Transparent Hugepage Support

Red Hat Inc.

Andrea Arcangeli
aarcange at redhat.com

KVM Forum 2010
Boston

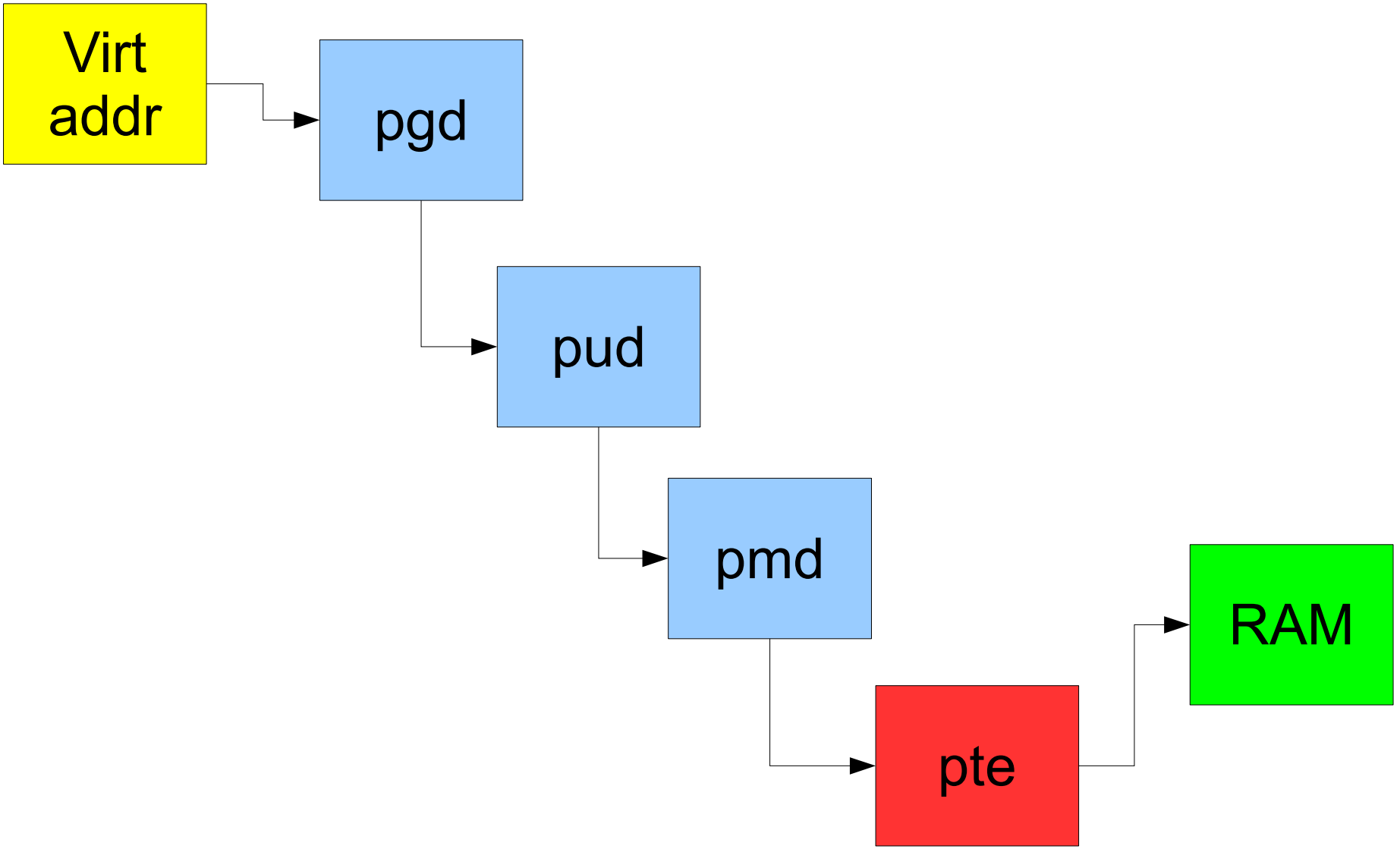
9 Aug 2010

redhat.

