Table of Contents

Experimenting with Huge Pages

23/03/2017

Building biglibs with huge pages in CMSSW

failed!

Just adding and rebuilding the biglib

```
[innocent@vinavx3 CMSSW_9_1_0_pre1]$ diff config/BuildFile.xml $CMSSW_RELEASE_BASE/config/BuildFi
40c40
<   <flags BIGOBJ_CXXFLAGS="-Wl,--exclude-libs,ALL -B /usr/share/libhugetlbfs  -Wl,--hugetlbfs-al
---
>   <flags BIGOBJ_CXXFLAGS="-Wl,--exclude-libs,ALL"/>
```

at run time got

```
libhugetlbfs: WARNING: Layout problem with segments 0 and 1:
   Segments would overlap
```

and no evidence of use of huge pages...

16/03/2017

HugePages strikes again

Following a very interesting seminar by Marco Guerri I investigated again HugePages for code and discovered that life is now a bit easier with new kernels and distributions... see https://github.com/libhugetlbfs/libhugetlbfs/blob/master/HOWTO

- *libhugetlbfs* is now in the distributions: CC7 for instance (under /usr/share/libhugetlbfs)
- it is easer to reserve huge pages
- one still has to mount hugetlbfs

configuration as root becomes

```
echo 1024 > /proc/sys/vm/nr_overcommit_hugepages
mkdir -p /mnt/hugetlbfs
mount -t hugetlbfs none /mnt/hugetlbfs
chmod 777 /mnt/hugetlbfs/
```

linking with libhugetlbfs loader

```
c++ -Wall -B /usr/share/libhugetlbfs  -Wl,--hugetlbfs-align ...
```

at runtime one needs

```
setenv HUGETLB_ELFMAP RW
```

I have used Marco's benchmark https://gitlab.cern.ch/snippets/216 to test it on my workstation

```
icache.py > icache.cpp
c++ -O0 -Wall icache.cpp -o icacheStandard
c++ -O0 -Wall icache.cpp -B /usr/share/libhugetlbfs -o icacheHuge -Wl,--hugetlbfs-align
```

```
setenv HUGETLB_ELFMAP RW
```

and observed

```
perf stat -e cycles -e iTLB-load-misses -e iTLB-loads ./icacheStandard

 Performance counter stats for './icacheStandard':

       73122850326      cycles
           1041337      iTLB-load-misses          #    0.17% of all iTLB cache hits
         599872659      iTLB-loads

      18.719051666 seconds time elapsed

perf stat -e cycles -e iTLB-load-misses -e iTLB-loads ./icacheHuge

 Performance counter stats for './icacheHuge':

       63403839270      cycles
            755673      iTLB-load-misses          # 50479.16% of all iTLB cache hits
              1497      iTLB-loads

      16.218030985 seconds time elapsed
```

(caveat iTLB-load-misses may be misleading: it actually counts the "itlb_misses.miss_causes_a_walk" so the sTLB-misses, while iTLB-loads counts the misses that actually hit the sTLB)

while **icacheHuge** was running I looked into *meminfo* and actually saw the huge pages in use

```
 cat /proc/meminfo | grep Huge
AnonHugePages:    104448 kB
HugePages_Total:       5
HugePages_Free:        1
HugePages_Rsvd:        1
HugePages_Surp:        5
Hugepagesize:       2048 kB
```

effect on Geant4

I have measured the number of iTLB-loads in CMS simulation and found to be 0.1% of the cycles Assuming (from the results about) that each iTLB-load cost about 15 cycles we can predict a speed gain of about 1.5% in using huge pages to allocate text-segments

how to verify the status of THP

```
egrep 'trans|thp' /proc/vmstat
```

06/04/2010

HugePages reloaded

Following the publication of a set of new article on lwn http://lwn.net/Articles/374424/ (part 1 and link to others) http://lwn.net/Articles/379748/ (part5) I tested again the use of hugepages in cmssw using pfmon to measure the amonut of DLTB misses. I just allocate the whole HEAP on hugepages thanks to the new library described in the articles above

