Simulation of Conditional Value-at-Risk via Parallelism on Graphic Processing Units

Hai Lan

Dept. of Management Science
Shanghai Jiao Tong University
Shanghai, 200052, China

July 22, 2012

Outline of Topics

- Motivation
- Sequential procedure of CVaR via nested simulation
- Paralleled nested simulation with screening
- Experimental performance
- Conclusion & Question

Motivations

- Some simulation procedures take days, weeks or even months to run.
- Traditional supercomputers are inaccessible for most researchers and practitioners.
- New development of high performance computing catches out attention:
  - cloud computing: heterogeneous, massive storage, massive computing
  - parallel computing on GPUs: homogeneous, data light, moderate computing
- Characteristics of simulation in financial engineering: data security,moderate size computing.

GPU Computing

Characteristics of GPU Computing: same program on different data, which makes it attractive for "embarrassingly paralleled" simulation work.

Reported work shows

- 20% – 800% faster for sorting (key only)
- 30 – 50 times faster for random number generating
- almost 100 faster for Monte Carlo simulation of Black Shorel model

GPU Computing

Characteristics of GPU Computing: same program on different data, which makes it attractive for "embarrassingly paralleled" simulation work.

Reported work shows

- 20% – 800% faster for sorting (key only)
- 30 – 50 times faster for random number generating
- almost 100 faster for Monte Carlo simulation of Black Shorel model

How about its effectiveness on complex simulation such as nested simulation?

Risk Measurement

- V is the dollar change of a portfolio over T time.
- Risk Measure: from distribution F_V (especially the loss part) $\rightarrow$ one number.

Risk Measurement

- V is the dollar change of a portfolio over T time.
- Risk Measure: from distribution F_V (especially the loss part) $\rightarrow$ one number.
- F_V^{-1} the right continuous inverse of F_V.

Risk Measurement

- V is the dollar change of a portfolio over T time.
- Risk Measure: from distribution F_V (especially the loss part) $\rightarrow$ one number.
- F_V^{-1} the right continuous inverse of F_V.
- $\mathrm{VaR}_p = F_V^{-1}(p)$ is the p-quantile of F_V.

Risk Measurement

- V is the dollar change of a portfolio over T time.
- Risk Measure: from distribution F_V (especially the loss part) $\rightarrow$ one number.
- F_V^{-1} the right continuous inverse of F_V.
- $\mathrm{VaR}_p = F_V^{-1}(p)$ is the p-quantile of F_V.
- $\mathrm{CVaR}_p = \int_p^1 F_V^{-1}(s)ds$.

Nested Simulation

Goal: provide a framework to estimate the market risk of an investment portfolio

Nested Simulation

Goal: provide a framework to estimate the market risk of an investment portfolio

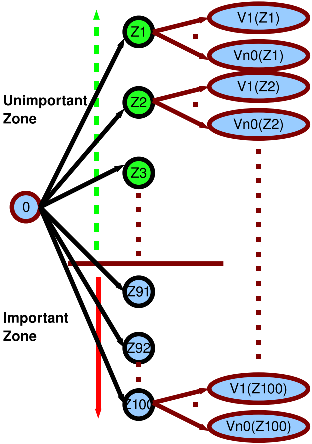
- outer level: generating $Z(T)$ from market scenario model in the real world probability measure.
  underlying stock price $S_i(T), i = 1, \cdots, k$.

Nested Simulation

Goal: provide a framework to estimate the market risk of an investment portfolio

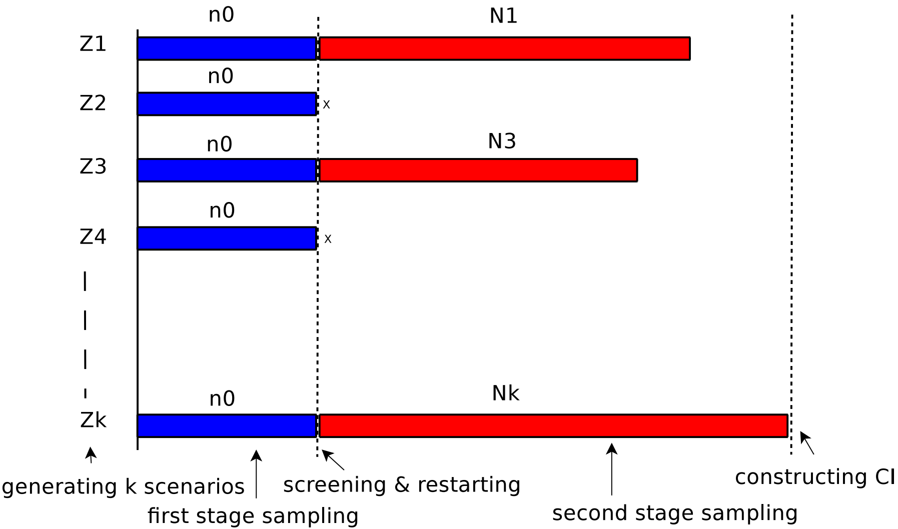
- outer level: generating $Z(T)$ from market scenario model in the real world probability measure.
  underlying stock price $S_i(T), i = 1, \cdots, k$.
- inner level: estimating V conditionally on $Z(T)$ by Monte Carlo Simulation in risk neutral probability measure.
  generate the underlying stock prices
  $S(T \leq t \leq U)|S_i(T),\ i = 1, \cdots, k$

