SOURCEBOOK OF PARALLEL COMPUTING

JACK DONGARRA
University of Tennessee

IAN FOSTER
Argonne National Laboratory

GEOFFREY FOX
Indiana University

WILLIAM GROPP
Argonne National Laboratory

KEN KENNEDY
Rice University

LINDA TORCZON
Rice University

ANDY WHITE
Los Alamos National Laboratory

MORGAN KAUFMANN PUBLISHERS
AN IMPRINT OF ELSEVIER SCIENCE

8.9 Deterministic Monte Carlo Methods and Parallelism

Michael Mascagni

Monte Carlo methods (MCMs) have been, and continue to be, very popular algorithms for solving a wide variety of problems in science, engineering, and technology. However, they are generally methods of last resort. As Mark Kác, a probability and Monte Carlo pioneer, put it, "You use Monte Carlo methods until you understand the problem." Yet there are clearly large classes of problems that remain poorly understood in the sense of Mark Kác. This is because MCMs remain the best approaches to certain classes of problems. While it is impossible to clearly identify the problem classes where MCMs are most effective, one can generally say that most problems that rely on MCMs for their solution either live in high dimensions or have extremely complicated geometries.

Given that MCMs will continue to dominate numerical approaches in certain application areas, it behooves MCM practitioners to optimize their computational methods as much as possible. This is especially evident when one considers that the U.S. Department of Energy (DoE) has claimed that MCMs have consistently consumed up to a half of their high-performance computing cycles since the beginning of DoE's supercomputing activities. A generic problem with MCMs is their slow convergence with respect to statistical error. Since MCMs are based on statistical sampling, a quantity of interest is known only within a statistically defined confidence interval. The width of such confidence intervals generically decreases as

$\mathcal{O}(N^{-1/2})$ with N random samples. Clearly, a modest improvement in this stochastic convergence rate would have a significant impact on scientific computing.

A generic approach to the acceleration of Monte Carlo convergence is through the use of so-called quasi-random numbers (QRNs). These are numbers that are highly uniformly distributed and thus preferred in MCMs where an even sampling of the computational space is more important than randomness. The classical MCM, numerical integration, is an example of an application that in reality requires uniformity, not randomness. With pseudorandom numbers (PRNs), N samples reduce the stochastic errors by $\mathcal{O}(N^{-1/2})$, while quasi–Monte Carlo methods can produce *deterministic* errors as small as $\mathcal{O}(N^{-1})$ in numerical integration. QRNs have also been used to accelerate the Monte Carlo convergence of other applications, and so they are sought after by computational scientists.

The purpose of this section is twofold. Primarily, it is to acquaint the reader with QRNs and the advances being made with quasi–MCMs in a variety of application areas. Secondarily we describe some of the problems inherent with applying QRNs to parallel computations and provide the reader with empirical evidence that quasi–MCMs are being applied to a broad spectrum of parallel Monte Carlo applications with some success. We begin with a standard introduction to QRNs via the numerical quadrature application. This introduces the discrepancy, a measure of the deviation of a point set from uniformity, and provides us with an understanding of how well QRNs can perform. This also gives us a clear demonstration of a Monte Carlo algorithm that in reality needs uniformity rather than randomness for optimal performance. Next, a problem associated with splitting QRN sequences for use on different problems (processors) is discussed. This shows that in this particular case, the ability to combine two or more results to obtain greater accuracy is equivalent to the problem of creating a parallel QRN generator. This is a special set of circumstances where the parallelization is required to advance capabilities for serial computation as well. We then review methods of quasi-random number generation and point out the deficiencies in currently available, free QRN software. Finally, we briefly present the results of a Markov chain computation for solving a problem in linear algebra via an MCM. Here we show (1) that one can parallelize the quasi–Monte Carlo approach to the problem, (2) that the parallel efficiency of the regular Monte Carlo approach is maintained by the quasi–Monte Carlo method, and (3) that the accelerated convergence of QRNs is maintained in this parallel context.

