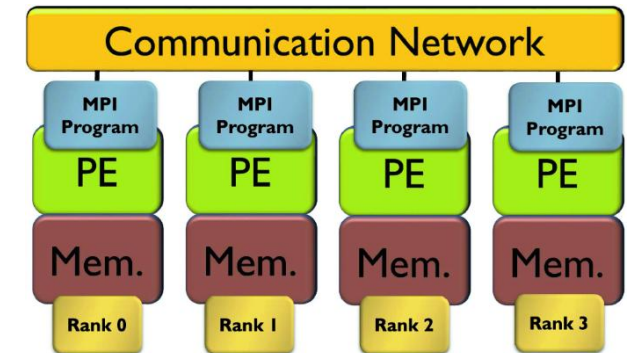
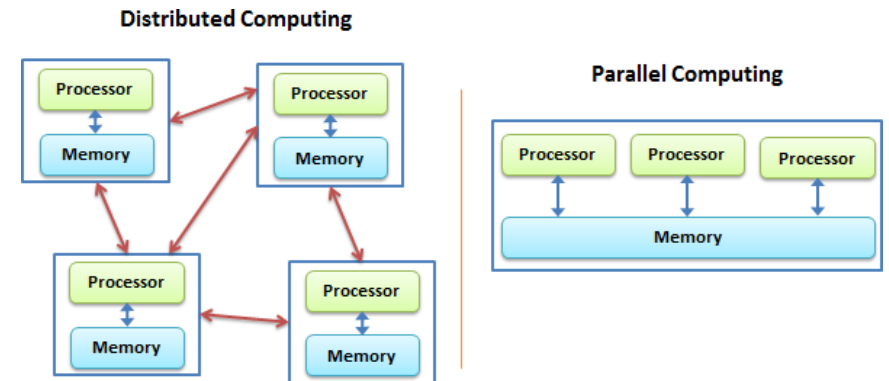
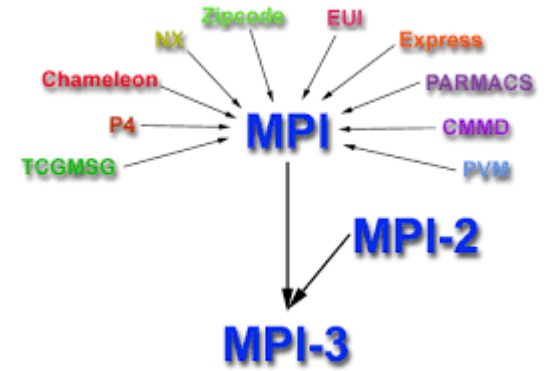


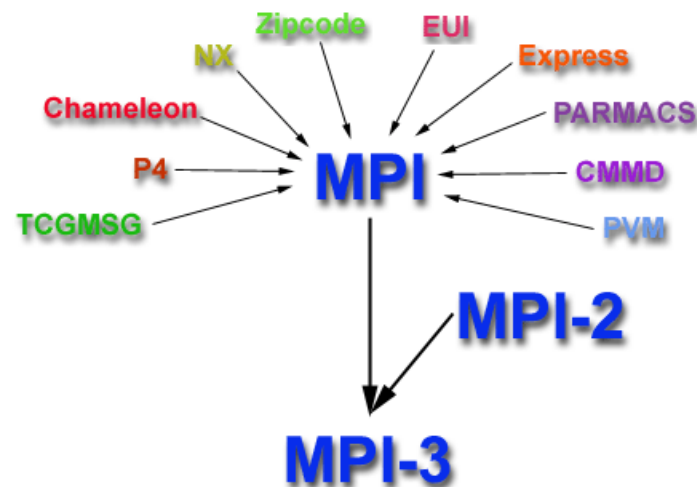
# MPI 병렬 프로그래밍



# MPI 튜토리얼 소개

## ● MPI의 역사(개략): A brief history of MPI

- 1990년대 이전에는 현재 MPI와 같은 스타일로 병렬 계산 프로그램을 작성할 수 없었음.
  - 서로 다른 컴퓨터 아키텍처에 대한 프로그래밍은 어렵고 지루하기 짝이 없는 일이고.
  - 몇 개의 라이브러리가 탄생했지만(PVM 등) 표준이 되지는 못함.
- 그 시대의 병렬 애플리케이션은 대부분 과학 및 연구 분야를 위한 것.
- 이 중 가장 일반적으로 채택된 라이브러리 모델은 **메시지 패싱 모델**. 이 모델을 사용하는 애플리케이션은 특정 작업을 수행하기 위해 프로세스 간에 메시지를 주고받는 방식.
- 이 모델은 병렬 애플리케이션에서 매우 잘 작동. 관리자 프로세스는 작업자 프로세스에 작업 명령이 포함된 메시지를 전달하여 작업자 프로세스에 작업을 할당할 수 있음.
- 예를 들어 병렬로 병합 정렬을 처리한다고 가정해 봅시다. 각 프로세스에서는 로컬 데이터를 정렬하고, 정렬된 로컬 데이터를 전체 병합하기 위해 인접한 프로세스에 결과를 전달함. 이처럼 많은 병렬 애플리케이션을 메시지 전달 모델로 표현할 수 있음.



# MPI 튜토리얼 소개

- 이 시기에 등장한 라이브러리는 메시지 패싱 모델이고 각 라이브러리 간의 기능 차이가 적었기 때문에 1992년 Supercomputing 컨퍼런스에서 각 라이브러리의 저자들이 모여 메시지 패싱의 실행 표준 인터페이스인 Message Passing Interface를 정의됨.
- 이를 계기로 프로그래머들은 모든 주요 병렬 아키텍처에 이식 가능한 병렬 애플리케이션을 작성할 수 있게 되었음.
- 또한, 현재도 일반적으로 사용되는 라이브러리에서 사용되는 기능이나 모델도 이때부터 등장함.
- 그리고 1994년에 이르러서야 완전한 인터페이스와 표준이 정의되었고, 이를 MPI-1이라고 칭함.
- 하지만 MPI는 특정 구현이 아닌 인터페이스의 정의임.
- 각 아키텍처에 맞는 인터페이스를 구현하는 것은 개발자에게 맡겨져 있었음.
- 하지만 다행히도 MPI의 완전한 구현을 사용할 수 있게 되기까지 1년 정도밖에 걸리지 않았음.
- 이 첫 번째 구현이 만들어진 후, MPI는 널리 채택되어 현재까지도 메시지 패싱 애플리케이션을 작성하는 데 있어 사실상 (사실상의) 표준이 되고 있음.

