

Parallel Programming (CS403)
Project-1
Simulation of Brownian Movement of a Stock
Price

Akhilesh Kumar (201351009)
Gaurav Tolani 201352021

September 28, 2016

Contents

1	Problem Statement	2
1.1	What is Brownian Movement?	2
1.2	What is Monte Carlo Simulation?	2
1.3	Geometric Brownian Motion	2
2	Algorithm Used	3
3	Output	3
3.1	Language Used	3
3.2	Outputs(Graphs and execution time for 'R')	3
3.3	Outputs(Graphs and execution time for 'C')	9
3.3.1	Time Matrix for different number of simulations	11
3.3.2	Graph Between number of iterations and corresponding time taken	12
3.3.3	Memory Allocation and Error Analysis	12
4	Task Dependency Graph	14
5	Conclusion	15
6	Future/Concluding Work	15
7	Appendix	15
7.1	How to install R	15
7.2	How to run the code	15

1 Problem Statement

1.1 What is Brownian Movement?

Standard definition of Brownian motion is:

Brownian motion is the random motion of particles suspended in a fluid (a liquid or a gas) resulting from their collision with the fast-moving atoms or molecules in the gas or liquid.

Dwelling into probability and statistics, Brownian motion is among the simplest of the continuous-time stochastic (or probabilistic) processes.

In mathematics, the Wiener process is a continuous-time stochastic process named in honor of Norbert Wiener. It is often called standard Brownian Motion.

1.2 What is Monte Carlo Simulation?

Monte Carlo methods (or Monte Carlo experiments) are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. Their essential idea is using randomness to solve problems that might be deterministic in principle.

A Monte Carlo simulation is an attempt to predict the future many times over. At the end of the simulation, thousands or millions of "random trials" produce a distribution of outcomes that can be analyzed.

1.3 Geometric Brownian Motion

A geometric Brownian motion (GBM) (also known as exponential Brownian motion) is a continuous-time stochastic process in which the logarithm of the randomly varying quantity follows a Brownian motion (also called a Wiener process) with drift. It is an important example of stochastic processes satisfying a stochastic differential equation (SDE); in particular, it is used in mathematical finance to model stock prices in the Black-Scholes model.

The geometric Brownian motion (GBM) is the most basic processes in financial modelling.

The formula for GBM is found below, where "S" is the stock price, " μ " (the Greek mu) is the expected return, " σ " (Greek sigma) is the standard deviation of returns, "t" is time, and " ε " (Greek epsilon) is the random variable:

$$\frac{\Delta S}{S} = \mu t + \sigma \varepsilon \sqrt{\Delta t} \quad (1)$$

If we change and rearrange the formula and take log of the above equation on both sides, the final equation which we will get is:

$$S(t) = S(0)e^{(\mu - \sigma^2/2)t + \sigma W(t)} \quad (2)$$

where, $W(t) = \text{Brownian Motion}$, $\sqrt{t}z(t)$, and

$z(t)$ = normal random variable b/w 0 and 1 with mean=0 and variance=1

In this problem, we will try to simulate the Brownian motion of a stock price for a given time using Monte Carlo simulations for different number of iterations and parallelize the algorithm to see the performance change.

2 Algorithm Used

An algorithm for simulating the stock price at time t , given that current price at time $t=0$ is S_0 is as follows:

1. Generate random variable $z \sim N(0,1)$
2. Set $\mu_t = (\mu - \sigma^2/2)t$ and $\sigma_t = \sigma t^{0.5}$
3. Set $S_t = S_0 \times e^{\mu_t + \sigma_t z}$

