PATCH: 1.9: gc_mark_children: Avoid gc_mark() tail recursion, use goto again.

07/16/2011 04:45 PM - kstephens (Kurt Stephens)

Status: Closed
Priority: Normal
Assignee: authorNari (Narihiro Nakamura)
Target version: 2.0.0

Description
Minor GC improvement.
Avoid recurring into gc_mark() when "goto again;" is sufficient.
-- KAS

History
#1 - 07/16/2011 06:12 PM - kosaki (Motohiro KOSAKI)
- Category set to core
- Status changed from Open to Assigned
- Assignee set to authorNari (Narihiro Nakamura)

#2 - 07/16/2011 08:04 PM - shyouhei (Shyouhei Urabe)
- I believe my compiler is smart enough to do that optimization and goto is considered harmful.

#3 - 07/17/2011 01:01 AM - kstephens (Kurt Stephens)
Not aware of any compiler that is smart enough to optimize away the second half of gc_mark() (lines 1616-1628), when tail called from gc_mark_children().

gc_mark_children() already uses goto.

#4 - 07/17/2011 01:15 AM - mame (Yusuke Endoh)
Personally I don't think goto matters so much in GC implementation. But I'm not sure if the patch is actually so effective. Did you benchmark? If you did, could you show it?

--
Yusuke Endoh mame@tsg.ne.jp

#5 - 07/17/2011 01:53 AM - normalperson (Eric Wong)
Kurt Stephens ks.ruby@kurtstephens.com wrote:

http://redmine.ruby-lang.org/issues/5033

In [ruby-core:36931], ko1 told us GC eats stack when marking nested objects. Kurt's patch should allow us to run smaller pthread stack sizes while still supporting deeply-nested structures.

Kurt: can you test a smaller stack size with your patch with some deeply-nested objects?

Thanks, I'm excited about this patch :D (but unlikely to have time to test until next week).

#6 - 07/19/2011 10:50 PM - kstephens (Kurt Stephens)
I don't know of a reliable means to record max stack depth, but the speed isn't necessarily better or worse:

bash ./profile-gc-mark

- export 'CFLAGS=-O2 -I/opt/local/include' LDFLAGS=-L/opt/local/lib
CFLAGS=-O2 -I/opt/local/include
LDFLAGS=-L/opt/local/lib
for branch in trunk trunk-gc-mark-optimization
git checkout trunk Switched to branch 'trunk' Your branch is ahead of 'origin/trunk' by 10 commits.
prefix=/Users/stephens/local/ruby-trunk
make test
make test
real    0m51.808s
user    0m19.760s
sys 0m13.118s

make test
real    0m49.938s
user    0m19.598s
sys 0m13.396s

for branch in trunk trunk-gc-mark-optimization
git checkout trunk-gc-mark-optimization Switched to branch 'trunk-gc-mark-optimization'
prefix=/Users/stephens/local/ruby-trunk-gc-mark-optimization
make test
make test
real    0m51.752s
user    0m19.526s
sys 0m13.336s

make test
real    0m50.157s
user    0m19.735s
sys 0m13.487s

BTW: make test-all in trunk hangs for me on OS X 64.

The space improvements would occur for NODES with deep obj->as.node.u3.node, arrays with deep last elements, and OBJECTS where the last IVAR is deep.

obj->as.file.fptr->write_lock, obj->as.regexp.src, obj->as.match.str, obj->as.rational.den, obj->as.complex.imag are likely to be not deep.

So maybe this patch is pointless, except for the removal of the unnecessary "long i" variable in the T_OBJECT case/loop.

#7 - 07/20/2011 08:29 AM - normalperson (Eric Wong)
Eric Wong normalperson@yhbt.net wrote:
Kurt Stephens ks.ruby@kurtstephens.com wrote:

http://redmine.ruby-lang.org/issues/5033

In [ruby-core:36931], ko1 told us GC eats stack when marking nested objects. Kurt's patch should allow us to run smaller pthread stack sizes while still supporting deeply-nested structures.

shyouhei appears right about compilers being able to optimize this for the easy cases.

However "inspect" on deeply-nested structures is still stack hungry and causes SystemStackErrors on my machine if I try to "p" a deeply-nested array or hash.

