NCHC Taiwania HPC系統
使用者操作說明

2018/11/13

章節內容

- 介紹
- Taiwania HPC系統
- 登入系統方法
- Linux基本指令
- Compile/Link
- PBS PRO Job操作
- 平行程式範例

介紹

- 國家高速網路與計算中心(NCHC) Taiwania HPC系統的使用說明

- 檔案下載目錄 : /pkg/mpi_sample/pdf/

Taiwania HPC 系統

- 計算節點
- 高性能檔案處理儲存設備
- 登入節點
- 資料傳輸節點

計算節點

Node Type	Total units (nodes)	Compute resources per unit (node)					
		CPU Sockets	CPU cores	Memory (GB)	Tesla P100	10Gbps interface	480 GB SSD
Thin nodes	438	2	40	192	-	-	-
Thin nodes	64	2	40	192	-	1	-
Fat nodes	64	2	40	384	-	-	-
Fat nodes	64	2	40	384	-	-	1
GPU nodes	64	2	40	192	4	-	-
Big memory node	1	4	96	6000	-	-	-

高性能檔案處理儲存設備

- 每個帳號，/home目錄免費提供100GB；/work1目錄免費提供1.5TB空間
- /project此空間提供研究群專案付費使用，提供研究成員長期存放共享資料
- 用戶請將計算前中後需要的檔案放在/work1/$USER，此空間的資料無備份，28天都沒有存取的檔案將會被系統偵測並清除。
- 需長期存放的資料請自行放置到/home或/project

Mount point	Capacity	Free quota	Purge policy	Suitable Policy
/home	500TB	100GB	N/A	code, program, small data
/work1	2.2 PB	1.5TB	28	Large data, intermediate files
/project	1.0 PB	N/A	N/A	Shared data, shared quota

- lfs quota 指令可查看自己帳號mount point容量配額和已使用有多少容量

```
$ lfs quota -hu $USER /home/$USER
Filesystem      used   quota   limit   grace   files   quota   limit   grace
/home/$USER   12.36M   100G    100G      -     868       0       0       -
```

登入節點

- 提交/管理HPC作業
- 可以存取保留在高速儲存系統上的檔案
- 編譯HPC應用程式
- 執行開發程式碼的除錯

Node Type	Node range	Total units (nodes)	Compute resources per unit (node)				
			CPU Sockets	CPU cores	Memory (GB)	Tesla P100	480 GB SSD
CPU login nodes	clogin1 - clogin2	2	2	40	384	-	1
GPU login nodes	glogin1	1	2	40	192	4	-

- 請不要將登入節點當成計算節點使用
- 140.110.148.11 clogin1.twnia.nchc.org.tw
- 140.110.148.12 clogin2.twnia.nchc.org.tw
- 140.110.148.15 glogin1.twnia.nchc.org.tw

資料傳輸節點

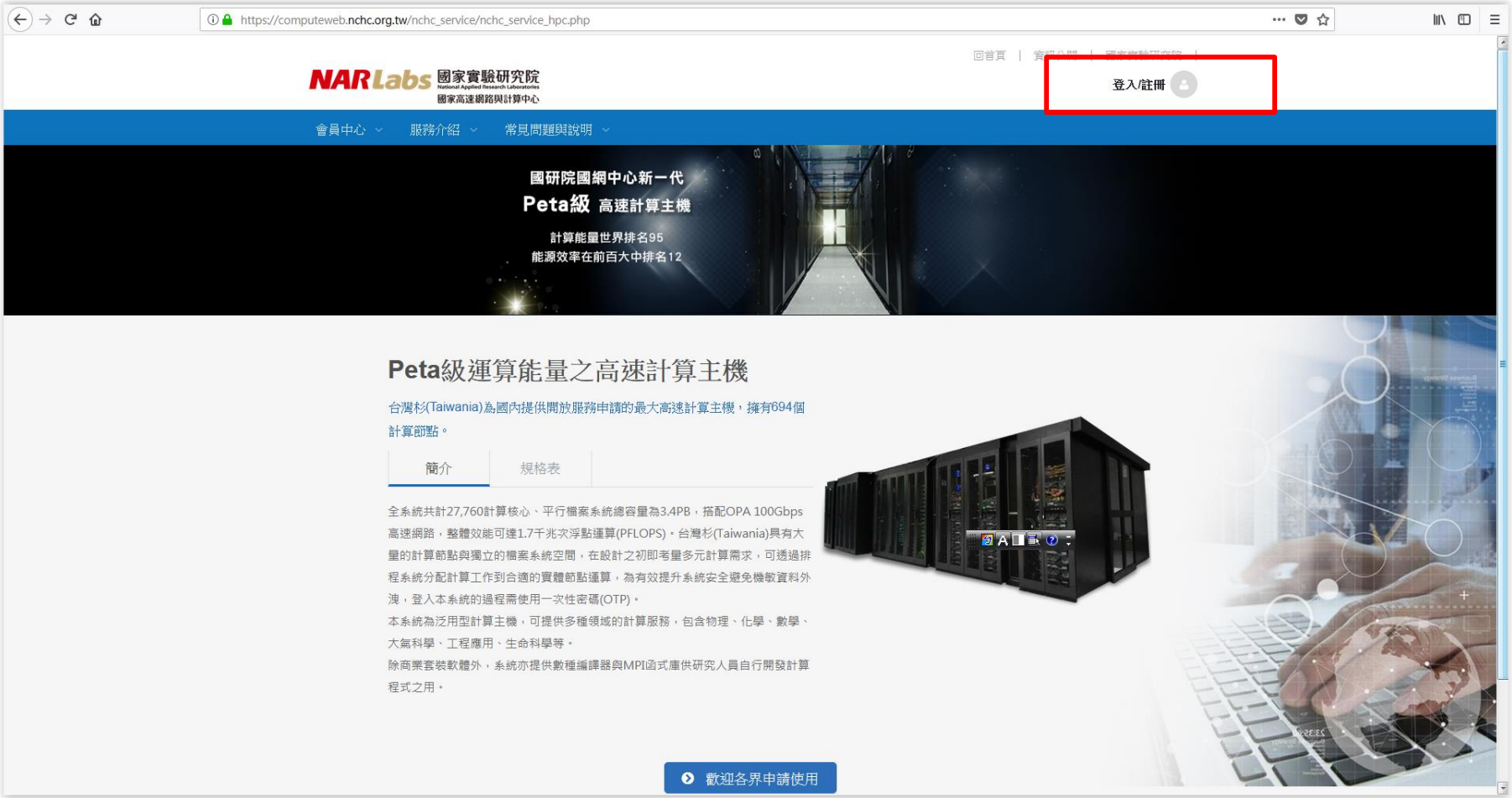
- 主要用途是傳輸資料，網路頻寬比登入節點快(網卡為40Gb)
- 只允許使用者使用scp/sftp存取資料，不能當登入節點使用
- 140.110.148.21 xdata1.twnia.nchc.org.tw
- 140.110.148.22 xdata2.twnia.nchc.org.tw

登入系統方法

- 註冊帳號
- 會員中心（Project, 錢包）
- 會員中心（OTP軟體&金鑰）
- 變更主機密碼
- 命令列登入
- 命令列登出
- 檔案傳輸

註冊帳號

- 註冊網址 https://iservice.nchc.org.tw/nchc_service/index.php

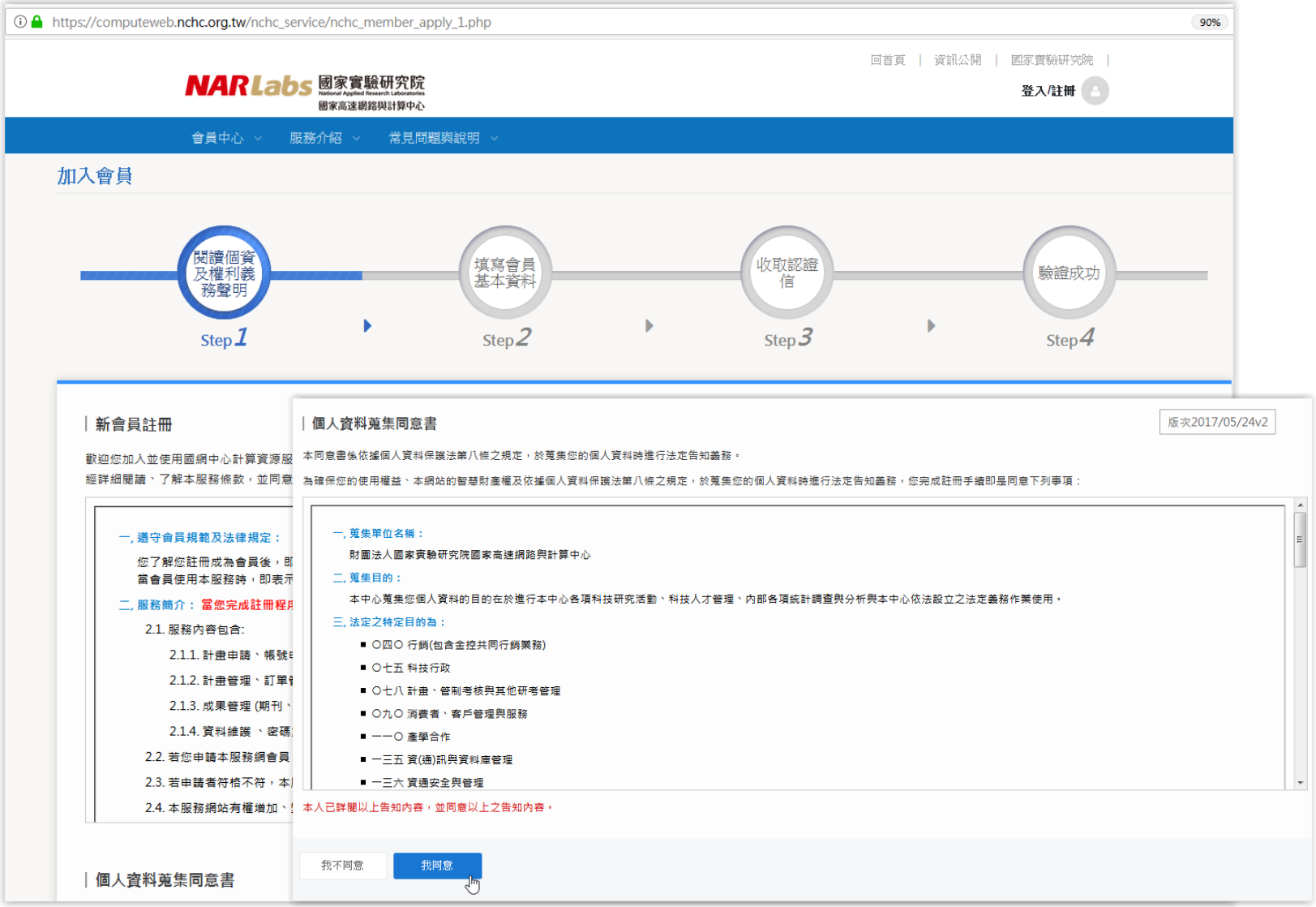


註冊帳號

■ [現在就加入會員]

註冊帳號（Step1）

■［個資同意書］



註冊帳號（Step2）

■ [填寫會員基本資料-會員帳號資料]

註冊帳號（Step2）

■[填寫會員基本資料-主機帳號資料]

填寫會員基本資料

會員資料　主機帳號資料

為了讓您體驗及熟悉主機之環境，特別貼心的為初次申請者，自動提供PETA 主機200元使用的額度，(使用期限為一年，申請一次為限)。
未來如額度不敷使用時，敬請透過此服務網提出計畫申請及購買使用額度。
以下是您未來登入主機的帳號資訊，您可以自行命名或直接採用本中心為您取的主機帳號名稱，此帳號如建立後，不提供更名之服務。

主機帳號資料

* 主機帳號	主機密碼	再次輸入密碼
	•••••••	•••••••

產生主機帳號

說明:
1. 主機帳號長度8-12個字元，限定小寫英數混合，首字英文。
2. 主機密碼長度至少需8字元，不可過於簡單
3. 主機密碼可為數字、英文字母(大小寫視為2種)、其他特殊字元等4種型式，至少須包含2種
4. 主機密碼有效期最長180天
5. 單一節點連續登入錯誤5次，將暫時鎖定主機帳號3分鐘，請改其他節點登入或等待三分鐘後再嘗試登入

填寫會員資料　下一步

註冊帳號（Step3）

■ [E-Mail收取認證信]

感謝您向國網中心申辦計算資源服務網會員，若上述基本資料無誤，為確保您的電子信箱無誤且為本人使用，請於 24 小時內點擊下列連結，完成會員帳號啟動程序。

https://computeweb.nchc.org.tw/nchc_service/nchc_member_apply_5.php?key=A_mQo32aH-C6AHmUpVfVC0fTzuGTaz8-GgYtk625Hx7KflS7Ypb9IR0oJUlpH_OAo1ofRUi_plY24sF6hW7PQw
請點此 連結完成電子信箱認證
*請於 24 小時內完成電子信箱認證。

*此信件為系統自動發出，請勿直接回覆。若上述連結無法正常開啟、資料有誤等任何相關疑問，歡迎您隨時透過以下方式與我們連絡，我們將盡快為您服務，謝謝！

若有任何問題，歡迎您隨時透過以下方式與我們連絡，我們將盡快為您服務，謝謝！

HPC 客服諮詢櫃檯網頁
E-mail：account@nchc.narl.org.tw
電話：03-5776085-442 呂小姐

國家高速網路與計算中心 計算資源服務小組 敬上

註冊帳號（Step4）

■［手機簡訊驗證］

會員中心（計劃管理 – Project）

■ 查看Project代號及錢包資訊

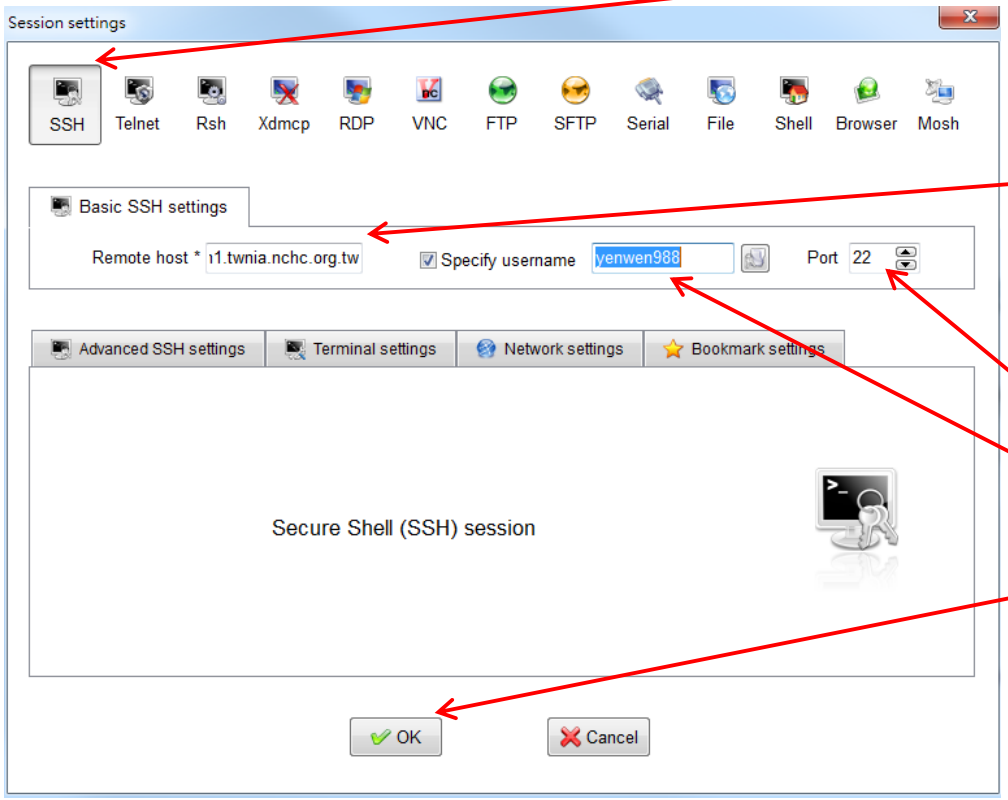
會員中心（會員資訊- One Time Pasword）

- 安裝OPT Authenticator軟體於手機或電腦
- 安裝後輸入自己的OTP金鑰

變更主機密碼

- [會員中心]-[會員資訊]-[主機帳號資訊]

