병렬 계산을 이용한 열방정식 풀기.

1. 처음

병렬 계산을 하기 전에 C언어를 이용하여 명시적 유한 차분법으로 하나의 열방정식을 풀어본다. 먼저 C로 열방정식을 이해한 다음 초기 조건만 다르게 하여 클러스터로 여러 개의 열방정식을 풀어보자.

2. C를 이용한 명시적 유한 차분법으로 열방적식 풀기

열방정식을 풀기 위한 자세한 이론은 앞서 다룬 Finite-Difference method을 보기로 하고 바로 식(1.10)을 적용한 matlabcode heatex.m(page10)를 C코드로 바꾸어 본다.

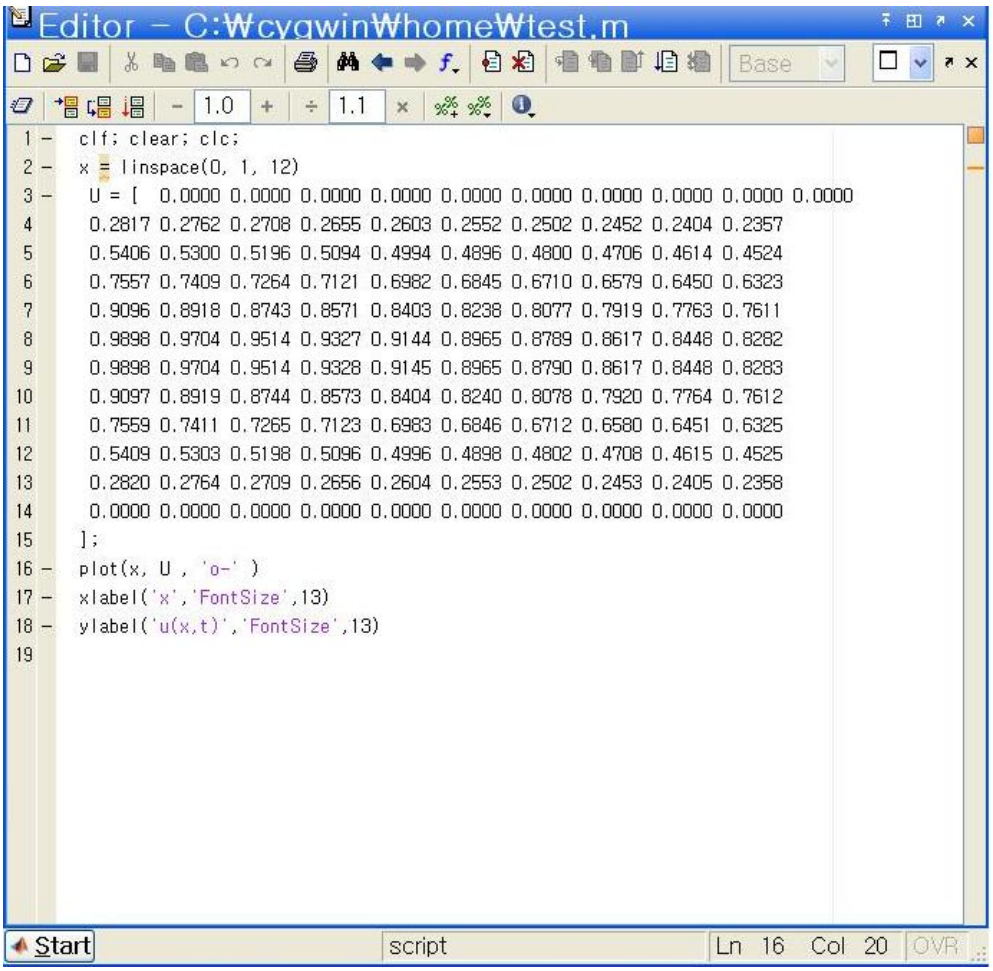
Heat.c

```
#include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 #include <math.h>

 #define N 12
 #define k 0.002
 #define Nt 10
 #define L 1
 #define h 0.0909
 #define pi 3.141596

int main()
 {
int i ,j;

double x ;
double U[N][Nt];

FILE *fp;                                                    /*ⓐ*/
char filename[256];
char buffer[10];
for ( i = 0; i <= N-1 ; i++) {
x = h * i;
U[i][0] = sin(( pi * x ));                                   /* initial condition */
}

for (j=0; j< Nt ; j++)
{
U[11][j]=0;
U[0][j]=0;
}

for ( j = 1; j < Nt ; j++)                                   /*eqn 6.11*/
{
for ( i = 1; i < N -1 ; i++)
U[i][j]=(k/(h*h))*U[i-1][j-1] + (1-2 * (k/(h*h)) )*U[i][j-1] + (k/(h*h))*U[i+1][j-1];
}

fp = fopen("test.m", "w");                                   /*ⓑ, create test.m file*/
```