Kurt: can you test a smaller stack size with your patch with some deeply-nested objects?

I was playing around with something like this (but did not get any useful results/conclusion either way):

def deeper!(array_or_hash, depth)
if depth > 6000
$last = array_or_hash[0] = {}  
else  
array_or_hash[0] = [ deeper!([], depth += 1) ]  
end  
end  
  
orig = {}  
deeper!(orig, 0)  
5000.times do |i|  
deeper!($last, 0)  
end  
p $$  
# give GC something to much on  
100000.times { [ i ] .i.to_s }  
p :done  

#8 - 07/20/2011 10:23 AM - kosaki (Motohiro KOSAKI)  

- git checkout trunk Switched to branch 'trunk'  
- make test  
  
real Â Â 0m49.938s  
user Â Â 0m19.598s  
sys Â Â 0m13.596s  

- git checkout trunk-gc-mark-optimization Switched to branch 'trunk-gc-mark-optimization'  
  
real Â Â 0m50.157s  
user Â Â 0m19.735s  
sys Â Â 0m13.487s  

Hmm...  
Don't you have any good result? Usually we reject the ticket if the  
optimization patch  
don't show any better result.  

I'm waiting someone which is interesting this ticket post the result.  
But, for remark, if nobody success to get it, I'll reject this.  

Thanks.  

#9 - 07/20/2011 03:21 PM - kstephens (Kurt Stephens)  

There is a time improvement for Arrays deeply nested at their tails:  

Stock:  

- ./miniruby -e '  
x = [ nil ]  
10000000.times do | i |  
x[0] = [ i ]  
x = x[0]  
end  
puts :OK  
system "ps -l -p $#{$$}"  
OK  
'  
UID PID PPID F CPU PRI NI  
SZ RSS WCHAN S ADDR TTY TIME CMD  
501 96240 84698 4006 0 31 0 3096 -  
S+ fda67e0 ttys009 0:02.28 ./miniruby -e "  
real 0m2.293s  
user 0m2.275s  
sys 0m0.013s  

- ./miniruby -e '  
x = [ nil ]  
10000000.times do | i |  
x[0] = [ i ]  
x = x[0]  
end  
puts :OK  
system "ps -l -p $#{$$}"  
'  
OK  
UID PID PPID F CPU PRI NI  
SZ RSS WCHAN S ADDR TTY TIME CMD  
501 20407 84698 4006 0 31 0 2447588 3096 -  
S+ fda67e0 ttys009 0:02.08 ./miniruby -e "  
real 0m2.293s  
user 0m2.275s  
sys 0m0.013s  

#10 - 07/20/2011 04:23 PM - kosaki (Motohiro KOSAKI)

real    0m2.096s
user    0m2.074s
sys 0m0.014s

#10 - 07/20/2011 04:23 PM - kosaki (Motohiro KOSAKI)

real    0m2.293s
user    0m2.275s
sys 0m0.013s

real    0m2.096s
user    0m2.074s
sys 0m0.014s

Nice ;)
I'm looking forward nari-san's responce.

#11 - 07/20/2011 07:09 PM - authorNari (Narihiro Nakamura)

Hi,
Kurt Stephens wrote:

Minor GC improvement.

Avoid recurring into gc_mark() when "goto again;" is sufficient.

-- KAS

Nice try!

I read your patch.
In some program, GC is improved.

$ cat r.rb
GC::Profiler.enable
x = ['s']
10_000_000.times do
  x[0] = x.dup
end
p GC::Profiler.total_time

origin: 0.28999999999999976
KAS's patch: 0.2299999999999993

I will accept this patch if GC performance is decrased in other programs.

#12 - 07/27/2011 12:57 PM - kstephens (Kurt Stephens)

There will be improvements for programs that have large numbers of Rational and Complex numbers. If someone has a suitable benchmark please let me know. Otherwise, I'll write something simple.

#13 - 10/05/2012 01:27 PM - authorNari (Narihiro Nakamura)

- Status changed from Assigned to Closed
- % Done changed from 0 to 100

I've committed part of your patch in r37075. Thanks!

Files

gc-mark-optimization.patch 3.17 KB 07/16/2011 kstephens (Kurt Stephens)