LRUMAP

$O(1)$ Massively-Scalable LRU

Nick Black

Georgia Institute of Technology
nickblack@linux.com

Abstract

Traditional $O(n)$ or even $O(\lg n)$ LRU implementations are too slow for backbone routers and IPS devices. We introduce LRUSET, a novel $O(1)$ true LRU scheme which becomes more space-efficient as the number of monitored sets increases.

Categories and Subject Descriptors I.5.3 [*Clustering*]: Algorithms

General Terms LRUMAP, networking algorithmics

Keywords LRUMAP, LRU, Pseudo-LRU, MROM, Precomputed Permutation Table (PPT)

1. Conventions

We will speak of n independent *sets*. It is assumed that all possible input values are somehow partitioned among these sets. For a set of TCP/UDP endpoint pairs, a set might be associated with each observed IP address pair (perhaps themselves backed by a one- or two-leveled LRU). For an associative cache, some subset of the address bits are used to index into a fixed number of sets. In an *order-r LRU*, each set contains r members. When $r = 1$, the system is said to be *direct-mapped*: replacement always occurs, and the number of sets n is equivalent to the system capacity. When r equals the capacity of the system, there can likewise be at most one set; such a system is said to be *fully associative*. Since $\lg r$ is a frequent term in LRU's complexity analysis, a system's order is almost always a power of 2[1].

[1] Though wasteful of space, other orders have been infrequently used to improve latency [5].

CS7260 May 5, Atlanta.

2. LRUMAP Data Structures

At LRUMAP's core lies a transition table shared among all sets, its size dependent upon r. This transition table is initialized at startup, and can be placed into constant memory once assembled (alternatively, it can be cast into cheap hardware using extremely reliable Masked-Or ROM (MROM) technology [4]). By *transition table*, we mean a family Σ of $|P|$ well-defined functions

$$\sigma_1(l), \ldots, \sigma_p(l) \longrightarrow P$$

where $p \in P$ and $1 \leq l \leq r$. P is the set of all permutations of r integers, and has $r!$ members. We call this the *Precomputed Permutation Table* (PPT). As it requires $\lceil \lg r! \rceil$ bits to encode p, the total size of this table is $r! r \lceil \lg r! \rceil$ bits.

True LRU assigns to each member in each set an index l, $0 \leq l \leq r-1$. Each set thus requires $\lg r$ bits to represent the relative ages of its r members. The central insight behind LRUMAP is that these members have been placed on a bijection against $0 \ldots r-1$; this is the classic definition of a *permutation* mapping. Each set's metastate can thus be considered completely described by a permutation of r, and $\lg r!$ bits are obviously sufficient to identify a set's LRU state. Just as in classic LRU, metastate has a variable cost linearly dependent on n, but each set as a whole maps into the precomputed permutation table using $\lceil \lg r! \rceil$ bits. It is simple to prove that the latter is strictly less than the former by properties of logarithms:

$$\begin{aligned} \lg ab &= \lg a + \lg b \\ \lg r! &= \lg (1 * 2 * \cdots * r - 1 * r) \\ \lg r! &= \sum_{i=1}^{r} \lg i \\ &= \sum_{i=1}^{r} \lg i \leq \sum_{i=1}^{r} \lg r \end{aligned}$$

3. LRUMAP Algorithms

There are two fundamental operations defined by an LRU system, which might be performed in a fused manner:

- `lookup(set,val)` $\rightarrow r$. Takes a value and searches the set's content for it, usually in parallel. If the value is

not present, the least-recently used (most stale) member will be replaced with this value. Returns the slot in which the value can now be found.

- `update(set,r)`→`set`. Takes a set and position, and updates the set's metastate based off a most recent reference to that position.

`lookup()`'s functionality is unchanged by LRUMAP. The amount of content memory used remains the same, as does the space of search algorithms (and the associated performance space). Typically, r is traded off against search latency and cost; a single-cycle search of more than a single SIMD unit is extremely expensive. As a result, r values for classic LRU are typically small, well within the range required by LRUMAP.

`update()` is the central interface to LRUMAP, and uniquely served. Rather than operating upon and updating a self-sufficient metastate description within the set, the single $\lceil \lg r! \rceil$-bit set value is used to index into the shared PPT. This uniquely identifies a function $\sigma_{idx}(l)$, to which the result from `lookup()` is provided. Functions from Σ are closed on P: this new result identifies a new permutation. Evaluation of $\sigma_{idx}(l)$ consists entirely of a small, constant number of arithmetic operations followed by a memory lookup, thus executing in $O(1)$ time.

4. Comparison to Classic LRU

The essential time and space complexities of classic LRU and LRUMAP are captured in Table 1, allowing for p updates performed in parallel. p can be made as large as r for those low values of r with which LRUMAP is applicable, but likely only at substantial unit cost. This is all the more true if multiple LRU sets are themselves to be searched and updated in parallel.

	LRU	LRUMAP
Time	$O(\lceil \frac{r}{p} \rceil), p \leq r$	$O(1)$
Space	$O(nr \lg r)$	$O(r! r \lceil \lg r! \rceil) + O(n \lceil \lg r! \rceil)$

Table 1. Essential properties of LRU/LRUMAP

As expected, we see in Figure 1 that LRUMAP saves significant space over LRU for large values of n and small values of r. At eighth order, LRUMAP takes over from LRU at $n = 111217$.

