ETH zürich

High Performance Computing for Science and Engineering I

P. Koumoutsakos, M. Troyer
ETH Zentrum, CLT F 12
CH-8092 Zürich

Fall semester 2014

Set 4 - Monte Carlo, OpenMP and Diffusion

Issued: October 17, 2014
Hand in: October 24, 2014, 8:00am

Question 1: Monte Carlo Estimators

In Monte Carlo simulations we generate a set of random variables $X_i = f(x_i)$. From this sample we want to extract good estimates of the properties of the function f. In the lecture it was shown that the sample mean $\overline{X}$ is a true (i.e. unbiased) estimator for the expectation value $E[X]$ and mentioned that the variance can be estimated from the sample mean of squares.

Show that the formula given in the lecture is an unbiased estimator, i.e. prove that

$$E\left[\frac{N}{N-1}\left(\overline{X^2}-\overline{X}^2\right)\right] = \mathrm{Var}X$$

Hint: You may use that $E\left[(X_i - E[X_i])^2\right] = \mathrm{Var}X$ and $E\left[(\overline{X} - E[X])^2\right] = \frac{1}{N}\mathrm{Var}X$.

Proof. In the following the sums are understood to run over the observations $1, \ldots, N$.

$$E\left[\frac{N}{N-1}\left(\overline{X^2}-\overline{X}^2\right)\right] = \frac{1}{N-1}E\left[\sum_i X_i^2 - \frac{1}{N}\left(\sum_i X_i\right)\left(\sum_j X_j\right)\right] \tag{1}$$

$$= \frac{1}{N-1}E\left[\sum_i X_i^2 - \frac{1}{N}\sum_i X_i^2 - \frac{1}{N}\sum_i\sum_{j\neq i} X_iX_j\right] \tag{2}$$

$$= \frac{1}{N}\sum_i E\left[X_i^2\right] - \frac{1}{N(N-1)}\sum_i\sum_{j\neq i} E[X_i]E[X_j] \tag{3}$$

$$= E\left[X^2\right] - E[X]^2 \equiv \mathrm{Var}X \qquad \square$$

In (1) we inserted the definition of the mean $\overline{X}$, in (2) we split the double sum into the part where $i = j$ and a remainder, such that in (3) we can use that X_i and X_j are uncorrelated for $i \neq j$ and hence $E[X_iX_j] = E[X_i]E[X_j]$.

Question 2: Parallel Monte Carlo integration

In the previous exercise we considered the two-dimensional diffusion equation given by

$$\frac{\partial\rho(x,y,t)}{\partial t} = D\left(\frac{\partial^2\rho(x,y,t)}{\partial x^2} + \frac{\partial^2\rho(x,y,t)}{\partial y^2}\right), \tag{4}$$

where $\rho(x, y, t)$ is a measure for the amount of heat at position (x, y) and time t and the diffusion coefficient D is constant and will be assumed to be equal to 1.

In this exercise we will consider the *steady state* of the two-dimensional diffusion equation (4) in domain $\Omega = [0, 1]^2$, which is obtained by assuming that the solution does not change over time, i.e. we set $\frac{\partial \rho(x,y,t)}{\partial t} \equiv 0$. Denoting the values of ρ at the boundary $\partial\Omega$ by the function $g(x, y)$, the resulting equations are given by

$$\begin{cases} \left(\dfrac{\partial^2 \rho(x,y)}{\partial x^2} + \dfrac{\partial^2 \rho(x,y)}{\partial y^2}\right) = 0, & \text{if } 0 < x, y < 1, \\ \rho(x, y) = g(x, y), & \text{if } x \in \{0, 1\} \text{ or } y \in \{0, 1\}. \end{cases} \tag{5}$$

If solution is required only at a *single* point $(x_0, y_0) \in \Omega$, then the value $\rho(x_0, y_0)$ can be approximated using the Feynman-Kac method with Monte-Carlo sampling. The algorithm is as follows:

1. Start a two-dimensional *random walk* from position $(x_0, y_0) \in \Omega$, making *random* steps of size $d > 0$ in the two-dimensional plane $\mathbb{R}^2$. Given a random uniformly distributed number a, one step can be computed by a incrementing x_0 and y_0 by

$$\Delta x = d\cos(2\pi a), \qquad \Delta y = d\sin(2\pi a). \tag{6}$$

2. Iteratively generate random steps (for every step, a *new* random number a is required) until the random walk crosses the boundary $\partial\Omega$, i.e. leaves the domain Ω. Denote the location where the random walk (which started at (x_0, y_0)) crosses the boundary by (x_c, y_c).
3. The solution of (5) for ρ at $(x_0, y_0) \in \Omega$ is then approximated as the *expectation* of the boundary values $g(x_c, y_c)$ taken at the point where the random walk is crossing the boundary, i.e.

