Reconfigurable Acceleration for Monte Carlo based Financial Simulation

G.L. Zhang, P.H.W. Leong, C.H. Ho, K.H. Tsoi,
C.C.C. Cheung*, D. Lee**,
*Ray C.C. Cheung****** and W. Luk***

The Chinese University of Hong Kong
Cluster Technology Ltd., Hong Kong*
Electrical Engineering Department, UCLA, USA**
Department of Computing, Imperial College, UK***

IEEE FPT December 2005

Talk outline

- achievements
- motivation
- financial introduction
- Monte Carlo (MC) & the BGM model
- new generic MC architecture
- BGM core architecture
- performance evaluation
- future work
- summary

Achievements

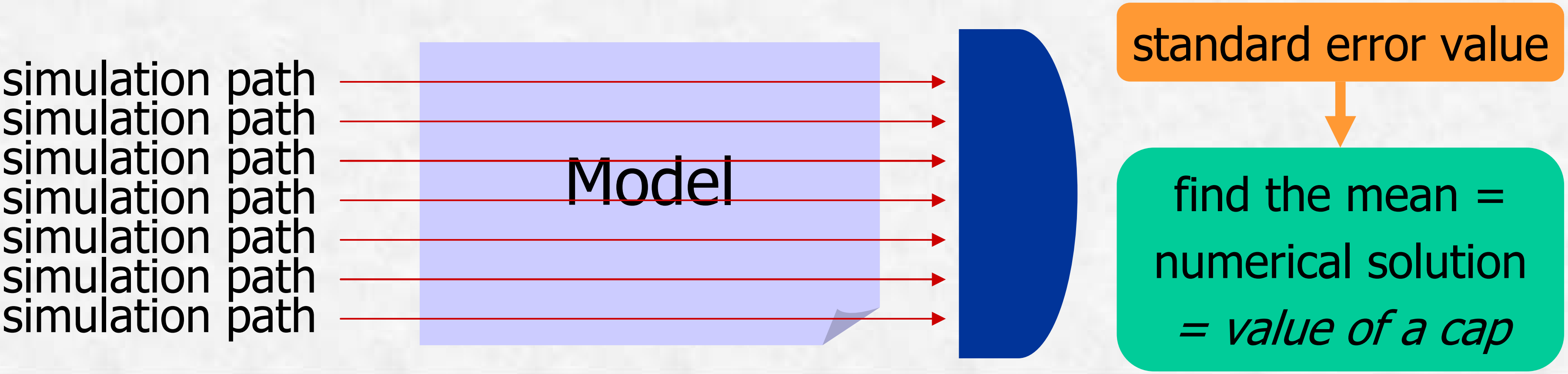
1. hardware accelerator for Monte Carlo (MC) simulation: on-chip processor + reconfigurable logic
2. generalized number system: simulate and optimize designs
3. apply this MC simulation to support Brace, Gatarek and Musiela (BGM) interest rate model
4. efficient Gaussian random number generators, fast division techniques
5. XC2VP30FPGA at 50MHz MC hardware design: 25x Pentium4 1500MHz

Motivation

- a cap gives the holder the right to stick with a specified rate; how much is it? (bank → holder)
- use Monte Carlo simulation to find this cap value
  - requires a large number of randomized runs
  - computation speed is the major barrier
- previous work about MC accelerations focus on
  - biochemical simulation
  - heat transfer, physics simulations
- this work describes MC acceleration and its application in financial modeling

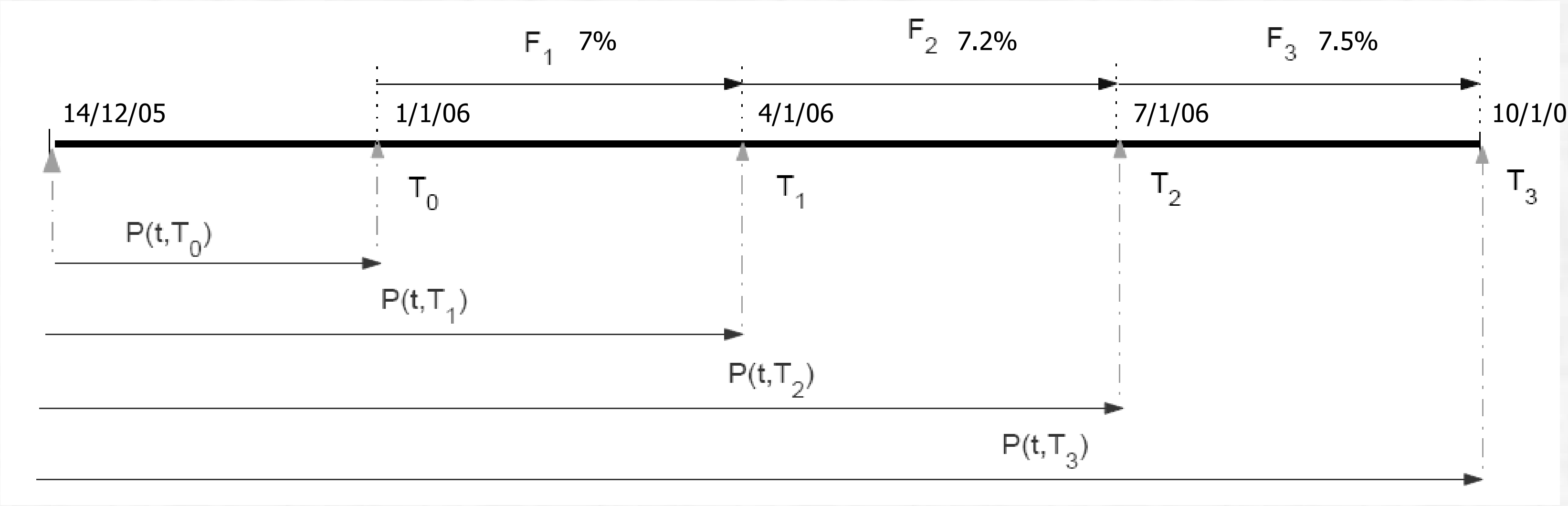
Monte Carlo introduction

- randomly generate values for uncertain variables over and over to simulate a model
- useful for obtaining numerical solutions to analytically difficult problems
- MC can discretize a probability space, using finite samples and find the average solution
- standard error reflects the accuracy of the mean value
  - batch size will vary this standard error value
  - random paths are divided into batches for the standard error
  - e.g. cap value = $10, standard error = 10 cents



Financial engineering introduction

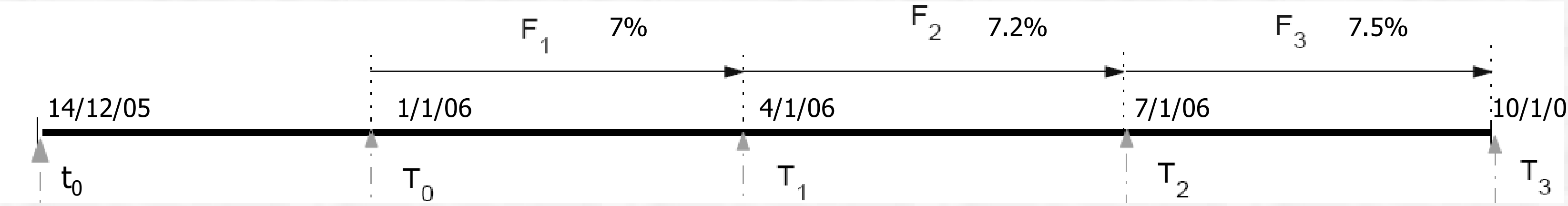
- Fn : forward rate (we can lock in at a later date)
- call option: holder has the right to buy asset at a price by a certain date
- BGM: a popular model to price interest rate derivates (e.g. caps, swap options, bond options)



