Ruby master - Feature #10423

[PATCH] opt_str_lit*: avoid literal string allocations

10/24/2014 03:25 AM - normalperson (Eric Wong)

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

Description

Patch also downloadable at: http://80x24.org/spew/m/opt_str_lit-v4%40m.txt
Broken-out commits in the "opt_str_lit-v4" branch of
git://bogomips.org/ruby.git (and http://bogomips.org/ruby.git)

This obsoletes Features #10326, #10329, and #10333

Changes since -v3

- cleanup tests to be more DRY
- optimize allocations for Hash#fetch, String#r?partition
- split opt_str_lit into 3 instructions to reduce insn complexity
  (opt_str_freeze, opt_aset_with, opt_aref_with) are still gone.

While having one instruction is appealing in some ways,
hiding compile-time-resolvable branches behind it is
misleading and it should be easier-to-follow when divided
into three methods.

1. opt_str_lit_recv: for "literal string" receivers
   This no longer optimizes the method dispatch away
   for String#freeze, the method call now always happens
   (but allocation is avoided)

2. opt_str_lit_tmask: the most heavily-used
   This optimizes allocations away for "literal string" arguments

3. opt_str_lit_data: currently only used for Time#strftime
   We may remove this (and the strftime optimization)
   if it is too rare to be useful.

Benchmarks:

"make rdoc" performance improved from 66.3 => 63.3 sec

vm2_* benchmarks:

We take a speed hit from losing opt_aref_with/opt_aset_with
but I think the flexibility of the new instructions is worth it.

