Praktikum im SS 2021
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: 25.11.2021

In the previous assignment, we focused on experimenting with the vector triad microbenchmark on GPU-based systems, by enabling offloading through OpenMP. In this assignment, we investigate the performance of the Matrix Multiplication microbenchmark ($\mathbf{A} = \mathbf{B} * \mathbf{C}$) as a compute-bound microbenchmark. For this, we use the same matrix multiplication microbenchmark you developed in Assignment #2 and adapt it to use OpenMP offloading directives to use GPU resources in BEAST, i.e., AMD MI50 and Nvidia V100 GPUs. Please refer to Assignment #4 for compiler usage for offloading.

1. **Basic Tasks**

   (a) **Porting with Optimal Data Transfer Policy:** The first step is to adapt your code from Assignment #2 to implement a matrix-matrix multiplication microbenchmark that offloads the computation to GPUs using OpenMP offloading. Use the compilers available on BEAST systems according to the instructions in Assignment #4, make sure that the multiplication is offloaded to GPU. From your results of Assignment #4 task (b), choose the best data initialisation policy and use this policy to initialise and migrate your matrices to GPU.

   (b) **Optimizations:** Unlike Assignment #2, where we used loop interchange (reordering the loops) as the first step, 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:

   i. Apply cache blocking (similar to Assignment #2)

   ii. Use appropriate loop scheduling policies and vectorization directives, based on your experience in Assignment #4.

   iii. Run similar experiments to those in Assignment #4 to measure FLOP rates and memory bandwidth utilization of your code on GPU.

iv. Explain your results and compare it to the results you achieved in Assignment #2, for matrix multiplication benchmark on CPUs.

(c) **Execution Configuration on Target Device:** In the lecture you learned OpenMP clauses that configure execution of teams and threads on target devices, and in Assignment #4 you learned how to use them in practice. Use these clauses for the following subtasks:

i. Try to achieve as much parallelism as you can by using OpenMP clauses by adjusting your loop iterations, and number of teams/threads.

ii. Use the combinations of league and team size configurations from the following set: $\{(t, T)\} = \{1, 2, 4, 8, 32, 64, 128, 256, 512,1024\} \times \{1, 2, 4, 8, 32, 64, 128, 256, 512, 1024\}$ and plot the flop rate vs. league/team size combination plot in 3D. Here, "t" denotes the number threads per team, "T" denotes the number of teams, and $\times$ denotes the cartesian cross operator. Make sure you are using vector lengths that are large enough.

iii. 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 tasks.

(d) **Power Measurements:** Repeat the power measurement experiments from Assignment #4 task (d) for the matrix matrix multiplication. Use the guidelines and commands in Assignment #4. For CPU only and single GPU versions of your code:

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

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