Transparent Hugepage Support
Red Hat, Inc.

Andrea Arcangeli
aarcange at redhat.com

Collaboration Summit
High Performance Computing track
San Francisco, CA

7 April 2011
Benefit of hugepages

➢ Boost CPU computing performance
  ➢ Enlarge TLB size
    ➢ TLB is also separate for 4k and 2m pages
  ➢ Speed up TLB miss
    ➢ Need 3 accesses to memory instead of 4 to refill the TLB
    ➢ Faster to allocate memory initially (minor)
    ➢ Page colouring inside the hugepage (minor)
➢ Cons
  ➢ clear_page/copy_page less cache friendly
  ➢ Slightly higher memory footprint in some case
TLB miss cost is 4 memory access

Virt addr -> pgd -> pud -> pmd -> pte -> RAM

TLB miss cost 4k pages
TLB miss cost is 3 memory access
NPT/EPT TLB miss cost: number of accesses to memory

- novirt THP on
- novirt THP off
- host THP on guest THP on
- host THP off guest THP on
- host THP on guest THP off
- host THP off guest THP off

0 5 10 15 20 25
Cache effect

➢ To access 16G of memory the CPU has to read
  ➢ 32MBytes worth of ptes with 4k pages (not counting pmd/pud/pgd)

➢ With hugepages the CPU will read only 64KBytes of hugepmd with hugepages

➢ 64KBytes
  ➢ fit into CPU cache

➢ 32MBytes
  ➢ don't fit into CPU cache
Limit of hugetlbfs

- Hugepages can be used with hugetlbfs
  - They can't be **swapped** out
  - They better be **reserved** at boot
- Hugepages and regular pages can't be mixed in the same vma (**only userland fallback**)
- If reservation is not used and dynamic allocation fails things go bad in KVM
- **Requires admin privilege** and libhugetlbfs
- Hugetlbfs is **growing like a second but inferior Linux VM** with its own paths, as people add more features to hugetlbfs to behave more like tmpfs
Hugetlbfs for database

- Reservation at boot time may not be big deal with database
  - 1 database
  - 1 machine
  - 1 database cache
  - 1 database cache size set in config file or GUI
  - 1 reservation of hugepages with known size
  - Swapping is still missing (some DBMS want to swap its shared memory)

- Hugetlbfs is usually ok only for database
Hypervisors and hugetlbfs

- Hugetlbfs is not good for KVM
  - **Unknown number** of virtual machines
  - **Unknown amount of memory** used by virtual machines
- We want to use as many hugepages as available to back guest physical memory (especially with NPT/EPT)
- Virtual machines are **started, shutdown, migrated on demand** by user or RHEV-M
- We need **overcommit (and KSM)** as usual
- We want all memory not allocated by the guest available to the host for caching
Hugetlbfs userbase

➢ Not many are using hugetlbfs on laptop/workstation/server
  ➢ Too many complications (not transparent)
  ➢ Too many disadvantages/limitations

➢ As opposed: even the OpenOffice used to prepare this presentation is backed by some Transparent Hugepage...
Transparent Hugepage design

- Any Linux process will receive 2M pages
  - if the `mmap` region is 2M naturally aligned
- Hugepages are **only** mapped by huge pmd
- When VM pressure triggers the hugepage are split
  - Then they can be **swapped out as 4k pages**
- Tries to **modify as little code as possible**
- Entirely **transparent to userland**
- Already working with **KVM with NPT/EPT and shadow MMU**
- Boost for page faults too and later the CPU accesses memory faster
THP on anonymous memory

- Current implementation **only covers anonymous memory** (MAP_ANONYMOUS, i.e. malloc())
  - KVM guest physical memory is incidentally backed by anonymous memory...
- In the future database may require tmpfs to use transparent hugepages too if they want to swap
  - database main painful limit of hugetlbfs is the lack of swapping
split_huge_page

➢ Low code impact

➢ Try to stay self contained
   ➢ If the code is not THP aware it's enough to call split_huge_page() to make it THP aware
     ➢ then it's business as usual

➢ 1 liner trivial change vs >100 lines of non trivial code

➢ Over time we need to minimize the use of split_huge_page

➢ Like the big kernel lock (lock_kernel() going away over time where avoidable)
collapse_huge_page/khugepaged

