International Journal of Scientific & Engineering Research, Volume 4, Issue 7, July-2013
ISSN 2229-5518

A STUDY ON THE EFFECT OF CACHE MEMORY IN COMPUTING ENVIRONMENT WITH PERFORMANCE ANALYSIS

S.V.SRIDHAR, DAYAKAR KONDAMUDI , HARISH KUMAR KOTHA , N.VAMSI KRISHNA

ABSTRACT: Processors are generally able to perform operations on operands faster than the access time of large capacity main memory. Though semiconductor memory which can operate at speeds comparable with the operation of the processor exists, it is not economical to provide all the main memory with very high speed semiconductor memory. The problem can be alleviated by introducing a small block of high speed memory called a *cache* between the main memory and the processor. The idea of cache memories is similar to virtual memory in that some active portion of a low-speed memory is stored in duplicate in a higher-speed cache memory. When a memory request is generated, the request is first presented to the cache memory, and if the cache cannot respond, the request is then presented to main memory.

Index Terms—memory, access, hit, miss. associative , cache , processor , data, address , performance , indexing , set ,

I.INTRODUCTION:

Why is this high speed memory necessary or beneficial? In today's systems , the time it takes to bring an instruction (or piece of data) into the processor is very long when compared to the time to execute the instruction. For example, a typical access time for DRAM is 60ns. A 100 MHz processor can execute most instructions in 1 CLK or 10 ns. Therefore a bottle neck forms at the input to the processor. Cache memory helps by decreasing the time it takes to move information to and from the processor. A typical access time for SRAM is 15 ns. Therefore cache memory allows small portions of main memory to be accessed 3 to 4 times faster than DRAM (main memory).

How can such a small piece of high speed memory improve system performance? The theory that explains this performance is called "Locality of Reference." The concept is that at any given time the processor will be accessing memory in a small or localized region of memory. The cache loads this region allowing the processor to access the memory region faster. How well contains over 90% of the addresses requested by the processor. This means that over 90% of So now the question, why not replace main memory DRAM with SRAM? The main reason is cost. SRAM is several times more expense than DRAM. Also, SRAM consumes more power and is less dense than DRAM. Now that the reason for cache has been established, let look at a simplified model of a cache system.

I.I FUNCTIONING and PERFORMANCE:

The performance of a cache can be quantified in terms of the hit and miss rates, the cost of a hit, and the miss penalty, where a cache hit is a memory access that finds data in the cache and a cache miss is one that does not. When reading, the cost of a cache hit is roughly the time to access an entry in the cache. The miss penalty is the additional cost of replacing a cache line with one containing the desired data.

Note that the approximation is an underestimate - control costs have been left out. Also note that only one word is being loaded from the faster memory while a whole cache block's worth of data is being loaded from the slower memory.

Since the speeds of the actual memory used will be improving 'independently", most effort in ache design is spent on fast control and decreasing the miss rates. We can classify misses into three categories, compulsory misses, capacity misses and conflict misses. Compulsory misses are when data is loaded into the cache for the first time (e.g. program start-up) and are unavoidable. Capacity misses are when data is reloaded because the cache is not large enough to hold all the data no matter how we organize the data (i.e. even if we changed the hash function and made it omniscient). All other misses are conflict misses - there is

Theoretically enough space in the cache to avoid the miss but our fast hash function caused a miss anyway.Which is the fetching a block when it is needed and is not already in the cache, i.e. to fetch the required block on a miss.This strategy is the simplest and requires no additional hardware or tags in the cache recording the references, except to identify the block in the cache to be replaced.

Prefetch

Which is fetching blocks before they are requested. A simple prefetch strategy is to prefetch the (*i*+1)th block when the *i*th block is initially referenced on the expectation that it is likely to be needed if the *i*th block is needed. On the simple prefetch strategy, not all first references will induce a miss, as some will be to prefetched blocks.

