CPC2021赛前培训第一期：

“神威·太湖之光”系统使用与并行编程

2021年6月23日（周三）20:00-21:00

适合第一次参赛、没接触过申威处理器并行编程的同学听讲

CPC 2021 报名正在进行中，微信扫码加好友咨询报名

CPC2021赛前培训第二期：

“神威·太湖之光”性能分析、优化与调试技术

2021年6月30日（周三）20:00-21:00

适合了解申威处理器并行编程的同学听讲

CPC 2021初赛赛题已发布，微信扫码查看赛题

国家超级计算无锡中心
National Supercomputing Center in Wuxi

“神威·太湖之光”系统使用与并行编程

张 炜

目录

“神威·太湖之光”超级计算机

- 全国产众核芯片（片上融合异构）
- 连续四次世界第一（2016-2017）
- 每秒12.5亿亿次计算（125 Pflops）
- 超过一千万核（260核*40960 CPU）

国产“神威•太湖之光”超级计算机

163,840 processes　　65 threads

racks　　*chips*　　*core-groups*　　*cores*　　*total number of cores*

$$40 \times 1{,}024 \times 4 \times 65 = 10{,}649{,}600$$

260-core Chip

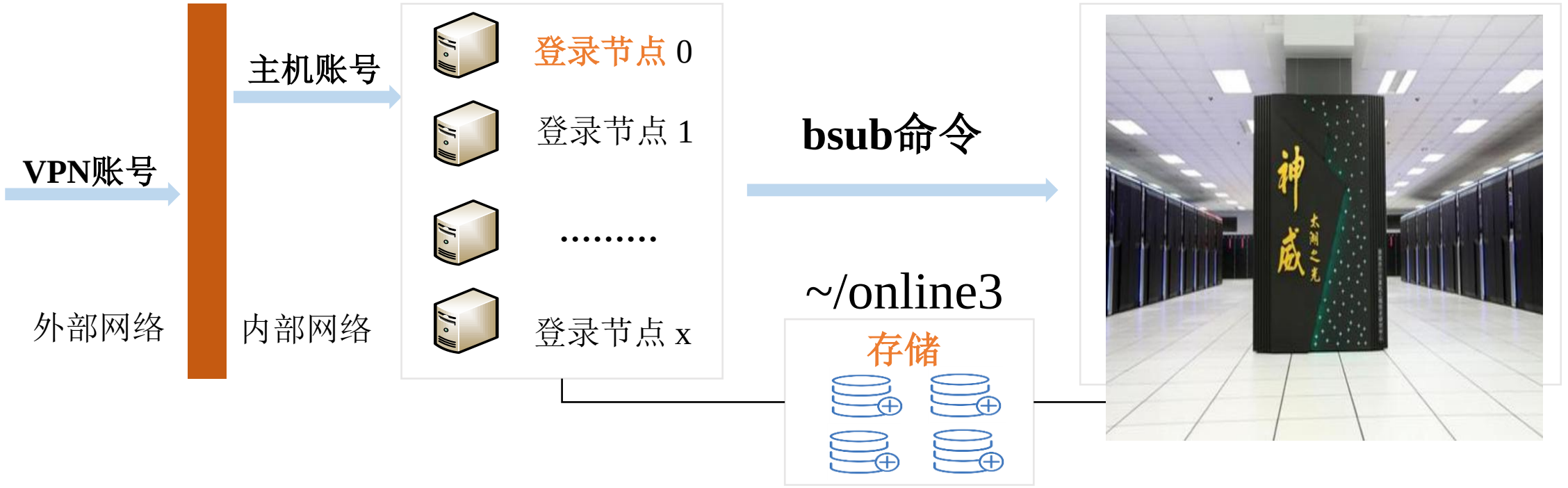
1,024x

Rack

40 x

System

太湖之光使用简单——两步登录、一条任务提交命令



- 程序要通过bsub命令运行，且要在online3目录下执行！
- 程序和数据都要放在online3目录下！

太湖之光使用简单——bsub任务提交命令

bsub -I -q q_sw_expr -b -n 1 -cgsp 64 -host_stack 1024 -share_size 6500 ./a.out

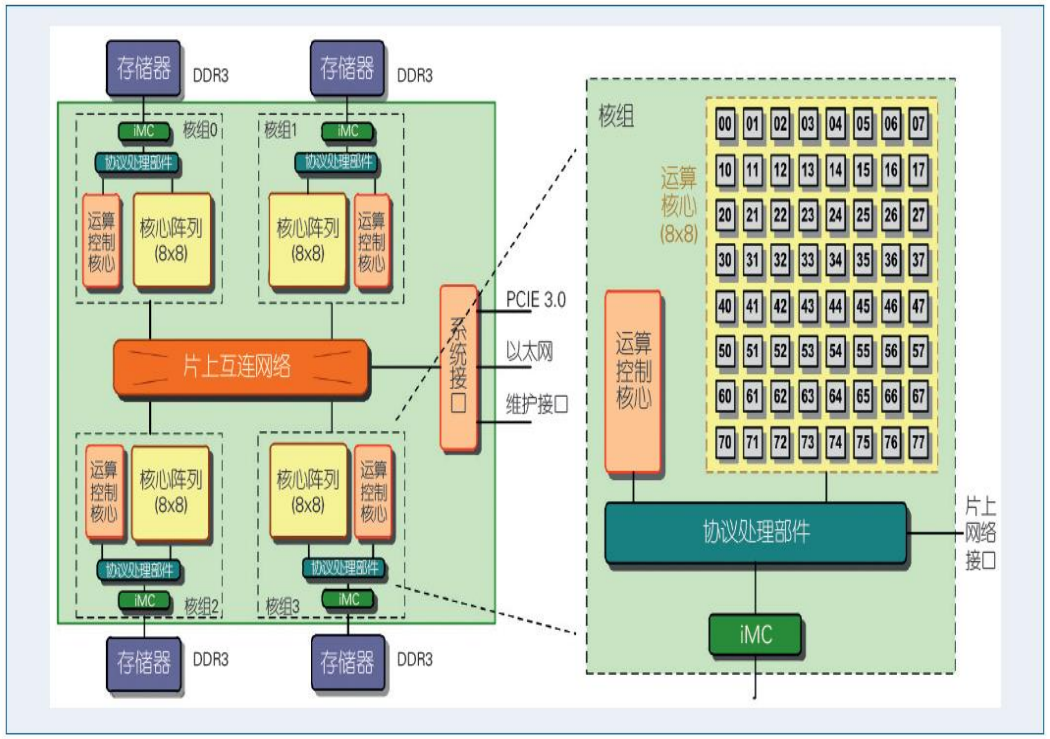
bsub命令

程序自身执行命令

- 参数1：**-I**，程序输出会打印到终端，终端关闭则程序终止运行，去掉该选项加上“**-o out.log**”输出会重定向到out.log文件，终端关闭不影响程序运行
- 参数2：**-q**，程序提交到 q_sw_expr 计算节点队列运行
- 参数3：**-n**，程序运行使用的核组数，最好是4的倍数，因为一个节点有4个核组
- 参数4：**-b**，从核函数栈变量放在从核局部存储上，为获取加速性能 必须的提交选项
- 参数5：**-cgsp**，使用从核的个数，一般设置为64
- 参数6：**-host_stack**，指定主核栈空间大小，默认为8M，一般设置为128MB以上
- 参数7：**-share_size**，指定核组共享空间大小，一般 share_size + host_stack 小于7500

太湖之光并行编程值得一试

- 独特的体系结构
- 自主研发的软件生态系统



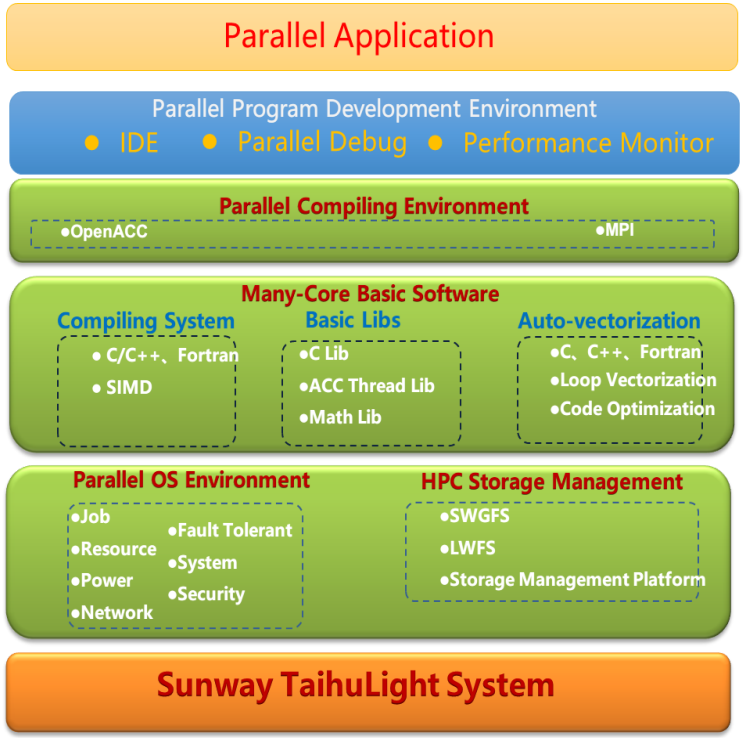
国产众核CPU基础软件

并行开发环境

并行语言及编译环境

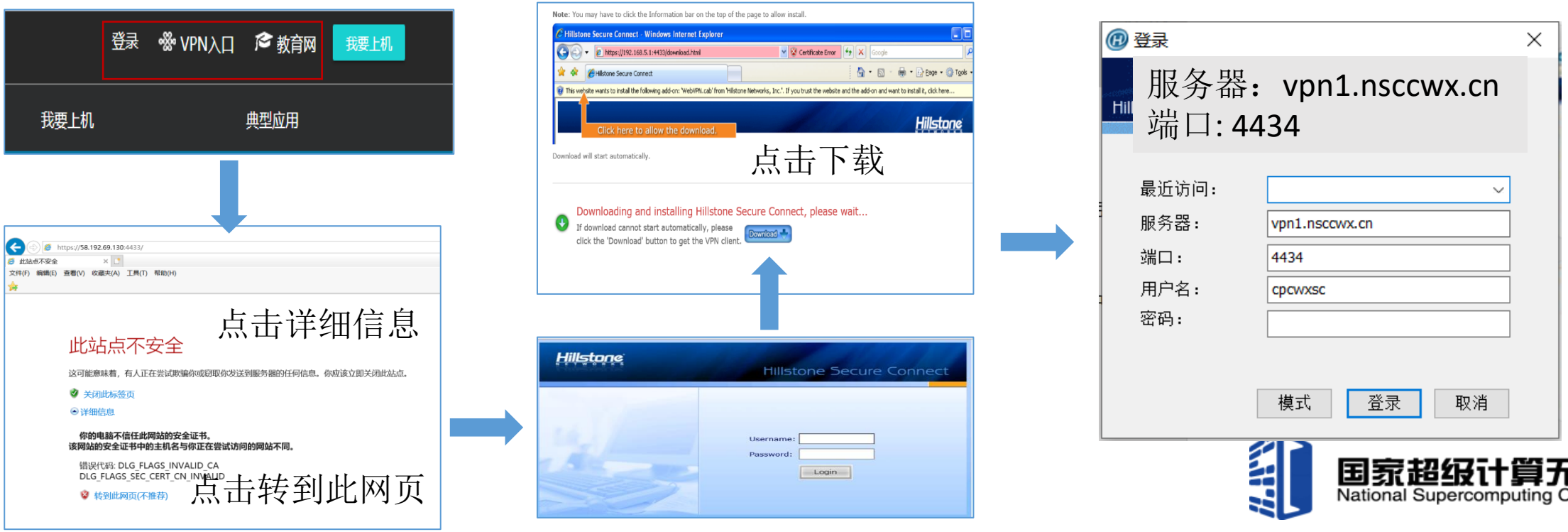
高性能存储系统

并行操作系统环境(神威睿思)



