Benchmarking code generation tool (**for d-cache misses)**

Documentation

Prepared by:

Apoorv Kumar

IIT Guwahati (09 batch)

Idea behind the tool :

We want to see the cache miss patterns with respect to memory access patterns. Interestingly , due to writeback/buffer options in caches , the cache miss due to writes is not the same as that due to reads. This tool tries to measure the cache miss patterns against following parameters.

1 . **BLOCKS**

This refers to the large chunk of memory that is accessed for a long duration. The memory accesses doesn't go beyond the specified range for very long. An example is running an application for some large duration , during which the access doesn't go beyond heap allocated to program.

2. **AMPLITUDE**

The range in which memory access remain inside a block is given by this parameter.

3. **AVG_MEM_JUMP (PER BLOCK)**

When the program counter jumps from one program to another , the jump is pretty large. This parameter takes in the average jump per shift in block.

4. **WRITE_PERCENT**

As said already this is bound to affect the cache misses in many architectures. We get the read percentage by calculating 100-WRITE_PERCENT

5. **DISTRIBUTION**

This gives the probability of accessing a memory location inside a block. Currently the only distribution supported is UNIFORM. We would like to include more real life distributions like pareto or normal/gaussian distributions (but it would need a bit of overhaul).

6. **ACCESSES**

This gives the number of accesses inside each block.

7. **CONTINUOUS_BLOCKS**

The idea behind this parameter is that some programs are very well behaved and access the memory in a very continuous way (example: merge sort) , thus in the block of that specific program , the entire block seems to be one continuous block. While some other programs (example: shell sort) accesses memory in a very jagged way , and thus it seems as if there were many unrelated continuous blocks within one block. Thus , this parameter controls the level of jaggedness within a block. Higher the number , greater the jaggedness.

These params can be changed in ***workload_memory.py***

Finally , the tool prints a graph that shows the memory access patterns , so we could better appreciate the influence of parameters. Play around with the parameters and see the difference.