8.9.1 Motivation for Using Quasi-Random Numbers

MCMs are based on mathematical processes that utilize random numbers. The computational requirement for random numbers in Monte Carlo applications has been satisfied with two types of computational random numbers: PRNs and QRNs. PRNs mimic the behavior of "real" random numbers in theoretical and empirical tests, whereas QRNs provide very uniformly distributed sets of numbers that may, in fact, perform poorly on tests of randomness. However, QRNs are more effective than PRNs in situations where the uniform distribution of points is important. Such applica-

tions include *the* canonical Monte Carlo application, the numerical evaluation of integrals. It is the case that many nonquadrature Monte Carlo computations can be mathematically viewed as numerical quadrature, and so many other types of Monte Carlo applications have seen performance improvement when PRNs have been carefully replaced with QRNs. In fact, many application areas that do not at face value seem to be anything like quadrature have been favorably impacted by the use of QRNs. These include simulations with random walkers in application areas as diverse as heat conduction [687], rarefied gas dynamics [165], particle transport [892], numerical linear algebra [650], and financial-instrument evaluation [166]. In addition, QRNs promise to improve the convergence of applications in quantum mechanics, materials science, biochemistry, and environmental remediation.

The mathematical motivation for QRNs can be found in the classic Monte Carlo application of numerical integration. For simplicity, we detail this for 1-D integration. Let us assume that we are interested in the numerical value of $I = \int_0^1 f(x)\,dx$, and we seek to optimize approximations of the form

$$I \approx \frac{1}{N}\sum_{n=1}^{N} f(x_n)$$

A solution to the optimization of the integration nodes, $\{x_n\}_{n=1}^N$, comes from the famous Koksma–Hlawka inequality. Let us define the star-discrepancy of a 1-D point set, $\{x_n\}_{n=1}^N$, by

$$D_N^* = D_N^*(x_1, \ldots, x_N) = \sup_{0 \le u \le 1} \left| \frac{1}{N} \sum_{n=1}^{N} \chi_{[0,u)}(x_n) - u \right|$$

where $\chi_{[0,u)}$ is the characteristic function of the half-open interval $[0, u)$. The term $\sum_{n=1}^N \chi_{[0,u)}(x_n)$ counts the number of x_n's in the interval $[0, u)$, and thus $\left|\frac{1}{N}\sum_{n=1}^N \chi_{[0,u)}(x_n) - u\right|$ measures the difference between the actual distribution of points in the interval $[0, u)$ and the uniform distribution on $[0, u)$. By taking the supremum, we are characterizing the distribution of the $\{x_n\}_{n=1}^N$ through its maximal deviation from uniformity. We thus have the remarkable theorem due to Koksma and Hlawka [571]: If $f(x)$ has bounded variation, $V(f)$, on $[0, 1)$, and $x_1, \ldots, x_N \in [0, 1]$ have star-discrepancy D_N^*, then

$$\left| \frac{1}{N}\sum_{n=1}^{N} f(x_n) - \int_0^1 f(x)\,dx \right| \le V(f)\, D_N^*$$

This simple bound on the integration error is a product of $V(f)$, the total variation of the integrand in the sense of Hardy and Krause, and D_N^*, the star-discrepancy of the integration points. A major area of research in Monte Carlo is variance reduction, which indirectly deals with minimizing $V(f)$. QRN generation deals with minimization of the other term.

Mathematically, QRNs produce point sets and sequences that have low discrepancy. Discrepancy is a quantitative measure of the uniformity of a point set. The star-discrepancy, introduced above, is merely one of many discrepancies that are used to measure uniformity of discrete measures [706]. For example, the star-discrepancy of a point set of N "real" random numbers in one dimension is $\mathcal{O}(N^{-1/2}(\log\log N)^{1/2})$, while the discrepancy of N QRNs can be as low as (N^{-1}).[2] In $s > 3$ dimensions, it is rigorously known that the discrepancy of a point set with N elements can be no smaller than a constant depending only on s times $N^{-1}(\log N)^{(s-1)/2}$. This remarkable result of Roth [810] has motivated mathematicians to seek point sets and sequences with discrepancies as close to this lower bound as possible. Since Roth's results, there have been many constructions of low discrepancy point sets that have achieved star-discrepancies as small as $\mathcal{O}(N^{-1}(\log N)^{s-1})$. Most notably, there are the constructions of Hammersley, Halton [431], Sobol' [139, 881], Faure [321, 351], and Niederreiter [140, 706].

