```
**********************************************************
   18983 Fri Jan 29 15:08:08 2016
new/usr/src/uts/i86pc/vm/vm_dep.h
5461 #pragma align before the declaration
**********************************************************
______unchanged_portion_omitted_

 458 /* allocation size to ensure vm_cpu_data_t resides in its own cache line */
 459 #define VM_CPU_DATA_PADSIZE                                             \
 460         (P2ROUNDUP(sizeof (vm_cpu_data_t), L2CACHE_ALIGN_MAX))

 462 /* for boot cpu before kmem is initialized */
 463 extern char     vm_cpu_data0[];

 462 /*
 463  * When a bin is empty, and we can't satisfy a color request correctly,
 464  * we scan.  If we assume that the programs have reasonable spatial
 465  * behavior, then it will not be a good idea to use the adjacent color.
 466  * Using the adjacent color would result in virtually adjacent addresses
 467  * mapping into the same spot in the cache.  So, if we stumble across
 468  * an empty bin, skip a bunch before looking.  After the first skip,
 469  * then just look one bin at a time so we don't miss our cache on
 470  * every look. Be sure to check every bin.  Page_create() will panic
 471  * if we miss a page.
 472  *
 473  * This also explains the '<=' in the for loops in both page_get_freelist()
 474  * and page_get_cachelist().  Since we checked the target bin, skipped
 475  * a bunch, then continued one a time, we wind up checking the target bin
 476  * twice to make sure we get all of them bins.
 477  */
 478 #define BIN_STEP        19

 480 #ifdef VM_STATS
 481 struct vmm_vmstats_str {
 482         ulong_t pgf_alloc[MMU_PAGE_SIZES];      /* page_get_freelist */
 483         ulong_t pgf_allocok[MMU_PAGE_SIZES];
 484         ulong_t pgf_allocokrem[MMU_PAGE_SIZES];
 485         ulong_t pgf_allocfailed[MMU_PAGE_SIZES];
 486         ulong_t pgf_allocdeferred;
 487         ulong_t pgf_allocretry[MMU_PAGE_SIZES];
 488         ulong_t pgc_alloc;                      /* page_get_cachelist */
 489         ulong_t pgc_allocok;
 490         ulong_t pgc_allocokrem;
 491         ulong_t pgc_allocokdeferred;
 492         ulong_t pgc_allocfailed;
 493         ulong_t pgcp_alloc[MMU_PAGE_SIZES];     /* page_get_contig_pages */
 494         ulong_t pgcp_allocfailed[MMU_PAGE_SIZES];
 495         ulong_t pgcp_allocempty[MMU_PAGE_SIZES];
 496         ulong_t pgcp_allocok[MMU_PAGE_SIZES];
 497         ulong_t ptcp[MMU_PAGE_SIZES];           /* page_trylock_contig_pages */
 498         ulong_t ptcpfreethresh[MMU_PAGE_SIZES];
 499         ulong_t ptcpfailexcl[MMU_PAGE_SIZES];
 500         ulong_t ptcpfailszc[MMU_PAGE_SIZES];
 501         ulong_t ptcpfailcage[MMU_PAGE_SIZES];
 502         ulong_t ptcpok[MMU_PAGE_SIZES];
 503         ulong_t pgmf_alloc[MMU_PAGE_SIZES];     /* page_get_mnode_freelist */
 504         ulong_t pgmf_allocfailed[MMU_PAGE_SIZES];
 505         ulong_t pgmf_allocempty[MMU_PAGE_SIZES];
 506         ulong_t pgmf_allocok[MMU_PAGE_SIZES];
 507         ulong_t pgmc_alloc;                     /* page_get_mnode_cachelist */
 508         ulong_t pgmc_allocfailed;
 509         ulong_t pgmc_allocempty;
 510         ulong_t pgmc_allocok;
 511         ulong_t pladd_free[MMU_PAGE_SIZES];     /* page_list_add/sub */
 512         ulong_t plsub_free[MMU_PAGE_SIZES];
 513         ulong_t pladd_cache;
```



```
 514         ulong_t plsub_cache;
 515         ulong_t plsubpages_szcbig;
 516         ulong_t plsubpages_szc0;
 517         ulong_t pfs_req[MMU_PAGE_SIZES];        /* page_freelist_split */
 518         ulong_t pfs_demote[MMU_PAGE_SIZES];
 519         ulong_t pfc_coalok[MMU_PAGE_SIZES][MAX_MNODE_MRANGES];
 520         ulong_t ppr_reloc[MMU_PAGE_SIZES];      /* page_relocate */
 521         ulong_t ppr_relocnoroot[MMU_PAGE_SIZES];
 522         ulong_t ppr_reloc_replnoroot[MMU_PAGE_SIZES];
 523         ulong_t ppr_relocnolock[MMU_PAGE_SIZES];
 524         ulong_t ppr_relocnomem[MMU_PAGE_SIZES];
 525         ulong_t ppr_relocok[MMU_PAGE_SIZES];
 526         ulong_t ppr_copyfail;
 527         /* page coalesce counter */
