Multi-Threaded Application Performance Analysis

Running Multi-threaded Benchmarks on Multicore Hardware

1. You will re-run the multi-threaded DaCapo benchmarks on Jikes again, but this time you will vary the number of application threads. Use the same Jikes setup as with project 3. But this time, download the new running script from the website (runJikesDaCapo_varyAppThreads.sh), which will allow you to vary the number of application threads of the benchmarks (four multi-threaded benchmarks are specified in the script). Notice that the parameters that you have to pass to the script have changed, and are now:

   ```
   <BENCHMARK> <COLLECTOR> <HEAP_MULTIPLIER> <NUM_APP_THREADS>
     <TIMING_ITERATION>
   ```

   You have to specify the benchmark (dacapo for all 4 benchmarks), collector (you can pick GenImmix), heap multiplier (2 times the minimum heap size is fine), then the number of application threads you want to use, then the timing iteration (2 is again fine because we are using replay compilation). Run the multi-threaded applications with 1, 2, 4, and 8 application threads. [If your machine has only 2 cores, you can skip the 8 application threads data point.] Look at the execution times as the number of application threads increases. Does it always go down? If no, why not? Are the benchmarks in general scalable as you increase the number of threads?
2. Based on Amdahl's Law [Speedup = 1 / (f/n + (1-f))], calculate for each of the multi-threaded benchmarks what f must be when going from 1 to 4 application threads, and from 1 to 8 (if applicable). Do you conclude that a large portion of these benchmarks is parallelized?

Visualizing Multi-threaded Benchmarks

3. Analyze the provided bottle graphs (download the 1 page from the website) for only the multi-threaded applications. The multi-threaded applications were run with 4 application threads and 2 garbage collection threads, using the GenImmix collector. How does this set of benchmarks do in regards to parallelism? In other words, does each of the applications scale well to 4 application threads? Do the bottle graphs reflect what you saw in the benchmark runs that you did in step 1?
4. Check out the pmd benchmark in particular. See the figure below in this document that comes from the bottle graph paper (Figure 11), which shows the bottle graphs for varying numbers of application threads for pmd. What can you conclude about pmd's scalability? Do you know what the bottleneck is or how to fix it based on just analyzing the bottle graph?
5. Check out the figure below in this document that comes from the bottle graph paper and compares the scaling of the GenImmix garbage collector when the number of threads is varied, keeping the application constant (Figure 3). What can you conclude about Jikes' GenImmix collector's scalability? Do you know how to improve it from this picture?

6. Now look at the speedup stacks paper that you downloaded from the course webpage (please do not give this paper out to others). Look particularly at Figure 3 (which shows speedup stacks for 4 DaCapo benchmarks when running with the GenImmix collector at 2x the minimum heap size) and Figure 4 (which shows supporting performance counter data). You can compare what the speedup stacks of these 4 benchmarks show about scalability with your experimental runs in step 1, and with the same benchmarks' bottle graphs from step 3. Do you see the same scalability problems with these benchmarks as in the bottle graphs? Or do you see different bottlenecks illuminated? Do speedup stacks give you enough information to eliminate these bottlenecks or know how to fix them? I want you to compare and contrast these 2 visualization tools and report on their advantages and disadvantages.

Multi-threaded Analysis Tool

7. **Choose one of the following performance analysis tools** below to explore. I would like each of you to pick a different one, so **please email me when you've made your choice** and I will tell other students that they cannot pick that tool. These tools all should work on multi-threaded applications, so try to download the tool (a trial version if it was developed by a company), and run it, if possible (meaning don't spend a whole day trying to get it to work if it's not working). Report back on what the tool is good for and what is it bad at. Can the tool you chose help you find and solve scalability issues such as the ones you found above? Is there something else you would have liked a multi-threaded performance analysis tool to provide you, as a developer and debugger, to be more useful to help improve scalability?
   - VTune (https://software.intel.com/en-us/intel-vtune-amplifier-xe)
   - VisualVM(http://visualvm.java.net/gettingstarted.html)
   - WAIT(https://wait.ibm.com/)
   - Jconsole (http://docs.oracle.com/javase/8/docs/technotes/guides/management/jconsole.html)
   - Jprofiler (http://www.ej-technologies.com/products/jprofiler/overview.html)
   - YourKit (https://www.yourkit.com/java/profiler/)
   - Valgrind (http://www.valgrind.org/)
   - Choose your own…

Additional Information

If you want to check out what the bottle graph paper's authors concluded about the GenImmix collector's scalability, and what was going on with pmd, check out the full paper (only after you've written your report):
http://soft.vub.ac.be/~jsartor/researchDocs/opsla100b-dubois.pdf

