수퍼컴퓨터를 이용한 병렬 몬테카를로 시뮬레이션

유 현 곤 (Hyun-Gon Ryu)
Dept. of Mathematics Yonsei Univ.

Email : yhgon@yonsei.ac.kr
2008년 7월 11일 금요일

목차

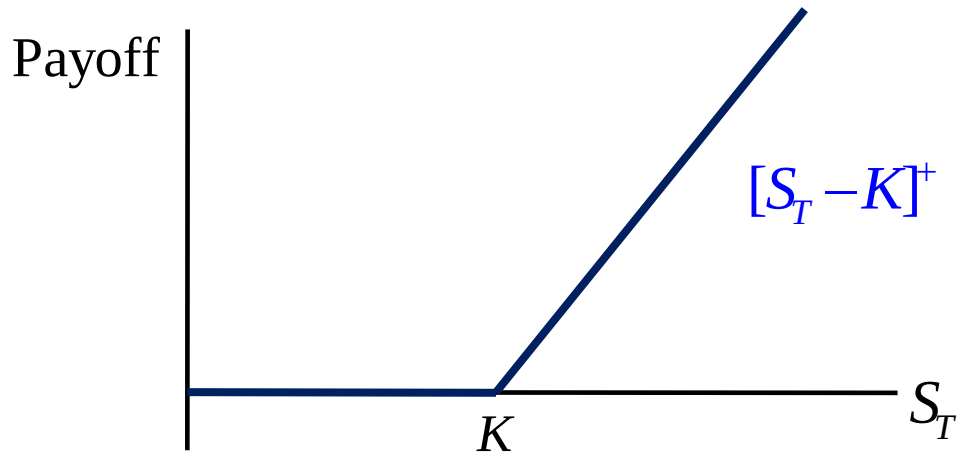
1. 기초 금융 공학,
몬테카를로 시뮬레이션

Introduction for Option Pricing

European Vanilla Call Option

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

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



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

Introduction for Option Pricing

European Vanilla 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

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

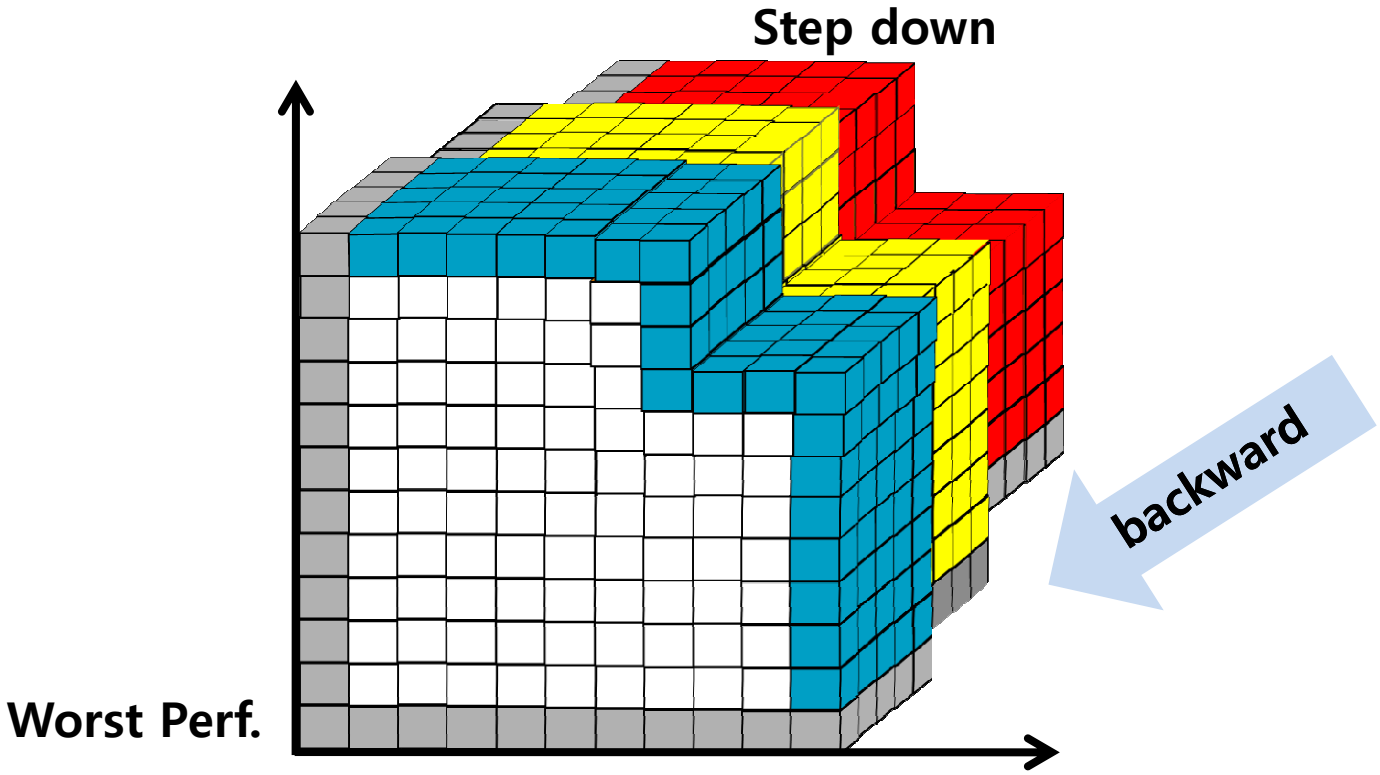
$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-x^2/2}dz \qquad d_1 = \frac{\ln(S_0/K) + (r-\sigma^2/2)\tau}{\sigma\sqrt{\tau}} \qquad d_2 = d_1 - \sigma\sqrt{\tau}$$

PDE method for ELS

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

With Boundary Value& Terminal Payoff

Domain for 2-Star n-chance Stepdown ELS



Monte Carlo Simulation

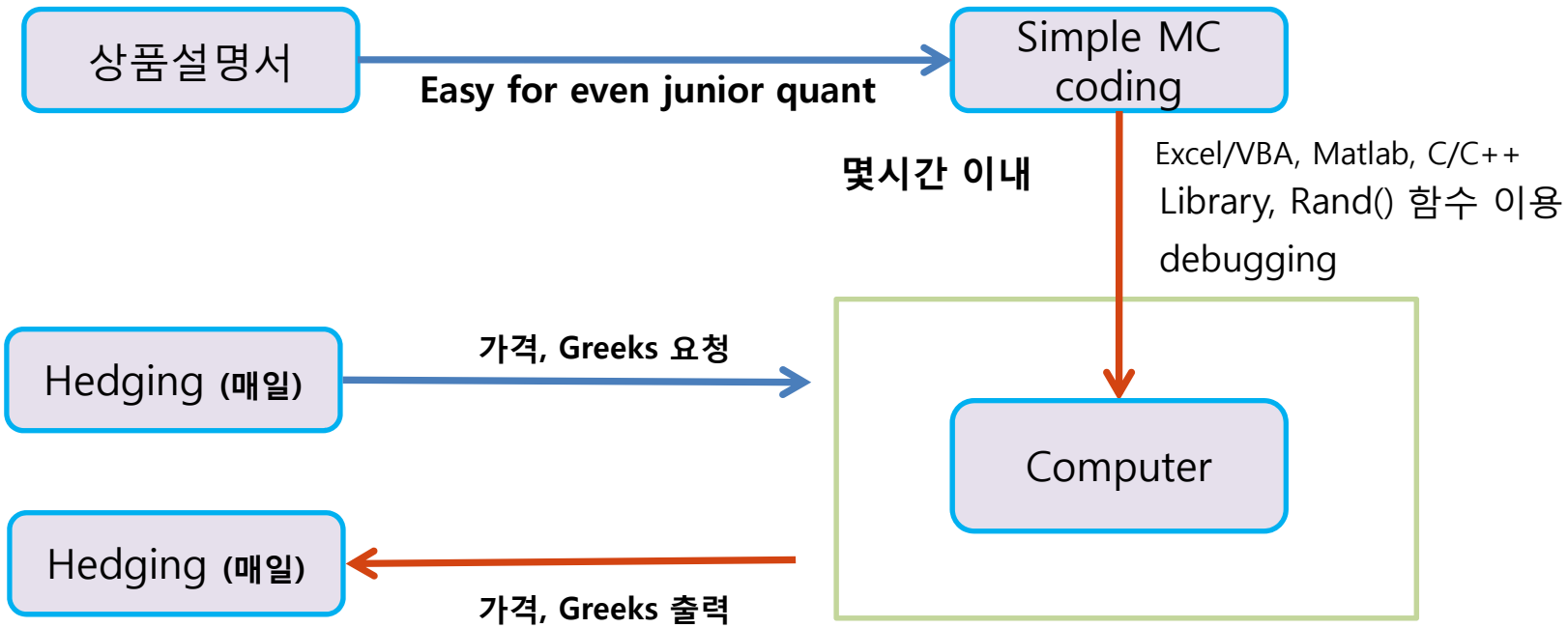
Law of Large Number : 많은 시행을 통한 평균을 통해 기대값 계산

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

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 in Finance



몬테카를로 시뮬레이션의 구현

상품 발행시
Pricing
Overview of Greek
Hedge Simulation
조기상환 확률계산

상품거래시
Pricing
Greeks Search
VaR계산, 위험관리

Pseudo Code for MC

Parameter Inputs

For-Loop of N {

XT simulation with Brown Motion Path

Transformation (Box-Muller, Inversion)

RNG (LCG32, RAND48, MWC, MT19937)

Compute Payoff

IF, Max function (payoff, Prepayment)

Sum

}

Average (1/N)

Print Results

Monte Carlo Simulation

Why is MC useful ?

- 적용이 쉽다. Easy to imply

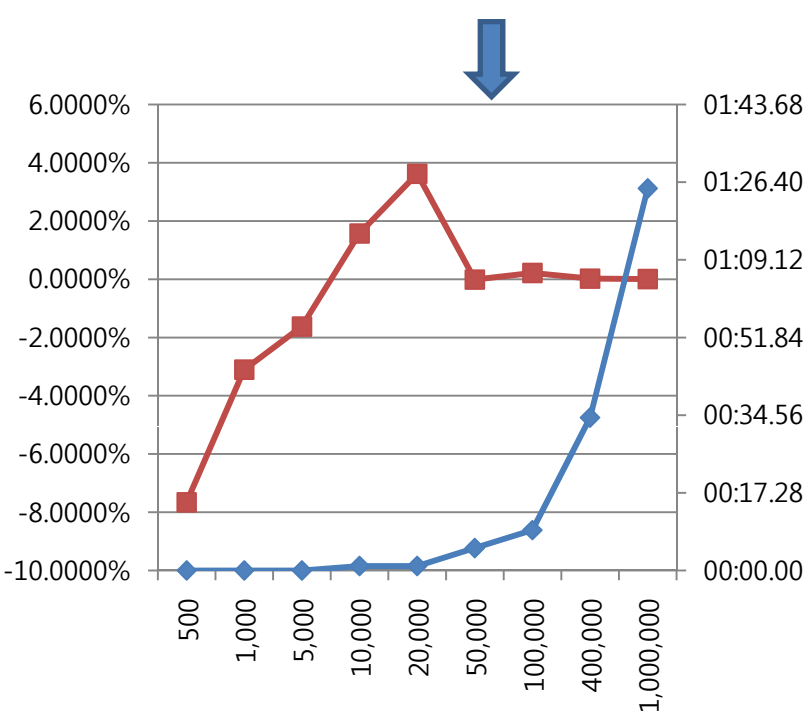
- 복잡한 상품의 평가가 가능하다.
  Complex structure : path-dependence, prepayment,
  complex payoff, etc.

다차원(다중자산) 문제를 풀 수 있다.
High demension : multi-asset problems (n>4)
➔ MC is the only solution (or sparse grid method)

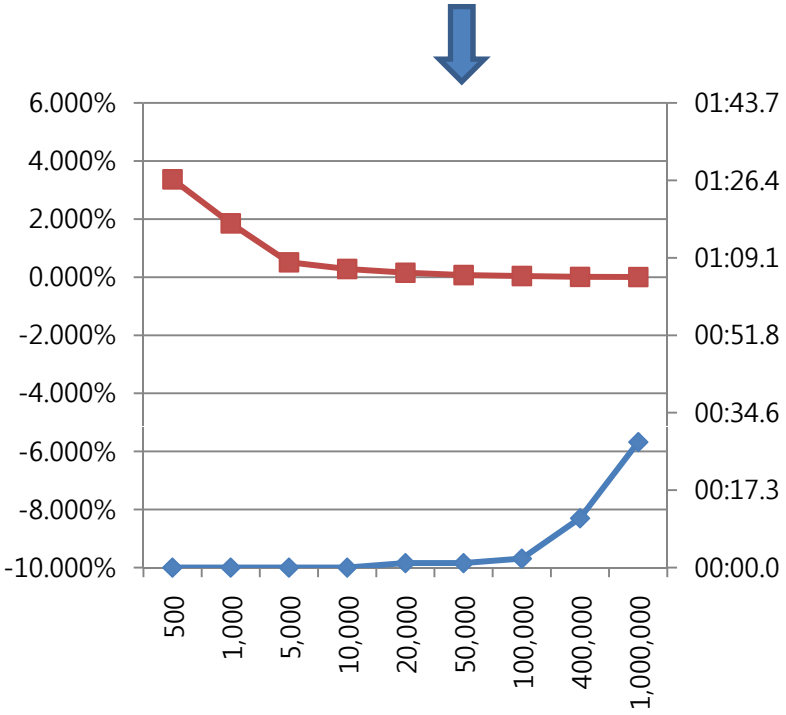
Convergence of Random Number (excel)

