User Manual to Generate Trace for Program Reconstruction

Traces include memory access trace and operator trace. They are obtained from the same instrumented code. It is highly recommended to refer to our example instrumented code to see how trace is produced, where instrument code is added etc.

Step0: add the following lines into the program.

#include “gentrace.h”

extern FILE\* fid;

also, add the following line into the beginning of the main() function:

fid = fopen(“memory.txt”, “w+t”);

Note0: currently the instrumentation should only appear in ONE file.

Step1: Instrument the original code so that the executable could dump memory access traces during execution. The memory trace is stored in memory.txt file. The way we instrument the code is to print out the address (or the content) of the accessed data in the memory using the following APIs.

double\* TnsMem(double\* pData);

double& TnsMem(double& rData);

int& TnsMem(int& rData);

double& TnsMemWr(double& rData);

double TnsMemC(double rConstant);

int TnsMemC(int rConstant);

int TnsMemIter(int i);

The first four APIs print out the address of the accessed data. The last three APIs just print out the value of the accessed data. The difference between TnsMemC and TnsMemIter is that TnsMemC actually prints out string “Constant+[value of the Constant]” while TnsMemIter prints out just the value of the constant.

Note 1: we only instrument assignment statements (since changes to internal data happen here!). Conditional statements as well as I/O statements are not yet allowed to be instrumented. Otherwise the reconstruction process will be confused.

e.g. the following statement should NOT be instrumented:

if ( (i\*N+j ) % 20 == 0) {

printf(“%0.2lf \n“, A[i\*N+j]);

}

Note 2: for floating operations, we focus on double precision data.

TnsMemWr function is added to the left hand side of each assignment statement (Write Operation).

TnsMem function is usually added to the right hand side of this assignment statement (Read Operation)

For example,

A[i] = B[i] + C[i];

in instrumented as follows:

TnsMemWr(A[i]) = TnsMem(B[i]) + TnsMem(C[i]);

Note 3: the offset/index of memory access should NOT be instrumented. For example:

A[i] = B[i] + C[i] should NOT be instrumented as

TnsMemWr(A[TnsMem(i)]) = TnsMem(B[TnsMem(i)] + TnsMem(C[TnsMem(i)];

This is because the Reconstruction tool is supposed to recover the “i” iterator.

Note 4: accesses to integer/floating point constants should always use TnsMemC() function. E.g.:

A[i] = 0.0;

B[i] = 0;

should be instrumented as:

TnsMemWr(A[i]) = TnsMemC(0.0);

TnsMemWr(B[i]) = TnsMemC(0);

Note 5: accesses to loop iterators on the right hand side should always use TnsMemIter() function.

e.g. the following original loop structure:

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

A[i] = i;

should be instrumented as

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

TnsMemWr(A[i]) = TnsMemIter(i);

Step2: Generate execution trace from the above instrumented code.

We have scripts available to perform this task. Simply run the script and feed the script with the instrumented code. The key execution information is recorded.

e.g.

./gen-ExecutionTrace.sh instrumentedFile.c

then ExecutionTrace file is generated and has the following structure:

68@0=0;

69@0=0;

72@0=0;

73@0=(0+0)/0/0;

74@0=(0+0)/0/0;

75@0=(0+0)/0/0;

76@0=(0+0)/0/0;

77@0=(0+0)/0/0;

78@0=0;

79@0=0;

82@0=(((DATA\_TYPE)0)\*0)/0;

125@0=0+0\*0+0\*0;

131@0=0+0\*0\*0;

136@0=0+0;

142@=0+0\*0\*0;

The numbers before “@” symbol are which line each statement is at.

The expressions after “@” symbol correspond to the statement but the memory accesses are anonymized to “0”.

Note 6: each assignment statement should only occupy one line.

e.g. the following statement

TnsMemWr(A[i]) = TnsMem(A[i-1]) +

TnsMem(A[i+1]) +

TnsMem(A[i]);

should be changed to occupy only one line:

TnsMemWr(A[i]) = TnsMem(A[i-1]) + TnsMem(A[i+1]) + TnsMem(A[i]);