IMPROVING GPU UTILIZATION WITH MULTI-PROCESS SERVICE (MPS)

PRIYANKA,
COMPUTE DEVTECH, NVIDIA
GPU parallelizable part
CPU parallel part
Serial part

STRONG SCALING OF MPI APPLICATION

Multicore CPU only
GPU accelerated CPU

With Hyper-Q/MPS
Available in K20, K40, K80
WHAT YOU WILL LEARN

- Multi-Process Server
- Architecture change (HyperQ - MPS)
- MPS implication on Performance
- Efficiently utilization of GPU under MPS
- Profile and Timeline
- Example
WHAT IS MPS

> CUDA MPS is a feature that allows multiple CUDA processes to share a single GPU context. Each process receives some subset of the available connections to that GPU.

> MPS allows overlapping of kernel and memcpy operations from different processes on the GPU to achieve maximum utilization.

> Hardware Changes - Hyper-Q which allows CUDA kernels to be processed concurrently on the same GPU.
REQUIREMENT

- Supported on Linux
- Unified Virtual Addressing
- Tesla with compute capability version 3.5 or higher, Toolkit - CUDA 5.5 or higher
- Exclusive-mode restrictions are applied to the MPS server, not MPS clients
ARCHITECTURAL CHANGE TO ALLOW THIS FEATURE
CONCURRENT KERNELS

- GPU can run multiple independent kernels concurrently
  - Fermi and later (CC 2.0)
  - Kernels must be launched to different streams
  - Must be enough resources remaining while one kernel is running

- While kernel A runs, GPU can launch blocks from kernel B if there are sufficient free resources on any SM for at least one B block
  - Registers, shared memory, thread block slots, etc.

- Max concurrency: 16 kernels on Fermi, 32 on Kepler
  - Fermi further limited by narrow stream pipe...
Kepler allows 32-way concurrency

- One work queue per stream
- Concurrency at full-stream level
- No inter-stream dependencies
CONCURRENCY UNDER MPS

Kepler allows 32-way concurrency

- One work queue per stream, 2 work queue per MPS Client
- Concurrency at 2 stream level per MPS client, total 32
  - Case 1: N_stream per MPS Client < N_channel (i.e. 2), no serialization
Kepler allows 32-way concurrency

- One work queue per stream, 2 work queue per MPS Client
- Concurrency at 2 stream level per MPS client, total 32
  - Case 2: \[ N\_stream > N\_channel \] - False dependency/serialization

Multiple Hardware Work Queues/Channel
HYPER Q/MPI (MPS): SINGLE/MULTIPLE GPUS PER NODE

MPS Server efficiently overlaps work from multiple ranks to single GPU

MPS Server efficiently overlaps work from multiple ranks to each GPU

Note: MPS does not automatically distribute work across the different GPUs. Inside the application user has to take care of GPU affinity for different mpi rank.
HOW MPS WORK

Process 1 initiated before MPS Server started

MPS Server

Many to one context mapping

All MPS Client Process started after starting MPS server will communicate through MPS server only

MPS Client

MPI Process 2 - Create CUDA context

Allows multiple CUDA processes to share a single GPU context
HOW TO USE MPS ON SINGLE GPU

• No application modifications necessary

• Proxy process between user processes and GPU

• MPS control daemon
  • Spawn MPS server upon CUDA application startup

• Setting
  • export CUDA_VISIBLE_DEVICES=0
  • nvidia-smi -i 0 -c EXCLUSIVE_PROCESS
  • nvidia-cuda-mps-control -d

• Enabled via environment variable (for CRAY)
  export CRAY_CUDA_PROXY=1
**USING MPS ON MULTI-GPU SYSTEMS**

**Step 1:** Set the GPU in exclusive mode
- `sudo nvidia-smi -c 3 -i 0,1`

**Step 2:** Start the mps daemon (In first window) & Adjust pipe/log directory
- `export CUDA_VISIBLE_DEVICES= ${DEVICE}`
- `export CUDA_MPS_PIPE_DIRECTORY=${HOME}/mps${DEVICE}/pipe`
- `export CUDA_MPS_LOG_DIRECTORY=${HOME}/mps${DEVICE}/log`
- `nvidia-cuda-mps-control -d`

**Step 3:** Run the application (In second window)
- `Mpirun -np 4 ./mps_script.sh`
- `NGPU=2`
- `lrank=$MV2_COMM_WORLD_LOCAL_RANK`
- `GPUID=$(($lrank%$NGPU))`
- `export CUDA_MPS_PIPE_DIRECTORY=${HOME}/mps${DEVICE}/pipe`