Vanilla Call Option Monte Carlo Simulation

N = 50000번 이상, 10만번 정도가 적당



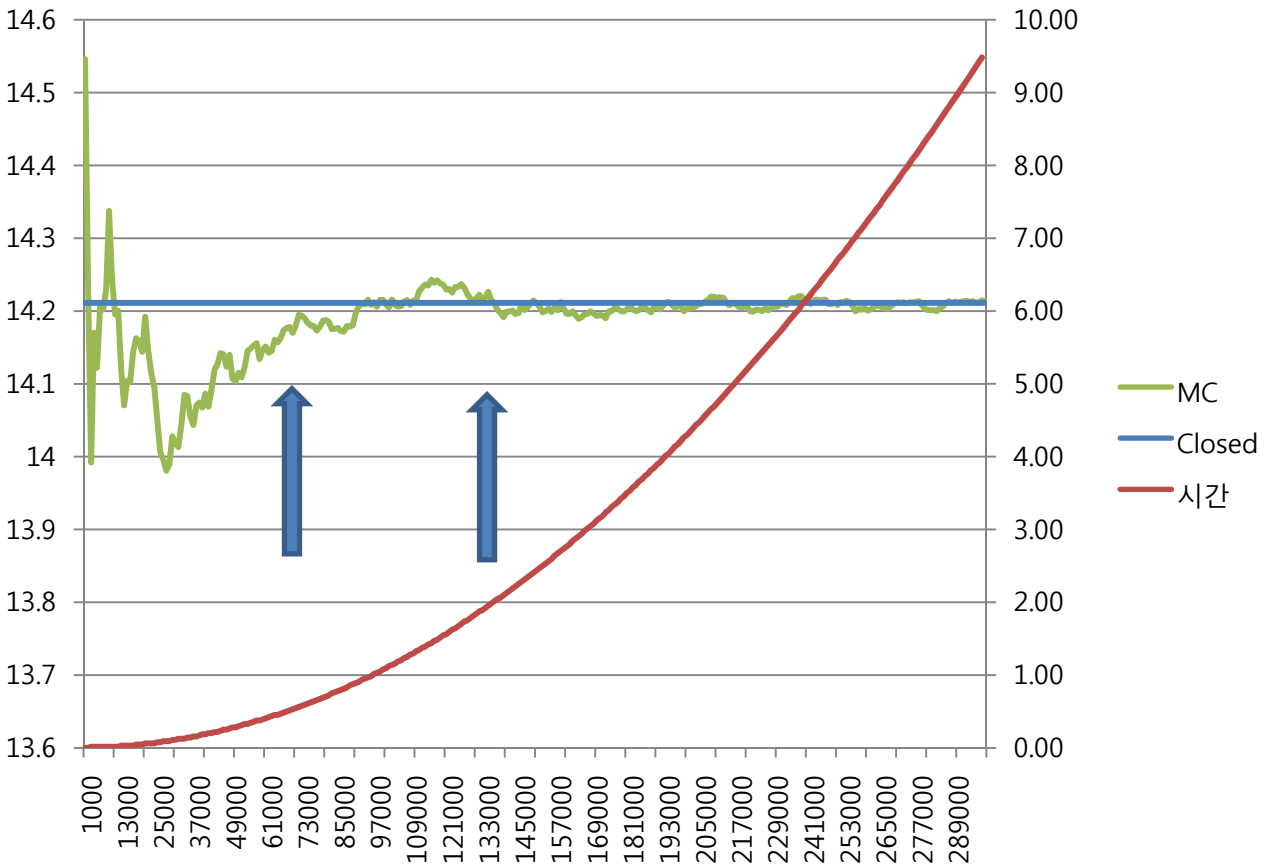
Pseudo Random Number
LCG 32



Quasi Random Number
1D Halton Sequence

European Call Option

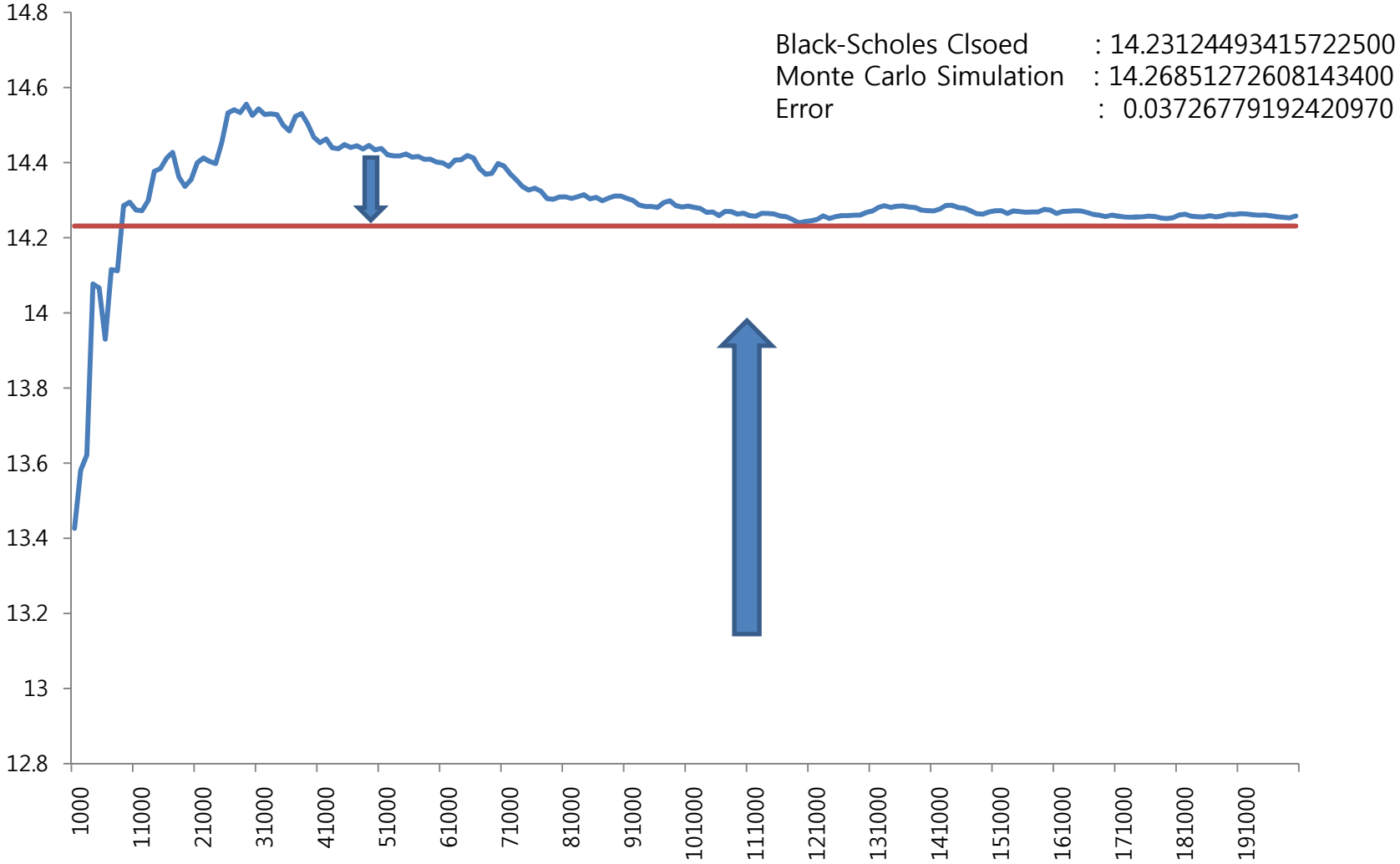
환경 : Intel CPU 3Ghz



Simple MC의 경우12만번 Simulation 하는 것이 적당 2 sec
하지만 계산시간의 제약으로 약 5만번 정도 돌림

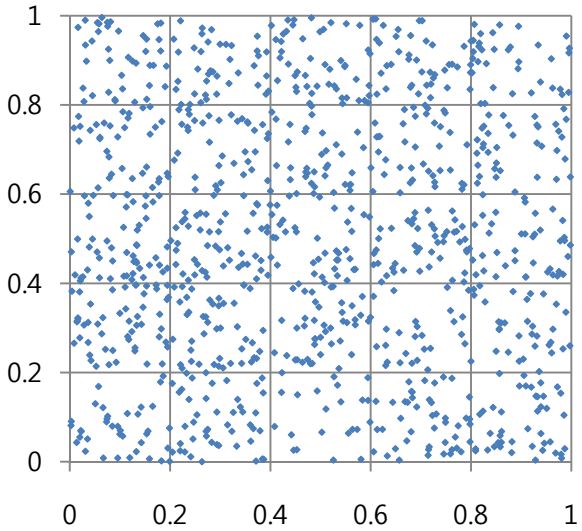
250 time step Path simulation

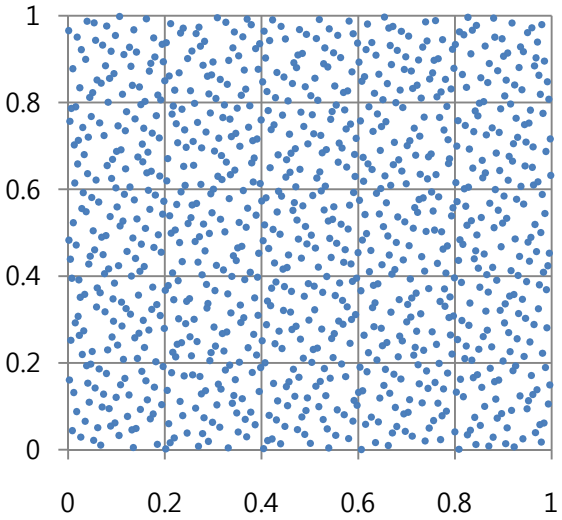
환경 : Intel CPU 3Ghz



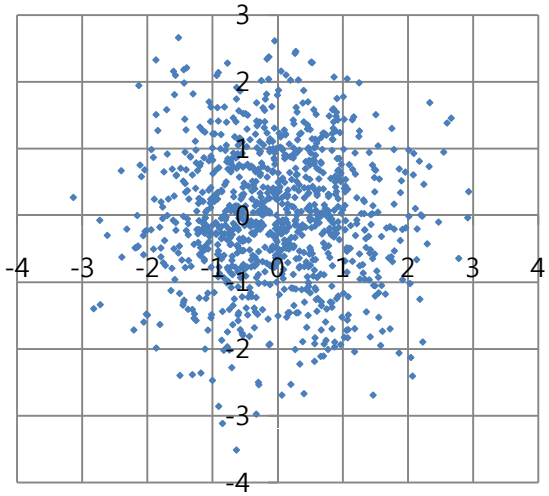
Simple MC의 경우10만 이상이 Simulation 하는 것이 적당 7.65sec
하지만 계산시간의 제약으로 약 5만번 정도만 돌리면 . . .

Pseudo vs. Quasi

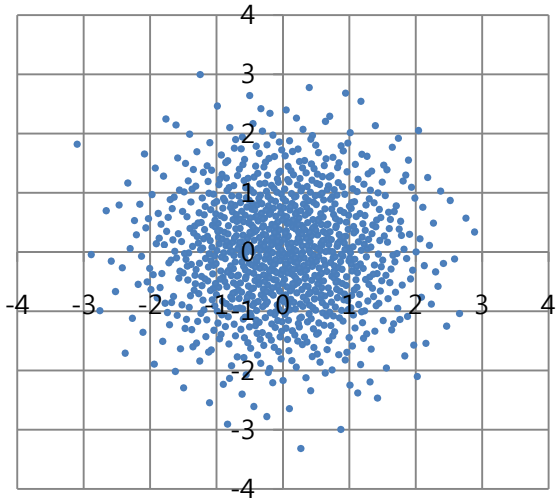




본 발표에서는 생략



LCG32



Halton Sequence

Monte Carlo Simulation

Why did MC not used commonly?

- 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 rather than FDM**

Convergence Ratio Chart

Speed chart

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)

2. 병렬컴퓨팅이란?

HPC vs. distribution/load balancing

병렬처리는 크게 3가지로 나뉘는데

하나의 방법은 **HPC**로, 하나의 작업을 여러 개로 쪼개어 계산하도록 함으로써 계산시간을 단축시킴

다른 하나의 방법은 여러 개의 작업을 여러 개의 서버에 **분산처리**시켜 전체작업시간을 단축시키는 기법

웹서버, 게임서버에 사용되는 로드벨런싱도 분산처리기법의 하나임

계산의 관점에서 병렬처리는 HPC를 의마함

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)

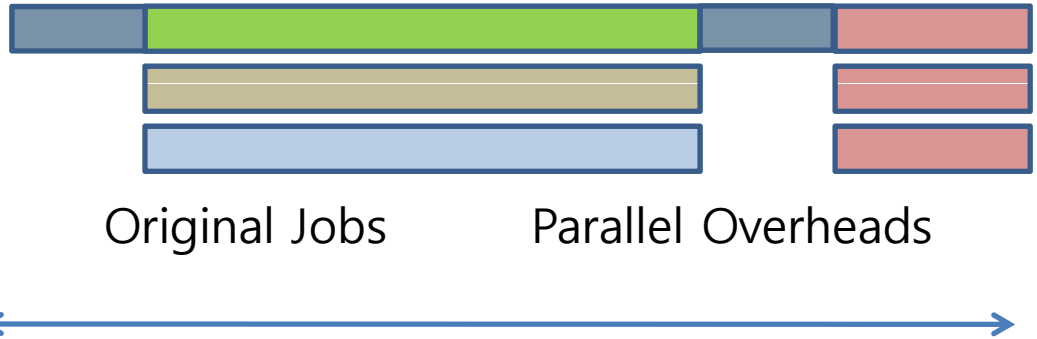
병렬처리(HPC – High performance Computing)

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



Wall Clock Time 2

KISTI 수퍼컴퓨터 4호기 Tachyon

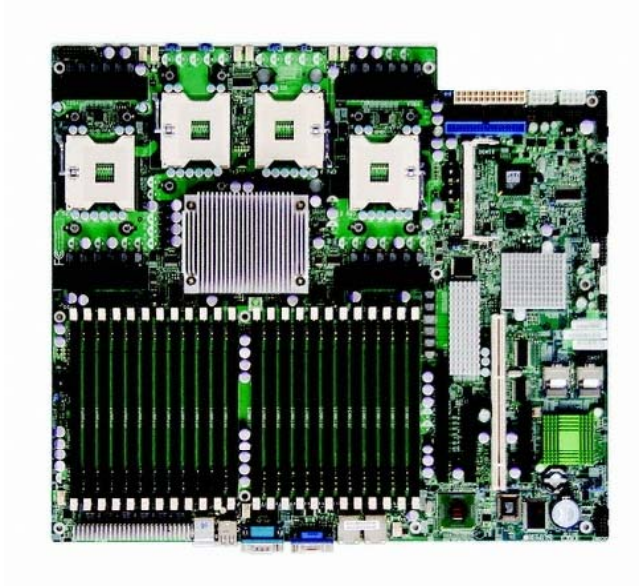
세계 수퍼컴퓨터 랭킹 130위

구 분	내 용
모델명	SUN Blade 6048
블레이드 노드	성능 : 24TFlops(Rpeak) 노드 : 188개(컴퓨팅)
CPU	AMD Opteron 2.0GHz 4개 노드당 : 16 Core (Quad Core) 총 : 3,008 개 (전체)
메모리	32GB(노드)
스토리지	207TB(디스크), 422TB(테이프)
노드 간 네트워크	Infiniband 4X DDR

금융권 HPC용 16코어 SMP머신

Blue Box : HP, IBM, SUN 등
도입비용 1700만원(프로모션견적) 가량

HP ProLiant DL580G5 Rack Type
4U case 사용
인텔 제온 E7340 CPU 4개 : 16코어
메모리 8GB



White Box :
비슷한 도입비용으로
동일 스펙에 Memory 48GB정도로 확장 가능
단, 유지 보수 등의 문제 발생

Node당 100Gflops 유지
Tachyon 1 노드와 유사한 성능
Tachyon 전체 시스템 대비 1/250 배 성능

N개의 노드 추가를 통해 MPI머신으로 성능향상 가능
대형시스템은 Tachyon과 유사한 blade 기반이 유리

수퍼컴퓨터 4호기 Tachyon on KISTI

구 분	내 용
계정	SRU 100시간 : 100만원 (기업) SRU 10시간 : 10만원(학생)
접속환경	ssh 접속 30분
실행환경	CentOS 4.4 - Linux 환경
컴파일러	GCC, PGI Compiler , Intel Compiler
병렬라이브러리	OpenMPI
수치라이브러리	IMKL, IMSL 등

계정 사용을 통해 HPC 도입 효과를 미리 검토해 볼 수 있음

작업노드(프로그래밍, 디버깅)

Tachyon Server는 SSH를 통해 접속해야 하고
30분까지로 접속시간이 제한되어 있음

따라서, 다음의 시스템에서 OpenMP, MPI 코딩을 작성, 디버깅하고
최종 결과물은 Tachyon에서 실행하는 방법을 사용한다.

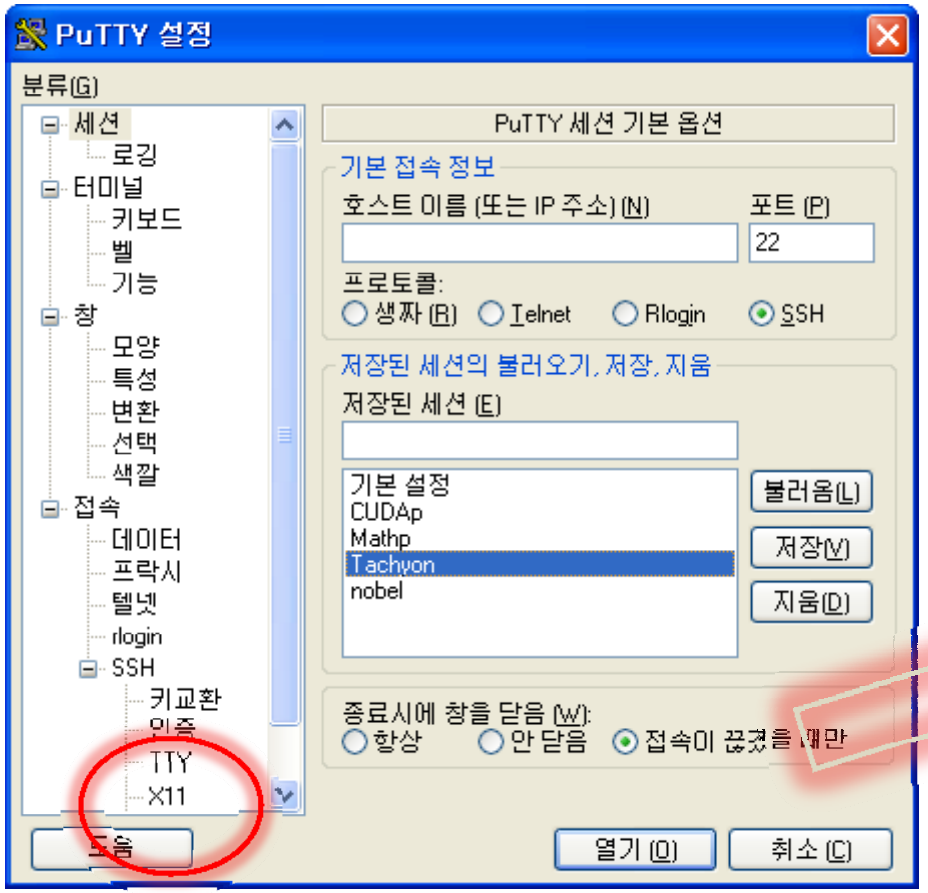
1. Windows XP 환경 (PC) : GUI 환경에서 programming
MS Visual Studio 2005+ MPICH2

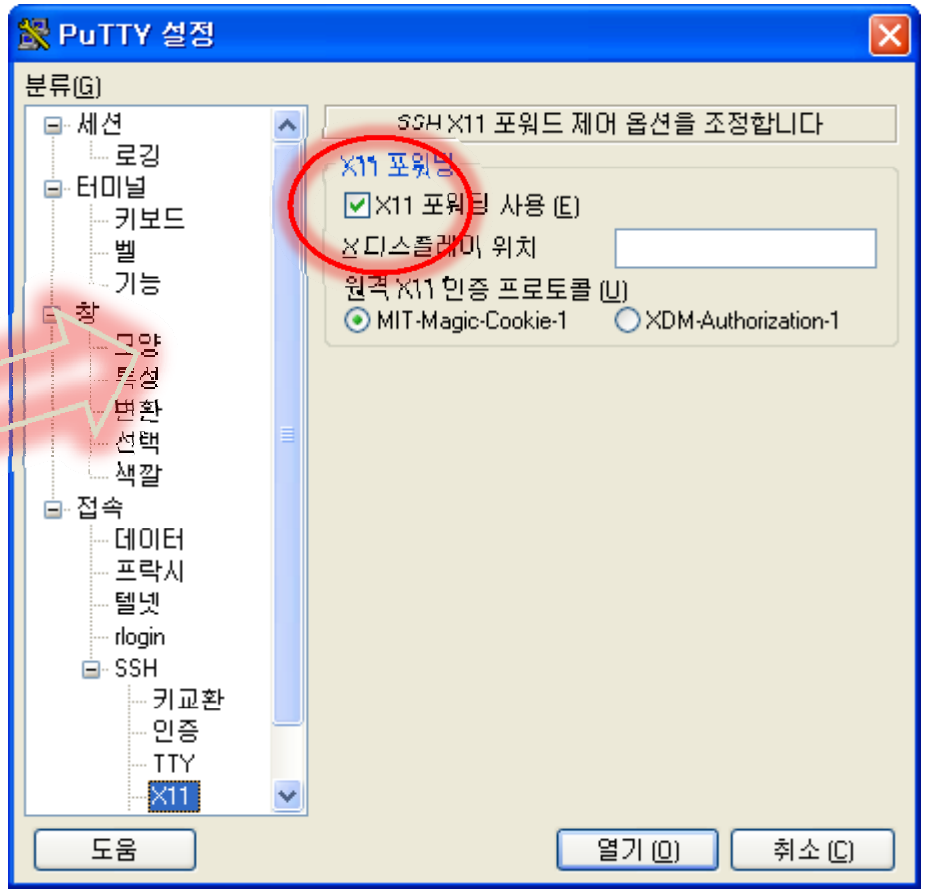
2. Linux 환경 (server) : linux에서 실행 테스트
ICC 10.1 + OpenMPI