Speedup ratio: compare with the result of `trunk' (greater is better)

```
name    built
loop_whileloop2         1.000
vm2_array*              0.997
vm2_array_delete_lit*   2.204
vm2_array_include_lit*  2.372
vm2_bigarray*           1.002
vm2_bighash*            1.004
vm2_case*               1.030
vm2_defined_method*     1.013
```

```
vm2_dstr*               0.977
vm2_eval*               1.019
vm2_hash_aref_lit*      0.767
vm2_hash_aset_lit*      0.828
vm2_hash_delete_lit*    2.254
vm2_method*             1.010
vm2_method_missing*     1.010
vm2_method_with_block*  0.996
vm2_mutex*              0.954
vm2_newlambda*          1.010
vm2_poly_method*        0.974
vm2_poly_method_ov*     0.990
vm2_proc*               1.003
vm2_raise1*             0.992
vm2_raise2*             0.993
vm2_regexp*             1.053
vm2_send*               1.000
vm2_set_include_lit*    1.051
vm2_str_delete*         1.365
vm2_str_eq1*            2.282
vm2_str_eq2*            2.700
vm2_str_eqq1*           2.062
vm2_str_eqq2*           2.421
vm2_str_fmt*            1.271
vm2_str_gsub_bang_lit*  1.650
vm2_str_gsub_bang_re*   1.153
vm2_str_gsub_re*        1.110
vm2_str_plus1*          1.496
vm2_str_plus2*          1.618
vm2_str_tr_bang*        1.527
vm2_strcat*             1.406
vm2_super*              1.005
vm2_unif1*              0.985
vm2_zsuper*             0.992

------------------------------------------------------------
raw data:

[["loop_whileloop2",
  [[0.09168246667832136,
    0.09153273981064558,
    0.09135112632066011,
    0.09144351910799742,
    0.0913569862022996],
   [0.09150420501828194,
    0.09135987050831318,
    0.09148290101438761,
    0.09135456290096045,
    0.091412821784615521]],
 ["vm2_array",
  [[0.6448190519586205,
    0.6377620408311486,
    0.6456073289737105,
    0.636782786808908,
    0.646114731207491,
   [0.6434593833982944,
    0.6479324344545603,
    0.6385641889646649,
    0.6491997549310327,
    0.661029470153152911],
 ["vm2_array_delete_lit",
  [[0.44069126434624195,
    0.43327025789767504,
    0.44154794327914715,
    0.43351241294294596,
    0.4343419009819627],
   [0.24649471696466208,
```

```
    0.24813515041023493,
    0.2801106022670865,
    0.24796569626778364,
    0.254449966363608841]],
["vm2_array_include_lit",
 [[0.42663843277841806,
    0.42771619465202093,
    0.4342082254588604,
    0.4192065382376313,
    0.4339462127536535],
   [0.24434014037251472,
    0.23243559524416924,
    0.23569373413920403,
    0.2295856410637498,
    0.235210455022752291]],
["vm2_bigarray",
 [[5.9676302736625075,
    5.921381871215999,
    5.893002421595156,
    5.898605763912201,
    5.92879247572273],
   [5.918988090008497,
    5.953728860244155,
    5.910121874883771,
    5.894348439760506,
    5.88202034495770911]],
["vm2_bighash",
 [[3.5214368999004364,
    3.5914944410324097,
    3.5528544737026095,
    3.6002828497439623,
    3.595806434750557],
   [3.6454197568818927,
    3.5987599324434996,
    3.631806828081608,
    3.5076274275779724,
    3.50609814841300251]],
["vm2_case",
 [[0.16305183991789818,
    0.16213963273912668,
    0.1626761993393302,
    0.1703133536502719,
    0.16371900774538517],
   [0.1630518063902855,
    0.16097174491733313,
    0.16129930969327688,
    0.16106242593377829,
    0.160072404891252521]],
["vm2_defined_method",
 [[2.4309032559394836,
    2.4551019677892327,
    2.4333217879757285,
    2.4307468980550766,
    2.4270543046295643],
   [2.3963797464966774,
    2.4110122229903936,
    2.592901307158172,
    2.7106671230867505,
    2.68695262726396321]],
["vm2_dstr",
 [[1.206045768223703,
    1.1181026576086879,
    1.29782950039953,
    1.074333599768579,
    1.060404953546822],
   [1.0911998134106398,
    1.3560991054400802,
```

```
    1.096764899790287,
    1.0906088771298528,
    1.0831655990332365]]],
["vm2_eval",
 [[12.45692038256675,
    12.536506623961031,
    12.868694677948952,
    12.425183075480163,
    12.422813648357987],
   [12.490455374121666,
    12.190652490593493,
    12.194032781757414,
    12.581901006400585,
    13.230092607438564]]],
["vm2_hash_aref_lit",
 [[0.2547442754730582,
    0.25785687286406755,
    0.2552897445857525,
    0.2547551356256008,
    0.2546374099329114],
   [0.3131725639104843,
    0.3109410647302866,
    0.30428329203277826,
    0.30984809901565313,
    0.30932857654988766]]],
["vm2_hash_aset_lit",
 [[0.3067243071272969,
    0.307466727681458,
    0.3059097733348608,
    0.32101333420723677,
    0.3114894386380911],
   [0.3522613048553467,
    0.3512485157698393,
    0.3521018158644438,
    0.35033643897622824,
    0.351638876833021641]]],
["vm2_hash_delete_lit",
 [[0.4271302009001374,
    0.5321928421035409,
    0.42596039827913046,
    0.4215333154425025,
    0.4221466425806284],
   [0.2427450241521001,
    0.23863658774644136,
    0.23870530910789967,
    0.23786015156656504,
    0.241129901260137561]]],
["vm2_method",
 [[1.2295677298679948,
    1.2423510188236833,
    1.2246184339746833,
    1.213058383204043,
    1.3788639837875962],
   [1.2468778286129236,
    1.2022472834214568,
    1.319407593458891,
    1.3409598469734192,
    1.2468722229823471]]],
["vm2_method_missing",
 [[1.8783203130587935,
    2.0230442117899656,
    1.7176201958209276,
    1.790073310956359,
    1.7432779222726822],
   [1.710430753417313,
    1.8266426706686616,
    1.7299889298155904,
```

```
    1.7529844669625163,
    1.7007915144786239]]],
["vm2_method_with_block",
 [[1.3849838990718126,
    1.345238379202783,
    1.3416069420054555,
    1.3473590090870857,
    1.3418097402900457],
   [1.3468122174963355,
    2.0074896160513163,
    1.3560442496091127,
    1.353834005072713,
    1.35251552145928141]]],
["vm2_mutex",
 [[0.6452304208651185,
    0.7461022492498159,
    0.7083938084542751,
    0.7519227871671319,
    0.7953951777890325],
   [0.6722144978120923,
    0.6991892093792558,
    0.699041954241693,
    0.8469044668599963,
    0.7956293905153871]]],
["vm2_newlambda",
 [[0.7562470361590385,
    0.7670294912531972,
    0.7572819162160158,
    0.8686427380889654,
    0.772026221267879],
   [0.7586023258045316,
    0.7559101320803165,
    0.7497995179146528,
    0.7534453617408872,
    0.75106958951801061]]],
["vm2_poly_method",
 [[2.1563803972676396,
    1.9838355090469122,
    2.0223182002082467,
    1.9927900172770023,
    2.055130822584033],
   [2.03470276389271,
    2.187690651975572,
    2.162980039604008,
    2.143593772314489,
    2.048294037580491]]],
["vm2_poly_method_ov",
 [[0.22944753244519234,
    0.2304667690768838,
    0.22847569175064564,
    0.22843772917985916,
    0.22901444043964148],
   [0.2298243623226881,
    0.2334523554891348,
    0.23789969086647034,
    0.23442872054874897,
    0.23049725405871868]]],
["vm2_proc",
 [[0.46109406277537346,
    0.4476210633292794,
    0.44980251509696245,
    0.4451689962297678,
    0.4452860997989774],
   [0.4545932151377201,
    0.4440324120223522,
    0.44979235529899597,
    0.4695265833288431,
```

```
    0.46356835961341861]],
 ["vm2_raise1",
  [[4.989030849188566,
    4.951415436342359,
    4.755166156217456,
    4.824490828439593,
    4.795467809773982],
   [4.797340838238597,
    4.799810292199254,
    4.792353850789368,
    4.966345896013081,
    5.0993040064349771]]],
 ["vm2_raise2",
  [[7.062344457022846,
    7.118377918377519,
    7.290325260721147,
    7.1321266973391175,
    7.341989707201719],
   [7.456813280470669,
    7.130581725388765,
    7.1143358228728175,
    7.1227226899936795,
    7.26876201946288351]]],
 ["vm2_regexp",
  [[1.0923050865530968,
    1.0876065697520971,
    1.0792832653969526,
    1.0728685557842255,
    1.098634515888989],
   [1.172722346149385,
    1.0436540711671114,
    1.0638451064005494,
    1.0391646642237902,
    1.0239013740792871]]],
 ["vm2_send",
  [[0.3253856906667352,
    0.31978365778923035,
    0.32439478673040867,
    0.3290995704010129,
    0.32464261166751385],
