OpenPOWER GPU-enabled architecture performance enhancement using the Engineering and Scientific Subroutine Library (ESSL) drop-in acceleration

Manish Modani
Aditya Nitsure
Thomas J. Heller
Joan McComb

July 31, 2017

This article illustrates the methodology to offload the part of computations to GPU without refactoring the applications. The Crossroads/NERSC-9 Memory Bandwidth benchmark is used to showcase the offload of dense matrix multiplication (DGEMM) computations on GPU by linking (compile time) the newer version of CUDA-enabled ESSL (IBM Scientific Library). The use of CUDA-enabled ESSL gives approximately six times performance gain over CPU-only code.

Introduction

Accelerated OpenPOWER systems based on the IBM, NVIDIA, and Mellanox collaboration offer a new potential for scalability and performance. These systems have GPUs (coprocessors) to increase the price or performance of the system. In case of accelerated systems, part of the application runs on processors and remaining part runs on coprocessors or GPUs.

To offload work on GPU, programmers should call the CUDA functions (for array declaration on GPU, copy data from CPU to GPUs and GPUs to CPUs, and so on) inside the applications. This is not trivial as it requires code modification, CUDA awareness, and compilation. Although OpenACC pragmas made life a bit easy (with instructions based offloading), it still requires efforts for the programmer to understand the application and its flow, and then decide what portion of the application can be offloaded to the GPUs.

CUDA-enabled ESSL

ESSL and Parallel ESSL are the collections of state-of-the-art mathematical subroutines specifically designed to improve the performance of engineering and scientific applications on the IBM® POWER® processor-based servers. ESSL and Parallel ESSL are commonly used in the aerospace, automotive, electronics, petroleum, utilities, and scientific research industries for applications.

To enable support for accelerated GPU-enabled OpenPOWER architectures, newer version (for example, version 5.5) of ESSL is now GPU enabled. This version of ESSL offloads a subset of the mathematical
functions on GPU to enable faster computations and then collects back the results on CPU. The process is seamless and transparent to users. There is no need to modify the application and compilation procedure. Only during the linking stage, you need to link CUDA-enabled ESSL (for example, libesslsmpcuda.so) instead of ESSL (for example, libessl.so). This article illustrates the process to exploit the performance of GPU-enabled OpenPOWER architecture using CUDA-enabled ESSL and performance gain.

**System Configuration**

To illustrate the advantage of CUDA-enabled ESSL, IBM S822LC is used. Table 1 provides the specifications of the system.

**Table 1. S822LC system details**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>RHEL 7.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores per node</td>
<td>20</td>
</tr>
<tr>
<td>CPU frequency</td>
<td>4.02 GHz</td>
</tr>
<tr>
<td>GPUs</td>
<td>NVIDIA Tesla P100</td>
</tr>
<tr>
<td>GPUs per node</td>
<td>4</td>
</tr>
<tr>
<td>ESSL</td>
<td>5.5</td>
</tr>
</tbody>
</table>

**Example program**

Crossroads/NERSC-9 DGEMM compute benchmark (version: 1.0.0)

The Crossroads/NERSC-9 Memory Bandwidth benchmark is a simple single-node multi-threaded dense-matrix multiply benchmark. The code is designed to demonstrate high floating-point compute rates on a system under sustained computation.

**Compilation**

**Figure 1. Makefile showing the linking of CPU-enabled ESSL library**
Figure 2. Makefile showing the linking of CPU- and GPU-enabled ESSL library

![Makefile screenshot]

Execution

This section describes the environment setting and execution of the DGEMM benchmark.

Environment setting

Following environment variable are set for the execution.

```bash
export PATH=/usr/local/cuda/bin:$PATH
export OMP_NUM_THREADS=20
export OMP_PLACES={0:20:8}
export OMP_WAIT_POLICY=active
export OMP_PROC_BIND=TRUE
unset CUDA_VISIBLE_DEVICES
export CUDA_VISIBLE_DEVICES=0,1,2,3
```

Run command

The executable/binary (mt-dgemm) takes the size of the matrix as an input argument. The following command can carry out a 4096 X 4096 size matrix multiplication.

```bash
./mt-dgemm 4096
```

Results

In the section, the performance of the DGEMM code is elaborated when DGEMM is:

- Compiled without ESSL
- Linked with ESSL
- Linked with CUDA-enabled ESSL (refer Table 2)

The GPU-enabled ESSL shows **five to six times better performance** than the CPU-only ESSL version.

Table 2. Performance comparison [Matrix size (N) = 4096]

<table>
<thead>
<tr>
<th></th>
<th>Time (in sec)</th>
<th>GFLOPS</th>
<th>FLOPS computed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without ESSL</td>
<td>1287.04</td>
<td>3.20</td>
<td>4124175237120</td>
</tr>
</tbody>
</table>

OpenPOWER GPU-enabled architecture performance enhancement using the Engineering and Scientific Subroutine Library (ESSL) drop-in acceleration
<table>
<thead>
<tr>
<th>ESSL SMP – CPU only</th>
<th>8.67</th>
<th>475.22</th>
<th>4124175237120</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSL SMP – CUDA enabled (with four GPUs)</td>
<td>1.62</td>
<td>2535.28</td>
<td>4124175237120</td>
</tr>
</tbody>
</table>

Figure 3. GPU utilization (in case of libesslcudasmp) through NVIDIA Visual Profiler (NVVP)

Resources

- Refer to the documentation on Using the ESSL SMP CUDA Library
- For benchmark code, connect with Joan McComb, Lead, ESSL Development.

© Copyright IBM Corporation 2017

Trademark
(www.ibm.com/developerworks/ibm/trademarks/)