

# 1 软件介绍

NAMD 是一个用于生物大分子大规模分子动力学的并行软件，支持 Charmm、Namd 和 Amber 等多种力场，由美国 Illinois 大学生物物理系和计算机系联合开发，旨在开发出高效的分子动力学并行程序，可支持 Charm++ 并行。目前 NAMD 还支持在 GPU 加速器上的运算。NAMD 具有非常强的大规模并行计算能力，已经实现了在上千个处理器上的并行计算，对包含超过三十万个原子的大分子系统进行模拟。NAMD 注册后可以免费下载使用：<http://www.ks.uiuc.edu/Research/namd/>

# 2 软件依赖

Fortran90 编译：操作系统自带的 GCC 编译器；

单精度 FFTW3 数学库：fftw3，编译时加--enable-float 选项；

GPU 节点 CUDA 驱动；

还依赖以下系统盘自带的安装包：

tcl-8.5.7-6.el6.x86\_64

tcl-devel-8.5.7-6.el6.x86\_64

numactl-devel-2.0.7-6.el6.x86\_64

操作系统：Ubuntu 16.04

# 3 安装步骤

## 3.1 CUDA

到 <https://developer.nvidia.com/cuda-downloads> 下载对应操作系统的 cuda 安装包。下载后执行：

```
chmod +x cuda_9.0.176_384.81_linux.run #使之具有可执行权限
sudo sh cuda_9.0.176_384.81_linux.run
```

然后按照相关的提示输入安装路径即可，本文选择默认路径。详细安装步骤可以参考 CUDA 9 安装手册。

环境变量：

```
cat /etc/profile.d/cuda-env.sh
```

```
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
export C_INCLUDE_PATH=/usr/local/cuda/include:$C_INCLUDE_PATH
```

## 4 NAMD 编译和运行

下载 NAMD 2.1.3 文件：

在 NAMD 官方网站可以下载，链接如下：

<http://www.ks.uiuc.edu/Research/namd/2.13/features.html>

在 release note 中，介绍了该版本的 NAMD 一些新特性，点击 download site 即可进入下载页面，注册后即可下载。

### NAMD 2.13 New Features

#### Ready for Testing

The following features will be released in NAMD 2.13, but are **available now** in the **nightly build version** on the [download site](#). These features are documented in the *Nightly Build User's Guide (online or 1.8M PDF)* and *release notes*.

#### GPU-accelerated simulations less limited by CPU performance

Contributed by Antti-Pekka Hynninen. Bonded forces and exclusions are now offloaded to the GPU by default. (May be disabled with "bondedCUDA 0". Note that this is a bit flag so default is "bondedCUDA 255", setting "bondedCUDA 1" will offload only bonds, not angles, dihedrals, etc.) Removes all load-balanced work from the CPU, leaving only integration and optional features. The only benefit is for machines that were limited by CPU performance, so a high-end dual-socket workstation with an older GPU may see no benefit, while a single-socket desktop with the latest 1080 Ti will see the most.

#### Improved GPU-accelerated performance for non-orthogonal cells

The "useCUDA2" and "usePMECUDA" kernels introduced in NAMD 2.12 now also support non-orthogonal periodic cells.

#### Gaussian accelerated MD (GaMD)

Contributed by Andrew Pang and Yi Wang.

#### Collective variables module improvements

Contributed by Giacomo Fiorin and Jerome Henin.

#### IMDIgnoreForces option for QwikMD

Blocks steering forces while still allowing pause/resume/exit commands.

#### Various psfgen improvements

When reading topology files, recognize READ statements even if previous END statement is missing, and warn and ignore on self and duplicate bonds. For topology file PRES (patch) entries support ATOM record with name but no type or charge to specify subsequent atom insertion order. Extend segment command to query charge.

#### Preliminary support for billion-atom systems

Fixed 32-bit integer overflows in psfgen, structure compression (takes 2 hours), and memopt build. Minimally tested on 10x10x10 grid of STMV with IdBalancer none.

#### Update to Charm++ 6.8.2

Various bug fixes and performance enhancements.

#### Require CUDA 8.0 or greater, Kepler or newer GPUs

Includes kernels compiled for Kepler, Maxwell, and Pascal GPUs. Support for Fermi GPUs was dropped in NAMD 2.12.