While QRNs do improve the convergence of some applications, it is by no means trivial to enhance the convergence of all MCMs. Even in the case of numerical integration, enhanced convergence is by no means assured in all situations with the naive use of QRNs. This fact was demonstrated through studies of the efficacy of QRNs in numerical integration [165, 166, 688, 691,] by carefully investigating the impact of dimensionality and smoothness of the integrand on convergence. In a nutshell, their results showed that at high dimensions ($s \approx> 40$), quasi–Monte Carlo integration ceases to be an improvement over regular Monte Carlo integration. Perhaps more startling was that a considerable fraction of the enhanced convergence is lost in quasi–Monte Carlo integration when the integrand is discontinuous. In fact, even in two dimensions one can lose the approximately $\mathcal{O}(N^{-1})$ quasi–Monte Carlo convergence for an integrand that is discontinuous on a curve such as a circle. In the best cases, the convergence drops to $\mathcal{O}(N^{-2/3})$, which is only slightly better than regular Monte Carlo integration.

8.9.2 Methods of Quasi-Random Number Generation

Perhaps the best way to illustrate the difference between QRNs and PRNs is with a picture. In Figure 8.13, we plot 4096 tuples produced by successive elements from a 64-bit PRN generator from the SPRNG library [651]. These tuples are distributed in a manner consistent with real random tuples. In Figure 8.14, we see 4096 quasi-random tuples formed by taking the second and third dimensions from the Sobol' sequence. It is clear that the two figures look very different and that Figure 8.14 is much more uniformly distributed. Both plots have the same number of points, and the largest "hole" in Figure 8.13 is much larger than that in Figure 8.14. This illustrates quite effectively the qualitative meaning of low discrepancy.

[2] Of course, the N optimal quasi-random points in $[0, 1)$ are the obvious: $\frac{1}{(N+1)}, \frac{2}{(N+1)}, \cdots, \frac{N}{(N+1)}$.

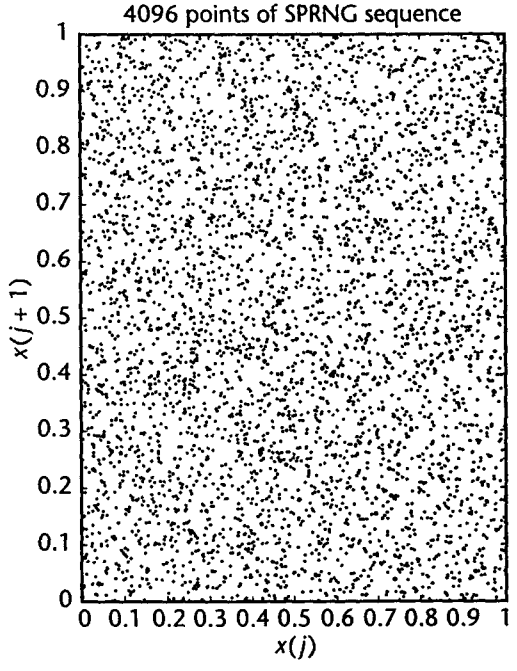

Figure 8.13 Tuples produced by successive elements from a SPRNG pseudorandom number generator.

The first QRN sequence was proposed by Halton [431] and is based on the Van der Corput sequence, with different prime bases for each dimension. The jth element of the Van der Corput sequence with base b is defined as $\phi_b(j-1)$, where $\phi_b(\cdot)$ is the radical inverse function and is computed by writing $j-1$ as an integer in base b, and then flipping the digits about the ordinal (decimal) point. Thus, if $j-1 = a_n \ldots a_0$ in base b, then $\phi_b(j-1) = 0.a_0 \ldots a_n$. As an illustration, in base $b=2$, the first elements of the Van der Corput sequence are $\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}$; while with $b=3$, the sequence begins with $\frac{1}{3}, \frac{2}{3}, \frac{1}{9}, \frac{4}{9}, \frac{7}{9}, \frac{2}{9}, \frac{5}{9}, \frac{8}{9}$. With $b=2$, the Van der Corput sequence methodically breaks the unit interval into halves in a manner that never leaves a gap that is too big. With $b=3$, the Van der Corput sequence continues with its methodical ways, but instead recursively divides intervals into thirds.

