Οδηγίες Χρήσης του Cluster Argo για την Ανάπτυξη CUDA προγραμμάτων

Το cluster ARGO αποτελείται από τον front-end (Dell OptiPlex 7050, Quad-Core Intel-Core i5-6500 @3.20GHz,16GB DDR4 2.4GHz) στον οποίο συνδεόμαστε και από 11 compute nodes σε rack για την εκτέλεση παράλληλων προγραμμάτων. Στους 10 compute nodes Dell PowerEdge εκτελούμε MPI και υβριδικά MPI+OpenMp προγράμματα. Στον 11ο Dell Precision 7920 Rack (Intel Xeon Silver 4114 2.2 GHz, 10Cores with hyperthreading, 16GB (2X8GB) DDR4 2666MHz με Dual Nvidia Quatro P4000, 8GB) θα τρέχουμε CUDA και OpenMp+CUDA προγράμματα.

Το cluster είναι διαθέσιμο για Δοκιμές, Ανάπτυξη, Debugging, Μετρήσεις, κλπ στους φοιτητές και ερευνητές του τμήματος, δεν έχει περιορισμούς, χρονικούς, διαθεσιμότητας πόρων, κλπ. Παρακαλώ, όμως όχι καταχρήσεις, οι πόροι είναι για όλους.

Συνδεθείτε στον λογαριασμό σας στον front-end και αντιγράψτε τον φάκελο CUDA-Demo στον λογαριασμό σας με εκτέλεση της εντολής:

```
[user@argo]# cp -r /etc/skel2/CUDA-Demo .
```

Ο φάκελος περιέχει δυο απλές εφαρμογές με σκοπό την παρουσίαση μεταγλώττισης προγραμμάτων CUDA και τον τρόπο εκτέλεσης στο κόμβο c10 του cluster, που έχει τις 2 GPUs. Η κάθε εφαρμογή βρίσκεται σε ξεχωριστό φάκελο (simple-add, dot-product).μαζί με ένα script (myPBSscript.sh) για την εκτέλεση της. Επιπλέον υπάρχουν δυο εφαρμογές σε υποκαταλόγους (simple_add1GPUOmp και simple_add2GPUOmp) που είναι τροποποιήσεις της αρχικής εφαρμογής (simple-add) με την χρήση του μοντέλου OpenMp για την εκτέλεση πολλαπλών νημάτων (simple_add1GPUOmp) και την χρήση των δυο GPUs παράλληλα (simple_add2GPUOmp).

Μεταγλώττιση CUDA προγραμμάτων

Η μεταγλωττιστή cuda προγραμμάτων γίνεται με χρήση του nvcc compiler, της Nvidia, ως εξής:

```
nvcc "flags" inputfiles -o outputfile
```

για παράδειγμα :

```
[user@argo]# nvcc dot-product.cu -o dot-product
```

Για την χρήση του μοντέλου OpenMp είναι απαραίτητη η χρήση της επιλογής -fopenmp στον compiler και στον linker για την μεταγλώττιση του προγράμματος. Η χρήση της επιλογής -fopenmp στον nvcc compiler γίνεται ως έξης:

```
nvcc -X -fopenmp inputfile.cu -o outputfile
```

*Σημείωση: μπορείτε να μεταγλωττίστε τις εφαρμογές simple-add, simple_add1GPUOmp και simple_add2GPUOmp χρησιμοποιώντας την εντολή **make**, όπου το εργαλείο make θα διαβάσει τις ιδιότητες (compiler flags, input files) από το Makefile αρχείο. Αφού τελειώσει η μεταγλώττιση το εργαλείο make επιστρέφει τα object files και το εκτελέσιμο. Μπορείτε να τροποποιήσετε το makefile για τις δικές σας εφαρμογές προκειμένου να γίνει η μεταγλώττιση τους με την εντολή make, αντικαθιστώντας τα ονόματα των αρχείων εισόδου και εξόδου ανάλογα με την εφαρμογή σας.*

Εκτέλεση CUDA προγραμμάτων

Όπως με τα MPI προγράμματα, για την εκτέλεση των CUDA προγραμμάτων, χρησιμοποιούμε τον PBS, ο οποίος λειτουργεί με σύστημα ουράς. Θα καταχωρήσετε την δουλειά προς εκτέλεση και θα εκτελεστεί όταν έρθει η σειρά της. Στην ουρά εισάγετε με ένα script με κατάληξη .sh, το οποίο περιέχει όλες τις πληροφορίες και παραμέτρους για την εκτέλεση που θέλουμε.

