

Ubuntu18.04安装和使用GPGPU-Sim_SharedMemory

Ubuntu18或16均可，主要是cuda和gcc、g++版本要安装正确。

Ubuntu18.04安装和使用GPGPU-Sim_SharedMemory

- 一、下载安装NVIDIA CUDA 4.0
- 二、下载和运行GPGPU-Sim
- 三、运行ispass-2009 benchmarks
- 四、popnet的使用
- 五、workspace文件夹的使用
- 六、多机应用矩阵乘 (MM) 示例

一、下载安装NVIDIA CUDA 4.0

1. 下载ubuntu linux 10.10 cuda toolkit和GPU Computing SDK code samples

[https://developer.nvidia.com/cuda-toolkit-40\](https://developer.nvidia.com/cuda-toolkit-40)

GPGPU-Sim只支持到cuda 4

2. 安装CUDA toolkit

```
1 | chmod +x cudatoolkit_4.0.17_linux_64_ubuntu10.10.run
2 | sudo ./cudatoolkit_4.0.17_linux_64_ubuntu10.10.run
```

默认安装在/usr/local/cuda，不用管他，直接enter。

3. 增加CUDA toolkit到 ~/.bashrc中，添加环境变量

.bashrc在根目录下，是隐藏文件，按control+H可看到

```
1 | echo 'export PATH=$PATH:/usr/local/cuda/bin' >> ~/.bashrc
2 | echo 'export
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib:/usr/local/cuda/lib64'
   >> ~/.bashrc
3 | source ~/.bashrc
```

可用vim查看：

```
1 | sudo vim ~/.bashrc
```

底部两行已加入路径即可。

4. 安装GPU Computing SDK code samples

```
1 | chmod +x gpucomputingsdk_4.0.17_linux.run
2 | sudo ./gpucomputingsdk_4.0.17_linux.run
```

默认安装在~/NVIDIA_GPU_Computing_SDK路径中，直接enter。

5.安装gcc-4.4和g++4.4(CUDA 4.0只支持gcc版本到4.4)

```
1 | sudo apt-get install gcc-4.4 g++-4.4
```

由于Ubuntu 18.04自带7.4.0版本gcc，所以无法安装，可通过以下方法修改：

```
1 | sudo vim /etc/apt/sources.list
```

底部插入两行代码：

```
1 | deb http://dk.archive.ubuntu.com/ubuntu/ trusty main universe
2 | deb http://dk.archive.ubuntu.com/ubuntu/ trusty-updates main universe
```

添加好后，保存退出。

更新apt源：

```
1 | sudo apt-get update
```

再重新安装gcc-4.4和g++-4.4即可。

```
1 | sudo apt-get install gcc-4.4 g++-4.4
```

6.改变系统中的gcc/g++为gcc-4.4/g++4.4

```
1 | sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-7 150
2 | sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.4 100
3 | sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-7 150
4 | sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.4 100
```

用update-alternatives选择4.4版本：

```
1 | sudo update-alternatives --config gcc
```

输入4.4版本前面对应的序号，然后enter。

二、下载和运行GPGPU-Sim

1.从GitHub下载GPGPU-Sim

```
1 | sudo apt-get install git
2 | git clone https://github.com/gpgpu-sim/gpgpu-sim_distribution.git
```

2.安装依赖

```

1 | sudo apt-get install build-essential xutils-dev bison zlib1g-dev flex
   | libglu1-mesa-dev
2 | sudo apt-get install doxygen graphviz
3 | sudo apt-get install python-pmw python-ply python-numpy libpng12-dev python-
   | matplotlib
4 | sudo apt-get install libxi-dev libxmu-dev freeglut3-dev

```

3.添加CUDA_INSTALL_PATH到~/.bashrc中

```

1 | echo 'export CUDA_INSTALL_PATH=/usr/local/cuda' >> ~/.bashrc
2 | source ~/.bashrc

