Progress Report

Ravi Ramaseshan (rramase@ncsu.edu)
Muhammad Latif (mmlatif@ncsu.edu)

Solved Issues:

Ravi:

1. Merged the following files from Swift with the current release (openimpact-1.0rc4):
    a. ./Lcode/Lcode/l_lcode.c
    b. ./Lcode/Lcode/l_pred_flow.c
    c. ./Lcode/Lcode/r_dataflow.c
    d. ./Lcode/codegen/Ltahoe/phase2_func.c
    e. ./Lcode/ codegen/Ltahoe/phase2_memstk.c
    f. ./Lcode/codegen/Regalloc/r_regalloc.c
    g. ./Lcode/codegen/Regalloc/r_regspill.c
    h. ./Lcode/opti/Lopti/l_disjvreg.c
    i. ./Lcode/sched/SM/sm.c
2. Read a document on Lcode and dumped the Lcode representation of a sample switch case program with the objective of becoming familiar with the internal representation of OpenImpact.

Muhammad:

3. Conducted further timing analysis and space analysis on two other possible methods to reduce instruction duplication for soft error detection in C.
    a. For pointer scheme
        i. Wrote a small routine which traversed an array of structure using pointer traversal
        ii. Repeated the routine a N (where N is a large number) to amortize any extra overheads.
        iii. Implemented hardware abstraction for the modulo 3 operation by repeating the modulo 3 operation N times, and subtracting the cost of modulo 3 operation from the actual cost of the routine
    b. MSB/LSB duplication
        i. Wrote a small routine which involved addition of integers whose result was always bounded by the half of the bits in an integer
        ii. Incorporated the MSB half duplication of the LSB half of the integer values
        iii. Calculated the times taken by normal operation, EDDI operation, our scheme

    From the results of the experiments, we can conclude that implementing these schemes in the compiler would produce substantial savings in terms of both execution time and duplicated instruction overhead.

Open Issues:

1. Get complete SWIFT code.
2. Get the Fault tolerant SWIFT version of OpenImpact to build.

Next Step:

1. Understand the OpenImpact code base and the SWIFT fault tolerance modules.

2. Understand the (duplicate) instruction generation for switch cases, for loops and pointer traversal.

3. Search for more constructs to reduce instruction duplication for soft error detection.