   [0.31982277520000935,
    0.32479251362383366,
    0.31968420650810003,
    0.3205655198544264,
    0.32221281714737415]]],
 ["vm2_set_include_lit",
  [[0.7017893251031637,
    0.7203339897096157,
    0.714060970582068,
    0.7558876499533653,
    0.7028816910460591],
   [0.6870902189984918,
    0.6755420127883554,
    0.6719080805778503,
    0.6900417571887374,
    1.14084662124514581]]],
 ["vm2_str_delete",
  [[0.671600878238678,
    0.6570039531216025,
    0.6776775699108839,
    0.6644531870260835,
    0.97628088388592],
   [0.508894819766283,
    0.5154125597327948,
    0.5064034061506391,
    0.5130562232807279,
    0.5056716529652476]]],
```

```
["vm2_str_eq1",
 [[0.42520802468061447,
   0.42302330397069454,
   0.42416366562247276,
   0.4216942973434925,
   0.4239108320325613],
  [0.23870286531746387,
   0.23699089046567678,
   0.2361262608319521,
   0.31979569140821695,
   0.26054865028709173]]],
["vm2_str_eq2",
 [[0.42534991446882486,
   0.4238837053999305,
   0.4327298905700445,
   0.42395070381462574,
   0.42501260805875063],
  [0.21500772424042225,
   0.21449210867285728,
   0.2152256891131401,
   0.21711387671530247,
   0.21528882998973131]]],
["vm2_str_eqq1",
 [[0.45151620265096426,
   0.4597599729895592,
   0.45280721783638,
   0.4582480965182185,
   0.45380780193954706],
  [0.26729187835007906,
   0.26626693457365036,
   0.2660442851483822,
   0.26674115750938654,
   0.27212413586676121]]],
["vm2_str_eqq2",
 [[0.45260279811918736,
   0.44771482422947884,
   0.4570333072915673,
   0.5335573730990291,
   0.4592890217900276],
  [0.2385527715086937,
   0.24223692156374454,
   0.24510123394429684,
   0.2744274791330099,
   0.24475668929517271]]],
["vm2_str_fmt",
 [[2.614012835547328,
   2.6062369514256716,
   2.5333361653611064,
   2.50343619287014,
   2.641179056838155],
  [1.9900542199611664,
   2.0983974151313305,
   1.9898552373051643,
   2.001521130092442,
   1.9963106652721763]]],
["vm2_str_gsub_bang_lit",
 [[1.1346126468852162,
   1.134159336797893,
   1.1467241132631898,
   1.170719631947577,
   1.149439207278192],
  [0.737271074205637,
   0.7308303005993366,
   0.730314377695322,
   0.7260715514421463,
   0.7231733025982976]]],
["vm2_str_gsub_bang_re",
```

```
 [[1.4583028750494123,
   1.4592840988188982,
   1.4590299287810922,
   1.4751051738858223,
   1.492074583657086],
  [1.3006651625037193,
   1.3510901927947998,
   1.2767879888415337,
   1.3884455161169171,
   1.277703725732863]]],
["vm2_str_gsub_re",
 [[1.7273316802456975,
   1.7103399503976107,
   1.7015334982424974,
   1.690703290514648,
   1.694624281488359],
  [1.5322920577600598,
   1.5391994584351778,
   1.5388264516368508,
   1.5372542077675462,
   1.5352015374228358]]],
["vm2_str_plus1",
 [[0.653569158166647,
   0.6917095249518752,
   0.6521412674337626,
   0.6697740163654089,
   0.7296328162774444],
  [0.46872936747968197,
   0.48235206957906485,
   0.46762560587376356,
   0.4766495209187269,
   0.4661587802693248]]],
["vm2_str_plus2",
 [[0.6671840893104672,
   0.6488652648404241,
   0.6509246686473489,
   0.6638444662094116,
   0.6557880509644747],
  [0.43602617271244526,
   0.4365514535456896,
   0.45153723005205393,
   0.437131704762578,
   0.44073084741830826]]],
["vm2_str_tr_bang",
 [[2.6861952347680926,
   2.816385295242071,
   2.919709806330502,
   2.712329135276377,
   2.6914397440850735],
  [1.79120559617877,
   1.8011440439149737,
   1.7935738116502762,
   1.7961191991344094,
   1.7948967870324851]]],
["vm2_strcat",
 [[0.7171547841280699,
   0.7207952421158552,
   0.7209635498002172,
   0.7520132083445787,
   0.7179927034303546],
  [0.5424934774637222,
   0.5365820089355111,
   0.5409346511587501,
   0.5369464093819261,
   0.5396620389074087]]],
["vm2_super",
 [[0.43017072789371014,
```

```
    0.4671447016298771,
    0.43235956504940987,
    0.43128165043890476,
    0.4320727000012994],
   [0.42843823600560427,
    0.4359660716727376,
    0.46027328819036484,
    0.43320534005761147,
    0.42836809623986481]],
 ["vm2_unif1",
  [[0.22890623100101948,
    0.23932419251650572,
    0.2282293662428856,
    0.2281231889501214,
    0.22829711250960827],
   [0.23160281032323837,
    0.23210297524929047,
    0.23051423579454422,
    0.23075581435114145,
    0.23018061928451061]],
 ["vm2_zsuper",
  [[0.44561540707945824,
    0.4586751004680991,
    0.468279717490077,
    0.44244454242289066,
    0.4582256320863962],
   [0.5099742524325848,
    0.44541092310100794,
    0.4480971107259393,
    0.4709290647879243,
    0.56506655551493171]]]]

Elapsed time: 646.078768307 (sec)
------------------------------------------------------------
benchmark results:
minimum results in each 5 measurements.
Execution time (sec)
name    trunk   built
loop_whileloop2 0.091   0.091
vm2_array*  0.545   0.547
vm2_array_delete_lit*   0.342   0.155
vm2_array_include_lit*  0.328   0.138
vm2_bigarray*   5.802   5.791
vm2_bighash*    3.430   3.415
vm2_case*   0.071   0.069
vm2_defined_method* 2.336   2.305
vm2_dstr*   0.969   0.992
vm2_eval*   12.331  12.099
vm2_hash_aref_lit*  0.163   0.213
vm2_hash_aset_lit*  0.215   0.259
vm2_hash_delete_lit*    0.330   0.147
vm2_method* 1.122   1.111
vm2_method_missing* 1.626   1.609
vm2_method_with_block*  1.250   1.255
vm2_mutex*  0.554   0.581
vm2_newlambda*  0.665   0.658
vm2_poly_method*    1.892   1.943
vm2_poly_method_ov* 0.137   0.138
vm2_proc*   0.354   0.353
vm2_raise1* 4.664   4.701
vm2_raise2* 6.971   7.023
vm2_regexp* 0.982   0.933
vm2_send*   0.228   0.228
vm2_set_include_lit*    0.610   0.581
vm2_str_delete* 0.566   0.414
vm2_str_eq1*    0.330   0.145
vm2_str_eq2*    0.333   0.123
```

```
vm2_str_eqq1*   0.360   0.175
vm2_str_eqq2*   0.356   0.147
vm2_str_fmt*    2.412   1.899
vm2_str_gsub_bang_lit*  1.043   0.632
vm2_str_gsub_bang_re*   1.367   1.185
vm2_str_gsub_re*    1.599   1.441
vm2_str_plus1*  0.561   0.375
vm2_str_plus2*  0.558   0.345
vm2_str_tr_bang*    2.595   1.700
vm2_strcat* 0.626   0.445
vm2_super*  0.339   0.337
vm2_unif1*  0.137   0.139
vm2_zsuper* 0.351   0.354

---
 benchmark/bm_vm2_array_delete_lit.rb  |   6 +
 benchmark/bm_vm2_array_include_lit.rb |   6 +
 benchmark/bm_vm2_hash_aref_lit.rb     |   6 +
 benchmark/bm_vm2_hash_aset_lit.rb     |   6 +
 benchmark/bm_vm2_hash_delete_lit.rb   |   6 +
 benchmark/bm_vm2_set_include_lit.rb   |   7 +
 benchmark/bm_vm2_str_delete.rb        |   6 +
 benchmark/bm_vm2_str_eq1.rb           |   6 +
 benchmark/bm_vm2_str_eq2.rb           |   6 +
 benchmark/bm_vm2_str_eqq1.rb          |   6 +
 benchmark/bm_vm2_str_eqq2.rb          |   6 +
 benchmark/bm_vm2_str_fmt.rb           |   5 +
 benchmark/bm_vm2_str_gsub_bang_lit.rb |   6 +
 benchmark/bm_vm2_str_gsub_bang_re.rb  |   6 +
 benchmark/bm_vm2_str_gsub_re.rb       |   6 +
 benchmark/bm_vm2_str_plus1.rb         |   6 +
 benchmark/bm_vm2_str_plus2.rb         |   6 +
 benchmark/bm_vm2_str_tr_bang.rb       |   7 +
 benchmark/bm_vm2_strcat.rb            |   7 +
 common.mk                             |  18 +-
 compile.c                             | 330 ++++++++++++++++++++++++++++++++----
 defs/id.def                           |  38 ++++
 defs/opt_method.def                   |  90 ++++++++++
 insns.def                             | 275 +++++++++++++++++++++++++-------
 template/opt_method.h.tmpl            | 111 ++++++++++++
 template/opt_method.inc.tmpl          |  42 +++++
 test/-ext-/symbol/test_type.rb        |   1 +
 test/objspace/test_objspace.rb        |   1 +
 test/ruby/envutil.rb                  |  20 +++
 test/ruby/test_hash.rb                |   2 +
 test/ruby/test_iseq.rb                |   1 +
 test/ruby/test_optimization.rb        | 162 +++++++++++++++++-
 vm.c                                  |  67 ++++++--
 vm_core.h                             |  44 ++++-
 vm_insnhelper.c                       |   8 +-
 vm_insnhelper.h                       |  27 +++
 36 files changed, 1089 insertions(+), 264 deletions(-)
```