## 4.1 multicore CUDA 版二进制 NAMD 可执行文件

我们在 NAMD 官网下载 2.13 版本的 multicore CUDA 版 NAMD，下载链接如下：

<http://www.ks.uiuc.edu/Development/Download/download.cgi>

点击 Linux-x86\_64-multicore-CUDA (NVIDIA CUDA acceleration) 即可下载。

由于 2.1.3 版本的 Linux-x86\_64-multicore-CUDA 是在 CUDA 8.0 下编译的二进制可执行文件，如果运行平台也是 CUDA 8.0，可以直接运行，如果是更高的版本，需要从源码编译安装，请参考下面的安

装方法。

## 4.2 编译 GPU 版 NAMD

解压缩 NAMD :

```
tar xvfNAMD_Git-2018-04-06_Source.tar.gz #此文件实际为 tar , 没有 gz 后缀 , 因此参数为 xvf , 不需要 xzvf。
```

NAMD 压缩文件中包含 Charm , 继续解压 :

```
cd NAMD_Git-2018-04-06_Source
tar xvf charm-6.8.2.tar
```

下载并安装依赖库

下载 TCL 和 FFTW , 下载链接如下 :

```
wget http://www.ks.uiuc.edu/Research/namd/libraries/fftw-linux-x86_64.tar.gz
wget http://www.ks.uiuc.edu/Research/namd/libraries/tcl8.5.9-linux-x86_64.tar.gz
wget http://www.ks.uiuc.edu/Research/namd/libraries/tcl8.5.9-linux-x86_64-threaded.tar.gz
```

安装命令如下 :

```
tar xzf fftw-linux-x86_64.tar.gz
mv linux-x86_64 fftw
tar xzf tcl*-linux-x86_64.tar.gz
tar xzf tcl*-linux-x86_64-threaded.tar.gz
mv tcl*-linux-x86_64 tcl
mv tcl*-linux-x86_64-threaded tcl-threaded
```

### 4.2.1 编译 charm++ 并行库

- charm++ 编译

```
tar xvf charm-6.8.2.tar
cd charm-6.8.2/
./build charm++ verbs-linux-x86_64 gcc smp --with-production #分布式计算模式
```

```
./build charm++ multicore-linux64 gcc --with-production #单节点 multicore 模式
```

这里选择的是 gcc 编译器，也可以使用其他编译器，如 icc，将上面的命令中的 gcc 替换为 icc 即可。

## 4.2.2 编译 GPU 版 NAMD 主程序

- 生成编译时参数

如果编译单节点模式：

```
./config Linux-x86_64-g++ --charm-arch multicore-linux64-gcc --with-cuda --cuda-prefix  
/usr/local/cuda  
cd Linux-x86_64-g++/  
make
```

如果编译多节点并行模式：

```
./config Linux-x86_64-g++ --charm-arch verbs-linux-x86_64-smp-gcc --with-cuda --cuda-  
prefix /usr/local/cuda  
cd Linux-x86_64-g++/  
make
```

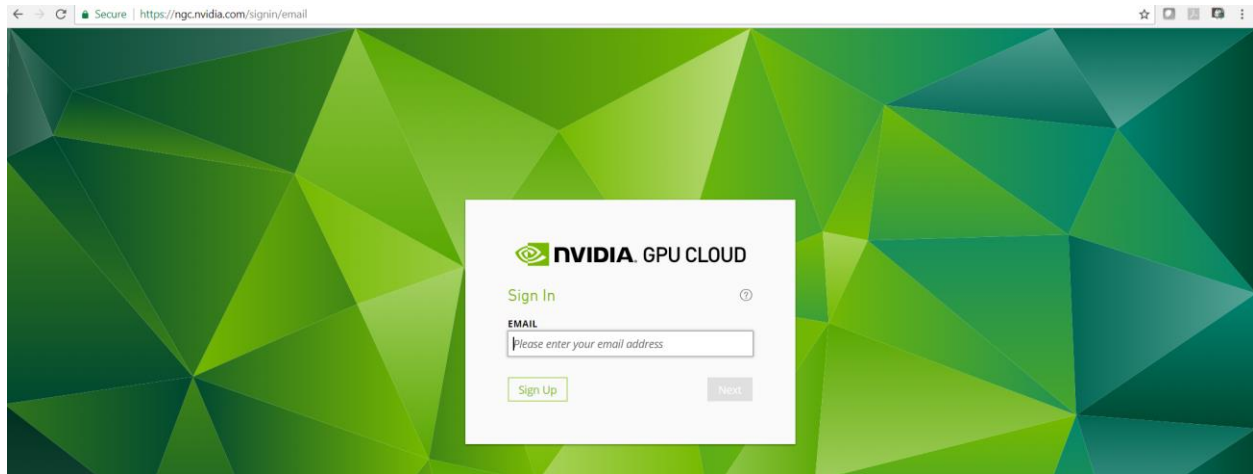
- 开始编译程序

```
make #编译完成后会生成 charmrun , namd2 等文件。
```