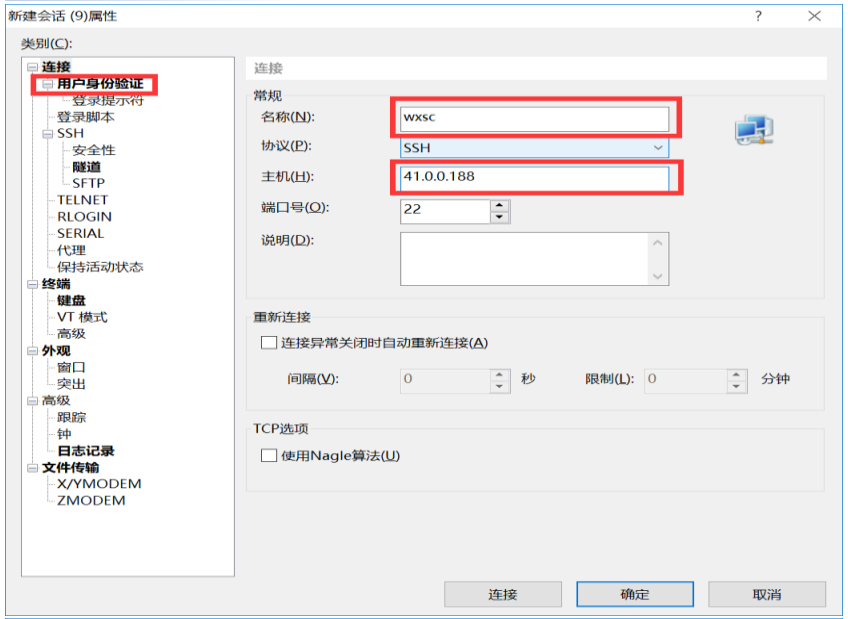
超算登录——VPN 登录

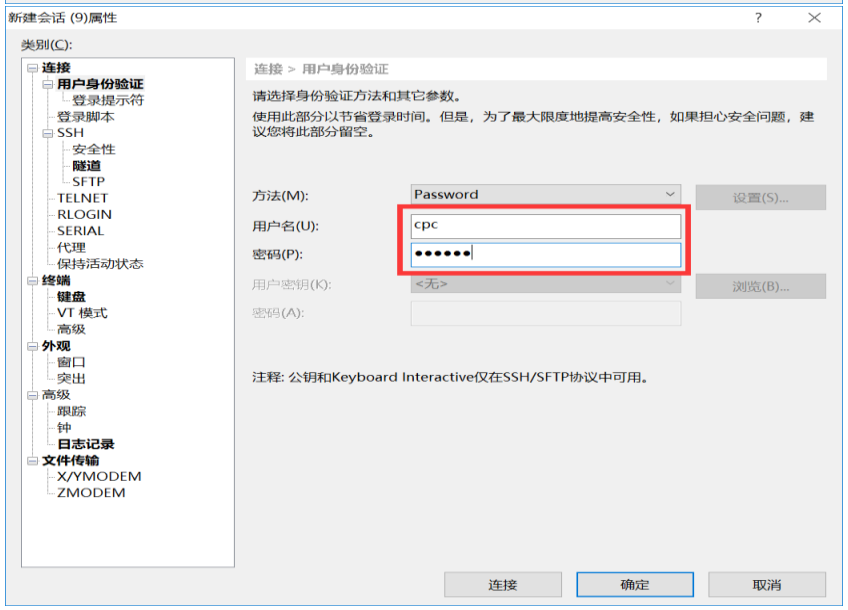
- 首次登录：使用 IE 浏览器访问 http://www.nsccwx.cn/
- 点击右上角“教育网”（非教育网也是这个入口），然后输入VPN账号和密码，进行VPN客户端下载与安装，中间会弹出一些询问或者警告信息，全部点击确定或者信任
- VPN客户端安装成功后，下次使用客户端连接VPN，不用再从官网登录



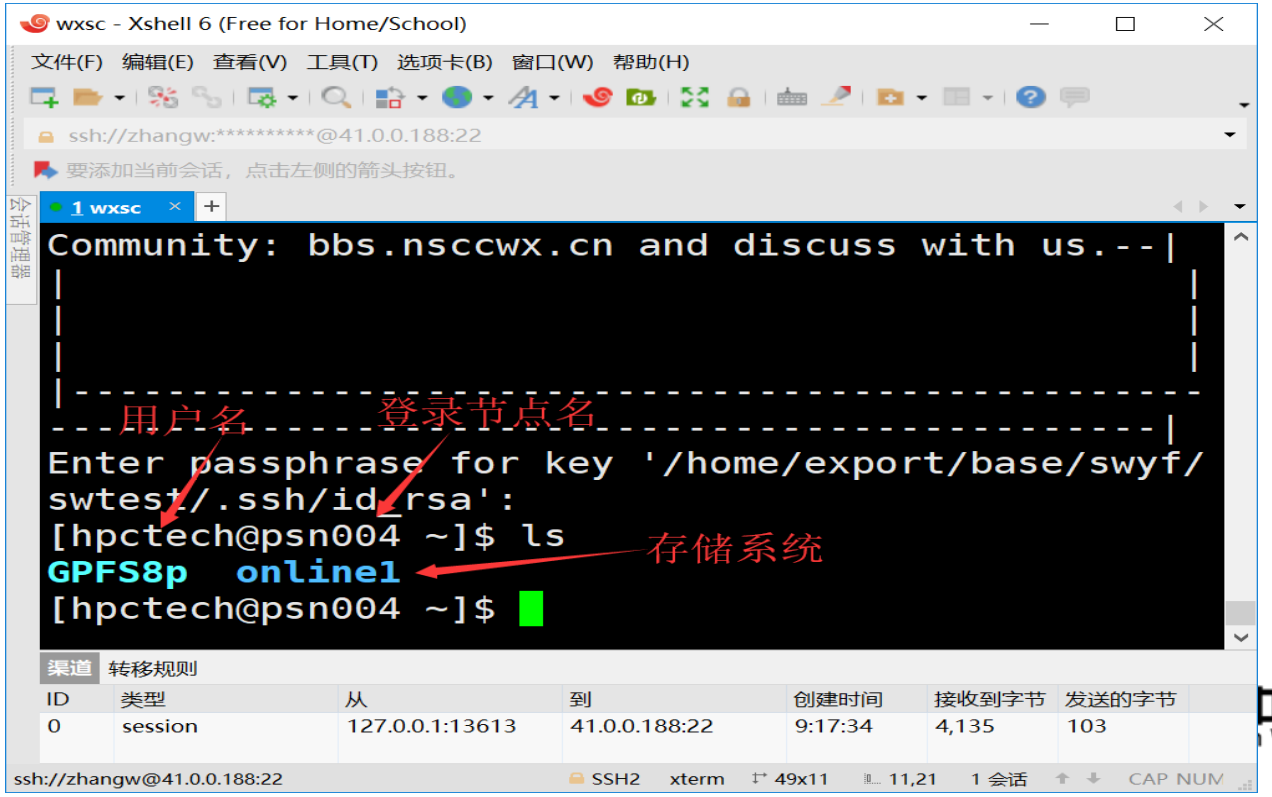
国家超级计算无锡中心
National Supercomputing Center in Wuxi

超算登录——集群登录



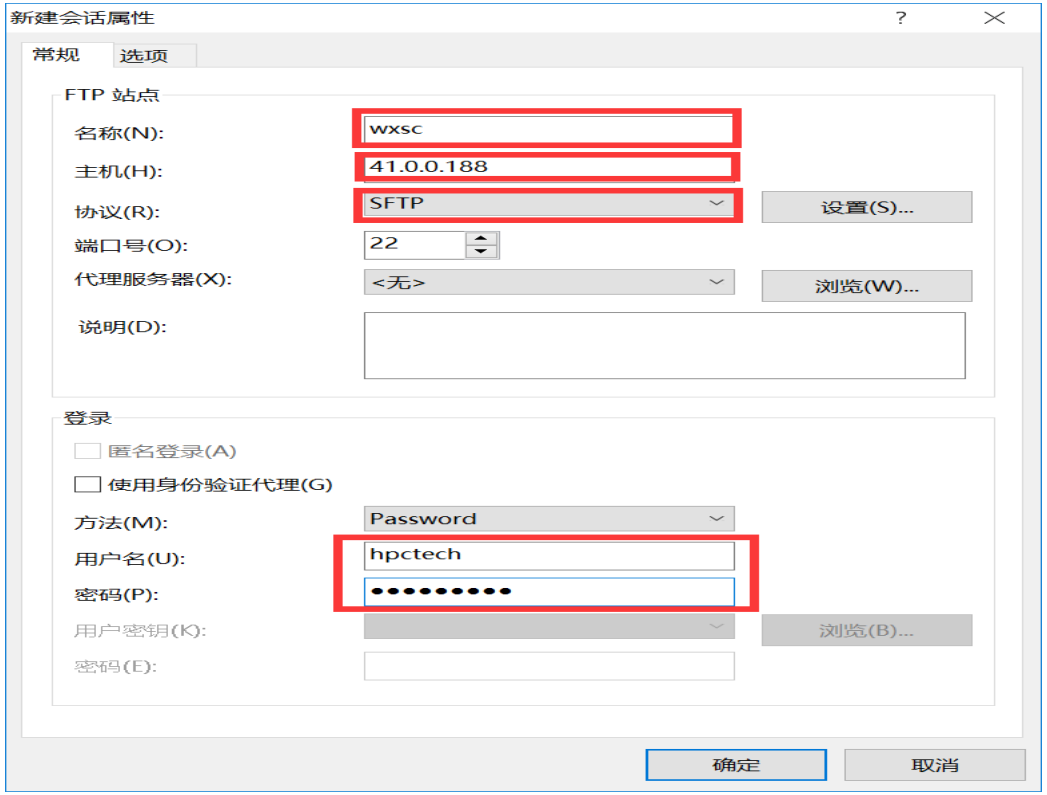


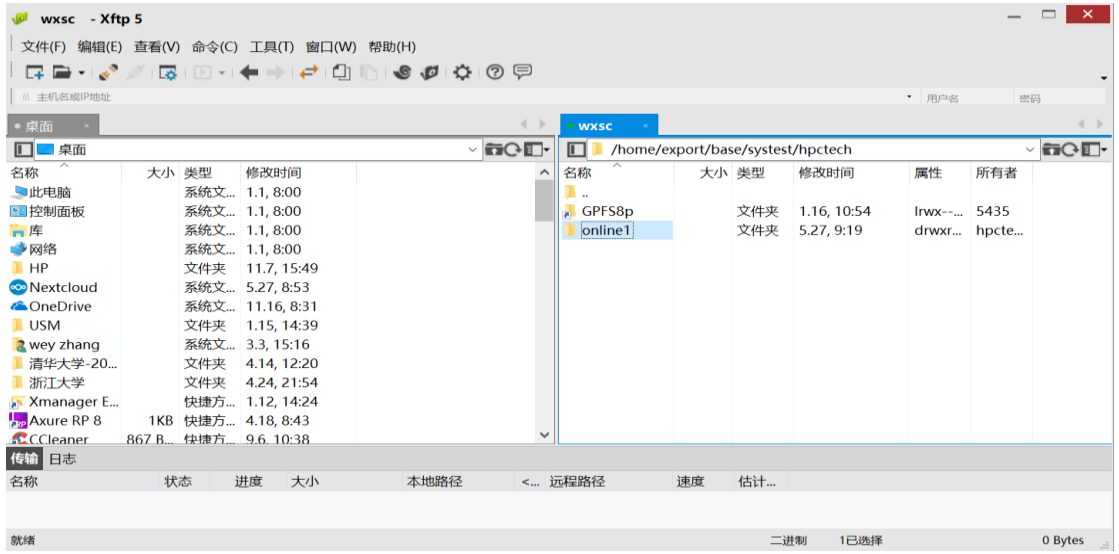
- 安装xshell 和xftp, https://www.netsarang.com/zh/free-for-home-school/
- 打开xshell，点击“新建”，输入主机IP：41.0.0.188
- 点击“用户身份验证”，输入用户名和密码
- 数据和作业提交都要在 online1 目录下