Benefit of hugepages

- Enlarge TLB size
  - TLB is 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
  - Initial page fault huge speed up (like 50% faster)
- Cons: clear_page/copy_page less cache friendly

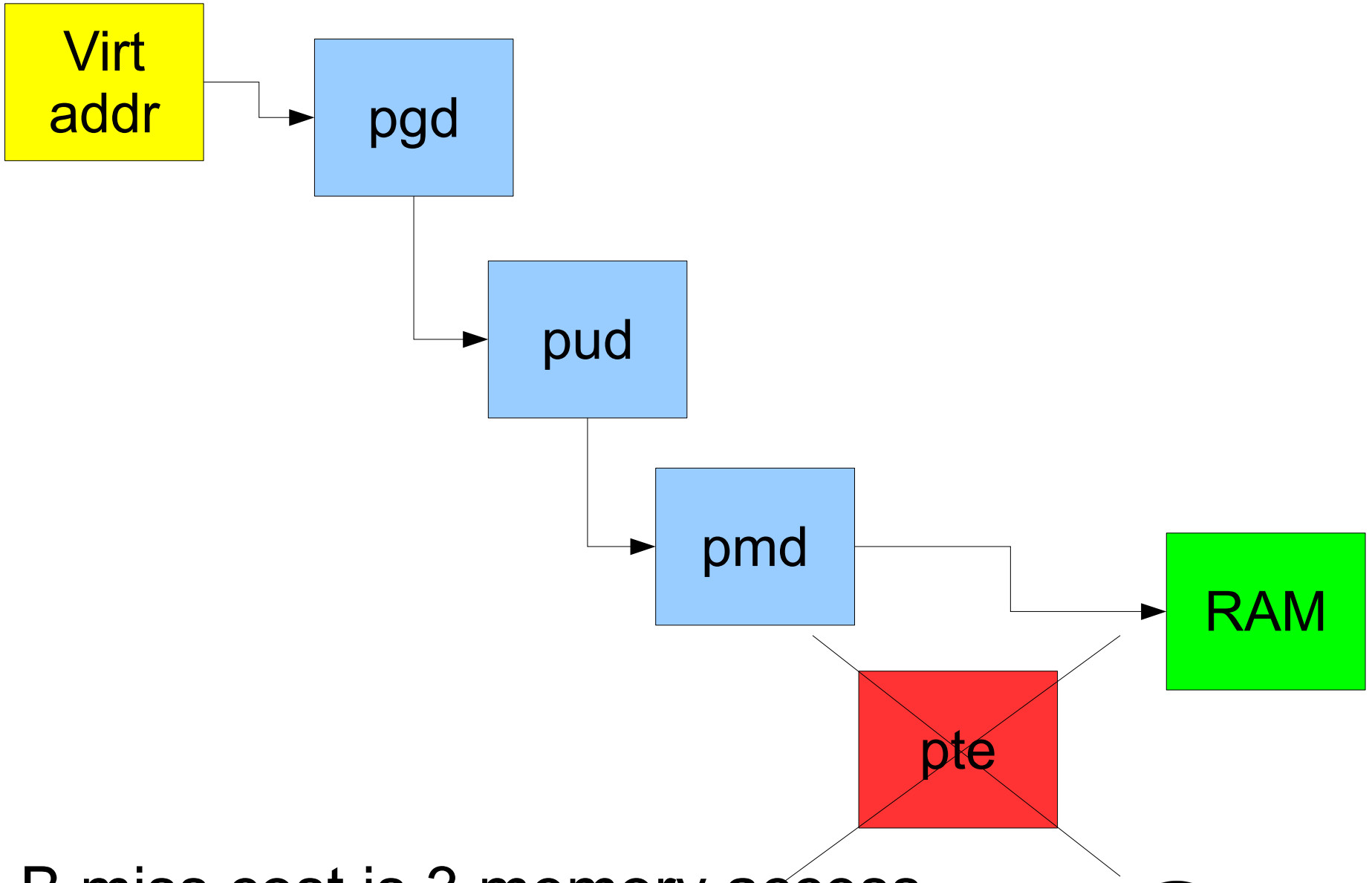
TLB miss cost 4k pages



TLB miss cost is 4 memory access

redhat

TLB miss cost 2M pages



TLB miss cost is 3 memory access

redhat

NPT/EPT TLB miss cost

- Guest THP off, KVM host THP off
  - 4 guest levels x 5 NTP accesses + 4 NPT accesses for final gpa->hpa translation = 24 memory accesses
- Guest THP off, KVM host THP on
  - 4 guest levels x 4 NTP accesses + 3 NPT accesses final gpa->hpa = 19 accesses
- Guest THP on, KVM host THP on
  - 3 x 4 + 3 = 15 accesses
- (not counting data access)

Cache effect

- To access 16G of memory the CPU has to read
  - 32MBytes worth of ptes (not counting pmd/pud/pgd)
  - With hugepages the CPU will read only 64KBytes of hugepages
- 64KBytes fits in CPU cache, 32MBytes don't...

redhat

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
  - If reservation is not used and dynamic allocation fails things go bad in KVM
  - Requires admin privilege and libhugetlbfs tricks
  - hugetlbfs is growing like a second but inferior Linux VM with its own paths, as people adds more features to hugetlbfs to behave more like tmpfs

redhat

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)

redhat

hugetlbfs poll!