$\tau = 1/4$

Cap valuation: simplified example

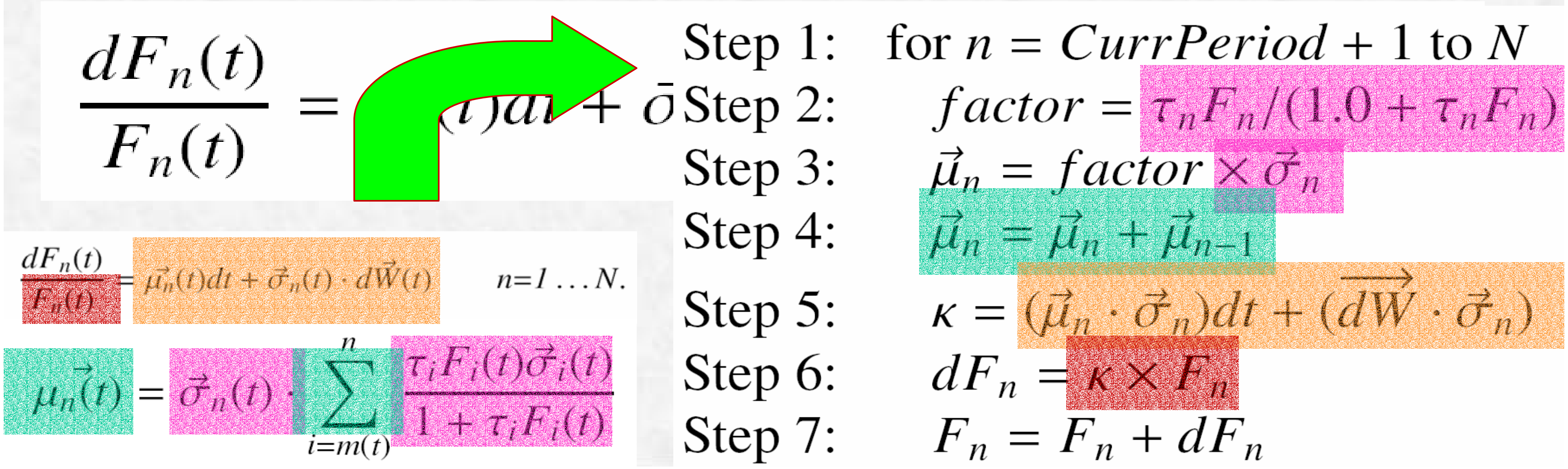
- suppose principal = $1000
- cap rates: 5% in $[T_0,T_1]$, 6% in $[T_1,T_2]$, 7% in $[T_2,T_3]$
- note that we do not know the value of forward rate
  - suppose they are 7%, 7.2%, 7.5%
  - payoff = 1k * ((7%-5%)*0.25year*$P(t_0,T_0)$+(7.2%-6%) *0.25year*$P(t_0,T_1)$+(7.5% -7%)*0.25year*$P(t_0,T_2)$)
  - $P(t_1,t_2)$ is the discounting factor from t_1 to t_2
  - if cap value = this payoff value (*accurate?*)
  - this is the reason why we need MC & the BGM model



τ = 1/4

The BGM model

- based on a stochastic differential equation (SDE)
- price of a cap is computed by using MC simulation
- BGM paths are generated according to the following equation
- involve vector products, divisions
- Noise term: $d\vec{W}(t)$ environment calibration: $\vec{\sigma}_n(t)$



$$\frac{dF_n(t)}{F_n(t)} = (t)dt + \bar{\sigma}$$

$$\frac{dF_n(t)}{F_n(t)} = \vec{\mu_n}(t)dt + \vec{\sigma}_n(t) \cdot d\vec{W}(t) \qquad n=1\ldots N.$$

$$\vec{\mu_n}(t) = \vec{\sigma}_n(t) \cdot \sum_{i=m(t)}^{n} \frac{\tau_i F_i(t) \vec{\sigma}_i(t)}{1 + \tau_i F_i(t)}$$

```
Step 1:  for $n = CurrPeriod + 1$ to $N$
Step 2:      $factor = \tau_n F_n / (1.0 + \tau_n F_n)$
Step 3:      $\vec{\mu}_n = factor \times \vec{\sigma}_n$
Step 4:      $\vec{\mu}_n = \vec{\mu}_n + \vec{\mu}_{n-1}$
Step 5:      $\kappa = (\vec{\mu}_n \cdot \vec{\sigma}_n)dt + (\overrightarrow{dW} \cdot \vec{\sigma}_n)$
Step 6:      $dF_n = \kappa \times F_n$
Step 7:      $F_n = F_n + dF_n$
```

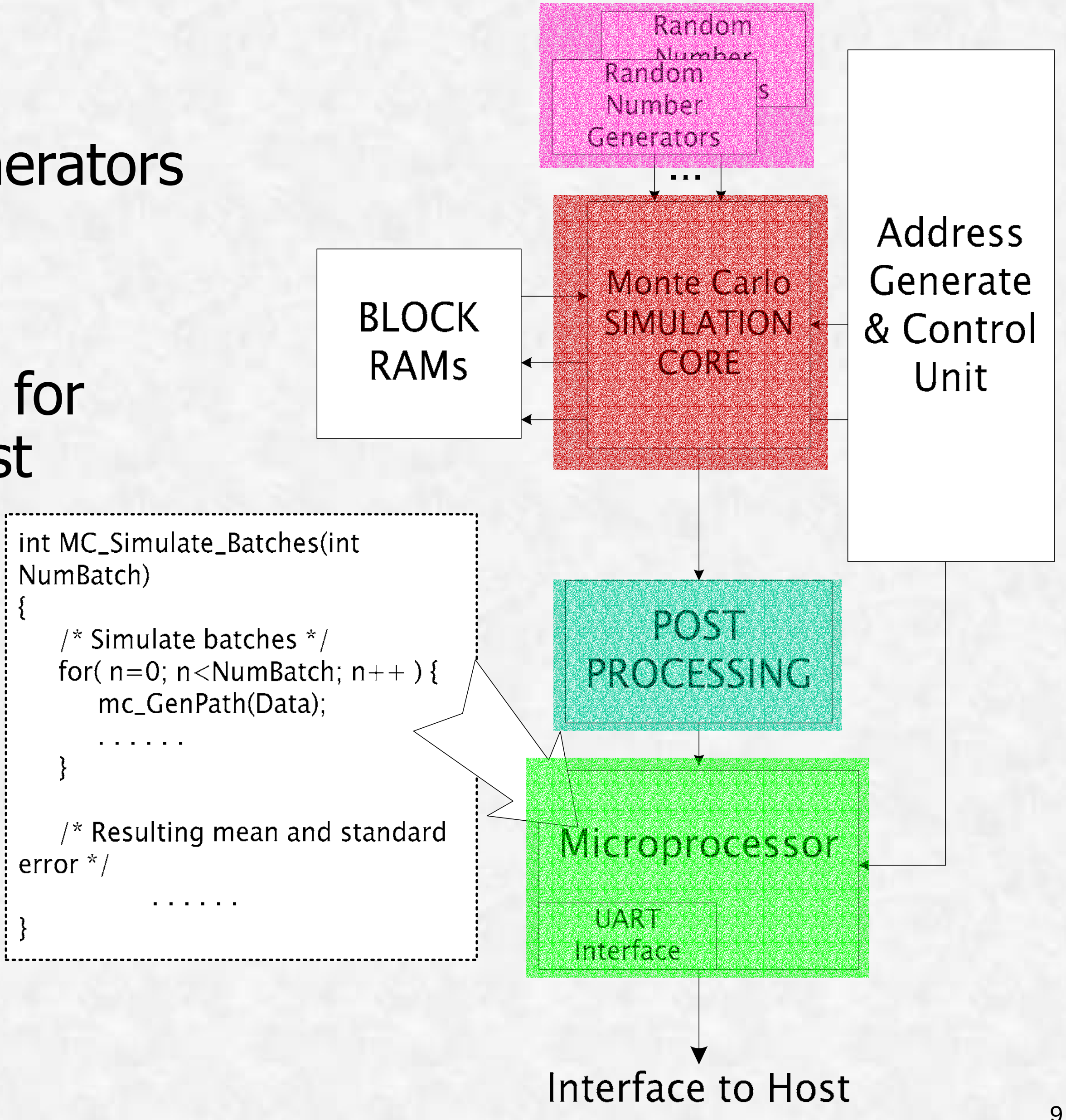
Our MC architecture

four components:

- random number generators
- MC simulation core
- post processing
- microprocessor core for control logic and host interface

e.g. calculate payoff values (caplet, swaplet)

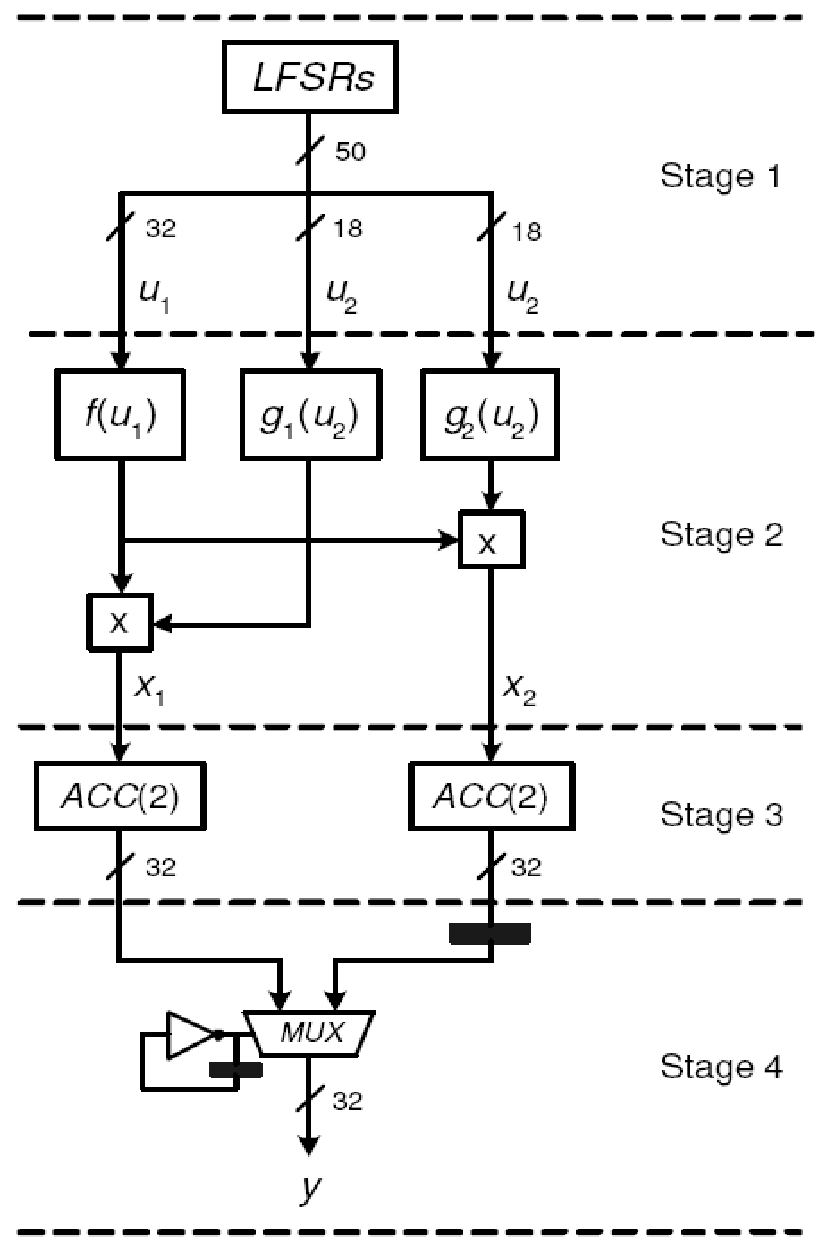
e.g. final cap pricing, standard error value



```
int MC_Simulate_Batches(int
NumBatch)
{
    /* Simulate batches */
    for( n=0; n<NumBatch; n++ ) {
        mc_GenPath(Data);
        . . . . . .
    }

    /* Resulting mean and standard
error */
                . . . . . .
}
```

