The Ministry of Education and Science of the Russian Federation

Lobachevsky State University of Nizhni Novgorod

Computing Mathematics and Cybernetics faculty

The competitiveness enhancement program

of the Lobachevsky State University of Nizhni Novgorod

among the world's research and education centers

Strategic initiative

“Achieving leading positions in the field of supercomputer technology

and high-performance computing”

NUMERICAL METHODS FOR SOLVING DIFFERENTIAL EQUATIONS

Practice 3. Solving Partial Differential Equations as Illustrated by the Problem of Compound Option Price Computation

Nizhni Novgorod

2014

OBJECTIVES

The purpose of this practice is the use of parallel algorithms of linear algebra for solving partial differential equations. This is illustrated by a quantitative finance problem.

ABSTRACT

This practice involves computation of the convertible option price whose behavior is described by means of a partial differential equation. To solve such an equation, the Crank-Nicolson scheme is used which involves solving a sequence of linear systems with a special tridiagonal matrix. The practice proposes to implement the tridiagonal matrix algorithm and parallelize the cyclic reduction method using OpenMP and TBB. You will compare efficiency of sequential implementation of methods and analyze scalability of parallel versions of the cyclic reduction method. You will also see the example how Intel Parallel Amplifier XE is used to find bottlenecks of the parallel version.

GUIDELINES

This practice involves computation of the convertible bond price whose behaviour is described by means of a partial differential equation.

The introductory part defines key notions used for numerical solution of differential equations. The following part states the problem of pricing *convertible bonds (CB)* - financial instruments that the holder can predetermined number of stocks with a pre-specified price, or to hold the bond till maturity to receive coupons and the principal [4]. The *CB price* is an amount that the holder can obtain by selling the bond. The CB price C_k at each time interval between two successive decisions of conversion t_k and t_{k+1} obeys a partial differential equations. The problem consists in determining the optimal CB price at each stage to enable the holder to covert the bond at best value. The stages are reviewed in reverse chronological order.

The practice shows how the Crank-Nicolson finite difference scheme is derived. This scheme is unconditionally stable, so it is often used to solve parabolic differential equations. Differencing requires M time partitions and J partitions for the bond value variation range (which is considered to be restricted). The difference scheme for the k^{th} time stage has the matrix form as follows:

$$A \times \left(C_{k,1}^{m}, C_{k,2}^{m}, \dots, C_{k,J-1}^{m}\right) = B \times \left(C_{k,0}^{m+1}, \dots, C_{k,J}^{m+1}\right) - \left(a_{21}C_{k,0}^{m}, 0, \dots, 0, a_{12}C_{k,J}^{m}\right),$$

where A is a tridiagonal matrix of order $(J-1)\times(J-1)$ and B is a tridiagonal matrix of order $(J+1)\times(J-1)$. Coefficients of matrices A and B have the same values at the same stage. Let us set the system $Ax = CB_{next}$. Its solving requires the use of the tridiagonal matrix algorithm and cyclic reduction method. See the section "Direct Methods for Solving Linear Systems" for a detailed description of these methods. This practice adapts the tridiagonal matrix algorithm and cyclic reduction method formulas for solving the posed problem.

It also features a sequential implementation of this problem. To enable sequential implementation, a number of auxiliary functions have to be developed for computation of coefficients for the matrices A and B, computation of boundary conditions, computation of the right-hand vector CB_{next} as a product of tridiagonal matrix and vector and CB pricing at the start time for each stock price value, computation of the interest rate paid to the holder at each moment of time, determination of the best CB price at the moment when the holder decides to sell the bond. One also has to implement the function for solving linear systems with the same values on their diagonals using the tridiagonal matrix algorithm and the function that implements the Crank-Nicolson scheme as applied to the initial problem. See a detailed description of the tridiagonal matrix algorithm and cyclic reduction method in the practice "Tridiagonal Matrix

Algorithm and Reduction Method as Applied to Linear System with Band Matrix". Solve the test problem to check the implementation for correctness.

Now let us modify the sequential version which uses the cyclic reduction method for solving linear systems. For each forward reduction iteration, equations that do not overlap with even-indexed ones are grouped in threes. From each group of three equations, odd-indexed variables are eliminated and variables are renumbered.The following iteration also eliminates the odd indexed variables. For each backward iteration, the values of odd variables are restored based on known values of even variables, i. e. the variables are recomputed in a reverse order. To check the implementation for correctness, you have to compare the results of the test problem with the tridiagonal matrix algorithm results.

After this, implement an OpenMP-based parallelized cyclic reduction algorithm for shared memory systems. As each next forward/backward iteration depends on the preceding one, computations cannot be parallelized by reduction level. However, within each iteration, an individual variable is eliminated and restored on its own thus enabling parallelization of embedded loops. For this purpose, insert OpemMP directive **`pragma omp parallel for`** before respective loops.

After this, implement an Intel TBB-based parallelized cyclic reduction algorithm for shared memory systems. This version also involves parallelization of embedded cyclic reduction iteration loops. To benefit from parallelization capabilities offered by TBB, initialize a **`tbb::task_scheduler_init`** class instance in the function **`main()`**. To parallelize loops with a known number of repeats, use **`tbb::parallel_for()`** whose inputs are the loop iteration space and function object. In this case, one can use the integrated single-dimension iteration space **`tbb::blocked_range`**. The function object is actually an extended loop body.