3 Output

3.1 Language Used

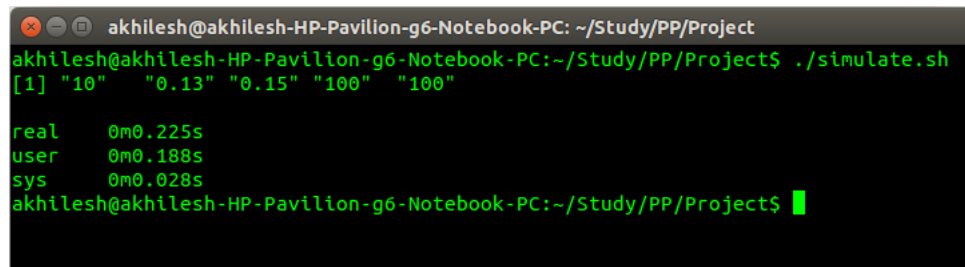
Due to time constraint, for the time being, we have implemented the serial code on the language R and C. R is a programming language and software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis.

Because of the some inbuilt features which R provide (eg. inbuilt function for generating random variable and graph plotting), the effort for making code was considerably reduced.

3.2 Outputs(Graphs and execution time for 'R')

Given the parameters,
 S_0 , initial price of stock= 10,
 μ , expected return=13%,
 σ , standard deviation of returns=15%,
time, $t=100$

1. For 100 simulations,



```
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC: ~/Study/PP/Project
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC:~/Study/PP/Project$ ./simulate.sh
[1] "10" "0.13" "0.15" "100" "100"

real    0m0.225s
user    0m0.188s
sys     0m0.028s
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC:~/Study/PP/Project$
```

Figure 1: Execution time for 100 simulations

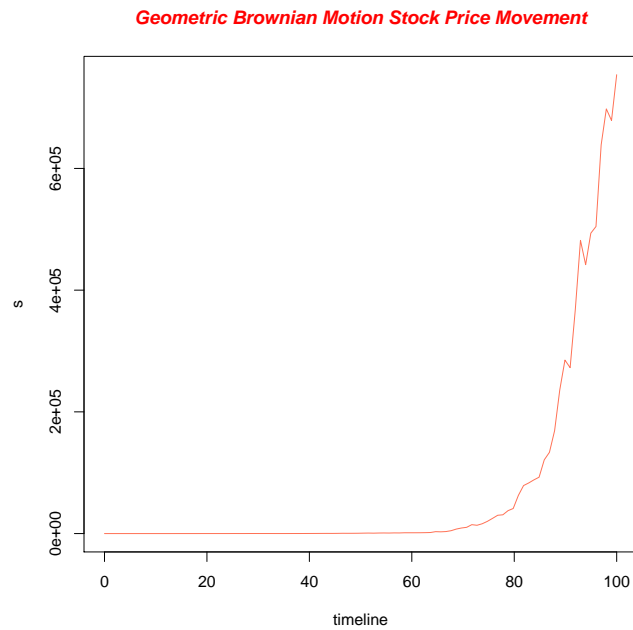


Figure 2: Brownian motion graph for 100 simulations

2. For 1000 simulations,

```
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC: ~/Study/PP/Project
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC:~/Study/PP/Project$ ./simulate.sh
[1] "10" "0.13" "0.15" "100" "1000"

real    0m0.255s
user    0m0.208s
sys     0m0.032s
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC:~/Study/PP/Project$
```