results for single and mutiple (5) processes

- no hugepages

```
pfmon command line: pfmon --with-header --aggregate-results --follow-all -e UNHALTED_CORE_CYCLES,
cmsRun recoqcd_RAW2DIGI_RECO_MC_one.py
1646660047412 UNHALTED_CORE_CYCLES
      1979891584 DTLB_MISSES:ANY
621.742u 3.730s 10:38.23 98.0%   0+0k 0+0io 22512pf+0w

pfmon command line: pfmon --with-header --aggregate-results --follow-all -e UNHALTED_CORE_CYCLES,
cmsRun recoqcd_RAW2DIGI_RECO_MC_multi.py

1740125704760 UNHALTED_CORE_CYCLES
      2042917165 DTLB_MISSES:ANY
654.580u 7.739s 3:18.09 334.3%   0+0k 0+0io 24619pf+0w
```

- hugepages

```
pfmon command line: pfmon --with-header --aggregate-results --follow-all -e UNHALTED_CORE_CYCLES,
/users/innocent/usrlocal/bin/hugectl --library-use-path --heap cmsRun recoqcd_RAW2DIGI_RECO_MC_on
1628420054827 UNHALTED_CORE_CYCLES
   1522338644 DTLB_MISSES:ANY
615.098u 3.509s 10:30.17 98.1%   0+0k 0+0io 23553pf+0w
pfmon command line: pfmon --with-header --aggregate-results --follow-all -e UNHALTED_CORE_CYCLES,
/users/innocent//usrlocal/bin/hugectl --library-use-path --heap cmsRun recoqcd_RAW2DIGI_RECO_MC_m
1720013658083 UNHALTED_CORE_CYCLES
      1342258581 DTLB_MISSES:ANY
647.690u 7.351s 3:11.93 341.2%   0+0k 0+0io 24344pf+0w
```

the observation is that **DTLB_MISSES** account to a bit more of 1% of the cycles (each DTLB miss costs about 10 cycles). Using hugepages for the heap reduces the **DTLB_MISSES** by more than 30%, still negligible for what CMSSW is concerned.

one shall also note that hugepages are **NOT** accounted in the standard memory reports

```
10823 innocent pfmon --with-header --aggre        0   772.0K   805.0K     1.3M
 9164 innocent -tcsh                              0     1.7M     1.9M     2.7M
 8953 innocent -tcsh                              0     2.8M     3.0M     3.8M
10974 innocent /usr/bin/python /afs/cern.c        0     7.2M     7.4M     8.3M
10824 innocent cmsRun recoqcd_RAW2DIGI_REC        0   286.8M   287.0M   288.1M
--------------------------------------------------------------------------------
   64 1                                           0   299.1M   300.1M   304.2M

[pcphsft50] ~ $ ~/w1/smem-0.9/smem -t -k
  PID User     Command                         Swap      USS      PSS      RSS
10977 innocent pfmon --with-header --aggre        0     1.1M     1.1M     1.7M
 9164 innocent -tcsh                              0     1.7M     1.9M     2.7M
 8953 innocent -tcsh                              0     2.8M     3.0M     3.8M
10991 innocent /usr/bin/python /afs/cern.c        0     7.5M     7.7M     8.6M
10978 innocent cmsRun recoqcd_RAW2DIGI_REC        0     4.3M    40.8M   226.8M
10984 innocent cmsRun recoqcd_RAW2DIGI_REC        0    63.0M   101.9M   296.8M
10981 innocent cmsRun recoqcd_RAW2DIGI_REC        0    63.2M   101.9M   296.7M
10985 innocent cmsRun recoqcd_RAW2DIGI_REC        0    63.2M   101.9M   296.7M
10983 innocent cmsRun recoqcd_RAW2DIGI_REC        0    63.2M   102.0M   297.0M
10982 innocent cmsRun recoqcd_RAW2DIGI_REC        0    63.2M   102.1M   297.1M
--------------------------------------------------------------------------------
   69 1                                           0   333.1M   564.2M     1.7G
%MSG-w MemoryCheck:  PostProcessPath 03-Apr-2010 17:30:55 CEST Run: 1 Event: 60
MemoryCheck: event : VSIZE 1004.96 0 RSS 226.828 0 HEAP-ARENA [ SIZE-BYTES 980017152 N-UNUSED-CHU
```

