Sunway TaihuLight

Quick Start Guide

National Supercomputing Center in Wuxi

1. System Introduction

“Sunway TaihuLight” Supercomputer System is a new generation super computer system, which is developed by National Research Center of Parallel Computer Engineering & Technology and under the supported of national 863 plan. The latest International TOP500 list showed that the system has a peak performance of 125.4 Pflop/s and a sustained performance of 93 Pflop/s, which are among the first in the world. It also has a performance per Watt of 60.5 billion, which can save energy more than 60% compared with other same order of magnitude. “Sunway TaihuLight” is the first super computer that has a peak performance of more than ten billion orders of magnitude in the world. It is also the first one that was all made of domestic processor in China.

“Sunway TaihuLight” was made of “Shenwei 26010” many-core processor which was supported by national major project. The processor was made by National High Performance Integrated Circuit Design Center. It used 64 bit Shenwei autonomous instruction system, 260 cores, and the core work frequency of 1.5 Hz and has a peak performance of 31,680 Flop/s. The performance index of “Shenwei 26010” is a world leading.

1.1. User Available Resources

1.1.1. Domestic High Speed Computing System:

- The number of Shenwei 26010 processors: 40960
- The number of Shenwei 26010 processor’s core groups: 163840 (basic unit fee)
- The peak performance of 125PFlops and is suited to science and engineering computing with source code

1.1.2. Commercial Auxiliary Computing System:

- 1000 common compute nodes，each has 24 cores and 128GB memory
- 32 fat nodes，each has 8 routers, 16 cores and 1TB memory
- The peak performance of 1PFlops, and is provided commercial software computing services such as industrial design.

1.1.3. Communication Network Bandwidth: Bi-directional 14GB/s

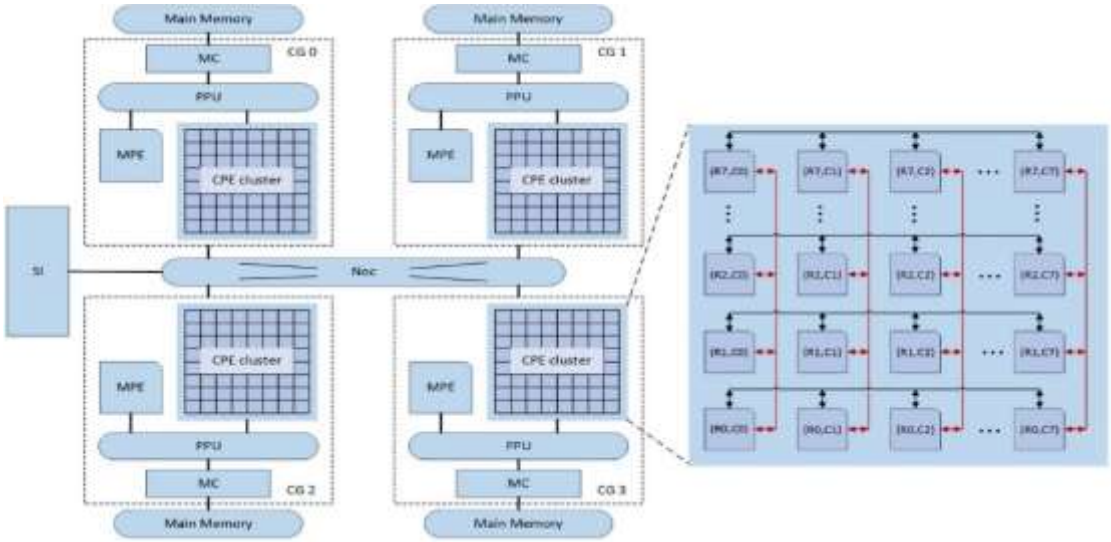
1.1.4. Disk Storage Space: Share 20PB

1.2. "Shenwei 26010" Many-Core Processor

Shenwei many-core processor uses the on-chip heterogeneous architecture and was composed of 4 heterogeneous groups (core group). Each core group contains 1 MPE core and 64 CPE cores. The whole chip contains 260 computing cores.

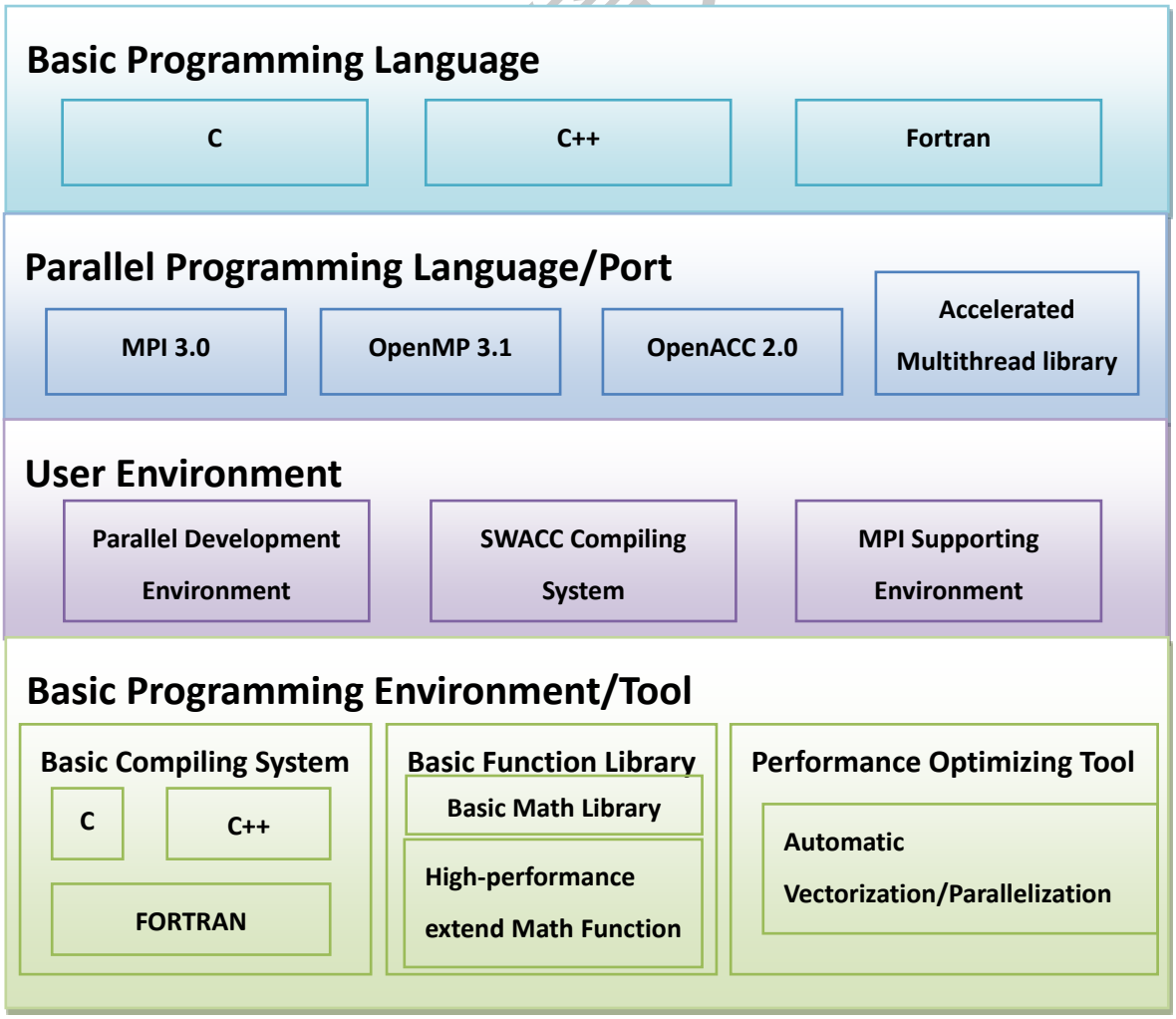
The dominant frequency of MPE core is 1.45GHz, the memory of each core is 8GB, L1cache is 32KB, L2cahe (hybrid data cache and instruction cache) is 256KB.

