Parallel Programming using MPICH2

차 례

1. Parallel Programming 환경 구축 - 1
2. Parallel Programming using MPICH2 - 13
3. 병렬처리 예제 프로그램 - 22
4. 참고자료 - 23

1. Parallel Programming 환경 구축

병렬 프로그래밍을 위하여 널리 사용되는 Message Programming Interface인 MPICH2를 Windows에서 사용하기 위하여 아래와 같이 환경설정을 하고, MPICH2를 설치한다.

MPICH2가 하나의 문제를 여러 컴퓨터로 처리하게 하기 위해서는 아래와 같은 환경이 필요하다.

작업에 참여하는 총 n대의 컴퓨터는

1. 동일 한 작업 그룹에 속해 있음
2. 동일 한 계정이름과 비밀번호를 사용
3. 방화벽이 없어야 한다.
4. MPICH2가 설치되어 있다.

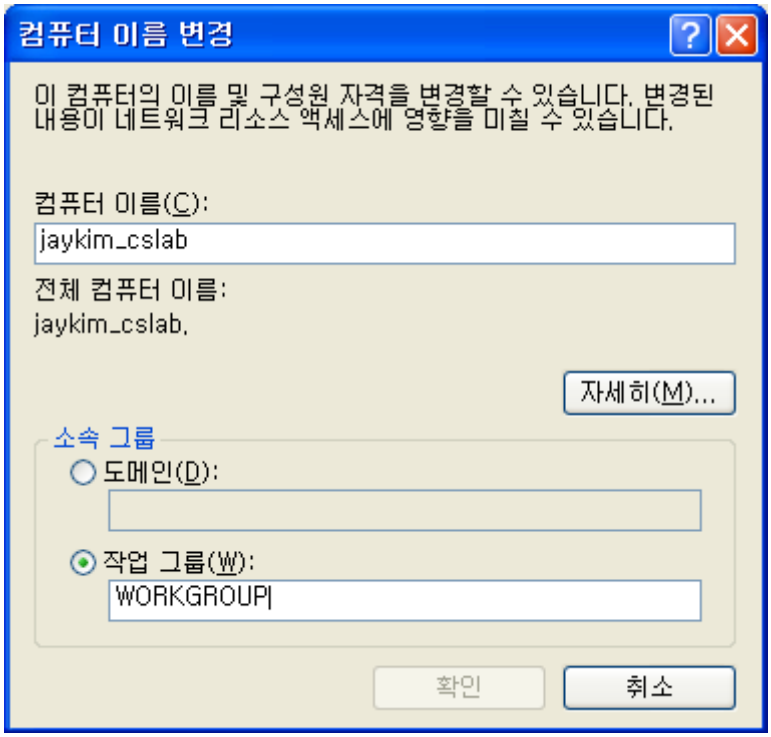
a. Windows 작업그룹 설정

MPICH2를 이용하여 여러 대의 컴퓨터를 하나의 시스템처럼 사용하기 위해서는 각 컴퓨터들은 동일한 작업그룹에 속해 있어야 한다. 아래 메뉴를 통하여 작업에 참여할 모든 컴퓨터들의 작업그룹을 동일한 이름으로 설정한다.

[참고] *MPICH2를 설치한 후 작업에 참여할 컴퓨터들을 설정할 때 이 곳에 설정되어 있는 컴퓨터 이름을 이용하기 때문에 가급적이면 컴퓨터 이름을 식별하기 쉬운 이름으로 설정하는 것이 좋다.*

아래 메뉴를 따라 작업그룹과 컴퓨터 이름을 설정하자.

"시작-> 제어판 -> 네트워크 연결 -> 고급 -> 네트워크 식별 -> 변경"



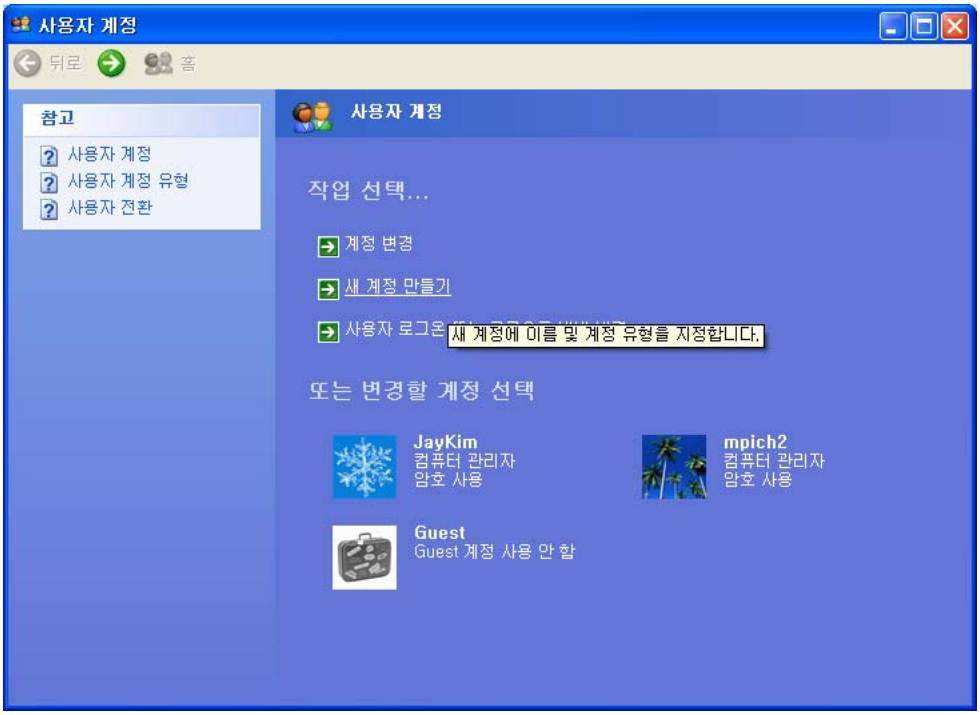
<작업 그룹, 컴퓨터 이름 설정 화면>

예를 들어 PC1, PC2, PC3, PC4, PC5의 이름을 가진 다섯 대의 컴퓨터를 동일한 작업 그룹으로 설정 했다면 "내 네트워크 환경 -> 작업 그룹 컴퓨터 보기" 에서 각각의 컴퓨터들의 컴퓨터 이름을 확인할 수 있어야 정상적으로 작업 그룹이 설정된 것이다.

b. Windows 계정 생성

MPICH2를 이용한 병렬처리 작업에 참여할 모든 컴퓨터는 동일한 계정과 비밀번호를 사용해야 한다. 예를 들면, MPICH2의 작업에 참여하는 5대의 컴퓨터 PC1~PC5는 "계정 : mpich2, 비밀번호 : campich2"와 같이 동일한 계정과 비밀번호를 사용해야 한다. 아래 메뉴에서 MPICH2를 위한 새로운 계정과 비밀번호를 생성한다.

"시작 -> 제어판 -> 사용자 계정 -> 새 계정 만들기"



<MPICH2의 사용을 위한 사용자 계정 생성>

[주의] 새로 생성한 계정은 필수적으로 비밀번호를 사용해야 한다. 비밀번호를 사용하지 않는 컴퓨터는 MPICH2에서 사용될 수 없다.

c. MPICH2 설치

이제 1.b 에서 새로 생성한 계정에 로그인 하여 아래 주소의 웹사이트에 접속하여 윈도우용 MPICH2 프로그램을 다운받고 설치하자.

MPICH2 웹사이트

http://www.mcs.anl.gov/research/projects/mpich2/

MPICH2 다운로드

http://www.mcs.anl.gov/research/projects/mpich2/downloads/tarballs/mpich2-1.0.7-win32-ia32.msi

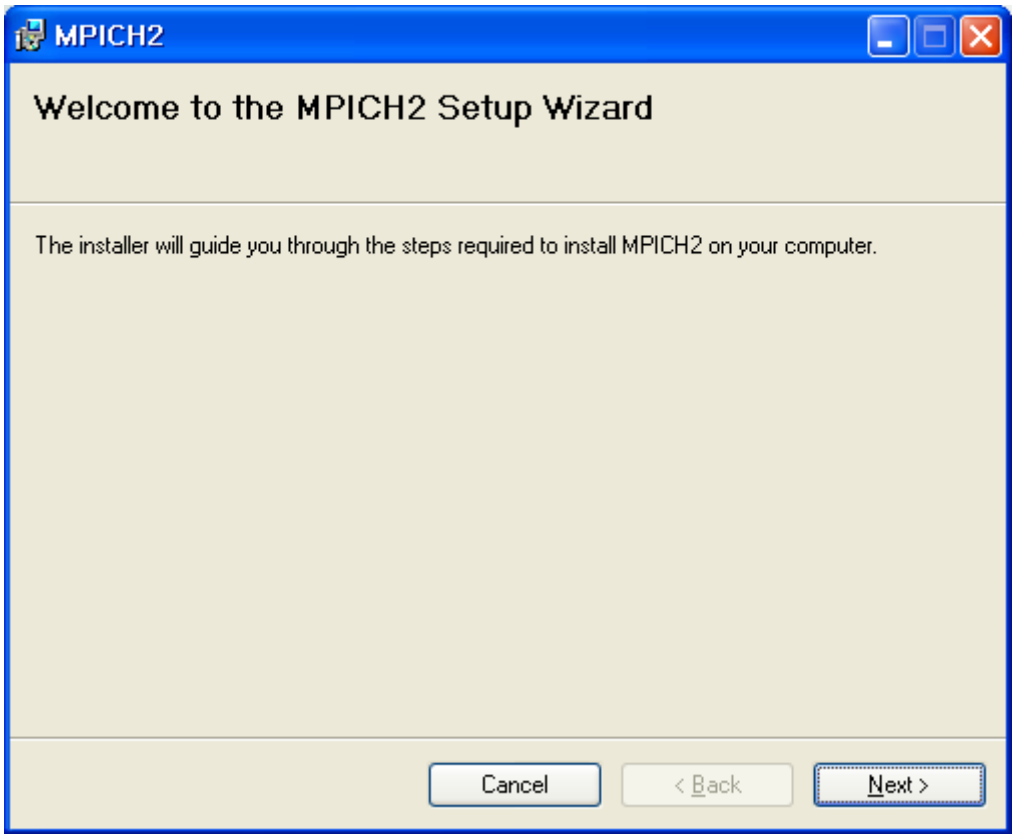
MPICH2가 정상적으로 설치되기 위해서는 아래 두 프로그램이 미리 설치되어 있어야 한다.

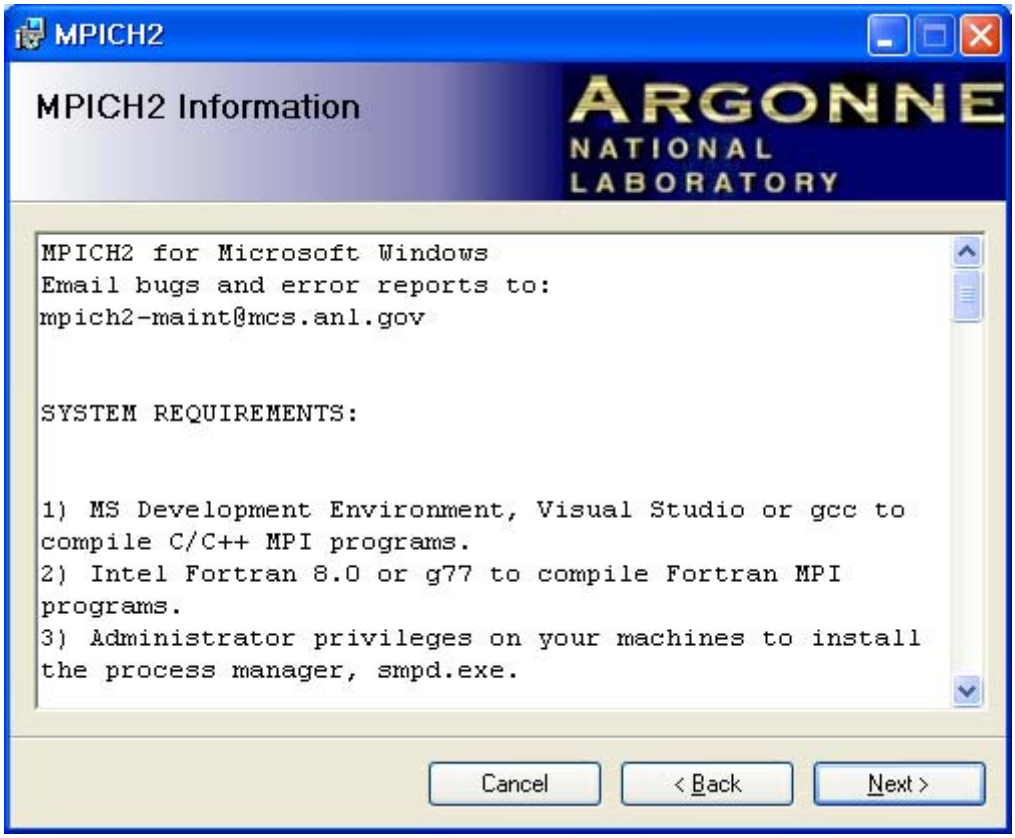
-Microsoft Visual C++ 2005 Redistributable Package (x86)

