加入predictor的数据处理

笔记本：	日常		
创建时间：	2019/8/2 23:13	**更新时间：**	2019/8/14 21:35
作者：	yizhi		
URL：	https://vimsky.com/article/3606.html		

加入predictor的数据处理

0.实验环境系统

Windows，Linux系统皆可。（Linux下已经写好makefile可以直接进行编译，Windows下需要再编译一次）

1.目录

为了使目录文件更加简洁，我新建了一个data文件夹，用来存放能量分级数据和input power trace。最后的输出结果一会存到相应的文件夹内，而不是保存在根目录。同时所有的源数据文件都需要存在data里面

data

data内部的命名规则与之前一样。

lenet_solar

lenet_tvrf

predictor

这里新加入了predictor文件夹，用来存放predictor的分级数据以及input power trace。

lenet_piezo

lenet_solar

内部的命名方式依旧与之前相同。

```
+data
|     |
|     +--lenet_solar
|     |
|     +--lenet_tvrf
|     |
|     +--predictor
|     |     |
|     |     +--lenet_piezo
|     |     |
|     |     +--lenet-solar
```

文件夹结构如图所示。

2.处理数据

(1) 首先打开能量分级文件

因为，延迟是固定的80ns，程序不需要再次输入，首先删除power consumption所在的列。 

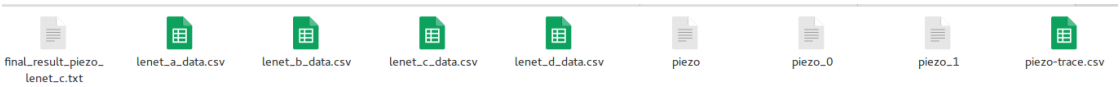
随后，复制所示区域（不需要第一列的行号），将区域中的数据粘贴到对应的csv文件中（这里是lenet_c_data.csv）。用文本编辑器打开，如图所示： 

其他网络其他策略的数据也用同样的方式进行处理。这样，需要的文件格式就整理好了。可以开始运行程序。

*PS: d策略有些不同，除了每一层的功耗还有最后一列的power pipeline，这里要将最后一列的值拷贝到第每一层的power consumption上。如图所示： 

