并行程序示例

上海交通大学高性能计算中心
http://hpc.sjtu.edu.cn

2014年3月3日更新

本文档介绍如何在π超级计算机上编译和提交并行作业任务。π支持OpenMP、MPI、CUDA等并行编程模型。再继续阅读本文档之前，您应该知道如何登录π、使用LSF提交作业、Module的基本概念。下面几个文档可以帮助您完成准备工作：

- 使用SSH登录高性能计算节点 **http://pi.sjtu.edu.cn/docs/SSH_ch**
- LSF作业管理系统使用方法 **http://pi.sjtu.edu.cn/docs/LSF_ch**
- 使用Environment Module设置运行环境 **http://pi.sjtu.edu.cn/docs/Module_ch**

本文档的所有示例代码均收录在登录节点`/lustre/utility/pi-code-sample`目录下。

1 OpenMP示例

Pi集群上GCC和Intel编译器都支持OpenMP扩展。示例代码`omp_hello.c`内容如下：

```
#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 */

}
```

1.1 使用GCC编译器

GCC编译OpenMP代码时加上`-fopenmp`：

```
$ gcc -fopenmp omp_hello.c -o ompgcc
```

在本地使用4线程运行程序：

```
$ export OMP_NUM_THREADS=4 && ./ompgcc
```

正式运行时需要提交到LSF作业管理系统，提交脚本`ompgcc.lsf`如下，仍使用4线程运行(增加约束条件让所有线程分配到一台物理机上)：

```
#BSUB -L /bin/bash
#BSUB -J HELLO_OpenMP
#BSUB -n 4
#BSUB -e %J.err
#BSUB -o %J.out
#BSUB -R "span[hosts=1]"
#BSUB -q cpu

export OMP_NUM_THREADS=4
./ompgcc
```

提交到LSF，查看程序输出：

```
$ dos2unix ompgcc.lsf && bsub < ompgcc.lsf
$ sleep 10 && bpeek
```

1.2 使用Intel编译器

Intel编译器`icc`编译OpenMP代码时需要使用`-openmp`参数。

```
$ module purge && module load icc/13.1.1 && icc -openmp omp_hello.c -o ompintel
```

在本地使用4线程运行：

```
$ module purge && module load icc/13.1.1/ && export OMP_NUM_THREADS=4 && ompintel
```

LSF作业脚本`ompintel.lsf`内容如下：

```
#BSUB -L /bin/bash
#BSUB -J HELLO_OpenMP
#BSUB -n 4
#BSUB -e %J.err
#BSUB -o %J.out
#BSUB -R "span[hosts=1]"
#BSUB -q cpu

MODULEPATH=/lustre/utility/modulefiles:$MODULEPATH
module purge
module load icc/13.1.1

export OMP_NUM_THREADS=4
./ompintel
```

提交到LSF：

```
$ dos2unix ompintel.lsf && bsub < ompintel.lsf
$ sleep 10 && bpeek
```

2 MPI示例

MPI (Message Passing Interface) 是并行计算中使用非常广泛的编程接口。它定义了一组标准的进程间消息传递接口，进程可以在同一节点或跨节点进行消息通信。软硬件厂商可以在保证MPI接口相容的前提下，设计自己的实现。π超级计算机上支持的MPI实现包括Intel MPI、MPICH2、OpenMPI、MVAPCH2等，用户调用不同的Module即可在不同的MPI实现间切换。用户在登录节点上选择需要的MPI环境，编译程序后提交给LSF作业管理系统。如果从原有的MPI实现切换到另一个MPI实现，MPI程序需要重新编译。

本节演示如何在不同的MPI环境下编译和运行名为`mpihello`的MPI程序。程序源代码`mpihello.c`内容如下：

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

#define MAX_HOSTNAME_LENGTH 256

int main(int argc, char *argv[])
{
    int pid;
    char hostname[MAX_HOSTNAME_LENGTH];

    int numprocs;
    int rank;

    int rc;

    /* Initialize MPI. Pass reference to the command line to
     * allow MPI to take any arguments it needs
     */
    rc = MPI_Init(&argc, &argv);

    /* It's always good to check the return values on MPI calls */
    if (rc != MPI_SUCCESS)
    {
        fprintf(stderr, "MPI_Init failed\n");
        return 1;
    }

    /* Get the number of processes and the rank of this process */
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* let's see who we are to the "outside world" - what host and what PID */
    gethostname(hostname, MAX_HOSTNAME_LENGTH);
    pid = getpid();

    /* say who we are */
    printf("Rank %d of %d has pid %5d on %s\n", rank, numprocs, pid, hostname);
    fflush(stdout);

    /* allow MPI to clean up after itself */
    MPI_Finalize();
    return 0;
}
```

2.1 将OpenMPI+GCC编译的程序提交到LSF

我们先尝试使用OpenMPI并行库和GCC编译器后端来构建程序：

```
$ module purge && module load openmpi/gcc/1.6.5 && mpicc mpi_hello.c -o hello_openmpi
```

为了验证程序的正确性，可先在登录节点上做小规模并行测试。注意：在登录节点上做并行测试，并行的核数请勿超过4核，执行时间不能超过15分钟。

测试运行需要准备`hosts.txt`，这个文件用来指定程序运行的主机。我们仅在本机做测试运行，因此machinefile内容只有`localhost`。

