Monte Carlo Financial Models on U50

Group 7

Member: 呂依凡 110061562

Introduction

In this project, I am going to implement Black-Scholes Model on the payoff price of European vanilla option. In the Black-Scholes model, the price can be considered as geometric Brownian motion. Then we can obtain (1).

$$\mathrm{dS} = \mathrm{rSdt} + \sigma\mathrm{Sdz} \tag{1}$$

In (1), S is stock price, r is the fixed interest rate, σ is the constant volatility and z is a Wiener process. The analytical solution for the stochastic differential equation (1) is

$$S_{t+\Delta t} = S_t e^{(r-\frac{1}{2}\sigma^2)\Delta t+\sigma\epsilon\sqrt{\Delta t}} \tag{2}$$

In (2), $\epsilon \sim N(0,1)$ is the standard normal distribution.

The European vanilla option can be exercised only at expiration date. As a result, only the stock price at the expiration date can affect the payoff price. Then the payoff price can be implemented as (3).

$$P_{call} = max\{S_T - K, 0\} \tag{3}$$

In (3), T is the pre-set time, S_T is the stock price at expiration date and K is the strike price.

Algorithm

Black-Sholes model includes three main steps. Each time interval (0, T) is first divided into M steps, and denoted as t_0, t_1, …, t_M. Secondly, M standard normally distributed independent random numbers ϵ are generated. Finally, S_{t_M} is computed by (2) from S_{t_0}. After going through the three steps, one "path" is completed. In the model, it needs to compute N paths. The algorithm is shown below.

Algorithm 1 Black-Scholes model

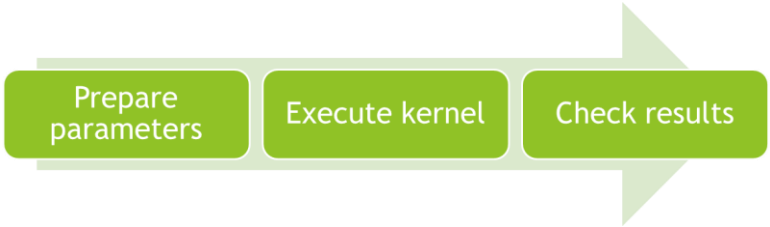
Input: parameters for the stock and option
Output: payoff price
Initialization: Random number generators
for $i = 1$ to N **do**
  for $k = 1$ to M **do**
    $U_1, U_2 \leftarrow MersenneTwist()$
    $\epsilon_1, \epsilon_2 \leftarrow BoxMuller(U_1, U_2)$
    $S_{t_{k+2}}, S_{t_{k+1}} \leftarrow Price(S_{t_k}, \epsilon_1, \epsilon_2)$
    $k+ = 2$
  end for
  $P_{option}[i] \leftarrow Option(S_t[], K)$
  i++
end for
return $P_{Call} = ave(P_{option})$

The random numbers ϵ is generated by Mersenne-Twister algorithm followed by Box-Muller transformation.

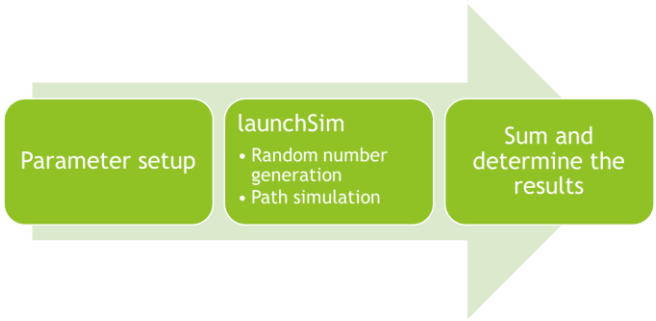
Hardware Implementation

In this project, I re-write the whole host program and adjust some of the pragma in order to make the project workable on the vitis 2021.2 and U50 platfrom.

System Structure



Kernel Execution Structure



1. Parameter setup

   The kernel is first setup the parameters. It uses sd to transfer all the float data into blackScholes model.

```
60          static const int SIMS_PER_GROUP =32;
61          stockData<float> sd(timeT,freeRate,volatility,initPrice,strikePrice);
62
63          float call0, put0;
64          blackScholes<SIMS_PER_GROUP,EuropeanOptionStatus<float>, float> bs0(sd, steps);
```

2. LaunchSim

   The kernel is then use launchSimulation in lauchSim.h to generate random number using box-Muller algorithm and use simulation offered by the class blackScholes to do path simulation. In launchSim, three functions are called, prng, BOX_MULLER, and simulation. I will introduce them in the following.

```
launchSimulation(call0, put0, g_id+0, bs0, sims*steps>>4, sims>>4);
```

```
19  void launchSim(DATA_T &pCall, DATA_T &pPut, RNG<DATA_T> &rng0,RNG<DATA_T> &rng1, Model &m, int numR, int sims){
20  #pragma HLS DATAFLOW
21          hls::stream<unsigned int> s_num0;
22          hls::stream<unsigned int> s_num1;
23  #pragma HLS STREAM variable=s_num0 depth=4 dim=1
24  #pragma HLS STREAM variable=s_num1 depth=4 dim=1
25          hls::stream<DATA_T> sRNG0;
26          hls::stream<DATA_T> sRNG1;
27  #pragma HLS STREAM variable=sRNG0 depth=4 dim=1
28  #pragma HLS STREAM variable=sRNG1 depth=4 dim=1
29
30          prng(rng0, rng1, s_num0, s_num1, numR);
31
32          RNG<DATA_T>::BOX_MULLER(s_num0, s_num1, sRNG0, sRNG1, numR>>1);
33
34          m.simulation(sRNG0,sRNG1, sims, pCall, pPut);
35  }
```

