instead of employing the theoretical result for the SFS (Kimura 1971) that assumes standard neutrality as done by Kim and Stephan (2002), Nielsen *et al.* (2005) use the empirical SFS of the entire dataset as neutral background. The first modification allows for applying the test to large-scale genome data, where θ can vary among regions. The second modification increases the robustness of the algorithm under demographic models (e.g., mild bottlenecks). It implicitly accounts for this, by using the empirical SFS that is obtained from the entire genome. Nielsen *et al.* (2005)

implemented their method in SweepFinder (http://people.binf.ku.dk/rasmus/webpage/sf.html). In the numerator of the CLR test, SweepFinder calculates the likelihood of a sweep at a certain position in the genome by maximizing α. The denominator (the neutral model) is given by the product of the empirical SFS over all SNPs. Since SNPs are assumed to be independent, the overall likelihood for the genetic hitchhiking model is calculated as product over the per-SNP likelihood scores .

With next generation sequencing technologies it has now become feasible to sequence whole genomes of thousands of individuals from a single species and to reliably detect the genomic locations of selective sweeps. Selective sweep prediction accuracy increases with the number of sequence samples. For instance, Jensen *et al.* (2007) showed that distinguishing selective sweeps from demographic events in samples of moderate size (50 samples) is easier than in smaller samples (12 samples). Nowadays, samples that comprise hundreds or even thousands (e.g., The 1000 Genomes Project Consortium 2012 https://1000genomes.org) of sequences are becoming available. Hence, selective sweep detection is expected to become more accurate. However, the increase in sample sizes and sequence lengths poses novel algorithmic, numerical, and computational challenges for selective sweep detection. Numerically stable implementations that can handle arithmetic over- and/or underflow are required. An efficient use of scarce computing and memory resources is also required. Furthermore, efficient parallel implementations are needed to analyze large datasets in reasonable times on state-of-the-art multi- and many-core processors.

At present only a handful of tools that scale to thousands of whole-genome sequences is available. The implementation of the CLR test by Kim and Stephan (2002) can only be used for analyzing small subgenomic regions. Jensen *et al.* (2007) and Pavlidis *et al.* (2010) used the ω-statistic (Kim and Nielsen 2004), which relies on the linkage-disequilibrium signature of a selective sweep to detect positively selected sites. The respective implementations are also only able to handle subgenomic regions. SweepFinder (Nielsen et al. 2005) can analyze whole genomes efficiently, but only for up to a few hundred sequences. For larger sample sizes, execution times increase substantially. Moreover, SweepFinder can not analyze samples exceeding 1,027 sequences because numerical problems associated to floating point underflow are not handled. Finally, SweepFinder only runs on a single core. To the best of our knowledge, the ω-statistic based OmegaPlus tool (Alachiotis et al. 2012) represents the sole publicly available high-performance implementation for detecting selective sweeps. OmegaPlus can efficiently analyze whole genomes from thousands of individuals by exploiting all available cores on a modern desktop or server.

New approaches

In the following, we describe SweeD (Sweep Detector), our open-source tool for the SFS-based rapid detection of selective sweeps at the whole-genome scale. The SweeD code is based on SweepFinder (Nielsen et al. 2005) and incorporates the following new features and algorithmic techniques: Via respective program parameters the SFS can be calculated analytically for demographic models that comprise an arbitrary number of instantaneous population size changes and, optionally, also an exponential growth as the most recent event. Thereby, a neutral SFS can be obtained without the need to compute the empirical average SFS for the genome.

Moreover, SweeD can analyze thousands of genomes because we adapted the numerical implementation of the arithmetic operations. For a large number of genomes, the double precision floating-point range is frequently not sufficient. This may lead to numerical over- or underflow. SweeD is able to analyze such large samples because it performs several calculations at the logarithmic scale.

The code also supports several additional input file formats for reading in simulated and real datasets.

Regarding real datasets, it supports the FASTA and VCF formats. The VCF format is widely used in next generation sequencing projects, such as, for instance, the 1000 Genomes project (http://www.1000genomes.org). With respect to simulated datasets, SweeD supports ms (Hudson 2002) and MaCS (Chen 2009) formats.

Furthermore, SweeD can exploit all available cores on a shared-memory multi-core processor to substantially expedite the analysis of huge datasets that comprise millions of SNPs and thousands of sequences.

Finally, SweeD offers a checkpointing capability that allows to restart (continue/resume) an analysis from the point where it failed, rather than running it again from scratch. This mechanism allows for saving CPU time and energy in the case of hardware failures or cluster queues with time limits.

Results and Discussion

In the following, we present a performance comparison between SweeD and SweepFinder, assess the efficiency of the parallel implementation, and provide a usage example.