Selective fetch:

Which is the policy of not always fetching blocks, dependent upon some defined criterion, and in these cases using the main memory rather than the cache to hold the information. For example, shared writable data might be easier to maintain if it is always kept in the main memory and not passed to a cache for access, especially in multi-processor systems. Cache systems need to be designed so that the processor can access the main memory directly and bypass the cache. Individual locations could be tagged as non-cacheable.

I.2 PENTIUM CACHE : A CASE STUDY

This section examines internal cache on the Pentium(R) processor. The purpose of this section is to describe the cache scheme that the Pentium(R) processor uses and to provide an overview of how the Pentium(R) processor maintains cache consistency within a system.

The above section broke cache into neat little categories. However, in actual implementations, cache is often a series of combinations of all the above mentioned categories. The concepts are the same, only the boundaries are different.

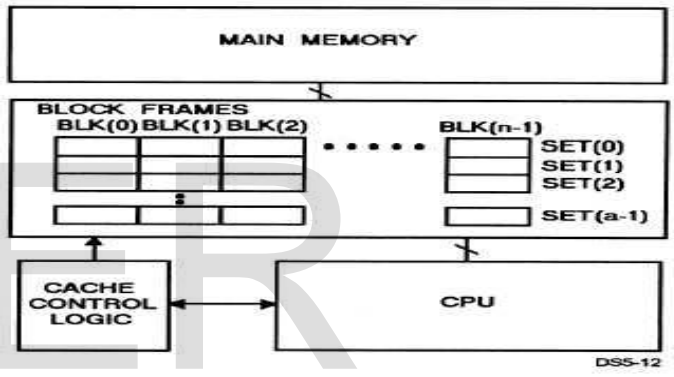
Pentium(R) processor cache is implemented differently than the systems shown in the previous examples. The first difference is the cache system is internal to the processor, i.e. integrated

helping to reduce the overall cost of the system. Another advantage is the speed of memory request responses. For example, a 100MHz Pentium(R) processor has an external bus speed of 66MHz. All external cache must operate at a maximum speed of 66mhz. However, an internal cache operates at 100MHz. Not only does the internal cache respond faster, it also has a wider data interface. An external interface is only 64-bits wide while the internal interface between the cache and processor prefetch buffer is 256-bits wide. Therefore, a huge increase in performance is possible by integrating the cache into the CPU.

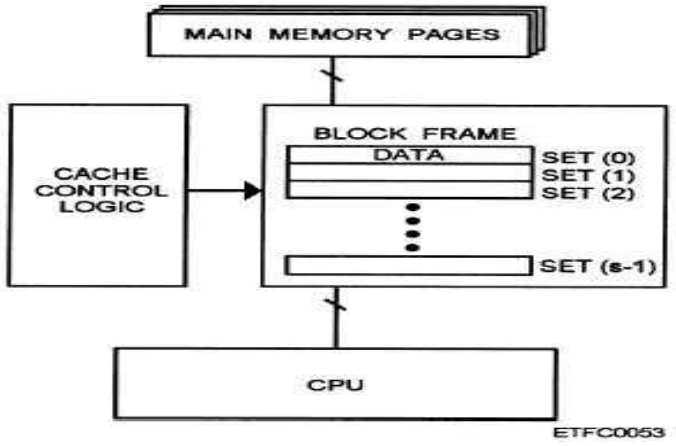
A third difference is that the cache is divided into two separate pieces to improve performance - a data cache and a code cache, each at 8K.. This division allows both code and data to readily cross page boundaries without having to overwrite one another.

The three different types of mapping used for the purpose of cache memory are as follow, Associative mapping, Direct mapping and Set-Associative mapping.

Associative mapping: In this type of mapping the associative memory is used to store content and addresses both of the memory word. This enables the placement of the any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.