The dominant frequency of CPE core is 1.45GHz. It can directly discrete access to main memory through gld/gst. It can also lot-sizing access main memory through DMA. We can communicate between the CPE core arrays by register communication mode. The local storage space size is 64KB and instruction storage space is 16KB.



1.3. Language Environment

"Sunway TaihuLight" system language environment mainly includes four components, as shown in the figure below.



1) Basic Programming Language

Provide mainstream basic programming language, including C, support standard C99; C++, support standard C++03 and provide SWGCC compiling environment (CPE core doesn't support C++) which supports standard C++11 standard. Fortran, support the main functions in the standard Fortran2003 to satisfy the need for actual projects.

2) Parallel Programming Language/Port

Provide international parallel programming standard support, including MPI3.0, OpenMP3.1, Pthreads and OpenACC2.0. Support message parallel model, shared programming model, accelerating programming model, satisfying the diversified need for transplantation and development of scientific computing projects. Provide the self-design of accelerating thread library programming interface at the same time, satisfying the development needs for some projects which pursue ultimate performance.

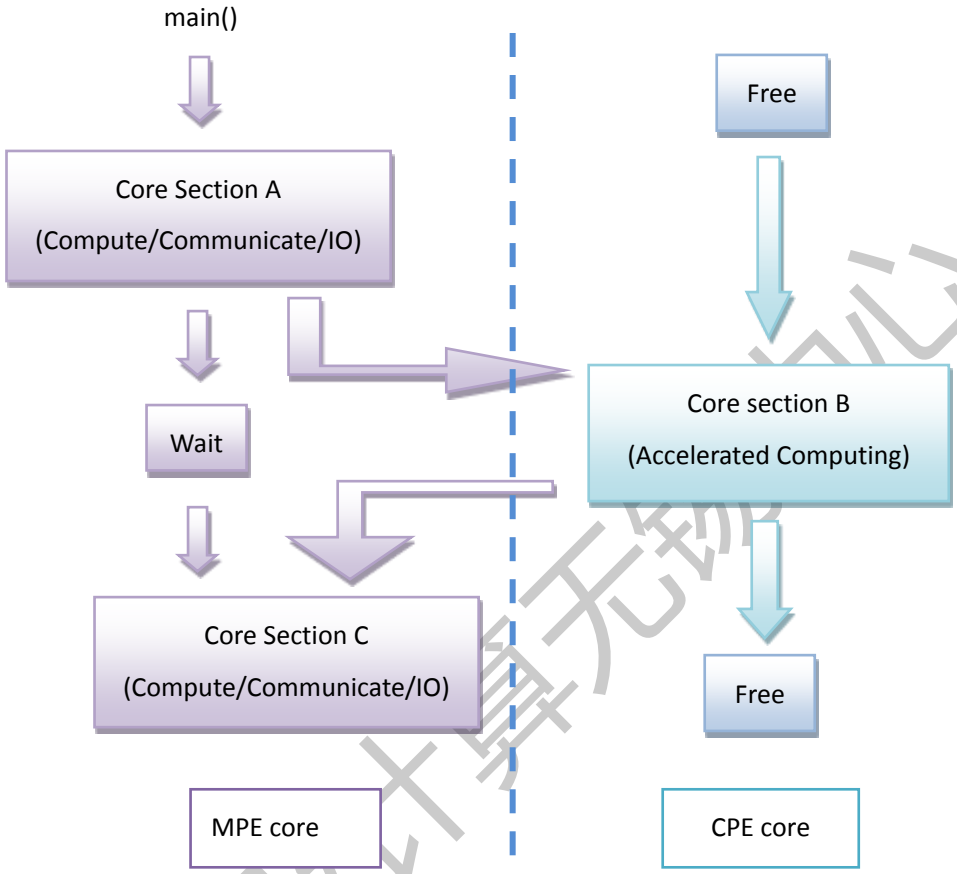
3) User Environment

Provide parallel development environment. Provide the using environment integrating editing, compiling, debugging, performance monitoring through a graphic user interface. Also support to set up basic developing environment by Sunway OpenACC compile system in character interface or MPI support environment to satisfy different user habits.

4) Basic Programming Environment

The basic programming environment is the foundation of all the superstratum and tools. Provide support such as basic language, main-CPE heterogeneous programming, basic function library, automatic vectorization/parallelization. Provide plenty efficient compiler optimizing method.

