

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

3

4 Pavlos Pavlidis<sup>1\*</sup>, Daniel Živković<sup>2</sup>, Alexandros Stamatakis<sup>1</sup>, Nikolaos Alachiotis<sup>1</sup>

5 <sup>1</sup>The Exelixis Lab, Scientific Computing Group, Heidelberg Insitute for Theoretical Studies (HITS  
6 gGmbH), Schloss-Wolfsbrunnenweg 35, D-69118, Heidelberg, Germany

7 <sup>2</sup>Section of Evolutionary Biology, Biocenter, University of Munich, D-82152 Planegg-Martinsried,  
8 Germany

9

10

11

12 **\*Corresponding author:**

13

14 Pavlos Pavlidis

15 pavlidisp@gmail.com

16 The Exelixis Lab, Scientific Computing Group,

17 Heidelberg Insitute for Theoretical Studies (HITS gGmbH),

18 69118 Heidelberg

19 Germany

20

21

22

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

25

26

27

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

29

30

31 Running Title: SweeD: Sweep Detector

32

33 ***Abstract***

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

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

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

51

## 52 **Introduction**

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

64 Neutral mutations are assumed to arise in a sufficiently large population at a rate of  $\theta/2$  , (  
65  $\theta = 4N\mu$  ,  $\mu$  being the mutation probability per site and per generation). Initially, they are present  
66 as a single copy. Thus, according to the infinitely-many sites model (Kimura 1969), they occur at  
67 previously monomorphic sites. The site frequency spectrum (SFS) of a population denotes the  
68 distribution of the expected number of polymorphic sites,  $\phi(x)dx$  , at which the mutant allele has a  
69 frequency in  $(x, x+dx)$  ,  $0 < x < 1$  . Kimura (1971) demonstrated that the SFS for the standard  
70 neutral model is given by  $\phi(x)dx = \theta/x dx$  . For the selective sweep model, Fay and Wu (2000) have  
71 shown that the frequency spectrum of neutral sites which are sufficiently close to the beneficial  
72 mutation shifts toward an excess of high- and low-frequency derived alleles in proportions  
73  $x\phi(x)dx = \theta dx$  and  $(\theta/x - \theta)dx$  , respectively. While the aforementioned neutral and selective  
74 models assume a constant population size, analytical results for the SFS have also been obtained for

75 scenarios in which the population is subject to deterministic size changes (Griffiths 2003). However,  
76 deriving an analytical approximation of site frequency spectrum when sites are subject to genetic  
77 hitchhiking (in populations with varying size over time) still remains a challenge.

78         Regarding analyses of DNA sequence samples, the sample SFS (and not the population SFS) is  
79 of interest. The sample SFS,  $f_{n,i}$ , is the distribution of the expected number of sites at which there  
80 are  $i$  derived alleles,  $1 \leq i \leq n-1$ , in a sample of  $n$  sequences. The relative frequencies are obtained  
81 from these absolute frequencies via division by the total number of segregating sites. If the mutant  
82 allele can not be distinguished from the wild type, the folded version of the SFS is used. Kim and  
83 Stephan (2002) interpreted  $f_{n,i}$  as the probability of observing a single site where  $i$  derived  
84 alleles are found in a sample of size  $n$ . The authors used the derivation of the SFS by Fay and Wu  
85 (2000) to develop the first composite likelihood ratio test (CLR) for detecting selective sweeps in  
86 typically small (up to a few hundred kilobases) genomic regions (henceforth called subgenomic  
87 regions). Nielsen et al. (2005) introduced two major modifications to the CLR method by Kim and  
88 Stephan (2002) for detecting selective sweeps in whole-genome data.