LRUMAP is even more effective at fourth order, requiring less space than LRU for all but trivially small n (see Figure 2).

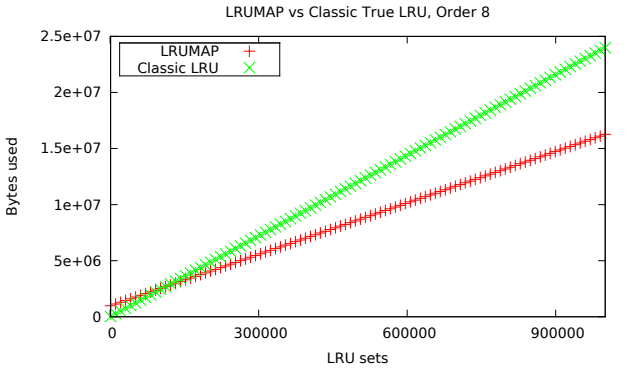

Figure 1. Eighth-order LRU

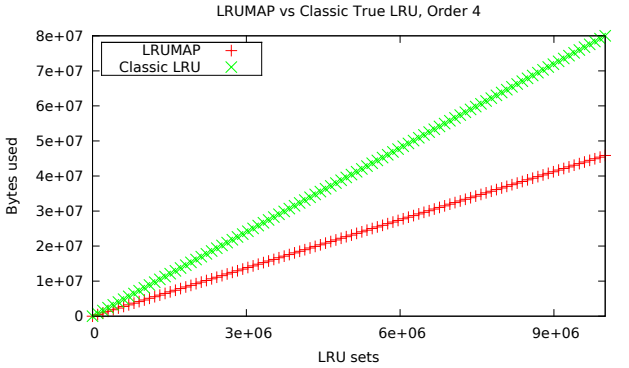

Figure 2. Fourth-order LRU

5. Beyond LRUMAP

5.1 Optimizations

LRUMAP as stated assigns no meaning to the relative locations of the permutation functions within Σ; the only requirement is that they occupy contiguous memory. This ensures that the `update()` function can be evaluated in constant time. By carefully placing the functions within the PPT, we can cut its size. For instance, every $\sigma \in \Sigma$ will map some input to itself; this corresponds to the case of the most recently used item being referenced. For each valid l, this will be true for exactly $\frac{r!}{r} r$ elements. Thus, by sorting the permutations by the l which they map to themselves, we can remove these elements—$\frac{1}{r}$ of the entries! This saves 25% and 12.5% of the space used by LRUMAP for $r = 4$ and $r = 8$, respectively.

5.2 Approximation

It may not be necessary to perform true, precise LRU. An entire family of schemes have been presented, approximating LRU in less time and/or space. VIA C3® and Intel processors of the Pentium® era [6] made use of *pseudo-LRU*, a direction vector-based scheme which requires only $O(\lg r)$ bits per set [2]. The PA-RISC 8600 [3] likewise used a proprietary *quasi-LRU* algorithm, with a similar reduction in bits per set. These schemes can be straightforwardly combined with LRUMAP to yield a new variant, wherein the LRUMAP entries represent and map among these algorithms' direction vectors rather than an order's permutations. There seems no advantage in doing so, however; Pseudo-LRUMAP will con-

sume strictly more space than Pseudo-LRU, and is unlikely to provide a speed advantage.

A direction vector is $\lg r$ bits, and the set of direction vectors is thus composed of $2^{\lg r} = r$ members. Just as before, we precompute a constant table, this time containing $r \lg r$-bit transitions for each of r entries. Each of n sets will require a $\lg r$-bit encoding of its current pseudo-LRU state. A single operation still suffices to update the metastate. Again allowing for p updates in parallel, we derive Table 2.

	Pseudo-LRU	Pseudo-LRUMAP
Time	$O(\lceil \frac{\lg r}{p} \rceil), p \leq \lg r$	$O(1)$
Space	$O(n \lg r)$	$O(r^2 \lg r) + O(n \lg r)$

Table 2. Essential properties of Pseudo-LRU/LRUMAP

In this case, however, the updates being performed involve $\lg r$ single bits. We've assumed r to be less than or equal to 8; even an 8-bit embedded processor could thus perform the updates in parallel. Pseudo-LRUMAP cannot be expected to provide any improvement over Pseudo-LRU.

5.3 Extensions

It is trivial to adapt LRUMAP to the Most-Recently-Used methodology, employed by page caches when a "looping sequential" [1] access pattern is detected.

References

[1] D. J. Dewitt and H.-T. Chou. An evaluation of buffer management strategies for relational database systems. *International Conference on Very Large Databases*, August 1985.

[2] J. Handy. *The Cache Memory Book*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, second edition, January 1998.

[3] K. A. Hurd. A 600 MHz 64b PA-RISC microprocessor. *IEEE Solid-state Circuits Conference*, February 2000.

[4] Integrated Circuit Engineering Corporation. *Memory 1997: Complete Coverage of DRAM, SRAM, EPROM, and Flash Memory ICs*. Smithsonian Integrated Circuit Engineering Collection. 1997.

[5] Intel Corporation. *Intel® Processor Identification and the CPUID Instruction*. Application Note 485. August 2009.

[6] T. Shanley. *80486 System Architecture*. PC System Architecture Series. Addison Wesley Longman, April 1995.