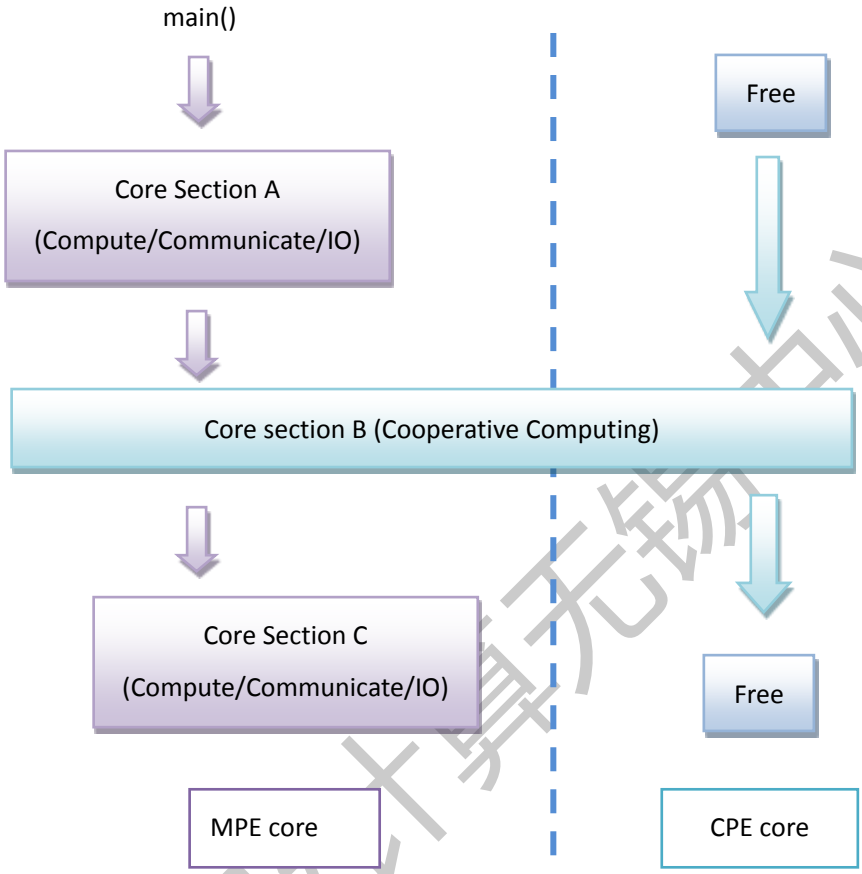
1.4. MPE-CPE Parallel Acceleration



Pic 2-1 MPE-CPE parallel acceleration schematic

Most projects use two-level parallel methods. When we use the many-core acceleration according to the calculation cores for the practical application, the MPE core mainly completes the computation and communication in the part that cannot use many-core parallel. The MPE cores wait when the CPE core is running task computation.

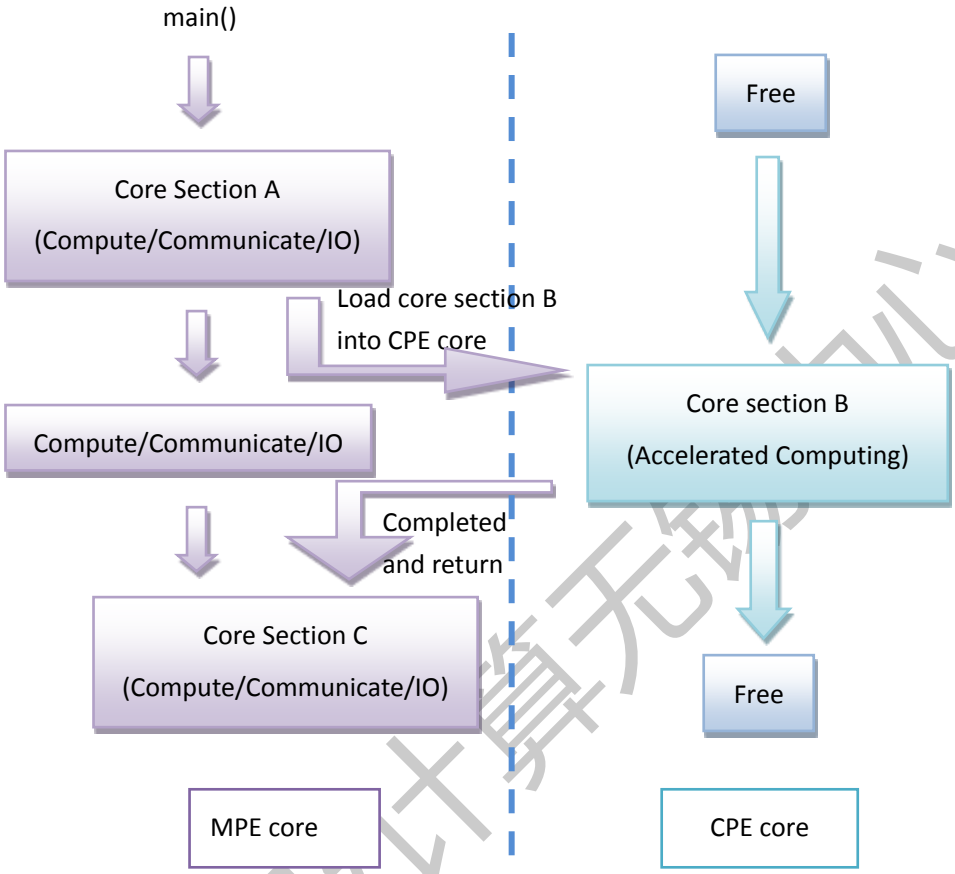
1.5. MPE-CPE Cooperative Parallel



Pic 2-2 MPE-CPE cooperative parallel schematic

The MPE cores and CPE cores run parallel computing as equal individuality, load allocation based on their respective computing, and calculate the core segments as a whole. If the project does not require high demand for LDM, the CPE cores can finish the computation of the whole core in the scope of LDM, reducing the cost of memory access and achieving good parallel effects.

