Development of a Parallel Fast Fourier Transform Algorithm for Derivative Pricing Using MPI

Sajib Barua
CS. Dept. University of Manitoba
Course Professor: Dr. Ruppa K. Thulasiram

Outline
- Introduction
- Background and related work
- Problem Statement
- Solution Strategy and Implementation
- Experimental Platform
- Results
- Conclusions and Future Work

Introduction
- Derivatives are known as options
- Two basic types of option
  - Call Option
  - Put Option
- Two types of option depending on the exercise time:
  - American Option
  - European Option
- Determining the optimum exercise policy is a key issue in pricing American option to achieve maximum profit.

Background and Related Work
Methods used for option pricing

- Binomial Lattice method
- Monte Carlo approach
- Finite-Difference method
- Fast Fourier transform
- Finite-element method

- FFT is used recently to study multifactor model for option pricing problems and
- FFT is highly suitable for parallel computing.

Parallel FFT Algorithm

- Most Common parallel FFT Algorithm:
  - Cooley-Tukey Algorithm
  - Gentleman-Sande Algorithm

- Two Schemes for FFT Algorithm
  - Recursive Scheme
  - Iterative Scheme

Problem Statement

- There are two distinct features in this project:
  - mathematical treatment of the option pricing problem
  - and its computation

- Design, development and implementation of FFT algorithm with improved data locality.
- Use of this algorithm in option pricing problem with appropriate mapping

Challenges

- Computation:
  - In parallel system two types of latencies are incurred
    - Communication latency
    - Synchronization latency
  - FFT is inherently a synchronous algorithm.
  - Communication latency is circumvented by providing good data locality.

- Mathematical treatment:
  - Applying appropriate risk-neutral probability for finishing in-the-money
  - obtaining the Fourier Transform of the call price function with a known risk-neutral density function
  - etc
Solution Strategy and Implementation

- Parallel FFT Algorithm
- FFT in Option Pricing
- A New Parallel FFT Algorithm

How FFT Equation can be parallized?

- A sequence of $X$ of length $n$
  - $X = <X[0], X[1], ..., X[n-1]>$
- The DFT of this sequence is
  - $Y = <Y[0], Y[1], ..., Y[n-1]>$

How FFT Equation can be parallized?

Contd.

- Overall sequential complexity is $O(n \log n)$
- FFT calculation can be parallelized
  - A parallel FFT calculation requires $\log N$ number of stages
  - A parallel FFT algorithm with block data distribution of $N$ data points with $P$ processors requires
    - $(\log N - \log P)$ number of iterations for local calculation called local algorithm
    - $\log P$ number of stages for communication among processors called remote algorithm

Fourier Analysis in Option Pricing

- Call value is a function of the log of its strike price.
- This function is not square integrable. Its Fourier transform cannot be calculated.
- To get the analytically solvable Fourier transform, this function is multiplied by an exponential to make it square integrable.
- The call price function is
  - $\Psi_T(v)$ is the Fourier transform of the call price

- $\phi(s) = \frac{e^{-\sigma^2 s^2 t}}{2\sqrt{\pi s}}$ is the characteristic function of the underlying asset.
Contd.

- If $M = e^{i\alpha \pi/2}$ and $\omega = e^{i}$ then

- The discrete form of this equation will be

- This DFT equation can be parallelized.
- Call price can be calculated accurately and efficiently from a good parallel FFT algorithm with an efficient data distribution.

A New Parallel FFT Algorithm

Improving Data Locality Approach

Cooley-Tukey Algorithm

Parallel FFT Algorithm

With Improved Data Locality

Iteration No = i
1st mode index = j
2nd mode index = 2 \* i

Iteration 0
Iteration 1
Iteration 2

P0
P1
P2

$\omega^0$
$\omega^1$
$\omega^2$
$\omega^3$

$\omega^4$
$\omega^5$
$\omega^6$
$\omega^7$

$\omega^0$
$\omega^1$
$\omega^2$
$\omega^3$

$\omega^4$
$\omega^5$
$\omega^6$
$\omega^7$

$\omega^0$
$\omega^1$
$\omega^2$
$\omega^3$

$\omega^4$
$\omega^5$
$\omega^6$
$\omega^7$
Parallel FFT Algorithm

Experimental Platform...

Experimental Testbed or Environment

- **Programming Language**
  - Unix C
  - Message Passing Interface (MPI)

- **Single Processor**

- **Bewoulf Cluster (Distributed Architecture)**
  - 2 no of processors
  - 4 no of processors
  - 8 no of processors
Performance result with varying processor size

<table>
<thead>
<tr>
<th>No of Processors</th>
<th>Execution Time (in Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^12</td>
<td>0.02535093</td>
</tr>
<tr>
<td>2^13</td>
<td>0.05539596</td>
</tr>
<tr>
<td>2^15</td>
<td>0.2773631</td>
</tr>
<tr>
<td>2^16</td>
<td>0.615428</td>
</tr>
<tr>
<td>2^17</td>
<td>1.332534</td>
</tr>
<tr>
<td>2^18</td>
<td>2.87687</td>
</tr>
<tr>
<td>2^19</td>
<td>6.18839</td>
</tr>
<tr>
<td>2^20</td>
<td>11.83851</td>
</tr>
</tbody>
</table>

Performance result with varying Data size

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Execution Time (in Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^12</td>
<td>1 Processor</td>
</tr>
<tr>
<td>2^13</td>
<td>2 Processors</td>
</tr>
<tr>
<td>2^15</td>
<td>4 Processors</td>
</tr>
<tr>
<td>2^16</td>
<td>8 Processors</td>
</tr>
</tbody>
</table>

Comparison of Cooley-Tukey and Parallel FFT Algorithm

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Execution Time (in Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^12</td>
<td>Parallel FFT</td>
</tr>
<tr>
<td>2^13</td>
<td>Cooley-Tukey</td>
</tr>
</tbody>
</table>

Conclusions & Future Work
1. A New Parallel FFT algorithm (with improved data locality)
- Reduce communication overhead
- Performance is more than 15% over Cooley-Tukey Algorithm with larger data sets.

2. Mathematical Contribution
- Applying appropriate risk neutral density function
- Specification of dampening co-efficient
- Mathematical derivation of the dampened call price function and its Fourier Transform

Future Work
- Better risk neutral density function
- For now the results are presented only in terms of the performance improvement of the FFT algorithm. The results need to be presented in terms of the option prices
- Implementation of our algorithm on shared-memory architecture (OpenMP)
- More efficient node distribution technique and tuning parallel FFT algorithm.

Questions…?