➢ “khugepaged” scans the virtual address space
  ➢ it collapses 512 4k pages in one 2M page
  ➢ it converts the 512 ptes to a huge pmd
➢ “khugepaged” can undo the effect of
  split_huge_page
  ➢ Like after swapin
THP sysfs enabled

➢ /sys/kernel/mm/transparent_hugepage/enabled
  ➢ [always] madvise never
    ➢ Try to use THP on every big enough vma to fit 2M pages
  ➢ always [madvise] never
    ➢ Only inside MAD_HUGEPAGE regions
      ➢ Applies to khugepaged too
  ➢ always madvise [never]
    ➢ Never use THP
      ➢ khugepaged quits
➢ Default selected at build time (enabled|madvise)

Copyright © 2011 Red Hat Inc.
THP kernel boot param

➢ To alter the default build time setting
  ➢ transparent_hugepage=always
  ➢ transparent_hugepage=madvise
  ➢ transparent_hugepage=never
    ➢ khugepaged isn't even started
khugepaged sysfs

- `/sys/kernel/mm/transparent_hugepage/khugepaged`
  - `pages_to_scan` (default 4096 = 16MB)
    - Number of pages to scan at each wakeup
  - `scan_sleep_millisecs` (default 10000 = 10sec)
    - How long before khugepaged is waken up to scan “pages_to_scan” virtual pages
      - 0 value run khugepaged at 100% load
  - `alloc_sleep_millisecs` (default 60000 = 60sec)
    - How long to wait before trying again allocating an hugepage in case of fragmentation
THP monitoring

$ grep Anon /proc/meminfo
AnonPages:       15719600 kB
**AnonHugePages:**  14436352 kB

$ cat /proc/`pgrep mutt`/smaps|grep Anon
Anonymous:         0 kB
AnonHugePages:     0 kB
Anonymous:         4 kB
AnonHugePages:     0 kB
Anonymous:         20 kB
AnonHugePages:     0 kB
Anonymous:        20 kB
AnonHugePages:     0 kB
Anonymous:       69400 kB
**AnonHugePages:**    67584 kB
[...]
THP vmstat

$ grep thp /proc/vmstat # during heavy swap

```
thp_fault_alloc 66608
  ➢ Transparent Hugepages allocated in page faults
  ➢ The higher the better
```

```
thp_fault_fallback 546
  ➢ Failure in allocating hugepage in fault → fallback to 4k
  ➢ The lower the better
```

```
thpCollapse_alloc 113
  ➢ Transparent Hugepages collapsed by khugepaged
```

```
thpCollapse_alloc_failed 5
  ➢ Failure in allocating hugepage in khugepaged
```

```
 thp_split 22608
  ➢ Number of split_huge_page()
  ➢ The lower the better
```
Optimizing apps for THP

➢ Not really required
  ➢ Mutt example → unmodified:
    - Anonymous: 69400 kB
    - **AnonHugePages**: 67584 kB

➢ `posix_memalign(&ptr, 2M, (size+2M-1) & ~(2M-1))`
  ➢ Allows max 2 more THP allocated per mapping
    ➢ Generally not very important
    ➢ Only KVM requires this: gfn → hva → pfn

➢ **Glibc could learn** to auto-align large mappings

➢ 4M for x86 32bit noPAE
madvise(MADV_HUGEPAGE)

➢ To use hugepages only in specific regions
  ➢ To avoid altering the memory footprint
  ➢ **Embedded systems** want to use it

➢ Makes a difference **only** when
  “/sys/kernel/mm/transparent_hugepage/enabled”
  is set to “madvise”

➢ Better than libhugetlbfs for embedded:
  ➢ swap enabled
  ➢ full userland transparency
  ➢ no root privilege
  ➢ no library dependency
Transparent Hugepages and KVM

➢ We need THP in both guest and host
  ➢ So the CPU can use the 2M TLB for the guest

➢ This shows the power of KVM design
  ➢ same algorithm
  ➢ same code
  ➢ same kernel image
  ➢ For both KVM hypervisor and guest OS
RHEL6  Linux Intel EP Specjbb Java Bare-Metal Huge/Transparent Huge Pages

