Compiler Techniques for Protection of Critical Instructions on Faulty Architectures

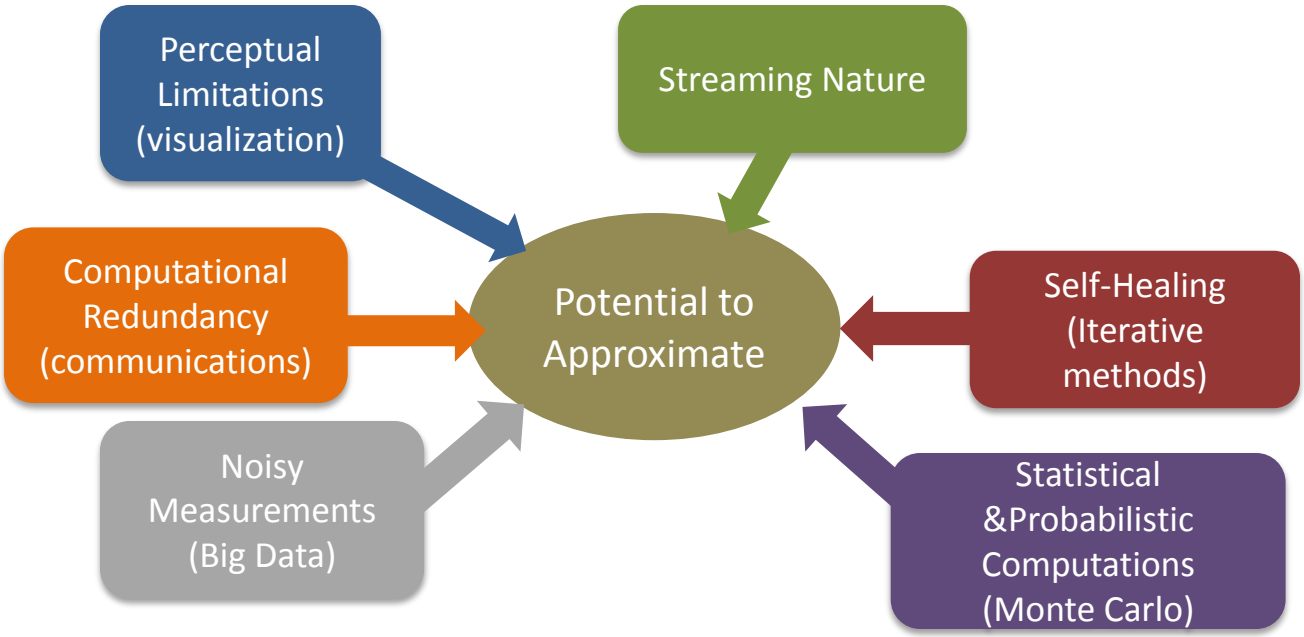
Konstantinos Parasyris, Vassilis Vassiliadis, Christos D. Antonopoulos, Spyridon Lalis, Nikolaos Bellas

Center Of Research and Technology, Hellas
& University of Thessaly

Motivation

CERTH / IRETETH

Inherent application error resieliency.




```
for ( i = 0 ; i < N; i++){
        x = *(A+i);
        y = *(B+i]);
        temp = x+ y
        *(c+i)=temp;
}
```

Not all instructions are equally critical.

Objective

- Protect Only critical instructions from faults
  - Decreases costs of correcting and detecting all errors
    - Less Protection-->Decreased Cost
- Non-Critical instructions vulnerable to errors
  - Rely on application resiliency for the remaining fault locations
    - Potential quality degradation.