23/07/2008

new performance test using 219

I run reco 219 on 200 events bbar as produced by RelVal.

standard

```
1732729539271 UNHALTED_CORE_CYCLES
1539713454028 INSTRUCTIONS_RETIRED
 230642073370 BRANCH_INSTRUCTIONS_RETIRED
   5670263475 MISPREDICTED_BRANCH_RETIRED
 603715307797 INST_RETIRED:LOADS
            0 SIMD_COMP_INST_RETIRED:PACKED_SINGLE:SCALAR_SINGLE:PACKED_DOUBLE:SCALAR_DOUBLE
 316752798881 INST_RETIRED:STORES
 144606669202 X87_OPS_RETIRED:ANY
 629460043234 RESOURCE_STALLS:ANY
   2522977571 BUS_TRANS_ANY:ALL_AGENTS
   2737708714 BUS_DRDY_CLOCKS:ALL_AGENTS
     69208177 BUS_BNR_DRV:ALL_AGENTS
  36272303280 LAST_LEVEL_CACHE_REFERENCES
    427617760 LAST_LEVEL_CACHE_MISSES
 217670175552 CPU_CLK_UNHALTED:BUS
--------------------------------------------------------------------------
Ratios:
                           CPI: 1.1254
            load instructions %: 39.210%
           store instructions %: 20.572%
  load and store instructions %: 59.782%
  resource stalls % (of cycles): 36.328%
         branch instructions %: 14.980%
% of branch instr. mispredicted: 2.458%
           % of l2 loads missed: 1.179%
              bus utilization %: 2.318%
         data bus utilization %: 1.258%
                bus not ready %: 0.064%
  comp. SIMD inst. ('new FP') %: 0.000%
  comp. x87 instr. ('old FP') %: 9.392%
```

with TCMALLOC

```
1584570646894 UNHALTED_CORE_CYCLES
1509316585174 INSTRUCTIONS_RETIRED
 224658865844 BRANCH_INSTRUCTIONS_RETIRED
   5025722897 MISPREDICTED_BRANCH_RETIRED
 597442500267 INST_RETIRED:LOADS
            0 SIMD_COMP_INST_RETIRED:PACKED_SINGLE:SCALAR_SINGLE:PACKED_DOUBLE:SCALAR_DOUBLE
 306462564773 INST_RETIRED:STORES
 144577985459 X87_OPS_RETIRED:ANY
 527521246432 RESOURCE_STALLS:ANY
   1961824853 BUS_TRANS_ANY:ALL_AGENTS
   2126966626 BUS_DRDY_CLOCKS:ALL_AGENTS
     67851327 BUS_BNR_DRV:ALL_AGENTS
  33442404368 LAST_LEVEL_CACHE_REFERENCES
    355141679 LAST_LEVEL_CACHE_MISSES
 199109646642 CPU_CLK_UNHALTED:BUS
--------------------------------------------------------------------------
Ratios:
                           CPI: 1.0499
            load instructions %: 39.584%
           store instructions %: 20.305%
  load and store instructions %: 59.888%
  resource stalls % (of cycles): 33.291%
         branch instructions %: 14.885%
% of branch instr. mispredicted: 2.237%
           % of l2 loads missed: 1.062%
              bus utilization %: 1.971%
```

```
        data bus utilization %: 1.068%
               bus not ready %: 0.068%
  comp. SIMD inst. ('new FP') %: 0.000%
  comp. x87 instr. ('old FP') %: 9.579%
```

23/07/2008

new performance test using 210(pre9)

I run reco 210_pre9 on 200 events bbar as produced by RelVal. I run the standard code, that applied the patch to `ROOT::Cintex::Allocate_code` to seve pages out of 2MB block pool then the same using tcmalloc and finally using hugepages

```
cat /proc/meminfo | grep -i huge
HugePages_Total:  1000
HugePages_Free:    678
HugePages_Rsvd:      1
HugePages_Surp:      0
Hugepagesize:     2048 kB
```

I even run 8 `cmsRun` at the same time.

the use of *hugepages* does not have any effect on performance. reco scales up to 8 cpus (on lxbuild114) w/o problems.

