Colloid Polym Sci (2008) 286:139–148
DOI 10.1007/s00396-007-1714-9

ORIGINAL CONTRIBUTION

Parallel computational techniques for the analysis of sedimentation velocity experiments in UltraScan

Emre Brookes · Borries Demeler

Received: 27 March 2007 / Revised: 20 May 2007 / Accepted: 5 June 2007 / Published online: 11 July 2007

Abstract The advent of parallel computing technology and low-cost computing hardware has facilitated the adoption of high-performance computing tools for the analysis of sedimentation data. Over the past 15 years, we have developed the UltraScan software (Demeler et al., http://ultrascan.uthscsa.edu) to support sedimentation analysis, experimental design, and data management. We describe here recent extensions and advances in methodology that have been adapted in UltraScan. High-performance computing methods implemented on parallel supercomputers utilizing grid computing technology are used to analyze sedimentation experiments at much higher resolution than was previously possible. We discuss the implementation of parallel computing in three novel algorithms used in UltraScan for modeling of sedimentation velocity experiments and provide guidelines for effective data analysis.

Keywords Two-dimensional spectrum analysis · Genetic algorithms · Monte Carlo · MPI

Introduction

Analytical ultracentrifugation (AUC) has long played a central role in the characterization of solutions containing colloidal suspensions and biological and synthetic polymers. The ability to separate individual components based on shape and mass by performing sedimentation experiments has provided significant insights into macromolecular solutions. In AUC experiments, transport due to sedimentation and diffusion is observed, and from these observations, mass and macromolecular shape parameters can be determined. We describe here recent advances in the methodology that extend significantly the resolution and detail that can be determined. These advances address a major difficulty: the convergence of optimization algorithms used for fitting experimental data. Especially for cases where heterogeneity is involved, accurate description of all of the components is a significant challenge, and existing methods for parameter estimation encounter a number of obstacles that are difficult to remove. Additional complexity is encountered when the heterogeneity is not only present in the mass distribution but also in the macromolecular shape. The reason for the difficulty to describe such heterogeneity is related to the complexity of the error surface during parameter optimization and the presence of noise in the data. We show that implementation of high-performance computing technology in conjunction with our new optimization methods can successfully address these challenges and provide superior results.

Computer performance has been increasing in accord with Moore's law [1], while simultaneously, the cost of computer hardware has been decreasing. Combined with the availability of modern computer languages and software libraries, these factors have made high-performance computing accessible to researchers. Computationally complex algorithms that were merely of theoretical interest just a few years ago can now be routinely implemented on ordinary computer hardware. In addition to the availability of faster computing hardware, the techniques of high-performance

E. Brookes
Department of Computer Science,
The University of Texas at San Antonio, One UTSA Circle,
San Antonio, TX 78249-1644, USA

B. Demeler (✉)
Department of Biochemistry, University of Texas Health Science Center at San Antonio,
7703 Floyd Curl Drive, MC 7760,
San Antonio, TX 78229-3901, USA
e-mail: demeler@biochem.uthscsa.edu

computing enable the implementation of parallel algorithms to accelerate demanding numerical calculations. Instead of running an algorithm on a single processor, new architectures allow the same algorithm to be calculated on multiple processors simultaneously. With careful design of a parallel algorithm, significant speed increases can be obtained. In this work, we describe how these advances in parallel computing technology can be appropriated by the field of AUC for the analysis of sedimentation experiments. The increased computational power afforded by parallel computing allows us to analyze sedimentation data at much higher resolution than is possible with traditional methods, without sacrificing computational speed.

Before our work, no attempts were made to parallelize algorithms used in the analysis of AUC experiments. To this end, we have developed three new parallel algorithms for AUC analysis, the 2-Dimensional Spectrum Analysis (2DSA) [2], the Genetic Algorithm for AUC analysis (GA) [3], and Monte Carlo (MC) versions of 2DSA and GA (2DSA-MC and GA-MC). These methods allow advanced analysis of the experimental data providing results that were not possible to attain before the advent of high-performance parallel computing.

In this paper, we will present a brief background on parallel processing, an overview of our analysis methods, a description of their parallel implementation, and a description of the methodology of sequentially executing the three methods to provide high-resolution results. Running jobs on high-performance parallel computing resources is historically a cumbersome process, requiring esoteric command-line batch packaging and submission of jobs. To address this issue, we have developed a convenient web interface to allow the researcher to easily process experiments from a web browser and obtain results in a format compatible with the UltraScan package.

Parallel processing background

A modern computer generally contains one or more main processors. Today's high-performance computer system typically consists of a *cluster* of computers connected with a fast communications network. Each processor is capable of executing the steps of an algorithm represented as a serial stream of instructions. To execute an algorithm on multiple processors in parallel requires that a set of instructions is made available to each processor and that some method of communication between the processors exists. A processor can be in one of three states: *computing*, *communicating*, or *idle*. During computing the processor is executing the algorithm, during communicating the processor is busy sending or receiving messages from other processors, and during idle cycles the processor is waiting for communications or has completed all work on the algorithm. If a processor must communicate intermediate results to other processors during the execution of a parallel algorithm, a processor waiting for results may sit idle. The balance of time spent among the three states of a group of processors is of critical importance to the design of parallel algorithms. Algorithms that spend a large percentage of time computing between communications are called *coarse-grained*, and those that must communicate frequently are termed *fine-grained*. Algorithms that run on multiple processors with very little or no communication are called *embarrassingly parallel*. The *efficiency* of a parallel algorithm is the time spent computing divided by the total time (communicating + computing + idle). Coarse-grained or embarrassingly parallel algorithms often exhibit a high efficiency. The efficiency of a parallel algorithm may change based on the size of the problem and number of processors used. A parallel algorithm that maintains a high efficiency with increasing numbers of processors and problem size is said to *scale efficiently*.