History

#1 - 10/24/2014 04:00 AM - ko1 (Koichi Sasada)

Thank you for your effort.

Could you show me an example of dis-assembled code for some cases? It will help us to understand your proposal.

#2 - 10/24/2014 08:00 AM - normalperson (Eric Wong)

Sure, I'll show each instruction with comments inline:

"%d" % 5

```
  [1,
```

```
  [:trace, 1],
  # 14 == OM_idMOD__String in generated opt_method.h
  # [:putstring, "%d"] was replaced with
  [:opt_str_lit_recv, ["%d", 14]],
  [:putobject, 5],
  [:opt_mod, {:mid=>:%, :flag=>256, :orig_argc=>1, :blockptr=>nil}],
  [:leave]]

  # I hope to optimize allocation on sprintf format strings too,
  # but that may be tricky...
```

str.tr("a", "A")

```
 [1,
  [:trace, 1],
  [:putself],
  [:opt_send_simple, {:mid=>:str, :flag=>280, :orig_argc=>0, :blockptr=>nil}],
  # both :opt_str_lit_tmask calls used to be :putstring
  # 55 == OM_idTr__String
  # 32 == OM_TMASK_String (1 << RUBY_T_STRING), also in opt_method.h
  [:opt_str_lit_tmask, ["a", 55, 32, 0]], # 0: offset to recv (str)
  [:opt_str_lit_tmask, ["A", 55, 32, 1]], # 1: offset to recv (str)
  [:opt_send_simple, {:mid=>:tr, :flag=>256, :orig_argc=>2, :blockptr=>nil}],
  [:leave]]]
```