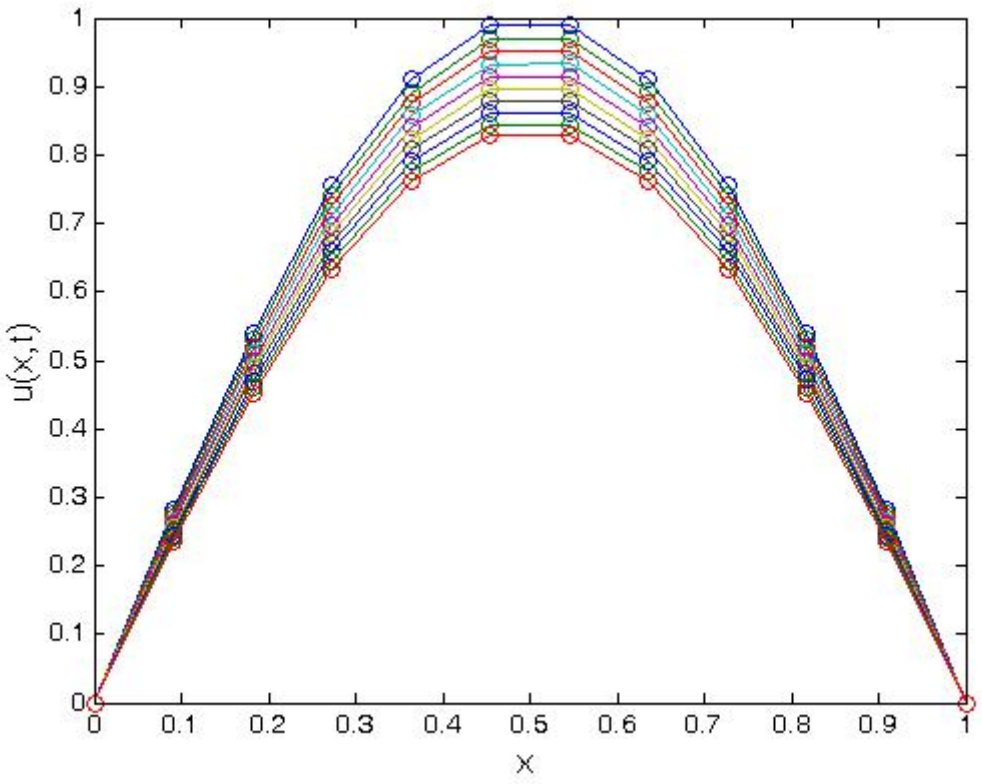
```
fprintf(fp, "clf; clear; clc; \n");
fprintf(fp, "x = linspace(0, %d, %d)\n ", L, N);
fprintf(fp, "U = [ ");

for (i = 0; i <= (N - 1) ; i++)
for (j = 0; j < Nt; j++)
{
fprintf(fp, " %0.4f", U[i][j]);
if (j == Nt-1 )
fprintf(fp, "\n");
}
fprintf(fp, "]; \n");
fprintf(fp, "plot(x, U , 'o-' ) \n");
fprintf(fp, "xlabel('x','FontSize',13) \n");
fprintf(fp, "ylabel('u(x,t)','FontSize',13) \n");
fclose(fp);                                                    /*ⓒ*/
return 0;
 }
```

C코드의 ⓐ FILE* fp 는 파일을 사용하기 위한 파일 포인터이다. FILE 이라고 하는 파일구조체를 가리키는 포인터 변수 fp를 선언하고 ⓑfopen()로 파일을 사용하고 fprintf 함수는 데이터를 서식화하여 파일로 출력한다. 입력이 끝나면 ⓒfclose로 닫아준다.

이 파일을 컴파일하고 실행을 해보면 test.m이라는 matlab파일이 생성되는 데 그 파일을 열어보면 아래와 같이 나온다.



이 matlab코드를 실행(F5)를 하면 아래와 같은 그래프가 나온다.



3. 병렬 계산하기

이제, 각 core별로 다른 초기조건을 가지는 열방적식을 풀어보자.