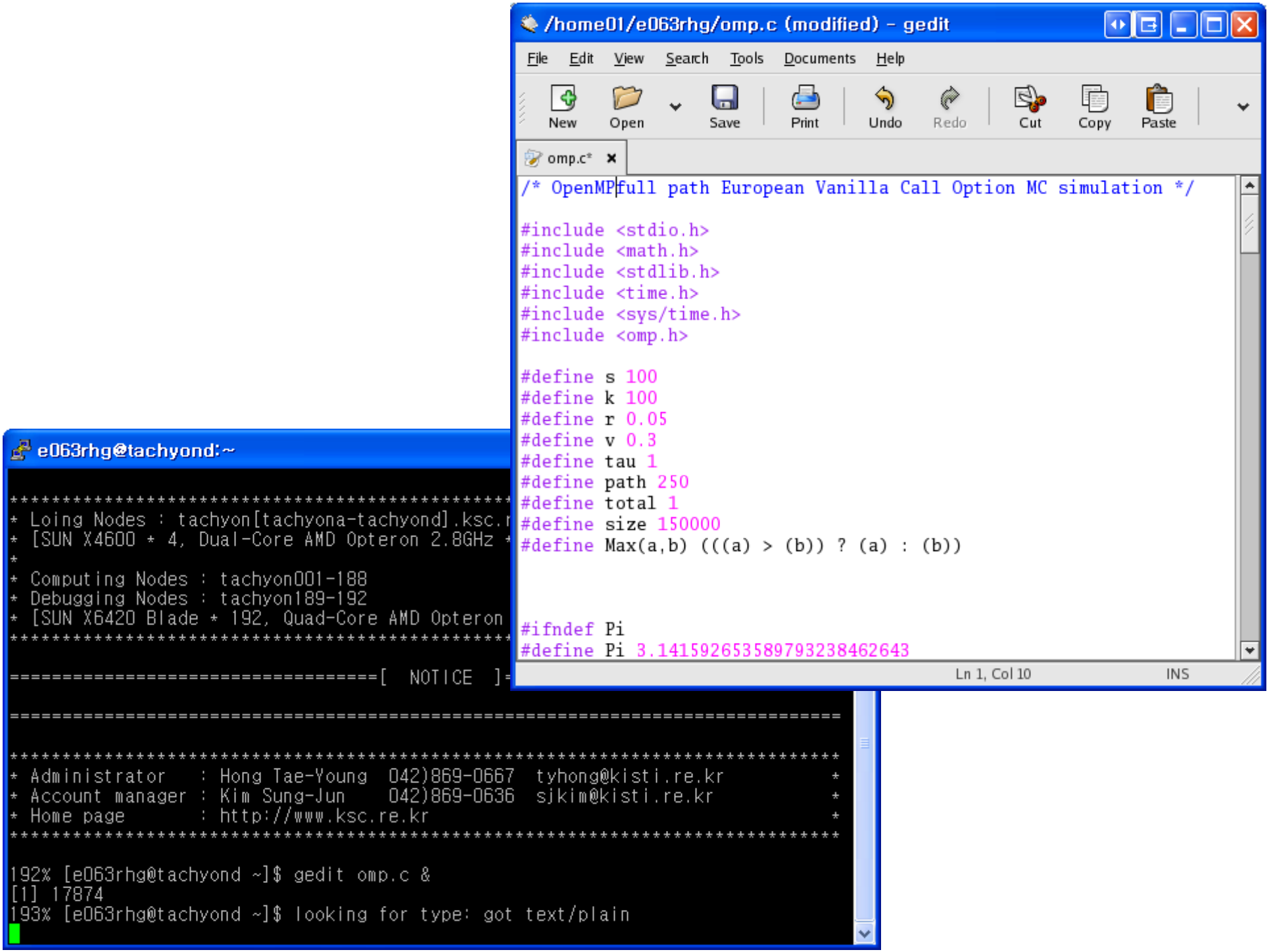
3. Tachyon Server : 최종 코드를 job_batch 실행

SSH 원격 접속환경

Windows 환경에서 PuTTY를 통해 접속






/home01/e063rhg/omp.c (modified) - gedit
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut Copy Paste
omp.c*
/* OpenMP full path European Vanilla Call Option MC simulation */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <omp.h>
#define s 100
#define k 100
#define r 0.05
#define v 0.3
#define tau 1
#define path 250
#define total 1
#define size 150000
#define Max(a,b) (((a) > (b)) ? (a) : (b))
#ifndef Pi
#define Pi 3.14159265358979323846264 3
Ln 1, Col 10
INS
e063rhg@tachyond:~
* Loing Nodes : tachyon[tachyona-tachyond].ksc.
* [SUN X4600 * 4, Dual-Core AMD Opteron 2.8GHz *
* Computing Nodes : tachyon001-188
* Debugging Nodes : tachyon189-192
* [SUN X6420 Blade * 192, Quad-Core AMD Opteron
[NOTICE]
* Administrator : Hong Tae-Young 042)869-0667 tyhong@kisti.re.kr *
* Account manager : Kim Sung-Jun 042)869-0636 sjkim@kisti.re.kr *
* Home page : http://www.ksc.re.kr *
192% [e063rhg@tachyond ~]$ gedit omp.c &
[1] 17874
193% [e063rhg@tachyond ~]$ looking for type: got text/plain

병렬 환경

- **SMP 머신 : OpenMP 프로그래밍**

  16 core 지원 : 2천만원 미만 (16배 성능향상 보장)

- **Cluster 머신 : MPI 프로그래밍**

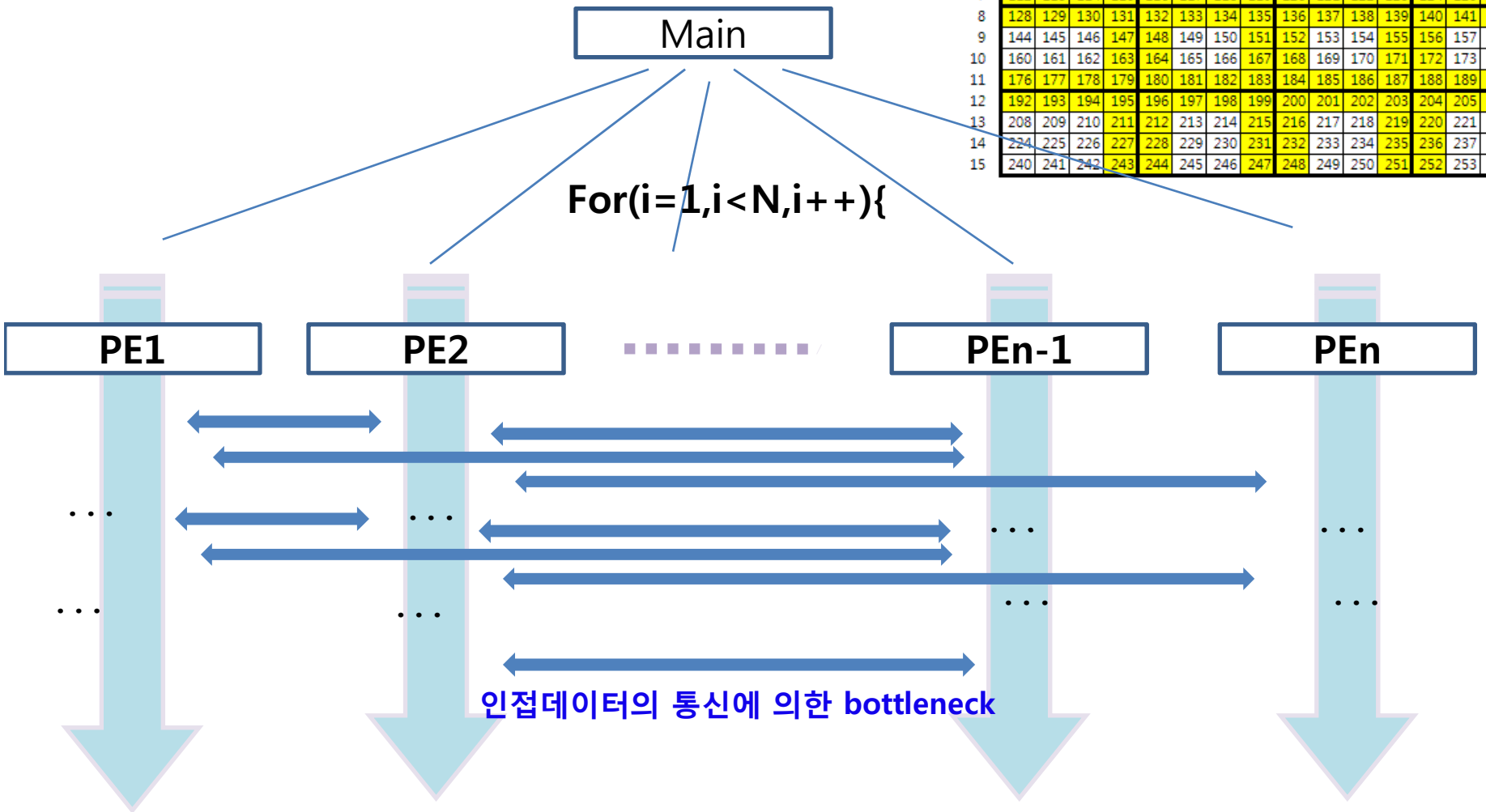
  64 core 지원 : 1억원 미만 : (60배 성능향상 보장)

→ OpenMP, MPI : (무료)

→ IMSL, IMKL : 병렬 라이브러리 : (유료)

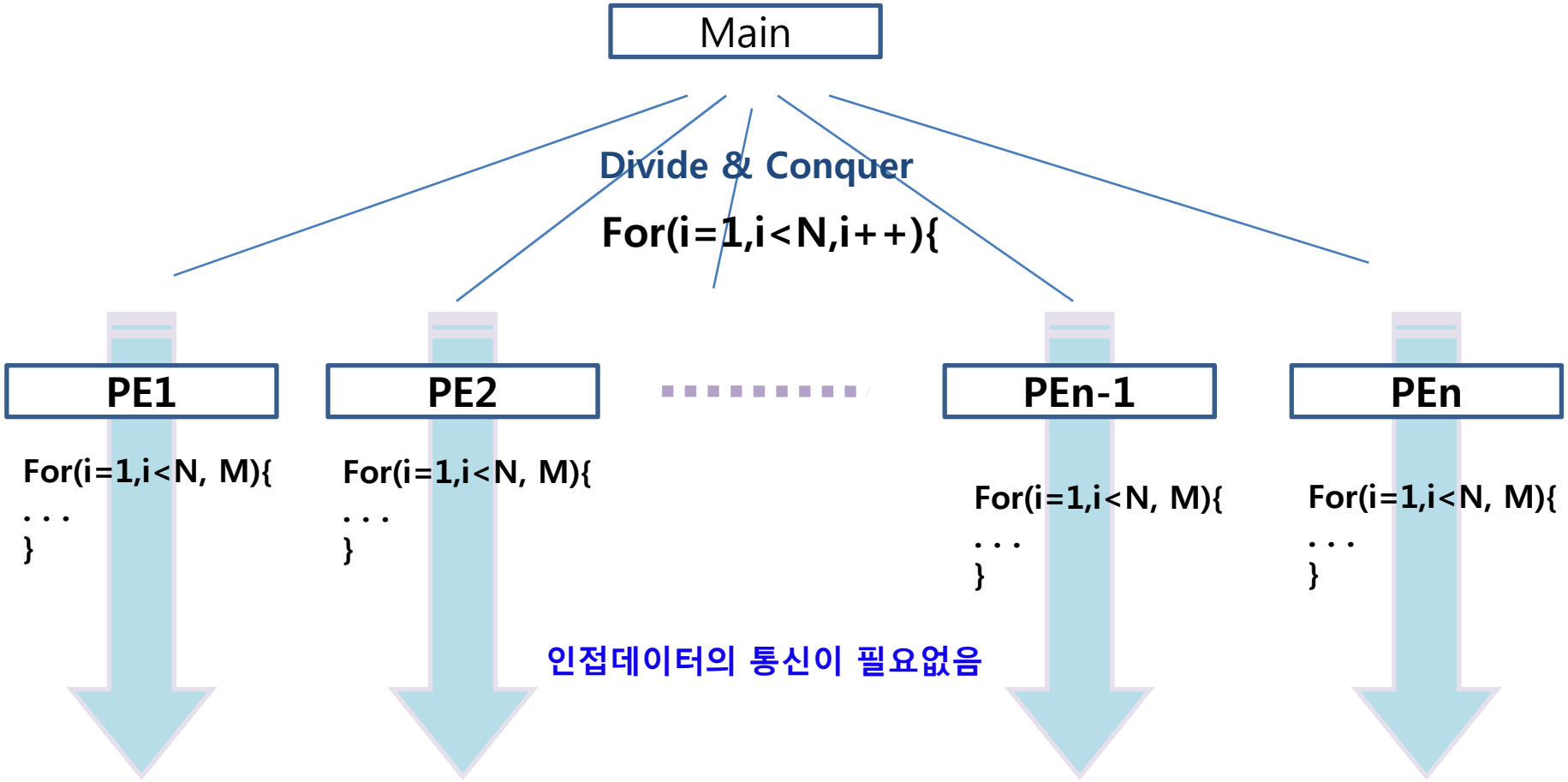
Parallel FDM, FEM (matrix)