Sequential Performance

For comparing the performance of SweeD versus SweepFinder, we generated simulated datasets with up to 1,000 sequences and 1,000,000 sites using msms (Ewing and Hermisson 2010). We slightly modified the source code of msms to obtain output files that can be parsed by SweepFinder (the modified version of msms is available at: http:/exelixis-lab.org/software.html). We generated datasets with and without selection. The programs were executed on an unloaded AMD Opteron 6174 processor with 12 cores running at 2.2 GHz under Ubuntu Linux.

As shown in Table 1 SweeD outperforms SweepFinder on all datasets. The total execution times for both programs increase with the number of sequences *and* the number of SNPs. Run-times are dominated by two computationally expensive parts in both programs: i) the pre-computation of a fixed number of likelihood values at given distances (in scaled units) around the position of the selective sweep, and ii) the computation of the CLR test at those positions as specified by the user via the `-grid` option. To precompute the likelihood values at certain distances around the position of the selective sweep, SweeD carries out the arithmetic operations in a different order than SweepFinder. SweeD employs a lookup table to store these intermediate results that can be reused for the precomputation of the constant, fixed likelihood values. In contrast, SweepFinder recalculates these intermediate constant values-on-the-fly. The performance benefit of using a lookup table can be observed when the number of sequences is increased, because the number of lookups (redundant recalculations in SweepFinder) is proportional to the number of sequences. For small numbers of sequences, lookups and recalculations need approximately the same time. As the number of sequences increases, the lookup-based approach outperforms the recalculation approach. SweeD and SweepFinder employ the same approach to compute the CLR test at a specific position. However, we optimized the CLR computation in SweeD via low-level technical optimizations. Nonetheless, the computation of the CLR test as such is only marginally faster in SweeD.

Table 1 also shows that, for a small number of sequences, SweeD becomes faster than SweepFinder as the number of SNPs increases . This is because the order and the number of operations at each position, where the CLR is calculated, is different in SweeD (see section **Arithmetic deviations from SweepFinder** for more details). We obtained speedups between 1.07X and 3.90X. For larger numbers of sequences (1,000), the speedup of SweeD over SweepFinder drops from 22X (10,000 SNPs) to 2.9X (1,000,000 SNPs) with an increasing number of SNPs because a larger fraction of overall execution time is spent for CLR computations.

Due to the aforementioned lookup table, SweeD requires more memory than SweepFinder. Figure 1 shows the peak memory consumption for SweeD and SweepFinder as a function of the number of sequences, when a dataset of 100 SNPs is analyzed (using the SF data format). For this specific dataset, SweeD consumes about 4.6 times more memory than SweepFinder. Nonetheless, the memory requirements increase linearly for both programs. Despite the larger memory footprint of SweeD, the additional memory for storing the lookup table is negligible with respect to the memory capacity of modern computers. For instance, storing a lookup table for a dataset with 10,000 sequences requires approximately 24 MB. Thus, the analysis of very large population genetics datasets is feasible. SweeD uses the same suite of parsers as OmegaPlus for ms, MaCS, VCF, and FASTA files. Since the parser suite is not yet fully optimized for memory efficiency, SweeD may exhibit temporary (during parsing and conversion into the internal SF data format) memory consumption peaks (depending on the input format), which exceed the amount of memory required for the actual computations.

Parallel Performance

To assess the parallel efficiency of SweeD, we generated datasets with up to 10,000 sequences and 1,000,000 sites. Figure 2 shows the respective speedups for up to 48 cores/threads (4 AMD Opteron 6174 processors) on simulated datasets with 100 and 10,000 sequences, and 10,000, 100,000, and 1,000,000 SNPs, respectively. The execution times for the sequential analysis of the dataset with 100 sequences are shown in Table 1. The datasets with 10,000 sequences as well as 10,000, 100,000, and 1,000,000 SNPs required 30,717, 32,299, and 37,212 seconds, respectively.

As can be observed in Figure 2A, the parallel implementation scales well with the number of cores, achieving speedups between 41X and 45X on 48 cores for the small sample of 100 sequences. In contrast, Figure 2B shows speedups that only range between 7X and 37X for the large sample of

10,000 sequences on 48 cores. This is due to the small amount of SNPs for the comparatively large number of sequences, which in turn leads to a significantly larger amount of time spent in the BFGS (Broyden-Fletcher-Goldfarb-Shanno, Fletcher 1987) algorithm that optimizes the neutral SFS. Specifically, the BFGS algorithm estimates the neutral SFS that maximizes the probability of the dataset (i.e., the overall likelihood) given the input SFS and the data. This step is needed because the input dataset may contain missing data, and thus the input SFS does not correspond precisely to the sample SFS. These likelihood computations have been parallelized. However, when the number of SNPs is small compared to the number of sequences, substantially more iterations (and hence thread synchronization events) are required for the BFGS algorithm to converge. This step cannot be further parallelized because the iterative optimization procedure uses the likelihood values sequentially, that is, there exists a hard-to-resove sequential dependency between iterations *i* and *i+1*.