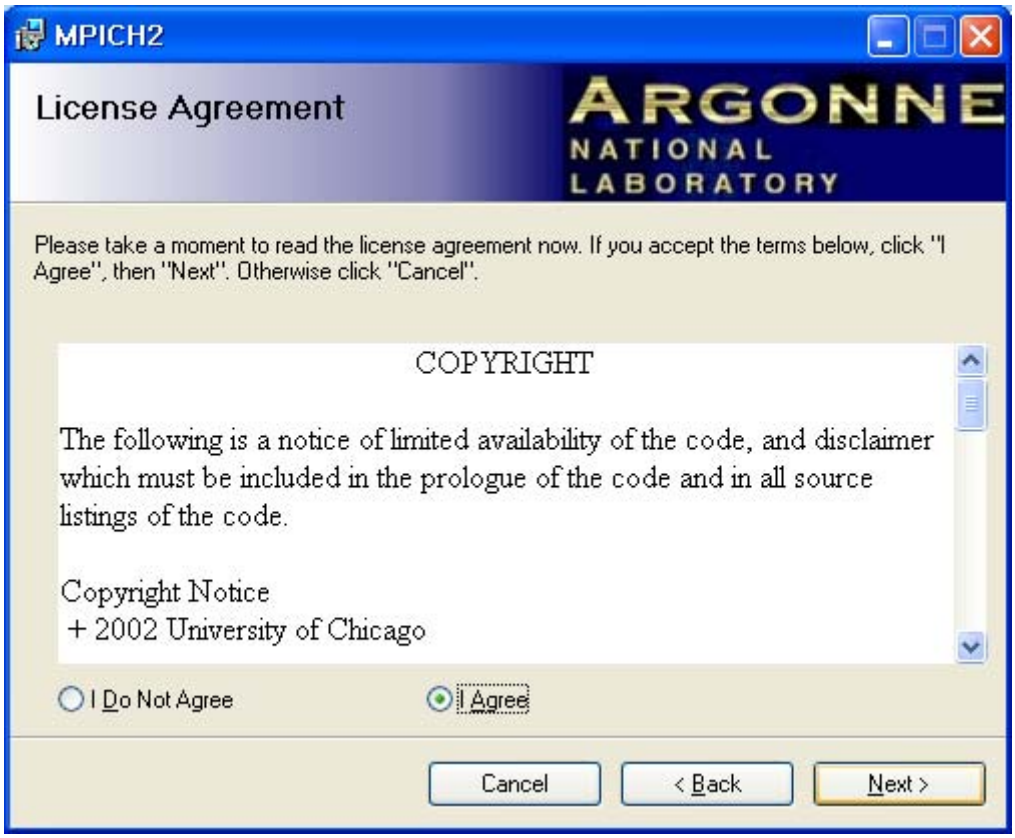
-Microsoft .NET Framework Version 2.0 Redistributable Package (x86)

위 두 프로그램이 설치되어 있지 않아 MPICH2가 설치되지 않으면 구글에서 위의 문장으로 검색을 하여 해당 프로그램을 다운로드 한다. 위 두 프로그램을 설치하였으면 MPICH2를 아래와 같이 설치하자.

(1) 기본적인 안내 메시지들은 Next를 클릭하여 넘어간다.

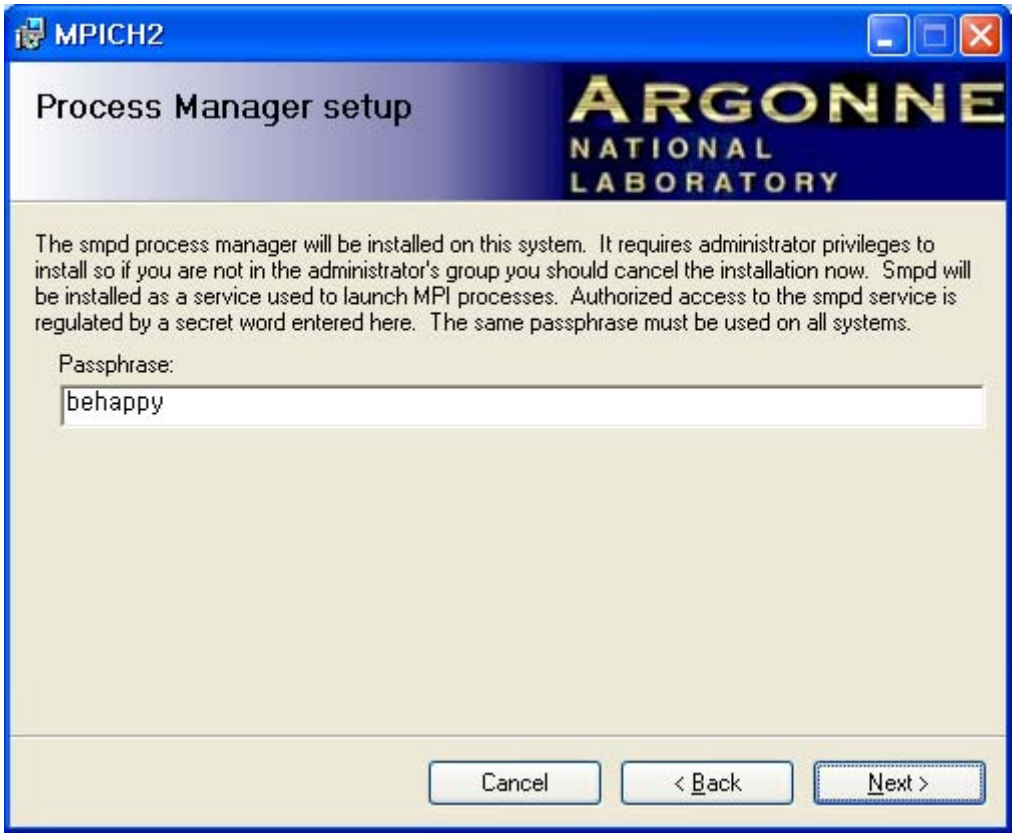



MPICH2
MPICH2 Information
ARGONNE
NATIONAL
LABORATORY
MPICH2 for Microsoft Windows
Email bugs and error reports to:
mpich2-maint@mcs.anl.gov
SYSTEM REQUIREMENTS:
1) MS Development Environment, Visual Studio or gcc to compile C/C++ MPI programs.
2) Intel Fortran 8.0 or g77 to compile Fortran MPI programs.
3) Administrator privileges on your machines to install the process manager, smpd.exe.
Cancel
< Back
Next >


MPICH2
License Agreement
ARGONNE
NATIONAL
LABORATORY
Please take a moment to read the license agreement now. If you accept the terms below, click "I Agree", then "Next". Otherwise click "Cancel".
COPYRIGHT
The following is a notice of limited availability of the code, and disclaimer which must be included in the prologue of the code and in all source listings of the code.
Copyright Notice
+ 2002 University of Chicago
I Do Not Agree
I Agree
Cancel
< Back
Next >

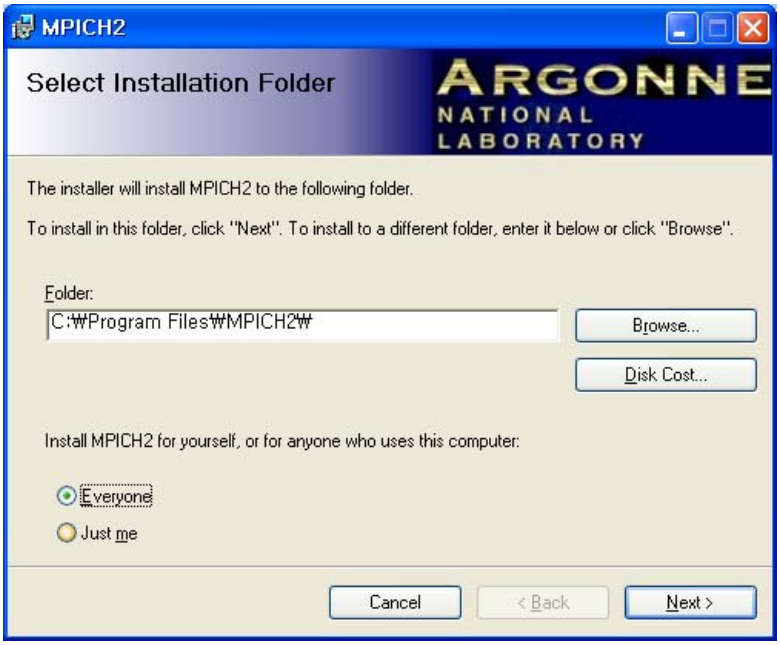
(2) Passphrase 설정

Passphrase를 설정하는 부분은 작업에 참여할 모든 컴퓨터들이 동일한 passphrase를 사용해야 하므로 주의하자.



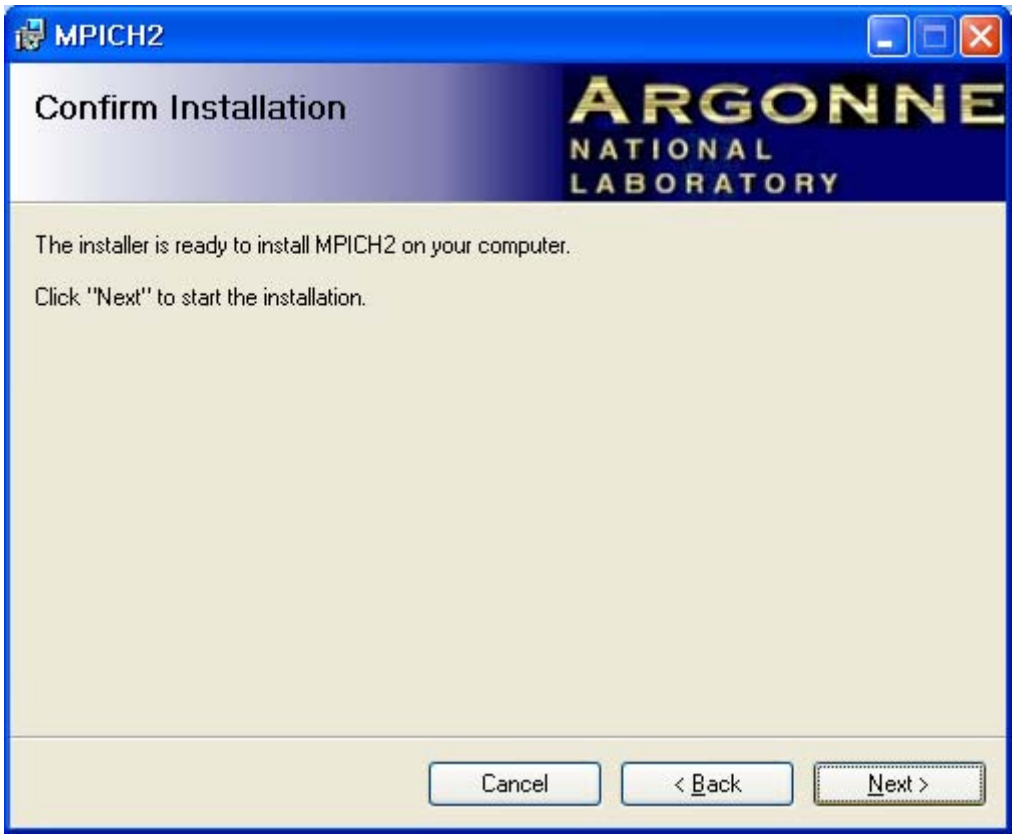
(3) MPICH2의 사용권한 설정

이 컴퓨터 내의 모든 계정에서 이 프로그램을 사용할 수 있게 할 것인지, 혹은 현재 이 인스톨러를 실행중인 계정에서만 사용할 수 있게 할 것인지 설정하는 부분이다. Everyone을 선택하자.