Για να καταχωρήσετε εργασία στην ουρά εκτελείτε:

```
[user@argo]# qsub myPBSScript.sh
```

σε αυτήν την φάση το κέλυφος επιστρέφει το έξης:

```
jobID.argo
```

όπου jobID είναι πενταψήφιος αριθμός που αναφέρεται στην εργασία που καταχωρήσαμε στην ουρά εκτέλεσης. Το jobID αλλάζει με κάθε καταχώρηση.

Μετά την εκτέλεση του script θα δείτε δυο νέα αρχεία στον κατάλογο (myGPUJob.o[jobID] και myGPUJob.e[jobID). Στο αρχείο myGPUJob.o[jobID] καταγράφεται η έξοδος του προγράμματος (stdout). Εάν προκύψει κάποιο σφάλμα κατά την εκτέλεση (runtime error) η έξοδος σφαλμάτων (stderr) καταγράφεται στο αρχείο myGPUJob.e[jobID]. Για παράδειγμα μετά την εκτέλεση του προγράμματος simple-add, το αρχείο myGPUJob.o[jobID], θα έχει το έξης περιεχόμενο:

```
Simple vector addition example (15000000 elements)
Total GPU memory -75956224, free -167706624

Vector addition on CPU
Execution time 17.02 msecs
Bandwith 9.848318 GB/sec
Printing last 6 elements
-466735552.0 -496735552.0 -526735552.0 -556735552.0 -586735552.0 -616735488.0

Vector addition on GPU
Execution time 1.19 msecs
Bandwith 141.228363 GB/sec
Printing last 6 elements
-466735552.0 -496735552.0 -526735552.0 -556735552.0 -586735552.0 -616735488.0
~
"myGPUJob.o59154" 14L, 456C
```

Η Δομή του PBS Script για καθαρά CUDA προγράμματα

Με έντονο χρώμα παρουσιάζονται οι τιμές που μπορείτε να αλλάξετε κατά περίπτωση:

```
#!/bin/bash

#Which Queue to use
#PBS -q GPUq

#Max Wall time, Example 1 Minute #
#PBS -l walltime=00:01:00

#How many nodes and tasks per node, Example 1 node with 1 CPU and 1 GPU#
#PBS -l select=1:ncpus=1:ngpus=1

# Only this job uses the chosen nodes
#PBS -l place=excl

#JobName #
#PBS -N myGPUJob

#Change Working directory to SUBMIT directory#
cd $PBS_O_WORKDIR

# Run executable #
./simple-add
```

Συλλογή πληροφοριών των CUDA προγραμμάτων με nvprof

Το profiling πρόγραμμα **nvprof** επιτρέπει την συλλογή και την προβολή στοιχείων της εκτέλεσης CUDA προγραμμάτων στο CPU και στο GPU, π.χ, εκτέλεση του πυρήνα, των μεταφορών μνήμης κ.λπ.
Η συλλογή δεδομένων γίνεται ως έξης:

```
bash$ nvprof ./executable
```

Για να χρησιμοποιήσετε το nvprof θα πρέπει να τροποποιήσετε το myPBSscript.sh ώστε να τρέχει την εφαρμογή χρησιμοποιώντας το nvprof. Ως παράδειγμα για την συλλογή πληροφοριών για το simple-add πρόγραμμα θα πρέπει να αλλάξουμε το executable του script:

```
# Run executable #
nvprof ./simple-add
```