For example, when we analyze the dataset with 10,000 sequences and 10,000 SNPs, the BFGS algorithm computes the likelihood of the input dataset conditional on the SFS 4,477,114 times, whereas only 396 such likelihood calculations are required for the dataset with 100 sequences and 10,000 SNPs.

The parallel efficiency of each iteration improves with an increasing number of SNPs because more computations are carried out per iteration/synchronization inbetween synchronization events. Therefore, for 10,000 SNPs and 10,000 sequences we observe the worst-case speedup of 7 due to an unfavorable combination of relatively few SNPs (low workload per iteration) and a large number of such parallel iterations (4,477,114). For the same sample size, but with 1,000,000 instead of 10,000 SNPs, the parallel efficiency improves and we obtain good speedups (37X).

Since a parallel implementation of SweepFinder is not available as a reference, we report on OmegaPlus performance as a rough reference. Compared to OmegaPlus, SweeD exhibits better parallel efficiency, since it scales well up to 48 cores in most cases. Parallel OmegaPlus only scales up to 12 cores (Alachiotis et al. 2012a). Note however that, for a single core or a small number of cores (up to

12 in our tests), OmegaPlus outperforms SweeD due to algorithmic innovations and because it mostly relyies on integer rather than on floating-point arithmetics.

Usage Example

To demonstrate the capability of SweeD to handle real-world genomic data, we downloaded and analyzed the chromosome 1 dataset from the 1000 Genome Project (http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase1/analysis_results/integrated_call_sets/). This dataset contains the genetic variation from 1092 humans, that is, the sample size is 2184. The size of the input file is 87 GB, and it comprises 2,896,960 SNPs. We carried out the analysis on an Intel Core i7-2600 processor with 4 cores (8 threads with hyperthreading) running at 3.4 GHz. We calculated the CLR test at 100,000 points (gridsize), and the SFS was obtained from the entire dataset. The total execution time was 8 hours and 15 minutes. In contrast to SweeD, SweepFinder fails to analyze this dataset because of the large sample size (see section **Arithmetic deviations from SweepFinder**). We also analyzed this dataset with OmegaPlus (command line flags: maxwin=280,000, minwin=1,000; see manual for further details on the OmegaPlus command line). OmegaPlus was faster than SweeD (total execution time: 2 hours and, 37 minutes). The OmegaPlus and SweeD output results are illustrated in Figure 3.

Conclusions and future work

SweeD is an improved and scalable implementation of SweepFinder that allows for analyzing thousands of genomes. In contrast to SweepFinder, SweeD can also analytically calculate the SFS based on a user-specified demographic model. It can also parse several common input file formats such as, ms, MaCS, FASTA, and VCF. Furthermore, SweeD leverages the computational power of multi-core systems, shows good speedups, and thereby substantially decreases the time-to-solution. Finally, a checkpointing mechanism allows to resume analyses from where they were interrupted in the case of hardware failures or queue limitations, leading to time and energy savings.

Regarding future work, we plan to parallelize the calculations of the theoretical SFS and employ an out-of-core (external memory algorithm) approach to make the calculations of the theoretical SFS feasible on off-the-shelf computers. Finally, we intend to evaluate the accuracy of scalable sweep-detection tools such as SweeD and OmegaPlus as a function of increasing sample size.

Materials and Methods

The SFS of samples for deterministically varying population size

Analytical results for sample frequency spectra can either be directly derived via the coalescent or be obtained via binomial sampling from the population version as derived within the diffusion framework. This is also the case for a neutral model of a population whose size varies over time. Here $\rho(t)=N(t)/N$ denotes the ratio between the ancestral and the current population size at time *t*. Changes in population size can be included into the standard neutral model as the harmonic mean of the relative population sizes via time-rescaling $t\rightarrow\int_0^t 1/\rho(s)\,ds$. Griffiths and Tavaré (1998) established the SFS within the coalescent framework, and Živković and Stephan (2011) found an equivalent solution based on diffusion theory (Evans et al. 2007) as

$$f_{n,i}=\frac{\theta}{i}\sum_{k=2}^{n}(-1)^k(2\mathrm{k}-1)\binom{k}{2}{}_3F_2(n-i+1,k,1-k\,;n+1,2\,;1)\int_0^\infty \exp\left(-\binom{k}{2}\int_0^t 1/\rho(s)\,ds\right)dt\quad,$$

where ${}_3F_2(a,b,c\,;d,e\,;z)=\sum_{l\geq 0}(a_{(l)}b_{(l)}c_{(l)})/(d_{(l)}e_{(l)})z^l/l\,!$ is a generalized hypergeometric function, in which $p_{(0)}=1$ and $p_{(l)}=p(p+1)\ldots(p+l-1),\ l\geq 1$. For the standard neutral model, this equation reduces to $f_{n,i} = \frac{\theta}{i}$. The relative frequency spectrum is obtained via division by the total number of segregating sites. The equation for the SFS can be applied to demographic models including various instantaneous size changes and multiple phases of exponential growth. It can also be used to calculate the composite likelihood of all considered sites of a dataset based on a given demographic model and in analogy to Kim and Stephan (2002).