Time.now.strftime("%Y")

```
 [1,
  [:trace, 1],
  [:getinlinecache, :label_9, 0],
  [:getconstant, :Time],
  [:setinlinecache, 0],
  :label_9,
  [:opt_send_simple, {:mid=>:now, :flag=>256, :orig_argc=>0, :blockptr=>nil}],
  # 63: OM_idStrftime__Time
  # for opt_str_lit_data, we must ensure class (Time) matches recv
  # because T_DATA may be almost anything
  [:opt_str_lit_data, ["%Y", 63, Time, 0]], # 0: offset to recv
  [:opt_send_simple,
   {:mid=>:strftime, :flag=>256, :orig_argc=>1, :blockptr=>nil}],
  [:leave]]]
```

In the future, I hope these optimizations can be applied to pure Ruby
classes and methods, as I mentioned in [ruby-core:65761]

I have been using the following insn-dump.rb script:

```
require 'pp'
pp RubyVM::InstructionSequence.compile(STDIN.read).to_a
```

#3 - 10/26/2014 01:32 AM - normalperson (Eric Wong)

It looks like Flonum was hiding my bug in on 64-bit systems :x
fix for 32-bit here:

```
--- a/compile.c
+++ b/compile.c
@@ -5467,7 +5467,7 @@ iseq_compile_each(rb_iseq_t *iseq, LINK_ANCHOR *ret, NODE * node, int poped)
        ADD_INSN1(ret, line, setn, INT2FIX(3));
        }
        flag = VM_CALL_ARGS_SKIP_SETUP;
-       ADD_SEND_R(ret, line, node->nd_mid, 2, 0, INT2FIX(flag));
+       ADD_SEND_R(ret, line, node->nd_mid, INT2FIX(2), 0, INT2FIX(flag));
        ADD_INSN(ret, line, pop);
        break;
     }
```

Unfortunately, I still get one failure due to strftime implementation
differences on 32 vs 64-bit:

```
  1) Failure:
TestRubyOptimization#test_time_opt_str_lit [/home/ew/ruby/test/ruby/test_optimization.rb:406]:
Time#strftime.
<88837> expected but was
```

```
<88832>.
```

I'll have to dig into it later (I'm disappointed that 64-bit needs more allocations, even).

#4 - 11/02/2014 10:07 PM - normalperson (Eric Wong)

- File opt_str_lit-v5.patch added

opt_str_lit-v5, changes since -v4:

- rebase ontop of r48239 ("Optimize keyword and splat argument")
  [Feature #10440]
- fix 32-bit compile [ruby-core:65902]
  and test for Time#strftime
- .gitignore: add /opt_method.h

#5 - 11/05/2014 11:14 AM - ko1 (Koichi Sasada)

Sorry for late, and thank you for catching up recent changes.

My comments:

1. (negative) Incompatibility

I find incompatibility:

```
def bar
  String.class_eval %q{
    def ==(other)
      self.upcase! # modify operation
      true
    end
  }
  'bar'
end

p('foo' == bar)

#=> (eval):3:in `upcase!': can't modify frozen String (RuntimeError)
#        from (eval):3:in `=='
#        from ../../gitruby/test.rb:11:in `<main>'
```

To solve this incompatibility, I have several ideas.

(1-1) Make new instruction to replace with receiver.

like that:

```
# 'foo' == 'bar'
putobject 'foo' # frozen
putobject 'bar' # frozen
opt_stringliteral data # data is described bellow
opt_eq
```

`data' includes (at least) three data, (mid, receiver location and usage bitmap).

For this time,
mid is :==,
receiver location is 1 (0 is top), and
usage bitmap is 3 (0x01 | 0x02)

```
# pseudo code

ressurrect_all(bits)
{
    i = 0;
    while (bits) {
    if (bits & 0x01) TOPN(i) = dup(TOPN(i));
    bits >> 1; i++;
    }
```

```
}

opt_stringliteral
(data)
()
()
{
    if (data->last_status == unkown) {
    recv = TOPN(data->recv_loc);
    klass = CLASS_OF(recv);

    if (klass's data->mid method is our target) {
        /* do nothing */
    }
    else {
        ressurrect_all(data->bitmap);
        last_status = failed;
    }
    }
    else {
    ressurrect_all(data->bitmap);
    }
}
```

(added 4th field to represent last try).

(1-2) Make new version of send' instruction which includesopt_stringliteral'.

(1-3) Ignore such incompatibility because nobody write such code

1. (negative) Live patching

(I need to announce it, sorry)

I don't introduce live patching codes because of the following reasons.

(2-1) It is difficult to translate C codes.

C codes means two meaning.
(2-1-1) Compiling bytecode data
(2-1-2) Compiling equivalent but fast C code (AOT/JIT compilation) (not limited to C code)

For (2-1-2), it is no problem if we disable such technique.

However, for (2-1-1), we want to expect the bytecode data on TEXT section (read only, easy for CoW).

This reason can be too fast optimization because there are no wide-spreadded used implementation. However, I want to make such implementation easier (in fact, I want to make both implementations, however there are stuck now).

(2-2) Copy on Write unfrindly

We can expect bytecode data is immutable, CoW frindly data. However, live patching prevents this optimization.

1. (neutral/not big issue) target methods

opt_method.def format is a bit difficult to read which methods are target. Maybe we can make more pretty like:

```
  Fixnum#+
  Array#[]
  Array#[]=
```

and translate to C code.

1. (neutral/not big issue) warning on VC (windows)

There are several warnings on VC:

```
c:\ko1\src\ruby\mswin64.git\opt_method.h(216) : warning C4146: □□□□□□□□□□
c:\ko1\src\ruby\mswin64.git\opt_method.h(221) : warning C4146: □□□□□□□□□□
c:\ko1\src\ruby\mswin64.git\opt_method.h(226) : warning C4146: □□□□□□□□□□
```

It says that "assigning to signed value".

