# Open-source code for self-consistent field theory calculations of block polymer phase behavior on graphics processing units*

**Guo Kang Cheong, Anshul Chawla, David C Morse and Kevin D Dorfman**

Department of Chemical Engineering and Materials Science, University of Minnesota – Twin Cities, 421 Washington Avenue SE, Minneapolis, Minnesota 55455, USA

E-mail: morse012@umn.edu

E-mail: dorfman@umn.edu

10 September 2021

**Abstract.** Self-consistent field theory (SCFT) is a powerful approach for computing the phase behavior of block polymers. We describe a fast version of the open-source Polymer Self-Consistent Field (PSCF) code that takes advantage of the massive parallelization provided by a graphical processing unit (GPU). Benchmarking double precision calculations indicate up to 30× reduction in time to converge SCFT calculations of various diblock copolymer phases when compared to the Fortran CPU version of PSCF using the same algorithms, with the speed-up increasing with increasing unit cell size for the diblock polymer problems examined here. Where double precision accuracy is not needed, single precision calculations can provide speed-up of up to 60× in convergence time. These improvements in speed within an open-source format open up new vistas for SCFT-driven block polymer materials discovery by the community at large.

*Keywords*: self-consistent field theory, block polymer, graphical processing unit, phase behavior

Submitted to: *Eur. J. Phys. E*

* Supplementary material in the form of executable files available from the Journal web page at xxx

## 1. Introduction

When a block polymer melt is cooled below its order-disorder transition temperature, it spontaneously undergoes microphase separation into an ordered morphology. The selection of a particular ordered structure at a given temperature can be tuned by changing the volume fraction of each block, the chain architecture, and the degree of segregation $\chi N$, where $\chi$ is the Flory-Huggins parameter and $N$ is the degree of polymerization [1]. This tunability forms the basis for using block polymers to design soft materials with particular physical and chemical properties [2]. An emerging challenge is determining what material to make from the vast design space afforded by block polymer chemistry [3]. Modern polymer synthesis techniques now provide a vast range of possible block chemistries and block polymer architectures (linear, star, miktoarm, bottle-brush) [4]. Even for the simplest case of a linear block polymer, the number of possible permutations grows factorially with increasing number of blocks and chemistries [3]. As a result, it is infeasible to synthesize and characterize every possible permutation in an attempt to discover materials with novel properties. It is exceedingly useful to have a guided strategy to systematically search such a large parameter space for a desired material property [5,6]. Any such strategy requires an efficient computational approach, a need that we address here through a fast version of an open-source code for self-consistent field theory (SCFT) calculations.

The key step in computational design of block polymer materials is identifying the stable morphology at a given state point, ideally via parameters that can be readily mapped to experimental studies. SCFT is well-suited to this purpose, and can also serve as the computing engine for inverse design strategies [7–13]. However, SCFT has been underutilized as a tool due to the lack of readily available open-source software [14]. The release of the Polymer Self Consistent Field (PSCF) code [15] improved this situation by providing an open-source SCFT solver along with guides for initializing calculations and examples of usage [14].

The original PSCF code [14] was designed to utilize only a single CPU core. This provides adequate speed for many tasks, and has allowed the code to be successfully used to analyze block copolymer phase behavior [16–34]. To determine the relative stability of different candidate morphologies in a region of the block polymer parameter space, an SCFT calculation must be performed for each competing morphology at many points in parameter space. If one has access to a CPU cluster, this task can be partially parallelized by assigning different morphologies and different regions of parameter space to different CPU cores. The single CPU implementation inevitably becomes a bottleneck, however, when each calculation requires a sufficiently long time to complete. The time required to solve the modified diffusion equation by a pseudo-spectral SCFT algorithm scales as $O(N_s N_n \log N_n)$ [17], where $N_n$ is the number of nodes in the spatial mesh and $N_s$ is the number of contour length integration steps. Some particularly important examples of computationally challenging problems involve candidate morphologies that have very large unit cells, such as Frank-Kasper phases and

Laves phases [21, 23, 24, 35–50]. Evaluating the relative stability of several competing Frank-Kasper phases is particularly challenging because of the need to use a large number nodes in the spatial mesh, and the need to test the dependence of free energy on both $N_n$ and $N_s$ to ensure sufficient accuracy when comparing phases with very similar free energies [23]. Obtaining a SCFT solution for Frank-Kasper phases in PSCF also generally requires many iterations to converge [14, 51]. Likewise, problems involving strongly segregated components and narrow interfaces are particularly challenging. Computation time also becomes a bottleneck for inverse design problems [7–13], since many SCFT calculations are required during the search of the design space. Clearly, PSCF stands to benefit from having a massively parallelized version which could be used to accelerate expensive calculations.

For this reason, we have implemented a GPU-accelerated version of PSCF. Delaney and Fredrickson previously described a GPU implementation of SCFT within their closed software package and found remarkable speed-ups of up to 60× for single precision calculations [52]. Following these authors, we also chose a GPU implementation over a multiple-CPU mesage passing implementation because the computational time of SCFT is dominated by Fast Fourier Transforms (FFT) [52]. FFTs are very effectively accelerated on GPUs, but suffer from significant communication costs in multi-CPU message passing implementations.