# MPI의 메시지 전달 모델 설계

- 병렬 프로그래밍의 메시지 전달 모델인 MPI 설계에서 몇 가지 고전적인 개념.
  - 첫 번째는 커뮤니케이터라는 개념
    - ✓ 커뮤니케이터(Communicators)는 서로 통신할 수 있는 프로세스 그룹. 이 프로세스 그룹에서는 각 프로세스에 고유한 랭크(rank)가 할당되어 있으며, 이 랭크로 서로를 구분하여 명시적으로 통신을 수행함.
- 통신의 기본은 프로세스 간의 송수신 작업.
  - 송신 측 프로세스는 랭크와 메시지를 식별하기 위한 고유한 태그를 사용하여 다른 프로세스에 메시지를 전송.
  - 수신 측 프로세스는 지정된 태그를 가진 메시지를 수신하여 데이터를 처리.
  - 이와 같이 하나의 발신자와 수신자가 관련된 통신을 **포인트 투 포인트 통신**이라고 함.
- 어떤 프로세스가 다른 모든 프로세스와 통신하고 싶을 때가 있음.
  - 예를 들어, 모든 관리자-프로세스 워커 프로세스에 데이터를 브로드캐스트하는 것을 생각해보자.
  - 일일이 송수신하는 코드를 일일이 작성하는 것은 번거롭고, 최적의 방법으로 네트워크를 이용하기도 어렵다.
  - MPI는 이러한 전체 프로세스를 포함한 다양한 종류의 집단 통신을 처리할 수 있음.

# MPI를 단일 머신에 설치하기

- MPI는 표준 규격을 의미하므로 MPI의 구현은 여러 가지가 있음. (MPICH, LAM/MPI, MPI/Pro, Open MPI 등)
- 여기서는 MPICH를 설치하는 방법을 소개.
- MPICH는 미국 아르곤 국립 연구소에서 주로 개발

## MPICH 설치

- 여기서는 3.3-2(2019년 11월 13일 릴리스)를 사용하며, tar.gz 파일을 다운로드하고 아래와 같이 압축을 푼다.

```
>>> tar -xzf mpich-3-3.2.tar.gz  
>>> cd mpich-3-3.2
```

- ./configure에서 make를 준비한다.
- 이때 머신의 권한이 없는 사용자 디렉토리에 설치하려면  
./configure --prefix=/installation/directory/path로 설치 디렉토리를 지정할 수 있다.
- Fortran 지원이 필요하지 않은 경우 ./configure --disable-fortran으로 설정한다.
- 사용 가능한 모든 옵션을 보려면 ./configure --help를 입력한다.

# MPI를 단일 머신에 설치하기

## Downloads | MPICH

←↻🏠🔒https://www.mpich.org/downloads/🔍☆⚙️📄🔖🔒

MPICH

High-Performance Portable MPI

HomeAboutDownloadsDocumentationSupportABI Compatibility InitiativeSupported Compilers


Downloads

MPICH is distributed under a **BSD-like license**. **NOTE: MPICH binary packages are available in many UNIX distributions and for Windows.** For example, you can search for it using “yum” (on Fedora), “apt” (Debian/Ubuntu), “pkg\_add” (FreeBSD) or “port”/“brew” (Mac OS). If available for your platform, this is likely the easiest installation method since it automatically checks for dependency packages and installs them. Otherwise you can use the [installation guide](#) for installing MPICH from the source code below.

Release	Platform	Download	Size
mpich-4.3.0 (stable release)	MPICH	<a href="#">[http]</a>	36 MB
hydra-4.3.0 (stable release)	Hydra (mpiexec)	<a href="#">[http]</a>	6 MB
libpmi-4.3.0 (stable release)	libpmi	<a href="#">[http]</a>	1 MB
mpich-testsuite-4.3.0 (stable release)	MPICH Testsuite	<a href="#">[http]</a>	3 MB
mpich-4.2.3 (legacy release)	MPICH	<a href="#">[http]</a>	38 MB
hydra-4.2.3 (legacy release)	Hydra (mpiexec)	<a href="#">[http]</a>	6 MB
libpmi-4.2.3 (legacy release)	libpmi	<a href="#">[http]</a>	1 MB
mpich-testsuite-4.2.3 (legacy release)	MPICH Testsuite	<a href="#">[http]</a>	3 MB

Search

MPICH2 was awarded an R&D100 award in 2005



“The Oscars of Invention” – The Chicago Tribune For 45 years, the prestigious R&D 100 Awards have been helping companies provide the important initial push a new product needs to compete successfully in the marketplace. The winning of an R&D 100 Award provides a mark of excellence known to industry, government, and academia as proof that the product is one of the most innovative ideas of the year.[Continue reading →](#)

# MPI를 단일 머신에 설치하기

## MPICH 설치

- 여기서는 3.3-2(2019년 11월 13일 릴리스)를 사용하며, tar.gz 파일을 다운로드하고 아래와 같이 압축을 푼다.

```
>>> ./configure
Configuring MPICH version 3.3.2
Running on system: Linux localhost.localdomain 5.8.18-100.fc31.x86_64 #1 SMP Mon Nov 2 20:32:55
UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
checking build system type... x86_64-unknown-linux-gnu
```

- configure가 종료되고 "Configuration completed."라고 표시되면 `make; sudo make install`을 사용하여 MPICH2를 빌드 및 설치한다.

```
>>> make; sudo make install
make
make all-recursive
```

# MPI를 단일 머신에 설치하기

## MPICH 설치

- 성공하면 `mpiexec --version`으로 설치한 정보를 출력할 수 있습니다.

```
>>> mpiexec --version
HYDRA build details:
  Version:                3.3.2
  Release Date:           Tue Nov 12 21:23:16 CST 2019
  CC:                     gcc
  CXX:                    g++
  F77:                    gfortran
  F90:                    gfortran
```