(4) 설치 시작

Next 를 클릭하면 설치가 시작된다. 설치가 완료되면 Close 를 클릭하여 인스톨러를 종료한다.



d. MPICH2 환경설정

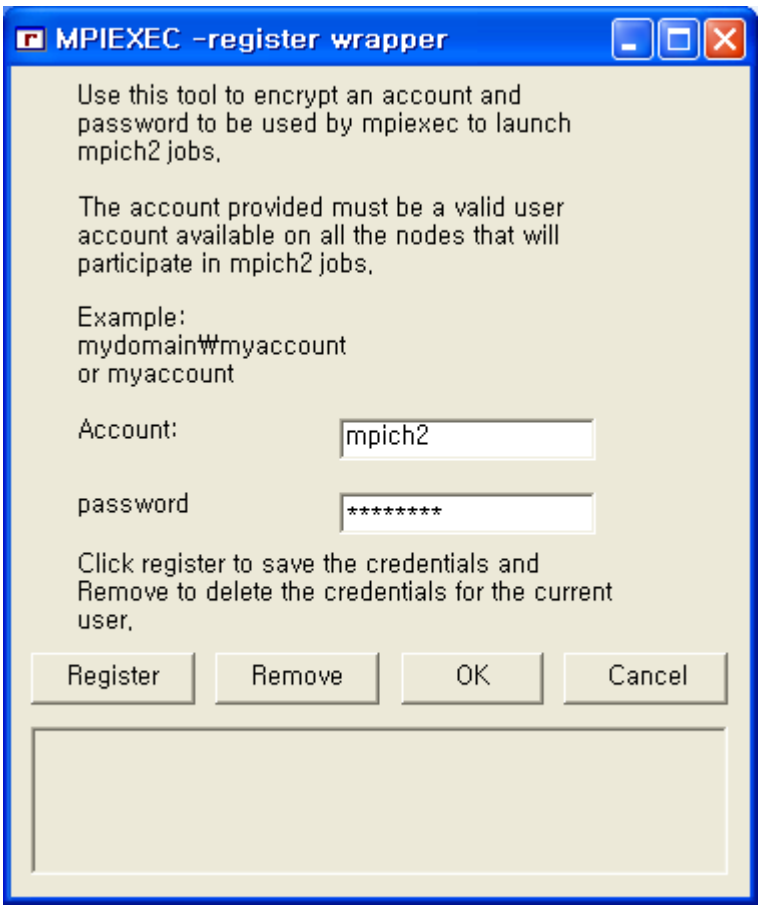
MPICH2를 모든 컴퓨터에 설치한 후에 MPICH2가 설치된 각 컴퓨터들이 실제로 연결이 되는지 확인하자. 아래 메뉴를 통하여 MPICH2 환경 설정 프로그램을 실행한다.

"시작 -> 모든 프로그램 -> MPICH2 -> wmpiconfig.exe"

(1) 계정과 비밀번호 등록

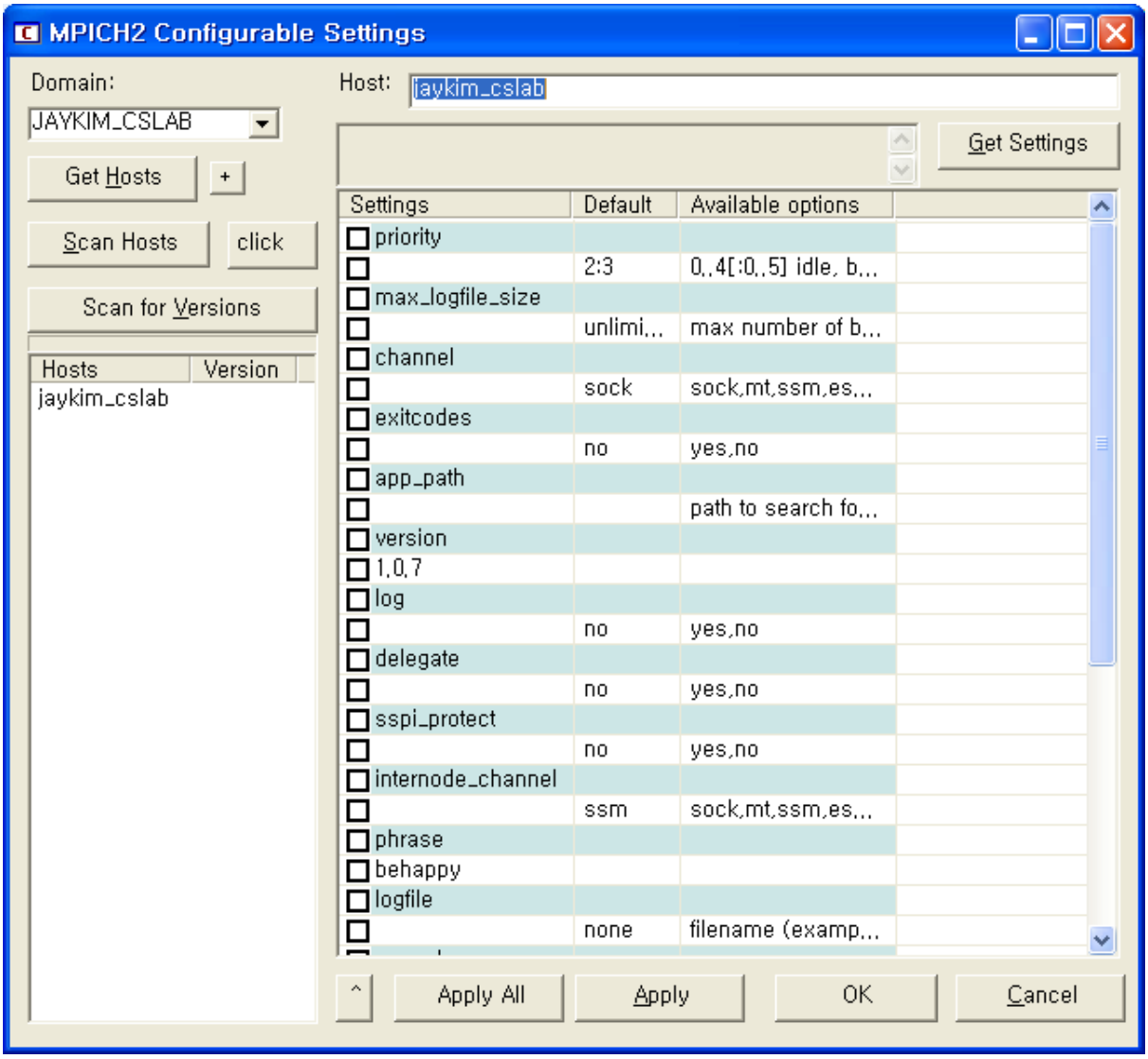
MPICH2 환경설정 프로그램을 처음 실행하면 계정과 비밀번호를 물어보는 창이 나온다. 이 때 이전 단계 1. b 에서 생성했던 윈도우 계정 이름과, 비밀번호를 입력하고 Register를 클릭한다.

다시 한번 강조하지만 지금 단계에서 입력하는 계정이름과 비밀번호를 이용하여 다른 모든 컴퓨터들에 접속하기 때문에 모든 컴퓨터들의 계정과 비밀번호는 동일해야 한다.



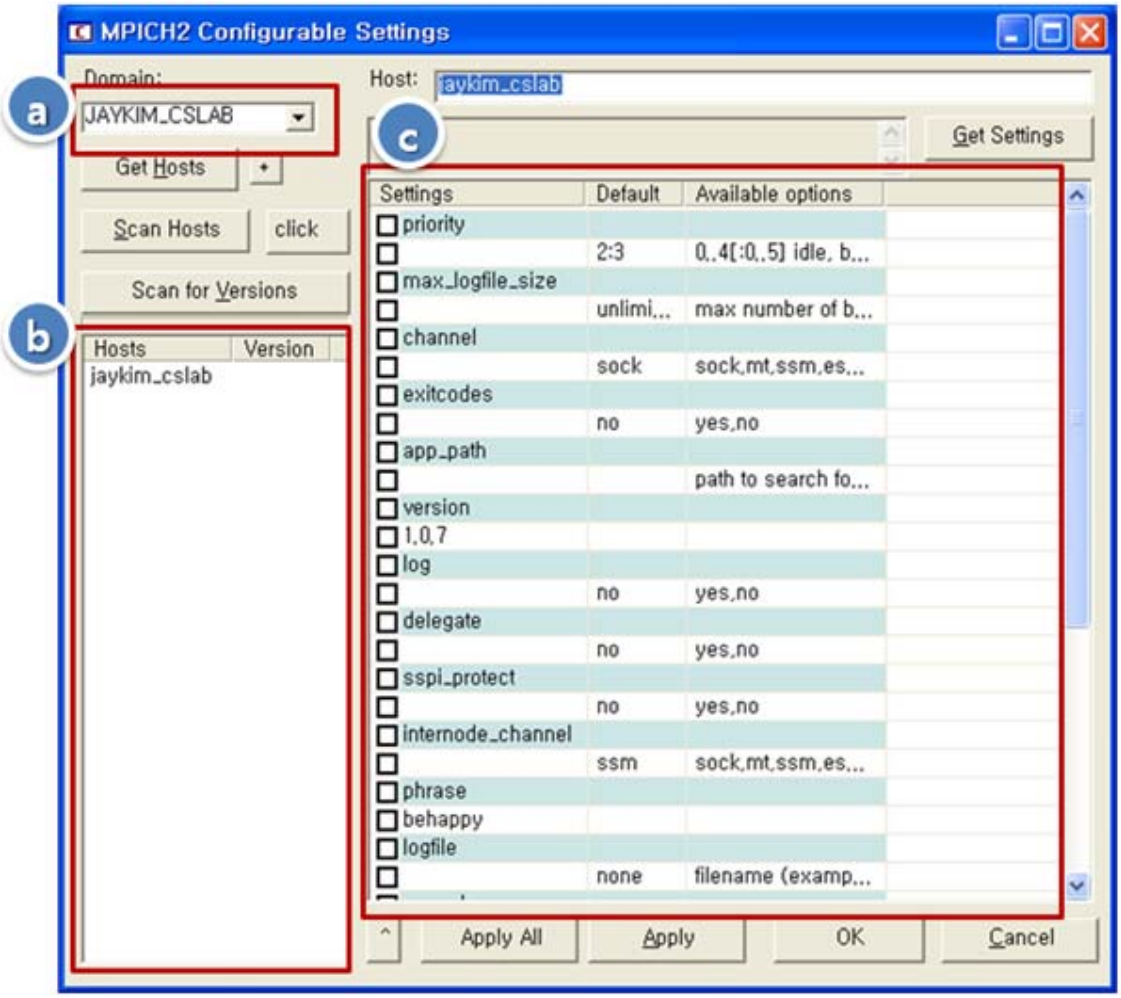
<윈도우 계정 이름과 비밀번호를 입력>

계정과 비밀번호를 입력하고 Register를 클릭한 후 wmpiconfig.exe 프로그램이 아래 그림과 같이 나타나야 한다.