22/07/2008

running with 210

```
cmsDriver.py BJets_Pt_50_120_cfi -s GEN -n 100
cmsDriver.py BJets_Pt_50_120_cfi -s SIM -n 100 --filein file:BJets_Pt_50_120_cfi_GEN.root
```

did not manage to get `--customize` to work copied Gabriele file locally run `cmsDriver` with `--no_exec` and used pico to mix the two...

at the end I used Sharham instruction.... and copied a sim-file from dbs

```
rfcp /castor/cern.ch/cms/store/relval/2008/7/20/RelVal-RelValBJets_Pt_50_120-1216579576/RelValBJe

cmsDriver.py   BJets_Pt_50_120_cfi -s GEN,SIM,DIGI,L1,DIGI2RAW -n 100 --conditions FrontierCondit
cp  $CMSSW_RELEASE_BASE/src/Configuration/Examples/python/RecoExample_cfg.py .
cmsRun RecoExample_cfg.py
```

16/07/2008

testing memory churn in multi-thread

six threads on lxbuild114 (8 core)

method	Real time	cpu/real		
	malloc		tcmalloc	
naive	81	534%	300	138%
boost-pool	35	462%	36	504%
large chunks	34	477%	36	506%

15/07/2008

summary of measurement of cms reco

time per event	malloc	tcmalloc
reco modules	2.8	2.5
output module	0.25	0.24

counter	malloc	tcmalloc
UNHALTED_CORE_CYCLES	970293569059	903705695000
INSTRUCTIONS_RETIRED	943404623915	931115219416
BRANCH_INSTRUCTIONS_RETIRED	153400155587	151185416169
MISPREDICTED_BRANCH_RETIRED	4283583203	4032950096
INST_RETIRED:LOADS	366093204853	363136780290
SIMD_COMP_INST_RETIRED	0	0
INST_RETIRED:STORES	198866522599	194064751706
X87_OPS_RETIRED:ANY	57650513756	57567303752
RESOURCE_STALLS:ANY	305133549321	259105502466
BUS_TRANS_ANY:ALL_AGENTS	1103247481	965820152
BUS_DRDY_CLOCKS:ALL_AGENTS	1164574768	1004396040
BUS_BNR_DRV:ALL_AGENTS	365249031	346899990
LAST_LEVEL_CACHE_REFERENCES	21200587333	20041091763
LAST_LEVEL_CACHE_MISSES	187148195	168698808
CPU_CLK_UNHALTED:BUS	138684654138	129149087560

ratio	malloc	tcmalloc
CPI	1.0285	0.9706
load instructions %	38.806%	39.000%
store instructions %	21.080%	20.842%
load and store instructions %	59.885%	59.842%
resource stalls % (of cycles)	31.448%	28.671%
branch instructions %	16.260%	16.237%
% of branch instr. mispredicted	2.792%	2.668%
% of l2 loads missed	0.883%	0.842%
bus utilization %	1.591%	1.496%
data bus utilization %	0.840%	0.778%
bus not ready %	0.527%	0.537%
comp. SIMD inst. ('new FP') %	0.000%	0.000%
comp. x87 instr. ('old FP') %	6.111%	6.183%

07/07/2008

allocating huge pages on 32 bit

tcmalloc fails this message:

```
Check failed: (fstatfs(hugetlb_fd, &sfs)) != -1: Bad address
```

06/07/2008 (sunday)

it's raining!