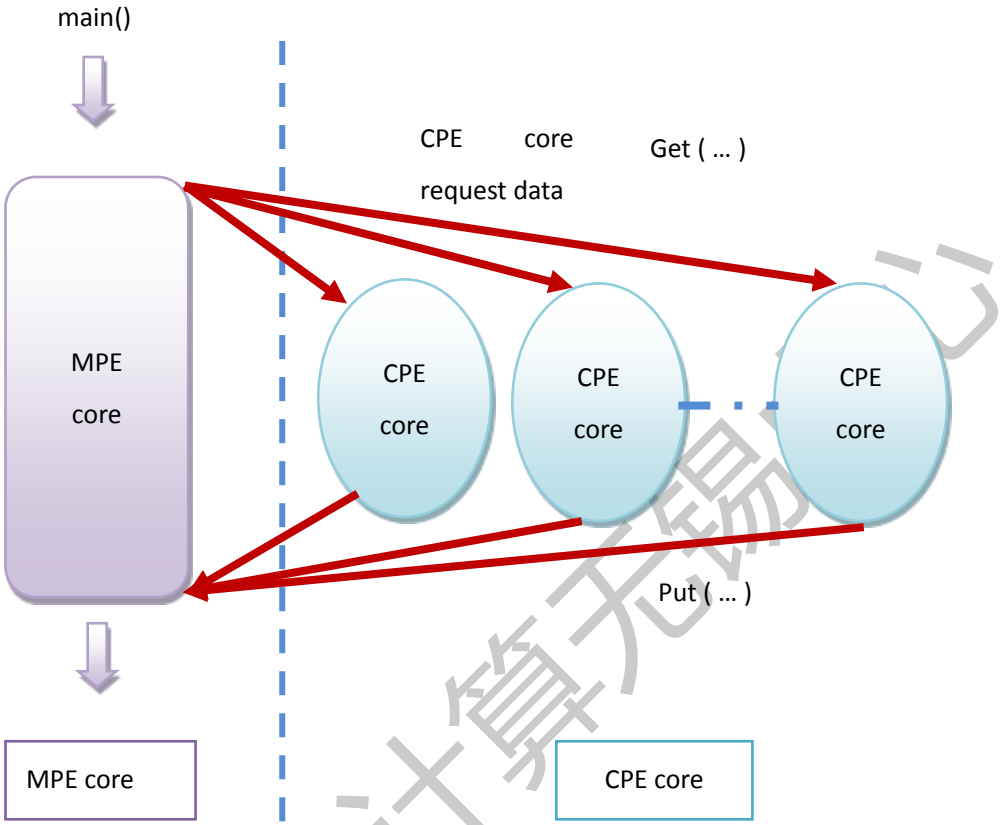
1.6. MPE-CPE Asynchronous Parallel



Pic 2-3 MPE-CPE asynchronous parallel schematic

At the same time when the CPE cores are executing the accelerating calculation, the MPE cores are executing other operations such as computation, communication, or I/O to improve the parallel efficiency of MPE-CPE asynchronous.

1.7. MPE-CPE Dynamic Parallel



Pic 2-4 MPE-CPE dynamic parallel accelerating schematic

The MPE cores are responsible for task allocation, while the CPE cores are responsible for obtaining new computing tasks, completing the calculation, and write back the calculation result. The method is suitable for the situation with an unfixed computing time for the computing tasks of the CPE cores. It can be considered to use two-level method with MPE-CPE parallel to run large scale parallel computing， The first level is the MPE-CPE parallel on process level. The second level is the MPE-CPE parallel in the core group of the heterogeneous many-core processor.

1.8. About Login

You need to use VPN to login, please see Appendix. Each user's username and password will be notified by e-mail. User shall change the initial password after first

login. The login account will become invalid after contest finish.

Tips: User could use remote login tool such as Xshell, SecureCRT and SSH Secure Shell Client to login.

1.4.1. Basic operation

User could log into the system by using Xshell5. The whole process will be described below using Xshell5 as an example.

1.4.2. Login

Login command: ssh 42.0.0.9. The system will prompt to input user name and password after the enter key is pressed, and then user shall be able to successfully log in to the system and enter the home directory at /home/export/base/asc2017/username.

User must pay attention to the login information, which may contain the system administrator's notice on system maintenance and update.

System will allocate quantitative space to each user by default. If the user needs a larger disk space, please send the application to the system administrator.

In order to ensure enough storage space and the data security and privacy, Please download or delete the relevant data before logging out. The platform is not responsible for long-term preservation of the user data.

1.9. Matters needing attention

Matters needing attention about the "Sunway TaihuLight" Platform:

1) It is suggested that the user shall immediately modify the password after log in. Password and account name shall be different. In the cases that the user forgets the password, please contact the maintenance personnel for resetting.

2) The user shall keep their own accounts and passwords, and do not disclose to others.

3) Make data backup timely to avoid data loss.

4) In order to ensure the reasonable usage of resource, the user shall log out after using. Besides, do not perform computation jobs that are irrelevant to the contest.

5) It is strictly prohibited to use scan, monitoring or camouflage kit to maliciously attack the system and to interfere with other users' normal jobs.

6) Please do not install or try unknown software, modify system configuration or run rogue program to avoid the system failure. In the cases that the user needs to use or install exterior file, please contact the maintenance personnel for approval.

7) For any good suggestions about system maintenance, please connect to system administrator.

Warning:

Behaviors such as deliberate damage of stored, processed or transmitted data and application programs in the system or intentionally making or transmitting destructive programs (computer virus) to affect the normal operation of system, are strictly prohibited.

2. Many-core Parallel Instances （Compile, Submit, Run）

2.1. Using Accelerated Programming Library

2.1.1. Fortran Instance

In this section, a calculation example is given that assign value to two arrays and solve the variance of the corresponding elements.

```
program main
implicit none
real ,dimension(imin:imax,jmin:jmax):: a，b，c
integer :: i,j,k          Load core section B
！init part               into CPE core
do j= jmin,jmax
do i= imin,imax
   a(i,j)=i+j-0.8888
   b(i,j)=i+j+7.7777
end do
end do          Record
                the result
！excute
do j= jmin,jmax
do i= imin,imax
   c(i,j)=a(i,j)*b(i,j)- b(i,j)*b(i,j)
end do
end do
end
```

The code was divided into two parts after many-core parallel, which are the code of MPE core and CPE core. The code of MPE core as follows:

```
program main
        implicit none
        integer:: i,j
        real,dimension(imin:imax,jmin:jmax)::a,b,c
        integer,external :: slave_fun              ! Specify the function name of CPE core
        common /shared_g1/ a,b,c                   ! The sared array
        do j= jmin,jmax
        do i= imin,imax
            a(i,j)=i+j-0.8888
            b(i,j)=i+j+7.7777
        end do
        end do
        call athread_init()                        ! Initialize CPE core
        call athread_spawn(slave_fun, 1)           ! The parallel task of CPE core start
        call athread_join()                        ! Wait for the end of CPE core task
        call athread_halt()                        ! Terminate CPE core
  end
```

The code of CPE core as follows:

```
subroutine fun
        implicit none
        real,dimension(imin:imax,jmin:jmax)::a,b,c
        common /shared_g1/ a,b,c                              ! The shared variable
        integer,dimension(imin:imax):: a_slave,b_slave,c_slave  ! Apply for the local variable
of CPE core
        integer i,j
        integer slavecore_id,reply
!$omp threadprivate (/local_g1/)                              ! The guide language of CPE core
        call get_myid(slavecore_id)                           ! Get logical id of CPE core
        do j= jmin,jmax
        if(mod(j,corenum)+1.eq.slavecore_id)then        ! Blinding the computation task of
CPE core
            reply=0
            call  athread_get(0,a(imin,j),a_slave(imin),  (imax-imin+1)*4,reply,0,0,0)! Read in
```

```
data
            call athread_get(0,b(imin,j),b_slave(imin), (imax-imin+1)*4,reply,0,0,0)
            do while (reply.ne.2)
            end do
            do i= imin,imax
               c_slave(i)=a_slave(i)*a_slave(i)+b_slave(i)*b_slave(i)
            end do
            reply=0
            call athread_put(0,c_slave(imin),c(imin,j), (imax-imin+1)*4,reply,0,0)！Write back
the data
            do while (reply.ne.1)
            end do
          end if
          end do
          return
end
```

Compile separately on MPE and CPE cores:

sw5f90 -slave -c slave.f90

sw5f90 -host -c master.f90

sw5f90 -O3 -hybrid *.o -o test

The code of CPE core was compiled by sw5f90 –slave, and the code of MPE core compiled by sw5f90 –host. Use sw5f90 –hybrid to link and generate the final executable file.