- Raise hand
  - who is running any applications under libhugetlbfs/hugetlbfs on his own production laptop/workstation/server
- Even the OpenOffice used to prepare this presentation is backed by some Transparent Hugepage...

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 don't want to alter behavior of boosted virtual machines and we need overcommit (and KSM) as usual

redhat.

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

redhat.

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)

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)

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

madvise(MADV_HUGEPAGE)

- To use hugepages only in specific regions
  - To avoid altering the memory footprint
  - Embedded systems want to use it
- Becomes effective when sysfs enabled is set to“madvise”
- Better than libhugetlbfs
  - swap
  - full userland transparency
  - no root privilege
  - no library dependency

redhat

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)

redhat

collapse_huge_page

- “khugepaged” scans the virtual address space
  - it collapses 512 4k pages in one 2M page
  - it converts the 512 ptes to a huge pmd
- “pages_to_scan”
- “scan_sleep_millisecs” (can be set to 0)
- “alloc_sleep_millisecs”
  - Throttle THP allocations in case of fragmentation

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

redhat

THP and kbuild

- GCC allocations are specially optimized (no glibc)
  - 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 @@
 #ifdef 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
```


redhat.

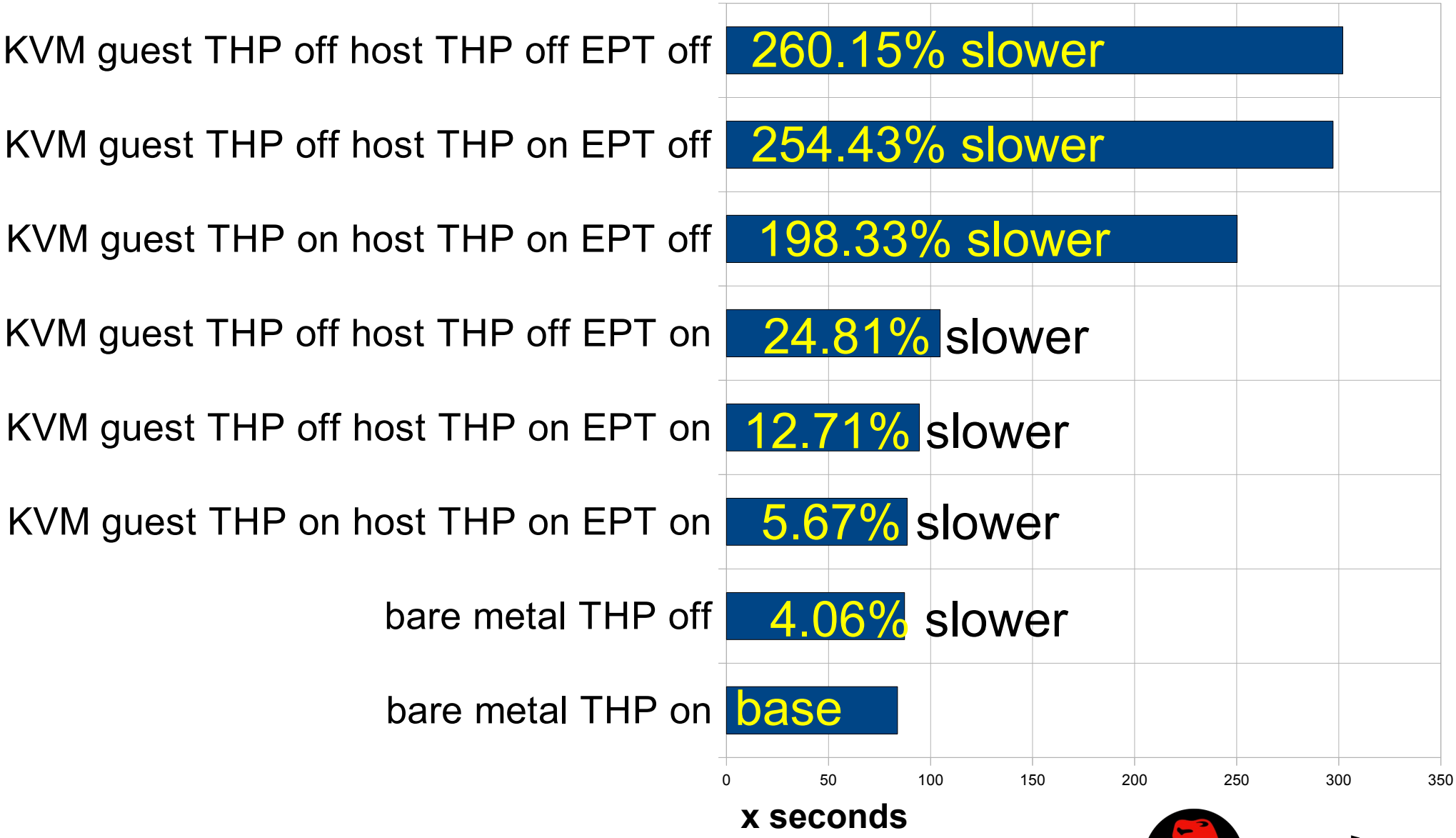
`perf` of kbuild (real life)

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   ( +-   0.058% )

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% )
```