**Step 4:** Profile the application (if you want to profile your mps code)
- `nvprof -o profiler_mps_mgpu$lrank.pdm ./application_exe`

---

**Not required in CUDA 7.0**

(for MV2_COMM_WORLD_LOCAL_RANK for mvapich2,OMPI_COMM_WORLD_LOCAL_RANK for openmpi)
NEW IN CUDA 7.0

Step 1: Set the GPU in exclusive mode

```bash
sudo nvidia-smi -c 3 -i 0,1
```

Step 2: Start the mps daemon (In first window) & Adjust pipe/log directory

```bash
export CUDA_VISIBLE_DEVICES= ${DEVICE}
nvidia-cuda-mps-control -d
```

Step 3: Run the application (In second window)

```bash
lrank=${OMPI_COMM_WORLD_LOCAL_RANK}
case ${lrank} in
  [0]) export CUDA_VISIBLE_DEVICES=0; numactl --cpunodebind=0 ./executable;;
  [1]) export CUDA_VISIBLE_DEVICES=1; numactl --cpunodebind=1 ./executable;;
  [2]) export CUDA_VISIBLE_DEVICES=0; numactl --cpunodebind=0 ./executable;;
  [3]) export CUDA_VISIBLE_DEVICES=1; numactl --cpunodebind=1 ./executable;
(esac
```
GPU UTILIZATION AND MONITORING MPI PROCESS RUNNING UNDER MPS OR WITHOUT MPS

- GPU Utilization by different MPI Rank under MPS
- GPU Utilization by different MPI Rank Without MPS

Two MPI Rank per processor sharing same GPU
Step 1: Launch MPS daemon
- $ nvidia-cuda-mps-control -d

Step 2: Run nvprof with --profile-all-processes
- $ nvprof --profile-all-processes -o application_exe_%p
- ======= Profiling all processes launched by user "user1"
- ======= Type "Ctrl-c" to exit

Step 3: Run application in different terminal normally
- $ application_exe

Step 4: Exit nvprof by typing Ctrl+c
- ==5844== NVPROF is profiling process 5844, command: application_exe
- ==5840== NVPROF is profiling process 5840, command: application_exe...
- ==5844== Generated result file: /home/mps/r6.0/application_exe_5844
- ==5840== Generated result file: /home/mps/r6.0/application_exe_5840
VIEW MPS TIMELINE IN VISUAL PROFILER

Import Multi-Process nvprof Data

Browse and Select Files
PROCESS SHARING SINGLE GPU WITHOUT MPS: NO OVERLAP

Process 1 - Create CUDA context
Process 2 - Create CUDA context

Allows multiple processes to create their separate GPU context

Kernel from Process 1
Kernel from Process 2

Two context corresponding to two different MPI Rank are created
PROCESS SHARING SINGLE GPU WITHOUT MPS: NO OVERLAP

Process 1 - Create CUDA context
Process 2 - Create CUDA context

Allows multiple processes to create their separate GPU context
PROCESS SHARING SINGLE GPU WITH MPS: OVERLAP

Allows multiple processes to share single CUDA Context.
PROCESS SHARING SINGLE GPU WITH MPS: OVERLAP

Process 1

MPS Server

Process 2

Allows multiple processes to share single CUDA Context.

Two process launch kernel in default stream.

Kernel from Process 1

Kernel from Process 2
CASE STUDY: HYPER-Q/MPS FOR ELPA
Sharing the GPU between multi MPI ranks increases GPU utilization.

Enables overlap between copy and compute of different processes.
EXAMPLE: HYPER-Q/PROXY FOR ELPA

Performance Improvement with MPS on single GPU

Problem Size 10K, EV-50%

- Hyper-Q with multiple MPI ranks on single node sharing same GPU under MPS leads to 1.5X speedup over multiple MPI rank per node without MPS

Performance Improvement with MPS on multiple GPU

Problem Size 15K, EV-50%

- Hyper-Q with half MPI ranks on single processor sharing same GPU under MPS leads to nearly 1.4X speedup over MPI rank per processor without MPS
CONCLUSION

- Best for GPU acceleration for legacy applications
- Enables overlapping of memory copies and compute between different MPI ranks
- Ideal for applications with
  - MPI-everywhere
  - Non-negligible CPU work
  - Partially migrated to GPU
REFERENCE

- S5117_JiriKraus_Multi_GPU_Programming_with_MPI

- Blog post by Peter Messmer of NVIDIA -
  http://blogs.nvidia.com/blog/2012/08/23/unleash-legacy-mpi-codes-with-
  keplers-hyper-q/
Email : priyankas@nvidia.com

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!