Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by JD Smith on Fri, 19 Mar 2004 01:11:11 GMT
View Forum Message <> Reply to Message

On Thu, 18 Mar 2004 21:40:38 +0100, Sidney Cadot wrote:

> Oliver Thilmann wrote:
>
>> Hi,
>>
>> your example is just generic for the kind of problem you want
>> to solve, I assume. Otherwise why not use a hash? A very
>> simple implementation (unsorted arrays) on a Pentium IV,
>> 2.6 GHz, IDL 6.0 yields
>>
>> 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)
>>
>> Is the access via call_function much faster?
>
> I'm afraid to sound terribly stupid here, but is there support for
> hashing in IDL? I haven't been able to find it.
>
> My laborious trick is nothing more than circumventing the lack of
> hashing as a standard feature in IDL (by piggybacking on the internal
> hashing IDL uses for function names). If you know a better way, I would
> be very much interested!

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.

What your method fails to offer that a real hash would is the ability to create new hash entries at run-time: all of your hash strings are fixed at runtime, which means you could transform them beforehand to integers (e.g. just use the sort index), and index a large static array instead, which would be orders of magnitude faster. Strings of course may be more convenient, but in the fixed string-space case they aren't technically necessary. Another variant on this would be akin to a C macro: just create a batch file for input using named variables like:

```
;; map_include.pro
dick=0
frank=1
harry=2
tom=3
...
zappa=11999

map=[456,222,789,123,...,777]


;; my map-using routine
pro mymaproutine
  @map_include
  print,map[zappa], ' is not as good as ',map[frank]
end
```

That will take a bit to compile, but once it compiles it should fly, and of course allows arrays, etc. If you insist on having a string mapper function, you can cheat:

```
function f, name
  @map_include
  ind=routine_names(name,FETCH=0)
  return,map[ind]
end
```

This doesn't really solve the real problem, which would include the ability to add more string keys at runtime, but it may be as much as you need. In the meantime, write your friendly IDL support techs and request a decent internal hash type, or at least a front end to a hash interface!

JD
