TECHNISCHE UNIVERSITÄT DRESDEN
Center for Information Services and High Performance Computing (ZIH)

OpenACC Support in Score-P and Vampir

Hands-On for the Taurus GPU Cluster

February 2016

Robert Dietrich (robert.dietrich@tu-dresden.de)
Ronny Tschüter (ronny.tschueter@tu-dresden.de)

ZIH
Center for Information Services & High Performance Computing

Simple Reduction with OpenACC

```
double reduceAddOpenACC_kernel(double *data, int size){
  double sum = data[0];

  #pragma acc kernels
  for (int i = 1; i < size; i++) {
      sum = sum + data[i];
  }

  return sum;
}
```

OpenACC kernels

```
double reduceAddOpenACC_parallel(double *data, int size){
  double sum = data[0];

  #pragma acc parallel loop copyin(data[1:size-1]) reduction(+:sum)
  for (int i = 1; i < size; i++) {
      sum = sum + data[i];
  }

  return sum;
}
```

OpenACC parallel

Login, Compile, Allocation

- Login to Taurus & load modules

```
> ssh taurus.hrsk.tu-dresden.de
> module add scorep/2.0oacc-pgi16.1-ompi-cuda7.0
  # loads dependencies: pgi/16.1 cuda/7.0.28 openmpi/1.10.2-pgi16.1 …
  # specifies OpenACC profiling library: libscorep_adapter_openacc_mgmt.so
```

- Compile

```
> scorep --cuda --openacc pgcc -acc -Minfo=all -ta=nvidia acc_reduction.c
  -o acc_reduction
  # “--cuda“ and “--openacc“ tells Score-P to link the CUDA and OpenACC adapter
```

- Allocate resources (1 core, 1 GPU, 1 hour) & get an interactive bash

```
> salloc --ntasks=1 --gres=gpu:1 --time=60
> srun --pty bash -l -i
```

Setup Tracing

- Setup Score-P measurement (`> scorep-info config-vars --full`)

```
> export SCOREP_ENABLE_TRACING=true
> export SCOREP_EXPERIMENT_DIRECTORY=scorep_acc_trace
> export SCOREP_TOTAL_MEMORY=50M
> export SCOREP_METRIC_PAPI=PAPI_FP_OPS
```

- Runtime filtering to reduce the trace file size

```
> export SCOREP_FILTERING_FILE=scorep.filt
```

```
> cat scorep.filt
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    acc_launch*
    cuEvent*
SCOREP_REGION_NAMES_END
```

Setup OpenACC Measurement

- Setup Score-P OpenACC measurement

```
> export SCOREP_OPENACC_ENABLE=yes #or a comma-separated list of ...

            regions: OpenACC regions
               wait: OpenACC wait operations
            enqueue: OpenACC enqueue operations (kernel, upload, download)
       device_alloc: Track allocations on the accelerator
  kernel_properties: Record kernel properties such as the kernel name as well
                     as the gang, worker and vector size for kernel launch
                     operations
     variable_names: Record variable names for OpenACC data allocation and
                     enqueue upload/download
```

Setup CUDA Measurement

- Setup Score-P CUDA measurement

```
> export SCOREP_CUDA_ENABLE=driver,kernel,memcpy,flushatexit

  runtime:        CUDA runtime API
  driver:         CUDA driver API
  kernel:         CUDA kernels
  kernel_counter: Fixed CUDA kernel metrics
  memcpy:         CUDA memory copies
  sync:           Record implicit and explicit CUDA synchronization
  idle:           GPU compute idle time
  pure_idle:      GPU idle time (memory copies are not idle)
  gpumemusage:    Record CUDA memory (de)allocations as a counter
  references:     Record dependencies between CUDA activities (CASITA)
  flushatexit:    Flush CUDA activity buffer at program exit


> export SCOREP_CUDA_BUFFER=1M

  # Total memory in bytes for the CUDA record buffer (per process)


> export SCOREP_CUDA_BUFFER_CHUNK=8k

  # Chunk size in bytes for the CUDA record buffer
```

Execution and Trace Visualization

- Execute the instrumented program

```
> salloc --ntasks=1 --gres=gpu:1 --time=60
> srun --pty bash -l -i
> ./acc_reduction
OR
> srun --ntasks=1 --cpus-per-task=1 --gres=gpu:1 --time=10 ./acc_reduction
```

- Trace visualization with Vampir

```
> module add vampir
> srun --ntasks=1 --cpus-per-task=2 --time=15 --pty --x11=first vampir
  scorep_acc_trace/traces.otf2
```