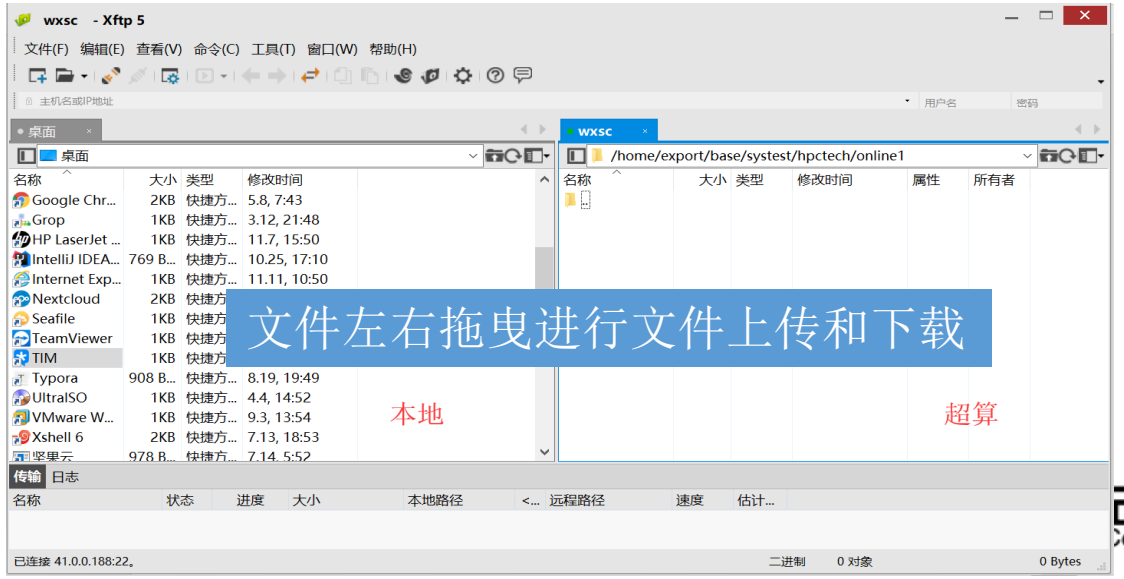


超算登录——数据传输

- 打开xftp，输入主机IP：41.0.0.188，用户名和密码
- 协议选择 SFTP
- 数据要放在在 online3 目录下





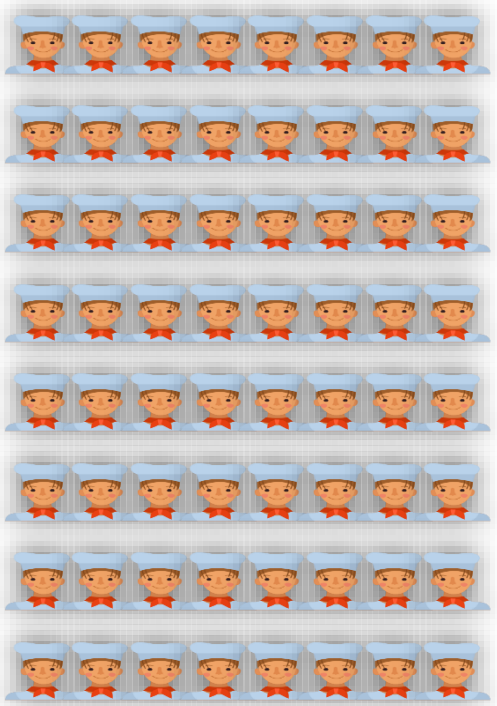


目录

国家超级计算无锡中心
National Supercomputing Center in Wuxi

炒菜任务：64道菜

领班主厨



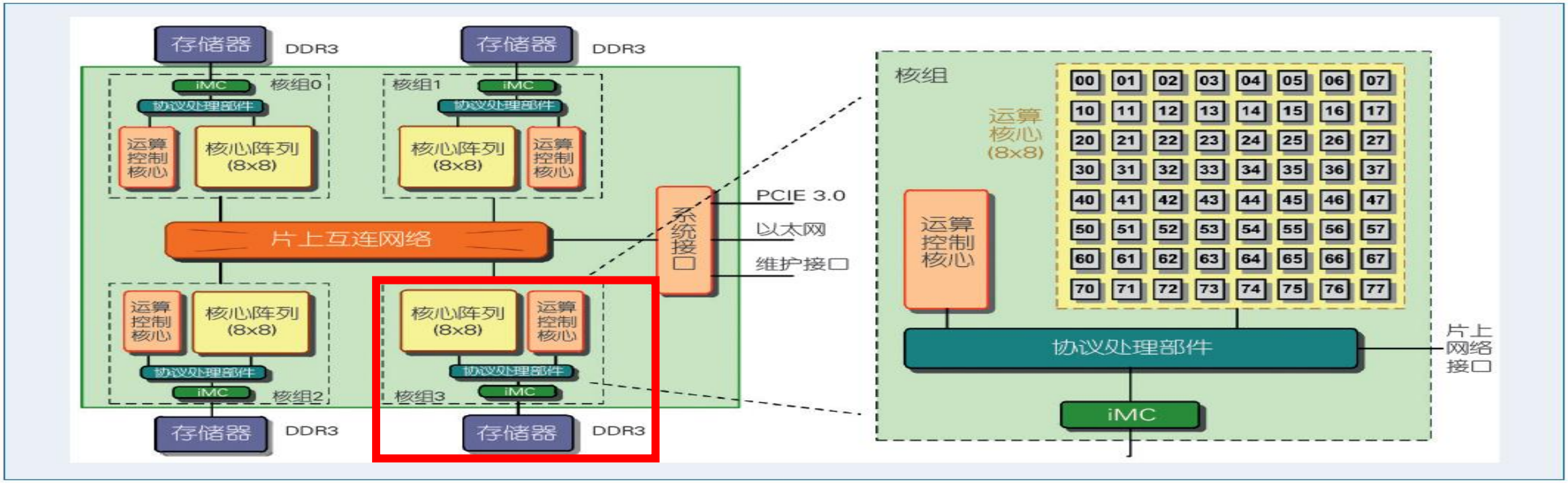
64个大厨

一般方案：主厨1分钟炒一道菜，要炒64次，需要64分钟

加速方案：增加64个大厨，每个大厨1分钟炒一道菜，1分钟即可炒好64道菜！

SW26010 处理器架构

- 一个CPU = 4个核组（4个厨房）
- 一个核组（厨房） = 1 主核（主厨） + 64 从核（大厨）



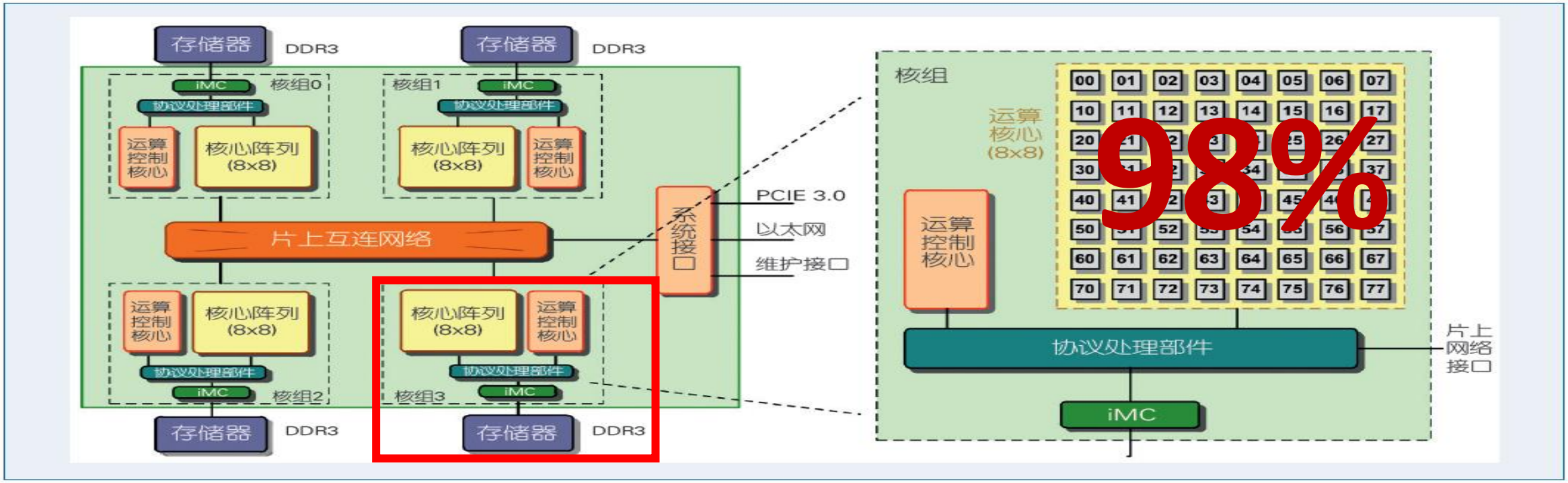
国家超级计算无锡中心
National Supercomputing Center in Wuxi

SW26010 处理器性能

主核浮点性能：$1 \times 1.45GHz \times 8(DP) \times 2(FMA) = \mathbf{23.2}GFLOPS$

从核浮点性能：$\mathbf{64} \times 1.45GHz \times 4(DP) \times 2(FMA) = \mathbf{742.4}GFLOPS$

（浮点性能是指表示每秒钟计算乘法和加法的总次数，FMA表示乘加融合，把乘法和加法融合为一条指令）



从核优化要点一：充分发挥从核计算性能

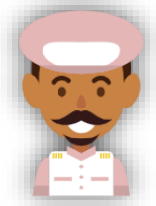
国家超级计算无锡中心
National Supercomputing Center in Wuxi

初始方案

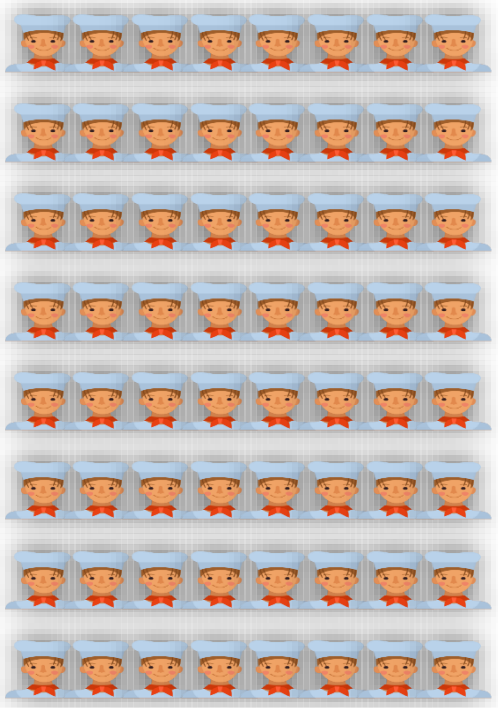
炒菜任务：64道菜

任务安排
00号大厨炒00道
01号大厨炒01道
.....
63号大厨炒63道

任务安排清单



领班主厨



64个大厨

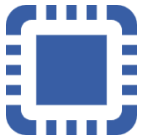
64道菜

- 每大厨炒一道菜，64个大厨“并行”炒菜
- 根据厨师编号和菜编号来分配任务

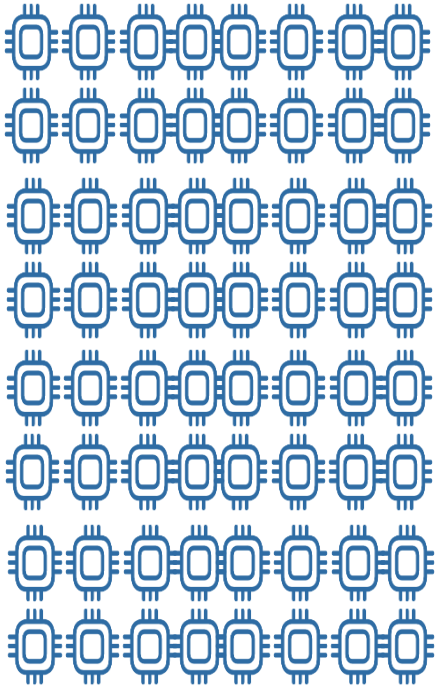
计算任务：64行矩阵加法

并行编程
00号核计算00行
01号核计算01行
.....
63号核计算63行

并行程序



一个“主”核



64个“从”核

第0行
第1行
⋮ ⋮
第62行
第63行

64行矩阵

- 每个从核计算1行，64个从核并行计算
- 根据从核编号和矩阵行号分配计算任务

从核Athread并行编程

从核并行编程关键点1：利用Athread接口，获得每个从核的ID号，再根据ID号分配计算任务

串行程序

```
#define J 64
#define I 1024
double a[J][I];
double b[J][I];
double c[J][I];
int main() {
    int i, j;
    ......
    for (i = 0; i < J; i++) {
        for (j = 0; I < N; j++) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    ......
}
```