Contribution

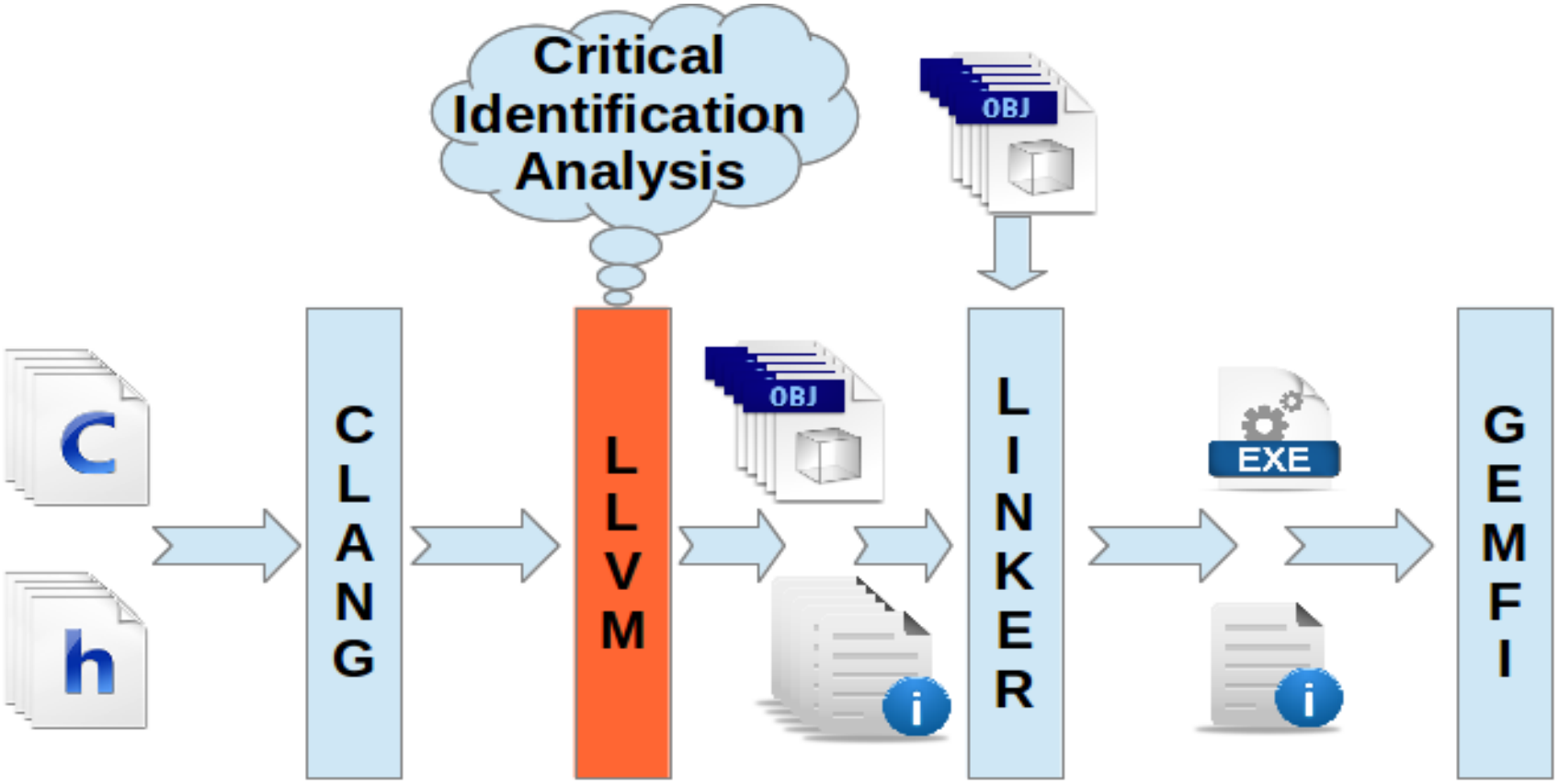
Compiler analysis to identify critical and non - critical instructions:

- LLVM back-end pass.

Experimental validation/evaluation of the pass.

Identified influence of code optimizations to the percentage of critical instructions.

Implementation Flow Chart



Critical Analysis Example 1

CERTH / IRETETH

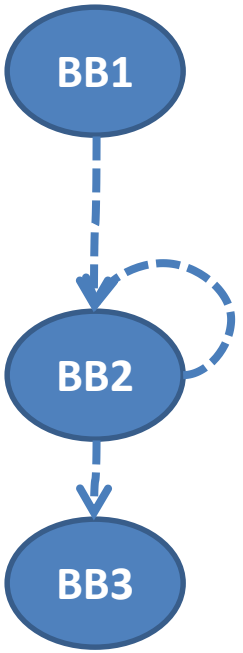
```
add $s1 $0 $0
for:
    beq $s0, $s1, end
    lw $t2, ($s2)
    lw $t3, ($s3)
    add $t4, $t3, $t2
    sw $t4, ($s4)
    addi $s2, $s2, 4
    addi $s3, $s3, 4
    addi $s4, $s4, 4
    addi $s1, $s1, 1
    j for
end:
```