The GPU-accelerated code that we focus on here differs from that of Delaney and Fredrickson [52] in two ways. First, the two codes use different iteration algorithms. The algorithms described by Delaney and Fredrickson obtain a solution of the self-consistent field equations through a relaxation scheme. Our implementation instead uses an Anderson mixing iteration scheme [53, 54] that can be used to optimize the unit-cell dimensions simultaneously with the solution of the self-consistent field equations [51, 55]. Secondly, and more importantly, our implementation is available for use by others as an open-source code. We found that our implementation obtains a speed-up of up to ∼60× relative to the existing Fortran PSCF code for sufficiently large single precision problems, comparable to the speed-up obtained by Delaney and Fredrickson.

Though we focus in this paper on a GPU-accelerated SCFT code for periodic structures, this code is being distributed [56] as part of a rewritten version of the PSCF package that also contains several CPU-based programs. While the original PSCF program was written in Fortran 90, the new package has been rewritten using C++ for code that is implemented on the CPU and CUDA for code that is implemented on a GPU.

## 2. Algorithm and implementation details

In the spirit of our prior publication on the PSCF Fortran code [14], we provide here a walk-through of the algorithms used in the code, focusing on design choices that affect performance. As part of this, we briefly reiterate some of the relevant governing equations [1, 14, 57, 58]. For simplicity, we present some equations in a form appropriate

to a one-component melt of linear block copolymers of equal length $N$, though the code can perform simulations of mixtures that can contain both branched and linear polymers.

The most computationally expensive part of SCFT is the solution of the modified diffusion equation (MDE). The forward propagator, $q(\mathbf{r}, s)$, of a linear multiblock polymer is computed from the solution of the MDE,

$$\frac{\partial q(\mathbf{r}, s)}{\partial s} = \left[\frac{b_\alpha^2}{6}\nabla^2 - \omega_\alpha(\mathbf{r})\right] q(\mathbf{r}, s) \quad . \tag{1}$$

The parameter $b_\alpha$ is the statistical segment length within a block containing monomers of type $\alpha$, $s \in [0, N]$ is a contour variable, $\mathbf{r}$ is the position variable, and $\omega_\alpha$ is the spatially-dependent potential field acting on monomers of type $\alpha$. The corresponding backward propagator, $q^\dagger(\mathbf{r}, s)$, follows the same MDE but with the sign of the time derivative in (1) reversed. In both cases, the initial condition is $q(\mathbf{r}, 0) = q^\dagger(\mathbf{r}, N_s) = 1$.

Solutions of the MDE are obtained in PSCF using the pseudospectral algorithm of Ranjan, Qin and Morse (RQM) [17]. This algorithm yields a discretization error of order $(\Delta s)^4$, where $\Delta s = N/N_s$ is the contour length step size, $N$ is chain contour length, and $N_s$ is the number of contour length steps. Typically, each chain is discretized into $N_s \sim 10^2$ steps. The RQM algorithm has been shown to be very effective for high accuracy SCFT calculations [59]. This algorithm involves six Fast Fourier Transform (FFT) operations at each step of integration along the contour length of the polymer, which dominate the cost of the algorithm [52]. In our code, NVIDIA's CUFFT library is used to perform all FFTs on the GPU. All other operations with a cost per contour step that is proportional to the number of grid points, such as point-wise multiplications, are also implemented on the GPU.

Following the solution of (1) for $q$ and the counterpart equation for $q^\dagger$, the volume fraction of each monomer at position $\mathbf{r}$ is computed as

$$\phi(\mathbf{r}) = \frac{1}{NQ}\int q(\mathbf{r}, s) q^\dagger(\mathbf{r}, s) ds \quad , \tag{2}$$

where the partition function for an unconstrained chain is

$$Q = \frac{1}{V}\int q(\mathbf{r}, N_s) d\mathbf{r} \quad . \tag{3}$$

The integral with respect to $s$ in Eq. (2) is computed using Simpson's rule, while the spatial average in Eq. (3) is computed as an average over mesh nodes [54]. Both types of integration are implemented on the GPU, where the division of work is at the level of an individual grid point.

The chemical potential fields in the modified diffusion equation must be chosen so as to satisfy the SCFT self-consistency condition,

$$\omega_\alpha(\mathbf{r}) = \sum_{\beta \neq \alpha} \chi_{\alpha\beta}\phi_\beta(\mathbf{r}) + \xi(\mathbf{r}) \quad , \tag{4}$$

for the chemical potential potential field, $\omega_\alpha$ for each monomer type $\alpha$. Here, $\xi(\mathbf{r})$ is a Lagrangian that enforces incompressibility constraint, which requires that

$$\sum_\alpha \phi_\alpha(\mathbf{r}) = 1 \tag{5}$$

at every position $\mathbf{r}$ in the unit cell.