먼저 C코드를 보자.

parallelheat.c

```
#include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 #include <math.h>
 #include "mpi.h"

 #define N 12
 #define k 0.002
 #define Nt 10
 #define L 1
 #define h 0.0909
 #define pi 3.141596

 int main(int argc, char **argv)
 {
int i ,j, t;
int p, rank;
double x ;
double U[N][Nt+1];
FILE *fp;
MPI_Init(&argc, &argv);
```

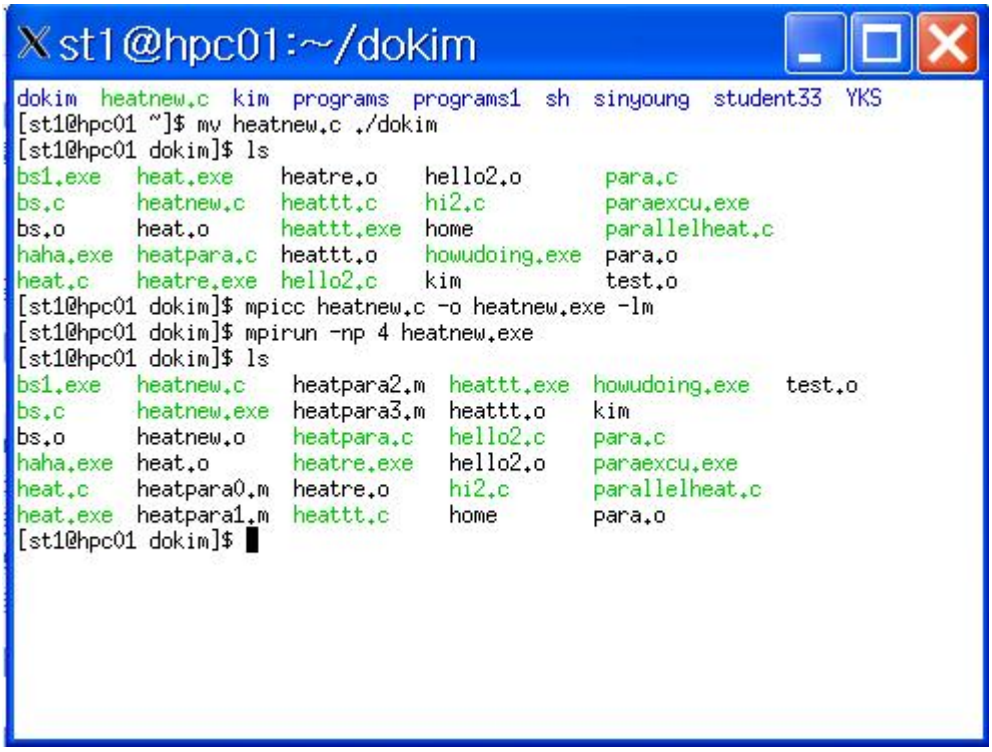
```
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
char filename[256];
char buffer[10];

for ( i = 0; i <= N-1 ; i++)
{
x = h * i;
U[i][0] = sin(( pi * x ))* rank ;
U[i][10]=0;
}
for (j=0; j<= Nt ; j++)
{
U[11][j]=0;
U[0][j]=0;
}
for ( j = 1; j < Nt ; j++)
for ( i = 1; i < N -1 ; i++)
U[i][j]=(k/(h*h))*U[i-1][j-1] + (1-2 * (k/(h*h)) )*U[i][j-1] + (k/(h*h))*U[i+1][j-1];
strcpy(filename, "heat");
        sprintf(buffer, "%d", rank);
        strcat(filename, buffer);
        strcat(filename, ".m");
        fp = fopen(filename, "wt");

fprintf(fp, "clf; clear; clc; \n");
fprintf(fp, "x = linspace(0, %d, %d)\n ", L, N);
fprintf(fp, "U = [ ");
for (i = 0; i <= (N - 1) ; i++)
for (j = 0; j < Nt; j++)
{
fprintf(fp, " %0.5f", U[i][j]);
if (j == Nt )
fprintf(fp, "\n");
}
fprintf(fp, "]; \n");
fprintf(fp, "plot(x, U , 'o-' ) \n");
fprintf(fp, "xlabel('x','FontSize',13) \n");
fprintf(fp, "ylabel('u(x,t)','FontSize',13) \n");
fclose(fp);
return 0;
 }
```

병렬 계산을 위해 앞의 C코드 heat.c에서 전처리기 #include "mpi.h"를 추가해주고 int main을 int main(int argc, char **argv)으로 바꾸어 주자. 그리고 프로세서의 수 변수 p와 프로세서의 고유번호 rank를 선언해주고 초기값을 다르게 해주기 위해 본래 초기값에 rank를 곱한 값을 각 각의 열방정식의 초기값으로 해둔다. 그러면 병렬계산을 위한 코드는 완성되었다. 다만 출력되는 파일을 구별하기 위해 strcpy와 strcat를 이용하여 출력되어 나오는 파일이름에 rank값을 붙여 준다.