We illustrate the three states of a parallel algorithm by means of a simple example. Calculation of the sum $\sum_{i=1}^{20} F(x_i)$ could be accomplished by adding the results of two partial sums $\sum_{i=1}^{10} F(x_i)$ and $\sum_{i=11}^{20} F(x_i)$. By calculating each partial sum on a different processor, the time required to calculate the total sum is cut in half. During the calculation, both processors are in the calculation state. To form the total sum, processor 2 needs to communicate the result of its partial sum to processor 1. During this time, both processors are in the communication state. Finally, the value of the two partial sums must be added by processor 1 to its partial sum to obtain the final result. During this time, processor 2 is in the idle state. Clearly, the goal of efficient parallel code design is to minimize the time spent in the communication and idle states. Efficient parallel algorithms have been designed for many problems including matrix–matrix, matrix–vector multiplication, Gaussian elimination, fast Fourier transforms, sorting, and searching [4]. Attempts at automatic parallelization have been made [5], but these tools generally offer much lower performance improvement than tedious hand coding. All our parallelization in UltraScan is hand coded.

Executing jobs on high-performance parallel systems is different than running a program on a desktop computer. Parallel machines are generally dedicated to a single process at a time, and execution is controlled through a *queue* mechanism, where requests are added to the queue and released from the queue when sufficient *resources* are available to execute the process. A compute request generally includes a list of resources required to execute the process, such as required memory, diskspace, and number of processors. Based on this information, the process

controlling queue will allocate resources and schedule job execution.

Analysis methods

The 2DSA, 2DSA-MC, GA, and GA-MC algorithms are all methods for determining the composition of a heterogeneous mixture of a colloidal or polymer solution. Parameters of interest include the number of different solutes present in the solution, each solute's partial concentration, as well as the shape and the molecular weight of each solute. This information can be extracted from AUC sedimentation velocity experimental data. The underlying method in each approach consists of determining some likely set of solutes G, solving the Lamm equation [6] for each solute, and using the nonnegative least squares [7] algorithm to reduce G to a set of contributing solutes G' (see Figs. 1 and 2). The goodness of fit of G' to the experimental data is measured by the root mean square deviation (RMSD). Each element g of the set G is a pair of parameters, a sedimentation coefficient s and a frictional ratio f/f_0. Equivalently, each element g of G is a point in the s, f/f_0 plane. If s and f/f_0 of a solute are known, the molecular weight of the solute can be computed. Previous methods could only estimate s and at best report an average f/f_0, but reliable molecular weights were not available.

2-Dimensional Spectrum Analysis In 2DSA, we place a two-dimensional grid on the s, f/f_0 plane (see Fig. 1). We constrain values of s with the enhanced van Holde–Weischet analysis [8]. f/f_0 is generally constrained to values between 1 (spherical) and 4 (rod-shape). In the serial implementation of the 2DSA algorithm, a uniform grid is placed within the constrained range of the s, f/f_0 plane. These grid-points define G from which G' can be computed. The serial algorithm faces a significant disadvantage: To assure that each solute g is correctly identified, the grid needs to be aligned with the parameters of the actual solute present in solution. This in turn requires a high-resolution grid, whose computation demands significant workspace memory. The parallel 2DSA algorithm solves this problem by dividing the high-resolution grid into multiple lower-resolution grids, which can be combined to form the high-resolution grid. The low-resolution grids are produced by moving an initial low-resolution grid incrementally in the s and f/f_0 directions until all desired grid points have been covered by grids $\{G_1, G_2,..., G_n\}$ (see Fig. 3). Each grid $\{G', G_1', G_2',..., G_n'\}$ can be independently computed on multiple processors. Next, we let G^1 be the union of $\{G', G_1', G_2',..., G_n'\}$ and compute $G^{1\prime}$ on one processor (see Fig. 4). This grid-level parallelization of the 2DSA approach represents a very efficient distribution of the calculation load. In general, G^1 may also be too large to compute on one processor, so we apply the method recursively in the multistage 2DSA method, which is shown in Fig. 5. For the highest quality result, result $G^{1\prime}$ can further be unioned back into each of G, G_1, G_2,..., G_n in an iterative method as shown in Fig. 6 where this process is repeated until no further change is observed, and the solution is converged on a stable grid solution. The final result from a 2DSA analysis is denoted by G_{2DSA}'. Parallelization of 2DSA is considered relatively

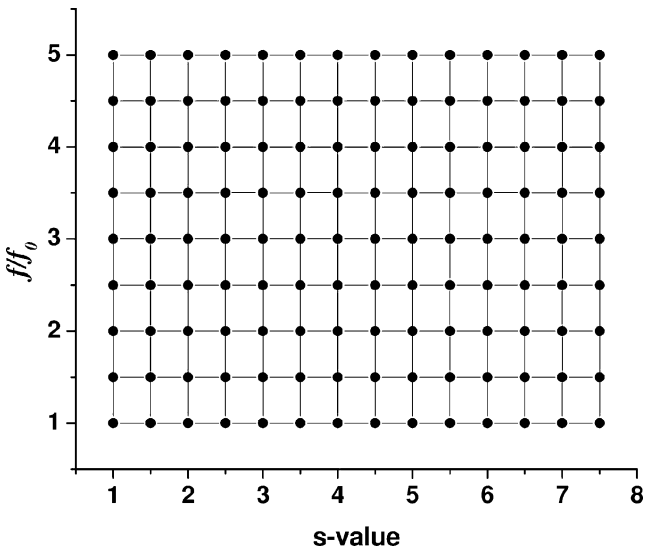

Fig. 1 An example of a two-dimensional grid used in 2DSA. This is a representation for the set G, a set of points representing likely solutes. G is evaluated for fitness to the experimental data, and the points contributing to the best fit solution are collected in G', a subset of G as shown in Fig. 2

