Blue Waters Petascale Semester Curriculum v1.0

Unit 4: OpenMP

Lesson 8: Markov chains, Matrix multiply

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

SERIAL AND PARALLEL MATRIX MULTIPLICATION

Learning Objectives

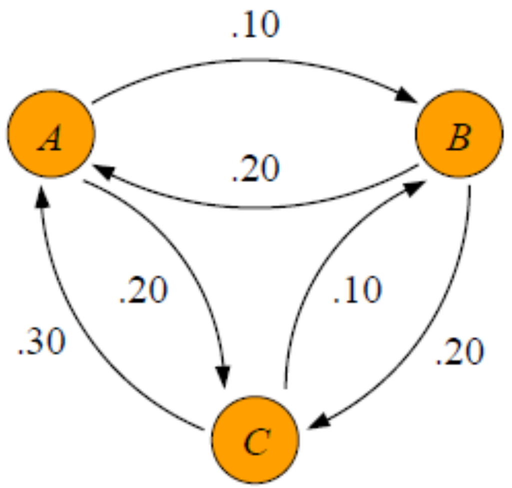
- The student should understand
  - The concept of a Markov Chain
  - The transition matrix
  - State changes using the transition matrix and a state vector (matrix-vector multiplication)
  - The steady state (iterated matrix-matrix multiplication)
  - How simple it ***can be*** to convert a serial program into a parallel program using OpenMP

Markov Chain

- A mathematical representation of a problem involving state transitions
- The transitions are based on a fixed probability
  - When the system is in one state the probability of moving to another state is fixed
- Useful to visualize a Markov chain with a state space diagram
  - Each state is represented as a node (vertex or circle)
  - Probabilities of transitioning from one state to another state
    - Weighted edges between the nodes

Markov Chain Example

- Three vats of water, *A*, *B*, and *C*
  - 10% of the water in *A* is transferred into *B* and 20% in *A* is transferred into to *C* in one hour
  - 20% of the water in *B* is transferred into *A* and 20% from *B* is transferred into *C* in one hour
  - 30% of the water in *C* is transferred into *A* and 10% from *C* is transferred into *B* in one hour
- The state-space diagram



Markov Chain Example

- The question to be answered is, how much water is in each vat after a certain number of hours
- The state space diagram can be represented as a two-dimensional matrix, called a transition matrix
  - The columns represent the transition amounts, or the probabilities
  - The columns must sum to one

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

  - T[r][c] represents the percentage of water that is taken from vat *c* and transferred to vat *r*

Markov Chain Example

- A vector is used to represent the initial amount of water in each vat
- For example, let vat *A* initially contain 5 gallons of water, vat *B* contain 10 gallons, and vat *C* contain 15 gallons

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

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

$$V_1 = TV_0$$

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

Markov Chain Example

- The previous equation states that after one hour vat *A* will contain 10 gallons, vat *B* will contain 8 gallons, and vat *C* will contain 12 gallons
- The amount of water in each vat after the next hour is found in a similar manner, using these new quantities as the initial conditions

$$V_2 = \mathrm{TV}_1 = T[TV_0] = \begin{bmatrix} 0.7 & 0.2 & 0.3 \\ 0.1 & 0.6 & 0.1 \\ 0.2 & 0.2 & 0.6 \end{bmatrix} \begin{bmatrix} 10 \\ 8 \\ 12 \end{bmatrix} = \begin{bmatrix} 12.2 \\ 7 \\ 10.8 \end{bmatrix}$$

- Likewise, the amount of water in each vat after *N* hours is

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

Markov Chain Example

- The previous equation demonstrates it is not necessary to computer each state to determine the state after *N* hours, but rather you need to multiply the transition matrix by itself *N* times

Matrix Multiplication

- When multiplying 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,

$$C = AB$$

- Where A, B, and C are matrices
- 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,

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

Matrix Multiplication

- Then

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

- Which can be written mathematically as

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

- Each element in the C matrix requires *n* multiplications
- Matrix multiplication is an n^3 operation, which is prohibitively expensive for large matrices

Matrix Multiplication

- The code to perform this computation is straightforward
  - For all rows and all columns, compute each element in the resulting matrix
  - This code assumes the matrix C is initialized with zeros

```
for (int r = 0; r < n; r++)
    for (int c = 0; c < n; c++)
        for (int k = 0; k < n; k++)
            C[r][c] += A[r][k] * B[k][c];
```