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

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

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

116 At present only a handful of tools that scale to thousands of whole-genome sequences is  
117 available. The implementation of the CLR test by Kim and Stephan (2002) can only be used for  
118 analyzing small subgenomic regions. Jensen *et al.* (2007) and Pavlidis *et al.* (2010) used the  $\omega$ -statistic  
119 (Kim and Nielsen 2004), which relies on the linkage-disequilibrium signature of a selective sweep to  
120 detect positively selected sites. The respective implementations are also only able to handle

121 subgenomic regions. SweepFinder (Nielsen et al. 2005) can analyze whole genomes efficiently, but  
122 only for up to a few hundred sequences. For larger sample sizes, execution times increase substantially.  
123 Moreover, SweepFinder can not analyze samples exceeding 1,027 sequences because numerical  
124 problems associated to floating point underflow are not handled. Finally, SweepFinder only runs on a  
125 single core. To the best of our knowledge, the  $\omega$ -statistic based OmegaPlus tool (Alachiotis et al. 2012)  
126 represents the sole publicly available high-performance implementation for detecting selective sweeps.  
127 OmegaPlus can efficiently analyze whole genomes from thousands of individuals by exploiting all  
128 available cores on a modern desktop or server.

129

### 130 ***New approaches***

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

138 Moreover, SweeD can analyze thousands of genomes because we adapted the numerical  
139 implementation of the arithmetic operations. For a large number of genomes, the double precision  
140 floating-point range is frequently not sufficient. This may lead to numerical over- or underflow. SweeD  
141 is able to analyze such large samples because it performs several calculations at the logarithmic scale.  
142 The code also supports several additional input file formats for reading in simulated and real datasets.

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

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

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

## 153 ***Results and Discussion***

154

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

### 157 **Sequential Performance**

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

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

181 Table 1 also shows that, for a small number of sequences, SweeD becomes faster than SweepFinder as  
182 the number of SNPs increases. This is because the order and the number of operations at each position,  
183 where the CLR is calculated, is different in SweeD (see section **Arithmetic deviations from**  
184 **SweepFinder** for more details). We obtained speedups between 1.07X and 3.90X. For larger numbers  
185 of sequences (1,000), the speedup of SweeD over SweepFinder drops from 22X (10,000 SNPs) to 2.9X  
186 (1,000,000 SNPs) with an increasing number of SNPs because a larger fraction of overall execution  
187 time is spent for CLR computations.



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

## 200 **Parallel Performance**

201

202 To assess the parallel efficiency of SweeD, we generated datasets with up to 10,000 sequences and  
203 1,000,000 sites. Figure 2 shows the respective speedups for up to 48 cores/threads (4 AMD Opteron  
204 6174 processors) on simulated datasets with 100 and 10,000 sequences, and 10,000, 100,000, and  
205 1,000,000 SNPs, respectively. The execution times for the sequential analysis of the dataset with 100  
206 sequences are shown in Table 1. The datasets with 10,000 sequences as well as 10,000, 100,000, and  
207 1,000,000 SNPs required 30,717, 32,299, and 37,212 seconds, respectively.

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

211 10,000 sequences on 48 cores. This is due to the small amount of SNPs for the comparatively large  
212 number of sequences, which in turn leads to a significantly larger amount of time spent in the BFGS  
213 (Broyden-Fletcher-Goldfarb-Shanno, Fletcher 1987) algorithm that optimizes the neutral SFS.  
214 Specifically, the BFGS algorithm estimates the neutral SFS that maximizes the probability of the  
215 dataset (i.e., the overall likelihood) given the input SFS and the data. This step is needed because the  
216 input dataset may contain missing data, and thus the input SFS does not correspond precisely to the  
217 sample SFS. These likelihood computations have been parallelized. However, when the number of  
218 SNPs is small compared to the number of sequences, substantially more iterations (and hence thread  
219 synchronization events) are required for the BFGS algorithm to converge. This step cannot be further  
220 parallelized because the iterative optimization procedure uses the likelihood values sequentially, that is,  
221 there exists a hard-to-resolve sequential dependency between iterations  $i$  and  $i+1$ .

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

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

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

235 12 in our tests), OmegaPlus outperforms SweeD due to algorithmic innovations and because it mostly  
236 relies on integer rather than on floating-point arithmetics.

### 237 **Usage Example**

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

### 250 **Conclusions and future work**

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

257 hardware failures or queue limitations, leading to time and energy savings.

