Critique of

“

‘Optimising the Idle Task and Other MMU Tricks’
by Dougan, Mackerras and Yodaiken.

”

by anonymous
11/11/1999

This paper presents a series of modifications made to Linux running on the PowerPC platform, specifically the PPC 603 and 604 chips. These are claimed to improve the general performance of Linux. The paper contains some good ideas, especially in its early stages, however the quality of the report decreases as it proceeds.

There are four main modifications:

- Reducing the frequency of TLB misses.
- Reducing the cost of TLB misses.
- Reducing the cost of TLB invalidations.
- Reducing the cost of the Linux ‘idle task’ (in cache terms), which clears free pages when the system has nothing else to do.

The term TLB above refers to both hardware and software TLBs.

This critique is split into three sections, which highlight notable aspects of the paper. These sections are entitled ‘The Good’, ‘The Bad’ and ‘The Ugly’. They are ordered in accordance with those of the paper.

The Good

The paper contains many aspects of a an excellent piece of research; some of these are observed here.

The abstract identifies the authors’ focus on optimising the critical path, which is a sensible place to direct optimisation efforts. For the most part this ideal is followed, especially in the work on improving TLB hit rates. Also the avoidance of memory accesses by using the PPC’s special-purpose kernel registers to save general-purpose registers in the TLB refill handler is nice.

The justification at the end of section 2 for these optimisations is also worthy — that TLBs aren’t getting much bigger so a greater understanding of them is the way forward.

Unfortunately, the paper as a whole is not exactly what this abstract leads us to believe — there is not any wonderful cache magic, such as trying to make time-local code/data also space-local (thus reducing TLB and cache entries), or preloading the cache with data that will be needed later. These ideas are relegated to the ‘Future Work’ section.

The paper makes the general point that page table structures should be flexible to exploit the characteristics of different OS's, and to allow experimentation with new page table structures. This means that chip hardware should not attempt to access page tables, but rather leave TLB reloading to software. This point is argued well and convincingly — although in the conclusion a trend towards hardware-loaded TLBs is identified, which is not a general industry trend outside the PPC platform. The fact that the 603 uses a software-loaded TLB whereas the 604 uses hardware allowed a great deal of meaningful comparison between the two approaches.

The summary of the PPC hardware page-table lookup in section 3 was concise and agreed with my understanding of it (I have some experience with the PPC platform). However, it would have been helpful to mention that both reference and dirty bits on PTEs are maintained by the hardware, as this is assumed later.

Shortly after this the paper remarks on the need for a backup page-table, when all entries cannot fit into the required hashed page table. This makes the hashed page table a (slow) form of second-level TLB, commonly called a software TLB (although here of course it is implemented in hardware). The desirability of this situation concurs with other research, such as clustered page tables (referenced) or guarded page tables (not referenced)

The decision to retain compatibility with the existing Linux source tree greatly improves the importance of this paper — it is not solely an exercise in optimisation, but rather the improvements made will be useful and transportable as the operating system on which they are based progresses.

The tuning of the PPC's VSIDs (similar to MIPS's ASIDs) to improve page-table hit rates is certainly a worthy concept, and although the process is not detailed in the paper, it appears to be a novel idea. However, the later incrementing of a process's VSID to invalidate its memory context is perhaps not the best approach — choosing another VSID that hashed to the same buckets for the same virtual addresses would likely produce less 'evicts'. The hash function used is not trivial but it certainly isn't MD5 — constructing equivalent VSIDs (especially for small hash tables) is definitely possible.

The Bad

This section contains things which, while they did not have a fatal effect on the paper, could certainly have been done better.

Firstly, the narrow choice of benchmarks (admitted at the start) compromises the claims of the paper. Using only the lmbench suite (with no reference to the work of Brown and Seltzer who question their global applicability) and an informal 'kernel compile' test, there are significant holes in their formal measurements. In particular, applications which do lots of I/O (such as a web server), and everyday programs with more sporadic TLB usage, are not represented.

The complete focus on the PPC architecture limits the generality of this research, despite claims at the start. In particular, arguments based on the exact structure of the hardware-accessed hashed page tables are not generalisable. Also the use of very large (24-bit) VSIDs is not common (compare MIPS 8-bit ASIDs). The properties of the BAT registers are however related to superpages, segments, and similar concepts on

other architectures, so work using these has greater general applicability.