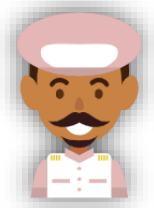
并行行程序

```
#define J 64
#define I 1024
extern double a[J][I], b[J][I], c[J][I];
void func() {

  //Step1 获得从核ID号
  my_id = athread_get_id(-1);

  //Step2 根据从核ID号拿矩阵中对应行的数据
  athread_get(……);
  athread_get(……);

  //Step3、开始计算，每个从核只要算一行
  for (i = 0; i < I; i++) {
        c_slave[i] = a_slave[i] + b_slave[i];
  }

  //Step4、每个计算好的数据，集成到矩阵中
  athread_put(……);
}
```

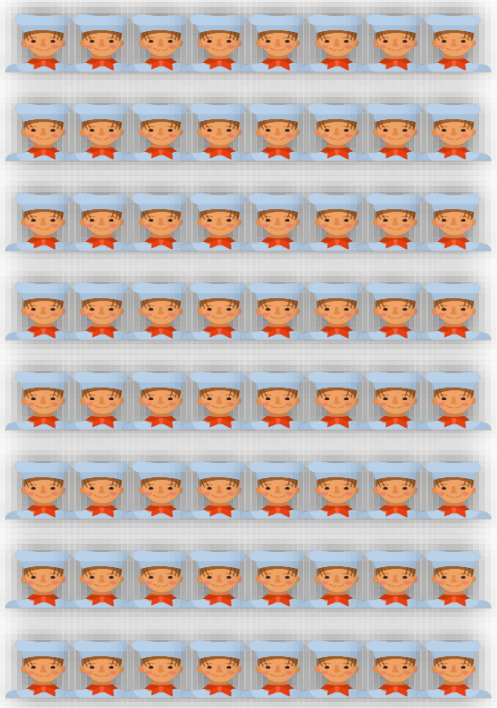
炒菜任务：64道菜

任务安排
00号大厨炒00道菜
01号大厨炒01道菜
……
63号大厨炒63道菜

任务安排清单



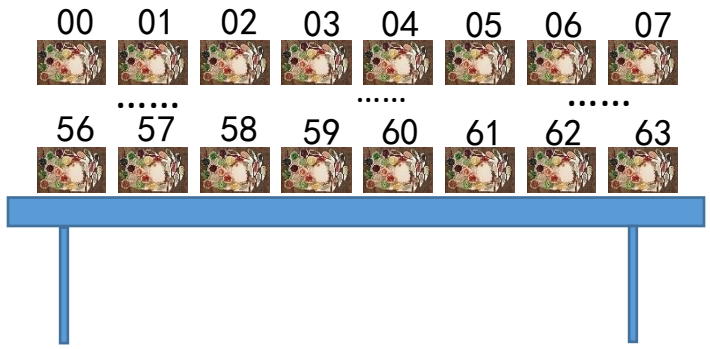
领班主厨



64个大厨

1. 领班主厨接到任务：要炒64道菜
2. 领班主厨安排炒菜任务：
   00号大厨炒第00道菜
   01号大厨炒第01道菜
   ……
   63号大厨炒第63道菜

00号大厨：
1. 开始炒菜
2. 发现第00道菜需要白鱼，先到食材架把白鱼取过来，然后继续炒菜
3. 发现第00道菜需要白虾，先到食材架把白虾取过来，然后继续炒菜
4. 发现第00道菜需要银鱼，先到食材架把银鱼取过来，然后继续炒菜
5. 炒菜结束



食材架

频繁到食材架拿食材，十分影响炒菜速度！

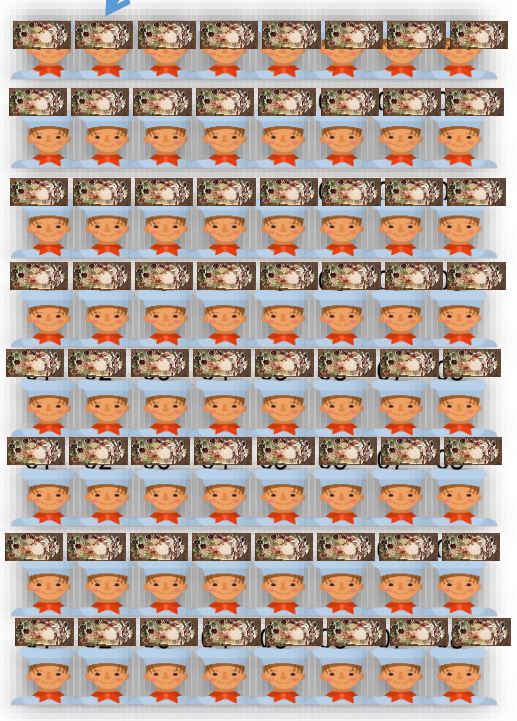
炒菜任务：64道菜

任务安排
00号大厨炒00道菜
01号大厨炒01道菜
……
63号大厨炒63道菜

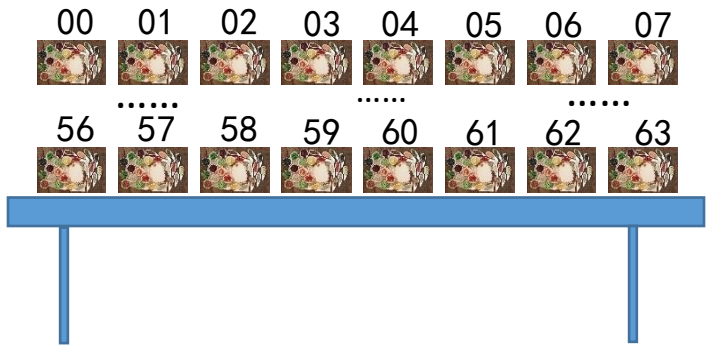
任务安排清单

领班主厨

砧板



64个大厨



食材架

1. 领班主厨接到任务：要炒64道菜
2. 领班主厨安排炒菜任务：
   00号大厨炒第00道菜
   01号大厨炒第02道菜
   ……
   63号大厨炒第63道菜

00号大厨：
1. 开始炒菜
2. 先把第00道菜需要的白鱼、白虾、银鱼等食材从食材架取到自己砧板上
3. 发现第00道菜需要白鱼，随手把砧板上的白鱼放到锅里面，继续炒菜
4. 发现第00道菜需要白虾，随手把砧板上的白鱼放到锅里面，继续炒菜
5. 发现第00道菜需要银鱼，随手把砧板上的白鱼放到锅里面，继续炒菜
6. 超菜结束

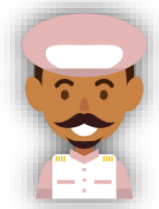
砧板先“缓存”食材，缩减拿食材的时间！

改进方案

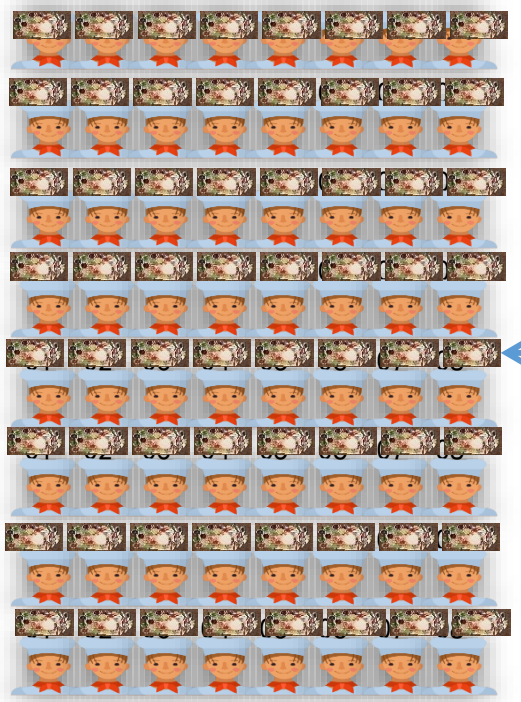
炒菜任务：64道菜

任务安排
00号大厨炒00道
01号大厨炒01道
……
63号大厨炒63道

任务安排清单



领班主厨



砧板缓存食材

64个大厨

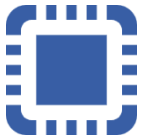
食材架

- 每个大厨炒1道菜
- 食材先从食材架缓存到砧板
- 炒菜需要食材要到的砧板上拿即可（速度快）

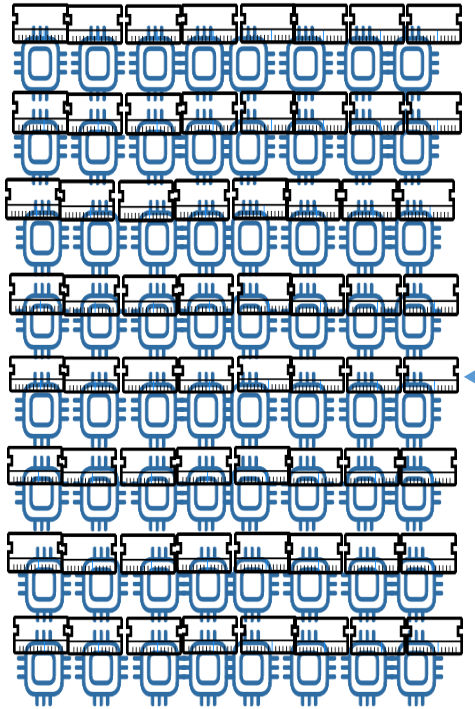
计算任务：64行矩阵加法

并行编程
00号核计算00行
01号核计算01行
……
63号核计算63行

并行程序



一个“主”核



局部存储缓存数据

64个从核

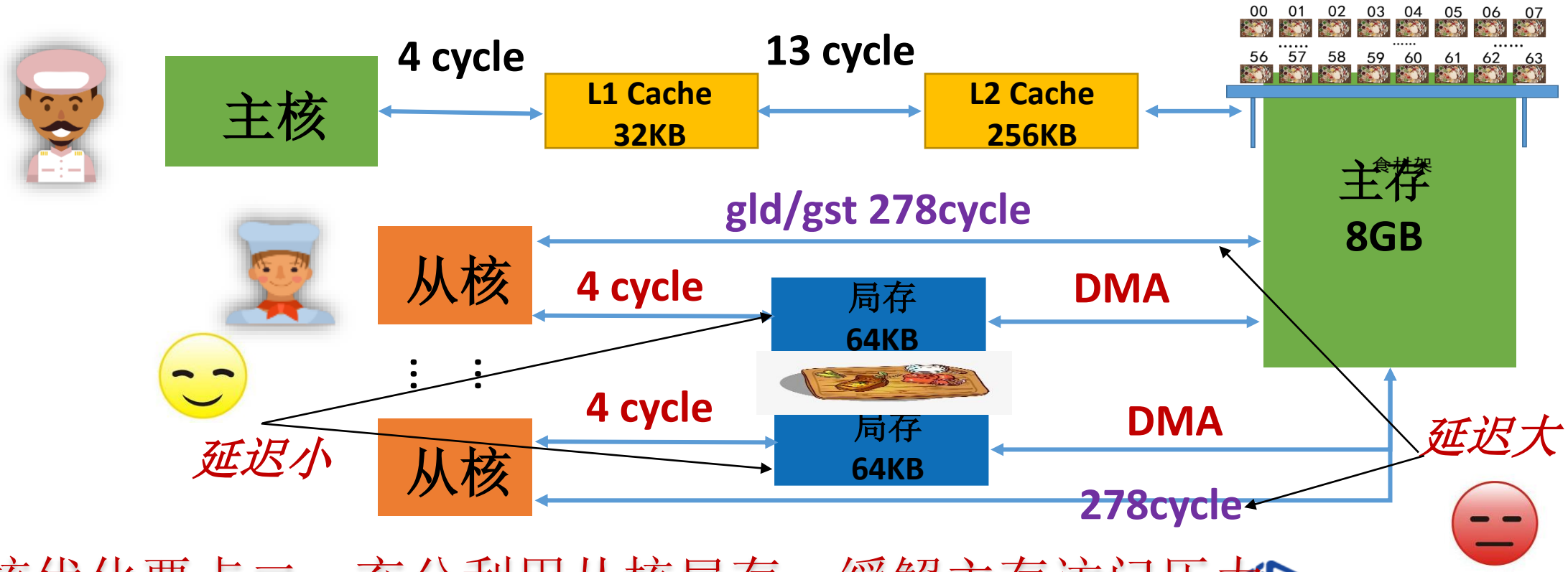
第0行
第1行
⋮ ⋮
第62行
第63行

内存

- 64个从核并行计算，每个从核计算1行
- 数据先从内存缓存到从核局部存储
- 计算需要数据，到从核局部存储中拿（延迟低）

SW26010 存储层次结构

- 每个从核有一个64KB的局部存储（LDM），LDM以草稿本 (Scratch Pad Memory, SPM) 的方式组织，由软件控制管理，属于一种可编程存储器（cache是由硬件控制，对程序员不可见）
- 从核可通过 gld/gst 离散访主存，也可以通过 DMA （ Direct Memory Access，直接内存访问）批量式访主存



