Issues on Massively Parallel Monte Carlo Simulation for ELS pricing

유 현곤
연세대학교 수학과

Email : yhgon@yonsei.ac.kr

2008년 6월 12일 목요일

연세대학교
YONSEI UNIVERSITY
금융퀀트연구센터

Agenda

1. Introduction for Option Pricing

2. Monte Carlo Simulation
   RNGs, Transformation, limits

3. Parallel Computing H/W (SMP, Cluster, Cell, CS, GPGPU)

4. Parallel Computing S/W (pMC)

5. CUDA technology H/W architecture & pMC

6. Implementation on ELS pricing and hedging

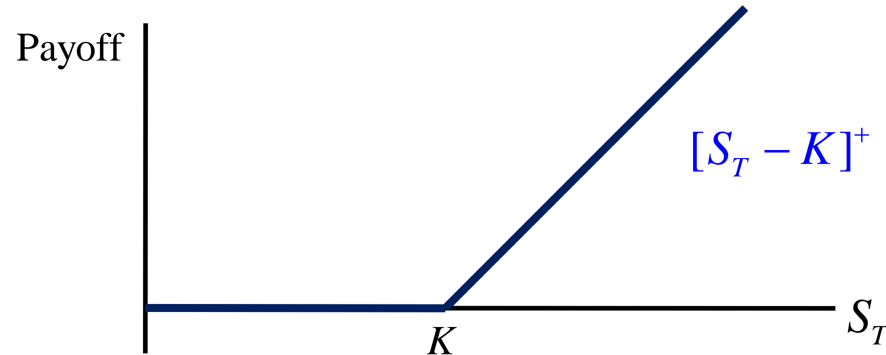
Part1 계산 재무에 대한 이해

Introduction for Option Pricing

European Call Option

1. asset dynamics GBM : $dS_t = \mu S_t dt + \sigma S_t dW_t$

2. payoff $[S_T - K]^+$



3. option price is $C_t = E^Q[e^{-r\tau}[S_T - K]^+]$

Introduction for Option Pricing

European Call Option

$$C_t = E^Q[e^{-r\tau}[S_T - K]^+]$$

4. By Ito's lemma

$$S_T{}^Q = S_0 e^{(r-\frac{1}{2}\sigma^2)\tau + \sigma W_\tau}$$

$$S_T = S_0 e^{(r-\frac{1}{2}\sigma^2)\tau + \sigma\sqrt{\tau}N(0,1)}$$

5. By Feynmann-Kac Theorem

BS-PDE : $f_t + rxf_x + \frac{1}{2}\sigma^2 x^2 f_{xx} = rf$

6. Closed Form Solution

$$C_t = S_0\Phi(d_1) - Ke^{-r\tau}\Phi(d_2)$$

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-x^2/2} dz \qquad d_1 = \frac{\ln(S_0/K) + (r - \sigma^2/2)\tau}{\sigma\sqrt{\tau}} \qquad d_2 = d_1 - \sigma\sqrt{\tau}$$

Pricing Methods

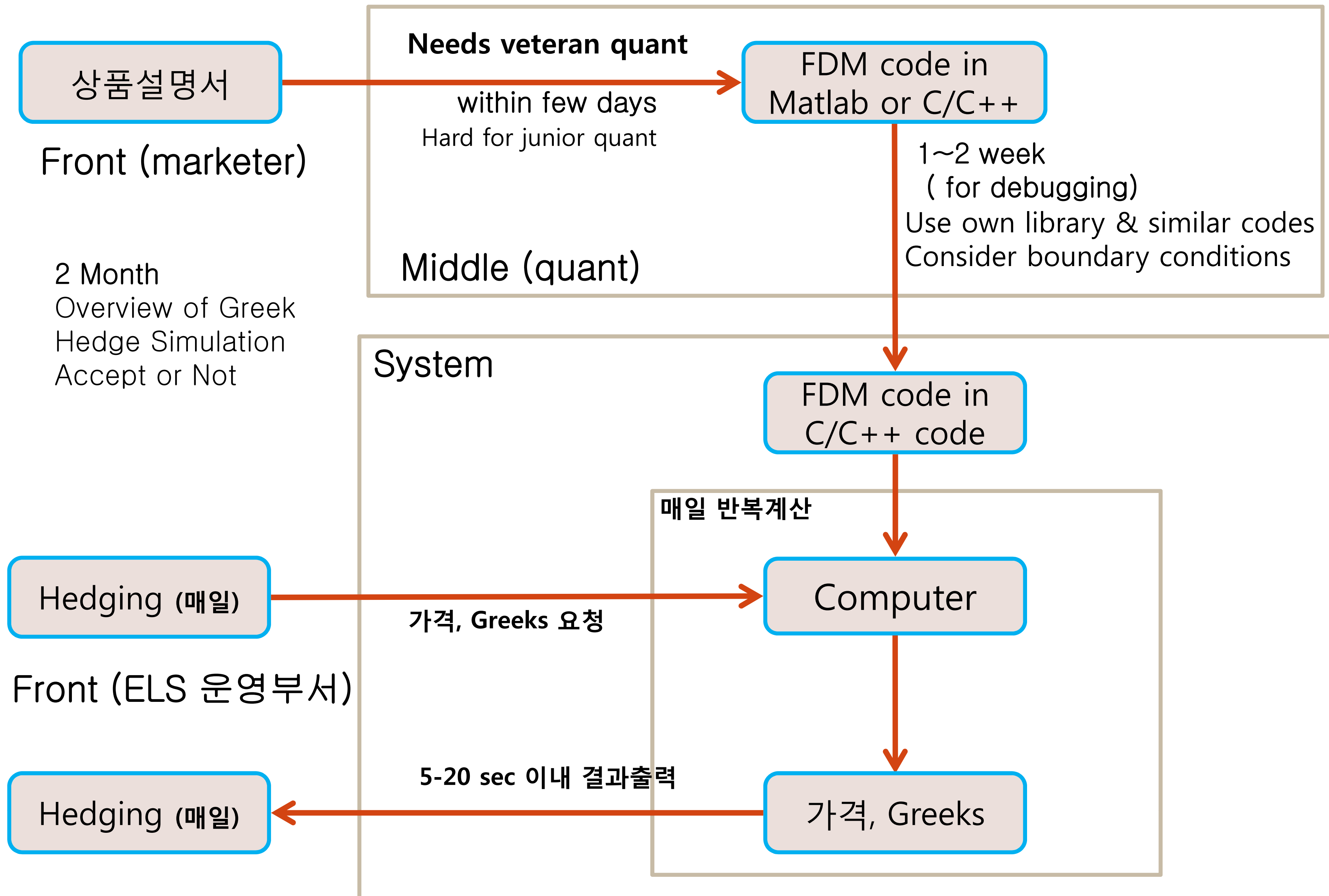
Closed form solution

Lattice(tree) method

Numerical Solution for PDE in FDM, FEM, Meshless

Monte Carlo Simulation

새로운 구조의 상품에 대한 분석작업 (FDM)



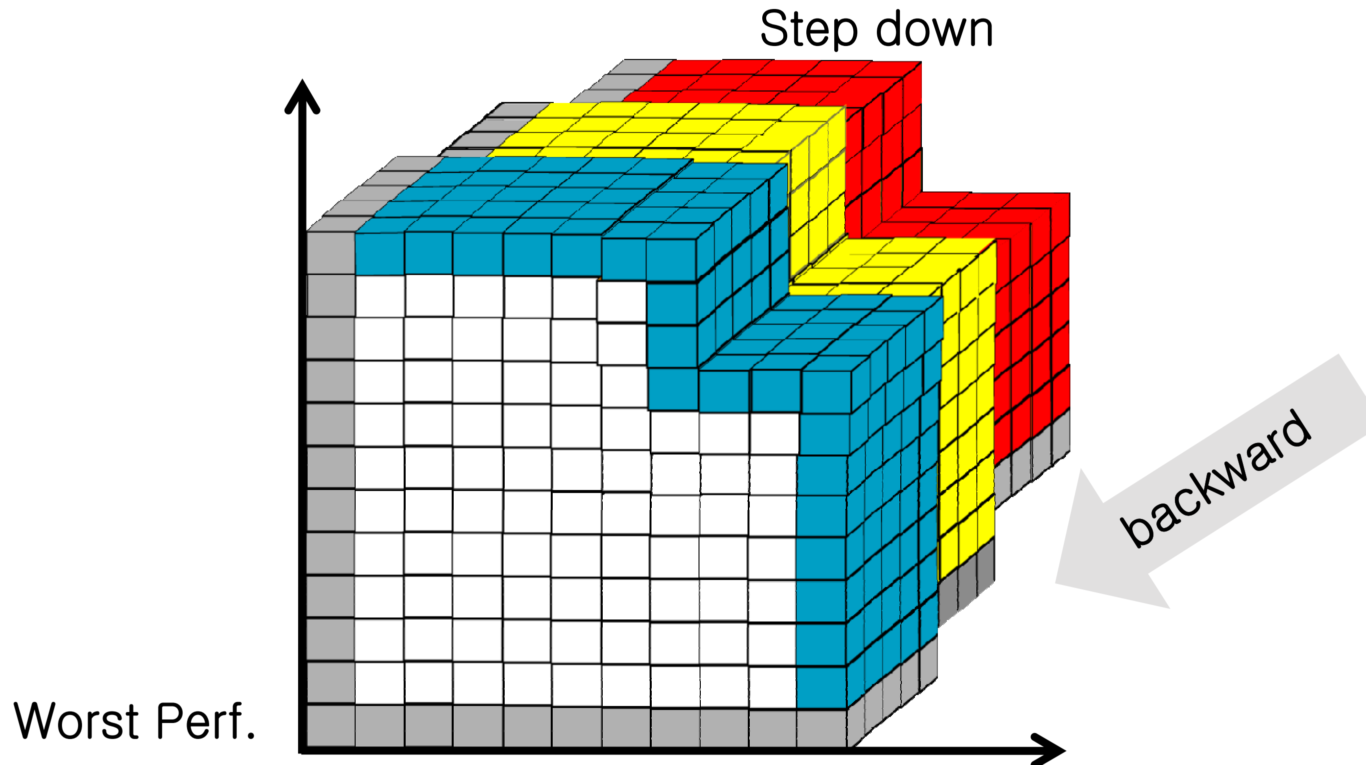
연세대학교
YONSEI UNIVERSITY
금융퀀트연구센터

PDE method for ELS

$$\frac{\partial f}{\partial t} + rx\frac{\partial f}{\partial x} + ry\frac{\partial f}{\partial y} + \tfrac{1}{2}\sigma_x^2\frac{\partial^2 f}{\partial x^2} + \rho\sigma_x\sigma_y\frac{\partial^2 f}{\partial x\partial y} + \tfrac{1}{2}\sigma_y^2\frac{\partial^2 f}{\partial y^2} = rf$$

With Boundary Value& Terminal Payoff

Domain for 2-Star n-chance Stepdown ELS



Monte Carlo Simulation

Law of Large Number : we can compute expectation

$$\mathrm{E}[e^{-r\tau}[S_T - K]^+] = \lim_{N\to\infty} \frac{1}{N} \sum_{i}^{N} e^{-r\tau}[S_{T_i} - K]^+$$

We can generate any process S_T **from given dynamics**

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

$$dS_t = \mu S_t dt + f(Y_t) S_t dW_t$$

Monte Carlo Simulation

Why MC is need ?

- Easy to imply

- Complex structure : path-dependence, prepayment, complex payoff, etc.

- High demension : multi-asset problems (n>4)

➔ MC is the only solution (or sparse grid method)

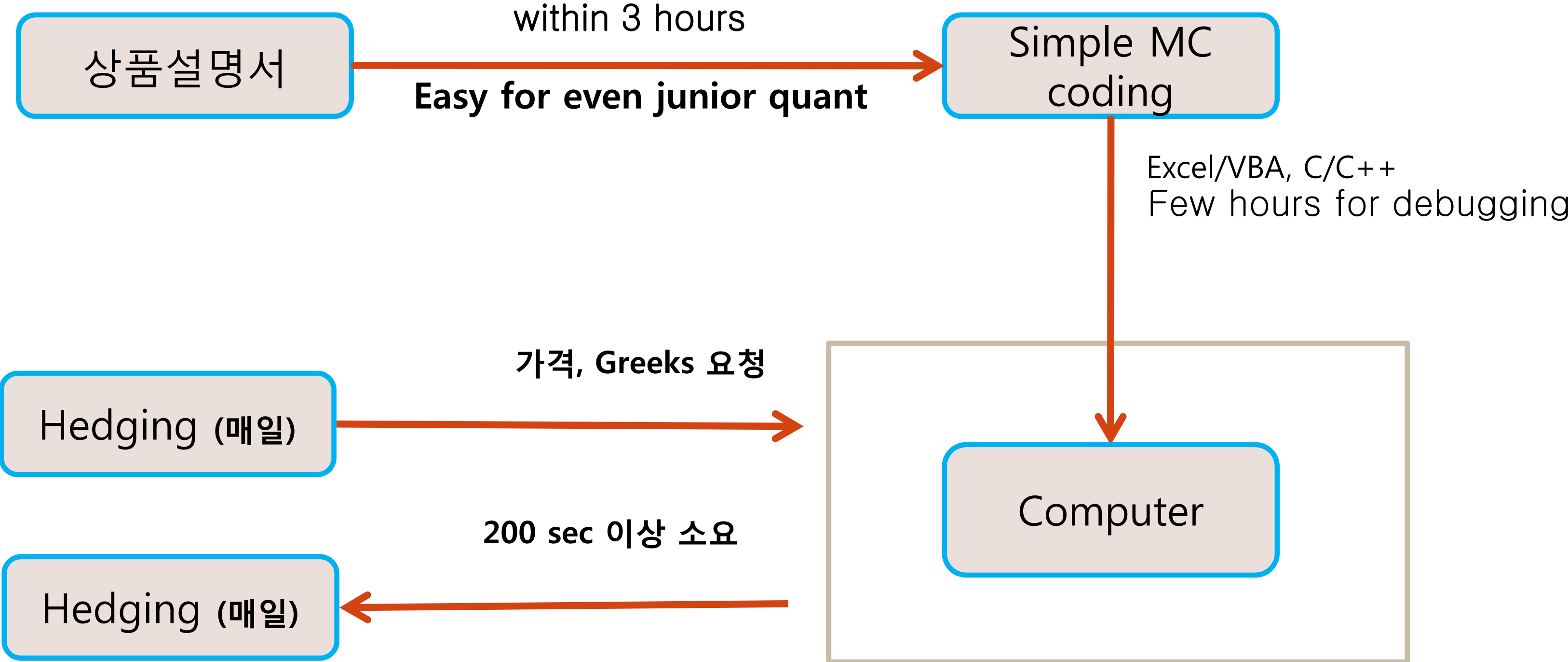
Why MC is not used ?

- Most contracts(95%) can be solved without MC. (closed, FDM)

- **Computation Time** : Too slow to get accurate results
  -1 minutes for each pricing. 4 minutes for greeks
  100 contracts : 7 hours to comutes (1 day : 6 hours)
  시나리오분석용 Greek plot 그리려면 하루종일걸림

- **Unstable sensitivity : non-smooth greeks plot**

Monte Carlo Simulation in Finance



계산시간으로 인하여 구현의 어려움

상품 발행시	상품거래시
Overview of Greek	Greeks Search
Hedge Simulation	VaR계산, 위험관리

Reasonable Solutions to speed up MC

1. **New theory in convergence**
   Malliavin Calculus, Operator Technique, Asymptotics

2. **Fast pseudo & quasi-RNGs, Transformation**

3. **Control Variate, Variance Reduction**

4. **Using Powerful Computer**

5. **Parallel Computing**
   HPC(use many CPUs) - OpenMP(SMP), MPI (Cluster)
   Alternative Method - IBM Cell BE, ClearSpeed, **GPGPU(CUDA)**

이상적 모델 : 1,2,3.4를 먼저 실시하고 5에 관심을 갖는다.

Part2 병렬처리

HPC와 분산처리의 차이

병렬처리는 크게 3가지로 나뉘는데

하나의 방법은 HPC로, 하나의 작업을 여러 개로 쪼개어 계산하도록 함으로써 계산시간을 단축시킴

다른 하나의 방법은 여러 개의 작업을 여러 개의 서버에 분산처리시켜 전체작업시간을 단축시키는 기법

웹서버, 게임서버에 사용되는 로드벨런싱도 분산처리기법의 하나임

계산의 관점에서 병렬처리는 HPC를 의미함

분산처리의 예
하나의 자산당 200초 걸리는 자산 200개가 있다. 총 40000초 걸림
200개의 서버에서 각 자산 정보를 보내서 돌린 결과를 받음
→병렬처리 할 필요없이 200초만에 모든 VaR 계산
Quant에게 유리(다시 코딩할 필요 없음), 회사입장 비용문제(수십억원)

금융권에서의 병렬화 진행상황

일반 프로그래밍

Excel/VBA 혹은 C/C++ 금융 프로그래밍

Excel2007에서 excel 내장 함수가 아닌 Excel/VBA, C/C++을 사용하면 오히려 속도가 느려지는 현상이 발생하는 경우가 있음 (excel에서 cell단위 자동분산처리함)

분산처리

여러명이 동시에 작업을 수행함, 혹은 여러대의 컴퓨터가 동시에 여러작업을 수행함

서버가 알아서 처리하므로 Quant는 신경쓸 필요 없음

현재 많은 중소형 증권사 (4년 전 대부분 금융권 – HPC컨퍼런스 Intel 한국지사 부사장曰)

HPC

Cluster기반 병렬시스템을 구축하였거나 구축하려는 추세 (ELS 라이센싱 대형 증권사)

외산 병렬 시스템 도입을 고려 (대형 은행)

병렬 프로그래밍이 가능한 퀀트 필요 – 인력이 절대적으로 부족함

금융관련 대학 : -- 연세대 수학과(cluster구축) 유일

비금융 대학 : 타 전공은 많은 편임 (CS, 기계 등)

대안적 가속처리 (IBM Cell, CS, CUDA 등)

최근 관심이 높아짐, 하지만, 역시 인력부족

금융관련 대학 : 연세대 수학과 유일 (CS, CUDA), 해외(Oxford Univ.)

비금융 대학 : 이화여대 그래픽연구실(CUDA) , 각대학 컴공, 물리, 천문대기, 기계 등

병렬처리(HPC – High performance Computing)

하나의 작업을 여러 개의 CPU(Cores)로
작업을 실행시켜 계산시간을 줄이는 방법

Wall Clock Time

Task를 병렬처리 에 의해 처리하는데 걸린 전체 시간

Wall Clock Time 1

Original Jobs　　Parallel Overheads

Wall Clock Time 2

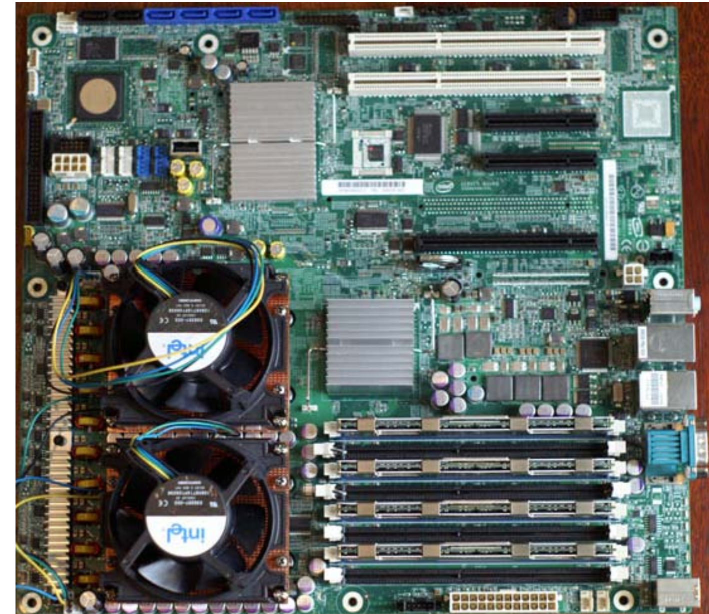
병렬 프로그래밍을 위한 시스템 구축

하드웨어 구조

HPC in SMP Machine

2 socket for quad core = 최대 8X 속도 향상

Each cores share system memory (upto 64GB)
Support pthread programming
standard OpenMP



메인보드의 한계로 현재는 8Core가 한계이다.

현대 대부분의 PC 및 Workstation은 SMP기반이다.

HPC in cluster

여러대의 SMP를 네트웍으로 연결

Each Nodes has 2-8 CPUs
4~10 Gflops in DP for each CPUs
Each Nodes connected by Infiniband, Gigbit Ethernet
1024+ nodes system is possible

Support standard OpenMP & MPI library
Benefit : develop envirinment, Technical supports from vender
PRNGs library for Monte Carlo simulation

MC의 경우 1만배 이상 속도 향상을 기대할 수 있음 : 하지만 구축 및 관리비용이 많음

Too expensive
기상청 15Tflops 서버 도입비용 – 200억 이상 (전기, 쿨링, 공간, 네트웍, 관리자 등 유지비 필요)

금융권에 필요한 성능의 서버
HPC for 400 Gflops 서버 도입비용 - 수억원 이상 (유지비, 설치공간 등 별도) IBM, HP 등
개발용 8 workstation : 300만원~1천만원 정도면 whitebox 구축(제작)가능
개발용 2 node 2*8 core cluster : 1천만원 정도면 whitebox 구축 가능

대안적 방법

PC용 CPU는 계산위주이 아닌 범용으로 개발됨

계산위주란?
빠른 처리속도와 더불어 CPU 내부의 On-chip 메모리가 큼
여러 개의 명령을 동시에 실행, ALU, FP 연산용 모듈이 많이 내장

Intel은 내부 메모리를 줄인 저가버전 등등을 출시 (가격경쟁을 위해)

따라서, 계산에 최적화된 하드웨어를 통한 병렬화 기법이
최근 HPC 업계에 주목받고 있음.

대표적인 예가
IBM Cell BE, Clearspeed, GPGPU 등임

대안1 IBM BladeCenter QS20 , QS21, QS22

Each Nodes has 3.2Ghz Cell BE processor
One Cell BE has 8 SPE units for computing

QS20 : 204 Gflops in SP, 21 Gflops in DP
QS21 : 408 Gflops in SP, 42 Gflops in DP
QS22 : 460 Gflops in SP, 217 Gflops in DP
QS21 : 5 Tflops in SP/ 588Gflops in DP in for blade chassis

Cell BE CPU ws developed for PS3

The initial target of Cell BE is entertainment – it does not need DP, so Cell BE provides very limited double precision in QS20, QS21, but QS22 support DP with 5 times fast.

IBM use this artitecture for building next generation world fastest super computer Roadrunner Project.

10X faster than normal CPU based cluster 하지만, 고가임.