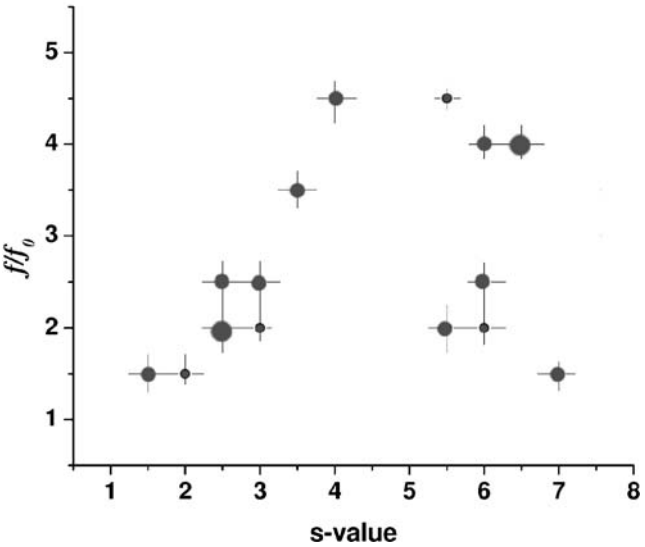

Fig. 2 The results of a 2DSA analysis using the set G of Fig. 1. This is a representation for the set G_{2DSA}', the best contributing components from a set G. The thickness of each point is proportional to the relative concentration of each component of G'

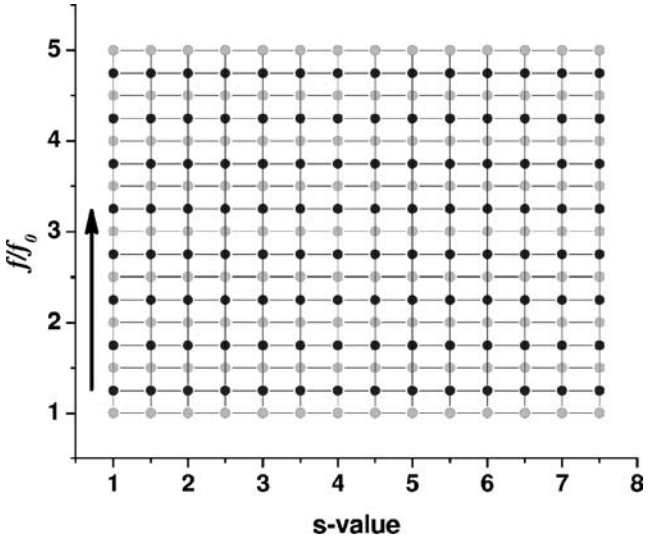

Fig. 3 The movement of a 2DSA grid G (see Fig. 1) to produce G_1. The *gray points* are the elements of G, and the *black points* are the elements of G_1. The parallel 2DSA algorithm will evaluate G_1' on a separate processor than G'

coarse grained and has been shown to scale efficiently on up to 512 processors [2].

Genetic Algorithm Unlike the single G as used in 2DSA, a genetic algorithm [9] contains a collection of sets G termed a *population* of *individuals*, where each individual represents a group of solutes, which approximate the entire colloidal sample. The *evolution* of genetic algorithm proceeds as follows: A population of individuals is randomly initialized. Then, a *generational loop* is run in which individuals are randomly selected for reproduction based upon *fitness*, and then the individuals are *recombined* or *mutated* to produce a new population in the next generation. The generational loop repeats for some predetermined number of *generations*. An individual's fitness is determined by the RMSD of G'. If *parsimonious regularization* (a method to find the set G with the fewest number of elements with a comparable RMSD [10]) is used, the fitness of the individual is penalized in relation to the number of elements in G. To accelerate convergence of the optimization, proper initialization of each G in the population is critical. We have found that excellent initialization results can be obtained by preprocessing the data with the 2DSA method. We use the 2DSA output $\left(G'_{\mathrm{2DSA}}\right)$ and draw a rectangular region in the s, f/f_0 plane around each element of $\left(G'_{\mathrm{2DSA}}\right)$ (see Fig. 7). This region is termed a *bucket*. We require the buckets to be nonoverlapping. If buckets overlap, we subdivide each overlapping bucket into multiple nonoverlapping buckets covering the same region. For a typical GA analysis, we start with a population of approximately 100 individuals G, each containing elements g that have been initialized with a random s, f/f_0 value drawn from each bucket identified above (see Fig. 8). Therefore, if the 2DSA method resulted in 30 individuals $\left(G'_{\mathrm{2DSA}}\right)$, our initialization will identify 30 buckets (or more, if subdivision is required due to overlap), and each of the 100 individuals G contains 30 or more elements g. When an individual G is randomly selected to mutate during the generational loop, an element of an individual G is randomly changed, but it will remain constrained by the range of the bucket associated

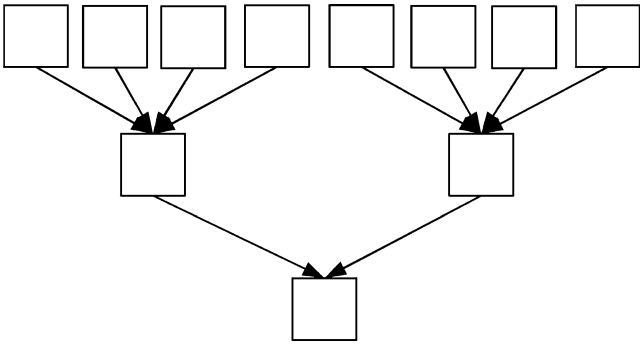

Fig. 5 The 2DSA multistage method. In this case, the size of the unions of the results from the *first row* (see Fig. 4) are too large to compute on a single processor and must be computed in groups