<wmpiconfig.exe 실행화면>

(2) MPICH2 환경설정



<wmpiconfig.exe 의 화면 구성>

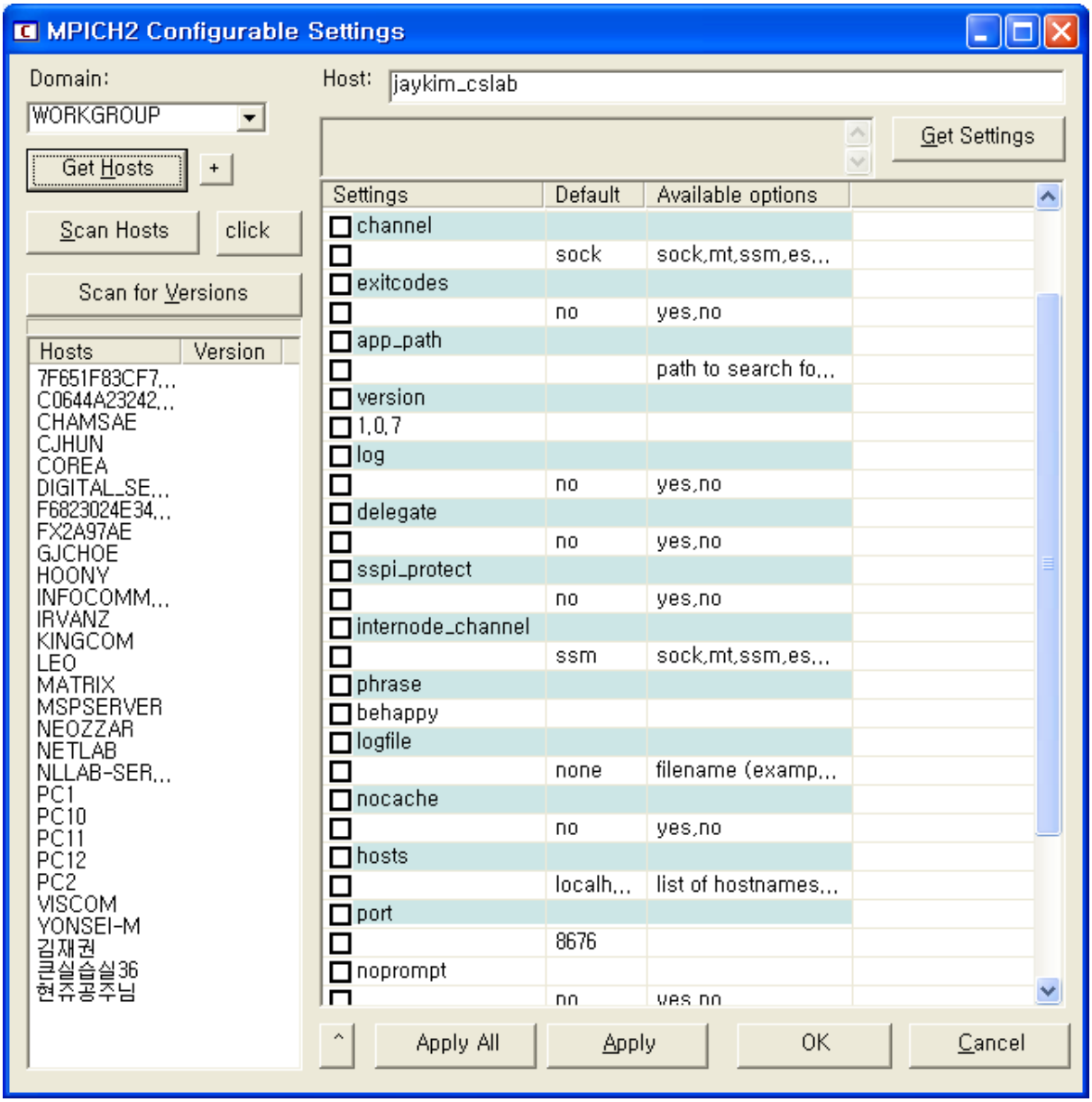
MPICH2 환경설정 프로그램은 위의 그림과 같이 크게 세가지 창이 있는데, 각각의 용도는 아래와 같다.

(a) 작업에 참여할 컴퓨터들의 작업그룹을 설정한다.

(b) (a)에서 선택한 작업그룹에 존재하는 Host의 list가 나타나는 창이다.

(c) 해당 호스트의 MPICH2 현재 설정 상태를 표시하고, 사용자가 수정할 수 있게 하는 창이다.

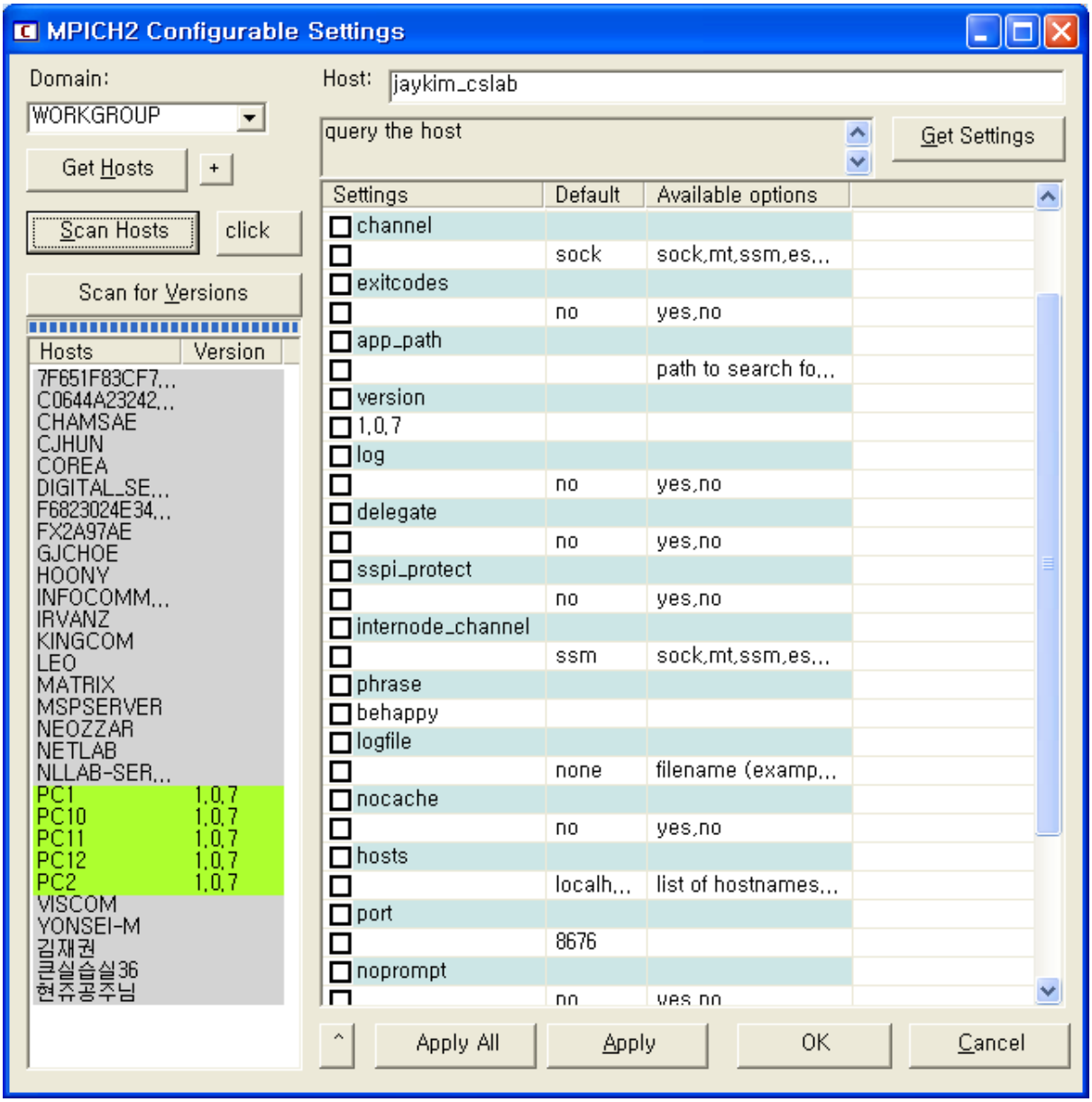
우선 1.a에서 설정한 작업 그룹 내에 있는 컴퓨터들 간의 연결 상태를 체크하기 위하여 위 그림에서 a로 표시된 부분에서 Domain을 이전에 설정한 작업그룹의 이름으로 선택하고, Get Hosts 버튼을 클릭하여 작업 그룹 내의 컴퓨터 목록을 생성한다.



<Domain을 작업그룹으로 설정하고 Get Hosts 버튼을 클릭한 모습>

이 시스템에서는 "WORKGROUP" 작업그룹 내에서 PC1, PC2, PC10, PC11, PC12 총 5대의 컴퓨터에 MPICH2가 모두 설치되어 있고, 사용자 계정과 비밀번호가 모두 동일하게 설정되어 있는 상태이다.

이 5대의 컴퓨터들이 올바로 설정되고 연결상태가 정상적인지 확인하기 위하여 Scan Hosts 버튼을 클릭한다.



<Scan Hosts 버튼을 클릭한 모습>

지금까지 해온 방법으로 미리 설정을 해둔 5대의 컴퓨터가 올바로 설정되었음을 컴퓨터 이름에 연두색으로 표시하여 나타내고 있다. MPICH2가 연결할 수 없는 컴퓨터는 회색으로 나온다. 연두색으로 표시된 컴퓨터는 현재 이 작업을 수행하는 컴퓨터에서 MPICH2를 이용하여 병렬처리 작업에 이용할 수 있는 컴퓨터이다.
만약 이 때 설정한 컴퓨터가 연두색으로 표시 되지 않는다면 앞의 과정을 다시 꼼꼼히 읽어본 후 다시 시도해본다.

지금까지의 과정으로 MPICH2의 환경설정은 모두 끝났다.

(3) 이 문서 작성에 사용된 시스템의 구성

-총 6대의 컴퓨터

PC1, PC2, PC10, PC11, PC12, jaykim_clsab(이 문서를 작성할 때 사용한 컴퓨터)

-작업그룹 이름

WORKGROUP

-6대의 컴퓨터의 사용자 계정과 비밀번호

계정 : **mpich2**

비밀번호 : **--------**

-방화벽 사용안함

-작업 그룹 내에 다른 컴퓨터들도 있지만 MPICH2가 설치되지 않았고 'mpich2' 사용자 계정이 없기 때문에 MPICH2의 작업에 참여할 수 없음

2. Parallel Programming Using MPICH2

이제 Parallel Programming을 위한 환경구축이 모두 끝났으니 간단한 예제를 실행해본다. 이 예제를 실행하는 과정에서 MPICH2를 이용한 프로그램을 컴파일 하는 방법, 컴파일 된 실행프로그램을 다수의 컴퓨터에서 실행하는 방법 등을 알 수 있을 것이다..

a. 예제 프로그램, test.c

test.c

```
#include <stdio.h>
#include "mpi.h"
#define MAX_DATA 100

main(int argc, char **argv){
	int rank;
	int size;
	char processor_name[MPI_MAX_PROCESSOR_NAME];
	int namelen;
	char buff[MAX_DATA];
	int i;
	MPI_Status status;

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

	if(rank==0){
		printf("### Process %d from %s ###\n", rank, processor_name); fflush(stdout);
		for(i=1;i<size;i++){
			MPI_Recv(buff, MAX_DATA, MPI_CHAR, i, 0, MPI_COMM_WORLD, &status);
			printf("%s\n", buff); fflush(stdout);
		}
	}
	else{
		sprintf(buff, "### Process %d from %s ###", rank, processor_name); fflush(stdout);
		MPI_Send(buff, MAX_DATA, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
	}
	MPI_Finalize();
}
```

b. 소스 컴파일

위 예제는 MPI 의 가장 기초적인 예제이다. 이를 컴파일하고 단일 컴퓨터 상에서 실행해보고 동작을 확인하고, 다수의 컴퓨터에서 실행해보자.

MPICH2 를 이용하여 작성한 프로그램을 Visual Studio 를 이용하여 컴파일 하고 실행할 때는 아래와 같은 절차가 필요하다.

(1) 프로젝트를 생성한다.
(2) include 파일 경로에 mpich2₩include 를 추가한다.
(3) library 파일 경로에 mpich2₩lib 를 추가한다.
(4) link 명령어에 mpi.lib 를 추가한다.
(5) 생성한 실행 파일을 실행할 모든 컴퓨터의 동일한 위치에 복사한다.

