***Note** - this feature will slow down context switches, and we don't recommend it's use*

A customer was worried about TLB miss performance. Without any performance numbers, they insisted on having the ability to set PPC BAT registers to point at particular regions of memory to avoid any TLB misses.

Previously, BAT registers were only used to map the system address area (first 256M), gain addressability to devices used by the callouts (via the startup callout_memory_map()/callout_io_map() functions), and (on SMP) the cpu_page.

All these are global quantities, so we didn't have to save/restore them across context switches that changed address spaces.

In order to avoid slowing down context switches when this feature is not being used, we put in gear that looked for shared memory objects with the SHMCTL_HIGHUSAGE flag set (see the shm_ctl() function on how to set that bit). This was implemented on top of the variable page size support (which the PPC 600 series family doesn't have)

When that flag is on, and the variable page size support can coalesce the mapping into something that a BAT register can use, it now takes an available BAT and points it at the memory on top of the regular page table entries.

Since BAT translations take priority over the TLB/page tables, you'll never get a TLB miss.

To avoid paying the performance cost of saving and restoring the BAT registers, when we don't need to, we change the memmgr.aspace function pointer to point at "vmm_aspace_bat" (in **memmgr/ppc/600/fam_pte.c**), but only when the SHMCTL_HIGHUSAGE flag actually causes a BAT to be used.

This function has the slower version of the address space switch which saves and restores the BAT registers.

Also, there are usually only 4 instruction and 4 data BAT registers on a PPC600 family chip (some implementations add extra BATs). The uniprocessor kernel needs 3 of them, and the SMP kernel needs 4. So unless you have extra 4 BAT registers then you can only use one BAT mapping, and only on the uniprocessor kernel.