$$\rho(x_0, y_0) \approx \mathbb{E}[g(x_c, y_c)]. \tag{7}$$

   The above expectation can be further approximated using the Monte-Carlo method.
4. Consider N independent random walks $X_1(x_0, y_0), \ldots, X_N(x_0, y_0)$, all starting form the *same* location (x_0, y_0), but finishing (crossing the boundary $\partial\Omega$) in different locations $((x_c^1, y_c^1), \ldots, (x_c^N, y_c^N))$. The approximation for $\rho(x_0, y_0)$ is then given by

$$E_N[\rho(x_0, y_0)] = \frac{1}{N}\sum_{i=1}^{N} g(x_c^i, y_c^i). \tag{8}$$

a) Write a serial program which approximates ρ at $x_0 = 0.3$ and $y_0 = 0.4$ using a simple Monte Carlo integration scheme (8) based on random walks with step size $d = 0.01$ and boundary conditions given by

$$g(x, y) = x, \qquad \forall x, y \in \Omega. \tag{9}$$

It should also estimate the error by employing the variance estimator you proved above (with notation $X_i = g(x_c^i, y_c^i)$).

solution code in `solution_code/simple_sampling_serial.cpp`

b) Check that your code works by comparing to the exact value of $\rho(0.3, 0.4) = 0.3$.

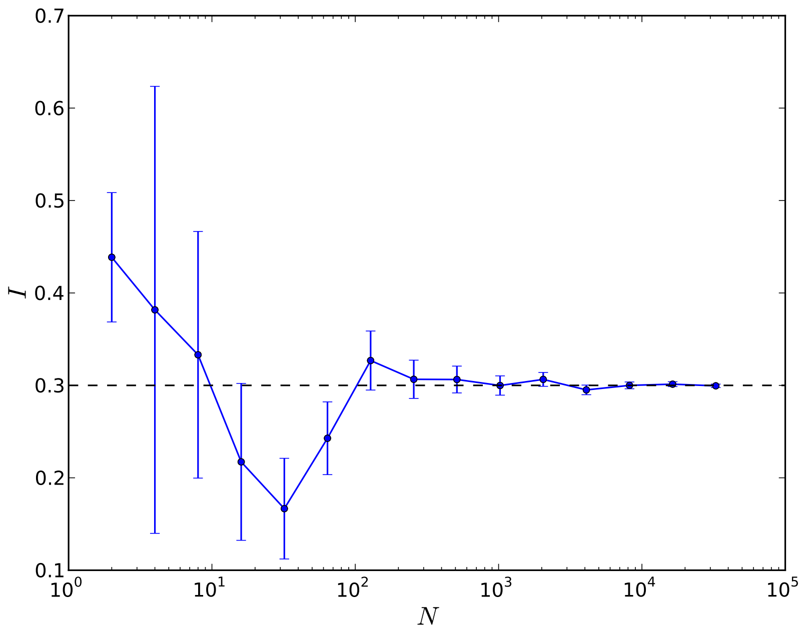

Figure 1: Approximation of $\rho(0.3, 0.4)$.

The result is plotted in figure 1

c) Run your code with different numbers of sample points ($N = 2^1, \ldots 2^{18}$) and plot the results with error bars. Is the true value of the integral always within the error bars? How do the true and estimated errors scale with the number of sample points?

The result is plotted in figure 1 and should also give $\rho(0.3, 0.4) \approx 0.3$.

- the majority of points are within one error bar of the exact result, but some are outside. We expect about one third of the points to be more than one standard error away from the correct value
- if the different runs had identical random number seeds (not the case here) we would see correlations between subsequent points
- the errors are stochastic and so decay with $1/\sqrt{N}$ for N samples (fig. 2)

d) Parallelize your code using OpenMP by splitting the sampling work (all random walks) over multiple threads. Make sure you do not introduce race conditions and that different threads use different random number sequences. Is the amount of computational work (in floating point operations) equal for all random walks? Could this cause load imbalance? Does the load balance decrease if a large number of random walks per thread is executed?

solution code in `solution_code/simple_sampling_omp.cpp`

e) Make a strong scaling plot up to 24 cores and comment on the result.