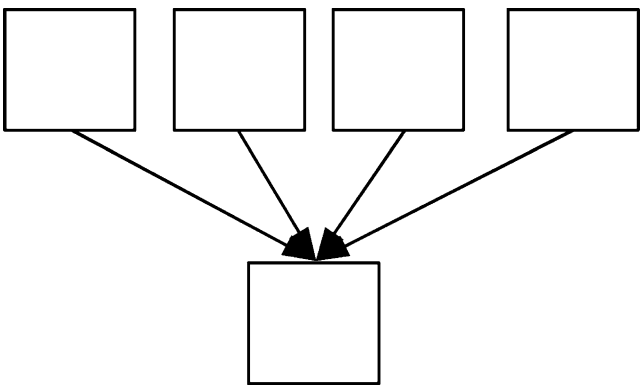

Fig. 4 The *first row of boxes* contains the sets G, G_1, G_2, and G_3. The *bottom box* contains G^1, the union of the sets G', G_1', G_2', and G_3'. The final result is $G^{1\prime}$. The first row is computed on different processors, but the final result is computed on one processor

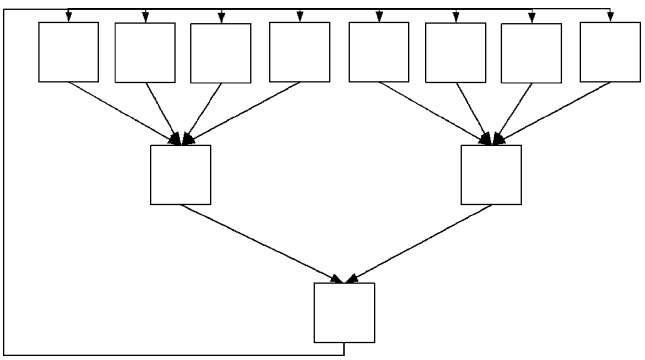

Fig. 6 The 2DSA iterative multistage method. In this case, the results of each multistage solution are unioned back into the initial sets G, G_1, G_2,...G_n, and the process is repeated until the final result is unchanging. This achieves the highest quality result

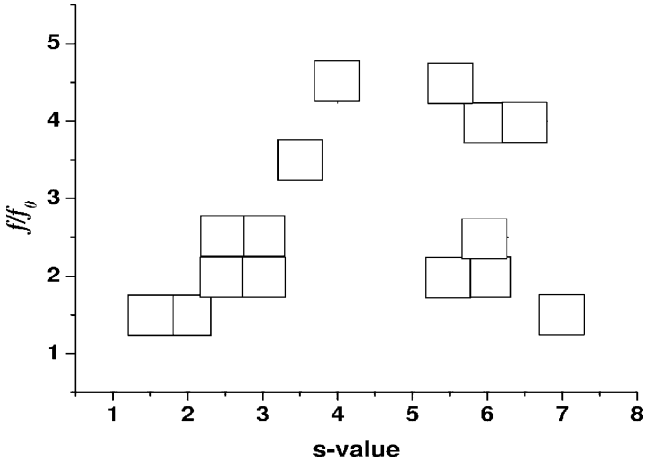

Fig. 7 The buckets produced around a 2DSA analysis for initialization of the *GA*. Each bucket is centered around an element of the 2DSA solution G'_{2DSA}

with its creation. Furthermore, during recombination, when two individuals are selected, the recombination will always be between associated buckets.

GA optimization involves the creation of multiple populations, termed demes (see Fig. 9). The purpose for maintaining multiple demes can best be understood by analogy with biological populations: A species derives benefit from genetic diversity, which is maintained by keeping subpopulations in geographically diverse regions. Parallelization is achieved by calculating each deme independently on a different processor. Each deme runs its own generational loop. When evaluating multiple demes, we also need to consider exchange of information between demes. Individuals within the deme are randomly selected for emigration to another deme with a user-selected migration rate probability. Migration occurs during the generational

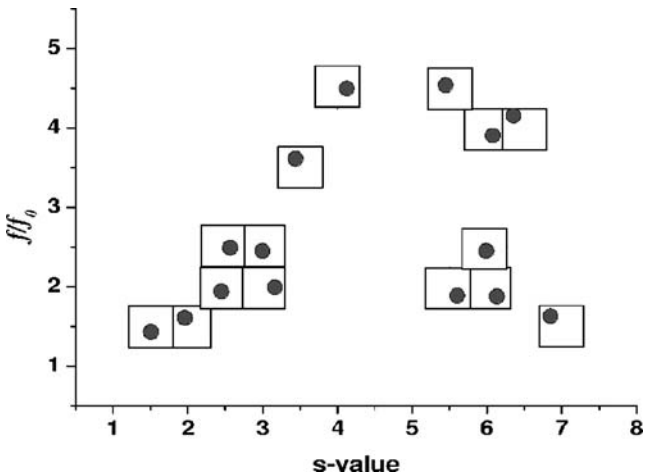

Fig. 8 An individual set *G* in the *GA* population. Each bucket (see Fig. 7) contains a set element constrained by the bucket. Different individuals in the *GA* population will have different elements, but they will all be constrained by the buckets, and exactly one element of set *G* will be in each bucket

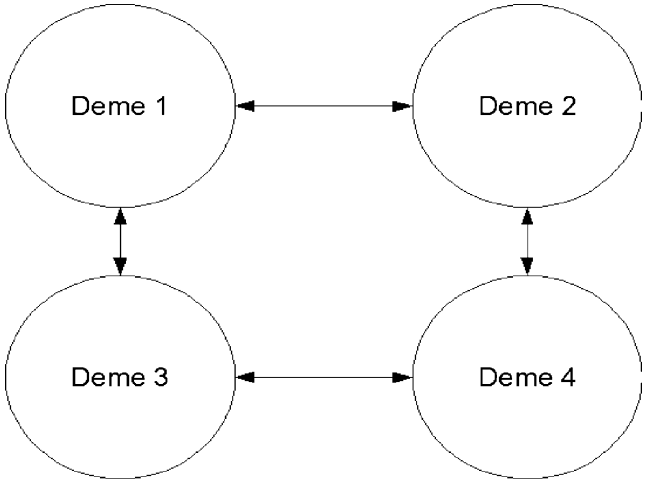

Fig. 9 A representation of four demes in a parallel *GA* job. Each deme contains its own population, a collection of individuals *G*, and runs on a separate processor. During the evolution of the *GA*, migration occurs between connected demes as represented by the *double arrows*. Migration is probabilistic and is controlled by the *GA* migration rate