	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



3. 병렬 몬테카를로 시뮬레이션

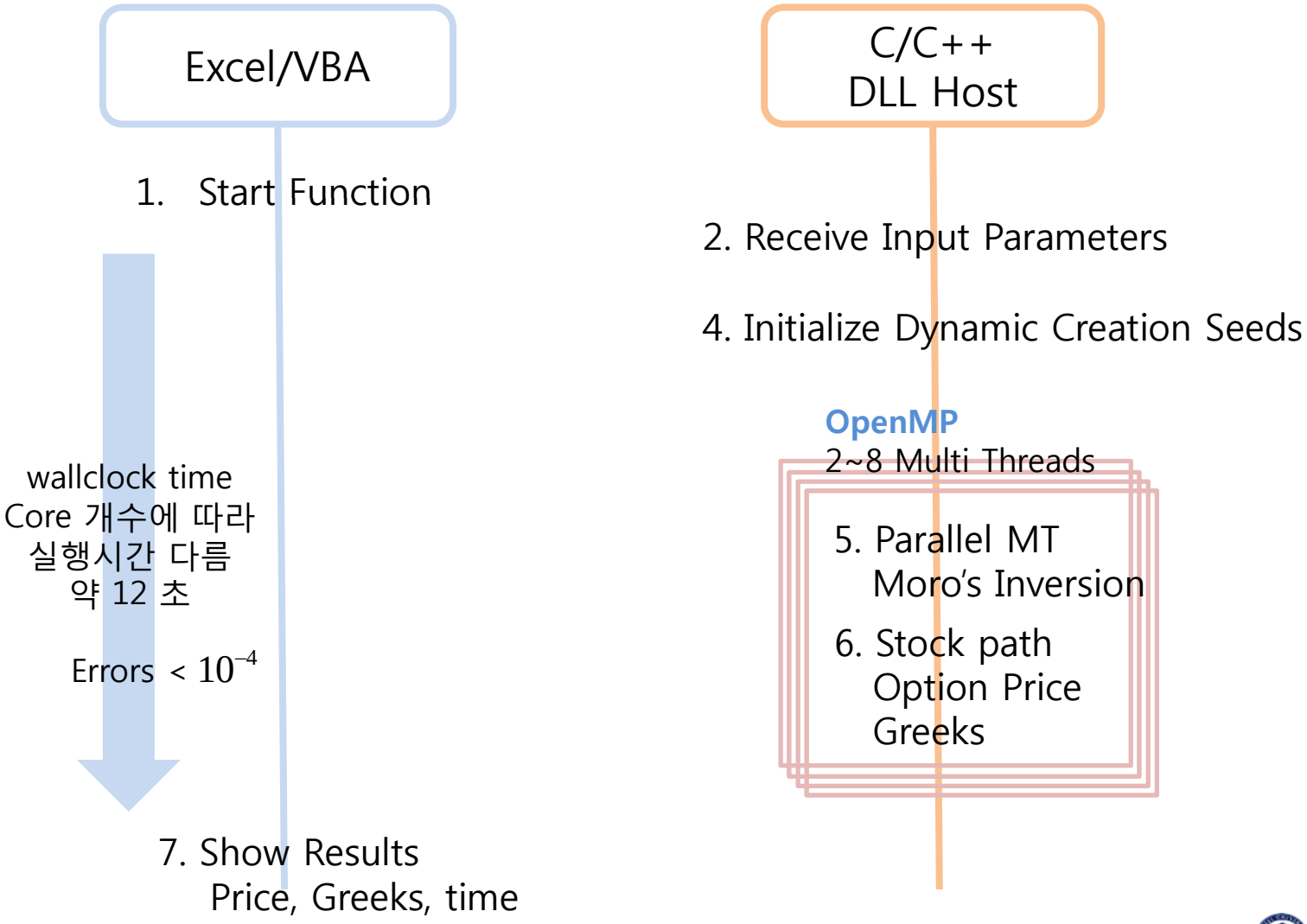
Parallel MC simulation



: MC 병렬 scailability가 좋다

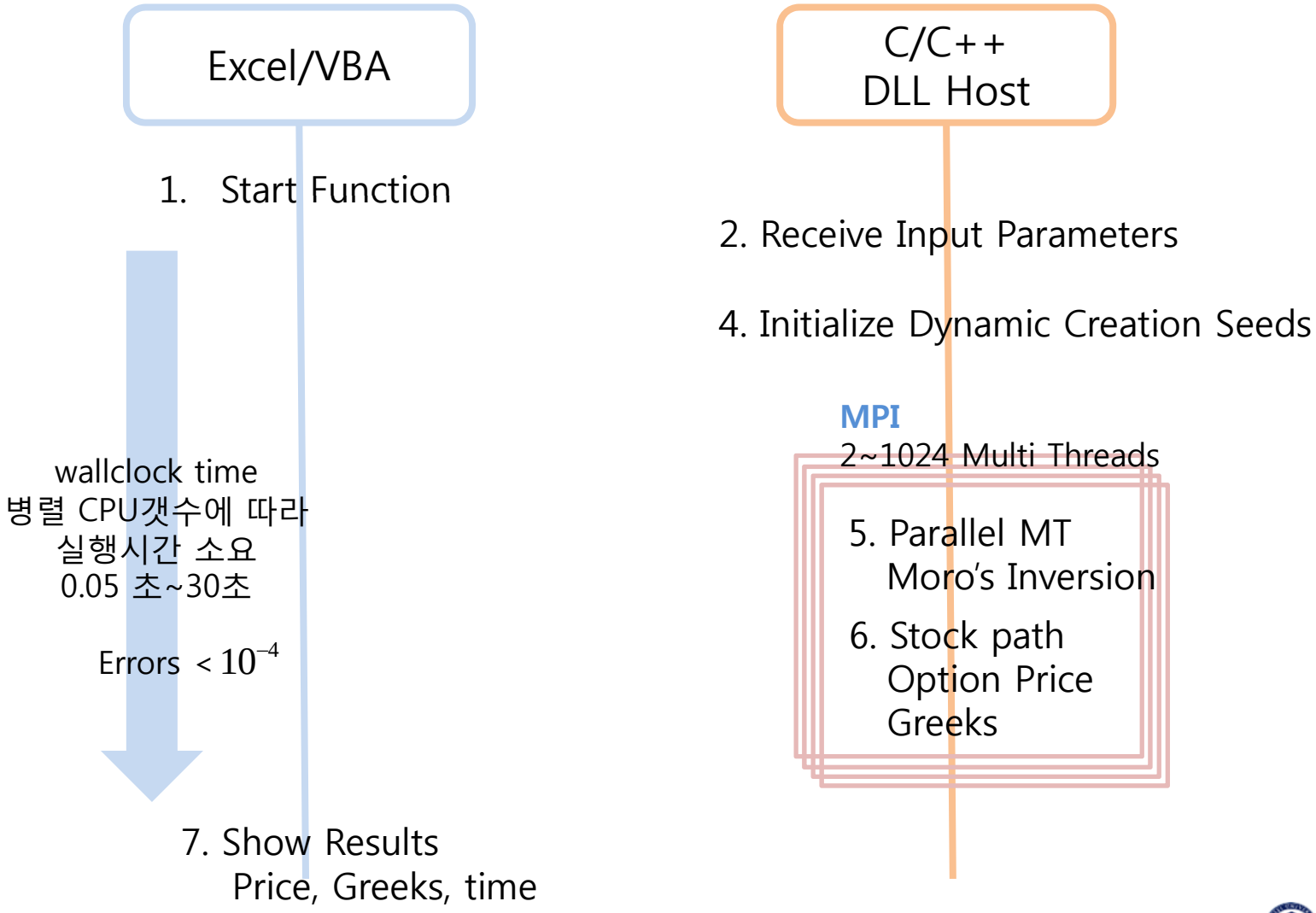
Parallel MC with OpenMP (SMP machine)

10만번 실행

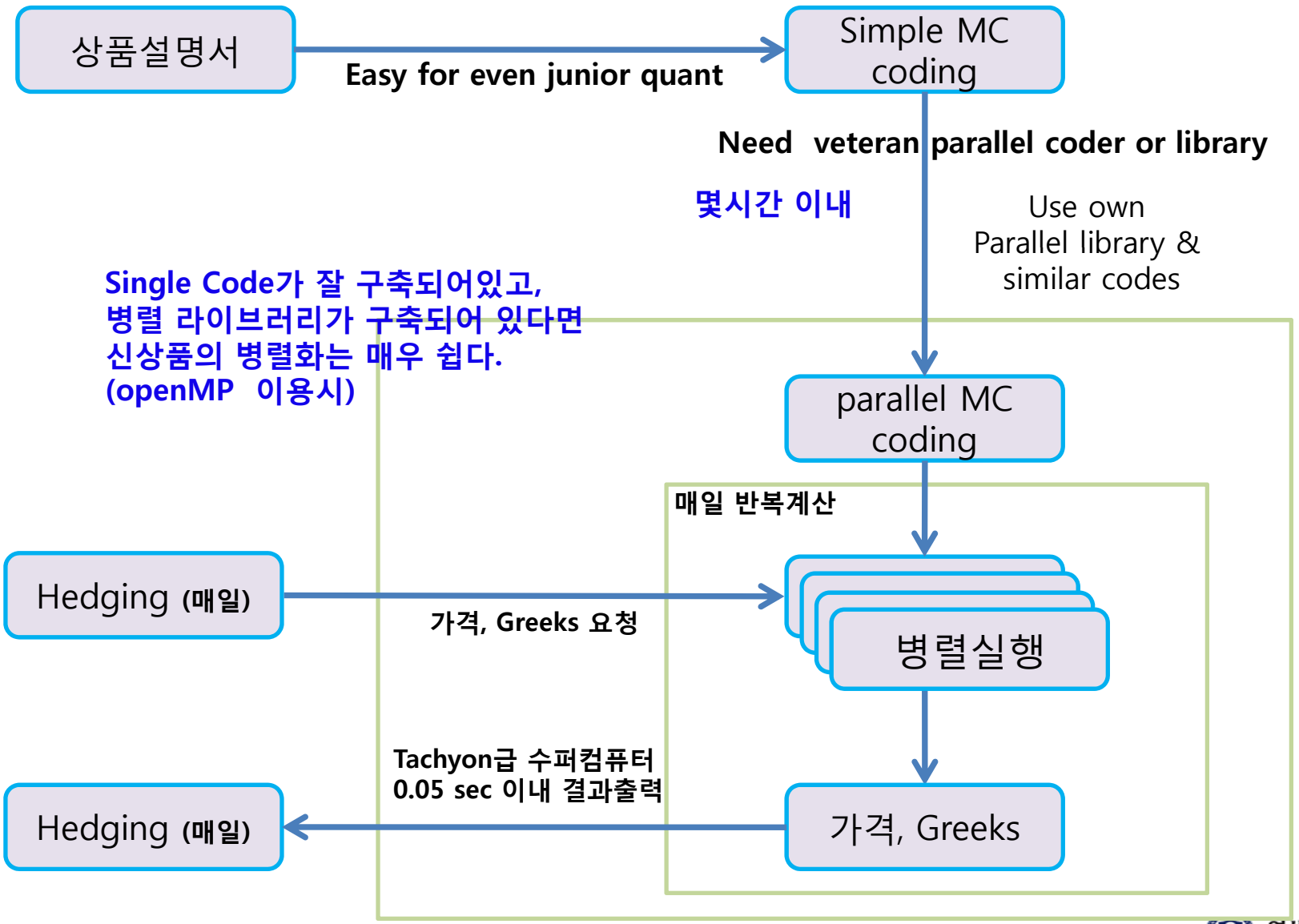


Parallel MC with MPI (Cluster Machine)

10만번 실행



Parallel Monte Carlo Simulation in Finance



LCG32

$$X_{n+1} = (a \cdot X_n + c) \bmod m$$

	M	a	c
Visual C/C++	2^{32}	214013	2531011
GNU C	2^{32}	69069	5
Unix (LCG48)	2^{48}	25214903917	11
ANSI C	2^{32}	1103515245	12345
Numerical Recipes	2^{32}	1664525	1013904223

Period : 2^{32}-1 = 4294967295 = 4.2 * 10^9 약 42억개

Split Method

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

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

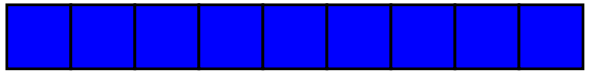
CPU ID 0 CPU ID 1 CPU ID 2

Multiseed Method

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

```
srand48p(123+(1+cpuID)*45);
```

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



CPU ID 0

CPU ID 1

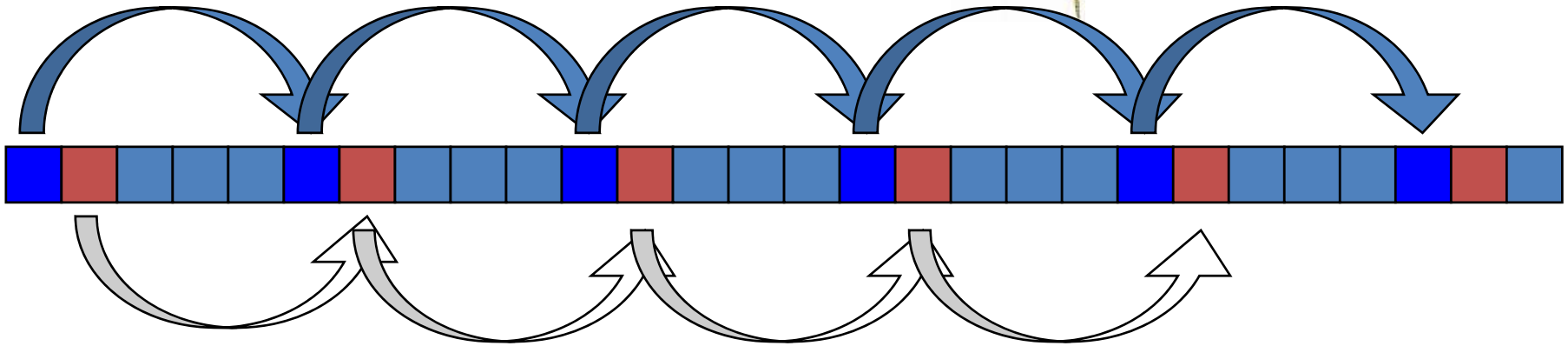
CPU ID 2

Leap Frog Method

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

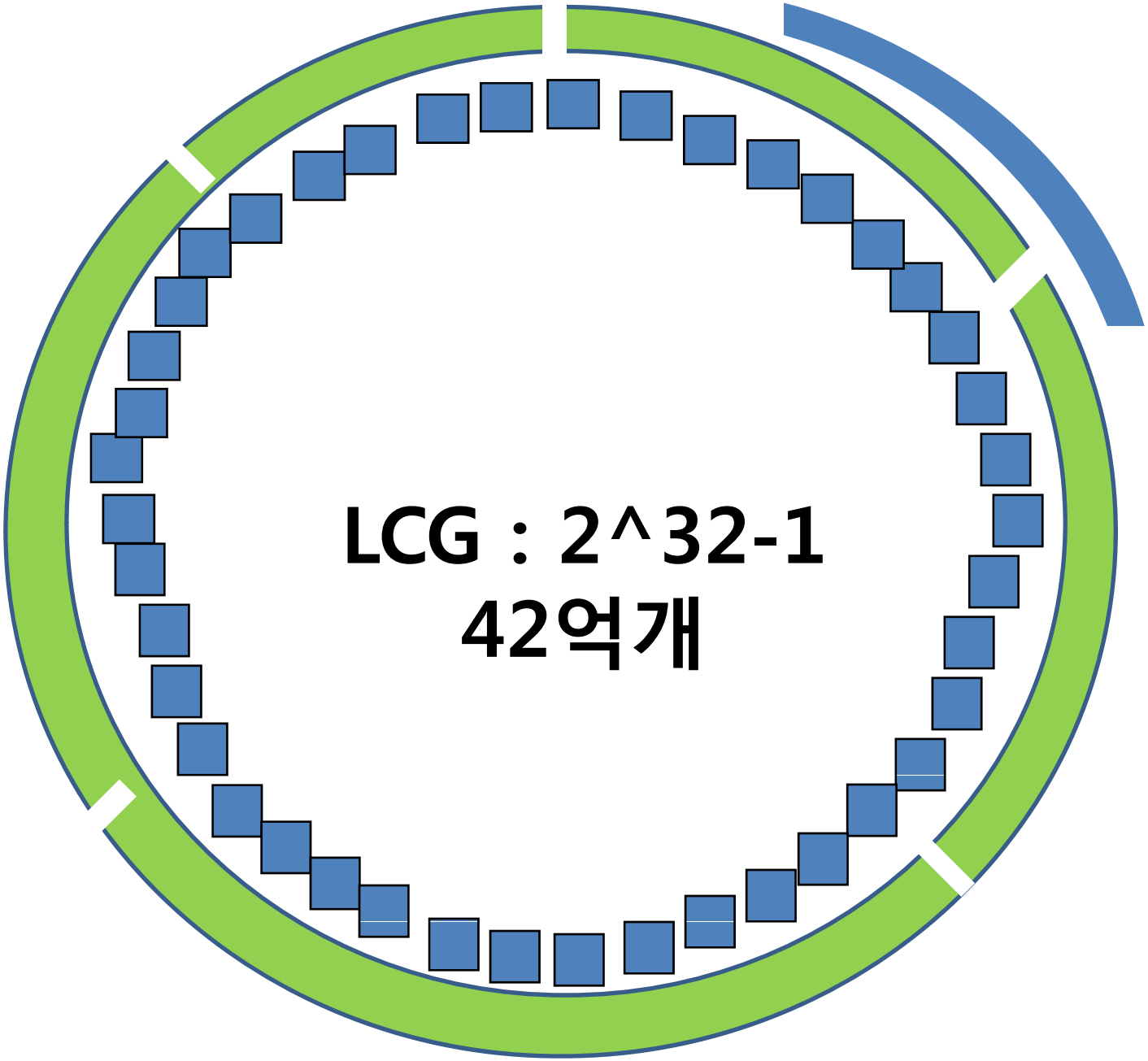
for (i=istart; i<N-CHUNKSIZE-1; CHUNKSIZE)

CPU ID 0



CPU ID 1

Problems in Parallel MC



Suitability for Parallel Method

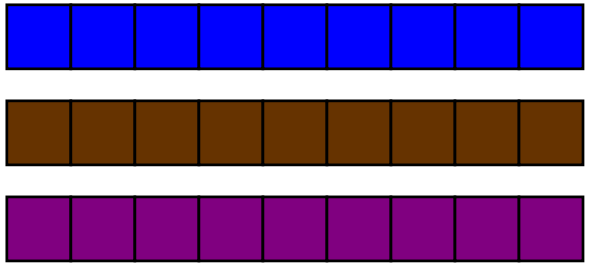
	Multiseed	Leapfrog	Splitting	D.C	J.Ah.
LCG	OK	OK	OK	X	-
RAND48	OK	OK	OK	X	-
MCG	OK	OK	OK	X	-
Lagged Pibonachi	OK	OK	OK	X	-
MWC	OK	OK	OK	X	-
MT19937	OK	X	X	OK	OK
Well RNG	OK	?	?		OK

