Praktikum im WS 2022
Quantitative Analyse von Hochleistungssystemen (LMU)
Evaluation of Modern Architectures and Accelerators (TUM)

Vincent Bode, MSc., Minh Chung, MSc., Dennis-Florian Herr, MSc., Bengisu Elis, MSc., Dr. Karl Fürlinger, Amir Raoofy, MSc., Dr. Josef Weidendorfer

Assignment 05 – Due: 08.12.2022

In the second assignment, you worked with GPUs in BEAST and specifically focused on the performance characteristics (peak performance, memory bandwidth) of the Vector Triad. In this assignment we focus on Matrix Multiplication microbenchmarks on GPUs. For this purpose, you will adapt the code to use OpenMP offloading directives and to utilize GPU resources in BEAST, i.e., AMD MI100 and Nvidia V100 GPUs.

Before going ahead with tests and analysis on the microbenchmarks, it is important to keep in mind that we investigate Matrix Multiplication as a compute-bound benchmark. This information may be helpful when explaining the observed results from the microbenchmarks.

Matrix Multiplication (MM)

For this set of tasks please use the provided matrix-matrix multiplication (MM) implementation similar to assignment #4 called `assignment5.cpp`. This implementation already offloads MM to GPU by OpenMP directives.

1. **Offloading with Optimal Data Transfer Policy:** Make sure that the MM computation is offloaded to GPU. Take a look back at task #3 in assignment #2 Based on the results of that tasks, choose the best data initialization policy and make sure this policy is used to initialize and migrate your matrices to GPU.

2. **Optimizations:** Unlike Assignment #4, where we used loop interchange (reordering the loops) as the first step, here we use another technique for optimization. For this, you need to introduce two variants of matrix multiplication benchmark, both with the same "ijk" loop ordering.

   - **Variant 1 :** Store the elements of both multiplicand $\mathbf{B}$ and multiplier $\mathbf{C}$ in row-major layout.
   - **Variant 2 :** Use row-major layout for multiplicand $\mathbf{B}$ and column-major layout for multiplier $\mathbf{C}$.

   For both variants, complete the following tasks:

(a) Apply cache blocking (similar to Assignment #4)

(b) Use appropriate loop scheduling policies and vectorization directives based on your experience in Part I of this Assignment.

(c) Run similar experiments to those in Part I task #6 of assignment #2 to measure FLOP rates and memory bandwidth utilization of your code on GPU.

(d) Explain your results and compare them to the results you achieved in Assignment #4, for matrix multiplication benchmark on CPUs.

3. **Execution Configuration on Target Device:** In assignment #2, you learned how to use OpenMP clauses that configure the execution of teams and threads on target devices in practice. Use these clauses for the following subtasks:

   (a) Make sure parallelism is achieved by using OpenMP clauses on the GPUs by setting the number of teams = 100 and the number of threads = 128.

   (b) Use all possible combinations of the league and team size combinations from the following set: $\{(t, T)\}$ = { 8, 16, 32, 40, 48, 64, 72, 80, 88, 96, 104, 112, 120, 128, 256, 512, 1024} $\otimes$ { 32, 64, 80, 128, 256, 512, 1024} and plot the flop rate vs. league/team size combination plot in 3D (or heatmap). Here, "t" denotes the number of threads per team, "T" denotes the number of teams. Perform this task only for a single matrix size for each thread and team size combination but make sure the matrix size is large enough. Explain your observations for each system on BEAST with GPUs and compare your results with the results of task #5 of assignment #2.

   (c) Find the optimum team number vs thread number configuration at which you get the maximum performance and explain the reason for it. You can use this optimal combination for the following task.

4. **Power Measurements:**

   To take power measurements, use The Data Center Data Base (DCDB) system monitoring framework installed on the BEAST gateway node.

   - To prepare and start using DCDB you should run the following command on the gateway:
     `source /opt/dcdb/dcdb.bash`.
   - You can list available ac power sensors by the following command:
     `dcdbconfig sensor list|grep power_ac`.
   - Sensor values and timestamps can be read by the following command:
     `dcdbquery /beast/<node name>/<sensor name> now-5m now`
     This command outputs the average power measurements from 5 minutes before until now, reported by DCDB every 5 seconds.

   Power Distribution Unit (PDU) is a device with multiple power outlets designed to distribute electric power, especially to racks of computers and networking equipment located within a data center. Power consumption from each outlet can be measured, for example, the power measured for by outlet #5 of PDU #3 is measured and stored in DCDDB as sensor "*(/beast-/pdu3/power5)*". Note that each BEAST node has 2 PDUs for redundancy, and the total power consumption of a node is the sum of measurements taken from both PDUs. The *power_ac* virtual sensor output performs this summation of two PDUs. In addition, the power measurements reported by DCDB are obtained by averaging PDU-internal power measurements that are taken every 30 msecs.

Also note that for correct and interference-free measurements, you need to determine and impose sleep time before and after your benchmark runs.

For CPU only and best-performing GPU versions of your code:

(a) Present your results for speed up and power consumption, in a table.

(b) Plot Mflops per Watt (Mflops/Watt) vs data set size results, similar to previous assignments' Mflops/sec vs data set size plots.