

## Re: Cracking DES with C++ is faster than Java?

**Source:** <http://coding.derkeiler.com/Archive/Java/comp.lang.java/2004-04/0503.html>

---

**From:** Andrew Swallow ([am.swallow\\_at\\_eatspam.btinternet.com](mailto:am.swallow_at_eatspam.btinternet.com))

**Date:** 04/28/04

Date: Wed, 28 Apr 2004 15:43:48 +0000 (UTC)

"Ernst Lippe" <[ernstl-at-planet-dot-nl@ignore.this](mailto:ernstl-at-planet-dot-nl@ignore.this)> wrote in message news:pan.2004.04.28.14.04.06.707444@ignore.this...

> On Wed, 28 Apr 2004 12:18:12 +0000, Andrew Swallow wrote:

>

> > "Jerry Coffin" <[jcoffin@taeus.com](mailto:jcoffin@taeus.com)> wrote in message

> > news:b2e4b04.0404271646.24e70541@posting.google.com...

> > [snip]

> >> *If anything, it looks to me like thie situation is getting worse: I*

> >> *don't think I've looked at anything in the last 5 years that was*

> >> *nearly AS CLOSE to optimal as Control Data's FORTRAN IV compiler*

> >> *typically produced 20+ years ago!*

> >>

> > *Computers can access fixed locations in memory*

> > *faster that relative locations.*

>

> *I assume that with "fixed location" you mean addresses that are known at link*

> *time, when the executable is produced.*

>

> *I don't think that your statement is true in general, it all depends on the*

> *circumstances. One of the disadvantages of using absolute memory addresses is*

> *that the size of instruction must always be larger than the number of bits in*

> *the address space. Many CPU's have built-in support for addressing*

> *small-sized offsets to a stack-pointer and in general the size of these*

> *instructions is smaller than the size of the instructions that use absolute*

> *addresses. When the instruction is longer, either it will take more time to*

> *fetch it from memory, or it will decrease the effective number of instructions*

> *that can be held in the instruction cache.*

>

Do not forget the time it required to load the pointers.

## comp.lang.java: Re: Cracking DES with C++ is faster than Java?

- > *Even on CPU architectures where both types of instructions have the same size,*
- > *there may not be any difference in execution speed. For most modern CPU's*
- > *memory access is the main bottle-neck. Simple arithmetic operations to*
- > *registers, such as adding a small offset, are so fast relative to the time*
- > *it*
- > *takes to fetch/store the contents of some address in memory, that there is*
- > *no*
- > *difference. Also, virtually all modern computers have memory caches, and*
- > *the*
- > *performance depends heavily on whether the contents are available in the*
- > *cache. But for the memory cache there is no difference between a fixed*
- > *address*
- > *and an address that has been computed dynamically, the only thing that is*
- > *important if that address has been used recently.*
- >
- >
- > *Recursion requires*
- > *local variables to be stored on the stack, i.e. to*
- > *be accessed using relative locations.*
- >
- > *The overhead of allowing recursion (if any) is minimal on modern*
- > *CPU's. Anyhow, all modern programming languages (even Fortran) allow it,*
- > *so*
- > *apparently language designers believe that the benefits are sufficiently*
- > *important.*
- >

Fortran IV did not support recursion. It was even designed so that its compilers did not need to use recursion. One of the reason that only simple calculations were permitted in array subscripts.

- > *Data structures*
- > *on the heap are even more complicated to access.*
- >
- > *In most cases this statement is false, in most computers data structures*
- > *on*
- > *the heap are simply identified by their address which is not more*
- > *complicated*
- > *than using absolute addresses or locations on the stack. I have the*
- > *feeling*
- > *that you believe that heaps only occur in languages, that have a*
- > *relocating*
- > *garbage collector (see below), but it is standard usage to refer to the*
- > *memory*
- > *area from which dynamically sized memory is allocated as "the heap". For*
- > *example, the malloc in C uses a heap.*
- >

## comp.lang.java: Re: Cracking DES with C++ is faster than Java?

As a user of programs written in C I have noticed. It does not matter if the runtime garbage collector is in the program or the operating system, both its time delays and probability of failure are still there. A construct that is definitely not suitable for use in realtime or safety critical programs.

- >
- > > *Dynamic storage is very bad, you can see the computer*
- > > *stop for several thousand million clock cycles whilst the*
- > > *garbage collector tries to find some more memory.*
- >
- > *What do you mean by dynamic storage? The standard meaning of the term is*
- > *storage where the actual memory size of an object is not known at compile*
- > *time. This is pretty common in most computer languages, and in most cases*
- > *this*
- > *is not at all a performance problem.*
- >
- > *You are probably referring to relocating garbage collectors that can*
- > *change*
- > *the actual memory addresses where objects are located. Traditionally,*
- > *these*
- > *used to have the problems that you describe, but there have been important*
- > *improvements (generation scavenging, multi-threading) and most of these*
- > *problems have disappeared.*
- >

Only yesterday did I saw my PC crash because of a stack overflow.

- > *Anyhow, I don't think that your argument that fixed addresses are*
- > *faster is very relevant. Virtually all computer programs are written*
- > *with procedures that can accept varying arguments (that was already*
- > *the case for these old Fortran programs). So there are hardly any*
- > *instances*
- > *where computers actually use fixed addresses, because most code*
- > *relies on variables that have been passed as arguments instead of*
- > *global variables.*
- >

One of the speed optimisations Fortran programmers made was to pass global variables around in the common blocks rather than as parameters.

> *Ernst Lippe*

Andrew Swallow

>