In a SCFT calculation with a flexible unit cell, the MDE and SCFT equations are supplemented by a requirement that parameters of the unit cell be chosen so as to minimize the free energy per unit volume or (in a one-component melt) per chain. Let $N_c$ denote the number of parameters (lengths and angles) required to describe the unit cell of a particular type (e.g., cubic, hexagonal, tetragonal, etc.), and let $\theta_1, \ldots, \theta_{N_c}$ denote a list of these parameters. In a block copolymer melt, the requirement of minimization of the free energy can be expressed as a requirement that

$$0 = -\frac{1}{Q}\frac{\partial Q}{\partial \theta_i} \tag{6}$$

for all $i = 1, \ldots, N_c$. The calculation of the stress in the unit cell in (6) is obtained using a perturbation theory that provides an analytical form of derivative $\partial Q/\partial \theta_i$ [55]. Efficient calculation of this derivative in a pseudo-spectral algorithm requires knowledge of the spatial Fourier transforms of the forward and reverse MDE solutions $q(\mathbf{r}, s)$ and $q^\dagger(\mathbf{r}, s)$ at each step of the contour length variable. Calculation of $\partial Q/\partial \theta_i$ thus requires the calculation of two additional FFTs per contour integration step, per trial choice of the chemical potential fields and unit cell parameters, in addition to the six FFTs required by the RQM algorithm for solving the MDE. These two additional FFTs are performed on the MDE solutions $q$ and $q^\dagger$ after each completion of the RQM algorithm for computing these quantities. To reduce the cost of these additional FFTs, solutions of $q$ and $q^\dagger$ are stored in a contiguous memory array that can then be pipelined into the CUFFT batching system. This batch processing of FFTs results in a speed-up of up to $2\times$ for this operation compared to performing separate FFTs.

Equations (1)-(6) constitute a set of nonlinear, non-local equations that need to be solved simultaneously to obtain a SCFT solution in an optimal unit cell. A calculation with a flexible unit cell typically begins with a guess for the potential fields $\omega_\alpha(\mathbf{r})$ and for the unit cell parameters. Reference [14] provides step-by-step recipes for generating initial chemical potential fields for both particle-forming phases and network phases. These initial guesses must then be iteratively adjusted until Equations (4)-(6) are satisfied. To obtain a SCFT solution in a fixed unit cell, one instead begins with an initial guess for the chemical potential fields and iteratively solves Equations (1)-(5), while treating the unit cell parameters as constants.

The self-consistent solution is found using an iterative procedure known as Anderson mixing [51, 53, 54, 60]. The form of Anderson mixing used in both the CPU code and the GPU code is the same as that presented by Arora *et al.* [51], which is only briefly summarized here. This algorithm is based on a reformulation of equations (4) and (5) for a system with $N_m$ distinct types of monomers as a set of $N_m$ independent equations per point in space (in a continuum) or per node on the FFT mesh (in a discretized solution), given by equations (10-12) in Ref. [51]. In a SCFT problem with a rigid unit cell, this formulation gives a system of $N_m N_f$ independent equations that must be satisfied by the chemical potential fields, where $N_f$ is the number of independent degrees of freedom used to represent the chemical potential and volume fraction fields

associated with each monomer type. The version of Anderson mixing implemented in the CPU code for periodic microstructures uses a representation of these fields as an expansion in symmetry-adapted basis functions. In this case, $N_f$ denotes the number of basis function used in the expansion, which is generally less than the number $N_n$ of nodes in the FFT mesh by a factor approximately equal to the number of symmetry elements in the space group. The version of Anderson mixing in the GPU-accelerated code, however, uses a representation that does not impose any space group symmetry, for which $N_f = N_n$. The solution of an SCFT problem with a flexible unit cell must also satisfy equation (6) for each of $N_c$ unit cell parameters, giving a system of $N_m N_f + N_c$ equations.

The resulting system of nonlinear equations can be expressed formally in all cases of interest as a requirement that

$$\mathbf{0} = \mathbf{R}(\mathbf{x}) \quad , \tag{7}$$

where $\mathbf{R}(\mathbf{x})$ denotes a column vector of residuals and $\mathbf{x}$ denotes vector of unknowns. In the case of a rigid unit cell, $\mathbf{x}$ is a vector with $N_m N_f$ elements corresponding to the coefficients required to specify the $N_m$ chemical potential fields, and $\mathbf{R}$ is a vector of $N_m N_f$ corresponding SCFT residuals. In the case of a flexible unit cell, $\mathbf{x}$ also contains an additional $N_c$ elements given by the values of the unit cell parameters, while $\mathbf{R}$ contains an additional $N_c$ elements containing derivatives of free energy with respect to particular unit cell parameters.

Anderson mixing is an iterative algorithm that retains a history of previous trial values of $\mathbf{x}$ and $\mathbf{R}$, and uses information from this history to construct new trial values of $\mathbf{x}$. The variant of the algorithm presented by Arora *et al.* [51] allows problems with fixed unit cell and flexible unit cells to be treated with analogous algorithms, simply by extending the number of elements in $\mathbf{x}$ and $\mathbf{R}$. After $k$ steps of iteration, it retains a history of the $K$ most recent previous values of $\mathbf{x}$ and $\mathbf{R}$. The number of previous trials retained is $K = \min(k+1, N_h)$, where $N_h$, the maximum history length, is a user-selected parameter. Storage of this history incurs a memory penalty of $\mathcal{O}(N_h N_f N_m)$. This is comparable to the memory required to store $q$ and $q^\dagger$ in calculations with $N_h \sim N_s \sim 10^2$.

