1. 运行方法：

在 HME 目录下运行，其他依赖环境参照段博 gitHub

```
sudo ./run.sh
```

然后在其他 shell 运行程序

2. 运行时要注意的问题

1） 应用程序与机器拓扑

机器上有 2 个 node，目前是要求程序在 node0 上跑，然后 node1 作为 NVM。node0 可能有 10 个 core 或者更多，目前程序只写了 8 个 core，主要是考虑到用其他 core 做统计工作，你也可以修改这部分。同时考虑到一个 core 有两个逻辑 cpu，为避免干扰只用一个。

综上，目前的模拟器设计为利用 node0 上的前 8 个 core，每个 core 里面只利用一个逻辑 cpu。具体地，在 delay_count.py 里面如下图，可以根据自己的需求修改。此结构只是测试机器的拓扑结构。

```
coreID=[0,2,4,6,8,10,12,14]
```

运行应用的时候需要将应用运行在这几个核上，可以运行 1 个，也可以运行多个，但限定在这几个逻辑 cpu 上。

可以通过修改 coreNUM 核 coreID 进行修改，满足自己的实验要求。

2） 参数的问题

修改是基于原来的版本，所以有些地方没改动，主要有一些几点没有做测试或者说疑惑的地方，需要你们自己决定

【1】NVM 时延参数（在 delay_count.py 中）

原参数设置为下图，参数准不准，需要自己测试

```
WDELAY = 1300
RDELAY = 200
PRECISION = 10000
```

【2】NVM 访问统计次数问题

原来的版本涉及到 3 个硬件时间来统计 NVM 的读写事件；这 3 个事件及其含义分别为

```
offcore_response.demand_rfo.llc_miss.remote_dram

    [Counts all demand data writes (RFOs) miss the L3 and the data is returned from remote dram]

offcore_response.all_reads.llc_miss.remote_dram

    [Counts all data/code/rfo reads (demand & prefetch) that miss the L3 and the data is
returned from remote dram]

unc_h_requests.writes_remote

        [write requests to remote home agent. Unit: uncore_ha]
```

第 1 个是 NVM 写次数，第二个是读次数；第三个中的 home agent 是什么意思我还没理解。而且原始版本中实现也是错误的，所以这里时延的计算简化成了，这里的误差可能需要进一步的讨论。

```
delay[i] = readCount[i] * RDELAY + writeCount[i]*WDELAY
```

【3】实验结果准确性问题

目前让应用在多个 node 上分配内存，采用的 numactl 的—membind 和—interleave，但即使是在真机上（不运行模拟器时），多种分配方案也没有达到预期效果（例如存在 membind 在 node1 上比在 node0 上速度更快的情况）。NVM 模拟器是通过 perf 采样的，所以结果偏差也很大。

如果有更好的方法测试应用在多个 node 上分配内存，希望大家提出。