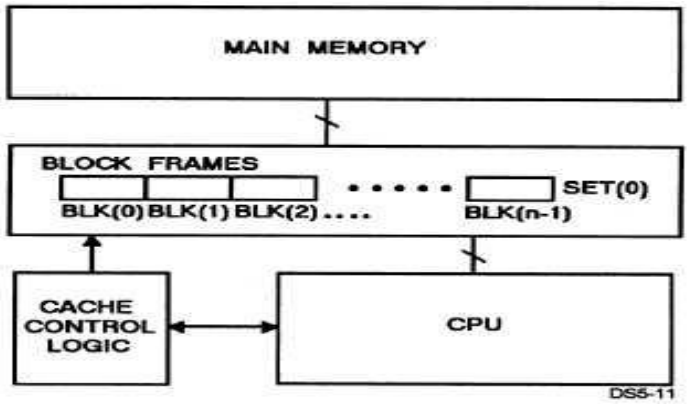


- Direct mapping: In direct mapping the RAM is made use of to store data and some is stored in the cache. An address space is split into two parts index field and tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.



- Set-associative mapping: This form of mapping is a modified form of the direct mapping where the

disadvantage of direct mapping is removed. Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address.



If the particular address is found in the cache, the block of data is sent to the CPU, and the CPU goes about its operation until it requires something else from memory. When the CPU finds what it needs in the cache, a hit has occurred. When the address requested by the CPU is not in the cache, a miss has occurred and the required address along with its block of data is brought into the cache according to how it is mapped. Cache processing in some computers is divided into two sections: **main** cache and **eavesdrop** cache. Main cache is initiated by the CPU within. Eavesdrop is done when a write to memory is performed by another requestor (other CPU or IOC). Eavesdrop searches have no impact on CPU performances.

II .CACHE MAPPING TECHNIQUES.—

Cache mapping is the method by which the contents of main memory are brought into the cache and referenced by the CPU. The mapping method used directly affects the performance of the entire computer system. new Word Press blog journalism server running on Sles 11 under VMware 3.5 U3. The server surprisingly only had a dozen users, surprising since a commercial Word Press provider i talked to had up to 50,000 hits per day and dozens of users on a box with 2 gigs of memory and no problems. Chances are its a memory leak, probably from a Word Press plug-in that's causing all the problems, however being a linux server there's other ways to manage the memory. The following is written for my staff to help bring them upto speed on memory and its troubleshooting.

III. PRACTIAL PERFORMANCE EVALUATION

Start by checking your server has enough memory, if processes are dying unexpectedly have a look at your /var/log/messages file and see if you are running out of memory or if processes are being killed of due to lack of memory.

I normally use the free command first to see how memory is being used, i like to use the –m flag to have the output formatted in megs to simply reading the information, e.g.:

[Server]

	total	used	free	shared	buffers	cached
Mem:	3777	3516	260	0	228	2921
-/+ buffers/cache:		366	3410			
Swap:	2055	0	2055			

I could go over the output in depth however there's a really easy way to understand what's happening, just look at the line:

	Used	Free
-/+ buffers/cache:	366	3410

The first value is how much memory is being used and the second value is how much memory can be freed for use by applications. As long as you have memory that can be used by applications you're generally fine. Another aspect to note is the output is the swap file:

	Total	Used	Free
Swap:	2055	0	2055

Swapping generally only occurs when memory usage is impacting performance, unless you manually change its aggressiveness, more on that later.

Swap

If your server is heavily using swap things are bad, you're running out of memory. The exception to this is where you have a distro with cache problems and may well decide to max swapiness to reduce the problems cache created. To find the space dedicated to swap type:

more /proc/swaps

```
<<-PRODUCTION->> :~ # more /proc/swaps
Filename                        Type            Size    Used    Priority
/dev/sda4                       partition       2104504 0       -1
```

To find your current level of swapiness type:

cat /proc/sys/vm/swappiness

```
<<-PRODUCTION->> :~ # cat /proc/sys/vm/swappiness
60
```