Another way to think of the Van der Corput sequence (with $b=2$) is to think of taking the bits in $j-1$ and associating with the ith bit the number v_i. Every time the ith bit is one, perform an exclusive-or in v_i, called the ith direction number. For the Van der Corput sequence, v_i is just a bit sequence with all zeroes except a one in the ith location counting from the left. Perhaps the most popular QRN sequence, the Sobol sequence, can be thought of in these terms. Sobol [881] found a clever way to define more complicated direction numbers than the "unit vectors" that define the

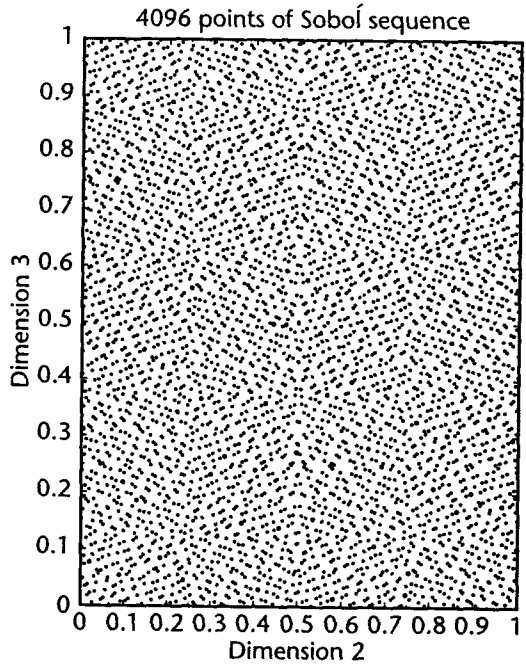

Figure 8.14 Tuples produced by the second and third dimension of the Sobol sequence.

Van der Corput sequence. Besides producing very good quality QRNs, the reliance on direction numbers means that the Sobol sequence is both easy to implement and very computationally efficient.

Since this initial work, Faure et al. [321], Niederreiter [705], and Sobol [881] chose alternate methods based on another sort of finite field arithmetic that utilizes primitive polynomials with coefficients in some prime Galois field. All of these constructions of quasi-random sequences have discrepancies that are $\mathcal{O}(N^{-1}(\log N)^s)$ [705]. What distinguishes them is the asymptotic constant in the discrepancy and the computational requirements for implementation. However, practice has shown that the provable size of the asymptotic constant in the discrepancy is a poor predictor of the actual computational discrepancy displayed by a concrete implementation of any of these QRN generators. There are existing implementations of the Halton, Faure, Niederreiter and Sobol sequences [139, 140, 351] that are computationally efficient. Each of these sequences is initialized to produce quasi-random s-tuples, and each one of these requires the initialization of s 1-D quasi-random streams. However, in practice the Sobol sequence has shown itself superior in quality and efficiency to these other methods. Thus, we will restrict our discussion to the Sobol generator.

8.9.3 A Fundamental Problem with Quasi-Random Numbers

QRNs are finely crafted mathematical objects that are hyperuniform. Recall the definition of the star-discrepancy of a set of points above. It is defined as the *supremum* of the difference between an empirical distribution of the set of points and the ideal uniform distribution. Clearly, a single misplaced point can lead to a serious degradation in this estimate. Thus, one should think of point sets (sets with N fixed numbers in them) of QRNs as sets that have completely filled all the holes in space at a given spatial scale. Similarly, sequences (sets with an extensible number of points) of QRNs are constructed so that the areas with the largest holes in space are exactly the next areas where points are to be placed.

The very highly structured nature of QRNs leads to an interesting problem. Let us perform a calculation with N QRNs from a given quasi-random sequence with given parameters and given initial values. Let's say we obtain the estimate q for some quantity of interest, Q. Theory tells us that in the best circumstances $|q - Q| = \mathcal{O}(N^{-1}(\log N)^k)$, for some k. However, the only practical way to continue this calculation is to continue with the $(N+1)$st QRN from the same sequence. If we choose another QRN sequence, or even the same sequence starting with other than the *next* unused point, we will get no guarantee of continued accelerated Monte Carlo convergence. In fact, using incompatible QRNs can lead to circumstances where convergence to the correct answer may no longer hold.

Clearly, this problem is equivalent to the problem of finding parallel streams of QRNs that can collectively be used together in a complementary fashion. Work in this area has shown that the gist of the above paragraph seems to be true, that is, at present one can do no better than to break up a single QRN sequence into nonoverlapping blocks for use in parallel. Schmid and Uhl [837] investigated the consequences of blocking QRN sequences versus using a leapfrog technique.[3] They determined that blocking from the same sequence led to acceptable results, whereas the leapfrog technique often caused problems with the subsequences. Clearly, more flexibility than this will be required if QRNs are to be used in calculations that terminate with a stochastic convergence condition.