```
$ echo "localhost" > hosts.txt
```

`mpirun`用于启动MPI并行程序。下面的命令启动`mpihello`并行程序，分配4个线程。

```
$ module purge && module load openmpi/gcc/1.6.5 && mpirun -np 4 -machinefile
hosts.txt ./mpihello
```

下面这个作业脚本`hello_openmpi.lsf`用于向LSF正式提交作业。脚本申请32线程，分配到2个节点上运行：

```
#BSUB -L /bin/bash
#BSUB -J HELLO_MPI
#BSUB -n 32
#BSUB -e %J.err
#BSUB -o %J.out
#BSUB -R "span[ptile=16]"
#BSUB -q cpu

MODULEPATH=/lustre/utility/modulefiles:$MODULEPATH
module purge
module load openmpi/gcc/1.6.5

mpirun ./hello_openmpi
```

提交作业，作业运行结束后可查看out和err文件的内容。

```
$ dos2unix hello_openmpi.lsf && bsub < hello_openmpi.lsf
$ sleep 10 && bpeek
```

2.2 将Intel MPI套件编译的程序提交到LSF

用户也可以使用Intel MPI库和Intel编译器构建应用。注意，要调用`icc`而非gcc作为后端编译器，必须使用`mpiicc`。

```
$ module purge && module load icc/13.1.1 impi/4.1.1.036 && mpiicc hello_mpi.c
-o hello_intel
```

在本地使用4线程测试运行，`hosts.txt`文件只有一行内容：`localhost`。

```
$ module purge && module load icc/13.1.1 impi/4.1.1.036 && mpirun -np 4 -machinefile
hosts.txt ./hello_intel
```

用于正式作业提交的LSF脚本`hello_intel.lsf`如下，使用了2个节点共32线程：

```
#BSUB -q cpu
#BSUB -J HELLO_MPI
#BSUB -L /bin/bash
#BSUB -o %J.out
#BSUB -e %J.err
#BSUB -n 32
#BSUB -R "span[ptile=16]"

MODULEPATH=/lustre/utility/modulefiles:$MODULEPATH
module purge
module load icc/13.1.1
module load impi/4.1.1.036

mpirun ./hello_intel
```

提交作业，等待结果：

```
$ dos2unix hello_intel.lsf && bsub < hello_intel.lsf
$ sleep 10 && bpeek
```

3 CUDA示例

这部分演示如何编译NVIDIA CUDA程序，并提交到LSF的gpu队列中运行。注意：登录节点只有CUDA软件开发环境，没有CUDA硬件加速卡，因而不能在登录节点执行CUDA应用程序，必须把作业提交到LSF的gpu队列运行。

示例的CUDA源程序名为`cudahello.cu`，内容如下：

```
#include <stdio.h>

const int N = 7;
const int blocksize = 7;

__global__
void hello(char *a, int *b)
{
 a[threadIdx.x] += b[threadIdx.x];
}

int main()
{
 char a[N] = "Hello ";
 int b[N] = {15, 10, 6, 0, -11, 1, 0};

 char *ad;
 int *bd;
 const int csize = N*sizeof(char);
 const int isize = N*sizeof(int);

 printf("%s", a);

 cudaMalloc( (void**)&ad, csize );
 cudaMalloc( (void**)&bd, isize );
 cudaMemcpy( ad, a, csize, cudaMemcpyHostToDevice );
 cudaMemcpy( bd, b, isize, cudaMemcpyHostToDevice );

 dim3 dimBlock( blocksize, 1 );
 dim3 dimGrid( 1, 1 );
 hello<<<dimGrid, dimBlock>>>(ad, bd);
 cudaMemcpy( a, ad, csize, cudaMemcpyDeviceToHost );
 cudaFree( ad );

 printf("%s\n", a);
 return EXIT_SUCCESS;
}
```

我们使用NVIDIA CUDA SDK提供的`nvcc`编译这段代码。

```
module purge && module load cuda/5.5 && nvcc cudahello.cu -o cudahello
```

由于登录节点没有安装CUDA加速卡，因此不能运行CUDA程序。CUDA程序必须提交到LSF作业管理系统的gpu队列运行。由于LSF对GPU卡的资源管理不够完善，因此指定资源时需要“折算”成相应的CPU数量。譬如，某个计算任务需要使用2块GPU卡(1个GPU节点有2块GPU卡)，则在作业脚本中应该申请1台GPU节点，合16核CPU。用于提交单节点CUDA作业的LSF作业控制脚本`cudahello.lsf`内容如下。

```
#BSUB -q gpu
#BSUB -J HELLO_CUDA
#BSUB -L /bin/bash
#BSUB -o %J.out
#BSUB -e %J.err
#BSUB -n 16
#BSUB -R "span[ptile=16]"

MODULEPATH=/lustre/utility/modulefiles:$MODULEPATH
module load cuda/5.5

./cudahello
```

将CUDA作业提交到gpu队列上：

```
$ dos2unix cudahello.lsf && bsub -q gpu < cudahello.lsf
$ sleep 10 && bpeek
```

4 参考资料

- “LLNL Tutorials: Message Passing Interface (MPI)” **https://computing.llnl.gov/tutorials/mpi/**
- “mpihello by ludwig Luis Armendariz” **https://github.com/ludwig/examples**
- “How to compile and run a simple CUDA Hello World” **http://www.pdc.kth.se/resources/computers/zorn/how-to/how-to-compile-and-run-a-simple-cuda-hello-world**