restrict

In the C programming language, **`restrict`** is a keyword, introduced by the C99 standard,[1] that can be used in pointer declarations. By adding this type qualifier, a programmer hints to the compiler that for the lifetime of the pointer, no other pointer will be used to access the object to which it points. This allows the compiler to make optimizations (for example, vectorization) that would not otherwise have been possible.

`restrict` limits the effects of pointer aliasing, aiding optimizations. If the declaration of intent is not followed and the object is accessed by an independent pointer, this will result in undefined behavior.

Optimization

If the compiler knows that there is only one pointer to a memory block, it can produce better optimized code. For instance:

```
void updatePtrs(size_t *ptrA, size_t *ptrB, size_t *val)
{
  *ptrA += *val;
  *ptrB += *val;
}
```

In the above code, the pointers `ptrA`, `ptrB`, and `val` *might* refer to the same memory location, so the compiler may generate less optimal code:

```
; Hypothetical RISC Machine.
ldr r12, [val]     ; Load memory at val to r12.
ldr r3, [ptrA]     ; Load memory at ptrA to r3.
add r3, r3, r12    ; Perform addition: r3 = r3 + r12.
str r3, [ptrA]     ; Store r3 to memory location ptrA, updating the value.
ldr r3, [ptrB]     ; 'load' may have to wait until preceding 'store' completes.
ldr r12, [val]     ; Have to load a second time to ensure consistency.
add r3, r3, r12
str r3, [ptrB]
```

However, if the `restrict` keyword is used and the above function is declared as

```
void updatePtrs(size_t *restrict ptrA, size_t *restrict ptrB, size_t *restrict val);
```

then the compiler is allowed to *assume* that `ptrA`, `ptrB`, and `val` point to different locations and updating the memory location referenced by one pointer will not affect the memory locations referenced by the other pointers. The programmer, not the compiler, is responsible for ensuring that the pointers do not point to identical locations. The compiler can e.g. rearrange the code, first loading all memory locations, then performing the operations before committing the results back to memory.

```
ldr r12, [val]  ; Note that val is now only loaded once.
ldr r3, [ptrA]  ; Also, all 'load's in the beginning ...
ldr r4, [ptrB]
add r3, r3, r12
add r4, r4, r12
```

```
str r3, [ptrA]  ; ... all 'store's in the end.
str r4, [ptrB]
```

The above assembly code is shorter because `val` is loaded only once. Also, since the compiler can rearrange the code more freely, the compiler can generate code that executes faster. In the second version of the above example, the `store` operations are all taking place after the `load` operations, ensuring that the processor won't have to block in the middle of the code to wait until the `store` operations are complete.

Note that the real generated code may have different behaviors. Benefit with the above mini-example tends to be small, and in real-life cases large loops doing heavy memory access tends to be what is really helped by restrict.

As mentioned above, how incorrect code behaves is undefined, the compiler only ensures the generated code works properly if the code follows the declaration of intent.

Support by C++ compilers

C++ does not have standard support for `restrict`, but many compilers have equivalents that usually work in both C++ and C, such as the GCC's and Clang's `__restrict__`, and Visual C++'s `__declspec(restrict)`. In addition, `__restrict` is supported by those three compilers. The exact interpretation of these alternative keywords vary by the compiler:

- In Unix-style compilers such as GCC and Clang, `__restrict` and `__restrict__` mean exactly the same as their C counterpart. Extensions include allowing them to be applied to reference types and `this`.[2]
- In Visual C++, multiple no-alias qualifiers are provided:
  1. `__declspec(restrict)` applies to the function declaration and hints that the *returned* pointer is not aliased.
  2. `__restrict` is used in the same place as `restrict`, but the no-alias hint does not propagate as in `restrict`. It is also extended for union types.

Compiler warnings

To help prevent incorrect code, some compilers and other tools try to detect when overlapping arguments have been passed to functions with parameters marked `restrict`.[3] The CERT C Coding Standard considers misuse of `restrict` and library functions marked with it (EXP43-C) a probable source of software bugs, although as of November 2019 no vulnerabilities are known to have been caused by this.[4]

References

1. Drepper, Ulrich (October 23, 2007). "Memory part 5: What programmers can do" (https://lwn.net/Articles/255364/). *What every programmer should know about memory*. lwn.net. "...The default aliasing rules of the C and C++ languages do not help the compiler making these decisions (unless restrict is used, all pointer accesses are potential sources of aliasing). This is why Fortran is still a preferred language for numeric programming: it makes writing fast code easier. (In theory the restrict keyword introduced into the C language in the 1999 revision should solve the problem. Compilers have not caught up yet, though. The reason is mainly that too much incorrect code exists which would mislead the compiler and cause it to generate incorrect object code.)"

2. "Restricted Pointers" (https://gcc.gnu.org/onlinedocs/gcc/Restricted-Pointers.html). *Using the GNU Compiler Collection (GCC)*.
3. "Warning Options: -Wrestrict" (https://gcc.gnu.org/onlinedocs/gcc-8.1.0/gcc/Warning-Options.html#index-Wrestrict). *GCC*. Retrieved 19 November 2019.
4. "EXP43-C. Avoid undefined behavior when using restrict-qualified pointers" (https://wiki.sei.cmu.edu/confluence/display/c/EXP43-C.+Avoid+undefined+behavior+when+using+restrict-qualified+pointers). *SEI CERT C Coding Standard*. Retrieved 19 November 2019.

- "ISO/IEC 9899:TC2 Committee Draft" (http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf) (PDF). ISO. May 6, 2005. pp. 108–112. Retrieved 2008-12-22.

External links

- Demystifying The Restrict Keyword (https://cellperformance.beyond3d.com/articles/2006/05/demystifying-the-restrict-keyword.html): explanation and examples of use
- Walls, Douglas. "How to Use the restrict Qualifier in C" (https://www.oracle.com/technetwork/server-storage/solaris10/cc-restrict-139391.html). Oracle™. Retrieved 2012-11-21.
- Restricted Pointers in C (https://www.lysator.liu.se/c/restrict.html): the original rationale behind the definition

---