CS 610 Semester 2023–2024-I: Assignment 3

24th September 2023

Due Your assignment is due by Oct 8, 2023, 11:59 PM IST.

General Policies

- You should do this assignment ALONE.
- Do not copy or turn in solutions from other sources. You will be PENALIZED if caught.

Submission

- Submission will be through Canvas.
- Submit a PDF file with name "<roll-no>.pdf". We encourage you to use the LaTeX typesetting system for generating the PDF file.
- Briefly explain your implementations, results, and other issues of interest (if any). Include the exact compilation instructions for each programming problem.
- Name each source file "<roll-no-probno>.cpp" where "probno" stands for the problem number (e.g., "23111545-prob2.cpp").
- You will get up to TWO LATE days to submit your assignment, with a 25% penalty for each day.

Evaluation

- Your solutions should only use concepts that have been discussed in class.
- Write your code such that the EXACT output format is respected.
- We will evaluate the implementations with our OWN inputs and test cases, so remember to test thoroughly.
- We may deduct marks if you disregard the listed rules.

Problem 1 **[20+40 marks]**

Consider the following code snippet.

```
#define N (1 << 12)

double x[N], y[N], z[N], A[N][N];

for (int i = 0; i < N; i++) {
 for (int j = 0; j < N; j++) {
  y[j] = y[j] + A[i][j]*x[i];
  z[j] = z[j] + A[j][i]*x[i];
 }
}
```

Perform loop transformations to improve the performance of the following code snippet for sequential execution on one core (i.e., no multithreading). We will provide a template code to help with the computation.

1. Modify the `optimized()` function in the template code and perform evaluations.
2. Implement a version of the best-performing variant from the previous step using AVX2 intrinsics. Implement it as a separate function.

Briefly explain the reason for the poor performance of the provided reference version, your proposed optimizations (with justifications) to improve performance, and the improvements achieved. Time the different versions and report the speedup results with the GNU GCC compiler. Note that GCC does not auto-vectorize at optimization level `-O2`, you need to pass `-O3`. Evaluate part (i) with both optimization levels `-O2` and `-O3`.

For part (i), you may use any transformation we have discussed in class. For example, you can try to align statically or dynamically allocated memory and are free to use constructs like `__restrict__` and `#pragma GCC ivdep`. No array padding or layout transformation of arrays (e.g., 2D $\rightarrow$ 1D) are allowed.

Some transformations will lead to speedup, some will not. Include all the transformations that you tried, even if they did not work. Finally, summarize the set of transformations (e.g., 32 times unrolling + `xyz` permutation) that gave you the best performance.

Problem 2 **[30 marks]**

Assume an array of type `long`. The size of the array is 2^{24}. Implement computing the sum of the elements of an array using the following strategies.

(a) OpenMP `parallel for` version *without* reductions. You are allowed to use synchronization constructs like `critical` or `atomic`.

(b) an OpenMP `parallel for` version *using* reductions,

(c) OpenMP tasks where each task will work on a sub-array of size 1024.

You are encouraged to implement additional versions for better performance, especially with Part (c).

Problem 3 **[20+40+40 marks]**

Implement versions of an *inclusive* prefix sum function using (i) SSE4 intrinsics, (iii) AVX2 instrinsics, and (iv) AVX-512 intrinsics. Compare the performance with the sequential and OpenMP versions, and report performance speedups.

Include the compiler version and the exact compilation command. It is okay to tweak the provided compilation command to suit your compiler version and the target architecture. You should refer to the GCC manual for details.

Use the template code provided with the description. There are multiple algorithms to compute prefix sum in parallel. Try to come up with implementations that perform the best for you (e.g., allow the compiler to vectorize the loops).

Refer to `https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#` for documentation on Intel Intrinsics.

Problem 4 **[30 marks]**

Parallelize the attached C code (`problem4-v0.c`) using OpenMP. Compare the performance of the OpenMP program with the sequential version. You may use any combination of valid OpenMP directives and clauses that you think will help.

You should submit a file `roll-no-problem4-v3.c[pp]`. You are allowed to switch to C++ for your solutions. Use a workstation in the KD lab for your performance evaluation. Report your speedup using the GNU gcc compiler. You can compile the reference code with `gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L prob4-v0.c -o prob4-v0.out`. Include the name of the workstation in your report. The TAs will reuse the same system to reproduce the performance results.