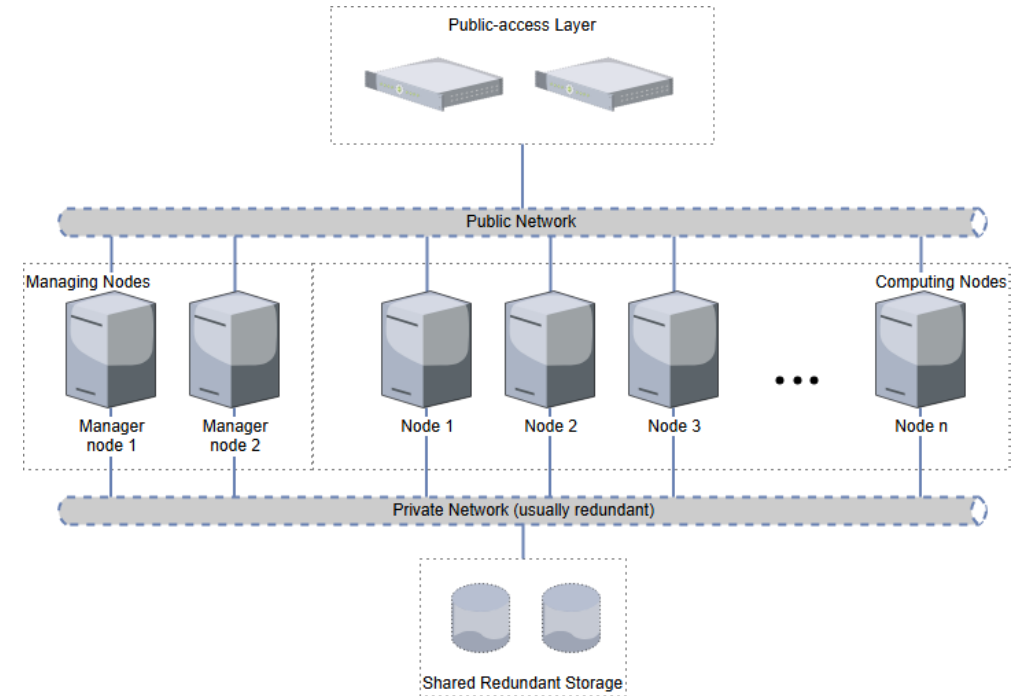
# MPI를 단일 머신에 설치하기

## MPICH 설치

- 성공하면 `mpiexec --version`으로 설치한 정보를 출력할 수 있습니다.

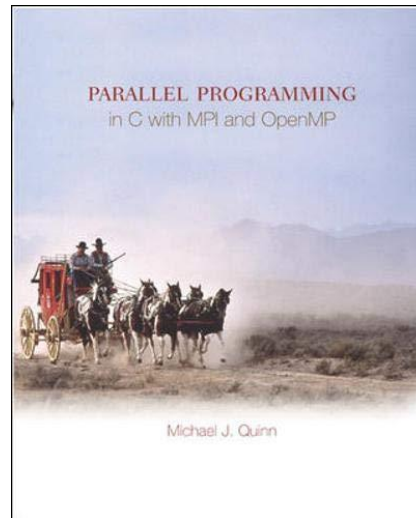
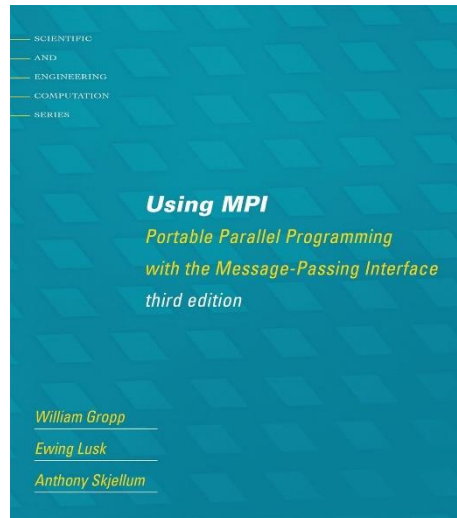
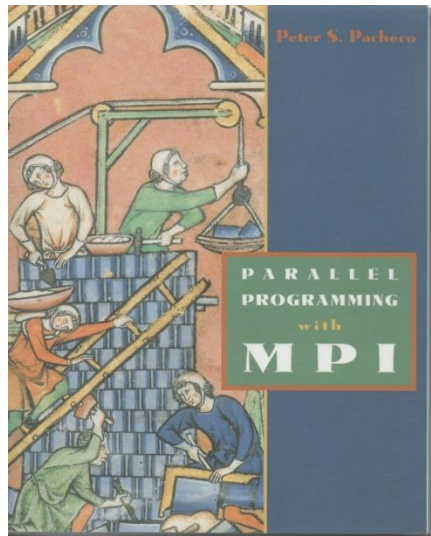
```
>>> mpiexec --version
HYDRA build details:
  Version:                3.3.2
  Release Date:           Tue Nov 12 21:23:16 CST 2019
  CC:                     gcc
  CXX:                    g++
  F77:                    gfortran
  F90:                    gfortran
```

# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)



- 이제 같은 작업을 한 대의 컴퓨터가 아닌 LAN에 연결된 노드에서 실행할 수 있도록 해 보자
- 단순화하기 위해 이번 레슨에서는 두 대의 컴퓨터를 고려, 더 많은 노드를 추가하는 것도 같은 방식으로 할 수 있음
- 리눅스 머신을 사용한다고 가정하고 두 대의 컴퓨터를 관리자(manager)라고 하고, 다른 한 대의 컴퓨터를 작업자(worker)라고 하자.

1. Parallel Programming with MPI 1st Edition
2. Using MPI - 3rd Edition and Using Advanced MPI - 1st Edition
3. Parallel Programming in C with MPI and OpenMP
4. MPI: The Complete Reference
5. Beginning MPI (An Introduction in C)



# MPI를 단일 머신에 설치하기

## Downloads | MPICH

←↻🏠🔒https://www.mpich.org/downloads/🔍☆⚙️📄🔖🔒

MPICH

High-Performance Portable MPI

HomeAboutDownloadsDocumentationSupportABI Compatibility InitiativeSupported Compilers


Downloads

MPICH is distributed under a **BSD-like license**. **NOTE: MPICH binary packages are available in many UNIX distributions and for Windows.** For example, you can search for it using “yum” (on Fedora), “apt” (Debian/Ubuntu), “pkg\_add” (FreeBSD) or “port”/“brew” (Mac OS). If available for your platform, this is likely the easiest installation method since it automatically checks for dependency packages and installs them. Otherwise you can use the [installation guide](#) for installing MPICH from the source code below.