命令列登入(可選擇使用MobaXterm、Putty等)



SSH

輸入下面的資訊

Host: 下列登入節點IP之一

- “clogin1.twnia.nchc.org.tw”
- “clogin2.twnia.nchc.org.tw”
- “glogin1.twnia.nchc.org.tw”

TCP port: 22”

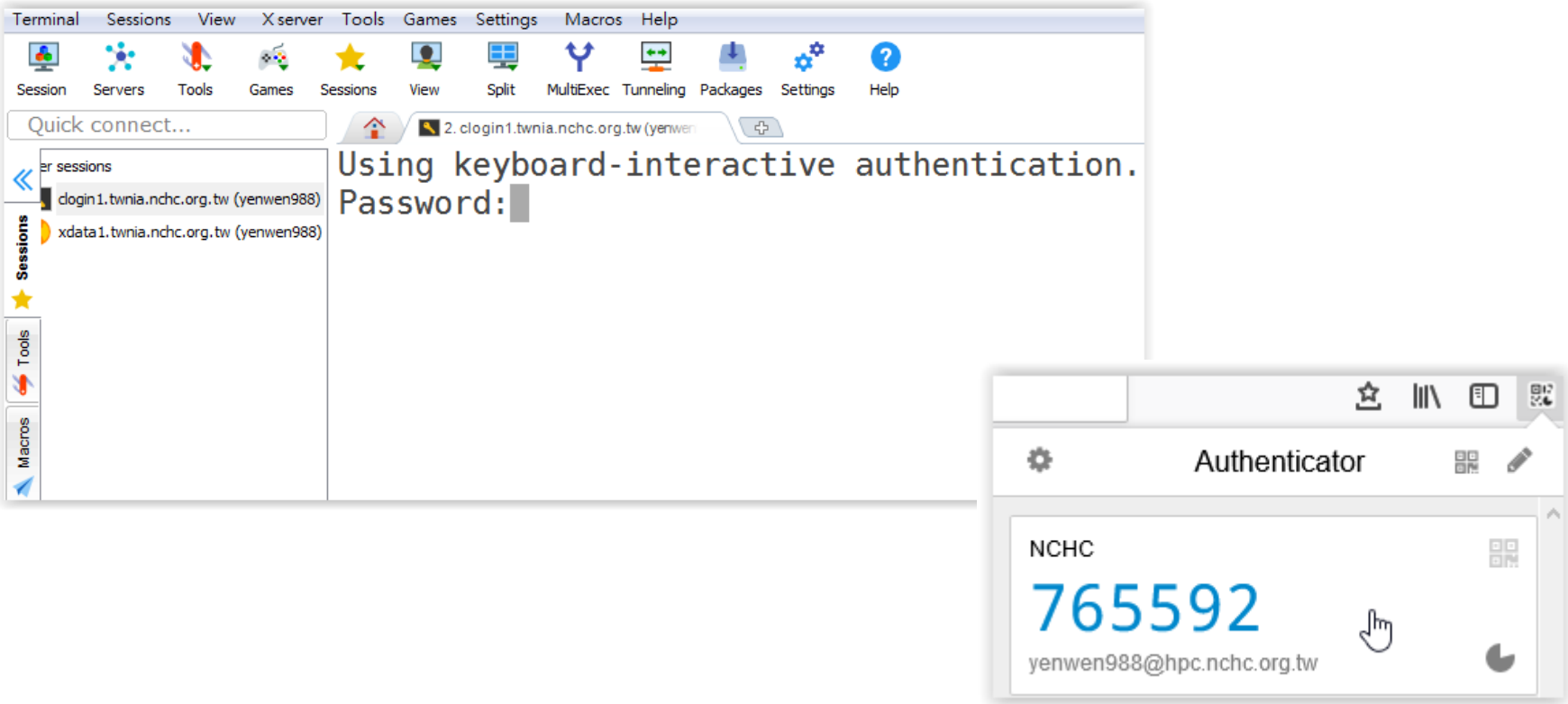
User name:

按OK

命令列登入

接下來，輸入您的主機帳號密碼接著輸入OTP動態密碼並按Enter鍵（密碼=帳號密碼+OTP動態密碼）

如下圖：



命令列登入

登入訊息

```
To run your jobs, use PBS Pro commands:

step 1: Prepare your job script first and specify Queue
and ProjectID in it.
        $ less /pkg/README.JOB.SCRIPT.EXAMPLE

        $ get_su_balance

        $ vi pbs_job.sh

step 2: Submit your job script to Torque and then
you'll get the job id.
        $ chmod u+x pbs_job.sh
        $ qsub pbs_job.sh

step 3: Trace job id and monitor your job.
        $ qstat -u your_account
        $ qstat -f

Other handy Torque commands:

      Terminate your job.
        $ qdel job_id

      Query available compute nodes.
        $ pbsnodes -a

      Display the list of all available Queues
        $ qstat -Q

Note:
1. New Queue ct160 has been released. (cpu cores:
min=2, max=160)
2. Do NOT use the login nodes for computation.
[yenwen988@clogin1 ~]$ get_su_balance
199,TRI107100,試用計畫(ISSUE)
[yenwen988@clogin1 ~]$ cat /etc/motd
```

命令列登出

- 執行登出或exit 指令

```
$ exit
```

檔案傳輸(Linux users)

- 使用scp指令存取傳輸節點的資料語法如下

```
$ scp [option] <source host>:<local path of directory or file>  \
  <destination host>:<remote path of directory or file>
```

常用的option如下:

-p　　保留原始檔案的修改時間、存取時間和模式。
-r　　遞歸複製整個目錄。

- 使用sftp指令列存取傳輸節點的資料語法如下

```
$ sftp [option] [username@]<destination host>
Connected to <destination host>.
sftp> get <remote path of directory or file>
                                        -> 下載檔案到本地端的目錄路徑
sftp> put <local path of directory or file>
                                        -> 上傳檔案到遠端的目錄路徑
sftp> bye                               -> 離開sftp
```

指令中常用的一些主要內部命令是：

cd *<path>*　　更改遠端的目錄路徑。
pwd　　顯示遠端的工作目錄。
lcd *<path>*　　更改本地端的目錄路徑。
lpwd　　顯示本地端的工作目錄。

檔案傳輸(Windows users)可選擇適合的軟體

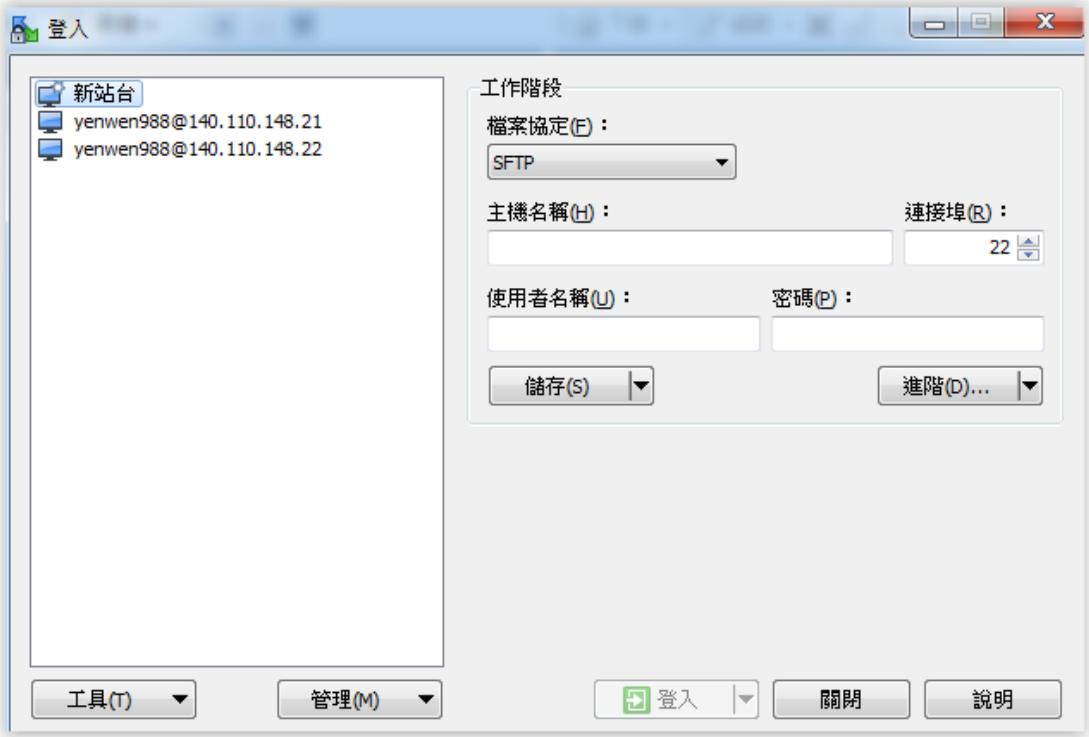
選擇適合的軟體如MobaXterm或WinSCP

Host name: “xdata1.twnia.nchc.org.tw”
“xdata2.twnia.nchc.org.tw”

Port number: 22

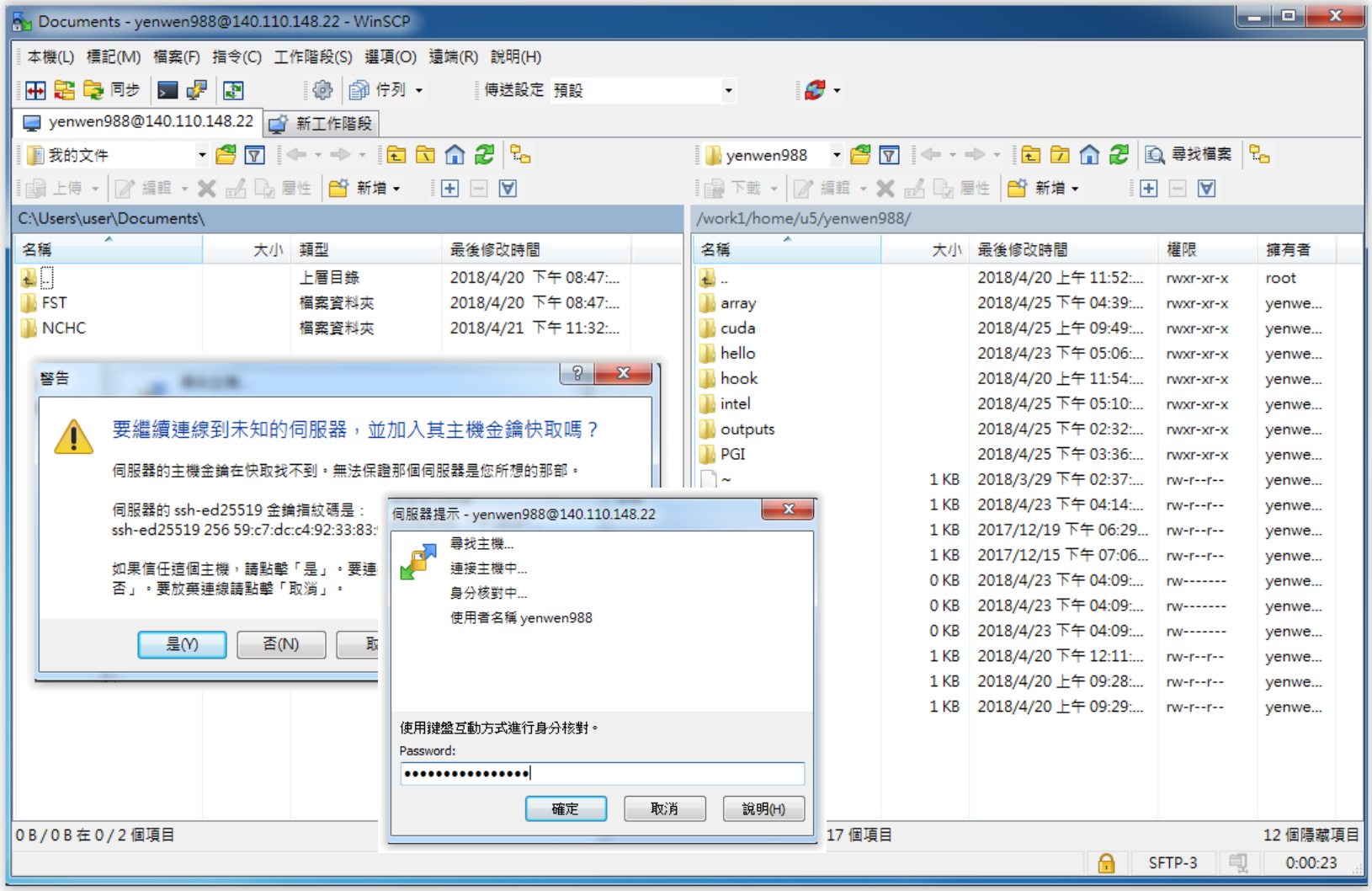
User name: 輸入主機帳號名稱

Password: 輸入主機帳號密碼+OPT動態密碼



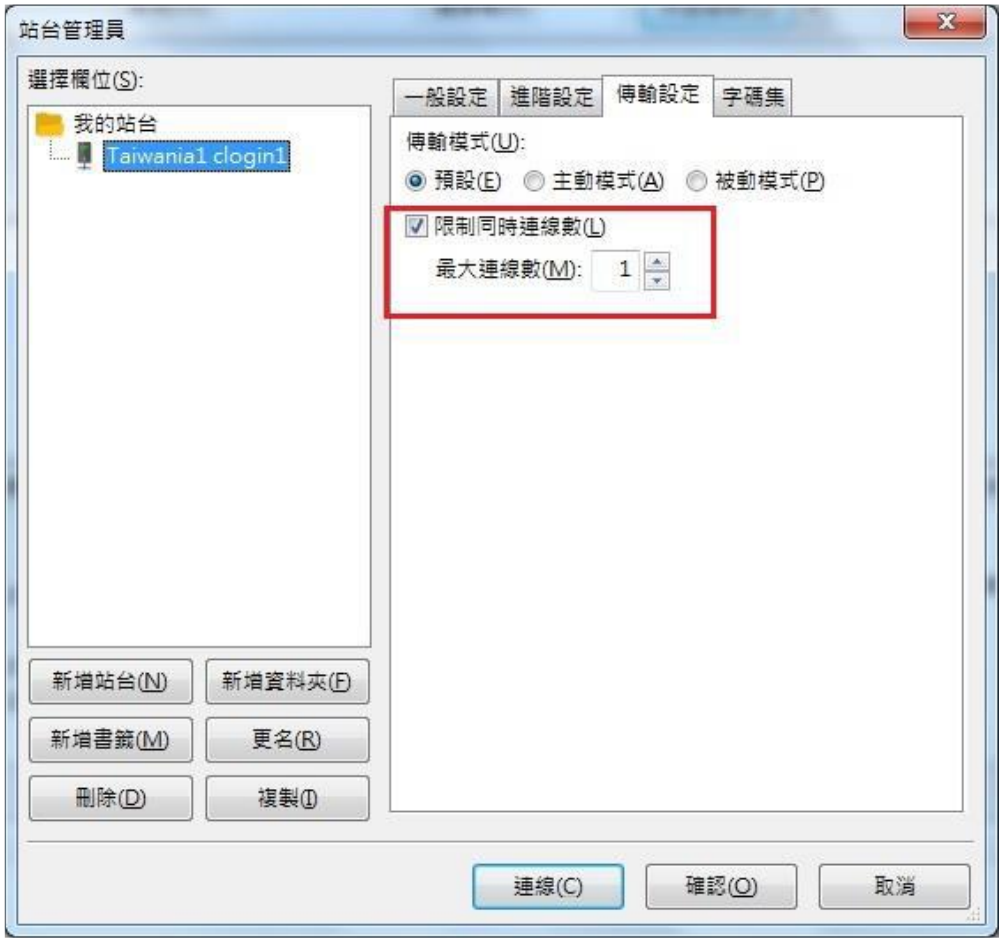
檔案傳輸(Windows users)

