Write-and-Do-Mini-Assignment#5: SCEV mini assignment version 2

Sai Harsha Kottapalli
CS17BTECH11036

SCEV Analysis
For analysis, i used “grep” for r.e. {.*,.*,.*} to get scev expressions. With this i tried to relate to the LLVM IR and get the following inference.
Only the equivalent code before and after optimizations are compared.
Used flag : **-mem2reg** for promoting memory to register which helps in easy comparison between .ll files and hence the following inference has been made.
When generating .ll file from .cpp file if we don’t add **-disable-O0-optnone** flag, it is very hard to correlate between the resultant .ll files.

hamiltonian-cycle-backtracking.cpp

{0,+,1}<nuw><nsw><%19> U: [0,-2147483648) S: [0,-2147483648) Exits: <<Unknown>> LoopDispositions: { %19: Computable }

{%2,+,4}<nsw><%19> U: full-set S: full-set Exits: <<Unknown>> LoopDispositions: { %19: Computable }

Here, SCEV is giving is the range and exit values of variables in LLVM IR, based on which we can decide which type of data type is sufficient.

Without SCEV,
%7 = sext i32 %.01 to i64
%8 = getelementptr inbounds i32, i32 %3, i64 %7*
store i32 -1, i32 %8, align 4*

With SCEV,
%7 = shl nuw nsw i64 %lsr.iv, 2
%scevgep = getelementptr i8, i8 %2, i64 %7*

%scevgep1 = bitcast i8 %scevgep to i32**
store i32 -1, i32 %scevgep1, align 4*

Associated SCEV,
{0,+,1}<nuw><nsw><%4> U: [0,6) S: [0,6) Exits: 5
LoopDispositions: { %4: Computable }

Here, we can notice that instead of using a 32 bit integer we use 8 bit integer which helps execute operations quicker and use less memory.

matrix-chain-multiplication.cpp

*{((4 * (zext i32 %1 to i64))<nuw><nsw> + %7)<nsw>,+,(4 * (zext i32 %1 to i64))<nuw><nsw>}<%8> U: [0,-3) S: [-9223372036854775808,9223372036854775805)*
*Exits: (((zext i32 %1 to i64) * (4 + (4 * (zext i32 (-1 + (1 smax %1))<nsw> to i64))<nuw><nsw>)<nuw><nsw>) + %7) LoopDispositions: { %8: Computable }*

If we notice here, exit consists of two SCEV expressions. This must be because there exists a pattern in which the nested loops proceeds, we can try loop-reduce to check if optimization use this analysis. We do notice the use of **uglygeps** and **bitcasts.**

Without SCEV,
%38 = sext i32 %.1 to i64
%39 = mul nsw i64 %38, %4
%40 = getelementptr inbounds i32, i32 %7, i64 %39*
%41 = sext i32 %.01 to i64
%42 = getelementptr inbounds i32, i32 %40, i64 %41*
%43 = load i32, i32 %42, align 4*
%44 = add nsw i32 %.01, 1
%45 = sext i32 %44 to i64
%46 = mul nsw i64 %45, %4
%47 = getelementptr inbounds i32, i32 %7, i64 %46*
%48 = sext i32 %28 to i64
%49 = getelementptr inbounds i32, i32 %47, i64 %48*
%50 = load i32, i32 %49, align 4*
%51 = add nsw i32 %43, %50
%52 = sub nsw i32 %.1, 1
%53 = sext i32 %52 to i64

```
%54 = getelementptr inbounds i32, i32* %0, i64 %53
%55 = load i32, i32* %54, align 4
%56 = sext i32 %.01 to i64
%57 = getelementptr inbounds i32, i32* %0, i64 %56
%58 = load i32, i32* %57, align 4
%59 = mul nsw i32 %55, %58
%60 = sext i32 %28 to i64
%61 = getelementptr inbounds i32, i32* %0, i64 %60
%62 = load i32, i32* %61, align 4
%63 = mul nsw i32 %59, %62
%64 = add nsw i32 %51, %63
%65 = sext i32 %.1 to i64
%66 = mul nsw i64 %65, %4
%67 = getelementptr inbounds i32, i32* %7, i64 %66
%68 = sext i32 %28 to i64
%69 = getelementptr inbounds i32, i32* %67, i64 %68
%70 = load i32, i32* %69, align 4
%71 = icmp slt i32 %64, %70
```