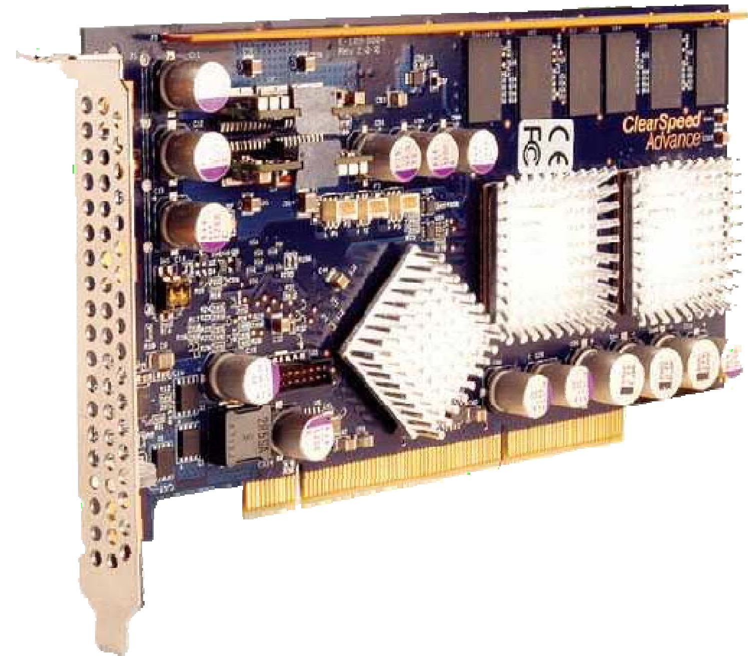
Cell BE SDK 개발자 거의 없음
- 금융 뿐만 아니라 국내 CS분야에도 극소수
- 게임 개발 분야에 극소수 종사

대안1 Mucury Cell Accelerator

2.8Ghz Cell BE processor in board
One Cell BE has 8 SPE units for computing

Cell BE processor at 2.8 GHz
SP 180 GFLOPS in PCI Express accelerator card
PCI Express x16 interface with raw data rate of 4 GB/s in each direction
Gigabit Ethernet interface
1-GB XDR DRAM, 2 channels each, 512 MB
4 GB DDR2, 2 channels each, 2 GB
162W for each board

대안2 ClearSpeed CSe620 Accelerate Board



Acceleration Board for HPC
Parallel 192= 2* 96 PE
66 GFlops in Double Precision

Support standard C/C++ SDK
Using CSCN compiler
15W per each board

cheaper than normal cluster solution
4 board system : 300Gflops

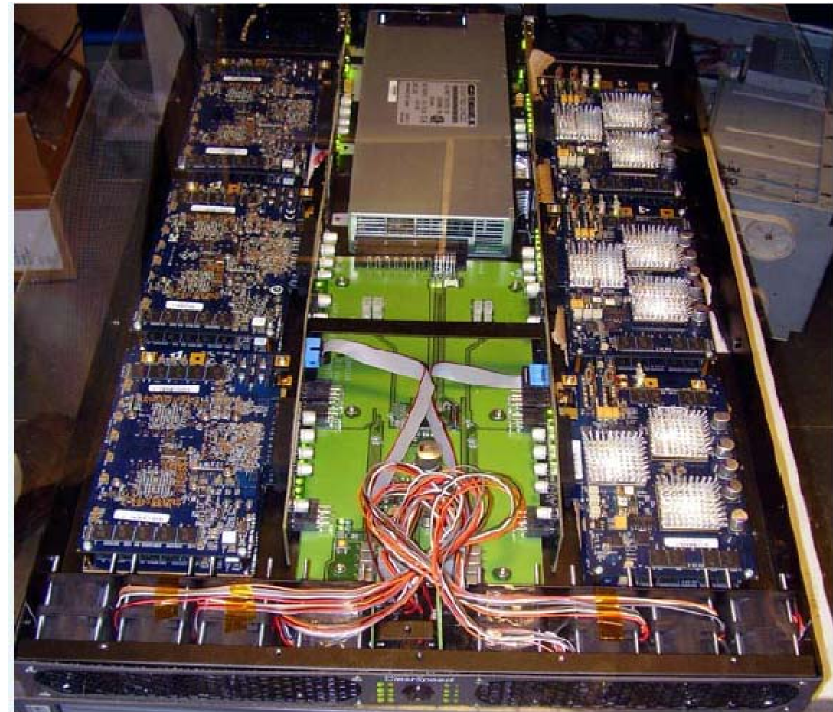
전력소모가 작고, 비교적 발열이 적어 PC, 4U 서버 등에 쉽게 장착 가능

대안2 CATs with Clearspeed

CS announce CATs in SC07

spec
12 CS board in 1U rack
Power : total 550W

12* 192 Pes = massively parallel
1 DP Tflops



하나의 서버에 최대 12개의 카드 장착가능

대안3 Nvidia Tesla C870, D870, S870

Computing by GPU
Nvidia 8800GTX chipset
Parallel 128 Stream Processors in one chip
500 GFlops in Single Precision

C870 : 1 GPU in board
D870 : 2 GPUs in case
S870 : 4 GPUs in 1U chassis

Support standard C/C++ SDK

CUDA

C870 system	SP 500Gflops	: 70 만원
C870 4board system	SP 2Tflops	: 300만원 + PC비용
S870 4GPUs system	SP 2Tflops	: 600 만원+ 서버비용

Cheap & powerful solution

몇 년 기다리면 빠른 컴퓨터가 나오지 않을까?

그냥 엔터치면 결과값이 나올텐데..
지금 병렬화를 배워서 금융에 적용해야하나?

현재 출시되는 모든 CPU는 Dual Core, Quad코어 기반임

추후 출시되는 CPU는 8 core, 16 core, 32 core로
연산속도 증가보다는 하나의 칩에 코어의 개수를
늘리는 방식으로 개발되고 있음

이러한 멀티코어 시스템의 성능을 100% 발휘하도록 하려면

모두 병렬 프로그래밍을 해주어야 함.

대안적 방법의 미래는?

Intel will launch (within 2 years)
DP 100 Gflops CPU with 8~16 cores in one chip

CS will launch (within few month)
DP 200 Gflops with 198 PEs in one board

Nvidia will launch (within 1 years)
SP 993 Gflops with 240 SPs in one board

IBM will launch (within 3 years)
SP 1 Tflops with some SPEs in one chip

AMD & Intel will also launch Stream processors

병렬 프로그래밍을 위한 S/W구현

Learning Curve (기초교육)

프로젝트, 경험

기초지식

Excel/VBA

C/C++언어

C언어 기초 지식 필요 (포인터, 구조체)

C++언어 기초 지식 필요 (template, STL 등)

Windows Visual Studio IDE, Linux Makefile 경험

excel dll link, excel winsock dll link

Closed form BS option pricing
Monte Carlo simulation
ELS pricing

Closed form BS option pricing
Monte Carlo simulation
RNG (LCG, MT19937)
ELS pricing

병렬처리 개념

CPU병렬화

OpenMP

MPI

1주일

Parallel Monte Carlo Simulation
RNG (split, multiseed, DC)
ELS pricing

CUDA

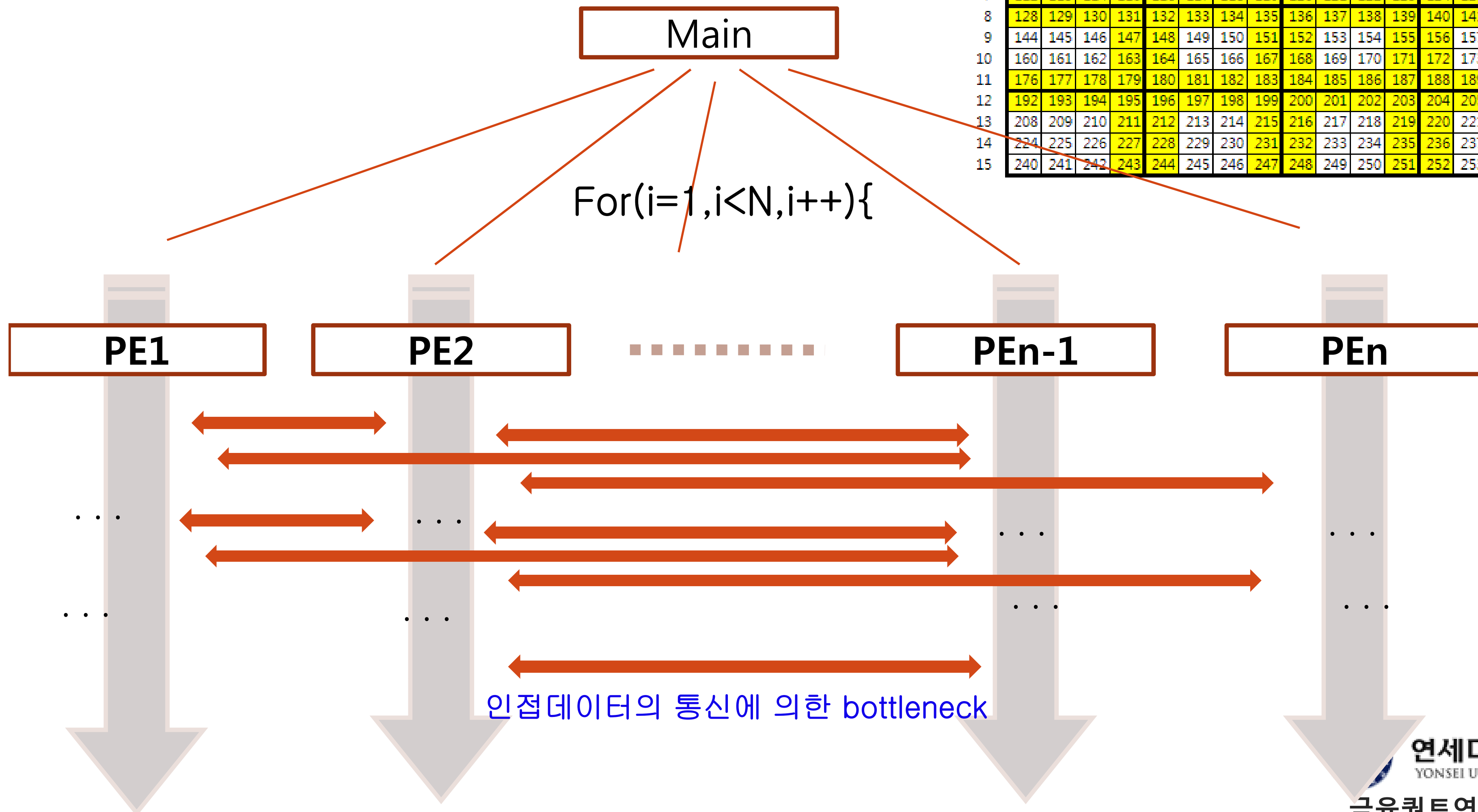
3주일

Massively pMC
ELS pricing

기초적인 CUDA ELS pricing 가능

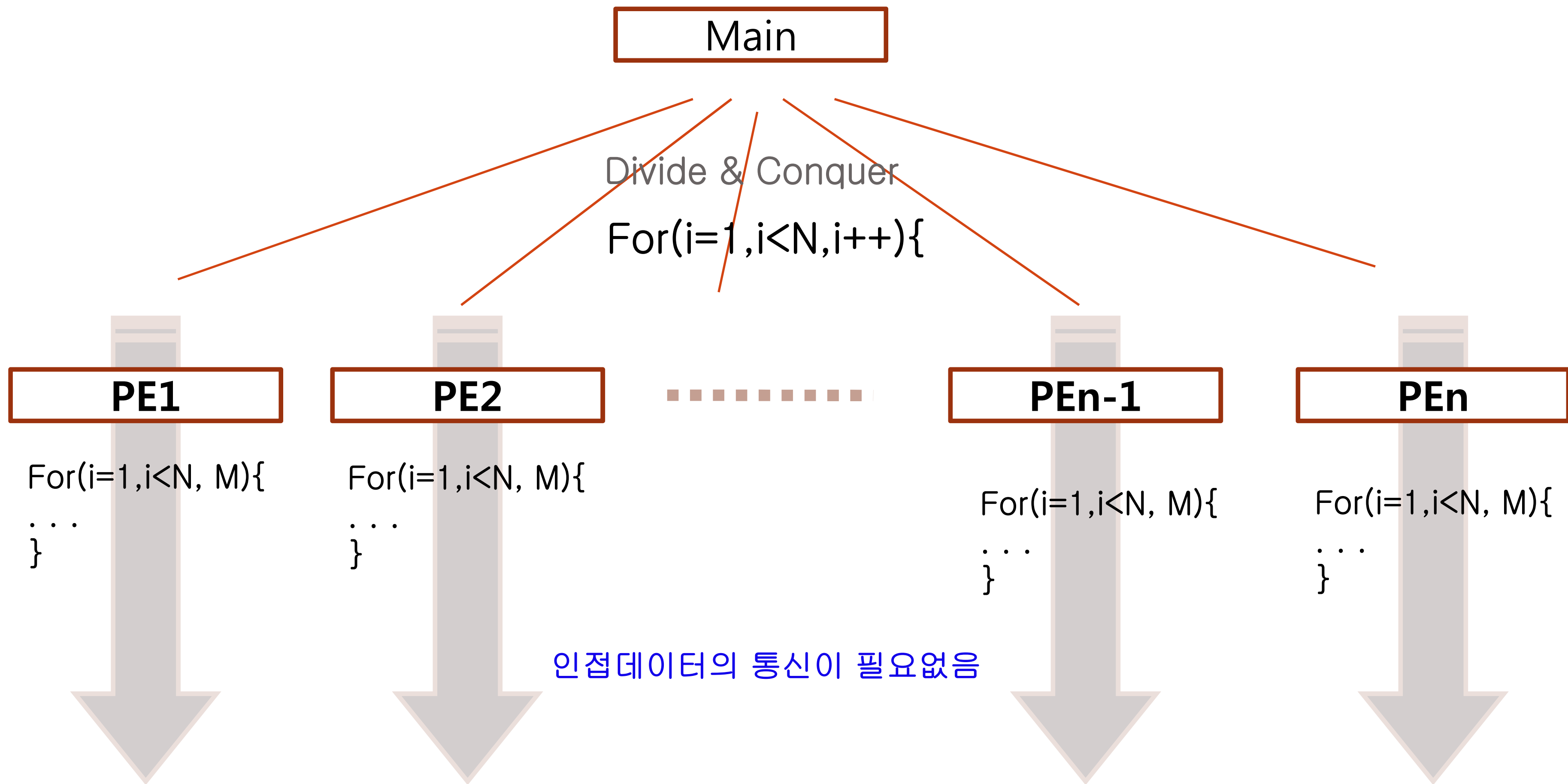
Parallel FDM, FEM (matrix)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
10	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
11	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
12	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
13	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
14	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
15	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255