Release	Platform	Download	Size
mpich-4.3.0 (stable release)	MPICH	<a href="#">[http]</a>	36 MB
hydra-4.3.0 (stable release)	Hydra (mpiexec)	<a href="#">[http]</a>	6 MB
libpmi-4.3.0 (stable release)	libpmi	<a href="#">[http]</a>	1 MB
mpich-testsuite-4.3.0 (stable release)	MPICH Testsuite	<a href="#">[http]</a>	3 MB
mpich-4.2.3 (legacy release)	MPICH	<a href="#">[http]</a>	38 MB
hydra-4.2.3 (legacy release)	Hydra (mpiexec)	<a href="#">[http]</a>	6 MB
libpmi-4.2.3 (legacy release)	libpmi	<a href="#">[http]</a>	1 MB
mpich-testsuite-4.2.3 (legacy release)	MPICH Testsuite	<a href="#">[http]</a>	3 MB

Search

MPICH2 was awarded an R&D100 award in 2005



“The Oscars of Invention” – The Chicago Tribune For 45 years, the prestigious R&D 100 Awards have been helping companies provide the important initial push a new product needs to compete successfully in the marketplace. The winning of an R&D 100 Award provides a mark of excellence known to industry, government, and academia as proof that the product is one of the most innovative ideas of the year.[Continue reading →](#)

# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)

## ❖ Step 1 : **hosts** 준비 - Configure your **hosts** file

- **hosts file**를 준비하자. 이것은 컴퓨터의 운영 체제가 호스트 이름을 IP 주소에 매핑하는 데 사용됩니다. 다음은 예시이다.

```
$ cat /etc/hosts
```

```
127.0.0.1    localhost
172.50.88.34 worker
```

- 이것은 Manager의 **hosts**이고, **worker**는 계산을 수행하고자 하는 다른 머신의 이름이다. 마찬가지로 Worker에서 **manager**도 마찬가지입니다.

## ❖ Step 2: 사용자 생성 - Create a new user

- 기존 사용자 계정으로 클러스터를 운영할 수도 있다. 하지만 설정을 단순화하기 위해 새로운 사용자 계정을 생성하는 것이 좋다.
- 모든 호스트에서 사용자 **mpiuser**를 생성한다.

```
$ sudo adduser mpiuser
```

- 프롬프트에 따라 진행. **adduser**가 아닌 **useradd** 명령을 사용하면 홈 디렉터리가 생성되지 않으므로 사용하지 마세요.

# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)

## ❖ Step 3: SSH 설정 - Setting up SSH

- 머신 간 통신이 가능하도록 SSH를 설정하고, NFS를 사용하여 데이터를 공유한다.

```
$ sudo apt-get install openssh-server
```

- 먼저 이 호스트에서 사용자를 mpiuser로 전환합니다.

```
$ su - mpiuser
```

- 이미 ssh 서버가 설치되어 있기 때문에 다른 머신에 ssh username@hostname으로 로그인할 수 있을 것이다.
- 비밀번호를 묻지 않도록 키를 생성하여 다른 머신의 authorized\_keys 목록에 복사한다. (이 과정은 공개키 인증을 가능하게 하는 과정이다. ➔비밀번호 인증을 금지하는 것이 아님을 주의해야 한다).

```
$ ssh-keygen -t dsa
```

- 이 예에서는 DSA 키를 생성했지만 RSA 키를 생성할 수도 있다. 더 높은 보안을 원한다면 RSA를 사용해도 좋지만 DSA로도 충분하다. 그리고 생성한 키를 다른 컴퓨터에 추가하고, worker 노드에 대해 복사한다.

```
$ ssh-copy-id worker #ip-address may also be used
```

# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)

- 각 Worker 머신과 자신의 사용자(localhost)에 대해 여기까지의 단계를 수행.
- 이제 openssh-server가 설정되어 Worker와 안전하게 통신할 수 있게 되었음. 모든 머신에 ssh를 한 번 실행한다.
- 그리고 known\_hosts 목록에 다른 호스트의 핑거프린트를 추가한다. 이 단계는 ssh 로그인에 문제가 발생하기 쉽기 때문에 중요한 단계이다.
- 비밀번호 없이 ssh를 할 수 있도록 한다.

```
$ eval `ssh-agent`  
$ ssh-add ~/.ssh/id_dsa
```

- 이제 비밀번호 프롬프트 없이 다른 컴퓨터에 로그인할 수 있을 것임.

```
$ ssh worker
```

- Note - 모든 워커 머신에서 mpiuser를 공통 사용자 계정으로 생성했다고 가정하며, Manager와 Worker에서 다른 이름의 사용자 계정을 생성한 경우 적절한 사용자 이름을 지정해야 함.

# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)

## ❖ Step 4: NFS 설정 - Setting up NFS

- manager는 NFS를 통해 디렉토리를 공유하고, 이를 worker가 마운트하여 데이터를 주고받는다.

### NFS-Server

- 패키지를 설치한다.

```
$ sudo apt-get install nfs-kernel-server
```

- 아직 mpiuser에 로그인한 상태라고 가정하고 cloud 라는 이름의 공유 폴더를 생성합니다.

```
$ mkdir cloud
```

- cloud를 공유하기 위해 /etc/exports를 다음과 같이 설정합니다.

```
$ cat /etc/exports  
/home/mpiuser/cloud *(rw,sync,no_root_squash,no_subtree_check)
```

```
/home/mpiuser/cloud 12.33.44.55(rw,sync,no_root_squash,no_subtree_check)
```



# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)

- \*부분에는 이 폴더를 공유하고자 하는 IP 주소를 지정할 수 있다. 이번에는 간단하게 \*로 설정하였음.
- rw: read/write를 허용하며, read만 할당하고 싶다면 ro를 설정하면 된다.
- sync: 변경 사항을 커밋한 후에만 공유 디렉토리에 변경 사항을 적용한다.
- no\_subtree\_check: 서브트리를 체크하지 않는다. 공유 디렉토리가 큰 파일 시스템의 하위 디렉터리인 경우, nfs는 그 위에 있는 모든 디렉토리의 권한과 세부 정보를 확인하기 위해 스캔을 수행한다. 서브트리 검사를 비활성화하면 NFS의 안정성은 향상되지만 보안이 저하될 수 있다.
- no\_root\_squash: root 계정으로 폴더에 연결할 수 있도록 허용한다.
- 참고 <https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-on-ubuntu-12-04>

➤ NFS를 실행합니다.

```
$ exportfs -a
```

- /etc/exports를 변경한 경우 이 명령이 필요함.
- 필요에 따라 nfs 서버를 재시작한다.

```
$ sudo service nfs-kernel-server restart
```

# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)

## NFS-worker

➤ 다음을 수행합니다.

```
$ sudo apt-get install nfs-common
```

➤ worker에서 같은 이름의 디렉터리를 생성합니다.

```
$ mkdir cloud
```

➤ 그리고 다음과 같이 공유 디렉터리를 마운트합니다.

```
$ sudo mount -t nfs manager:/home/mpiuser/cloud ~/cloud
```

➤ 정상적으로 장착되었는지 확인해 봅시다.

```
$ df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
manager:/home/mpiuser/cloud	49G	15G	32G	32%	/home/mpiuser/cloud

# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)

- 재부팅을 위해 공유 디렉터리를 수동으로 마운트할 필요가 없도록 하려면 /etc/fstab 파일에 다음과 같은 항목을 생성한다.