With SCEV,

```
%uglygep14 = getelementptr i8, i8* %lsr.iv1013, i64 %lsr.iv2
%uglygep1415 = bitcast i8* %uglygep14 to i32*
%41 = load i32, i32* %uglygep1415, align 4
%42 = sext i32 %lsr.iv7 to i64
%43 = mul nsw i64 %42, %4
%44 = getelementptr inbounds i32, i32* %7, i64 %43
%45 = sext i32 %30 to i64
%46 = getelementptr inbounds i32, i32* %44, i64 %45
%47 = load i32, i32* %46, align 4
%48 = add nsw i32 %41, %47
%49 = sub nsw i32 %.1, 1
%50 = sext i32 %49 to i64
%51 = getelementptr inbounds i32, i32* %0, i64 %50
%52 = load i32, i32* %51, align 4
%uglygep = getelementptr i8, i8* %lsr.iv3, i64 %lsr.iv2
%uglygep4 = bitcast i8* %uglygep to i32*
%53 = load i32, i32* %uglygep4, align 4
%54 = mul nsw i32 %52, %53
%55 = sext i32 %30 to i64
```

%56 = getelementptr inbounds i32, i32 %0, i64 %55*
%57 = load i32, i32 %56, align 4*
%58 = mul nsw i32 %54, %57
%59 = add nsw i32 %48, %58
%60 = sext i32 %.1 to i64
%61 = mul nsw i64 %60, %4
%62 = getelementptr inbounds i32, i32 %7, i64 %61*
%63 = sext i32 %30 to i64
%64 = getelementptr inbounds i32, i32 %62, i64 %63*
%65 = load i32, i32 %64, align 4*
%66 = icmp slt i32 %59, %65

Associated SCEV,
*{((4 * (zext i32 %1 to i64))<nuw><nsw> + %7)<nsw>,+,(4 * (zext i32 %1 to i64))<nuw><nsw>}<%22> U: [0,-3) S: [-9223372036854775808,9223372036854775805) Exits: {((4 * (zext i32 %1 to i64))<nuw><nsw> + %7)<nsw>,+,(4 * (zext i32 %1 to i64))<nuw><nsw>}<%22> LoopDispositions: { %34: Invariant, %22: Computable, %19: Variant }*

The number of cycles required has been reduced with the help of uglygeps and bitcast from SCEV analysis.

m-coloring-problem.cpp

Without SCEV,
%8 = sext i32 %0 to i64
%9 = getelementptr inbounds [4 x i8], [4 x i8] %1, i64 %8*
%10 = sext i32 %.01 to i64
%11 = getelementptr inbounds [4 x i8], [4 x i8] %9, i64 0, i64 %10*
%12 = load i8, i8 %11, align 1*
%13 = trunc i8 %12 to i1

With SCEV,
%scevgep2 = getelementptr i8, i8 %scevgep1, i64 %lsr.iv*
%9 = load i8, i8 %scevgep2, align 1*
%10 = trunc i8 %9 to i1

Associated SCEV,

(sext i32 %0 to i64) U: [-2147483648,2147483648) S: [-2147483648,2147483648) Exits: (sext i32 %0 to i64) LoopDispositions: { %5: Invariant }
*((4 * (sext i32 %0 to i64))<nsw> + %1)<nsw> U: full-set S: full-set Exits: ((4 * (sext i32 %0 to i64))<nsw> + %1)<nsw> LoopDispositions: { %5: Invariant }*

Here too we notice that the total number of instructions as well as cycles are reduced with the help of scev analysis.