Have you measure performance improvement on non-microbenchmarks?

#6 - 11/06/2014 12:31 AM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> Sorry for late, and thank you for catching up recent changes.

No problem. I had some time today. I will be busy and have little
computer time for the next 5-7 days.

> My comments:
>
> 1. (negative) Incompatibility
>
> To solve this incompatibility, I have several ideas.
>
> (1-1) Make new instruction to replace with receiver.

I added a new opt_str_lit_yoda instruction. It only handles one receiver
location for now, I'm not sure if other cases (e.g. gsub) can trigger
the incompatibility.

patch: http://80x24.org/spew/m/38d41def78f16bca14fbc7fccd160ef0a8d06479.txt

Right now, I do not optimize str.gsub("bar", method_call)', yet,
only optimizing:str.gsub(anything, "bar")'

```
"foo" == bar
```

Compiles to:

[:putobject, "foo"],
[:putself],
[:opt_send_without_block,
{:mid=>:bar, :flag=>280, :orig_argc=>0, :blockptr=>nil}],
[:opt_str_lit_yoda, ["foo", 17, 1]],
[:opt_eq, {:mid=>:==, :flag=>256, :orig_argc=>1, :blockptr=>nil}],

> (1-3) Ignore such incompatibility because nobody write such code

However, I hope we can ignore the strange case and new insn.

> 1. (negative) Live patching

I've removed live patching for now:
http://80x24.org/spew/m/86739be08df5b4128e84bce15a2853cd7947a6fa.txt
It was a good learning experience, at least.

> 1. (neutral/not big issue) target methods
>
> opt_method.def format is a bit difficult to read which methods are
> target. Maybe we can make more pretty like:

I prefer to leave the format unchanged for now, because compile.c still
uses id{PLUS,ASET,AREF} names.

> 1. (neutral/not big issue) warning on VC (windows)

Untested patch (cast to unsigned):
http://80x24.org/spew/m/30c309d1ad24319ad53813f8814e09be0554ccb0.txt

> Have you measure performance improvement on non-microbenchmarks?

"make rdoc" performance improved from 66.3 => 63.3 sec in original test

However, the difference seems is smaller now: 66.1 => 64.9 sec

Full patch for convenience:
http://80x24.org/spew/m/opt-str-lit-v7%40m.txt

#7 - 11/06/2014 08:05 AM - ko1 (Koichi Sasada)

Eric Wong wrote:

> My comments:
>
> 1. (negative) Incompatibility
>
> To solve this incompatibility, I have several ideas.
>
> (1-1) Make new instruction to replace with receiver.

I added a new opt_str_lit_yoda instruction. It only handles one receiver
location for now, I'm not sure if other cases (e.g. gsub) can trigger
the incompatibility.

patch: http://80x24.org/spew/m/38d41def78f16bca14fbc7fccd160ef0a8d06479.txt

Right now, I do not optimize str.gsub("bar", method_call)', yet,
only optimizing:str.gsub(anything, "bar")'

"foo" == bar

Compiles to:

[:putobject, "foo"],
[:putself],
[:opt_send_without_block,
{:mid=>:bar, :flag=>280, :orig_argc=>0, :blockptr=>nil}],
[:opt_str_lit_yoda, ["foo", 17, 1]],
[:opt_eq, {:mid=>:==, :flag=>256, :orig_argc=>1, :blockptr=>nil}],

I raed the implementation of "opt_str_lit_yoda". Is this a not bitmap version (single string) of "opt_stringliteral"?
You don't use bitmap, because of implementation reason?

BTW, why the name "yoda"?

> (1-3) Ignore such incompatibility because nobody write such code

However, I hope we can ignore the strange case and new insn.

Me too.
(but also I think such restriction is challenging)

> 1. (negative) Live patching

I've removed live patching for now:
http://80x24.org/spew/m/86739be08df5b4128e84bce15a2853cd7947a6fa.txt
It was a good learning experience, at least.

Thank you.

Please do not think that I stick to avoid live patching.
If we can have huge performance improvement, we need to consider about it.

> 1. (neutral/not big issue) target methods
>
> opt_method.def format is a bit difficult to read which methods are
> target. Maybe we can make more pretty like:

I prefer to leave the format unchanged for now, because compile.c still
uses id{PLUS,ASET,AREF} names.

Yes, because it is C source code.
In the definition code, we don't need to continue such unreadable format.

But pretty printing is not big issue, we can change any time.

1. (neutral/not big issue) warning on VC (windows)

Untested patch (cast to unsigned):
http://80x24.org/spew/m/30c309d1ad24319ad53813f8814e09be0554ccb0.txt

Not solved yet :(

```
C:\ko1\src\ruby\mswin64.git\opt_method.h(216) : warning C4146: 符号付きの値を代入する変数は、符号付き型にキャストしなければなりません。
C:\ko1\src\ruby\mswin64.git\opt_method.h(221) : warning C4146: 符号付きの値を代入する変数は、符号付き型にキャストしなければなりません。
C:\ko1\src\ruby\mswin64.git\opt_method.h(226) : warning C4146: 符号付きの値を代入する変数は、符号付き型にキャストしなければなりません。
```

Maybe it says "a variable which assign signed value, you need to cast to signed value."

Have you measure performance improvement on non-microbenchmarks?

"make rdoc" performance improved from 66.3 => 63.3 sec in original test

However, the difference seems is smaller now: 66.1 => 64.9 sec

What does it means? what is "now"?

#8 - 11/07/2014 09:37 AM - ko1 (Koichi Sasada)

I tested your patch (v5).