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

## 262 ***Materials and Methods***

### 263 **The SFS of samples for deterministically varying population size**

264

265 Analytical results for sample frequency spectra can either be directly derived via the coalescent or be  
266 obtained via binomial sampling from the population version as derived within the diffusion framework.

267 This is also the case for a neutral model of a population whose size varies over time. Here

268  $\rho(t) = N(t)/N$  denotes the ratio between the ancestral and the current population size at time  $t$ .

269 Changes in population size can be included into the standard neutral model as the harmonic mean of

270 the relative population sizes via time-rescaling  $t \rightarrow \int_0^t 1/\rho(s) ds$ . Griffiths and Tavaré (1998)

271 established the SFS within the coalescent framework, and Živković and Stephan (2011) found an

272 equivalent solution based on diffusion theory (Evans et al. 2007) as

273 
$$f_{n,i} = \frac{\theta}{i} \sum_{k=2}^n (-1)^k (2k-1) \binom{k}{2} {}_3F_2(n-i+1, k, 1-k; n+1, 2; 1) \int_0^\infty \exp\left(-\binom{k}{2} \int_0^t 1/\rho(s) ds\right) dt$$

274 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

275 function, in which  $p_{(0)} = 1$  and  $p_{(l)} = p(p+1)\dots(p+l-1)$ ,  $l \geq 1$ . For the standard neutral model,

276 this equation reduces to  $f_{n,i} = \frac{\theta}{i}$ . The relative frequency spectrum is obtained via division by the  
277 total number of segregating sites. The equation for the SFS can be applied to demographic models  
278 including various instantaneous size changes and multiple phases of exponential growth. It can also be  
279 used to calculate the composite likelihood of all considered sites of a dataset based on a given  
280 demographic model and in analogy to Kim and Stephan (2002).

## 281 **Implementation**

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

285

## 286 *Optional computation of the SFS for a given demographic model*

287 A new feature of SweeD that is not available in SweepFinder is the calculation of the theoretical  
288 sample SFS for a user-specified demographic model. The model can comprise an arbitrary number of  
289 instantaneous population size changes and, optionally, an exponential growth as the most recent event.  
290 For the calculation of the theoretical sample SFS, numerical issues can arise for samples exceeding 60  
291 sequences. To solve recurrent issues with numerical precision that are related to the harmonic sum  
292 representation of the SFS, we used the MPFR (Multiple-Precision Floating-point library with correct  
293 Rounding, Fousse et al. 2007) library. The MPFR library can be used to conduct arbitrary precision  
294 floating-point operations where required. Using arbitrary precision arithmetics, however, leads to  
295 increased run times and memory requirements for the analytical computation of the SFS compared to  
296 double precision floating point arithmetics. Although the run time differences are negligible for small

297 sample sizes (up to approximately 50 sequences), computing times can increase substantially (up to 5  
298 times in Figure 4B) with the number of sequences. We employ a lookup table to alleviate this  
299 performance issue by avoiding frequent re-computations of these values. This approach reduces run  
300 times by a factor that is approximately equal to the number of sequences. However, the size of the  
301 lookup table also increases quadratically with the number of sequences and may induce excessive  
302 memory requirements (Figure 4A).

303

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

### 309 **Parallelization**

310 Multi-core systems can run several threads of execution in parallel which can decrease run  
311 times of an application. However, substantial changes to the sequential code may be required to obtain  
312 an efficient parallel algorithm. Therefore, we focused on parallelizing the most compute-intensive parts  
313 of SweeD. As already described, SweeD computes the likelihood and optimizes the  $\alpha$ -parameter of the  
314 CLR test at several positions of the alignment. Since the CLR calculations at different positions (CLR  
315 positions) are independent, they are equally divided among the available cores. However, there is load  
316 imbalance among CLR computations because the inference of  $\alpha$ -parameters at CLR positions that are  
317 located close to a selected site requires a larger amount of arithmetic operations. When a CLR position  
318 is located near a positively selected site, the  $\alpha$ -parameter value that maximizes the likelihood of the  
319 sweep model is smaller ( $\alpha$  is inversely proportional to the selection coefficient). However, the size of a

320 genomic region that a selective sweep may affect is inversely proportional to  $\alpha$ . Thus, more SNPs are  
321 required to compute  $\alpha$ , when the  $\alpha$  value decreases. Therefore, we distribute CLR positions in a cyclic  
322 way to cores such as to improve load balance. We plan to test whether more elaborate load balancing  
323 schemes, such as dynamic scheduling or guided scheduling can further improve load balance.