## 4.3 NAMD 基于 NGC Docker 安装

在 NVIDIA GPU Cloud ( NGC ) 中，包含 Namd 的 docker 镜像文件，可以直接下载，导入 Linux docker 环境就可以使用。

## 注册 NGC



填写注册信息：



DEEP LEARNING EVERYWHERE, FOR EVERYONE

### Sign Up ?

<b>NAME</b> <input type="text" value="Please enter your full name"/>	<b>ROLE</b> <input type="text" value="Please select your job role"/>
<b>COMPANY</b> <input type="text" value="Please enter your company or organization"/>	<b>INDUSTRY</b> <input type="text" value="Please select your industry"/>
<b>EMAIL</b> <input type="text" value="Please enter your email address"/>	<b>COUNTRY</b> <input type="text" value="Please select your country"/>

I agree to the [NVIDIA ACCOUNT TERMS OF USE](#)

By obtaining any third party software containers through NGC, I agree that NVIDIA will share my registration information with such third party software providers, who may use my information as permitted by their privacy policies.

Send me NVIDIA GPU Cloud updates and enterprise news

[Cancel](#)

## 登录 NGC 系统

注册 NGC 后，登录 NGC 系统，就可以看到下图中所有的 HPC Apps 的 docker image 镜像。

Repositories

- nvidia
  - caffe
  - caffe2
  - cntk
  - cuda
  - digits
  - mxnet
  - pytorch
  - tensorflow
  - tensorrt
  - theano
  - torch
- hpc
  - candle
  - gamess
  - gromacs
  - lammps
  - lattice-microbes
  - namd**
  - reion
  - vmd
- nvidia-hpcvis

hpc/namd

```
docker pull nvcv.io/hpc/namd:2.12-171025
```

### NAMD

NAMD is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems. NAMD uses the popular molecular graphics program VMD for simulation setup and trajectory analysis, but is also file-compatible with AMBER, CHARMM, and X-PLOR.

[See here](#) for a document describing prerequisites and setup steps for all HPC containers.

[See here](#) for a document describing the steps to pull NGC containers.

