

## **Programming Project 1: Due Date: Sunday Oct 8th, 19.55 (100 points)**

To be conducted in groups of 4. When you run your programs, you may compare them against other nodes in your group, if the specifications of other machines (nodes) are different. Again, please produce an extensive report that displays the results of your experimentation, describes your code, and answers all questions addressed.

### **Part 1: (60 points)**

**Question:** Write an ISPC program that computes the square root of 20 million random numbers (real, not necessarily integers) between 0 and 8 (*Hint:* Use Newton's method to solve the equation:  $\frac{1}{x^2} - S = 0$ ). Please use multiple iterations for sqrt to converge to an accurate solution (accurate solution is one that gives  $< 10^{-4}$  difference compared to the true value).

ISPCs SPMD execution model (and libraries), like “foreach”, implement SIMD processing. ISPC also provides a mechanism that “launches ISPC tasks”, utilizing this way multiple cores in an ISPC computation. Check the `launch[num]` command in function `mandelbrot_ispc_withtasks` and in the Intel ISPC Spec. This command launches *num* tasks. Each task defines a computation that will be executed by a “thread” of ISPC gangs/instances. As given by the function `mandelbrot_ispc_task`, each task computes a region of the final image in any order (in parallel on different CPU cores).

What is the ISPC implementation speedup for single CPU core (no tasks launched) and when using multiple cores (with tasks)? What is the speedup due to SIMD parallelization? What is the speedup due to multi-core parallelization? Extend your code to utilize 2, 3, 4, 5, 6, 7, 8 threads, partitioning the computation accordingly. In your write-up, produce a graph of speedup compared to a sequential implementation (which you need to implement as well) as a function of the number of cores and threads used. Is speedup linear in the number of cores used? In the number of threads used? Please justify why yes or why not.

Then write another version of the sqrt function using AVX intrinsics. You may consult the Intel Intrinsics Guide: <https://software.intel.com/sites/landingpage/IntrinsicsGuide/>.

If you wish to tell the ISPC compiler to generate AVX, rather than SSE instructions, please see the Makefile comment about ISPC's `--target=avx2-i32x8` compiler flag. For slightly better performance use `--target=avx2-i32x16`, which will use a gang size of 16 instances and use two AVX instructions to implement an operation for the entire gang. (more details here: <https://ispc.github.io/ispc.html#selecting-the-compilation-target>).

## **Part 2: (40 points)**

**Question:** Implement a simple **password cracker** using ISPC. You are going to make a password which is between 1-8 ascii characters. You will then use one of the popular hash functions to convert your password to a hash. You are expected to do adequate research on this topic and list your results in a detailed report. So, let's assume that the attacker (you-the group-the programmer) is exposed only to the hash of your password and to no other information. Another question to answer: what is a hash function? What are the properties of a hash function? When the attacker gets a hashed password can he directly know what the hash function is? Or can he retrieve or investigate the hash? How does the attacker eventually obtain the hash so that the attacker can brute-force or even use a rainbow table to try any combination of characters through the hash in order to obtain the same hash value? So, you are asked to do exactly the same. You are to implement your own password cracker based on the rules above. You are provided the range of the password length, which is to be between 1-8 characters. Provided this information, your group should brute force the password, i.e., take every possible combination of words between the mentioned length, convert it into a hash and compare it with the provided hash. Yet a couple of extra questions to answer: i) When to use a Dictionary attack vs. a Rainbow Table attack? ii) What are the resource requirements for each type of attack? Which uses more storage? Which requires more pre-computation? Which requires more analysis time? (per-hash vs. batch cracking)

This task can be easily parallelized. You are expected to use and exploit exactly the concepts worked upon in Project 1 Part 1 in the best possible way to provide the fastest times for your password cracker. And you may select various and diverse ways of your problem decomposition and assignment with the end goal to achieve optimal workload utilization which may of course – under circumstances – which do you believe? – lead also to fastest times of password cracking. To facilitate your understanding on the concept of assignment, let's consider a scenario for which thread 1 checks only combinations of length 1, thread 2 does that for 2 letter words, thread 3 for 3 letter words etc. So, you first generate the hashes and then matching them in bulk via your dedicated hardware (what type of HW in particular should you be using here?).

Similar to Part 1: What is the ISPC implementation speedup for single CPU core (no tasks launched) and when using multiple cores (with tasks)? What is the speedup due to SIMD parallelization? What is the speedup due to multi-core parallelization? Extend your code to utilize 2, 3, 4, 5, 6, 7, 8 threads, partitioning the computation accordingly. In your write-up, produce a graph of speedup compared to a sequential implementation (which you need to implement as well) as a function of the number of cores and threads used. Is speedup linear in the number of cores used? In the number of threads used? Please justify why yes or why not.

You may also conduct some search on the web and utilize in addition an existing software tool for password cracking. Does your tool lend itself easily to parallel execution? You are expected to do some research and find one that does. Conduct a similar experiment as the one with your own password cracker. Compare the results and discuss them in your report.

**Note:** Also experiment with a number of different password characters and report the difference in the timing results.

All groups in addition to your own imaginary passwords, also use the following to compare against all groups and all varying implementations: bv37qi#f. Apply a number of popular hashes and obtain a number of versions of the password. And you take it from there...

**You are encouraged to run your code on the orbit testbed in WINLAB. Use your ORBIT account per group. Time slots are reserved everyday normally between 20.00-12.00am but also randomly earlier in the day. The ISPC compiler is already installed in “mariasfirstimage” but you may download ISPC at <http://ispc.github.com/>.**