loop. At the end of the generational loop, the emigrants are communicated to a master process, which stores all emigrants. At the beginning of the generational loop, the master process is asked for the list of any immigrants. By brokering emigrants through the master process, all deme calculations can proceed asynchronously without the need to wait for the completion of another deme. This eliminates the requirement for keeping all demes at the same generation, and any deme can receive immigrants from any other deme at earlier or later generations without penalty. Running multiple demes implies a migration topology, where migration can only occur between certain processors. In our GA, we run a bidirectional ring topology, where each deme's individuals can migrate between two neighbors. GA deme parallelization is embarrassingly parallel and scales efficiently. The final result for a GA is denoted G'_{GA}.

Monte Carlo analysis The MC method is used to amplify true signal away from background noise and to provide parameter value distributions, which can be used to evaluate the confidence intervals of each parameter. Our MC approach can be applied to either 2DSA or GA optimization results. The MC process is started by performing a 2DSA or GA optimization, resulting in a best fit G'. Visual inspection of residual bitmaps and run patterns should confirm that only random deviations are present in the residuals before any MC analysis is attempted.

MC is performed by creating a vector of absolute values of the residuals. Residuals are calculated from the difference between the G' model and the experimental data and represent an estimate of the magnitude of random noise present at each point in the experimental data. This vector is then slightly smoothed with a Gaussian kernel. Next, for

each vector element, we define a new, randomly generated residual with a zero mean and a standard deviation equivalent to the vector entry at each point. The new randomly generated residuals are added to the corresponding points in the best fit G' solution from either 2DSA or GA, generating a new and equivalent pseudo-experimental dataset. During parallel execution of MC, the pseudo-experimental data are produced by a master process and communicated to all other processors. Each pseudo-experimental dataset is refitted with the 2DSA or GA optimization method, producing a series of new G'_{MC1}, G'_{MC2}, …, G'_{MCn}. The final result G'_{MC} is the union of G' and the results of every iteration of the MC. The MC methods scale efficiently. More detail on the MC methods is available in [11].

Implementation

Our parallel analysis methods require high-performance computing resources. These resources consist of multiple clusters and a software module that submits analyses to these resources. Communication between processors within a cluster is accomplished with the MPI (MPI Forum, http://www.mpiforum.org) message passing interface library. We have developed a convenient web interface, available to the public, to submit analyses to both our local cluster and multiple remote clusters. All clusters addressable by UltraScan are part of the Texas Internet Grid for Research and Education (TIGRE; http://www.hipcat.net/projects/tigre). TIGRE is a large grid computing infrastructure developed by the consortium for High-Performance Computing in Texas (HiPCaT: Consortium for High-Performance Computing across Texas, http://www.hipcat.net). The TIGRE infrastructure includes a Globus Toolkit (http://www.globus.org) based software stack, which provides grid computing middleware. This middleware facilitates communication and data exchange among all clusters within TIGRE, maximizes load balancing, and permits sharing of resources. Our software uses it to submit jobs in a uniform way to multiple remote clusters. In this section, we will describe how the analysis requests are processed.

Before analysis, all experimental data are edited and committed to the UltraScan Laboratory Information Management System (LIMS) database as explained in [12]. The researcher begins by logging into the web interface, selecting the experiments to analyze as well as the analysis method (2DSA or GA). At this stage, all required analysis parameters are entered via a web form. If MC analysis is desired, the number of MC iterations is entered as well. Then, the job is submitted to one of the clusters by the researcher. Results compatible with UltraScan are e-mailed to the researcher upon job completion. The user imports the e-mailed results into the UltraScan software where they can be visualized in 3D by a C++ graphical user interface module. There are no special requirements for the user's computer. Any PC with access to Internet e-mail and web browsing that is capable of running UltraScan locally can process jobs on our high-performance clusters and then view the results locally. UltraScan versions for all major operating systems and hardware platforms are available for free download from our web site (Demeler et al., http://ultrascan.uthscsa.edu).

The mechanism that enables this apparent simplicity is shown in Fig. 10. When the user submits a job, the web server sends the user's request to us_gridpipe, which is a *named pipe*. This is a special type of file that simply holds written data until they are read. The PERL [13] script us_gridpipe.pl *daemon*, a program that is always running, reads from us_gridpipe and manages the queue for local MPI jobs and controls startup of TIGRE jobs. Upon receipt of the researcher's job request, the us_gridpipe.pl daemon will first execute us_gridcontrol to collect the experimental data from the LIMS database in preparation for job execution. When us_gridcontrol completes, it informs the us_gridpipe.pl daemon via the named pipe that all of the experimental data have been extracted from the database and placed into a file on the disk. The us_gridpipe.pl daemon then determines from the request if the process should be executed locally via MPI or submitted to remote TIGRE resources. For MPI jobs, the job is placed in an MPI job queue maintained by us_gridpipe.pl. If only a single request is waiting in the job queue (the one just added), the MPI job is directly executed on the local cluster using all processors specified for either a 2DSA or GA analysis in a machine file. For 2DSA jobs, the number of grid repetitions is split among the available processors, and for GA jobs, the number of processors required equals the number of demes+1. When the MPI job completes, results are e-mailed to the researcher, and the us_gridpipe.pl daemon is informed that the job has completed so that it can remove the job from its MPI job queue and start the next MPI job if another job is in the queue. For TIGRE jobs, us_gridpipe.pl places the job request into a list of TIGRE jobs and begins execution of the PERL script us_tigre_job.pl, which controls TIGRE job execution. TIGRE resources are shared, and it is important to select the number of processors carefully. The authors have developed a formula to compute the optimal number of processors to achieve a specific processor utilization [2], and this computation is performed for TIGRE jobs. Once the number of processors is known, us_tigre_job.pl sends the experimental data and user's web request parameters to the user-selected cluster and submits the job to the appropriate queue. The TIGRE job is monitored until completion. Upon completion, us_tigre_job.pl retrieves the result data from the target cluster, sends the results e-mail to