<br/>

### 1. Running NAMD

TAG	SIZE	USER	LAST MODIFIED	PULL
2.12-171025	1.47 GB		November 21, 2017	↓

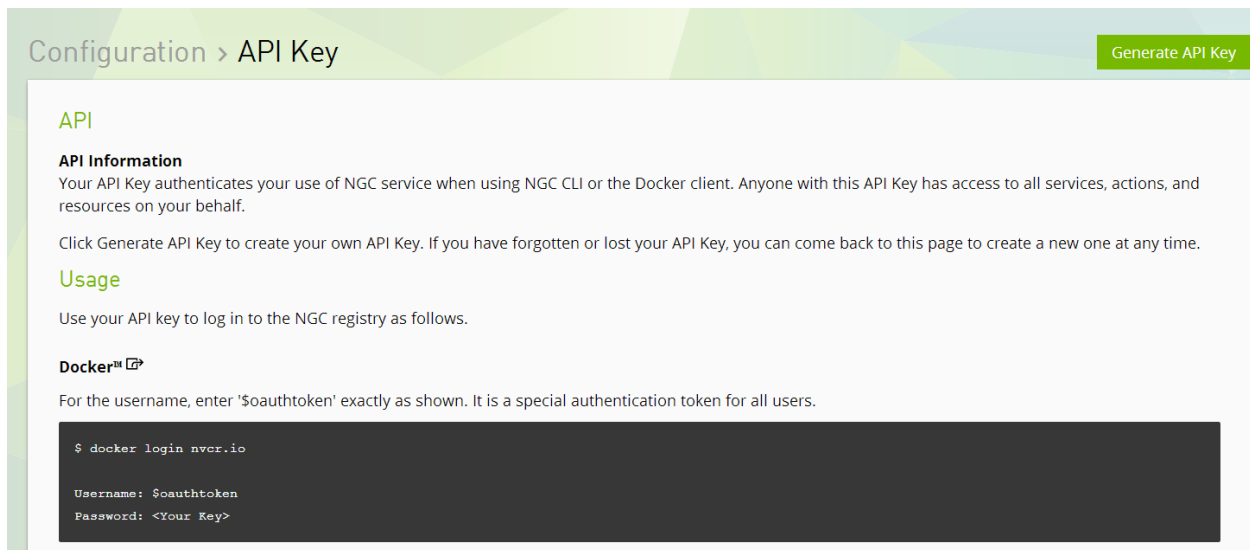
## Docker image 下载

下载 image 镜像之前，先要获取 API key：

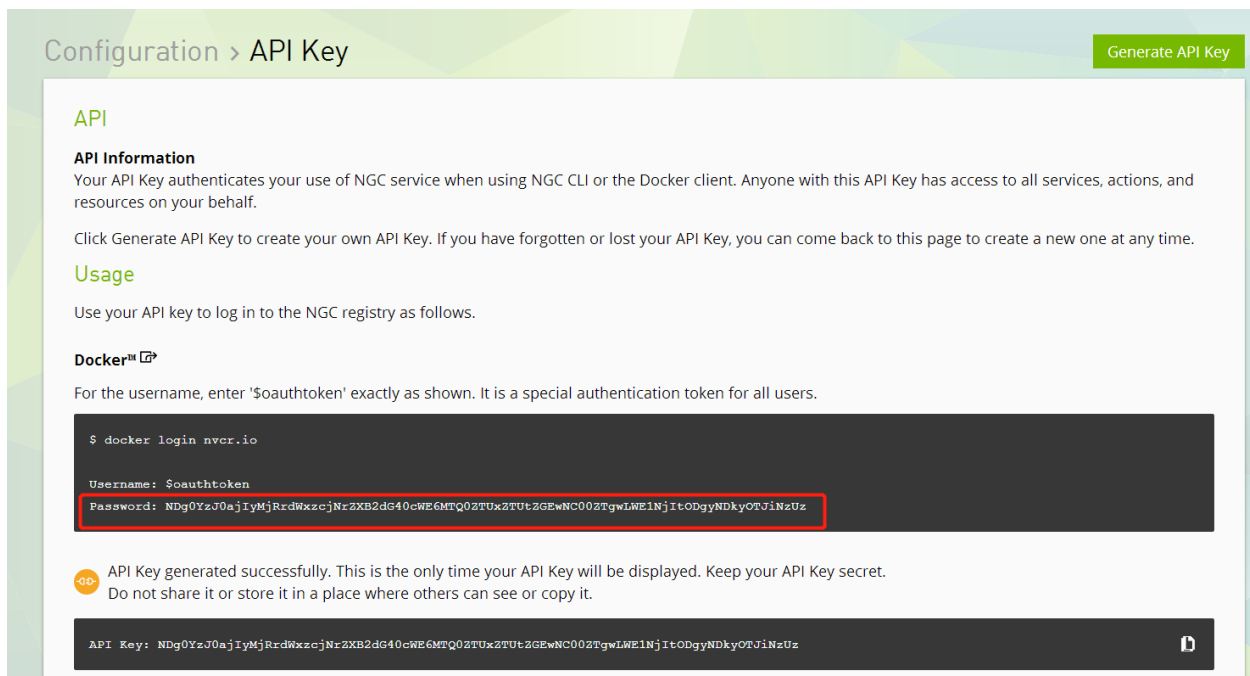
hpc/namd

```
docker pull nvcv.io/hpc/namd:2.12-171025
```

点击 Get API Key，即可获得：



点击 “Generate API Key”，并点击弹出对话框中的 “Confirm”，系统会生成一个 API Key 作为 nvcr.io 的登录密码，并复制该 Password，用于登录：



在 Linux 客户端登录方式如下，作者是在 DGX-1 平台下，Ubuntu 16.04 的系统演示的，如下：

```
dgxsa@dgx1:~$ docker login nvcr.io
Username ($oauthtoken): $oauthtoken
Password:
Login Succeeded
dgxsa@dgx1:~$
```

登录成功以后，就可以进行 Namd 容器下载，如下：

Repositories

- nvcr.io
  - caffe
  - caffe2
  - cntk
  - cuda
  - digits
  - mxnet
  - pytorch
  - tensorflow
  - tensorrt
  - theano
  - torch
- hpc
  - candle
  - games
  - gromacs
  - lammmps
  - lattice-microbes
  - namd**
  - relicon
  - vmd
- nvidia-hpcvis

hpc/namd

```
docker pull nvcr.io/hpc/namd:2.12-171025
```

[See here](#) for a document describing prerequisites and setup steps for all HPC containers.  
[See here](#) for a document describing the steps to pull NGC containers.