```

4.编译GPGPU_Sim

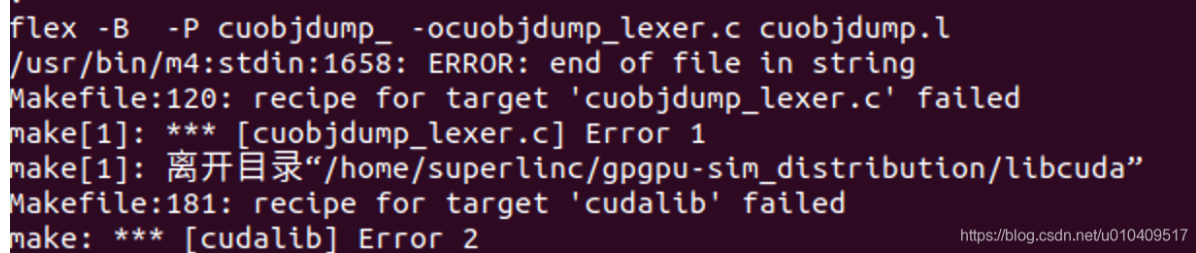
在编译之前，先将[Chiplet GPGPU-Sim SharedMemory](#)文件夹下的Source Code文件夹重命名为SourceCode，如果路径有空格可能会影响后续操作。

```

1 | source setup_environment
2 | make
3 | make docs

```

如果make结束出现错误，如图2-1



```

flex -B -P cuobjdump_ -ocuobjdump_lexer.c cuobjdump.l
/usr/bin/m4:stdin:1658: ERROR: end of file in string
Makefile:120: recipe for target 'cuobjdump_lexer.c' failed
make[1]: *** [cuobjdump_lexer.c] Error 1
make[1]: 离开目录"/home/superlinc/gpgpu-sim_distribution/libcuda"
Makefile:181: recipe for target 'cudalib' failed
make: *** [cudalib] Error 2

```

<https://blog.csdn.net/u010409517>

图2-1 编译gpgpu-sim出现错误

移除cuobjdump.l:109-111行，再make就不会出现错误了。

5.运行GPGPU_Sim

在gpgpu-sim_distribution-master下运行cuda程序，

程序示例：

```

1 | #include "cuda_runtime.h"
2 | #include "device_launch_parameters.h"
3 | #include <stdio.h>
4 |
5 | __global__ void kernel(void) {
6 |
7 | }
8 |
9 | int main() {
10 |
11 |     kernel << <1, 1 >> > ();
12 |     printf("Hello world!\n");
13 |     return 0;
14 |
15 | }