The default value is 60. However different systems require different levels of swapiness, a server is not the same as home computer. The value ranges between 0 and 100. At 100 the server will try and swap inactive pages, at 0 applications that want ram will shrink the ram to a tiny fraction of cache, i.e. 0 less likely to swap, 100 very likely. You can change the value by echoing a new one to the /proc/sys/vm/swapiness file,

e.g. echo 10 > /proc/sys/vm/swapiness

To change the default level on boot edit the /etc/sysctl.conf file (since kernel 2.6)

e.g. vm.swappiness = 10

IV. MEMORY PROCESS ALLOCATION

Along with other aspects of the server, Virtual memory statistics can be reported with vmstats, its main use for memory diagnosis is that it reports page-ins and page-outs as they happen. The best way to see this is by delaying the output of vmwstat and it comes with options to do this, otherwise it just reports averages since the last boot. State the delay in seconds after the command followed by the number of updates you wish to use, e.g. vmstat 2 4 runs vmstat with a 2 second delay with 4 updates and so on e.g.

```
<<-PRODUCTION->> :~ # vmstat 2 4
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 155612 205936 3139932    0    0     4    20    5    3  1  1 98  0  0
 0  0      0 155612 205936 3139932    0    0     0     0  257   46  0  0 100  0  0
 0  0      0 152488 205936 3139940    0    0     0     0  287  145  2  2 96  0  0
 0  0      0 150100 205944 3139940    0    0     0    50  267   77  1  1 98  0  0
```

read the man for detailed info if need be, otherwise just look at:

free – free memory
si – page ins
so – page outs

Page ins are expected e.g. when starting an application and its information is paged in Regular page outs are not wanted, occasional page outs are expected as the kernel frees up memory. If page outs occure so often the server is spending more time managing paging than running apps performance suffers, this is refered to as thrashing. At this point you could use top and ps to identify the processes that are causing problems.

To see where all your memory is going the easiest way is to use the top command, then press m to sort by memory, press q or crtl+c to exit the top screen.

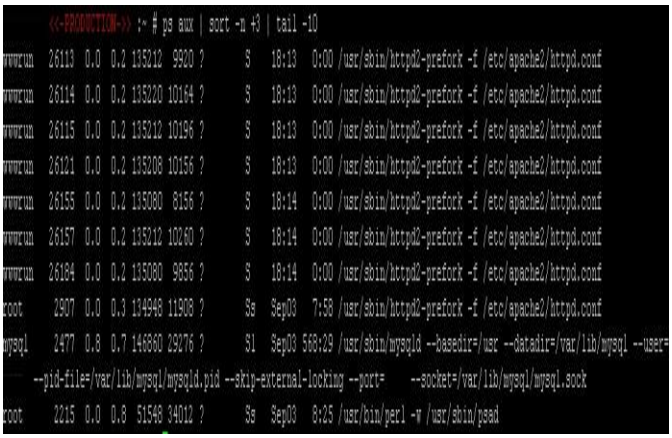
```
top - 18:06:23 up 45 days, 22:49,  1 user,  load average: 0.02, 0.11, 0.06
Tasks:  80 total,   2 running,  78 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.0%us,  0.3%sy,  0.0%ni, 98.6%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3868296k total,  3601640k used,   266656k free,   234172k buffers
Swap:  2104472k total,        0k used,  2104472k free,  2992424k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2215 root      15   0 51420  33m 1832 S  0.0  0.9   7:44.61 psad
 2477 mysql     16   0  134m  26m 4916 S  0.0  0.7 530:31.58 mysqld
 2907 root      16   0  131m  11m 5080 S  0.0  0.3   7:14.53 httpd2-prefork
11120 wwwrun    15   0  133m  11m 4224 S  0.7  0.3   0:00.08 httpd2-prefork
11351 wwwrun    16   0  133m  11m 4180 S  0.0  0.3   0:00.03 httpd2-prefork
11473 wwwrun    16   0  132m  11m 4180 S  0.0  0.3   0:00.03 httpd2-prefork
10808 wwwrun    15   0  132m  10m 3324 S  0.0  0.3   0:00.08 httpd2-prefork
10559 wwwrun    16   0  131m  10m 3304 S  0.0  0.3   0:00.10 httpd2-prefork
11112 wwwrun    16   0  131m  10m 3280 S  0.0  0.3   0:00.06 httpd2-prefork
11364 wwwrun    16   0  131m  10m 3264 S  0.0  0.3   0:00.05 httpd2-prefork
11511 wwwrun    15   0  131m  10m 3172 S  0.0  0.3   0:00.01 httpd2-prefork
11509 wwwrun    16   0  131m  10m 3172 S  0.0  0.3   0:00.02 httpd2-prefork
11282 wwwrun    16   0  131m  10m 3160 S  0.0  0.3   0:00.03 httpd2-prefork
11361 wwwrun    16   0  131m  10m 3140 S  0.0  0.3   0:00.00 httpd2-prefork
11512 wwwrun    16   0  131m  10m 3120 S  0.0  0.3   0:00.01 httpd2-prefork
11603 wwwrun    16   0  131m  9m 3112 S  0.0  0.3   0:00.01 httpd2-prefork
11362 wwwrun    16   0  131m  9m 3128 S  0.0  0.3   0:00.00 httpd2-prefork
```

