Ruby master - Feature #12180

switch id_table.c variant

03/15/2016 06:51 PM - funny_falcon (Yura Sokolov)

Status:	Closed	
Priority:	Normal	
Assignee:	ko1 (Koichi Sasada)	
Target version:		

Description

Currently used variant is 'binary search in small table + hash for large tables'.
But for contemporary CPU it may be better to do linear scan for small tables.

It is already implemented in id_table.c and numbered as 35.

Tested with simple Redmine installation on Intel Haswell i7-4770 CPU @ 3.40GHz

- trunk: Requests per second: 27.79 [#/sec] (mean)
- with switched implementation: Requests per second: 28.87 [#/sec] (mean)

Associated revisions

Revision 4c2d014e - 01/25/2017 03:03 AM - ko1 (Koichi Sasada)

swithc id_table data structure.

- id_table.c: swtich to "simple open addressing with quadratic probing" by Yura Sokolov. For more detail measurements, see [Feature #12180]
- id_table.c: remove other algorithms to simplify the source code.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@57416 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 57416 - 01/25/2017 03:03 AM - ko1 (Koichi Sasada)

swithc id_table data structure.

- id_table.c: swtich to "simple open addressing with quadratic probing" by Yura Sokolov. For more detail measurements, see [Feature #12180]
- id_table.c: remove other algorithms to simplify the source code.

Revision 57416 - 01/25/2017 03:03 AM - ko1 (Koichi Sasada)

swithc id_table data structure.

- id_table.c: swtich to "simple open addressing with quadratic probing" by Yura Sokolov. For more detail measurements, see [Feature #12180]
- id_table.c: remove other algorithms to simplify the source code.

Revision 57416 - 01/25/2017 03:03 AM - ko1 (Koichi Sasada)

swithc id_table data structure.

- id_table.c: swtich to "simple open addressing with quadratic probing" by Yura Sokolov. For more detail measurements, see [Feature #12180]
- id_table.c: remove other algorithms to simplify the source code.

History

#1 - 03/15/2016 07:11 PM - funny_falcon (Yura Sokolov)

Well, in fact '22' implementation (simple open addressing with quadratic probing) is even faster:

Requests per second: 29.67 [#/sec] (mean)

And it uses just 0.5-1% more memory.

#2 - 03/16/2016 02:45 AM - ko1 (Koichi Sasada)

Hi,

> simple Redmine installation

What is this benchmarking?

Thanks,
Koichi

#3 - 03/16/2016 05:25 AM - funny_falcon (Yura Sokolov)

I just install Redmine - the same software that powers bugs.ruby-lang.org ,
then add several test issues, and bench it with 'apache benchmark':
ab -n 1000 -c 10 http://localhost:3000/projects/general/issues
(general is a name of test project)

#4 - 03/16/2016 05:36 PM - funny_falcon (Yura Sokolov)

In other words, big web applications are too big for global method cache, and too polymorphic for inline cache.
So general method lookup is inevitable.
With current choice for id_table implementation (34) it takes about 4.5-6%CPU.
With implementation 22 it takes just 2.5-3% CPU.

#5 - 10/07/2016 12:14 PM - funny_falcon (Yura Sokolov)

Please, reconsider benchmarking it with realworld applications.

#6 - 11/08/2016 05:19 AM - ko1 (Koichi Sasada)

I measured them like st_table:
https://gist.github.com/ko1/b8714907c328bc29ba9501cfcb1abc8a

And ffalcon's 22 implementation is best. We need to switch to it.

Naruse-san, can I switch it now?

#7 - 11/08/2016 05:23 AM - ko1 (Koichi Sasada)

Yura: can you make a table more small with a few entries?

And do you have a commit permission?

#8 - 12/21/2016 10:21 AM - hsbt (Hiroshi SHIBATA)

I asked this feature to ko1. He said that it's better to merge at Ruby 2.5.

We will evaluate this after Ruby 2.4.0 release.

#9 - 01/19/2017 05:27 AM - shyouhei (Shyouhei Urabe)

- Assignee set to ko1 (Koichi Sasada)

- Status changed from Open to Assigned

#10 - 01/20/2017 03:17 AM - shyouhei (Shyouhei Urabe)

We looked at this issue in yesterday's developer meeting.

(As ko1 measured before,) we confirmed that ffalcon's implementation is the fastest. Ko1 is assigned to switch to that.

#11 - 01/20/2017 03:46 PM - funny_falcon (Yura Sokolov)

Excuse me for not reacting on discussion.
I forgot to turn on mail notifications for this issue.

> can you make a table more small with a few entries?

It could be 25% smaller for any size (on x86_64) at cost of second memory lookup
if key and value stored in separate arrays. It should be measured.
Single pointer still could be used as with "USE_CALC_VALUES".

ko1, will you just switch id or remove other implementations?
if first, I may prepare patch for current source, otherwise I will wait for commit.

> And do you have a commit permission?

No I haven't. And I doubt I should have.

#12 - 01/20/2017 05:11 PM - funny_falcon (Yura Sokolov)

Oh, to reduce hash size either max serial_id should be 0x7fffffff instead of 0xffffffff
(cause 1 bit is stolen for collision check),
or collision bit removed (it will increase time for "miss" and decrease for "hit").

#13 - 01/25/2017 03:03 AM - ko1 (Koichi Sasada)

- Status changed from Assigned to Closed

Applied in changeset r57416.

---

swithc id_table data structure.

- id_table.c: swtich to "simple open addressing with quadratic probing" by Yura Sokolov. For more detail measurements, see [Feature #12180]
- id_table.c: remove other algorithms to simplify the source code.

#14 - 01/25/2017 03:07 AM - ko1 (Koichi Sasada)

Yura:
Sorry for my laziness. I committed it (remain only your algorithm) and cleanup source code.
Pls make another ticket to improve id_table structure.

#15 - 01/31/2017 02:14 PM - funny_falcon (Yura Sokolov)

Created https://bugs.ruby-lang.org/issues/13174 with implementation smaller in memory.

Files

0001-id_table.c-switch-id_table-variant.patch	767 Bytes	03/15/2016	funny_falcon (Yura Sokolov)