연세대학교
YONSEI UNIVERSITY
금융퀀트연구센터

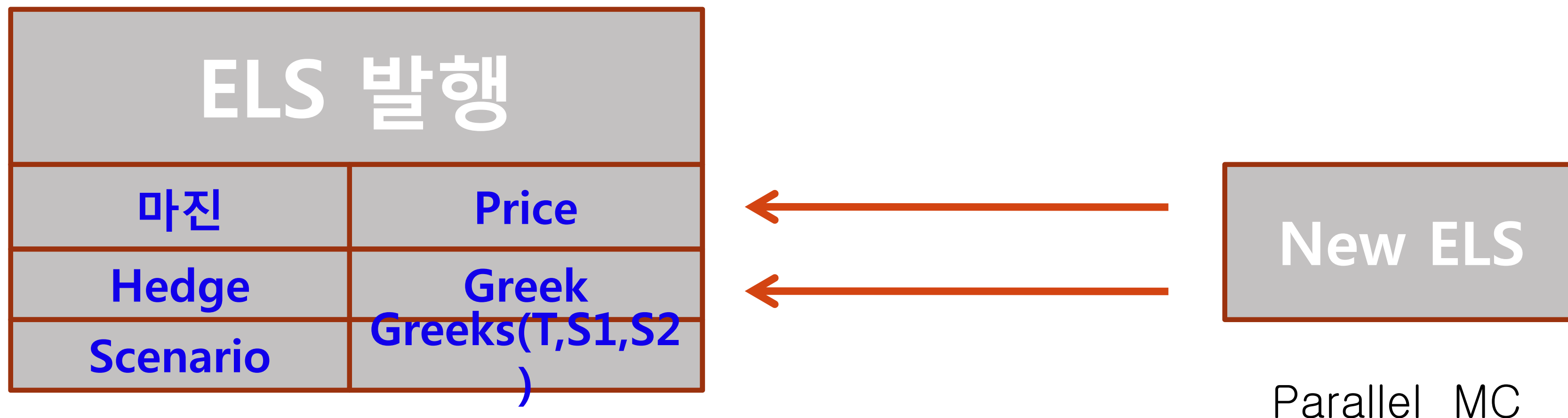
Parallel MC simulation



: MC 병렬 scailability가 가장 좋다

Application for Massively Parallel MC

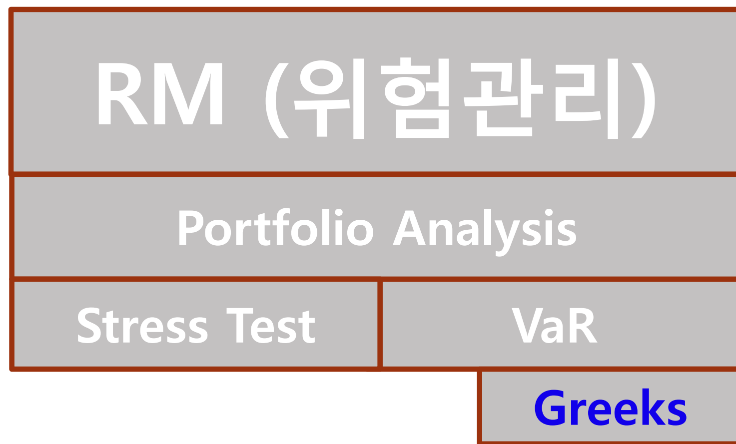
신규 상품 발생시 Front 입장



속도는 single FDM보다 빠르고 개발주기는 훨씬 짧음

Application for Massively Parallel MC

VaR을 통한 위험관리시



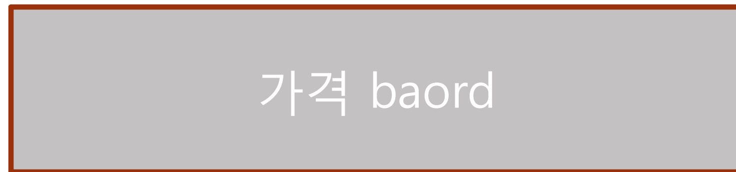
Parallel MC

위험관리용 계산시간을
획기적으로 단축시킴

Parallel Tasks

Application for Massively Parallel MC

채평사

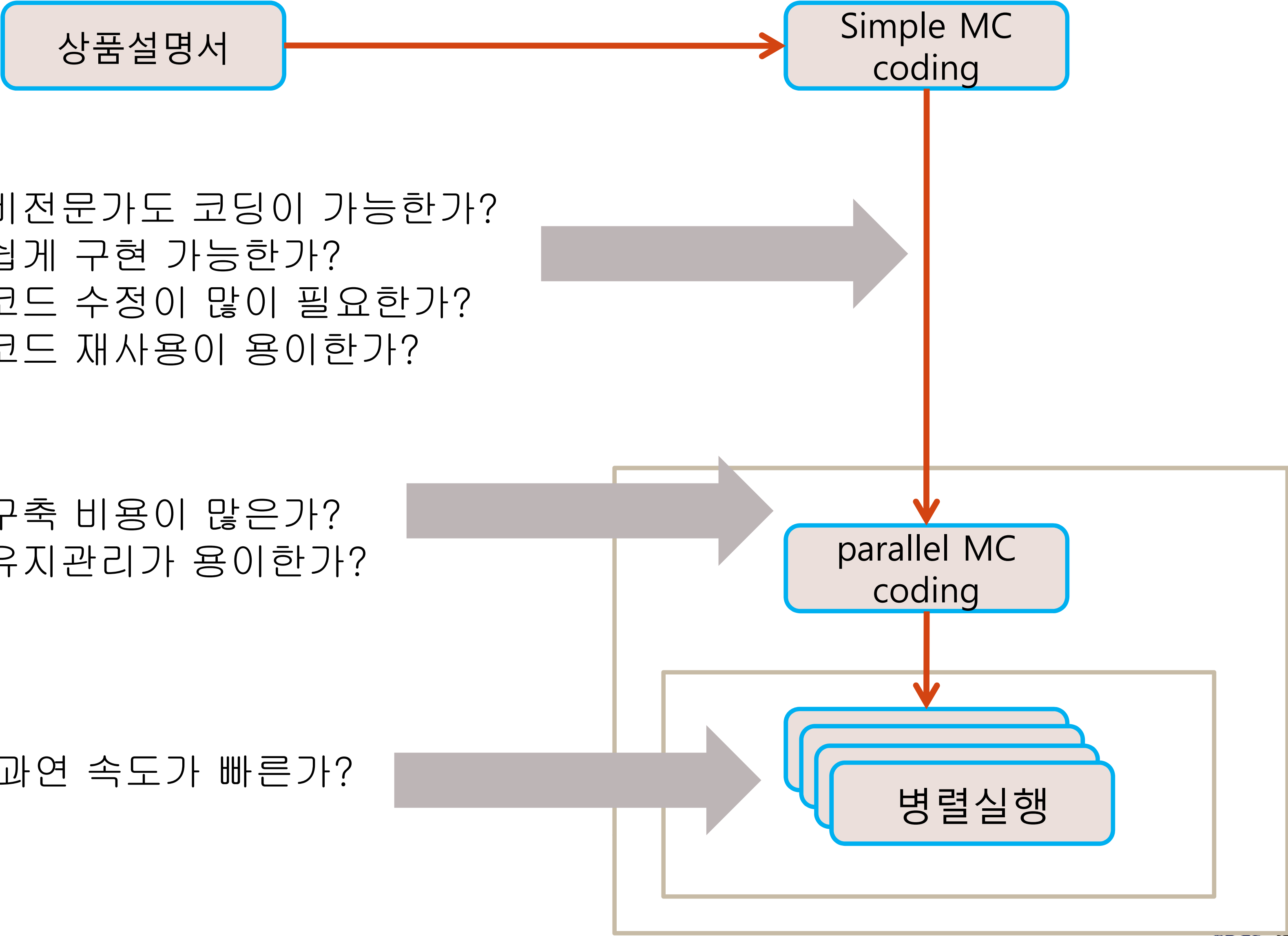


Parallel MC를 사용하여
기존 MC의 가속화

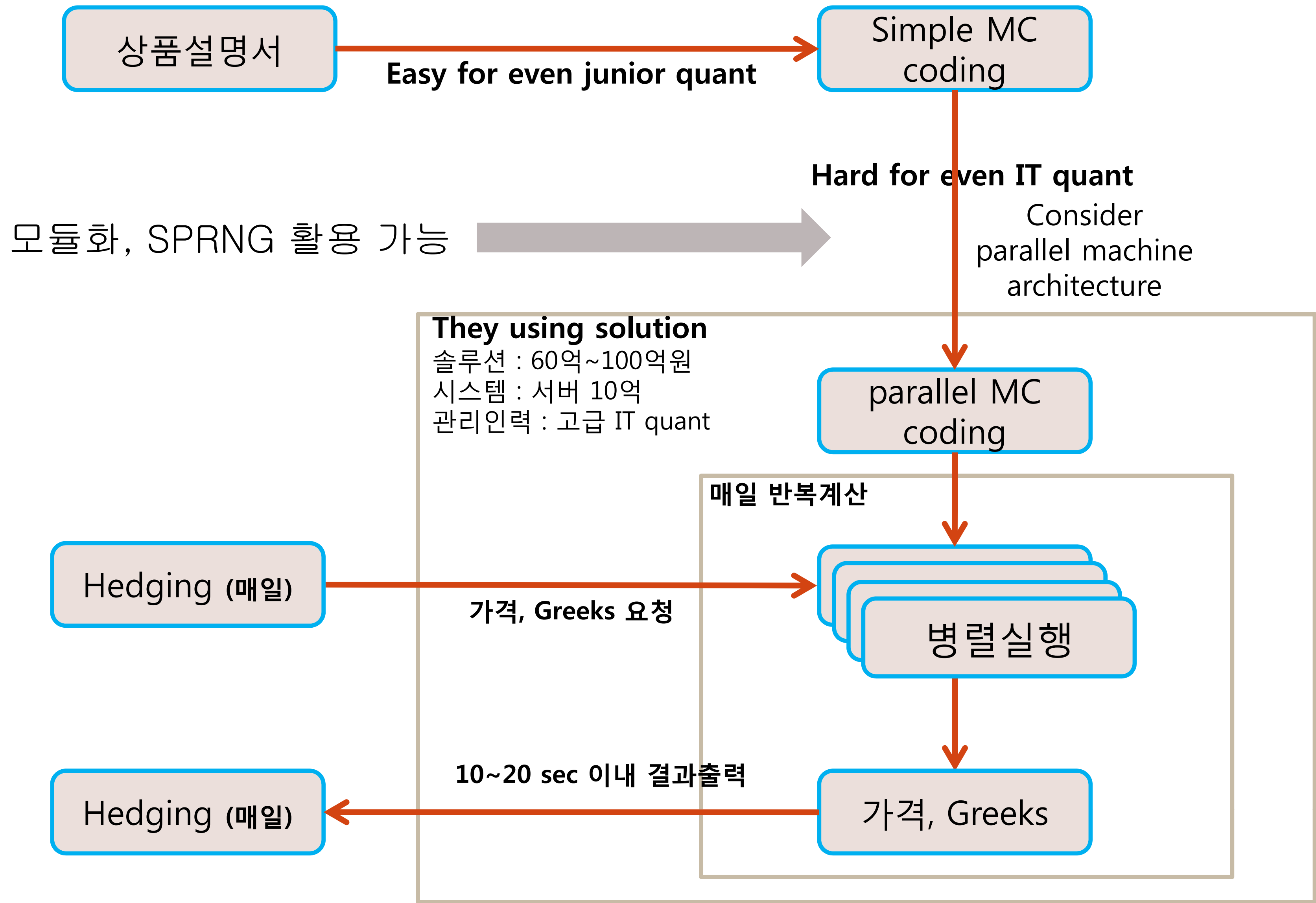
수많은 상품들

Parallel MC

Parallel MC구현의 용이성

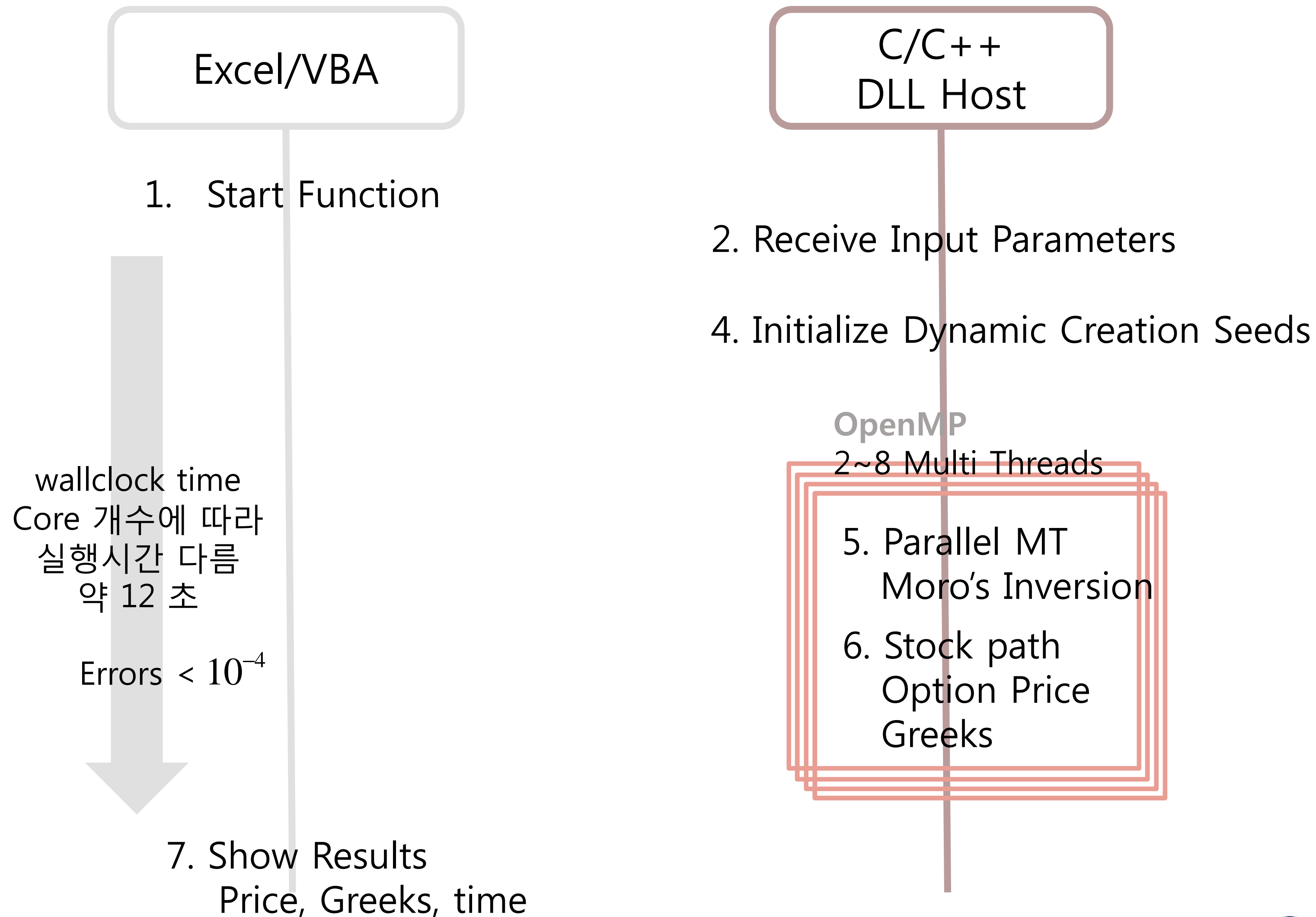


Cluster 기반 pMC



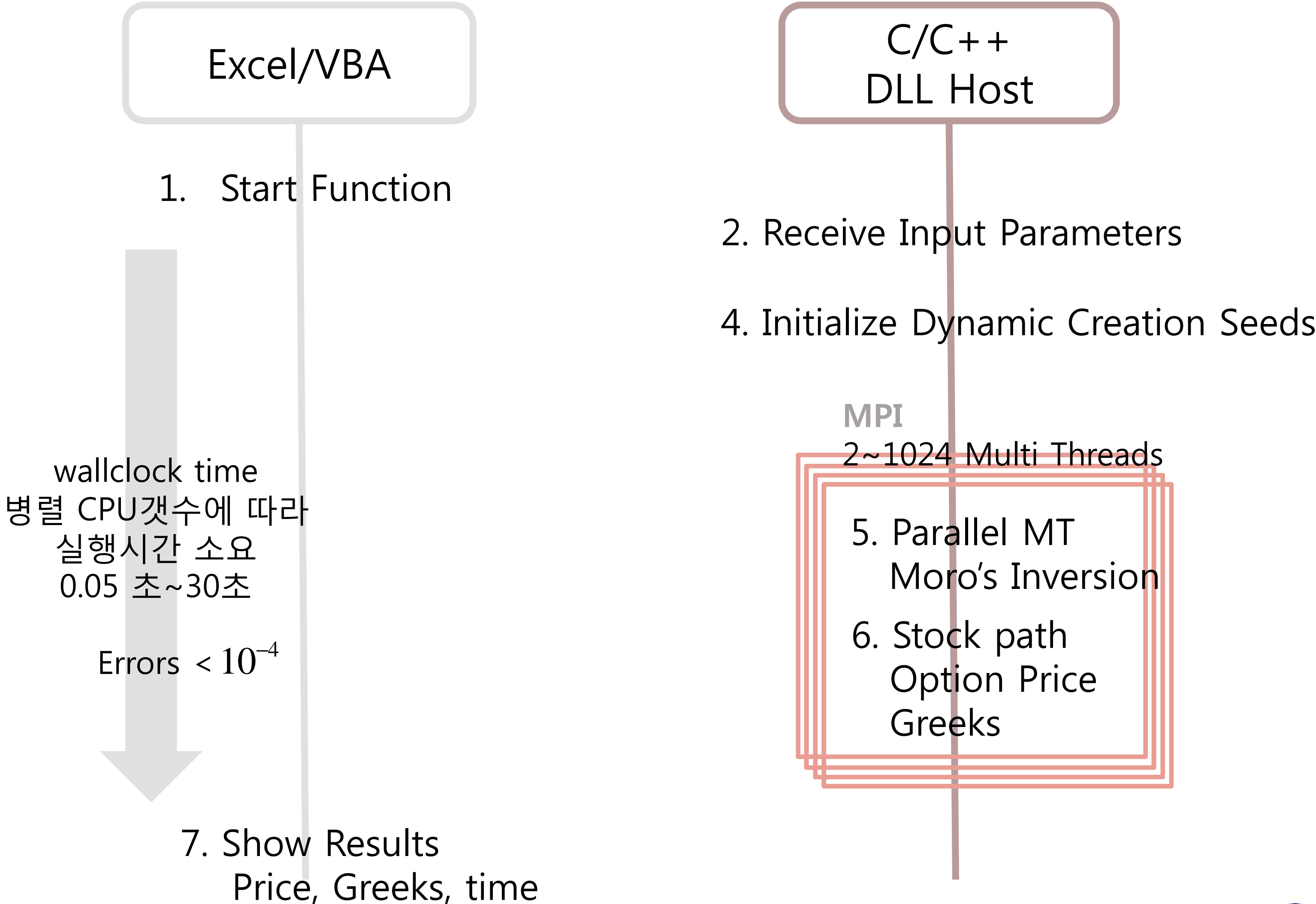
Parallel MC with OpenMP (SMP machine)

10만번 실행



Parallel MC with MPI (Cluster Machine)

10만번 실행



MPI MC examples for Pi calculation

```
#include <math.h>
#include "mpi.h"
#include "mpe.h"
#define CHUNKSIZE      1000
#define INT_MAX 1000000000

/* message tags */
#define REQUEST  1
#define REPLY    2
int main( int argc, char *argv[] )
{
    int iter;
    int in, out, i, iters, max, ix, iy, ranks[1], done, temp;
    double x, y, Pi, error, epsilon;
    int numprocs, myid, server, totalin, totalout, workerid;
    int rands[CHUNKSIZE], request;
    MPI_Comm world, workers;
    MPI_Group world_group, worker_group;
    MPI_Status status;

    MPI_Init(&argc,&argv);
    world  = MPI_COMM_WORLD;
    MPI_Comm_size(world,&numprocs);
    MPI_Comm_rank(world,&myid);
    server = numprocs-1;                /* last proc is server */
    if (myid == 0)
        sscanf( argv[1], "%lf", &epsilon );
    MPI_Bcast( &epsilon, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD );
    MPI_Comm_group( world, &world_group );
    ranks[0] = server;
    MPI_Group_excl( world_group, 1, ranks, &worker_group );
    MPI_Comm_create( world, worker_group, &workers );
    MPI_Group_free(&worker_group);
    if (myid == server) {               /* I am the rand server */
                        do {
                            MPI_Recv(&request, 1, MPI_INT, MPI_ANY_SOURCE, REQUEST,
                                         world, &status);
                            if (request) {
                                    for (i = 0; i < CHUNKSIZE; ) {
                                            rands[i] = random();
                                                    if (rands[i] <= INT_MAX) i++;
                                    }
                                    MPI_Send(rands, CHUNKSIZE, MPI_INT,
                      status.MPI_SOURCE, REPLY, world);
                            }
                        }
                        while( request>0 );
    }
else {                                                             /* I am a worker process */
        request = 1;
                        done = in = out = 0;
                        max  = INT_MAX;         /* max int, for normalization */
        MPI_Send( &request, 1, MPI_INT, server, REQUEST, world );
        MPI_Comm_rank( workers, &workerid );
                        iter = 0;
                        while (!done) {
                            iter++;
                            request = 1;
                            MPI_Recv( rands, CHUNKSIZE, MPI_INT, server, REPLY,
                                         world, &status );
                            for (i=0; i<CHUNKSIZE-1; ) {
                                x = (((double) rands[i++])/max) * 2 - 1;
                                y = (((double) rands[i++])/max) * 2 - 1;
                                 if (x*x + y*y < 1.0)
                                            in++;
                                 else
                                            out++;
                            }
                            MPI_Allreduce(&in, &totalin, 1, MPI_INT, MPI_SUM,
                                                    workers);
                            MPI_Allreduce(&out, &totalout, 1, MPI_INT, MPI_SUM,
                                                    workers);
                            Pi = (4.0*totalin)/(totalin + totalout);
                            error = fabs( Pi-3.141592653589793238462643);
                            done = (error < epsilon || (totalin+totalout) > 1000000);
                            request = (done) ? 0 : 1;
                            if (myid == 0) {
                                        printf( "₩rpi = %23.20f", Pi );
                                        MPI_Send( &request, 1, MPI_INT, server, REQUEST,
                                                    world );
                            }
                            else {
                                        if (request)
                                            MPI_Send(&request, 1, MPI_INT, server, REQUEST,
,
                                                      world);
                            }
                        }
                        MPI_Comm_free(&workers);
    }

    if (myid == 0) {
        printf( "₩npoints: %d₩nin: %d, out: %d, <ret> to exit₩n",
                     totalin+totalout, totalin, totalout );
                getchar();
    }
    MPI_Finalize();
```

연세대학교
YONSEI UNIVERSITY
금융퀀트연구센터

대안적 system 구축

테스트서버/워크스테이션
CentOS 5/Windows
Dualcore PC with PCI-e slot
1개의 8600GT 이상의 graphic card

계산서버
RHEL 5
1U Quad core 서버 with 2 PCI-e slot
1U Nvidia Tesla S870 4 GPU system

개발용 client
네트웍 접속 가능한 notebook도 가능

구축비용 비교적 저렴

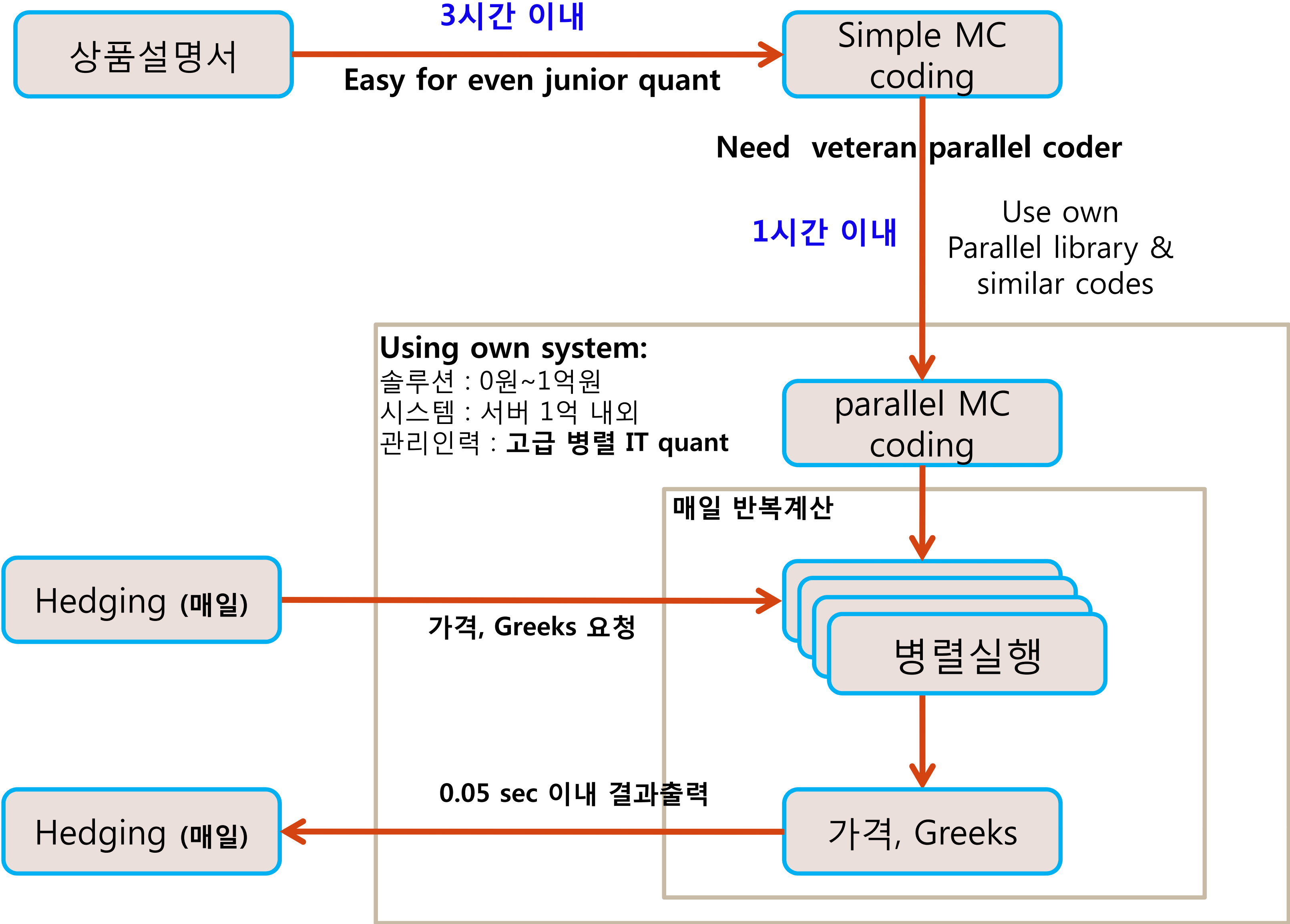
테스트서버/워크스테이션
CentOS 5/Windows
Dualcore PC with 1 PCI-e slot
1개의 CS e620 가속보드

계산서버 1
RHEL 5
4U Quad core 서버 with 2 PCI-e slot
4-6 CS e620 가속보드 슬롯에 장착

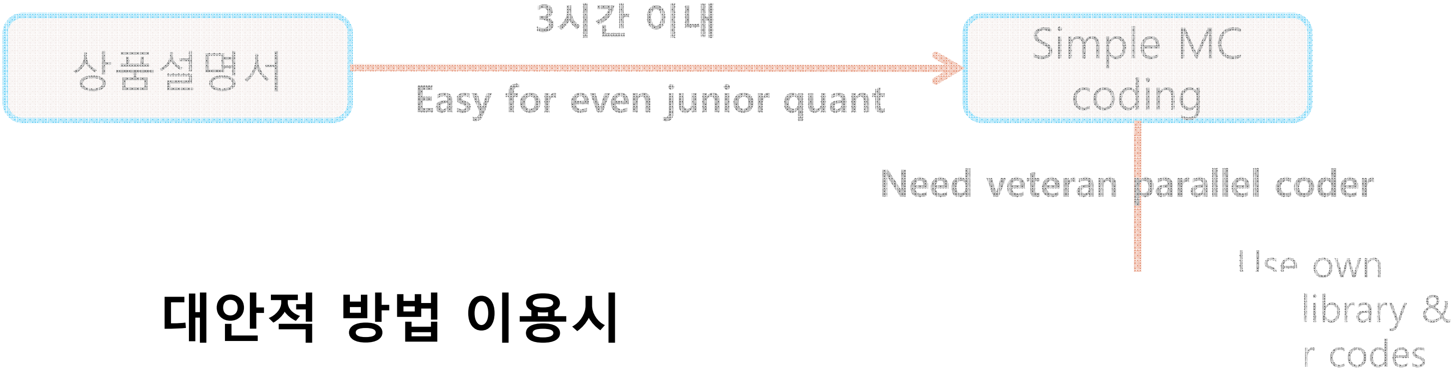
계산서버 2
RHEL 5
4U Quad core 서버 with 2 PCI-e slot
1U CATs 12개의 CS e620 가속보드 장착

개발용 client
네트웍 접속 가능한 notebook도 가능

Parallel Monte Carlo Simulation in Finance



Parallel Monte Carlo Simulation in Finance



대안적 방법 이용시

장점 :
- 서버 도입비용 절감
- 계산속도 향상 (Massively pMC 가능)

단점 :
개발인력 부족
→ 연세금융퀀트연구센터

기존 코드의 재활용 가능성

Parallel MC

기 구축된 Template, pMC library 이용시
- 상품설명서 확인후 single 알고리즘 개발하면 병렬화 완료
- 병렬코딩 시간을 획기적으로 단축

→모듈 설계시 하드웨어, 알고리즘, 확장성을 모두 고려해야함
- quant, IT quant 영역을 넘어섬
- 현재 모듈화 작업중 (open source 예정) pSMT19937 활용
- 금융실무자측과의 communication 필요

Parallel FDM & FEM

자신이 사용하는 2star CN FDM, FEM 코드의 linear solver 부분을 Matrix형태(Ax=b)로 구현하였다면 약간의 코드수정(pLU, pCG 사용)만으로 병렬 가속화 가능 (연구 예정)

병렬화 이전의 matrix generation이 더욱 복잡함

Parallel RNGs

병렬 Pseudo Random Number Generator

병렬 Pseudo Random Number Generation에 대한 연구가 많이 진행되었음.