Implementation

SweeD is implemented in C and has been developed and tested on Linux platforms. The parallel SweeD version uses Posix threads (Pthreads). The checkpointing procedure relies on the DMTCP (Distributed MultiThreaded CheckPointing, Ansel et al. 2009) library.

Optional computation of the SFS for a given demographic model

A new feature of SweeD that is not available in SweepFinder is the calculation of the theoretical sample SFS for a user-specified demographic model. The model can comprise an arbitrary number of instantaneous population size changes and, optionally, an exponential growth as the most recent event. For the calculation of the theoretical sample SFS, numerical issues can arise for samples exceeding 60 sequences. To solve recurrent issues with numerical precision that are related to the harmonic sum representation of the SFS, we used the MPFR (Multiple-Precision Floating-point library with correct Rounding, Fousse et al. 2007) library. The MPFR library can be used to conduct arbitrary precision floating-point operations where required. Using arbitrary precision arithmetics, however, leads to increased run times and memory requirements for the analytical computation of the SFS compared to double precision floating point arithmetics. Although the run time differences are negligible for small sample sizes (up to approximately 50 sequences), computing times can increase substantially (up to 5 times in Figure 4B) with the number of sequences. We employ a lookup table to alleviate this performance issue by avoiding frequent re-computations of these values. This approach reduces run times by a factor that is approximately equal to the number of sequences. However, the size of the lookup table also increases quadratically with the number of sequences and may induce excessive memory requirements (Figure 4A).

However, the implementation of the theoretical sample SFS is useful since it does not require using additional programs. For instance, one could use ms (Hudson 2002) to simulate thousands of samples (typically > 10,000) and then compute the average SFS using some ad hoc implementation.. Furthermore, the option to calculate the theoretical sample SFS is useful when a representative average genome SFS is not available (e.g., when sub-genomic regions are analyzed).

Parallelization

Multi-core systems can run several threads of execution in parallel which can decrease run times of an application. However, substantial changes to the sequential code may be required to obtain an efficient parallel algorithm. Therefore, we focused on parallelizing the most compute-intensive parts of SweeD. As already described, SweeD computes the likelihood and optimizes the α-parameter of the CLR test at several positions of the alignment. Since the CLR calculations at different positions (CLR positions) are independent, they are equally divided among the available cores. However, there is load imbalance among CLR computations because the inference of α-parameters at CLR positions that are located close to a selected site requires a larger amount of arithmetic operations. When a CLR position is located near a positively selected site, the α-parameter value that maximizes the likelihood of the sweep model is smaller (α is inversely proportional to the selection coefficient). However, the size of a genomic region that a selective sweep may affect is inversely proportional to α. Thus, more SNPs are required to compute α, when the α value decreases. Therefore, we distribute CLR positions in a cyclic way to cores such as to improve load balance. We plan to test whether more elaborate load balancing schemes, such as dynamic scheduling or guided scheduling can further improve load balance.

Arithmetic deviations from SweepFinder

Since SweeD mainly represents a re-engineered version of SweepFinder, one would expect to obtain *exactly* the same output from both programs, when the same input data is analyzed. However, both SweeD and SweepFinder, heavily rely on floating-point arithmetics, which are not associative. In other words the following equality does not hold under floating-point arithmetics: A + (B + C) = (A + B) + C. Therefore, the order of floating point operations affects the final result. For each CLR position both SweeD and SweepFinder compute the probability of each SNP (under the sweep and the neutral model) in a certain region around the CLR position. To calculate these probabilities, SweepFinder moves from left to right along the genome, whereas SweeD moves from the CLR position toward the boundaries of the region. Consequently, the order of operations is different. Therefore, slight numerical deviations between the respective results are to be expected.

There are two additional factors that contribute to the numeric differences between SweeD and SweepFinder. First, logarithmic operations are required in SweeD to ensure scalability for a large number (thousands) of sequences. To avoid arithmetic underflow as frequently observed in SweepFinder, several multiplications are implemented as sums of logarithms in SweeD. When the number of sequences is large, the operands in these multiplications approach the lower limit of the double-precision floating-point range, which can result in floating-point underflows. This is the main reason why SweepFinder cannot analyze datasets that comprise more than 1,027 sequences and exits with a failing assertion: “SweepFinder: SweepFinder.c:365: get_pstar: Assertion `sum <= 1.0 && sum

> 0.0' failed".