4. 실행하기

앞의 병렬 계산을 하는 방법과 같은 방법으로 한다. "scp 파일명.c st1@163.152.62.225:"이라고

써서 파일을 클러스터에 올리고 mpicc로 컴파일한 후 mpirun으로 실행하면 된다.



mpirun –np 4 heatnew.exe를 클러스터에 실행시킨 결과
heatpara0.m heatpara1.m heatpara2.m heatpara3.m의 matlab파일이 생성되었다.

Parallel programming with MPI

1. 개요.

MPI_Send와 MPI_Recv를 사용해서 병렬 계산을 해본다.

2. 파일 올리기

병렬 계산을 하기 위해서는 파일이 클러스터에 올라가 있어야 한다. 파일을 올리는 방법은 시그윈에서 "scp 파일명.c st1@163.152.62.225:"이라고 쓰면 파일이 클러스터에 올라가게 된다.

3. 컴파일하기

"st1@hpc01:~]$"에서 "mpicc 대상파일이름.c –o 컴파일된파일이름.exe"이라 입력하면 된다.

4. 실행하기

컴파일이 되었으면 "mpirun –np 24 컴파일된파일.exe"을 입력하여 실행한다.
Mpirun –np [프로세서의 수를 쓴다. (숫자,예 4나 24)]

5. 예시 파일

각 process가 병렬 연산하여 결과 값을 보여주는 예를 보자.

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

main(int argc, char* argv[]){
int my_rank;
int p;
int source;
int dest;
int tag=0;
char message[100];
MPI_Status status;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);

if (my_rank !=0){
sprintf(message,"greeting from process %d!",my_rank);
dest=0;

