Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by justspam03 on Fri, 19 Mar 2004 10:06:02 GMT
View Forum Message <> Reply to Message

JD Smith <jdsmith@as.arizona.edu> wrote in message news:<pan.2004.03.19.01.11.11.181250@as.arizona.edu>...

>>>
>>> Setting 12000 random values (key: string, value: integer):
>>> mean 0.15 ms per entry (total ~2 seconds)
>>> Random access of 12000 values from this set:
>>> mean 0.3 ms per access, (total ~3.5 s)
>>>
>
> I think he means just using linear search, ala WHERE. This
> technically is a form of hashing: it just happens to utilize just one
> hash bucket (alright, a useless form ;).
>
> Anyway, that's some clever use of function name searching for free
> hashing. IDL does not expose any internal hashing functionality, but
> Craig wrote a hash object which works reasonably well. I don't find
> it on his site, but perhaps he'd be willing to share.
>

JD is right of course in that this not a hash, but an associative array and the linear execution time ('set' has to check for existing keys) is due to the use of 'where'.

Performance was good enough for me, though (with only a handful of keys per array), so I didn't care any further about using hashes.

You can find a hash_table implementation on the RSI user contribution site. Quick performance test for 12000 sets/gets:

#hashes	set/get per entry (ms)
13	6
101	0.8
1001	0.15
12000	0.08/0.05

Talking about it:
How would you calculate a hash value from a string? In C I would base it on the ASCII value of the chars, but in IDL? Above mentioned implementation converts the string via byte() and then loops over the resulting array. Is there a faster way (loops always take so long)?
Cheers
Oliver