Subset-sum.cpp

Without SCEV,

```
%45 = sext i32 %.0 to i64
%46 = getelementptr inbounds i32, i32* %0, i64 %45
%47 = load i32, i32* %46, align 4
%48 = sext i32 %3 to i64
%49 = getelementptr inbounds i32, i32* %1, i64 %48
store i32 %47, i32* %49, align 4
%50 = sext i32 %.0 to i64
%51 = getelementptr inbounds i32, i32* %0, i64 %50
%52 = load i32, i32* %51, align 4
%53 = add nsw i32 %4, %52
%54 = icmp sle i32 %53, %6
```

With SCEV,

```
%indvars.iv = phi i64 [ %indvars.iv.next, %63 ], [ %42, %41 ]
%45 = icmp slt i64 %indvars.iv, %43
```

Associated SCEV,
{%5,+,1}<nsw><%42> U: full-set S: full-set Exits: (%2 smax %5) LoopDispositions: { %42: Computable }

This SCEV reduces the instructions required.

topological-sorting.cpp

Without SCEV,

```
%.01 = phi i32 [ 0, %7 ], [ %16, %15 ]
%9 = getelementptr inbounds %class.Graph, %class.Graph* %0, i32 0, i32 0
%10 = load i32, i32* %9, align 8
%11 = icmp slt i32 %.01, %10
%13 = sext i32 %.01 to i64
%14 = getelementptr inbounds i8, i8* %6, i64 %13
store i8 0, i8* %14, align 1
```

With SCEV,

```
%scevgep7 = getelementptr i8, i8* %6, i64 %lsr.iv5
store i8 0, i8* %scevgep7, align 1
```

Associated SCEV,

{(8 + %13)<nsw>,+,24}<nuw><%20> U: full-set S: full-set Exits: (8 + (24 * ((-8 + (8 * (sext i32 %1 to i64)))<nsw> /u 8)) + %13) LoopDispositions: { %20: Computable }

{(32 + %13),+,24}<nw><%20> U: full-set S: full-set Exits: (32 + (24 * ((-8 + (8 * (sext i32 %1 to i64)))<nsw> /u 8)) + %13) LoopDispositions: { %20: Computable }

The SCEV's calculated for the two loops are used in the loop-reduce optimization

transitive-closure-of-a-graph.cpp

Without SCEV,

```
%9 = sext i32 %.01 to i64
%10 = getelementptr inbounds [4 x i32], [4 x i32]* %0, i64 %9
%11 = sext i32 %.02 to i64
%12 = getelementptr inbounds [4 x i32], [4 x i32]* %10, i64 0, i64 %11
%13 = load i32, i32* %12, align 4
%14 = sext i32 %.01 to i64
%15 = getelementptr inbounds [4 x [4 x i32]], [4 x [4 x i32]]* %2, i64 0, i64 %14
%16 = sext i32 %.02 to i64
%17 = getelementptr inbounds [4 x i32], [4 x i32]* %15, i64 0, i64 %16
store i32 %13, i32* %17, align 4
```

With SCEV,

%scevgep14 = getelementptr [4 x i32], [4 x i32] %lsr.iv12, i64 0, i64 %lsr.iv9*
%9 = load i32, i32 %scevgep14, align 4*
%scevgep11 = getelementptr [4 x [4 x i32]], [4 x [4 x i32]] %lsr.iv7, i64 0, i64 0, i64 %lsr.iv9*
store i32 %9, i32 %scevgep11, align 4*

Associated SCEV,

%12 = getelementptr inbounds [4 x i32], [4 x i32] %10, i64 0, i64 %11*
--> {{%0,+,16}<nsw><%3>,+,4}<nsw><%6> U: full-set S: full-set Exits: {(16 + %0),+,16}<nw><%3> LoopDispositions: { %6: Computable, %3: Variant }

Number of instructions as well as cycles are reduced.
If we notice the later sext instructions are removed too because they don't effect the variable.