Blue Waters Petascale Semester Curriculum v1.0

Unit 4: OpenMP

Lesson 8: Markov chains, Matrix multiply

Instructor Guide

Developed by Paul F. Hemler for the Shodor Education Foundation, Inc.

Browse and search the full curriculum at http://shodor.org/petascale/materials/semester-curriculum

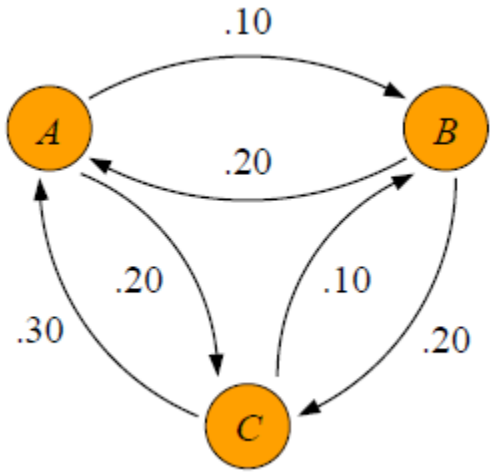
We welcome your improvements! You can submit your proposed changes to this material and the rest of the curriculum in our GitHub repository at https://github.com/shodor-education/petascale-semester-curriculum

We want to hear from you! Please let us know your experiences using this material by sending email to petascale@shodor.org

Markov Chains

Paul F. Hemler, Hampden-Sydney College

A Markov chain is a mathematical representation of transitions from one state to another state, which can be used to model real world phenomena including species loss in a community and gene location in DNA[1]. The transitions are based on a fixed probability so that when the system is in one state the probability of moving to another state is fixed. One way to visualize a Markov chain uses a state space, where the states are represented as circles and the probability of going into another state are edges. For example, suppose there were three vats of water, A, B, and C, containing 5 gallons, 10 gallons, and 15 gallons, respectively. After each hour, 10% of the water in A is transferred into to B and 20% in A is transferred into to C, 20% from B is transferred into A and 20% from B is transferred into C, and 30% from C is transferred into A and 10% from C is transferred into B. The following state space represents these transitions.



The question to be answered is, how much water is in each vat after a certain number of hours. A Markov Chain can be used to mathematically model this question.

The state space diagram can be represented as a two-dimensional matrix, called a transition matrix, where the columns represent the transition amounts, or the probabilities. Since the columns represent probabilities the elements in each column must sum to one because there must be a next state, even if it is the same state. The transition matrix for the above state space diagram is,

$$T = \begin{bmatrix} 0.7 & 0.2 & 0.3 \\ 0.1 & 0.6 & 0.1 \\ 0.2 & 0.2 & 0.6 \end{bmatrix}$$

The first column of the matrix indicates that 70% of the water stays in vat A after one hour, 10% of the water goes into vat B, and 20% of the water goes into vat C. Likewise for the second column, which represents vat B and the third column, which represents vat C. To say this another way, a given cell in the transition matrix, T[r][c] represents the percentage of water that is taken from vat *c* and transferred to vat *r*.

Using a vector to represent the initial amount of water in each vat,

$$V_0 = \begin{bmatrix} 5 \\ 10 \\ 15 \end{bmatrix}$$

The amount of water in each vat is determined by multiplying the transition matrix by the vector, or

$$V_1 = TV_0$$

Or, more specifically,

$$V_1 = \mathrm{TV}_0 = \begin{bmatrix} 0.7 & 0.2 & 0.3 \\ 0.1 & 0.6 & 0.1 \\ 0.2 & 0.2 & 0.6 \end{bmatrix} \begin{bmatrix} 5 \\ 10 \\ 15 \end{bmatrix} = \begin{bmatrix} 10 \\ 8 \\ 12 \end{bmatrix}$$

This equation says that after one hour there will be 10 gallons in vat *A*, 8 gallons in vat *B*, and 12 gallons in vat C.

The power using the Markov Chain model is that it does not matter where the various states came from, their next state is fully defined by the transition matrix. This means that to find the next state, or the contents of the vats after one more hour is determined using V_1 as the initial state. That is,

$$V_2 = \mathrm{TV}_1 = \begin{bmatrix} 0.7 & 0.2 & 0.3 \\ 0.1 & 0.6 & 0.1 \\ 0.2 & 0.2 & 0.6 \end{bmatrix} \begin{bmatrix} 10 \\ 8 \\ 12 \end{bmatrix} = \begin{bmatrix} 12.2 \\ 7 \\ 10.8 \end{bmatrix}$$

The process can be continually repeated to find the contents of the vats after any number of hours. Calculating the intermediate vat contents after two hours can be written as,

$$V_2 = T[TV_0] = TTV_0 = T^2V_0$$

The contents of the vats can be determined after *N* hours, without determining the intermediate values since,

$$V_n = T \ldots [TV_0] = T \ldots TV_0 = T^N V_0$$

where, T^N means multiplying T by itself N times. That is, an iterated matrix multiplication.

It is well know that further iterating the transition matrix does not cause it to change after some number of iterations. The resulting columns represent the long-term probabilities of state changes and when multiplied by the initial vector results in the final states. For example, the above transition matrix after 20 iterations is:

$$T^{20} = \begin{bmatrix} 0.4666 & 0.4666 & 0.4666 \\ 0.2 & 0.2 & 0..2 \\ 0.3333 & 0.3333 & 0.3333 \end{bmatrix}$$

And the steady-state vector is:

$$V_{20} = \begin{bmatrix} 14 \\ 6 \\ 10 \end{bmatrix}$$

This means the steady-state contents of vat A is 14 gallons, vat B is 6 gallons, and vat C is 10 gallons, regardless of the starting quantities.