<br/>

### 1. Running NAMD

There are two options to run the NAMD container.

- You can run NAMD in detached mode from the nvidia-docker run command
- You can start the container in interactive mode and run NAMD interactively within the container

<br/>

TAG	SIZE	USER	LAST MODIFIED	PULL
2.12-171025	1.47 GB		November 21, 2017	↓

将命令 “docker pull nvcr.io/hpc/namd:2016.4” 复制到 Linux terminal：

```
dgxsa@dgx1:/raid/chengyi$ docker login nvcr.io
Username ($oauthtoken):
Password:
Login Succeeded
dgxsa@dgx1:/raid/chengyi$ docker pull nvcr.io/hpc/namd:2.12-171025
```

## 启动 docker 并运行 Namd

使用 nvidia-docker 命令查看 Namd 容器镜像信息。



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
qp_lammps_180403	v1.0	95f9899acc84	5 days ago	5.5GB
qp_caffe2_mpi_ofed	v1.0	f36e5f65e417	7 days ago	7.39GB
qp_caffe2_mpi	v1.0	050f87845aab	8 days ago	6.59GB
nvcr.io/nvidia/caffe2	18.03-py3	e223f0e24295	5 weeks ago	3.01GB
nvcr.io/nvidia/tensorrt	18.03-py2	2d6903917375	5 weeks ago	4.33GB
nvcr.io/nvidia/digits	18.02	4f7dc6bf79a6	7 weeks ago	5.45GB
nvcr.io/nvidia/tensorflow	18.02-py3	57ae51ee8b74	7 weeks ago	2.91GB
nvcr.io/nvidia/tensorflow	18.02-py2	5729fa7f740d	7 weeks ago	2.91GB
nvcr.io/nvidia/pytorch	18.02-py3	0fa1c8e0011b	7 weeks ago	5.76GB
nvcr.io/nvidia/caffe	18.02-py2	fc16e9d37dff	8 weeks ago	3.29GB
nvcr.io/nvidia/caffe2	18.02-py3	13aa25b41ad1	8 weeks ago	2.87GB
nvcr.io/nvidia/caffe2	18.02-py2	bfe2278714cd	8 weeks ago	2.86GB
nvcr.io/nvidia/theano	18.02	77ac9f9b866d	8 weeks ago	3.86GB
nvcr.io/nvidia/mxnet	18.02-py3	9731cb1dbf8b	8 weeks ago	2.81GB
nvcr.io/nvidia/cntk	18.02-py3	d1cb490099fd	8 weeks ago	6.4GB
nvcr.io/nvidia/tensorrt	18.02-py2	23f146705293	8 weeks ago	4.31GB
nvcr.io/nvidia/caffe	18.01-py2	c81ff7e540db	3 months ago	3.25GB
nvcr.io/nvidia/mxnet	18.01-py2	ffe7de2f34c3	3 months ago	2.64GB
nvcr.io/nvidia/tensorflow	18.01-py2	377b46c75bfc	3 months ago	2.88GB
nvcr.io/nvidia/pytorch	18.01-py3	27f7f6895e25	3 months ago	4.73GB
nvcr.io/nvidia/caffe2	18.01-py2	fa0881b1b245	3 months ago	2.82GB
nvcr.io/hpc/namd	2.12-171025	9a6c17154075	4 months ago	2.91GB
nvd1.githost.io:4678/dgx/tensorrt	17.12-stage	fd28d0eaae4b	4 months ago	4.08GB
nvcr.io/nvidia/tensorflow	17.12	19afd620fc8e	4 months ago	2.88GB
nvd1.githost.io:4678/dgx/cuda	9.0-cudnn7-devel-ubuntu16.04	634be617d3ed	4 months ago	1.75GB
nvcr.io/hpc/gromacs	2016.4	d5711c31ecde	4 months ago	2.86GB
nvcr.io/hpc/lammps	patch230ct2017	d8ae9ee6bf06	4 months ago	3.48GB
nvcr.io/nvidia/pytorch	17.11	fb6782fcd54	5 months ago	4.76GB

启动 docker 镜像：

```
dgxsa@dgx1:~$ nvidia-docker run --name MyNamd -v /home/dgxsa/chengyi/share:/data -it nvcr.io/hpc/namd:2.12-171025 /bin/bash
```