The references section could probably contain a great deal more. In the related work section, the paper claims that ‘There has been surprisingly little published experimental work on OS memory management design’. I find this statement difficult to believe, in light of such papers as ‘GPTs on MIPS R4600…’ by J. Liedtke and K. Elphinstone in 1996, as well as a number of other papers some not related to µ-kernels present in the bibliography of the Adv. OS lecture on page tables. I also find it hard to believe that concepts such as a lazy 2nd-level TLB flush do not have immediate analogues in other caches, although this is not acknowledged by the authors.

The results of the different optimisations are presented in the paper irrespective of each other. However, since there is such a reliance on both the system being optimised (Linux) and the hardware platform it is running on (PPC), a further analysis of the interactions and results when all the modifications are combined is necessary. This is not done adequately.

The tone of the report is inconsistent. Judging by the introduction to PPC memory management, only a general knowledge of that architecture and operating systems seems to be assumed. However many later topics are referenced without an introduction, and the meaning of the topic only becomes apparent having read a few sentences on. This is annoying and unnecessary — some examples are the Linux ‘get_free_frame’ function, the purpose of its idle task, and the PPC ‘dcbz’ instruction (which sets up and clears a data cache line without any memory accesses, so it can be written back later when it is replaced — a faster form of clearing memory. The instruction is not even named so readers cannot easily look it up)

The Ugly

This section contains failings rather more serious to the use and understanding of the paper.

There is an absence or bad presentation of results throughout the paper. Hard data is not given where needed, and there is no collation of results, or detailed breakdown. The paper contains only three tables.

An example of this occurs in section six. At the bottom of page four, the text talks about the 603 ‘keeping pace’ with the more ample 604. The table overleaf does not show this.

In section 7 on page 5 mention is made of a ratio of ‘hash table reloads to evicts’, with the value of 90%. Since an evict **is** a hash table reload, this does not make sense until the following page, where it is defined around the other way. Page 5 also says that hash table usage finally ends up at 5% from 15%, whereas the better ‘final’ result is actually the 15% usage. Halfway through the first column of page 6, the paper starts reading as if the rest of section 7 did not exist. Also the table at the top of page 6 compares a 603(lazy) with a 604(tune), when they should both be (lazy). On a higher level, the conclusions drawn by section 7 highlight more problems with narrow benchmarks.

At the end of section 8 there are no results to justify the assertions of ‘cache misuse’, but strong qualifications are used ‘This conflict is non-productive and causes many unnecessary cache misses’

At the end of section 9, there are no final results for the overall improvements made by not caching writes of zeros to pages in the idle task, even though it is asserted that it incurs no overhead when the results are not used.

And at the end of the paper itself, where some detailed examination could be expected in the Analysis section, only a conclusion is found there.

The basis for section 8 (which has no results, only assertions) is also suspect. It calculates there there will be 18 new cache entries created for each hash table miss. This may be so, but the relevant quantum when dealing with caches is cache lines. Since the 8 PTEs in a bucket are contiguous, there would almost certainly all fit into the same cache line. Thus only 4 cache lines (1 for each bucket, and two for the two-level page table) would be required.

This perhaps explains why there are no results for this section — caching page tables is by no means a violation of the principle of locality, as a process is more likely to fault on a nearby page than one further away, and if the PTEs are already in the cache due to being in the same cache line as the previous PTE (this is Linux hierarchical PTE — the hashed page table PTE would likely not be in the cache) then that's a memory access saved.

In sections 7 and 8, there is a great deal of confusion about the meaning of the term 'TLB'. At the start of section 7, the paper clearly indicates that the PPC hash table is not considered a TLB. However, throughout the section, it uses TLB to mean either (and sometimes both) the H/W TLB and the S/W hashed page table 'TLB', without saying which. Then in section 8 there is the sentence 'By caching the page-tables we cause the TLB to pollute the cache…', which doesn't make sense for either meaning of TLB.

Conclusion

Despite these grievances, the paper overall is a worthy piece of research, with some interesting and repeatable ideas. If the authors had been more diligent in their measuring and reporting of results it would be far more useful — but nevertheless they deserve credit for their efforts, and the future work section suggests a more complete and refined paper may be published in the future.

I can't conclude this report however without a mention of the pointless µ-kernel bashing. I am personally no big fan of µ-kernels, but in both the Abstract and the Analysis sections there are unjustified criticisms of them — particularly the comparison of Linux to Mach-based systems. Also the reference of the '*Personal Communication with Steve Jobs*' in the Analysis swipe instead of one of Liedtke's factual performance-based papers — even though he is referenced elsewhere — is unforgivable.