當連線建立後，WinSCP出現畫面如下所示



檔案傳輸(Windows users)

- 如果是使用FileZilla軟體，請將[檔案]-[站台管理員]-[傳輸設定]的[限制同時連線數]-[Enable]，[最大連線數]設為[1]（因OTP issue）



練習

- 1. 完成申請會員帳號&主機帳號
- 2. 讓手機可使用Authenticator程式，並將OTP金鑰加入
- 3. 完成主機帳號登入系統

Linux基本指令

- 常用指令
- vim
- 查看空間使用
- 查看說明
- 練習

Linux基本指令（常用指令）

- pwd : 顯示所在資料夾路徑
- mkdir : 建立資料夾
- ls : 列出所有檔案與資料夾
- ls -al : 列出所有檔案與資料夾之詳細資訊
- cd ~ : 切換到家目錄
- cd .. : 切換到上層資料夾
- cd - : 切換回先前的資料夾

詳請參考鳥哥的Linux私房菜 http://linux.vbird.org/

Linux基本指令（常用指令）

- mv　　：移動檔案或重新命名

  $ mv a.txt b.txt

- rm　　：移除檔案

  $ rm filename

- rm -r　：移除檔案資料夾
- rm -rf　：強制移除檔案或資料夾(不會詢問)

Linux基本指令（常用指令）

- cat : 列出檔案內容（more ; less）

```
$ cat filename
$ more filename
$ less filename
```

- grep : 搜尋字串

```
$ cat filename |grep string
$ grep string filename
```

- cp : 複製檔案目錄

```
$ cp 來源 目地
$ cp -rp 來源 目地 : 連同權限與目錄一起copy
```

Linux基本指令（常用指令）

- 環境變數 PATH ： 查詢可執行程式的路徑

```
$ echo $PATH
$ export PATH=$PATH:/xxx
```

- 環境變數 LD_LIBRARY_PATH ： 查詢動態函式庫的路徑

```
$ echo $LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/xxx
```

- which ： 尋找執行檔路徑

```
$ which gcc
```

Linux基本指令（vim）

- 命令模式(command mode)：在這個模式下，無法輸入文字，但可用x刪除文字。

- 輸入模式(Insert mode)：在command mode先按下i、a或o這三個鍵其中一個進入輸入模式，便能開始輸入文字。
  - i表示insert
  - a表示append
  - o表示會新增一行並開始輸入

  按下ESC鍵，可退回至命令模式。

- 底線命令模式(Last line mode)：在命令模式按下冒號:可進入底線命令模式。
  - :q 不保存,直接退出
  - :q! 不保存，並強制退出
  - :wq或:x保存，並退出
  - :w 保存文件,但不退出
  - :wq!強制保存，並退出

  按ESC鍵可退回至命令模式。

Linux基本指令（查看空間使用）

- /home目錄每個帳號，免費提供100GB
- /work1目錄每個帳號，免費提供1.5TB

```
$ lfs quota -hu $USER /home/$USER
Disk quotas for usr $USER
Filesystem     used   quota   limit   grace   files   quota   limit   grace
/home/$USER  12.36M  100G    100G      -      868      0       0       -
```

```
$ lfs quota -hu $USER /work1/$USER
Disk quotas for usr $USER
Filesystem    used   quota   limit   grace   files   quota   limit  grace
/work1/$USER  2M     1.5T    1.5T      -     189      0       0      -
```

Linux基本指令（查看說明）

- man 指令
- 指令 --help
- Google

練習（複製檔案）

- 請在家目錄建一個1113目錄, 試著將/pkg/mpi_sample下的所有的目錄copy到自己家目錄下的1113目錄

```
$ ls /pkg/mpi_sample
$ cd ~
$ pwd
$ mkdir 1113
$ cp -rp /pkg/mpi_sample/*  ~/1113
$ ls 1113
array_mpi_gcc   cuda     depend     mpi_gcc   mpi_pgi  openmp_mpi
array_mpi_intel cuda_mpi hello_world mpi_intel openmp   pdf
```

Compile/Link

- 環境模組Environment modules
- 對應各種語言編譯指令對照表
- Intel Compiler
- PGI Compiler
- Compile with CUDA
- mpirun
- 練習

環境模組Environment modules

- 目前帳號已載入模組

```
$ module list
Currently Loaded Modulefiles:
  1) gcc/6.3.0
```

環境模組Environment modules

系統提供的模組

```
$ module avail
------------------------------ /cm/shared/modulefiles ------------------------------------------------
abaqus/2017                          biology/Python3/default              iozone/3_465
adf/2017.112                         biology/rosetta/gcc/2018.09.60072    lapack/gcc/64/3.7.0
amber/16/gcc/gpu                     biology/VCFtools/v0.1.13             lsdyna/R10.0.0
amber/16/gcc/multigpu                blacs/openmpi/gcc/64/1.1patch03      lsdyna/R10.1.0
amber/16/gcc/parallel                blas/gcc/64/3.7.0                    lsdyna/R8.1.0
amber/16/gcc/serial                  bonnie++/1.97.1                      mlib/OpenBLAS-0.2.20
anaconda2/5.1.10                     calculix/calculix213                 mvapich2/gcc/64/2.2rc1
anaconda3/5.1.10                     calculix/calculix214                 namd/2.12/cpu
ansys/CFX/v162                       chem/cp2k/r18428                     namd/2.12/gpu
ansys/CFX/v170                       cuda/8.0.61                          netcdf/gcc/64/4.6.0
ansys/CFX/v182                       cuda/9.1.85                          netperf/2.7.0
ansys/EM/v17.0                       default-environment                  openmpi/gcc/64/1.10.3
ansys/EM/v19.0                       dock/6.8                             openmpi/gcc/64/1.10.4
ansys/Fluent/v162                    fftw2/openmpi/gcc/64/double/2.1.5    openmpi/open64/64/1.10.3
ansys/Fluent/v170                    fftw2/openmpi/gcc/64/float/2.1.5     openmpi/pgi/2.1.2/2017
ansys/Fluent/v182                    fftw3/openmpi/gcc/64/3.3.6           petsc/openmpi/gcc/3.8.0
ansys/ICEMCFD/v162                   gaussian/g09                         pgi/17.10
ansys/ICEMCFD/v170                   gaussian/g16                         python3/3.5.6
ansys/ICEMCFD/v182                   gdb/7.12.1                           scalapack/openmpi/gcc/64/2.0.2
ansys/workbench_APDL/v162            hdf5/1.10.0                          singularity/2.5.2
ansys/workbench_APDL/v170            hdf5/1.10.1                          tcad/G_2012.06
ansys/workbench_APDL/v182            hwloc/1.11.6                         tcad/I_2013.12
biology/BEDTools/v2.27.1             intel/2017_u4                        tcad/J_2014.09
biology/Java/jdk_10.0.2              intel/2018_init                      tcad/K_2015.06
biology/Java/jdk_8u181               intel/2018_u1                        tcad/L_2016.03
```

環境模組Environment modules

- 載入compiler、library、applications所需要的模組

```
$ module load <module name>
$ module load intel/2018_u1
$ module list
Currently Loaded Modulefiles:
  1) gcc/6.3.0        2) intel/2018_u1
$ which icc
/pkg/intel/2018_u1/compilers_and_libraries_2018.1.163/linux/bin/intel64/icc
```

- 增加其他模組

```
$ module add <module name>
$ module add cuda/8.0.61
$ module list
Currently Loaded Modulefiles:
  1) gcc/6.3.0        2) intel/2018_u1    3) cuda/8.0.61
$ which nvcc
/pkg/cuda/8.0.61/bin/nvcc
```

環境模組Environment modules

卸載模組

```
$ module unload <module name>
$ module unload cuda/8.0.61
$ module list
Currently Loaded Modulefiles:
  1) gcc/6.3.0        2) intel/2018_u1
```

卸載所有已載入模組

```
$ module purge
$ module list
No Modulefiles Currently Loaded.
```

查看模組名稱的描述

```
$ module whatis intel/2018_u1
intel/2018_u1          : adds Intel Parallel Studio XE 2018 update1 to your
environment variables.
```

環境模組Environment modules

預設的環境模組

```
$ cat .bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
module load gcc
```

環境模組Environment modules

- 此系統提供模組部分摘錄描述如下表
- openmpi/gcc和openmpi/pgi一次只能載入其中的一個模組
- intel/2017_u4，intel/2018_init和intel/2018_u1等不同版本，也是一次只能載入其中的一個模組

Module Name	Description
blacs/openmpi/gcc/64/1.1patch03	Blacs library
bonnie++/1.97.1	Bonnie++ library
cuda/8.0.61	CUDA library for GPU
gdb/7.12.1	GNU Cross Compilers
intel/2017_u4	Intel Parallel Studio XE 2017 update 4
intel/2018_init	Intel Parallel Studio XE 2018 Initial
intel/2018_u1	Intel Parallel Studio XE 2018 update 1
iozone/3_465	File system benchmark tool
mvapich2/gcc/64/2.2rc1	MVAPICH MPI library
netcdf/gcc/64/4.6.0	Network Common Data Form library
netperf/2.7.0	Network benchmark
openmpi/gcc/64/1.10.3	GCC compiled OpenMPI
openmpi/pgi/2.1.2/2017	PGI compiled OpenMPI
petsc/openmpi/gcc/3.8.0	PETSc data structure library
pgi/17.10	PGI compilers and development tools

對應各種語言編譯指令對照表

- Serial program & Thread parallel program

	GCC	intel	pgi	CUDA
module example	gcc/6.3.0	intel/2018_u1	pgi/17.10	cuda/8.0.61
C	gcc	icc	pgcc	nvcc
C++	g++	icpc	pgc++	nvcc
Fortran	gfortran	ifort	pgf77 pgf90 pgf95 pgfortran	-

對應各種語言編譯指令對照表

MPI program

	Intel MPI	Openmpi/gcc	Openmpi/pgi
Module example	intel/2018_u1	openmpi/gcc/64/1.10.4	openmpi/pgi/2.1.2/2017
C	mpiicc mpicc mpicxx	mpicc mpicxx	mpicc mpicxx
C++	mpiicpc	mpic++	mpic++
Fortran	mpiifort mpifc mpif90 mpif77	mpif77 mpif90 mpifort	mpifort mpif90 mpif77

Intel Compiler (Serial program)

- 載入英代爾編譯器環境
- 選擇版本相配的的模組

```
$ module purge

$ module load intel/2018_u1

$ module list
Currently Loaded Modulefiles:
  1) intel/2018_u1

$ which icc
/pkg/intel/2018_u1/compilers_and_libraries_2018.1.163/l
inux/bin/intel64/icc
```

Intel Compiler (Serial program)

- Serial program
- compile/link C 、C++、Fortran程式

```
$ icc -o sample.exe sample.c
$ ls -l sample.exe
-rwxr-xr-x 1 yenwen988 TRI107100 27464 Aug 21 10:54
sample.exe
```

```
$ icpc -o sample.exe sample.c
```

```
$ ifort -o sample.exe sample.f
```

Intel Compiler (OpenMP program)

- Thread parallel program (OpenMP)
- compile/link C 、C++、Fortran程式

```
$ icc   -qopenmp  -o sample_omp.exe sample_omp.c
```

```
$ icpc  -qopenmp  -o sample_omp.exe sample_omp.c
```

```
$ ifort -qopenmp  -o sample_omp.exe sample_omp.f
```

Intel Compiler (MPI program)

- MPI 平行處理型程式
- compile/link C 、C++、Fortran程式

```
$ mpiicc -o sample_mpi.exe sample_mpi.c
```

```
$ mpiicpc -o sample_mpi.exe sample_mpi.c
```

```
$ mpiifort -o sample_mpi.exe sample_mpi.f
```

PGI Compiler (Serial program)

- 載入編譯器環境
- 載入PGI compiler模組

```
$ module purge

$ module load pgi/17.10

$ module list
Currently Loaded Modulefiles:
  1) pgi/17.10

$ which pgcc
/pkg/pgi/17.10/linux86-64/17.10/bin/pgcc
$ which pgc++
/pkg/pgi/17.10/linux86-64/17.10/bin/pgc++
```

PGI Compiler (Serial program)

- Serial program
- compile/link C 、C++、Fortran程式

```
$ pgcc -o sample.exe sample.c
```

```
$ pgc++ -o sample.exe sample.c
```

```
$ pgfortran -o sample.exe sample.f
```

PGI Compiler (OpenMP program)

- Thread parallel program (OpenMP)
- compile/link C 、C++、Fortran程式

```
$ pgcc -mp -o sample_omp.exe sample_omp.c
```

```
$ pgc++ -mp -o sample_omp.exe sample_omp.c
```

```
$ pgfortran -mp -o sample_omp.exe sample_omp.f
```

PGI Compiler (MPI program)

- 載入編譯器環境
- 載入PGI compiler模組

```
$ module load pgi/17.10 openmpi/pgi/2.1.2/2017
$ module list
Currently Loaded Modulefiles:
  1) pgi/17.10                2) openmpi/pgi/2.1.2/2017

$ which pgcc
/pkg/pgi/17.10/linux86-64/17.10/bin/pgcc
$ which pgc++
/pkg/pgi/17.10/linux86-64/17.10/bin/pgc++

$ which mpicc
/pkg/pgi/17.10/linux86-64/2017/mpi/openmpi-2.1.2/bin/mpicc
```

PGI Compiler (MPI program)

- MPI 平行處理型程式
- compile/link C 、C++、Fortran程式

```
$ mpicc -o sample_mpi.exe sample_mpi.c
```

```
$ mpic++ -o sample_mpi.exe sample_mpi.c
```

```
$ mpifort -o sample_mpi.exe sample_mpi.f
```

CUDA Compiler (thread parallel program)

- 載入編譯器環境
- 選擇版本相配的的模組

```
$ module purge
$ module load cuda/8.0.61

$ module list
Currently Loaded Modulefiles:
  1) cuda/8.0.61

$ which nvcc
/pkg/cuda/8.0.61/bin/nvcc
```

CUDA Compiler (thread parallel program)

- CUDA program
- compile/link C 、C++、Fortran程式

```
$ nvcc -o sample.exe sample.cu
```

```
$ nvcc -o sample.exe sample.cu
```

```
$ module add pgi
$ pgfortran -o  sample.exe sample.cuf
```

mpirun (Intel MPI)

mpirun (Intel)

- 編譯完成的MPI執行檔，需用mpirun command來執行MPI執行檔
- mpirun有許多option可活用，load Intel MPI module後，直接執行mpirun就可以看到option說明

```
$ module load intel/2018_u1
$ module list
Currently Loaded Modulefiles:
  1) intel/2018_u1
$ which mpirun
/pkg/intel/2018_u1/compilers_and_libraries_2018.1.163/linux/mpi/intel64/bin/mpirun
$ which mpiexec
/pkg/intel/2018_u1/compilers_and_libraries_2018.1.163/linux/mpi/intel64/bin/mpiexec
$ mpirun

Usage: ./mpiexec [global opts] [exec1 local opts] : [exec2 local opts] : ...
Global options (passed to all executables):
 Other global options:
    -f {name} | -hostfile {name}     file containing the host names
 Other local options:
    -n/-np {value}                   number of processes
    {exec_name} {args}               executable name and arguments
```

mpirun (GCC OpenMPI)

mpirun (GCC)

```
$ module load openmpi/gcc/64/1.10.4
$ module list
Currently Loaded Modulefiles:
  1) openmpi/gcc/64/1.10.4
$ which mpirun
/usr/mpi/gcc/openmpi-1.10.4-hfi/bin/mpirun
$ man mpirun
SYNOPSIS
       Single Process Multiple Data (SPMD) Model:
       mpirun [ options ] <program> [ <args> ]

QUICK SUMMARY
       If you are simply looking for how to  run  an  MPI  application,  you
       probably want to use a command line of the following form:

           % mpirun [ -np X ] [ --hostfile <filename> ]  <program>

       This will run X copies of <program> in your current run-time environ-
       ment , scheduling (by default)  in  a  round-robin fashion by CPU slot.
       -hostfile, --hostfile <hostfile>
              Provide a hostfile to use.
       Setting MCA parameters:
       -mca, --mca <key> <value>
              Send arguments to various MCA modules.
```