Gaussian random number generation

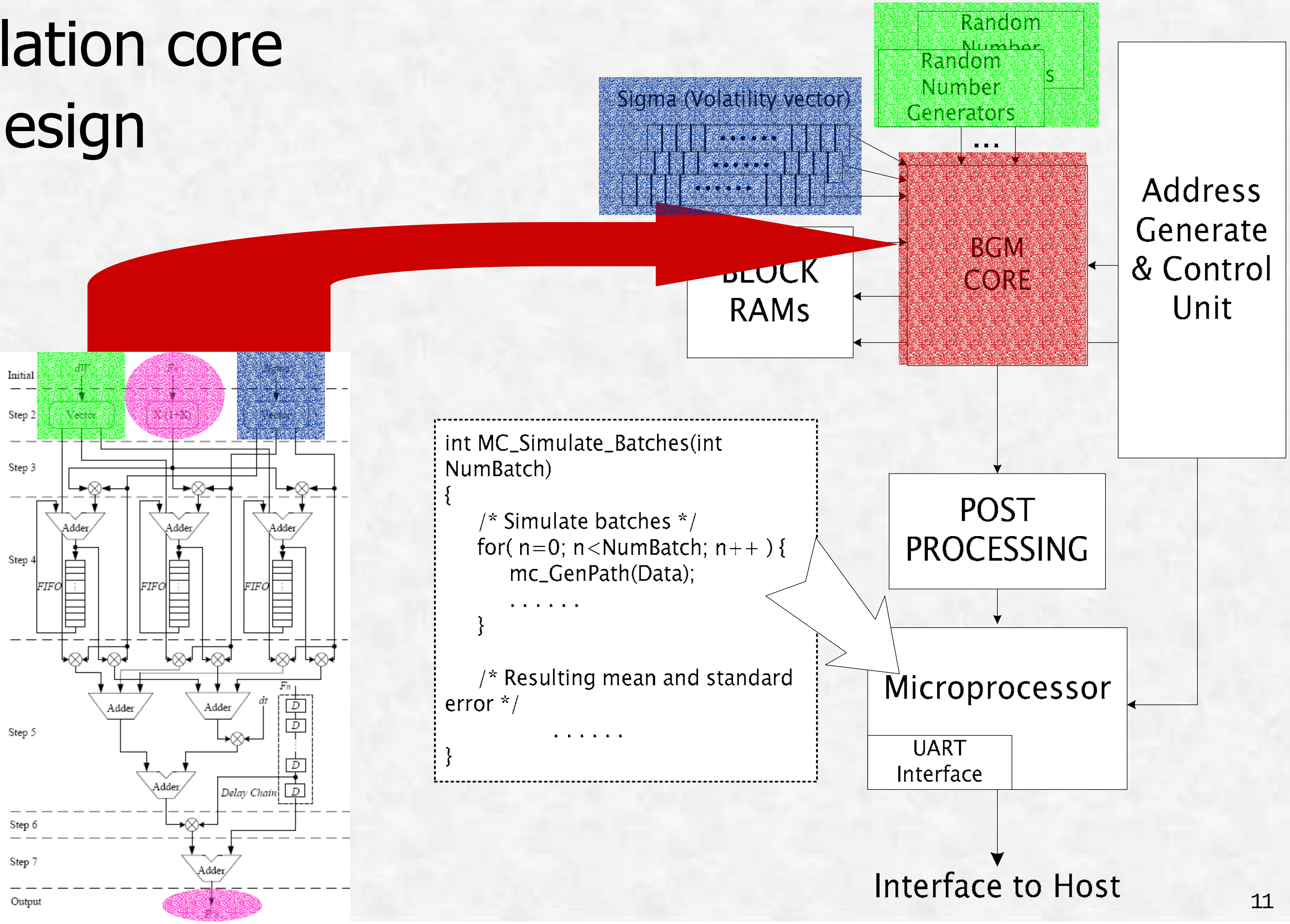
- consists of 4 stages
- use LFSR to generate uniform random numbers
- function evaluations of f, g_1 and g_2 and multiplications
  - piecewise approximation
- accumulation step to overcome the quantization and approximation errors
- noise generation at one sample per clock cycle