```
$ cat /etc/fstab
#MPI CLUSTER SETUP
manager:/home/mpiuser/cloud /home/mpiuser/cloud nfs
```

## Step 5: MPI 프로그램 실행 - Running MPI programs

- 이제 MPICH2 설치 패키지 mpich2/examples/cpi에 포함된 샘플 프로그램을 병렬로 실행하여 정상 여부를 확인해보자.
- 자신의 코드를 컴파일하고 싶다면, 코드를 mpi\_sample.c로 설정하고 아래와 같은 방법으로 컴파일하여 mpi\_sample을 생성할 수 있다.

```
$ mpicc -o mpi_sample mpi_sample.c
```

- 그리고 실행 파일을 공유 디렉터리 cloud에 복사합니다. 물론 NFS 공유 디렉터리 내에서 코드를 컴파일해도 무방함.

```
$ cd cloud/
$ pwd
/home/mpiuser/cloud
```

- 자신의 머신(관리자)에서만 실행할 때는 다음과 같이 합니다.

```
$ mpirun -np 2 ./cpi # No. of processes = 2
```

# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)

## ➤ 클러스터로 실행해 보자

```
$ mpirun -np 5 -hosts worker,localhost ./cpi  
#호스트명 대신 IP 주소도 괜찮습니다.
```

## ➤ 미리 준비된 호스트 파일을 사용할 수도 있습니다.

```
$ mpirun -np 5 --hostfile mpi_file ./cpi
```

## ➤ 예시, hostfile

```
# This is an example hostfile.  Comments begin with #  
# 도메인 주소 혹은 IP 주소, 혹은 등록된 호스트 이름  
# The following node is a single processor machine:  
foo.example.com  
# The following node is a dual-processor machine:  
bar.example.com slots=2  
# The following node is a quad-processor machine, and we absolutely  
# want to disallow over-subscribing it:  
yow.example.com slots=4 max-slots=4
```

# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)

- 이제 관리자가 연결한 머신에서 MPI 프로그램이 실행될 것임!

## 일반적인 오류 및 팁 - Common errors and tips

- 실행파일을 실행하려는 모든 머신에서 MPI의 버전이 동일한지 확인.
- manager의 hosts 파일에는 manager와 워커 노드의 로컬 네트워크 IP 주소를 기술한다. worker에는 manager와 자신의 항목이 포함되어야 한다.
- 예를 들어, manager에는 다음과 같은 hosts가 필요합니다.

```
$ cat /etc/hosts
127.0.0.1      localhost
#127.0.1.1    1944

#MPI CLUSTER SETUP
172.50.88.22   manager
172.50.88.56   worker1
172.50.88.34   worker2
172.50.88.54   worker3
172.50.88.60   worker4
172.50.88.46   worker5
```

# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)

- 이때 worker3에 필요한 최소 hosts는 다음과 같다.

```
$ cat /etc/hosts
127.0.0.1      localhost
#127.0.1.1    1947

#MPI CLUSTER SETUP
172.50.88.22   manager
172.50.88.54   worker3
```

- MPI를 사용하여 프로세스를 병렬로 실행하려는 경우, 로컬에서만, 로컬 노드와 원격 노드의 조합으로 프로세스를 실행할 수 있다. 원격에서만 프로세스를 실행할 수는 없다.
- 예시로, 다음 호출은 정상.

```
$ mpirun -np 10 --hosts manager ./cpi
# To run the program only on the same manager node
```

- 예시로, 다음도 호출은 정상.

```
$ mpirun -np 10 --hosts manager,worker1,worker2 ./cpi
# To run the program on manager and worker nodes.
```

# LAN 상에서 MPI 클러스터 구축(Running an MPI Cluster within a LAN)

➤ 하지만 manager에서 다음 호출을 할 수 없다(worker1에서 호출하면 좋겠지만).

```
$ mpirun -np 10 --hosts worker1 ./cpi  
# Trying to run the program only on remote worker
```

# 첫 MPI 프로그램 - MPI Hello World

- 여기서는 기본적인 MPI Hello World 애플리케이션을 만드는 방법과 MPI 프로그램을 실행하는 방법을 설명
- MPI 초기화 및 여러 프로세스에 걸쳐 MPI 작업을 실행하는 기본 사항을 설명.

## Hello, World!: Hello world code examples

[mpi\\_hello\\_world.c](https://github.com/mpitutorial/mpitutorial/tree/gh-pages/tutorials/mpi-hello-world/code) <https://github.com/mpitutorial/mpitutorial>

<https://github.com/mpitutorial/mpitutorial/tree/gh-pages/tutorials/mpi-hello-world/code>

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    // MPI환경 초기화
    MPI_Init(NULL, NULL);


    // Get the number of processes
    // 프로세스의 수를 얻는다.
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    // 이 커뮤니케이터에서 자신의 rank를 얻는다.
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    // 이 프로세서의 이름을 얻는다
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    // Hello world 메시지 출력
    printf("Hello world from processor %s, rank %d out of %d\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    // MPI 환경을 마무리
    MPI_Finalize();
}
```





# 첫 MPI 프로그램 - MPI Hello World

- MPI 프로그램에서는 MPI 헤더를 포함해야 함(`#include <mpi.h>`).
- 다음으로 MPI 환경을 초기화해야 한다.

## `MPI_Init(int* argc, char*** argv)`

- **MPI\_Init**: MPI의 글로벌 변수와 내부 변수를 초기화 및 설정한다. 이 프로그램은 모든 프로세스에 각각 Rank를 할당하고, 이를 모두 포함하는 커뮤니케이터를 생성한다.
- MPI\_Init은 특별히 설정할 인수가 없다. 추가 파라미터는 향후 구현에서 필요할 경우를 대비하여 예약되어 있다.
- **MPI\_Init**뒤에 두 개의 함수를 호출함. 이 두 함수는 거의 모든 MPI 프로그램에서 호출되는 코드이다.

## `MPI_Comm_size( MPI_Comm communicator, int* size)`

- **MPI\_Comm\_size**는 커뮤니케이터의 크기를 반환한다.
- 샘플에서 인수로 제공한 `MPI_COMM_WORLD`(MPI에 의해 초기화되는 변수)는 작업의 모든 프로세스를 포함하므로, 이 호출은 작업에 요청한 프로세스 수를 반환한다.