   Prng:

```
void prng(RNG<DATA_T> &rng0,RNG<DATA_T> &rng1, hls::stream<unsigned int> &sRNG0,hls::stream<unsigned int> &sRNG1, int nums
        for(int i = 0 ; i < nums>>2;i++){
#pragma HLS PIPELINE
                unsigned int r0, r1, r2, r3;
                rng0.extract_number(&r0, &r1);
                rng1.extract_number(&r2, &r3);
                sRNG0.write(r0);
                sRNG0.write(r2);
                sRNG1.write(r1);
                sRNG1.write(r3);
        }
}
```

extract_number is used to prepare the seed for box-muller algorithm. The INLINE pragma is used to break the function structure to decrease the use of hardware resources.

```
        void extract_number(uint* num1,uint* num2){
#pragma HLS INLINE
                uint id1=increase(1), idm=increase(RNG_MH), idm1=increase(RNG_MHI);

                uint x = this->seed,x1=this->mt_o[this->index],x2=this->mt_e[id1],
                        xm=this->mt_o[idm],xm1=this->mt_e[idm1];

                x = (x & upper_mask)+(x1 & lower_mask);
                uint xp = x >> 1;
                if ((x & 0x01) != 0)
                        xp ^= RNG_A;
                x = xm ^ xp;

                uint y = x;
                y ^= ((y >> RNG_U)&RNG_D);
                y ^= ((y << RNG_S)&RNG_B);
                y ^= ((y << RNG_T)&RNG_C);
                y ^= (y >> RNG_L);
                *num1 = y;
                mt_e[this->index]=x;

                x1 =( x1 & upper_mask) + (x2 & lower_mask);
                uint xt = x1 >> 1;
                if ((x1 &0x01) != 0)
                        xt ^= RNG_A;
                x1 = xm1 ^ xt;
                uint y1 = x1;
                y1 ^= ((y1 >> RNG_U)&RNG_D);
                y1 ^= ((y1 << RNG_S)&RNG_B);
                y1 ^= ((y1 << RNG_T)&RNG_C);
                y1 ^= (y1 >> RNG_L);
                *num2 = y1;
                mt_o[this->index]=x1;

                this->index=id1;
                this->seed=x2;
        }
```

BOX_MULLER:

In this function, INLINE is still used to break function structure.

```
        void BOX_MULLER(DATA_T*data1, DATA_T*data2,DATA_T ave, DATA_T deviation){
#pragma HLS INLINE
                static const DATA_T _2PI= 2*3.14159265358979323846f;
                //      static const DATA_T MINI_RNG = 2.328306e-10;

                uint num1,num2;
                DATA_T tp,tmp1,tmp2;
                extract_number(&num1,&num2);
                ap_ufixed<32,0> f_tmp1, f_tmp2;
                f_tmp1(31, 0)=num1;
                f_tmp2(31, 0)=num2;
                tmp1 = f_tmp1.to_float();
                tmp2 = f_tmp2.to_float();
#ifdef __DOUBLE_PRECISION__
                tp=sqrt(fmax(-2*log(tmp1),0)*deviation);
                *data1=cos(_2PI*tmp2)*tp+ave;
                *data2=sin(_2PI*tmp2)*tp+ave;
#else
                tp=sqrtf(fmaxf(-2.0f*logf(tmp1),0.0f)*deviation);
                *data1=cosf(_2PI*tmp2)*tp+ave;
                *data2=sinf(_2PI*tmp2)*tp+ave;
#endif
        }
```

Simulation:

In simulation, path_sum and sum are called to complete the algorithm we have explained in algorithm section.

```
        void simulation(hls::stream<DATA_T>& s_RNG0,hls::stream<DATA_T>& s_RNG1, int sims,  DATA_T &pCall, DATA_T &pPut)
        {
                DATA_T call = 0, put = 0;
                hls::stream<DATA_T> prices;
#pragma HLS STREAM variable=prices depth=NUM_SIMS dim=1
#pragma HLS DATAFLOW
                path_sim(s_RNG0, s_RNG1, prices, sims);
                sum(prices, call, put, sims);
                pCall= call;
                pPut = put;
        }
```

```
56        void path_sim(hls::stream<DATA_T>& s_RNG0,hls::stream<DATA_T>& s_RNG1, hls::stream<DATA_T>& prices, int sims){
57                OptionStatus stockPrice[NUM_SIMS];
58  #pragma HLS ARRAY_PARTITION variable=stockPrice cyclic factor=2 dim=1
59                for(int j=0;j<NUM_SIMS;j++)
60  #pragma HLS PIPELINE
61                        stockPrice[j].init(data.price);
62
63                for(int k=0;k<sims/NUM_SIMS;k++) {
64                        for(int s=0; s <NUM_STEPS;s++){
65                                for(int j=0;j<NUM_SIMS/2;j++) {
66  #pragma HLS PIPELINE
67                                        DATA_T r0 = s_RNG0.read();
68                                        DATA_T r1 = s_RNG1.read();
69                                        update(stockPrice[j*2], r0);
70                                        update(stockPrice[j*2+1], r1);
71                                }
72                        }
73                        for(int j=0;j<NUM_SIMS;j++){
74  #pragma HLS DEPENDENCE variable=stockPrice array inter RAW false
75  #pragma HLS PIPELINE
76                                DATA_T price = stockPrice[j].valid? stockPrice[j].price():data.strikePrice;
77                                prices.write(price);
78                                stockPrice[j].init(data.price);
79                        }
80                }
81        }
```

```
82        void update(OptionStatus &option, DATA_T r){
83  #pragma HLS INLINE
84                const DATA_T Dt = data.timeT / (DATA_T)NUM_STEPS,
85                                        Rdt = 1+data.freeRate*Dt,
86                                        SqrtV = data.volatility * sqrtf(Dt);
87                option.stockPrice *= Rdt +r *SqrtV;
88                option.update();
89        }
90        void sum(hls::stream<DATA_T> &prices, DATA_T &call, DATA_T&put, int sims){
91                for(int j=0;j<sims;j++) {
92  #pragma HLS PIPELINE
93                        DATA_T price = prices.read();
94                        call+=executeCall(price);
95                        put+=executePut(price);
96                }
97        }
```