#### 324 **Arithmetic deviations from SweepFinder**

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

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

343 > 0.0' failed”.

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

### 351 **Checkpoint and restart capability**

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

359           SweeD uses the open-source checkpointing library DMTCP (Ansel et al. 2009) for this purpose.  
360 With the respective makefiles (with the file extension .CHECKPOINTS), users can compile the  
361 checkpointable version of SweeD: SweeD-C. Note that the non-checkpointable version does not  
362 require the DMTCP library and is hence easier to compile and install. The checkpointable version takes  
363 one additional input parameter, the checkpointing interval, which defines how often checkpoints are  
364 created and stored during the execution of SweeD-C. To enable checkpointing, the `dmtcp_coordinator`  
365 process has to be started before executing SweeD-C. Subsequently, the program can be invoked as



366 usual (with the additional parameter for the checkpointing interval). When an unexpected event such as  
367 a queue time-out or an electricity or processor failure interrupts the execution of the program, the user  
368 will be able to resume the execution by using the restart script provided with the DMTCP library.

### 369 **Command line arguments and output files**

370 SweeD is a command line tool and requires at least three parameters for a typical analysis: i) a name  
371 for the run (*-name*), ii) the name of the input file (*-input*), and iii) the number of CLR positions (*-grid*).  
372 In addition to the input file format of SweepFinder (see **SweeD manual Section at [http://exelixis-](http://exelixis-<br/>373 lab.org/software.html)**

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

375

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

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

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

379

380 In the first example, SweeD is called with the minimum number of parameters to compute the CLR at  
381 10,000 positions along the dataset as provided in file.sf. In the second example, the parallel version of  
382 SweeD is called. Hence we need an additional parameter to specify the number of cores/threads that  
383 shall be used. In the last example, we start the checkpointable version. This requires the additional  
384 parameter that specifies how frequently (in seconds) checkpoints should be stored. For more examples  
385 and a detailed description of all supported command line parameters please refer to the manual  
386 (<http://exelixis-lab.org/software.html>).

387

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

393

394

395

396

397

398

399

400

401

402

403 **References**

- 404 Alachiotis N, Pavlidis P, Stamatakis A. 2012. Exploiting Multi-grain Parallelism for Efficient  
405 Selective Sweep Detection. In: Algorithms and Architecture for Parallel Processing (ICA3PP).  
406 Vol. 7439. p. 56–68.
- 407 Alachiotis N, Stamatakis A, Pavlidis P. 2012. OmegaPlus: A Scalable Tool for Rapid Detection  
408 of Selective Sweeps in Whole-Genome Datasets. *Bioinformatics (Oxford, England)* 28:2274–  
409 2275.
- 410 Ansel J, Arya K, Cooperman G. 2009. DMTCP: Transparent Checkpointing for Cluster  
411 Computations and the Desktop. In: 23rd IEEE International Parallel and Distributed  
412 Processing Symposium (IPDPS'09). p. 1–12.
- 413 Chen GK, Marjoram P, Wall JD. 2009. Fast and flexible simulation of DNA sequence data.  
414 *Genome research* 19:136–42.
- 415 Evans SN, Shvets Y, Slatkin M. 2007. Non-equilibrium theory of the allele frequency  
416 spectrum. *Theoretical population biology* 71:109–19.
- 417 Ewing G, Hermisson J. 2010. MSMS: a coalescent simulation program including  
418 recombination, demographic structure and selection at a single locus. *Bioinformatics (Oxford,*  
419 *England)* 26:2064–5.
- 420 Fay JC, Wu CI. 2000. Hitchhiking under positive Darwinian selection. *Genetics* 155:1405–13.