Cluster구축시 rand()함수가 아닌 Library 활용가능

무료라이브러리

SPRNG

- MPI기반 병렬 RNG가 구현된 Library

-인터넷에 소스코드까지 무료로 공개되어있음

상용 라이브러리

IMKL – (Intel Math Kernel Library) – LCG, MWC, MT19937 등 지원

IMSL, 등

Split Method

```
for (i=0; i<N-1; i++ )
```

```
for (i=istart; i<iend; i++ )
```

CPU ID 0 CPU ID 1 CPU ID 2

Multiseed Method

for (i=0; i<N-1; i++)

```
srand48p(12345+(1+cpuID)*6789);
```

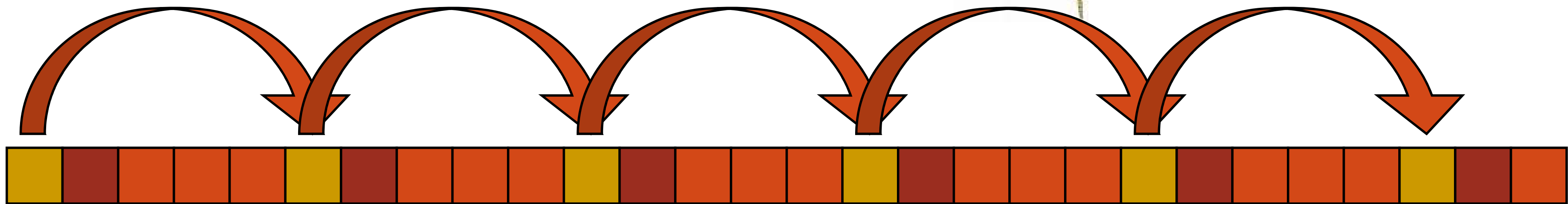
for (i=0; i<CHUNKSIZE-1;)

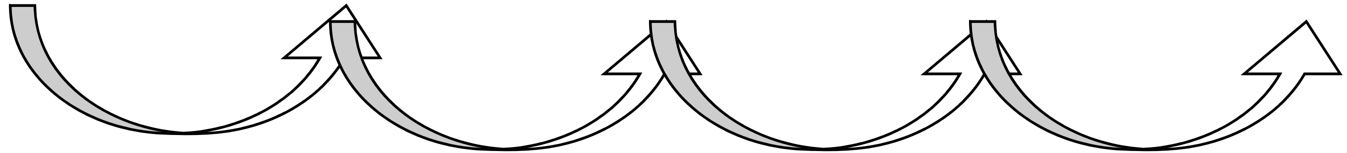
Leap Frog Method

for (i=0; i<N-1; i++)

for (i=istart; i<N-CHUNKSIZE-1; CHUNKSIZE)

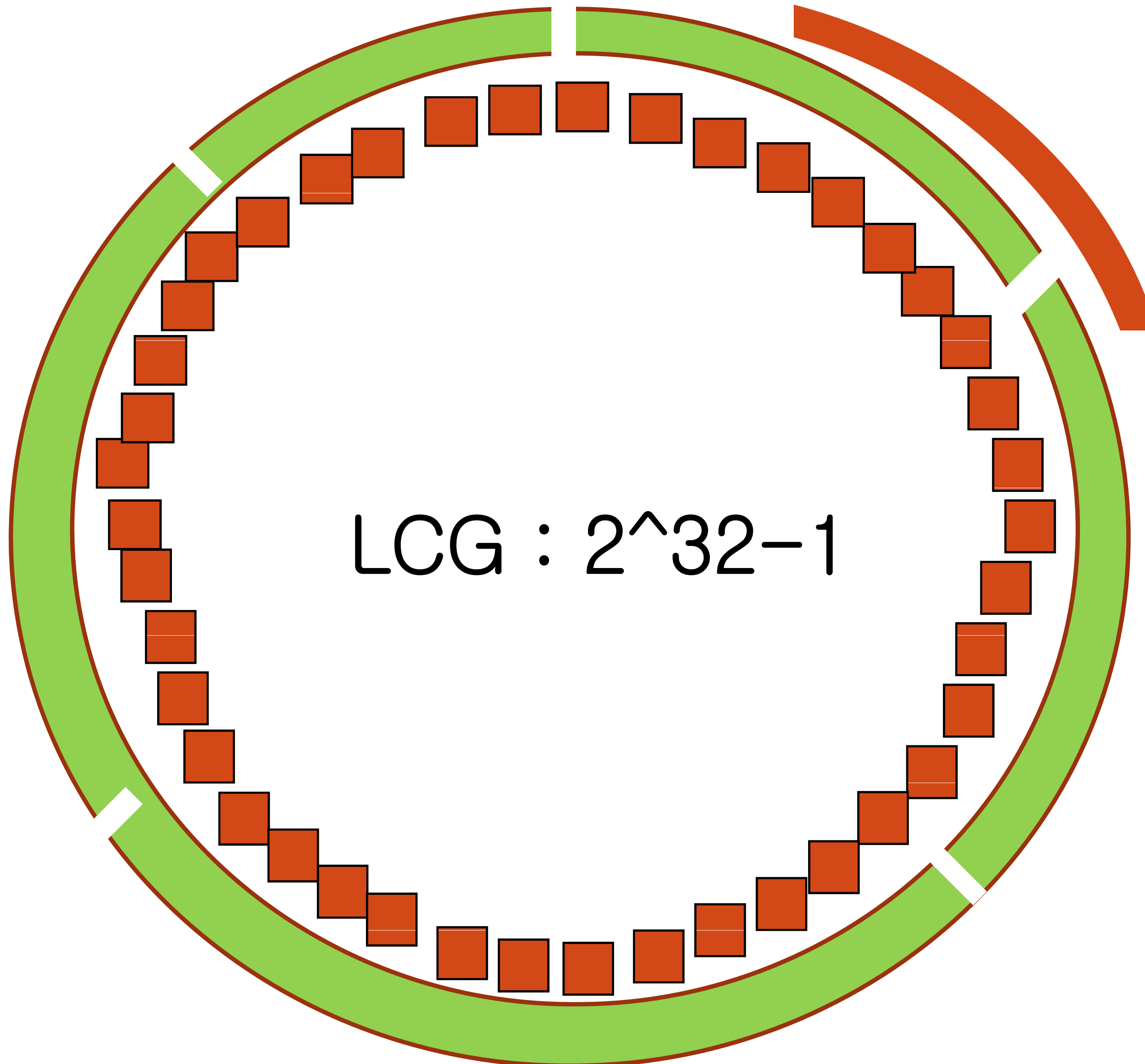
CPU ID 0





CPU ID 1

Problems in Parallel MC



Suitability for Parallel Method

	Multiseed	Leapfrog	Splitting	D.C	J.Ah.
LCG	OK	OK	OK	X	-
RAND48	OK	OK	OK	X	-
MCG	OK	OK	OK	X	-
Lagged Pibonachi	OK	OK	OK	X	-
MWC	OK	OK	OK	X	-
MT19937	OK	X	X	OK	OK
Well RNG	OK	?	?		OK

Dynamic Creation

for (i=0; i<N-1; i++)

Complex dividing for non-overlapping cycle

for (i=0; i<CHUNKSIZE-1;)

CPU ID 0

CPU ID 1

CPU ID 2

Jump-Ahead Method

for (i=0; i<N-1; i++)

Divide sequence with GF(2) polynomial

for (i=0; i<CHUNKSIZE-1;)

CPU ID 0

CPU ID 1

CPU ID 2

병렬 MT19937

Single version : Mutsuo Saito and Makoto Matsumoto 최초 제시

알고리즘 차원에서 624개의 병렬 random sequence 지원

Dynamic Creation 활용시 massively parallel random sequence 생성가능

MT19937 알고리즘을 이해하고 소스를 분석해야 병렬 MC를 제대로 구현할 수 있음., MT19937의 경우 소스코드(89줄)가 공개되었는데, 분석을 안함

진입장벽이 존재

학생들 : Excel Rand()함수를 쓰면 되지~~

Quant : 바쁜 업무와 MC의 천시

MC 이론은 특별한거 없고, 랜덤넘버는 그냥 라이브러리를 쓰면 되지~~

Linear Solver (LU 등)는 C/C++로 직접 구현하면서 RNG는 그냥 갖다 쓰는 경우가 많음.

고급 RNG 이론의 어려움 (TGRSF알고리즘 등은 GF(2) 등 정수론 기반으로 이론 전개)

RNG 쪽의 Bible인 the art of computer programming은 assamber 언어로 작성되었음

MT코드의 난해함 (XOR, 이진 shift 연산 사용)

Algorithm for MT19937

Step 0. $\mathbf{u} \leftarrow \underbrace{1\cdots1}_{w-r}\underbrace{0\cdots0}_{r}$;(bitmask for upper $w-r$ bits)

$\mathbf{ll} \leftarrow \underbrace{0\cdots0}_{w-r}\underbrace{1\cdots1}_{r}$;(bitmask for lower r bits)

$\mathbf{a} \leftarrow a_{w-1}a_{w-2}\cdots a_1a_0$;(the last row of the matrix A)

Step 1. $i \leftarrow 0$

$\mathbf{x}[0], \mathbf{x}[1], \cdots, \mathbf{x}[n-1] \leftarrow$ "any non-zero initial values"

Step 2. $\mathbf{y} \leftarrow (\mathbf{x}[i] \text{ AND } \mathbf{u}) \text{ OR } (\mathbf{x}[(i+1) \bmod n] \text{ AND } \mathbf{ll})$;(computing $(\mathbf{x}_i^u|\mathbf{x}_{i+1}^l)$)

Step 3. $\mathbf{x}[i] \leftarrow \mathbf{x}[(i+m) \bmod n] \text{ XOR } (\mathbf{y} >> 1)$

$$\text{XOR} \begin{cases} 0 & \text{if the least significant bit of } \mathbf{y} = 0 \\ \mathbf{a} & \text{if the least significant bit of } \mathbf{y} = 1 \end{cases}$$;(multiplying A)

Step 4. ;(calculate $\mathbf{x}[i]T$)

$\mathbf{y} \leftarrow \mathbf{x}[i]$

$\mathbf{y} \leftarrow \mathbf{y} \text{ XOR } (\mathbf{y} >> u)$;(shiftright $\mathbf{y}$ by u bits and add to $\mathbf{y}$)

$\mathbf{y} \leftarrow \mathbf{y} \text{ XOR } ((\mathbf{y} << s) \text{ AND } \mathbf{b})$

$\mathbf{y} \leftarrow \mathbf{y} \text{ XOR } ((\mathbf{y} << t) \text{ AND } \mathbf{c})$

$\mathbf{y} \leftarrow \mathbf{y} \text{ XOR } (\mathbf{y} >> l)$

output $\mathbf{y}$

Step 5. $i \leftarrow (i+1) \bmod n$

Step 6. Goto Step 2.

Full Source Code for MT19937

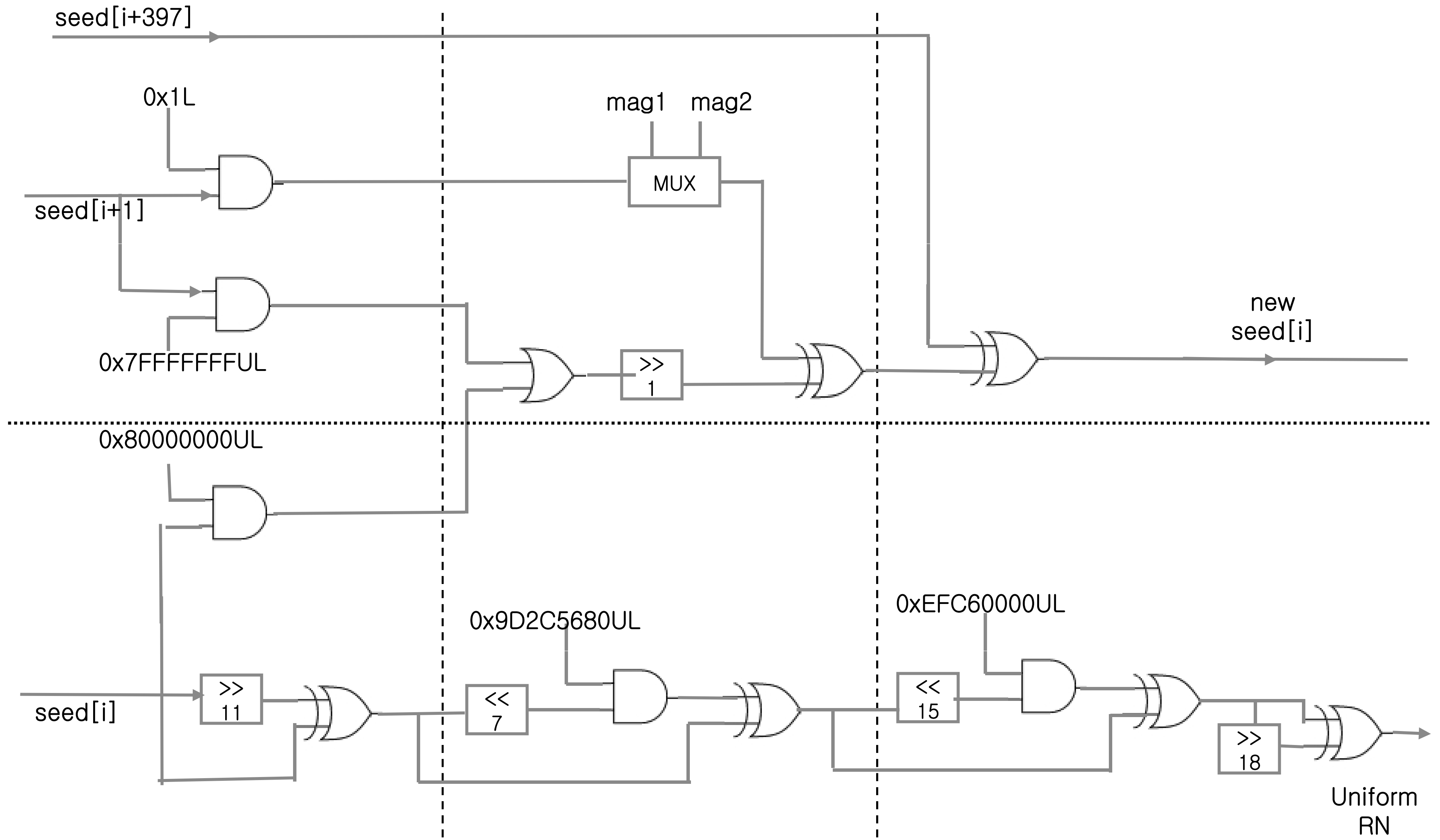
```
#include <stdio.h>
#define N 624
#define M 397
#define MATRIX_A 0x9908b0dfUL
#define UPPER_MASK 0x80000000UL
#define LOWER_MASK 0x7fffffffUL

static unsigned long mt[N];
static int mti=N+1;

/* initializes mt[N] with a seed */
void init_genrand(unsigned long s)
{
    mt[0]= s & 0xffffffffUL;
    for (mti=1; mti<N; mti++) {
        mt[mti] =
                        (1812433253UL * (mt[mti-1] ^ (mt[mti-1] >> 30)) +
mti);
    }
}
```

```
void init_by_array(unsigned long init_key[], int key_length)
{
    int i, j, k;
    init_genrand(19650218UL);
    i=1; j=0;
    k = (N>key_length ? N : key_length);
    for (; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1664525UL))
          + init_key[j] + j; /* non linear */
        mt[i] &= 0xffffffffUL;
        i++; j++;
        if (i>=N) { mt[0] = mt[N-1]; i=1; }
        if (j>=key_length) j=0;
    }
    for (k=N-1; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1566083941UL))
          - i; /* non linear */
        mt[i] &= 0xffffffffUL;
        i++;
        if (i>=N) { mt[0] = mt[N-1]; i=1; }
    }

    mt[0] = 0x80000000UL;
}
```

```
unsigned long genrand_int32(void)
{
    unsigned long y;
    static unsigned long mag01[2]={0x0UL, MATRIX_A};
    /* mag01[x] = x * MATRIX_A  for x=0,1 */

    if (mti >= N) {
        int kk;
        if (mti == N+1)
            init_genrand(5489UL);
        for (kk=0;kk<N-M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        for (;kk<N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1UL];
        mti = 0;
    }
    y = mt[mti++];
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);
    return y;
}
```

```
long genrand_int31(void){
    return (long)(genrand_int32()>>1);
}
/* generates a random number on [0,1]-real-interval */
double genrand_real1(void) {
    return genrand_int32()*(1.0/4294967295.0);
    /* divided by 2^32-1 */
}
/* generates a random number on [0,1)-real-interval */
double genrand_real2(void) {
    return genrand_int32()*(1.0/4294967296.0);
    /* divided by 2^32 */
}
/* generates a random number on (0,1)-real-interval */
double genrand_real3(void) {
    return (((double)genrand_int32()) +
0.5)*(1.0/4294967296.0);
    /* divided by 2^32 */
}
```

SMP19937



연세대학교 YONSEI UNIVERSITY 금융퀀트연구센터

MT19937 알고리즘 설명

Step1 : Seed 생성
최초 624개의 32bit random 정수를 생성 - 대부분 Knuth가 제시한 방법사용

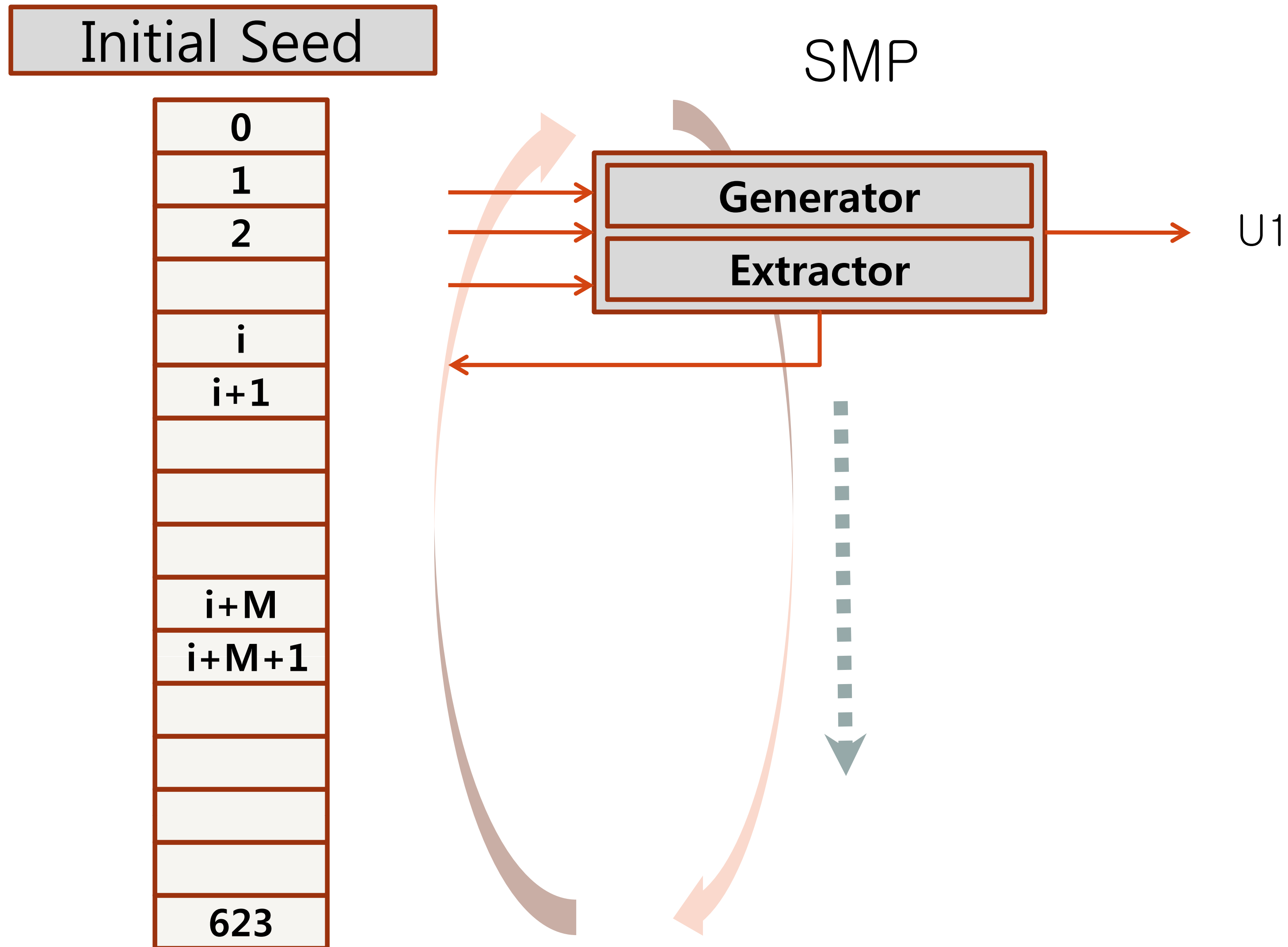
Step2 : XOR연산
0,1번째 정수를 받아와서 앞부분, 뒷부분, 397번째 정수를 받아와서 XOR 연산
자세한 연산의 자세한 내용은 앞의 그림 참조

Step3 : XOR 연산결과 저장
XOR 연산결과를 0번째 정수에 저장(update)