After the first $N_h$ iterations, each iteration of the Anderson-mixing update algorithm as described by Stasiak and Matsen [54] requires computation of $N_h$ inner products of pairs of residual vectors, at a cost of $\mathcal{O}(N_h N_f N_m)$. We achieved this time complexity by following their suggestion of storing the inner products of pairs of residual vectors.

In our implementation, these operations are performed on the GPU. While the Anderson mixing update algorithm constitutes only a small portion of the computation time in SCFT, we were able to obtain a modest additional speed-up in the GPU code by offloading these update operations to the GPU.

In contrast to the CPU code, which uses double precision, the GPU-accelerated code can be compiled in either single precision or double precision. For professional grade GPUs that are designed for scientific computing, such as as NVIDIA's Tesla, double

precision calculations take roughly twice as much time as corresponding single precision calculations. For more widely available GPUs that were not designed for computation, such as the GTX series, use of double precision can cause a slow down of 32-fold in the ideal case where everything else such as bandwidth is equal. The flexibility of our code reduces the burden of needing sophisticated hardware and thus allows efficient use of the code on more common, less expensive GPUs.

To further take advantage of our choice of hardware, we have also chose to limit the simulation grid to sizes of $N_n = 2^n$. This allows us to write GPU kernels that both require less communication time with the CPU and have a higher computational speed.

## 3. Code Performance

To ensure the efficacy of this code, it is important to address both the speed-up and the accuracy of the calculation. The performance of GPU-accelerated SCFT calculations are well documented by Delaney and Fredrickson [52] for their particular implementation. We have performed basic performance testing here as well, as our implementation of the field relaxation is somewhat different than Delaney and Fredrickson [52], and we also test their conjecture that single-precision codes can be polished to higher accuracy by a subsequent double-precision calculation. We further provide results for GPU-accelerated solutions for Frank-Kasper phases, which have very large unit cells.

For the purpose of comparing the older PSCF Fortran code to the new GPU code, we used the diblock case studies that are distributed with [14] as a benchmarking tool. The files necessary for running the calculations are also included with this paper as the online supplementary material. The SCFT solution provides the free energy of a particular morphology with an optimized domain size, which allows for easy comparison between different implementations. The diblock case studies involve BCC, FCC, gyroid, A15, and the Frank-Kasper $\sigma$ phase. Except where noted otherwise, each calculation proceeded until the root-mean-squared magnitude of elements in the residual vector $\mathbf{R}$, as defined by Arora *et al.* [14], is less than $10^{-5}$. All calculations were performed with a Intel® Xeon® Processor E5-2630 v2 CPU, coupled with a Tesla K40 GPU for the GPU code. The computational time is taken from an average of three trials. All results reported here for a CPU program were obtained with the original Fortran PSCF code.

We performed four sets of calculations to benchmark the code. The first calculation (with the label 'Symmetry') uses the Fortran CPU code with the workflow that was recommended in [14] and simultaneous relaxation of the fields and unit-cell stress [51]. This calculation uses symmetry-adapted basis functions to represent all fields within the Anderson Mixing iteration algorithm, thus constraining the solution to have a specified space group symmetry [14]. We also examined the performance of the CPU code but with no imposed symmetry ('No-Symmetry'), by setting the space group to the identity group. Finally, we tested the performance of both double and single precision calculation on the GPU ('GPU-D' and 'GPU-S' respectively). The 'No-Symmetry' calculations yield the closest equivalence between the CPU and the GPU codes, since

**Table 1.** Grid size and computational time needed to reach convergence for each morphology using the three different SCFT implementations. The corresponding number of Anderson mixing iterations for each implementation is in parenthesis next to the computational time. The 'Symmetry' and 'No-Symmetry' implementations are CPU-based with the conditions as described in Section 3. 'GPU-D' and 'GPU-S' is the double precision and single precision, respectively, GPU-based implementation described in Section 2.

| | | Computational Time [s] (# Iterations) | | | |
|---|---|---|---|---|---|
| Phase | Grid Size, $N_n$ | Symmetry | No-Symmetry | GPU-D | GPU-S |
| BCC | 32×32×32 | 27.9 (51) | 80.7 (56) | 5.4 (49) | 4.3 (68) |
| FCC | 64×64×64 | 256.9 (53) | 580.9 (54) | 20.3 (51) | 11.0 (67) |
| A15 | 64×64×64 | 418.0 (84) | 951.0 (93) | 31.4 (79) | 15.7 (92) |
| $\sigma$ | 64×64×32 | 372.0 (177) | 846.3 (183) | 41.9 (175) | 21.2 (198) |
| Gyroid | 32×32×32 | 21.7 (44) | 47.4 (45) | 4.9 (44) | 3.7 (58) |

we have not yet implemented an iteration algorithm that enforces space group symmetry during iteration in the GPU-based code. The relative speed of the GPU code and CPU code with no imposed symmetry is thus the best measure of the speed-up obtained for identical algorithms as a result of the change in hardware alone. The speed of the GPU code relative to the CPU code with symmetry is instead the speed-up obtained for these crystal structures by switching to the current GPU code from the fastest available CPU implementation.

