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

目录

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:

```
$ bsub < ompgcc.lsf
```

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：

```
bsub < ompintel.lsf && bjobs
```

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。

```
$ bsub < hello_openmpi.lsf && bjobs
```

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 ./h
```

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：

```
$ bsub < hello_intel.lsf && bjobs
```

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：

```
$ bsub -q gpu < cudahello.lsf
```

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