从核优化要点二：充分利用从核局存，缓解主存访问压力

从核Athread并行编程

从核并行编程关键点2：利用Athread接口，把每个从核计算需要的数据拿到从核局部存储

串行程序

```
#define J 64
#define I 1024
double a[J][I];
double b[J][I];
double c[J][I];
int main() {
    int i, j;
    ......
    for (i = 0; i < J; i++) {
        for (j = 0; I < N; j++) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    ......
}
```

并行行程序

```
#define J 64
#define I 1024
extern double a[J][I], b[J][I], c[J][I];
void func() {

  //Step1 获得从核ID号
  my_id = athread_get_id(-1);

  //Step2 根据从核ID号取矩阵中对应行的数据
  athread_get(主存地址、从核地址、数据量);
  athread_get(主存地址、从核地址、数据量);

  //Step3、开始计算，每个从核只要算一行
  for (i = 0; i < I; i++) {
        c_slave[i] = a_slave[i] + b_slave[i];
  }

  //Step4、每个计算好的数据，集成到矩阵中
  athread_put(……);
}
```

从核Athread并行编程基本流程

程序分拆分成主核上运行、从核上运行两部分，两部分保存成独立的文件，单独编译，最后链接成可执行文件

```c
#define J 64
#define I 1024
double a[J][I];
double b[J][I];
double c[J][I];
int main() {
    int i, j;
    ......
    for (i = 0; i < J; i++) {
        for (j = 0; I < N; j++) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    ......
}
```

host.c

```c
#define J 64
#define I 1024
double a[J][I];
double b[J][I];
double c[J][I];
int main() {
    int i, j;
    ......



    ......
}
```

slave.c

```c
for (i = 0; i < J; i++) {
    for (j = 0; I < N; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

```c
#define J 64
#define I 1024
extern double a[J][I], b[J][I], c[J][I];
void func() {


  //Step1 获得从核ID号
  my_id = athread_get_id(-1);

  //Step2 根据从核ID号拿矩阵中对应行的数据
  athread_get(……);
  athread_get(……);


  //Step3、开始计算，每个从核只要算一行
  for (i = 0; i < I; i++) {
        c_slave[i] = a_slave[i] + b_slave[i];
  }

  //Step4、每个计算好的数据，集成到矩阵中
  athread_put(……);
}
```

核心工作：利用Athread接口，把要在从核上运行的代码，改造成64个从核高效并行

从核Athread并行编程

- 利用Arhread接口将程序改造成可以在从核上并行运行的程序
- 程序分拆分成主核上运行、从核上运行两部分，两部分保存成独立的文件，单独编译，最后链接成可执行文件

串行程序

```
#define J 64
#define I 1024
double a[J][I];
double b[J][I];
double c[J][I];
int main() {
    int i, j;
    ......
    for (i = 0; i < J; i++) {
        for (j = 0; I < N; j++) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    ......
}
```

从核函数

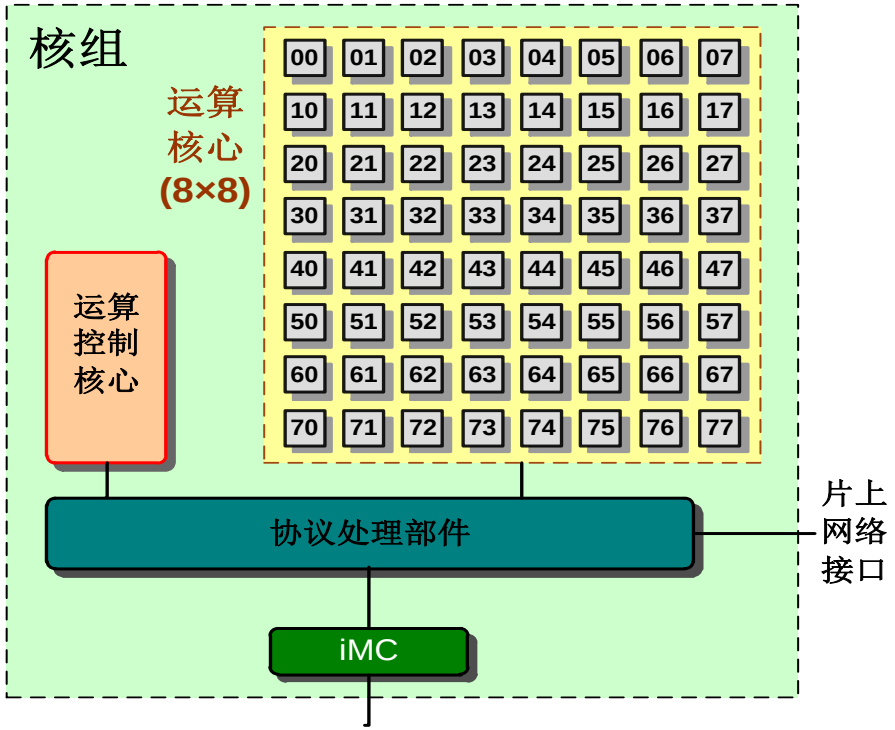
```
#define J 64
#define I 1024
extern double a[J][I], b[J][I], c[J][I];
void func() {
    volatile int i, j,my_id, get_reply, put_reply;


}
```

从核函数
单独保存成文件

slave.c

从核Athread并行编程

- **double a_slave[I]** 在从核局部存储中声明数组，用于缓存在主存中的计算所需数据
- **athread_get_id(-1)** 获取从核 ID 号，根据 ID 号控制每个从核的计算任务，在本例中，my_id = 0 的从核计算第 0 行， my_id = 1 的从核计算第 1 行，以此类推



从核函数

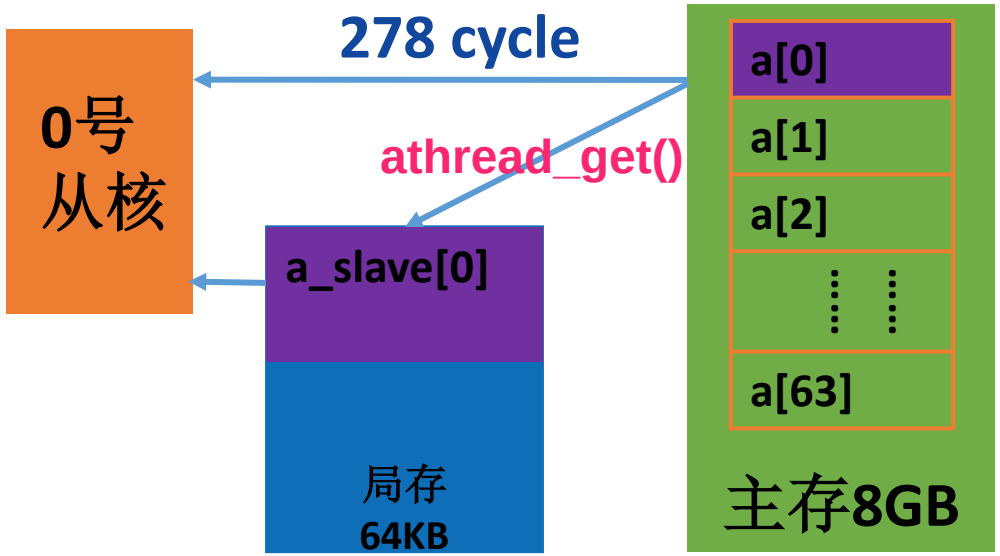
```
#define J 64
#define I 1024
extern double a[J][I], b[J][I], c[J][I];
void func() {
 volatile int i, j,my_id, get_reply, put_reply;

  //1、在从核局部存储声明数组，用于缓存计算所需数据
  double a_slave[I], b_slave[I], c_slave[I];
  //2、获得从核ID号
  my_id = athread_get_id(-1);

}
```

从核函数单独保存成文件

slave.c

从核Athread并行编程

- 从核计算前，先把需要的数据（数组a和b的my_id行）从主存缓存到局部存储（LDM）
- **athread_get(……)** 从主存缓存数据到从核LDM
- **athread_get(……)** 关键参数： & a[my_id][0]主存地址（从哪个位置开始缓存数据）、从核地址&a_slave[0]（缓存的数据存储在哪个位置）、字节长度 I*8（缓存多少数据）、 & get_reply 回答字 （标识缓存操作是否完成）
- **athread_get** 每完成一次数据缓存操作，回答字 get_reply 自动加 1，通过检测回答字的值是否等于调用次数判断数据是否缓存完毕



从核函数

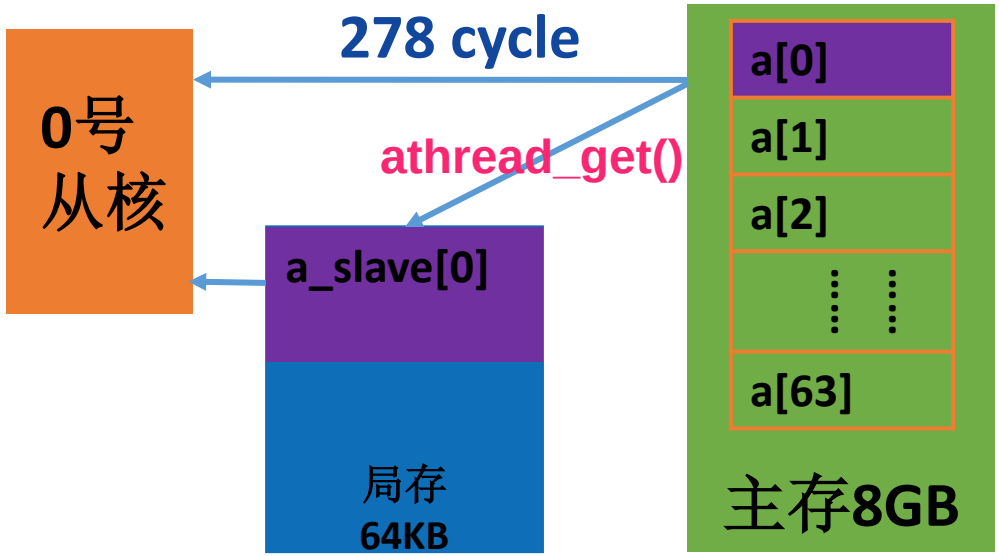
```
#define J 64
#define I 1024
extern double a[J][I], b[J][I], c[J][I];
void func() {
 volatile int i, j,my_id, get_reply, put_reply;

  //1、在从核局部存储声明数组，用于缓存计算所需数据
  double a_slave[I], b_slave[I], c_slave[I];
  //2、获得从核ID号
  my_id = athread_get_id(-1);