Step4 : XOR연산에 의해 수행된 Uniform Random Number를 출력함

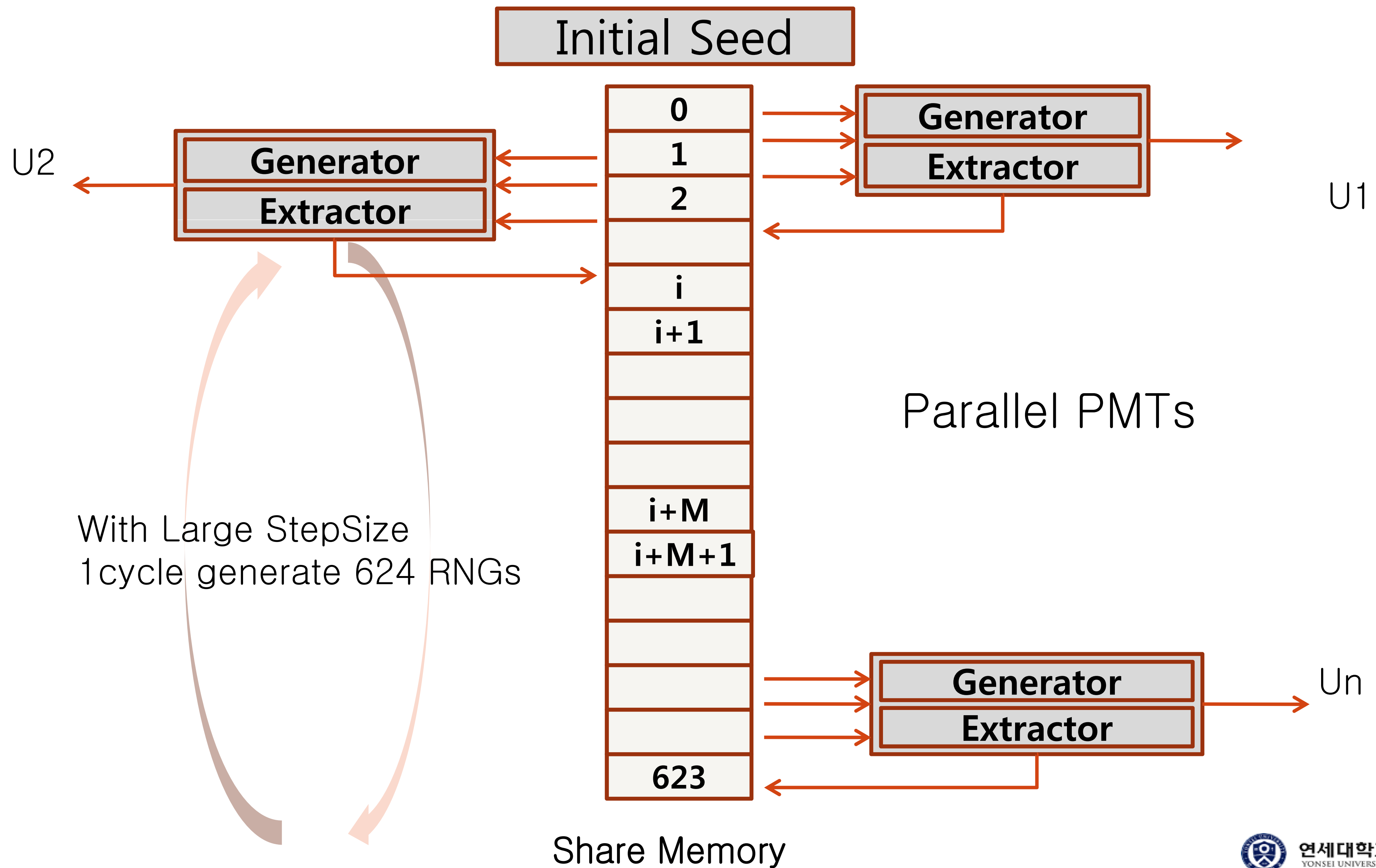
Step2-4 반복
1,2, 398번째 정수를 이용, XOR이후 1번째 정수를 update

sMT architecture of MT19937



연세대학교
YONSEI UNIVERSITY
금융퀀트연구센터

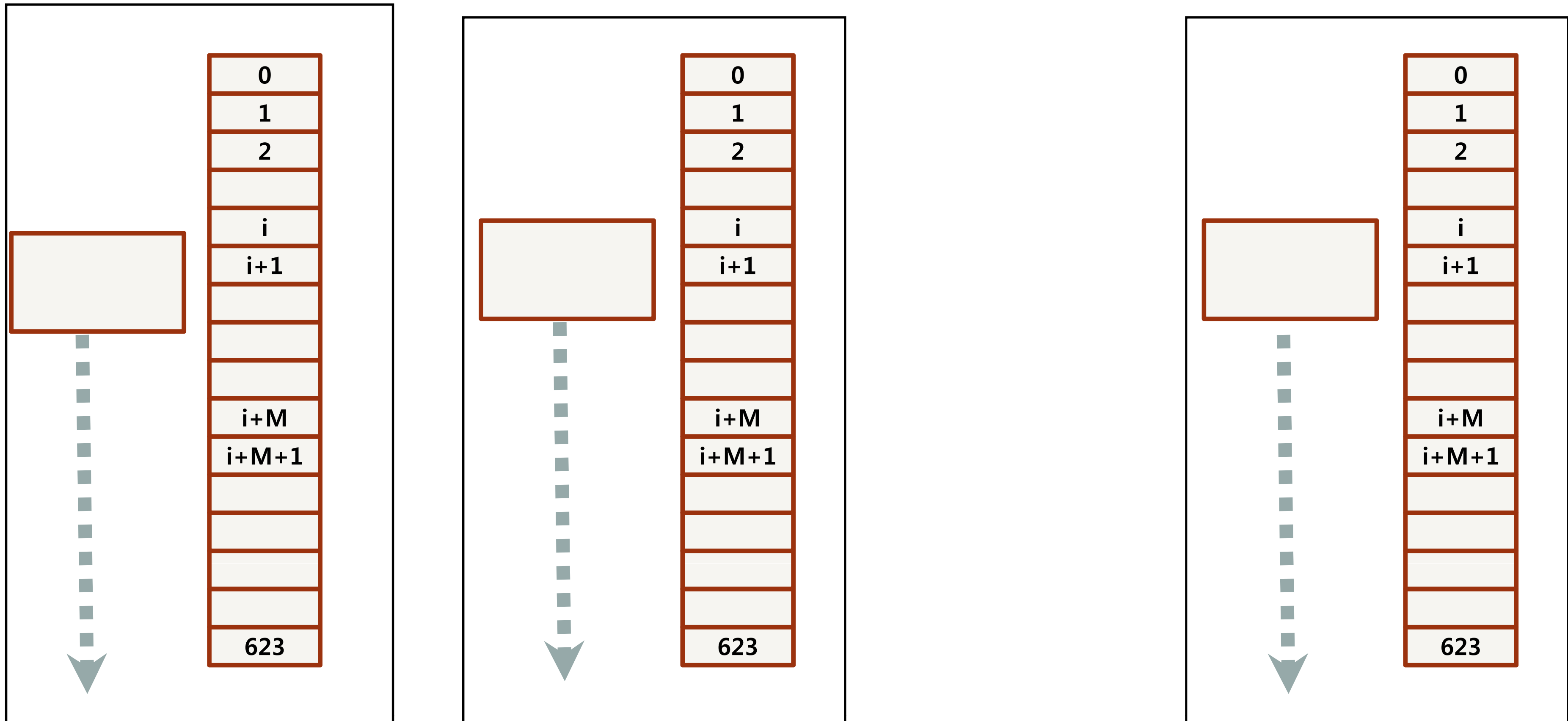
Shared pMT architecture of MT19937



Parallel sMP MT19937

Each PEs execute SMP independently

We will use this model



Part4 CUDA technology

CUDAp workstation in 연세대학교 수학과

Fedora Linux + Intel Compiler + CUDA SDK

도입비용 : 총 400만원(2008년), Tesla C870 3개 장착 : 이론상 1.5Tflops in SP

비교 - 서울대 수퍼컴퓨터 3호기 : 27억원(2000년) 5.6Tflops in DP

CUDA Technology

Nvidia Graphic Cards (8800GTX, Tesla C870)

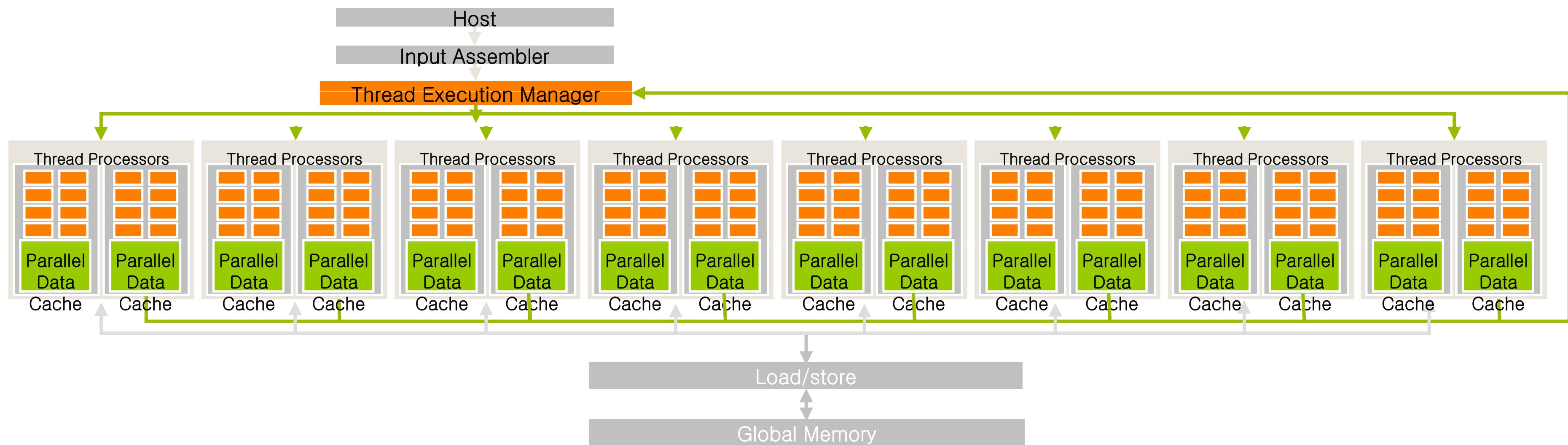
H/W
128 stream processors : 500Gflops in SP
Each unit share Global Memory 1.5GB
Memory Bandwidth : 72 GB/s

S/W
Support C/C++
NVCC compiler support CUDA SDK

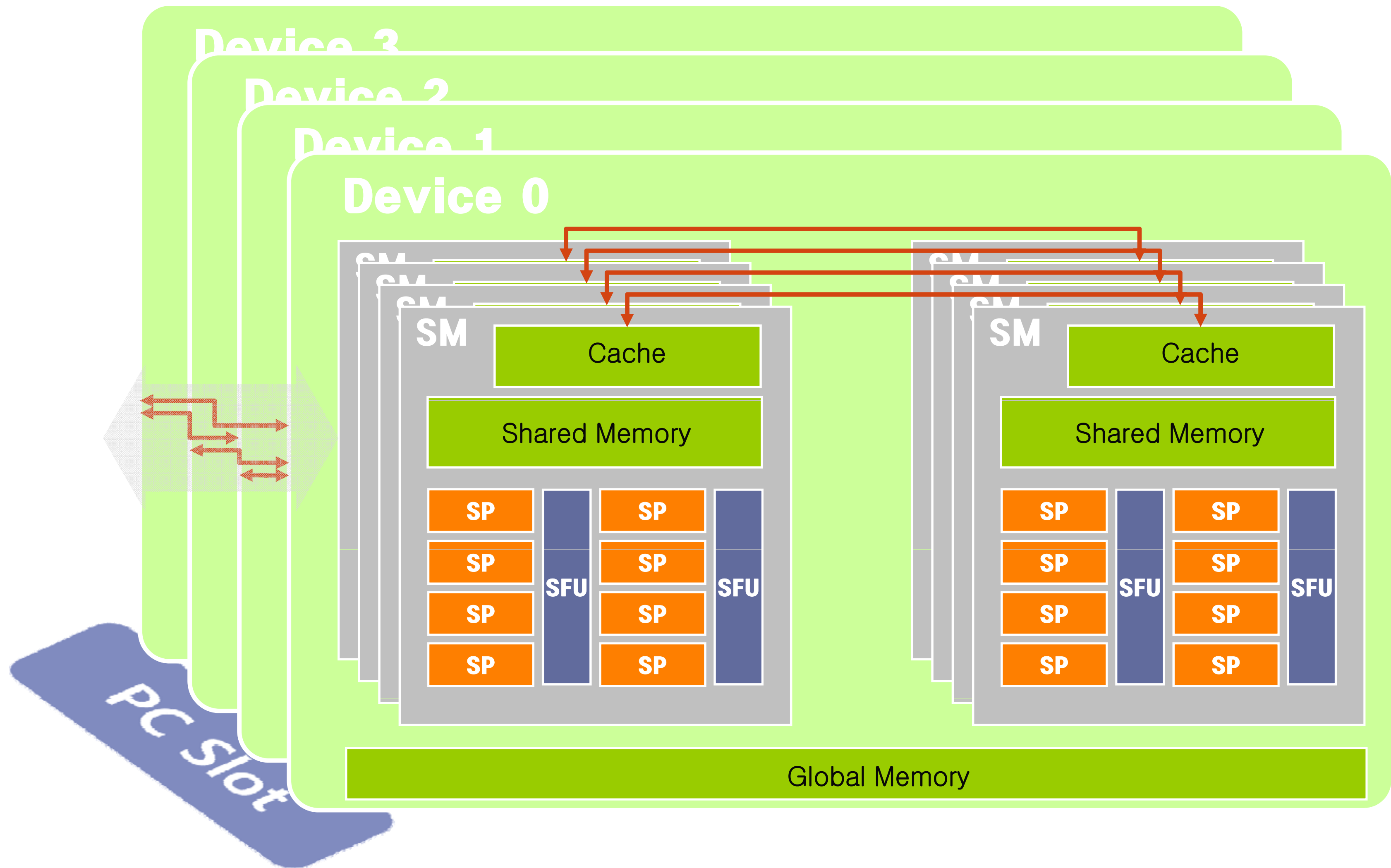
CUDA does not support Double Precision.
It is critical for accuracy in round-off errors in FDM, FEM
Monte Carlo Simulation, it is not critical rather than FDM, FEM.
But in financial market industry require double precision

G80 Device in Tesla Inner

- Processors – execute computing threads
- Thread Execution Manager issues threads
- 128 Thread Processors grouped into 16 Multiprocessors (SMs)
- Parallel Data Cache (Shared Memory) enables thread cooperation

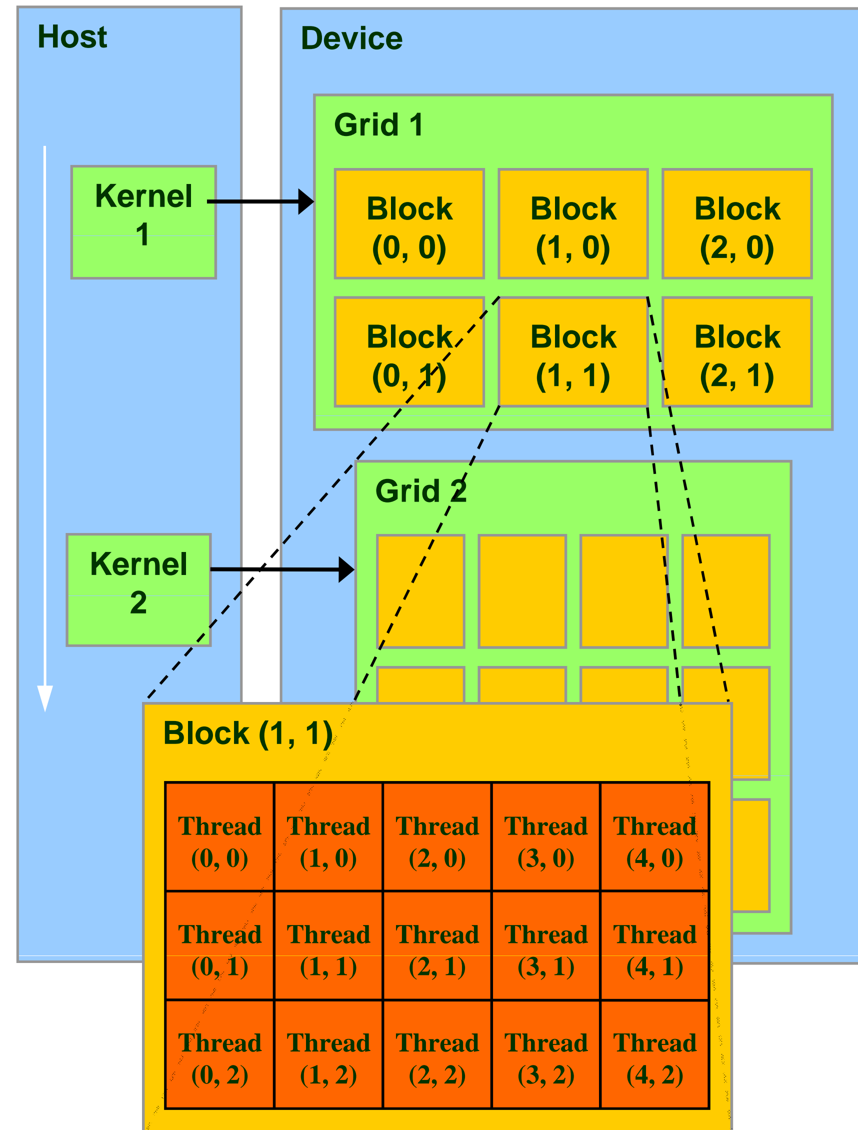


G80 Device In Tesla H/W

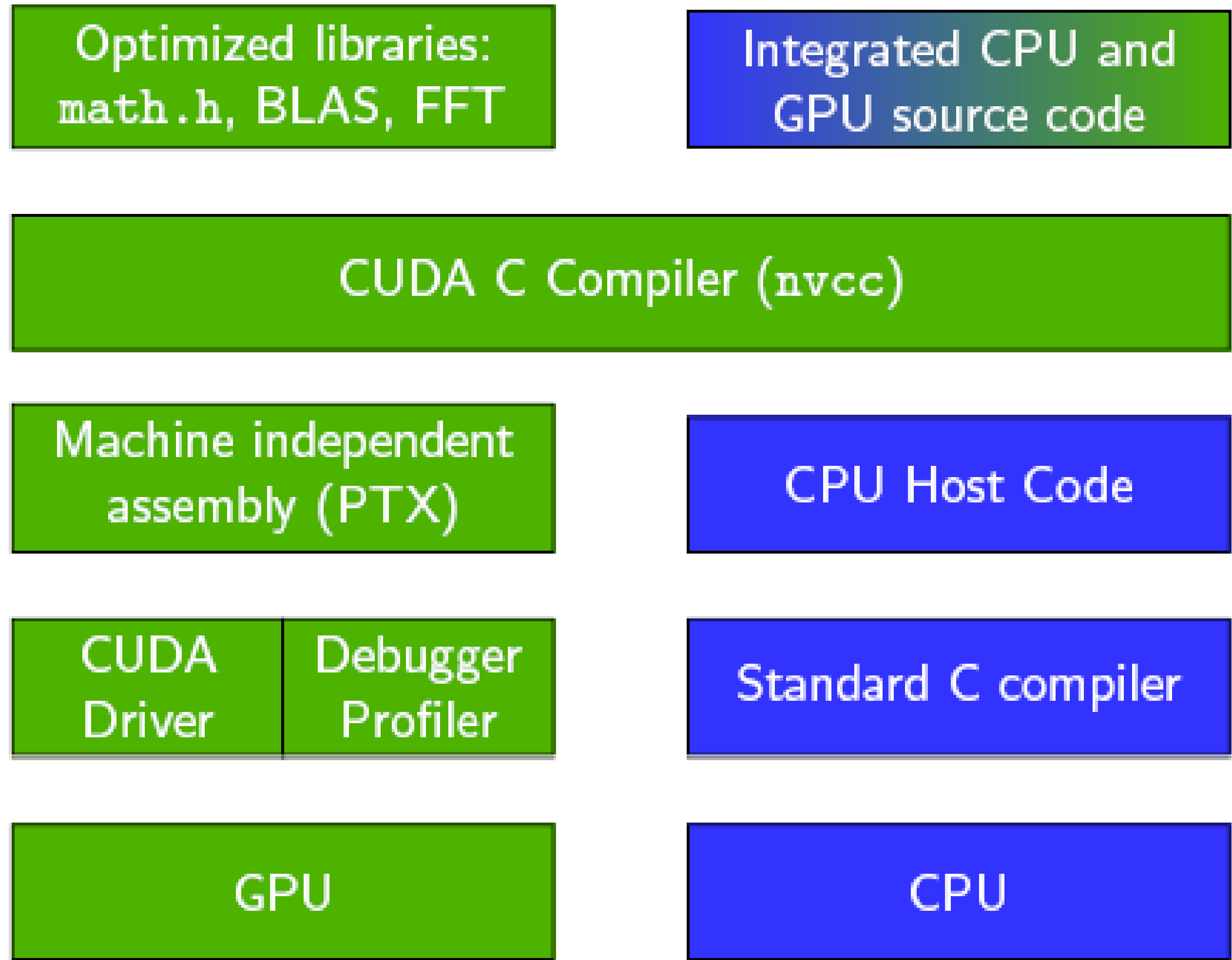


SDK architecture

Threads blocks for SMs
SM launch Waps of threads



CUDA compiler



연세대학교
YONSEI UNIVERSITY
금융퀀트연구센터

CUDA syntax

```
Function_name <<<a, b, c>>>(variables )

CuMalloc(A,size)
CuMemcpy(A,B,method)
__global__ function (variable) { }
__device__  function (variable) { }
__share__ variable;
__ global__ variable;
```

CUDA algorithm example

single

```
void add_matrix
( float* a, float* b, float* c, int N ) {
   int index;
   for ( int i = 0; i < N; ++i )
      for ( int j = 0; j < N; ++j ) {
           index = i + j*N;
          c[index] = a[index] + b[index];
      }
   }

int main() {
   add_matrix( a, b, c, N );
}
```

CUDA

```
__global__ add_matrix
                ( float* a, float* b, float* c, int N ) {
       int i = blockIdx.x * blockDim.x + threadIdx.x;
       int j = blockIdx.y * blockDim.y + threadIdx.y;
       int index = i + j*N;

       if ( i < N && j < N )
       c[index] = a[index] + b[index];
      }

int main() {
   dim3 dimBlock( blocksize, blocksize );
   dim3 dimGrid( N/dimBlock.x, N/dimBlock.y );
   add_matrix<<<dimGrid, dimBlock>>>( a, b, c, N );
}
```

Single Precision vs. Double Precision

Single Precision : 32bit
Error : 24 mantisa
10^-7

double Precision : 64bit
Error : 24 mantisa
10^-15

Round-off errors in single precision (example in Knuth)
(11111113. **[+]** - 11111111.) **[+]** 7.5111111 = 2.0000000 **[+]** 7.5111111 = 9.5111111.
11111113. **[+]** (- 11111111. **[+]** 7.5111111) = 11111113. **[+]** - 111111103. = 10.000000.