Dynamic Creation

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

Complex dividing for non-overlapping cycle

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

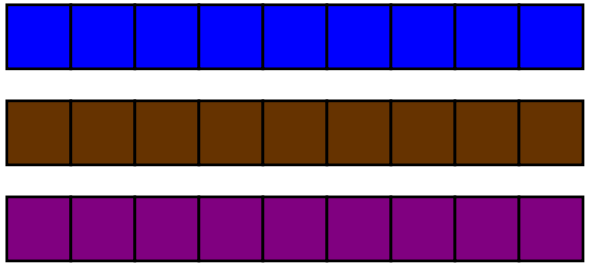


Jump-Ahead Method

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

Divide sequence with GF(2) polynomial

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

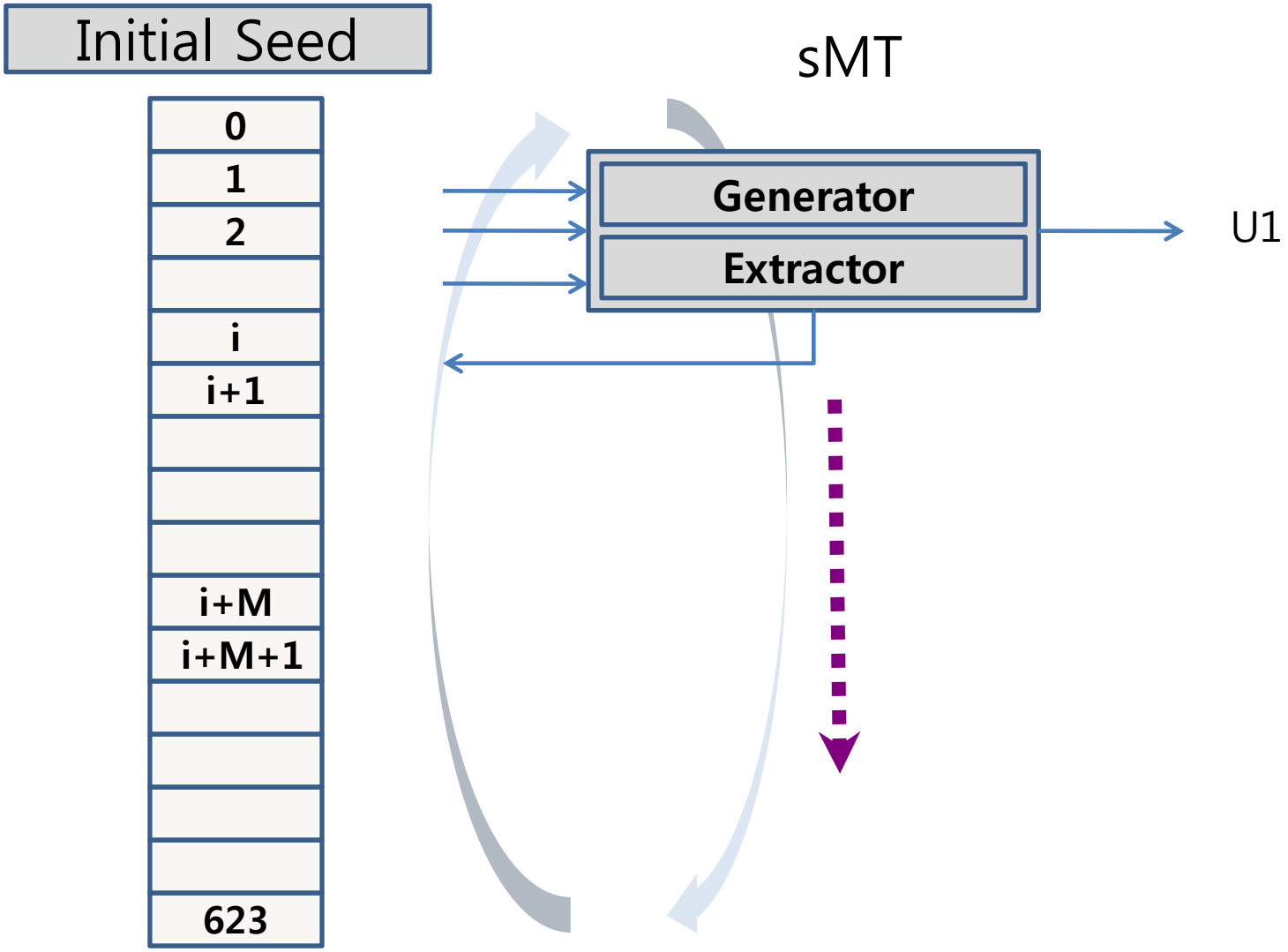


CPU ID 0

CPU ID 1

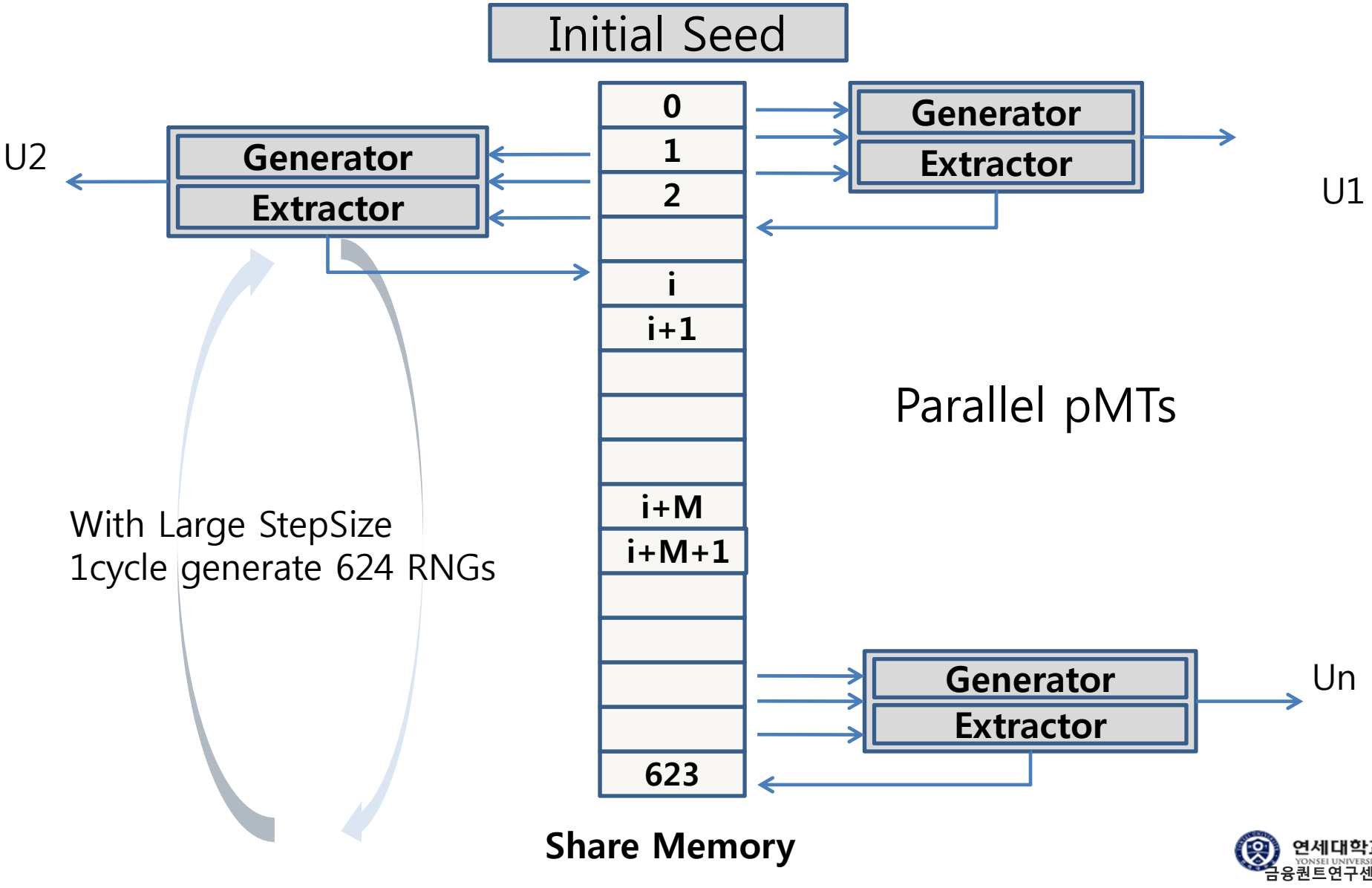
CPU ID 2

sMT architecture of MT19937



MT19937 병렬화

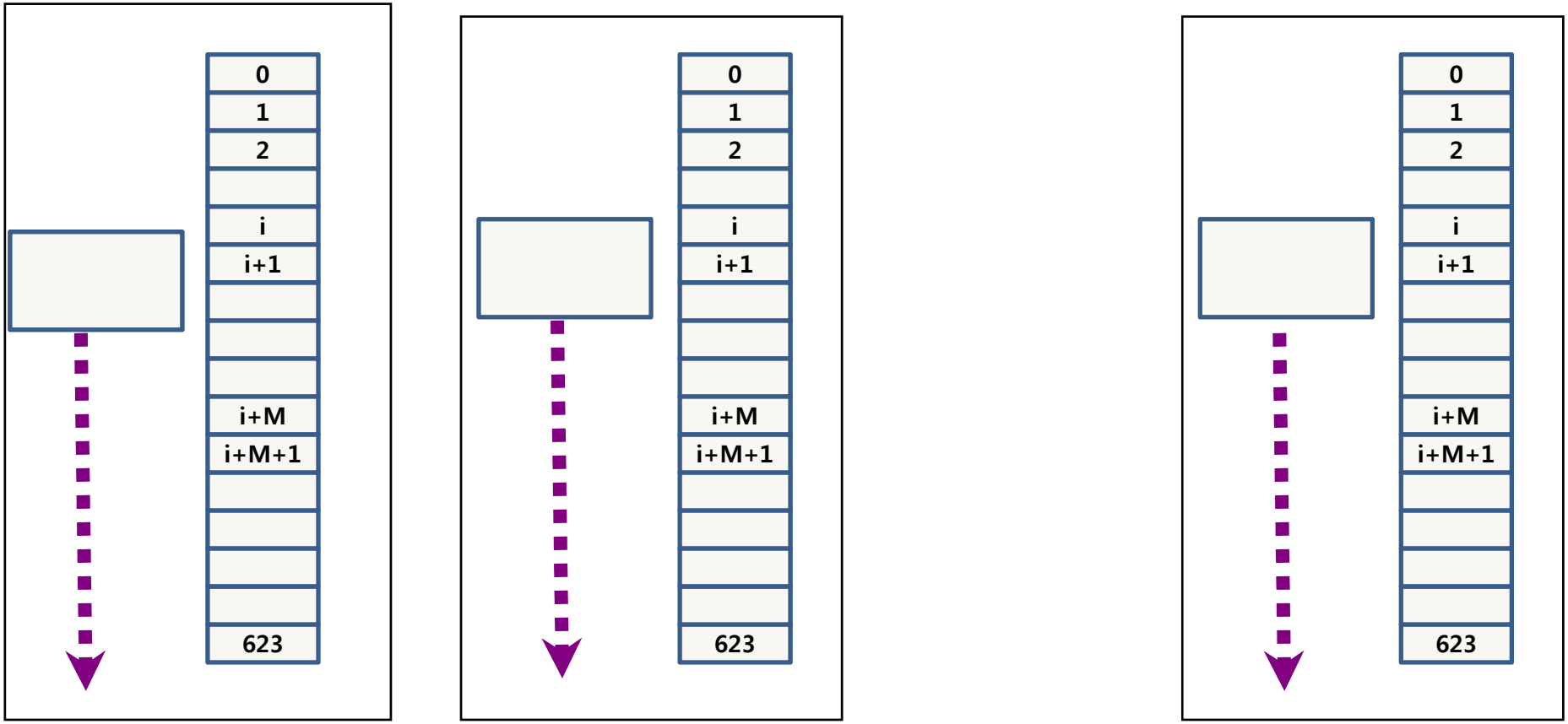
Shared pMT architecture of MT19937



Parallel sMT MT19937

Each Cores execute sMT independently

We will use this model



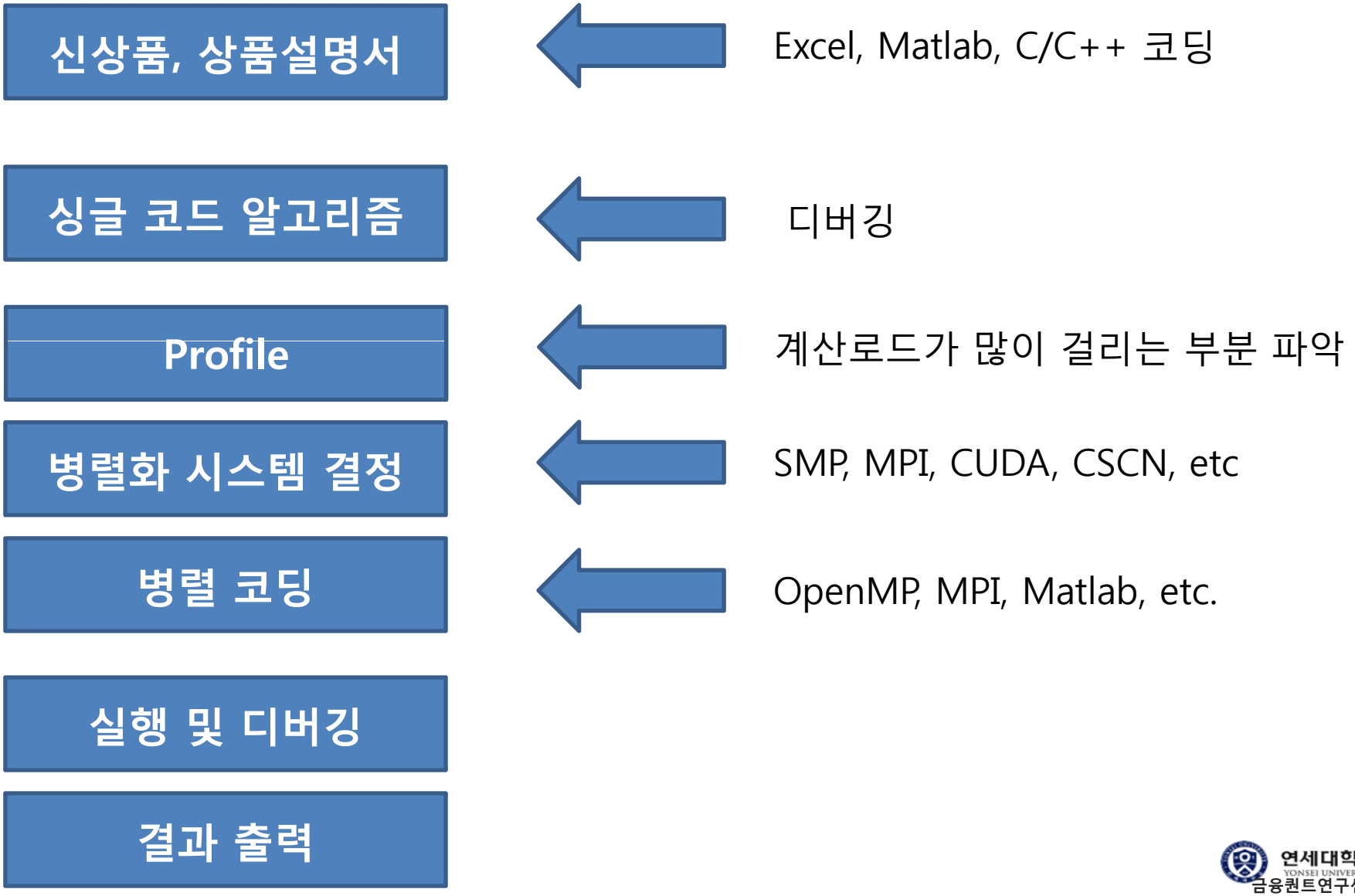
Parallel Quasi RNG

For Loop Nsim simulation
For Loop for T simulation of path

→Method 1 base p에 의한 병렬화
Split Halton with Nsim /16 -- 16 is # of core
Core has own Base **3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59**
Scramble each Sequence with T times

→Method 2 : 준주기성을 고려한 n 병렬화
select large prime such as 23
Compute Split K= Nsim/16 * 24
n=(K*coreID + i)
Parallely Compute Halton(23,n)
Scramble each Sequence with T times

실제 병렬화


신상품, 상품설명서
Excel, Matlab, C/C++ 코딩
싱글 코드 알고리즘
디버깅
Profile
계산로드가 많이 걸리는 부분 파악
병렬화 시스템 결정
SMP, MPI, CUDA, CSCN, etc
병렬 코딩
OpenMP, MPI, Matlab, etc.
실행 및 디버깅
결과 출력

금융문제의 병렬화 기법

사용

방법1
총 M회 시뮬레이션을 병렬처리
각 N개의 Core가 M/N 회 시뮬레이션 실행
각 Core가 독립적으로 Boxmuller 및 RNG 생성
→ 1/N으로 계산 시간 단축
→ 장점 : single code의 90% 이상 그대로 사용

방법2
1. RNG를 병렬화하여 미리 생성
2. N개의 시뮬레이션을 병렬처리
3. N개의 시뮬레이션 평균

→ 조기상환시 RNG 생성 시간 낭비

방법3 Table Technique
RNG를 병렬화하여 미리 생성(1회)
→ memory에 load하여 필요시 사용

각 상품별로 (K회)
M/N회의 시뮬레이션을 병렬처리

→ 많은 상품을 계산해야하는
시스템에서 적합
1/K*N 으로 계산시간 단축
충분한 메모리가 확보 및 분산처리
Memory access bottleneck 발생

모듈화 작업시
single code를 그대로 사용가능

Single Code

병렬 몬테카를로 시뮬레이션 예제

1. **Full-path 유럽형 Vanilla Call 옵션** : reference & benchmark 용
S : 100 , K : 100 , r : 0.03 , v : 0.3, T : 1

2. **ELS pricing** : 삼성증권 1909호 주식연계증권

3년 만기
2 star : POSCO 일반주, S-Oil

12 chance : 디지털 옵션

Double Barrier
Down barrier : 장중체크 (둘 중 하나라도 하락한계)
Up barrier : 매일 종가 체크(두개다 상승한계)
Step-down : 없음, 동일한 조기상환 조건 (Digital Option 형태)

지급 : 조기상환시 +2 영업일

ELS 예제의 경우 실무 경험 부족으로 (본인)

-실무 파트의 현실과 경험 부족으로 실무에서 사용하는 코드와 다를 가능성이 있음

- 이자율, 날짜계산, vol설정 등 파라미터 설정에 대한 노하우 부족

- 실제 업무용 시스템과 연구용 코드의 괴리 가능성

- 실제 포트폴리오에서의 Greeks 관리의 복잡성 등

아직 완성된 프로젝트가 아님

- 체계적 노하우, 코드관리 능력 등 부족

OpenMP를 이용한 병렬화

OpenMP 개념

메모리를 공유한 SMP 머신 (core2 duo 등의 형태)에서 병렬코딩함.
16코어 머신을 이용하면 MC의 경우 약 16배의 속도 향상 기대

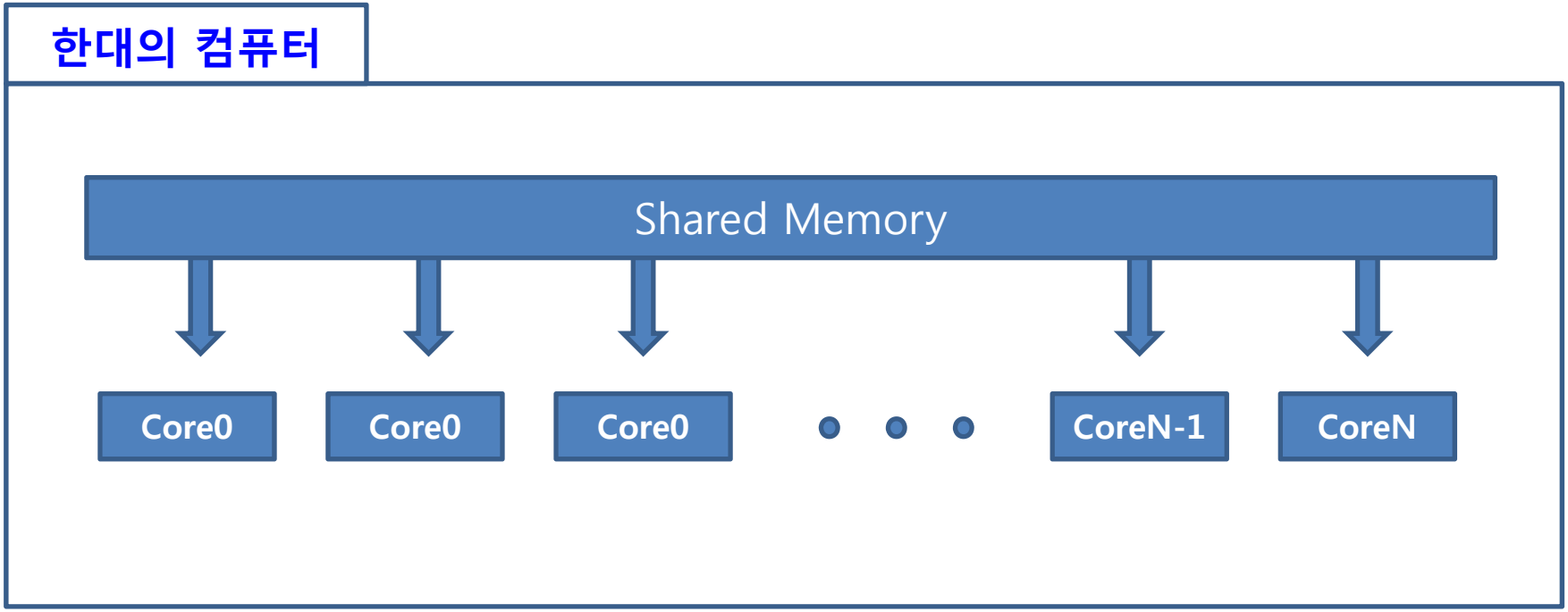
C/C++ 언어와 Fortran 언어를 지원함

Windows 환경은 Visual Studio 2003 이상에서 기본 지원
Unix 환경의 경우 최신 버전 기본 지원
Linux의 경우 GCC 2.4.2 이상에서 지원