redhat.

kbuild bench (shorter is better)



redhat

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

redhat

`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)

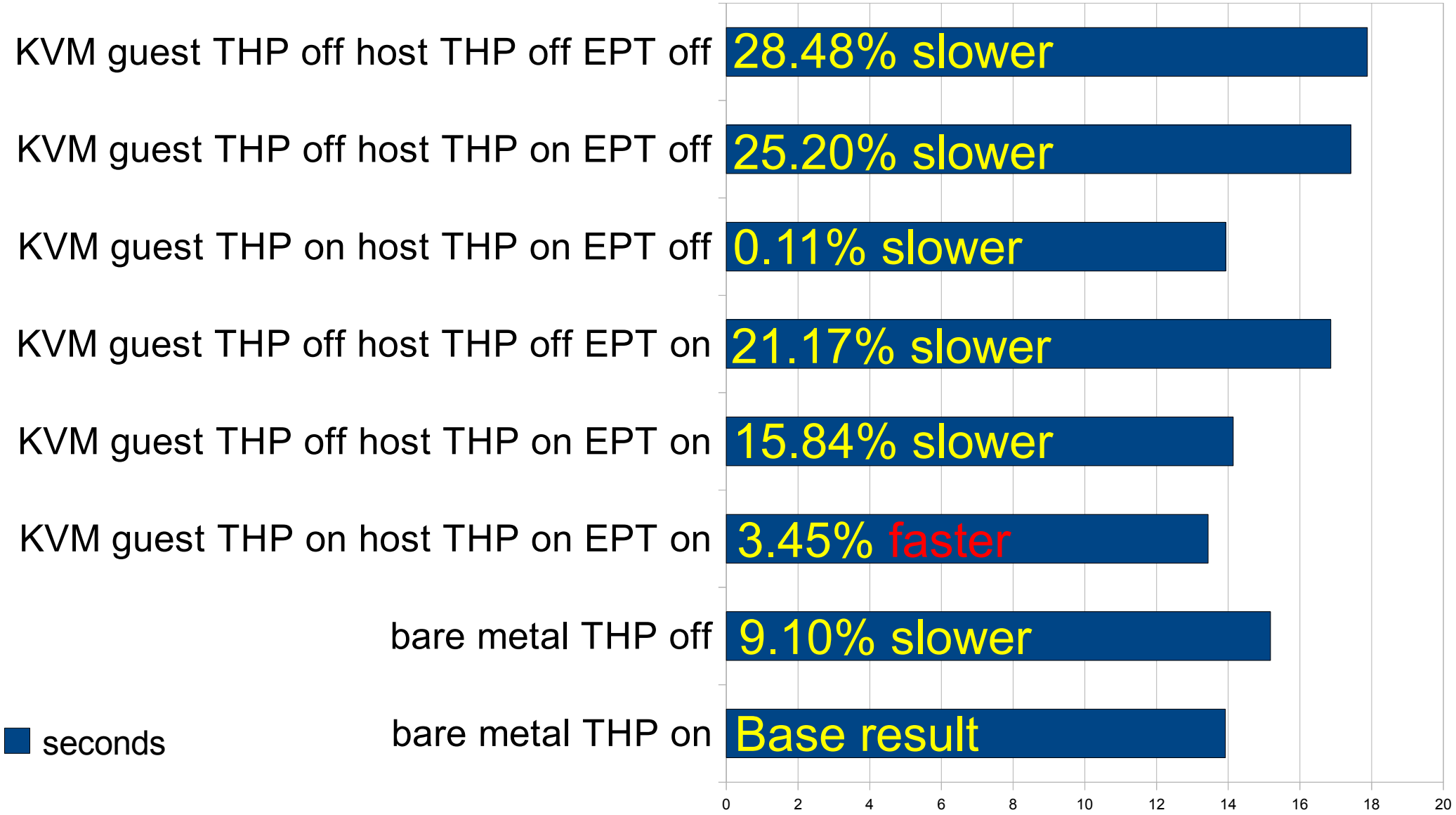
```
 40746051351  cycles                    ( +-   5.597% )
 36394696366  instructions              #      0.893 IPC     ( +-   0.007% )
  9602461977  dTLB-loads                ( +-   0.006% )
    45123574  dTLB-load-misses          ( +-   0.614% )

13.920436128  seconds time elapsed      ( +-   5.600% )
```