  //3、从主存缓存计算所需数据到LDM，a和b数组的第my_id行
  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);
   /*每完成一次athread_get操作，回答字get_reply自动加1，回答字等于2，
说明两次athread_get操作完成，可以进行下一步计算，否则将等待完成*/
  while (get_reply != 2);
```

从核函数单独保存成文件

slave.c

从核Athread并行编程

- 从核计算前，先把需要的数据（数组a和b的my_id行）从主存缓存到局部存储（LDM）
- **athread_get(……)** 从主存缓存数据到从核LDM
- **athread_get(……)** 关键参数： & a[my_id][0]主存地址（从哪个位置开始缓存数据）、从核地址&a_slave[0]（缓存的数据存储在哪个位置）、字节长度 I*8（缓存多少数据）、 & get_reply 回答字 （标识缓存操作是否完成）
- **athread_get** 每完成一次数据缓存操作，回答字 get_reply 自动加 1，通过检测回答字的值是否等于调用次数判断数据是否缓存完毕



从核函数

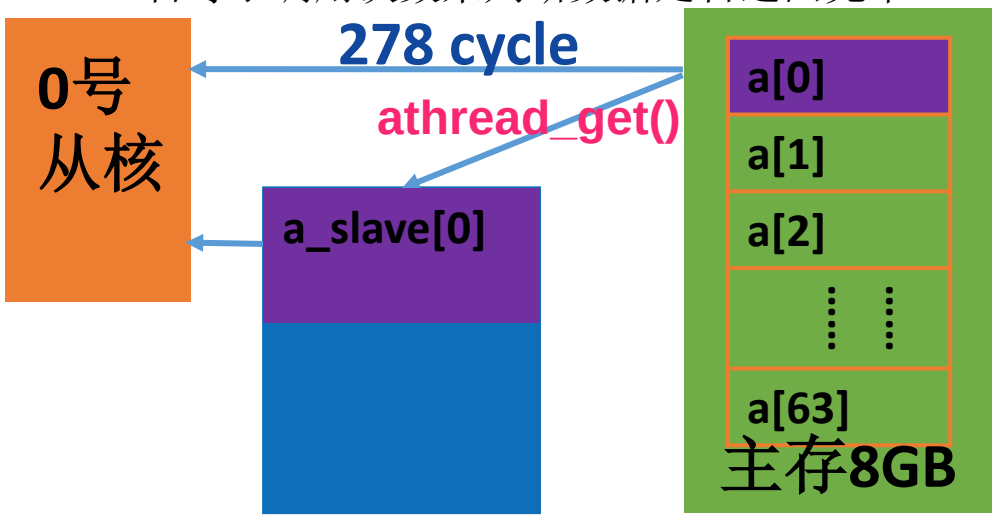
```
#define J 64
#define I 1024
extern double a[J][I], b[J][I], c[J][I];
void func() {
 volatile int i, j,my_id, get_reply, put_reply;

  //1、在从核局部存储声明数组，用于缓存计算所需数据
  double a_slave[I], b_slave[I], c_slave[I];
  //2、获得从核ID号
  my_id = athread_get_id(-1);

  //3、从主存缓存计算所需数据到LDM ， a和b数组的第my_id行
  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);
   /*每完成一次athread_get操作，回答字get_reply自动加1，回答字等于2，
说明两次athread_get操作完成，可以进行下一步计算，否则将等待完成*/
  while (get_reply != 2);

  //4、开始计算，每个从核负责算一行
  for (i = 0; i < I; i++) {
          c_slave[i] = a_slave[i] + b_slave[i];
  }
```

从核函数单独保存成文件

slave.c

从核Athread并行编程

- 从核计算结果存储在从核局部存储（LDM）c_slave 数组
- **athread_put (……)** 将从核LDM中的数据，返回到主存中
- **athread_put(……)** 关键参数： &c_slave[my_id][0] 从核地址（返回哪个位置的数据）、主存地址 &c[my_id][0]（返回的数据存储在主存中哪个位置）、字节长度 I*8（返回多少数据量）、& put_reply 回答字 （标识操作是否完成）
- athread_put 每完成一次数据返回操作，回答字 put_reply 自动加1，通过检测回答字的值是否等于调用次数来判断数据是否返回完毕



从核函数

```
#define I 1024
extern double a[J][I], b[J][I], c[J][I];
void func() {
 volatile int i, j,my_id, get_reply, put_reply;

  //1、在从核局部存储声明数组，用于缓存计算所需数据
  double a_slave[I], b_slave[I], c_slave[I];
  //2、获得从核ID号
  my_id = athread_get_id(-1);

  //3、从主存缓存计算所需数据到LDM ， a和b数组的第my_id行
  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);
   /*每完成一次athread_get操作，回答字get_reply自动加1，回答字等于2，
说明两次athread_get操作完成，可以进行下一步计算，否则将等待完成*/
  while (get_reply != 2);

  //4、开始计算，每个从核负责算一行
  for (i = 0; i < I; i++) {
          c_slave[i] = a_slave[i] + b_slave[i];
  }

  //5、计算结束，把计算结果c_slave存储到主存中c数组的my_id行
  athread_put(PE_MODE, &c_slave[0], &c[my_id][0], I * 8,
          &put_reply, 0, 0);
  /*每完成一次athread_put操作，回答字put_reply自动加1，回答字等于1，说
明athread_put操作完成，可以进行下一步计算，否则将等待完成*/
   while (put_reply != 1);
}
```

从核函数单独保存成文件

slave.c

从核Athread并行编程

- 主核函数调用从核函数，要用到四个 Athread 接口
- 通过定义在主存的全局变量（共享变量）来传递参数，也可以通过athread_spawn传递

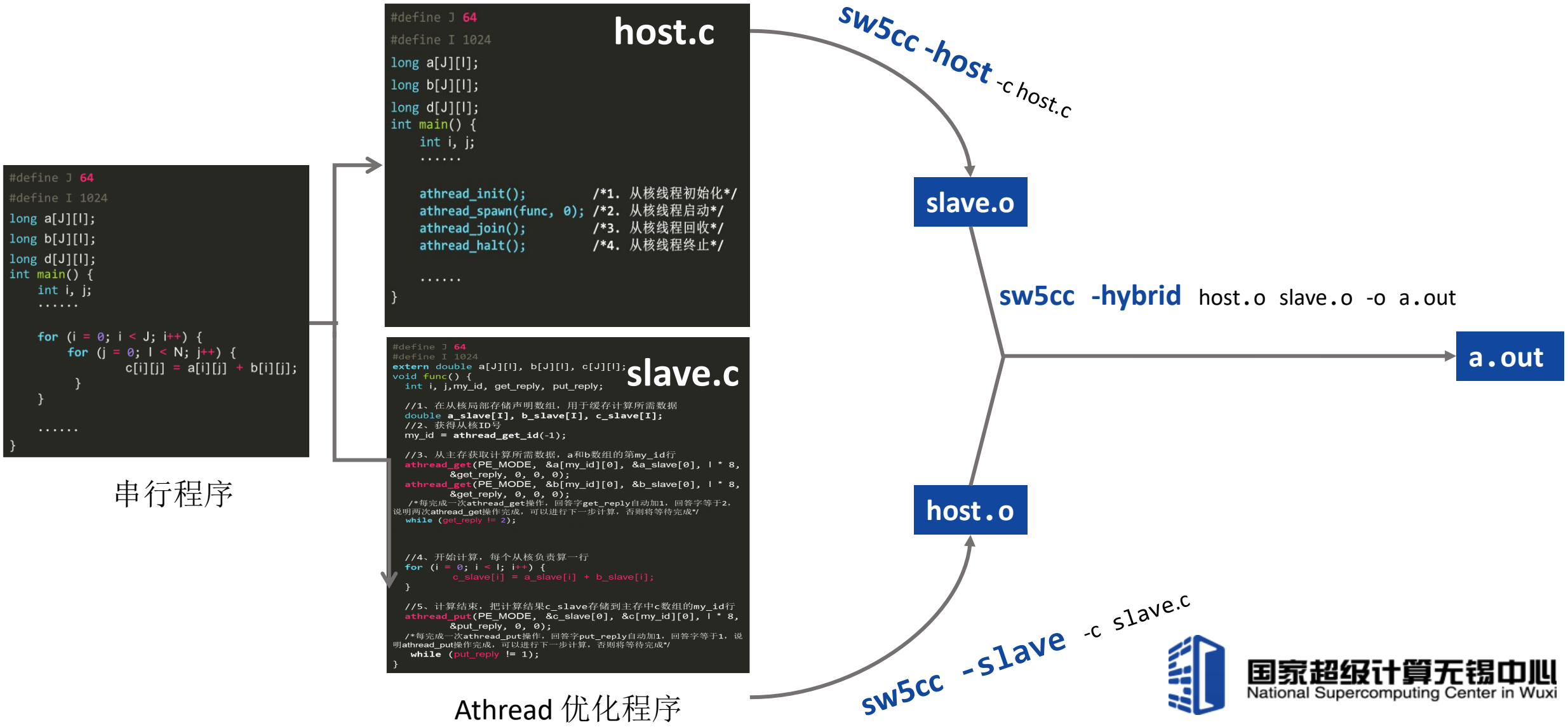
主核函数

```
#define J 64
#define I 1024
extern SLAVE_FUN(func)(); //从核函数声明
long a[J][I];
long b[J][I];
long c[J][I];
int main() {
    int i, j;
    ......

    athread_init();           /*1. 从核线程初始化*/
    athread_spawn(func, 0); /*2. 从核线程启动*/
    athread_join();           /*3. 从核线程回收*/
    athread_halt();           /*4. 从核线程终止*/

    ......
}
```

从核函数

```
#define J 64
#define I 1024
extern double a[J][I], b[J][I], c[J][I];
void func() {
  int i, j,my_id, get_reply, put_reply;

  //1、在从核局部存储声明数组，用于缓存计算所需数据
  double a_slave[I], b_slave[I], c_slave[I];
  //2、获得从核ID号
  my_id = athread_get_id(-1);

  //3、从主存获取计算所需数据，a和b数组的第my_id行
  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);
   /*每完成一次athread_get操作，回答字get_reply自动加1，回答字等于2，
说明两次athread_get操作完成，可以进行下一步计算，否则将等待完成*/
  while (get_reply != 2);


  //4、开始计算，每个从核负责算一行
  for (i = 0; i < I; i++) {
          c_slave[i] = a_slave[i] + b_slave[i];
  }

  //5、计算结束，把计算结果c_slave存储到主存中c数组的my_id行
  athread_put(PE_MODE, &c_slave[0], &c[my_id][0], I * 8,
          &put_reply, 0, 0);
  /*每完成一次athread_put操作，回答字put_reply自动加1，回答字等于1，说
明athread_put操作完成，可以进行下一步计算，否则将等待完成*/
   while (put_reply != 1);
}
```

Athread 程序编译



Athread 程序运行

bsub -I -q q_sw_expr -b -n 1 -cgsp 64 -host_stack 1024 -share_size 6500 ./a.out

- 参数1：**-I**，程序输出会打印到终端，终端关闭则程序终止运行，去掉该选项加上“**-o out.log**”输出会重定向到out.log文件，终端关闭不影响程序运行
- 参数2：**-q**，程序提交到 q_sw_expr 计算节点队列运行
- 参数3：**-n**，程序运行使用的核组数，最好是4的倍数，因为一个节点有4个核组
- 参数4：**-b**，从核函数栈变量放在从核局部存储上，为获取加速性能 必须的提交选项
- 参数5：**-cgsp**，使用从核的个数，一般设置为64
- 参数6：**-host_stack**，指定主核栈空间大小，默认为8M，一般设置为128MB以上
- 参数7：**-share_size**，指定核组共享空间大小，一般 share_size + host_stack 小于7500

程序管理

- 常用的作业管理命令
  - 我的作业怎么样了，bjobs，可查询到作业状态和作业号
  - 我要停止提交的作业，bkill 作业号
  - 我有哪些计算队列可用？有多少节点？ qload -w

```
[swtest@psn002 ~]$ qload -w
+=============================================================================================================+
QUEUE_NAME                  CONFIG IDLE   BUSY   BOOT   SLEEP DOWN   SOFTFT HARDFT IDLEMPE IDLESPE   BUSYMPE BUSYSPE
+=============================================================================================================+
q_sw_expr                   64     56     7      0      0     0      0      1      224     14336     25      84
q_x86_expr                  3      0      3      0      0     0      0      0      0       0         72      0
```

- 每个账号默认有两个测试队列，队列是包含一系列计算节点的集合
- q_sw_expr，采用国产处理器的测试队列，程序只能运行1小时，最多使用16个节点
- q_sw_cpc_1，采用国产处理器的比赛专用队列，程序运行时间无限制，最多使用16个节点
- CONFIG：总节点数，IDEL空闲节点数，BUSY已使用节点数，其它参数普通用户不用关心

Athread常用接口说明

```
int athread_get_id(-1);
   /*
        获取从核ID号
  */
```

```
int athread_get(dma_mode mode,
                    void *src, void *dest, int len, void *reply, /*重点关注*/
                    char mask,int stride,int bsize);
/*
   作用：从核局存LDM接收主存MEM数据，将MEM的数据get到LDM指定位置
   dma_mode mode：DMA传输命令模式          void *src：DMA传输主存源地址；
   void *dest：   DMA传输本地局存目标地址；    int len：DMA传输数据量，以字节为单位；
   void *reply：  DMA传输回答字地址，必须为局存地址，地址4B对界；
   char mask：    DMA传输广播有效向量，有效粒度为核组中一行，某位为1表示对应的行传输有效，
                  作用于广播模式和广播行模式；
   int stride：   主存跨步，以字节为单位；
   int bsize：    行集合模式下，必须配置，用于指示在每个从核上的数据粒度大小；
                  其它模式下，在DMA跨步传输时有效，表示DMA传输的跨步向量块大小，以字节为单位。
*/
```

Athread常用接口说明