allocating huge pages

fasten your seatbelt

following Giulio's receipt I tried to convince tcmalloc to use hugetlbpages

I used lxbuild066 (standard slc4) and pcphsft50 (new kernel)

setting up

```
sudo /bin/tcsh -f -c "echo 1000 > /proc/sys/vm/nr_hugepages"
sudo /bin/tcsh -f -c "mkdir -p /mnt/hugepage;mount -t hugetlbfs -o uid=1039,gid=1399,mode=0775 no
setenv TCMALLOC_MEMFS_MALLOC_PATH /mnt/hugepage/
```

caveat lxbuild machines have little contigous memory to allocate hugetlb so one gets on 12 pages (even only 6 on lxbuild065)

```
cat /proc/meminfo | grep -i huge
HugePages_Total:    12
HugePages_Free:     12
Hugepagesize:     2048 kB
```

this is enough for a small test with scimark2:

```
unsetenv TCMALLOC_MEMFS_MALLOC_PATH
[lxbuild066] ~/public/multicore > ~/w1/scimark2/scimark2 -large > scimark2_tc&
[1] 2257
[lxbuild066] ~/public/multicore > cat /proc/meminfo | grep -i huge
HugePages_Total:    12
HugePages_Free:     12
Hugepagesize:     2048 kB
[lxbuild066] ~/public/multicore > cat /proc/meminfo | grep -i huge
HugePages_Total:    12
HugePages_Free:     12
Hugepagesize:     2048 kB
[lxbuild066] ~/public/multicore >
[1]    Done                          ~/w1/scimark2/scimark2 -large > scimark2_tc
[lxbuild066] ~/public/multicore > cat scimark2_tc
**                                                              **
** SciMark2 Numeric Benchmark, see http://math.nist.gov/scimark **
** for details. (Results can be submitted to pozo@nist.gov)     **
**                                                              **
Using       2.00 seconds min time per kenel.
Composite Score:          473.27
FFT             Mflops:    55.27    (N=1048576)
SOR             Mflops:   846.92    (1000 x 1000)
MonteCarlo:     Mflops:   191.06
Sparse matmult  Mflops:   455.11    (N=100000, nz=1000000)
LU              Mflops:   818.00    (M=1000, N=1000)
-----------------------------------------------------
-----------------------------------------------------

setenv TCMALLOC_MEMFS_MALLOC_PATH /mnt/hugepage/testTC
[lxbuild066] ~/public/multicore > ~/w1/scimark2/scimark2 -large > scimark2_tc_hugetlb &
[1] 3585
at /proc/meminfo | grep -i huge
HugePages_Total:    12
```

```
HugePages_Free:      4
Hugepagesize:     2048 kB
[lxbuild066] ~/public/multicore > cat /proc/meminfo | grep -i huge
HugePages_Total:    12
HugePages_Free:      4
Hugepagesize:     2048 kB
[1]    Done                          ~/w1/scimark2/scimark2 -large > scimark2_tc_hugetlb
[lxbuild066] ~/public/multicore > cat scimark2_tc_hugetlb
**                                                              **
** SciMark2 Numeric Benchmark, see http://math.nist.gov/scimark **
** for details. (Results can be submitted to pozo@nist.gov)     **
**                                                              **
Using       2.00 seconds min time per kenel.
Composite Score:          452.73
FFT             Mflops:    34.39    (N=1048576)
SOR             Mflops:   849.27    (1000 x 1000)
MonteCarlo:     Mflops:   191.06
Sparse matmult  Mflops:   485.31    (N=100000, nz=1000000)
LU              Mflops:   703.61    (M=1000, N=1000)
```

so it works, but it is slower! btw scimark2 is faster with standard malloc (mainly because of the `MonteCarlo` that does not stress memory much)

```
unsetenv LD_PRELOAD $TCMALLOC_ROOT/lib/libtcmalloc_minimal.so
[lxbuild066] ~/public/multicore > ~/w1/scimark2/scimark2 -large
**                                                              **
** SciMark2 Numeric Benchmark, see http://math.nist.gov/scimark **
** for details. (Results can be submitted to pozo@nist.gov)     **
**                                                              **
Using       2.00 seconds min time per kenel.
Composite Score:          522.15
FFT             Mflops:    55.42    (N=1048576)
SOR             Mflops:   851.63    (1000 x 1000)
MonteCarlo:     Mflops:   403.66
Sparse matmult  Mflops:   471.89    (N=100000, nz=1000000)
LU              Mflops:   828.16    (M=1000, N=1000)
```