Financial Monte Carlo simulation use math function of sqrt, ln, exp, sin, cos
or numerical approximation for transformation and cdf
Random number generator use divide

In this situation , with 10^-7 rounding
we will generate same r.v. in different values and lose the accuracy

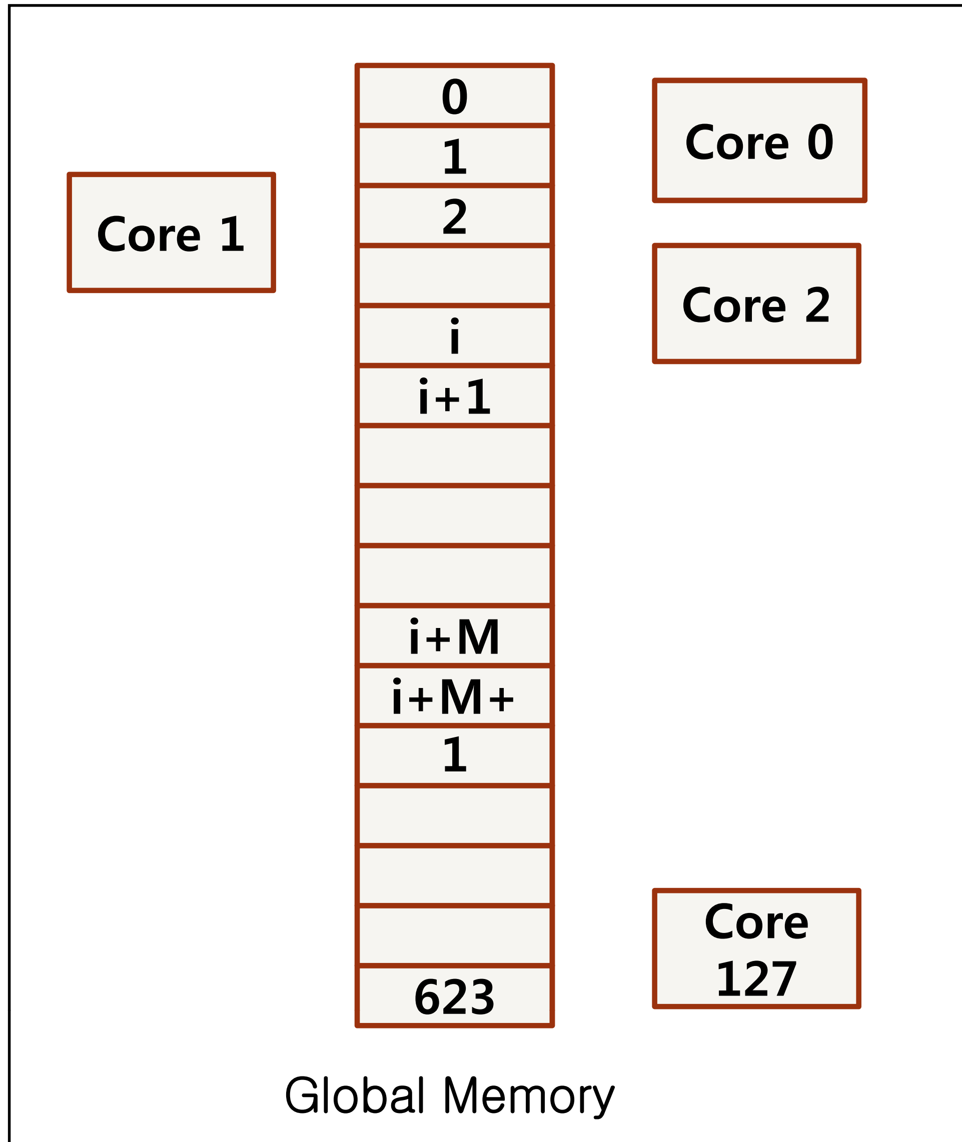
But, in Financial Industry,
they allow 1BP in Hedge Vol : that means they allow 10^-4 errors in price.
If we can control the errors 10^-5 in single precision in MC. (not FDM, FEM)

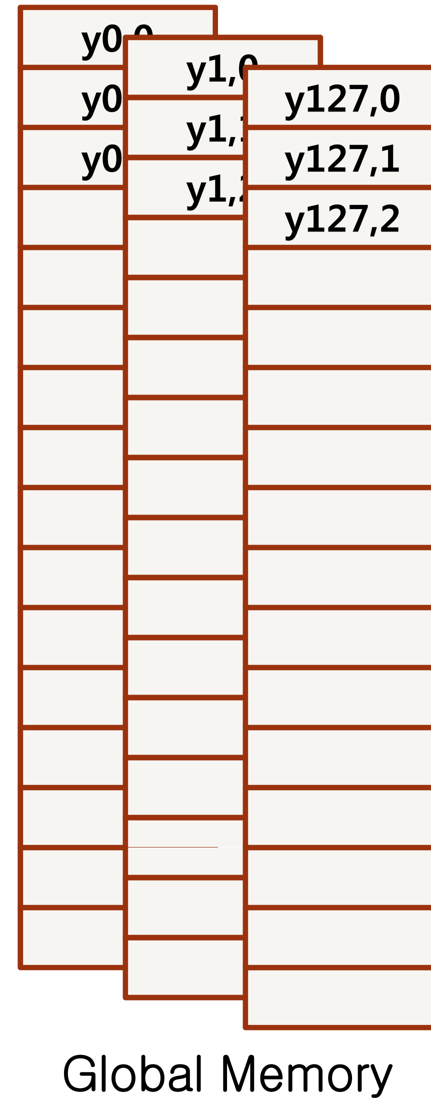
Implementation for pMC on CUDA

Massively pMC (H/W)

- 다양한 문제를 고민해줘야함
- RNG 자체의 분석 필요 (SMT19937 분석)
- 비 금융적 문제임
- Parallel H/W에 관련된 문제가 대부분
- Knuth, the art of computer programming

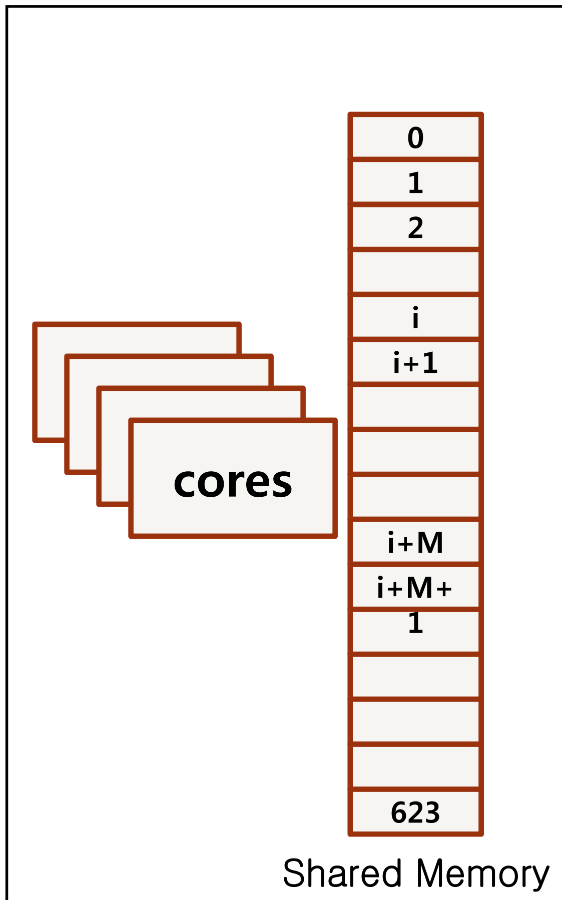
CUDA 방법I -- PMT19937

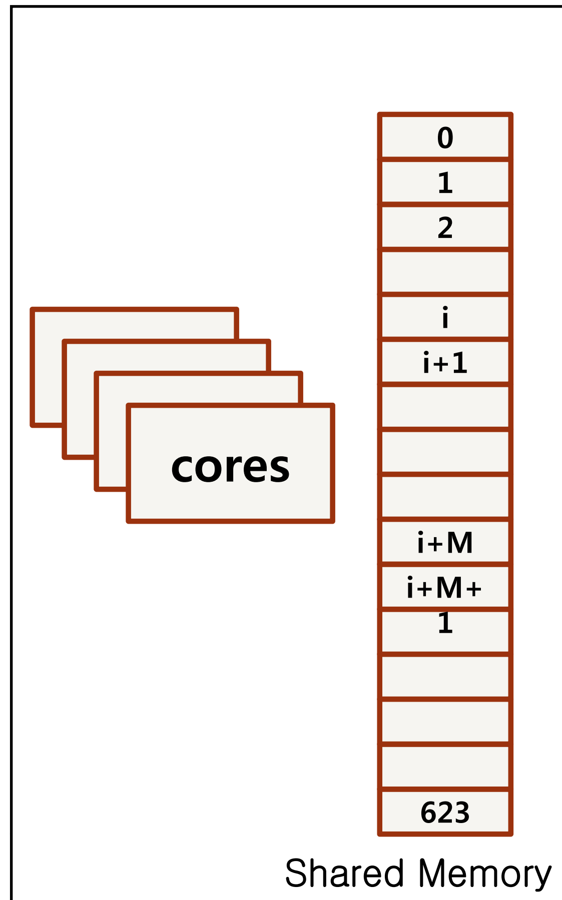


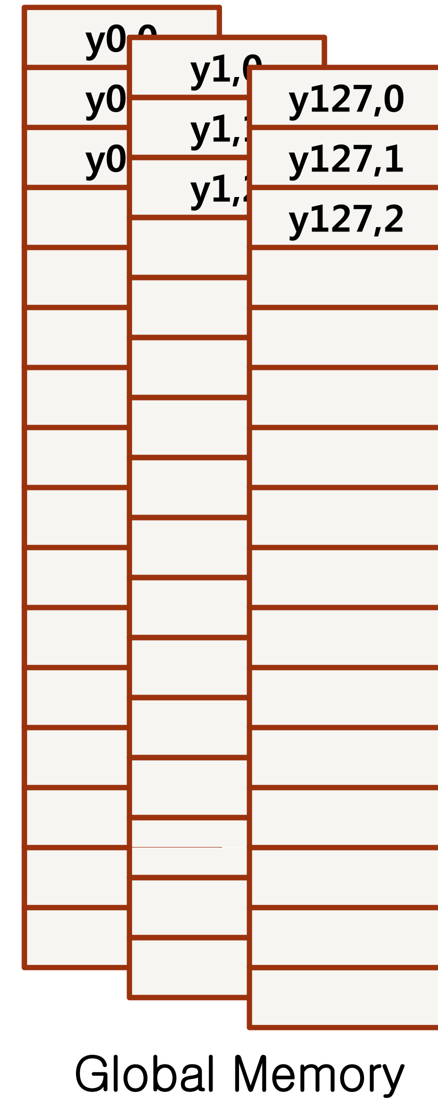


CUDA방법 II -- PMT19937

각 16 Block은 Multi Seed 이용

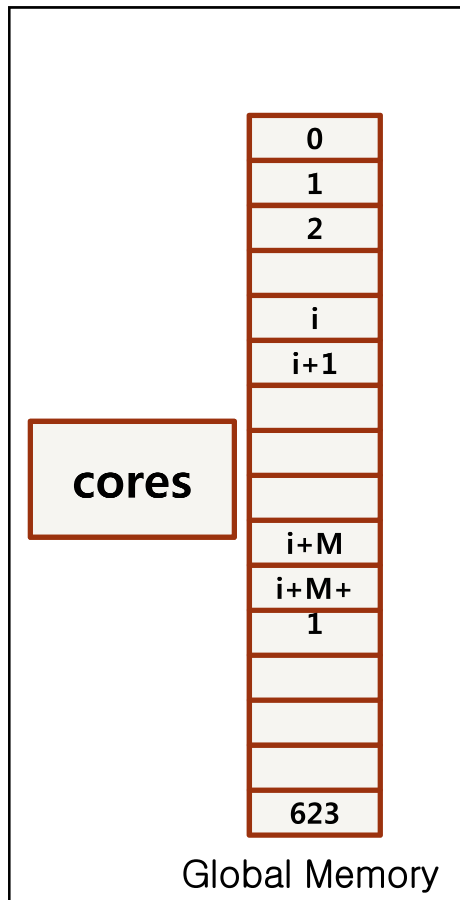


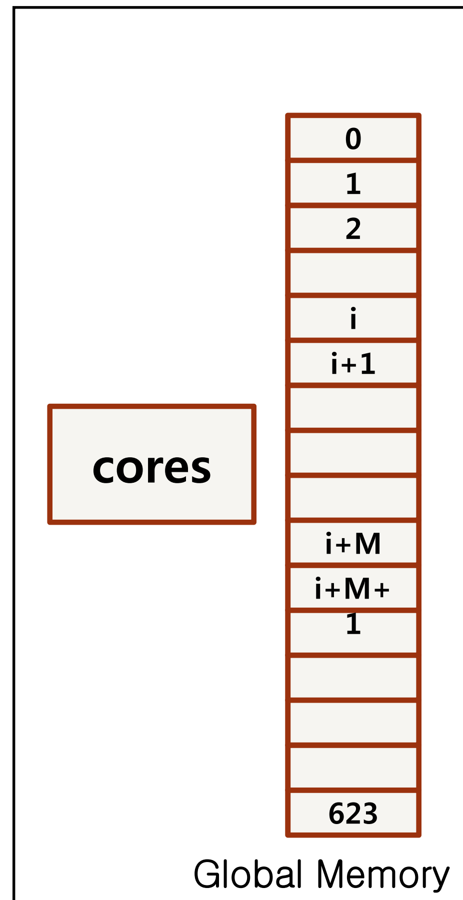




CUDA 방법III -- parallel SMT

각 128 SPs가 Multi Seed 이용





New approach for massively parallel SPMD

1. **Job split method**

   Simulate Option Price in Outer Loop.

   each simulation, call RNG

   it has big overheads of function call

   Wall Clock Time >> Simulation Time / (# of cores)

2. **Pre-generation method**

   generate full # of U R.V. for simulation

   Transfer U R.V. to N R.V.

   each simulation, they use R.V.s which were already generated.

   limit of memory bandwidth : 60GB/s : needs 0.025 sec for 1.5GB

   limit of memory size : 2억개의 R.V.s /each device

   Wall Clock Time = Simulation Time / (# of cores) + Memcpy Time

3. **Avoid Round-off Errors**

$$\mathrm{E}[e^{-r\tau}[S_T - K]^+] \approx \frac{1}{M}\sum_{k}^{M}\left(\frac{1}{N_k}\sum_{i}^{N_k} e^{-r\tau}[S_{T_i} - K]^+\right)$$

New approach for massively parallel SPMD

4. **Mixed Precision Method (float-float approach)**
   emulate double precision with two single precision operation
   DP is 4~10 times slower than SP.
   DP RNGs much more slower than SP RNGs
      - same period but : 24bit or 48bit mantisa in SP, 54bit mantisa in DP
      - heavy function call of ln, sin, cos in Box-muller

5. **Mixed Precision Method (GPU-CPU approach)**
   use 64bit CPU FP in DP computation
   It need comunicating between Host and Device.
   DP in CPU is slower than DP in GPU
   Tt is easy to imply

6. **Fixed-Point Method**
   convert float point data to integer data.
   ALU is faster than FP
   Fixed Point Method is useful in LCG, but is not suitable in Box-muller

New approach for massively parallel SPMD

7. **Inter-correlation between parallel cores**
   split wthout careful considering,
   the inter-correlation between parallel cores occure.
   To avoid this, we need to parallelize with considering algorithms of RNG
   for different RNGs, we apply different method of parallelizations

8. **Overlapping in splitted random sequence**
   Pseudo RNGs has limited period such as 2^31, etc.
   In massively parallel generating,
   each stream of random number may overlapp with each others.
   To avoid this, use long period RNGs or dynamic creation method.

Benchmark for RNGs with CUDA

Rand48() + Memcpy(DtoH)

CPU : 최신 AMD 페놈 2.5Ghz
GPU : Tesla C870

Size: 122880000 random numbers

CPU rand48

time : 2260.000000 ms

Samples per second: 5.2512821E+07

GPU rand48

time : 20.000000 ms

Samples per second: 6.144000E+09

Copying random GPU data to CPU

time : 540.000000 ms

속도향상 : 최대 113 X

Thread에 따라 80X 정도

Benchmark for RNGs with CUDA

MT19937 + Box-Muller + Memcpy

Size: 122880000 random numbers

CPU MT19937 RN generation

time: 1460.000000 ms

Samples per second: 8.4164384E+07

원래 MT알고리즘이 LCG보다 속도 빠름
LCG 2260ms vs. MT 1460ms

그런데 가속화 성능은 LCG가 더좋음
LCG 113X vs. MT 24X
LCG 20ms vs. 48ms

왜?

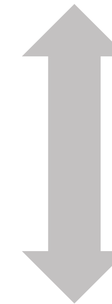
Size: 122880000 random numbers

GPU MT19937 RN generation

time : 199.955994 ms

Samples per second: 6.145352E+08

속도향상 : 7.3 X



Sample 개수 조정에따라 따라 속도향상됨 Why?

Generated samples : 100007936

RandomGPU() time : 47.960999 ms

Samples per second: 2.085193E+09

속도향상 : 24.7 X

Occupacy for thread & Performence

3.) GPU Occupancy Data is displayed here and in the graphs:	
Active Threads per Multiprocessor	760
Active Warps per Multiprocessor	24
Active Thread Blocks per Multiprocessor	4
Occupancy of each Multiprocessor	100%
Maximum Simultaneous Blocks per GPU	64

(Note: This assumes there are at least this many blocks)

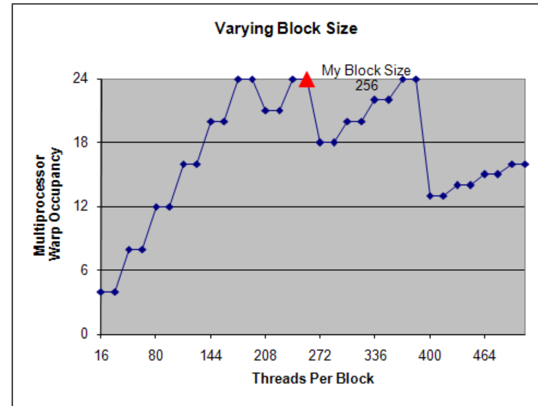
Physical Limits for GPU:	G80
Multiprocessors per GPU	16
Threads / Warp	32
Warps / Multiprocessor	24
Threads / Multiprocessor	768
Thread Blocks / Multiprocessor	8
Total # of 32-bit registers / Multiprocessor	8192
Shared Memory / Multiprocessor (bytes)	16384

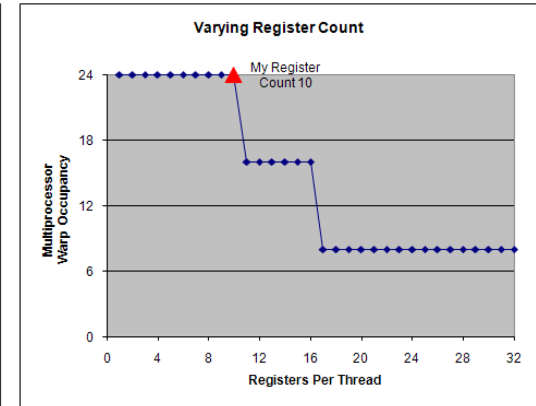
Allocation Per Thread Block	
Warps	6
Registers	1920
Shared Memory	4096

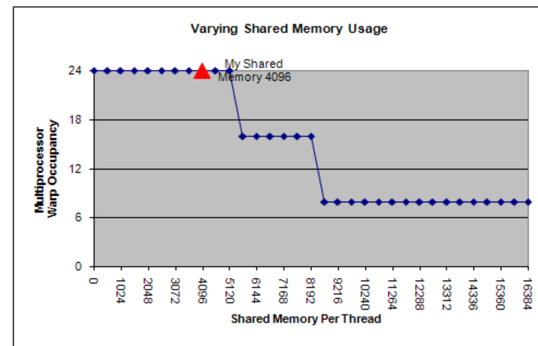
These data are used in computing the occupancy data in blue

Maximum Thread Blocks Per Multiprocessor	Blocks
Limited by Max Warps / Multiprocessor	4
Limited by Registers / Multiprocessor	4
Limited by Shared Memory / Multiprocessor	4

Thread Block Limit Per Multiprocessor is the minimum of these 3







<<<16,256>>>

Performance from Operation Cycles & latency

Operation
type conversion : 4 cycle

float add/mul/mad : 4 cycle
floating div : 36 cycle

integer add, bit operation : 4 cycle
integer comare, min, max : 4 cycle
integer mul : 16 cycle

Memory
4 cycles for issuing a read from global memory
4 cycles for issuing a write to shared memory

400~600 cycle for reading a float from global memory
→Need thread schedule strategy

Benchmark for European Vanilla

GPU MT19937 RN generation

time : 199.955994 ms

Samples per second: 6.145352E+08

GPU BoxMuller transformation

time : 98.652000 ms

Samples per second : 1.245591E+09

Copying random GPU data to CPU

time : 517.033020 ms

GPU Monte-Carlo simulation...

Total GPU time : 56.948002 ms

Options per second : 1.755988E+01

Total : 872.589 ms = 0.8 sec

Non-full path generation

Implementation on ELS pricing and hedging

상품설명서 : **삼성증권 제1909호 주식연계증권**

3년 만기

2 star : POSCO 일반주, S-Oil

12 chance : 디지털 옵션

Double Barrier

Down barrier : 장중체크 (둘중 하나라도 하락한계)

Up barrier : 매일 종가 체크(두개다 상승한계)

Step-down : 없음

지급 : 조기상환시 +2 영업일

vba03-els.xls [호환 모드] - Microsoft Excel

2Stock StepDown(6Change) & Double Barrier ELS 가격평가

이자율 0.05 부스트래핑이 필요함.

편의상 주어졌다고 생각함

		Code	Vol	Correlation	배당률
자산1	POSCO	005490	0.25	0.5	0.01
자산2	SK	003600	0.3		0.01

SK
SK네트웍스
SK텔레콤
SKC
S-Oil
강원랜드
고려아연
국민은행

발행일
발행기준일
만기일
만기평가일

자산1현가
자산2현가

상환조건

금리	연이율 7.0%
Upbarrier	기준가의 120%
Upbarrier	2.0 배지급
knock-In 조건	기준가의 60%
Knock-In금리	이표 4.5%
만기이표금리	3년수익률 42%
Up만기이표금리	3년수익률 84%

stepdown

1	기준가의 90%
2	기준가의 90%
3	기준가의 85%
4	기준가의 85%
5	기준가의 80%
6	기준가의 80%

중도상환	날짜	행사가격	연금리	이표	Upbarr	연금리	이표
1	2008-08-01	90%	7.0%	7.0%	120%	14.0%	14.0%
2	2009-02-01	90%	7.0%	14.0%	120%	14.0%	28.0%
3	2009-08-01	85%	7.0%	21.0%	120%	14.0%	42.0%
4	2010-02-01	85%	7.0%	28.0%	120%	14.0%	56.0%
5	2010-08-01	80%	7.0%	35.0%	120%	14.0%	70.0%
6	2011-02-01	80%	7.0%	42.0%	120%	14.0%	84.0%

30%
25%
20%
15%
10%
5%
80%
85%
90%
110%
36개월
30개월
24개월
18개월
12개월
6개월

ELS Tmp MC Sheet3

준비 100%

연세대학교 YONSEI UNIVERSITY 금융퀀트연구센터

Excel 화면

삼성증권 1909회 ELS (95형)

face value 10000

	POSCO	S-Oil	비율
최초기준가	539000	68700	
행사가격	512050	65265	95%
상승한계	549780	70074	102%
하락한계	323400	41220	60%
현재가	539000	68700	
발행일	2008-05-13	기간	
만기	2011-05-09	3년	
현재날짜	2008-05-14		

Volatility 추정

	POSCO	S-Oil
Hvol	0.410461777	0.373543706
EWMA	0.385880533	0.448304863

Correaltion 추정

0.328060489

할인율 0.06

조기상환일		days	networkdays		지급 : 년12%	처음부터	처음부터	현재부터
1차	2008-08-08	64	61		0.03	60	366	360
2차	2008-11-07	129	123		0.06	122	738	732
3차	2009-02-09	195	185		0.09	184	1110	1104
4차	2009-05-08	259	247	1년	0.12	246	1482	1476
5차	2009-08-07	324	310		0.15	309	1860	1854
6차	2009-11-09	390	375		0.18	374	2250	2244
7차	2010-02-09	456	439		0.21	438	2634	2628
8차	2010-05-07	519	498	2년	0.24	497	2988	2982
9차	2010-08-09	585	563		0.27	562	3378	3372
10차	2010-11-09	651	626		0.30	625	3756	3750
11차	2011-02-09	717	689		0.33	688	4134	4128
만기	2011-05-09	780	749	3년		748	4494	4488

Excel/VBA Pseudo code for ELS (single)

Parameter 입력

Loop 10만번~ 50만번
- XT simulation 실행
  - Boxmuller,
  - RNG(SMT19937)
- ELS pricing Routine
  - IF문을 통해 조기사환, 만기조건 고려
  - 조기상환시 시뮬레이션 정지

평균값을 구함

결과 출력

CPU single 실행속도

- 엑 셀 : 50만번 시뮬레이션(Pricing만) : 약 4~5분(조기 상환되었는데도 느림)

  → Excel이 실행되는 CPU는 성능이 안좋음.