練習

- cd到之前copy過來的1113/hello_world目錄下，載入Intel MPI module，將hello_mpi.c compile/link成執行檔，然後在login node試著執行Intel MPI hello-world程式，如下：

```
$ cd ~/1113/hello_world/
$ module purge
$ module load intel/2018_u1
$ module list
$ which mpicc
$ mpicc -o hello_mpi_intel.exe hello_mpi.c
$ ls -l hello_mpi_intel.exe
$ mpirun -np 2 ./hello_mpi_intel.exe
Hello world from processor clogin2, rank 0 out of 2 processors
Hello world from processor clogin2, rank 1 out of 2 processors
```

PBS PRO Job 操作

- What is PBS pro
- Queue Name
- Queue list
- PBS Pro job operation
- 練習MPI hello-world

What is PBS pro

- PBS Professional 高性能計算負載管理軟體
- PBS Professional 用於高性能計算（HPC）環境的負載管理器和作業調度器。
- PBS Professional® 是一款負載管理軟體，旨在提高生產力、優化資源利用率和效率並簡化 HPC 集群、雲端和超級電腦的管理工作。
- PBS Professional 能夠自動執行作業的調度、管理、監控和報告工作。

Queue Name

■ 胖節點的Quene name

Queue name	Type	Resource Range (CPU cores)	Memory per node	Resource Range (GPUs)	Resource range (SSD)	Max Walltime per job
serial	execution	1	384GB	—	1	96:00:00
cf40	execution	2-40	384GB	—	1	96:00:00
cf160	execution	2-160	384GB	—		96:00:00
cf1200	execution	161-1200	384GB	—		48:00:00

Queue Name

■ 瘦節點的Quene name

Queue name	Type	Resource Range (CPU cores)	Memory per node	Resource Range (GPUs)	Resource range (SSD)	Max Walltime per job
ct160	execution	2-160	192GB	-		96:00:00
ct400	execution	161-400	192GB	—		96:00:00
ct800	execution	401-800	192GB	—		72:00:00
ct2k	execution	801-2000	192GB	—		48:00:00
ct6k	execution	2001-6000	192GB	—		24:00:00
ctest	execution	2-80	192GB	—		00:30:00

Queue Name

- GPU節點的Quene name

Queue name	Type	Resource Range (CPU cores)	Memory per node	Resource Range (GPUs)	Resource range (SSD)	Max Walltime per job
gtest	execution	1-8	192GB	1-8		00:30:00
gp4	execution	1-40	192GB	1-4		96:00:00
gp16	execution	41-160	192GB	5-16		96:00:00
gp32	execution	161-320	192GB	17-32		48:00:00

Queue List

■ 列出單一Queue的詳細資訊(例如ctest)

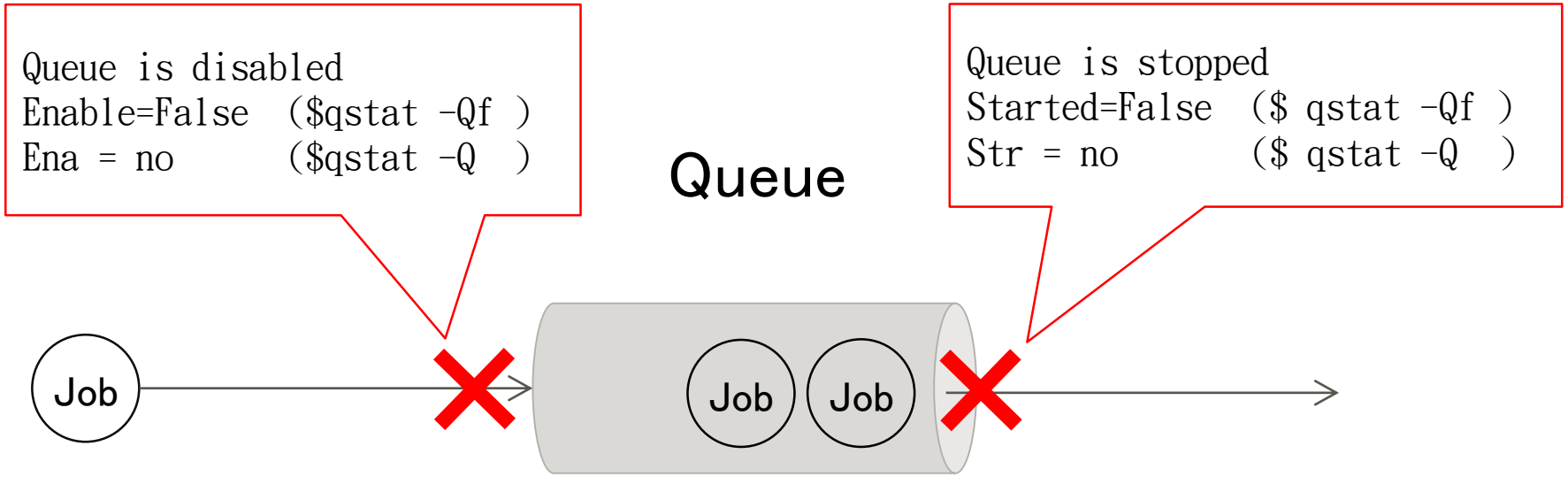
```
$ qstat -Qf ctest
Queue: ctest
    queue_type = Execution
    total_jobs = 25
    state_count = Transit:0 Queued:0 Held:25 Waiting:0 Running:0 Exiting:0 Begu
        n:0
    resources_max.ncpus = 80
    resources_max.walltime = 00:30:00
    resources_min.ncpus = 2
    resources_default.ncpus = 2
    acl_group_enable = True
    acl_groups = +,-MST107024
    default_chunk.Qlist = ctest
    resources_assigned.mpiprocs = 0
    resources_assigned.ncpus = 0
    resources_assigned.nodect = 0
    enabled = True
    started = True
```

enabled=True :job可以排入ctest Queue
started=True :排入Queue的job可以被執行

在此ctest queue的job ncpus 最多不能超過80
在此ctest queue的job ncpus 最少需求要2
在此ctest queue的job walltime 30分鐘

Queue List

- Queue is disabled & stopped



Queue List

- 列出所有Queue資訊

```
$ qstat -Q
Queue              Max   Tot Ena Str   Que   Run   Hld   Wat   Trn   Ext Type
---------------- ----- ----- --- --- ----- ----- ----- ----- ----- ----- ----
ct16k                0     0  no  no     0     0     0     0     0     0 Exec
gp128                0     0  no  no     0     0     0     0     0     0 Exec
ct160                0    56 yes yes     2    54     0     0     0     0 Exec
gtest                0     0 yes yes     0     0     0     0     0     0 Exec
gp4                  0    37 yes yes    11    24     0     0     0     0 Exec
gp16                 0     1 yes yes     1     0     0     0     0     0 Exec
gp32                 0     0 yes yes     0     0     0     0     0     0 Exec
ct6k                 0     2 yes yes     2     0     0     0     0     0 Exec
serial               0     5 yes yes     0     5     0     0     0     0 Exec
cf40                 0   316 yes yes   200   116     0     0     0     0 Exec
cf160                0    72 yes yes     2    70     0     0     0     0 Exec
cf1200               0     1 yes yes     0     1     0     0     0     0 Exec
ct800                0     1 yes yes     0     1     0     0     0     0 Exec
ct2k                 0     0 yes yes     0     0     0     0     0     0 Exec
ct400                0    11 yes yes     0    11     0     0     0     0 Exec
ctest                0    25 yes yes     0     0    25     0     0     0 Exec
```

Queue List

qstat -Q 欄位資訊說明

```
$ qstat -Q gp4
Queue              Max   Tot Ena Str   Que   Run   Hld   Wat   Trn   Ext Type
---------------- ----- ----- --- --- ----- ----- ----- ----- ----- ----- ----
gp4                  0    37 yes yes    11    24     0     0     0     0 Exec
```

Queue	Queue name.
Max	Maximum number of jobs allowed to run concurrently in the queue.
Tot	Total number of jobs in the queue.
Ena	Whether the queue is enabled or disabled.
Str	Whether the queue is started or stopped.
Que	Number of queued jobs.
Run	Number of running jobs.
Hld	Number of held jobs.
Wat	Number of waiting jobs.
Trn	Number of jobs being moved (transiting.)
Ext	Number of exiting jobs.
Type	Type of queue: execution or routing.

Queue List

- 可利用grep列出欲查詢queue name資訊

```
$ qstat -Q gtest
Queue              Max   Tot Ena Str   Que   Run   Hld   Wat   Trn   Ext Type
---------------- ----- ----- --- --- ----- ----- ----- ----- ----- ----- ----
gtest                0     0 yes yes     0     0     0     0     0     0 Exec

$ qstat -Q serial
Queue              Max   Tot Ena Str   Que   Run   Hld   Wat   Trn   Ext Type
---------------- ----- ----- --- --- ----- ----- ----- ----- ----- ----- ----
serial               0     5 yes yes     0     5     0     0     0     0 Exec

$ qstat -Q |grep yes| grep cf
cf40                 0   316 yes yes   198   118     0     0     0     0 Exec
cf160                0    71 yes yes     2    69     0     0     0     0 Exec
cf1200               0     1 yes yes     0     1     0     0     0     0 Exec

$ qstat -Q |grep yes| grep gp
gp4                  0    37 yes yes    11    24     0     0     0     0 Exec
gp16                 0     1 yes yes     1     0     0     0     0     0 Exec
gp32                 0     0 yes yes     0     0     0     0     0     0 Exec
ngsgp4               0     0 yes yes     0     0     0     0     0     0 Exec
```

PBS Pro job operation (PBS job script語法)

- PBS job script
  - Shell specification
  - PBS directives
  - Programs or commands

```
#!/bin/bash                                  -> 宣告這個 script 使用的 shell 名稱

#PBS -l walltime=00:01:00
#PBS -l select=2:ncpus=16:mpiprocs=16
#PBS -N sample_job
#PBS -q ctest                                -> PBS 指令的option和參數
#PBS -P TRI654321
#PBS -j oe
#PBS -o outdir
#PBS -e outdir

module load intel/2018_u1
cd $PBS_O_WORKDIR                              -> shell script內容
cat $PBS_NODEFILE
echo $PBS_O_WORKDIR
date

mpirun ./myprogram
```

PBS Pro job operation（提交）

- 提交PBS job的2種格式（a）&（b）
- （a-1）建立PBS job script檔案

```
$ vim example01.sh
#!/bin/bash
#PBS -P sample_project
#PBS -N sample_job
#PBS -l select=2:ncpus=4:mpiprocs=4
#PBS -l walltime=00:01:00
#PBS -q ctest
#PBS -j oe
module purge
module load intel/2018_u1
cd $PBS_O_WORKDIR

mpirun ./myprogram
```

- (a-2)提交job

```
$ qsub example01.sh
20.srvc1
```

PBS Pro job operation（提交）

- (b) Submit command (qsub)直接帶PBS指令option和參數

  不建議使用此方法，不易進行後續troubleshooting

```
$ qsub -l select=1:ncpus=1 -q serial -P xxx -N test-job -j oe  ./example01.sh
21.srvc1
```

PBS Pro job operation (Submission Options)

- Job Submission Options 摘錄一些說明如下表

Option	Value	Description
-P	project	obtain project ID by get_su_balance command
-N	job_name	Specifying a job name
-q	queue_name	Specifying the queue name
-l	resource_list	Requesting job resources
-j	oe eo n	Merging standard output and error files
-o	path	Specifying path for standard output files
-e	path	Specifying path for standard error files
-M	e-mail_address	Setting email recipient list
-m	mail_events	Specifying email notification

PBS Pro job operation (Submission Options)

- Job Submission Options摘錄一些說明如下表(continue..)

Option	Value	Description
-a	[[[[YYYY]MM]DD]hhmm[.SS]]	Deferring execution
-J	<X-Y [:z]>	Defining job array
-h		Holding a job (delaying execution)
-W depend=	dependency_list	Specifying job dependencies

PBS Pro job operation (Submission Options)

- Job Submission Options摘錄一些說明如下表(continue..)

Option	Value	Description
-W depend=	dependency_list	Specifying job dependencies

Value	Description
after:arg_list	This job may be scheduled for execution at any point after all jobs in arg_list have started execution.
afterok:arg_list	This job may be scheduled for execution at any point after all jobs in arg_list have terminated with no errors.
afternotok:arg_list	This job may be scheduled for execution at any point after all jobs in arg_list have terminated with errors.

PBS Pro job operation（環境變數）

- PBS Pro 提供許多環境變數，摘錄一些說明如下表，可活用於 PBS job script

Variable	Description
PBS_JOBID	Job identifier given by PBS when the job is submitted.
PBS_JOBNAME	Job name given by user
PBS_NODEFILE	Name of file containing the list of nodes assigned to the job
PBS_O_PATH	Value of PATH taken from user's submission environment
PBS_O_WORKDIR	Absolute path to directory where qsub is run
TMPDIR	Pathname of job's scratch directory

PBS Pro job operation (Job Placement)

Requesting Job Resources: Job Placement

Using the place statement
Usage: -l place=[type]
Example: qsub -l place=scatter pbs_job.sh

type	free	Place job on any host(s) (default)
	pack	All chunks will be taken from one host.
	scatter	Only one chunk is taken from a host

PBS Pro job operation（資源選取）

- host base resource (Submitting Multiprocessor Jobs)
- Requesting number of MPI processes per chunk
  - Using the **mpiprocs** resource

For example:

1. -l select=2:ncpus=2:mpiprocs=2,place=scatter

$ cat $PBS_NODEFILE →

```
cn0101
cn0101
cn0102
cn0102
```

2. -l select=4:ncpus=4:mpiprocs=4,place=scatter

$ cat $PBS_NODEFILE →

```
cn0101
cn0101
cn0101
cn0101
cn0102
cn0102
cn0102
cn0102
cn0103
cn0103
cn0103
cn0103
cn0104
cn0104
cn0104
cn0104
```

PBS Pro job operation（資源選取）

- #PBS -l resource option (host base resource)

```
#PBS -l select=1:ncpus=1
-> sequential job (1 core)

#PBS -l select=2:ncpus=8:mpiprocs=8
-> MPI job
(Select 2 chunks with 8 CPUs each for a total of 16 MPI processes)

#PBS -l select=2:ncpus=8:mpiprocs=1:ompthreads=8
-> MPI/OpenMP Hybrid job
 (Request two chunks, each with 1 MPI tasks and 8 threads per
task)

#PBS -l select=2:ncpus=4:ngpus=4:mpiprocs=4
-> MPI/CUDA Hybrid job
 (Request 2 chunks,each chunk with 4 MPI processes,4 gpus per
chunk)
```

PBS Pro job operation（資源選取）

- #PBS -l resource option (host base resource)

```
#PBS -l select=1:ncpus=1
-> sequential job
(選擇1個chunk,每個chunk用1個core去執行1個MPI process)

#PBS -l select=2:ncpus=8:mpiprocs=8
-> MPI job
 (選擇2個chunks,每個chunk用8個cores去執行8個MPI processes,總共16個
MPI processes)

#PBS -l select=2:ncpus=8:mpiprocs=1:ompthreads=8
-> MPI/OpenMP Hybrid job
 (選擇2個chunks,每個chunk用8個cores去執行1個MPI process,但此MPI
process會fork出8個OpenMP threads,總共16個OpenMP threads)

#PBS -l select=2:ncpus=4:ngpus=4:mpiprocs=4
-> MPI/CUDA Hybrid job
 (選擇2個chunks,每個chunk用4個cores加4個GPU去執行4個MPI processes,總共
8個MPI processes)
```