## MPI\_Comm\_rank(MPI\_Comm communicator, int\* rank)

- **MPI\_Comm\_rank**: 커뮤니케이터 내 해당 프로세스의 rank를 반환한다.
- 커뮤니케이터 내의 각 프로세스에는 0부터 순서대로 rank가 할당된다. 랭크(rank)는 주로 메시지 송수신에 사용된다.

- 다음 함수는 실제 코드에서 많이 사용되지는 않음 → 주요 확인용.

## MPI\_Get\_processor\_name(char\* name, int\* name\_length)

- **MPI\_Get\_processor\_name**은 프로세서의 이름을 가져옴.

## MPI\_Finalize()

- MPI\_Finalize는 MPI 환경을 정리하는 함수.
- 이 함수 이후에는 MPI 함수를 사용할 수 없다.

# 애플리케이션 실행하기: Running the MPI hello world application

## ➤ Makefile 만들기 (빌드 만들기)

```
EXECS=mpi_hello_world
```

```
MPICC?=mpicc
```

```
all: ${EXECS}
```

```
mpi_hello_world: mpi_hello_world.c
```

```
    ${MPICC} -o mpi_hello_world mpi_hello_world.c
```

```
clean:
```

```
    rm ${EXECS}
```

- 이 makefile은 MPICC 환경 변수가 설정되어 있다면 이를 사용하며, MPICH2를 로컬 디렉토리에 설치한 경우 MPICC 환경 변수를 적절한 mpicc 바이너리 경로를 가리키도록 설정해야 한다.
- mpicc는 필요한 library와 include를 수행해 주는 gcc의 래퍼이다.

```
>>> export MPICC=/home/kendall/bin/mpicc
```

```
>>> make
```

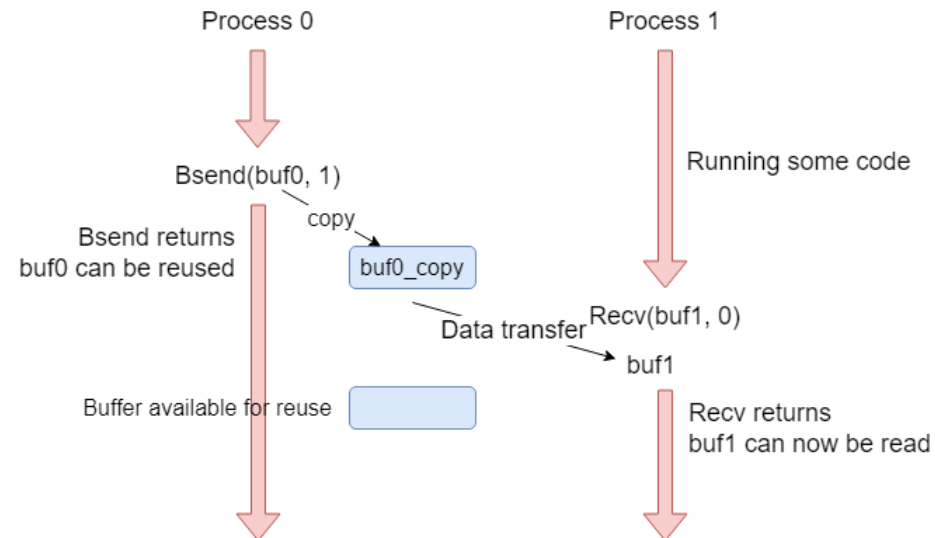
```
/home/kendall/bin/mpicc -o mpi_hello_world mpi_hello_world.c
```

# 애플리케이션 실행하기: Running the MPI hello world application

- 프로그램 컴파일이 완료되어 실행할 준비가 완료! 여러 노드 클러스터에서 MPI 프로그램을 실행하려면 호스트 파일을 설정해야 한다는 점에 유의하자.
- 단일 머신에서 MPI를 실행하는 경우 다음 정보는 무시해도 됨
- `host_file`에는 MPI 작업이 실행될 모든 컴퓨터의 호스트명이 포함되어 있다.
- 실행을 용이하게 하기 위해 이러한 호스트에 SSH 액세스가 가능한지 확인해야 합니다.
- 또한 SSH의 암호 프롬프트를 피하기 위해 [set up an authorized keys file](#)을 설정해야 한다. 예를 들어 `host_file`의 예를 들어보자.

```
>>> cat host_file
cetus1
cetus2
cetus3
cetus4
```

# Blocking point-to-point communication



## MPI의 send와 receive - MPI Send and Receive

- 송수신(send, recv)은 MPI의 가장 기본적인 개념으로, MPI의 거의 모든 기능은 send, recv를 통해 구현할 수 있다.
- 블로킹 송수신에 대해 설명하고 MPI의 데이터 전송의 기본 개념을 설명.

# MPI를 이용한 송수신 개요: Overview of sending and receiving with MPI

- 먼저 프로세스 A가 프로세스 B에게 메시지를 보내고 싶다고 가정. 프로세스 A는 프로세스 B에 보낼 데이터를 모두 하나로 묶어 (팩) 버퍼에 저장함. 이 버퍼는 데이터를 하나의 메시지로 묶어 놓았기 때문에 봉투(envelopes)라고 부르기도 함(우편 편지가 봉투에 포장되는 것과 같은 이치).
- 데이터가 버퍼에 담겨지면 통신 장치(대부분 네트워크)가 메시지를 적절히 라우팅한다. 메시지의 목적지는 프로세스의 rank에 따라 정의됨. 메시지는 B로 라우팅됨. 그런 다음 프로세스 B는 A로부터 데이터를 수신할 의사가 있음을 명확하게 통지 (acknowledge)해야 한다. 통지가 완료되면 데이터가 전송된 것으로 간주하고, 프로세스 A에 데이터가 전송되었음을 알리고 A의 차단이 완료되어 다음 처리로 넘어갈 수 있다.
- 다음은 A가 B에게 여러 종류의 메시지를 전송한다고 가정하면, B가 이러한 메시지를 구분하기 위해 특별한 방법을 취하지 않더라도 MPI는 메시지에 ID(태그 - tags)를 붙여 송수신할 수 있다. 프로세스 B는 특정 태그의 메시지만 요청할 수 있고, 다른 태그의 메시지는 B가 수신 요청을 할 때까지 네트워크 계층에 의해 버퍼링된다.