Matrix-matrix multiplication

To multiply two matrices together, the number of columns of the matrix on the left must be the same as the number of rows of the matrix on the right. The final result of multiplying two matrices together is a third matrix, and the process can be described by the equation,

$$C = AB.$$

Matrix multiplication is a mechanical process where each element in the product matrix is formed by multiplying all the elements of a row with the corresponding elements of a column. For example, given the following two matrices,

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

The product matrix is given by:

$$C = \begin{bmatrix} 1(5) + 2(7) & 1(6) + 2(8) \\ 3(5) + 4(7) & 3(6) + 4(8) \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

In this module we will just multiply square matrices, which means all matrices have the same number of rows as columns. Formally, the *(i, j)* element in the resulting matrix *C* is determined by forming the sum of the element by element multiplication the i^{th} row of matrix *A* by the j^{th} column of matrix *B*. This can be written mathematically as:

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$$

This equation shows that each element in the product matrix requires *n* multiplications and *n-1* additions. Since there are n^2 computed elements, we say matrix multiplication is an n^3 operation, where *n* is the size of the matrix. When the matrices get large, matrix multiplication becomes prohibitively expensive and its execution time is greatly improved using parallel computing.

Program Design

The program is named ***matMult.c*** and it optionally takes two arguments. The first argument is the size of the square matrix and is expected to be 100, 200, …, 1400. The second argument is the number of threads that should be used when performing those parts of the program that can be made parallel. Some small default values are used if an argument is not specified.

The program then allocates the dynamic memory required to store four square matrices of the specified size, *n*. Row pointers are first allocated and then a block to hold all the elements is then allocated. This added level of indirection allows each element in the matrix to be accessed with two indices, as

A[i][j].

Some variables are then declared and initialized and the number of specified threads to be used in the parallel sections are then requested. At this point the program enters into a parallel section to resolve

the row pointers to refer to the first element in each row and each element is initialized to 0.0. This is all done in parallel using OpenMP. The statement,

```
#pragma omp parallel shared(A, B, C, D, r)
```

spawns the requested number of threads, with will be functioning until the closing brace of the parallel region. The statement

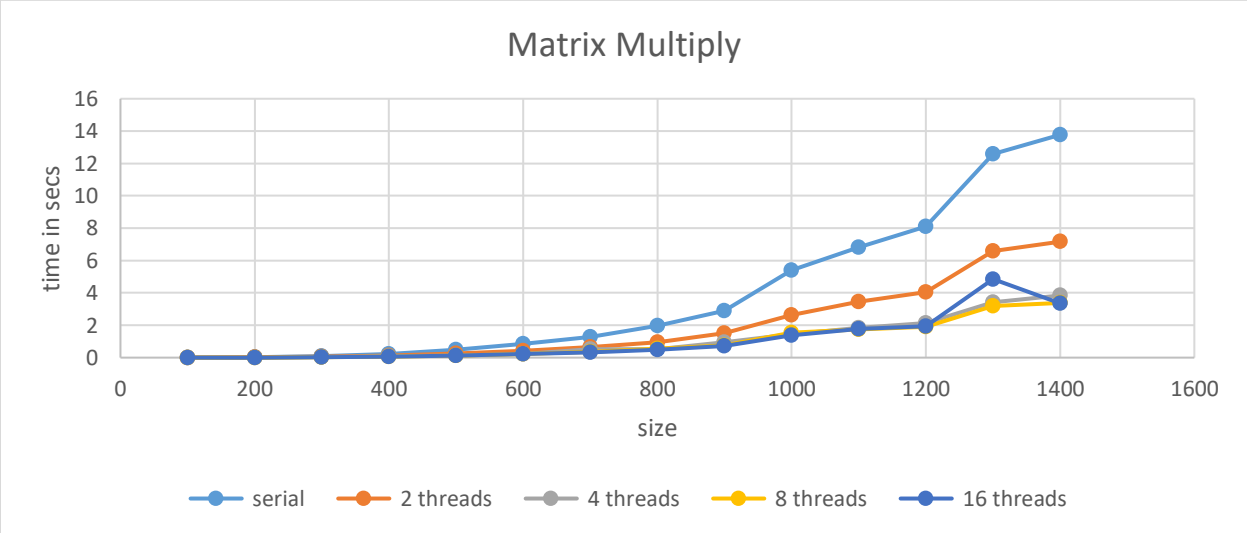
```
#pragma omp for
```

tells OpeMP to break the for loop into a number of blocks equal to the number of threads and gives each thread its own block to process. Once the row pointers are resolved and initialized, the elements of the two matrices A and B are then given some fractional values in parallel. One feature of OpenMP is that loop variables are made private to each thread without making a specific request, otherwise the row and column indices (r and c) would be required since each thread uses the same variable name but it must not be shared.

Once the two matrices A and B have been initialized a function to perform serial matrix multiplication is called and timed.

Results

A BASH shell script (runMatMult.sh) was written to automate determining the running time of matrix multiplication with various size matrices and various number of threads for the parallel version. A plot of the running time is shown in the figure below.



Clearly visible in the plot is the dramatic improvement in running time for the parallel version of matrix multiplication. This plot shows the computer performing the computation had hardware support for only four threads so when running with more threads than that no improvement was observed.

References

1) http://shodor.org/petascale/materials/UPModules/probableCause/