OpenMP 명령어를 통해 컴파일러가 자동으로 병렬화

For Loop를 쉽게 병렬화 할 수 있고,
각 Core가 메모리를 공유하기 때문에 병렬 코딩이 매우 편함

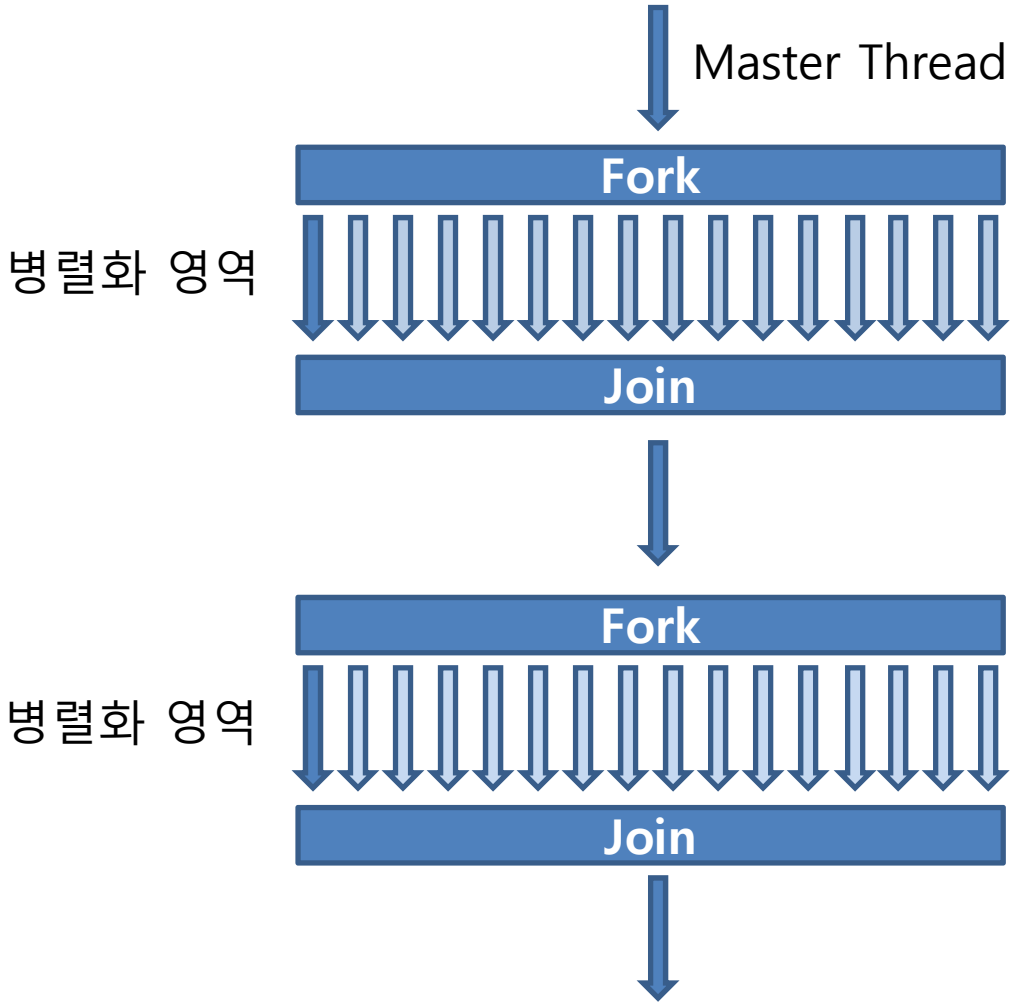
SMP



각 코어가 모두 하나의 메인보드 위에 있기 때문에 모두 같은 메모리를 사용
특별히, 통신 코딩을 할 필요가 없음

➔ OpenMP 코딩의 편리함

OpenMP 개념



OpenMP 병렬화 예약어들

omp_set_num_threads(16);　　총 16개의 thread 사용설정

totalN=omp_get_num_threads;　전체 병렬화 개수 파악

tid=omp_get_thread_num();　각 병렬 프로세서 번호 인식

#pragma omp　지시어

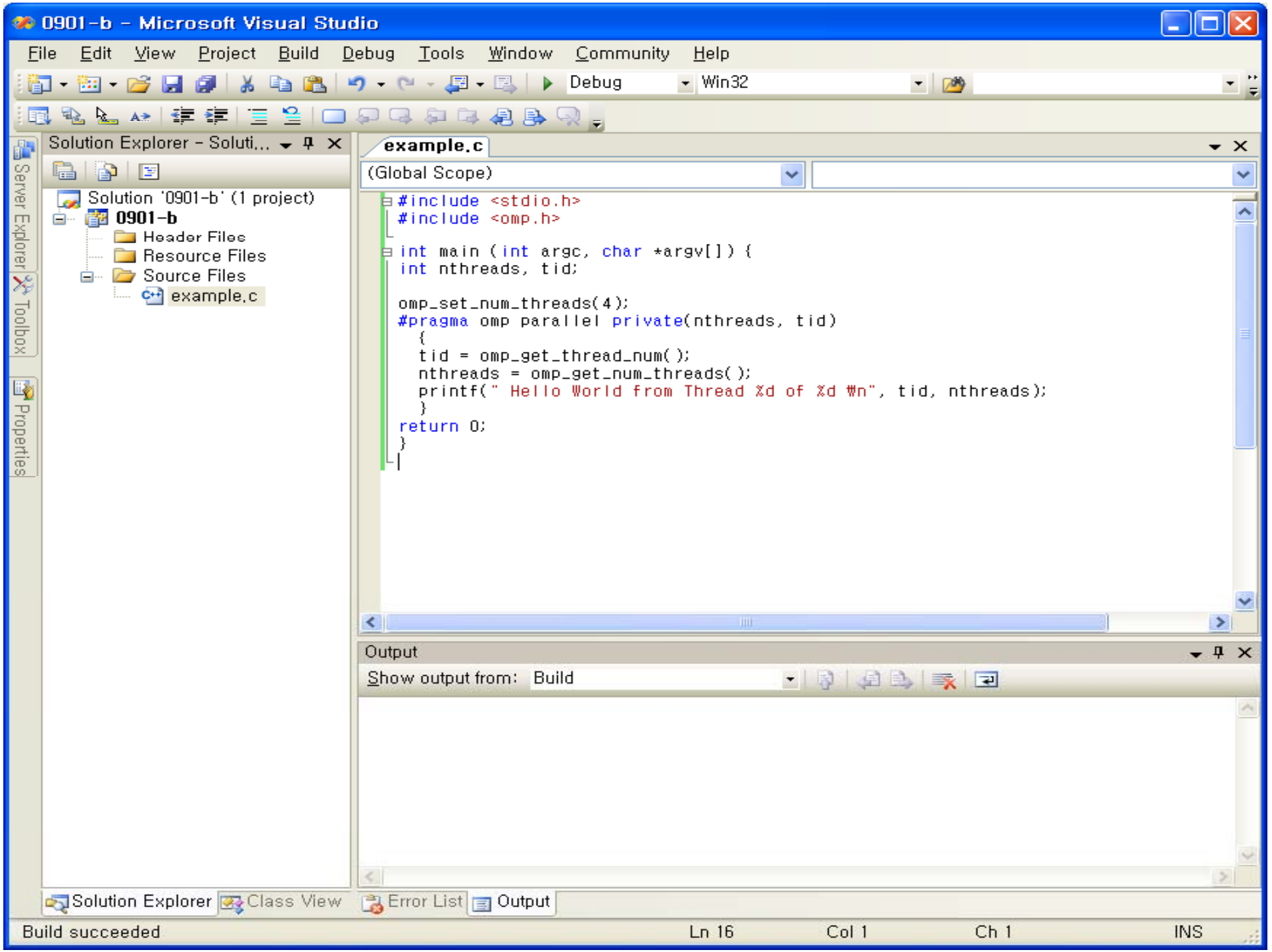
#ifdef_OPENMP　순차 프로그래밍에서도 사용가능하도록 프로그래밍

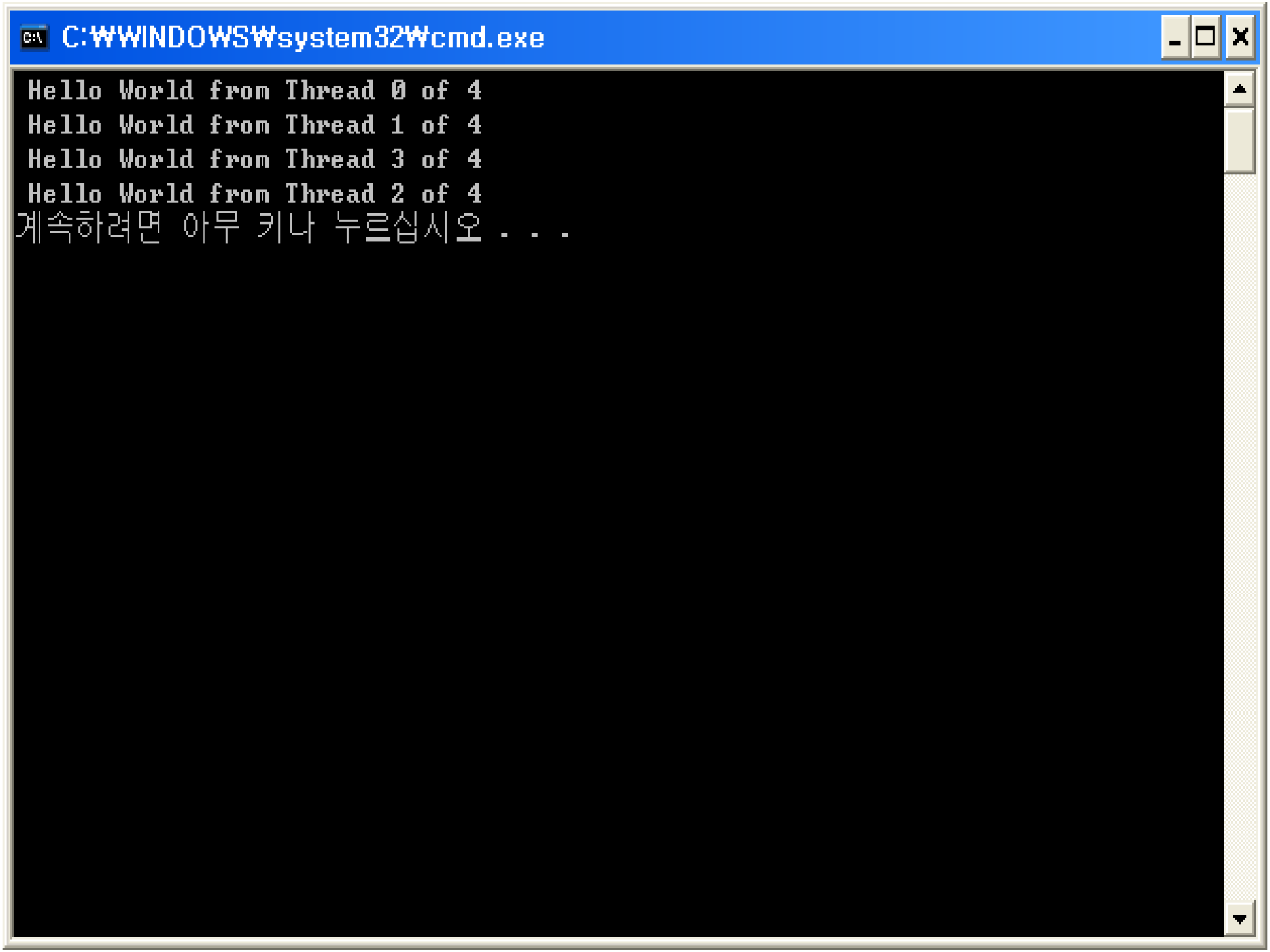
parallel for critical
private() shared() schedule()

OpenMP 예제1

```
#include <stdio.h>
#include <omp.h>

int main (int argc, char *argv[]) {
int nthreads, tid;

omp_set_num_threads(4);

#pragma omp parallel private(nthreads, tid)
  {
  tid = omp_get_thread_num();
  nthreads = omp_get_num_threads();
  printf(" Hello World from Thread %d of %d ₩n", tid, nthreads);
  }
return 0;
}
```

병렬화 영역


0901-b - Microsoft Visual Studio
File Edit View Project Build Debug Tools Window Community Help
Debug
Win32
Solution Explorer - Soluti...
Solution '0901-b' (1 project)
0901-b
Header Files
Resource Files
Source Files
example.c
Server Explorer
Toolbox
Properties
example.c
(Global Scope)
#include <stdio.h>
#include <omp.h>
int main (int argc, char *argv[]) {
int nthreads, tid;
omp_set_num_threads(4);
#pragma omp parallel private(nthreads, tid)
{
tid = omp_get_thread_num();
nthreads = omp_get_num_threads();
printf(" Hello World from Thread %d of %d \n", tid, nthreads);
}
return 0;
}
Output
Show output from: Build
Solution Explorer
Class View
Error List
Output
Build succeeded
Ln 16
Col 1
Ch 1
INS


C:₩WINDOWS₩system32₩cmd.exe
Hello World from Thread 0 of 4
Hello World from Thread 1 of 4
Hello World from Thread 3 of 4
Hello World from Thread 2 of 4
계속하려면 아무 키나 누르십시오 . . .

OpenMP 병렬화 방법

```
for(i=0; i<m; i++)
        {
        a[i] = b[i*n]*c[0];
        for(j=1; j<n; j++)
            a[i] += b[i*n+j]*c[j];
        }
```

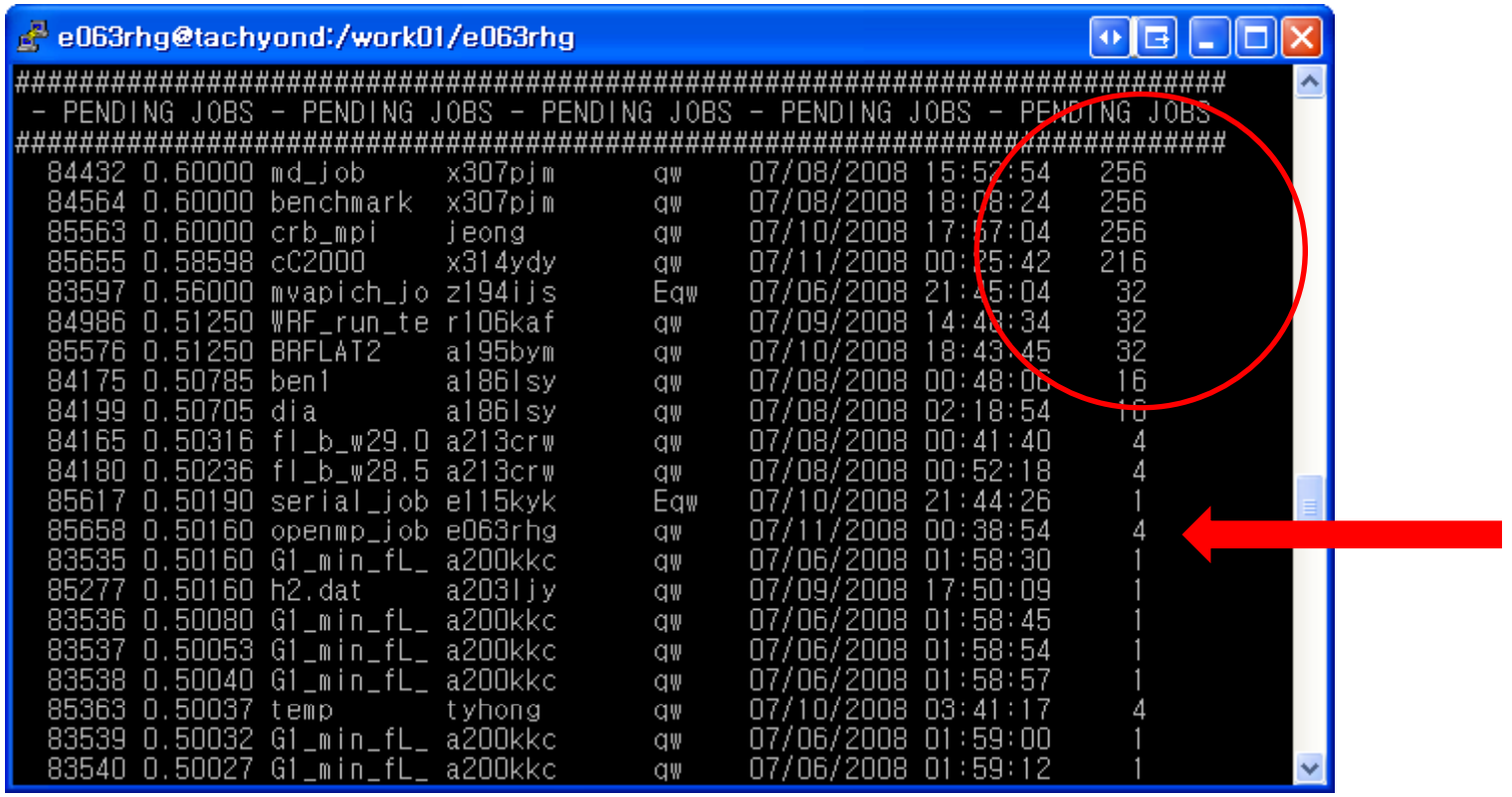

자동 병렬화

```
#include <omp.h>

#pragma opm parallel for shared(m,n)  private(i,j)
    for(i=0; i<m; i++)
        {
        a[i] = b[i*n]*c[0];
        for(j=1; j<n; j++)
            a[i] += b[i*n+j]*c[j];
        }
```