# MPI를 이용한 송수신 개요: Overview of sending and receiving with MPI

- MPI send 함수와 recv 함수의 정의를 살펴보자.

```
MPI_Send(  
    void* data,  
    int count,  
    MPI_Datatype datatype,  
    int destination,  
    int tag,  
    MPI_Comm communicator)
```

```
MPI_Recv(  
    void* data,  
    int count,  
    MPI_Datatype datatype,  
    int source,  
    int tag,  
    MPI_Comm communicator,  
    MPI_Status* status)
```

- 첫 번째 인수는 데이터 버퍼의 주소.
- 두 번째와 세 번째 인수는 버퍼에 있는 요소의 수와 유형. MPI\_Send는 정확한 수의 요소를 전송하고 MPI\_Recv는 "최소" 개수의 요소를 수신
- 네 번째와 다섯 번째 인수는 send 또는 recv할 프로세스의 rank와 태그를 지정한다.
- 여섯 번째 인수는 커뮤니케이터를 지정하고, recv에만 포함된 마지막 인자 status는 수신된 메시지에 대한 정보 포인터이다.



# 기본 MPI 데이터 유형 - Elementary MPI datatypes

- MPI\_Send와 MPI\_Recv 함수는 메시지의 데이터 구조를 C 언어의 데이터 타입이 아닌 더 높은 수준의 데이터 타입으로 지정할 수 있음.
- 예를 들어, 프로세스가 하나의 정수를 전송하는 경우 count1과 MPI\_INT 데이터 타입을 사용한다.
- 기본적인 MPI 데이터 타입과 이에 해당하는 C 언어의 데이터 타입은 다음과 같다.

MPI 데이터 유형	C언어 데이터 유형
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	char

이 데이터 유형은 초보자를 위한 MPI 튜토리얼에서만 사용.

기초를 배운 후에는 복잡한 메시지를 처리하기 위해 자신만의 MPI 데이터 유형을 만들 수 있음

# MPI send / recv program

## send\_recv.c.

```
// Find out rank, size
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

int number;
if (world_rank == 0) {
    number = -1;
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
} else if (world_rank == 1) {
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n",
           number);
}
```

- MPI\_Comm\_rank와 MPI\_Comm\_size는 월드 사이즈와 프로세스 rank를 구하는 함수이다.
- 프로세스 0은 정수 데이터를 -1로 초기화하고 이 값을 프로세스 1에 전송한다.
- 명령문에서 알 수 있듯이 프로세스 1은 if else if로 분기하여 숫자를 수신하고, 수신한 값도 출력한다.
- 송수신하고자 하는 것은 하나의 INT뿐이므로 각 프로세스는 하나의 MPI\_INT를 송수신한다.
- 또한, 메시지를 식별하기 위해 태그 번호 0을 지정했다.
- 전송되는 메시지 종류는 한 종류이므로 프로세스는 태그 번호에 이미 정의된 상수 MPI\_ANY\_TAG를 사용해도 된다.

# MPI send / recv program

➤ Mpirun을 위한 run.py를 사용하여 실행할 수도 있다.

```
>>> ./run.py send_recv  
mpirun -n 2 ./send_recv  
Process 1 received number -1 from process 0
```

# MPI ping - MPI ping pong program

- 다음 예제는 ping 프로그램. 이 예제에서 프로세스는 MPI\_Send와 MPI\_Recv를 사용하여 중지될 때까지 메시지를 주고받는다 (ping\_pong.c 참조). 코드의 주요 부분은 다음과 같다.

```
// Ping pong example with MPI_Send and MPI_Recv. Two processes
//ping pong a number back and forth, incrementing it until it reaches
a given value.
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
    const int PING_PONG_LIMIT = 10;

    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    // Find out rank, size
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // We are assuming 2 processes for this task
    if (world_size != 2) {
        fprintf(stderr, "World size must be two for %s\n", argv[0]);
        MPI_Abort(MPI_COMM_WORLD, 1);
    }
}
```

```
int ping_pong_count = 0;
int partner_rank = (world_rank + 1) % 2;
while (ping_pong_count < PING_PONG_LIMIT) {
    if (world_rank == ping_pong_count % 2) {
        // Increment the ping pong count before you send it
        ping_pong_count++;
        MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0,
MPI_COMM_WORLD);
        printf("%d sent and incremented ping_pong_count %d to %d\n",
world_rank, ping_pong_count, partner_rank);
    } else {
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0,
MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        printf("%d received ping_pong_count %d from %d\n",
world_rank, ping_pong_count, partner_rank);
    }
}
MPI_Finalize();
}
```

# MPI ping - MPI ping pong program

- 코드의 주요 부분은 다음과 같다.

```
int ping_pong_count = 0;
int partner_rank = (world_rank + 1) % 2;
while (ping_pong_count < PING_PONG_LIMIT) {
    if (world_rank == ping_pong_count % 2) {
        // Increment the ping pong count before you send it
        ping_pong_count++;
        MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
        printf("%d sent and incremented ping_pong_count "
               "%d to %d\n", world_rank, ping_pong_count,
               partner_rank);
    } else {
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("%d received ping_pong_count %d from %d\n",
               world_rank, ping_pong_count, partner_rank);
    }
}
```

- 이 코드는 두 개의 프로세스로만 실행된다고 가정.
- 첫 번째 프로세스는 간단한 연산을 통해 상대방의 rank를 결정한다. ping\_pong\_count는 0으로 초기화하고, 프로세스의 각 전송 단계마다 +1씩 증가한다. 그리고 송신 측과 수신 측을 번갈아 가며 담당한다.
- 마지막으로 limit 횟수에 도달하면(내 코드에서는 10) 프로세스가 작동을 멈춘다.

# MPI ping - MPI ping pong program

➤ 출력은 다음과 같다.

```
>>> ./run.py ping_pong
0 sent and incremented ping_pong_count 1 to 1
0 received ping_pong_count 2 from 1
0 sent and incremented ping_pong_count 3 to 1
0 received ping_pong_count 4 from 1
0 sent and incremented ping_pong_count 5 to 1
0 received ping_pong_count 6 from 1
0 sent and incremented ping_pong_count 7 to 1
0 received ping_pong_count 8 from 1
0 sent and incremented ping_pong_count 9 to 1
0 received ping_pong_count 10 from 1
1 received ping_pong_count 1 from 0
1 sent and incremented ping_pong_count 2 to 0
1 received ping_pong_count 3 from 0
1 sent and incremented ping_pong_count 4 to 0
1 received ping_pong_count 5 from 0
1 sent and incremented ping_pong_count 6 to 0
1 received ping_pong_count 7 from 0
1 sent and incremented ping_pong_count 8 to 0
1 received ping_pong_count 9 from 0
1 sent and incremented ping_pong_count 10 to 0
```