Second, SweeD implements a linear instead of a cubic spline interpolation. Both SweepFinder and SweeD calculate the probability *P(b)* of observing a SNP with a frequency *b* at *k* fixed distances *d* (as scaled by α). For all other values of α*d*, *P(b)* is calculated by interpolating the probability values of the *k* fixed distances. SweepFinder uses *k* := 60 in conjunction with a cubic spline interpolation. We observed that the spline function calculates erroneous values for *k* := 60. By increasing the value of *k*, we found that, using a linear interpolation between distance points is sufficiently accurate to calculate *P(b)*. Thus, we use *k:=300* and a linear instead of a cubic spline in SweeD.

Checkpoint and restart capability

Due to the typical time limitations imposed by job submission queues on cluster systems, a checkpointing and restart capability represents an important feature of scientific codes. In typical cluster installations, job queues have 24 or 48 hour time limits. A job submitted to a 24-hour queue is killed immediately, if it takes longer, effectively wasting the energy spent during the past 24 hours, since the user will have to resubmit the job to a queue with a higher time limit, say 48 hours. However, if the application is checkpointed, the user can resume the job from the point, where its execution was interrupted to achieve time and energy savings.

SweeD uses the open-source checkpointing library DMTCP (Ansel et al. 2009) for this purpose. With the respective makefiles (with the file extension .CHECKPOINTS), users can compile the checkpointable version of SweeD: SweeD-C. Note that the non-checkpointable version does not require the DMTCP library and is hence easier to compile and install. The checkpointable version takes one additional input parameter, the checkpointing interval, which defines how often checkpoints are created and stored during the execution of SweeD-C. To enable checkpointing, the dmtcp_coordinator process has to be started before executing SweeD-C. Subsequently, the program can be invoked as usual (with the additional parameter for the checkpointing interval). When an unexpected event such as a queue time-out or an electricity or processor failure interrupts the execution of the program, the user will be able to resume the execution by using the restart script provided with the DMTCP library.

Command line arguments and output files

SweeD is a command line tool and requires at least three parameters for a typical analysis: i) a name for the run (*-name*), ii) the name of the input file (*-input*), and iii) the number of CLR positions (*-grid*). In addition to the input file format of SweepFinder **(see SweeD manual Section at http://exelixis-lab.org/software.html).**

In the following we provide a few example command line invocations:

i) SweeD -name test -input file.sf -grid 10000

ii) SweeD-P -name test -input file.sf -grid 10000 -threads 4

iii) SweeD-C -name test -input file.sf -grid 10000 -checkpoint 1200

In the first example, SweeD is called with the minimum number of parameters to compute the CLR at 10,000 positions along the dataset as provided in file.sf. In the second example, the parallel version of SweeD is called. Hence we need an additional parameter to specify the number of cores/threads that shall be used. In the last example, we start the checkpointable version. This requires the additional parameter that specifies how frequently (in seconds) checkpoints should be stored. For more examples and a detailed description of all supported command line parameters please refer to the manual **(http://exelixis-lab.org/software.html).**

SweeD generates two output files: i) an information file that provides information regarding the dataset (number of sequences, sites, etc.) and the analysis (e.g., execution time), and ii) a report file that contains the likelihood value and α-parameter for each CLR position. Finally, a warning file might be written, when ms or MaCS input file formats are used to report possible conflicting SNP positions, that is, SNPs that refer to the same alignment site.

References

Alachiotis N, Pavlidis P, Stamatakis A. 2012. Exploiting Multi-grain Parallelism for Efficient Selective Sweep Detection. In: Algorithms and Architecture for Parallel Processing (ICA3PP). Vol. 7439. p. 56–68.

Alachiotis N, Stamatakis A, Pavlidis P. 2012. OmegaPlus: A Scalable Tool for Rapid Detection of Selective Sweeps in Whole-Genome Datasets. Bioinformatics (Oxford, England) 28:2274–2275.

Ansel J, Arya K, Cooperman G. 2009. DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop. In: 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS’09). p. 1–12.

Chen GK, Marjoram P, Wall JD. 2009. Fast and flexible simulation of DNA sequence data. Genome research 19:136–42.

Evans SN, Shvets Y, Slatkin M. 2007. Non-equilibrium theory of the allele frequency spectrum. Theoretical population biology 71:109–19.

Ewing G, Hermisson J. 2010. MSMS: a coalescent simulation program including recombination, demographic structure and selection at a single locus. Bioinformatics (Oxford, England) 26:2064–5.

Fay JC, Wu CI. 2000. Hitchhiking under positive Darwinian selection. Genetics 155:1405–13.

Fletcher R. 1987. *Practical methods of optimization*. New York: John Wiley & Sons, Ltd

Fousse L, Hanrot G, Lefevre V, Pélissier P, Zimmermann P. 2007. MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Transactions on Mathematical Software 33:1–15.

Griffiths RC, Tavaré S. 1998. The age of a mutation in a general coalescent tree. Communications in Statistics. Stochastic Models 14:273–295.

Griffiths RC. 2003. The frequency spectrum of a mutation, and its age, in a general diffusion model. Theoretical population biology 64:241–51.

