Faster than C

Judging the performance of programming languages, usually C is called the leader, though Fortran is often faster. New programming languages commonly use C as their reference and they are really proud to be only so much slower than C. Few language designer try to beat C.

What does it take for a language to be faster than C?

Better Aliasing Information

Aliasing describes the fact that two references might point to the same memory location. For example, consider the canonical memory copy (not `memcpy` from `stdlib.h`!):

```
void* memcopy(void* dst, const void* src, size_t count) {
   while (count--) *dst++ = *src++;
   return dst;
}
```

Depending on the target architecture, a compiler might perform a lot of optimizations with this code. For example, on a modern x86 with the SSE instruction MOVDQU, it could copy 16 Byte blocks instead of 4 Byte (`sizeof(void*)`). Unfortunately, no. Due to aliasing, `dst` could for example be `src+1`. In this case, the result must be the first word `*src` repeated `count` times at `dst`. The compiler is not allowed to use `MOVDQU` due to the semantics of C.

In C99 the `restrict` keyword was added, which we could use here to encode that `src` and `dst` are different from all other references. This mechanism helps in some cases, but not in our example.

Fortran semantics say that function arguments never alias and there is an array type, where in C arrays are pointers. This is why Fortran is often faster than C. This is why numerical libraries are still written in Fortran. However, it comes at the cost of pointer arithmetic.

A language which wants to be faster than C should provide semantics where aliasing can be better analyzed by the compiler.

Push Computation to Compile-Time

Doing things at compile time reduces the run time. Of course, C compilers do this for trivial cases like `1+2`, where the addition is already handled at compile time.

However, languages with nice meta-programming support enable the programmer to do similar application specific optimizations. A simple example, we could optimize `fib(20)` to `6765`, without the compiler knowing about Fibonacci numbers.

For a real example, the Eigen C++ library for linear algebra uses C++ templates to avoid copies and be lazy about computations. Of course, Lisp is the grandfather of this technique with its macro system. For example, there is a nice anecdote about a student using Scheme for an assignment. Basically, the programmer can modify the abstract syntax tree during compilation. The trade-off with such meta programming features is complexity. Programmer underestimate the difficulty to write correct macros like they underestimate the difficulty to write correct concurrent programs.

A language designer should think about meta programming. Something Turing-complete like C++ templates, seems to be beneficial for performance.

Runtime Optimization

At runtime there is dynamic information which is not available to a static compiler. Any specific example could be duplicated by a C program, but usually it is not feasible. The trick of profile-guided optimizations solves only a small part of the problem.

What becomes especially easy at runtime is whole-world optimization. While this is possible statically, the C semantics (compilation units) and the mandatory preprocessor make it difficult for the compiler. Even Python can beat C by inlining across file borders.

Of course, there are downsides to using a JIT and especially in systems- and embedded programming it is not appropriate. So there might be examples where Java, C# or others beat C, but they do not threaten C's niche.

Instruction Level Parallelism

Modern CPUs provide SIMD instructions for vectors, like Intel's AVX extension. While this can be use in C with intrinsics, it could be made much easier in another language. Exploiting it automatically by the compiler has not been that successful so far, so language support should be beneficial.

Conclusion

Aliasing information is the only one where I am certain about speed improvements, because it is impossible to reach Fortran-speed in C. The other ideas are more about making it easier to write faster programs.

Discussion on Hacker News and Reddit Programming.

Language designers, also read about my five mistakes in programming language design.

An older version of this has been published in Hacker Monthly (issue 26).

Also interesting: We Need More Compute Power, The Spirit of C, and A Better C.