- 421 Fletcher R. 1987. *Practical methods of optimization*. New York: John Wiley & Sons, Ltd
- 422 Fousse L, Hanrot G, Lefevre V, Pélissier P, Zimmermann P. 2007. MPFR: A multiple-precision  
423 binary floating-point library with correct rounding. *ACM Transactions on Mathematical*  
424 *Software* 33:1–15.
- 425 Griffiths RC, Tavaré S. 1998. The age of a mutation in a general coalescent tree.  
426 *Communications in Statistics. Stochastic Models* 14:273–295.
- 427 Griffiths RC. 2003. The frequency spectrum of a mutation, and its age, in a general diffusion  
428 model. *Theoretical population biology* 64:241–51.
- 429 Hudson RR. 2002. Generating samples under a Wright-Fisher neutral model of genetic  
430 variation. *Bioinformatics (Oxford, England)* 18:337–8.
- 431 Jensen JD, Thornton KR, Bustamante CD, Aquadro CF. 2007. On the utility of linkage  
432 disequilibrium as a statistic for identifying targets of positive selection in nonequilibrium  
433 populations. *Genetics* 176:2371–9.
- 434 Kim Y, Nielsen R. 2004. Linkage disequilibrium as a signature of selective sweeps. *Genetics*  
435 167:1513–1524.
- 436 Kim Y, Stephan W. 2002. Detecting a local signature of genetic hitchhiking along a  
437 recombining chromosome. *Genetics* 160:765–777.
- 438 Kimura M. 1969. The number of heterozygous nucleotide sites maintained in a finite

439 population due to steady flux of mutations. *Genetics*:893–903.

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

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

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

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

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

451 The 1000 Genomes Project Consortium. 2012. An integrated map of genetic variation from  
452 1,092 human genomes. *Nature* 491:56–65.

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

455

456 **Tables**

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

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

458 \* SweepFinder terminated abruptly due to a failed assertion: "SweepFinder: SweepFinder.c:595: ln\_likelihood: Assertion `pr >=0.0 &&

459 pr<1.00000001' failed"