Hudson RR. 2002. Generating samples under a Wright-Fisher neutral model of genetic variation. Bioinformatics (Oxford, England) 18:337–8.

Jensen JD, Thornton KR, Bustamante CD, Aquadro CF. 2007. On the utility of linkage disequilibrium as a statistic for identifying targets of positive selection in nonequilibrium populations. Genetics 176:2371–9.

Kim Y, Nielsen R. 2004. Linkage disequilibrium as a signature of selective sweeps. Genetics 167:1513–1524.

Kim Y, Stephan W. 2002. Detecting a local signature of genetic hitchhiking along a recombining chromosome. Genetics 160:765–777.

Kimura M. 1969. The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations. Genetics:893–903.

Kimura M. 1971. Theoretical foundation of population genetics at the molecular level. Theoretical population biology 2:174–208.

Maynard Smith J, Haigh J. 1974. The hitch-hiking effect of a favourable gene. Genetical research 23:23–35.

Nielsen R, Williamson S, Kim Y, Hubisz MJ, Clark AG, Bustamante C. 2005. Genomic scans for selective sweeps using SNP data. Genome research 15:1566–75.

Pavlidis P, Jensen JD, Stephan W. 2010. Searching for footprints of positive selection in whole-genome SNP data from nonequilibrium populations. Genetics 185:907–22.

Seward J, Nethercote N. 2005. Using Valgrind to detect undefined value errors with bit-precision. In: Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC ’05). p. 2--2

The 1000 Genomes Project Consortium. 2012. An integrated map of genetic variation from 1,092 human genomes. Nature 491:56–65.

Živković D, Stephan W. 2011. Analytical results on the neutral non-equilibrium allele frequency spectrum based on diffusion theory. Theoretical population biology 79:184–91.

Tables

Table 1: Total execution times and speedups for simulated datasets with and without selection.

		SweepFinder		SweeD		Speedup	
Sequences	**SNPs**	**Neutral**	**Selection**	**Neutral**	**Selection**	**Neutral**	**Selection**
50	**10,000**	199.908	434.744	142.200	399.440	1.406	1.088
50	**100,000**	2005.075	4380.188	1085.240	3563.890	1.848	1.229
50	**1,000,000**	34563.920	52560.680	8881.410	32466.250	3.892	1.619
100	**10,000**	207.123	427.885	142.650	400.050	1.452	1.070
100	**100,000**	1924.353	3695.948	1082.370	2890.020	1.778	1.279
100	**1,000,000**	32140.840	45531.370	9013.630	23762.100	3.566	1.916
500	**10,000**	984.357	869.217	158.730	181.100	6.201	4.800
500	**100,000**	2548.083	2991.866	1121.820	1841.540	2.271	1.625
500	**1,000,000**	23431.980	45118.190	9091.370	16684.070	2.577	2.704
750	**10,000**	2382.910	2418.270	186.660	231.510	12.766	10.446
750	**100,000**	4172.555	4657.067	1120.780	1810.410	3.723	2.572
750	**1,000,000**	29006.060	-*	9181.570	20601.350	3.159	-*
1,000	**10,000**	5375.578	5385.314	244.460	270.410	21.990	19.915
1,000	**100,000**	7031.194	7435.575	1173.320	1751.660	5.993	4.245
1,000	**1,000,000**	27360.160	29893.300	9214.810	13036.350	2.969	2.293

* SweepFinder terminated abruptly due to a failed assertion: “SweepFinder: SweepFinder.c:595: ln_likelihood: Assertion `pr >=0.0 && pr<1.00000001' failed”

Figure Legends

Figure 1

Figure1: Comparison of peak memory consumption between SweeD and SweepFinder. Simulated datasets of 100 SNPs and 25, 50, 100, 200, and 400 respective sequences were used for the measurements. Memory consumption was quantified with the massif tool of the valgrind software (Seward and Nethercote 2005). SweeD consumes more memory than SweepFinder due to the lookup table implementation.

Figure 2

Figure 2: Speedup measurements using up to 48 cores for the analysis of simulated datasets consisting of 100 (A) and 10000 (B) sequences with 10,000, 100,000 and 1,000,000 SNPs, respectively.

Figure 3

Figure 3: Genome-scan for selective sweeps of the human chromosome 1. The x-axis denotes the position on chromosome 1, and the y-axis shows the ω-statistic (A) and the CLRs evaluated by SweeD (B), respectively.

Figure 4

Figure 4: Comparison of memory consumption (A) and run-time (B) of SweeD (where the average SFS is computed by the data itself) and SweeD using the MPFR library to calculate the analytical SFS.

Simulated standard neutral datasets of 500 SNPs and 25, 50, 100, 200, and 400 sequences were used for the measurements. Memory consumption was quantified with the massif tool of the valgrind software (Seward and Nethercote 2005).