RHEL5.5 /6 SPECjbb Scaling Intel EX

- RHEL5.4
- RHEL6
- RHEL5 Huge
- RHEL6 Huge
- R6 vs R5
RHEL6/6.1 KVM Linux Intel Westmere EP
Specjbb transparent hugepages/unfair_spin

RHEL6/6.1 SPECjbb
24-cpu, 24 vcpu Westmere EP, 24GB
THP and kbuild

➢ GCC allocations are specially optimized (gcc isn't using glibc malloc)
  ▶ Requires a small tweak to gcc

➢ Heavily parallel

➢ Heavily MMU intensive

➢ Worst case benchmark for THP, especially on bare metal
  ▶ Small working set for each task
  ▶ It even includes `make clean` etc...

➢ Phenom X4 kbuild (no virt)
  ▶ 2.5% faster with THP
gcc patch (trivial)

-450,6 +450,11 @@
#define BITMAP_SIZE(Num_objects) \ 
 (CEIL ((Num_objects), HOST_BITS_PER_LONG) * sizeof(long))

+ifdef __x86_64__
+define HPAGE_SIZE (2*1024*1024)
+define GGC_QUIRE_SIZE 512
+endif
+
/* Allocate pages in chunks of this size, to throttle calls to memory
 allocation routines. The first page is used, the rest go onto the
 free list. This cannot be larger than HOST_BITS_PER_INT for the
 @@ -654,6 +659,23 @@
 ifndef HAVE_MMAP_ANON
 char *page = (char *) mmap (pref, size, PROT_READ | PROT_WRITE,
 MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
+ifdef HPAGE_SIZE
 + if (!((size & (HPAGE_SIZE-1))) &&
 + page != (char *) MAP_FAILED && (size_t) page & (HPAGE_SIZE-1)) {
 + char *old_page;
 + munmap(page, size);
 + page = (char *) mmap (pref, size + HPAGE_SIZE-1,
 + PROT_READ | PROT_WRITE,
 + MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
 + old_page = page;
 + page = (char *) (((size_t)page + HPAGE_SIZE-1)
 + & ~(HPAGE_SIZE-1));
 + if (old_page != page)
 + munmap(old_page, page-old_page);
 + if (page != old_page + HPAGE_SIZE-1)
 + munmap(page+size, old_page+HPAGE_SIZE-1-page);
 + }
+endif
```
`perf` of kbuild (real life)

24-way SMP (12 cores, 2 sockets) 16G RAM host, 24-vcpu 15G RAM guest

====== build ======
#!/bin/bash
make clean >/dev/null; make -j32 >/dev/null

THP always host (base result)
Performance counter stats for './build' (3 runs):

    4420734012848  cycles                     ( +-   0.007% )
    2692414418384  instructions             #      0.609 IPC     ( +-   0.000% )
       696638665612  dTLB-loads                 ( +-   0.001% )
       2982343758  dTLB-load-misses           ( +-   0.051% )

     83.855147696  seconds time elapsed   ( +-.058%)

THP never host (slowdown 4.06%)
Performance counter stats for './build' (3 runs):

    4599325985460  cycles                     ( +-   0.013% )
    2747874065083  instructions             #      0.597 IPC     ( +-   0.000% )
       710631792376  dTLB-loads                 ( +-   0.000% )
       4425816093  dTLB-load-misses           ( +-   0.039% )

     87.260443531  seconds time elapsed   ( +-   0.075% )

Copyright © 2011 Red Hat Inc.
```
kbuild bench
build time: lower is better

- KVM guest THP off host THP off EPT off: 260.15% increase
- KVM guest THP off host THP on EPT off: 254.43% increase
- KVM guest THP on host THP on EPT off: 198.33% increase
- KVM guest THP off host THP off EPT on: 24.81% increase in build time
- KVM guest THP off host THP on EPT on: 12.71% increase in build time
- KVM guest THP on host THP on EPT on: 5.67% increase in build time
- bare metal THP off: 4.06% increase in build time
- bare metal THP on: base

x seconds
qemu-kvm translate.o

- Phenom X4 qemu-kvm translate.o build (no virt)
  - 10% faster with THP
  - this is a single gcc task running
    - Not parallel
    - no `make -jxx`
    - no `make clean`
  - Will follow the result on 24-way SMP
`perf` profiling of `translate.o`