BB1 GEN={ $s4, $s3, $s2, $s1, $s0 }

BB2 GEN={ $s4, $s3, $s2, $s1, $s0 }

BB3 GEN={ }

```
for ( x = 0 ; x < N ; x++)
        C[x]= A[x] + B[x];
```



Critical Analysis Example 2

CERTH / IRETETH

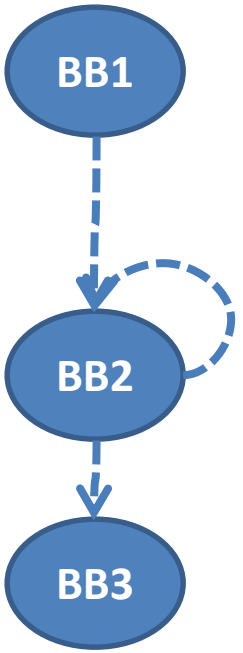
```
add $s1 $0 $0
for:
    beq $s0, $s1, end
    lw $t2, ($s2)
    lw $t3, ($s3)
    add $t4, $t3, $t2
    sw $t4, ($s4)
    addi $s2, $s2, 4
    addi $s3, $s3, 4
    addi $s4, $s4, 4
    addi $s1, $s1, 1
    j  for
end:
```

BB1 GEN={
$s4
$s3
$s2
$s0
}

BB2 GEN={
$s4
$s3
$s2
$s1
$s0
}

BB3 GEN={
}

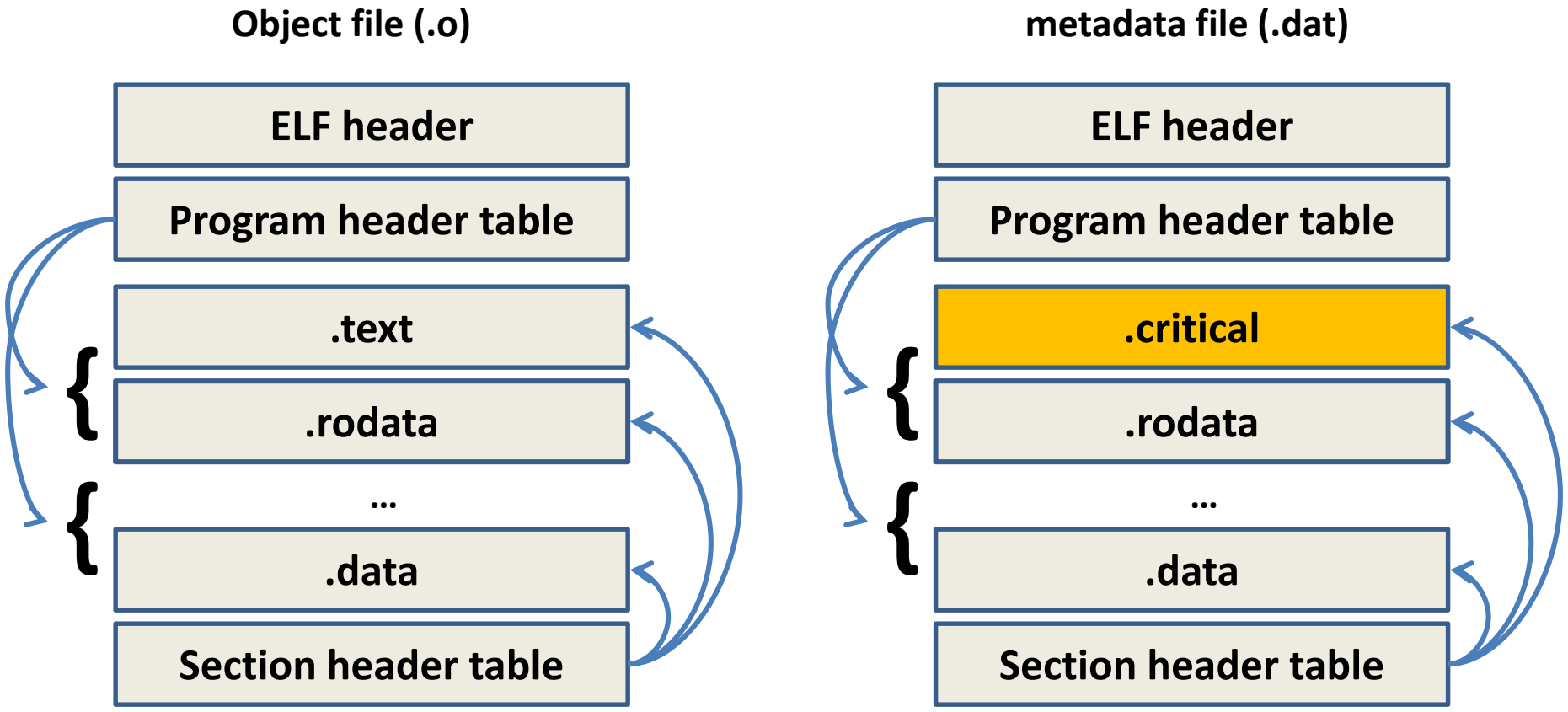
```
for ( x = 0 ; x < N ; x++)
        C[x]= A[x] + B[x];
```



