

并行程序示例

上海交通大学高性能计算中心

<http://hpc.sjtu.edu.cn>

2013 年 11 月 8 日更新

目录

| | | |
|----------|---|----------|
| 1 | OpenMP 示例 | 2 |
| 2 | MPI 示例 | 2 |
| 2.1 | 编译源代码 | 2 |
| 2.2 | 使用 <code>mpirun</code> 在本地测试运行 | 3 |
| 2.3 | 提交到 LSF 作业管理系统 | 4 |
| 2.4 | 使用 <code>module</code> 简化环境设置 | 5 |
| 3 | CUDA 示例 | 6 |
| 3.1 | 编译源代码 | 6 |
| 3.2 | 提交到 LSF 作业系统的 <code>gpu</code> 队列运行 | 7 |
| 3.3 | 使用 <code>module</code> 简化环境设置 | 8 |
| 4 | 参考资料 | 9 |

1 OpenMP 示例

2 MPI 示例

这部分演示如何使用 Intel 编译器编译一个名为`mpihello`的 MPI 程序，然后在本地测试运行，最后向 LSF 作业管理系统提交正式作业。使用 Environment Module 可以简化环境参数设定，我们也会给出使用 module 的过程。

2.1 编译源代码

`mpihello`程序的源文件为`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;
}
```

我们使用 Intel MPI 并行开发环境提供的 `mpicc` 对其编译, `mpicc` 会自动调用后端编译器 `icc`。使用 `icc` 前需要运行脚本使环境变量生效。

```
$ source /lustre/utility/intel/composer_xe_2013.3.163/bin/compilervars.sh intel64
$ source /lustre/utility/intel/mkl/bin/intel64/mklvars_intel64.sh
$ source /lustre/utility/intel/impi/4.1.1.036/bin64/mpivars.sh
$ mpicc -o mpihello mpihello.c
```

2.2 使用 `mpirun` 在本地测试运行

测试运行需要准备 `machinefile`, 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" > machinefile.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。注意: 在 π 集群中, `mpirun` 只能用作小于 4 线程的测试, 大规模正式作业必须提交到 *LSF*。

```
$ mpirun -np 4 -machinefile machinefile.txt ./mpihello
Rank 0 of 4 has pid 90037 on mu05
Rank 1 of 4 has pid 90038 on mu05
Rank 2 of 4 has pid 90039 on mu05
Rank 3 of 4 has pid 90040 on mu05
```

2.3 提交到 LSF 作业管理系统

大规模正式作业必须提交到 LSF 作业管理系统。

提交的作业由 LSF 作业管理系统统一调度运行, 用户不需要手动书写 machinefile。作业控制脚本 mpihello.lsf 内容如下:

```
#BSUB -J HELLO_MPI
#BSUB -o job.out
#BSUB -e job.err
#BSUB -n 256

source /lustre/utility/intel/composer_xe_2013.3.163/bin/compilervars.sh intel64
source /lustre/utility/intel/mkl/bin/intel64/mklvars_intel64.sh
source /lustre/utility/intel/impi/4.1.1.036/bin64/mpivars.sh

MPIRUN=`which mpirun`
EXE="./mpihello"

cat /dev/null > nodelist

for host in `echo $LSB_MCPU_HOSTS | sed -e 's/ /:/g' | sed 's/:n/\nn/g'`
do
echo $host >> nodelist
done

$MPIRUNP -machinefile nodelist $EXE
```

将作业提交到名为 cpu 的队列上:

```
$ bsub -q cpu < mpihello.lsf
```

作业运行结束后, 在工作目录下前查看输出结果 job.out、job.err。

2.4 使用 module 简化环境设置

使用 Environment Module 可以简化环境变量的书写，过程如下。

1. 编译源程序。

```
$ module load icc/13.1.1
$ module load mkl/11.0.3
$ module load impi/4.1.1.036
$ mpiicc -o mpihello mpihello.c
```

2. 在本地测试运行。

```
$ mpirun -np 4 ./mpihello
```

3. 提交到 LSF 作业管理系统，所用的作业控制脚本如下。注意，增加了 *BSUB -L*、*MODULEPATH*、*module* 若干行。

```
#BSUB -J HELLO_MPI
#BSUB -L /bin/bash
#BSUB -o job.out
#BSUB -e job.err
#BSUB -n 256

MODULEPATH=/lustre/utility/modulefiles:$MODULEPATH
module load icc/13.1.1
module load mkl/11.0.3
module load impi/4.1.1.036

MPIRUN=`which mpirun`
EXE="./mpihello"

cat /dev/null > nodelist

for host in `echo $LSB_MCPU_HOSTS | sed -e 's/ /:/g' | sed 's/:n/\nn/g'`
do
echo $host >> nodelist
done

$MPIRUN -machinefile nodelist $EXE
```

提交作业:

```
$ busb -q cpu < mpihello.lsf
```

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。

3.1 编译源代码

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` 对其编译。使用 `nvcc` 前需要设置一些环境变量。

```
$ export PATH=/lustre/utility/cuda-5.0/bin/:$PATH
$ export C_INCLUDE_PATH=/lustre/utility/cuda-5.0/include/:$C_INCLUDE_PATH
$ export CPLUS_INCLUDE_PATH=/lustre/utility/cuda-5.0/include/:$CPLUS_INCLUDE_PATH
$ export LIBRARY_PATH=/lustre/utility/cuda-5.0/lib64/:$LIBRARY_PATH
$ export LD_LIBRARY_PATH=/lustre/utility/cuda-5.0/lib64/:$LD_LIBRARY_PATH
$ nvcc -o cudahello cudahello.cu
```

3.2 提交到 LSF 作业系统的 gpu 队列运行

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。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 -J HELLO_CUDA
#BSUB -L /bin/bash
#BSUB -o job.out
#BSUB -e job.err
#BSUB -n 1

export PATH=/lustre/utility/cuda-5.0/bin/:$PATH
```

```
export C_INCLUDE_PATH=/lustre/utility/cuda-5.0/include/:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=/lustre/utility/cuda-5.0/include/:$CPLUS_INCLUDE_PATH
export LIBRARY_PATH=/lustre/utility/cuda-5.0/lib64/:$LIBRARY_PATH
export LD_LIBRARY_PATH=/lustre/utility/cuda-5.0/lib64/:$LD_LIBRARY_PATH

./cudahello
```

将 CUDA 作业提交到gpu队列上:

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

作业运行结束后, 在工作目录下查看输出结果job.out、job.err。

3.3 使用 module 简化环境设置

使用 Environment Module 可以简化环境变量的书写, 过程如下。

1. 编译源程序。

```
$ module load cuda/5.0
$ nvcc -o cudahello cudahello.cu
```

2. 提交到 LSF 的 gpu 队列, 使用的作业脚本为cuda.lsf, 内容如下:

```
#BSUB -J HELLO_CUDA
#BSUB -L /bin/bash
#BSUB -o job.out
#BSUB -e job.err
#BSUB -n 1

MODULEPATH=/lustre/utility/modulefiles:$MODULEPATH
module load cuda/5.0

./cudahello
```

提交到 gpu 队列:

```
$ bsub -q gpu < cuda.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>