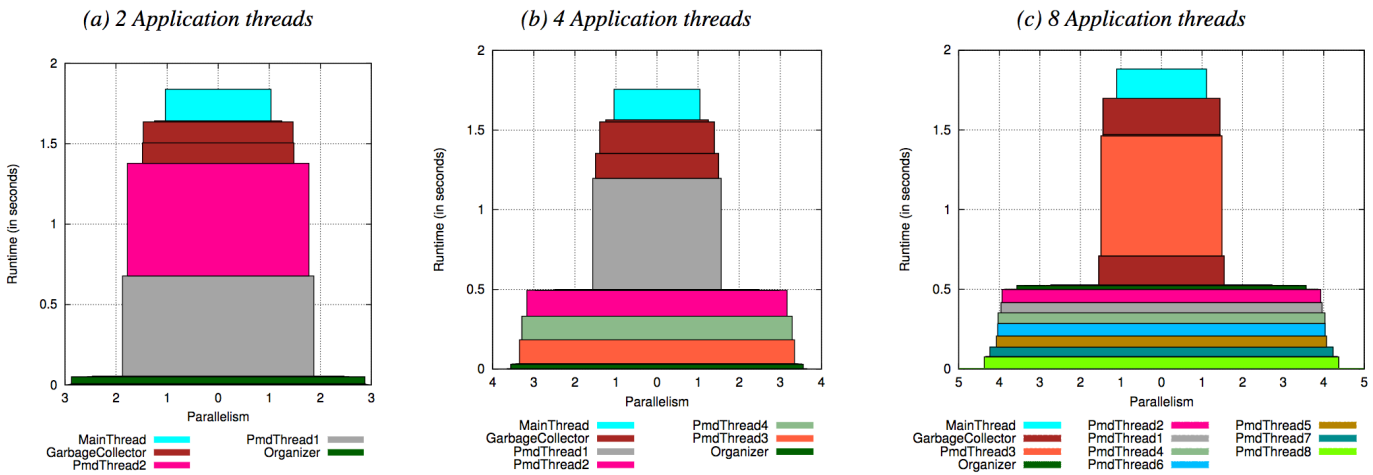

Figure 11. Pmd: scaling of application threads with 2 GC threads (default input set).

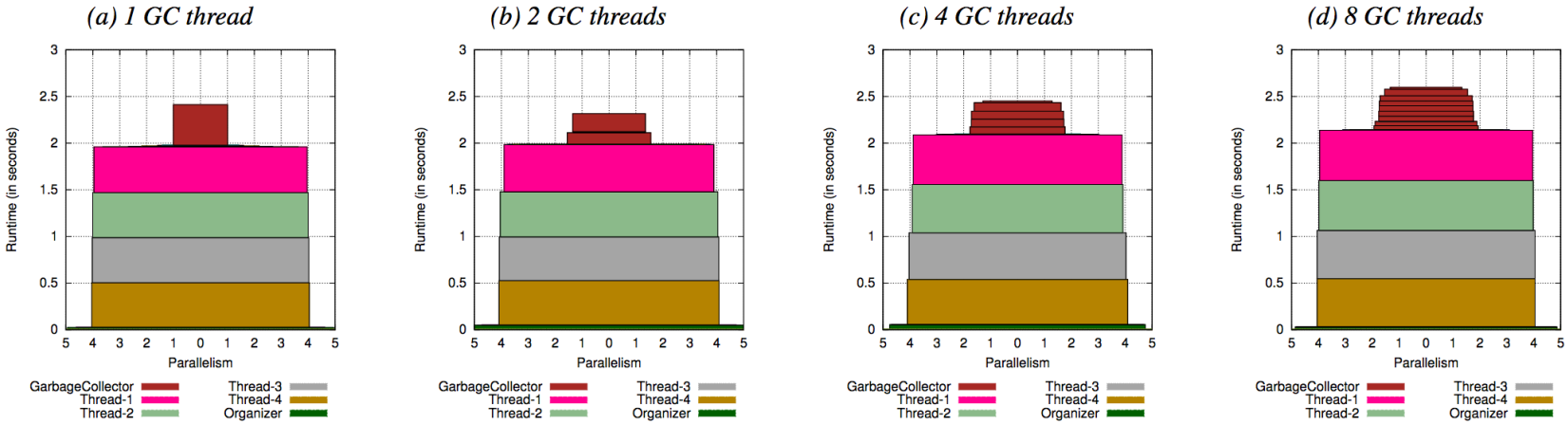

Figure 3. Xalan: scaling of GC threads with 4 application threads.