This practice includes efficiency analysis of the implemented algorithms. It also gives theoretical estimate of method runtimes depending on the number of partitions for the bond J value variation range. This estimate equals to $10J + O(1)$ for the tridiagonal matrix algorithm and $11J + O(\log J)$ for the cyclic reduction method. However, experimental results showed that the tridiagonal matrix algorithm runtime for matrices of greater dimensions is twice as long as that of the cyclic reduction method. This may be explained by the fact that despite the invariable, the number of operations for both algorithms in asymptotics is proportional to $O(J)$. Therefore, to compare the results one should consider the number of operations performed by the software implementation. Another theory is that the architecture used for experimental purposes has a significant influence on the result. To check it, use Intel Parallel Amplifier XE in the Basic Hostspots mode. Profiling showed that the average tridiagonal matrix algorithm operation takes about 2 clock periods, while a cyclic reduction method operation takes about 1 clock period. In this case, cyclic reduction requires about 1.3 more instructions. As the linear system is solved more than once, cyclic reduction requires less time to determine the best bond value than the tridiagonal matrix algorithm.

After this, analyze scalability of the OpenMP and TBB-based cyclic reduction method parallel implementations. Experimental results demonstrate low scalability of the program: in case of OpenMP, it reaches 2 for 4 through 8 threads and 1.3 through 1.7 for 4 through 8 threads. To analyze the experimental results, use Intel Parallel Amplifier XE tool in the Concurrency mode. The program profile shows high parallelism-related contingencies, i. e.thread expectation, renewal and stopping between loop iterations. Then analyze the character of caching, especially the number of cache misses. For this purpose, create a user-defined analysis type for selected events in Intel Parallel Amplifier XE. The results show thread competition for cache that affects the application efficiency.

RECOMMENDATIONS FOR STUDENTS

See [4] for a detailed problem description and concepts of financial mathematics. See [2, 3] for the methods for solving differential equations by the finite difference method. See [1] for the methods for solving linear systems and descriptions of the tridiagonal matrix algorithm and cyclic reduction method in particular.

REFERENCES

1. Golub G.H., Van Loan Ch. F. Matrix Computations. The John Hopkins University Press, 1996.
2. Hoffman J.D. Numerical Methods for Engineers and Scientists, 2nd Edition. New York: CRC Press, 2001.
3. Kincaid D.R., Cheney E.W. Numerical Analysis: Mathematics of Scientific Computing, 3rd Edition. Pacific Grove: Brooks Cole, 2001.
4. Gong. P, He. Z and Zhu. SP. Pricing convertible bonds based on a multi-stage compound option model, Physica A, 336, 2006, 449-462.

PRACTICE

1. Substantiate applicability of the tridiagonal matrix algorithm for solving linear systems with a tridiagonal matrix resulting from the Crank-Nicolson scheme construction for the problem of CB pricing.
2. Evaluate the approximation error for the Crank-Nicolson scheme constructed for the problem of CB pricing.
3. Add the respective Intel MKL function to solve the tridiagonal system. Evaluate efficiency of the library functions as compared to the tridiagonal matrix algorithm and the sequential cyclic reduction method.
4. Implement the parallel tridiagonal matrix algorithm. Evaluate the algorithm efficiency as compared to the ordinary tridiagonal matrix algorithm and parallel cyclic reduction method. Explain the experimental results.
5. Implement the block tridiagonal matrix algorithm. Evaluate the algorithm efficiency as compared to all the above methods for solving tridiagonal systems. Explain the experimental results. Evaluate scalability of the implemented algorithm.

TEST

1. How many times does one solve the differential equation describing bond price variation if the problem is posed for m^* interest payments within the CB lifetime and n holder's decisions to convert the bond ($n = km^*, k$ is an integer)?
    a. k
    b. m^*
    c. $+ n$
2. What is the order of solving the problem of pricing convertible bonds for each stage$[t_k, t_{k+1}]$ between two successive decisions of bond conversion?
    a. forward time
    b. + backward time
    c. the problem is solved only once
3. What is the size of a linear system matrix resulting from the use of the Crank-Nicolson scheme?
    a. *+ (J - 1)×(J - 1), J* is the number of partitions of the bond value variation range

b. $(M - 1) \times (M - 1)$, M is the number of partitions of the current time interval
c. $J \times J$, J is the number of partitions of the bond value variation range

4. How many cyclic reduction method iterations are required to solve a $N \times N$ linear system?
   a. + $\log_2(N)$
   b. $N / 2$
   c. $N / \log_2(N)$
5. How many forward operations of the tridiagonal matrix algorithm are required to solve a tridiagonal linear system with a $N \times N$ matrix?
   a. + $2N + 3$
   b. $8N - 14$
   c. $3N + 5$
6. What parameter of **`parallel for`** OpenMP directive has to be indicated to set the number of threads for parallel operation of the loop?
   a. threads_num
   b. there is no such a parameter
   c. + num_threads
7. What is the purpose of **`tbb::task_scheduler_init`** class?
   a. Creation of threads
   b. Creation of internal structures for the thread scheduler
   c. + Creation of threads and internal structures for the thread scheduler.
8. What is the function of the **`tbb::affinity_partitioner`** object?
   a. It divides the parallel loop iterations among threads
   b. + It divides and allocates the parallel loop iterations among threads
   c. It divides the parallel loop iterations among threads and selects the best size (**`grain_size`**) for each thread
9. What does L2_RQSTS.MISS mean when the program is analyzed using Intel Parallel Amplifier XE?
   a. + Number of L2 cache misses
   b. Number of L2 cache hits that did not hit L1 cache
   c. Number of RFO requests that did not hit L2 cache
10. What event enables finding the number of loads leading to last level cache misses in the course of program analysis using Intel Parallel Amplifier XE?
    a. MEM_LOAD_RETIRED.LLC_MISS_PS
    b. MEM_LOAD_RETIRED.L2_MISS
    c. + MEM_LOAD_RETIRED.LLC_MISS