For more detailed information you can always use ps aux and see which process are using memory and how much. Apache and mysql are normally top users, along with psad for busy web servers.

To sort the output of ps by memory you are supposed to be able to use : ps aux –sort pmem
however i find this does not work on all flavours on linux so i prefer to use the sort command to sort by memory usage order : *ps aux | sort –n +3*

Then if i just want to look at the top 10 memory hogs or the top memory hog i do a further pipe and use the tail command, e.g. to find the 10 highest memory consuming process: *ps aux | sort –n +3 | tail -10*



If you want to monitor a processes memory usage then look at the pid for the process and setup a cron job to pipe the output of the command ps ev –pid=<PID> to a file you can check later. If you want to check memory usage change straight away keep entering the command: *ps ev –pid=<PID>*

Once you know the process that is responsible for the memory problems you can optimise it, or kill it. Here's a few common tricks for processes that can use a lot of memory

Java

Java memory heaps need a limit to their sizes set by passing a –Xmx option else the heap increases until you're out of memory. Custom Java apps should be able to use the java command line –XmxNNm. NN = number of megs. With JBoss and Tomcat check the settings in your relevant JBoss (48m to 160m recommended) or Tomcat files (48m to 96m recommended).

A rough way to work out the largest size you can set is to stop the java process's then look at the free –m output for buffers as shown earlier and subtract the used from the free to allow for unexpected memory usage, the resultant number is the max memory you could set.

However keep in mind these are just guidelines, It's up to you to decide how high to set the memory limit for the heap since only you really know how much memory you have on the server and how memory the java process needs.

Apache

Apache when it loads starts multiple servers and distributes the traffic amongst these 'servers', the memory usage can grow large as each loads libraries for php and perl. You can adjust the number spawned with the settings:

StartServers
MinSpareServers
MaxSpareServers

These are in the httpd file. However depending on the distro you might need to adjust the prefork values, google for your os. The maxclients value can be worked out by finding out the memoty usage of the largest apache client, stopping apache, looking at free memory and dividing by the free memory by the memory usage size of the largest apache thread. Apache has default configuration for small, medium and large servers. For many of you out there hosting your own low traffic site you'll get better performance used the settings optimised for small servers.

SQL

However in some cases the problem is down to the cache.

Reducing cached memory

Linux memory management tries to minimise disk access. To do this it will use any unused ram to cache, this is because reading from disk is slow compared to reading from memory. When the cache is used up the data that has been there the longest is freed, theoretically data that is used often will not be removed whilst data that is no longer needed slowly gets moved out of the cache. When an application needs memory the kernel should reduce the size of the cache and free up memory. This is why people sometimes get confused when using the free command, since linux uses memory for cache it can appear to the untrained eye that most of the memory has been used up. This is in fact normal; it's when the server can no longer free memory from the cache that problems occur.