The command of submit job is as follow:

bsub -I -b -q queue_name -n 1 -cgsp 64 -share_size 4096 -host_stack 128./test

In the code above, "-I" option represents that submitting interactive job to make the job output at the submitting window.

"-b" represents that the stack variables of the CPE core function are placed in the local storage of the CPE cores, and this option is the mandatory submitting option to achieve accelerating performance.

"-q" represents submitting job to the assigned queue. "-n" represents the number of MPE core required.

"-cgsp" represents the number of CPE core required by each CG. When you input, this parameter must be <=64.

"-share_size" represents the shared memory space of the core groups, which commonly can be used up to 7600MB.

"-host_stack" represents the stack space of the MPE core, which is 8M by default， and commonly set to be more than 128MB.

As we can see from the code of representative many-core above, when using many-core parallel, we need to share the data relating with core computing through common. The code of MPE core initialize the accelerating thread library firstly, then transfer the interface of the CPE core program that needs to be executed, then wait for the end of CPE core computation，and finish the accelerating thread library in the end. The program of the CPE core mainly achieves the read-in, computation and write-back of the shared data.

For the MPE-CPE cores, the shared memory space is visible, which also means that the MPE and CPE cores can directly visit and modify the data from the shared memory space. When we write programs with Fortran，data sharing usually realize by common. So we must keep the consistency of the common name and array size. When we achieve the array sharing of the data through the way of common， there are requirements about the size of the array -- the size of the array must be fixed. For those arrays with dynamic sizes， they can directly use common to share data pointer，or let common point to the Cray pointer of this array. Three examples of common are given as follows.

…	…	…

n=10;m=10 real v(name) common/fun_v / v …	real,pointer v(:,:) common/fun_v / v …	real v_host(n,m) pointer(v_p,v_host) common/fun_v / v_p v_p=loc(v(1,1)) …

There are differences between the DMA communication performances of the three examples of common. The experiment result states that the DMA performance of array pointer is the worst, cray pointer is better and static array is the best.

2.1.2. C Instance

The code of MPE core as follows:

```
#include <stdlib.h>
#include <stdio.h>
#include <athread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

extern SLAVE_FUN(func)();
static inline unsigned long rpcc()
{
        unsigned long time;
        asm("rtc %0": "=r" (time) : );
        return time;
}

#define J 64
#define I 1000
double a[J][I],b[J][I],c[J][I],cc[J][I];
double check[J];
unsigned long counter[J];
int main(void)
```

```
{
        int i,j;
        double checksum;
        double checksum2;
        unsigned long st,ed;

printf("!!!!!!!!!! BEGIN INIT !!!!!!!!!!\n");fflush(NULL);
        for(j=0;j<J;j++)
        for(i=0;i<I;i++){
                a[j][i]=(i+j+0.5);
                b[j][i]=(i+j+1.0);
        }
        st=rpcc();
        for(j=0;j<J;j++)
        for(i=0;i<I;i++){
                cc[j][i]=(a[j][i])/(b[j][i]);
        }
        ed=rpcc();
        printf("the host counter=%ld\n",ed-st);
        checksum=0.0;
        checksum2=0.0;
        athread_init();

        st=rpcc();
        athread_spawn(func,0);//fflush(NULL);
        athread_join();
        ed=rpcc();
        printf("the manycore counter=%ld\n",ed-st);
printf("!!!!!!!!!! END JOIN !!!!!!!!!\n");fflush(NULL);
        for(j=0;j<J;j++)
        for(i=0;i<I;i++){
                checksum=checksum+c[j][i];
                checksum2=checksum2+cc[j][i];
        }
        printf("the master value is %f!\n",checksum2);
```

```
        printf("the manycore value is %f!\n",checksum);
        athread_halt();
printf("!!!!!!!!!! END HALT !!!!!!!!!\n");fflush(NULL);
    }
```

The code of CPE core as follows:

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "slave.h"

#define J 64
#define I 1000

__thread_local volatile unsigned long get_reply,put_reply;
__thread_local volatile unsigned long start,end;
__thread_local int my_id;
__thread_local double a_slave[I],b_slave[I],c_slave[I];
extern double a[J][I],b[J][I],c[J][I];
extern unsigned long counter[64];

void func()
{
        int i,j;
my_id = athread_get_id(-1);
        get_reply = 0;
        athread_get(PE_MODE,&a[my_id][0],&a_slave[0],I*8,&get_reply,0,0,0);
        athread_get(PE_MODE,&b[my_id][0],&b_slave[0],I*8,&get_reply,0,0,0);
        while(get_reply!=2);

        for(i=0;i<I;i++){
                c_slave[i]=a_slave[i]/b_slave[i];
        }
        put_reply=0;
```

```
        athread_put(PE_MODE,&c_slave[0],&c[my_id][0],I*8,&put_reply,0,0);
        while(put_reply!=1);
}
```

The process of compile and link:

```
sw5cc -host -c master.c
sw5cc -slave -c slave.c
sw5cc -hybrid master.o slave.o -o test
```

Submit options are the same as the instance of Fortran，and need not be repeated here.

2.2. Using OpenACC

Here is how to use OpenACC to port the program above to the many-core platform.

```
program main
        implicit none
        real ,dimension(1:512,1:64):: a, b, c
        integer :: i,j
        !init part
        do j= 1,64
        do i= 1,512
            a(i,j)=i+j-0.8888
          b(i,j)=i+j+7.7777
        end do
        end do
        !excute
        !$ACC PARALLEL LOOP COPYIN(a, b) COPYOUT(c) LOCAL(i)
        do j= 1,64
        do i= 1,512
            c(i,j)=a(i,j)*b(i,j)- b(i,j)*b(i,j)
        do
```

