Praktikum im WS 2022/2023
Quantitative Analyse von Hochleistungssystemen (LMU)
Evaluation of Modern Architectures and Accelerators (TUM)

Vincent Bode, MSc., Minh Chung, MSc., Sergej Breiter, MSc., Bengisu Elis, MSc., Dr. Karl Fürlinger, Amir Raoofy, MSc., Dr. Josef Weidendorfer

Assignment 00 – Due: 27.10.2022

This warm-up assignment tries to make sure everything is fine with the BEAST system. The 1st part is intended to check system access, module availability, and some exploring hints with compiler flags that might be helpful for further assignments. The 2nd part is to familiarize yourself with the basic usage of a CPU profiling tool - **Likwid** [1]; further profiling tools for CPU-GPU will be introduced in future assignments.

1. **System Access and Module Check**

   (a) Make sure you can login to all the provided systems on BEAST and should ensure the right node (`username @nodename>`, e.g., `@nodename` can ONLY be one of `ice1, rome2, thx2, cs2`). Furthermore, you should check other critical jobs running by other users by `top` or `htop` before running your experiment of assignments. On the GPU side (Rome or Thx nodes), we can use `rocm-smi` or `nvidia-smi` to check GPU-status.

   (b) Make sure the module is loaded. By default, the standard GNU compiler is provided. For example, to check and load a different available version:

```
$ lscpu                        # show details of system architecture
$ module avail                 # show all compilers/packages available
$ module load <package-name>   # to load a package
$ module unload <package-name> # to unload it
```

   (c) Additionally, play around with some optimization options from compiler.

Listing 1: A code snippet about branching miss/hit

```
int i;
int sum = 0;
for (i = 0; i < N; i++)
    if (i && 0)        // FFFF...
    // if (i || 1)     // TTTT...
    // if (i && (i%2)) // FTFT...
    // if (rand()%2)   // Random TF...
        sum++;
```

[1] https://github.com/RRZE-HPC/likwid/wiki

The snippet shows a small example that intends to see how many branches miss/hit on modern processors. Now let's explore it with different optimization flags (a reference code is in `\src`). To see what happens with these flags (e.g., *assembly* code comparison), you can check at `https://godbolt.org/`. Lets choose a system, compile and run the code, then quickly answer the questions in a short reference report in `\report`.

2. **Profiling Tools - Likwid**
   Profiling is an essential part of understanding and optimizing performance. Profiling means counting a set of performance events that occur during execution. Today's processors support hardware performance events via their performance monitoring units (PMUs). Events are counted using the available (limited number) hardware performance monitoring counters of a PMU.

   Likwid is a set of command line performance analysis tools. The tool `likwid-perfctr`[2] facilitates configuring and accessing performance counters and is introduced step by step.

   (a) Login to one of the BEAST systems and type the following commands:

```
$ module load likwid             # required on every login
$ likwid-perfctr -e | less       # list available performance events
$ likwid-perfctr -a              # list available performance groups
```

   The list of available performance events can be overwhelming and depends on the architecture.

   (b) To facilitate profiling, Likwid provides predefined *performance groups*. Multiple events are measured simultaneously in a group to derive particular performance metrics from them. Type the following commands:

```
$ likwid-perfctr -g MEM -H       # show memory bandwidth group
$ likwid-perfctr -g FLOPS_DP -H  # show double precision flops group
```

   This shows the formulas used by Likwid to calculate memory bandwidth utilization (data volume per time transferred between memory and CPU) and double precision flop rate (floating-point operations per time)–two examples for useful performance metrics– from hardware performance events on the current architecture.

   (c) In the next step, we measure performance groups during a single-threaded run of the `copy` and `peakflops` benchmarks from the Likwid benchmark suite. The option `-C 0`[3] specifies to pin the thread to core 0, and also to measure performance events on core 0.

```
# copy 1GB of data:
$ likwid-perfctr -C 0 -g MEM      likwid-bench -t copy      -w C0:1GB:1
# double precision (scalar) floating-point operations on 1MB of data:
$ likwid-perfctr -C 0 -g FLOPS_DP likwid-bench -t peakflops -w C0:1MB:1
```

   (d) Compare the performance metrics calculated by software (output of `likwid-bench`) to the corresponding performance metrics derived from hardware performance events (output of `likwid-perfctr`). The flop rates shown by the `peakflops` benchmark and `likwid -perfctr` should closely match. Did you find a mismatch in the `copy` benchmark for memory bandwidth? Can you think of why this may be the case?

   Submission: your group just needs to submit the reference report in `\report` by the deadline.

[2]see `$ likwid-perfctr -h` or `$ man likwid-perfctr`
[3]see `$ man likwid-pin`