THP never bare metal (9.10% slower)

```
 44492051930  cycles                    ( +-   5.189% )
 36757849113  instructions              #      0.826 IPC     ( +-   0.001% )
  9693482648  dTLB-loads                ( +-   0.004% )
    63675970  dTLB-load-misses          ( +-   0.598% )

15.188315986  seconds time elapsed      ( +-   5.194% )
```


redhat

kbuild “EPT off”

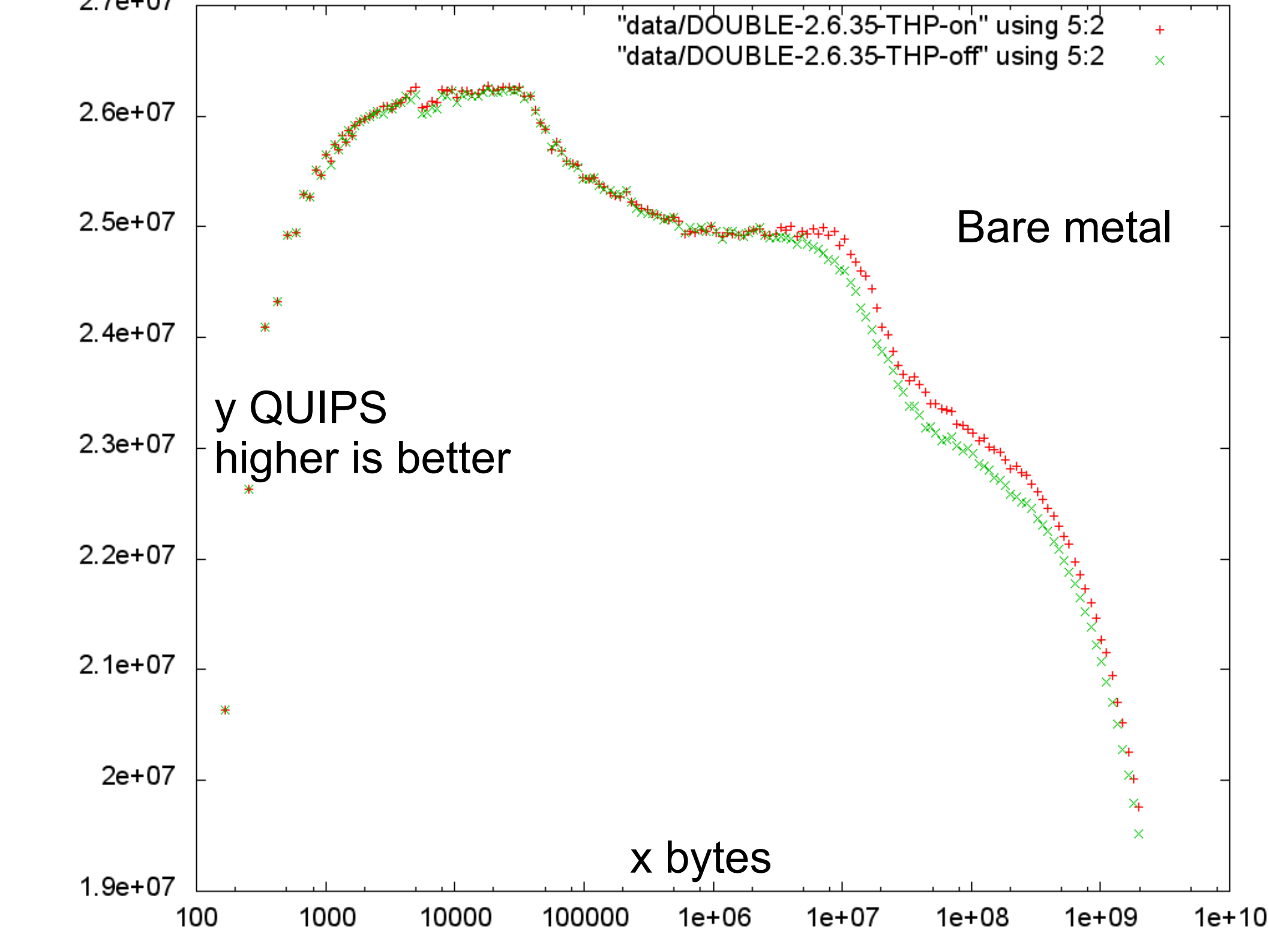


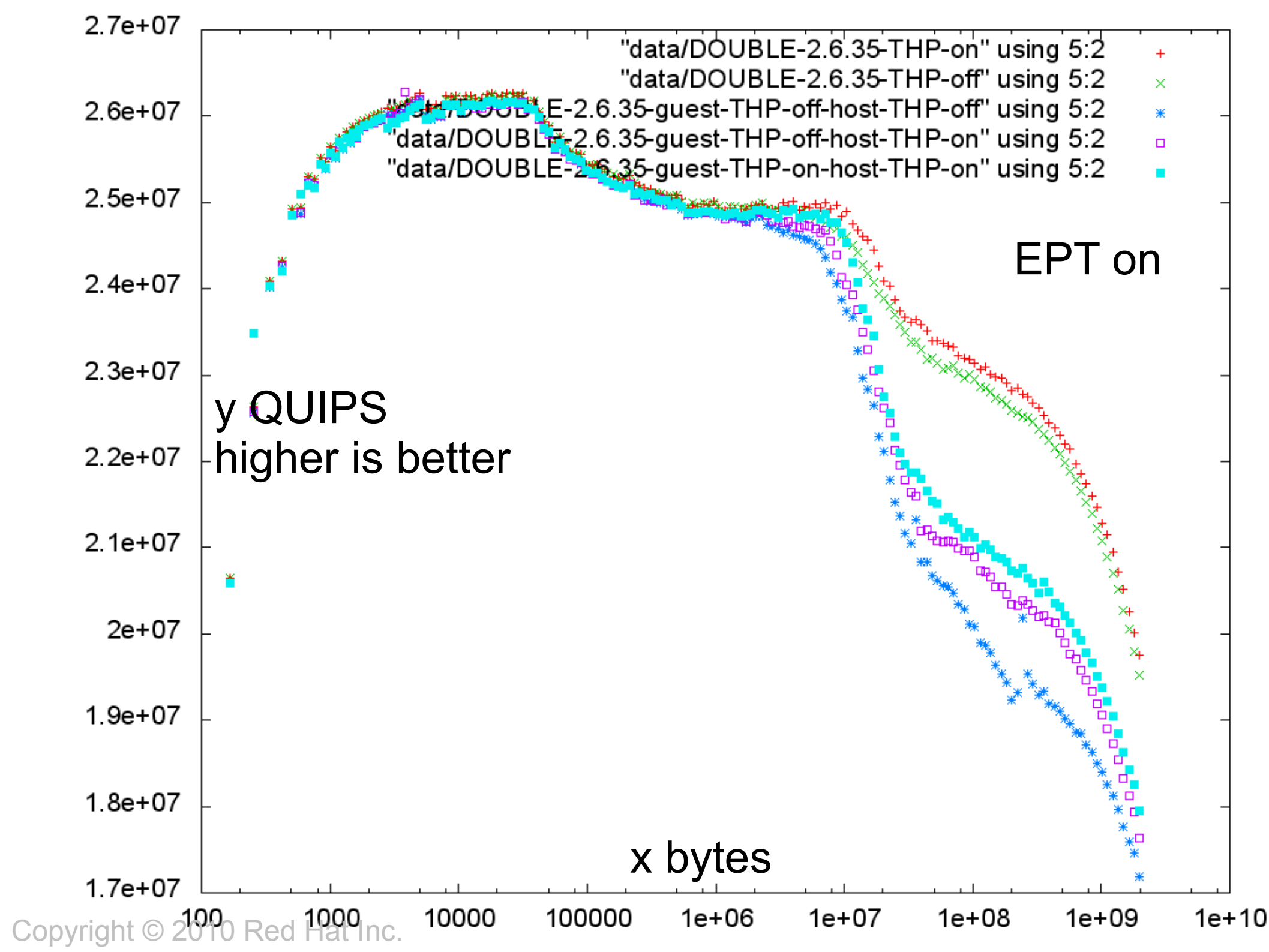
redhat

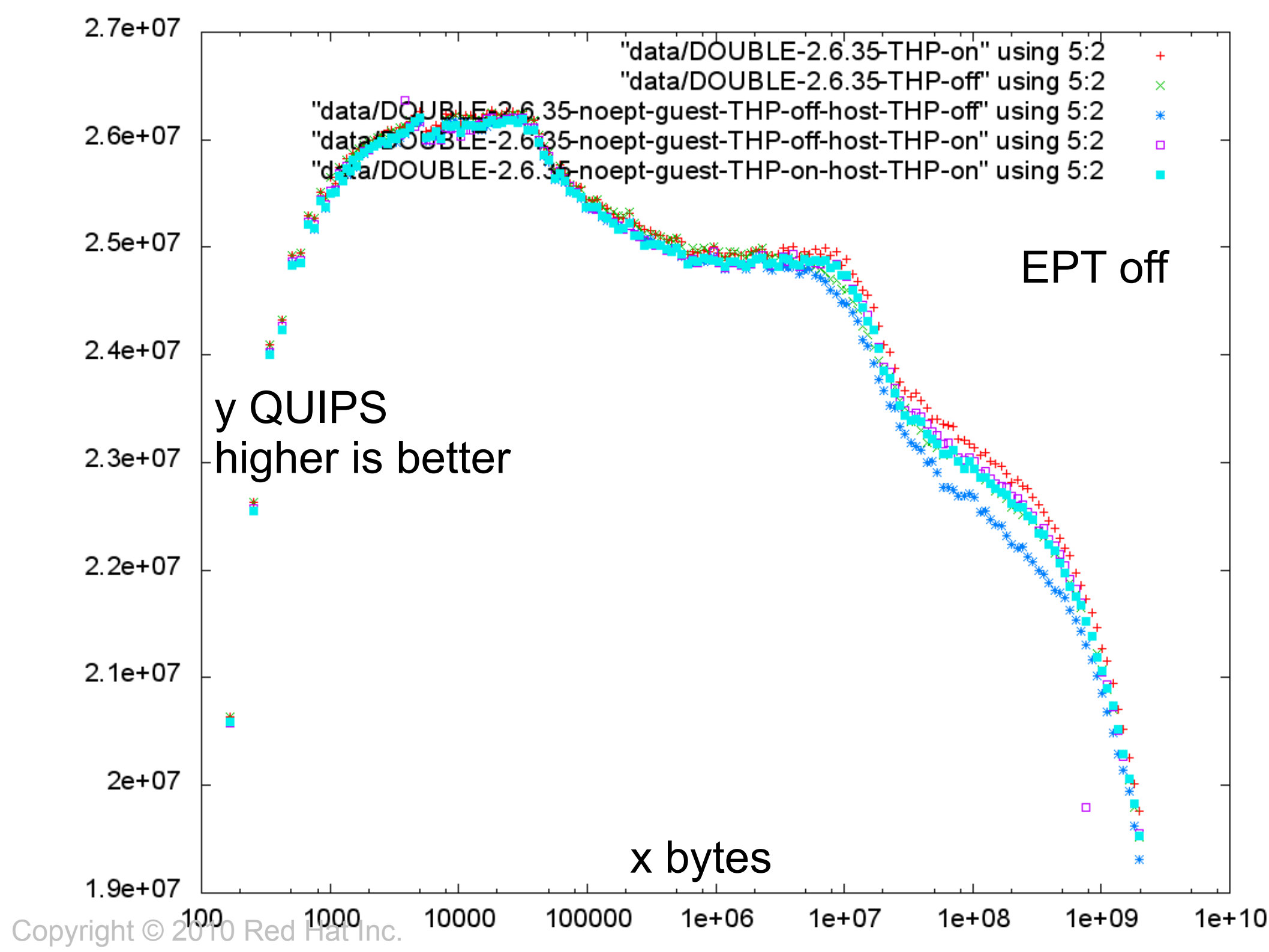
Hierarchical INTegration

- No THP related modification required
- Performance of scientific computing (Y)
  - In function on the memory size (X)
    - Show cache sizes etc..
- No MMU guest mangling (optimal for EPT off)

redhat.