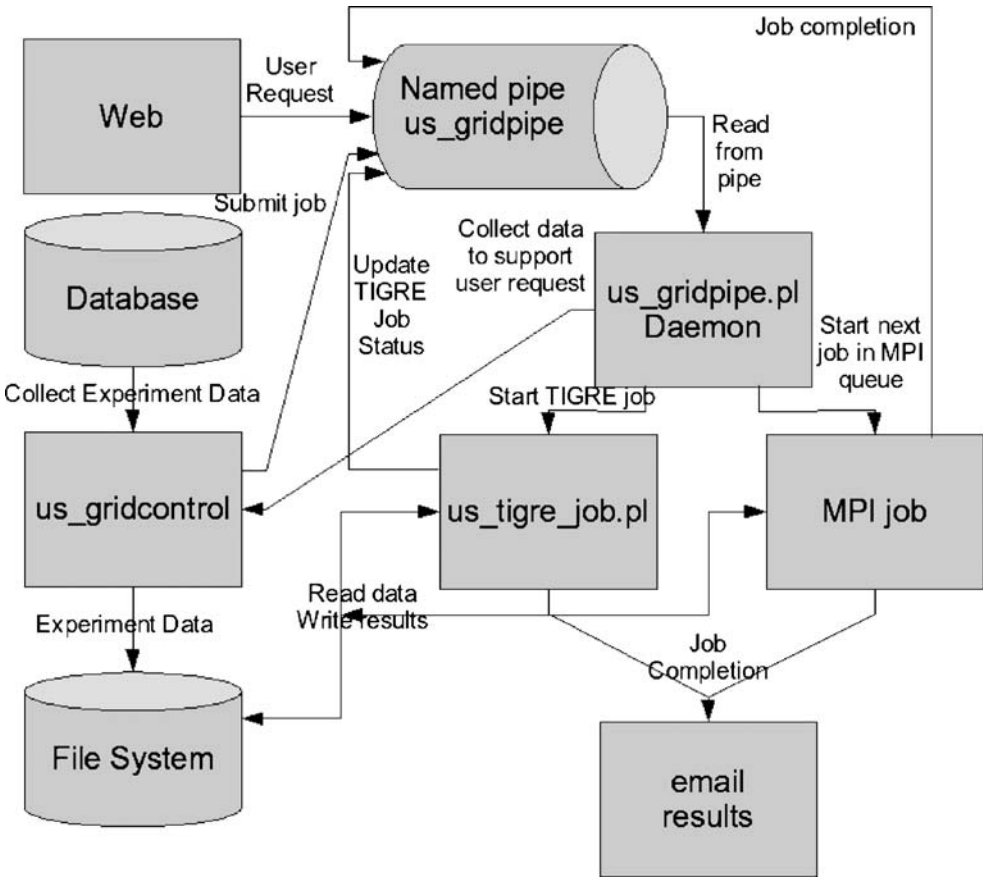

Fig. 10 A flow diagram of mechanism behind submission of parallel analysis jobs through the web interface. Job submission starts with a request entered from the web, and it is eventually passed to us_gridpipe.pl, which manages the job execution by requesting us_gridcontrol to collect the experimental data, and then submits the job to a local MPI queue or a remote TIGRE queue (see text). Upon completion, results are emailed

the researcher, and informs the us_gridpipe.pl daemon that the TIGRE job is finished. The us_tigre_job.pl script will also collect all run-time statistics and store them in a database. At this point, the us_gridpipe.pl removes the TIGRE jobs from its TIGRE job list, and us_tigre_job.pl exits. The us_gridpipe.pl daemon also accepts requests to obtain information about its MPI job queue and TIGRE job list, and this is available for viewing directly from the web interface. All required parallelization modules are available for the Linux operating system, and can be downloaded for free from the UltraScan web site (http://www.ultrascan.uthscsa.edu).

Methodology

We have adapted the UltraScan software to allow the user to follow the typical flow of information in the analysis environment. We now describe how the user will interact with the remote computing platform and identify the actions required to improve the data in a step-by-step procedure:

Step 1—LIMS data import The first step after data acquisition is the association of experimental data with related information and storage of these data in the UltraScan LIMS. The LIMS is described in detail in [12]. One purpose of the LIMS is to combine experimental data with other information important for hydrodynamic corrections, such that hydrodynamic corrections can be performed on the fly without the user's intervention. In addition, users can provide experimental designs and protocols, project descriptions, gel images, and absorbance profiles to further specify experimental conditions.

Step 2—editing a binary copy of the data After all data have been entered into the database, a copy of the data is retrieved and edited. This will create a binary copy of the data, which contains all data structures necessary for any UltraScan analysis method. Moreover, these binary data are a self-contained description of the entire experiment suitable for distribution among different compute clusters.

Step 3—setting s-value limits For a velocity experiment, the first method employed is a preliminary analysis with the enhanced van Holde–Weischet method [8]. This approach is used to define a diffusion-corrected $G(s)$ distribution, which provides the sedimentation coefficient limits of the data.

Step 4—single-pass 2DSA with time-invariant noise correction and meniscus fitting After committing the binary copy of the data to the LIMS, the s-value limits are applied

to the 2DSA parameter range and the data can be analyzed with a high-resolution 2DSA pass. Simultaneously, meniscus position and time-invariant noise correction should be selected. We found a 0.01-cm meniscus range with ten increments is generally sufficient to find the most accurate meniscus position. RMSD values from each meniscus iteration can be fitted to a second-order polynomial, and the minimum value will correspond to the best meniscus value. Data analysis results are sent by e-mail to the investigator in the form of a small e-mail attachment (typically 0.3–20 KB, containing the sedimentation and diffusion coefficient, and the partial concentration for each solute determined during analysis, as well as the time and radially invariant noise vectors), who will import the results into the UltraScan software and use the Finite Element Data Viewer within UltraScan to display the results. After inspection of the data, a utility module in UltraScan is used to subtract the time-invariant noise vector from the edited data and to update the meniscus position, if necessary, producing an improved rendition of the data that is now free of scratches and other time-invariant noise contributions, and contains an optimized meniscus position. The improved data are now recommitted to the LIMS.