MC architecture – using BGM

MC simulation core

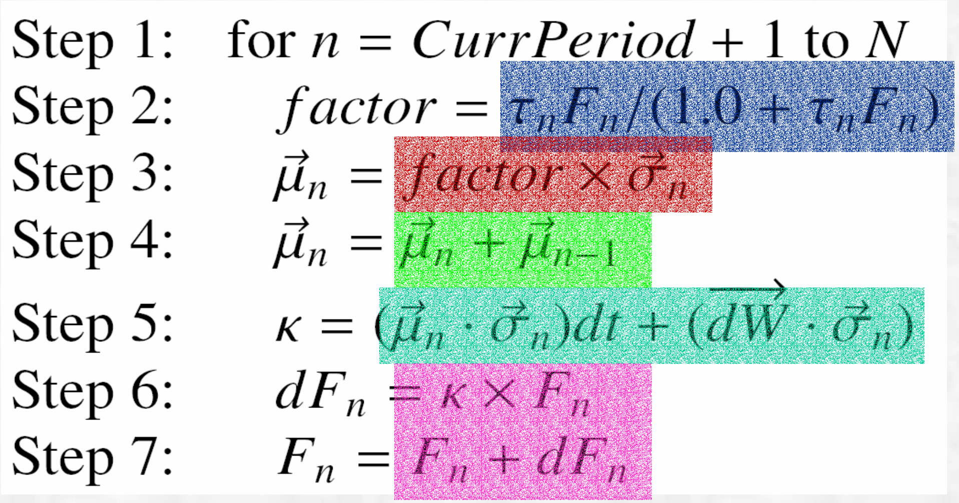
- BGM design



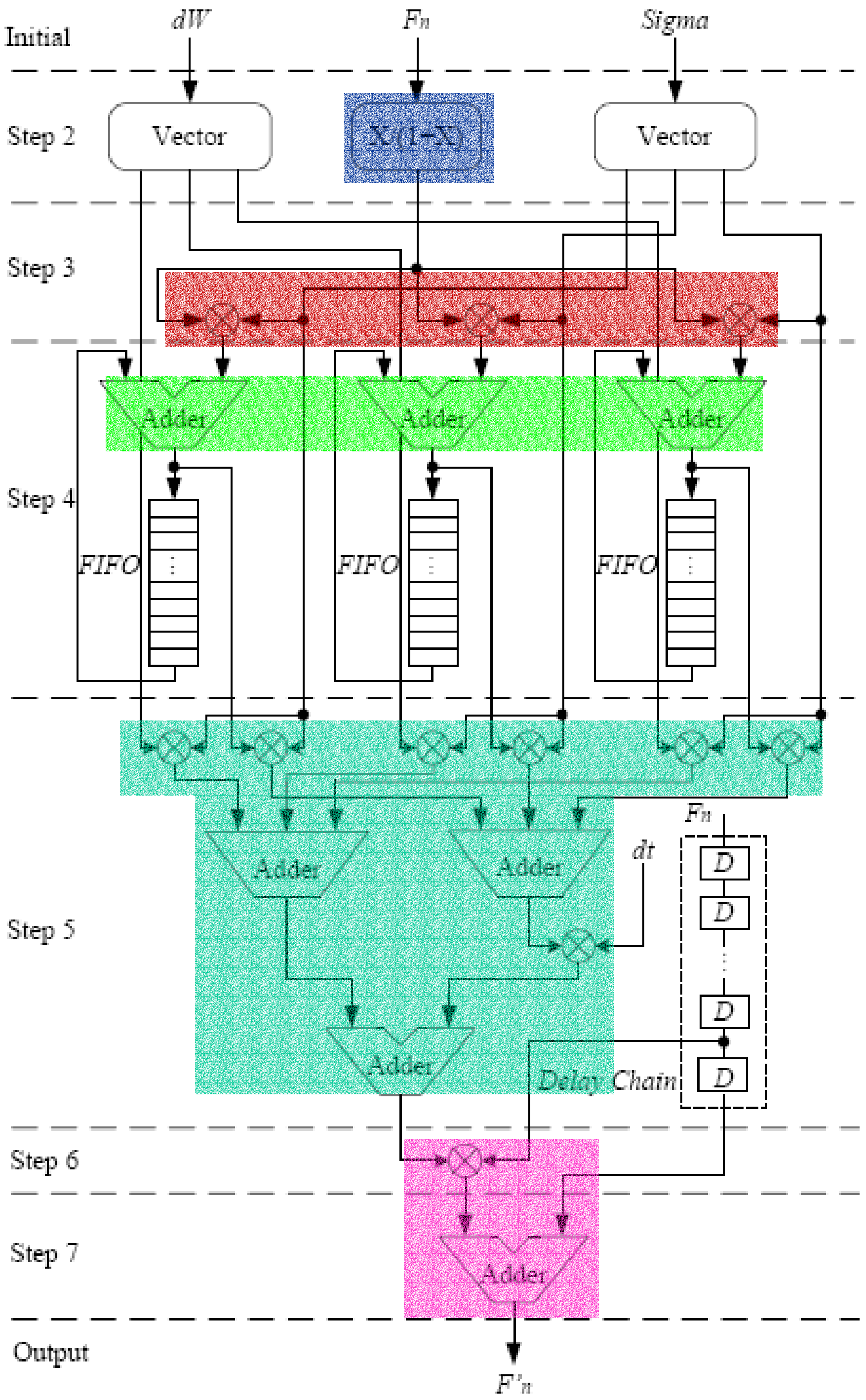
```
int MC_Simulate_Batches(int
NumBatch)
{
    /* Simulate batches */
    for( n=0; n<NumBatch; n++ ){
        mc_GenPath(Data);
        ......
    }

    /* Resulting mean and standard
error */
            ......
}
```

BGM core architecture

- mapping the pseudo code into hardware
- fully pipelined design
- generate F'_n



```
Step 1: for $n = CurrPeriod + 1$ to $N$
Step 2:     $factor = \tau_n F_n/(1.0 + \tau_n F_n)$
Step 3:     $\vec{\mu}_n = factor \times \vec{\sigma}_n$
Step 4:     $\vec{\mu}_n = \vec{\mu}_n + \vec{\mu}_{n-1}$
Step 5:     $\kappa = (\vec{\mu}_n \cdot \vec{\sigma}_n)dt + (\overrightarrow{dW} \cdot \vec{\sigma}_n)$
Step 6:     $dF_n = \kappa \times F_n$
Step 7:     $F_n = F_n + dF_n$
```

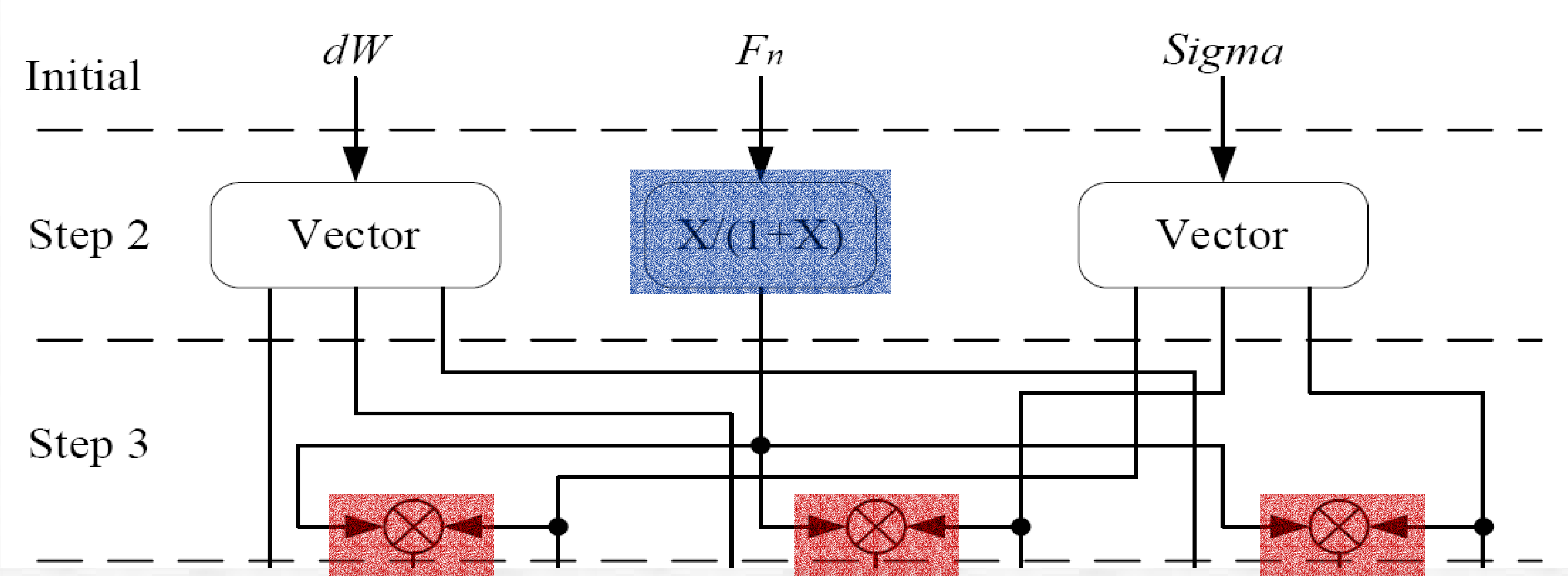


BGM core – Step 1 to 3

- initialize the Brownian motion parameter (*dW*), volatility vector (*sigma*), forward rate (F_n)
- “vector” blocks convert *dW* and *sigma* to scalars

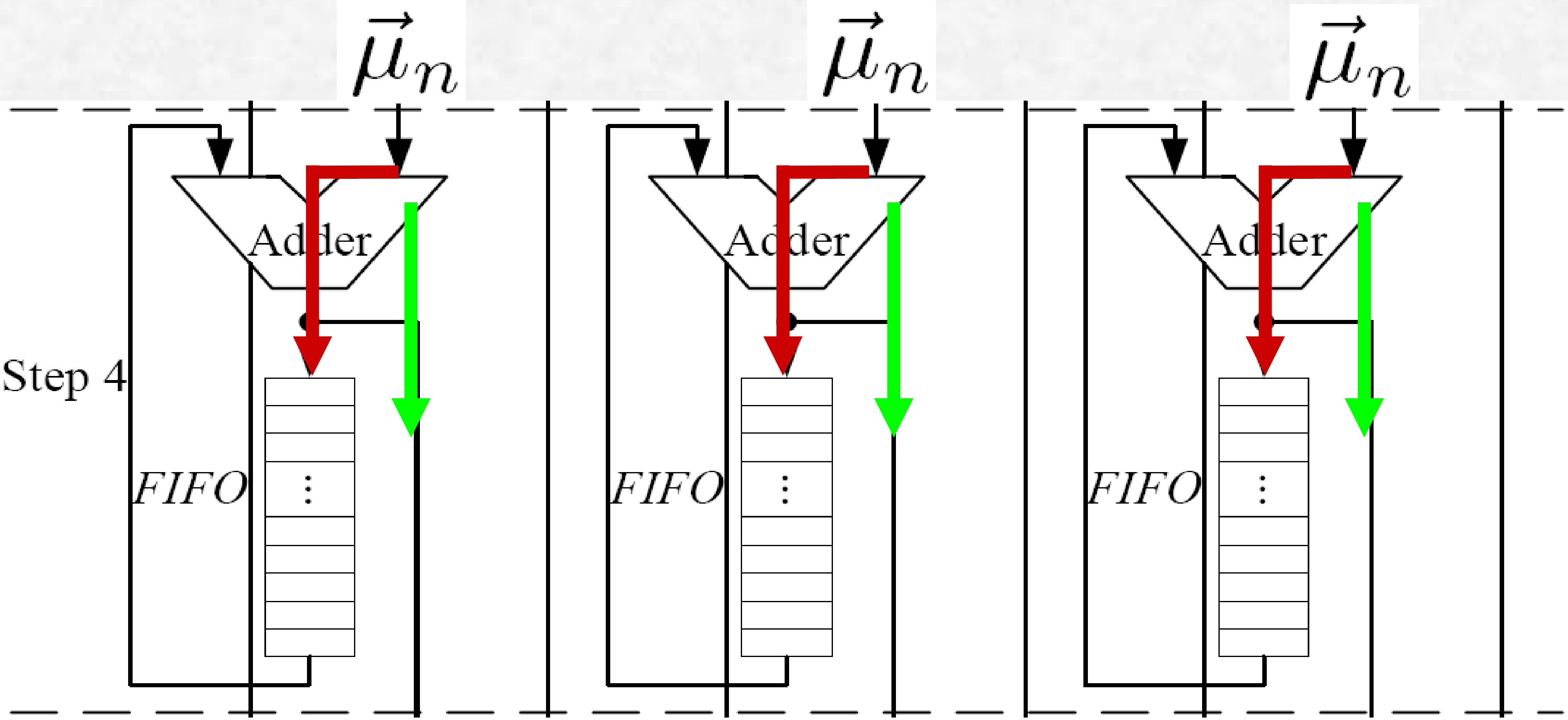
Step 2: $factor = \tau_n F_n/(1.0 + \tau_n F_n)$