Μετά την εκτέλεση του προγράμματος το αρχείο myGPUJob.e[jobID] περιέχει στοιχεία που κατέγραψε το nvprof, για το παράδειγμα simple-add η έξοδος θα είναι παρόμοια με:

```
==92147== NVPROF is profiling process 92147, command: ./simple_add
==92147== Profiling application: ./simple_add
==92147== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   67.81%  16.992ms         2  8.4962ms  8.2164ms  8.7759ms  [CUDA memcpy HtoD]
                   27.60%  6.9176ms         1  6.9176ms  6.9176ms  6.9176ms  [CUDA memcpy DtoH]
                    3.49%  874.04us         1  874.04us  874.04us  874.04us  kadd(float*, float*, float*, int)
                    1.10%  275.30us         1  275.30us  275.30us  275.30us  [CUDA memset]
      API calls:   85.51%  192.32ms         1  192.32ms  192.32ms  192.32ms  cudaMemGetInfo
                   10.83%  24.349ms         3  8.1163ms  7.0949ms  8.8986ms  cudaMemcpy
                    2.15%  4.8280ms         3  1.6093ms  253.12us  2.2961ms  cudaFree
                    0.50%  1.1275ms         1  1.1275ms  1.1275ms  1.1275ms  cudaThreadSynchronize
                    0.33%  751.59us         3  250.53us  187.04us  358.72us  cudaMalloc
                    0.33%  744.14us         2  372.07us  369.39us  374.75us  cuDeviceTotalMem
                    0.28%  619.08us       194  3.1910us     271ns  128.65us  cuDeviceGetAttribute
                    0.03%  59.868us         2  29.934us  25.861us  34.007us  cuDeviceGetName
                    0.03%  57.410us         1  57.410us  57.410us  57.410us  cudaMemset
                    0.01%  20.936us         1  20.936us  20.936us  20.936us  cudaLaunchKernel
                    0.01%  17.997us         2  8.9980us  2.3010us  15.696us  cuDeviceGetPCIBusId
                    0.00%  3.4700us         3  1.1560us     411ns  2.3260us  cuDeviceGetCount
                    0.00%  1.8730us         4     468ns     334ns     669ns  cuDeviceGet
                    0.00%  1.0470us         2     523ns     441ns     606ns  cuDeviceGetUuid
                    0.00%     248ns         1     248ns     248ns     248ns  cudaGetLastError
```

Προσαρμογή PBS script για Υβριδικά προγράμματα OpenMp + CUDA (1 ή 2 GPUs)

1. OpenMp single GPU

Στην εφαρμογή **simple_add1GPUOmp** η εκτέλεση των πράξεων της CPU γίνεται σε πολλαπλά νήματα. Η ουρά GPUq επιτρέπει την χρήση μέχρι ncpus=20 πυρήνων (υπάρχουν 10 cores, αλλά λόγω hyperthreading φαίνονται 20). Ο έλεγχος αριθμού νημάτων γίνεται με την τροποποίηση του script, συγκριμένα ελέγχεται από την τιμή της μεταβλητής ompthreads=x ή την μεταβλητή ncpus=x σε περίπτωση που δεν οριστεί η μεταβλητή ompthreads.

Για παράδειγμα το παρακάτω script επιτρέπει στην εφαρμογή να αξιοποιήσει 20 νήματα σε 10 πυρήνες.

```
#!/bin/bash

# Which Queue to use, DO NOT CHANGE #
#PBS -q GPUq

# Max Wall time, Example 1 Minute #
#PBS -l walltime=00:01:00

# How many nodes and tasks per node,  1 node with 10 cpus 2 tasks(threads) per cpu#
#PBS -lselect=1:ncpus=10:ompthreads=20:ngpus=1

# Only this job uses the chosen nodes
#PBS -lplace=excl

# JobName #
#PBS -N myGPUJob

#Change Working directory to SUBMIT directory
cd $PBS_O_WORKDIR

# Run executable #
./simple_add
```