Figure 1 provides the computational time for two of the case studies, and Table 1 provides the computational time and the number of iterations needed to reach convergence for each of the case studies. Overall, the GPU-accelerated calculations are much faster than the CPU calculations. The speed-up accrued by shifting to the double precision GPU-based SCFT code is around two orders of magnitude, about 30× for A15. The change from double precision to single precision provides an additional 2× speed up per iteration, consistent with our choice of GPU. Taking into account both the change in hardware and the use of single precision calculations, the magnitude of our largest speed-up is 60×, similar to that by Delaney and Fredrickson [52]. Speed aside, Anderson mixing is crucial to the SCFT solutions of Frank-Kasper phases because it is a Jacobian-free method, and thus uses much less memory than, for example, using a Newton-Raphson scheme to update the fields [14].

A common observation is that the relative speed-up obtained by using GPUs tends to increase up until the problem becomes large enough to hide latency. This is observed in our calculations of the case study examples where the largest speed-up are with examples of 64×64×64 grid size, while only moderate speed-up is seen for 32×32×32 grids. Note that, since the algorithm dictates that the computational grid is limited to sizes of $2^n$, we slightly modified the unit cell of the PSCF-distributed case study example of BCC to $32^3$ from the previous value of $36^3$. The improvement with increasing grid

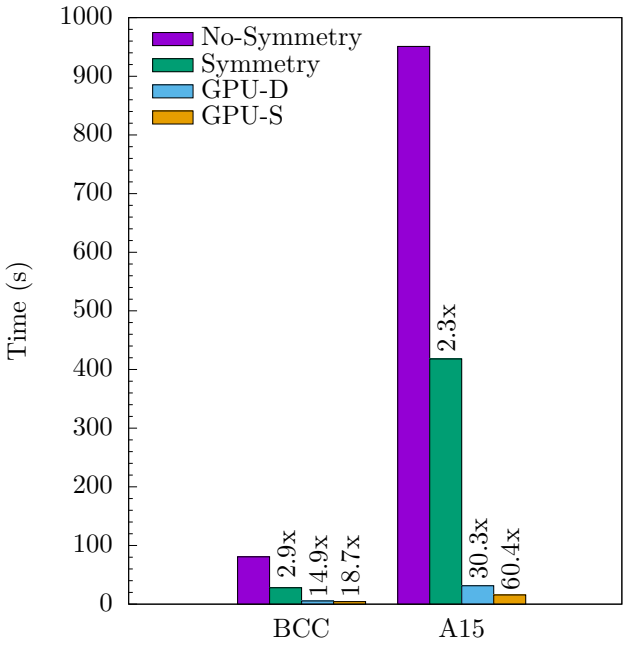


**Figure 1.** Execution time for the SCFT solution for the BCC and A15 phase. The labels above the bars corresponds to the speed-up, which is defined as time to compute the solution with no-symmetry constraints divided by execution time for the other approaches. The grid sizes and number of iterations for each solution are in Table 1, along with similar speed-up results for other morphologies.

size is especially important for SCFT solutions for Frank-Kasper phases, which have large unit cells with a large number of particles [61] and thus require many grid points to resolve the core-matrix interfaces of the particles.

The difference in speed between the 'Symmetry' and 'No-Symmetry' calculations is a result of the use of symmetry-adapted basis functions on the cost of the operations required for Anderson-Mixing update operations. In the CPU code, symmetry adapted basis functions are used to represent fields in the outer Anderson-Mixing iterator algorithm, but not within the pseudo-spectral algorithm used to solve the MDE. Use of symmetry-adapted basis functions thus has no effect on the time required to solve the MDE, but does affect the time required by the Anderson Mixing update operation itself, because it drastically reduces the number of degrees of freedom required to represent the chemical potential field and the corresponding residual vector.

We have not yet implemented any symmetry constraints in the GPU-accelerated code. It is illuminating to estimate the speed-up that might be obtained if this feature were implemented. Comparing the 'Symmetry' and 'No-Symmetry' calculations reveals the effect of including symmetry-constrained basis functions in Anderson mixing. For example, the symmetry-adapted basis for the A15 'Symmetry' case reduces the 262 144 nodes to a mere 6 017 basis function coefficients, a 44× reduction. The use of basis functions allows inner products of residual vectors to be computed as summations over basis functions rather than summations over grid points, and dramatically reduces the

number of unknowns in the required solution of a system of linear equations. The fact that the solution with 'Symmetry' is roughly twice as fast as the 'No Symmetry' case reflects the fact that, in the 'No Symmetry' case, the cost of these update operations has become comparable to the cost of the solution of the MDE, though the cost of the update operations becomes negligible for high symmetry (e.g., cubic) phases when symmetry is imposed. While these observations suggest that a further speed-up is possible by implementing these symmetry constraints within the GPU code, we do point out that the effectiveness of the GPU acceleration will be decreased concomitantly with the reduction in the number of equations to solve. Thus the speed-up is not multiplicative.