In sum function, the executeCall and executePut are called to determine which price result should be kept as shown in formula (3).

```
98          DATA_T executeCall(DATA_T& price){
99  #pragma HLS INLINE
100                 if(price > data.strikePrice){
101                         return (price - data.strikePrice);
102                 }
103                 else
104                         return 0;
105         }
106         DATA_T executePut(DATA_T& price){
107 #pragma HLS INLINE
108                 if(price < data.strikePrice){
109                         return (data.strikePrice - price);
110                 }
111                 else
112                         return 0;
113         }
114 };
```

3. Sum and determine the results

```
pCall[g_id] = (call0
              +call1
              +call2
              +call3
              +call4
              +call5
              +call6
              +call7
              +call8
              +call9
              +call10
              +call11
              +call12
              +call13
              +call14
              +call15
              )/sims;
```

```
pPut[g_id] =(put0
             +put1
             +put2
             +put3
             +put4
             +put5
             +put6
             +put7
             +put8
             +put9
             +put10
             +put11
             +put12
             +put13
             +put14
             +put15
             )/sims;
```

Main function

I use the host code from lab3 and adjust the kernel-related part.

The kernel interface is shown below. According to the kernel interface, the host function needs to prepare 2 axi master for pCall and pPut and 8 axi lite for the rest of parameters.

```
33  void blackEuro(float *pCall, float *pPut,   // call price and put price
34          float timeT,                // time period of options
35          float freeRate,         // interest rate of the riskless asset
36          float volatility,           // volatility of the risky asset
37          float initPrice,            // stock price at time 0
38          float strikePrice,// strike price
39          int steps,
40          int sims,
41          int g_id)
42  {
43  #pragma HLS INTERFACE m_axi port=pCall bundle=gmem
44  #pragma HLS INTERFACE s_axilite port=pCall bundle=control
45  #pragma HLS INTERFACE m_axi port=pPut bundle=gmem
46  #pragma HLS INTERFACE s_axilite port=pPut bundle=control
47  #pragma HLS INTERFACE s_axilite port=timeT bundle=control
48  #pragma HLS INTERFACE s_axilite port=freeRate bundle=control
49  #pragma HLS INTERFACE s_axilite port=volatility bundle=control
50  #pragma HLS INTERFACE s_axilite port=initPrice bundle=control
51  #pragma HLS INTERFACE s_axilite port=strikePrice bundle=control
52  #pragma HLS INTERFACE s_axilite port=steps bundle=control
53  #pragma HLS INTERFACE s_axilite port=sims bundle=control
54  #pragma HLS INTERFACE s_axilite port=g_id bundle=control
55  #pragma HLS INTERFACE s_axilite port=return bundle=control
```

The code below allocates memory for storing pCall and pPut and create write only buffer for them.

```
604     cout << "All parameters are successfully generated. ";
605
606     cout << "HOST-Info: Allocating memory for h_call    ... ";
607     void *ptr=nullptr;
608     if (posix_memalign(&ptr,4096,num_runs*sizeof(float))) {
609         cout << endl << "HOST-Error: Out of Memory during memory allocation for h_call" << endl << endl;
610         return EXIT_FAILURE;
611     }
612     cout << "h_call Allocated" << endl;
613     h_call = reinterpret_cast<float*>(ptr);
614
615     cout << endl;
616
617     cout << "HOST-Info: Allocating memory for h_put    ... ";
618     if (posix_memalign(&ptr,4096,num_runs*sizeof(float))) {
619         cout << endl << "HOST-Error: Out of Memory during memory allocation for h_put" << endl << endl;
620         return EXIT_FAILURE;
621     }
622     cout << "h_put Allocated" << endl;
623     h_put = reinterpret_cast<float*>(ptr);
```

```
625     // ------------------------------------------------------------------
626     // Step 4.2: Create Buffers in Global Memory to store data
627     //             o) d_call     - stores h_call (W)
628     //             o) d_put      - stores h_put (W)
629     // ------------------------------------------------------------------
630     cl_mem d_call, d_put;
631
632     d_call = clCreateBuffer(Context, CL_MEM_WRITE_ONLY | CL_MEM_USE_HOST_PTR, num_runs * sizeof(float),\
633                             h_call, &errCode);
634     if (errCode != CL_SUCCESS) {
635         cout << endl << "Host-Error: Failed to allocate Global Memory for d_call" << endl << endl;
636         return EXIT_FAILURE;
637     }
638
639     d_put = clCreateBuffer(Context, CL_MEM_WRITE_ONLY | CL_MEM_USE_HOST_PTR, num_runs * sizeof(float), \
640                            h_put, &errCode);
641     if (errCode != CL_SUCCESS) {
642         cout << endl << "Host-Error: Failed to allocate Global Memory for d_put" << endl << endl;
643         return EXIT_FAILURE;
644     }
```