Step 3: $\vec{\mu}_n = factor \times \vec{\sigma}_n$



BGM core: Step 4: $\vec{\mu}_n = \vec{\mu}_n + \vec{\mu}_{n-1}$

- a vector addition
- depth of FIFO is decided by the number of BGM paths

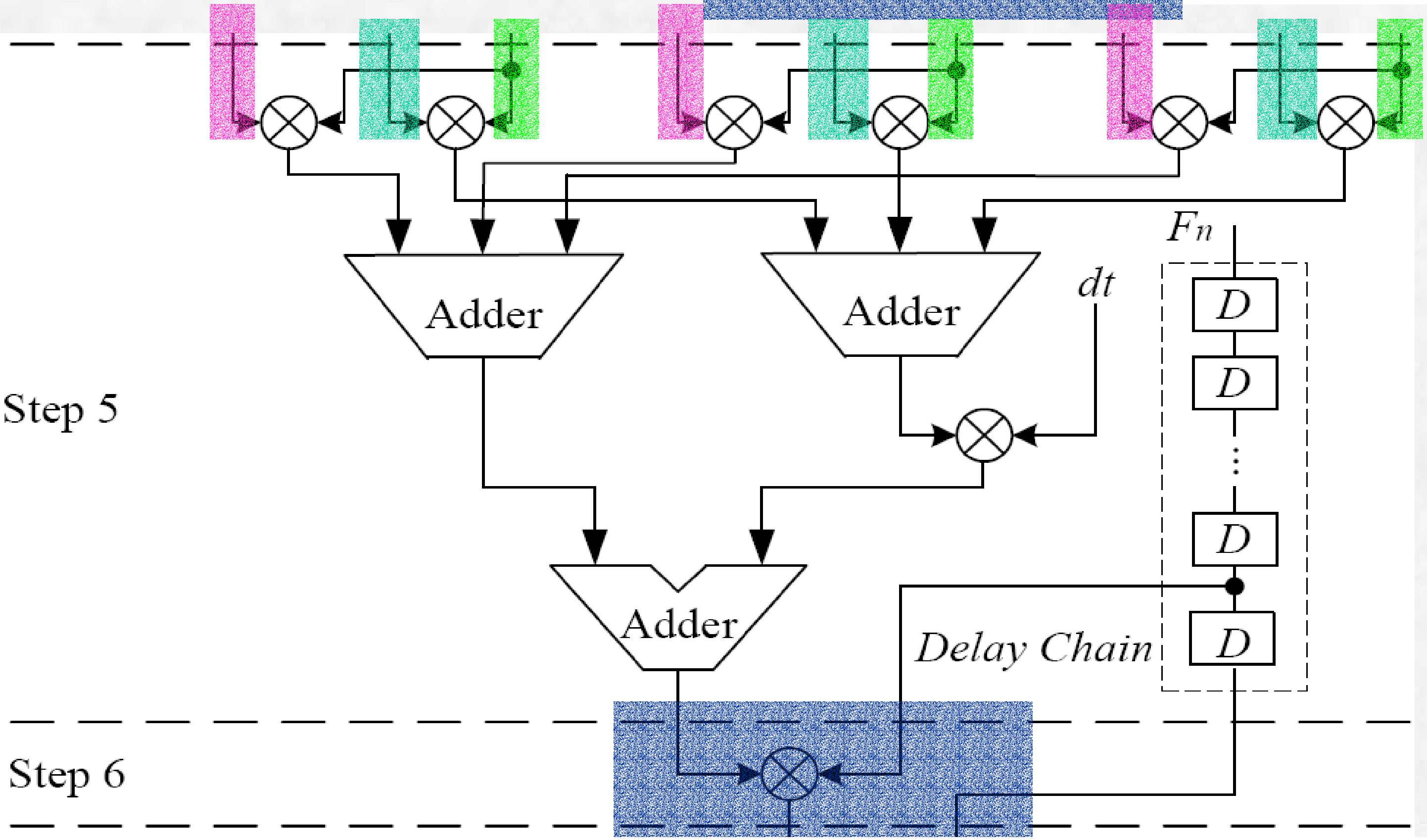


BGM core – Step 5 & 6

- two vector multiplications (dot product)
- $x_1y_1 + x_2y_2 + x_3y_3$

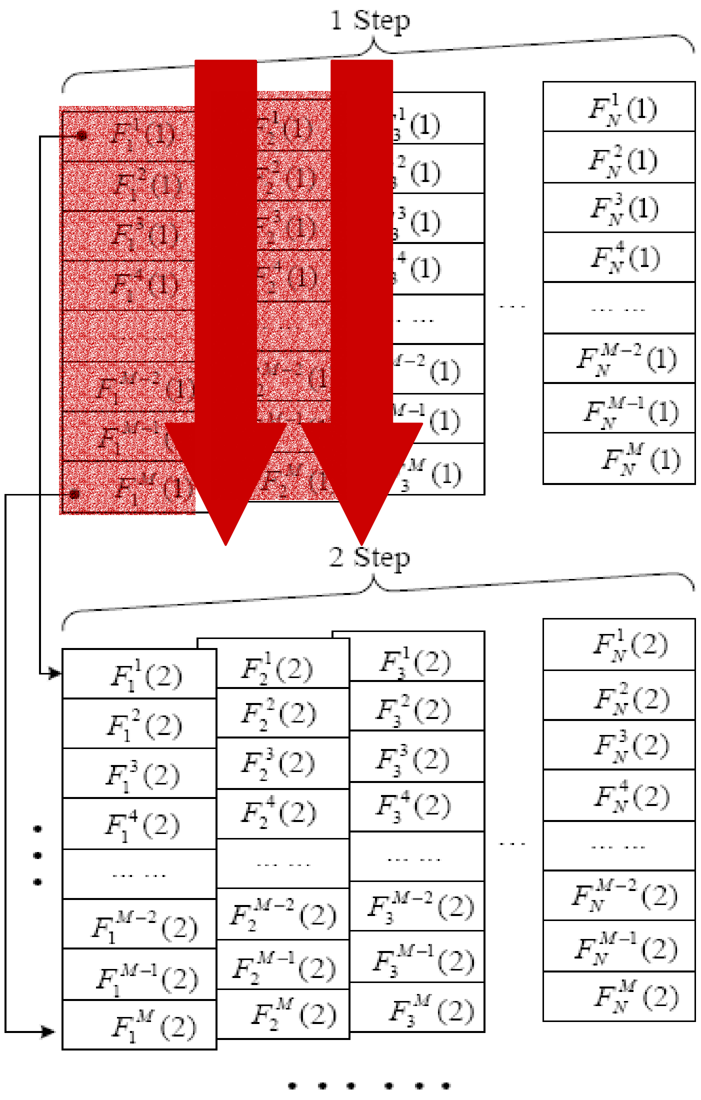
$$\kappa = (\vec{\mu}_n \cdot \vec{\sigma}_n)dt + (\overrightarrow{dW} \cdot \vec{\sigma}_n)$$

$$dF_n = \kappa \times F_n$$



2-D data flow of the BGM simulation

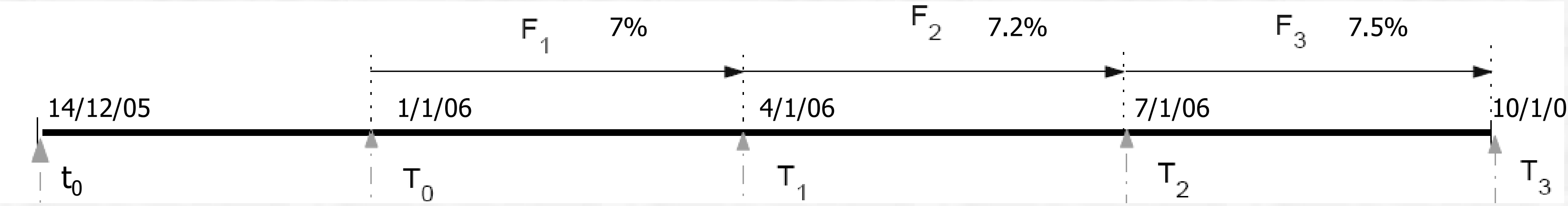
- MC simulation generates a set of independent random forward rate paths
- different pipelined stage computes a different path
- resolve the data dependencies between different paths