시스템 : Intel Core2Duo 2.0Ghz dual core (but single using)

(장중 모니터링 6회 실시) 즉 50만번 시뮬레이션 시

총 사용 RNG 개수 :500000*250*3*6*2 = 4500000000 개 : 45*10^8

조기상환이 되도 계속 RNG를 생성시킨 경우(비교를 위해)

C MT19937 이용시	최적화			최적화중
RNG	CPU 53초,	GPU1 7.3초,	GPU2 2.1초	
BM 변환	CPU 20초,	GPU1 3.6초,	GPU2 1.1초	
Copy	CPU 0초 ,	GPU1 18.3초,	GPU2 6.2초	
MC	CPU 18초,	GPU1 1.2초,	GPU2 0.4초	
총	90초,	30초(2.5배)	9.8초(9배)	7.1초(12배)

C언어 : AMD 페놈 2.5Ghz GPU : Tesla C870

CPU single 실행속도

대부분 1-2 회차에서 조기상환됨

총 사용 RNG 개수 : 500000*120*6*2 = 약 720000000개

전체 계산량의 1/6로 감소 : 이론상 CPU에서 15 sec 가능??

불가능함.

원인 : Function call에 의한 오버헤드 (IF문 사용)로 인한 속도 증가 어려움

병렬처리시에도 비슷한 문제 발생, 더욱 복잡한 문제들 발생가능성 있음.

작업목표

BenchMark
Single 5분
SMP 서버 사용시 7.5배 속도 향상 (예상)
2노드 Cluster 이용시 15배 속도 향상 (예상)

- Tesla C870을 이용하여 2sec 안에 ELS price 계산 가능?
- 즉, 100배 이상 속도 향상 가능?
- 작업

1. 상품설명서 -> VBA코딩 (날짜 고민, 조기상환 고려, option base 1)
2. VBA 코드 - > C코드 (날짜 fix했음 : 작업의 편의상, option base 0)
3. C 코드 -> CUDA 병렬화
   __kernel__ 함수 코딩 (병렬구조, RNG 특징 고려)

결론부터 말하면…

- Tesla C870을 이용하여 2sec 안에 ELS price 계산 가능?
- 즉, 100배 이상 속도 향상 가능?

C870 하나의 보드 이용시

- 병렬화시 overlapping, cyclic 문제가 발생할 수 있는
- LCG를 이용시
- RNG 생성시 112배 빨라짐
- 전체 프로세스에서는 **80배** 향상됨 (업무적으로 쓰기에는 부적합할 것으로 생각됨.)

- MT19937 이용시,
- 현재 **12배** 정도 가속됨. (최적화 안되었음, 계속적인 연구 중)

- 조기종료를 고려할 경우에도 CPU조기종료 대비 약 12배 정도(약간씩 오차있음) 빠름

VBA Single MC 코딩 (old version)

Break문 사용 안하고 조기상환시 종료

If문 call 횟수 줄이이기 12회

```
For i = 1 To Nsim ' 이제부터는 daily simulation
xt1 = basis_xt1 ' 시초가
xt2 = basis_xt2 '시초가
DB_flag(i) = 100  ' 200이면 hit.
prestrike(i) = 100  '조기상환이 아직 없었음
price(i) = 0
  For K = 1 To KK  ' KK*JJ = 전체 3년간의 daily기간이 됨 계산의 편의상 조기상환일까지 균등분할함
  UB_flag(K) = 100 ' 초기값 설정 200이면 hit,
  If prestrike(i) = 100 Then  ' 조기상환 발생시 이후 시뮬레이션 안하기 위한 코드
    For L = 1 To LL ' 각 조기상환일  12*63 = 252*3
      For m = 1 To MM  ' daily 하루 모니터링 횟수를 나타냄
        coNormal1 = Application.NormSInv(Rnd())
        coNormal2 = corr * coNormal1 + Sqr(1 - corr ^ 2) * Application.NormSInv(Rnd())
        'Xt+1  = Xt + r*Xt*dt + vol Xt * sqr(dt) Normal
        xt1 = xt1 + r * xt1 * dt + vol1 * xt1 * Sqr(dt) * coNormal1
        xt2 = xt2 + r * xt2 * dt + vol2 * xt2 * Sqr(dt) * coNormal2
        If (xt1 <= DB_Xt1) Or (xt2 <= DB_Xt2) Then ' DB 장중모니터
          DB_flag(i) = 200 ' 전체 기간에 1개
        Else
        End If
      Next m
        If ((xt1 >= UB_Xt1) And (xt2 >= UB_Xt2)) Then    ' UB 종가모니터(daily)
          UB_flag(K) = 200  '각 만기일마다 1개씩 존재
        End If
    Next L
          '만기 지급시
          If (K = 12) Then
            If (((xt1 >= Strike_Xt1) And (xt2 >= Strike_Xt2)) Or (UB_flag(K) = 200)) Then
              price(i) = discount(K) * Face_value * (1 + 0.03 * K)
              counter1 = counter1 + 1 ' 만기 수익지급
Else
worst_perf = Application.Min((xt1 - basis_xt1) / basis_xt1, (xt2 - basis_xt2) / basis_xt2)
                If (DB_flag(i) = 200) Then
                  price(i) = discount(K) * Face_value * (1 + worst_perf)
                  If Face_value * (1 + worst_perf) < 0 Then  ' 손실 100%
                    price(i) = 0
                  End If
                  counter2 = counter2 + 1 ' 만기 손실발생
                Else
                  price(i) = discount(K) * Face_value '만기 원금만 지급
                  counter3 = counter3 + 1
                End If
            End If
          Else  ' 만기가 아니라면
            If (((xt1 >= Strike_Xt1) And (xt2 >= Strike_Xt2)) Or (UB_flag(K) = 200)) Then

              price(i) = discount(K) * Face_value * (1 + 0.03 * K)  '조기상환 지급액
              counter4 = counter4 + 1   ' 조기상환 횟수 파악용
              prestrike(i) = 200 '  prepayment strike 발생여부 저장
              pretime(i) = K
            End If
          End If

    Else  ' 조기상환 발생시 뒤쪽 시뮬레이션 안함 코드
    counter5 = counter5 + 1
    End If ' 조기상환 발생시 뒤쪽 시뮬레이션 안함 코드
  Next K   ' 여기까지 작동시키면 price 1회 구해졌음
sum1 = sum1 + price(i)
sum2 = sum2 + pretime(i)
Next i
```

VBA Single MC 코딩 (new version)

```
kstart = 1
For K = 1 To KK ' 매일 매일 시간이 흘러가는 것을 고려함
pretime(K) = (totaldaycount - Cells(14 + K, 7)) / totaldaycount
pretime2(K) = Cells(14 + K, 7)
' MsgBox (pretime(K))
rrr(K) = r ' after Bootstrapping, we will change this code
If Cells(14 + K, 7) <= 0 Then  ' <, <= 뭐를 써야 정확할지 체크 필요
discount(K) = 0
kstart = K + 1  ' 현재날짜 고려하여, 조기상환일을 지나쳤으면 생략되도록
'MsgBox (K)
Else
discount(K) − Exp(-1 * pretime(K) * rrr(K))
End If
'MsgBox (discount(K))
upbarrierflag(K) = 100

Next K

'MsgBox (kstart)

Range("j43").Value = Now()

'simulation 시작
Randomize  'seed를 초기화 함
For i = 1 To Nsim ' 이제부터는 daily simulation
   xt1 = basisxt1 ' 시초가
   xt2 = basisxt2 '시초가
   price(i) = 0
   downbarrierflag(i) = 100  ' 200이면 hit.
   upbarrierflag2(i) = 100
   prestrikeflag(i) = 100  '조기상환이 아직 없었음

      For j = 1 To totaldaycount '만기까지 남은 기간 daily
         If prestrikeflag(i) = 100 Then ' 전기에서 조기상환되지 않았다면
' 장중 모니터링 ================================================================
For m = 1 To MM  ' MM은 daily 장중 모니터링 횟수를 나타냄
   coNormal1 = Application.NormSInv(Rnd())
   coNormal2 = corr * coNormal1 + Sqr(1 - corr ^ 2) * Application.NormSInv(Rnd())  ' 추후 MT19937 & BMM으로 바꿈
   'Xt+1  = Xt + r*Xt*dt + vol Xt * sqr(dt) Normal
   xt1 = xt1 + r * xt1 * dt + vol1 * xt1 * Sqr(dt) * coNormal1
   xt2 = xt2 + r * xt2 * dt + vol2 * xt2 * Sqr(dt) * coNormal2
   If (xt1 <= DownbarrierXt1) Or (xt2 <= DownbarrierXt2) Then ' downbarrier 장중모니터
      downbarrierflag(i) = 200 ' 전체 기간에 1개
   Else
   End If
Next m
```

Break문 사용 안하고 조기상환 시 종료

If문 call 횟수는 늘어났음

```
' 각 상환일별 체크 ============================================================
For K = kstart To KK - 1 '각 상환일별 조기상환여부체크 1~11, or 8~11 현재날짜에 따라 kstart값 달라
   If pretime2(K) = j Then
      If ((xt1 >= UpbarrierXt1) And (xt2 >= UpbarrierXt2)) Then    ' upbarrier 종가모니터(daily)
         upbarrierflag(K) = 200  '각번 상환일에 조기상환되었음
         upbarrierflag2(i) = 200 ' 이번 simulation에서  조기상환되었음

      End If
      If (((xt1 >= StrikeXt1) And (xt2 >= StrikeXt2)) Or (upbarrierflag(K) = 200)) Then
         price(i) = discount(K) * Facevalue * (1 + 0.03 * K)  '조기상환 지급액
         counter4 = counter4 + 1   ' 조기상환 횟수 파악용
         prestrikeflag(i) = 200 '  prepayment strike 발생여부 update
         pretimeNum(i) = K ' 정확한 조기상환 회차 저장시킴
      Else
      prestrikeflag(i) = 100
      End If
   End If
Next K
End If '1회때 상환되었는데, 2회째 다시 상환되는 설정을 꺼야함 : 작업완료.
      Next j   ' daily simulation 완료
      '만기 지급시=========================================
If prestrikeflag(i) = 100 Then '11기까지 조기상환이 안이루어졌다면
      If ((xt1 >= UpbarrierXt1) And (xt2 >= UpbarrierXt2)) Then    ' upbarrier 종가모니터(daily)
         upbarrierflag(KK) = 200  '각번 상환일에 조기상환되었음
      End If
   If (((xt1 >= StrikeXt1) And (xt2 >= StrikeXt2)) Or (upbarrierflag(KK) = 200)) Then
      price(i) = discount(KK) * Facevalue * (1 + 0.03 * KK)
      counter1 = counter1 + 1 ' 만기 수익지급
   Else
         worstperf = Application.Min((xt1 - basisxt1) / basisxt1, (xt2 - basisxt2) / basisxt2)
         If (downbarrierflag(i) = 200) Then
            price(i) = discount(KK) * Facevalue * (1 + worstperf)
            If Facevalue * (1 + worstperf) < 0 Then  ' 손실 100% 이상일때는 0원지급
               price(i) = 0
            End If
            counter2 = counter2 + 1 ' 만기 손실발생
         Else
            price(i) = discount(KK) * Facevalue '만기 액면만 지급
            counter3 = counter3 + 1
         End If
   End If
Else
' debug용조기상환이 이루어졌다면
   pretime2flag(i) = 200
   If pretime2flag(i) - prestrikeflag(i) = 0 Then
   pretime3flag(i) = 300
   End If
End If
```

연세대학교
YONSEI UNIVERSITY
금융퀀트연구센터

VBA -> C/C++ 변환

Excel의 networkdays 함수를 직접 구현해주어야함.
(어려운 점 : 실무 경험이 부족하여 1일 이상 소요 아직도 완벽하지 않음)

배열을 사용시 언어의 특징 고려해야함
variable(i) → variable[i]
Option base 1을 option base 0 로 바꿔줘야함

For문, if문의 형식을 맞춰줘야함. 특히 bracket { } 에서 오류 많이 발생
특히 연산자 &&, ||, !, == 오류 주의

대소문자 오류 체크 (C언어는 변수명과 함수명의 대소문자에 민감함)

C언어 DLL을 이용한 excel link도 사용가능

VBA -> C/C++ 변환

엑셀 **DLL** 사용 **- 3**월 세미나에서 **^^**

Public Declare Function myrand Lib "D:\work\Excel\rngdll.dll" () As Long
Public Declare Function mysrand Lib "D:\work\Excel\rngdll.dll" (ByVal seed As Long)
Public Declare Function myurand Lib "D:\work\Excel\rngdll.dll" () As Double

Public Declare Function mysrandMT Lib "D:\work\Excel\mtrng.dll" (seed As Long) As Long
Public Declare Function myrandMT Lib "D:\work\Excel\mtrng.dll" () As Double

Public Declare Function myrandulogit Lib "D:\work\Excel\logitrng.dll" () As Double
Public Declare Function myrandlogit Lib "D:\work\Excel\logitrng.dll" () As Double
Public Declare Function mysrandlogit Lib "D:\work\Excel\logitrng.dll" (seed As Long)

C언어 Single MC 코딩 (old version)

```
for (i = 1; i<= Nsim;i++){  // 이제부터는 daily simulation
    xt1 = basis_xt1; // 시초가
    xt2 = basis_xt2; //시초가
    DB_flag[i] = 100;  // 200이면 hit.
    prestrike[i] = 100;  //조기상환이 아직 없었음
    price[i] = 0;
        for(k = 1 ; k<= KK; k++ ){   // 만기까지 daily기간
            UB_flag[k] = 100; // 초기값 설정 200이면 hit,
            if (prestrike[i] == 100) {   // 조기상환 발생시 이후 시뮬레이션 안함
                for (l = 1; l<= LL; l++){ // 각 조기상환일  12*63 = 252*3
                    for (m = 1; m<= MM; m++){  // daily 하루 모니터링 횟수를 나타냄
                        coNormal1 = boxmuller();
                        coNormal2 = corr * coNormal1 + Sqr(1 - corr ^ 2) * boxmuller();
                       if (xt1 <= DB_Xt1) Or (xt2 <= DB_Xt2) {
                           DB_flag[i] = 200; // DB 장중모니터  // 전체 기간에 1개
                       }
                    } // Next m
                        if ((xt1 >= UB_Xt1) && (xt2 >= UB_Xt2)) {
                            UB_flag[k] = 200;    // UB 종가모니터(daily)
                        }
                        xt1 = xt1 + r * xt1 * dt + vol1 * xt1 * sqrt(dt) * coNormal1;
                        xt2 = xt2 + r * xt2 * dt + vol2 * xt2 * sqrt(dt) * coNormal2;
                } // Next l
                if (k = 12) { //만기 지급시
                    if (((xt1 >= Strike_Xt1) && (xt2 >= Strike_Xt2))
                        || (UB_flag[k] == 200)) {
                        price[i] = discount[k] * Face_value * (1 + 0.03 * k);
                        counter1 = counter1 + 1; // 만기 수익지급
 }
                    else{
worst_perf = min((xt1 - basis_xt1) / basis_xt1, (xt2 - basis_xt2) / basis_xt2);
                        if (DB_flag[i] = 200) {
                            price[i] = discount[k] * Face_value * (1 + worst_perf);
                            if (Face_value * (1 + worst_perf) < 0 ) {// 손실 100%
                                price[i] = 0;
                            }
                             counter2 = counter2 + 1; // 만기 손실발생
                        }
                        else{
                             price[i] = discount[k] * Face_value; //만기 원금만 지급
                             counter3 = counter3 + 1;
                        }
                    }
                }
                else{  // 만기가 아니라면
                    if (((xt1 >= Strike_Xt1) && (xt2 >= Strike_Xt2))
                        || (UB_flag[k] == 200)) {
                        price[i] = discount[k] * Face_value * (1 + 0.03 * k);
                        counter4 = counter4 + 1;   // 조기상환 횟수 파악용
                        prestrike[i] = 200; //  prepayment strike 발생여부 저장
                        pretime[i] = k;
                    }
                }
            }else{// Else   // 조기상환 발생시 뒤쪽 시뮬레이션 안함 코드
                counter5 = counter5 + 1
            } // 조기상환 발생시 뒤쪽 시뮬레이션 안함 코드
        }//Next k  // 여기까지 작동시키면 price 1회 구해졌음
    sum1 = sum1 + price[i];
    sum2 = sum2 + pretime[i];
}//Next i
```

C언어 Single MC 코딩 (New)

Single Code가 완벽해야 병렬화에 유리
병렬화를 하는 과정에서 single 코드 변경시
변수명 등 전체적 구조를 다시 바꿔줘야함

```
// simulation 실시
  for (i = 1; i<= Nsim;i++){  // 이제부터는 daily simulation
     xt1 = basis_xt1; // 시초가
     xt2 = basis_xt2; //시초가
     DB_flag = 100;  // 200이면 hit.
     UB_flag = 100; // 초기값 설정 200이면 hit,
     prestrike_flag = 100;  //조기상환이 아직 없었음
     price = 0;
     for(j = 1 ; j<= totaldyaount; j++ ){   // 만기까지 daily기간
        if (prestrike_flag == 100) {   // 조기상환 발생시 이후 시뮬레이션 안함
           for (m = 1; m<= MM; m++){  // daily 하루 모니터링 횟수를 나타냄
              coNormal1 = myrandom();
              coNormal2 = corr * coNormal1 + Sqrt(1 - corr ^ 2) * myrandom();
              xt1 = xt1 + xt1 *rdt + xt1 * vsqdt1 * coNormal1;
              xt2 = xt2 + xt2 *rdt + xt2 * vsqdt2 * coNormal2;
             if (xt1 <= DB_Xt1) Or (xt2 <= DB_Xt2) {
                DB_flag = 200; // DB 장중모니터  // 전체 기간에 1개
             }
           } // Next m 장중 simulation 완료

           for(k = kstart ; k<= KK-1; k++ ){   // 각 조기상환일 Loop
              if (pretime[k]==j){  //조기상환일이라면  값 받아옴
                 if ((xt1 >= UB_Xt1) && (xt2 >= UB_Xt2)) {
                    UB_flag = 200;    // UB 종가모니터(daily)
                 }
                 if (((xt1 >= Strike_Xt1) && (xt2 >= Strike_Xt2))
                 || (UB_flag == 200)) {
                    price = discount[k] * Face_value * (1 + premium * k); // premium
                    counter4 = counter4 + 1;   // 조기상환 횟수 파악용
                    prestrike_flag = 200; //  prepayment strike 발생여부 저장
                    pretime = k;
                 }else{
                    prestrike_flag=100;
                 }
              }

           }// Next k  각 조기 상환일별 체크
        } //조기상환이 일어나지 않았는지 여부 체크
     }// Next j daily simulation 완료

        if (prestrike_flag == 100) { //만기 지급시
           if ((xt1 >= UB_Xt1) && (xt2 >= UB_Xt2)) {
              UB_flag = 200;    // UB 종가모니터(daily)
           }

           if (((xt1 >= Strike_Xt1) && (xt2 >= Strike_Xt2))
              || (UB_flag == 200)) {
              price = discount[KK] * Face_value * (1 + premium * k);
              counter1 = counter1 + 1; // 만기 수익지급
           }else{
worst_perf = min((xt1 - basis_xt1) / basis_xt1, (xt2 - basis_xt2) / basis_xt2);
              if (DB_flag == 200) {
                 price = discount[KK] * Face_value * (1 + worst_perf);
                 if (Face_value * (1 + worst_perf) < 0 ) {// 손실 100%
                    price = 0;
                 } // 손실 100%
                 counter2 = counter2 + 1; // 만기 손실발생
              }else{ //down barrier를 안쳤다면
                 price = discount[k] * Face_value; //만기 원금만 지급
                 counter3 = counter3 + 1;
              }
           }
        } 만기 지급시
        sum1 = sum1 + price;
        sum2 = sum2 + pretime;
     }//Next i
// return or  memory access
return sum1
}
```

연세대학교
YONSEI UNIVERSITY
금융퀀트연구센터

C언어 Single -> CUDA 병렬 코딩

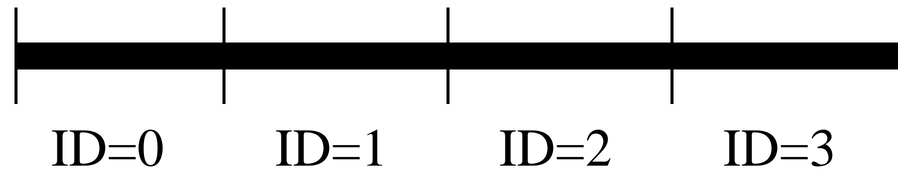
병렬 MC아이디어를 사용

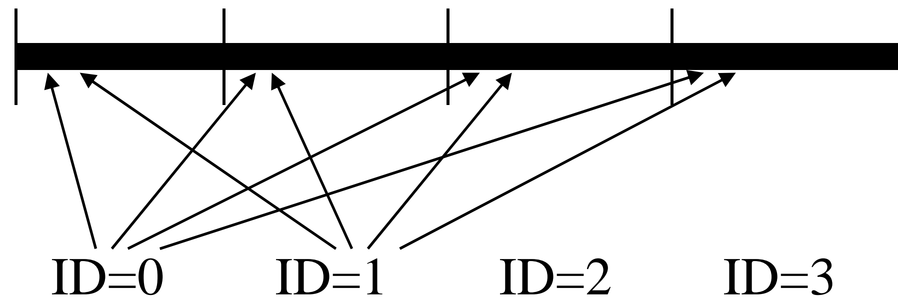
CPU영역과 GPU 영역을 정확히 구분해줘야함

하드웨어 및 CUDA 자체의 특징을 고려해 줘야함.
thread 고민
memory 고민

Thread control on SPMD algorithms

How do we divide job (FOR LOOP) ?