Figure 3: Execution time for 1000 simulations

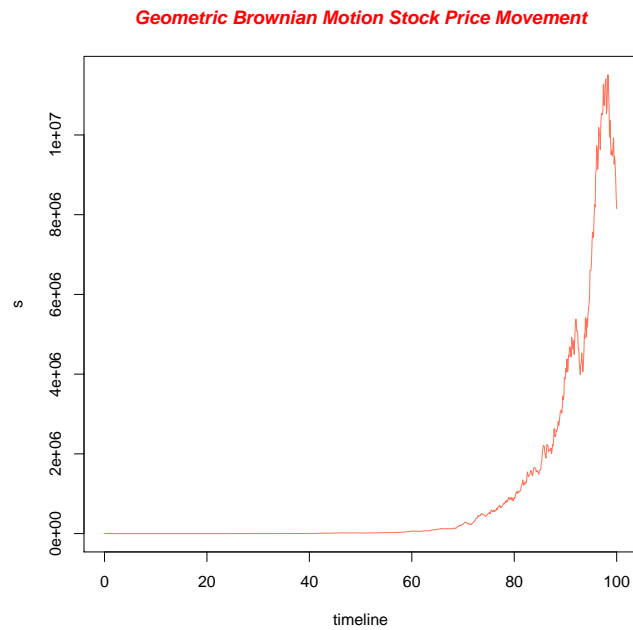


Figure 4: Brownian motion graph for 1000 simulations

3. For 10000 simulations,

```
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC: ~/Study/PP/Project
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC:~/Study/PP/Project$ ./simulate.sh
[1] "10"      "0.13"    "0.15"    "100"     "10000"

real    0m0.405s
user    0m0.356s
sys     0m0.036s
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC:~/Study/PP/Project$ █
```

Figure 5: Execution time for 10000 simulations

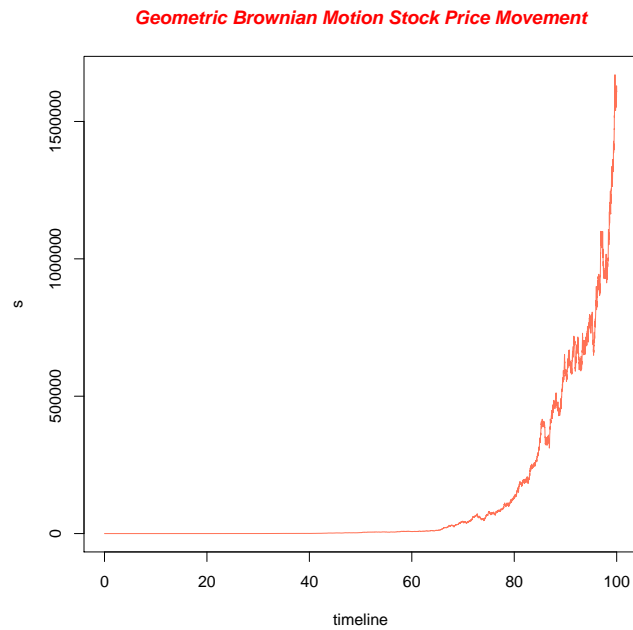


Figure 6: Brownian motion graph for 10000 simulations

4. For 100000 simulations,

```
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC: ~/Study/PP/Project
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC:~/Study/PP/Project$ ./simulate.sh
[1] "10"      "0.13"    "0.15"    "100"     "100000"

real    0m1.937s
user    0m1.880s
sys     0m0.040s
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC:~/Study/PP/Project$
```