Object File Creation

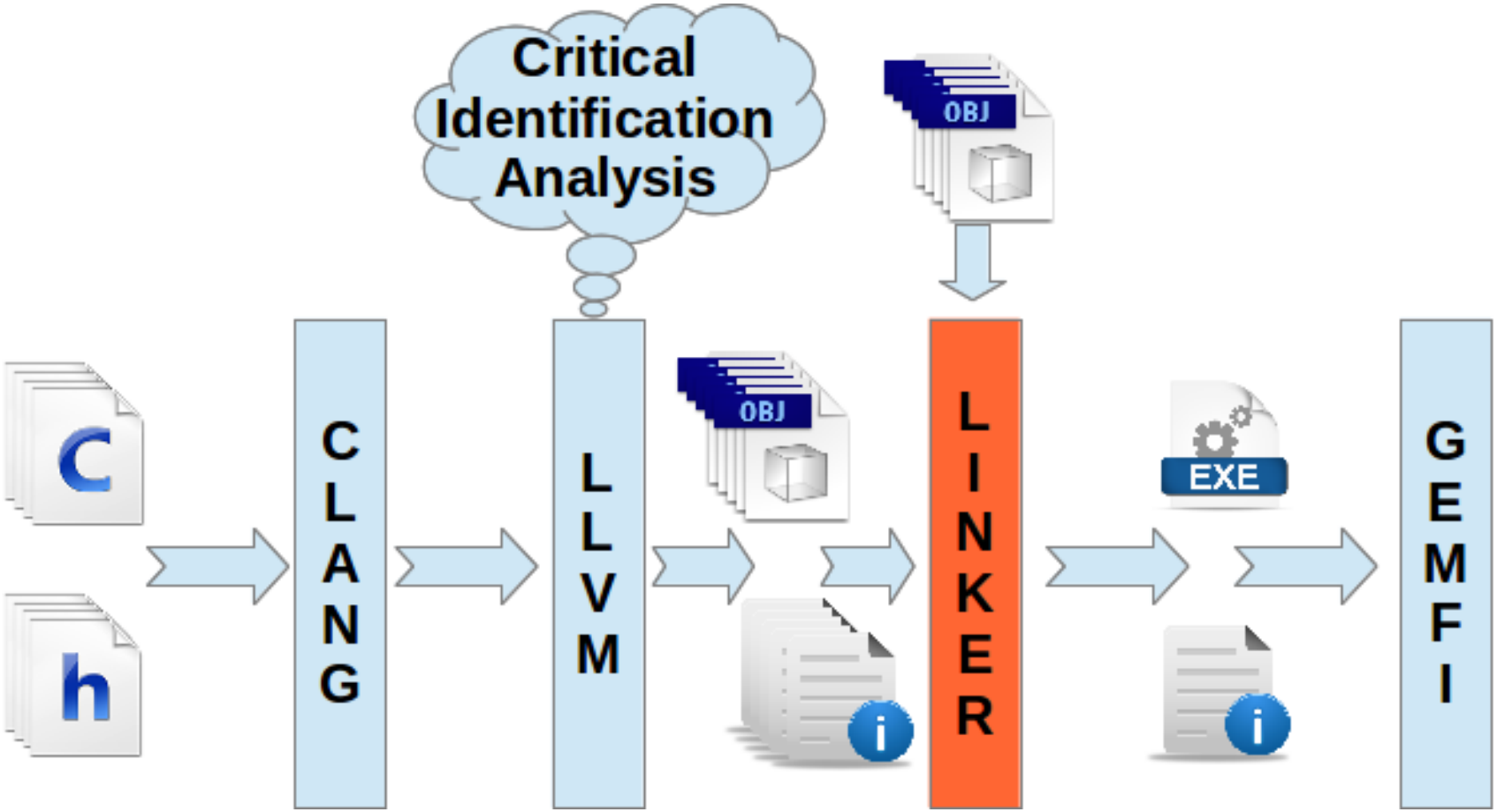
CERTH / IRETETH

Each Object file is accompanied by a metadata file that contains information about the criticality of instructions.



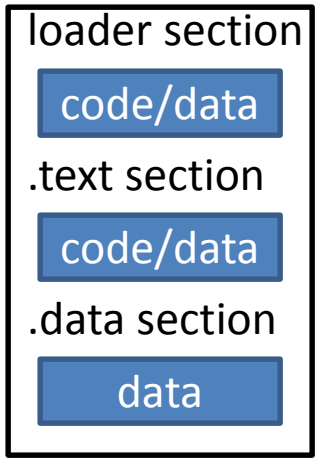
Implementation Flow Chart

CERTH / IRETETH

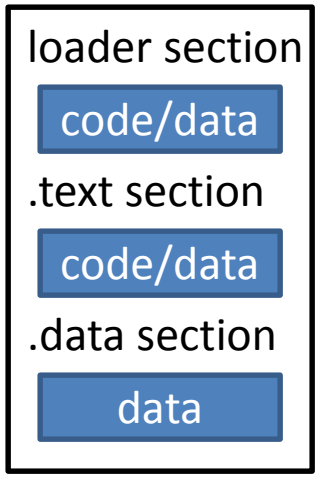


Linker Extensions

file1.o



file2.o

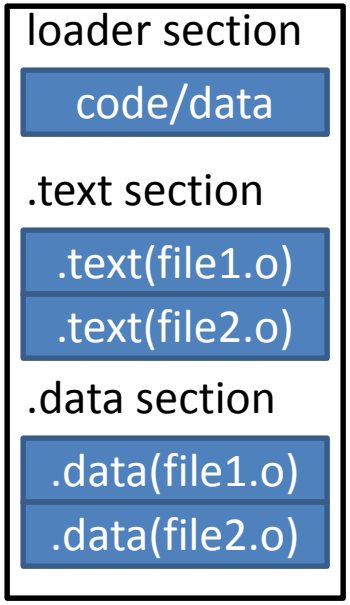


Original Liker:

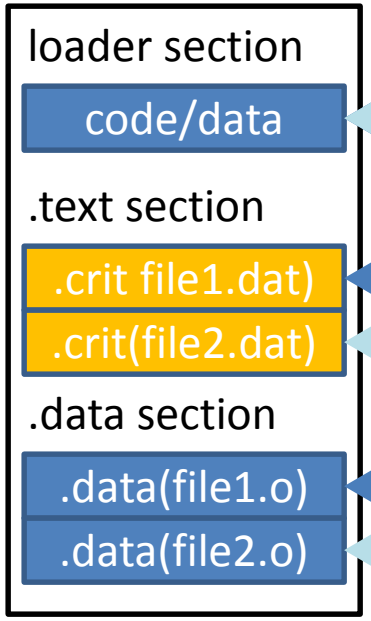
Input: object files, libraries

Output: single Executable

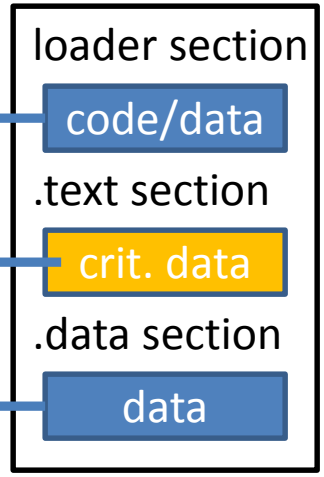
a.out



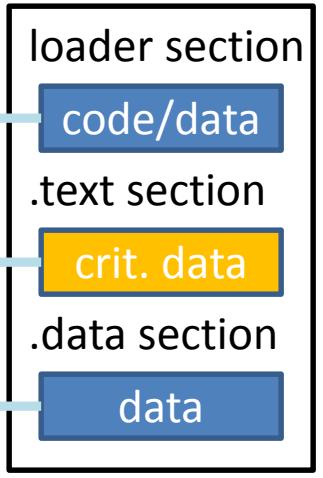
a.dat



file1.dat



file2.dat



Extended Linker:

Input: objectfiles, libraries,

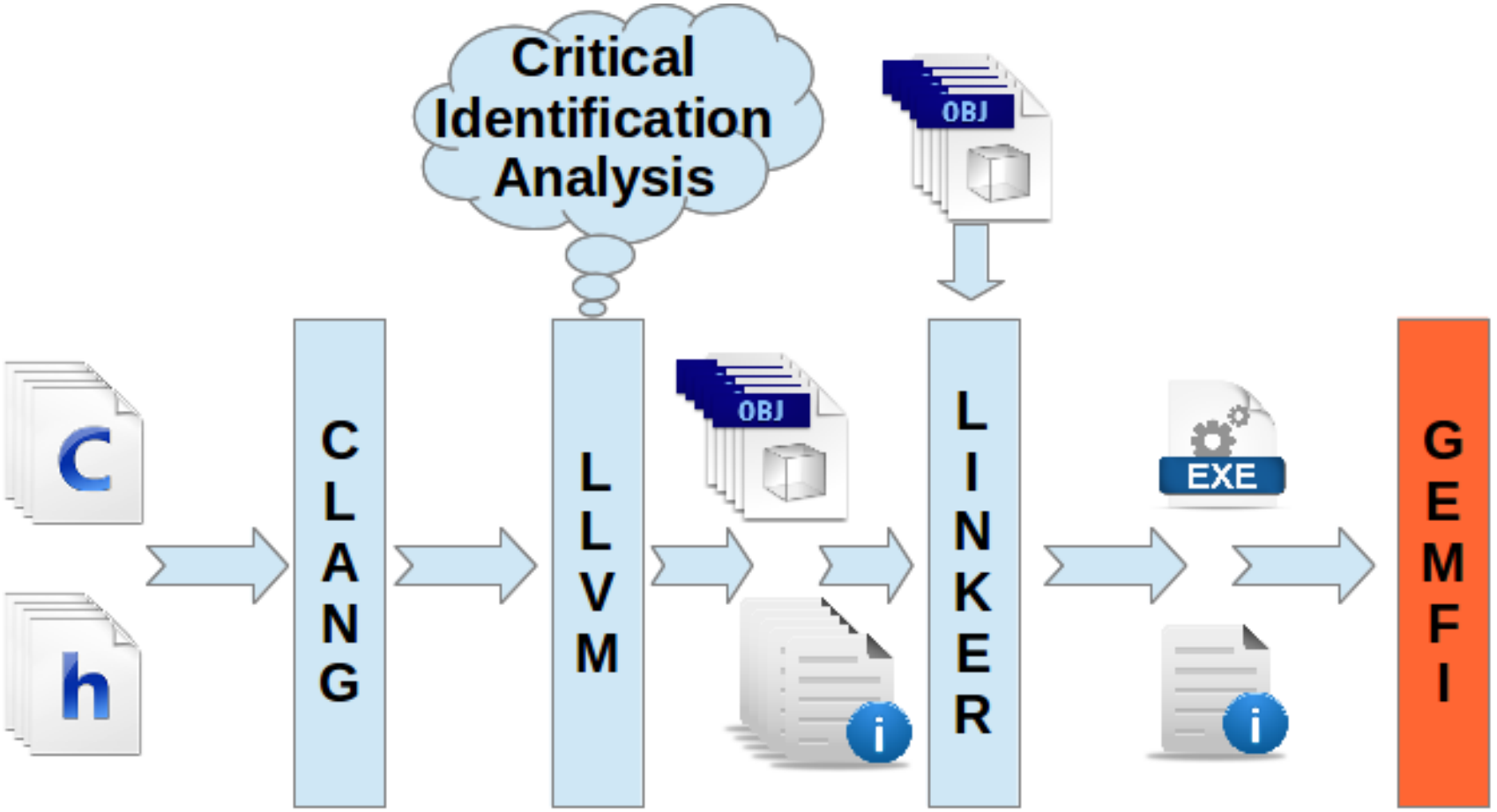
metadata files

Output: single Executable

single metadata file

Implementation Flow Chart

CERTH / IRETETH



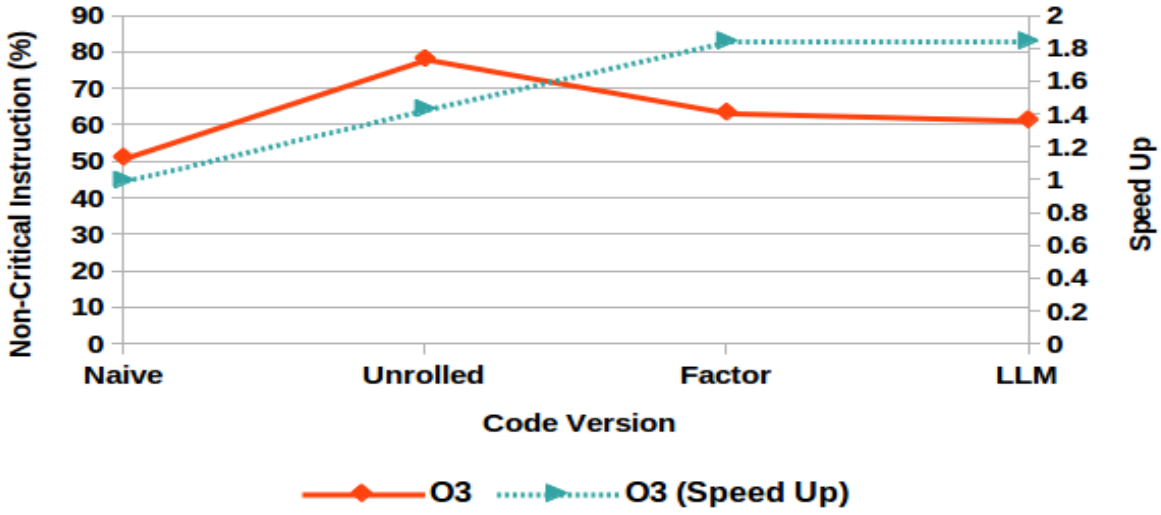
Experimental Evaluation methodology

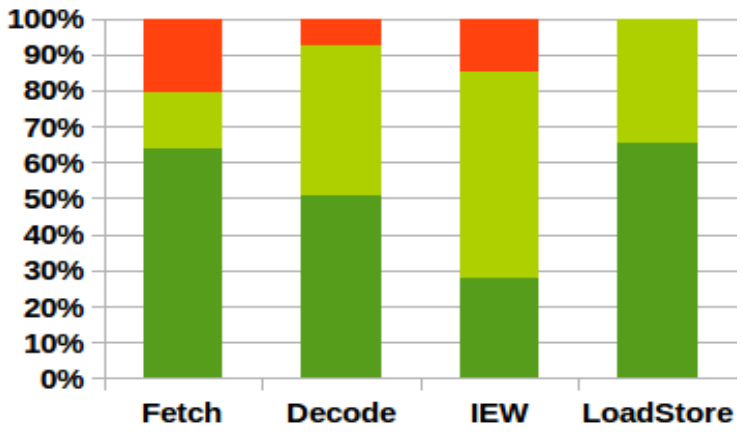
CERTH / IRETETH

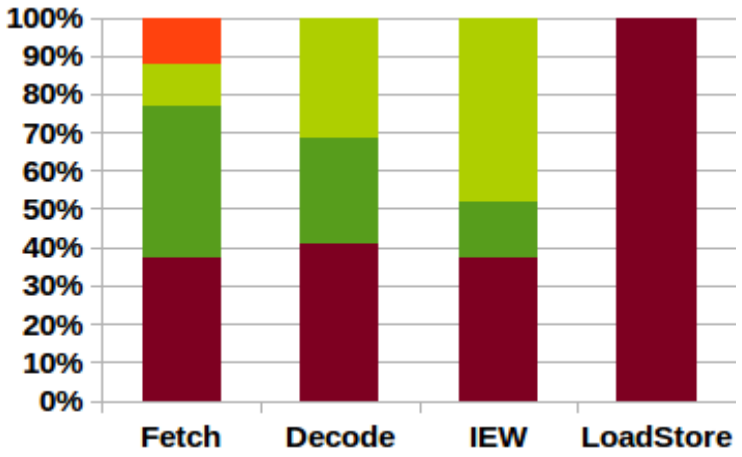
- Benchmarks
  - DCT
  - Sobel
  - Blackscholes
- Optimize applications to increase percentage of non critical instructions
- Fault Injection