```

保存为hello.cu格式。

终端运行：

```
1 | nvcc hello.cu -o hello.out
```

生成一个hello.out文件

```
1 | ./hello.out
```

但还不能运行GPGPU_Sim，要将/configs/GTX480文件夹下的三个文件都复制到程序中，如图2-1所示

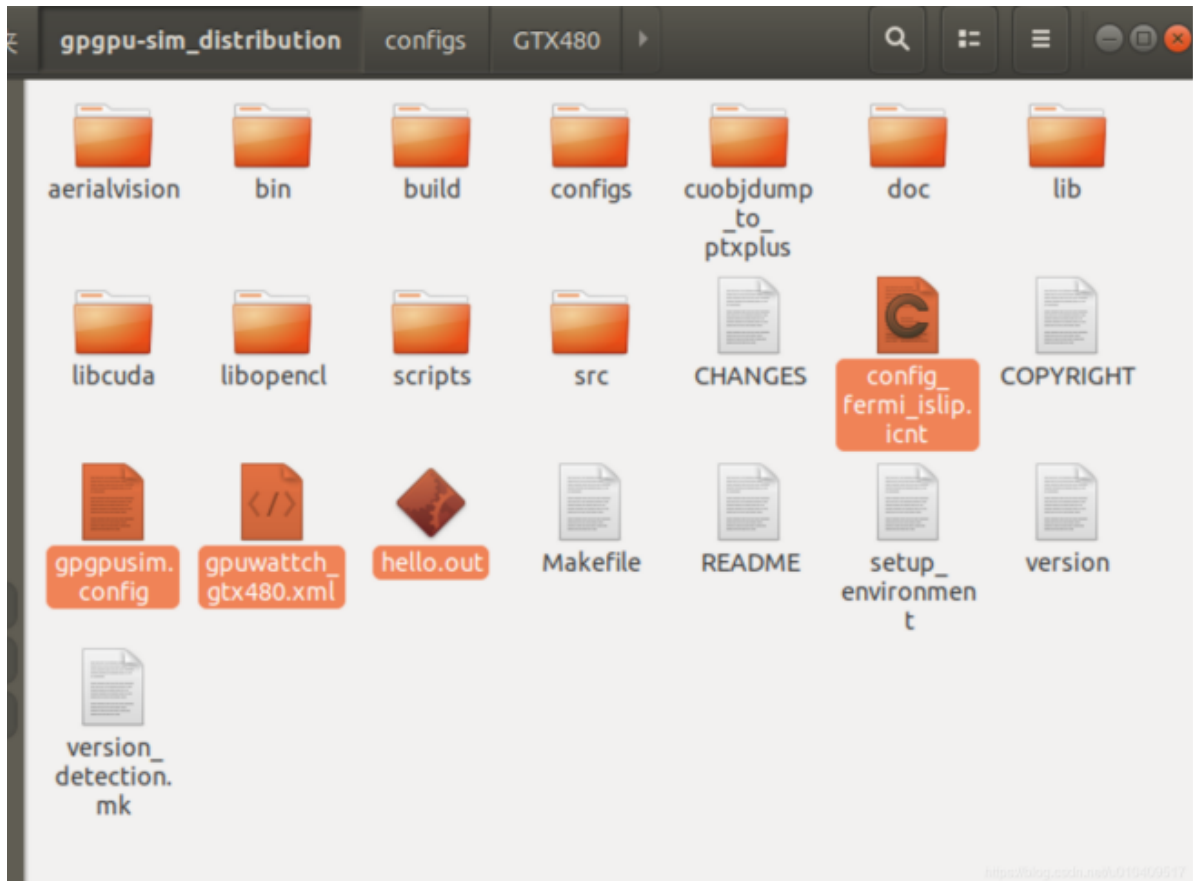


图2-2 运行GPGPU_Sim所需的文件

在此路径中运行：

```
1 | source setup_environment
```

会发现出现一大堆信息，最后可以看到运行时间，速率等信息，以及最后的输出。至此，GPGPU_Sim安装运行完毕。

三、运行ispass-2009 benchmarks

进入[Chiplet GPGPU-Sim SharedMemory/benchmark/](#)，并给文件夹内的文件赋予可执行权限。

```
1 | sudo chmod =R 777 ispass2009-benchmarks-master/  
2 | cd ispass2009-benchmarks-master/
```

1.编译ispass2009-benchmarks

打开Makefile.ispass-2009文件在文件开头添加：

```
1 | CUDA_INSTALL_PATH=/usr/local/cuda
2 | NVIDIA_COMPUTE_SDK_LOCATION=/[PATH]/NVIDIA_GPU_Computing_SDK
```

[PATH] 替换为安装 CUDA 过程中个人自定义的 CUDA SDK 路径。

比如我的路径为：

```
1 | NVIDIA_COMPUTE_SDK_LOCATION=/home/fj5/NVIDIA_GPU_Computing_SDK
```

然后用编辑器打开Makefile.ispass-2009文件，修改OPENMPI_BINDIR路径：

```
1 | change
2 | export OPENMPI_BINDIR=/usr/lib64/mpi/gcc/openmpi/bin/;
3 | to
4 | export OPENMPI_BINDIR=/usr/bin/;
```

保存并退出。

2.AES 错误修正

进入AES目录下，用编辑器打开Makefile文件修改LINKFLAGS。

```
1 | change
2 | LINKFLAGS      := -L$(BOOST_LIB) -lboost_filesystem$(BOOST_VER)
3 | to
4 | LINKFLAGS      := -L$(BOOST_LIB) -lboost_filesystem$(BOOST_VER) -
  | lboost_system
```