I can measure the microbenchmark improvements (compare with current trunk).

```
name    ruby 2.2.0dev (2014-11-06 trunk 48295) [x86_64-linux]   built-ruby
loop_whileloop2 0.128   0.111
vm2_str_delete* 0.752   0.549
vm2_str_eq1*    0.421   0.225
vm2_str_eq2*    0.418   0.156
vm2_str_eqq1*   0.461   0.233
vm2_str_eqq2*   0.454   0.201
vm2_str_fmt*    3.075   2.334
vm2_str_gsub_bang_lit*  1.453   1.084
vm2_str_gsub_bang_re*   1.743   1.532
vm2_str_gsub_re*        2.054   1.876
vm2_str_plus1*  0.750   0.536
vm2_str_plus2*  0.785   0.475
vm2_str_tr_bang*        3.652   2.711
vm2_strcat*     0.839   0.624

Speedup ratio: compare with the result of `ruby 2.2.0dev (2014-11-06 trunk 48295) [x86_64-linux]' (greater is
better)
name    built-ruby
loop_whileloop2 1.152
vm2_str_delete* 1.369
vm2_str_eq1*    1.873
vm2_str_eq2*    2.688
vm2_str_eqq1*   1.982
vm2_str_eqq2*   2.260
vm2_str_fmt*    1.318
vm2_str_gsub_bang_lit*  1.340
vm2_str_gsub_bang_re*   1.138
vm2_str_gsub_re*        1.095
vm2_str_plus1*  1.398
vm2_str_plus2*  1.651
vm2_str_tr_bang*        1.347
vm2_strcat*     1.344
```

However, I can't see big performance improvement on the following something like big benchmarks.

```
# current trunk
$ make gcbench-rdoc
      user     system      total        real
102.080000   0.910000 102.990000 (103.618734)
GC total time (sec): 4.637481483002263


# with patch
```

```
$ make gcbench-rdoc
...
      user     system      total        real
104.770000   1.000000 105.770000 (106.152441)
GC total time (sec): 4.424793113999028
```

And discourse bench (thansk @samsaffron !)

```
current trunk:
---
categories_admin:
  50: 112
  75: 116
  90: 121
  99: 137
home_admin:
  50: 92
  75: 96
  90: 99
  99: 108
topic_admin:
  50: 49
  75: 51
  90: 53
  99: 59
categories:
  50: 89
  75: 91
  90: 98
  99: 107
home:
  50: 62
  75: 64
  90: 68
  99: 74
topic:
  50: 21
  75: 22
  90: 23
  99: 30
timings:
  load_rails: 3355
ruby-version: 2.2.0-p-1
rss_kb: 211068
pss_kb: 206877
virtual: physical
architecture: amd64
operatingsystem: Ubuntu
physicalprocessorcount: 1
kernelversion: 3.13.0
processor0: Intel(R) Core(TM) i7-4930K CPU @ 3.40GHz
memorysize: 15.60 GB

modified (v5)
---
categories_admin:
  50: 112
  75: 116
  90: 121
  99: 140
home_admin:
  50: 92
  75: 95
  90: 99
  99: 107
topic_admin:
  50: 49
  75: 51
  90: 54
  99: 59
categories:
  50: 89
  75: 91
  90: 99
  99: 106
```

```
home:
  50: 62
  75: 63
  90: 68
  99: 74
topic:
  50: 22
  75: 22
  90: 23
  99: 30
timings:
  load_rails: 3368
ruby-version: 2.2.0-p-1
rss_kb: 210184
pss_kb: 205978
virtual: physical
architecture: amd64
operatingsystem: Ubuntu
physicalprocessorcount: 1
kernelversion: 3.13.0
processor0: Intel(R) Core(TM) i7-4930K CPU @ 3.40GHz
memorysize: 15.60 GB

categories_admin:
  50: 112
  75: 116
  90: 121
  99: 137
```

means that accessing "categories_admin" many times (5000 accesses, in this time),
and 50% of accesses are lower than 112 ms, 75% of accesses are lower than 116ms and so on.

Anyone else can test it on your application?

#9 - 11/11/2014 08:58 AM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> Eric Wong wrote:
>
> > My comments:
> >
> > 1. (negative) Incompatibility
> >
> > To solve this incompatibility, I have several ideas.
> >
> > (1-1) Make new instruction to replace with receiver.
>
> I added a new opt_str_lit_yoda instruction. It only handles one receiver
> location for now, I'm not sure if other cases (e.g. gsub) can trigger
> the incompatibility.
>
> patch: http://80x24.org/spew/m/38d41def78f16bca14fbc7fccd160ef0a8d06479.txt
>
> Right now, I do not optimize str.gsub("bar", method_call)', yet,
> only optimizing:str.gsub(anything, "bar")'
>
> "foo" == bar
>
> Compiles to:
>
> [:putobject, "foo"],
> [:putself],
> [:opt_send_without_block,
> {:mid=>:bar, :flag=>280, :orig_argc=>0, :blockptr=>nil}],
> [:opt_str_lit_yoda, ["foo", 17, 1]],
> [:opt_eq, {:mid=>:==, :flag=>256, :orig_argc=>1, :blockptr=>nil}],

I raed the implementation of "opt_str_lit_yoda". Is this a not bitmap version (single string) of "opt_stringliteral"?
You don't use bitmap, because of implementation reason?

Right, it only affected methods with one string literal arg.

It was a little easier to implement without bitmap.

> BTW, why the name "yoda"?

Named after the syntax of the Star Wars character:
https://en.wikipedia.org/wiki/Yoda_Conditions

> > (1-3) Ignore such incompatibility because nobody write such code
>
> However, I hope we can ignore the strange case and new insn.