The threads only need to add their sums in the end which scales perfectly up to 24 threads.

Question 3: Metropolis Monte Carlo paper

Please read the original paper [1] on Metropolis Monte Carlo algorithm [1] :

[1]a VPN connection is required for access

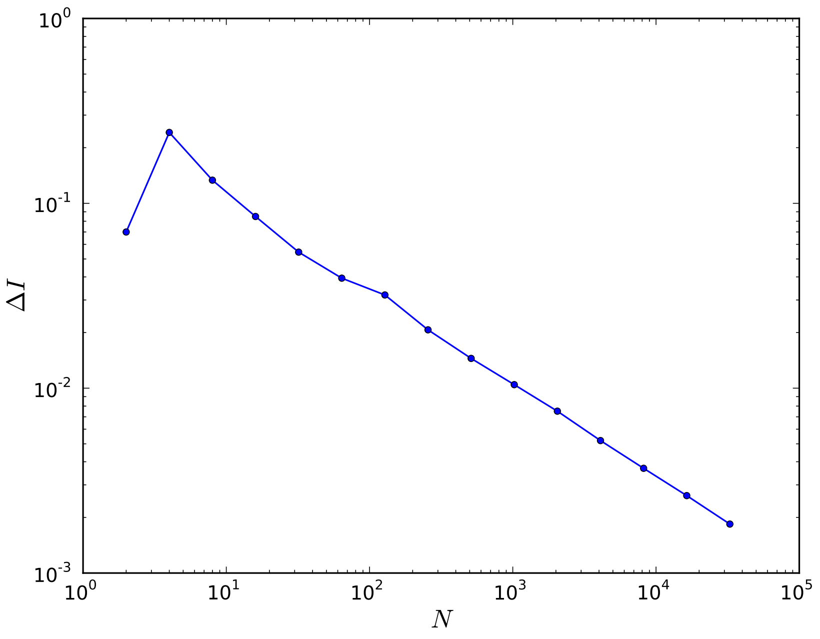

Figure 2: Errors of $E_N[\rho(0.3, 0.4)]$.

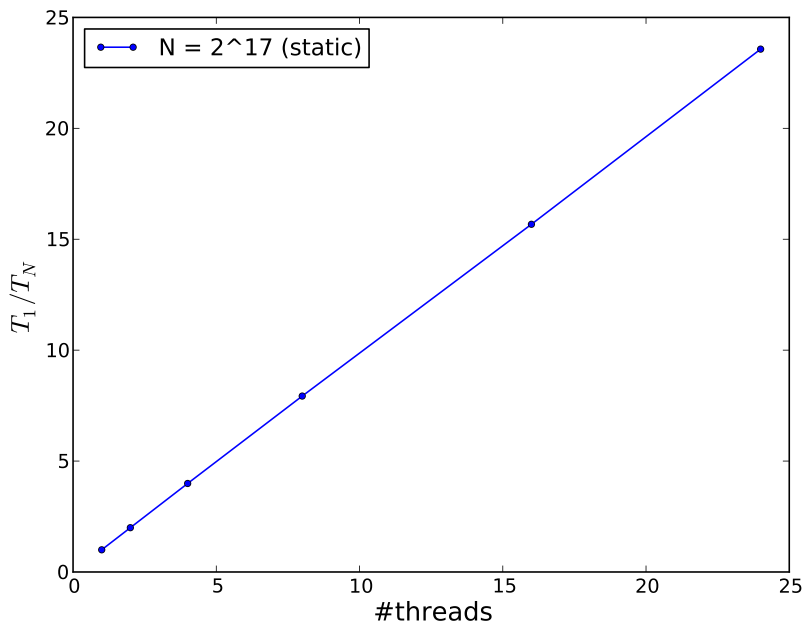

Figure 3: Strong scaling for $N = 2^{18}$ samples

[1] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller (1953). "Equation of State Calculations by Fast Computing Machines". *Journal of Chemical Physics*, 21 (6): 1087–1092. DOI: `http://dx.doi.org/10.1063/1.1699114`

and answer the following question:
What potential energy $V(|r_1 - r_2|)$ is used in the simulations done by Metropolis et al.?

Answer: $V(r) = \infty$ for $r \leq d_0$ and 0 otherwise.

Summary

Summarize your answers, results and plots into a PDF document. Furthermore, elucidate the main structure of the code and report possible code details that are relevant in terms of accuracy or performance. Send the PDF document and source code to your assigned teaching assistant.