Figure 7: Execution time for 100000 simulations

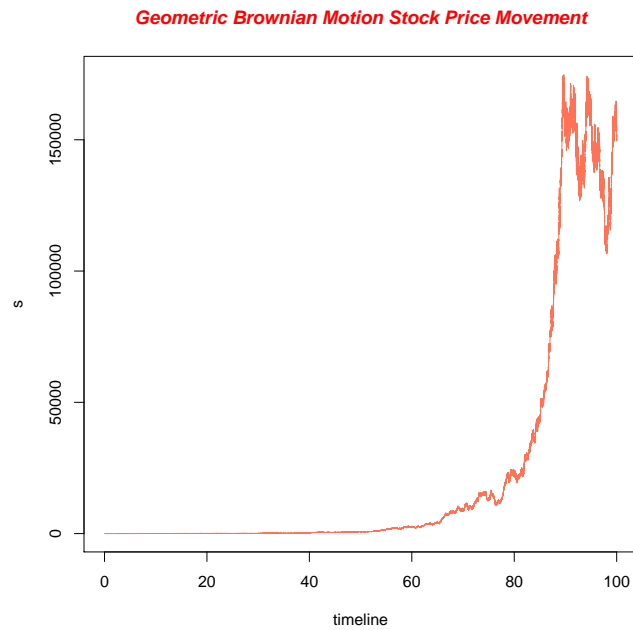


Figure 8: Brownian motion graph for 100000 simulations

5. For 1000000 simulations,

```
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC: ~/Study/PP/Project
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC:~/Study/PP/Project$ ./simulate.sh
[1] "10"      "0.13"     "0.15"     "100"      "1000000"

real    0m15.607s
user    0m15.436s
sys     0m0.140s
akhilesh@akhilesh-HP-Pavilion-g6-Notebook-PC:~/Study/PP/Project$
```

Figure 9: Execution time for 1000000 simulations

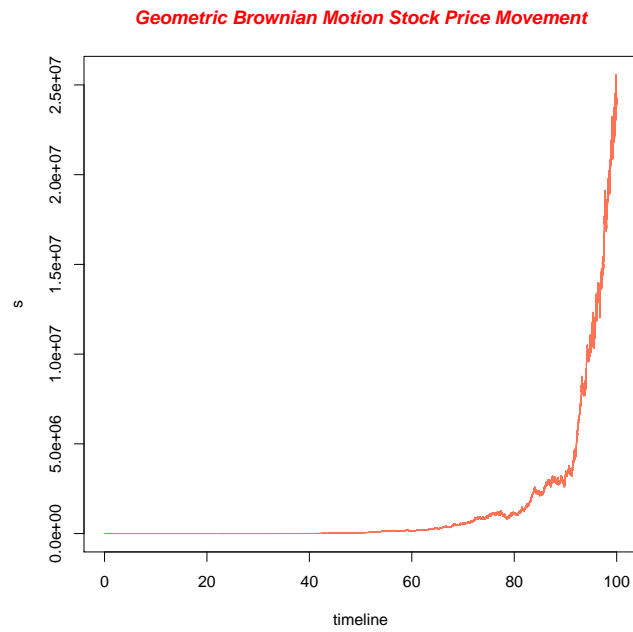


Figure 10: Brownian motion graph for 1000000 simulations

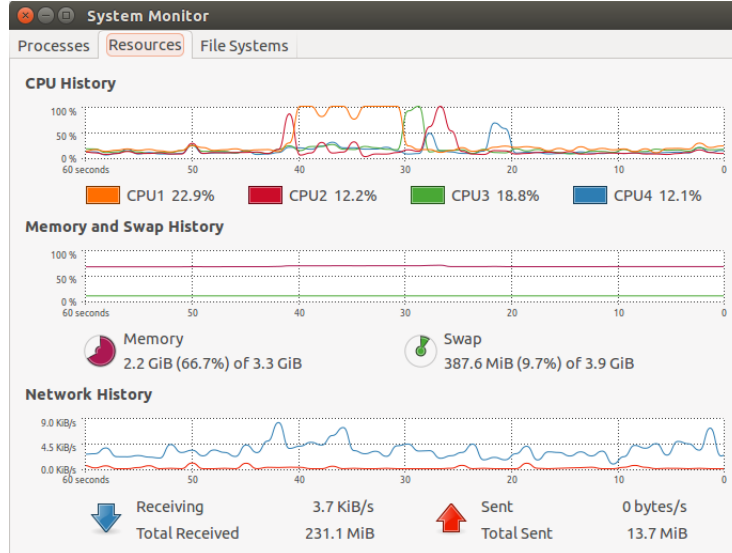


Figure 11: System Monitor for 1000000 simulations

3.3 Outputs(Graphs and execution time for 'C')

Given the parameters,

S_0 , initial price of stock= 10,

μ , expected return=13%,

σ , standard deviation of returns=15%,

time, $t=100$

1. For 100 simulations,

```
gaurav@Gaurav:~/Desktop/1$ gcc GBM_seq.c -o GBM_seq -lm
gaurav@Gaurav:~/Desktop/1$ time ./GBM_seq
Enter Initial stock price : $10
Enter the expected return: 13
Enter the standard deviation of returns: 15
Enter the time: 100
Enter number of simulations: 100

real    0m5.313s
user    0m0.000s
sys     0m0.000s
```