MPI_Send(message,strlen(message)+1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
} else {
for (source =1; source < p; source++) {
        MPI_Recv(message,100, MPI_CHAR, source, tag, MPI_COMM_WORLD, &status);
printf("%s₩n", message);
}
}
MPI_Finalize();
-----------------------------------------------------------
```

#include <string.h> 는 strlen을 사용하기 썼다.
#include "mpi.h" MPI를 쓰기 위해서는 반드시 mpi.h을 기입하자.
int my_rank;는 MPI는 process를 구별하기 위해 각각의 process에 rank라는 고유한(unique) integer를 갖는다. rank값은 0번부터 n-1까지 갖는다.
int p; processes의 개수를 알려준다.
int source;는 send의 rank을 의미한다. Source는 메시지를 보낸 process의 identification을 위한 것이다.

int dest; 받는 곳의 rank이다. (destination)
int tag=0; sender가 보내는 tag을 receiver가 같은 tag로 받게 된다. Tag는 올바르게 받은 지 확인하기 위한 꼬리표로써 MPI_Send와 MPI_Recv의 tag가 같아야 한다.

char message[100]; 메시지를 입력 받는다.

MPI_Init(&argc,&argv); mpi를 시작하게 하는데 다른 어떤 함수보다 먼저 호출되어야 하고 한 번만 호출되어야 한다.

MPI_Comm_rank(MPI_COMM_WORLD, &my_rank); commucator에서 rank를 알려준다. (Find process rank)
MPI_Comm_size(MPI_COMM_WORLD, &p); commucator내의 processes의 수를 알려준다.
MPI_COMM_WORLD는 communicator이다. processor간에 메시지를 주고 받을 수 있는 process들의 집합이다. Default 값은 MPI_COMM_WORLD이다. 간단한 프로그램에서는 MPI_COMM_WORLD도 충분하다.]

if (my_rank !=0){
sprintf(message,"greeting from process %d!",my_rank);
dest=0;
rank가 0이 아니면 그 rank값의 message에 greeting from process을 할당하고 %d에 rank값을 넣는다.

MPI_Send(ⓐmessage, ⓑstrlen(message)+1, ⓒMPI_CHAR, ⓓdest, ⓔtag, ⓕMPI_COMM_WORLD);
ⓐ: Address of send buffer
ⓑ: Number of items to send
ⓒ: datatype MPI_는 잊지 말고 붙인다.
ⓓ rank의 destination (도착지)
ⓔ send의 꼬리표
ⓕ communicator이다.

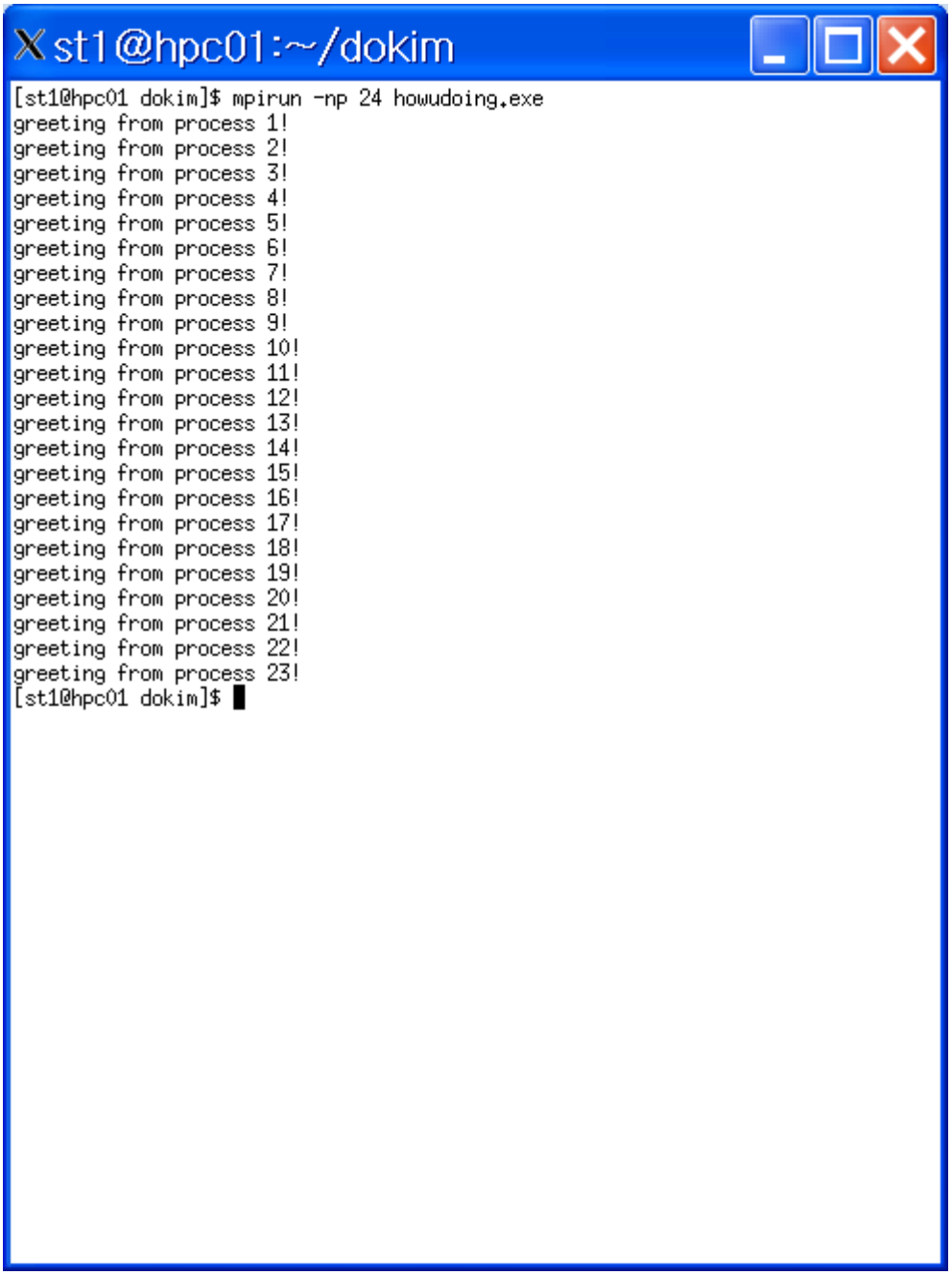
for (source =1; source < p; source++) {
MPI_Recv(ⓐmessageⓑ,100, ⓒMPI_CHAR, ⓓsource, ⓔtag, ⓕMPI_COMM_WORLD, ⓖ&status); printf("%s₩n", message);
클러스터는 계산을 임의로 하게 되어서 source 순서로 출력하는 과정이다.
ⓐ: Address of receive buffer
ⓑ: Maximum Number of items to receive
ⓒ: datatype이다.
ⓓ rank의 source (출발지)
ⓔ Message의 꼬리표이다.

ⓕ communicator

ⓖ status after operation이다.

MPI_Finalize(); mpi를 종료시킨다.

6. 아래는 이 파일을 실행 시켰을 때 나오는 결과이다.



20DEC08

contact: donsen2@hotmail.com