#### docker run Options

- -i -t or -it : 交互式，连接到一个"tty"
- --name : 给容器命名
- -v /home/dgxsa/chengyi/share:/data : 将 host 主机的/home/dgxsa/chengyi/myshare 存储目录映射到容器的 data 目录。

```
dgxsa@dgx1:/raid/chengyi$ nvidia-docker run --name MyNamd -v /home/dgxsa/chengyi/share:/data -it nvcr.io/hpc/namd:2.12-171025 /bin/bash
root@fd6c96ca0779:/#
root@fd6c96ca0779:/#
```

启动容器以后，就可以像在一台 Linux 服务器上操作了，里面已经配置好了所有运行环境，如果需要安装其他软件，可以使用命令：`apt-get install xxxx` 进行安装。

如果想退出容器，可以使用命令：`Ctrl+D`

如果想删除容器，可以使用命令：`nvidia-docker rm fd6c96ca0779`；`fd6c96ca0779` 为 docker-ID

如果想退出容器登录界面，但保持容器后台运行，可以使用命令：`Ctrl+P`，然后 `Ctrl+Q`

#### Tips:

也可以将 NAMD 的 container 中 `/opt/namd` 文件夹下的可执行文件拷贝下来直接运行。

## 4.4 运行 GPU 版 NAMD

### 4.4.1 配置 NAMD 文件

- 使用 NAMD 网站上的标准算例，例如 apoa1，然后修改算例的输入文件
  - 注释掉 par\_all22\_prot\_lipid.xplor，CUDA 版 NAMD 不支持 NBFIX
  - 修改 apoa1.namd 控制文件

```
numsteps 1000 #总共模拟 1000 步
outputtiming 100 #每 100 步输出一次时间信息
outputenergies 100 #每 100 步输出一次能量，小于 60 时不能跑在 GPU 上
```

### 4.4.2 命令运行 NAMD

- 在 GPU 节点上运行 namd 程序

当 namd 程序在 GPU 节点上运行的时候，每个进程都会在 GPU 上启动相应的线程在 GPU 上，使用 nvidia-smi 程序查看 NAMD 程序在 GPU 上的运行状态。

- multicore 运行

```
/opt/namd/namd-multicore +p40 +setcpuaffinity +idlepoll
/namd_bench/apoa1/apoa1.namd
```

- 跨节点运行
  - 创建 nodelist 文件，按照如下的格式写

如果是单个节点 16 核 charmrun 并行，格式如下：

```
dgxsa@dgx1:/raid/chengyi/share/test/apoa1$ cat nodelist
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
host dgx1
```

```
/raid/chengyi/share/software/namd/charmrun ++nodelist nodelist ++p 16 ++ppn 16
/raid/chengyi/share/software/namd/namd2.12.171025-verbs +setcpuaffinity +pemap 1-7,9-
15 +commap 0,8 +devices 0,1,2,3 apoa1.namd
```

8个节点计算，每个节点2个进程，如下：

```
dgxsa@dgx1:/raid/chengyi/share/test/apoa1$ cat nodelist
host node01
host node01
host node02
host node02
host node03
host node03
host node04
host node04
host node05
host node05
host node06
host node06
host node07
host node07
host node08
host node08
```

跨界点运行 namd 程序，命令如下：

```
/raid/chengyi/share/software/namd/charmrun ++nodelist nodelist ++p 16 ++ppn 2
/raid/chengyi/share/software/namd/namd2.12.171025-verbs +setcpuaffinity +pemap 1-7,9-
15 +commap 0,8 apoa1.namd
```

### {charmOpts}的参数说明：

- ++nodelist {nodeListFile} – 多节点运行需要指定的节点列表文件
- Charm++ 也支持++mpiexec 参数，用于作业调度系统。
- ++p \$totalPes – 指定总的 PE 线程数
  - ++ppn \$pesPerProcess – 每个节点的线程数，推荐： $\#ofCoresPerNode/\#ofGPUsPerNode - 1$ ，即 $[(\text{每节点核数}) / (\text{每节点 GPU 数})] - 1$ ，留一个核心用于通信。
  - 总进程数为 $\$totalPes/\$pesPerProcess$

### {namdOpts}的参数说明：

- NAMD 继承{charmOpts}后面设置的参数，如：'+p', '+ppn', '+p'
- 如果没有{charmOpts}，采用 multi-core 计算，使用'+p' 设置计算的核数。
- '+setcpuaffinity' 选项是为了核绑定，不会到处跳动。
- '+pemap #-#' – 这是设置 thread 线程和 CPU 核心的映射。
- '+commap #-#' – 这是设置通信线程的范围。
- 范例：双 CPU，每 CPU 16 核心，参数设置如下：

```
+setcpuaffinity +pemap 1-15,17-31 +commap 0,16
```