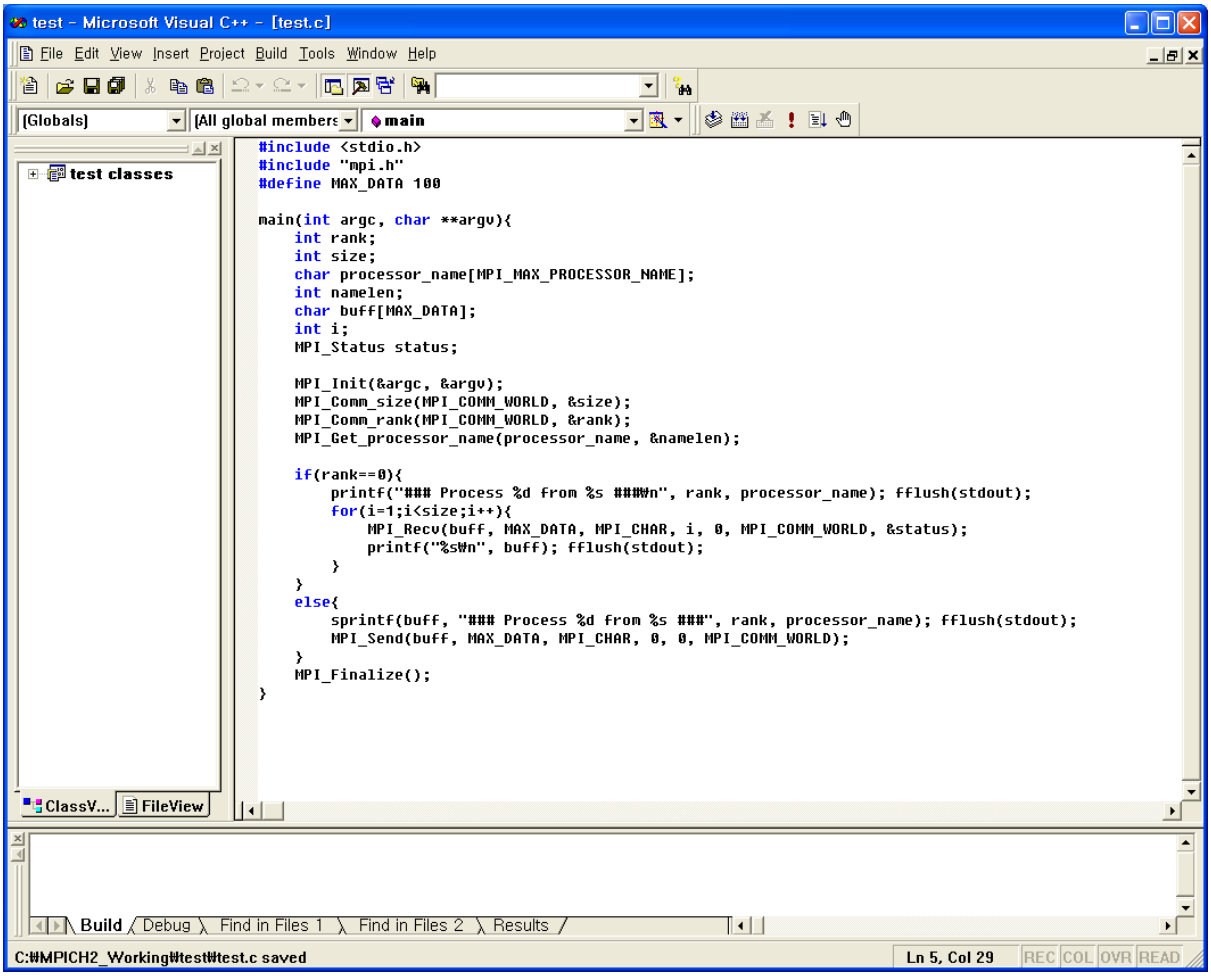
(6) mpiexec 명령어를 이용하여 프로그램을 실행한다.

* MPICH2 의 기본 설치 폴더는 "C:₩Program Files₩MPICH2" 이다.

Visual Studio 를 이용하여 프로젝트를 생성하고 위 예제를 컴파일 해보자.

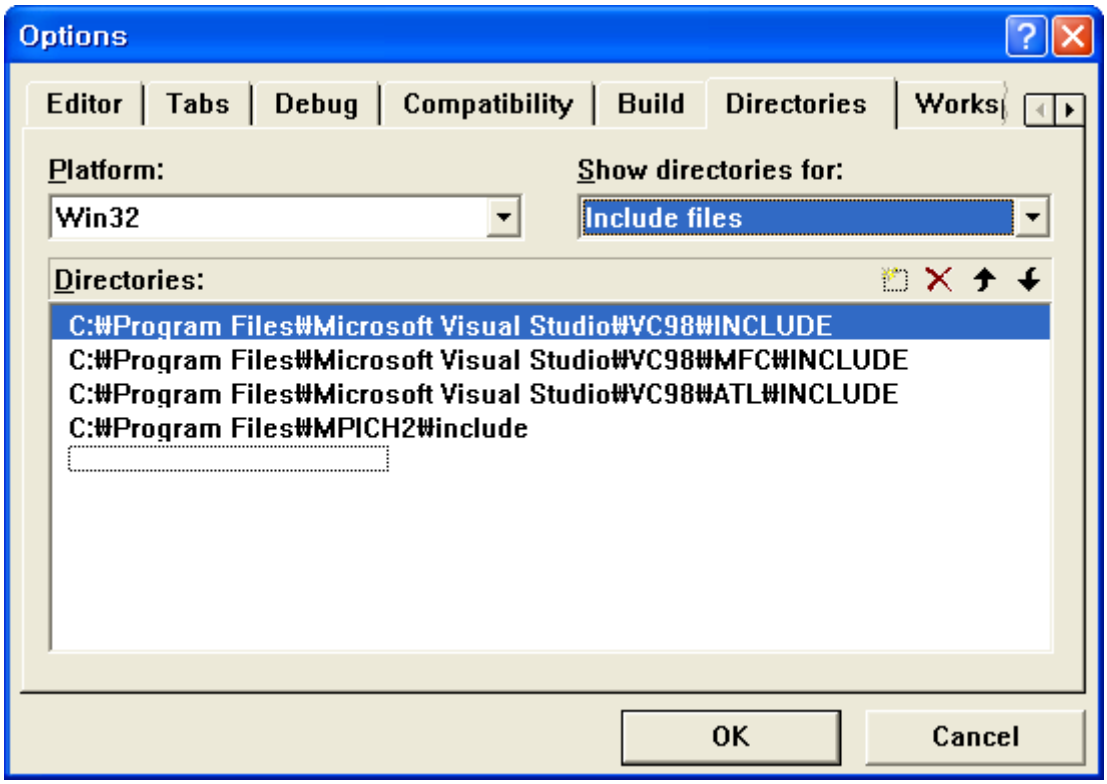
(1) 프로젝트를 생성한다.

프로젝트 파일을 임의의 디렉토리, C:₩MPICH2_Working₩test 에 저장한다.

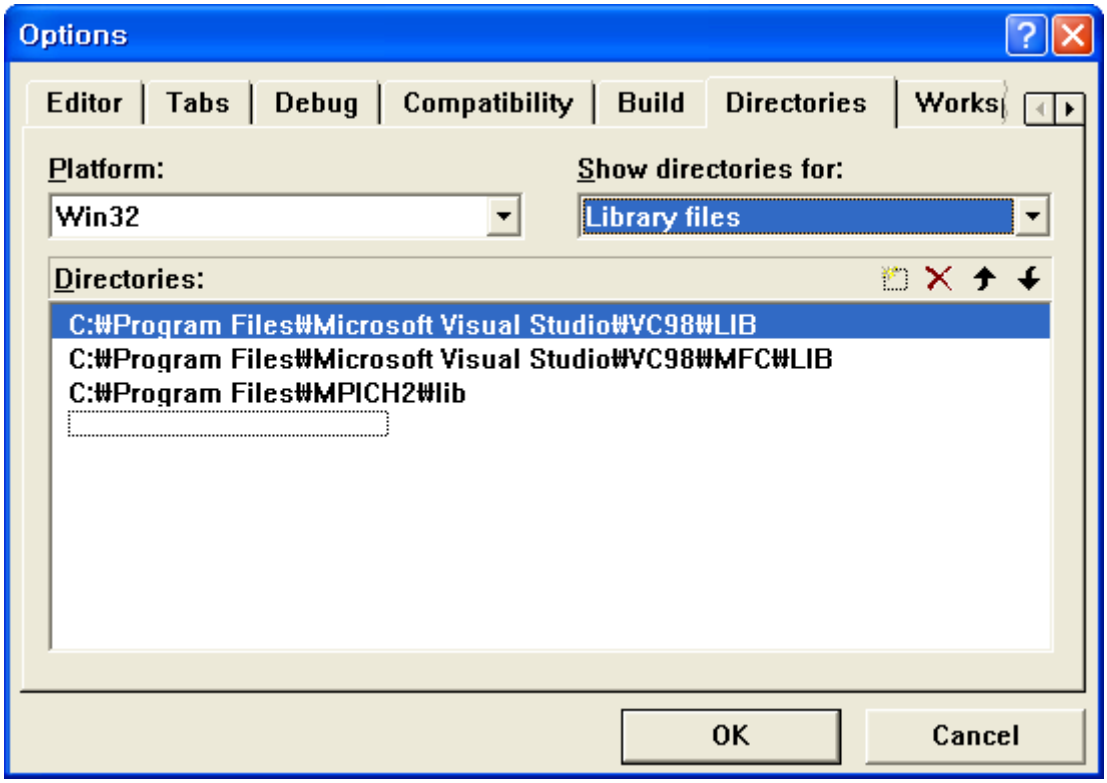


(2) include, library 파일 경로에 mpich2₩include, mpich2₩lib 를 추가한다.

Visual Studio 메인 메뉴 -> Tools -> Options -> Directories



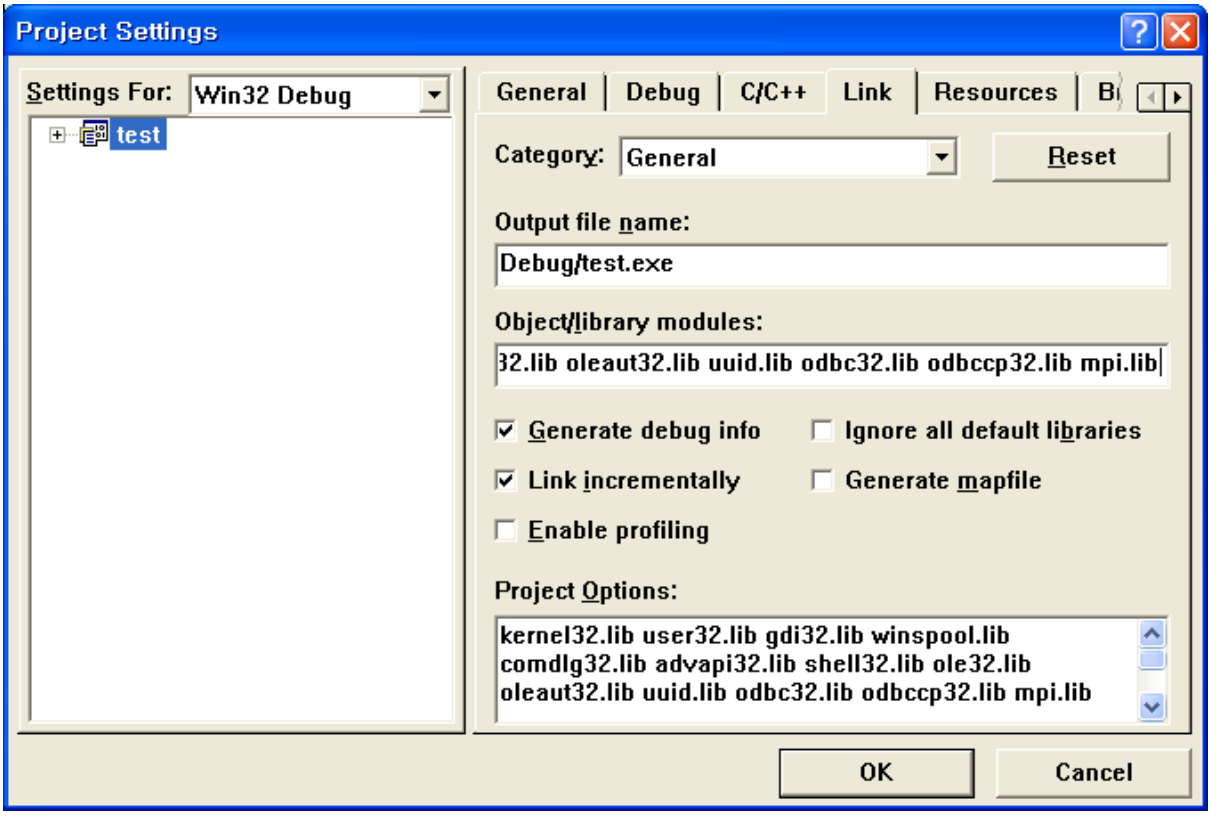
위 그림과 같이 MPICH2 의 include 파일의 위치를 추가한다.