1 Step
$F_1^1(1)$ $F_1^2(1)$ $F_1^3(1)$ $F_1^4(1)$ $F_1^{M-2}(1)$ $F_1^{M-1}(1)$ $F_1^M(1)$
$F_N^1(1)$ $F_N^2(1)$ $F_N^3(1)$ $F_N^4(1)$ … … $F_N^{M-2}(1)$ $F_N^{M-1}(1)$ $F_N^M(1)$
2 Step
$F_1^1(2)$ $F_1^2(2)$ $F_1^3(2)$ $F_1^4(2)$ … … $F_1^{M-2}(2)$ $F_1^{M-1}(2)$ $F_1^M(2)$
$F_2^1(2)$ $F_2^2(2)$ $F_2^3(2)$ $F_2^4(2)$ … … $F_2^{M-2}(2)$ $F_2^{M-1}(2)$ $F_2^M(2)$
$F_3^1(2)$ $F_3^2(2)$ $F_3^3(2)$ $F_3^4(2)$ … … $F_3^{M-2}(2)$ $F_3^{M-1}(2)$ $F_3^M(2)$
$F_N^1(2)$ $F_N^2(2)$ $F_N^3(2)$ $F_N^4(2)$ … … $F_N^{M-2}(2)$ $F_N^{M-1}(2)$ $F_N^M(2)$

M: the number of paths
N: the number of standard forward rates

Revisit: Cap pricing

- suppose principal = $1000
- cap rates: 5% in $[T_0,T_1]$, 6% in $[T_1,T_2]$, 7% in $[T_2,T_3]$
- note that we do not know the value of forward rate
  - suppose they are 7%, 7.2%, 7.5%
  - payoff = 1k * ((7%-5%)*0.25year*$P(t_0,T_0)$+(7.2%-6%) *0.25year*$P(t_0,T_1)$+(7.5% -7%)*0.25year*$P(t_0,T_2)$)
  - $P(t_1,t_2)$ is the discounting factor from t_1 to t_2
  - if cap value = this payoff value (*accurate?*)
  - this is the reason why we need MC & the BGM model

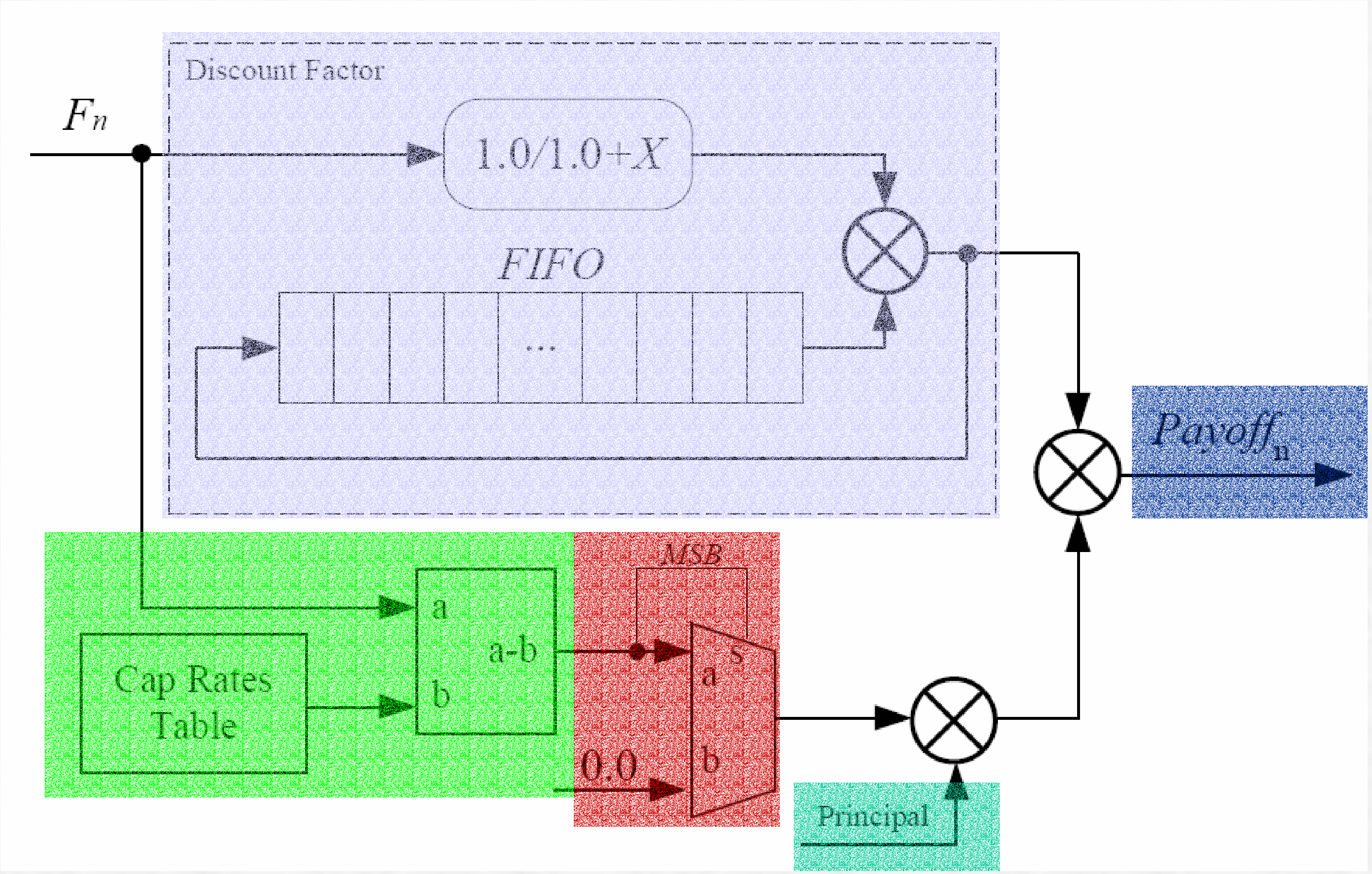


τ = 1/4

Post-processing – Cap pricing

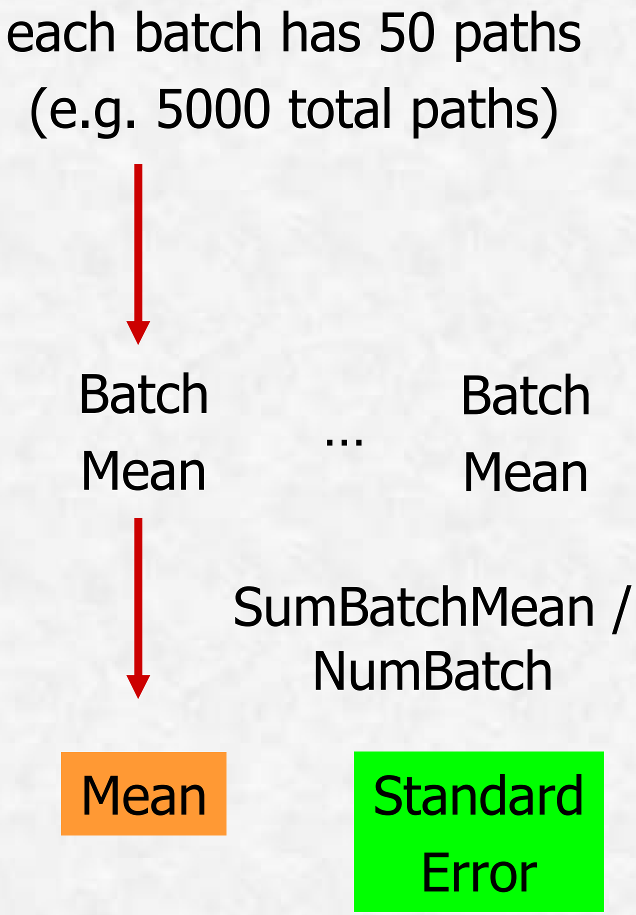
- implement this equation for cap pricing

$$payoff_n = principal \times \tau_n \times max(F_n(t_n) - \text{cap rate}, 0.0)$$