PBS Pro job operation（資源選取）

- #PBS -l resource option（job-wide resource）

```
#PBS -l walltime=1:00:00
-> processing wall time is one hour
```

PBS Pro job operation（資源選取上限）

- ncpus, ngpus 不可超過每個計算節點的資源上限
  (40ncpus/1 node；4ngpus/1 GPU node)
  請注意: ncpus數值要小於等於40(資源上限)
  ngpus數值要小於等於4(資源上限)
- 錯誤範例:Job送出後之錯誤訊息如下
  #PBS -l select=1:ncpus=80:ngpus=8 （錯誤）

```
qsub: Job violates queue and/or server resource limits
```

- 範例修正後如下:
  #PBS -l select=2:ncpus=40:ngpus=4 （正確）

PBS Pro job operation (E-mail通知)

- PBS指令帶option -M E-mail設定收件人

```
# PBS -M user@example.com
```

- PBS指令帶option -m 加參數(a,b,e)指定發送E-mail時間點

```
# PBS -m be
```

- 發送E-mail時間點參數

Mail point argument	Description
a	job被批次系統中斷時發送E-mail
b	job 開始執行時發送E-mail
e	job 結束時發送E-mail
n	不發送E-mail

PBS Pro job operation (Output/Error file)

- 當PBS job結束後，在提交job的目錄下($PBS_O_WORKDIR)，預設會有2個檔案產生，如下：

  standard output stream to Output file

  standard error stream to Error file

- Output file format : <job_name>.o<jobid>

- Error file format : <job_name>.e<jobid>

PBS Pro job operation(Output/Error file合併)

- #PBS -j oe 把2個檔案合併成1個檔案到standard output file

```
# PBS -j oe
```

- #PBS -j eo 把2個檔案合併成1個檔案到standard error file

```
# PBS -j eo
```

- option for standard output and standard error files

option	Description
oe	Both files are merged into standard output file
eo	Both files are merged into standard error file
n	Both files are not merged (default)

PBS Pro job operation (Status)

- 常見的3項Job Status
- job在“ctest”Queue中排隊queue

```
$ qstat ctest
Job id                    Name             User            Time Use S Queue
------------------------------------------------------------------------------
12.localhost              example01        user01                 0 Q ctest
```

- 執行中run

```
$ qstat ctest
Job id                    Name             User            Time Use S Queue
------------------------------------------------------------------------------
12.localhost              example01        user01                 0 R ctest
```

- 完成Finished

```
$ qstat -H ctest
Job id                    Name             User            Time Use S Queue
------------------------------------------------------------------------------
12.localhost              example01        user01          00:00:55 F ctest
```

PBS Pro job operation (Status)

- -H to view jobs that have been deleted or finished

```
$ qstat -u $USER -H
srvc1:
                                                                Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
224225.srvc1    yenwen98 gtest    simple_cud    --    2  40    --  00:10 F   --
```

- -x to view all jobs, regardless of state

```
$ qstat -x 304293.srvc1
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
304293.srvc1      hello-world-mpi  yenwen988         00:00:00 F ctest
```

- -s show comment ; -w Allows display of wider fields

```
$ qstat -x 362950.srvc1 -sw
srvc1:
                                                                                Req'd  Req'd   Elap
Job ID                          Username        Queue           Jobname         SessID   NDS  TSK   Memory Time  S Time
------------------------------- --------------- --------------- --------------- ----
362950.srvc1                    yenwen988       gtest           simple_cuda_mpi   152073    2    8    --  00:10 F 00:00:07
Job run at Wed Aug 29 at 09:10 on (cn1017:ncpus=4:ngpus=4)+(cn1018:ncpus=4:ngpus=4) and finished
```

PBS Pro job operation (Status)

- qstat –xf <job ID>

```
$ qstat -xf 1200144.srvc1
Job Id: 1200144.srvc1
    Job_Name = intel-openmp-mpi-hello-world
    Job_Owner = yenwen988@clogin1
    resources_used.cpupercent = 0
    resources_used.cput = 00:00:00
    resources_used.mem = 932kb
    resources_used.ncpus = 8
    resources_used.vmem = 13232kb
    resources_used.walltime = 00:00:02
    job_state = F
    queue = ctest
    server = srvc1
    Checkpoint = u
    ctime = Mon Oct 22 15:21:22 2018
    Error_Path = clogin1:/home/yenwen988/1022/openmp_mpi/intel-
openmp-mpi-hello-world.e1200144
    exec_host = cn0305/1*4+cn0307/1*4
    exec_vnode = (cn0305:ncpus=4)+(cn0307:ncpus=4)
    Hold_Types = n
    Join_Path = oe
    Keep_Files = n
    Mail_Points = a
    mtime = Mon Oct 22 15:21:28 2018
    Output_Path = clogin1:/home/yenwen988/1022/openmp_mpi/intel-
openmp-mpi-hello-world.o1200144
    Priority = 0
    qtime = Mon Oct 22 15:21:23 2018
    Rerunable = True
    Resource_List.mpiprocs = 2
    Resource_List.ncpus = 8
    Resource_List.nodect = 2
    Resource_List.place = scatter
    Resource_List.select = 2:ncpus=4:mpiprocs=1:ompthreads=4
    Resource_List.walltime = 00:01:00
    stime = Mon Oct 22 15:21:26 2018
    session_id = 241235
    jobdir = /home/yenwen988
    substate = 92
    Variable_List = PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,
        PBS_O_HOME=/home/yenwen988,PBS_O_LOGNAME=yenwen988,

PBS_O_WORKDIR=/home/yenwen988/1022/openmp_mpi,PBS_O_LANG=en_US.UTF-
8,

PBS_O_PATH=/pkg/intel/2018_u1/compilers_and_libraries_2018.1.163/li
nux
/bin/intel64:/pkg/intel/2018_u1/compilers_and_libraries_2018.1.163/
linux/mpi/intel64/bin:/usr/lib64/qt-
3.3/bin:/usr/local/bin:/usr/bin:/usr/lo
cal/sbin:/usr/sbin:/sbin:/usr/sbin:/cm/local/apps/environment-
modules/3

.2.10/bin:/opt/pbs/bin:/home/yenwen988/.local/bin:/home/yenwen988/b
in,
        PBS_O_MAIL=/var/spool/mail/yenwen988,PBS_O_QUEUE=ctest,
        PBS_O_HOST=clogin1
    comment = Job run at Mon Oct 22 at 15:21 on
(cn0305:ncpus=4)+(cn0307:ncpus=4) and finished
    etime = Mon Oct 22 15:21:23 2018
    run_count = 1
    Stageout_status = 1
    Exit_status = 0
    Submit_arguments = hello_openmp_mpi_intel.sh
    pset = Leaf_sw=THIN_L108B
    history_timestamp = 1540192888
    project = ACD107023
```

PBS Pro job operation（删除job）

qdel指令格式

```
$ qdel <job ID>
```

範例

```
$ qdel 51
$ qdel 1234[].server
```

qdel指令带option “-W force” for 強制删除

```
$ qdel -W force <job ID>
```

PBS Pro job operation (Releasing jobs)

- Allowing job(s) that are held to be eligible for execution
- Using qrls command
- If no -h option is specified, the USER hold will be released.

```
Usage:   qrls [-h hold_list] job_id
Example: qrls  24.srvc1
```

複習Intel MPI hello-world (compile/link)

- cd到之前copy過來的1113/mpi_intel目錄下,載入Intel MPI module,將hello_mpi.c compile/link成執行檔,然後試著用mpirun執行Intel MPI hello-world程式,如下:

```
$ cd ~/1113/mpi_intel/
$ module purge
$ module load intel/2018_u1
$ module list
$ which mpicc
$ mpicc -o hello_mpi_intel.exe hello_mpi.c
$ ls -l hello_mpi_intel.exe
$ mpirun -np 2 ./hello_mpi_intel.exe
Hello world from processor clogin2, rank 0 out of 2 processors
Hello world from processor clogin2, rank 1 out of 2 processors
```

練習Intel MPI hello-world（編輯PBS job）

- 請用vim編輯hello_mpi_intel.sh將project ID換成自己的project ID

```
$ get_su_balance
$ vim hello_mpi_intel.sh
#!/bin/bash
#PBS -P ACD107023
#PBS -N intel-mpi-hello-world
#PBS -q ctest
#PBS -l select=2:ncpus=4:mpiprocs=4
#PBS -l place=scatter
#PBS -l walltime=00:01:00
#PBS -j oe
module purge
module load intel/2018_u1
module list
cd $PBS_O_WORKDIR
mpirun ./hello_mpi_intel.exe
```

練習Intel MPI hello-world（提交job;看結果）

- qsub提交job，qstat檢查status，檢查g輸出結果

```
$ qsub hello_mpi_intel.sh
464115.srvc1
$ qstat -x 464115.srvc1 -sw
Job run at $DATE on (cn0426 :ncpus=4)+(cn0428 :ncpus=4) and finished
$ ls
hello_mpi.c          hello_mpi_intel.sh          intel-mpi-hello-world.o464115  hello_mpi_intel.exe
$ more intel-mpi-hello-world.o464115
Currently Loaded Modulefiles:
  1) intel/2018_u1
Hello world from processor cn0426, rank 0 out of 8 processors
Hello world from processor cn0426, rank 1 out of 8 processors
Hello world from processor cn0426, rank 2 out of 8 processors
Hello world from processor cn0426, rank 3 out of 8 processors
Hello world from processor cn0428, rank 4 out of 8 processors
Hello world from processor cn0428, rank 5 out of 8 processors
Hello world from processor cn0428, rank 6 out of 8 processors
Hello world from processor cn0428, rank 7 out of 8 processors
```

練習（qstat）

- 1. 列出目前系統所有的queue

  ```
  $ qstat -Q
  ```

- 2. 列出ct400 queue的詳細資料 & ncpus最小需求為多少

  ```
  $ qstat -Qf ct400 |grep min
    resources_min.ncpus = 161
  ```

- 3. 查詢自己qsub過的PBS job 且status已Finished的紀錄

  ```
  $ qstat -u $USER -H
  ```

- 4. 查詢目前已加入ctest queue的nodes有幾台

  ```
  $ pbsnodes -a |grep ctest |wc -l
  $ 3
  ```

PBS dependent jobs範例

- 例如有下列3個PBS scripts要依序執行

  1:pre_job.sh -> 2:main_job.sh -> 3:post_job.sh

```
$ vim go.sh
#!/bin/bash
JID1=`qsub -h pre_job.sh`
JID2=`qsub -W depend=afterok:$JID1 main_job.sh`
qsub -W depend=afterok:$JID2 post_job.sh
qrls $JID1
$ chmod u+x go.sh
$ ./go.sh
```

1) qsub pre_job.sh加option -h先hold住job
2) qsub main_job.sh加option -W depend=afterok:$JID1
3) qsub post_job.sh加option -W depend=afterok:$JID2
4) relase hole on job pre_job.sh

練習(PBS dependent jobs & PBS環境變數)

- 請練習用PBS dependent job範例，依序submit 3個job script,
  每個job script各執行Intel MPI hellow-world,
  並將結果輸出到同一個檔案/work1/$USER/output file

練習(PBS dependent jobs & PBS環境變數)

- cd到之前copy過來的1113/depend目錄下,
  請用vim編輯job scrip將project ID換成自己的project ID
  check job scrip內容，如下操作:

```
$ cd ~/1113/depend
$ ls
go.sh  hello_mpi_intel.exe  main_job.sh  post_job.sh  pre_job.sh
$ get_su_balance
$ vim pre_job.sh
$ vim main_job.sh
$ vim post_job.sh
export TMPDIR=/work1/$USER
cd $TMPDIR
cat /dev/null > output
echo PBS_JOBNAME=$PBS_JOBNAME >> output
echo PBS_JOBID=$PBS_JOBID >> output
mpirun $PBS_O_WORKDIR/hello_mpi_intel.exe >>output
```

練習(PBS dependent jobs & PBS環境變數)

- check go.sh script內容 & 執行go.sh & qstat -u $USER，check PBS job script的standard output file操作如下:

```
$ cat go.sh
#!/bin/bash
JID1=‘qsub -h pre_job.sh‘
JID2=‘qsub -W depend=afterok:$JID1 main_job.sh‘
qsub -W depend=afterok:$JID2 post_job.sh
qstat -u $USER
qrls $JID1
$ ./go.sh ;qstat -u $USER
630691.srvc1
$ ls
 main-job-script.o630690  post-job-script.o630691  pre-job-
script.o630689
$ cat pre-job-script.o630689
Original TMPDIR=/var/tmp/pbs.630689.srvc1
after export TMPDIR=/work1/yenwen988
$ cat /work1/$USER/output
```

練習(PBS dependent jobs & PBS環境變數)

- check /work1/$USER/output file的內容:

```
$ cat /work1/$USER/output
PBS_JOBNAME=pre-job-script
PBS_JOBID=630689.srvc1
Hello world from processor cn0429, rank 2 out of 4 processors
Hello world from processor cn0429, rank 3 out of 4 processors
Hello world from processor cn0428, rank 0 out of 4 processors
Hello world from processor cn0428, rank 1 out of 4 processors
PBS_JOBNAME=main-job-script
PBS_JOBID=630690.srvc1
Hello world from processor cn0429, rank 2 out of 4 processors
Hello world from processor cn0429, rank 3 out of 4 processors
Hello world from processor cn0428, rank 0 out of 4 processors
Hello world from processor cn0428, rank 1 out of 4 processors
PBS_JOBNAME=post-job-script
.......
```

平行程式平台與環境

- 平行運算
- MPI / OpenMP / CUDA
- OpenMP
- MPI
- MPI vs OpenMP
- Hybrid OpenMP+MPI
- CUDA

平行運算

- 平行運算是指將原來需要在一個CPU執行的程式分散在多個CPU同時執行
- 為什麼要平行運算
  - 節省時間
  - 即時性
  - 使用更多來自網路上的資源

平行程式平台與環境

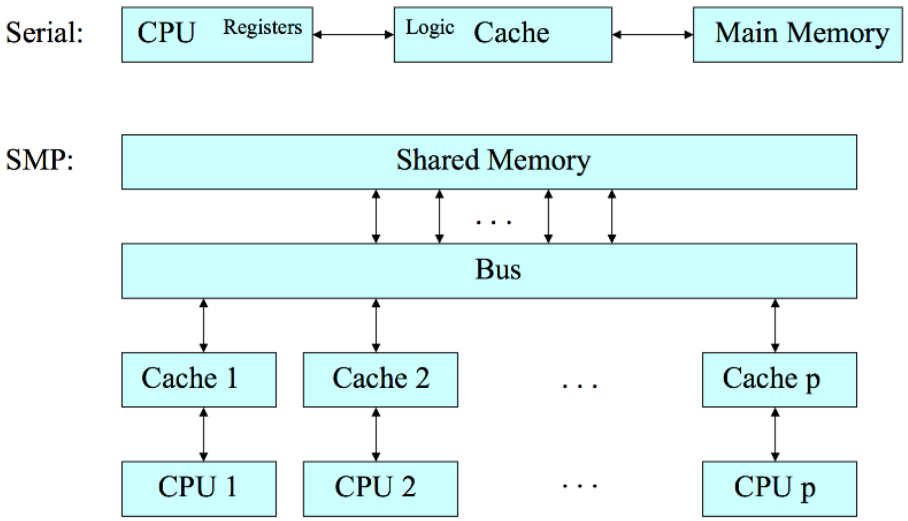
- MPI / OpenMP / CUDA

	MPI	OpenMP	CUDA
硬體架構	多電腦系統架構	多核心CPU處理架構	NVIDIA GPU架構
執行緒數目	核心數量	核心數量	極大量
軟體撰寫難度	普通	低	視需求而定