수퍼컴퓨터 Tachyon OpenMP job batch scheduler

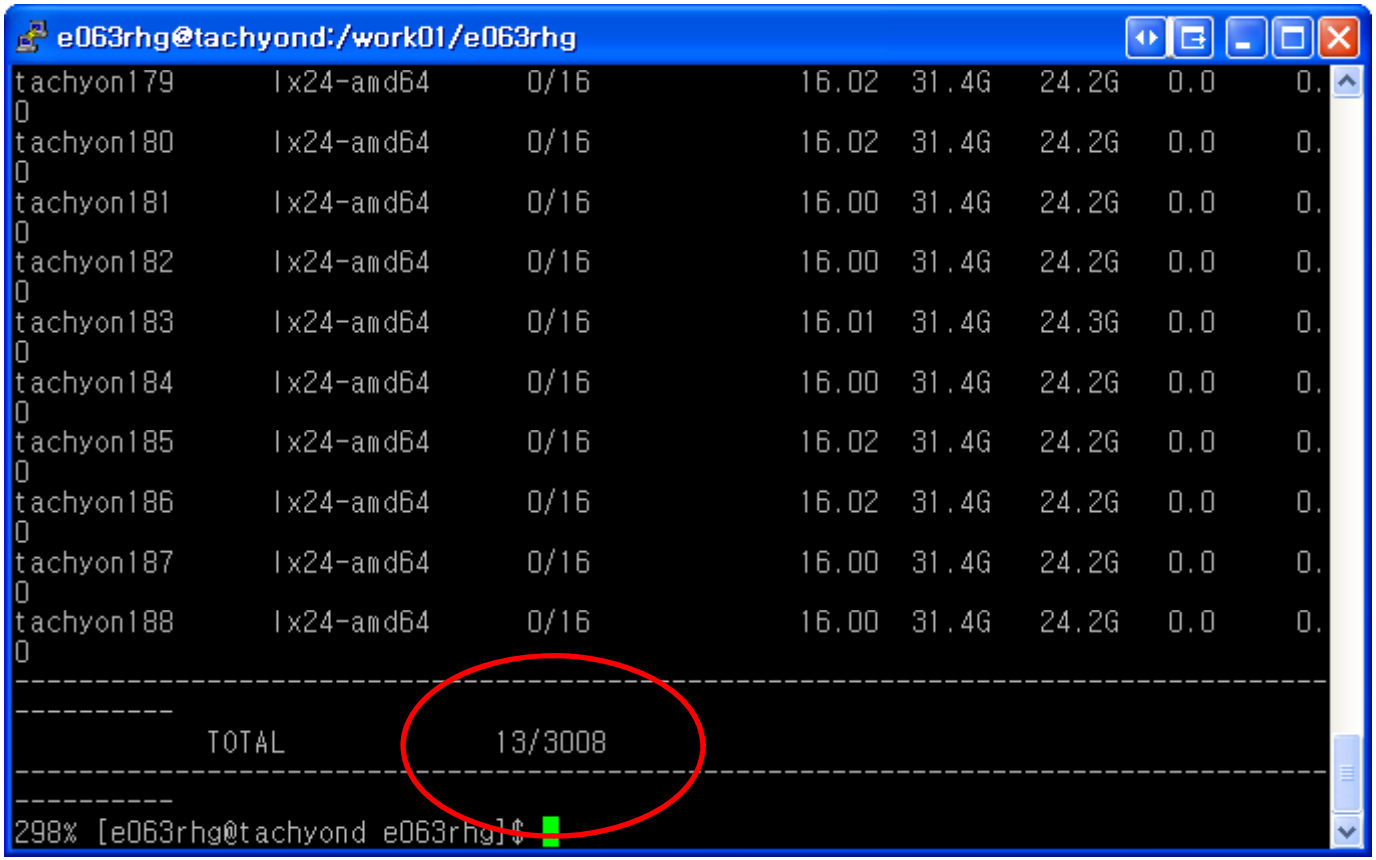
```
282% [e063rhg@tachyond e063rhg]$ vi a.sh
#!/bin/bash
#$ -V
#$ -cwd
#$ -N openmp_job
#$ -pe openmp 4
#$ -q small
#$ -R yes
#$ -wd /work01/e063rhg/
#$ -l h_rt=00:01:00
#$ -M myEmailAddress
#$ -m e
export OMP_NUM_THREADS=4
/work02/e063rhg/omp.exe >result.txt
```

Job scheduler에서 대기중인 사람들



예상실행시간 5sec의 job을 dir 40분 기다려야함
Scheduler 조정을 통해 제일 적게 기다리는 편

Tachyon idle core 현황



6월 30일부터 현재까지 계속 job이 waiting 상태로,
수퍼컴퓨터 사용의 의미가 없음 OpenMP, MPI test 불가

Job Schedule 실행된 모습


```
================================
Black-Scholes : 14.23124493415723357
1
3     -1073749904      566413.67638386972248554     566413.67638386972248554

3     -1073749904      1132306.36511257803067565    565892.68872870830819011

3     -1073749904      1701503.34236495988443494    569196.97725238185375929

3     -1073749904      2254231.80765507556498051    552728.46529011556413025

14.29527750057961200      0.06403256642237842
elapsed time is 44.892820
```

```
#pragma omp parallel shared(fsum, N)  private(tid,fsum_local,NNN)
{
   NNN = 0;
   fsum_local = 0;
   tid = omp_get_thread_num();
   printf("%d \t",omp_get_num_threads());

#pragma omp for
   for ( i=0; i< Nsim ;i++ )
   {
      xt1=s;
      xt2=s;
      for(m=0; m<path; m++)
      {
         xx1 = rand()/(RAND_MAX+1.0);if(xx1==0.0) xx1=0.0000000000001;
         xx2 = rand()/(RAND_MAX+1.0);if(xx2==0.0) xx2=0.0000000000001;
         normal1=sqrt(-2.0*log(xx1 ))*cos(2.0*3.14159265358979323846*xx2 );
         xt1= xt1 + r * xt1 * dt + v*xt1*sqrt(dt)*normal1;
      }
      oprice=Max(xt1-k,0);
      fsum_local =fsum_local+ oprice;
      //printf("%f",op);
   }
#pragma omp barrier
#pragma omp critical (update_sum)
   {
      fsum +=fsum_local;
      stop=clock();
      printf("\n%d \t %20.17f \t %20.17f \n",tid,fsum, fsum_local) ;
   }
} // end of OpenMP
results = (double) 1/Nsim *  exp(-1* r * tau)* fsum;
stop=clock();
htime = 0.001*difftime(stop,start); //windows
printf("\n%19.17f \t %19.17f     %7.5f \n",results,results-bsp,  htime) ;
```

전체 병렬화 영역

Loop 병렬화 부분

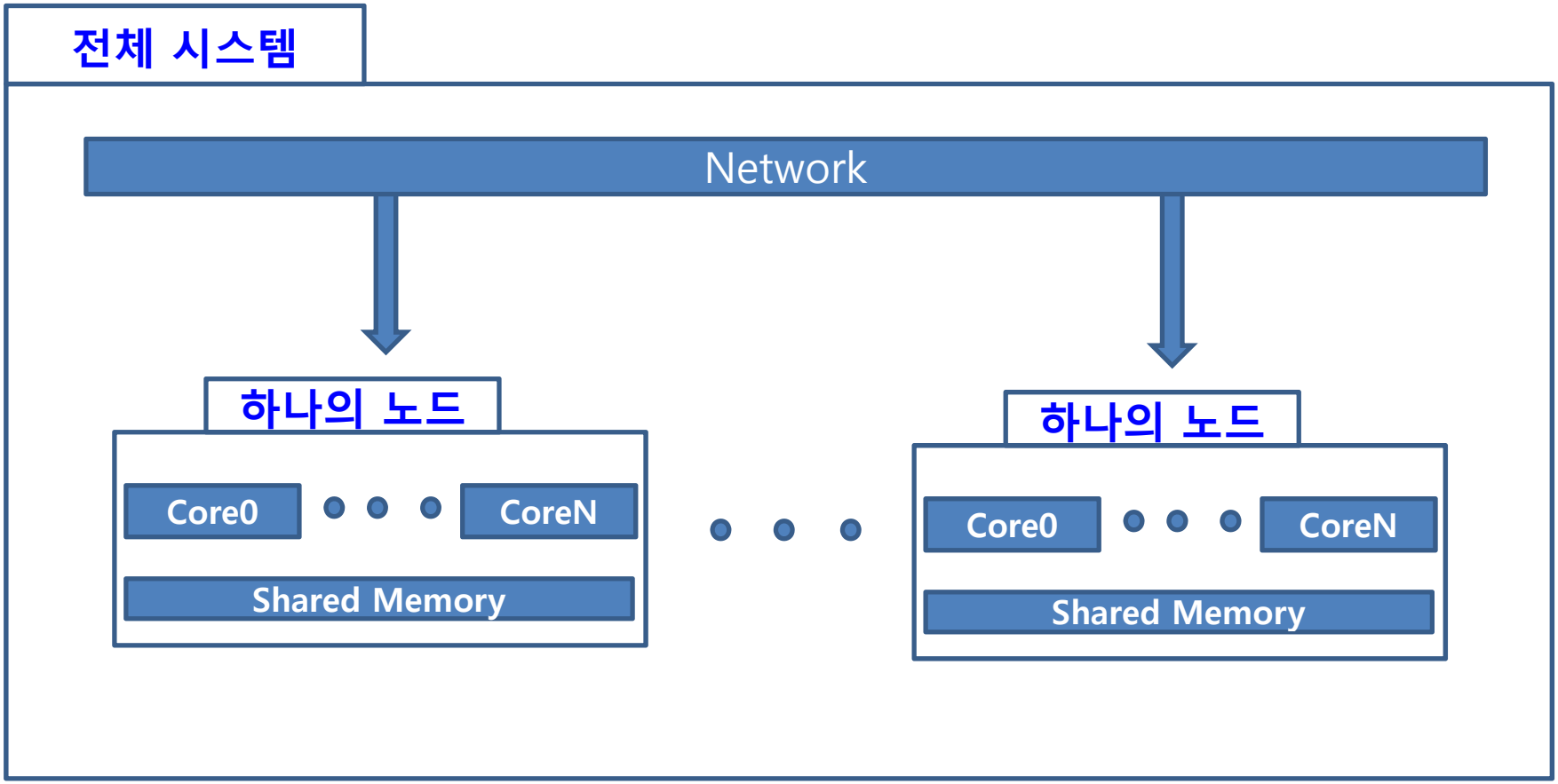
결과 Reduction 부분

OpenMP 결과

1. 수퍼컴퓨터에서는 1~16배까지 core 개수를 늘리면 scailability가 증가하지만 유휴 CPU가 부족하여 scailability를 테스트 하기 어려웠음.

2. CUDAp, MATHp, Windows PC 등에서 1~4 core 기반의 scailability 체크 가능

Cluster



각 노드에서 각각 프로그램을 실행해줘야함 (mpirun)
각 노드의 메모리는 서로 공유하지 않으므로 서로 통신해 줘야함

➔ MPI 코딩의 복잡함

MPI 개념

MPIRUN
MPI로 병렬 프로그래밍된 실행 명령을 각각의 node에 복사하여
각각의 노드가 병렬 명령을 실행할 수 있도록 지원함

MPI setting에 각각의 node 설정을 해주어야함 : 관리의 복잡성 발생

각 Node는 네트웍을 통해 연결되어 있으므로, 서로 메모리를 공유하지 않음.
각 메모리는 서로 독립적으로 작동하므로 PDE solving, Matrix 병렬화 등에서는
서로의 메모리 정보를 서로 통신을 통해 업데이트 해줘야함

MPI는 병렬화 보다는 오히려 통신 프로그래밍을 쉽게 해준 역할을 함.
→OpenMP와는 다르게 프로그래머가 병렬화 작업, 메모리 공유를 직접 해줘야 함

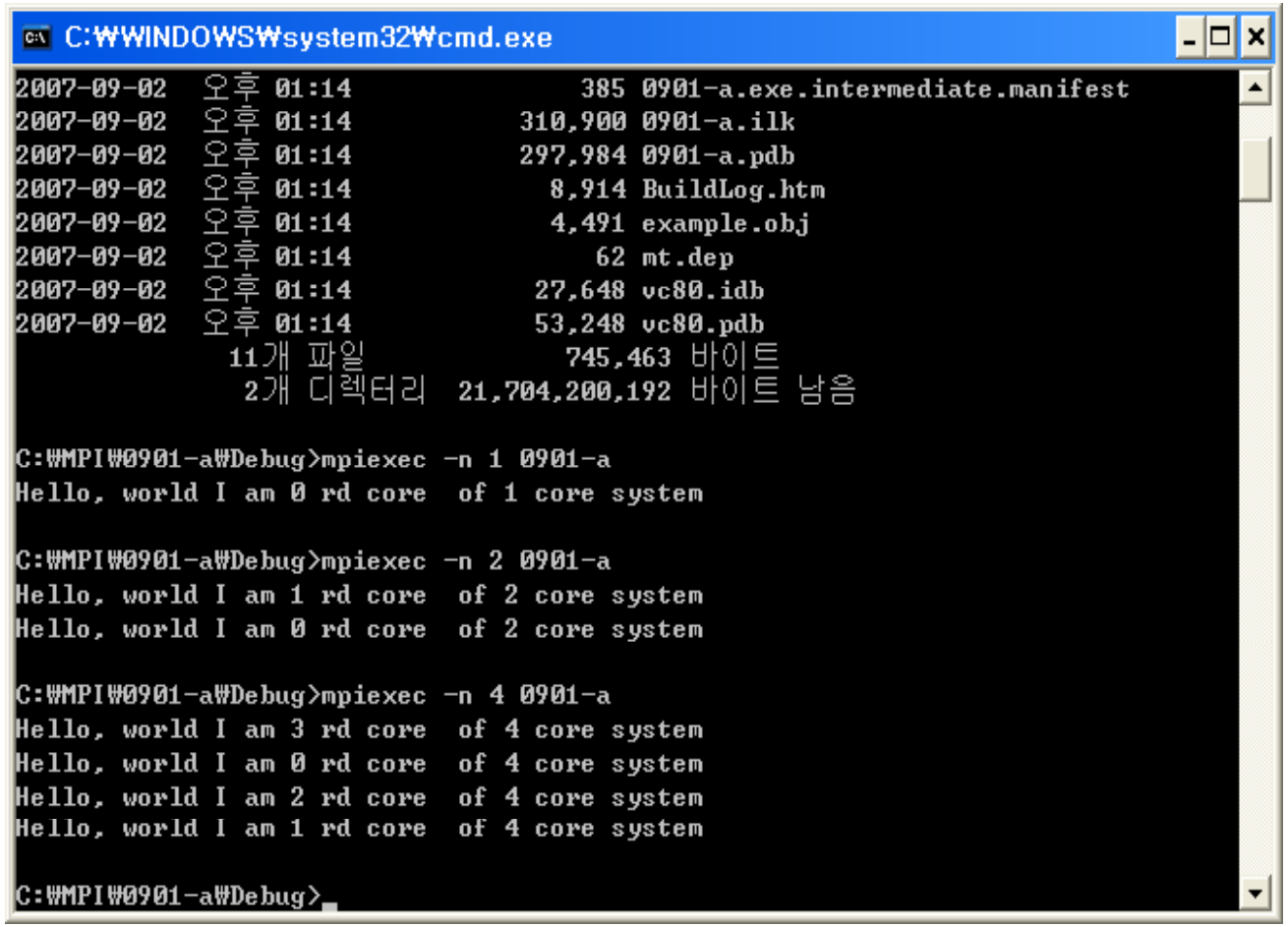
MPIEXEC, POE 등의 MPI Launcher를 통해 실행해줘야한다.
전용서버의 경우 scheduler 대신 plink를 활용하면 유용하다.

