Homework #1: Hardware and Code Optimization

**Introduction**

Interactive access to a set of compute nodes allows for users to quickly compile, run and validate MPI and other applications in rapid succession, which is very useful for development. The app, “idev”, stands for “Interactive Development”, and was created by TACC to allow users to acquire compute nodes for interactive access. In this assignment, the “idev” application will be used to access the “Cascade Lake development queue” to edit, compile and run a program, “vector.c” to study code optimization. The icc compiler will be used to explore the effect that different compilation schemes have on the run-time/performance of the program.

**Results and Discussion**

**Instructions: Logon and “idev”**

1. **Starting an interactive session**

When the command “idev -p development” is run in Frontera, an interactive session is started in the terminal. As shown in the figure below, a command prompt appears in the interactive development session and status messages are presented that report the condition of the job running on a masternode and the condition of the initialization of a development environment.

1. **What does the option “-p” do?**

* The “-p” flag creates a development queue in the environment

1. **What does the command “hostname” do?**

* The command “hostname” returns the computer’s DNS (Domain Name System), which is a name given to a computer, and attached to a network to uniquely identify that computer over the network.

1. **– f) Add the name of the compute node to your report and report multiple hostnames if necessary.**

* After running the command inside the author’s interactive session, the names of the two compute nodes the author ran are “c202-001.frontera.tacc.utexas.edu”, and “c203-005.frontera.tacc.utexas.edu”, and “c203-001.frontera.tacc.utexas.edu”, “c203-018.frontera.tacc.utexas.edu”,

**Instructions: Setup**

1. **Create a directory where you will conduct the experiment. The directory where the author conducted the experiment is specified through the following full path:**

**/home1/07674/jcam98/hwk\_1\_h\_code\_op**

1. **What is the size of the arrays in the “vector.c” file?**

A:

|  |  |  |  |
| --- | --- | --- | --- |
| Array Size Type | a | B | c |
| Number of Elements | 1024 | 1048576 | 1024 |
| Number of Bytes | 8.0 kB | 8.0 Mb | 1. kB |

1. **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?**

A: The size of a “Level 3” cache of single socket of a Frontera Cascade Lake node is 38.5 Megabytes which is significantly larger than the size of arrays “a”, and “b”, and approximately four times the size of array “B”.

The total size of the three arrays is roughly one-quarter of the socket’s full memory.

1. **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 innermost 2 loops? You may use “matrix” and “vector” in your explanation.**

**A: The purpose of the innermost 2 loops in the nested for loop between the start and the end of the timer is to iterate through the entries in the vectors “a”, and “c”, and matrix “B” to update the value of “a” according to the operation “a = B \* c”.**

1. **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?**

**A: The third and outer loop is used**

**Instructions: Compiling and Executing**

1. **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.**

A:

|  |  |  |
| --- | --- | --- |
| Executable (by compiler flag) | Minimum Run-Time (sec) | Average Run-Time (sec) |
| -01 | 31.614 | 31.771 |
| -02 | 3.341 | 3.355 |
| -02 -no-vec | 6.469 | 6.470 |

**5b) Explain the purpose of the two compilation flags in the command “-O2 -qopt-report-phase=vec -qopt-report=3” that create a detailed optimization report.**

**A: The first compilation flag specifies the phase of the optimization; in this case, the report characterizes optimization performed on vectors in the “vector.c” program. The second flag tells the compiler the level of detail with which to express the report in. A level of 3 indicates that information will be presented in a moderate level of detail.**

**6. Inspect the Optimization Report**

1. **Report the file name that contains the optimization report.**

**A: The file name that contains the report is “vector.optrpt”**

1. **What are the line numbers of the loop nest (3 loops) of interest?**

**A: The line numbers of the loop nest (loops of interest that were vectorized) are 26,36, and 48.**

1. **The optimization report mentions loop reordering.**
2. **Which loops does the compiler reorder?**

**A: The compiler reordered the loops that began at lines 34 and 37.**

1. **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.**

**A:** for (n = 0; n < iter; n++) {

a[0] = 0.;

for(i = 0; i < N; i++)

for(j = 0; j < N; j++)

a[i] = a[i] + B[i][j]\*c[j];

1. **Explain how the reordering changes the access pattern to the arrays a and b. Please discuss access to both arrays separately.**

**A: Reordering the for loops impacts the access pattern to the arrays as it influences the speed with which data can move from the memory to the cache, and ultimately to the CPU. This in turn impacts the amount of time it takes for the CPU to process the arithmetic operation in the final line of the for loop. The ordering of the loops affects the distance that the data of the loop counters “i” and “j” must move to and from the main memory and the CPU, and the greater the distance, the greater the run-time of the loops. When the loops are ordered such that the outermost loop iterates over the loop counter “i” and the innermost loop iterates over the loop counter “j”, the compiler first accesses elements in array “c” and then respectively “B” and “a” as the arrays that operate on the loop counter “j” are accessed first. The converse is true for an ordering where the innermost loop iterates over “i” and the outermost loop over “j”, namely, the compiler accesses elements in array “a”,”B”, and “c” respectively.**

1. **Why is the access pattern of the reordered code better, i.e. why/how does this reordering lead to better performance?**

**A: The access pattern of the reordered code leads to better performance because the data of the loop counter “j” , that are used in the more computationally-intensive, arithmetic operation (B[i][j] \* c[j]), travel over a lesser distance to and from the main memory and the CPU than the data of the loop counter, “i” that are used in the less computationally-intensive arithmetic operation (addition and assignment respectively from right to left). Reordering the loop so that the innermost loop counter iterates over the loop counter (j) used in the more computationally-intensive arithmetic operation helps to minimize the distance that the data must travel between cycles/iterations and thus, improves latency and minimizes runtime.**

**7) What does the option “-no vec” do? Use the man page for “icc” or the internt as a resource. Why is the execution time with the flag “no-vec” larger?**

**A:**

**8) Why is the execution time with “-O1” larger? You may create an optimization report with “-O1” to understand the difference between “-O1” and “-O2”.**

**A:**