"Plain" Nested Simulation



- Simulate scenarios $Z_1, \cdots, Z_k$. Unknown true value $V_i = E[X|Z = Z_i]$.
- Simulate payoffs $X_{i1}, \cdots, X_{in_0}$ given Z_i. Estimate V_i by $(X_{i1} + \cdots + X_{in_0})/n_0$.

Nested Simulation with Screening



Screening and Restarting

- Motivation: Only the loss part is interesting.
- Identified as subset selection with uncommon variance

Screening and Restarting

- Motivation: Only the loss part is interesting.
- Identified as subset selection with uncommon variance
- The comparison of Scenario i and j is taken as a hypothesis test:

$$H_o : V_i > V_j \text{ v.s.} H_1 : V_i \leq V_j$$

  we say, j is beaten by i if H_o is true.
- Screening can be taken as multiple comparisons of each pairs of market scenarios. A market scenario will survive from screening only when it is beaten less than $\lceil kp \rceil$ times of such comparisons.

Screening and Restarting

- Motivation: Only the loss part is interesting.
- Identified as subset selection with uncommon variance
- The comparison of Scenario i and j is taken as a hypothesis test:

$$H_o : V_i > V_j \text{ v.s.} H_1 : V_i \leq V_j$$

  we say, j is beaten by i if H_o is true.
- Screening can be taken as multiple comparisons of each pairs of market scenarios. A market scenario will survive from screening only when it is beaten less than $\lceil kp \rceil$ times of such comparisons.
- Restarting is used to avoid the selection bias inherited from screening.

Embarrassingly Paralleled: Generating Scenarios and 1st Stage Conditional Payoffs

Random Number Generator on GPU:

- long period, multiple streams and sub-streams.
- 32 bits computation only
- combined multiple recursive generator

Embarrassingly Paralleled: Generating Scenarios and 1st Stage Conditional Payoffs

Random Number Generator on GPU:

- long period, multiple streams and sub-streams.
- 32 bits computation only
- combined multiple recursive generator

Gaussian Random Variable: Muller-Box method to generate normal variables

Embarrassingly Paralleled: Generating Scenarios and 1st Stage Conditional Payoffs

Random Number Generator on GPU:

- long period, multiple streams and sub-streams.
- 32 bits computation only
- combined multiple recursive generator

Gaussian Random Variable: Muller-Box method to generate normal variables

Each thread equipped with one random number generator starting with the same/different seeds when common/independent random numbers applied.

Embarrassingly Paralleled: Generating Scenarios and 1st Stage Conditional Payoffs

Random Number Generator on GPU:

- long period, multiple streams and sub-streams.
- 32 bits computation only
- combined multiple recursive generator

Gaussian Random Variable: Muller-Box method to generate normal variables

Each thread equipped with one random number generator starting with the same/different seeds when common/independent random numbers applied.

Experiments show 50-60 times faster than the sequential procedure.

Parallel Screening 1: Strategy

Sequential procedure:

- for any scenario $i = 1, \ldots, k$, compute at least $\lceil kp \rceil$ variances of difference $S_{ij}^2 = \sum_{l=1}^{n_0} \frac{(X_{il} - X_{jl})^2}{n_0 - 1} - \frac{\sum_{l=1}^{n_0}(X_{il} - X_{jl}) \sum_{l=1}^{n_0}(X_{il} - X_{jl})}{n_0(n_0 - 1)}$
- scenario i is screened out if it is beaten no less than $\lceil kp \rceil$ times.

thus, a lot of data communication has to be done. To save data communication, we adopt a "divide-and-conquer" strategy.