- 想要操作多台電腦來寫平行程式，可以使用MPI
- 想利用電腦多核心來寫平行程式，可以使用OpenMP
- 想要在NVIDIA的顯示卡上寫平行程式，則可以使用CUDA

平行程式平台與環境

■ OpenMP （Open Multi-Processing）



OpenMP（Open Multi-Processing）是一套支援跨平台共享記憶體方式的多執行緒並行的編程API，使用C,C++和Fortran語言，可以在大多數的處理器體系和作業系統中執行，包括Solaris, AIX, HP-UX, GNU/Linux, Mac OS X, 和Microsoft Windows。包括一套編譯器指令、庫和一些能夠影響執行行為的環境變數。(節錄自維基百科)

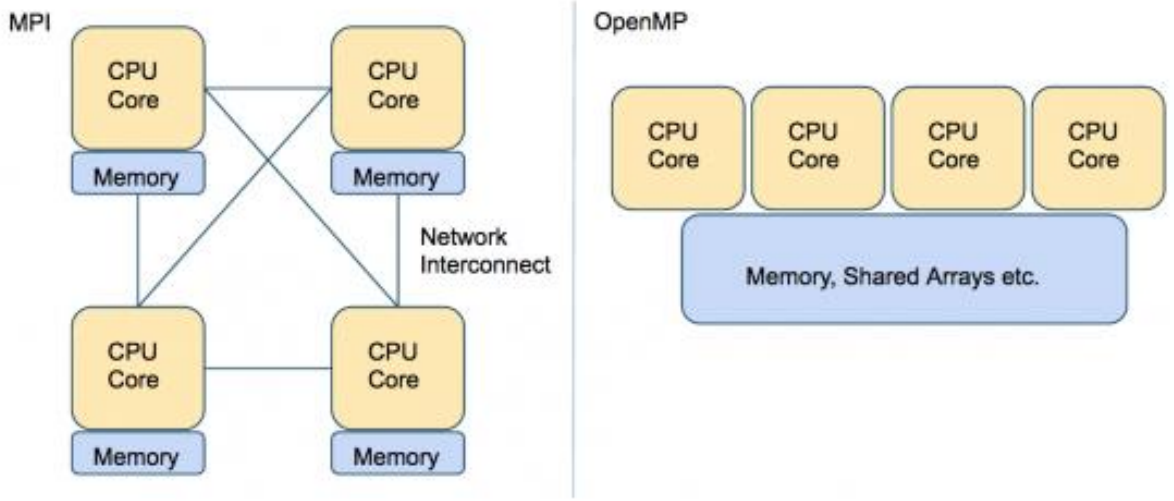
平行程式平台與環境

- MPI (Message Passing Interface)

訊息傳遞介面/介面（英語：Message Passing Interface，縮寫MPI）是一個平行計算的應用程式介面（API），常在超級電腦、電腦叢集等非共享內存環境程式設計。主要的功能是在行運算之間的各個node 的資料交換
(節錄自維基百科)

平行程式平台與環境

- MPI vs OpenMP



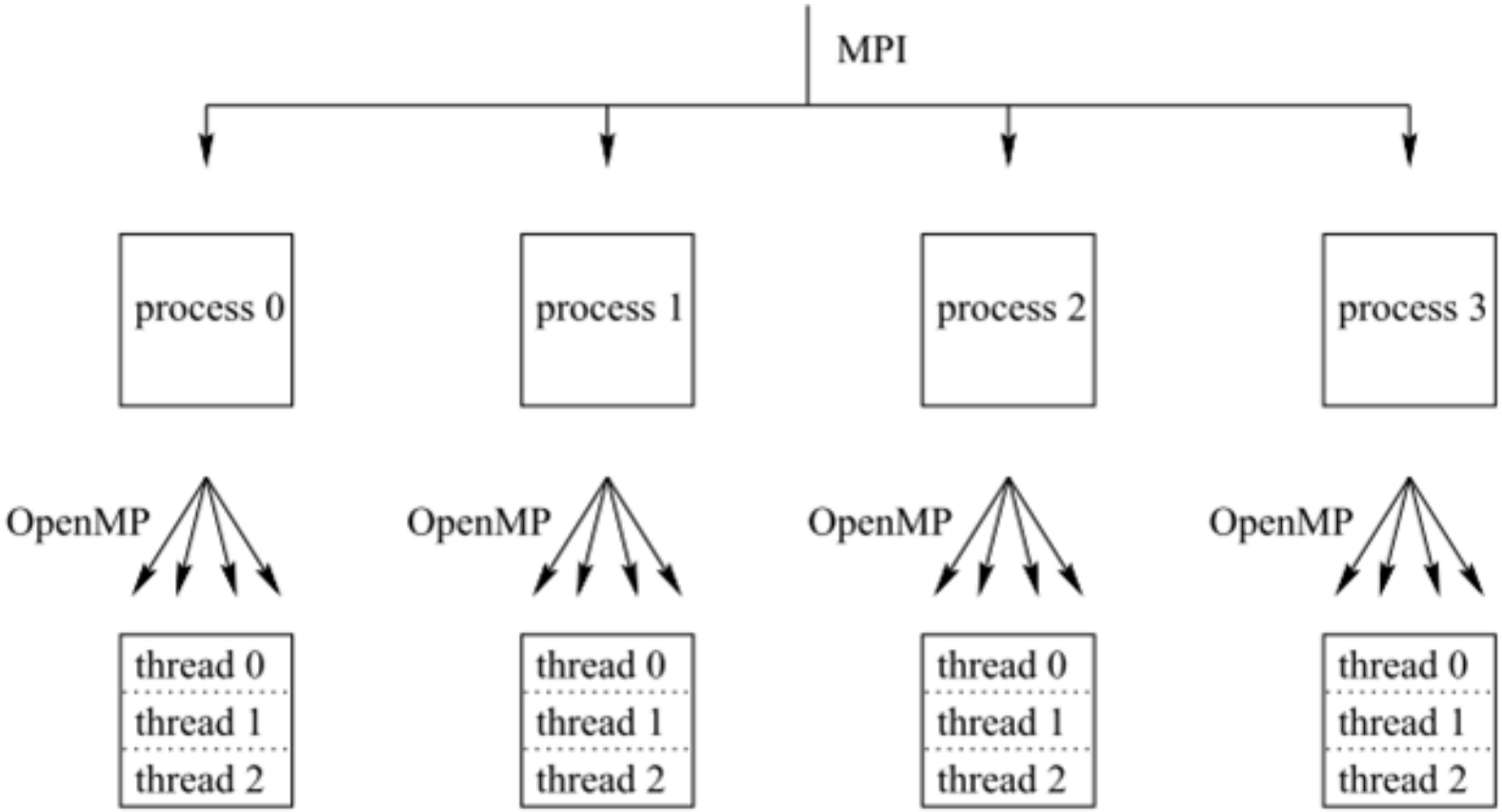
- MPI程式屬於Distributed memory架構，多台機器透過網路對同一個程序做處理，所以可以利用網路串連各式各樣的機器。
- OpenMP為SMP(Shared Memory Processors)架構，多顆處理器對同一塊記憶體作存取，通常處理器跟記憶體是在一台機器上，所以這種架構只能在單一機器上運作

平行程式平台與環境

- OpenMP (Shared memory) vs MPI (Distributed Memory)
  - Shared memory架構
  - 優點是在記憶體的存取時比較容易，而且因為是在同一塊記憶體上面操作，所以對於記憶體的存取通常也比較快。
  - 缺點就是很難擴充處理器，如果還要增加處理器的話，會造成處理器到記憶體data path的overhead，處理器愈多，效能愈差
  - Distributed Memory架構
  - 優點是可以很容易利用網路將所有的機器串連在一起，而且可以很快速的存取記憶體，並不會造成處理器到記憶體之間的overhead。
  - 缺點就是設計師要花更多時間在處理器與處理器之間的資料交換，還有網路速度的問題。網路速度現在在平行電腦上一般都使用光纖加快網路存取速度。

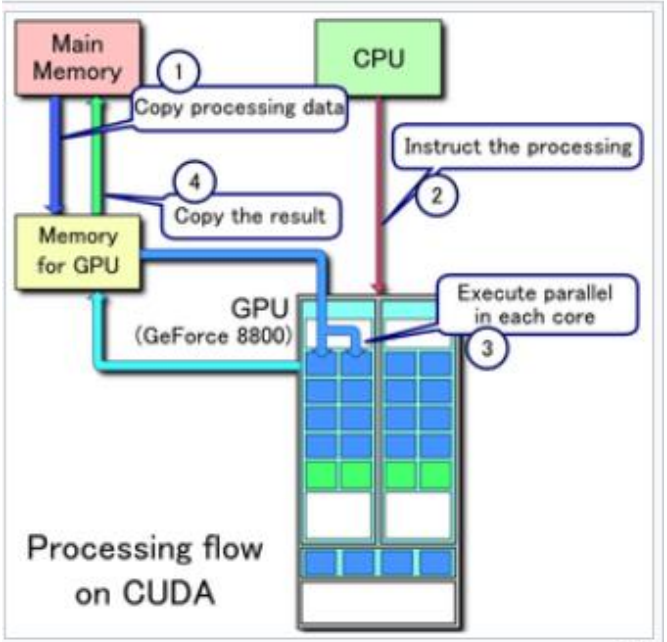
平行程式平台與環境

- Hybrid OpenMP+MPI



平行程式平台與環境

■ CUDA



CUDA處理流例子
1.將主記憶體的處理資料複製入視訊記憶體中
2. CPU指令驅動GPU
3. GPU每一核心並列處理
4. GPU將視訊記憶體結果傳回主記憶體

CUDA (Compute Unified Device Architecture) 是NVIDIA提出，可在NVIDIA圖形處理器進行平行運算的計算環境。程式設計者可以利用CUDA的C語言擴充 (extension) 直接用C語言寫程式，設計資料分配 (data decomposition) 及程式流程將運算工作分配到上千個執行緒 (threads)及圖形處理器中數以百計的計算核心 (cores)。(節錄自維基百科)

平行程式實作範例

- OpenMP範例（GCC & Intel）
- MPI範例（Intel）
- Hybrid OpenMP+MPI範例（Intel）
- CUDA範例
- Hybrid CUDA+MPI範例（GCC）
- JOB Array說明&範例

平行程式平台與環境

小提醒

- 選擇正確登入節點(glogin clogin)
- /pkg/mpi_sample目錄下,有放上課要demo MPI job的檔案 要練習的請自行copy檔案到自己的家目錄
- 載入必要module
- 用which command確認一下compiler例如mpiicc 或 mpicc
- 請儘量用intel complier來進行編譯，效能上較優。
- 選擇正確(所需)的Queue
- 在intel MPI下，使用大規模計算(大約100個節點以上)需要導入此環境變數，才不會造成執行錯誤
  export I_MPI_HYDRA_BRANCH_COUNT=-1
- 如果要執行大量IO處理程試時，儘量放置/work1 目錄下執行但需注意有28天存放限制，執行完成後可將結果放回/home目錄。

OpenMP Hello word範例

- OpenMP Hello world source code

```
$ more hello_openmp.c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

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

/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(nthreads, tid)
  {

  /* Obtain thread number */
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);

  /* Only master thread does this */
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }

  }  /* All threads join master thread and disband */

}
```

OpenMP Hello word範例

- How to Compile/Link OpenMP program

Compiler example	Option	OMP_NUM_THREADS not set
gcc/6.3.0 (gcc, g++, gfortran)	-fopenmp	as many threads as available cores
intel/2018_u1 (icc, icpc, ifort)	-qopenmp -fopenmp	as many threads as available cores
pgi/17.10 (pgcc, pgc++, pgfortran)	-mp	one threads

OpenMP Hello word範例 (GCC)

- GCC compiler example

```
$ module list
1) gcc/6.3.0
```

- Compile/Link

```
$ gcc -o hello_openmp_gcc.exe hello_openmp.c -fopenmp
```

- 直接執行,可看到共40個threads (40 threads/1node)

```
$ ./hello_openmp_gcc.exe
Hello World from thread = 8
Hello World from thread = 6
Hello World from thread = 39
…
…
```

OpenMP Hello word範例 (GCC)

- export OMP_NUM_THREADS=4 後執行結果可看到4個 threads

```
$ export OMP_NUM_THREADS=4

$ ./hello_openmp_gcc.exe
Hello World from thread = 0
Number of threads = 4
Hello World from thread = 3
Hello World from thread = 1
Hello World from thread = 2
```

OpenMP Hello word範例（Intel）

- Intel compiler example

```
$ module list
Currently Loaded Modulefiles:
  1) intel/2018_u1
```

- Compile/Link

```
$ icc -o hello_openmp_intel.exe hello_openmp.c -qopenmp
```

OpenMP Hello word範例（Intel）

- 直接執行,可看到共40個threads（40 threads/1 node）

```
$ unset OMP_NUM_THREADS
$ env |grep OMP

$ ./hello_openmp_intel.exe
Hello World from thread = 30
Hello World from thread = 14
Hello World from thread = 35
…
```

- export OMR_NUM_THREADS=4後執行結果可看到4個 threads

```
$ export OMP_NUM_THREADS=4

$ ./hello_openmp_intel.exe
Hello World from thread = 0
Number of threads = 4
Hello World from thread = 3
Hello World from thread = 1
Hello World from thread = 2
```

OpenMP Hello word範例 (GCC)

- 建立PBS job scrip 選擇1個node 8個threads

```
$ cat hello_openmp_gcc.sh
#!/bin/bash
#PBS -P ACD107023
#PBS -N hello-world-openmp
#PBS -l select=1:ncpus=8:mpiprocs=1:ompthreads=8
#PBS -l walltime=00:01:00
#PBS -q ctest
#PBS -j oe
module purge
module load gcc/6.3.0
cd $PBS_O_WORKDIR
./hello_openmp_gcc.exe
```

OpenMP Hello word範例 (GCC)

- 執行job 與顯示結果，可看出用8threads來執行

```
$ qsub hello_openmp_gcc.sh
410734.srvc1
$ cat hello-world-mpi.o362951
Hello World from thread = 3
Hello World from thread = 0
Number of threads = 8
Hello World from thread = 7
Hello World from thread = 5
…
```

練習 (PGI OpenMP Hello_World)

- 請練習用PBS script提交job，使用20個threads之PGI OpenMP程式

練習 (PGI OpenMP Hello_World)

- cd到之前copy過來的1113/openmp目錄下, 載入pgi/17.10 module, 將hello_openmp.c compile/link成執行檔, 然後試著執行PGI OpenMP hello-world程式, 如下:

```
$ cd ~/1113/openmp
$ ls
$ module purge
$ module load pgi/17.10
$ which pgcc
/pkg/pgi/17.10/linux86-64/17.10/bin/pgcc
$ pgcc -o hello_openmp_pgi.exe hello_openmp.c -mp
$ ls -l hello_openmp_pgi.exe
$ export OMP_NUM_THREADS=20
$ ./hello_openmp_pgi.exe
```

練習 (PGI OpenMP Hello_World)

- 請用vim編輯hello_openmp_pgi.sh將project ID換成自己的project ID，ncpus & ompthreads改成20

```
$ get_su_balance
$ vim hello_openmp_pgi.sh
#!/bin/bash
#PBS -P ACD107023
#PBS -N pgi-openmp-hello-world
#PBS -q ctest
#PBS -l select=1:ncpus=20:mpiprocs=1:ompthreads=20
#PBS -l walltime=00:01:00
#PBS -j oe
module purge
module load pgi/17.10
module list
cd $PBS_O_WORKDIR
./hello_openmp_pgi.exe
```

練習 (PGI OpenMP Hello_World)