MPI 예제1

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[]) {
        int rank, size;

        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        MPI_Comm_size(MPI_COMM_WORLD,&size);

        printf("Hello, world I am %d rd core  of %d core system₩n",rank,size);

        MPI_Finalize();
        return 0;
}
```

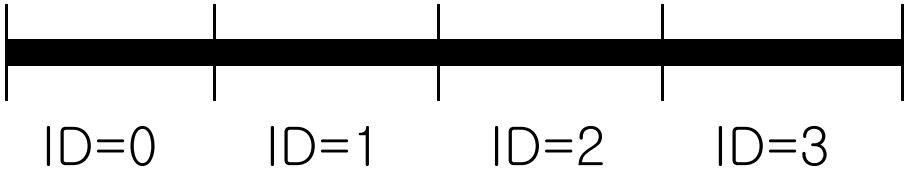
MPIexec

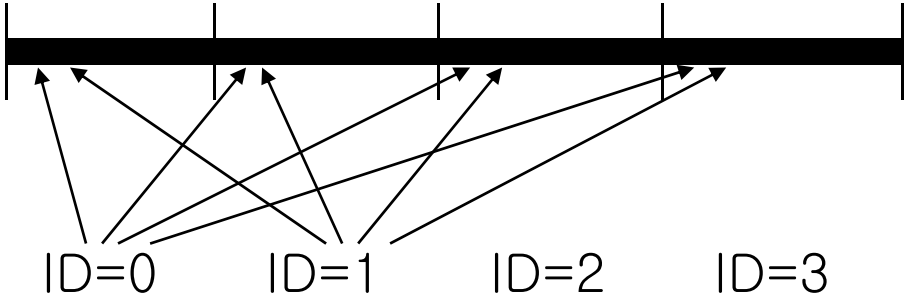
다음은 mpiexec를 통해 core를 1개, 2개 4개로 확장시켰을 때의 실행 결과를 나타내고 있다.



Job 분할 방법

How do we divide job (FOR LOOP) ?





Loop 분할 방법들

단순분할

1

```
for (i = 0; i < Nsim/N; i++) {
        ….
   }
```

Block 분할

2

```
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++) {
        ….
   }
```

순환분할

3

```
for (i = Tid; i < Nsim; i+= N) {
        ….
}
```

블록-순환분할

4

```
for (i = n1*block*myrank;
        i < n2; i+=nprocs*block) {

   for (j = jid; j < min(ij+block-1); i+=n2N) {
        ….
}
```

para_range(n1,n2,n3,n4,n5,n6) 함수

Para_range 함수는 크게 3가지의 병렬화 방법 중 Method 2의 기법으로
For Loop를 균등 분할함

```
void para_range(int lowest, int highest,
                int nprocs, int myrank,
                int *start, int *end) {
    int wk1, wk2;

    wk1 = (highest - lowest + 1) / nprocs;
    wk2 = (highest - lowest + 1) % nprocs;
    *start = myrank * wk1 + lowest + ( (rank<wk2) ? myrank : wk2);
    *end = *start + wk1 - 1;
    if(wk2 > rank) *end = *end + 1;

}
```

$$\sum_{i=1}^{N} a(i) = \sum_{i=1}^{n_1} a(i) + \sum_{i=n_2+1}^{n_2} a(i) + \cdots + \sum_{i=n_{k-1}+1}^{N} a(i)$$

Core 0 Core 1 Core k-1

MPI 예제2 (MPI 통신)

```
#include <mpi.h>
#include <stdio.h>
#define n 100000

void para_range(int, int, int, int, int*, int*);
int min(int, int);

void main (int argc, char *argv[]){
    int i, nprocs, myrank ;
    int ista, iend;
    double a[n], sum, tmp;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    para_range(1, n, nprocs, myrank, &ista, &iend);

    for(i = ista-1; i<iend; i++) a[i] = i+1;
        sum = 0.0;
        for(i = ista-1; i<iend; i++) sum = sum + a[i];
            MPI_Reduce(&sum, &tmp, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
              sum = tmp;
              if(myrank == 0) printf("sum = %f ₩n" , sum);

MPI_Finalize();
}
```

MPI 예제3 Pi계산

```
#include <math.h>
#define n 100000
main(){
    int i,istep,itotal[10],itemp;
    double r, seed, pi, x, y, angle;
    pi = 3.1415926;
    for(i=0;i<10;i++) itotal[i]=0;
    seed = 0.5; srand(seed);
    for(i=0; i<n; i++){
       x = 0.0; y = 0.0;
       for(istep=0;istep<10;istep++){
          r = (double)rand();
          angle = 2.0*pi*r/32768.0;
          x = x + cos(angle);
          y = y + sin(angle);
       }
    itemp = sqrt(x*x + y*y);
    itotal[itemp]=itotal[itemp]+1;
    }

    for(i=0; i<10; i++){
        printf(" %d :", i);
        printf("total=%d₩n",itotal[i]);
    }
}
```

```
#include <mpi.h>
#include <stdio.h>
#include <math.h>
#define n 100000

void para_range(int, int, int, int, int*, int*);
int min(int, int);

main (int argc, char *argv[]){
    int i, istep, itotal[10], iitotal[10], itemp;
    int ista, iend, nprocs, myrank;
    double r, seed, pi, x, y, angle;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    para_range(0, n-1, nprocs, myrank, &ista, &iend);

    pi = 3.1415926;
    for(i=0; i<10; i++) itotal[i] = 0;
       seed = 0.5 + myrank; srand(seed);
       for(i=ista; i<=iend; i++){
          x = 0.0; y = 0.0;
          for(istep=0; istep<10; istep++){
             r = (double)rand();
             angle = 2.0*pi*r/32768.0;
             x = x + cos(angle); y = y + sin(angle);
          }
          itemp = sqrt(x*x + y*y);
          itotal[itemp] = itotal[itemp] + 1;
      }


MPI_Reduce(itotal, iitotal, 10, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    for(i=0; i<10; i++){
        printf(" %d :", i);
        printf(" total = %d₩n",iitotal[i]);
    }
MPI_Finalize();
```

MPI예제4 (Full Path 유렵형 Vanilla Call옵션)

```
for ( i=0; i< Nsim ;i++ )
   {
      xt1=s;
      xt2=s;
      for(m=0; m<path; m++)
      {
         xx1 = rand()/(RAND_MAX+1.0);if(xx1==0.0) xx1=0.0000000000001;
         xx2 = rand()/(RAND_MAX+1.0);if(xx2==0.0) xx2=0.0000000000001;
         normal1=sqrt(-2.0*log(xx1 ))*cos(2.0*3.14159265358979323846*xx2 );
         xt1= xt1 + r * xt1 * dt + v*xt1*sqrt(dt)*normal1;
      }
      oprice=Max(xt1-k,0);
      fsum =fsum+ oprice;
}
results = (double) 1/Nsim *  exp(-1* r * tau)* fsum;
stop=clock();
htime = 0.001*difftime(stop,start); //windows
printf("\n%19.17f \t %19.17f     %7.5f \n",results,results-bsp,  htime) ;
```

Loop분할필요

병렬 RNG필요
multiseed method

MPI reduce 필요

MPI화 코드

```
#include <mpi.h>
#include <stdio.h>
#include <math.h>
#define n 150000
void para_range(int, int, int, int, int*, int*);
int min(int, int);

main (int argc, char *argv[]){
    int i,  m;
    int ista, iend, nprocs, myrank;
    double r, seed, pi, x, y, angle;
    Nsim = n;
    금융관련 변수 설정은 모두 생략함 (single 코드와 동일)

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    para_range(0, Nsim, nprocs, myrank, &ista, &iend);
        seed = 0.5 + myrank; srand(seed);
        for ( i=ista; i< iend ;i++ )
        {
            xt1=s;
            xt2=s;
            for(m=0; m<path; m++)
            {
                xx1 = rand()/(RAND_MAX+1.0);if(xx1==0.0) xx1=0.0000000000001;
                xx2 = rand()/(RAND_MAX+1.0);if(xx2==0.0) xx2=0.0000000000001;
                normal1=sqrt(-2.0*log(xx1 ))*cos(2.0*3.14159265358979323846*xx2 );
                xt1= xt1 + r * xt1 * dt + v*xt1*sqrt(dt)*normal1;
            }
            oprice=Max(xt1-k,0);
            fsum_local =fsum_local+ oprice;
        }
    MPI_Reduce(&fsum_local, &fsum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Finalize();
 results = (double) 1/Nsim *  exp(-1* r * tau)* fsum;
 printf(" Option Price : %23.17f " results);
}
```

병렬화 블록

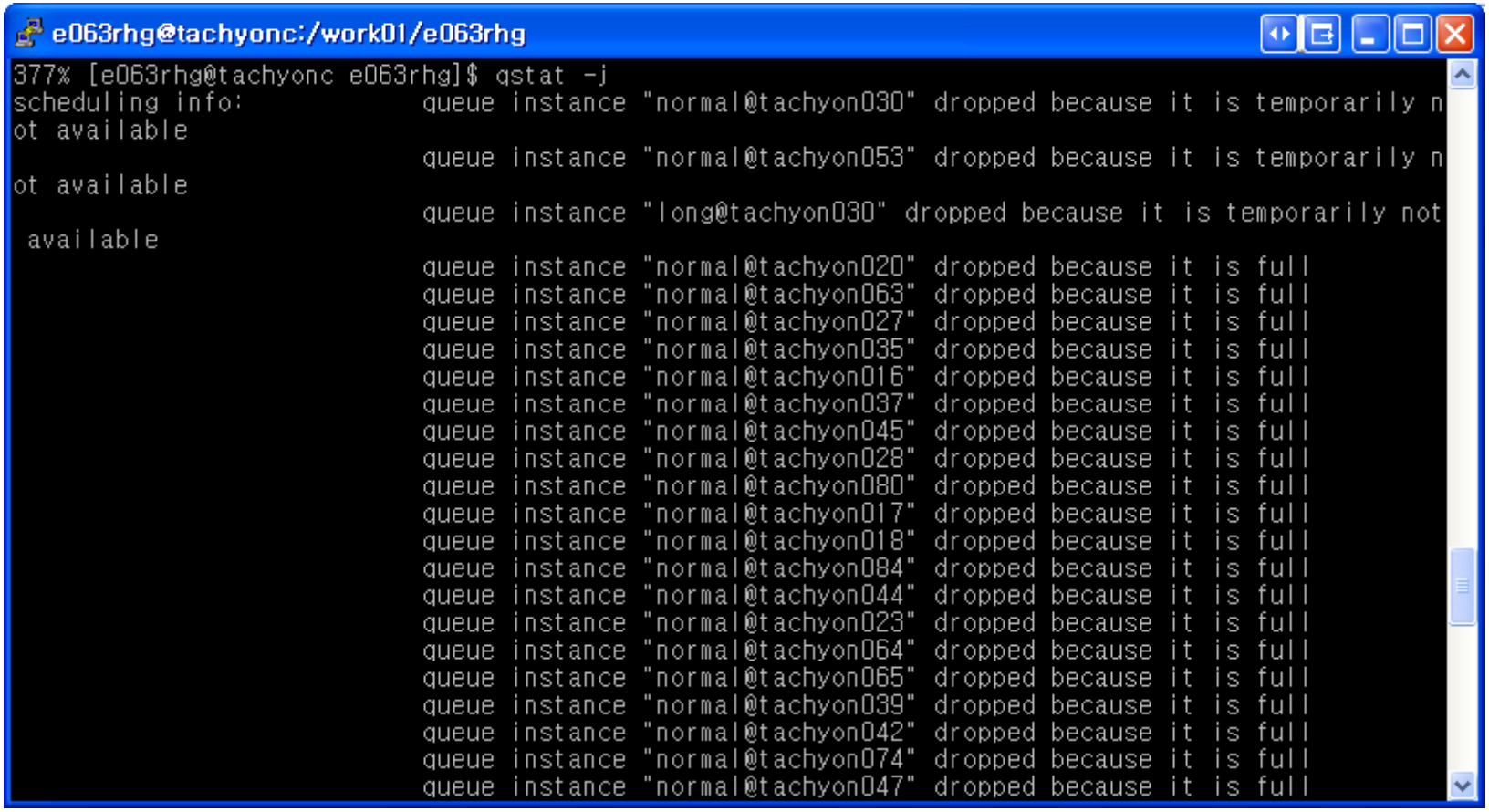
Multiseed방법사용

Single code 그대로 사용

Core간 데이터 통신

상황에 따라
추가적 RNG코딩

Tachyon : all node is full



MPI 결과

수퍼컴퓨터 센터의 tachyon, nobel 서버 모두 busy

➔ 연세대학교 수학과 PDE팀 4 node 8 core server
➔ kisti Hamel Cluster (5년 이상 된 Cluster로 성능이 많이 떨어지만 사용자 적음) 에서 테스트

Linear Scailability : CPU 개수와 속도향상의 선형성이 보장됨

2. Monte Carlo Simulation은 병렬화의 Scailability가 매우 좋은 편임
N개의 Core 사용시 1/N으로 계산시간 단축효과, 1/sqrt(N)의 정확도 향상

현재의 정확도를 유지하면서 속도를 향상시킴 : 1/N

수퍼컴퓨터를 이용한 몬테카를로 병렬화

1. 계산 전용 서버는 자체 구축 필요 (KISTI의 경우 Job schedule : 최소 30분)

2. MPI의 경우 직접 병렬코딩을 하고, MPIRUN을 실행해 줘야 하지만
잘 만들어진 single 코드가 존재하는 경우 몬테카를로 병렬화는 어렵지 않다.

3. 병렬화를 통한 속도 향상
200초 걸리는 시뮬레이션 문제의 경우

2노드	32 core system 구축시 6.3초	0.5억원
8노드	128 core system 구축시 1.6초	2억원 +알파
16노드	256 core system 구축시 0.8초 예상됨	4억원 +알파
32노드	512 core system 구축시 0.4초 예상됨	8억원 +알파

➔ 32노드의 사용시 MC의 편리함과 FDM 수준의 속도를 얻을 수 있음

Parallle Quasi MC를 사용할 경우 8노드 정도로 FDM 수준의 속도향상 예상됨

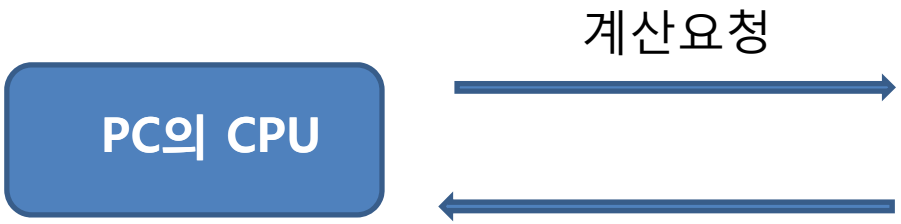
추가적인 결과 (대안적 수퍼컴퓨팅)

성능과 비용을 고려한 병렬컴퓨팅 (그린컴퓨팅)

1. 게임용 그래픽카드를 이용한 병렬MC (CUDA)
   (2008년 6월 13일 확률론 워크샵 발표)

2. PS3 클러스터를 통한 병렬MC (IBM Cell BE chip에 대한 연구로 확대 예정)

3. 가속보드를 통한 연산가속 (Clearspeed CSe620보드를 통한 가속)



게임산업의 발전괘 개발로
그래픽칩이 CPU보다 성능이 높음

CUDAp workstation at Yonsei Math-Finance Lab

계산용 서버 : Fedora Linux + Intel Compiler + CUDA
Tesla C870 3개 장착 : 이론상 SP 1.5Tflops in SP

보조용 워크스테이션 Windows XP + Visual Studio 2003 + CUDA
Tesla C870 1개 장착 : 이론상 SP 0.5Tflops in SP

비교
Kisti 수퍼컴퓨터 4호기 : 전체 : DP 24Tflops(40억 이상),
하나의 노드 : DP 0.1Tflops(2천만원)

Algorithm for ELS

Parameter input

Loop Parallelization
XT simulation
Boxmuller,
RNG(parallel SMT19937)
ELS pricing Routine
IF flag : prepayment, payoff

average

results

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;
}
```

Single & CUDA ELS 실행속도

삼성증권 제1909회 ELS

MT19937 이용 (10만회)

	CPU	병렬화	1차 최적화	현재 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 vs. Tesla C870

Futher Research

1. Hybrid Method with OpenMP, MPI & CUDA

MT19937대신 MT607 이용 시 추가적으로 4배 이상 추가 속도향상 가능 (작업중)
pMC with MultiGPU :

16배 성능의 1node 서버 비용으로 16노드 정도의 200배 이상 속도향상 예상

→ Pseudo RNG를 이용한 MC simulation : 0.45 sec 이내 pricing 가능함을 의미

단점 : C870은 현재 **single precision**만 지원. 차기H/W에서 double precision 지원예정
single precision : 소수점 7자리의 누적오차
double precisoin : 소수점 15자리 누적오차
MC 기법을 이용한 1BP 이하 수준의 에러를 확보하기 위해서는
5만~10만번 정도 double Precision RNG 시뮬레이션이 적당할 것으로 생각됨

➔ 바로 적용 불가 (H/W 출시 : 6개월 이상, S/W 안정화 : 계속 연구 중)
S1070 출시 : double precision 사용시, 2 sec 정도로 예상됨(테스트 예정)

3. Module화 작업
Parallel Quasi Monte, New RNGs, Fast Convergence

4. Module화 작업
 좀더 쉽게 병렬화 응용 :
CUDA의 경우 99%이상 single code 그대로 이용할 수 있도록 작업 중
금융쪽에서 OpenMP 혹은 MPI의 직접 적용 가능성 연구
plink, 소켓통신을 이용한 excel base 원격 MPIRUN 환경 구축
Greek Search, RM 등 금융 시스템의 병렬화 연구

5. 다양한 병렬화 환경 연구
CPU, GPU, 가속보드 등의 병렬처리 노하우
 병렬화 자문 경험 - LG전자연구소 병렬화 작업 자문,
KISTI 수퍼컴퓨터센터 병렬시스템 도입 자문 등
 병렬화 협력 – 이화여대 그래픽연구실, CUDA포럼 등

6. Parallel FDM & FEM 계획 중
Parallel LU, parallel CG solver 연구 및 병렬 PDE solver

The End

경청해 주셔서 감사합니다.

연세대 수학과 박사과정 유현곤
E-mail : yhgon@yonsei.ac.kr

Resume

학력 :

1997.3-2002.2 **연세대학교 상경계열 97학번** (경영학, 경제학 이중전공)

2006.3-2008.2 **연세대학교 수학과 석사**
논문 Option Pricing with Stochastic Volatility under Markovian Regimes Switching

2008.3- 현재 **연세대학교 수학과 박사과정 1년차 재학 중** (약 3년 후 졸업 예정^^)

관심분야 :
금융수학, 금융공학 - Structured derivatives(ELS 등), efficient Greek search, Stochastic Volatility, Parallel Computing (MC, FDM, FEM, Meshless) 등