To ensure the accuracy of our calculations, we examined the resulting volume fraction profile for each block in the converged solution as well as the final free energies. Density profiles were examined using the visualizer developed for [14], which is also available online [15]. In all cases, the converged solutions were found to yield the expected morphology. The free energies obtained from the CPU and GPU codes were very similar, with the largest fractional error in free energy for both double precision and single precision GPU calculations with respect to the 'Symmetry' calculation being an order of magnitude less than the convergence criteria described earlier.

For cases where speed is the key criterion, our results indicate that single precision GPU calculations are preferred. However, there are situations where higher accuracy may be required, for example when comparing different Frank-Kasper phases with nearly degenerate free energies [23]. Delaney and Fredrickson [52] proposed that the solution generated by a single precision GPU calculation could be used as the initial condition for a subsequent double precision calculation to provide the desired accuracy while retaining the speed of the original GPU calculation during the early stages. We tested this proposition by first computing a single precision GPU calculation to the standard tolerance of $10^{-5}$ for our work, and then using that solution with a double precision GPU calculation to reach a tolerance of $10^{-6}$. The timing results of this calculation are the 'Refine' method in Fig. 2. We then repeated the calculation by performing it entirely in double precision, which is denoted as the 'Direct' method in Fig. 2. Both methods lead to similar run-times. Note that the comparison in Fig. 2 uses identical Anderson mixing parameters when switching from single precision to double precision, and none of the history information from the single precision calculation is transferred to the double precision calculation. Some additional speed-up of the 'Refine' calculation might be possible by additional optimization of the algorithm.

## 4. Conclusion

This paper provides a brief presentation of a new open-source implementation of SCFT with Anderson mixing using GPU acceleration. The present contribution focuses on using this code to compute three-dimensional, periodic morphologies in diblock copolymer melts. The speed-up of the GPU code is found to increase with the size of the problem, which is important for investigating the properties of complex phases in

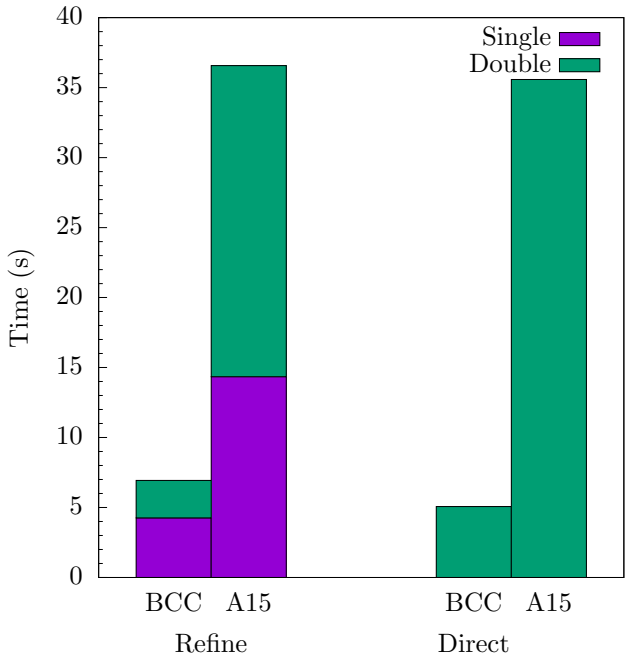


**Figure 2.** Execution time for the SCFT solution for the BCC and A15 phase with a convergence criteria of $10^{-6}$. The set of histograms labeled 'Refine' is the time taken to run a single precision solver to an accuracy of $10^{-5}$ and then refine that solution to an accuracy of $10^{-6}$ using the double precision solver. The set labeled 'Direct' is the time taken using a double precision solver from the outset.

block polymers [21, 23, 24, 37–50]. Eventually, the GPU acceleration will saturate with problem size, but none of the problems we have explored thus far have reached such a large system.

While we have focused here on SCFT computations of bulk phases of block copolymer melts, the present GPU-accelerated implementation can be modified in a straightforward manner to address other problems of interest. The simplest case is employing it as the compute engine within an algorithm for inverse design of bulk phases, such as particle-swarm optimization [11]. By adding appropriate non-periodic boundary conditions [62, 63] the code could be used to compute pattern formation in thin films. Similar to the case for bulk phases, such a code could also be used as the compute engine for inverse design of such patterns [7–9, 12, 13]. Likewise, modifying the code to self-consistently compute the pressure field would allow it to serve as the inner loop for Monte Carlo Field Theoretic Simulations [64, 65]. In all of these cases, the key computational bottleneck is obtaining a fast self-consistent field solver. We are optimistic that all of these applications will be enabled by the availability of this open-source, GPU-accelerated SCFT code.