Program running on the PowerPC

- calculate the means and standard errors of the randomised trial runs



each batch has 50 paths

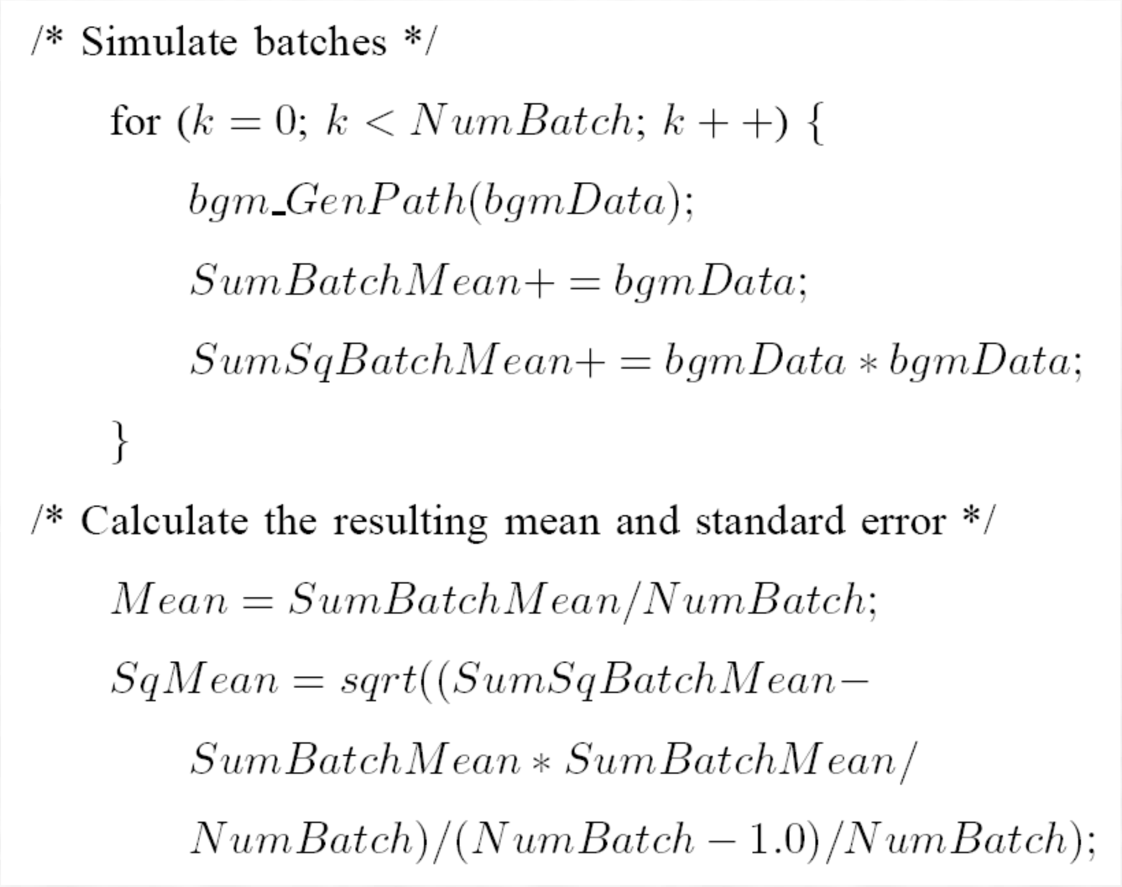
(e.g. 5000 total paths)

Batch Mean … Batch Mean

SumBatchMean / NumBatch

Mean

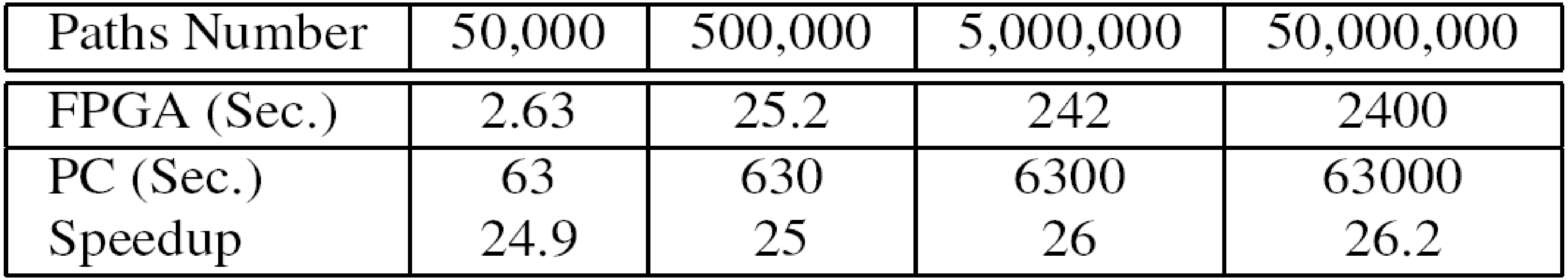
Standard Error



```
/* Simulate batches */
    for ($k = 0$; $k < NumBatch$; $k++$) {
        $bgm\_GenPath(bgmData)$;
        $SumBatchMean+ = bgmData$;
        $SumSqBatchMean+ = bgmData * bgmData$;
    }
/* Calculate the resulting mean and standard error */
    $Mean = SumBatchMean/NumBatch$;
    $SqMean = sqrt((SumSqBatchMean-$
        $SumBatchMean * SumBatchMean/$
        $NumBatch)/(NumBatch - 1.0)/NumBatch)$;
```

Performance evaluation

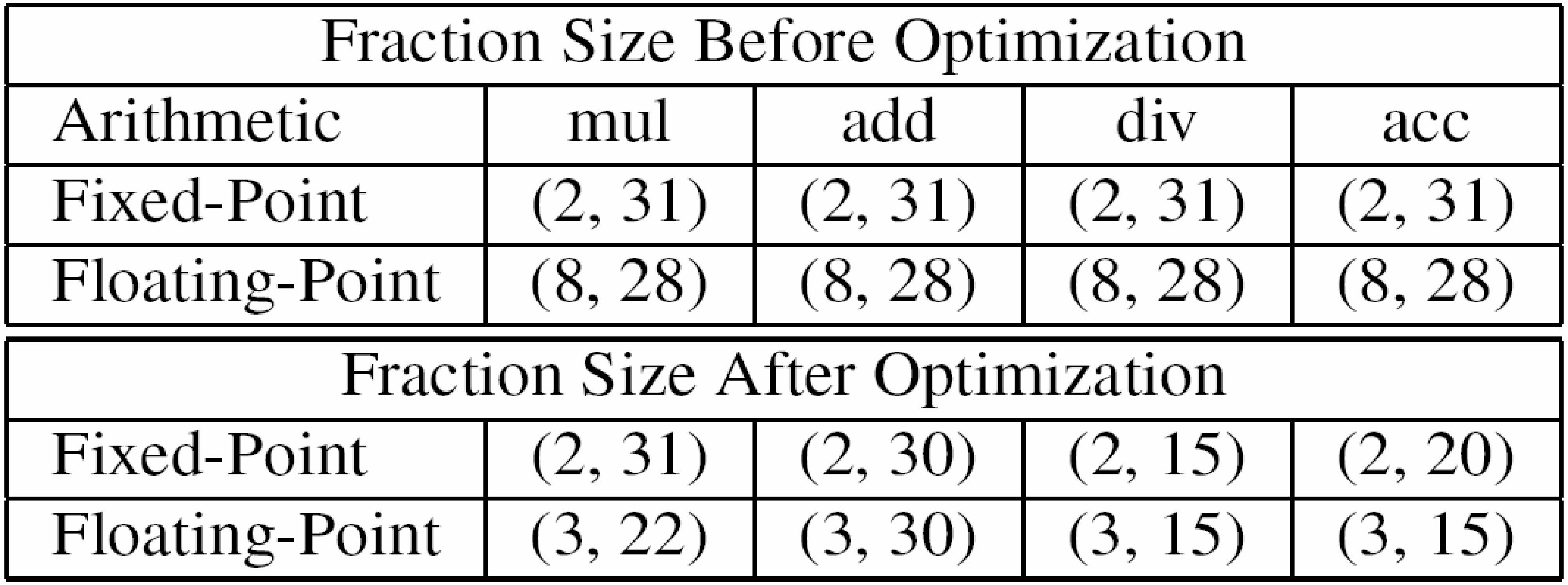
- use Xilinx ML310 system, XC2VP30 device, with two embedded PowerPC cores
- compare with a P4 1.5GHz machine
  - GCC 3.3 compilation with -O3 optimization
- experimental results show a scalable speedup to the software implementation
- NumPath-per-Batch = 50



Paths Number	50,000	500,000	5,000,000	50,000,000
FPGA (Sec.)	2.63	25.2	242	2400
PC (Sec.)	63	630	6300	63000
Speedup	24.9	25	26	26.2

Wordlength optimisation

- given: financial applications require at least 4 decimal place accuracy
- use “computer arithmetic synthesis tool”(CAST) for software simulation
  - support fixed-point, floating-point simulations to generate generic VHDL descriptions



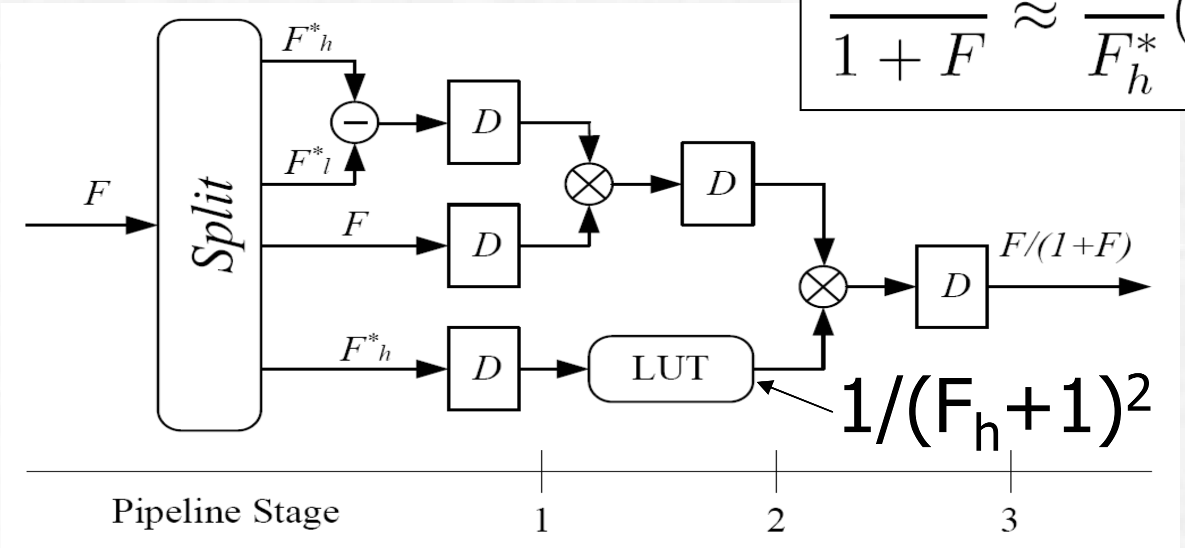
Fraction Size Before Optimization				
Arithmetic	mul	add	div	acc
Fixed-Point	(2, 31)	(2, 31)	(2, 31)	(2, 31)
Floating-Point	(8, 28)	(8, 28)	(8, 28)	(8, 28)