The code below prepares the rest of the parameters and set kernel arguments.

```
594     cout << "HOST-Info: generate parameters    ... ";
595     cl_float initprice = 100;           // -s
596     cl_float strikeprice = 110;             // -k
597     cl_float rate = 0.05;           // -r
598     cl_float volatility = 0.2;  // -v
599     cl_float time = 1.0;                // -t
600     int num_runs=16;
601     cl_uint sims = 1024, steps=1024;
```

```
668     int Nb_Of_Mem_Events = 4,
669         Nb_Of_Exe_Events = num_runs;
670
671     cl_event Mem_op_event[Nb_Of_Mem_Events],
672              K_exe_event[Nb_Of_Exe_Events];
673
674     #ifdef ALL_MESSAGES
675     cout << endl;
676     cout << "HOST-Info: ============================================================ " << endl;
677     cout << "HOST-Info: (Step 5) Run Application                                      " << endl;
678     cout << "HOST-Info: ============================================================ " << endl;
679     #endif
```

```
681     // ----------------------------------------
682     // Step 5.1: Set Kernel Arguments
683     // ----------------------------------------
684     #ifdef ALL_MESSAGES
685     cout << "HOST-Info: Setting Kernel arguments ..." << endl;
686     #endif
687     errCode  = false;
688
689     errCode |= clSetKernelArg(K_blackEuro,  0, sizeof(cl_mem), &d_call);
690     errCode |= clSetKernelArg(K_blackEuro,  1, sizeof(cl_mem), &d_put);
691     errCode |= clSetKernelArg(K_blackEuro,  2, sizeof(cl_float), &time);
692     errCode |= clSetKernelArg(K_blackEuro,  3, sizeof(cl_float), &rate);
693     errCode |= clSetKernelArg(K_blackEuro,  4, sizeof(cl_float), &volatility);
694     errCode |= clSetKernelArg(K_blackEuro,  5, sizeof(cl_float), &initprice);
695     errCode |= clSetKernelArg(K_blackEuro,  6, sizeof(cl_float), &strikeprice);
696     errCode |= clSetKernelArg(K_blackEuro,  7, sizeof(cl_uint), &steps);
697     errCode |= clSetKernelArg(K_blackEuro,  8, sizeof(cl_uint), &sims);
698
699     if (errCode != CL_SUCCESS) {
700         cout << endl << "Host-ERROR: Failed to set Kernel arguments from 0 to 8" << endl << endl;
701         return EXIT_FAILURE;
702     }
```

```
704     errCode = clEnqueueMigrateMemObjects(Command_Queue, 1, &d_call, CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED, \
705     0, NULL, &Mem_op_event[0]);
706     if (errCode != CL_SUCCESS) {
707         cout << endl << "Host-Error: Failed Migrate d_call without migrating content" << endl << endl;
708         return EXIT_FAILURE;
709     }
710
711     // ----------------------------------------------------
712
713     errCode = clEnqueueMigrateMemObjects(Command_Queue, 1, &d_put, CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED, \
714     0, NULL, &Mem_op_event[1]);
715     if (errCode != CL_SUCCESS) {
716         cout << endl << "Host-Error: Failed Migrate d_put without migrating content" << endl << endl;
717         return EXIT_FAILURE;
718     }
```

The code below is submitting kernel for execution

```
728     // ----------------------------------------
729     // Step 5.3: Submit Kernels for Execution
730     // ----------------------------------------
731
732     cout << "HOST-Info: Submitting Kernel K_blackEuro ..." << endl;
733     errCode  = false;
734     for (int i=0; i < num_runs; i++) {
735         errCode |= clSetKernelArg(K_blackEuro,  9, sizeof(int), &i);
736         errCode |= clEnqueueTask(Command_Queue, K_blackEuro, 0, NULL, &K_exe_event[i]);
737     }
738     if (errCode != CL_SUCCESS) {
739         cout << endl << "HOST-Error: Failed to submit K_blackEuro" << endl << endl;
740         return EXIT_FAILURE;
741     }
```

<u>Kernel Function</u>

HLS allocation:

The origin pragma has wrong syntax:

```
56  #pragma HLS ALLOCATION instances=mul limit=1 operation
57  #pragma HLS ALLOCATION instances=fmul limit=1 operation
58  #pragma HLS ALLOCATION instances=fexp limit=1 operation
59  #pragma HLS ALLOCATION instances=fdiv limit=1 operation
```

```
WARNING: [HLS 207-5551] unexpected pragma argument 'instances', expects function/operation
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/launchSim.h:39:24)
WARNING: [HLS 207-5551] unexpected pragma argument 'instances', expects function/operation
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/blackEuro.cpp:57:24)
WARNING: [HLS 207-5551] unexpected pragma argument 'instances', expects function/operation
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/blackEuro.cpp:58:24)
WARNING: [HLS 207-5551] unexpected pragma argument 'instances', expects function/operation
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/blackEuro.cpp:59:24)
WARNING: [HLS 207-5551] unexpected pragma argument 'instances', expects function/operation
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/blackEuro.cpp:60:24)
```

The adjusted pragma:

```
62  #pragma HLS ALLOCATION operation instances=mul limit=1
63  #pragma HLS ALLOCATION operation instances=fmul limit=1
64  #pragma HLS ALLOCATION operation instances=fexp limit=1
65  #pragma HLS ALLOCATION operation instances=fdiv limit=1
```

Stream:

The origin pragma:

```
49  #pragma HLS STREAM variable=prices depth=NUM_SIMS dim=1
```

```
WARNING: [HLS 207-5505] the 'dim' option to 'Stream' pragma is not supported and will be ignored
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/blackScholes.h:49:56)
WARNING: [HLS 207-5505] the 'dim' option to 'Stream' pragma is not supported and will be ignored
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/launchSim.h:23:49)
WARNING: [HLS 207-5505] the 'dim' option to 'Stream' pragma is not supported and will be ignored
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/launchSim.h:24:49)
WARNING: [HLS 207-5505] the 'dim' option to 'Stream' pragma is not supported and will be ignored
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/launchSim.h:27:48)
WARNING: [HLS 207-5505] the 'dim' option to 'Stream' pragma is not supported and will be ignored
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/launchSim.h:28:48)
```

The adjusted pragma:

```
50    #pragma HLS STREAM variable=prices depth=NUM_SIMS
```

Function extract:

The origin pragma:

```
95          {
96  #pragma HLS FUNCTION_EXTRACT
97  #pragma HLS DATAFLOW
98                  launchSimulation(call0, put0, g_id+0, bs0, sims*steps>>4, sims>>4);
99                  launchSimulation(call1, put1, g_id+1, bs1, sims*steps>>4, sims>>4);
100                 launchSimulation(call2, put2, g_id+2, bs2, sims*steps>>4, sims>>4);
101                 launchSimulation(call3, put3, g_id+3, bs3, sims*steps>>4, sims>>4);
102                 launchSimulation(call4, put4, g_id+4, bs4, sims*steps>>4, sims>>4);
103                 launchSimulation(call5, put5, g_id+5, bs5, sims*steps>>4, sims>>4);
104                 launchSimulation(call6, put6, g_id+6, bs6, sims*steps>>4, sims>>4);
105                 launchSimulation(call7, put7, g_id+7, bs7, sims*steps>>4, sims>>4);
106                 launchSimulation(call8, put8, g_id+8, bs8, sims*steps>>4, sims>>4);
107                 launchSimulation(call9, put9, g_id+9, bs9, sims*steps>>4, sims>>4);
108                 launchSimulation(call10, put10, g_id+10, bs10, sims*steps>>4, sims>>4);
109                 launchSimulation(call11, put11, g_id+11, bs11, sims*steps>>4, sims>>4);
110                 launchSimulation(call12, put12, g_id+12, bs12, sims*steps>>4, sims>>4);
111                 launchSimulation(call13, put13, g_id+13, bs13, sims*steps>>4, sims>>4);
112                 launchSimulation(call14, put14, g_id+14, bs14, sims*steps>>4, sims>>4);
113                 launchSimulation(call15, put15, g_id+15, bs15, sims*steps>>4, sims>>4);
114         }
```

```
WARNING: [HLS 207-5566] unknown HLS pragma ignored
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/blackEuro.cpp:97:9)
WARNING: [HLS 207-5554] Only for-loops and functions support the dataflow
(/users/course/2022S/HLS17000000/g110061562/final/Vitis_V3/blackEuro_kernels/src/blackEuro.cpp:98:9)
```

The adjusted pragma:

```
102   //  {
103   //#pragma HLS FUNCTION_EXTRACT
104   //#pragma HLS DATAFLOW
105           launchSimulation(call0, put0, g_id+0, bs0, sims*steps>>4, sims>>4);
106           launchSimulation(call1, put1, g_id+1, bs1, sims*steps>>4, sims>>4);
107           launchSimulation(call2, put2, g_id+2, bs2, sims*steps>>4, sims>>4);
108           launchSimulation(call3, put3, g_id+3, bs3, sims*steps>>4, sims>>4);
109           launchSimulation(call4, put4, g_id+4, bs4, sims*steps>>4, sims>>4);
110           launchSimulation(call5, put5, g_id+5, bs5, sims*steps>>4, sims>>4);
111           launchSimulation(call6, put6, g_id+6, bs6, sims*steps>>4, sims>>4);
112           launchSimulation(call7, put7, g_id+7, bs7, sims*steps>>4, sims>>4);
113           launchSimulation(call8, put8, g_id+8, bs8, sims*steps>>4, sims>>4);
114           launchSimulation(call9, put9, g_id+9, bs9, sims*steps>>4, sims>>4);
115           launchSimulation(call10, put10, g_id+10, bs10, sims*steps>>4, sims>>4);
116           launchSimulation(call11, put11, g_id+11, bs11, sims*steps>>4, sims>>4);
117           launchSimulation(call12, put12, g_id+12, bs12, sims*steps>>4, sims>>4);
118           launchSimulation(call13, put13, g_id+13, bs13, sims*steps>>4, sims>>4);
119           launchSimulation(call14, put14, g_id+14, bs14, sims*steps>>4, sims>>4);
120           launchSimulation(call15, put15, g_id+15, bs15, sims*steps>>4, sims>>4);
121   //  }
```

