Project 2

- Part I Open MP

Problem 1 (Data Parallelism): The code from the last assignment models a sparse matrix vector multiply (updated in sparse_matvec.c). The matrix is sparse in that many of its elements are zero. Rather than representing all of these zeros which wastes storage, the code uses a representation called Compressed Row Storage (CRS), which only represents the nonzeros with auxiliary data structures to keep track of their location in the full array.

Given youwrite.c, develop an OpenMP implementation of this code for 4 threads. You will also need to modify the initialization code as described below, and add timer functions. You will need to evaluate the three different scheduling mechanisms, static, dynamic and guided, and for two different chunk sizes of your choosing.

I have provided three input matrices, sm1.txt, sm2.txt2 and sm3.txt3, which were generated from the MatrixMarket (see http://math.nist.gov/MatrixMarket/). The format for these is a sorted coordinate representation (row, col, value) and will need to be converted to CRS. Measure the execution time for the sequential code and all three parallel versions, all three data set sizes and both chunk sizes.

You will turn in the code, and a brief README file with the 21 different timings and an explanation of which strategies performed best and why.

Project 2, cont.

- Part I Open MP, cont.

Problem 2 (Task Parallelism): Producer-consumer codes represent a common form of a task parallelism where one task is "producing" values that another thread "consumes". It is often used with a stream of data to implement pipeline parallelism.

The program prodcons.c implements a producer/consumer sequential application where the producer is generating array elements, and the consumer is summing up their values. You should use OpenMP parallel sections to implement this producer-consumer model. You will also need a shared queue between the producer and consumer tasks to hold partial data, and synchronization to control access to the queue. Create two parallel versions: producing/consuming one value at a time, and producing/consuming 128 values at a time.

Measure performance of the sequential code and the two parallel implementations and include these measurements in your README file.

Project 2, cont.

- Part II Thread Building Blocks

As an Academic Alliance member, we have access to Intel assignments for ThreadBuildingBlocks. We will use the assignments from Intel, with provided code that needs to be modified to use TBB constructs. You will turn in just your solution code.

Problem 3 (Problem 1 in TBB.doc, Using parallel_for)

Summary: Parallelize "mxm_serial.cpp"

Problem 4 (Problem 3 in TBB.doc, Using recursive tasks)

Summary: Modify implementation in rec_main.cpp

All relevant files prepended with rec_ to avoid conflict.

Problem 5 (Problem 4 in TBB.doc, Using the concurrent_hash_map container)

Summary: Modify implementation in chm_main.cpp

All relevant files prepended with chm_ to avoid conflict.

Project 2, cont. Using OpenMP

- You can do your development on any machines, and use compilers available to you. However, the final measurements should be obtained on the quadcore systems in lab5. Here is how to invoke OpenMP for gcc and icc.
  - gcc: gcc –fopenmp prodcons.c
  - icc: icc –openmp prodcons.c