```
int athread_put(dma_mode mode,
                void *src, void *dest, int len, void *reply,  /*重点关注*/
                int stride,int bsize)
/*
   作用：从核局存LDM接收主存MEM数据，将MEM的数据get到LDM指定位置
   dma_mode mode：DMA传输命令模式          void *src：DMA传输主存源地址；
   void *dest：DMA传输本地局存目标地址；      int len：DMA传输数据量，以字节为单位；
   void *reply：DMA传输回答字地址，必须为局存地址，地址4B对界；
   int stride：主存跨步，以字节为单位；
   int bsize：行集合模式下，必须配置，用于指示在每个从核上的数据粒度大小；
              其它模式下，在DMA跨步传输时有效，表示DMA传输的跨步向量块大小，以字节为单位。
*/
int athread_spawn(start_routine fpc, void * arg)
/*
  作用：在当前进程中添加受控线程组。最先执行的任务由函数fpc指定，函数fpc的参数由arg提供
  start_routine fpc：函数指针；
  void * arg：函数f的参数起始地址
*/
```

SW26010编译环境

- **基础编译器支持C、C++、和Fortran程序**
  - C编译器：sw5cc
  - C++编译器：sw5CC
  - Fortran编译器：sw5f90
- **通过不同选项区分不同代码**
  - 运算控制核心代码生成：-host
  - 运算核心代码生成：-slave
  - 混合链接：-hybrid
- **MPI编译器**
  - Fortran: mpif90
  - C语言: mpicc
  - C++: mpiCC
- **OpenACC编译器**
  - C语言: swacc
  - Fortran: swafort

语言/ 目标核心	运算控制核心	运算核心	混合链接
C	sw5cc -host	sw5cc -slave	sw5cc -hybrid
C++	sw5CC -host	不支持	sw5CC -hybrid
Fortran	sw5f90 -host	sw5f90 -slave	sw5f90 -hybrid

SW26010编译器快速入门

- 一个纯主核程序的编译示例

```
sw5cc -host -c hello.c
sw5cc -hybrid hello.o -o a.out
```

- 一个异构从核程序的编译示例

```
sw5cc -host -c host.c
sw5cc -slave -c slave.c
sw5cc -hybrid host.o slave.o -o a.out
```

- OpenACC编译示例

```
swacc hello.c -o hello.out          swafort hello.f90 -o hello.out
```

- MPI编译示例

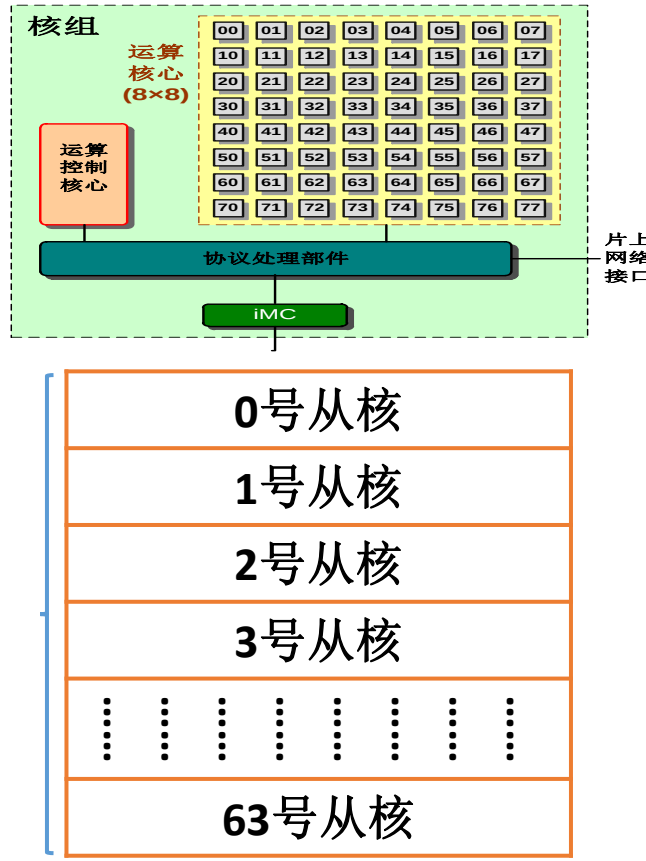
```
mpicc hello.c -o hello.out          mpif90 hello.f90 -o hello.out
```

OpenACC 从核并行编程

在计算密集区域前加一些 编译标记，OpenACC 编译器会根据标记含义自动将代码编译成可利用从核并行计算的程序

```
#define M 64
long a[M][N];
long b[M][N];
long c[M][N];
int main() {
        int i, j;
        ……
        #pragma acc parallel loop copyin(a,b) copyout(c)
        for (i = 0; i < M; i++) {
                for (j = 0; j < N; j++) {
                        c[i][j] = a[i][j] + b[i][j];
                }
        }
        return 0;
}
/*
编译：swacc matadd.c -o test
运行：bsub -I -q q_sw_expr -b -n 1 -cgsp 64 -host_stack 1024 -share_size 6500 ./test
*/
```

每个从核计算两行



国家超级计算无锡中心
National Supercomputing Center in Wuxi

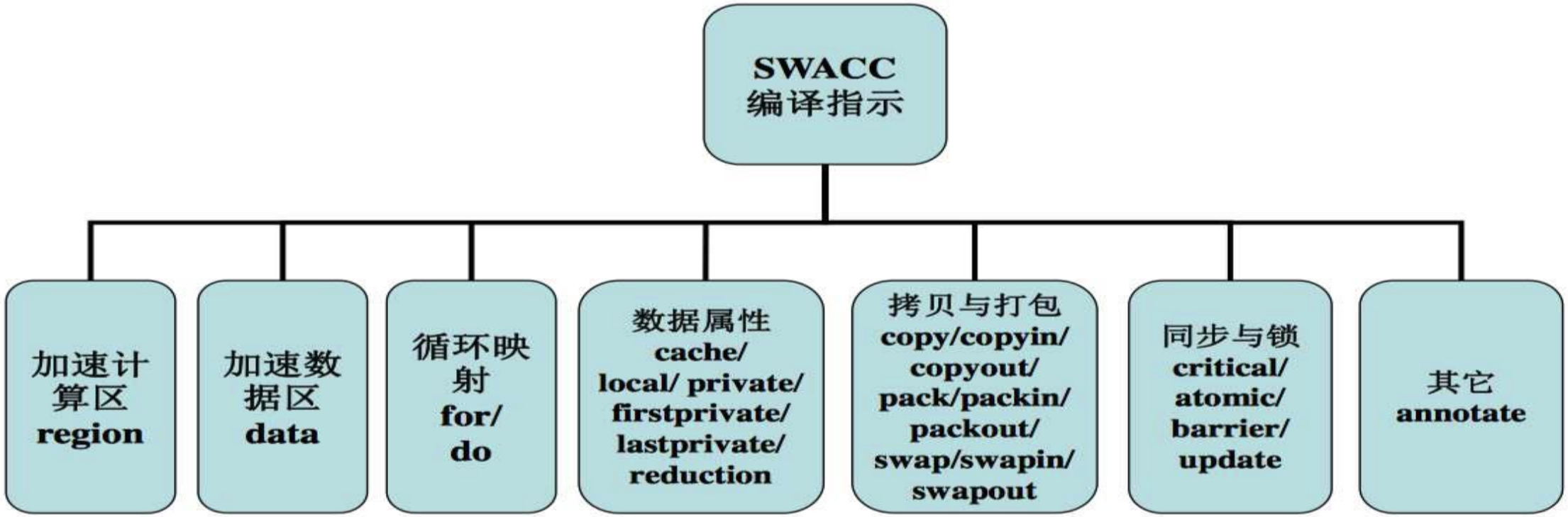
OpenACC编译指示字句一览

C语言示例：

```c
#pragma acc parallel 子句列表
for (i = 0; i < M; i++)
{
        d[i] = a[i] + b[i];
}
```

Fortran示例：