```
        end do
        !$ACC END PARALLEL LOOP
end program
```

Mark by OpenACC accelerating compile instruction before and after the loop in the execution part that needs to use many-core accelerating execution. As the bold part in the code shows, the PARALLEL shows that this part of code is the parallel code that needs accelerating execution. LOOP shows that the j loop closely below needs to parallel partitioning between accelerating threads. COPYIN(a,b) shows that array a, b need to be copied to LDM, because array a, b is read-only, we do not need to update it back to the MPE memory. COPYOUT(c) shows that array c needs to be copied back to the MPE memory after the end of computation. LOCAL(i) shows that variable i is private to the accelerating thread and put into LDM.

When we use swafort to compile, the running method is similar to that in 3.1. The running result of this program is that if we execute with 64 accelerating threads. Every thread will execute one of the 64 j loops, and respectively will apply for memory space in LDM for the data corresponding array a, b, c, then copy and computing this data.

The initialization part of the code can be ported with this method as well.

3. Compiling Environment

3.1. Domestic High Speed Computing system

3.1.1. Basal Compilers

Compiling commands:

Language/ Target core	Computing Control Core	Computing Core	Hybrid Link
C	sw5cc -host	sw5cc -slave	sw5cc -hybrid
C++	sw5CC -host	Not supported	sw5CC -hybrid
Fortran	sw5f90 -host	sw5f90 -slave	sw5f90 -hybrid

Common Compiling Options：

Option	Function
-c	Generate an intermediate target file for each source file, but do not link.
-g	Generate debug symbol information for subsequent debugging.
-I<dir>	Add search <path> for the preprocessing header file.
-l<library>	Specify the library files that need to link in the link stage.
-L<dir>	Add search <path> for the link stage.
-lm	Use libm math library in the link stage. Required to call this library if we use functions such as exp, log, sin, cos, etc. in C.
-o <filename>	Specify the name of the generated executable (library) files.
-O1~-O3	Generate the executable code of the advanced optimization.
-pg	To generate feedback information for the gprof analysis program.
-msimd	Open the function module of simd.

C compile and link process:

```
sw5cc -host -c master.c
sw5cc -slave -c slave.c
sw5cc -hybrid master.o slave.o -o test
```

The compile link process for Fortran is similar, and need not be repeated here.

3.1.2. OpenACC Compiler

FORTRAN compiler: **swafort[options] filename**

FORTRAN pragma format: add "!$acc pragma name [clause[[,]clause]...]"

C compiler：**swacc[options] filename**

C pragma format：add "#pragma acc pragma [clause[[,] clause]...]"

Compile option：

Compile option	Instruction
-|--(h|help)	Display help
-SCFlags	Assigned option will be transmitted to the serial compiler of the device program. We need to use comma to separate if we are transmitting multiple options at the same time. No space is allowed in the middle. For example: swafort -SCFlags -extend_source,-O3 hello.c
-HCFlags	Assigned option will be transmitted to the basic compiler of the host program. We need to use comma to separate if we are transmitting multiple options at the same time. No space is allowed in the middle.
-LFlags	Assigned option will be transmitted to the linker. We need to use comma to separate if we are transmitting multiple options at the same time. No space is allowed in the middle.
-priv	**Auxiliary option:** conduct privatized variable analysis in the accelerating zone, give the read-write attribute of the variable, and finish analysis if encountering a unidentified function.
-priv:ignore_call	**Auxiliary option:** conduct privatized variable analysis in the accelerating zone, give the read-write attribute of the variable, ignore the impact of possible function call in the acceleration zone.
-arrayAnalyse	**Auxiliary option:** conduct array access mode analysis. It can give advices for the arrays that can be copied.

-ldmAnalyse	**Auxiliary option:** conduct device memory usage analysis. After the compiled, it will provide feedback of the memory usage and relevant suggestions in the computation zones when running.
-preinline	**Auxiliary option:** conduct inline to the functions in function call statement which contains #inline instructions. Do not deal with other accelerating instructions. Mainly used for the inline of the auxiliary functions.
-preinline:all	**Auxiliary option:** On the basis of –preinline, conduct inline to all functions of same name in a single function.
-dmaReuse	Mained at the "data copy" instruction，automatically conduct data reuse optimization.
-autoSwap	Automatically conduct transposition and optimization for appropriate arrays.
-v|-version	Display the version number of the compiler and library when running.
-Minfo	Display the compiler's analysis information of programs.
-keep	Keep the intermediate file when compiling.
-dumpcommand [dumpfile]	Export the compile commands for the intermediate files to dumpfile.

3.1.3. MPI

FORTRAN compiler：**mpif90[options] filename**

C compiler：**mpicc[options] filename**

Common compile option: support compile options such as -O2, -O3, -convert big_endian, -ftz.

Add "-OPT:IEEE_arith=2" to support the computing standard of IEEE754 float-point when compiling.

3.2. Commercial Auxiliary Computing System

3.2.1. Intel Basal Compiler

FORTRAN compiler：**ifort [options] filename**

C compiler： **icc[options] filename**

Compile option：

Compile option	Instruction
-O1	Maximized optimization speed, but it will turn off some options which increase file size but provide little speed improvement.
-O2	Maximized optimization speed (default).
-O3	Maximized optimization speed, and turn on more radical options that may not improve performance of all the programs.
-O	Same as -O2.
-Os	Turn on optimization option，but it will turn off some options that increases file size but provides little speed improvement.
-O0	Turn off optimization option.
-fast	Same as turn on -xHOST -O3 -ipo -no-prec-div –static.
-Ofast	Same as turn on -O3 -no-prec-div optimizations.
-fno-alias	Do not use code obfuscation(to protect code，prevent decompilation).
-fno-fnalias	Do not use internal function obfuscation. Only obfuscate the external call.
-nolib-inline	Disable the internal expansion of the Intrinsic functions.
-ftz	Set abnormal values to 0.

3.2.2. Intel Parallel Compiler

FORTRAN compiler：**mpif90[options] filename**

C compiler：**mpicc[options] filename**

The compiling options reference the one at 4.2.1 Intel Basal Compiler.

4. Job Management System Quick Tour

4.1. Job Submission（bsub）

Function	**Submit job to system**
Command Format	Usage: bsub [-h][-v] bsub [-f sub_script] bsub [-I] [-p] //list job's nodename and spe_map [-q queue_name]

	[-n num_procs [-master]] [-np node_mpes] [-mpecg mpe_cgs] [-cgsp spe_in_cg] [-J jobname] [-jobtype job_type] // available type: COMM / I/O / COMP / MEM / MIX [-mpmd] [-node nodelist] [-o outfile] [-b] [-parse] [-PARSE <all | master | slave>] [-quick] [-share_size size] [-host_stack size] [command [argument...]]