  - 2 versions of each Application.
    - With Critical Instructions and only non critical

DCT

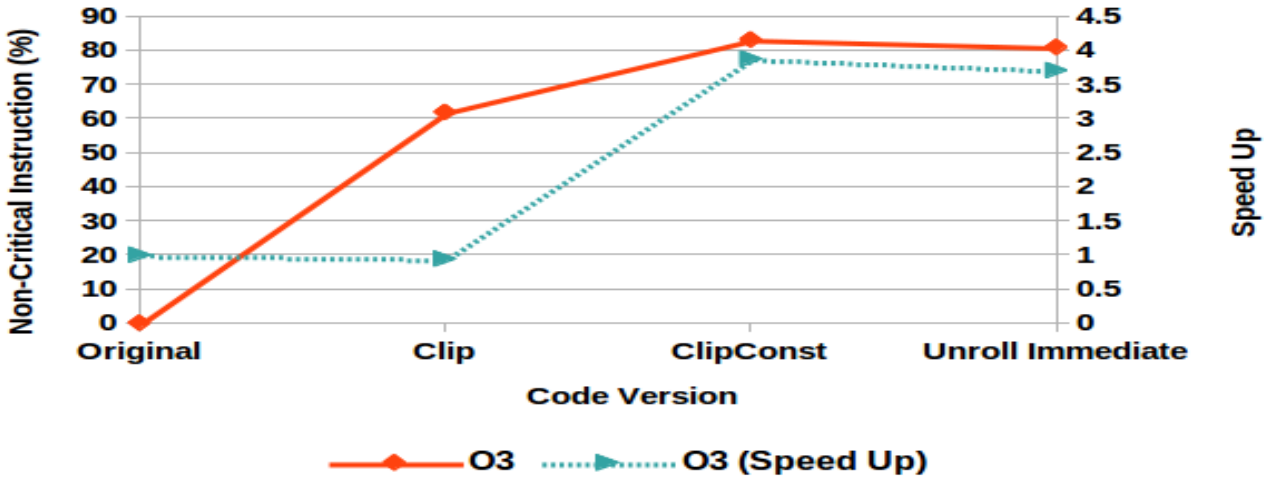
CERTH / IRETETH







Sobel




```
Sblx = SobelX(x,y)
Sbly = SobelY(x,y)
res = sqrt(Sblx² + Sbly²)
if ( res > 255)
   res = 255
out[y][x]=res
```

```
Sblx = SobelX(x,y)
Sbly = SobelY(x,y)
res = sqrt(Sblx² +
   Sbly²)
out[y][x]=CLIP(res)
```

Conclusions

- Increased Error Resiliency through ISA extensions.
  - Fetch/ Decode Stages not fully protected by failures even with Instruction Set extensions.
- Increasing number of non-critical Instructions through code Optimizations/transformations
- CISC like instruction sets hide critical information that could be exploited by the compiler.

Questions ?

This work has been partially supported by the EC within the 7th Framework Program under the FET-Open grant agreement SCoRPiO, grant number 323872.

SCoRPiO

Significance-Based Computing for Reliability and Power Optimization

SEVENTH FRAMEWORK PROGRAMME