8.9.4 State-of-the-Art Quasi-Random Number Generators

A serious problem with using QRNs in both serial and parallel Monte Carlo applications is the lack of good quality, widely available QRN generation software. At present, good implementations of the Sobol', Faure, and Halton sequences exist, but there is no software that provides the facilities necessary for simple parallel use of such generators. In addition, generators with certain desirable properties are not freely available. One of the most popular application areas for QRNs is currently in

[3] Suppose we have k QRN subsequences of length N we wish to create. In blocking, the first subsequence consists of the first N numbers, the second subsequence of the next N numbers, and so on. When using the leapfrog technique, the ith subsequence is $\{x_i, x_{i+N}, x_{i+2N}, \ldots, x_{i+(N-1)N}\}$.

Table 8.2 Implementation using MPI of the power Monte Carlo algorithm (PMC) and power quasi–Monte Carlo algorithm (PQMC) for calculating the dominant eigenvalue of a matrix of size 2000 using PRNs and Sobol QRNs.

		Number of processors				
		1	2	3	4	5
PMC	Time (s)	168	84	56	42	33
	Efficiency		1	1	1	1.01
	λ_{max}	62.48	61.76	63.76	61.3151	61.39
PQMC	Time (s)	177	87	70	57	44
	Efficiency		1.01	0.84	0.77	0.80
	λ_{max}	64.01	64.01	64.01	64.01	64.01

Note: The number of Markov chains (realizations) used is 100,000, and the exact value is $\lambda_{max} = 64.00$.

financial mathematics. However, some of the canonical problems are often set in very high-dimensional spaces. For example, the pricing of a mortgage-backed security made up of 30-year home mortgages is a 360-dimensional problem [689].[4] At present, there is no high-quality QRN software that produces sequences in such high dimensions. In fact, to our knowledge the only publicly available Sobol QRN generation software allows for sequences up to dimension 41.

8.9.5 A Parallel Quasi–Monte Carlo Application

Given this brief introduction to QRNs, we wish to illustrate their utility on a parallel application. We present our results for an MCM for the computation of the extremal eigenvalue of a sparse, square matrix [271]. The method we employ is a stochastic version of the well-known power method. It is based on the repeated application of the matrix, which is ideal for Markov-chain–based MCMs. A good description of both the MCM and the application of QRNs to this problem can be found elsewhere [648, 649, 650]. The computations presented here were implemented in parallel using MPI on an IBM SP-2 located at the Florida State University's School of Computational Science and Information Technology.

MCMs are "naturally parallel."[5] They allow us to compute with minimal communication. In our case, we need only pass the nonzero elements of the sparse matrix A to every processor (Table 8.2). Then we compute a total of N Monte Carlo realizations on p processors. Each processor gets N/p realizations, and we collect the results at the end. The only communication here is at the beginning and at the end of the

[4] Thirty-year mortgages are paid monthly, giving 360 payment periods during the mortgage's life. This accounts for the 360-dimensionality of the mortgage-backed security problem.

[5] This distinction of MCMs as "naturally parallel" was first used by Malvin Kalos.