Thread control on SPMD algorithms

1

```
__Global__ ELS (parameters){
   Tid= blockIdx.x*blockDim.x + threadIdx.x;
   N=blockDim.x *threadDim;
    for (i = 0; i < Nsim/N; i++) {
        ….
    }
}
```

3

```
__Global__ ELS (parameters){
 Tid= blockIdx.x*blockDim.x + threadIdx.x;
   N=blockDim.x *threadDim;
   for (i = Tid; i < Nsim; i+= N) {
        ….
   }
}
```

2

```
__Global__ ELS (parameters){
   Tid= blockIdx.x*blockDim.x + threadIdx.x;
   N=blockDim.x *threadDim;
   int rem = Nsim %   N;
   i_start = Tid  * (Nsim /N);
   i_end = i_start + (Nsim /N);
   if (Tid == (N-1))  i_end = N;
   for (i = i_start; i < i_end; i++) {
       ….
   }
}
```

1,3 method is good

3 method is recommanded

연세대학교
YONSEI UNIVERSITY
금융퀀트연구센터

CUDA 고급 메모리 관리

CUDA의 제한사항

__global__, __device__ 함수 내부에서는 static 변수의 선언 불가

Pointer는 global 메모리영역만 지정 가능

global 함수는 무조건 void 함수임.

사용방법

__global__ 함수 밖에서 static 변수를 미리 정의해 줘야함.

__global__ 에서는 메모리를 참조해야함.

Global memory의 포인터 주소를 참조

사용 예)

```
Main(){
__device__  static mt[624*N];                    global memory에 할당됨
__device__  __shared__ static mti,bmused,y1,y2,r1,r2;
__device__  static seed;

MT19937 <<<32,8>>>(void)
}
__kernel__  MT19937(){
Mt[block*BlockID+ 0]=seed;
...
}
```

CUDA 고급 메모리관리 shared pMT19937구현시

Nvidia G80 chip 스펙

16 KB

on chip shared memory : 16384 bytes per Block (1 cycle)
on device global memory : 512~1.5 GB (200 cycle)

Shared Memory가 상당한 고가임

필요한 Static Variables
1개의 SMT19937 처리시 필요한 변수들
배열 : MT[624] – 19968 bytes
변수 : mti, bmused, y1,y2,r1,r2 – 192 bytes

20 KB

배열만으로 H/W제공 shared memory size보다 많이 필요

→ Shared memory(fast)가 아닌 Global memory(slow)사용
mti,bmused, y1,y2,r1,r2 등은 shared memory 사용가능

→ Global memory 사용에 의한 Bottleneck 발생 5배 이상 속도저하

CUDA 차세대 버전에서 필요한 Memory Size

차세대 Nvidia H/W에서의 필요 shared memory size

1차 21504 byte per Block 지원시
고속 shared pMT19937 구현가능

```
__shared__ static mt[624];
__shared__ static mti,bmused,y1,y2,r1,r2;
__device__ static seed;
```

차세대 GPU에서 Global Memory를 DDR5로 제공하여 bandwidth를 높이는 경우에도 추가적인 속도 향상 기대됨

2차 161280 byte per Block 지원시
고속 fully pMT19937 구현 가능

```
__device__ static mt[624*N];
__shared__ static mti,bmused,y1,y2,r1,r2;
__device__ static seed;
```

Case I Algorithm for ELS

Parameter 입력

Loop 병렬화
- RNG(PMT19937) -- shared / global memory version
- Box Muller 실행

Loop 병렬화
- XT simulation 실행 <- 만들어진 RNG 사용함
- ELS pricing Routine
  - IF문을 통해 조기사환, 만기조건 고려

평균값을 구함

결과 출력

Case I　CUDA구조

템플릿

```
Main(){
Parameter & memory copy;
ELS_bodyGPU<<<A,B>>>();
get results;
Print results;
}

__global__ ELS_bodyGPU(){
ELS_kernel();
}

__device__ ELS_singleGPU(){
Option pricing algorithm;
}
```

모듈화

```
__device__  boxmuller(){
}
__device__  MT19937(){
}

void ymalloc() {
}
void yhtod() {
}
void ydtoh() {
}
void ytstart() {
}
```

Case I Pseuco code for CUDA ELS

```
Main(){
   cudaMalloc((void**)  &MTd, 624*sizeof(float));
   cudaMalloc((void**)  &a, 1024*1024*2*sizeof(float));
   Dim3 DimGrid();
   Dim3 DimBlock();
   Random_GPU <<<DimGrid,DimBlock>>>  (parameters);
   Boxmuller_GPU <<<DimGrid,DimBlock>>>  (parameters);
   ELS <<<DimGrid,DimBlock>>>  (parameters);
   sum( option[k] ) /N;  //N is 128

}
__Global__ ELS (parameters){
   Tid= blockIdx.x*blockDim.x + threadIdx.x;
   N=blockDim.x *threadDim;
For( I<0, I< Nsim/N;I++){
      For(j<0,j<Totalday){
         For(k=0;k<monitor;k++){
            Norm1=a[N*i+ blockIdx.x*blockDim.x + threadIdx.x];
            Norm2= a[N*(i+N)+ blockIdx.x*blockDim.x + threadIdx.x];
            Xt1(I)= Xt1+MuT*dt + SigmaT*Norm1
            Xt2(I)= Xt1+MuT*dt + SigmaT*Norm2
            if (Xt1(I) <DB1 and Xt2(I) <=DB2 ) down_flag=1;
            }
         if (Xt1(I) >DB1 and Xt2(I) >=DB2 ) up_flag=1;
         if(j=pre1){ }
         if(j=pre2){ }
         }
         option = ;
         sum = sum+option;
         option = 1/(Nsim/N)*sum;
   Return option[tid];
}
```

```
__device__  float Box_Muller( float a1, float a2){
r-=sqrt(-2.0f *logf(u1));
Float phi = 2*PI*u2;
a1=r*__cosf(phi);
a2=r*__sinf(phi);
}

__ device__ float BoxMuller_GPU(){
Return Box_Muller( a[N*i+Tid], a[N*(i+1)+Tid );
}


 __ device__ random_GPU(){
//Use static variables for each threads
Algorithms for MT RNG

a[K*i+Tid]=y;  //K=2N
}
```

연세대학교
YONSEI UNIVERSITY
금융퀀트연구센터

Case I 분석

Random Number Generation이 대부분 ELS pricing의 계산시간을 차지

장점
CUDA를 이용한 가장 빠른 parallel RNG 생성기법임
함수call에 의한 오버헤드 발생이 거의 없음

단점
미리 RNG를 생성하기 때문에 Memory Size의 제약을 받음
RNG를 Global Memory에 생성후 Xt생성시 Global Memory에서
불러와야 하기 때문에 계속적인 Memory bottle neck 발생

Critical한 단점
조기상환의 경우 미리 만들어 놓은 RNG를 사용할 필요없음
조기상환형 문제 해결을 위한 방법을 고민해야함

→ 조기상환형의 경우 속도향상을 기대하기 어려움 (현재)
조기상환형 상품을 Case I의 알고리즘으로 구축시 GPU를 사용할 필요가 없음

조기상환일별 RNG를 재추출 하는 방법으로 속도향상기대 (코딩이 복잡해짐)

Case Ⅱ Algorithm for ELS

Parameter 입력

Loop 병렬화
- XT simulation 실행
  - Boxmuller,
  - RNG(parallel SMT19937)
- ELS pricing Routine
  - IF문을 통해 조기사환, 만기조건 고려

평균값을 구함

결과 출력

Case II Pseuco code for CUDA ELS

```
Main(){
   cudaMalloc((void**)  &MTd, 624*sizeof(float));
   Dim3 DimGrid();
   Dim3 DimBlock();
   ELS_body <<<DimGrid,DimBlock>>>  (parameters);
   sum( option[k] ) /N;  //N is 128
}

__global__ ELS_ body(parameters){
 Tid= blockIdx.x*blockDim.x + threadIdx.x;
  N=blockDim.x *GridDim;
Initialize();
ELS_ kernel(parameters);
}

__ device__   Initialize(){
Initialize DC of MT19937
}
__ device__   float Box_Muller(){
U1= MyRand();U2= MyRand();
If( used =1){ return} else {return }
}
```

```
__device__ ELS_ kernel(parameters){
For( I<0, I< Nsim/N;I++){
      For(j<0,j<Totalday){
         For(k=0;k<monitor;k++){
            Norm1(i)=Box_Muller();
            Norm2(I)=Box_Muller();
            Xt1(I)= Xt1+MuT*dt + SigmaT*Norm1
            Xt2(I)= Xt1+MuT*dt + SigmaT*Norm2
            if (Xt1(I) <DB1 and Xt2(I) <=DB2 ) down_flag=1;
            }
         if (Xt1(I) >DB1 and Xt2(I) >=DB2 ) up_flag=1;
         if(j=pre1){ }
         if(j=pre2){ }
         }
         option = ;
         sum = sum+option;
         option = 1/(Nsim/N)*sum;
   Return option[tid];
}

__ device__   float MyRand(){
Return MT19937()/4294967296.0;
}
__ device__   float MT19937(){
//Use static variables for each threads
Algorithms for MT RNG
Return y;
}
```

SPDM1 방법론 사용

Case Ⅱ single 코드의 재사용

Closed

Single code와 동일한 구조

Case II 분석

장점

RNG 저장에 메모리를 사용하지 않아 이론상 최대 주기까지 RNG 생성 가능

조기상환시 더 이상 RNG를 생성 안함

Single Program 알고리즘을 거의 수정하지 않고 사용할 수 있음

→ 모듈화 가능

단점

하나의 코어에서 SMT를 돌리기 위해 필요한 메모리 용량이 큼

Dynamic Creation을 통해 균등분할해줘야함

고려할 점

고난위도 thread 컨트롤이 필요 (모듈제작시)

(CUDA의 자동 thread 생성기능이 오히려 속도저하를 불러올 수 있음)

Kernel수준의 SMT코딩 필요, Global 함수에서 통합관리

We need static variables but cuda do not support !!!

실제 코드 main(){}

_ _

Closed

}

실제 코드 ELS_bodyGPU

쓰레드 생성과 파라미터 넘김

Closed

} // __global__ void ELS_bodyGPU() 종료

ELS_singleGPU

```
for(k = kstart ; k<= KK-1; k++ ){   // 각 조기상환일 Loop
            if (pretime[k]==j){  //조기상환일이라면 값 받아옴
              if ((xt1 >= UB_Xt1) && (xt2 >= UB_Xt2)) {
```

Closed

```
    }
} // Next m 장중 simulation 완료
```

```
} // __device__ void ELS_single_GPU 종료
```

Case II 분석

진행상황 – 현재 작업 진행중 (디버그중) : 현재 전체 코드 491 line
모듈화 작업을 병행하다보니 코드 길이가 길어짐

SPMD 1 방법을 이용했음 (모듈화 용이)
GPU에서 single code를 실행시켰을 경우 CPU대비 1.8배 느림
이론상 하나의 보드에서 약 3-40배 정도 성능향상 기대됨
현재 코드에서 약 12배 가속됨

현재 코드 전체를 변경할 예정
SPMD 2 방법을 이용하고 Memory공유시 SPMD1에 비해 4배 속도향상 기대됨

이외에 몇가지 속도 향상 기법이 존재 대부분 메모리, thread 관리 기법임.

왜 128개의 SP를 쓰는데 128배가 안나오지?

이론상
128개의 SP가 128개의 SFU를 가지고 있지 않고 32개의 SFU만 가짐.

GPU : 500Gflops, 최고성능 CPU : 40 Gflops : 약 40배 차이
일반성능 CPU 5 Gflops :약 100배 차이

일반 CPU 1core와 GPU 전체와 비교하면 100배 정도 성능 차이가 남

GPU의 clock speed가 CPU의 clock speed보다 떨어지기 때문에
GPU의 1개의 SP와 CPU 1개의 Core를 비교하면 CPU core 1개의 성능이 더좋음

따라서 GPU의 1개 core대비 SP의 성능 0.4~0.8 * 128 = 약 40~80배 정도
성능 향상 효과가 있음.

최초 출시되었을 때 CPU대비 100배 속도 상향이 가능했지만,
현재는 40배 정도 향상시키면 최적코드로 생각됨

multiGPU 이용시 N배의 scailability 보장함 (MC 방법론의 경우)

결론

CUDA를 이용한 MC 병렬화 (MT19937 이용시)

1. 현재 12배, 좀더 최적화하면 ELS pricing 속도 약 20배 향상 예상
   - 1개의 보드 장착 200sec → 17.00 sec (현재)
   - 1개의 보드 장착 200sec → 10.00 sec (최적화중)

2. multiGPU 시스템 연구예정
   - 4개의 보드 장착된 S870 1대 이용 200sec → 2.50 sec (예상) 80배
   - 4개의 보드 장착된 S870 2대 이용 200sec → 1.25 sec (예상) 160배

3. shared memory size의 제한으로 고속화 불가
   - block당 2.7 KB 가능하다면 약 40배 속도 향상 200sec → 4.00 sec (예상)

4. Single code의 재사용이 용이한 모듈화 가능

5. MC 방법은 다른 방법론에 비해 병렬화 성능이 좋은편임
   FDM, FEM의 경우 병렬처리로 6배 이상 빨라지면 대성공.

결과 On-going Project

- 1차 - 무식하게 병렬화 (완료 4, 9배, 12배)
  - 속도 저하(thread가 다시 thread 생성)

- 2차 – 병렬구조를 고려함 (현재 진행중 목표 20배)

- 3차 – 메모리 한계를 고려함 (현재 진행중 목표 40배)

- 4차 – multiGPU 고려함 (현재 진행중 목표 80배)

연구 계획

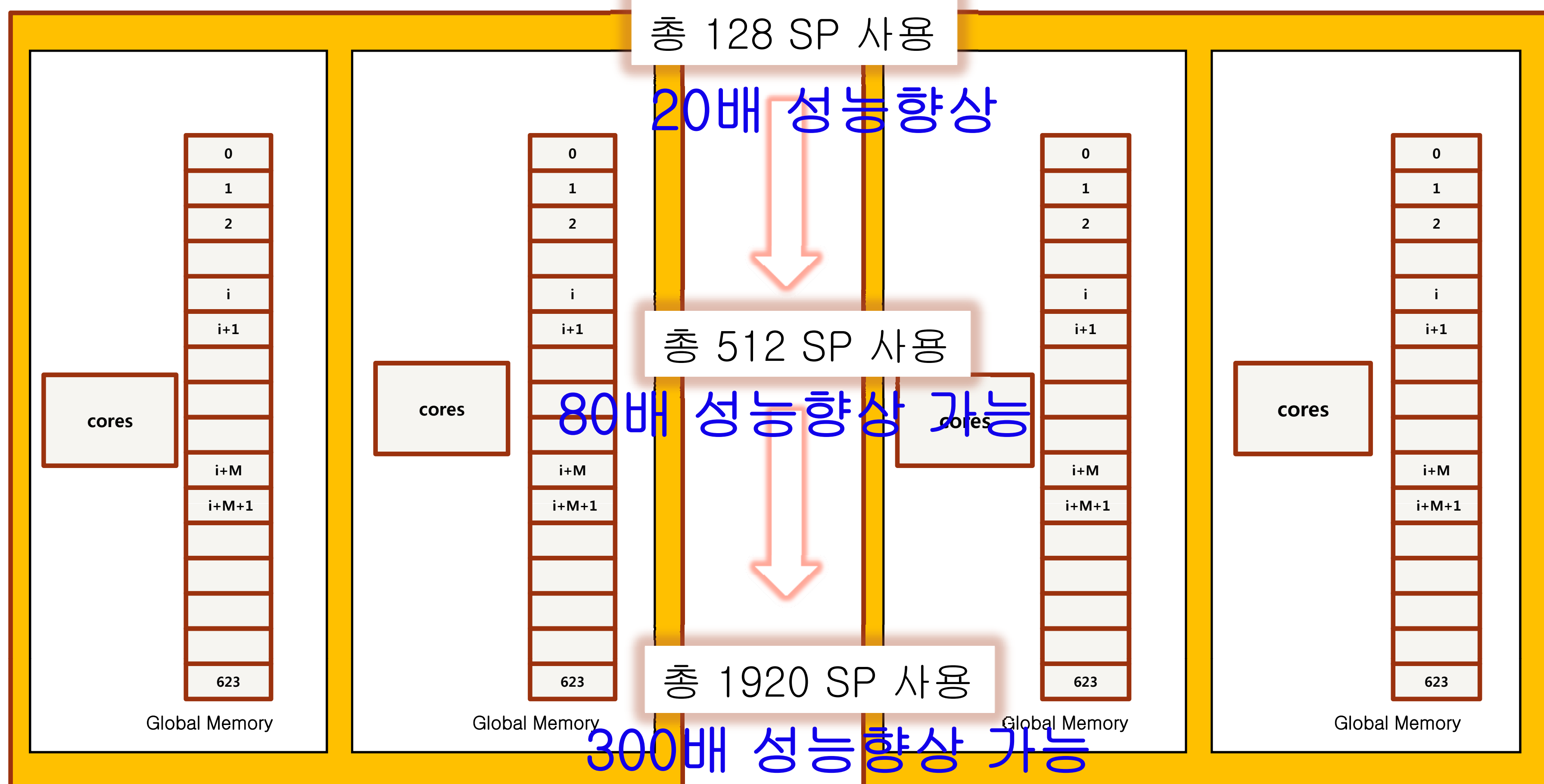
Further Research I   MultiGPU for parallel SMT

모듈화 작업병행

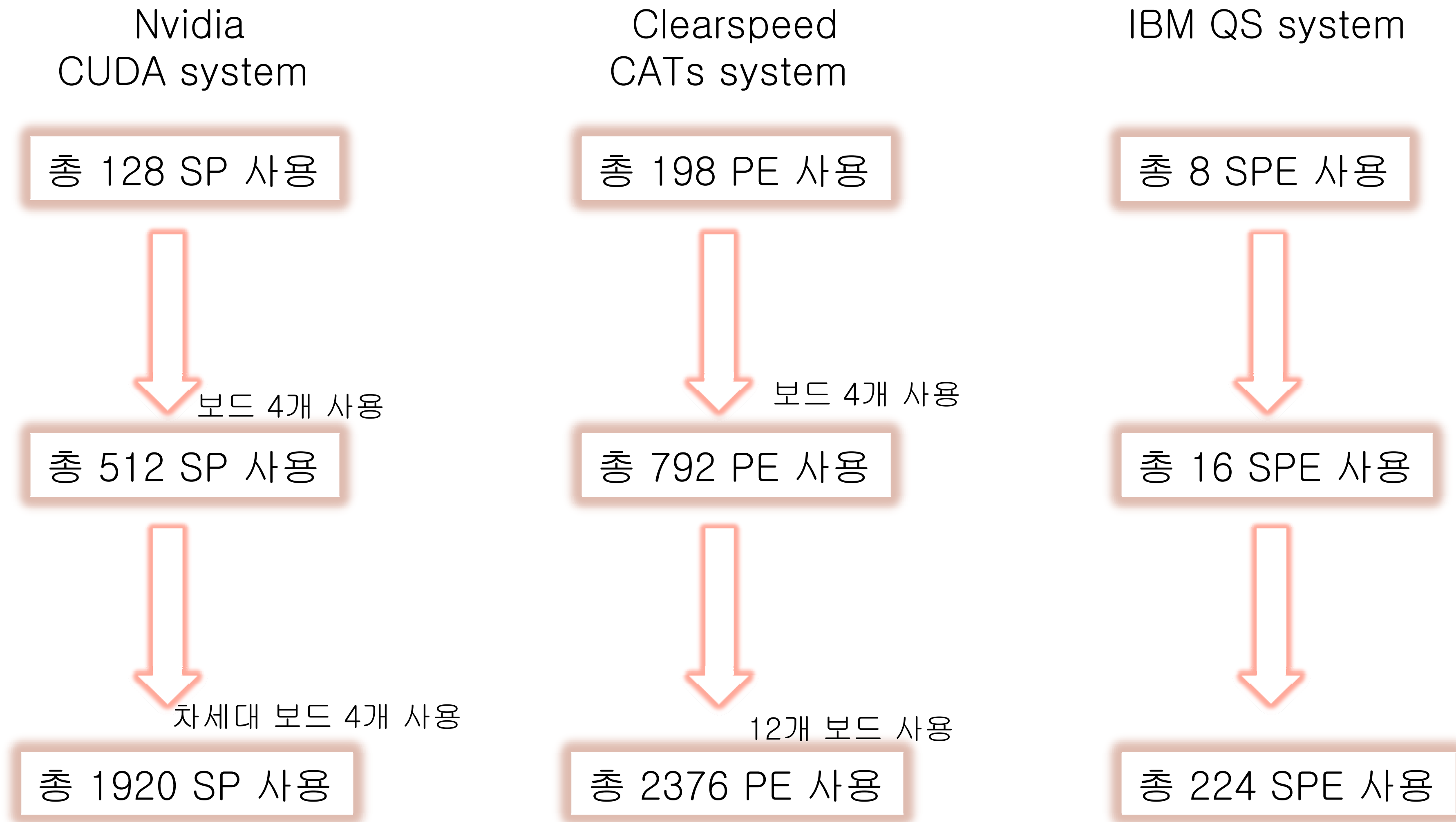
split for 4 multiGPU numbers with pThread, OpenMP or MPI

128 PE execute ELS pricing & SMT19937 algorithm

128 PE execute ELS pricing & SMT19937 algorithm



Further Research II CATs system & IBM Cell



300배 성능향상 가능

Further Research III speed up pMC

- Massively **Parallel** **Quasi RNG**

c.f) prime numbers

500th prime number : 3511, 1000th prime number : 7829 < 2^13

1000 dimensional Halton sequence is possible.. 32비트 연산가능

We need to check the correlations of dimension with each subsequences.

- **New Pseudo RNGs** suitable for massively parallel MC

MT607 is very fast, better than rand48

MT4093 need 512 bytes per block with 128 distributed sequence

MT132049, MT216091 has very large period.

With 128, 64, 32bit integer op, we can generate of 6752 parallel pMC

- New theory in convergence

Malliavin Calculus, Operator Technique, Asymptotics

- Variance Reduction, Control Variate etc.

1000 dimensional Quasi RNG가 구현된다면
CUDA, CS보드에서 4개의 보드 장착시 RNG number가 많이 필요
없기 때문에 가속성능이 우수할 것으로 생각됨.

Further Research V Parallel FDM, FEM

예정

We will test 2 stock parallel FDM, FEM method

Direct parallel algorithm for explicit FDM
Parallel LU solver for CN FDM
CG algorithm for FEM (연세대학교 PDE팀과 공동 연구 구상 중)
Meshless (연세대학교 PDE팀과 공동 연구 구상 중)

The End

경청해 주셔서 감사합니다.