 528         ulong_t page_ctrs_coalesce[MMU_PAGE_SIZES][MAX_MNODE_MRANGES];
 529         /* candidates useful */
 530         ulong_t page_ctrs_cands_skip[MMU_PAGE_SIZES][MAX_MNODE_MRANGES];
 531         /* ctrs changed after locking */
 532         ulong_t page_ctrs_changed[MMU_PAGE_SIZES][MAX_MNODE_MRANGES];
 533         /* page_freelist_coalesce failed */
 534         ulong_t page_ctrs_failed[MMU_PAGE_SIZES][MAX_MNODE_MRANGES];
 535         ulong_t page_ctrs_coalesce_all; /* page coalesce all counter */
 536         ulong_t page_ctrs_cands_skip_all; /* candidates useful for all func */
 537         ulong_t restrict4gcnt;
 538         ulong_t unrestrict16mcnt;       /* non-DMA 16m allocs allowed */
 539         ulong_t pgpanicalloc;           /* PG_PANIC allocation */
 540         ulong_t pcf_deny[MMU_PAGE_SIZES];       /* page_chk_freelist */
 541         ulong_t pcf_allow[MMU_PAGE_SIZES];
 542 };
______unchanged_portion_omitted_
```

```
**********************************************************
   30541 Fri Jan 29 15:08:08 2016
new/usr/src/uts/sun4/vm/vm_dep.h
5461 #pragma align before the declaration
**********************************************************
______unchanged_portion_omitted_

 695 /* allocation size to ensure vm_cpu_data_t resides in its own cache line */
 696 #define VM_CPU_DATA_PADSIZE                                             \
 697         (P2ROUNDUP(sizeof (vm_cpu_data_t), L2CACHE_ALIGN_MAX))

 699 /* for boot cpu before kmem is initialized */
 700 extern char     vm_cpu_data0[];

 699 /*
 700  * Function to get an ecache color bin: F(as, cnt, vcolor).
 701  * the goal of this function is to:
 702  * - to spread a processes' physical pages across the entire ecache to
 703  *      maximize its use.
 704  * - to minimize vac flushes caused when we reuse a physical page on a
 705  *      different vac color than it was previously used.
 706  * - to prevent all processes to use the same exact colors and trash each
 707  *      other.
 708  *
 709  * cnt is a bin ptr kept on a per as basis.  As we page_create we increment
 710  * the ptr so we spread out the physical pages to cover the entire ecache.
 711  * The virtual color is made a subset of the physical color in order to
 712  * in minimize virtual cache flushing.
 713  * We add in the as to spread out different as.  This happens when we
 714  * initialize the start count value.
 715  * sizeof(struct as) is 60 so we shift by 3 to get into the bit range
 716  * that will tend to change.  For example, on spitfire based machines
 717  * (vcshft == 1) contigous as are spread bu ~6 bins.
 718  * vcshft provides for proper virtual color alignment.
 719  * In theory cnt should be updated using cas only but if we are off by one
 720  * or 2 it is no big deal.
 721  * We also keep a start value which is used to randomize on what bin we
 722  * start counting when it is time to start another loop. This avoids
 723  * contigous allocations of ecache size to point to the same bin.
 724  * Why 3? Seems work ok. Better than 7 or anything larger.
 725  */
 726 #define PGCLR_LOOPFACTOR 3

 728 /*
 729  * When a bin is empty, and we can't satisfy a color request correctly,
 730  * we scan.  If we assume that the programs have reasonable spatial
 731  * behavior, then it will not be a good idea to use the adjacent color.
 732  * Using the adjacent color would result in virtually adjacent addresses
 733  * mapping into the same spot in the cache.  So, if we stumble across
 734  * an empty bin, skip a bunch before looking.  After the first skip,
 735  * then just look one bin at a time so we don't miss our cache on
 736  * every look. Be sure to check every bin.  Page_create() will panic
 737  * if we miss a page.
 738  *
 739  * This also explains the '<=' in the for loops in both page_get_freelist()
 740  * and page_get_cachelist().  Since we checked the target bin, skipped
 741  * a bunch, then continued one a time, we wind up checking the target bin
 742  * twice to make sure we get all of them bins.
