GPU Accelerated Implementation of ONETEP

Karl A. Wilkinson*

February 11, 2016

1 Introduction

An OpenACC implementation of ONETEP is available to allow execution on machines containing graphical processing units based accelerators (GPUs). GPUs are highly parallel and are well suited to algorithms such as the fast fourier transforms (FFTs) within ONETEP during the calculation of properties such as the local potential integrals and the charge density.

However, the connection of the accelerators to the host machine through the peripheral component interconnect express (PCIe) bus introduces a bottleneck when large amounts of data are transferred. Currently, this is an issue when moving the fine grid FFT boxes from the accelerator to the host machine but future generations of hardware, and developments within ONETEP are expected to reduce this issue and improve performance significantly.

This work has been published in the Journal of Computational Chemistry. More detailed information is available in this publication: http://onlinelibrary.wiley.com/doi/10.1002/jcc.23410/abstract However, significant performance improvements have achieved since the publication of this article. It should be noted that this feature of the ONETEP package is under development.

2 Compilation

Compilation of the OpenACC implementation of ONETEP is only currently supported by the compilers from the Portland Group International (PGI). Relatively few changes are required in order to perform the compilation: The flag:

```
-DGPU_PGI
```

should be used and additional variable describing the flags and libraries need to be defined:

```
ACCFLAGS = -ta=nvidia -Mcuda=6.5
ACCLIBS = -lcufft -lcudart
```

Here, we are utilising the CUDA 6.5 runtime libraries as they are the most up to date version available on the TITAN supercomputer at the Oak Ridge National laboratories, your local machine may have a more up to data version available.

Further examples of complete config files for the desktops at the University of Southampton and the Wilkes cluster at the University of Cambridge follow:

```
################  Southampton desktop ################
F90 = pgf90
MPIROOT=/local/scratch/kaw2e11/software/openmpi_1.6.4/pgi/
FFTWROOT=/local/scratch/kaw2e11/software/fftw/pgi/
FFLAGS = -DGPU_PGI -DFFTW3 -DMPI -I$(MPIROOT)include -I$(FFTWROOT)include -I$(MPIROOT)lib/
OPTFLAGS = -O3 -fast
DEBUGFLAGS = -g -C
```

*karl.wilkinson@uct.ac.za

```
MPILIBS= -L$(MPIROOT)lib/ -lmpi_f90 -lmpi_f77 -lmpi -lopen-rte -lopen-pal -ldl -Wl,
--export-dynamic -lnsl -lutil -ldl
ACCFLAGS = -ta=nvidia -Mcuda=6.5
ACCLIBS = -L/usr/lib64/nvidia -L$(CUDAROOT)/lib64/ -lcufft -lcudart
LIBS = $(MPILIBS) -llapack -lblas -L$(FFTWROOT)lib/ -lfftw3_omp -lfftw3 -lm

################ WILKES ################
FC := mpif90
F90 := $(FC)
FFLAGS = -DGPU_PGI -DFFTW3_NO_OMP -DMPI -DNOMPIIO -Mdalign
MKLPATH=${MKLROOT}/lib/intel64
LIBS=  -L$(MKLROOT)/lib/intel64 -lmkl_intel_lp64 -lmkl_core -lmkl_sequential -lpthread -lm
OPTFLAGS = -O3 -m64
WARNINGFLAGS = -Wall -Wextra
DEBUGFLAGS =
COMPILER = PORTLAND-pgf90-on-LINUX
ACCFLAGS = -acc -ta=nvidia:cc35 -Mcuda=6.5
ACCLIBS = -lcufft -lcudart
```

Unfortunately, attention should be paid to to the version of the compilers and libraries used as, due to the speed at which the OpenACC approach is evolving, it is a common for functionality to break. As such, this document will be regularly updated with details of combinations of compiler and library versions that are known to be stable.

3 Execution

Use of the OpenACC implementation of ONETEP does not require any changes to the ONETEP input files. However, job submission does change significantly in some platforms.

3.1 CUDA Multi Process Service

The CUDA Multi Process Service (MPS) daemon controls the way MPI processes see GPUs and allows multiple MPI processes to use a single GPU wherein the hyperqueue scheduler is used to utilise the hardware much more efficiently than when a single process is used per GPU. As a single MPI process does not provide sufficient computation to fully utilize a GPU, it is critical to use this technology to achieve optimal performance.

However, attention must be paid to ensure that GPU memory is not exhausted. Currently, the usage is reported but these safety checks need to be extended to allow a graceful exit should the total memory be exhausted.

Below are examples for the usage of MPS during job submission on Wilkes and TITAN:

3.1.1 Wilkes

On Wilkes, job submission is performed using: sbatch slurm_submit.tesla

where: slurm_submit.tesla is:

```
#!/bin/bash
#SBATCH -J MPS_test
#SBATCH -A SKYLARIS-GPU
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --time=00:30:00
#SBATCH --no-requeue
#SBATCH -p tesla

. /etc/profile.d/modules.sh
```

```
module purge
module load default-wilkes
module unload intel/impi intel/cce intel/fce cuda
module load pgi/14.7
module load mvapich2/2.0/pgi-14
ulimit -s unlimited

numnodes=$SLURM_JOB_NUM_NODES
numtasks=$SLURM_NTASKS
mpi_tasks_per_node=$(echo "$SLURM_TASKS_PER_NODE" | sed -e  's/^\([0-9][0-9]*\).*$/\1/')
JOBID=$SLURM_JOB_ID

cd $SLURM_SUBMIT_DIR

application="onetep.wilkes.gpu.cuda55"

echo "JobID: $JOBID"
echo "Time: `date`"
echo "Running on master node: `hostname`"
echo "Current directory: `pwd`"

if [ "$SLURM_JOB_NODELIST" ]; then
       #! Create a machine file:
       export NODEFILE=`generate_pbs_nodefile`
       cat $NODEFILE | uniq > machine.file.$JOBID
       echo -e "\nNodes allocated:\n================"
       echo `cat machine.file.$JOBID | sed -e 's/\..*$//g'`
fi

echo -e "\nnumtasks=$numtasks, numnodes=$numnodes, mpi_tasks_per_node=$mpi_tasks_per_node (OMP_NUM_T

# Start MPS deamons...
srun -N$SLURM_JOB_NUM_NODES -n$SLURM_JOB_NUM_NODES ./run_MPS.sh

echo -e "\nExecuting program:\n==================\n\n"

mpirun -np ${SLURM_NTASKS} -ppn ${mpi_tasks_per_node} --genvall -genv MV2_RAIL_SHARING_LARGE_MSG_THR


echo -e "\n\n>>> Program terminated! <<<\n"
echo -e "Time: `date` \n\n"

# Kill MPS deamons
srun -N$SLURM_JOB_NUM_NODES -n$SLURM_JOB_NUM_NODES ./kill_MPS.sh
```

This file, and the following files, were obtained from the Wilkes systems administrators. It is advisable to contact system administrators if you have any questions regarding the submission process.

Here, the files: run_MPS.sh and kill_MPS.sh manage the initialisation and termination of the MPS deamon and the run_app.sh controls the allocation of MPI processes to the correct GPUs. For reference, the contents of those files are as follows, again, it is advisable to speak with your systems administrator about equivalent scripts for other machines (For example, run_app.sh assumes the use of MVAPICH2).

```
#####run_MPS.sh
#!/bin/bash

# Number of gpus with compute_capability 3.5  per server
```

```
NGPUS=2

# Start the MPS server for each GPU
for ((i=0; i< $NGPUS; i++))
do
 echo "[CUDA-PROXY] Setting MPS on ‘hostname‘ for GPU $i..."
 mkdir /tmp/mps_$i
 mkdir /tmp/mps_log_$i
 export CUDA_VISIBLE_DEVICES=$i
 export CUDA_MPS_PIPE_DIRECTORY=/tmp/mps_$i
 export CUDA_MPS_LOG_DIRECTORY=/tmp/mps_log_$i
 nvidia-cuda-mps-control -d
done

exit 0
```

```
###/run_app.sh
#!/bin/bash

# Important note: it works properly when MV2_CPU_BINDING_LEVEL=SOCKET && MV2_CPU_BINDING_POLICY=SCAT

lrank=$MV2_COMM_WORLD_LOCAL_RANK
grank=$MV2_COMM_WORLD_RANK

case ${lrank} in
0|2|4|6|8|10)
  export CUDA_MPS_PIPE_DIRECTORY=/tmp/mps_0
  export MV2_NUM_HCAS=1
  export MV2_NUM_PORTS=1
  export MV2_IBA_HCA=mlx5_0
  echo "[CUDA-PROXY] I am globally rank $grank (locally $lrank ) on ‘hostname‘ and I am using GPU 0"
  "$@"
  ;;
1|3|5|7|9|11)
  export CUDA_MPS_PIPE_DIRECTORY=/tmp/mps_1
  export MV2_NUM_HCAS=1
  export MV2_NUM_PORTS=1
  export MV2_IBA_HCA=mlx5_1
  echo "[CUDA-PROXY] I am globally rank $grank (locally $lrank ) on ‘hostname‘ and I am using GPU 1"
  "$@"
  ;;
esac
```

```
##kill_MPS.sh
#!/bin/bash

echo "[CUDA-PROXY] Kill nvidia-cuda-mps-control on ‘hostname‘..."
killall -9 nvidia-cuda-mps-control

# this waiting time is to let killall have effect...
sleep 3

echo "[CUDA-PROXY] Clean /tmp on ‘hostname‘..."
rm -rf /tmp/mps_*
rm -rf /tmp/mps_log_*

exit 0
```

3.1.2 TITAN

Job submission on TITAN is somewhat more straightforward and the following script may be used directly. The important line is: `export CRAY_CUDA_PROXY=1` which enables the use of MPS.

```
#!/bin/bash
#PBS -A CODENAME
#PBS -N MgMOF74_111_SP
#PBS -j oe
#PBS -l walltime=1:30:00,nodes=XNUMNODES
#PBS -l gres=atlas1%atlas2

PROJECT=chm113

source $MODULESHOME/init/bash
module load cudatoolkit
#module swap PrgEnv-pgi/5.2.40 PrgEnv-intel/5.2.40

export CRAY_CUDA_PROXY=1

EXEDIR=/lustre/atlas/scratch/kaw2e11/chm113/binaries
#EXE=onetep.4313.titan.cpu.intel
EXE=onetep.4313.titan.gpu.pgi

##########################################
SOURCEDIR=/ccs/home/kaw2e11/BENCHMARKS/PGI_GPU/benchmark-XTOTALMPI-XNUMNODES-XMPIPERNUMANODE
INPUT=G_222_80_D2.dat
INFO=PGI_GPU-XTOTALMPI-XNUMNODES-XMPIPERNUMANODE
##########################################

BASENAME=‘basename $INPUT‘-$INFO
OUTPUT=$BASENAME.out

cd $MEMBERWORK/$PROJECT/
mkdir dir-$BASENAME
cd dir-$BASENAME

cp $SOURCEDIR/* $MEMBERWORK/$PROJECT/dir-$BASENAME

aprun -n XTOTALMPI -S XMPIPERNUMANODE -j 2 $EXEDIR/$EXE $INPUT &> $OUTPUT

cd ..
```