Show directories for: 메뉴에서 Library files 를 선택하고 위 그림과 같이 MPICH2 의 lib 파일의 위치를 추가한다.

(3) link 명령어에 mpi.lib 를 추가한다.

Visual Studio 메인 메뉴 -> Project -> Settings -> Link



Link 탭의 Object/library modules: 텍스트 입력 창의 가장 뒤쪽에 "mpi.lib" 를 타이핑 하여 추가한다.

(4) 컴파일 환경 설정 완료

이 과정을 마치면 MPICH2 를 이용하여 작성한 프로그램을 컴파일 할 수 있다. 컴파일을 하여 실행파일을 생성하자.

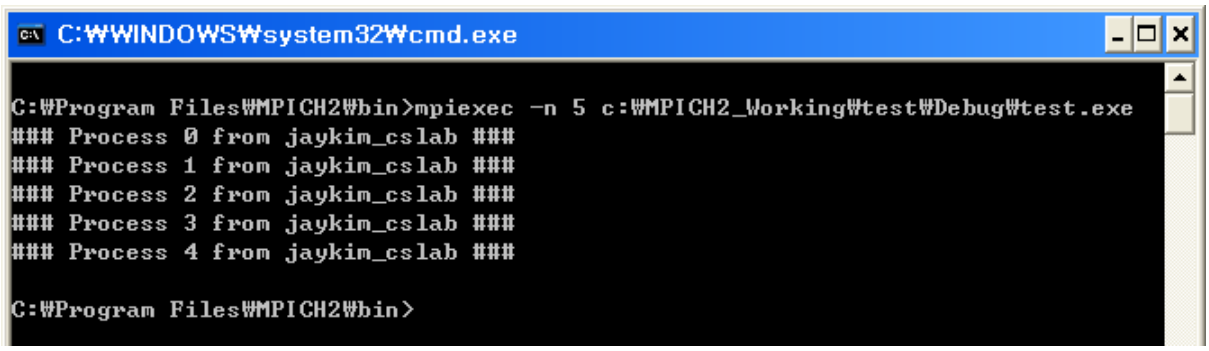
* 생성한 실행파일의 위치를 잘 기억하도록 하자.
  여기서는 생성한 실행파일의 위치를 c:₩MPICH2_Working₩test₩debug 폴더로 가정한다.

c. 실행

MPICH2 를 이용하여 작성한 프로그램은 실행할 때 별도의 프로그램이 필요하다. MPICH2 의 기본 설치 폴더 내에는 'bin' 'lib' 'include' 'examples' 등의 폴더가 있다. 우선 MPICH2 로 작성한 프로그램을 실행하기 위해 필요한 프로그램은 bin 폴더 내에 위치한다. Console 프로그램을 열고, cd c:₩program files₩MPICH2₩bin 명령어를 입력하여 MPICH2 폴더로 이동하고 아래 명령어를 입력한다.

[참고] 환경변수 설정에서 MPICH2 의 bin 폴더를 path 에 추가해 두면 mpiexec 명령어를 편하게 사용할 수 있다.

C:₩Program Files₩MPICH2₩bin>mpiexec –n 5 c:₩MPICH2_Working₩test₩debug₩test.exe



<실행 결과>

MPICH2 로 작성한 프로그램을 실행시켜주는 mpiexec 명령어를 이용하여 위에서 작성한 예제를 실행 하였다. 여기서 사용한 –n 5 파라미터는 이 프로그램을 5 개의 프로세스로 생성 하라는 의미이다. 즉, 이 명령어가 수행되면 test.exe 프로그램의 프로세스가 5 개가 생성되는 것이다. 이렇게 생성된 프로세스는 smpd.exe 에 의하여 관리되기 때문에 서로 통신(Message Passing)이 가능하다. 그리고 각 프로세스 마다 고유한 ID 도 할당 받기 때문에 각각의 프로세스를 구분할 수 있다. 자세한 사항은 소스 분석을 통하여 설명하도록 하겠다.

d. 소스 분석

test.c

```
#include <stdio.h>
#include "mpi.h"
#define MAX_DATA 100

main(int argc, char **argv){
        int rank;
        int size;
        char processor_name[MPI_MAX_PROCESSOR_NAME];
        int namelen;
        char buff[MAX_DATA];
        int i;
        MPI_Status status;

        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Get_processor_name(processor_name, &namelen);
        /* 여기까지 모든 프로그램이 동일하게 수행된다. rank 는 각 프로세스를 구별하기 위해
           사용된다. 주로 rank 0 은 특별한 용도로 사용된다.*/
        /* 이 부분부터 프로세스의 rank 에 따라 수행할 코드가 달라진다. */
        // rank 가 0 번인 프로세스가 수행할 코드
        if(rank==0){
                printf("### Process %d from %s ###\n", rank, processor_name); fflush(stdout);
                for(i=1;i<size;i++){
                        MPI_Recv(buff, MAX_DATA, MPI_CHAR, i, 0, MPI_COMM_WORLD, &status);
                        printf("%s\n", buff); fflush(stdout);
                }
        }
        //rank 가 0 이 아닌 프로세스가 수행할 코드, 이 경우에는 rank 가 1~4 인 경우
        else{
                sprintf(buff, "### Process %d from %s ###", rank, processor_name); fflush(stdout);
                MPI_Send(buff, MAX_DATA, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
        }
        MPI_Finalize();
}
```

위 소스 중 주요 MPICH2 Function 과 기능은 아래와 같다.

```
MPI_Init();
MPI_Comm_size();
MPI_Comm_rank();
MPI_Recv();
MPI_Send();
MPI_Finalize();
```

이 예제와 같은 프로그래밍 Model 을 SMPD(Single Program Multiple Data)이라 한다. 즉, 같은 소스 코드의 프로그램을 이용하여 여러 개의 프로세스를 생성하는 것이다. 동일한 소스 코드로 다수의 프로세스가 생성되기 때문에 각각의 프로세스를 구분할 수 있어야 각 프로세스 별로 작업을 달리 할당하거나 하는 일을 할 수 있을 것이다.

가장 먼저, MPI_Comm_rank 함수는 mpiexec 프로그램을 이용해 생성된 다수의 프로세스를 식별할 수 있는 고유번호를 할당해 준다. 즉, mpiexec –n 5 test.exe 명령어를 이용하여 5 개의 프로세스를 생성한다면 0~4 의 rank 가진 프로세스가 생성되는 것이다.

MPI_Comm_size 함수는 mpiexec 명령어를 이용해 몇 개의 프로세스가 생성 되었는지를 알려준다. mpiexec –n 5 test.exe 로 실행한 경우에는 size 의 값이 5 가된다.

MPI_Send 함수는 목적지 프로세스의 고유번호(rank)와 전송할 데이터에 관한 정보들을 파라미터로 받아 해당 목적지 프로세스에게 메시지를 전달하는 함수이다. 즉, 이 함수를 통하여 같은 영역(MPI_COMM_WORLD) 내에 있는 다른 프로세스에게 메시지를 전달하는 것이다.

MPI_Recv 함수는 메시지를 보내는 프로세스의 정보와 수신된 메시지를 저장할 변수를 파라미터로 입력 받고, 메시지를 보내는 프로세스가 Send 함수를 이용하여 메시지를 전송하면 전송받은 데이터를 지정한 변수(버퍼)에 저장한다.

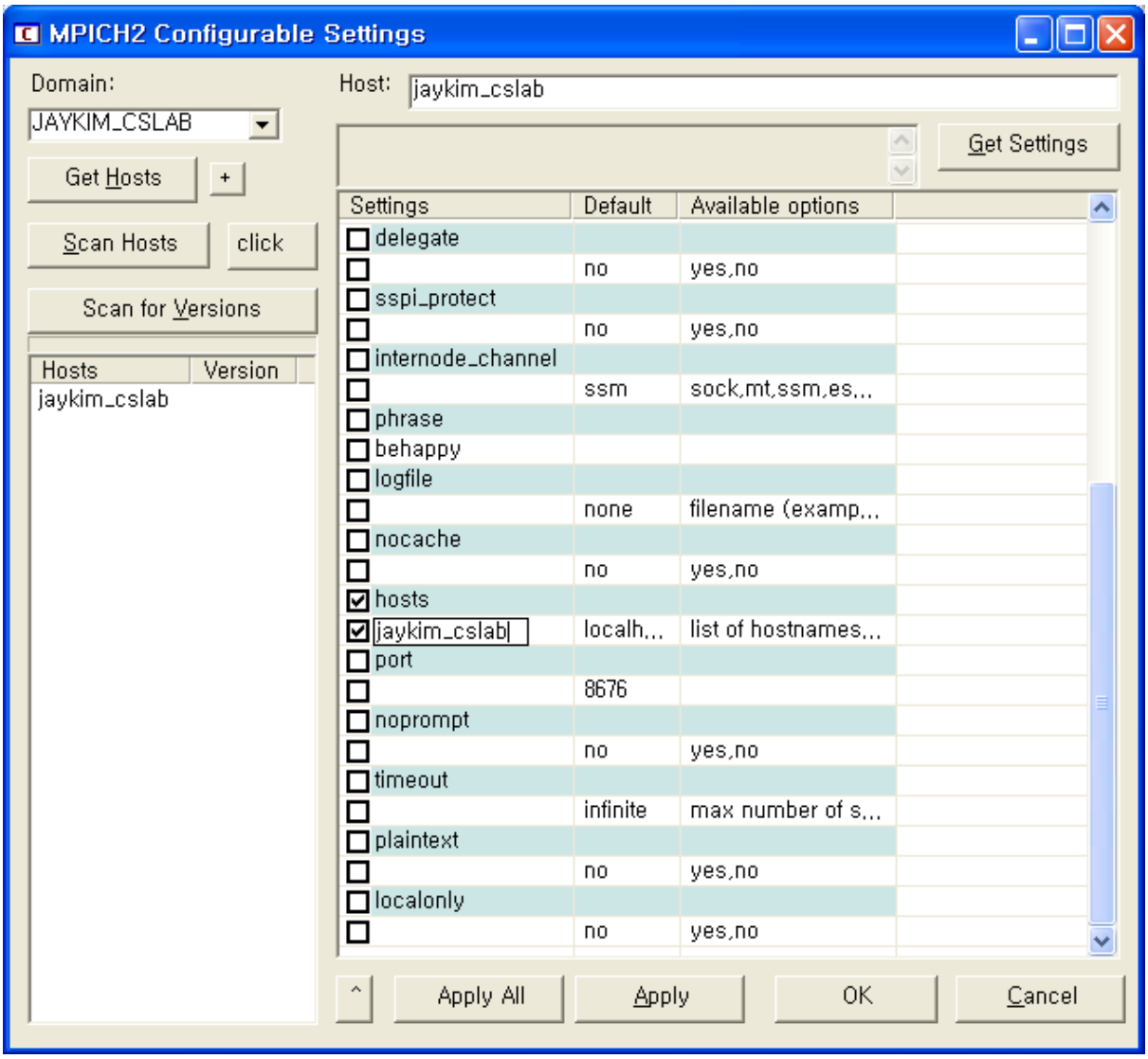
MPI_Init 함수는 MPI 를 이용하기 전에 항상 호출해야 하는 함수이다.
MPI_Finalizer 함수는 MPI 를 이용한 프로그램이 종료되기 전에 항상 호출해야 한다.

위와 같은 MPI 함수를 이용하여 이 예제 프로그램은, rank 가 0 인 프로세스(프로세스 0)는 4 개의 프로세스로부터 메시지가 전송되기를 기다리고, 프로세스 1 ~ 프로세스 4 는 프로세스 0 에게 메시지를 전송한다. 프로세스 0 이 프로세스로부터 메세지를 받은 후 수신한 메세지 내용을 출력하는 동작을 하고 있다.

