Suggested Questions

It is not necessary to do all the activities and answer all the questions. For example, you can skip problem 1 and go directly to Problem 2; you can skip problem 4, or problem 5 or 6.

Problem 1: Explain the performance of the three initial code versions

1. Compile, run and profile the three files implementing the C code versions: ***HeatBase.c***, ***HeatBaseT.c*** and ***Heat.c***. You can find the Linux commands for compiling, running and profiling in a LOG file published in the Campus Virtual. Try to run the corresponding Linux commands by yourself and compare the published results with yours. If you find any significant difference, please ask your teacher.

Note: The number of **cache references** measured by **perf** approximates the number of times that the processor has requested to read/write a value from/to memory and the data was not present in the Level-1 and Level-2 cache memories, but it was present in the Level-3 cache memory (with a capacity of 18MiBytes = 18×1024×1024 Bytes).

Note: The number of **cache misses** measured by **perf** indicates an approximation of the number of times that the processor has requested to read/write a value from/to main memory, and the data was not present in the Level-1, Level-2 and Level-3 cache memories, and had to be read from or written to Main Memory (MM).

2. From the performance results shown in the Campus Virtual, compute the average number of clock cycles required for executing each stencil operation, the average number of machine instructions executed for each stencil operation, the IPC ratio, the average number of cache references per stencil operation, and the average number of cache misses per stencil operation.

3. From the data computed in the previous question, the data obtained using the **perf record** and **perf report** tools (published in the Campus Virtual), and the different values of "*system time*" provided by the profiler, explain the performance of each version.

Problem 2: Optimize your code

4. Modify the code **Heat.c** to include optimized versions as the ones described in the complementary document as **HeatV2**, **HeatV3** and **HeatV4**. For these versions use the problem size: **N**=100,000, **T**= 100,000.

5. Improve the code with optimized versions as the ones described in the complementary document as **HeatV5** and **HeatV6**. For these versions use the problem size: **N**=100,000, **T**= 100,000.

Problem 3: Check the effect of the vector size on performance

6. Since the problem size is proportional to both **N** and **T**, we fix the product of **N** and **T** to 10 giga (thousands of millions), and executions will **always perform 10 giga stencils**. Execute your best performing version of the code to process 10G stencils for **N**= 5K, 10K, 100K, 1M and 10M. The performance results for **HeatV4** are published in Campus Virtual.

7. From the performance results shown in the Campus Virtual (or your own results if you have designed a better version), compute the average number of clock cycles required for executing each stencil operation, the average number of machine instructions executed for each stencil operation, the IPC, the average number of cache references per stencil operation, and the average number of cache misses per stencil operation.

8. From the data computed in the previous question, explain the performance of each execution with a different **N** value.

Problem 4: Analyze the performance characteristics of the Memory Hierarchy

Version **V4** of the program performs two floating-point additions and two floating-point multiplications on each iteration of the inner loop; i.e., performs a ratio of 4 FLOAT operations per stencil. We can also consider that each iteration of the inner loop reads 3 data elements and writes one data element, which accounts for 4 × sizeof (float) = 16 Bytes per stencil.

9. Indicate the theoretical **arithmetic intensity** of the program. It is the ratio of FLOAT operations per Byte transferred between the processor and the memory. Compute the actual **FLOP/second** ratio achieved by the best performing case for the V4 version (it is achieved for a small vector size). Also, compute the actual memory bandwidth, measured as **GB/s**, considering the best performing case. In this scenario, the L1 cache memory is practically the only element responsible for providing all the data to the processor core.

When the vector size grows, part of the data moved between the processor and memory comes from the L2 cache, L3 cache or even main memory. Still, part of the data requested by the program for reading is provided by the L1 cache.

10. Estimate the amount of data moved between the processor and the L2 cache, the L3 cache or the Main Memory, and compute the maximum effective bandwidth achieved by the executions of version V4 of the program.

Problem 5: Design a multi-threaded version

11. Parallelize version V4 or V5 using the **loop parallelism** directives provided by OpenMP and execute the program for **N**= 10K, 100K, 1M and 10M. Scale the problem size so that the product of **N** and **T** is always 100 Giga. Check with your teacher the performance results obtained.

Problem 6: Design a Massively-Parallel version for GPU using OpenACC

Our goal is to add **OpenACC** directives to versions V4 or V5 to create a GPU-accelerated code. The next commands try to achieve that goal using the program file ***HeatACC.c***

```
$ module add nvhpc/21.2                                    # install NVIDIA tools
$ nvc -V                                                   # Look at CPU microarchitecture

$ nvc -fast -acc=host -Minfo=all HeatACC.c -o HeatCPU
       # compile for host CPU. A long message appears explaining optimizations
       # the executable HeatCPU can be executed, but it is slower than
       # the executable compiled with gcc. This is something out of scope

$ nvc -fast -tp=nehalem -acc -gpu=cc80 -Minfo=accel HeatACC.c -o HeatGPU
       # compile for GPU of generation cc80, and CPU with microarchitecture nehalem
   Heat_V4:
        98, Generating implicit copyin(Uin[:N+2]) [if not already present]
            Generating implicit copyout(Uout[1:N]) [if not already present]
       100, Complex loop carried dependence of Uin-> prevents parallelization
            Loop carried dependence of Uout-> prevents parallelization
            Loop carried backward dependence of Uout-> prevents vectorization
            Accelerator serial kernel generated
            Generating Tesla code
           100, #pragma acc loop seq
       100, Loop carried dependence of Uout-> prevents parallelization
```

12. The previous compilation command generates a message summarizing the results of the parallelization analysis of the inner loop of version V4. First, implicit copies of data between **host** and **device** have been generated. Look carefully at the line into the code where these copies are inserted, and the size of the blocks copied. Second, the inner loop is found to contain recurrent dependences, though we know that this is not true, and the iterations are completely independent. Modify the code using a **parallel loop** OpenACC construct instead of a **kernel** construct. Once the compiler message indicates that the loop has been parallelized you can execute the program using the next **sbatch** command, which sends your execution **job** to a remote queue and remote computer where it will eventually run. Results from the execution will appear on file ***Result.txt***. You can check the state of the job with the command **squeue**. If something goes wrong, you can cancel your job using the command **scancel <jobid>**, where ***jobid*** is the job identifier, a number. If you want to run this command remotely (from outside of the Laboratory), then you should change **cuda-int.q** by **cuda-ext.q** and **aolin-gpu-4** by **aolin-gpu-3**.

```
$ sbatch -p cuda-int.q -w aolin-gpu-4 --gres=gpu:1 -o Result.txt runGPU.txt HeatGPU 10000000
   1000 4
       # submit execution job to partition cuda-int.q and computer node aolin-gpu-1

$ sbatch -p cuda-int.q -w aolin-gpu-4 --gres=gpu:1 -o Profile.txt profGPU.txt HeatGPU 10000000
   1000 4
       # submit profiling job to GPU node
```

13. Explicitly handle the data copies in a more efficient way by including your own **data copy** OpenACC constructs in the source file. Ask your teachers for help, especially when trying to understand the profiling results.