Figures

Figure 1

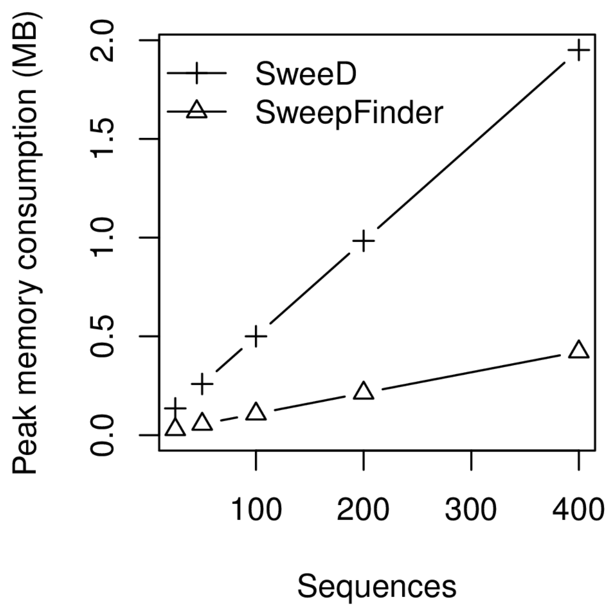


Figure 2

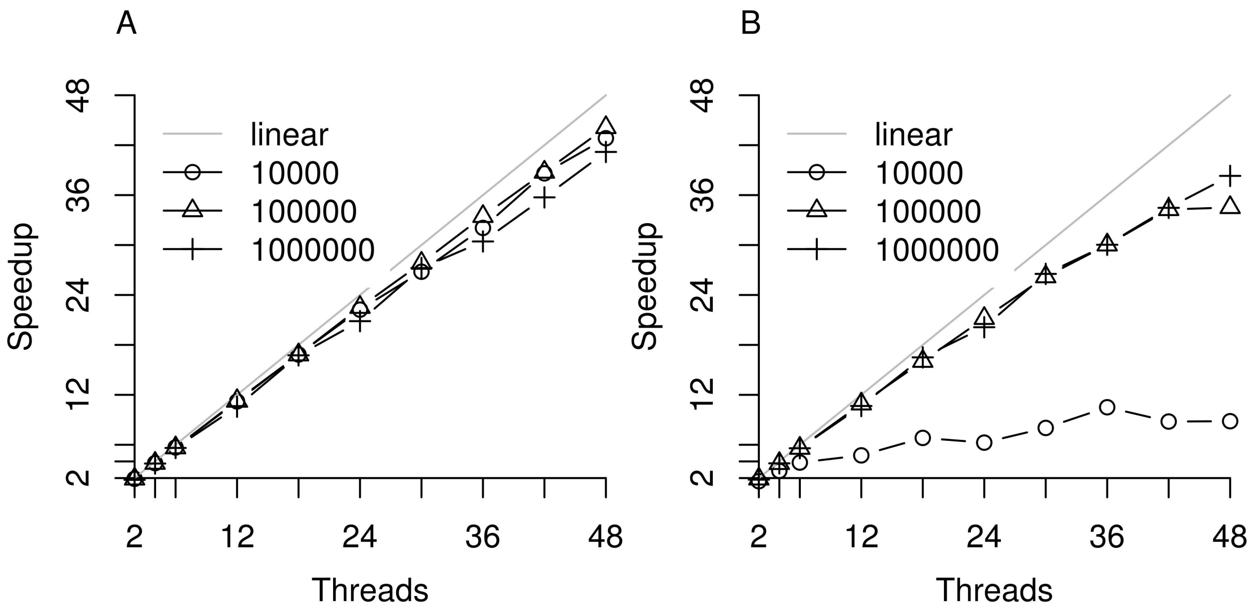


Figure 3

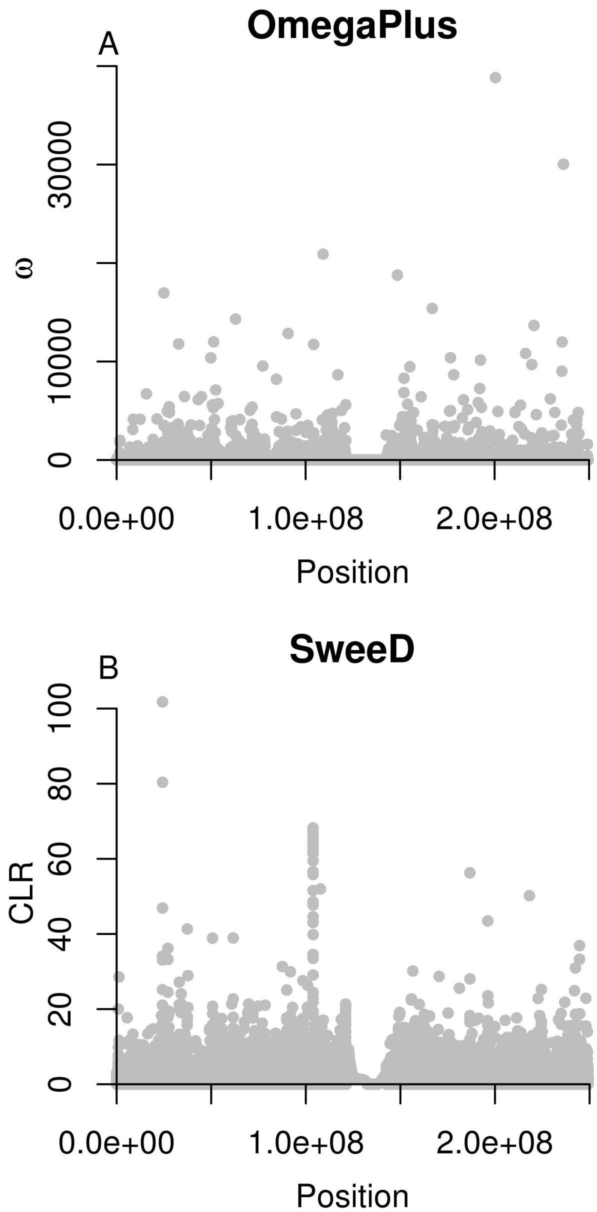
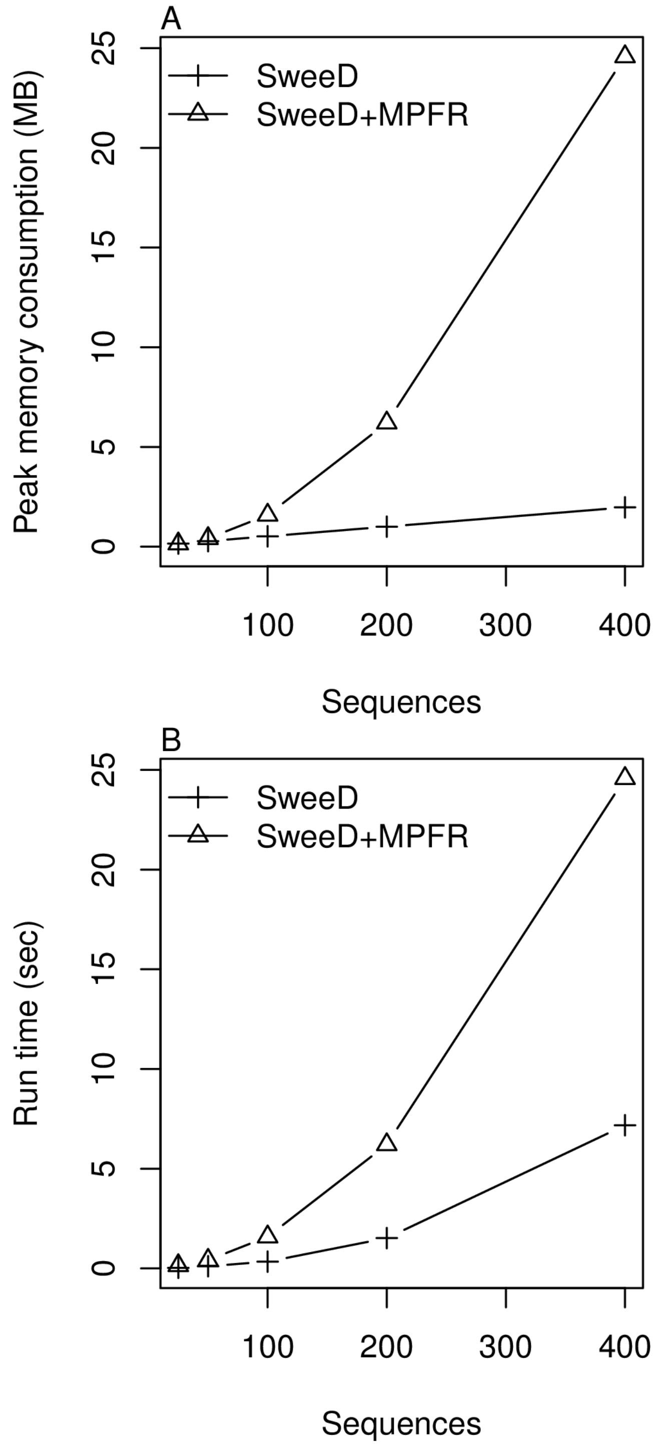

OmegaPlus
A
ω
0
10000
30000
0.0e+00
1.0e+08
2.0e+08
Position
SweeD
B
CLR
0
20
40
60
80
100
0.0e+00
1.0e+08
2.0e+08
Position

Figure 4


A
SweeD
SweeD+MPFR
Peak memory consumption (MB)
0
5
10
15
20
25
100
200
300
400
Sequences
B
SweeD
SweeD+MPFR
Run time (sec)
0
5
10
15
20
25
100
200
300
400
Sequences