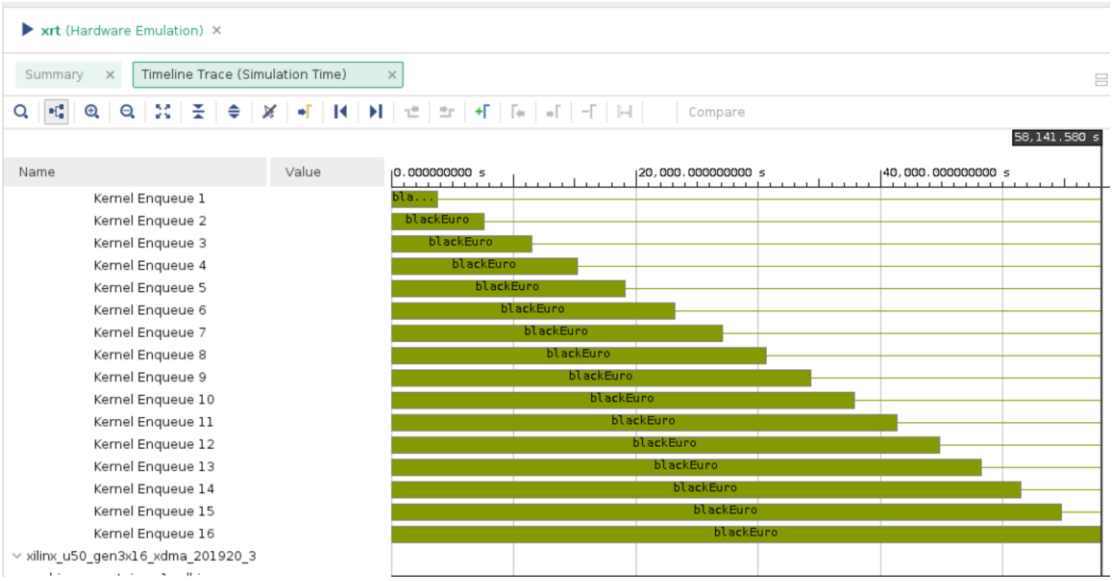
```
18      template<typename DATA_T, typename Model>
19    void launchSim(DATA_T &pCall, DATA_T &pPut, RNG<DATA_T> &rng0,RNG<DATA_T> &rng1, Model &m, int numR, int sims){
20    #pragma HLS function_instantiate variable=rng0
21    #pragma HLS DATAFLOW
```

I add function_instantiate to make 16 launchSimulation run in parallel.