3.DG 错误修正

进入DG目录下，用编辑器打开Makefile文件，找到第54和56行INCLUDES：

```
1 | // line 54
2 | change
3 | INCLUDES = -Dp_N=$(N) -DNDG3d -DCUDA -I/opt/local/include -
  | I/usr/include/malloc -I$(HDRDIR) -I/opt/mpich/include
4 | to
5 | INCLUDES = -Dp_N=$(N) -DNDG3d -DCUDA -I/opt/local/include -
  | I/usr/include/malloc -I$(HDRDIR) -I/opt/mpich/include -
  | I/usr/lib/openmpi/include
6 |
7 | // line 56
8 | change
9 | INCLUDES = -Dp_N=$(N) -DNDG3d -DCUDA -I/opt/local/include -
  | I/usr/include/malloc -I$(HDRDIR)
10 | to
11 | INCLUDES = -Dp_N=$(N) -DNDG3d -DCUDA -I/opt/local/include -
  | I/usr/include/malloc -I$(HDRDIR) -I/usr/lib/openmpi/include
```

4.WP 错误修正

进入WP根目录，用编辑器打开makefile文件，修改第75行。

```

1 // line 75
2 change
3 NVOPT = $(DEVICEEMU_NVCC) $(PROMOTE) $(DEBUGDEBUG) $(DEBUGOUTPUT) \
4         -DXXX=$(XXX) -DYYY=$(YYY) -DMKX=$(MKX) --host-compilation
5 'C++' --use_fast_math
6 to
7 NVOPT = $(DEVICEEMU_NVCC) $(PROMOTE) $(DEBUGDEBUG) $(DEBUGOUTPUT) \
         -DXXX=$(XXX) -DYYY=$(YYY) -DMKX=$(MKX) --use_fast_math

```

然后找到GPGPULINK:

```

1 change
2 #GPGPULINK = -L$(CUAHOME)/lib64/ -lcudart -
3 L$(NVIDIA_COMPUTE_SDK_LOCATION)/C/lib/ -lcutil_x86_64 -lm -lz -ldl -lGL -
4 lstdc++ $(NEWLIBDIR) $(LIB) # /usr/lib64/libstdc++.so.6
5 to
6 GPGPULINK = -L$(CUAHOME)/lib64/ -lcudart -
7 L$(NVIDIA_COMPUTE_SDK_LOCATION)/C/lib/ -L$(NVIDIA_COMPUTE_SDK_LOCATION)/C/lib
8 -lm -lz -ldl -lGL -lstdc++ $(NEWLIBDIR) $(LIB) # /usr/lib64/libstdc++.so.6

```

5.执行编译命令:

```
1 make -f Makefile.ispass-2009
```

如果修改错误后还是编译不了的,就注释掉:

```

1 #$(SETENV) make noinline=$(noinline) -C AES
2 #$(SETENV) make noinline=$(noinline) -C DG/3rdParty/ParMetis-3.1
3 #$(SETENV) make noinline=$(noinline) -C DG
4 #$(SETENV) make noinline=$(noinline) -C WP

```

编译生成的二进制文件在.../bin/release/中。

6.链接GPU配置文件

```

1 cd /home/gpgpu-sim_distribution
2 source setup_environment
3 cd ispass2009-benchmarks/
4 ./setup_config.sh GTX480

```

7.运行基准测试, 比如NN

```

1 cd NN/
2 sh README.GPGPU-Sim

```

四、popnet的使用

1.对popnet进行编译

```

1 cd SourceCode/popnet-master
2 make

```

2.通信延迟的计算

在每次gpgpu-sim进行片外dram访问时，都在popnet发一个包，就当作往其他芯粒发包。即仿真器会于trace文件夹中的文件bench.src_x.src_y在添加一条trace记录。包的格式是：T sx sy dx dy n，

T: 表示数据包发出时的时间

sx sy: 表示数据源Chiplet的地址

dx dy: 表示数据目标Chiplet的地址

n: 表示包大小

在程序运行完成之后，将所有bench.*.*文件中的trace记录合并，并根据数据包发出时间进行排序，生成文件命名为bench并置于本目录下。然后进入popnet所在目录，执行指令得到通信延迟。

```
1 | ./popnet -A 9 -c 2 -v 3 -B 12 -O 12 -F 4 -L 1000 -T 20000 -r 1 -I ./address -R 0
```