460

461

462 **Figure Legends**

463

464 **Figure 1**

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

470

471 Figure 2

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

474

475 Figure 3

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

479

480 **Figure 4**

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

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

486



487 *Figures*

488

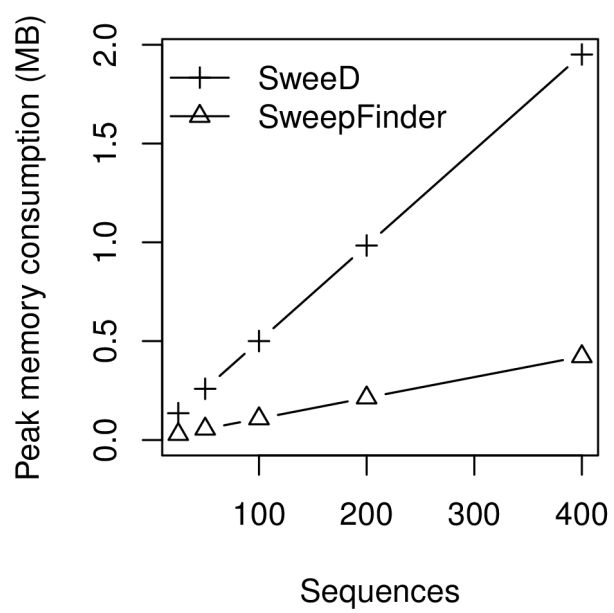
489

490

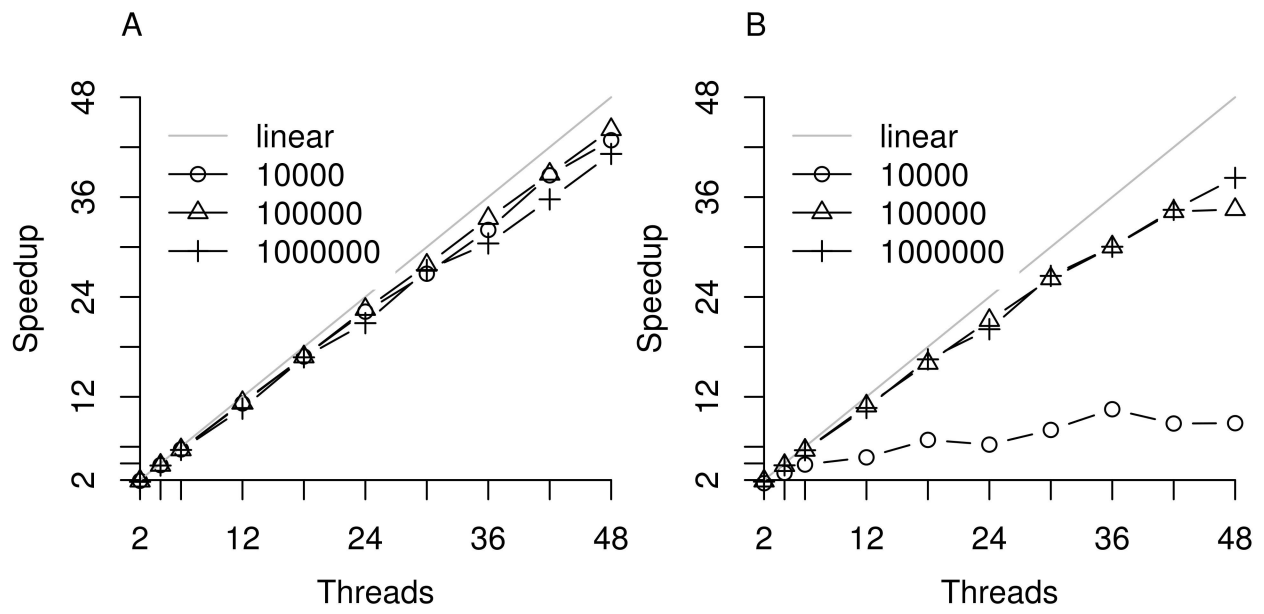
491 Figure 1

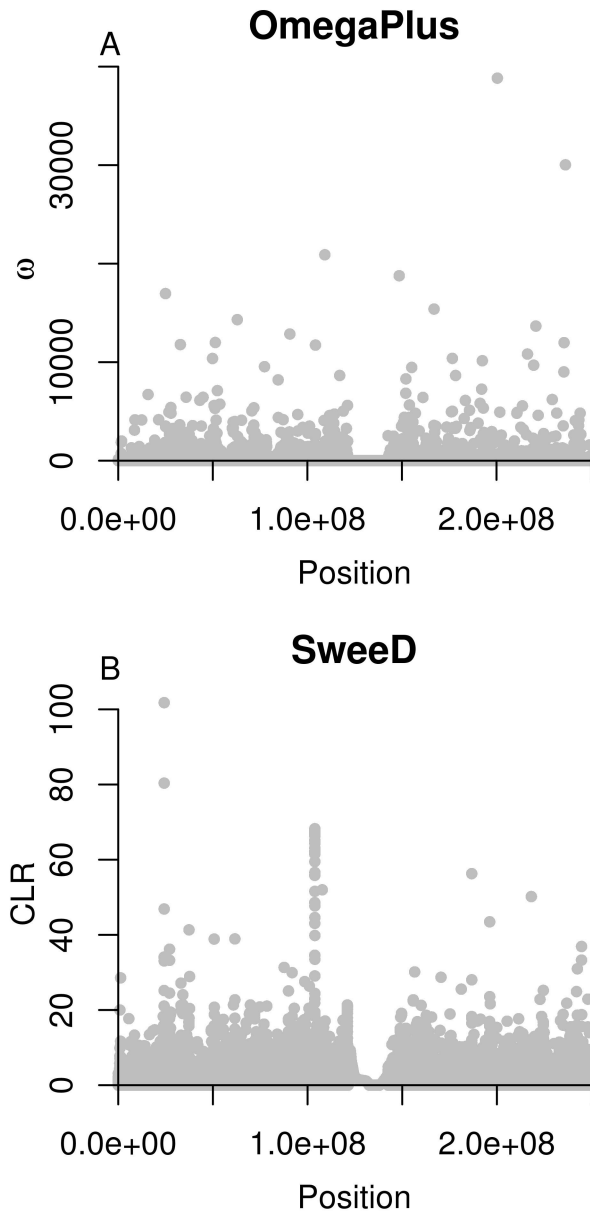
492

493



495 Figure 2





498

499

500

501

502

503

504

505

506

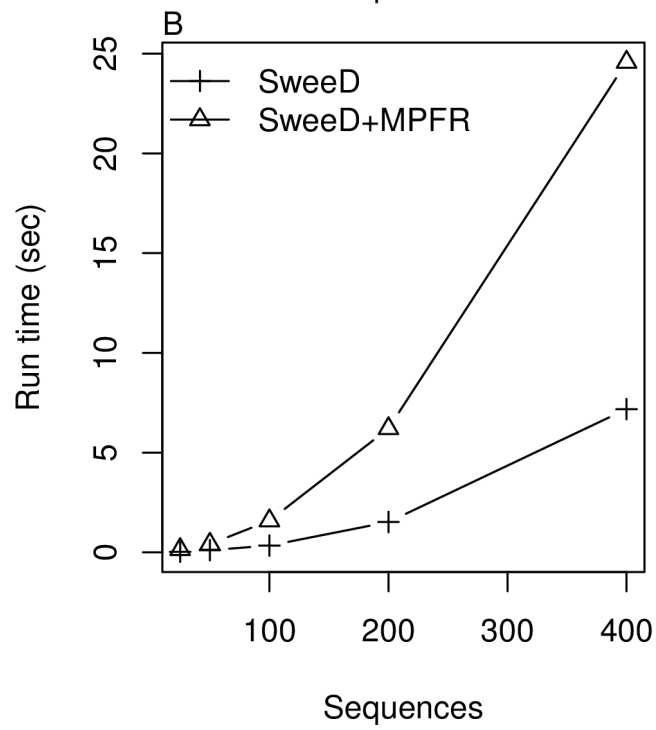
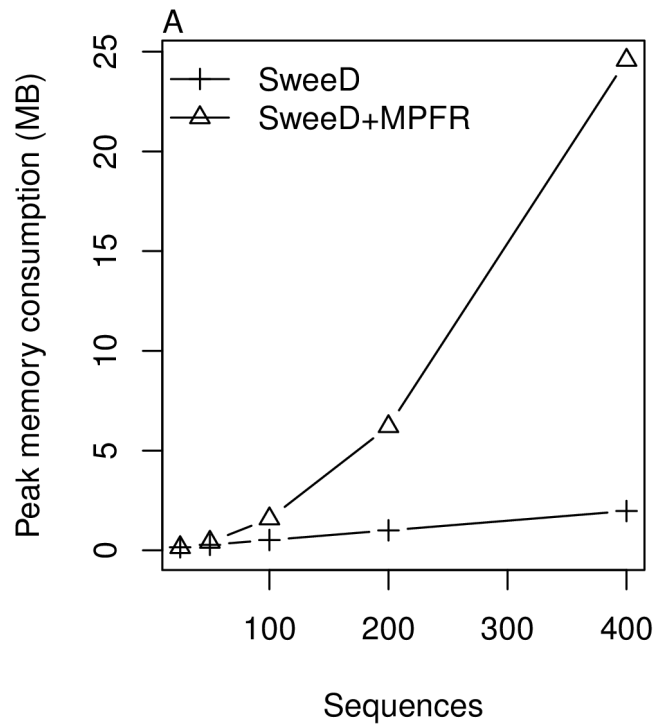
507

508 Figure 4

509

510

511



513

514