summary of the measurement below

Using **tcmalloc** cms reconstruction for b-jets is about 10% faster. a system analysis shows that the number of *retired instruction* is essentially the some. Most of the saving comes from less *resources stalls* and some less *miss-predicted branches*

measuring cms reconstruction

these instructions are valid for CMSSW 2_0_9 run the preparatory sequence `cmsDriver.py B_JETS -s STEP -n 100` with `STEP` = `GEN, SIM, DIGI` than run `~/pyModules/pfmon_deluxe.py cmsDriver.py B_JETS -s RECO -n 100`

results

```
TimeReport ---------- Event  Summary ---[sec]----
TimeReport CPU/event = 3.046750 Real/event = 3.059505

TimeReport ---------- Path   Summary ---[sec]----
TimeReport             per event          per path-run
TimeReport        CPU       Real        CPU       Real Name
TimeReport   2.798615   2.811101   2.798615   2.811101 reconstruction_step

TimeReport -------End-Path   Summary ---[sec]----
TimeReport             per event       per endpath-run
TimeReport        CPU       Real        CPU       Real Name
```

```
TimeReport   0.247656   0.247859   0.247656   0.247859 outpath


970293569059 UNHALTED_CORE_CYCLES
943404623915 INSTRUCTIONS_RETIRED
153400155587 BRANCH_INSTRUCTIONS_RETIRED
  4283583203 MISPREDICTED_BRANCH_RETIRED
366093204853 INST_RETIRED:LOADS
           0 SIMD_COMP_INST_RETIRED:PACKED_SINGLE:SCALAR_SINGLE:PACKED_DOUBLE:SCALAR_DOUBLE
198866522599 INST_RETIRED:STORES
 57650513756 X87_OPS_RETIRED:ANY
305133549321 RESOURCE_STALLS:ANY
  1103247481 BUS_TRANS_ANY:ALL_AGENTS
  1164574768 BUS_DRDY_CLOCKS:ALL_AGENTS
   365249031 BUS_BNR_DRV:ALL_AGENTS
 21200587333 LAST_LEVEL_CACHE_REFERENCES
   187148195 LAST_LEVEL_CACHE_MISSES
138684654138 CPU_CLK_UNHALTED:BUS
--------------------------------------------------------------------------
Ratios:
                           CPI: 1.0285
            load instructions %: 38.806%
           store instructions %: 21.080%
  load and store instructions %: 59.885%
  resource stalls % (of cycles): 31.448%
         branch instructions %: 16.260%
% of branch instr. mispredicted: 2.792%
           % of l2 loads missed: 0.883%
              bus utilization %: 1.591%
         data bus utilization %: 0.840%
                bus not ready %: 0.527%
  comp. SIMD inst. ('new FP') %: 0.000%
  comp. x87 instr. ('old FP') %: 6.111%
```