Parallel Screening 1: Strategy

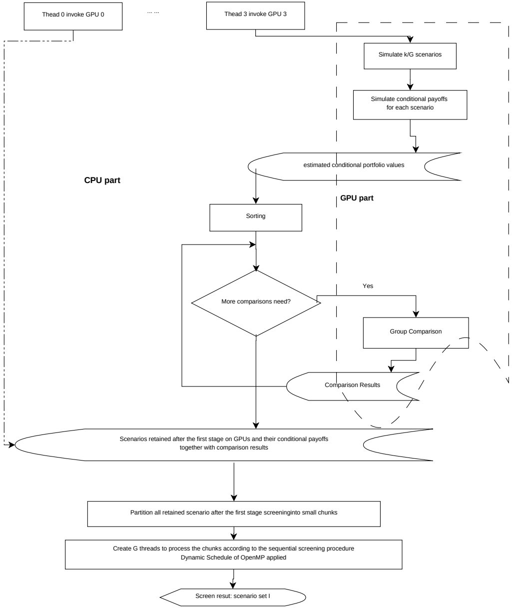
Sequential procedure:

- for any scenario $i = 1, \dots, k$, compute at least $\lceil kp \rceil$ variances of difference $S_{ij}^2 = \sum_{l=1}^{n_0} \frac{(X_{il} - X_{jl})^2}{n_0 - 1} - \frac{\sum_{l=1}^{n_0}(X_{il} - X_{jl}) \sum_{l=1}^{n_0}(X_{il} - X_{jl})}{n_0(n_0 - 1)}$
- scenario i is screened out if it is beaten no less than $\lceil kp \rceil$ times.

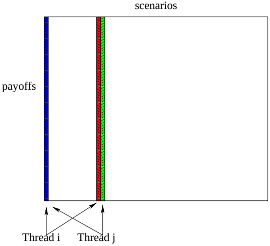
thus, a lot of data communication has to be done. To save data communication, we adopt a "divide-and-conquer" strategy.

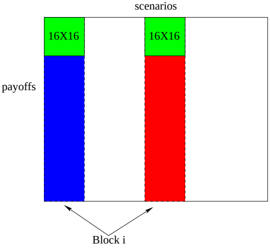
- randomly group the total k market scenario into G groups, each GPU processes one group.
- apply screening procedure within each group to eliminate some unimportant market scenarios.
- collect remaining scenarios from each GPU and perform screening again.

Parallel Screening 2: Flow Chart

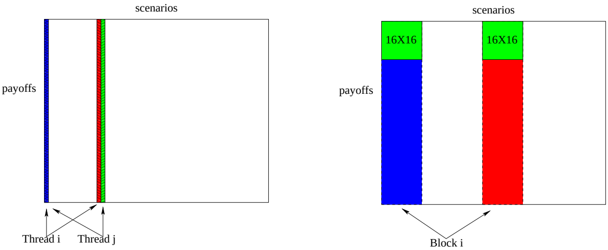


Parallel Screening 3: Implementation on GPU



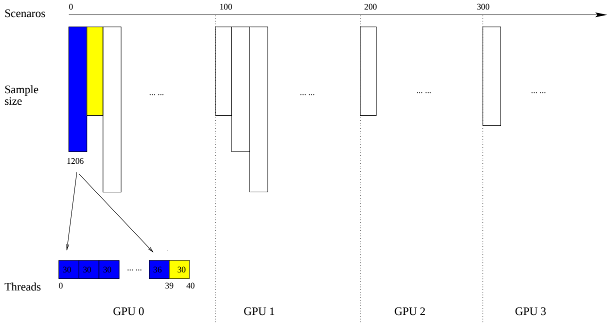


Parallel Screening 3: Implementation on GPU

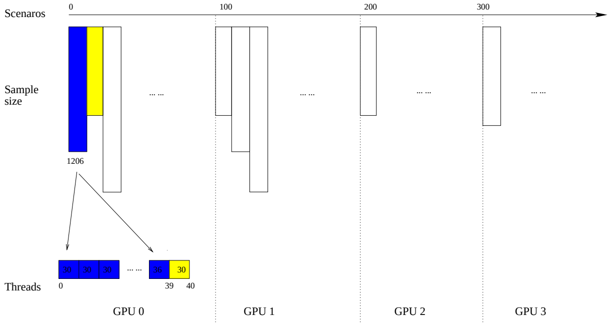


Scheme 1 is about 2 times faster than CPU only, scheme 2 is 30-50 times faster.

Second Stage Sampling



Second Stage Sampling

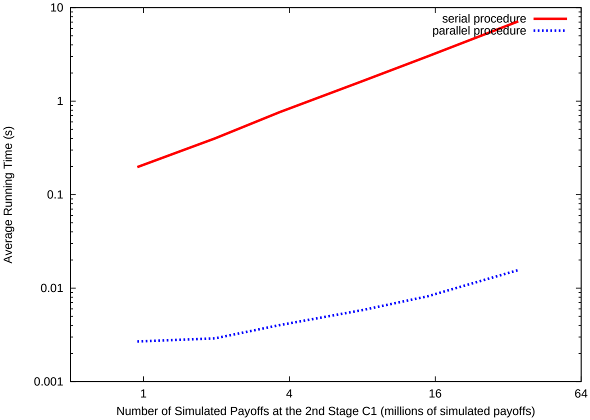


- balance the uneven load among different GPUs and threads
- experiments show 100-400 times faster