- submit PBS job script hello_openmp_pgi.sh check result

```
$ qsub hello_openmp_pgi.sh
523333.srvc1
$ ls
$ more pgi-openmp-hello-world.o523333
Currently Loaded Modulefiles:
  1) pgi/17.10
Hello World from thread = 15
Hello World from thread = 6
Hello World from thread = 2
Hello World from thread = 12
Hello World from thread = 4
Hello World from thread = 16
Hello World from thread = 10
Hello World from thread = 8
Hello World from thread = 3
Hello World from thread = 18
Hello World from thread = 7
......
```

MPI Hello word範例

- Hello world source code

```
$ more hello_mpi.c
#include <mpi.h>
#include <stdio.h>

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

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

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

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

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

MPI Hello word範例

- Hello world source code

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

main (int argc, char **argv)
{
  MPI_Init(&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank (MPI_COMM_WORLD, &myid);
  . . .
  . . .
  MPI_Finalize();
  return 0;
}
```

啟動該程式在多個 CPU 上的平行計算工作

得知參與平行計算的 CPU 個數 (nproc)

得知我是第幾個 CPU (myid) from 0

結束平行計算工作

MPI Hello word範例（Intel）

- Intel compiler example

```
$ module load intel/2018_u1
```

- Compile/Link

```
$ mpicc -o hello_mpi_intel.exe hello_mpi.c
$ mpiicc -o hello_mpi_intel.exe hello_mpi.c
```

- 在login node用mpirun 來跑MPI hello world程式

```
$ mpirun -np 2 ./hello_mpi_intel.exe
Hello world from processor clogin1, rank 1 out of 2 processors
Hello world from processor clogin1, rank 0 out of 2 processors
```

MPI Hello word範例（Intel）

■ 建立PBS Job script（Intel MPI example）

```
$ get_su_balance
467974.3332,ACD107023,國網中心內部計畫

$ vim hello_mpi_intel.sh
#!/bin/bash
#PBS -P ACD107023
#PBS -N intel-mpi-hello-world
#PBS -q ctest
#PBS -l select=2:ncpus=4:mpiprocs=4
#PBS -l place=scatter
#PBS -l walltime=00:01:00
#PBS -j oe

module purge
module load intel/2018_u1
module list
cd $PBS_O_WORKDIR

mpirun ./hello_mpi_intel.exe
```

MPI Hello word範例（Intel ）

- qsub pbs job script

```
$ qsub hello_mpi_intel.sh
452785.srvc1

$ qstat -xsw 452785.srvc1

srvc1:
                                                                                                                Req'd  Req'd
Elap
Job ID                        Username        Queue           Jobname         SessID   NDS  TSK   Memory Time  S
Time
-----------------------  --------------- --------------- --------------- -------- ---- ----- ------ ----- - -----
452785.srvc1                 yenwen988       ctest           intel-mpi-hello    62538    2     8    --  00:01 F
00:00:02
   Job run at Sun Sep 09 at 20:34 on (cn0427:ncpus=4)+(cn0431:ncpus=4) and finished
```

MPI Hello word範例（Intel ）

- 顯示結果，共執行8次Hello world

```
$ more intel-mpi-hello-world.o452785
Currently Loaded Modulefiles:
  1) intel/2018_u1
Hello world from processor cn0431, rank 4 out of 8 processors
Hello world from processor cn0431, rank 5 out of 8 processors
Hello world from processor cn0431, rank 6 out of 8 processors
Hello world from processor cn0431, rank 7 out of 8 processors
Hello world from processor cn0427, rank 0 out of 8 processors
Hello world from processor cn0427, rank 1 out of 8 processors
Hello world from processor cn0427, rank 2 out of 8 processors
Hello world from processor cn0427, rank 3 out of 8 processors
```

練習 (GCC MPI Hello-World)

- 請練習用PBS script提交job，選擇2個chunks，每個chunk用4個cores去執行4個MPI processes, 總共8個MPI processes

練習 (GCC MPI Hello-World)

- cd到之前copy過來的1113/mpi_gcc目錄下,載入 module openmpi/gcc/64/1.10.4，將hello_mpi.c compile/link成執行檔,然後用mpirun -np 2 跑GCC MPI hello-world程式,如下:

```
$ cd ~/1113/mpi_gcc
$ ls
$ module purge
$ module load openmpi/gcc/64/1.10.4
$ which mpicc
$ mpicc -o hello_mpi_gcc.exe hello_mpi.c
$ ls -l hello_mpi_gcc.exe
$ mpirun -np 2 ./hello_mpi_gcc.exe
Hello world from processor cloginl, rank 0 out of 2 processors
Hello world from processor cloginl, rank 1 out of 2 processors
```

練習 (GCC MPI Hello-World)

- 請用vim編輯hello_mpi_gcc.sh將project ID換成自己的project ID， & select=2:ncpus=4:mpiprocs=4

```
$ get_su_balance
$ vim hello_mpi_gcc.sh
#!/bin/bash
#PBS -P ACD107023
#PBS -N gcc-mpi-hello-world
#PBS -q ctest
#PBS -l select=2:ncpus=4:mpiprocs=4
#PBS -l place=scatter
#PBS -l walltime=00:01:00
#PBS -j oe
module purge
module load openmpi/gcc/64/1.10.4
module list
cd $PBS_O_WORKDIR
PROCESSES=`cat $PBS_NODEFILE | wc -w`
mpirun -np $PROCESSES -hostfile $PBS_NODEFILE \
       --mca pml cm --mca mtl psm2 \
       ./hello_mpi_gcc.exe
```

練習 (GCC MPI Hello-World)

- submit PBS job script hello_mpi_gcc.sh & check result

```
$ qsub hello_mpi_gcc.sh
1373136.srvc1
$ ls
$ gcc-mpi-hello-world.o1373136
$ cat gcc-mpi-hello-world.o1373136
Currently Loaded Modulefiles:
  1) openmpi/gcc/64/1.10.4
Hello world from processor cn0321, rank 0 out of 8 processors
Hello world from processor cn0321, rank 2 out of 8 processors
Hello world from processor cn0321, rank 3 out of 8 processors
Hello world from processor cn0321, rank 1 out of 8 processors
Hello world from processor cn0322, rank 7 out of 8 processors
Hello world from processor cn0322, rank 4 out of 8 processors
Hello world from processor cn0322, rank 5 out of 8 processors
Hello world from processor cn0322, rank 6 out of 8 processors
```

Hybrid OpenMP+MPI Hello word範例

- OpenMP+MPI Hello world source code

```
$ more hello_openmp_mpi.c
#include <stdio.h>
#include "mpi.h"
#include <omp.h>

int main(int argc, char *argv[]) {
  int numprocs, rank, namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int iam = 0, np = 1;

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

  #pragma omp parallel default(shared) private(iam, np)
  {
    np = omp_get_num_threads();
    iam = omp_get_thread_num();
    printf("Hello from thread %d out of %d from process %d out of %d on %s\n",
           iam, np, rank, numprocs, processor_name);
  }

  MPI_Finalize();
}
```

Hybrid OpenMP+MPI Hello word範例（Intel）

- Load module

```
$ module load intel/2018_u1
```

- Compile/Link

```
$ mpiicc -o hello_openmp_mpi.exe hello_openmp_mpi.c -fopenmp
```

- 在login node用mpirun來跑OpenMP+MPI程式

```
$ export OMP_NUM_THREADS=4
$ mpirun -np 2 ./hello_openmp_mpi_intel.exe
Hello from thread 0 out of 4 from process 0 out of 2 on clogin1
Hello from thread 2 out of 4 from process 0 out of 2 on clogin1
Hello from thread 3 out of 4 from process 0 out of 2 on clogin1
Hello from thread 1 out of 4 from process 0 out of 2 on clogin1
Hello from thread 0 out of 4 from process 1 out of 2 on clogin1
Hello from thread 1 out of 4 from process 1 out of 2 on clogin1
Hello from thread 2 out of 4 from process 1 out of 2 on clogin1
Hello from thread 3 out of 4 from process 1 out of 2 on clogin1
```

Hybrid OpenMP+MPI Hello word範例（Intel ）

- 建立PBS job script 選擇2個node 4個therads

```
$ vim hello_openmp_mpi_intel.sh
#!/bin/bash
#PBS -P ACD107023
#PBS -N intel-openmp-mpi-hello-world
#PBS -q ctest
#PBS -l select=2:ncpus=4:mpiprocs=1:ompthreads=4
#PBS -l place=scatter
#PBS -l walltime=00:01:00
#PBS -j oe

module purge
module load intel/2018_u1
module list
cd $PBS_O_WORKDIR

mpirun ./hello_openmp_mpi_intel.exe
```

Hybrid OpenMP+Intel MPI Hello word範例

- 執行job 與 job status

```
$ qsub hello_openmp_mpi_intel.sh
452159.srvc1

$ qstat -xsw 452159.srvc1

srvc1:
                                                                                  Req'd  Req'd   Elap
Job ID                          Username        Queue           Jobname         SessID   NDS  TSK   Memory Time  S Time
------------------------------- --------------- --------------- --------------- -------- ---- ----- ------ ----- - -----
452159.srvc1                    yenwen988       ctest           intel-openmp-mp    38406    2     8    --  00:01 F 00:00:02
 Job run at Sun Sep 09 at 14:53 on (cn0458:ncpus=4)+(cn0459:ncpus=4) and finished
```

Hybrid OpenMP+MPI Hello word範例（Intel ）

- Check結果，可看出用4threads來執行, 並在二個節點上執行

```
$ more intel-openmp-mpi-hello-world.o452159
Currently Loaded Modulefiles:
  1) intel/2018_u1
Hello from thread 0 out of 4 from process 0 out of 2 on cn0458
Hello from thread 3 out of 4 from process 0 out of 2 on cn0458
Hello from thread 1 out of 4 from process 0 out of 2 on cn0458
Hello from thread 2 out of 4 from process 0 out of 2 on cn0458
Hello from thread 1 out of 4 from process 1 out of 2 on cn0459
Hello from thread 2 out of 4 from process 1 out of 2 on cn0459
Hello from thread 0 out of 4 from process 1 out of 2 on cn0459
Hello from thread 3 out of 4 from process 1 out of 2 on cn0459
```

CUDA Hello word範例

- 小提醒
  - 登入GPU login node（glogin1）可方便測試執行檔
  - 編譯時需load cuda module
  - PBS script job派送時，需選擇含有GPU的queue
  - ngpus資源最大為4，若需求超過4GPU時，請用select來達成需求

CUDA Hello word範例

- GPU中允許active很多threads，下面的例子在kernel函式helloFromGPU(),指定了一個含有2塊block，每個block 3個threads

```
#include <stdio.h>

// __global__:
// function will be called on the CPU and executed on the GPU
__global__ void helloFromGPU()
{
    // blockIdx.x for block index
    // threadIdx.x for thread index
    printf("Hello World from GPU! block %d thread %d \n", blockIdx.x, threadIdx.x);
}

int main(void)
{
    // hello from cpu
    printf("\nHello World from CPU!\n\n");

    // kernel(GPU code) configuration: total threads = blockNum x threadNum
    int blockNum = 2;
    int threadNum = 3;

    // hello from gpu: call from the host thread to the code on the device side
    helloFromGPU <<<blockNum, threadNum >>>();
    cudaDeviceReset(); //destroy and clean up all resources
    //cudaDeviceSynchronize();

    return 0;
}
```

CUDA Hello word範例

- Load module

```
$ module load cuda/9.1.85
```

- Compile/Link

```
$ nvcc hello_cuda.cu -o hello_cuda.exe
```

- 直接執行,可以看到每一行對應到方才active的block & thread (2*3=6) (blockNum* threadNum)

```
$ ./hello_cuda.exe

Hello World from CPU!

Hello World from GPU! block 0 thread 0
Hello World from GPU! block 0 thread 1
Hello World from GPU! block 0 thread 2
Hello World from GPU! block 1 thread 0
Hello World from GPU! block 1 thread 1
Hello World from GPU! block 1 thread 2
```

CUDA Hello word範例

- 建立PBS job script 選擇1個CPU ,1個GPU

```
$ cat hello_cuda.sh
#!/bin/bash
#PBS -P ACD107023
#PBS -N hello-world-cuda-simple
#PBS -l select=1:ncpus=1:ngpus=1
#PBS -l walltime=00:01:00
#PBS -q gp4
#PBS -j oe
module purge
module load cuda/9.1.85
cd $PBS_O_WORKDIR
./hello_cuda.exe
```

CUDA Hello word範例

- 執行job 與 job status

```
$ qsub hello_cuda.sh
424634.srvc1
$ qstat -xs 424634.srvc1

srvc1:
                                                          Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
424634.srvc1    yenwen98 gp4      hello-worl    --    1   1    --  00:01 Q   --
   Not Running: Insufficient amount of resource: Qlist
```

CUDA Hello word範例

- 查看結果，每個hello world對應一個GPU thread

```
$ more hello-world-cuda-simple.o424634

Hello World from CPU!

Hello World from GPU! block 1 thread 0
Hello World from GPU! block 1 thread 1
Hello World from GPU! block 1 thread 2
Hello World from GPU! block 0 thread 0
Hello World from GPU! block 0 thread 1
Hello World from GPU! block 0 thread 2
```

Hybrid CUDA+openMPI (simpleMPI)範例

- 範例使用OpenMPI來編譯和執行CUDA MPI程式
- 在/pkg/mpi_sample/cuda_mpi目錄下,有此範例的source code
- 請cd 到之前copy過來的目錄

```
$ cd ~/1113/cuda_mpi
$ ls
Makefile  readme.txt  run_4gpu.sh  simpleMPI.cpp  simpleMPI.cu
simpleMPI_cuda_gcc.sh  simpleMPI.h
```

Hybrid CUDA+openMPI (simpleMPI)範例

- 載入環境模組 openmpi/gcc/64/1.10.4 & cuda/8.0.61
- 用which指令檢查你是否正確的載入了nvcc 和mpicxx

```
$ module purge
$ module load openmpi/gcc/64/1.10.4
$ module load cuda/8.0.61
$ module list
Currently Loaded Modulefiles:
  1) openmpi/gcc/64/1.10.4   2) cuda/8.0.61
$ which mpicxx
/usr/mpi/gcc/openmpi-1.10.4-hfi/bin/mpicxx
$ which nvcc
/pkg/cuda/8.0.61/bin/nvcc
```

- 用mpicxx compile simpleMPI.cpp產生object code simpleMPI_mpi.o
- 用nvcc compile simpleMPI.cu產生object code simpleMPI_cu.o

```
$ mpicxx -o simpleMPI_mpi.o -c simpleMPI.cpp  -I/pkg/cuda/8.0.61/samples/common/inc
$ nvcc -o simpleMPI_cu.o -c simpleMPI.cu -gencode arch=compute_60,code=sm_60
```

Hybrid CUDA+openMPI (simpleMPI)範例

- 檢查被編譯出來的檔案
- Linking之前所產生的2個objet file檔案產生執行檔

```
$ ls *.o
simpleMPI_cu.o  simpleMPI_mpi.o
$ mpicxx -o simpleMPI simpleMPI_mpi.o simpleMPI_cu.o  -
L/pkg/cuda/8.0.61/lib64 -lcudart
$ ls -l simpleMPI
-rwxr-xr-x 1 yenwen988 TRI107100 117728 Aug 27 15:20 simpleMPI
```

- 在glogin1用mpirun來跑CUDA+openMPI程式

```
$ mpirun -np 2 ./run_4gpu.sh
Running on 2 nodes
Average of square roots is: 0.667279
PASSED
$
```

Hybrid CUDA+openMPI (simpleMPI)範例

- 建立一個run_4gpu.sh的script,將processes分散到不同的4個GPU device

```
$ vim run_4gpu.sh
#!/bin/bash

#load cuda library
module load cuda/8.0.61
#location of Binary
EXEC_DIR=`pwd`
APP=$EXEC_DIR/simpleMPI
lrank=$OMPI_COMM_WORLD_LOCAL_RANK
case ${lrank} in
[0])
  export CUDA_VISIBLE_DEVICES=0
  $APP
  ;;
[1])
  export CUDA_VISIBLE_DEVICES=1
  $APP
  ;;
[2])
  export CUDA_VISIBLE_DEVICES=2
  $APP
  ;;
[3])
  export CUDA_VISIBLE_DEVICES=3
  $APP
  ;;
esac
```