# Ring Program

- MPI\_Send와 MPI\_Recv를 사용하여 두 개 이상의 프로세스에서 통신하는 것을 생각해 보자.
- 이 예시에서는 값을 모든 프로세스에서 링 모양으로 전달할 것이다. ring.c를 살펴보자. 코드의 주요 부분은 다음과 같습니다.

```
int token;
if (world_rank != 0) {
    MPI_Recv(&token, 1, MPI_INT, world_rank - 1, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process %d received token %d from process %d\n",
           world_rank, token, world_rank - 1);
} else {
    // rank0은 토큰을 -1로 초기화한다
    token = -1;
}
MPI_Send(&token, 1, MPI_INT, (world_rank + 1) % world_size,
         0, MPI_COMM_WORLD);

// 여기서 프로세스 0은 링크의 마지막 수신 처리를 수행한다.
if (world_rank == 0) {
    MPI_Recv(&token, 1, MPI_INT, world_size - 1, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process %d received token %d from process %d\n",
           world_rank, token, world_size - 1);
}
```

- 여기서 프로세스 0은 링크에서 마지막 수신 처리를 수행한다. 링 프로그램은 프로세스 0에서 값을 초기화하고, 그 값을 다음 프로세스에 전달.
- 프로세스 0이 마지막 프로세스로부터 값을 받으면 프로그램이 종료. 데드락이 발생하지 않도록 특수한 로직으로 되어 있음.
- 프로세스 0은 (마지막 프로세스로부터) 값을 차단적으로 수신하기 전에 첫 번째 전송을 완료한 것으로 보장. 다른 모든 프로세스는 링을 따라 값을 전달.
- MPI\_Recv(인접한 낮은 Rank의 프로세스로부터 수신)으로 값을 받은 후 MPI\_Send(인접한 높은 Rank의 프로세스에 값 전송)를 호출한다.
- 메시지가 전송될 때까지 블로킹(Blocking)한다. 따라서 printf는 값이 전달되는 순서대로 실행되며, 5개의 프로세스를 사용하는 경우 출력은 다음과 같다.

```
>>> ./run.py ring
Process 1 received token -1 from process 0
Process 2 received token -1 from process 1
Process 3 received token -1 from process 2
Process 4 received token -1 from process 3
Process 0 received token -1 from process 4
```

[Introduction to Parallel Programming with MPI](#)

[Cornell Virtual Workshop: MPI Point-to-Point](#)

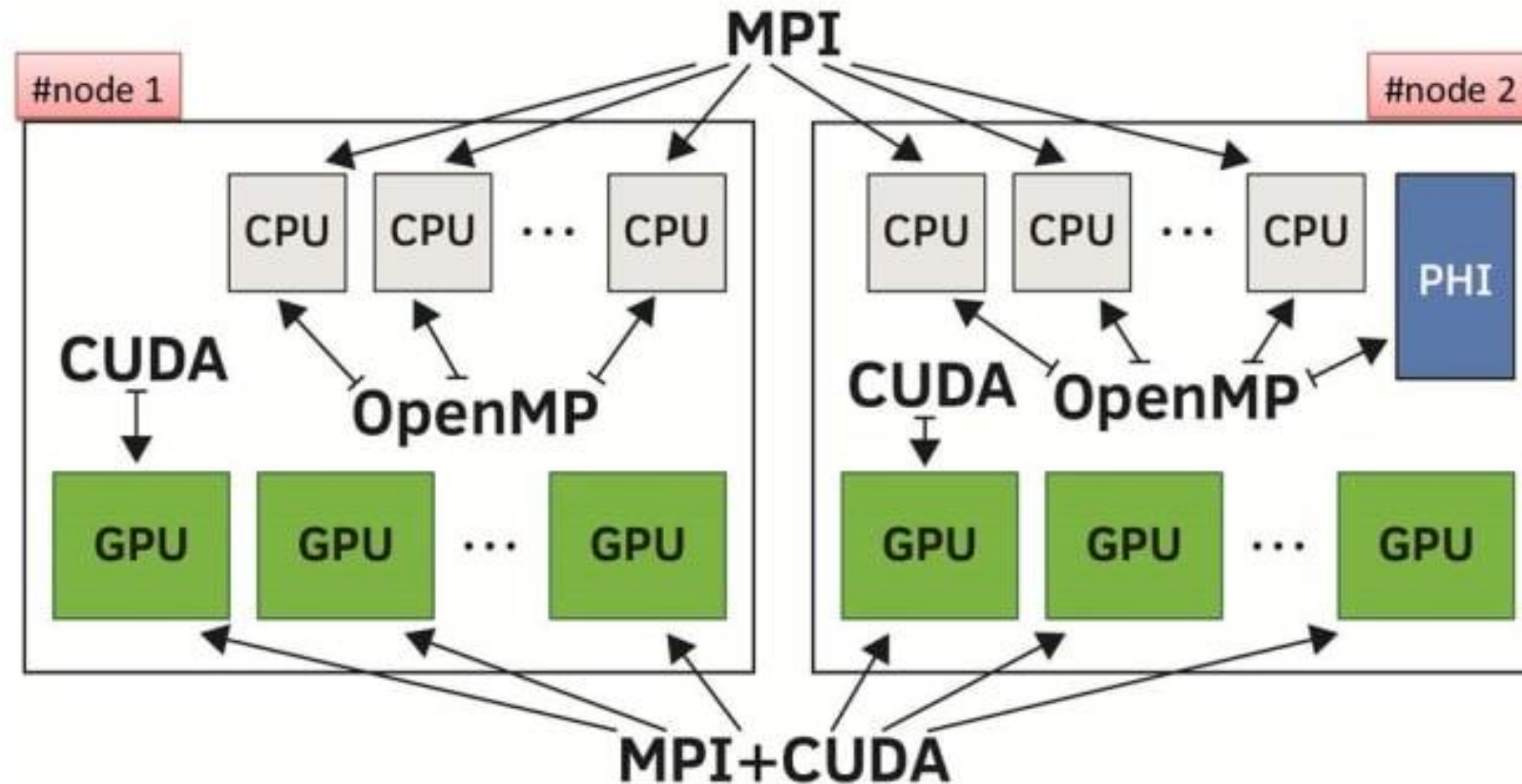
[Parallel Programming With MPI :: High Performance Computing](#)

[MPIの環境構築や基本コマンドのまとめ #並列処理 - Qiita](#)



# Parallel Technologies: Levels of Parallelism

## Parallel technologies: levels of parallelism



How to control hybrid hardware:  
MPI – OpenMP – CUDA - OpenCL ...