各参数含义详见popnet文件夹下的README文件。

五、workspace文件夹的使用

先编译好GPGPU-Sim和popnet，然后再用compile.sh编译workspace文件夹下的几个文件，starter.c文件是作为启动各个benchmark的文件，controller.c/controller2.c是作为通信协议。然后就使用命令：

```
1 | ./S AES BFS MM mesh
```

前面几个是benchmark（AES,BFS,MM），最后一个拓扑结构。

六、多机应用矩阵乘（MM）示例

1.仿真器系统运行架构

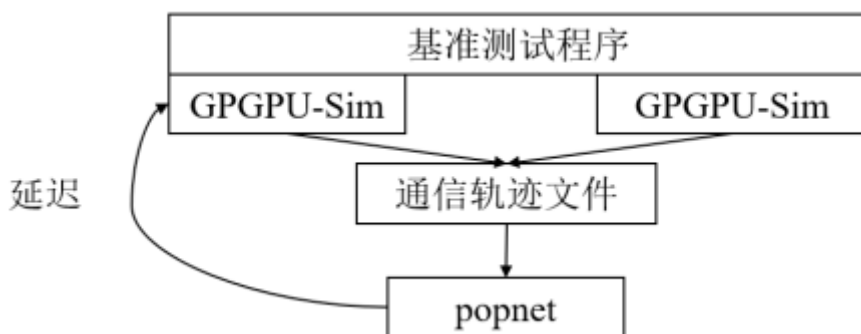


图6-1 系统架构

仿真器运行的架构如图6-1，基准测试程序作为应用运行在多个GPGPU-Sim上，GPGPU-Sim产生trace，输入到popnet，popnet反馈延迟给GPGPU-Sim。

2.矩阵乘编译与运行

$$A = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} B = \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

$$\begin{aligned} A \times B &= A_{00} \times B_{00} + A_{01} \times B_{10} \\ &\quad + A_{00} \times B_{01} + A_{01} \times B_{11} \\ &\quad + A_{10} \times B_{00} + A_{11} \times B_{10} \\ &\quad + A_{10} \times B_{01} + A_{11} \times B_{11} \end{aligned}$$

图6-2 矩阵乘示例图

```
1 | cd Chiplet_GPGPU-Sim_SharedMemory/benchmark/ispass2009-benchmarks-master/MM
```

如图6-2所示，假设矩阵A和B大小400×400，则将mm.cu中的第9、10行设为400

```
1 | vim mm.cu
2 | //line 9 10
3 | #define Row 400
4 | #define Col 400
```

运行mm.cu得到等式左边乘法的运行时间，输出文件nohup.out在MM目录下。

```
1 | cd Chiplet_GPGPU-Sim_SharedMemory/SourceCode/workspace/
2 | sh compile.sh
3 | ./S MM mesh
```

然后由图6-2所示，容易得到A₀₀、B₀₀等小矩阵的大小为200×200，则则将mm.cu中的第9、10行设为200

```
1 | cd Chiplet_GPGPU-Sim_SharedMemory/benchmark/ispass2009-benchmarks-master/MM
2 | vim mm.cu
3 | //line 9 10
4 | #define Row 200
5 | #define Col 200
```

假设开启两个gpgpusim，则每个gpgpusim需要运行四次200×200的矩阵乘法，一个gpgpusim运行三次200×200矩阵加法，一个gpgpusim运行四次200×200矩阵加法。矩阵加法函数如下：


```
1  __global__ void matrix_add_gpu(int *M,int *N,int *P,int width)
2  {
3      int i = threadIdx.x + blockDim.x * blockIdx.x;
4      int j = threadIdx.y + blockDim.y * blockIdx.y;
5
6      if (i < width && j < width)
7      {
8          P[i][j] = M[i][j] + N[i][j]
9      }
10 }
```

运行方法同上，结束后将两个gpgpusim的总耗时相加，得到等式右边计算花费的时间，与等式左边计算花费的时间做对比。

参考资料：

1. <https://blog.csdn.net/u010409517/article/details/91050129>
2. <https://learnku.com/articles/39866>