Step 5—updating the LIMS with the noise-free data After subtraction of the time-invariant noise, the edited dataset in the LIMS can now be replaced with the improved version and henceforth be used for all additional analysis methods. Because the time-invariant noise is eliminated from the dataset, all subsequent analysis methods no longer have to consider time-invariant noise, which improves calculation speed significantly, especially for large datasets. This is especially important for MC analysis methods, which require many iterations of analysis.

Step 6—2DSA-MC analysis At this point, a 100- to 200-iteration 2DSA-MC analysis of the data should be performed. This analysis will amplify actual data signals and minimize the effect of noise contributions in the data. A 3D plot of this analysis often provides a good qualitative view of the data.

Step 7—initialization of the GA The results obtained in the 2DSA-MC analysis can be visualized with the Finite Element Viewer in UltraScan, and the processed distribution file can be imported into the GA initialization routine. Using the bucket selection method explained earlier, likely solute groupings are identified and written to a disk file. This file is uploaded to the web interface of the GA routine and used to constrain the GA. Completion of the GA will result in a parsimonious solute distribution, eliminating all buckets that are not required for a description of the experimental data.

Step 8—performing the GA-MC analysis The final step in the analysis will further refine the parsimonious solution by classifying the contributions of random noise to the identified parameters. The MC analysis will result in confidence limits for each parameter and provide reliable statistics for each value.

Step 9—global analysis Additional improvements can be obtained by combining experimental data from multiple experiments in step 8. Combined experiments should include data from different speeds to maximize the signal from both sedimentation and diffusion coefficients. The LIMS version of UltraScan permits addition of multiple experiments in the analysis queue to achieve a truly global analysis.

Results

To exemplify the importance of our methods, we have simulated a four-solute system containing time-invariant and random noise (Solute 1: MW=25 kDa, f/f_0=1.2; Solute 2: MW=75 kDa, f/f_0=1.6; Solute 3: MW=150 kDa, f/f_0=2.0; Solute 4: MW=300 kDa, f/f_0=2.4). This system exemplifies the general case of a polymer unit that forms end-to-end associations as could be found in either a mixture of DNA fragments, a mixture of disordered or unfolded proteins, or an amyloid or fibril forming macromolecular mixture. We analyzed the system using our parallel method and compared the results to a commonly used traditional method. We report the results along with execution times for various numbers of processors in Tables 1 and 2. From these results, it is clear that significant improvements are realized by employing our parallel methods. Furthermore, we demonstrate that the increased computational demand by our parallel methods can be effectively addressed by scaling the parallel computing infrastructure to produce compute times on the same order of magnitude as traditional methods provide.

Due to varying lengths of the fragments in our simulated example, the frictional ratios increase with molecular weight. This produces a heterogeneity both in molecular weight as well as in shape, which is poorly modeled by traditional methods such as C(M) [14]. The parallel approach proposed in our work aims to address the general case of solving systems that display heterogeneity in molecular weight *and* in frictional parameters, for which the C(M) and C(s) methods are not suitable. If implemented correctly (see references [2, 3]), the method treats the two-dimensional problem of shape and molecular weight with a high-resolution approach that appropriately deals with experimental noise and eliminates false positives. As a result, our methods show a significant improvement in

Table 1 Comparison of sedimentation velocity analyses for a simulated mixture of four macromolecular fragments that are heterogeneous in shape with molecular weights listed in the target column

Solute		Target	C(M) no reg	C(M) reg	Parallel method
	RMSD		0.01217	0.01228	0.00987
1	MW	25	16.26 (0.914)	15.42 (3.69)	24.9 (0.007)
			36.97 (1.37)	38.40 (7.93)	
	Conc.	0.25	0.11 (NA)	0.09 (NA)	0.25 (0.0015)
			0.14 (NA)	0.22 (NA)	
2	MW	75	52.81 (1.35)	72.22 (10.1)	74.9 (0.107)
			76.37 (2.35)		
	Conc.	0.25	0.14 (NA)	0.43 (NA)	0.24 (0.0024)
			0.33 (NA)		
3	MW	150	123.85 (2.67)	122.46 (6.57)	145.0 (0.051)
	Conc.	0.25	0.28 (NA)	0.28 (NA)	0.26 (0.0018)
4	MW	300	–	–	304.3 (0.339)
	Conc.	0.25	–	–	0.25 (0.00078)

Our parallel method is compared to a traditional C(s) analysis [14], without and with 95% regularization. Standard deviations for each parameter are listed in parentheses. Molecular weights are reported in kDa. Concentrations are reported in optical density units. Standard deviations for the partial concentration measurements made by SEDFIT (Schuck,http://www.analyticalultracentrifugation.com) were not available from the software.

RMSD, parsimony in the number of solutes, and in the determination of the solute's actual parameters.

In our chosen example, we observe a 24% improvement in RMSD with our parallel method, and the actual target values are faithfully reproduced, while the C(M) analysis not only reports false positive components with incorrect molecular weights but it also fails to detect the appropriate partial concentrations and misses entirely the fourth species. Standard deviations are much narrower for the parallel method, and molecular weights are well described by the confidence intervals. The failure of the C(M) approach is independent of regularization. It is obvious from the reported execution times that a computational price is paid for the increase in resolution and accuracy, which renders processing of sedimentation velocity data impractical for a single-processor computer. As we demonstrate here, this limitation is effectively removed by our parallel implementation. We have performed a detailed analysis of the scalability of our approach in [2], which proves that our parallel approach scales linearly with the number of processors used in the calculation. Thus, we have shown that, with the appropriate high-performance computing infrastructure, our approach reduces computing times to practical levels, which are comparable to traditional methods.