This GPU code is being distributed as part of a new C++/CUDA version of PSCF that also includes several C++ codes. This rewritten package provides a common framework for implementation of different SCFT solvers for different boundary conditions, algorithms, geometries, and choices of hardware, based on a common set

of classes for shared aspects of SCFT. All programs in the package allow simulation of systems containing arbitrarily complicated acyclic branched block copolymers (e.g., star copolymers) in addition to the linear block copolymers and homopolymers allowed in the older Fortran code. The package currently contains a specialized CPU-based solver for one-dimensional problems and CPU-based programs for problems with periodic boundary conditions in 1, 2 and 3 dimensions in addition to the GPU-accelerated code for periodic problems described here. The one-dimensional solver allows the use of Cartesian, cylindrical or spherical coordinates, and was designed to allow the treatment of flat or curved interfaces and spherical or cylindrical micelles. The CPU and GPU codes for periodic boundary structures both use algorithms analogous to those used in the original Fortran PSCF program. These new CPU and GPU codes both provide efficient algorithms to optimize unit cell parameters so as to minimize the free energy. The new C++ CPU code, like the Fortran code, also allows the user to constrain the solution to have a specified space group symmetry. The C++ CPU code for periodic boundary conditions is intended to eventually supplant the Fortran code, but is still missing a few features of the Fortran code. Currently, most important of such missing features are an algorithm to efficiently continue solutions along a line in parameter space (the SWEEP command of the Fortran program), and a specialized code for point-particle solvents. The GPU code also does not yet allow the user to explicitly constrain the space group symmetry during iteration. The source code of the new package in its present form is available online [56].

## Acknowledgments

This work was support by the National Science Foundation DMR-1725272. Computational resources were provided in part by the Minnesota Supercomputing Institute, which receives capital equipment funding from the NSF through the UMN MRSEC program (Grant No. DMR-1420013).

## Author contribution statement

GKC, DCM and KDD designed the research; DCM wrote most of the C++ codes; GKC and AC wrote and tested the GPU codes; KDD and DCM supervised the research; all authors analyzed the results; KDD and GKC wrote the manuscript with input from AC and DCM.

## References

[1] Fredrickson G H 2006 *The Equilibrium Theory of Inhomogenous Polymers* (Oxford Science Publication)
[2] Bates F S and Fredrickson G H 1999 *Phys. Today* **52** 32–36
[3] Bates F S, Hillmyer M A, Lodge T P, Bates C M, Delaney K T and Fredrickson G H 2012 *Science* **336** 434–440

[4] Bates C M and Bates F S 2017 *Macromolecules* **50** 3–22
[5] Miskin M Z, Khaira G, de Pablo J J and Jaeger H M 2016 *Proc. Natl. Acad. Sci. USA* **113** 34–39
[6] Gadelrab K R, Hannon A F, Ross C A and Alexander-Katz A 2017 *Mol. Sys. Des. Eng.* **2** 539–548
[7] Hannon A F, Gotrik K W, Ross C A and Alexander-Katz A 2013 *ACS Macro Lett.* **2** 251–255
[8] Hannon A F, Ding Y, Bai W, Ross C A and Alexander-Katz A 2014 *Nano Lett.* **14** 318–325
[9] Paradiso S P, Delaney K T and Fredrickson G H 2016 *ACS Macro Lett.* **5** 972–976
[10] Tsai C L, Delaney K T and Fredrickson G H 2016 *Macromolecules* **49** 6558–6567
[11] Khadilkar M R, Paradiso S, Delaney K T and Fredrickson G H 2017 *Macromolecules* **50** 6702–6709
[12] Ouaknin G Y, Laachi N, Delaney K T, Fredrickson G H and Gibou F 2018 *J. Comput. Phys.* **375** 1159–1178
[13] Zhang R, Zhang L, Lin J and Lin S 2019 *Phys. Chem. Chem. Phys.* **21** 7781–7788
[14] Arora A, Qin J, Morse D C, Delaney K T, Fredrickson G H, Bates F S and Dorfman K D 2016 *Macromolecules* **49** 4675–4690
[15] http://pscf.cems.umn.edu
[16] Tyler C, Qin J, Bates F and Morse D C 2007 *Macromolecules* **40** 4654–4668
[17] Ranjan A, Qin J and Morse D 2008 *Macromolecules* **41** 942–954
[18] Qin J, Bates F and Morse D 2010 *Macromolecules* **43** 5128–5136
[19] Glaser J, Medapuram P, Beardsley T, Matsen M and Morse D 2014 *Physical Review Letters* **113** 068302
[20] Medapuram P, Glaser J, Beardsley T, Matsen M and Morse D 201 *Macromolecules* **48** 819–839
[21] Chanpuriya S, Kim K, Zhang J, Lee S, Arora A, Dorfman K D, Delaney K T, Fredrickson G H and Bates F S 2016 *ACS Nano* **10** 4961–4972
[22] Radlauer M R, Sinturel C, Asai Y, Arora A, Bates F S, Dorfman K D and Hillmyer M A 2017 *Macromolecules* **50** 446–458
[23] Kim K, Schulze M W, Arora A, III R M L, Hillmyer M A, Dorfman K D and Bates F S 2017 *Science* **356** 520–523
[24] Kim K, Arora A, III R M L, Liu M, Li W, Shi A C, Dorfman K D and Bates F S 2018 *Proc. Natl. Acad. Sci. USA* **115** 847–854
[25] Burke C J and Grason G M 2018 *J. Chem. Phys.* **148** 174905
[26] Burns A B and Register R A 2916 *Macromolecules* **50** 8106–8116
[27] Prasad I, Seo Y, Hall L M and Grason G M 2017 *Phys. Rev. Lett.* **118** 247801
[28] Prasad I, Jinnai H, Ho R M, Thomas E L and Grason G M 2018 *Soft Matter* **14** 3612–3623
[29] Dane C, Register R A and Priestley R D 2018 *Phys. Rev. Lett.* **121** 247801
[30] Christie D, Register R A and Priestley R D 2018 *ACS Cent. Sci.* **4** 504–511
[31] Yadav M, Bates F and Morse D 2018 *Physical Review Letters* **121** 127802
[32] Burns A B, Christie D, Mulhearn W D and Register R A 2019 *J. Polym. Sci. B Polym. Phys.* **57** 932–940
[33] Yadav M, Bates F and Morse D 2019 *Macromolecules* **52** 4091–4102
[34] Feng X, Burke C J, Zhuo M, Guo H, Yang K, Reddy A, Prasad I, Ho R M, Avgeropoulos A, Grason G M and Thomas E L 2019 *Nature* **575** 175–179
[35] Frank F C and Kasper J S 1958 *Acta Cryst.* **11** 184–190
[36] Frank F C and Kasper J S 1959 *Acta Cryst.* **12** 483–499
[37] Lee S, Bluemle M J and Bates F S 2010 *Science* **330** 349–353
[38] Lee S, Leighton C and Bates F S 2014 *Proc. Natl. Acad. Sci. USA* **111** 17723–17731
[39] Xie N, Li W, Qiu F and Shi A C 2014 *ACS Macro Lett.* **3** 906–910
[40] Gillard T M, Lee S and Bates F S 2016 *Proc. Natl. Acad. Sci. USA* **113** 5167–5172
[41] Liu M, Li W, Qiu F and Shi A C 2017 *Soft Matter* **12** 6412–6421
[42] Liu M, Qiang Y, Li W, Qiu F and Shi A C 2016 *ACS Macro Lett.* **5** 1167–1171
[43] Schulze M W, Lewis III R M, Lettow J H, Hickey R J, Gillard T M, Hillmyer M A and Bates F S 2017 *Phys. Rev. Lett.* **118** 207801
[44] Takagi H, Hashimoto R, Igarashi N, Kishimoto S and Yamamoto K 2017 *J. Phys. Condens. Matter*