Score-P Profiling

- Setup Score-P profiling

```
> export SCOREP_ENABLE_PROFILING=true
> export SCOREP_ENABLE_TRACING=false
> export SCOREP_EXPERIMENT_DIRECTORY=scorep_acc_profile

> ./acc_reduction

> scorep-score scorep_acc_profile/profile.cubex
```

```
Estimated aggregate size of event trace:                   1177 bytes
Estimated requirements for largest trace buffer (max_buf): 1177 bytes
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       7MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=7MB to avoid intermediate
flushes
 or reduce requirements using USR regions filters.)

flt     type max_buf[B] visits time[s] time[%] time/visit[us]  region
         ALL      1,176     49    0.27   100.0         5462.34  ALL
         USR      1,176     49    0.27   100.0         5462.34  USR
```

- Profile visualization with Cube

```
> module add cube

> srun --ntasks=1 --cpus-per-task=1 --time=15 --pty --x11=first cube
  scorep_acc_profile/profile.cubex
```

Score-P OpenACC Hands-On

- Compile CMAKE projects with Score-P (> scorep-wrapper --help)

e.g. Intel C/C++ Compiler

```
> cmake \
> -DCMAKE_CXX_COMPILER=scorep-icpc \
> -DCMAKE_C_COMPILER=scorep-icc \
> -DCUDA_HOST_COMPILER=/sw/global/compilers/intel/2015/.../icc \
> -DMPI_C_COMPILER=scorep-mpicc \
> -DMPI_CXX_COMPILER=scorep-mpicxx
```

e.g. PGI Fortran Compiler

```
> cmake \
> -DCMAKE_Fortran_COMPILER=scorep-pgf90 \
> -DMPI_Fortran_COMPILER=scorep-mpif90
```

- Set Score-P wrapper flags

```
> make SCOREP_WRAPPER_INSTRUMENTER_FLAGS="--cuda --openacc"
```

Advanced Instrumentation

- Selected Score-P compiler wrapper options (`scorep --help`)

```
--nocompiler    Disables compiler instrumentation.
(useful for C++ programs with many short running functions)

--verbose       Shows what scorep actually does.

--user          Enables user instrumentation. (see Score-P documentation)

--openacc       Enables OpenACC instrumentation.

--cuda          Enables CUDA instrumentation.

--opencl        Enables OpenCL instrumentation.
```

Automatic Trace Analysis with CASITA (1)

- CASITA workflow (for OpenACC & CUDA)

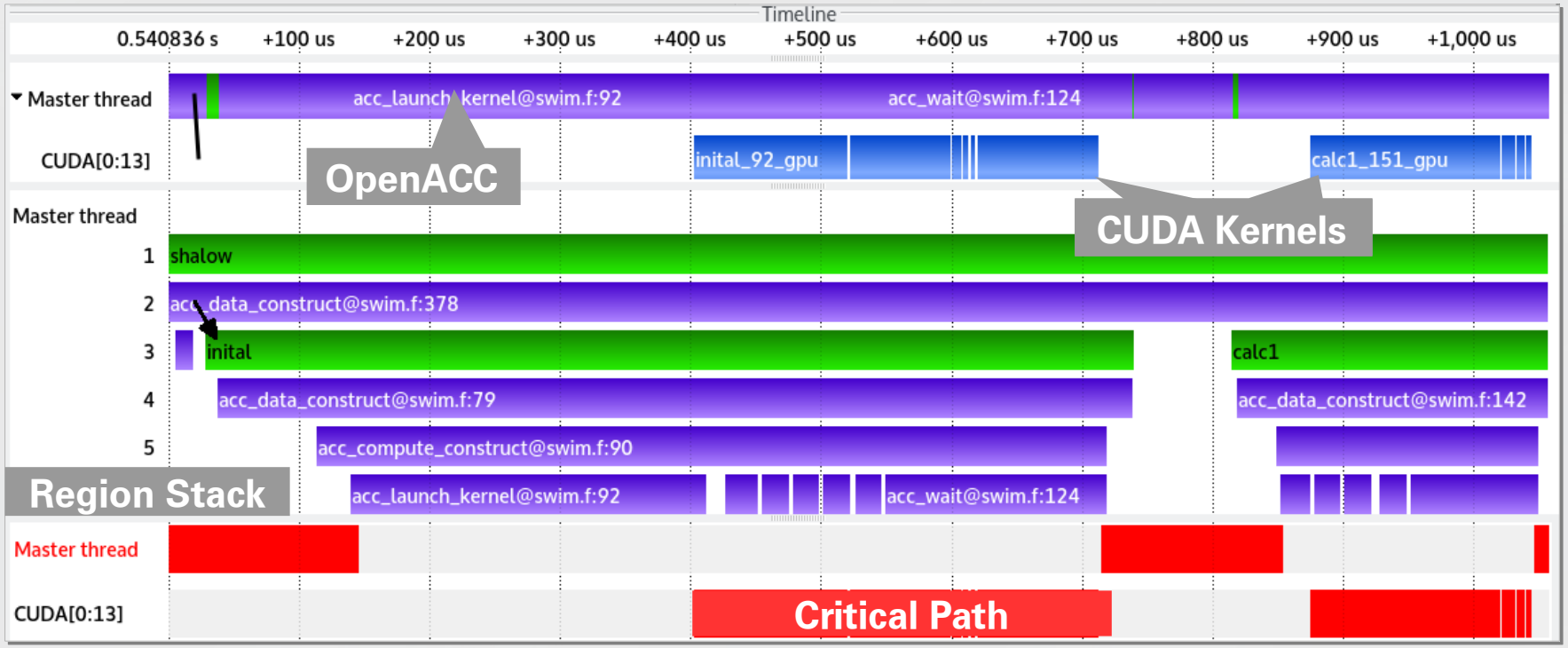
```
> export SCOREP_CUDA_ENABLE=driver,kernel,memcpy,flushatexit,references

> casita --help
  -s … create summary file
  -o … generate OTF2 trace file with analysis counters
  …

> mpirun -n N casita scorep_oacc/traces.otf2 -s -o scorep_oacc/casita.otf2
   N … same number of processes as used for program execution
```