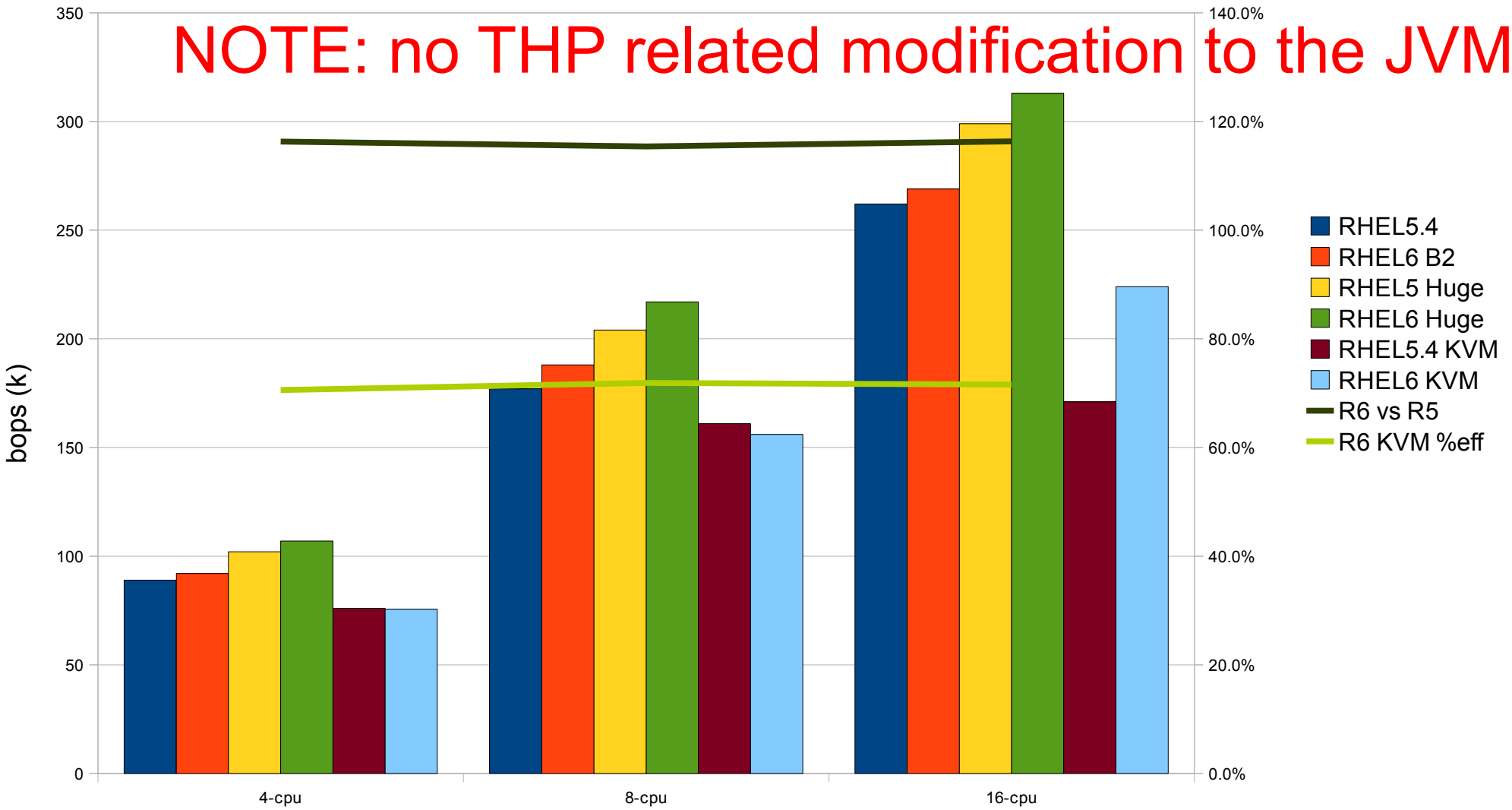

"data/DOUBLE-2.6.35-THP-on" using 5:2
"data/DOUBLE-2.6.35-THP-off" using 5:2
Bare metal
y QUIPS
higher is better
x bytes
2.7e+07
2.6e+07
2.5e+07
2.4e+07
2.3e+07
2.2e+07
2.1e+07
2e+07
1.9e+07
100
1000
10000
100000
1e+06
1e+07
1e+08
1e+09
1e+10





SPECjbb
hugetlbfs RHEL5 vs THP RHEL6

RHEL5.5 /6 SPECjbb Scaling Intel EX



redhat

Other results

- KVM with THP on guest and host
  - sometime faster than bare metal w/o THP
- “/usr/bin/sort -b 1200M /tmp/largerand” no virt
  - 6% faster with THP (reported on lkml)
- Bare metal SPECJBB
  - 18%?!? faster
- Vmware workstation SPECJBB with hugetlbfs in guest
  - 22% faster with THP (reported on lkml)

Transparent Hugepages future

- Enabled by default in RHEL6 (guest & host)
- Memory compaction included in 2.6.35
  - Memory compaction motivated by THP
- Hopefully THP will be merged in 2.3.36-rc?
- KSM must learn about transparent hugepages
- Remove split_huge_page in mremap
- glibc?
- Possibly expand into tmpfs but hugetlbfs remains:
  - some archs can't mix different page sizes
  - Too big page_size isn't allocatable

redhat.

Q/A

- You're very welcome!
- Open http://git.kernel.org and then search “aa.git”
- **http://git.kernel.org/?p=linux/kernel/git/andrea/aa.git;a=shortlog**
- **First: git clone git://git.kernel.org/pub/scm/linux/kernel/git/andrea/aa.git**
- **Later: git fetch; git checkout -f origin/master**