**29** 204002
[45] Lewis III R M, Arora A, Beech H K, Lee B, Lindsay A P, Lodge T P, Dorfman K D and Bates F S 2018 *Phys. Rev. Lett.* **121** 208002
[46] Reddy A, Buckley M B, Arora A, Bates F S, Dorfman K D and Grason G M 2018 *Proc. Natl. Acad. Sci. USA* **115** 10233–10238
[47] Bates M W, Lequieu J, Barbon S M, Lewis III R M, Delaney K T, Anastasaki A, Hawker C J, Fredrickson G H and Bates C M 2019 *Proc. Natl. Acad. Sci. USA* **116** 13194–13199
[48] Jeon S, Jun T, Jo S, Ahn H, Lee S, Lee B and Ryu D Y 2019 *Macromol. Rapid Commun.* 1900259
[49] Takagi H and Yamamoto K 2019 *Macromolecules* **52** 2007–2014
[50] Zhao M and Li W 2019 *Macromolecules* **52** 1832–1842
[51] Arora A, Morse D C, Bates F S and Dorfman K D 2017 *J. Chem. Phys.* **146** 244902
[52] Delaney K T and Fredrickson G H 2013 *Comput. Phys. Commun.* **184** 2102–2110
[53] Matsen M W 2009 *Eur. Phys. J. E* **30** 361–369
[54] Stasiak P and Matsen M W 2011 *Eur. Phys. J. E* **34** 110
[55] Tyler C A and Morse D C 2003 *Macromolecules* **36** 8184–8188
[56] Repository https://github.com/dmorse/pscfpp. Test reported here use tagged version v0.7.
[57] Fredrickson G H, Ganesan V and Drolet F 2002 *Macromolecules* **35** 16–39
[58] Ceniceros H D and Fredrickson G H 2004 *Multiscale Model Simul.* **2** 452–474
[59] Audus D J, Delaney K T, Ceniceros H D and Fredrickson G H 2013 *Macromolecules* **46** 8383–8391
[60] Thompson R B, Rasmussen K O and Lookman T 2004 *J. Chem. Phys.* **120** 31–34
[61] Dutour Sikirić M, Delgado-Friedrichs O and Deza M 2010 *Acta Crystallogr. Sect. A Found. Crystallogr.* **66** 602–615
[62] Matsen M W 1997 *J. Chem. Phys.* **106** 7781–7791
[63] Bosse A W, García-Cervera C J and Fredrickson G H 2007 *Macromolecules* **40** 9570–9581
[64] Stasiak P and Matsen M W 2013 *Macromolecules* **46** 8037–8045
[65] Vorselaars B, Stasiak P and Matsen M W 2015 *Macromolecules* **48** 9071–9080