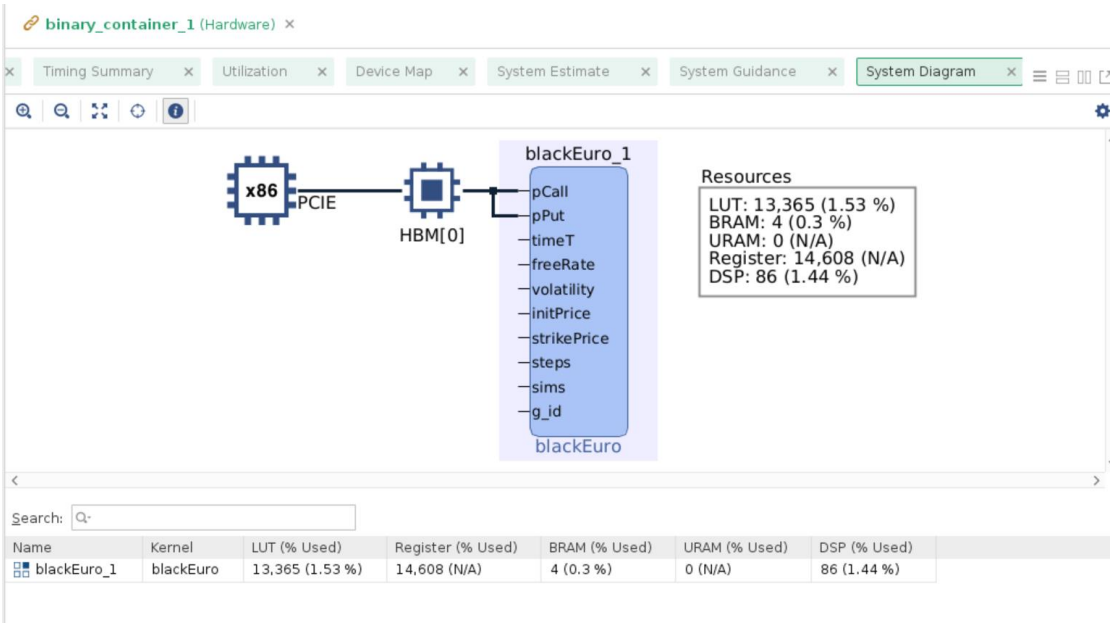
Results

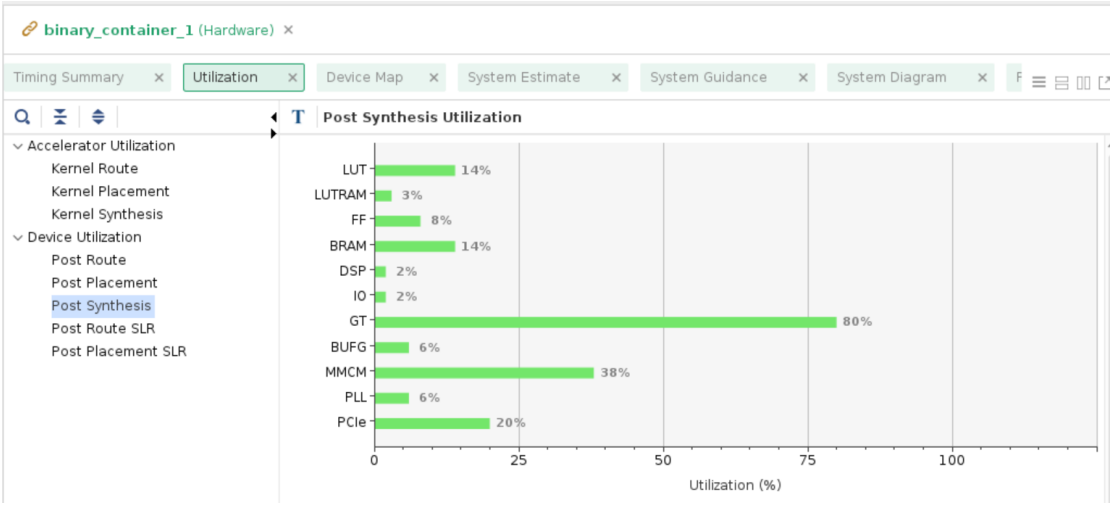
```
HOST_Info: Waiting for application to be completed ...
The 1st call price is 6.26604
The 1st put price is 10.8582
The 2nd call price is 6.4935
The 2nd put price is 10.7894
The 3th call price is 6.45818
The 3th put price is 10.7916
The 4th call price is 6.36644
The 4th put price is 11.173
The 5th call price is 6.3803
The 5th put price is 11.2333
The 6th call price is 6.42416
The 6th put price is 11.3428
The 7th call price is 6.49111
The 7th put price is 11.2489
The 8th call price is 6.46597
The 8th put price is 11.3446
The 9th call price is 6.28576
The 9th put price is 11.4276
The 10th call price is 6.26392
The 10th put price is 11.5545
The 11th call price is 1.34525e-43
The 11th put price is 1.66313e+22
The 12th call price is 6.31635
The 12th put price is 11.4937
The 13th call price is 6.40219
The 14th call price is 0
The 14th put price is 7.18483e+22
The 15th call price is 0
The 15th put price is 0
The 16th call price is 6.26251
The 16th put price is 11.8452
the sum of call price is: 82.8764
the sum of put price is: 8.84795e+22
the call price is: 4.92716      the difference with the reference value is 592.716%
the put price is: 5.26027e+21   the difference with the reference value is 1185.82%
```

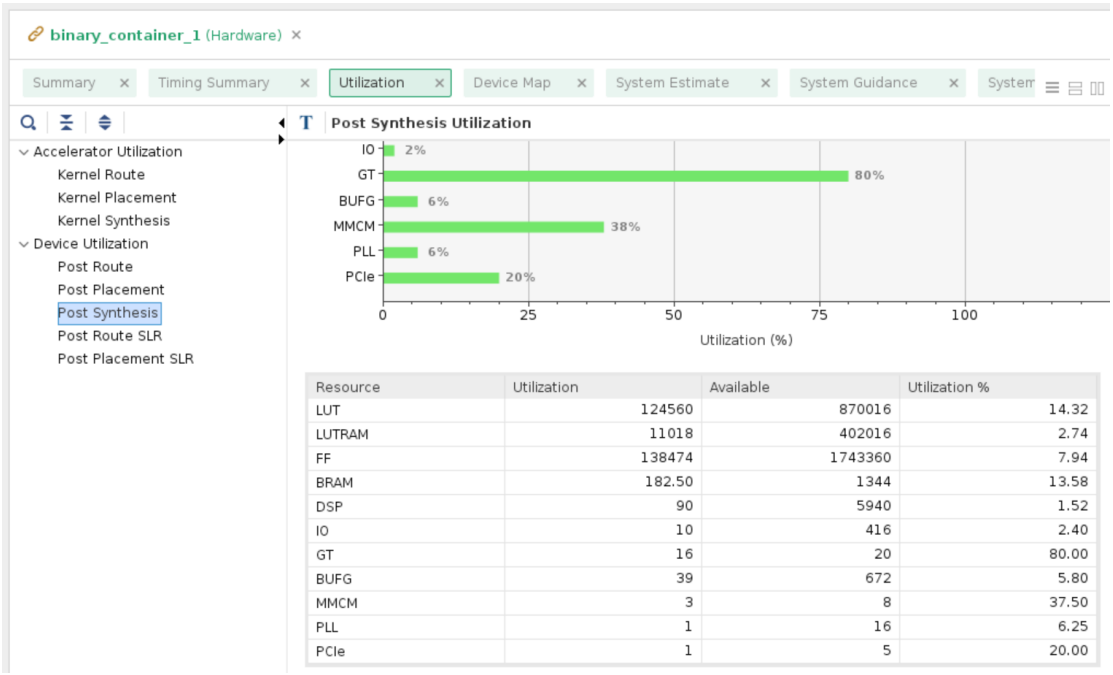
Hardware emulation timeline trace:



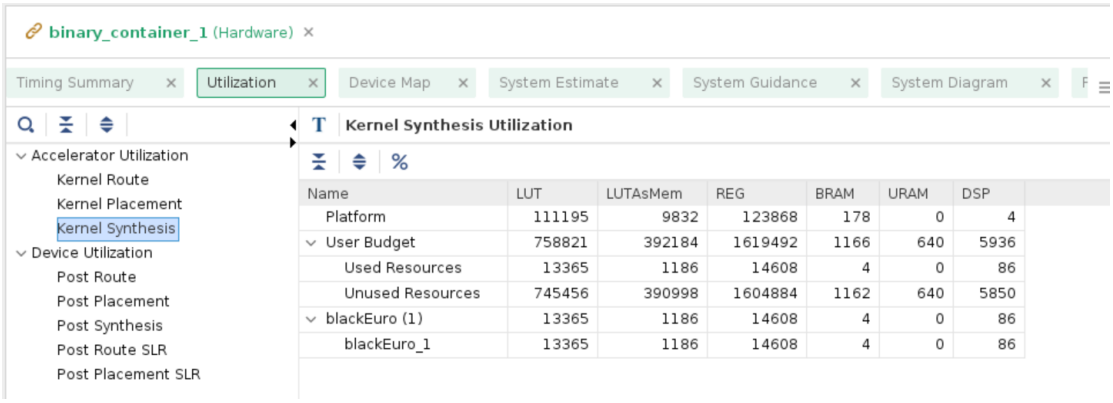
Hardware utilization:



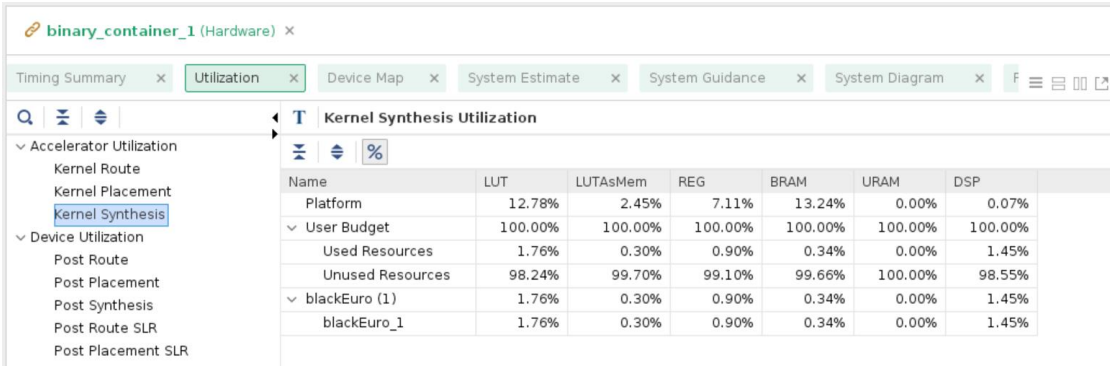




Resource	Utilization	Available	Utilization %
LUT	124560	870016	14.32
LUTRAM	11018	402016	2.74
FF	138474	1743360	7.94
BRAM	182.50	1344	13.58
DSP	90	5940	1.52
IO	10	416	2.40
GT	16	20	80.00
BUFG	39	672	5.80
MMCM	3	8	37.50
PLL	1	16	6.25
PCIe	1	5	20.00



Name	LUT	LUTAsMem	REG	BRAM	URAM	DSP
Platform	111195	9832	123868	178	0	4
User Budget	758821	392184	1619492	1166	640	5936
Used Resources	13365	1186	14608	4	0	86
Unused Resources	745456	390998	1604884	1162	640	5850
blackEuro (1)	13365	1186	14608	4	0	86
blackEuro_1	13365	1186	14608	4	0	86



Name	LUT	LUTAsMem	REG	BRAM	URAM	DSP
Platform	12.78%	2.45%	7.11%	13.24%	0.00%	0.07%
User Budget	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Used Resources	1.76%	0.30%	0.90%	0.34%	0.00%	1.45%
Unused Resources	98.24%	99.70%	99.10%	99.66%	100.00%	98.55%
blackEuro (1)	1.76%	0.30%	0.90%	0.34%	0.00%	1.45%
blackEuro_1	1.76%	0.30%	0.90%	0.34%	0.00%	1.45%

Reference

L. Ma, F. B. Muslim and L. Lavagno, "High Performance and Low Power Monte Carlo Methods to Option Pricing Models via High Level Design and Synthesis," 2016 European Modelling Symposium (EMS), 2016, pp. 157-162, doi:

10.1109/EMS.2016.036.
https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/pragma-HLS-inline
https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/pragma-HLS-allocation
https://www.boledu.org/textbooks/hls-textbook/io-interface/axi-lite
https://github.com/dr-liangma/FinancialModels_AmazonF1