Parameter Specification	-h Display help instruction. -I Submit interactive job, let the job submit to the job submission window. The job will be batch if there is not such option. -q Submit job to the specified queue. Must be selected. -p Print the node list and bitmaps that the job allocates in the job output. -n Specify the number of all the MPE core in need. -np Specify the number of MPE cores that each node uses -cgsp Specify the number of CPE cores that each CG needs, the parameter must be <=64. -node Specifies the node to run the job (CG list). -o Direct the stdout and stderr output of the job specified file, options. -b Allocate the CPE core's stack to the cache. -share_size Specify the size of the shared space for the core group. -host_stack Specify the size of the stack space for the MPE core, 8M by default.
Instance	Submit interactive job "myjob" to queue. This job uses 1 node and 4 MPE cores:

	bsub –I –q queue –n 1 –np 4 ./myjob A line of prompt message containing jobid will be displayed after the successful submission of the job. The message contains the job id, such as "Job <102> is submitted to queue <queue>". In this instance, jobid is 102, and it is globally unique. Once the work is submitted successfully, various operations of the user can be achieved through this jobid.
Consideration	1） -I and –o usually are not recommended to be used together, because using –o will not display the printing of the program on the screen output. 2） With each success submission of the job, there will always has a jobid. It is the only feature that this job can be distinguished from other jobs. The system will ensure the consistency of the jobid. In the life cycle of the job, any operation on the job needs jobid as parameter. 3） All parameters of this command are required before the user program. 4） User uses on SN.

4.2. Job Terminated（bkill）

Function	Terminate the work
Command Format	bkill [-h] [-J jobName] [-q queue] [jobId]
Parameter Specification	-h Help information. -q Operate the job in the specified queue. -J Operate the job with the specified job name. jobId Represent the job id identifier. Instruction: When jobId is specified, ignore –q option.
Instance	bkill 1234 Terminate the job with id 1234 in the queue. bkill –q q1 Terminate all the job of the current user in the queue q1.
Consideration	1）To the normal users, even though using –q parameter, they can only operate the job of himself in the specified queue. **2）Use on SN.**

4.3. Job Status Queries（bjobs）

Function	Query the status information of the job in the queue
Command Format	bjobs [-h] [-l] [-w] [-d | -e | -p | -r] [-q queue_name] [jobId]
Parameter Specification	-q Specify queue to query. -l Display job details by long format. -w Display with fill length. When the length of the column value exceeds the column width, do not intercept according to the column. -d Displays the recently successfully completed jobs. -e Displays the recently aborted jobs. -p Display the job with pend status. -r Display the job running at the moment. jobid: Job id.
Instance	bjobs 1234 Query job running status information with id 1234.
Consideration	1）Normal users can only view job information belong to the current user. 2）To the job in pend status, if the job stays at pend status for a long time, we can use bjobs –l jobid to view the reason why the job could not be scheduled to run. 3）**Use on SN.**

4.4. Job Submission Instance

Domestic platform：

Foreground submit：bsub –I -b -m 1 -p -q q_sw_expr -host_stack 1024 -share_size 6500 -n 1024 -np 4 -cgsp 1 ./wrf.exe

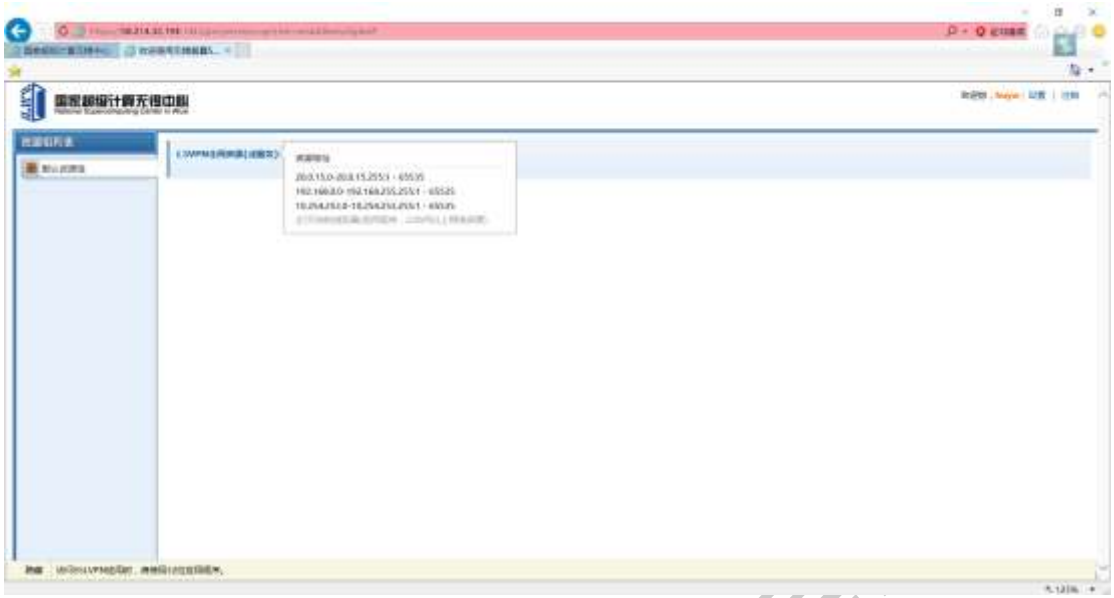
Background submit：bsub -o run.log -b -m 1 -p -q q_sw_expr -host_stack 1024 -share_size 6500 -n 1024 -np 4 -cgsp 1 ./wrf.exe

Appendix: Windows connect VPN

1. Open the Internet Explorer: http://www.nsccwx.cn/, select the network at the top right:

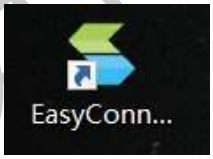
2. Follow the prompts to download the client control:

3. Gain the resource address and connect VPN successfully:

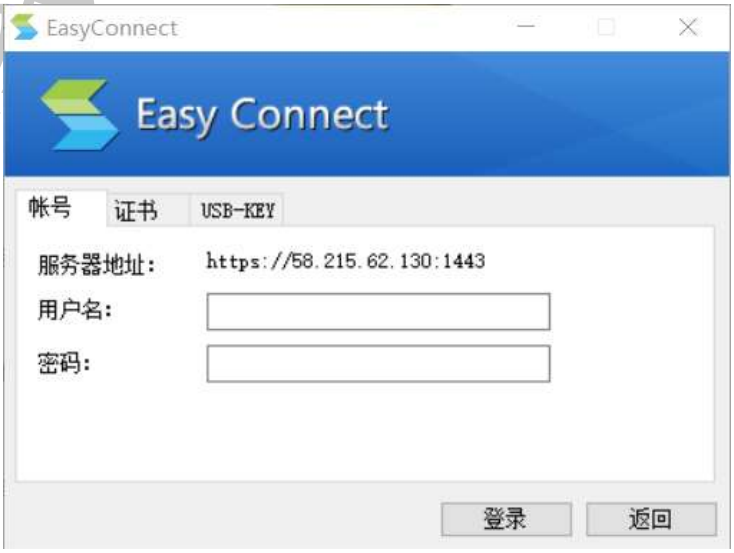


4. Run the installation package to install:

5. When you first connect VPN successfully, you can use the EasyConnect to connect next time:



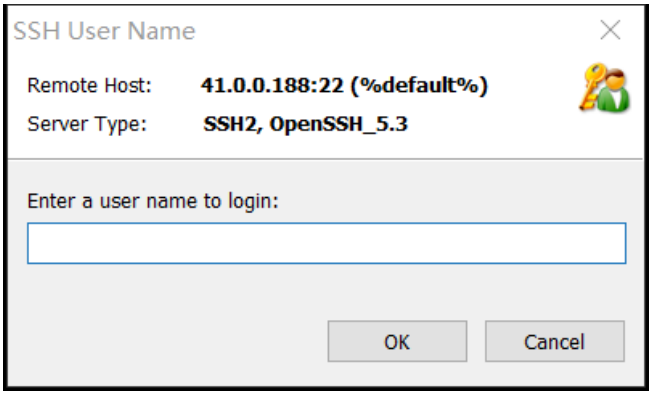
6. Enter the VPN account password:



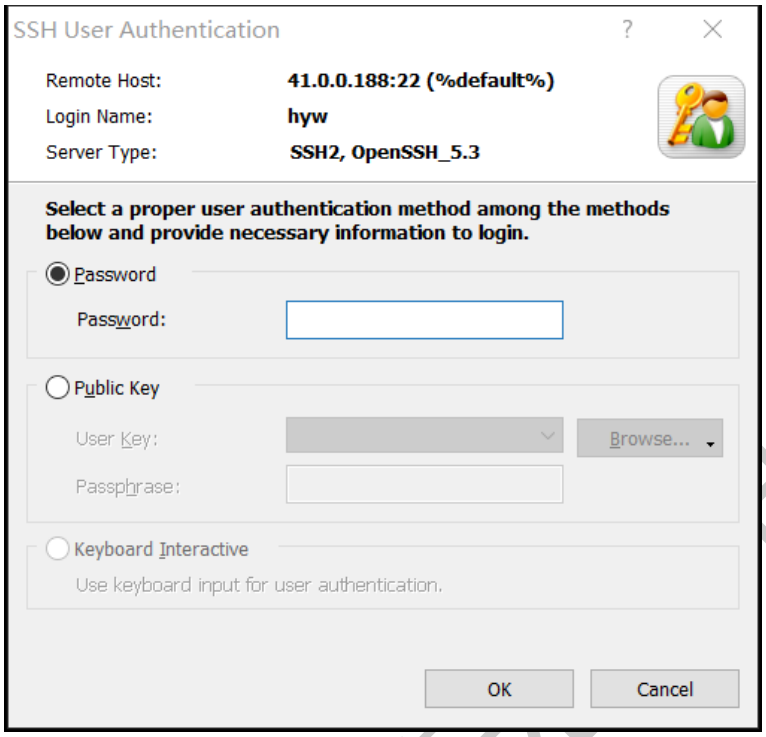
7. Prompt login successfully:

8. Open a terminal and connect to 41.0.0.188, click "Accept and Save":

9. Enter the user name:



10. Enter the password:



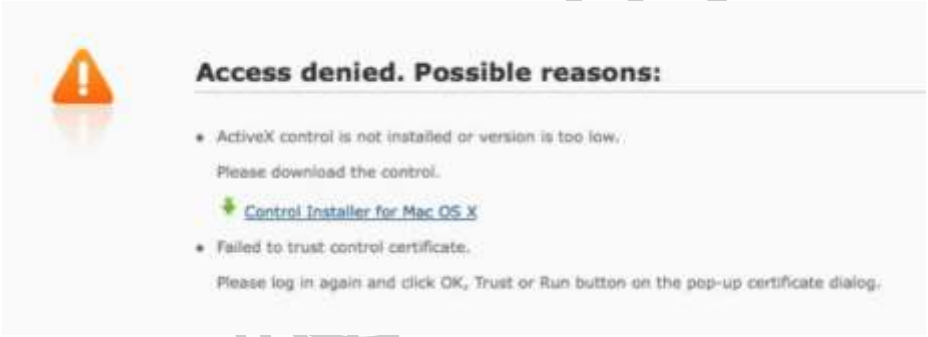
Mac OS connect VPN

1. Open the Safari: http://www.nsccwx.cn/, select the network at the top right:

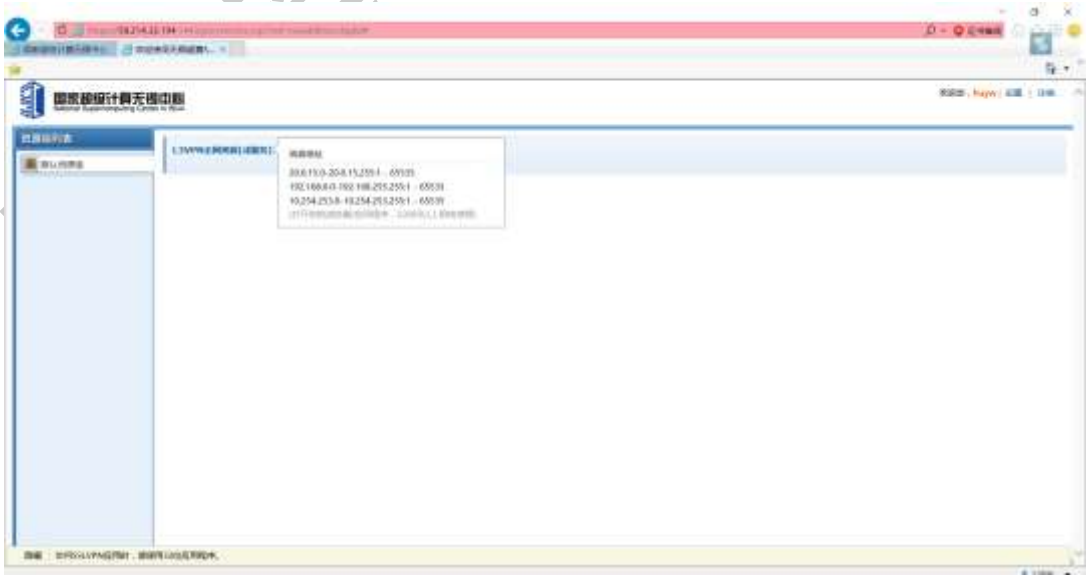
2. When Safari prompt "Safari can't verify the identity of the website…", choose "continue":

3. Enter the username and password:

4. Follow the prompts to download the client control:



5. Gain the resource address and connect VPN successfully:



6. Open MAC terminal. Use the command: ssh, connect to 41.0.0.188, enter your username and password.