- Supported paradigms: MPI, OpenMP, CUDA (and OpenACC)
- Rating identifies regions with high potential to optimize the program execution

```
Running CASITA 1.4.1 with 1 analysis processes
Generate optimization rating:
                                  Activity Group  Instances Time (sec)  Time on CP  Fraction CP Fraction Global Blame    Rating
                                     main_36_gpu          1   0.000017    0.000017        0.00%               100.00%  1.000038
                          cuDevicePrimaryCtxRetain          1   0.274880    0.274880       62.82%                 0.00%  0.628184
                       cuDevicePrimaryCtxRelease            1   0.105226    0.105226       24.05%                 0.00%  0.240474
                                            main          1   0.023916    0.023916        5.47%                 0.00%  0.054655
                                   cuMemHostAlloc         1   0.023641    0.023641        5.40%                 0.00%  0.054026
                                    cuMemFreeHost         1   0.007100    0.007100        1.62%                 0.00%  0.016226
                                    cuMemAlloc_v2         4   0.000730    0.000730        0.17%                 0.00%  0.001669
      acc_data_construct_enter@vecAddOpenACC.c:35         1   0.000383    0.000383        0.09%                 0.00%  0.000876
                                                      ...
                  acc_upload@vecAddOpenACC.c:35           2   0.000015    0.000015        0.00%                 0.00%  0.000033
-------------------------------------------------
                                  Sum of Top 20          52   0.437539    0.437479       99.98%               100.00%
CASITA analysis took 0.040000 seconds.
```

Automatic Trace Analysis with CASITA (2)

- CASITA output trace can be visualized with Vampir



- Analysis is based on wait states:
  - **Critical path analysis**
  - Cause effect analysis / Blame shifting to quantify load imbalances

Build Score-P with OpenACC Support (for PGI)

- Score-P configuration (configure --help=recursive)

```
> configure --prefix=/scorep/install/dir \
  --with-nocross-compiler-suite=pgi \
  --with-openacc-include=/path/to/pgi/include \ #openacc.h & #acc_prof.h
  --with-libcudart=/path/to/cuda/toolkit \
  --with-libcudart-include=/path/to/pgi-cuda/include \ #PGI patched CUDA head.
  --with-libOpenCL=/path/to/opencl \
  --enable-shared
```

  Make sure that paths point to CUDA libraries and PGI CUDA headers of the same CUDA version!

- Setup Score-P environment (module) for OpenACC
  - Set ACC_PROFLIB to point to Score-P's OpenACC measurement library
    > setenv ACC_PROFLIB /path/to/scorep/lib/libscorep_adapter_openacc_event.so*
    > export ACC_PROFLIB=/path/to/scorep/lib/libscorep_adapter_openacc_event.so*