Ph.D. Qualifying Examination Answers

School of Computer and Infomation Science
Syracuse University

Sung-Hoon Ko

(shko@npac.syr.edu)

1) Describe at least two applications(not just algorithms!) that require parallel random number generator. Explain characteristics of the generation that are important in each case.

ans :

(1) We can evaluate one-dimensional integral by the Monte Carlo method.

$$\int_0^1 f(x)\,dx \tag{1}$$

Evaluating this integral by Monte Carlo amounts to generating random numbers, r_i, in the range 0 to 1, and for each one computing the score, $f(r_i)$. If this is done many times, the average score tends to the value of the integral. The parallel implementation involves having each node generate events and keep its own estimate of the average score. At the end, the N scores are averaged.

(2) Calculation of π via Monte Carlo.
Consider a circle of unit radius centered at the origin with square circumscribed about it. This circle forms the dart board. Darts are thrown so that they always hit the square, but have an equal probability of hitting any part of the square. During this process, we keep a score which is incremented by 1 for each dart inside the unit circle, and 0 for darts outside the circle. If we then divide the total score by the total number of darts thrown, we obtain an estimate of the ratio of the area of the circle to the are of the square. The ratio of the areas is $\pi/4$, so that the total score divided by the total number of trials($\times 4$) yields an estimate of π. Here the throws of the darts are independent and so we will have each node of the concurrent processor run its own dart board simulation and make its own estimate of π. At the end, the N estimates will be globally combined.

2) Answer these questions on cluster algorithms.
a) Describe algorithms for identifying and labeling connected components(or cluster) of an undirected graph, using
i. breadth-first search
ii. depth-first search

ans :
a)

i. breadth-first search

This algorithm uses the idea of a rooted tree. Each site i belonging to cluster α is assigned a cluster label m_t^α. The cluster α may initially be assigned several different cluster labels, which are given as a set of natural numbers $\{m_1^\alpha, m_2^\alpha, ..., m_s^\alpha, ..., m_t^\alpha\}$. In this set only one number is regarded as the proper cluster label which we designate as m_s^α. This is the smallest number in the set. A connection between the label m_t^α at any site, and the proper cluster label m_s^α, is provide by an array N. This is constructed so that only $N(m_s^\alpha)$ is positive and denotes the number of sites in the cluster, while the remaining $N(m_s^\alpha)$ provide the links between the m_t^α and the proper cluster label.

ii. depth-first search

This algorithm is so called Ants-in-the-labyrinth. An ant is put somewhere on the lattice, and notes which of the neighboring sites are connected to the site it is on. At the next time-step this ant places children on each of these connected sites which are not already occupied. The children then proceed to reproduce likewise until the entire cluster is populated. In order to label all the clusters, we start by giving every site a negative label, set the initial cluster label to be zero, and then loop through all the sites in turn. If a site's label is negative then the site has not already been assigned to a cluster, so we place an ant on this site, give it the current label, and let it reproduce, passing the label on to all its offspring. When this cluster is identified we increment the cluster label and carry on, repeating the ant-colony birth, growth and death cycle until all the clusters have been identified.

2)

b) For ONE of these algorithms, explain how it could be parallelized.

ans :

We begin by assigning each site i a unique cluster label S_i. At each step of the algorithm, in parallel, every site looks in turn at each of its neighbors in the positive directions. If it is bonded to a neighboring site n which has a different cluster label S_n, then both S_i and S_n are set to the minimum of the two. This is continued until nothing changes, by which time all the cluster will have been labeled with the minimum initial label of all the sites in the cluster.

2)

c) Would you expect the parallel algorithm to perform better on a SIMD or a MIMD machine? Why?

ans :

This algorithm is more effective on a coarse grain MIMD machine. Since we can label the clusters on the sub-lattice on each processor using fast, efficient sequential algorithm. The local label propagation is done only on the edges of the sub-lattice, to match up labels of clusters which cross processor boundaries. So we have good load balancing, and label of largest cluster is propagated across number of processors, not number of sites.

3) Answer these four questions on random number generation
a) Describe two common algorithms used in random number generators.

ans :

(1) Linear Congruential Generators(LCG).
LCG produce a sequence X_i of random integers using the relation

$$X_i \;=\; (aX_{i-1} + c)\, mod\, M$$

where a is the multiplier, M is the modulus, and c is an additive constant. The maximum periods is M for suitably chosen parameters(M-1 if c=0). LCG work well for most application but are known to have some major defects. The main problem is that the least significant bits of the numbers produced are correlated. Another problem is that many commonly-used LCG use a modulus M that is a power of 2, since it is fast and convenient to implement this on a computer. This approach produce highly correlated low-order bits, as well as long-range correlations for intervals that are a power of 2.