repeated with TCMALLOC

```
TimeReport ---------- Event  Summary ---[sec]----
TimeReport CPU/event = 2.776694 Real/event = 2.781829

TimeReport ---------- Path   Summary ---[sec]----
TimeReport             per event          per path-run
TimeReport        CPU       Real        CPU       Real Name
TimeReport   2.536359   2.541528   2.536359   2.541528 reconstruction_step

TimeReport -------End-Path   Summary ---[sec]----
TimeReport             per event       per endpath-run
TimeReport        CPU       Real        CPU       Real Name
TimeReport   0.240255   0.240235   0.240255   0.240235 outpath


903705695000 UNHALTED_CORE_CYCLES
931115219416 INSTRUCTIONS_RETIRED
151185416169 BRANCH_INSTRUCTIONS_RETIRED
  4032950096 MISPREDICTED_BRANCH_RETIRED
363136780290 INST_RETIRED:LOADS
           0 SIMD_COMP_INST_RETIRED:PACKED_SINGLE:SCALAR_SINGLE:PACKED_DOUBLE:SCALAR_DOUBLE
194064751706 INST_RETIRED:STORES
 57567303752 X87_OPS_RETIRED:ANY
259105502466 RESOURCE_STALLS:ANY
   965820152 BUS_TRANS_ANY:ALL_AGENTS
  1004396040 BUS_DRDY_CLOCKS:ALL_AGENTS
   346899990 BUS_BNR_DRV:ALL_AGENTS
 20041091763 LAST_LEVEL_CACHE_REFERENCES
   168698808 LAST_LEVEL_CACHE_MISSES
129149087560 CPU_CLK_UNHALTED:BUS
--------------------------------------------------------------------------
```

```
Ratios:
                             CPI: 0.9706
            load instructions %: 39.000%
           store instructions %: 20.842%
  load and store instructions %: 59.842%
  resource stalls % (of cycles): 28.671%
         branch instructions %: 16.237%
% of branch instr. mispredicted: 2.668%
           % of l2 loads missed: 0.842%
              bus utilization %: 1.496%
         data bus utilization %: 0.778%
                bus not ready %: 0.537%
  comp. SIMD inst. ('new FP') %: 0.000%
  comp. x87 instr. ('old FP') %: 6.183%
```

04/07/2008

TCMalloc

I've installed TCMalloc (using Giulio's receipt) as

```
setenv TCMALLOC_ROOT /afs/cern.ch/user/i/innocent/w1/tcmalloc
mkdir -p $TCMALLOC_ROOT
cd $TCMALLOC_ROOT
wget http://google-perftools.googlecode.com/files/google-perftools-0.97.tar.gz
tar xzvf google-perftools-0.97.tar.gz
cd google-perftools-0.97
./configure --prefix $TCMALLOC_ROOT --enable-frame-pointers
make
make install
```

to use it just

```
setenv TCMALLOC_ROOT /afs/cern.ch/user/i/innocent/w1/tcmalloc
setenv LD_PRELOAD $TCMALLOC_ROOT/lib/libtcmalloc_minimal.so
```

for 32 bit the best in to initialize a CMSSW first and then

```
setenv TCMALLOC_ROOT /afs/cern.ch/user/i/innocent/w1/tcmalloc32
mkdir -p $TCMALLOC_ROOT
cd $TCMALLOC_ROOT
wget http://google-perftools.googlecode.com/files/google-perftools-0.97.tar.gz
tar xzvf google-perftools-0.97.tar.gz
cd google-perftools-0.97
linux32 ./configure --prefix $TCMALLOC_ROOT --enable-frame-pointers
linux32 make
linux32 make install
```

`ldd` should look like this

```
ldd /afs/cern.ch/user/i/innocent/w1/tcmalloc32/lib/libtcmalloc_minimal.so
   linux-gate.so.1 =>  (0xf7f8d000)
   libstdc++.so.6 => /afs/cern.ch/cms/sw/slc4_ia32_gcc345/external/gcc/3.4.5-cms/lib/libstdc++.so
   libm.so.6 => /lib/tls/libm.so.6 (0xf7e48000)
   libc.so.6 => /lib/tls/libc.so.6 (0xf7d1c000)
   libgcc_s.so.1 => /afs/cern.ch/cms/sw/slc4_ia32_gcc345/external/gcc/3.4.5-cms/lib/libgcc_s.so.1
   /lib/ld-linux.so.2 (0x008bc000)
```

Once preloaded one will get `ERROR: ld.so: object '/afs/cern.ch/user/i/innocent/w1/tcmalloc32/lib/libtcmalloc_minimal.so' from LD_PRELOAD`

`cannot be preloaded: ignored.` for every possible shell command, just ignore...

-- VincenzoInnocente - 02-Mar-2011

This topic: LCG > VIHugePages
Topic revision: r4 - 2017-03-26 - VincenzoInnocente

Copyright &© 2008-2020 by the contributing authors. All material on this collaboration platform is the property of the contributing authors.
Ideas, requests, problems regarding TWiki? Send feedback