Parallel Matrix Multiplication

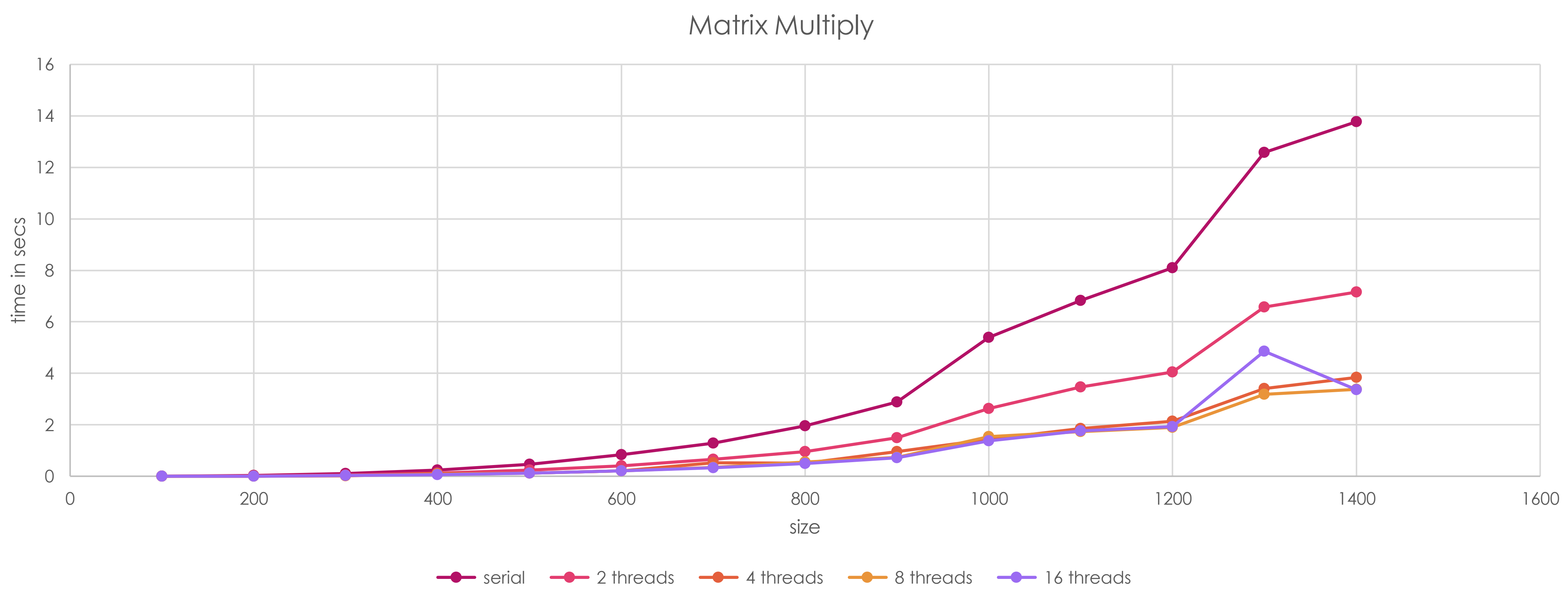
- Matrix multiplication is an example of an embarrassingly parallel problem
  - All computations are independent and can therefore be performed simultaneously
- The power of OpenMP is to simply convert serial code for an embarrassingly parallel problem into a parallel solution by adding a single directive to the serial code

```
#pragma omp parallel for shared(A, B, C, n)
    for (int r = 0; r < n; r++)
        for (int c = 0; c < n; c++)
            for (int k = 0; k < n; k++)
                C[r][c] += A[r][k] * B[k][c];
```

Parallel Matrix Multiplication

- The pragma tells the compiler to break up the *A* matrix into a number of blocks containing a similar number of rows, and have independent threads perform the computation on each block of rows simultaneously
  - Each thread performs fewer computations compared to the serial case, therefore improving the performance of the matrix multiply code
  - There is some overhead of performing the computation in parallel
    - The speedup is less than one over the number of threads
    - The speedup becomes more significant as the size of the matrix increases
    - The number of hardware threads also limits any performance improvement
- The following plot shows the time to perform matrix multiplication on various sized matrices using different numbers of threads

Parallel Matrix Multiplication Efficiency



Conclusion

- Matrix multiplication is the fundamental processing step when a problem can be mathematically modeled with a Markov Chain
- Matrix multiplication is an embarrassingly parallel computational problem that can be easily made parallel with OpenMP
- Dramatic improvements in computation time can be observed with a parallel version of matrix multiplication, especially as the size of the matrices get large