

G

Loop Unrolling

This appendix describes how to use an optimization called loop unrolling, and provides background information on how the compiler implements this feature. It contains the following sections:

§G.1: *Overview*

§G.2: *Selecting Loops for Unrolling: Pragma Loop_factor*

§G.3: *How the Compiler Unrolls Loops*

G.1 Overview

Loop unrolling (or *loop unwinding*) is a compiler optimization based on the following idea: If the next iteration of a loop does not rely on results of the prior iteration, two or more iterations can execute at the same time. This works only for loops with *constant stride*, that is, those whose loop counter is incremented by a compile-time constant for every iteration.

Performance considerations Loop unrolling is most effective in code running on machines with a pipelined architecture. You might realize only a small saving in execution time on machine architectures that do not have pipelined instructions. On machines with an instruction cache, loop unrolling can actually degrade performance if the unrolled loop exceeds the size of the cache.

G.2 Selecting Loops for Unrolling: Pragma Loop_factor

Pragma Loop_factor Use pragma Loop_factor to specify criteria for selecting loops to be potentially unrolled. This is the syntax:

```
#pragma Loop_factor(loop_count, size)
```

where the arguments have the following meanings:

loop_count Maximum number of times a loop will be unrolled
size Upper limit on the size of loops to be considered for unrolling

size refers to the number of intermediate-language instructions generated by the compiler; this number corresponds loosely to the number of individual machine instructions.

You can specify **#pragma Loop_factor** any number of times, with different values for *size*. You can have any number of loop factors in effect at the same time, each with a different *size*. However, a pragma with the same *size* as a previous pragma replaces that previous pragma. The default for *loop_count* depends on the number of instructions in the loop:

Number of Instructions	5	4	3	2
Times Unrolled	3	4	8	15

To disable a previously specified pragma, use **#pragma Loop_factor (0, -size)**, where *size* is the same size specified in the pragma you want to disable.

Specify multiple Loop_factor pragmas If you specify **#pragma Loop_factor(n, 0)**, *n* becomes the default number of times a loop is unrolled regardless of size. If you also specify (for instance) **#pragma Loop_factor(x, 100)**, loops with fewer than 100 intermediate instructions will be unrolled *x* number of times, and loops with more than 100 intermediate instructions will be unrolled *n* number of times.

Specify loop unrolling on the command line You can also specify loop unrolling on the compiler command line, with option **-Hunroll=loop_count, size**. For example:

```
hc cwhetd.c -Hunroll=3,20 -Hunroll=4,10
```

is equivalent to:

```
#pragma Loop_factor(3,20)
#pragma Loop_factor(4,10)
```

G.3 How the Compiler Unrolls Loops

Requirements for unrolling The loops to be unrolled must have certain characteristics, in addition to meeting the selection criteria you specify with **#pragma Loop_factor**.

To be a candidate for unrolling, the following conditions must be met:

- The loop control variable must not be modified in the body of the loop.
- The loop must have constant stride. That is, the controlling loop variable must increase or decrease by a constant amount each time around the loop.
- The resulting *loop_count* must be greater than 1 (one); otherwise, the loop is not unrolled.
- The loop must not contain **switch** statements, calls to other functions, and the like; only loops containing simple instructions are unrolled.

Number of times a loop is unrolled

The loop count determines how many times a loop is unrolled. A general loop — that is, a loop for which the number of iterations is not known — is unrolled *loop_count* times, and then a remainder of the iterations (up to *loop_count* - 1) are executed by a following loop that is not unrolled.

A constant loop (one where the number of iterations is known) with less than *loop_count* iterations is unrolled *n* times, and no loop-test conditional jumps are performed. The same happens for constant loops with more than *loop_count* iterations but less than *loop_count* * 2 iterations.

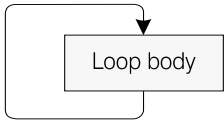
The following example illustrates what might happen for a constant loop with more than *loop_count* * 2 iterations.

Suppose you specify:

```
#pragma Loop_factor(3,100)
```

and the following **for** loop occurs somewhere in your program:

```
for (i = 1; i <= 8; i++) {
```



```
}

```

The diagram shows a rectangular box labeled "Loop body". A curved arrow starts from the bottom of the box, goes down and then left, and then curves up and right to point at the top of the box, representing the iteration of the loop.

If code generated for the body of the loop amounts to less than 100 intermediate instructions, the loop might be unrolled as follows:

```
for ( i = 1; i <= 6; i++ ) {
```

Loop body

```
  i + 1;
```

Loop body

```
  i + 1;
```

Loop body

```
  i + 1;  
}
```

Loop body

```
  i + 1;
```

Loop body

```
  i + 1;
```