Freeing cache memory therefore does not usually make your computer faster, but the converse, linux becomes slower having to re read information to the cache. Ironic then that some of the latest distro's of linux, namely SUSE and Mandriva seem to have forgotten this, there are numerous reports of these, and other linux distro's, deciding cached memory is too important to free up for actual processes. Luckily a solution was added in kernel 2.6.16 allowing us to free cached memory by writing to /proc/sys/vm/drop_caches. There are three options depending on what you need to do, clean the cache, free dentries and inodes, and free cache, dentries and inodes, we run sync first to ensure all cached objects are freed as this is a non-destructive operation and dirty objects are not freed:Tofree cache enter: sync; echo 1 > /proc/sys/vm/drop_caches dentries and inodes *:* sync; echo 2 > /proc/sys/vm/drop_caches pagecache, dentries and inodes*:* sync; echo 3 > /proc/sys/vm/drop_caches You can automate these in a cron job e.g. hourly if you have the

misfortune to use a distro with problems. Another issue with cache is that if you copy a large amount of data, e.g. a file tree, the copied data will end up in the cache flushing out your existing cache. There is an interesting article on improving linux performance by selectively preserving cache state at:

V. OOM – 32 bit system memory problems (64 bit safe)

If you are running 32 bit linux and have enough memory then you might be a victim of the out of memory (oom) killer. However in 64 bit linux all memory is low memory so you are safe from Oom, and out of memory errors are really down to out of memory problems!

SOLUTION:

Oom problems can be easily solved by:
running the hugemem kernel
editing /etc/sysctl.conf with the below line to make the kernel more aggressive about recovering low memory:
vm.lower_zone_protection = 250 or finally
editing /etc/sysctl.conf to disable oom on boot with the line:
vm.oom-kill = 0

CAUSE:

Oom kills processes on servers even when there is a large amount of memory free. Oom problems are caused by low memory exhaustion. Systems that are victim to Oom suffer more as memory is increased since they have kernels where memory allocation is tracked using low memory, so the more memory you have the more low memory is used up and the more you have problems. When low memory starts running out Oom starts killing processes to keep memory free!

VI.DIAGNOSIS

To check low and high memory usage, use the command lines below, though the info is from a 64 bit system since I'm sensible J

[Server] <<-PRODUCTION->> :~ # egrep 'High|Low'

/proc/meminfo

HighTotal: 0 kB

HighFree: 0 kB

LowTotal: 3868296 kB

LowFree: 271872 kB

[Server] <<-PRODUCTION->> :~ # free -lm

	total	used	free	shared	buffers	cached
Mem:	3777	3512	265	0	228	2919
Low:	3777	3512	265			
High:	0	0	0			
-/+ buffers/cache:		364	3413			
Swap:	2055	0	2055			

VII. DETAILED MEMORY INFORMATION

To obtain detailed memory information type cat /proc/meminfo *e.g.:*

I was going to type something up when i found a nice explanation on red hats site which i've quoted and amended where relevant below:

The information comes in the form of both high-level and low-level statistics. First we will discuss the high-level statistics

```
<<-PRODUCTION->> :/ # cat /proc/meminfo
MemTotal:      2075592 kB
MemFree:        202368 kB
Buffers:        280500 kB
Cached:        1376220 kB
SwapCached:          0 kB
Active:        1045228 kB
Inactive:       680636 kB
HighTotal:     1179588 kB
HighFree:       142252 kB
LowTotal:       896004 kB
LowFree:         60116 kB
SwapTotal:     2104504 kB
SwapFree:      2104504 kB
Dirty:             132 kB
Writeback:           0 kB
AnonPages:       68676 kB
Mapped:          17940 kB
Slab:           138344 kB
CommitLimit:   3142300 kB
Committed_AS:   327932 kB
PageTables:        924 kB
VmallocTotal:  114680 kB
VmallocUsed:     7380 kB
VmallocChunk:  107148 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
Hugepagesize:     4096 kB
```