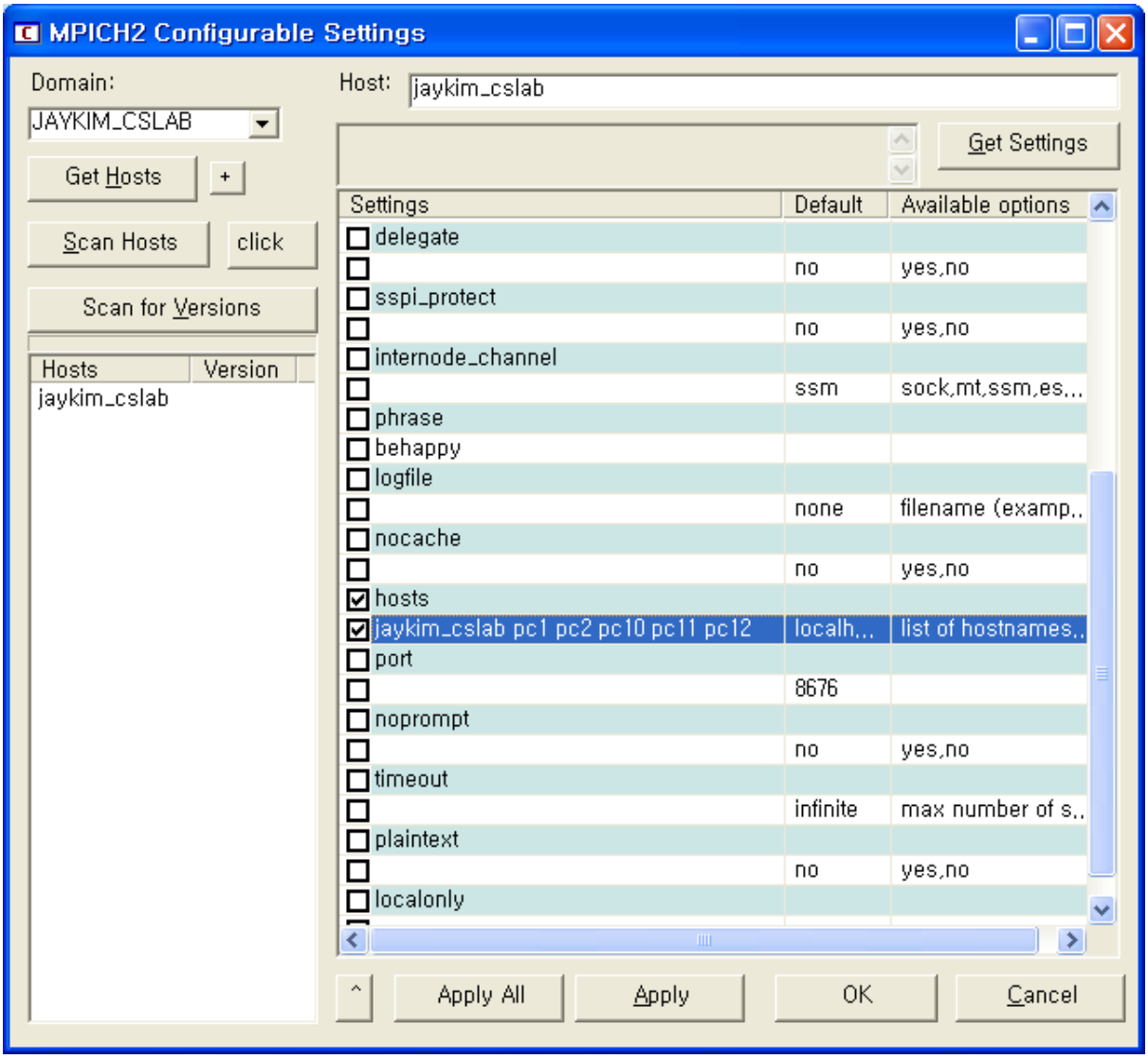
여기서 중요한 사실은 각각의 프로세스들이 동일한 소스 프로그램을 이용하여 생성된다는 점과(SMPD Model), 각 프로세스는 하나의 컴퓨터 내의 메모리에 모두 생성될 수 도 있고, 여러 대의 독립적인 컴퓨터의 메모리에서 각각 생성될 수 도 있다. 각각의 프로세스가 하나의 컴퓨터 내의 메모리에 생성하는 경우는 방금 해보았고, 이번에는 5 개의 프로세스를 각각 5 대의 별도의 컴퓨터의 메모리에 생성해 보겠다. 이렇게 생성한 프로세스들 역시 Send, Recv 함수를 이용하여 서로 통신할 수 있다. 이런 경우에는 MPICH2 의 환경설정 프로그램을 이용하여 설정을 변경해줘야 한다.

e. 여러 대의 컴퓨터에 프로세스를 생성하기 위한 환경설정

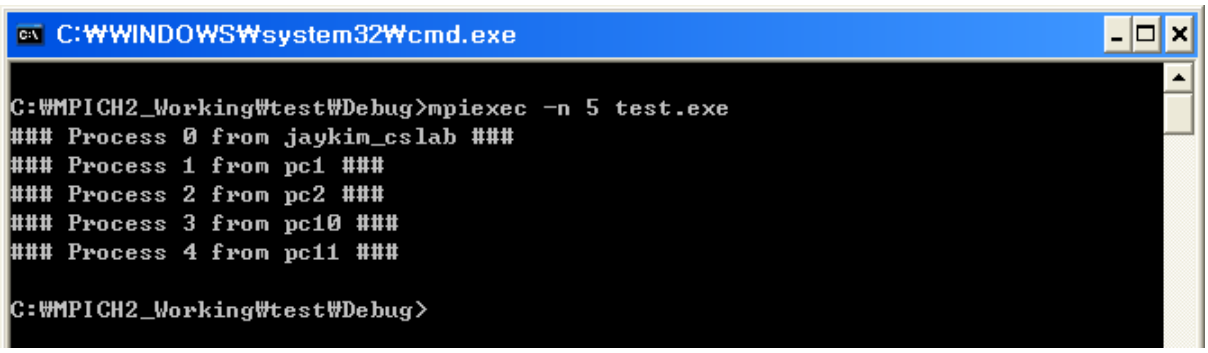
Wmpiconfig.exe 을 실행한다.



위와 같은 화면이 나오면 가운데에 위치한 창의 메뉴중 'hosts' 의 정보를 변경해줘야 한다. 이 메뉴를 이용하여 MPICH2 를 이용하여 프로세스를 생성할 때 참여할 컴퓨터들을 정할 수 있다. 미리 설정해둔 5 대의 컴퓨터 이름을 아래와 같이 추가 기입하고 Apply 버튼을 클릭하여 설정을 저장한다.. (ex, PC1 PC2 PC10 PC11 PC12)



이제 다수의 프로세스를 다수의 컴퓨터에서 생성할 수 있게 된다. console 에서 아래 명령어를 입력하고 실행 해보자.

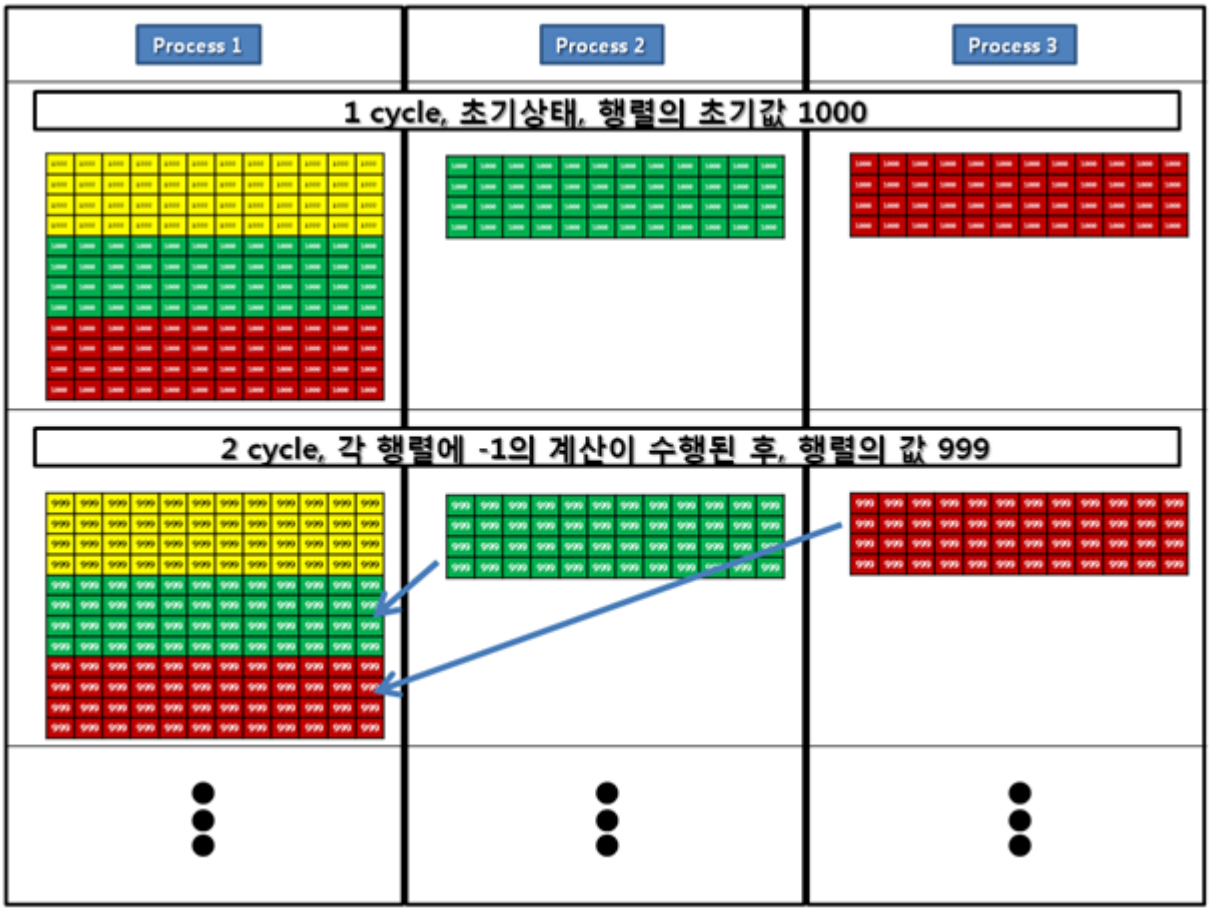


5 개의 프로세스들이 각각 다른 컴퓨터에서 생성되고 서로 통신하는 모습을 볼 수 있다. 이러한 Message Passing 을 이용하여 병렬처리를 수행할 수 있다.

3. 병렬처리 예제 프로그램

첨부한 예제 프로그램은 특정 값(ex. 1000)으로 초기화되어 있는 12X12 행렬의 값을 특정 값(ex. 1)으로 감산하여 0 이하가 될 때까지 계산하는 프로그램이다. 12 개의 행을 다수의 프로세스가 병렬로 계산한 후 이 결과 값을 Master 프로세스(rank 가 0 인)에게 돌려주고 Master 프로세스는 이 값을 자신의 행렬에 반영하는 방식으로 동작한다.

아래 그림은 12X12 행렬이 1000 으로 초기화 되어있고, 각 행렬의 값이 1 씩 감소하는 과정을 나타낸 예이다. 3 개의 프로세스(컴퓨터)가 작업에 참여하고 프로세스당 4 개의 행을 계산한다.



1 cycle 에서는 모든 행렬의 값이 1000(초기값)이고, 다음 cycle 에서 각 프로세스가 1 을 감소한 값을 master process(process 1)로 전달한다. 효율적인 메모리 사용을 위해, Master process 는 12X12 행렬의 모든 값을 가지고 있고, process2, process3 은 자신의 계산 영역의 행렬만 가지고 있는다. 그리고 자신의 계산 영역의 행렬 원소들을 계산 하고 그 결과를 process1 에게 전송하고, process1 은 전달 받은 값을 자신이 가지고 있는 행렬에 반영한다. 즉, 2 cycle 이 되면 process1~process3 이 가지고 있는 행렬의 값은 모두 999 가 된다. 행렬의 값에 1 을 빼는 계산은 프로세스 2~3 에서 수행되고, 그 계산의 결과 값이 process1 으로 전달된다. 이런 계산을 반복하여 process1 에 있는 행렬의 모든 값이 0 이하가 될 때까지 반복한다.

[참고] 각각의 프로세스가 행렬의 값에 -1 을 하는 계산은 너무 간단한 계산이기 때문에 병렬처리의 효과를 얻을 수 없다. 실제 소스에서는 보다 복잡한 실수 연산인, 0.0001 을 100000 번 빼는 등의 연산을 하여 각 프로세스에게 보다 복잡한 연산을 수행하도록 하였다.