Fraction Size After Optimization				
Fixed-Point	(2, 31)	(2, 30)	(2, 15)	(2, 20)
Floating-Point	(3, 22)	(3, 30)	(3, 15)	(3, 15)

Fast division

- evaluate “F / (F+1)” where F is between 0 and 1
- use Hung’s method* with a *small lookup table* with 3 pipeline stages, two multiplications and one subtraction

$$\frac{F}{1+F} \approx \frac{F}{F_h^*}(1 - \frac{F_l}{F_h^*}) = \frac{F(F_h^* - F_l^*)}{F_h^{*2}}$$

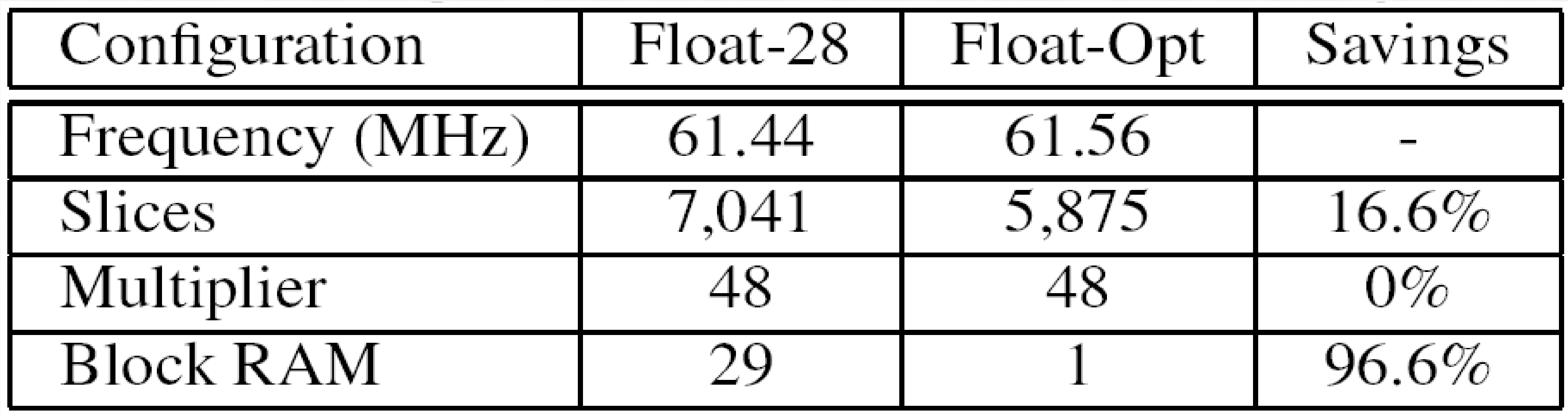


$1/(F_h+1)^2$

* P. Hung, H. Fahmy, O. Mencer, M.J. Flynn, “Fast division algorithm with a small lookup table,” *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems and computers*, Vol. 2, pp. 1465.1468, May 1999.

Configuration optimization

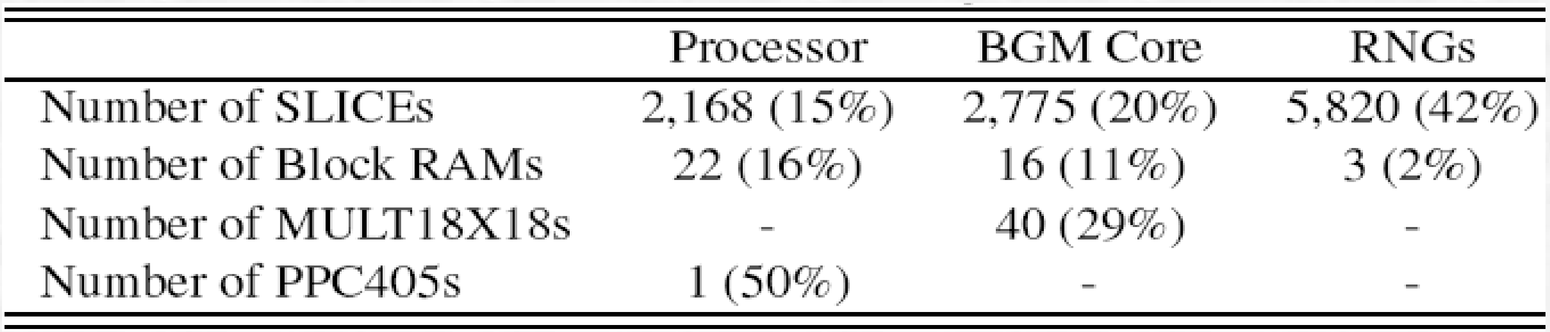
- VHDL implementation, synthesize with Xilinx XST, and implement using Xilinx EDK
- slices reduction due to the smaller exponent and fraction bit-widths
- significant BRAM reduction due to the bit-width reduction for the divider (by 2^8)



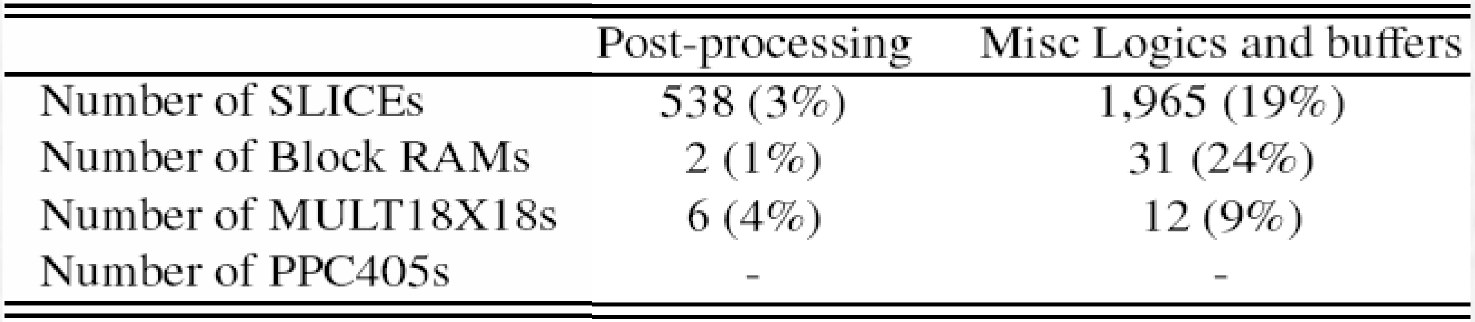
Configuration	Float-28	Float-Opt	Savings
Frequency (MHz)	61.44	61.56	-
Slices	7,041	5,875	16.6%
Multiplier	48	48	0%
Block RAM	29	1	96.6%

Device utilization summary

- device utilization of XC2VP30FF896
  - – number of SLICEs: 13,266 / 13,696 (96%)
  - – number of Block RAMs: 74 / 136 (54%)
  - – number of MULT18x18: 58 / 136 (42%)



	Processor	BGM Core	RNGs
Number of SLICEs	2,168 (15%)	2,775 (20%)	5,820 (42%)
Number of Block RAMs	22 (16%)	16 (11%)	3 (2%)
Number of MULT18X18s	-	40 (29%)	-
Number of PPC405s	1 (50%)	-	-



	Post-processing	Misc Logics and buffers
Number of SLICEs	538 (3%)	1,965 (19%)
Number of Block RAMs	2 (1%)	31 (24%)
Number of MULT18X18s	6 (4%)	12 (9%)
Number of PPC405s	-	-

Future work

- apply run-time reconfiguration technique
- extension work to cover other financial models
- integration MC simulation core with embedded system
- two embedded PowerPC in ML310
  - one for MC simulation,
  - one for embedded Linux,
  - communicate with other FPGA boards,
  - virtually unlimited scalability

Summary

1. hardware accelerator for Monte Carlo (MC) simulation: on-chip processor + reconfigurable logic
2. generalized number system: simulate and optimize designs
3. apply this MC simulation to support Brace, Gatarek and Musiela (BGM) interest rate model
4. efficient Gaussian random number generator, fast division techniques
5. XC2VP30FPGA at 50MHz MC hardware design: 25x Pentium4 1500MHz