- GPU 选项：
- '+devices {CUDA IDs}' – 指定 NAMD 调用的 GPU ID

## 4.4.3 作业运行脚本

### 4.4.3.1 Slurm 脚本范例

```
#!/bin/bash
#SBATCH --job-name namdtest
#SBATCH --partition longqueue
#SBATCH --nodes 2
```

```

#SBATCH --ntasks-per-node 20
#SBATCH --time 00:10:00
#SBATCH --output namd-test.${SLURM_JOBID}.out
# choose version of NAMD to use
export NAMD_DIR=/projects/username/NAMD/NAMD_2.11_Linux-x86_64-ibverbs-smp
export PATH=$PATH:$NAMD_DIR
cd /scratch/bhaddad/NAMD/Coeus_test
# generate NAMD nodelist
for n in `echo $SLURM_NODELIST | scontrol show hostnames`; do
  echo "host $n ++cpus 19" >> nodelist.${SLURM_JOBID}
done
# calculate total processes (P) and procs per node (PPN)
PPN=`expr $SLURM_NTASKS_PER_NODE - 1`
P="$(($PPN * $SLURM_NNODES))"
charmrun ++mpiexec ++remote-shell srun
/home/bhaddad/NAMD_2.12_Linux-x86_64-verbs-smp/namd2 ++p $P ++ppn $PPN
+setcpuaffinity +isomalloc_sync test.conf

```

#### 4.4.3.2 PBS 脚本范例

```

#!/bin/bash
#PBS -q gpuqueue
#PBS -l nodes=2:ppn=4
#PBS -l walltime=100:00:00
#PBS -e ${PBS_JOBID}.err
#PBS -o ${PBS_JOBID}.out
cd $PBS_O_WORKDIR
chmod +x job.sh
./job.sh

```

job.sh:

```

#!/bin/sh -x
PROC_NUM=20 #number_of_procesor_same_as_in_PBS
echo "*****"
echo "Running on: $HOSTNAME"
echo "*****"
if [ "x$PBS_NODEFILE" != "x" ] ; then
  echo "PBS Nodefile: $PBS_NODEFILE"
  HOST_NODEFILE=$PBS_NODEFILE
  fi
  if [ "x$HOST_NODEFILE" = "x" ]; then
    echo "No hosts file defined. Exiting..."
    exit
  fi
echo "Creating host file..."
export NODES=`cat $PBS_NODEFILE`
export NODELIST=nodelist

```

```

echo group main > $NODELIST
for node in $NODES ; do
echo host $node ++shell ssh >> $NODELIST
done
echo "Nodelist file:"
cat $NODELIST
export NAMD_DIR=/projects/chengyi/NAMD/NAMD_2.13_Linux-x86_64-ibverbs-smp
export PATH=$PATH:$NAMD_DIR
charmrun ++remote-shell ssh ++nodelist $NODELIST +p$PROC_NUM namd2
apoal.namd

