**Georgia Institute of Technology**

**School of Computer Science**

**ACT (Alternative Computing Technology) Lab**

**R2 User Study**

**Goal:**

The goal of this user study is to evaluate how R2 language can help programmers annotate approximations on source codes compared to EnerJ language.

The process has three steps:

1. A subject should understand each benchmark and what is the target of approximation.
2. At first, she will try to annotate the source codes with EnerJ annotations. The time to take for annotating will be measured and it is used for comparison data.
3. For the same benchmark, she will use R2 language extensions to approximate the same part of programs. The time to take for annotating will also be measured.

**Annotation:**

By default, every program is completely accurate.

1. EnerJ

**@Approx**: type qualifier attaching to variables that users want to make approximate

e.g. **@Approx** int VARIABLE\_NAME;

**Endorsements.endorse**(VARIABLE\_NAME);

Users can annotate primitive type variables and arrays using **@Approx** but it is not allowed to use **@Approx** for class instances or class definitions. The users should use **Endorsements.endorse()** for a primitive type variable or an array element.

1. R2

**Relax.relax(**VARIABLE\_NAME**)**: function call to relax accuracy requirements for VARIABLE\_NAME and all data/operations driving into the VARIABLE\_NAME

**Restrict.restrict(**VARIABLE\_NAME**)**: function call to restrict approximations for VARIABLE\_NAME and all data/operations driving into the VARIABLE\_NAME

Users should use **Relax.relax()** only for primitive type variables.

**Requirements:**

Users should avoid to approximate:   
 (1) loop conditional  
 (2) array indices  
 (3) if-conditional that is control dependent to program flow   
  
\*\*\* if-conditional that is only control dependent to data operations can be approximated since the control dependency can be converted into data dependency

**Benchmark:**

There are four benchmarks that are used in the user study. The first one is for demonstration so that we can explain how EnerJ and R2 work to the subjects.

1. mc (MonteCarlo) – LOC: 165

**Description:** Monte Carlo integration approximates the value of Pi by computing the integral of the quarter circle y = sqrt(1 - x^2) on [0,1]. It chooses random points with the unit square and computes the ratio of those within the circle.

**Program Output:** *out* (estimated PI value)

**Expected # of annotations:**

1. EnerJ: 7 @Approx, 2 Endorsements.endorse() (Random.java: 16)
2. R2: 1 Relax.relax()
3. sor (Jacobi Successive Over-relaxation) – LOC: 188

**Description:** SOR on a 100x100 grid exercises typical access patterns in finite difference applications, for example, solving Laplace's equation in 2D with Drichlet boundary conditions.

**Program Output**: *g\_i\_j* (all elements in output matrix G[][])

**Expected # of annotations:**

1. EnerJ: 12 @Approx, 1 Endorsements.endorse()
2. R2: 1 Relax.relax()
3. smm (Sparse matrix multiply) – LOC: 201

**Descripion:** Sparse matrix multiply uses an unstructured sparse matrix stored in compressed-row format with a prescribed sparsity structure.

**Program Output:** *y\_i* (all elements in output vector y[])

**Expected # of annotations:**

1. EnerJ: 8 @Approx, 1 Endorsements.endorse()
2. R2: 1 Relax.relax()
3. fft (Fast Fourier Transform) – LOC: 289

**Description:** FFT performs a one-dimensional forward transform of 4K complex numbers. This kernel exercises complex arithmetic, shuffling, non-constant memory references and trigonometric functions. The first section performs the bit-reversal portion (no flops) and the second performs the actual Nlog(N) computational steps.

**Program Output:** *x\_i* (all elements in output vector x[])

**Expected # of annotations:**

1. EnerJ: 29 @Approx, 3 Endorsements.endorse()
2. R2: 1 Relax.relax()