Scientific and Technical Computing
Homework #2 (Hardware and Code Optimization)

Part 1: Compiler

General Description

In this homework you will compile and execute code on Frontera's Cascade Lake nodes. You will report execution times and inspect optimization reports. Based on the data collected you will reason about optimization strategies.

General Instruction

Everything in blue will require a response in your report.

Posting date: 10/26/20

Return date: 11/09/20, afternoon (2 weeks after posting)

How to submit: Submit your pdf file through Canvas

What to submit: Short document maybe 2 pages.

Instructions: Logon and 'idev'

1. Logon to Frontera
   `ssh <username>@frontera.tacc.utexas.edu`
2. Queue: You will use the Cascade Lake development queue for the exercise.
3. Use 'idev' to start an interactive session
   a. Review the document https://portal.tacc.utexas.edu/software/idev
   b. While you can edit and compile code on the login node (that is the main purpose of the login nodes) you *must not* run the code on the login nodes. Executing code on the login nodes will impede other users and you will get *incorrect timing results.*
   c. Use the 'idev' tool for the interactive session
      Review the 'idev' help page by typing **`idev -h`**
      Start the interactive session by typing: **`idev -p development`**
      What does the option **`-p`** do?
      You will get the prompt back when the interactive session has started. Note that it may take a while for the interactive session to start. You may have to continue your work when the system is less busy.
      On the login node you may check the status of the queue by typing
      `showq -l | grep devel`

Trouble shooting

1. You may be asked for the number of 'tasks per node'. Any input will do, i.e. just hit return, or enter 1, or enter 56.
2. The 'idev' tool selects a project to account for the usage. Most likely you will have only one project, which will then be automatically selected. If you are prompted for a project, select the appropriate project.
3. If you get errors you may remove the idev resource file
   `rm -f ~/.idevrc`

Hint

Your interactive session will expire after 30 minutes. Your session will close and you will be back on the login node. If your session expires type **`idev -p development`** again to start a new interactive session.

a. You can extend the time to 2 hours by adding the option **`-m 120`** to the **`idev`** command.

e. Familiarize yourself with the command **`hostname`** (man page or internet search). What does the command 'hostname' do?
f. Inside the interactive session type **`hostname`**. Add the name of the compute node to your report. Example:
g. Report multiple hostnames, if you are using multiple interactive sessions. Use as many interactive sessions as necessary.

Instructions: Setup

- Create a directory where you will conduct the experiment. Choose a name and add the location (full path) to your report. You may use the command **`pwd`** to get the full path.
- Copy the source file: **`cp ~train00/stc20/vector.c .`**
  Hint: do not forget the destination, i.e. the dot '.' at the end.
- Inspect the source file. The function **`gettimeofday()`** is being used to derived the elapsed time. Only the code between the timing statements is of interest. Answer these questions:
  1. What is the size of the arrays **`a`**, **`b`**, and **`c`**? Report the size in terms of (1) number of elements and (2) in number of bytes, K-bytes, or M-bytes? Report the information in a table.
  2. How does the total size of the 3 arrays compare to the size of the 'Level 3' cache of a single socket of a Frontera Cascade Lake node? Refer to the Stampede2 user guide for the size of the L3 cache: https://portal.tacc.utexas.edu/user-guides/frontera
  3. We are diagnosing and timing the code between the timer calls. There are 2 calls, one to start the timer, one to stop it. What is the purpose of the 2 innermost loops? You may use 'matrix' and 'vector' in your explanation.
  4. Why is there a third and outer loop? What does the outer loop do? Use the timings that you are reporting to explain the purpose of the outermost loop. What would happen, if the outermost loop were removed?

Instructions: Compiling and Executing

1. Generally the code is compiled with this command:
   `icc -xcore-avx512 vector.c`
   Add additional flags as necessary and instructed.
2. Create 3 different executables adding these options
   a. **`-O1`**
   b. **`-O2`**
   c. **`-O2 -no-vec`**
3. Run the executables and report the timing. Run every executable at least 3 times. Create a table and report the minimum time and the average time for each executable.
4. Now compile with
   a. **`-O2 -qopt-report-phase=vec -qopt-report=3`** to create a detailed optimization report.
   b. Explain the purpose of these 2 compilation flags that generate the report. Use the man page for the compiler (**`man icc`**, or **`icc -help`**) and/or the internet as a resource.
5. Inspect the optimization report
   a. Report the file name that contains the optimization report.
   b. What are the line numbers of the loop nest (3 loops) of interest?
   c. The optimization report mentions loop reordering.
   d. Which loops does the compiler reorder?

e. What is the actual reordered code? Add the 5 lines of source code that reflect the actual loop order, to your report. Clarification: Your report should contain 5 lines of code that reflect the loop structure (and the loop kernel) **after** reordering. See above. Either order is fine.
f. Explain how the reordering changes the access pattern to the arrays a and b. Please discuss access to both arrays separately.
g. Why is the access pattern of the reordered code better, i.e. why/how does this reordering lead to better performance?

6. What does the option –no-vec do? Use the man page for 'icc' or the internet as a resource. Why is the execution time with the flag –no-vec larger?
7. Why is the execution time with –O1 larger? You may create an optimization report with **–O1** to understand he difference between **–O1** and **–O2**.

Best of luck!