Figure 12: Execution time for 100 simulations

2. For 1000 simulations,

```
gaurav@Gaurav:~/Desktop/1$ gcc GBM_seq.c -o GBM_seq -lm
gaurav@Gaurav:~/Desktop/1$ time ./GBM_seq
Enter Initial stock price : $10
Enter the expected return: 13
Enter the standard deviation of returns: 15
Enter the time: 100
Enter number of simulations: 1000

real    0m4.979s
user    0m0.004s
sys     0m0.004s
```

Figure 13: Execution time for 1000 simulations

3. For 10000 simulations,

```
gaurav@Gaurav:~/Desktop/1$ gcc GBM_seq.c -o GBM_seq -lm
gaurav@Gaurav:~/Desktop/1$ time ./GBM_seq
Enter Initial stock price : $10
Enter the expected return: 13
Enter the standard deviation of returns: 15
Enter the time: 100
Enter number of simulations: 10000

real    0m5.783s
user    0m0.024s
sys     0m0.000s
```

Figure 14: Execution time for 10000 simulations

4. For 100000 simulations,

```
gaurav@Gaurav:~/Desktop/1$ gcc GBM_seq.c -o GBM_seq -lm
gaurav@Gaurav:~/Desktop/1$ time ./GBM_seq
Enter Initial stock price : $10
Enter the expected return: 13
Enter the standard deviation of returns: 15
Enter the time: 100
Enter number of simulations: 100000

real    0m5.803s
user    0m0.232s
sys     0m0.000s
```

Figure 15: Execution time for 100000 simulations

5. For 1000000 simulations,

```
gaurav@Gaurav:~/Desktop/1$ gcc GBM_seq.c -o GBM_seq -lm
gaurav@Gaurav:~/Desktop/1$ time ./GBM_seq
Enter Initial stock price : $10
Enter the expected return: 13
Enter the standard deviation of returns: 15
Enter the time: 100
Enter number of simulations: 1000000

real    0m9.808s
user    0m2.208s
sys     0m0.008s
```