(2) Lagged Fibonnaci Generators.
The sequence is defined by

$$X_i \;=\; X_{i-p} \odot X_{i-q}$$

where p and q are the lags, $p > q$, and $\odot$ is any binary arithmetic operation, such as addition or subtraction modulo M, multiplication modulo M, or the bitwise exclusive OR function. The arithmetic operations are done modulo any large integer value M, or modulo 1 if the X's are represented as floating point numbers in the interval [0,1), as can be done if operation is addition or subtraction. Multiplication must be done on the set of odd integers. The modulus function is not required when using XOR. This method requires storing the p previous values in the sequence in a array called lag table. If M is taken to be 2^b(i.e the X's have b-bit precision), the maximal period is obtained if the lags are the exponent of a primitive polynomial. In the case the period is $2^p - 1$ for XOR, $(2^p-1)2^{b-1}$ for addition and subtraction, and $(2^p-1)2^{b-3}$ for multiplication.

3)

b)For ONE of these algorithms, explain how it could be parallelized.

ans :

We can parallelize a sequential generator by taking the elements of the sequence of pseudo-random numbers it generates and distributing them among the processors. One of these method is leapfrog method.

The first N numbers in the sequence are given as seeds to the N processors, and then each processor leap-frogs to the $(i+N)th$ term in the sequence, instead of the $(i+1)st$. so the sequence is split among processors. In order to use this method we need to be able to easily jump ahead in the sequence. This can be done quite easily for linear congruential generators, and merely involves replacing the multiplier a and the additive constant c by new values $A = a^N$ and $C = c\sum_{i=0}^{N-1} a^i = c(a^N - 1)/(a-1)$ (both modulus M).

3)

c) What kind of problems might arise in parallelizing a random number generator, in both the efficiency of the implementation and the quality of the random numbers produced on each processor?

ans :

One of the reasons good parallel random number generators are so hard to create is that any small correlations that exist in the sequential generators may be amplified by the method used to distribute the sequence among the processors, producing stronger correlations in the subsequences on each processor. Also, the method used to initialize a parallel random number generator(i.e to specify the seeds for each processor) in important, since any correlations between the seeds on different processors could produce strong inter-processor correlations.

It is quite efficient if we choose modulus that is a power of 2 in leap frog method. However this is known to have correlations between elements in the sequence that are a power of 2 apart. For many parallel computers the number of physical processors is a power of 2, and this is also often the case for the pseudo-random numbers generated on a given processor may be more strongly correlated than the sequence on a single processor.

3)

d) Describe some tests you might perform to check the quality of a parallel random number generator.

ans :

It is useful to apply physical tests that are related to the various computational science applications for which random numbers are commonly used. As with the statistical test, these tests generally compare results obtained from using a pseudo-random number generator with known exact results.

(1) Ising model Monte Carlo tests:

One of the method for testing random number generators is parallel monte carlo simulation of the 2-dimensional Ising model. This model has been solved exactly for a finite lattice of grid points, so that values of the energy and the specific heat of the system calculated from the monte carlo simulation can be compared with the known exact values. There are several different monte carlo algorithms that can be used to simulate the Ising model; the Metropolis algorithm which updates a single site of the lattice at a time, the Swendseng-Wang algorithm which form clusters of sites to be updated collectively, and the Wolff algorithm which updates a single cluster of sites.

(2)Using replicated sequential test programs:

This method use standard sequential tests on the random number streams on each processor, or the combined stream from all processors. The simplest type of empirical test is to use the pseudo-random number generators to compute a result and a statistical error for that result, and then compute it with the known exact value which would occur for a truly random sequence.

4) Describe the issues that are attractive and not so attractive in the use of Java in High Performance Computing.

ans :

attractive part:

1. Java is a descendent of C++, C and C++ are used increasingly in scientific programming.
2. Java omits various features of C and C++ that are considered difficult - notably, pointers. Poor compiler analysis has often been blamed on these features.
3. Java comes with builtin multi-threading. On a shared memory platform independent threads may be scheduled on a different processors by a suitable runtime.
4. it's simple, efficient, object-oriented and platform neutral.

Not attractive:

1. Java was not specifically designed for computationally intensive numerical applications.
2. Java lacks the extensive scientific libraries of language like Fortran and C
3. Java has relatively low performance than C and Fortran code. Performance is improved in new "just-in-time" interpreters.

5) Compare the style of messages seen in MPI and those used on the Internet. What are the difference and why?

ans :

MPI specifies the buffer by starting address, datatype, and count. Data type are constructed recursively from native datatype, contiguous array of datatypes, strided blocks of datatypes, indexed array of blocks of datatypes and general structures. Count is number of datatype elements.

In general the messages will have complex types and the data may not be contiguous in memory. Java does not have a linear memory model and Java array has no uniquely defined address in memory, because the garbage collector is allowed to relocate objects unpredictably during program execution to avoid fragmentation of its workspace.