除了第一列和最后一列，复制到相应的csv文件里，余下部分与abc相同。

2.1. predictor的处理数据

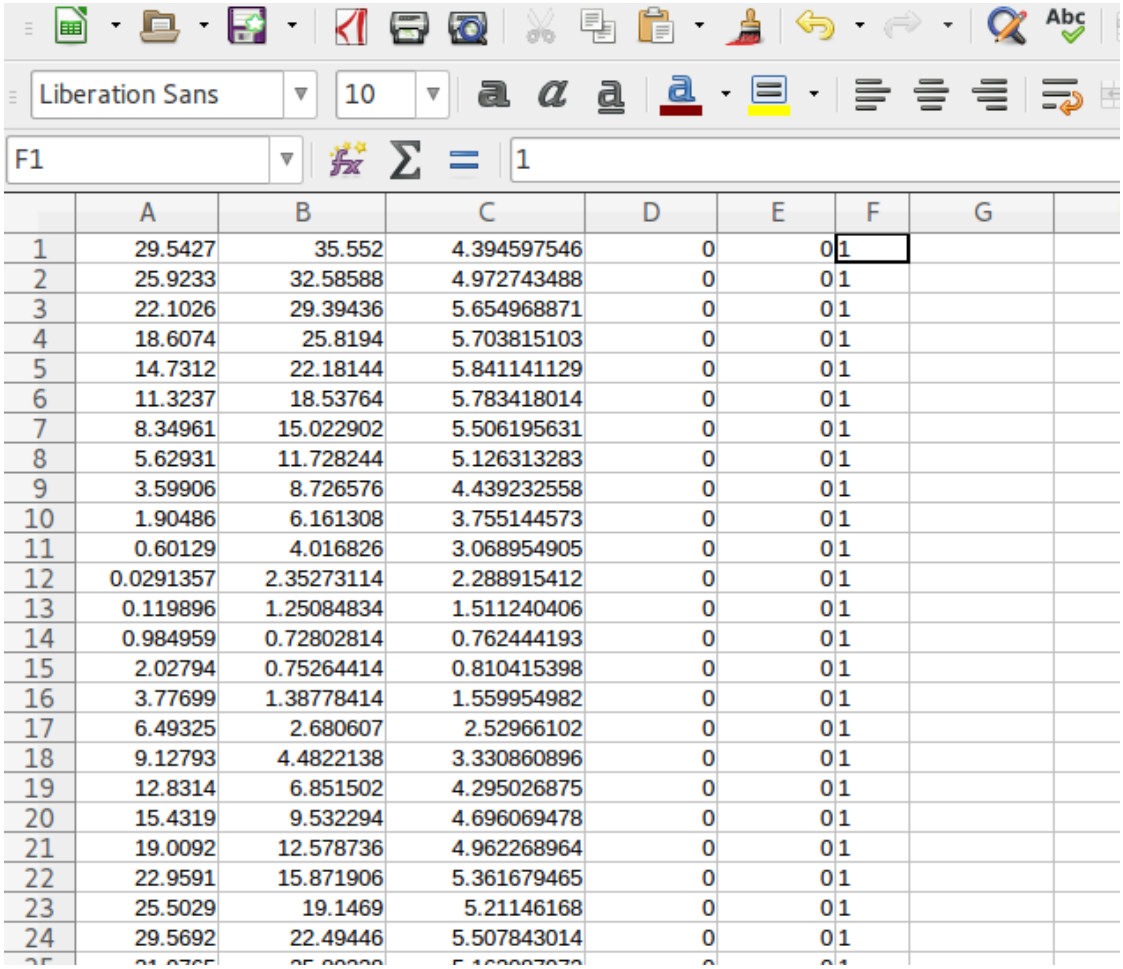


上图为lenet_piezo文件夹内的文件，可以看到，最终的结果也保存在这里。

首先处理predictor的能量数据。

	A	B	C	D	E	F	G
1	energy level	average power	variation of power	target level	infer level	judge	
2	29.5427	35.552	4.394597546	0	0	true	
3	25.9233	32.58588	4.972743488	0	0	true	
4	22.1026	29.39436	5.654968871	0	0	true	
5	18.6074	25.8194	5.703815103	0	0	true	
6	14.7312	22.18144	5.841141129	0	0	true	
7	11.3237	18.53764	5.783418014	0	0	true	
8	8.34961	15.022902	5.506195631	0	0	true	
9	5.62931	11.728244	5.126313283	0	0	true	
10	3.59906	8.726576	4.439232558	0	0	true	
11	1.90486	6.161308	3.755144573	0	0	true	
12	0.60129	4.016826	3.068954905	0	0	true	
13	0.0291357	2.35273114	2.288915412	0	0	true	
14	0.119896	1.25084834	1.511240406	0	0	true	
15	0.984959	0.72802814	0.762444193	0	0	true	
16	2.02794	0.75264414	0.810415398	0	0	true	
17	3.77699	1.38778414	1.559954982	0	0	true	
18	6.49325	2.680607	2.52966102	0	0	true	
19	9.12793	4.4822138	3.330860896	0	0	true	
20	12.8314	6.851502	4.295026875	0	0	true	
21	15.4319	9.532294	4.696069478	0	0	true	
22	19.0092	12.578736	4.962268964	0	0	true	
23	22.9591	15.871906	5.361679465	0	0	true	
24	25.5029	19.1469	5.21146168	0	0	true	
25	29.5692	22.49446	5.507843014	0	0	true	
26	31.9765	25.80338	5.163087972	0	0	true	
27	34.4494	28.89142	4.678213719	0	0	true	
28	35.9832	31.49624	4.144822711	0	0	true	

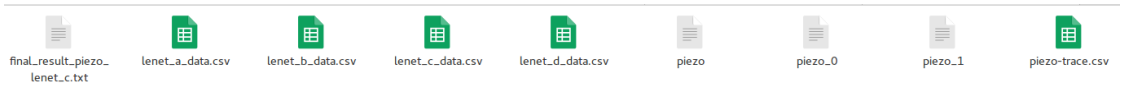
可以看到predictor有两列能量数据，我们只需要第一列和最后一列。
首先将最后一列的true替换成1，将false替换成0。然后删除第一行。



	A	B	C	D	E	F	G
1	29.5427	35.552	4.394597546	0	0	1	
2	25.9233	32.58588	4.972743488	0	0	1	
3	22.1026	29.39436	5.654968871	0	0	1	
4	18.6074	25.8194	5.703815103	0	0	1	
5	14.7312	22.18144	5.841141129	0	0	1	
6	11.3237	18.53764	5.783418014	0	0	1	
7	8.34961	15.022902	5.506195631	0	0	1	
8	5.62931	11.728244	5.126313283	0	0	1	
9	3.59906	8.726576	4.439232558	0	0	1	
10	1.90486	6.161308	3.755144573	0	0	1	
11	0.60129	4.016826	3.068954905	0	0	1	
12	0.0291357	2.35273114	2.288915412	0	0	1	
13	0.119896	1.25084834	1.511240406	0	0	1	
14	0.984959	0.72802814	0.762444193	0	0	1	
15	2.02794	0.75264414	0.810415398	0	0	1	
16	3.77699	1.38778414	1.559954982	0	0	1	
17	6.49325	2.680607	2.52966102	0	0	1	
18	9.12793	4.4822138	3.330860896	0	0	1	
19	12.8314	6.851502	4.295026875	0	0	1	
20	15.4319	9.532294	4.696069478	0	0	1	
21	19.0092	12.578736	4.962268964	0	0	1	
22	22.9591	15.871906	5.361679465	0	0	1	
23	25.5029	19.1469	5.21146168	0	0	1	
24	29.5692	22.49446	5.507843014	0	0	1	

然后就变成的上图的样子，现在我们将第一行复制到“能量名称” + "_0"（这里是piezo_0），将第二行复制到“能量名称” + "_1"（这里是piezo_1）