Figure 16: Execution time for 1000000 simulations

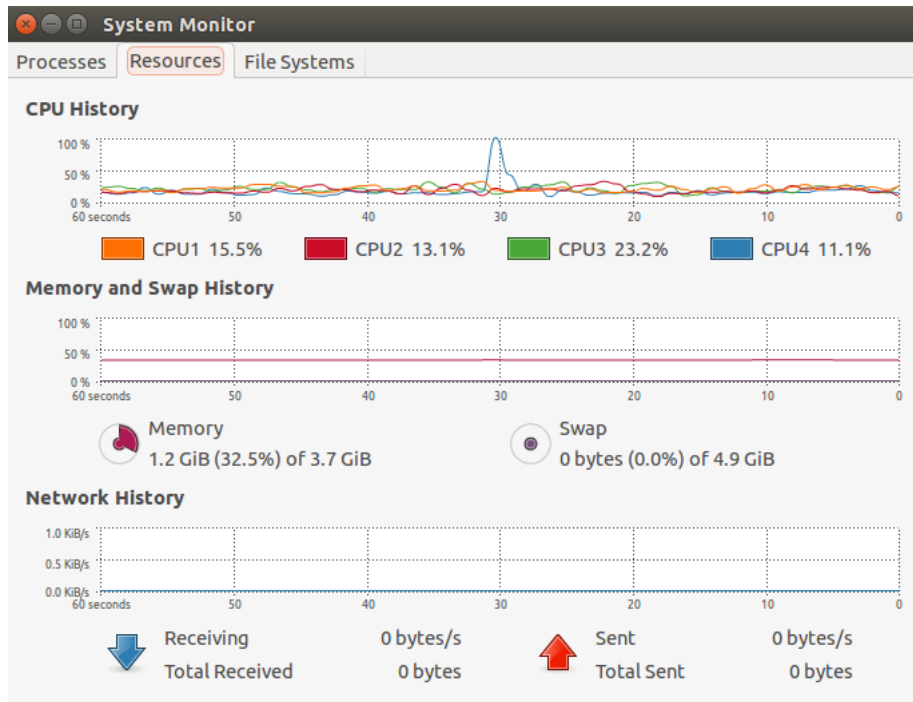


Figure 17: System Monitor for 1000000 simulations

3.3.1 Time Matrix for different number of simulations

Number of Iterations	100	1000	10000	100000	1000000
Time(In Seconds)	5.313	4.979	5.783	5.803	9.808

3.3.2 Graph Between number of iterations and corresponding time taken

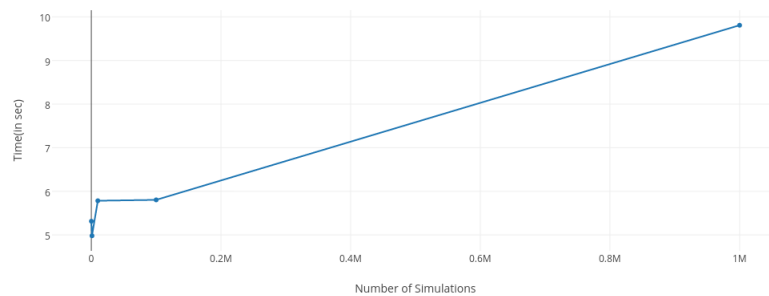


Figure 18: Graph between Number of Simulations and Time(In seconds)

3.3.3 Memory Allocation and Error Analysis

Valgrind : It is a memory tool that will alert you about memory bugs and point you to where the problem might be.