Hybrid CUDA+openMPI (simpleMPI)範例

■ 建立PBS job script

```
$ vim simpleMPI_cuda_gcc.sh
#!/bin/bash
#PBS -P ACD107023
#PBS -N simple_cuda_mpi_job
#PBS -l select=2:ncpus=4:ngpus=4:mpiprocs=4
#PBS -l walltime=00:01:00
#PBS -q gtest
#PBS -j oe

module purge
module load openmpi/gcc/64/1.10.4
module load cuda/8.0.61
cd $PBS_O_WORKDIR

mpirun -np 8 -hostfile $PBS_NODEFILE \
    --mca pml cm --mca mtl psm2 \
        ./run_4gpu.sh
```

- #PBS -P XXX 請改成自己的project ID
- #PBS -l select=2:ncpus=4:ngpus=4:mpiprocs=4

選擇2個chunks,每個chunk用4個cores加4個GPU去執行4個MPI processes,總共8個MPI processes

Hybrid CUDA+openMPI (simpleMPI)範例

執行job

```
$ qsub simpleMPI_cuda_gcc.sh
362950.srvc1
```

顯示job (status is Queue)

```
$ qstat -xsw 362950.srvc1

srvc1:
                                                                                Req'd  Req'd   Elap
Job ID                          Username        Queue           Jobname         SessID   NDS  TSK   Memory Time  S Time
------------------------------- --------------- --------------- --------------- -------- ---- ----- ------ ----- - -----
362950.srvc1                    yenwen988       gtest           simple_cuda_mpi      --     2     8    --  00:10 Q  --
  Not Running: Insufficient amount of resource: Qlist
```

Hybrid CUDA+openMPI (simpleMPI)範例

■ 顯示job (status is Finished) & 查看結果

```
$ qstat -xsw 362950

srvc1:
                                                                                  Req'd  Req'd   Elap
Job ID                          Username        Queue           Jobname         SessID   NDS  TSK   Memory Time  S Time
------------------------------- --------------- --------------- --------------- -------- ---- ----- ------ ----- - -----
362950.srvc1                    yenwen988       gtest           simple_cuda_mpi   152073    2     8    --  00:10 F
00:00:07
   Job run at Wed Aug 29 at 09:10 on
(cn1017:ncpus=4:ngpus=4)+(cn1018:ncpus=4:ngpus=4) and finished


$ more outputs/362950.srvc1.OU
Running on 4 nodes
Average of square roots is: 0.667305
PASSED
```

練習

- 可測試上述內容，但由於GPU queue 常常busy，可能需等待數天才能執行。

Job Array: Concept

- 使用相同的PBS脚本，提交多個作業，每個作業運行環境基本一致，除了個別運行參數有所不同。

- What is a job array?
  - Collection of subjobs differing by a single parameter

- Why use job arrays?
  - Allows users to group similar jobs and submit as one job
  - Allows users to query, modify, and display the set as a single unit

- Multiple uses
  - Job arrays are a great way to organize the execution of multiple short jobs
    - Jobs with similar properties
    - Jobs using similar data with different algorithms
    - Jobs using a serial input file numbering system, e.g. file01, file02, file03
  - Scientific applications
    - BLAST
    - Molecular modeling
    - Drug screening
    - Discrete optimization

Array job範例

- Intel MPI Hello world demo Array job（內含3個sub job）
- 請cd到之前copy過來的目錄array_mpi_intel
- 載入 module intel/2018_u1，將hello_mpi.c compile/link成執行檔
- check array sub job script

```
$ cd ~/1113/array_mpi_intel
$ module purge
$ module load intel/2018_u1
$ mpicc hello_mpi.c -o hello_mpi_intel.exe
$ vim hello_mpi_intel1.sh
mpirun  ./hello_mpi_intel.exe
$ vim hello_mpi_intel2.sh
mpirun  ./hello_mpi_intel.exe
$ vim hello_mpi_intel3.sh
mpirun  ./hello_mpi_intel.exe
```

Array job範例

■ 建立PBS job script array.sh

```
$ vim array_intel.sh
#!/bin/bash
#PBS -l walltime=00:01:00
#PBS -l select=3:ncpus=3:mpiprocs=3
#PBS -l place=scatter
#PBS -N intel-mpi-array-job-hello
#PBS -q ctest
#PBS -P ACD107023
#PBS -j oe
#PBS -J 1-3

module purge
module load intel/2018_u1
module list
cd $PBS_O_WORKDIR
echo "Main script: index $PBS_ARRAY_INDEX  “

./hello_mpi_intel$PBS_ARRAY_INDEX.sh
```

執行檔案變數1-3

執行檔路徑與檔名變數

Array job範例

- 執行job

```
$ qsub array_intel.sh
452844[].srvc1
```

- 顯示job (status = Finished)

```
$ qstat -x 452844[].srvc1
Job id              Name              User              Time Use S Queue
----------------  ----------------- ----------------  -------- - -----
452844[].srvc1     intel-mpi-array   yenwen988                0 F ctest
```

Array job範例

- 查看結果（intel-mpi-array-job-hello.o452844.1）

```
$ more intel-mpi-array-job-hello.o452844.1
Currently Loaded Modulefiles:
  1) intel/2018_u1
Main script: index  1
Hello world from processor cn0319, rank 3 out of 9 processors
Hello world from processor cn0319, rank 4 out of 9 processors
Hello world from processor cn0319, rank 5 out of 9 processors
Hello world from processor cn0320, rank 6 out of 9 processors
Hello world from processor cn0320, rank 7 out of 9 processors
Hello world from processor cn0320, rank 8 out of 9 processors
Hello world from processor cn0315, rank 0 out of 9 processors
Hello world from processor cn0315, rank 1 out of 9 processors
Hello world from processor cn0315, rank 2 out of 9 processors
```

Array job範例

- 查看結果（intel-mpi-array-job-hello.o452844.2）

```
$ more intel-mpi-array-job-hello.o452844.2
Currently Loaded Modulefiles:
  1) intel/2018_u1
Main script: index  2
Hello world from processor cn0333, rank 0 out of 9 processors
Hello world from processor cn0333, rank 1 out of 9 processors
Hello world from processor cn0337, rank 6 out of 9 processors
Hello world from processor cn0333, rank 2 out of 9 processors
Hello world from processor cn0337, rank 7 out of 9 processors
Hello world from processor cn0337, rank 8 out of 9 processors
Hello world from processor cn0335, rank 3 out of 9 processors
Hello world from processor cn0335, rank 4 out of 9 processors
Hello world from processor cn0335, rank 5 out of 9 processors
```

Array job範例

- 查看結果（intel-mpi-array-job-hello.o452844.3）

```
$ more intel-mpi-array-job-hello.o452844.3
Currently Loaded Modulefiles:
  1) intel/2018_u1
Main script: index  3
Hello world from processor cn0333, rank 0 out of 9 processors
Hello world from processor cn0337, rank 6 out of 9 processors
Hello world from processor cn0333, rank 1 out of 9 processors
Hello world from processor cn0337, rank 7 out of 9 processors
Hello world from processor cn0333, rank 2 out of 9 processors
Hello world from processor cn0337, rank 8 out of 9 processors
Hello world from processor cn0335, rank 3 out of 9 processors
Hello world from processor cn0335, rank 5 out of 9 processors
Hello world from processor cn0335, rank 4 out of 9 processors
```

Array job範例

- Submit 10 jobs with consecutive index numbers ， 執行檔名1~10

```
#!/bin/sh
#PBS -N Simn1010Jobs
#PBS -J 1-10
echo "Main script: index " $PBS_ARRAY_INDEX
/opt/AppA - input /home/user01/runcase1/scriptlet_$PBS_ARRAY_INDEX
```

- Submit 5 jobs with odd indices (1,3,5,7,9),執行檔名1,3,5,7,9

```
#!/bin/sh
#PBS -N SimOddJobs
#PBS -J 1-10:2
echo "Main script: index " $PBS_ARRAY_INDEX
/opt/AppA - input /home/user01/odd/scriptlet_$PBS_ARRAY_INDEX
```

- Submit 5 jobs with even indices (2,4,6,8,10), 執行檔名2,4,6,8,10

```
#!/bin/sh
#PBS -N SimEvenJobs
#PBS -J 2-10:2
echo "Main script: index " $PBS_ARRAY_INDEX
/opt/AppA -input /home/user01/even/scriptlet_$PBS_ARRAY_INDEX
```

Array job 範例

- **qstat -t** Shows state of and time used by job array and subjobs

```
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------- -------- - -----
0[].pbsworks      test             user01                  0 B workq
0[1].pbsworks     test             user01           00:00:00 R workq
0[2].pbsworks     test             user01                  0 Q workq
```

- **qstat -J** Shows state only

```
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------- -------- - -----
0[].pbsworks      test             user01                  0 B workq
```

- **qstat -p** Shows the % completed (completed / total)

```
Job id            Name             User              % done  S Queue
----------------  ---------------- ---------------- -------- - -----
0[].pbsworks      test             user01                 0  B workq
```

練習（Array job）

- 送出一個內含5個sub job的array job，用GCC OpenMPI的 mpirun來執行hello-world執行檔，每個sub job選3個nodes，每個node用3個ncpus去run 3個mpiprocs。(每個sub job總共9個mpiprocs分散在3個nodes;總共5個sub job)
- 請cd到之前copy過來的目錄array_mpi_gcc
- $ cd ~/1113/array_mpi_gcc
- $ module load openmpi/gcc/64/1.10.4
- $ mpicc hello_mpi.c -o hello_mpi_gcc.exe（產生執行檔）
- $ get_su_balance
- $ vim array_gcc.sh （修改project ID & host base resource & $PBS_ARRAY_INDEX）
  ./hello_mpi_gcc$PBS_ARRAY_INDEX.sh $PROCESSES $PBS_NODEFILE
- $ cat hello_mpi_gcc1.sh（查看sub job 內容）
  mpirun -np $1 -hostfile $2 --mca pml cm --mca mtl psm2 ./hello_mpi_gcc.exe
- $ cp hello_mpi_gcc1.sh hello_mpi_gcc4.sh（再多產生一份sub job4）
- $ cp hello_mpi_gcc1.sh hello_mpi_gcc5.sh（再多產生一份sub job5）
- $ qsub array_gcc.sh ;qstat -u $USER -J
- $ qstat -x jobid[]
- $ cat jobname.ojobid.index

PBS常見正常訊息

- Queue資源不足，job排隊等待中

```
$ qstat  -xsw 398061.srvc1

srvc1:
                                                                                  Req'd  Req'd   Elap
Job ID                          Username        Queue           Jobname         SessID   NDS  TSK   Memory Time  S Time
------------------------------- --------------- --------------- --------------- -------- ---- ----- ------ ----- - -----
398061.srvc1                    yenwen988       gtest           simple_cuda_mpi      --     2     8    --  00:10 Q  --
   Not Running: Insufficient amount of resource: Qlist
```

- job ID xxx has finished,請用 qstat -x job_id

```
$ qstat  398830.srvc1
qstat: 398830.srvc1 Job has finished, use -x or -H to obtain historical job information
```

PBS常見錯誤訊息

- 資源選項key錯（例如#PBS -l select=2:ncpus=2:mpiproc=4）

```
qsub: Unknown resource: mpiproc
```

- 資源選項錯誤（例如#PBS -l select=2:ncpus=41.. ）

```
qsub: Job violates queue and/or server resource limits
```

PBS常見錯誤訊息

- Queue name key錯(例如#PBS -q cf80)

```
qsub: Unknown queue
```

- 未指定project id (例如###PBS -P XXX)

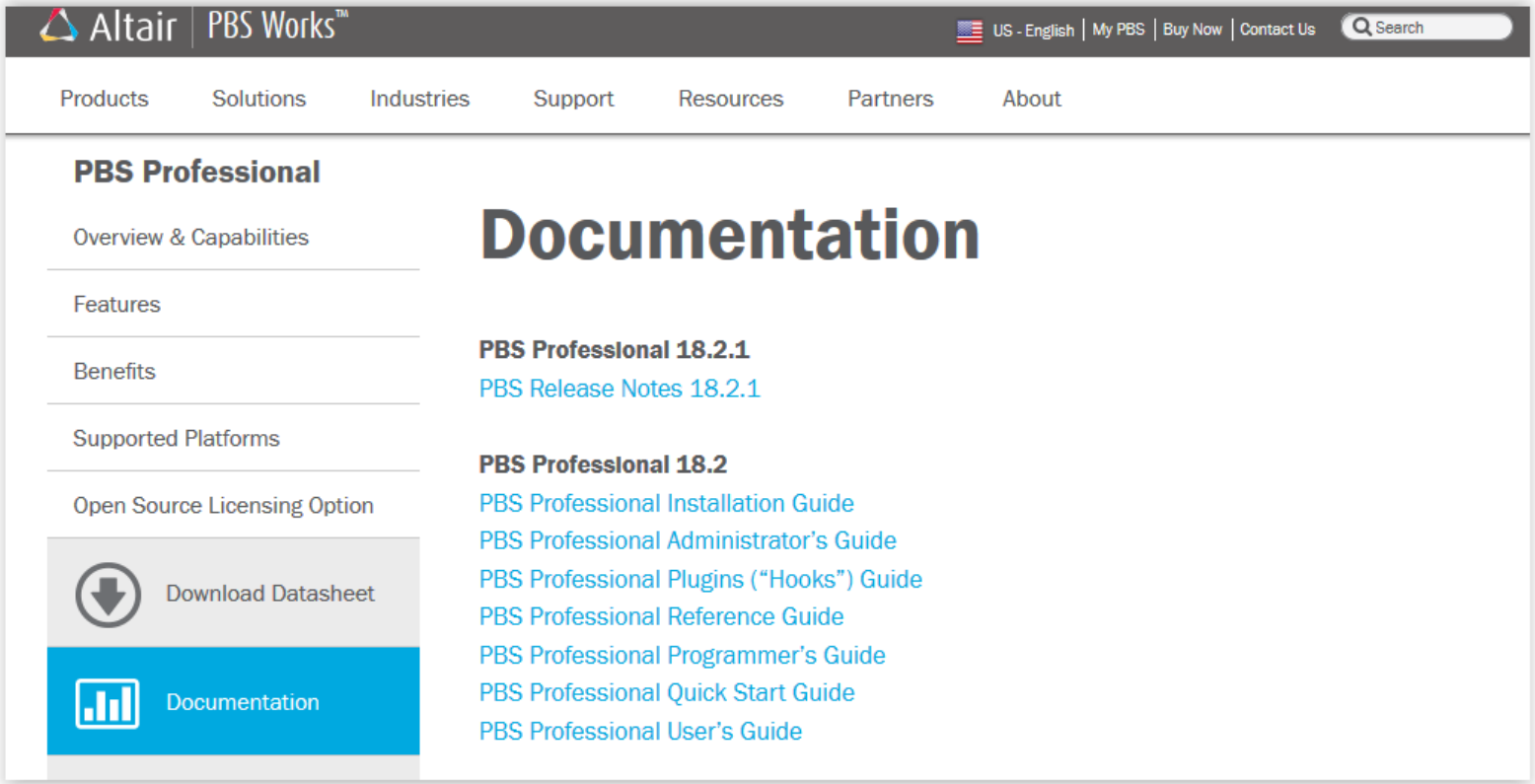
```
qsub: The job is rejected as Project has not been specified
```

- PBS job script MPI module沒有load (例如# module load intel/2018_u1)

```
$ more outputs/398440.srvc1.OU
/var/spool/pbs/mom_priv/jobs/398440.srvc1.SC: line 15: mpirun: command not found
```

PBS Professional文件參考資料鏈結

- https://www.pbsworks.com/PBSProductGT.aspx?n=PBS-Professional&c=Overview-and-Capabilities&d=PBS-Professional,-Documentation



FUJITSU

shaping tomorrow with you