Στην έξοδο του προγράμματος βλέπουμε τον χρόνο εκτέλεσης σε ένα νήμα και 8 νήματα αντίστοιχα:

```
Simple vector addition example (15000000 elements)
Total GPU memory -75956224, free -194969600

Vector addition on CPU
Execution time 16.76 msecs

Vector addition on CPU (multithreads)
Execution time 7.86 msecs

Vector addition on GPU
Execution time 1.15 msecs
```

2. OpenMp multiple GPU

Η εφαρμογή **simple_add2GPUOmp** αξιοποιεί τις δυο κάρτες γραφικών στον κόμβο για την εκτέλεση του πυρήνα (kernel). Με χρήση του μοντέλου OpenMp το πρόγραμμα διακλαδώνεται σε δυο νήματα, και το κάθε νήμα καλεί μια GPU για την επεξεργασία ενός μέρος των δεδομένων.
Για την προβολή των δεδομένων για τις δυο κάρτες γραφικών από το nvprof χρησιμοποιούμε την επιλογή –print-summary-per-gpu. Όπως παρουσιάζεται στο παρακάτω script.

```
#!/bin/bash

# Which Queue to use, DO NOT CHANGE #
#PBS -q GPUq

# Max Wall time, Example 1 Minute #
#PBS -l walltime=00:01:00
```

```
# How many nodes and tasks per node, 1 node with 20 tasks/threads 2 GPU
#PBS -lselect=1:ncpus=20:ompthreads=20:ngpus=2 -lplace=shared

# Only this job uses the chosen nodes
#PBS -lplace=excl

# JobName #
#PBS -N myGPUJob

#Change Working directory to SUBMIT directory
cd $PBS_O_WORKDIR

# Run executable #
nvprof --print-summary-per-gpu ./simple_add
```

Αφού τελειώσει η εκτέλεση του προγράμματος στο αρχείο myGPUJob.e[jobID] καταγράφονται τα δεδομένα των δυο GPUs

```
==162941== NVPROF is profiling process 162941, command: ./simple_add
==162941== Profiling application: ./simple_add
==162941== Profiling result:

==162941== Device "Quadro P4000 (0)"
            Type  Time(%)     Time     Calls       Avg       Min       Max  Name
 GPU activities:   56.09%  10.631ms         2  5.3156ms  5.1780ms  5.4531ms  [CUDA memcpy HtoD]
                   40.87%  7.7475ms         1  7.7475ms  7.7475ms  7.7475ms  [CUDA memcpy DtoH]
                    2.31%  437.56us         1  437.56us  437.56us  437.56us  kadd(float*, float*, float*, int)
                    0.73%  139.19us         1  139.19us  139.19us  139.19us  [CUDA memset]

==162941== Device "Quadro P4000 (1)"
            Type  Time(%)     Time     Calls       Avg       Min       Max  Name
 GPU activities:   65.24%  9.6868ms         2  4.8434ms  4.8413ms  4.8455ms  [CUDA memcpy HtoD]
                   30.88%  4.5855ms         1  4.5855ms  4.5855ms  4.5855ms  [CUDA memcpy DtoH]
                    2.94%  436.28us         1  436.28us  436.28us  436.28us  kadd(float*, float*, float*, int)
                    0.94%  139.03us         1  139.03us  139.03us  139.03us  [CUDA memset]
```

Στην έξοδο του προγράμματος παρουσιάζεται ο χρόνος εκτέλεση σε 2 νήματα στον επεξεργαστή και τις δυο κάρτες γραφικών:

```
Simple vector addition example (15000000 elements)
Total GPU memory -75956224, free -194969600

Vector addition on CPU
Execution time 16.52 msecs

Vector addition on GPU
CPU(0)|GPU(0):Execution time 0.58 msecs
```

Ο χρόνος εκτέλεσης του πυρήνα (kernel) στην έξοδο του προγράμματος διαφέρει από τα δεδομένα του nvprof διότι υπολογίζεται από τον επεξεργαστή “συμπεριλαμβάνει την κλήση του πυρήνα και την επιστροφή του ελέγχου”.