High-Level Statistics

```
MemTotal:      2075592 kB
MemFree:        202368 kB
Buffers:        280500 kB
Cached:        1376220 kB
SwapCached:          0 kB
```

MemTotal: Total usable ram (i.e. physical ram minus a few reserved bits and the kernel binary code)
MemFree: Is sum of LowFree+HighFree (overall stat)
Buffers: Memory in buffer cache. mostly useless as metric nowadays
Cached: Memory in the pagecache (diskcache) minus SwapCache
SwapCache: Memory that once was swapped out, is swapped back in but still also is in the swapfile (if memory is needed it doesn't need to be swapped out AGAIN because it is already in the swapfile. This saves I/O)

Detailed Level Statistics

VM Statistics

```
Active:        1045228 kB
Inactive:       680636 kB
```

VM splits the cache pages into "active" and "inactive" memory. The idea is that if you need memory and some cache needs to be sacrificed for that, you take it from inactive since that's expected to be not used. The vm checks what is used on a regular basis and moves stuff around.

When you use memory, the CPU sets a bit in the pagetable and the VM checks that bit occasionally, and based on that, it can move pages back to active. And within active there's an order of "longest ago not used" (roughly, it's a little more complex in reality). The longest-ago used ones can get moved to inactive. Inactive is split into two in the above kernel (2.4.18-24.8.0). Some have it three.

Active: Memory that has been used more recently and usually not reclaimed unless absolutely necessary. Inactive — The total amount of buffer or page cache memory, in kilobytes, that are free and available. This is memory that has not been recently used and can be reclaimed for other purposes.

4 Conclusion

While cache size only had a limited impact on the synthetic benchmarks such as PCMark05, the performance difference in most real-life benchmarks was significant. This was surprising at first, because experience tells us that performance differences can typically be found in most synthetic benchmarks, while little of it is eventually reflected in real-life benchmarks

From this perspective, upgrading the L2 cache from up to 4 MB to a maximum of 6 MB for the upcoming 45-nm dual core Penryn processors (Core 2 Duo E8000 series) makes a lot of sense. Not only does the shrink from 65 to 45 nm give Intel more headroom to increase the cache size, but the company will again offer more performance thanks to the increased cache size. However, the most important benefit is due to how Intel can offer more processor variants with6 MB, 4 MB, 2 MB or even 1 MB L2 cache. In doing so, Intel utilizes an even higher percentage of the dies on a wafer despite some scattered defects that might have forced Intel to throw dies away in the past. Large cache sizes seem to be both important for both performance and Intel's balance sheet.

When a request is made of the system the CPU requires instructions for executing that request. The CPU works many times faster than system RAM, so to cut down on delays, L1 cache has bits of data at the ready that it anticipates will be needed. L1 cache is very small, which allows it to be very fast. If the instructions aren't present in L1 cache, the CPU checks L2, a slightly larger pool of cache, with a little longer latency. With each *cache miss* it looks to the next level of cache. L3 cache can be far larger than L1 and L2, and even though it's also slower, it's still a lot faster than fetching from RAM.

5 References

[1] K. Hwang, Advanced Computer Architecture, McGraw-Hill, 1993.

[2] J.L. Hennessy and D.A. Patterson, Computer Architecture A Quantitative Approach, Second Ed.,Morgan Kaufmann, CA, 1996.

[3] M.D. Hill, Aspect of Cache Memory and Instruction Buffer Performance, Ph.D. Thesis, University of Calif. at Berkeley, Computer Science Division, Tech. Rep. UCB/CSD 87/381, 1987.

[4] N.P. Jouppi, "Improving direct mapped cache performance by the addition of a small fully associative

[5]cache and prefetching buffers," Proc. 17th Annual Int'l Symposium on Computer Architecture, Seattle, pp