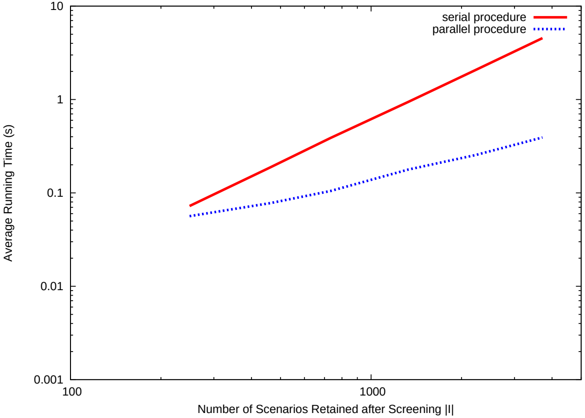
Experimental Setting

- hardware: one 3.0G Hz Core 2 Duo CPU, four GeForce 9800 GPUs, 6 GB memory.
- software: gcc 4.2.4 on Ubuntu 64 bits version.
- testing portfolios: shorting one put option and a portfolio with 8 options.
- number of replications: 100 and more.

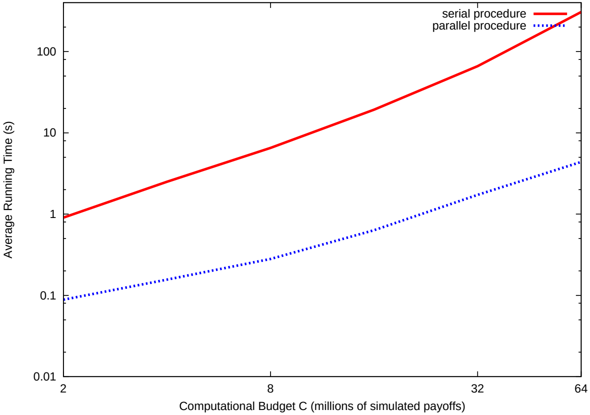
Speedup of Sampling



Speedup of Computing the CI



Speedup in Running Time



Conclusion

Conclusion

- For complex simulation procedure, like nested simulation with screening, GPU computing can still help to improve the overall performance.

Conclusion

- For complex simulation procedure, like nested simulation with screening, GPU computing can still help to improve the overall performance.
- In a parallel computing environment, simulation procedure, like ranking and selection (screening here) need to be trimmed or even redesigned to achieve really high performance.

Conclusion

- For complex simulation procedure, like nested simulation with screening, GPU computing can still help to improve the overall performance.
- In a parallel computing environment, simulation procedure, like ranking and selection (screening here) need to be trimmed or even redesigned to achieve really high performance.
- Parallelism on GPU is very similar as that on CPUs and distributed computing, data communication and synchronization are the bottlenecks of scalability.

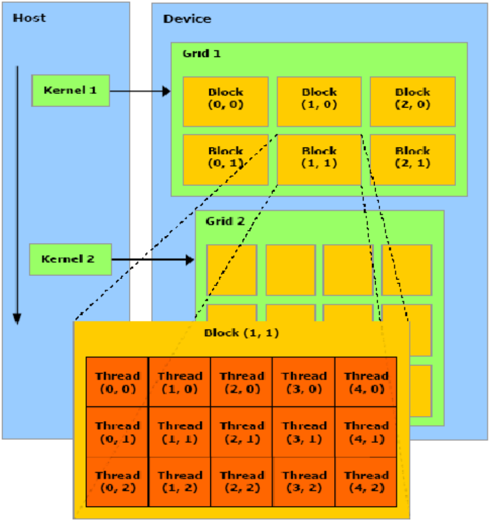
Conclusion

- For complex simulation procedure, like nested simulation with screening, GPU computing can still help to improve the overall performance.
- In a parallel computing environment, simulation procedure, like ranking and selection (screening here) need to be trimmed or even redesigned to achieve really high performance.
- Parallelism on GPU is very similar as that on CPUs and distributed computing, data communication and synchronization are the bottlenecks of scalability.
- Research opportunities exist in providing standard scientific computing package like BLAST on GPU computing.

Multi-GPU and OpenMP

- OpenMP
  - shared memory(fast cached)
  - high performance yet limited scalability
  - branching adaptability
- CUDA on GPU
  - many cores
  - shared memory (cached and un-cached)
  - fast on-board memory (register, local, share, global, const ,texture)
  - hardware thread management (zero overhead)
- 4 GPUs allowed on one computer. Affordable personal supercomputer.

CUDA Program Model



The host issues a succession of kernel invocations to the device. Each kernel is executed as a batch of threads organized as a grid of thread blocks

16 successive threads is called half-wrap, processed simultaneously on one SP.

- no diversity
- access successive memory in one command

Memory Model