Leaks vs Errors : Leaks occur when you allocate(malloc) memory that you fail to free and Errors occur when you attempt to read or write memory that you do not have access to (e.g. writing off the end of an array)

```
gaurav@Gaurav:~/Desktop/1$ gcc GBM_seq.c -o GBM_seq -lm -g
gaurav@Gaurav:~/Desktop/1$ valgrind ./GBM_seq
==12955== Memcheck, a memory error detector
==12955== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==12955== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==12955== Command: ./GBM_seq
==12955==
Enter Initial stock price : $10
Enter the expected return: 13
Enter the standard deviation of returns: 15
Enter the time: 10
Enter number of simulations: 100
==12955==
==12955== HEAP SUMMARY:
==12955==   in use at exit: 0 bytes in 0 blocks
==12955==   total heap usage: 3 allocs, 3 frees, 24,000,352 bytes allocated
==12955==
==12955== All heap blocks were freed -- no leaks are possible
==12955==
==12955== For counts of detected and suppressed errors, rerun with: -v
==12955== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 19: Valgrind Memory Check Analysis of Serial Code

Explanation: There are Generally Two types of errors in memory check method

(i)Invalid Read of size X(When we try to access a memory that we don't have access to).

(ii)Invalid write of size X(When we try to write into a memory that we don't have access to)

In the serial code of GBM, we neither have any Invalid reads not Invalid writes(refer Figure 19).

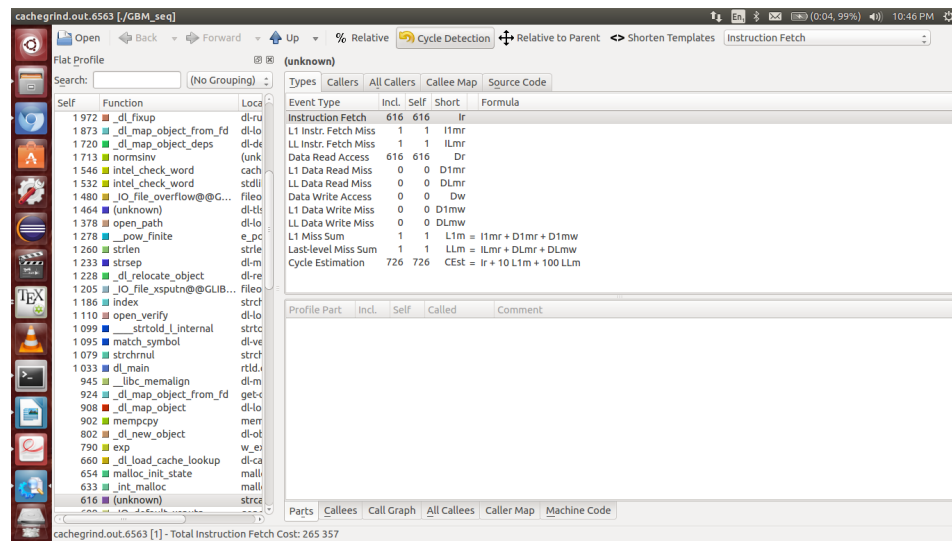


Figure 20: KCache Grind Analysis of Serial code

4 Task Dependency Graph

As There is only quantity that is being calculated for each simulation i.e. $W(t)$.

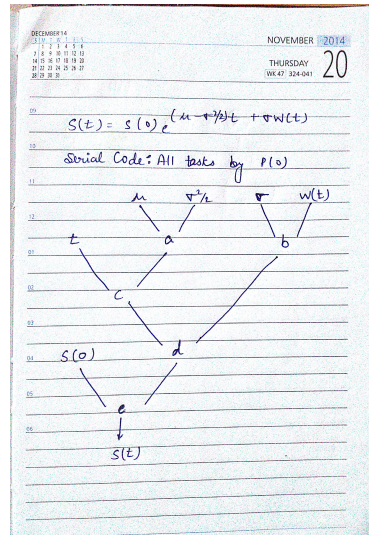


Figure 21: Task Dependency Graph for Serial Code

5 Conclusion

We observe that with increasing number of simulations, execution time also increase. Also, we also observe that while executing the code for 10,00,000 simulations, one or two cores reach their 100% processing power. This shows that other cores are idle and their execution capabilities are not utilized by the code. So, the code written is not optimised to utilize the full potential of the machine.

6 Future/Concluding Work

As of now, the code written is in R programming language. A serial code for this algorithm will be written in C and then will be evaluated for performance. After that, using parallel programming APIs like openMP, we will try to parallelize the code and compute the speedup. We can also compare whether for this kind of computational heavy algorithm, parallel code in C is better or serial code in R is better, performance wise.

7 Appendix

7.1 How to install R

Run the following commands in your machine(OS: Ubuntu 14.04):

1. `sudo sh -c 'echo "deb http://cran.rstudio.com/bin/linux/ubuntu trusty/"`
`>> /etc/apt/sources.list'`
2. `gpg --keyserver keyserver.ubuntu.com --recv-key E084DAB9`
3. `gpg -a --export E084DAB9 -- sudo apt-key add -`
4. `sudo apt-get update`
5. `sudo apt-get -y install r-base`

7.2 How to run the code

Attached are two files:

- GBMStock.R
- simulate.sh

In the simulate.sh file, there are 5 args which are required by the R Script to run. The args stands for:

GBMStock.R "Initial Price of Stock" "mu" "sigma" "Time" "Iterations"

In the terminal, run the **simulate.sh** file by manipulating the arguments and the output will be shown there. In the directory where the R Script is present, a new file "Rplots.pdf" will be created with the corresponding graph in it.