Project Title: saveCHIMP (save CHIp Multi Processor)
Team Name: DAT
Team Members: Arjun Khurana, Dong-hyeon Park, Timothy Wong

Problem to be addressed

As computer architecture becomes more complicated, the amount of time and effort spent on verifying a system will not scale with the growth of complexity. Current testing relies on random or directed tests that blindly explore the design space, which is inefficient. There is a need to find a way to limit the amount of testing to cases that are actually relevant to everyday operation.

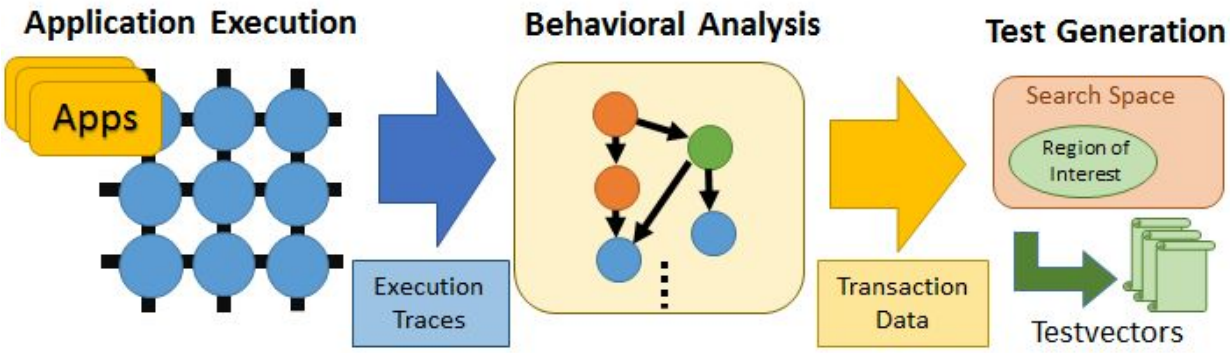
Why does this problem matter?

If testing is not limited to a relevant pool of scenarios, the time and resources spent on verifying complex systems in the future will grow without bound. There is always a tradeoff between correctness and resources spent on verification. If we could characterize the relevant behaviors that will be exercised by the system, and generate tests that specifically target those behaviors, we could significantly reduce the amount of tests required, while guaranteeing greater correctness. While our solution shares some similarities with previous works such as Inferno, it is different in that we focus on verifying the entire CMP system and how cores interact with each other, rather than focusing on a single design.

Idea/solution to be investigated by the project

We want a framework to do the following:

1. Given a program/software expected to run on the system, identify the characteristic transactions and behaviors that are exercised by the program.
2. Represent the transactions/behaviors in a model that can help us identify a search space.
3. Generate several series of targeted tests for the particular search space we are interested in. We will focus our effort in this project on applying our idea on a homogeneous system (e.g. a network of CPUs).



How do you plan to develop the project?

We will use gem5 or MARSS as our platform to simulate the behavior of a homogeneous system. We plan to focus on the x86 ISA, with the MESI_CMP_directory memory consistency protocol on a 8x8 mesh network. The MARSS platform is simpler, but may not support a NoC configuration. The applications we will be using as our target workloads are PARSEC and SPLASH benchmarks. We will log the simulation traces and architectural states and extract the simulation traces from gem5. We will then develop a behavioral analyzer that uses the traces to extract transactions from it. Finally, we will develop a test generator that uses the transactional data from the behavioral analysis to generate test vectors.

How do you plan to evaluate the project's results?

After we identify the characteristic behaviors of our system, we can design a series of test cases that specifically target those behaviors. In order to show that our test cases are effective in verifying the relevant behaviors, we will compare our sets of tests with randomly generated tests. We plan to inject a number of bugs into our baseline system to assess how well each set of tests can bring out these bugs. To quantify our results, we will plot a graph of the number of bugs caught vs. testing duration. We expect the line of the random test cases to be constant along the time axis, while our specific test cases exhibits some increasing trend. If that is the case, then we will have shown that we are on the right track for developing a general algorithm to limit the amount of testing to only relevant cases.

Timeline

10/9 - Project Outline
10/16 - Benchmarks ran on gem5 and trace/simulation data collected.
10/22 - Characterize the behaviors of benchmarks
10/23 - ========== Checkpoint 1 ==========
10/30 - Investigate different ways of characterizing applications and HW activity.
11/6 - Finalize behavioral characterization solution/idea. Start work on test generator.
11/12 - Finish design of test generation framework based on behavior analysis.
11/13 - ========== Checkpoint 2 ==========
11/20 - Testbench setup with bug-injection on test system. Start data collection.
12/3 - Evaluate preliminary data. Adjust characterizer and test generator as needed.
12/4 - ========== Checkpoint 3 ==========
12/9 - Optimize solution, collect additional data.
12/10 - ========== Checkpoint 4 ==========
12/11 - Stop technical work. Start writing paper and work on presentation.
12/16 - Final Presentation and Report due