 743  */
 744 #define BIN_STEP        20

 746 #ifdef VM_STATS
 747 struct vmm_vmstats_str {
 748         ulong_t pgf_alloc[MMU_PAGE_SIZES];      /* page_get_freelist */
 749         ulong_t pgf_allocok[MMU_PAGE_SIZES];
 750         ulong_t pgf_allocokrem[MMU_PAGE_SIZES];
```



```
 751         ulong_t pgf_allocfailed[MMU_PAGE_SIZES];
 752         ulong_t pgf_allocdeferred;
 753         ulong_t pgf_allocretry[MMU_PAGE_SIZES];
 754         ulong_t pgc_alloc;                      /* page_get_cachelist */
 755         ulong_t pgc_allocok;
 756         ulong_t pgc_allocokrem;
 757         ulong_t pgc_allocokdeferred;
 758         ulong_t pgc_allocfailed;
 759         ulong_t pgcp_alloc[MMU_PAGE_SIZES];     /* page_get_contig_pages */
 760         ulong_t pgcp_allocfailed[MMU_PAGE_SIZES];
 761         ulong_t pgcp_allocempty[MMU_PAGE_SIZES];
 762         ulong_t pgcp_allocok[MMU_PAGE_SIZES];
 763         ulong_t ptcp[MMU_PAGE_SIZES];           /* page_trylock_contig_pages */
 764         ulong_t ptcpfreethresh[MMU_PAGE_SIZES];
 765         ulong_t ptcpfailexcl[MMU_PAGE_SIZES];
 766         ulong_t ptcpfailszc[MMU_PAGE_SIZES];
 767         ulong_t ptcpfailcage[MMU_PAGE_SIZES];
 768         ulong_t ptcpok[MMU_PAGE_SIZES];
 769         ulong_t pgmf_alloc[MMU_PAGE_SIZES];     /* page_get_mnode_freelist */
 770         ulong_t pgmf_allocfailed[MMU_PAGE_SIZES];
 771         ulong_t pgmf_allocempty[MMU_PAGE_SIZES];
 772         ulong_t pgmf_allocok[MMU_PAGE_SIZES];
 773         ulong_t pgmc_alloc;                     /* page_get_mnode_cachelist */
 774         ulong_t pgmc_allocfailed;
 775         ulong_t pgmc_allocempty;
 776         ulong_t pgmc_allocok;
 777         ulong_t pladd_free[MMU_PAGE_SIZES];     /* page_list_add/sub */
 778         ulong_t plsub_free[MMU_PAGE_SIZES];
 779         ulong_t pladd_cache;
 780         ulong_t plsub_cache;
 781         ulong_t plsubpages_szcbig;
 782         ulong_t plsubpages_szc0;
 783         ulong_t pfs_req[MMU_PAGE_SIZES];        /* page_freelist_split */
 784         ulong_t pfs_demote[MMU_PAGE_SIZES];
 785         ulong_t pfc_coalok[MMU_PAGE_SIZES][MAX_MNODE_MRANGES];
 786         ulong_t ppr_reloc[MMU_PAGE_SIZES];      /* page_relocate */
 787         ulong_t ppr_relocok[MMU_PAGE_SIZES];
 788         ulong_t ppr_relocnoroot[MMU_PAGE_SIZES];
 789         ulong_t ppr_reloc_replnoroot[MMU_PAGE_SIZES];
 790         ulong_t ppr_relocnolock[MMU_PAGE_SIZES];
 791         ulong_t ppr_relocnomem[MMU_PAGE_SIZES];
 792         ulong_t ppr_krelocfail[MMU_PAGE_SIZES];
 793         ulong_t ppr_copyfail;
 794         /* page coalesce counter */
 795         ulong_t page_ctrs_coalesce[MMU_PAGE_SIZES][MAX_MNODE_MRANGES];
 796         /* candidates useful */
 797         ulong_t page_ctrs_cands_skip[MMU_PAGE_SIZES][MAX_MNODE_MRANGES];
 798         /* ctrs changed after locking */
 799         ulong_t page_ctrs_changed[MMU_PAGE_SIZES][MAX_MNODE_MRANGES];
 800         /* page_freelist_coalesce failed */
 801         ulong_t page_ctrs_failed[MMU_PAGE_SIZES][MAX_MNODE_MRANGES];
 802         ulong_t page_ctrs_coalesce_all; /* page coalesce all counter */
 803         ulong_t page_ctrs_cands_skip_all; /* candidates useful for all func */
 804 };
______unchanged_portion_omitted_
```