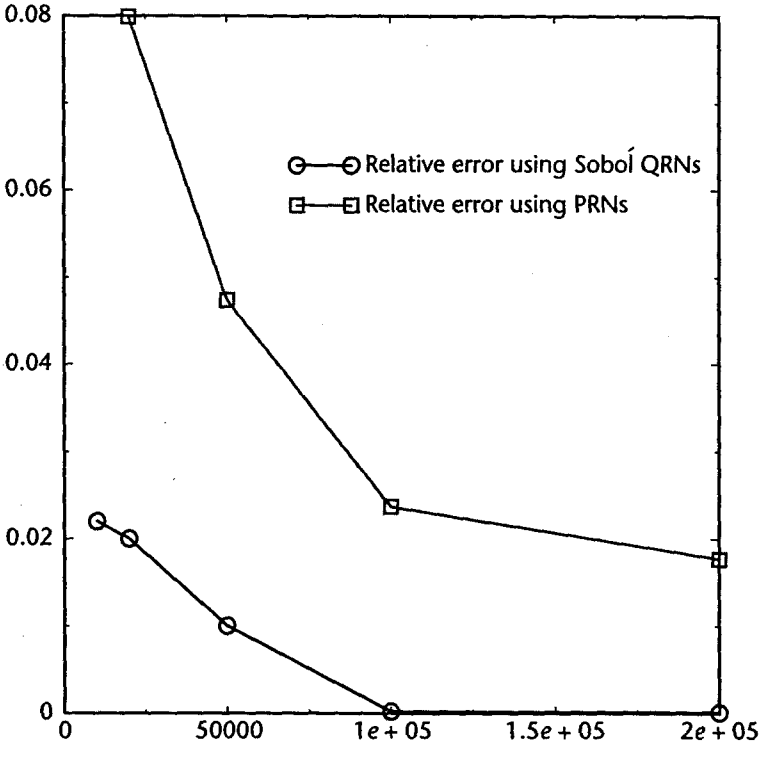

Figure 8.15 Relative errors in computing the dominant eigenvalue for a sparse matrix of size 2000×2000. Markov chains are realized using PRNs and Sobol QRNs.

program execution; this provides a very efficient parallel implementation. Note that in our empirical analysis, we use the standard definition of *parallel efficiency*, E.

Figure 8.15 presents the relative errors of the power MC algorithm and power quasi-Monte Carlo algorithm (using the Sobol sequence) for computing the dominant eigenvalue for a sparse square matrix of size 2000. Note that with 20,000 points from our Sobol sequence, we achieve an accuracy that would require 100,000 or more PRNs. The fact that QRNs can achieve accuracy similar to PRNs for this kind of calculation, while using only a fraction of the time, is the significant reason for using QRNs.

In this computation, we knew beforehand how many QRNs would be used in the entire calculation, and we neatly broke the sequences into same-sized subsequences. Clearly, it is not expected that this information will be known beforehand. Providing QRNs that can help extend calculations easily remains the major challenge to widespread parallel use of QRNs.

We have shown that one can parallelize the quasi–Monte Carlo approach to the calculation of the extremal eigenvalue of a matrix. We have also shown that the

parallel efficiency of the regular Monte Carlo approach is maintained by the quasi–Monte Carlo method; however, there is some slight degradation. Finally, perhaps the most important fact is that the accelerated convergence of QRNs is maintained in this parallel context.

8.9.6 State of the Art

We have introduced the reader to the concept of quasi–MCMs and QRNs. These are powerful techniques for accelerating the convergence of ubiquitous MCMs. However, even though quasi–MCMs can often be made to converge much faster than ordinary MCMs, the ability to improve the accuracy of quasi–MCMs as readily as ordinary MCMs is not here yet. Nonetheless, for certain applications it is possible to accelerate the convergence of Monte Carlo applications with QRNs and to take advantage of their natural parallelism. At present, there are a variety of Monte Carlo applications that benefit from QRN acceleration. Most notable, perhaps, is the calculation of financial derivatives [303]. In the future, we expect to see considerable benefits to other Monte Carlo applications, and hence to scientific computing.

8.10 Quasi–Real Time Microtomography Experiments at Photon Sources

Gregor von Laszewski, Mei-Hui Su, Joseph Insley, Ian Foster, and Carl Kesselman

Computed microtomography (CMT) is a powerful tool for obtaining nondestructively a 3-D view of the internal structure of opaque objects [421]. In contrast to the widespread use of this technique in the millimeter scale as part of diagnostic procedures in hospitals, we are interested in the investigation of objects on the micrometer scale.

One application of this method is quality control during the production of 3-D semiconductor wafers. Being able to visualize the details of chip wafers in all three dimensions allows engineers to improve the chip design before production. Other examples can be found in the field of earth science, where common tasks include investigation of the interior of very small meteorites and study of the enclosures of very tiny materials in opaque diamonds formed 100,000 years ago, in order to determine more about the origin and development of the Earth.

The energy and the infrastructure necessary to conduct such experiments can be provided by using x-ray beams at synchrotrons. The use of x-rays for investigating the internal structure of materials at the micron scale has grown rapidly over the past decade as a result of the availability of synchrotron radiation sources. One such facility is the Advanced Photon Source (APS) at Argonne National Laboratory.

A typical computed microtomography experiment at the APS proceeds as follows. A sample is mounted in the experiment station, parameters are adjusted, and the sample is illuminated by a collimated beam of x-rays. Data are collected for multiple sample orientations by using a charge-coupled device. A time-consuming reconstruction process is then used to obtain a 3-D representation of the raw data with spatial