Conclusion

In this paper, we have described high-performance computing extensions to UltraScan that address problems in the parameter optimization of AUC experiments where macromolecular solutions heterogeneous in mass and shape are measured. The 2DSA analysis method determines shape and molecular weight information from AUC sedimentation velocity experiments using a fixed grid approach. The parallel application of 2DSA enables extremely fine resolution results and provides a signal amplification to minimize the effects of stochastic noise. Because 2DSA has been designed for efficient parallel computation, analysis results are obtained in acceptable computation time. These runs typically take between 5 and 20 min on a 44-processor Opteron cluster. The MC extension to 2DSA will multiply the time of a single run by the number of MC iterations. The benefits of MC are that the signal-to-noise ratio is increased in direct proportion to the number of MC iterations, and thus, statistically significant confidence intervals can be obtained for both shape and molecular weight of all solutes in the mixture.

The GA analysis method also determines shape and molecular weight information. In contrast to 2DSA, GA

Table 2 Results along with execution times for various numbers of processors

Execution time with different numbers of processors	C(M) no reg	C(M) reg	Parallel method
Execution time 1 processor	2 m	2 m	2,732 m
Execution time 36 processors	–	–	95 m
Execution time 51 processors	–	–	66 m
Execution time 128 processors	–	–	27 m
Execution time 256 processors	–	–	15 m
Execution time 512 processors	–	–	9 m

uses floating parameter values and can obtain even finer resolution than are obtained with the fixed grid 2DSA method. The high number of degrees of freedom present and the stochastic nature of the GA method make GA a slower method than 2DSA. Using the 2DSA result to initialize the GA provides significant speed increase for the GA because the GA can then focus on searching for solutions in a bucket-constrained area. A major benefit of the GA besides the floating parameter space is the ability of the GA to perform parsimonious regularization. This addition, based upon Occam's Razor, finds solutions of greater simplicity with equivalent RMSD. Use of parsimonious regularization also speeds up the GA, as it pressures evolution towards smaller population members. GA is still generally slower that 2DSA, but the quality of result is unsurpassed. The MC method also benefits the GA method by increasing the signal-to-noise ratio and providing statistical confidence intervals.

We have documented in "Implementation" an overview of the significant work we have done to allow simple web access to our advanced analysis methods. Through the step-by-step methodology described in "Methodology", the researcher can obtain the highest quality results combining the methods of 2DSA, GA, and MC to determine statistically confident shape and molecular weight distributions from sedimentation velocity experiments. All methods presented here are programmed in UltraScan, which can be downloaded freely from http://www.ultrascan.uthscsa.edu. A web portal suitable for implementation on a local Linux cluster is available for download from the same web site.

Acknowledgment We would like to thank Josh Wilson, Yu Ning, and Bruce Dubbs for contributions to the web interface code. This research has been supported by NSF Grant DBI-9974819, NIH Grant 1 R01 RR022200-01A1, and the San Antonio Life Science Institute with Grant #10001642, all to B.D. The parallel calculations were performed on the Linux cluster at the Bioinformatics Core Facility at the University of Texas Health Science Center and on the Lonestar cluster at TACC through NSF Teragrid Allocation # TG-MCB070038. We gratefully acknowledge support by the Robert J. Kleberg Jr. and Helen C. Kleberg Foundation.

References

1. Moore GE (1965) Cramming more components onto integrated circuits. Electronics 38(8) (April)
2. Brookes EH, Boppana RV, Demeler B (2006) Computing large sparse multivariate optimization problems with an application in biophysics. In: SuperComputing 2006 Conference Proceedings. ACM, IEEE, November
3. Brookes EH, Demeler B (2006) Genetic algorithm optimization for obtaining accurate molecular weight distributions for sedimentation velocity experiments. In: Analytical Ultracentrifugation VIII. Prog Colloid & Polym Sci 131:78–82 (Springer)
4. Grama A, Gupta A, Karypis G, Kumar V (2003) Introduction to parallel computing, 2nd edn. Addison-Wesley, Boston
5. Hall MW, Anderson JM, Amarasinghe SP, Murphy BR, Liao S-W, Bugnion E, Lam MS (1996) Maximizing multiprocessor performance with the SUIF compiler. IEEE Computer, December
6. Lamm O (1929) Die Differentialgleichung der Ultrazentrifugierung. In: Ark Mat Astrol Fys 21B:1–4
7. Lawson CL, Hanson RJ (1974) Solving least squares problems. Prentice Hall, New Jersey
8. Demeler B, van Holde KE (2004) Sedimentation velocity analysis of highly heterogeneous systems. In: Anal Biochem 335:279–288
9. Holland JH (1992) Adaptation in natural and artificial systems, 2nd edn. MIT, Cambridge, MA
10. Brookes E, Demeler B (2007) Parsimonious regularization using genetic algorithms applied to the analysis of analytical ultracentrifugation experiments. In: Proceedings of Genetic and Evolutionary Computation Conference 2007 (in press)
11. Demeler B, Brookes E (2007) Monte Carlo analysis of sedimentation experiments. Prog Colloid & Polym Sci (in press) DOI 10.1007/s00396-007-1699-4
12. Demeler B (2005) UltraScan: a comprehensive data analysis software package for analytical ultracentrifugation experiments. Royal Society of Chemistry, UK
13. Wall L, Christiansen T, Orwant J (2000) Programming PERL, 3rd edn. O'Reilly and Associates, Sebastopol, California
14. Schuck P (2000) Size-distribution analysis of macromolecules by sedimentation velocity ultracentrifugation and Lamm equation modeling. Biophys J 78(3):1606–1619