```

## 4.5 标准算例测试

### 4.5.1 测试环境

NVIDIA DGX-1 平台，操作系统：ubuntu 16.04 无盘

CPU：Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz

内存：512G DDR 4 内存

GPU：8\*Tesla V100 SXM2

硬盘：480G SSD

GPU 驱动：CUDA-9.0

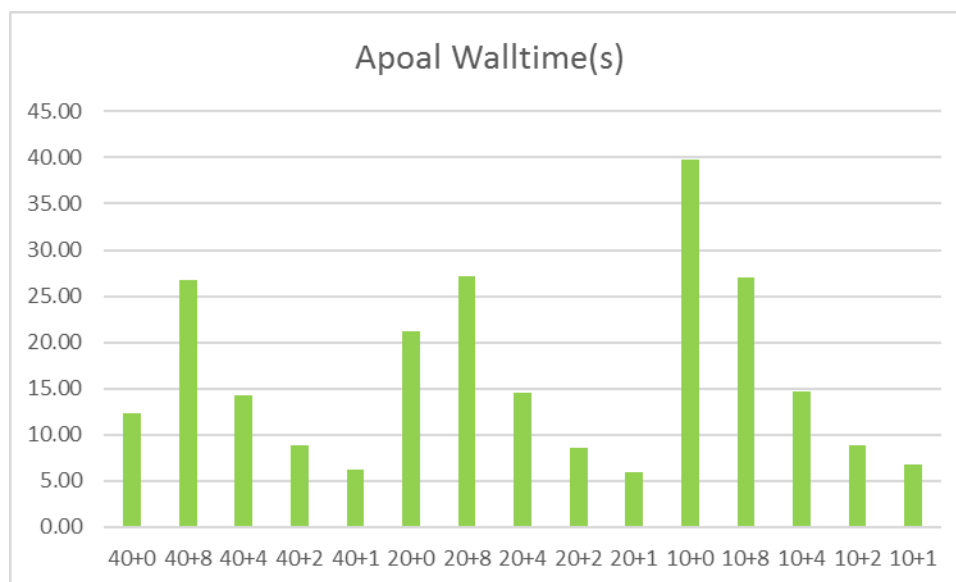
算例：apoal 标准算例和 VDPV1 算例。

### 4.5.2 测试结果

由于 Apoal 算例比较小，GPU 利用率较低，所以加速效果不明显，对于计算规模更大的算例，可以获得明显的加速。

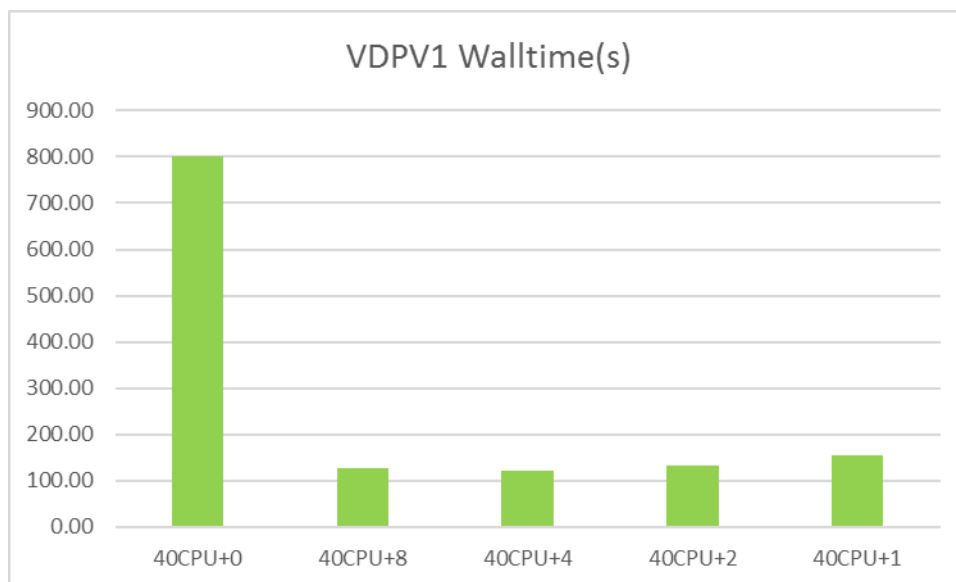
CPU 进程数+GPU 数	Walltime(s)	Speedup
40+0	12.31	
40+8	26.73	0.46
40+4	14.28	0.86
40+2	8.79	1.40
40+1	6.19	1.99
20+0	21.23	

20+8	27.19	0.78
20+4	14.52	1.46
20+2	8.58	2.48
20+1	5.93	3.58
10+0	39.80	
10+8	27.04	1.47
10+4	14.74	2.70
10+2	8.81	4.52
10+1	6.74	5.90



下面是另外一个测试算例 VDPV1，共 33 万个原子，计算量而比较大：

CPU 进程数+GPU 数	Walltime(s)	speedup
40CPU+0	801.06	
40CPU+8	127.22	6.30
40CPU+4	121.26	6.61
40CPU+2	132.48	6.05
40CPU+1	154.39	5.19



从这个算例可以发现，1块 V100 可以获得 5.2 倍的加速，加速效果非常明显，此时通过 nvidia-smi 查看发现，GPU 的实际利用率大约 70~80%，2 块 GPU 就只能获得 6 倍的加速，此时每块 GPU 的实际利用大约时候 40-50%。当 4 块以上 GPU 并行时，GPU 的利用率只有 20% 左右，因此不能获得更高的加速，NAMD 计算时，计算量和 GPU 数量匹配很重要。在 Apoal 算例中，GPU 的利用率都在 10% 以下，因此加速效果不是很明显。

注：NAMD 在物理服务器上直接运行性能较好，使用 apaol 算例，40 核，8 块 GPU，和物理机相比，docker container 会有 11% 左右的损失。