```fortran
!$ACC PARALLEL子句列表
        do m=1,N5
                axvalue2 = a(m) + 1.11
        enddo
!$ACC END PARALLEL
```

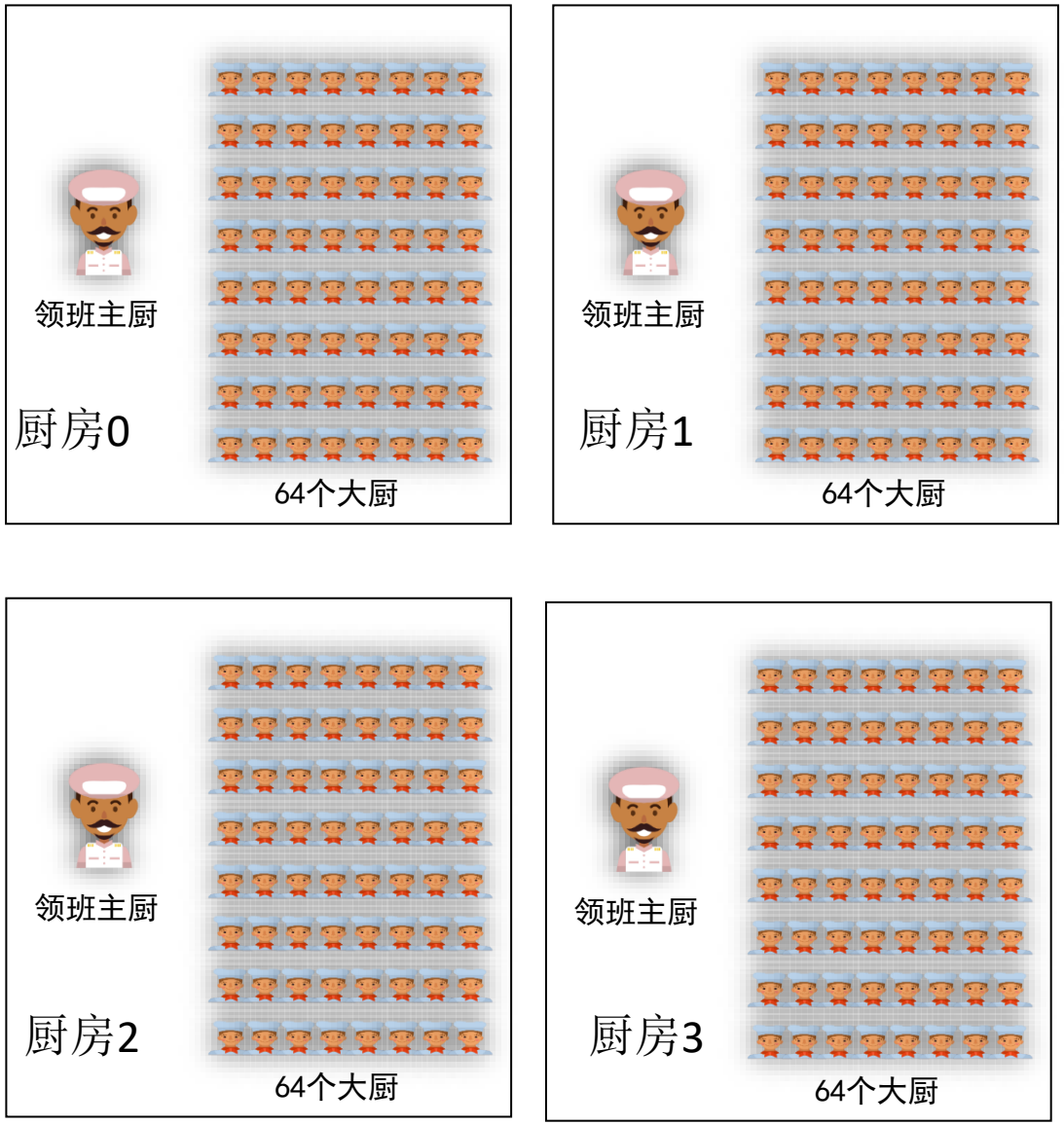


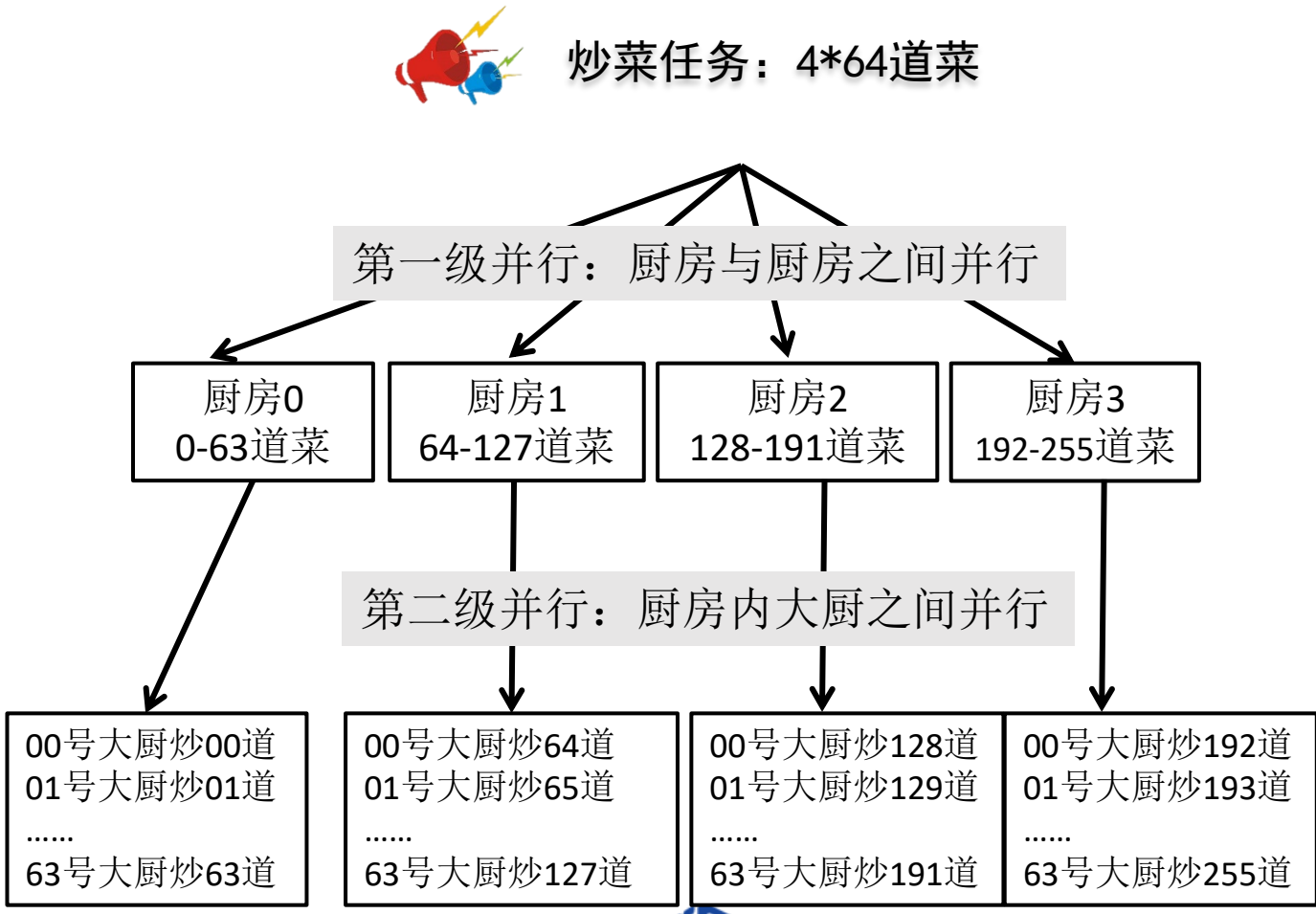
OpenACC常用编译指示字句

parallel	用在一个代码段之前，表示这段代码将作为加速任务加载到从核上执行
parallel loop	parallel 和 loop 指示的组合，用在一个循环之前，表示循环的代码将被多个加速线程并行执行
local	变量列表中的变量、数组等，在从核LDM中申请空间
copy/copyin/copyout	可指定把标量、数组、数组的全部或一部分在主存和局存之间拷贝
pack/packin/packout	把多个变量打包为一个大变量，以减少拷贝数据到局存的操作次数
swap/swapin/swapout	对数组进行转置，以消除不连续的访问
tile	将循环按指定的块大小分割
collapse	循环合并
annotate	对循环映射、数组大小、数组属性(如只读、访问特点)等进行补充说明

详见神威太湖之光OpenACC用户手册

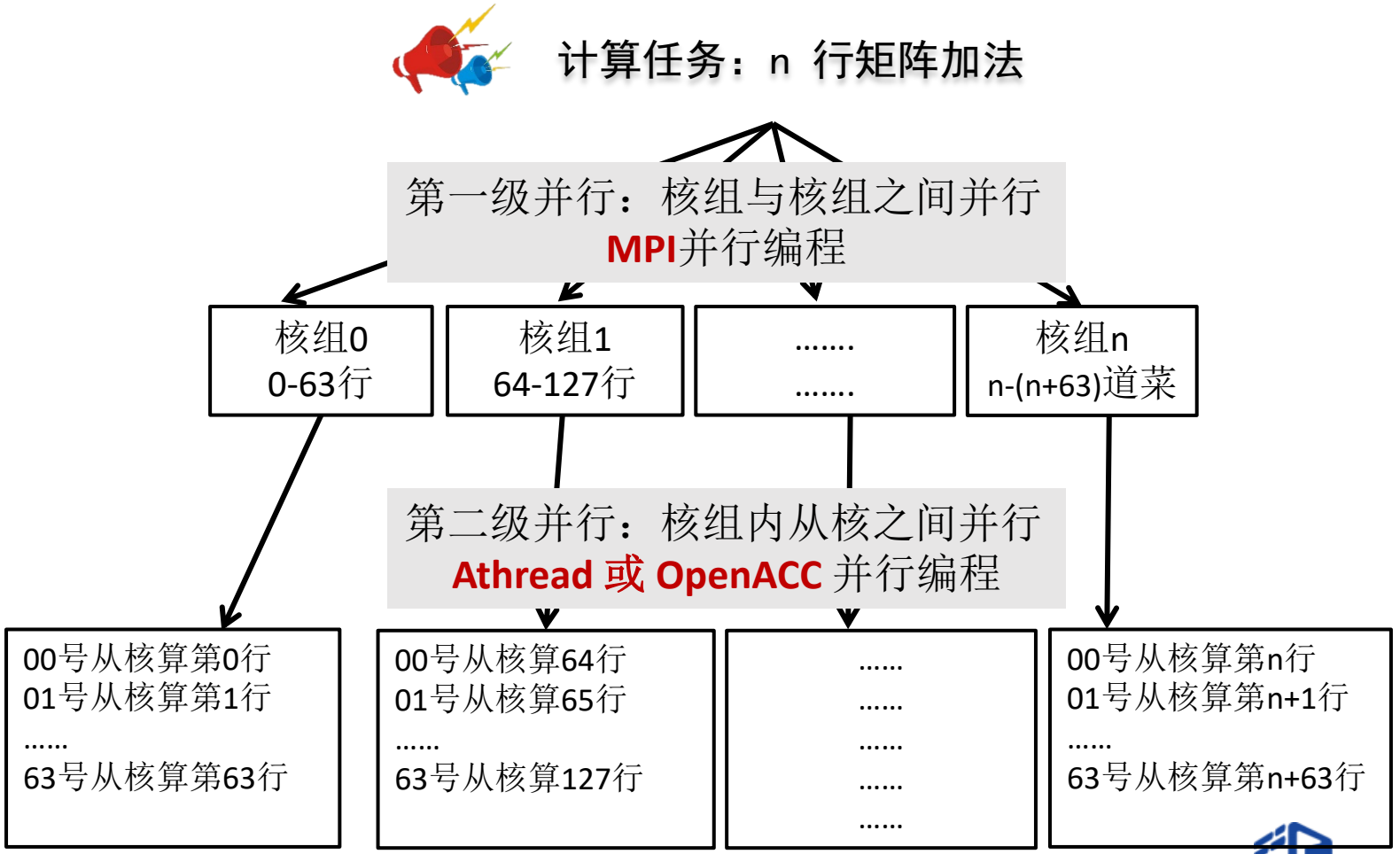
SW26010 并行编程模式——两级并行





国家超级计算无锡中心
National Supercomputing Center in Wuxi

SW26010 并行编程模式——两级并行



参考资料

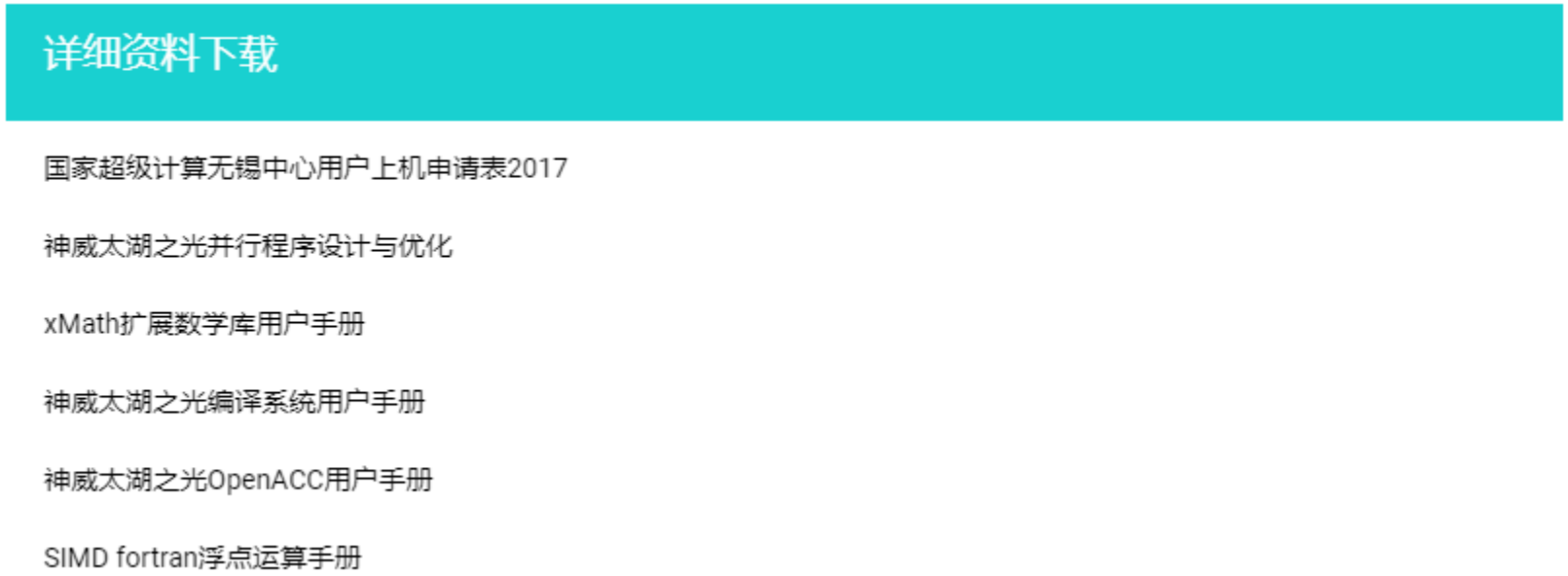
- 示例程序路径： /usr/sw-mpp/example/Athread_OpenACC

```
Athread
    EX1-mpi-athread-c
    EX2-mpi-athread-c++
    EX3-mpi-athread-fortran
    EX4-mpi-athread-allshare
    EX5-mpi-athread-allshare-master
    EX6-mpi-pthread-athread-allshare
    EX7-perf
    EX8-mpi-athread-doublebuffer
    EX9-mpi-athread-register
    lecture-demo
OpenACC
    1-matmax
    2-matadd
    3-spmz
    4-matmul
```

课程演示例子

参考资料

www.nsccwx.cn



swDNN、swCaffe

https://github.com/feifeibear/swDNN

国家超级计算无锡中心
National Supercomputing Center in Wuxi

参考资料

无锡超算微课堂

已群发消息

07月31日
发送完毕

原创 10分钟入门Athread并行编程

07月25日
发送完毕

原创 10分钟入门OpenACC并行编程

07月18日
发送完毕

原创 10分钟了解申威26010处理器的架构及性能特性

07月15日
发送完毕

原创 10分钟带你把超算用起来

国家超级计算无锡中心
National Supercomputing Center in Wuxi

THANK YOU