[실험 결과, 소스 프로그램 첨부]

4. 참고자료

Web Sites

http://www.mcs.anl.gov/research/projects/mpich2/

http://www.mpi-forum.org/docs/docs.html

http://www.mcs.anl.gov/research/projects/mpich2/documentation/index.php?s=docs

http://www-unix.mcs.anl.gov/mpi/

http://socmaster.homelinux.org/~hongiiv/296

https://computing.llnl.gov/tutorials/mpi/

http://www.parawiki.org/index.php/MPI

http://helpdesk.ksc.re.kr/technote/mpi_data.htm

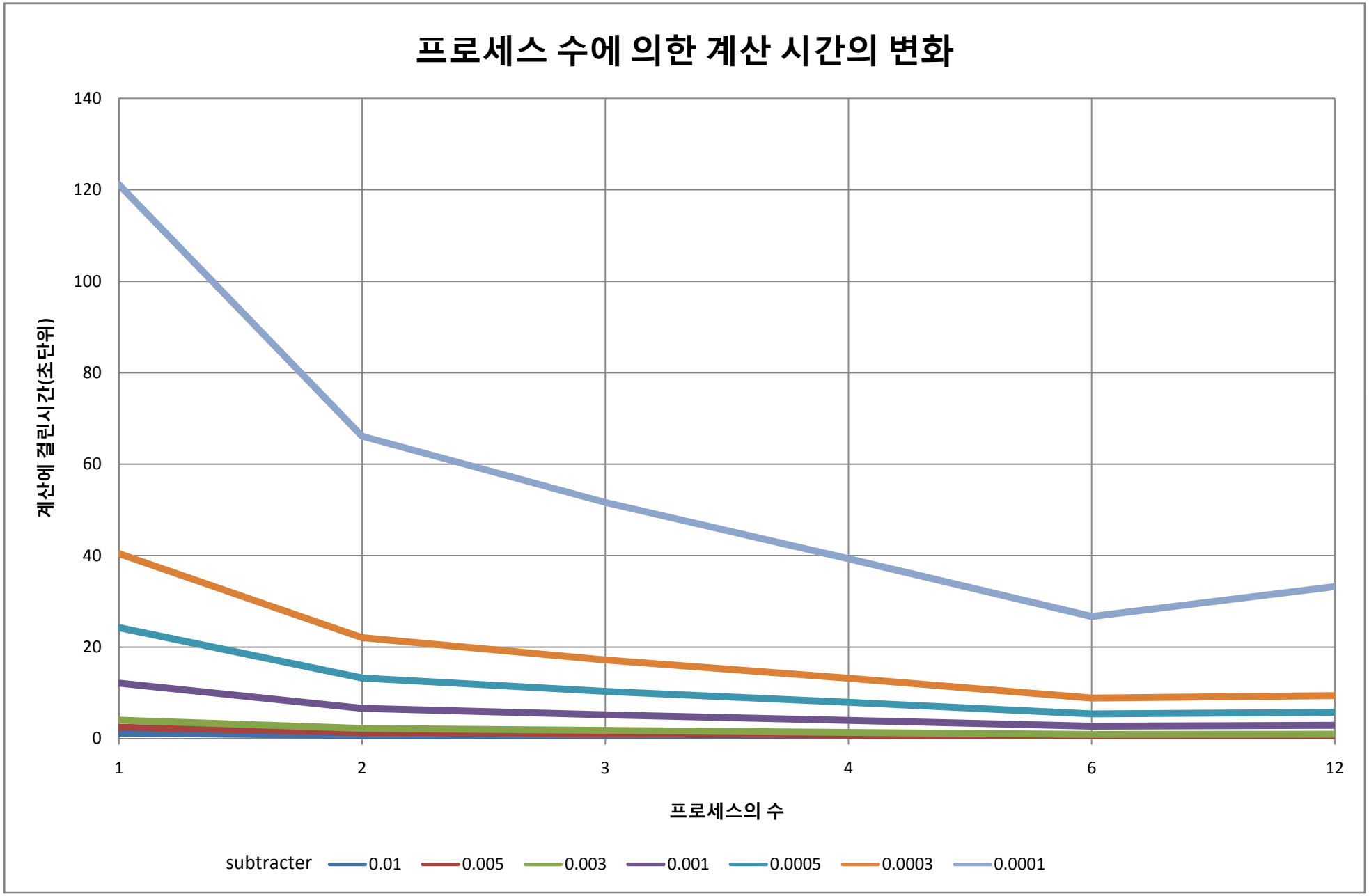
Books

Parallel Programming with MPI, Peter S. Pacheco

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

//parameter
#define ROW_MAX 12
#define COL_MAX 12

#define MAX_DATA 100

int control_array(int type, double value, int row, int col);

//기본 자료구조
double array[ROW_MAX][COL_MAX];
double t_array[ROW_MAX][COL_MAX];
//parameter
double init_value = 1000.0;
double subtracter = 1;

int cycle_counter=0;

main(int argc, char **argv){
    //mpich2
    int rank;
    int size;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int namelen;
    double starttime=0.0, endtime;
    MPI_Status status;
    //mpich2 buffer
    int buf[MAX_DATA];
    char char_buf[MAX_DATA];
    //작업분할
    int divider=0;
    int index_i=0, index_j=0;
    int index_buf[MAX_DATA];
    //array
    int local_row_start, local_row_end;
    int isZero=1, isZero_gather;
    //control
    int i, j, k,l;
    int print=1;
    int done=0;

    //mpich2
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(processor_name, &namelen);

    //계산에 참여하는 프로세스 수는 ROW_MAX를 정확히 나눌수 있는 수 이어야한다.
    if(ROW_MAX % size !=0){
        printf("Error!\n"); fflush(stdout);
        MPI_Finalize();
        return -1;
    }

    //divider에 프로세스당 처리할 row의 수
    divider=ROW_MAX/size;

    if(rank==0){
        //parameter broadcasting
        printf("Initial Value : "); fflush(stdout);
        scanf("%lf", &init_value);
        printf("Subtracter : "); fflush(stdout);
        scanf("%lf", &subtracter);
        printf("Print results[yes=1,no=0]? "); fflush(stdout);
        scanf("%d", &print);
        //start clock
        starttime=MPI_Wtime();
        //broadcasting
        MPI_Bcast(&init_value, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        MPI_Bcast(&subtracter, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

        //(1) 작업분할, 각프로세스가 처리할 index의 범위를 할당
        for(i=0;i<size*2;i+=2){
            index_buf[i]=i/2*divider;
```

```
            if(i/2 == size-1)
                //마지막 프로세스는 나머지 Index를 다 처리
                index_buf[i+1]=ROW_MAX-1;
            else
                index_buf[i+1]=(i/2*divider)+(divider-1);
        }
        printf("### rank[%d]-[%s] - Received Index Range = [%d] ~ [%d]\n", rank, processor_name, index_buf[0], index_buf[1]);
        //index할당
        for(i=1;i<size;i++){
            buf[0]=index_buf[i*2];
            buf[1]=index_buf[i*2+1];
            MPI_Send(buf, 2, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
        //각 프로세스에 할당한 Index Range를 확인(디버깅)
        for(i=1;i<size;i++){
            MPI_Recv(char_buf, MAX_DATA ,MPI_CHAR, i, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            printf("%s\n",char_buf); fflush(stdout);
        }
        //배열초기화
        control_array(1, init_value, ROW_MAX, COL_MAX);
        //print
        if(print) control_array(0, 0, ROW_MAX, COL_MAX);

        //(2)작업시작
        local_row_start=index_buf[0];
        local_row_end=index_buf[1];

        //할당받은 index범위 내의 행렬 값들을 계산한다.
        while(1){
            isZero=1;
            cycle_counter++;
            for(i=local_row_start;i<=local_row_end;i++){
                for(j=0;j<COL_MAX;j++){
                    for(k=0;k<100;k++)
                        for(l=0;l<100;l++){
                            t_array[i][j] = t_array[i][j]-subtracter;
                        }
                        if(t_array[i][j]> 0) isZero=0;//계산 값중 0이하가 아닌것이 하나라도 있으면 0으로 셋
                }
            }
            //각 프로세스가 계산한 값을 array에 모두 저장, MPI_Gather
            MPI_Gather(t_array, ROW_MAX/size*COL_MAX, MPI_DOUBLE, array, ROW_MAX/size*COL_MAX, MPI_DOUBLE, 0, MPI_COMM_WORLD);
            //각 프로세스의 isZero값 더하여 종료조건 계산
            MPI_Reduce( &isZero, &isZero_gather, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
            if(print) control_array(0, 0, ROW_MAX, COL_MAX);
            if(isZero_gather==size)
                break;
        }
        printf("### rank[%d]-[%s] - Finished!!!\n", rank, processor_name); fflush(stdout);

        endtime=MPI_Wtime();
        for(i=1;i<size;i++){
            MPI_Recv(char_buf, MAX_DATA ,MPI_CHAR, i, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            printf("%s\n", char_buf);
        }
        printf("\n### Time = %f\n", endtime-starttime); fflush(stdout);

    }

    else{
        MPI_Bcast(&init_value, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        MPI_Bcast(&subtracter, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
        //배열초기화
        control_array(1, init_value, ROW_MAX, COL_MAX);
        //할당받은 Index Range를 서버에게 전달(디버깅)
        MPI_Recv(buf, 2,MPI_INT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        sprintf(char_buf, "### rank[%d]-[%s] - Received Index Range = [%d] ~ [%d]", rank,processor_name, buf[0], buf[1]);
        MPI_Send(char_buf, MAX_DATA, MPI_CHAR, 0, 0, MPI_COMM_WORLD);

        local_row_start = buf[0];
```

```c
        local_row_end = buf[1];
        while(done!=1){
            //할당받은 index범위 내의 행렬 값들을 계산한다.
            isZero=1;
            for(i=0;i<ROW_MAX/size;i++){
                for(j=0;j<COL_MAX;j++){
                    for(k=0;k<100;k++)
                        for(l=0;l<100;l++){
                            array[i][j] = array[i][j]-subtracter;
                        }
                        if(array[i][j] > 0) isZero=0;//계산 값중 0이하가 아닌것이 하나라도 있으면 0으로 셋
                }
            }
            MPI_Gather(array, ROW_MAX/size*COL_MAX, MPI_DOUBLE, array, ROW_MAX/size*COL_MAX, MPI_DOUBLE, 0, MPI_COMM_WORLD);
            MPI_Reduce( &isZero, &isZero_gather, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
            done=isZero;
        }
        sprintf(char_buf, "### rank[%d]-[%s] - Finished!!! ", rank, processor_name);
        MPI_Send(char_buf, MAX_DATA, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 1;
}

int control_array(int type, double value, int row, int col){
    //type==0 : print
    //type==1 : initialize with the value of the parameter, value
    int i,j;
    if(type==0) printf("### %d Cycle\n", cycle_counter);
    for(i=0;i<row;i++){
        for(j=0;j<col;j++){
            if(type==0)         { printf("%.2lf ", array[i][j]); fflush(stdout); }
            else if(type==1)    { array[i][j]=value; t_array[i][j]=value;}
            else                return -1;
        }
        if(type==0)             { printf("\n"); fflush(stdout);}
    }
    if(type==0) { printf("\n"); fflush(stdout); }
    return 1;
}
```


프로세스 수에 의한 계산 시간의 변화
계산에 걸린시간(초단위)
140
120
100
80
60
40
20
0
1
2
3
4
6
12
프로세스의 수
subtracter
0.01
0.005
0.003
0.001
0.0005
0.0003
0.0001

X축 : 계산에 사용한 감수(이 수를 이용하여 행렬의 초기값에서 0이 될 때 까지 뺀다)
Y축 : 계산에 참여한 프로세스(컴퓨터)의 수,
(프로세스수가 6이상일 경우는 1대의 컴퓨터에 2개 이상의 프로세스를 생성하여 처리함)

	0.01	0.005	0.003	0.001	0.0005	0.0003	0.0001
1	1.208702	2.445478	4.051017	12.122096	24.246863	40.43595	121.173168
2	0.664042	1.331564	2.213143	6.629686	13.23974	22.06656	66.152345
3	0.518425	1.037648	1.757446	5.201913	10.289312	17.21393	51.641147
4	0.399318	0.794012	1.316017	3.981628	7.90923	13.179899	39.318848
6	0.274386	0.541879	0.894247	2.719146	5.382371	8.856445	26.6861
12	0.302999	0.58894	0.964869	2.918447	5.733778	9.391288	33.206671