24-way SMP (12 cores, 2 sockets) 16G RAM host, 24-vcpu 15G RAM guest

THP always bare metal (base result)

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Relative Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>40746051351</td>
<td>(+- 5.597%)</td>
</tr>
<tr>
<td>Instructions</td>
<td>36394696366</td>
<td># 0.893 IPC (+- 0.007%)</td>
</tr>
<tr>
<td>dTLB-loads</td>
<td>9602461977</td>
<td>(+- 0.006%)</td>
</tr>
<tr>
<td>dTLB-load-misses</td>
<td>45123574</td>
<td>(-+ 0.614%)</td>
</tr>
</tbody>
</table>

13.920436128 seconds time elapsed (+- 5.600%)

THP never bare metal (9.10% slower)

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Relative Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>44492051930</td>
<td>(+- 5.189%)</td>
</tr>
<tr>
<td>Instructions</td>
<td>36757849113</td>
<td># 0.826 IPC (+- 0.001%)</td>
</tr>
<tr>
<td>dTLB-loads</td>
<td>9693482648</td>
<td>(+- 0.004%)</td>
</tr>
<tr>
<td>dTLB-load-misses</td>
<td>63675970</td>
<td>(+- 0.598%)</td>
</tr>
</tbody>
</table>

15.188315986 seconds time elapsed (+- 5.194%)

Copyright © 2011 Red Hat Inc.
kbuild “EPT off”
built time: lower is better

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Build Time Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>KVM guest THP off host THP off EPT off</td>
<td>28.48% increase</td>
</tr>
<tr>
<td>KVM guest THP off host THP on EPT off</td>
<td>25.20% increase</td>
</tr>
<tr>
<td>KVM guest THP on host THP on EPT off</td>
<td>0.11% increases</td>
</tr>
<tr>
<td>KVM guest THP off host THP off EPT on</td>
<td>21.17% increase</td>
</tr>
<tr>
<td>KVM guest THP off host THP on EPT on</td>
<td>15.84% increase in</td>
</tr>
<tr>
<td>KVM guest THP on host THP on EPT on</td>
<td>3.45% decrease in</td>
</tr>
<tr>
<td>bare metal THP off</td>
<td>9.10% increase in</td>
</tr>
<tr>
<td>bare metal THP on</td>
<td>Base result</td>
</tr>
</tbody>
</table>

seconds

0 2 4 6 8 10 12 14 16 18 20

Copyright © 2011 Red Hat Inc.
Phoronix test suite


- IS.C test of NASA's OpenMP-based performance boost more than 20%
  - No virt
  - On Thinkpad T16 notebook
    - Core 2 Duo T9300
    - 4GB of RAM
  - A bigger boost is expected on server/virt
Other results

- “/usr/bin/sort -b 1200M /tmp/largerand” no virt
  - 6% faster with THP (reported on lkml)
- Vmware workstation SPECJBBB with hugetlbfs in guest
  - 22% faster with THP (reported on lkml)
Transparent Hugepages status

- Fully merged in 2.6.38 upstream
- Memory compaction included in 2.6.35
  - Memory compaction motivated by THP
- THP enabled by default in RHEL6 (guest & host)
- KSM fully THP aware (2.6.38 and RHEL6.1)
  - Mix of PageKsm, PageTransHuge and regular anon pages in the same vma
    - All 3 kind of anonymous pages swappable
- mprotect/mincore/memcg THP support in 2.6.38
- /proc/<pid>/smaps support in 2.6.39-rc
THP future optimizations

➢ mremap THP support + tlb boost ready for -mm
➢ Remove tlb flush in pmdp_splitting_flush_notify()
➢ Avoid some unnecessary split_huge_page:
  ➢ migrate_pages()/move_pages() syscall
➢ More glibc awareness for automatic alignments of large mmap
➢ pagecache
  ➢ tmpfs
  ➢ swapcache (i.e. native THP swapping)
  ➢ Maybe filebacked mappings?
Q/A

- You're very welcome!

- Latest development THP code
  - http://git.kernel.org and then search “aa.git”
  - First: git clone
    git://git.kernel.org/pub/scm/linux/kernel/git/andrea/aa.git
  - Later: git fetch && git checkout -f origin/master