下面就是处理完成的样子。



lenet_b_data.csv 等文件与之前的处理方法一样，不再赘述。
第一个文件final_result_piezo_lenet_c.txt就是实验得出的结果（之前结果都是保存在根目录下的，为了保持根目录的简洁，我把最后的结果移到了相对应的文件夹内）。

predictor的数据都要放在predictor的文件夹下

4. 运行

因为写好了脚本，直接运行脚本即可。
名为run_acc.sh

```
Search   Tools   Documents   Help
NET_TYPE="pv"
SCHEME="a"
ENG_SRS="piezo"
#mkdir "exec_"$NET_TYPE
#cd "exec_"$NET_TYPE
#cp -rf ../accelerator .


echo "Starting Exec"

printf "thermal y c lenet" | ./accelerator
printf "thermal y c pv" | ./accelerator
printf "thermal y c fr" | ./accelerator
printf "thermal y c hg" | ./accelerator
echo "Done Exec solar"

printf "piezo y c lenet" | ./accelerator
printf "piezo y c pv" | ./accelerator
printf "piezo y c fr" | ./accelerator
printf "piezo y c hg" | ./accelerator
echo "Done Exec piezo"



echo "Ececution Done for a"
```

内容如上图所示。
printf为程序的输入，
第一列代表所用到的energy trace， 第二列代表是否使用predictor，第三列代表使用的策略，最后一列为当前所模拟的网络。

abc,d,e几种方式的输入略有不同，但都可以在终端输出的结果或是脚本里看到

4.1. 运行可执行文件

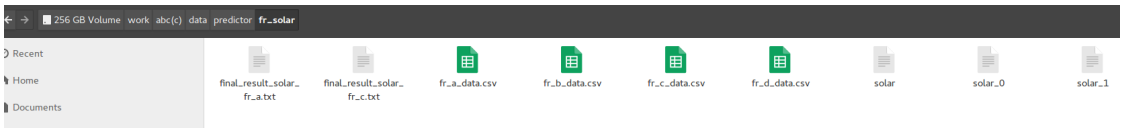
如果想单独运行程序，可直接运行可执行文件accelerator
运行后会出现要求输入power trace， 这里输入tvrf。
输入是否需要使用predictor。

```
✔ abc(c) ./accelerator
input trace file name (piezo, solar, thermal, tvrf, wifi-h, wifi-o):tvrf
trace file name is:tvrf
predictor? (y or n):
```

随后，需要输入运行策略。这里的例子是abc的例子，只能运行abc策略，d和e的在其他程序里，大体步骤一样。
输入网络名。
结束。

4. 结果

最后的输出结果都保存在对应策略对应能量对应网络下的文件夹里。



如图所示，前两个文件就是最终的实验结果。

详细的实验数据和最终的结果都在最后一行，如图所示。

total work times:	11582611	total middle work times:	838	addition work time:	0	c work time:	11582611
input power larger than83.3times and proportion:	1575	0.875	power utilization(cal):	0.678627	power utilization(trans):	1	power utilization(total): 0.839314
power utilization(cal when input power larger than threshold):	0.678627	power utilization(total when input power larger than threshold):	0.839314				
throughput(per sec):	11634089271	throughput(per J):	1250586918				

5.统计结果代码

```
    //final result
    statis_total_times          = power_cycle * power_space;
    statis_total_input_energy = statis_total_input_energy * statis_total_times / 1000000;
    statis_thr_per_sec          = network_cal_times * statis_total_work_times / statis_total_times;
    statis_thr_per_J            = network_cal_times * statis_total_work_times / statis_total_input_energy;

    std::ofstream outfile;
    outfile.open(final_file, std::ios::app);
    outfile << "total work times: ," << statis_total_work_times
            << "," << "total middle work times: ," << statis_total_mid_work_times
            << "," << "addition work time: ," << statis_total_add_work_times
            << "," << "c work time: ," << statis_total_c_work_times << std::endl;
    outfile << "input power larger than " << threshold << " times and proportion: ," << statis_total_large_enr_times
            << "," << statis_total_large_enr_times / power_cycle
            << "," << "power utilization(cal): ," << statis_total_enr_rate_cal / statis_total_enr_rate_con
            << "," << "power utilization(trans): ," << statis_total_enr_rate_trans
            << "," << "power utilization(total): ," << statis_total_enr_rate / statis_total_enr_rate_con << std::endl;
    outfile << "power utilization(cal when input power larger than threshold): ,"
            << statis_total_enr_rate_cal / statis_total_large_enr_times
            << "," << "power utilization(total when input power larger than threshold): ,"
            << statis_total_enr_rate / statis_total_large_enr_times << std::endl;
    outfile << "throughput(per sec): ," << statis_thr_per_sec << ","
        << "throughput(per J): ," << statis_thr_per_J << std::endl;
    outfile.close();
    return 0;
}
```

源代码的最后一部分，完成了所有统计结果的计算，如上图所示。

如果想计算新的统计结果，可以用以上的变量重新计算输出。