> Me too.
> (but also I think such restriction is challenging)
>
> > 1. (negative) Live patching
>
> I've removed live patching for now:
> http://80x24.org/spew/m/86739be08df5b4128e84bce15a2853cd7947a6fa.txt
> It was a good learning experience, at least.

> Thank you.
>
> Please do not think that I stick to avoid live patching.
> If we can have huge performance improvement, we need to consider about it.

OK, we can also make a compile-time option.

> > Have you measure performance improvement on non-microbenchmarks?
>
> "make rdoc" performance improved from 66.3 => 63.3 sec in original test
>
> However, the difference seems is smaller now: 66.1 => 64.9 sec

> What does it means? what is "now"?

I was not able to replicate the original results, probably because other
changes in trunk affected performance.

I'll check other issues tomorrow.

About the discourse benchmark: it seems discourse and libraries already
use .freeze frequently; so the performance may not be noticeable in
application which is already optimized by Ruby experts.

In our stdlib, I hope to make parts of r47073, r47072 unnecessary
and revert the "literal".freeze calls.

For instance, rack.git has commit dc53a8c26dc55d21240233b3d83d36efdef6e924
("Less allocated objects on each request"), which is ugly, but
apparently important for performance.

My goal is to make optimization transparent to non-experts
(and prettier code for all)

#10 - 11/12/2014 10:08 PM - normalperson (Eric Wong)

ko1@atdot.net wrote:

> > 1. (neutral/not big issue) warning on VC (windows)
>
> Untested patch (cast to unsigned):
> http://80x24.org/spew/m/30c309d1ad24319ad53813f8814e09be0554ccb0.txt

> Not solved yet :(

```
C:\ko1\src\ruby\mswin64.git\opt_method.h(216) : warning C4146: 符号付きの値を代入する変数は、符号付き型にキャストしなければなりません。
C:\ko1\src\ruby\mswin64.git\opt_method.h(221) : warning C4146: 符号付きの値を代入する変数は、符号付き型にキャストしなければなりません。
C:\ko1\src\ruby\mswin64.git\opt_method.h(226) : warning C4146: 符号付きの値を代入する変数は、符号付き型にキャストしなければなりません。
```

> Maybe it says "a variable which assign signed value, you need to cast to signed value."

Maybe this works:

```
http://80x24.org/spew/m/opt_str_lit+signed_enum@7.txt
```

But I am thinking of trying an entirely different approach to this optimization (also able to target non-cfunc)

#11 - 11/13/2014 04:28 AM - normalperson (Eric Wong)

normalperson@yhbt.net wrote:

> But I am thinking of trying an entirely different approach to this optimization (also able to target non-cfunc)

I had an idea to do lazy/late rb_str_resurrect before ci->call by defining + setting a STR_LITERAL flag (FL_USER19), but that is also complicated and I haven't been able to implement it, yet.

The current patches can only handle cfunc on core classes, and even handling of Time (T_DATA) is ugly. I'm afraid this type of transparent optimization is not effective for real-world performance unless it can be applied more broadly to non-core classes.

#12 - 11/15/2014 10:48 PM - normalperson (Eric Wong)

2 patch series:

1. http://80x24.org/spew/m/opt_str_lit-v8-prepare@0.txt
   (allow disabling existing optimizations)

2. http://80x24.org/spew/m/opt_str_lit-v8@1.txt
   (enable new optimizations)

Changes since -v7:

- opt_str_lit*: avoid string literal allocations
- rb_undo_opt_str_lit is now a VM option
- template/opt_method.h.tmpl: explicitly cast to signed
- remove Time#strftime and opt_str_lit_data insns (too specialized)
- keep opt_aref_with and opt_aset_with (for speed)

Rebased on top of r48453

I decided to keep opt_{aref,aset}_with despite being less elegant to avoid performance regressions from trunk (I removed opt_str_freeze, however).

I also reinstated bytecode patching to disable optimization as a VM option since we do not have CoW-friendly bytecode or JIT, yet.

Also pushed to my git repo:

git://80x24.org/ruby.git opt_str_lit-v8

Performance difference is still minor, "make gcbench-rdoc" results:

before:
user   system   total   real
60.630000   0.720000 61.350000 (61.424623)
GC total time (sec): 4.268761704000086

VmHWM: 290040 kB

Summary of rdoc on 2.2.0dev   61.42462271545082   4.268761704000086   145
(real time in sec, GC time in sec, GC count)

after:
user system total real
58.830000 0.700000 59.530000 (59.601977)
GC total time (sec): 4.048253329998705

VmHWM: 289536 kB

Summary of rdoc on 2.2.0dev 59.60197714343667 4.048253329998705 131
(real time in sec, GC time in sec, GC count)

Eventually I'd like this to work more generically (for pure-Ruby classes)
and perform lazy dup at method dispatch, but that is a lot of work.

#13 - 11/16/2014 09:28 AM - normalperson (Eric Wong)

Updated 2/2 in patch series, 1/2 unchanged:

1. http://80x24.org/spew/m/opt_str_lit-v8-prepare@0.txt
   (allow disabling existing optimizations) [unchanged]

2. http://80x24.org/spew/m/opt_str_lit-v9@1.txt
   Changes since -v8:

   - rebased on r48458 and updated deps

#14 - 01/05/2018 09:01 PM - naruse (Yui NARUSE)

- Target version deleted (2.2.0)

Files

opt_str_lit-v4.patch	76.7 KB	10/24/2014	normalperson (Eric Wong)
opt_str_lit-v5.patch	78.2 KB	11/02/2014	normalperson (Eric Wong)