A full description of required metadata

0.1. What are the Required Metadata?

Our proposed method requires two types of metadata. The first type of metadata, such as *DeQWordCount*, is required for address translation between L1 and L2. *DeQWordCount* determines how many words (of L1 values) can fit in a quantized L2 block. *DeQWordCount* is an integer number. Access to *DeQWordCount* is in the critical path as it is required for sending a miss request to lower level cache (L2). Hence, *DeQWordCount* is stored in the TLB so that it can be accessed when the processor accesses the TLB for the traditional virtual to physical address translation. The second type of metadata, including *Step-size*, *Bit-width*, and *Mid*, is required for data conversion, We introduce a new hardware module, the `MetaData-Buffer`, to store the metadata required for data conversion. In the following sections we introduce all the fields in the TLB extensions and `MetaData-Buffer` and explains the hardware overhead of the extra metadata.

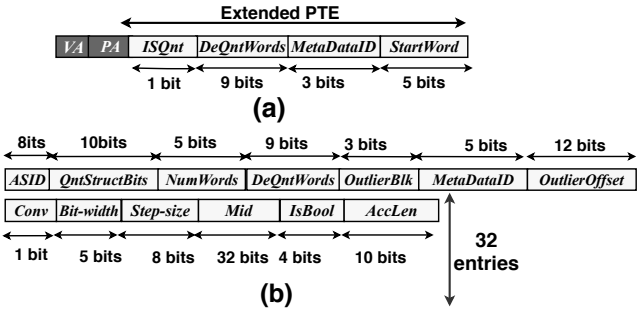

Figure 1: Metadata stored in (a) TLB, (b) MetaData-Buffer

0.1.1. TLB Entry Extension We extended TLB entries to store four fields (Figure 1 (a)): (i) *IsQuant* indicates whether the page is quantized or not, (ii) *DeQWordCount* is the number of dequantized words that fits in a quantized block, (iii) *MetaDataID* is the identifier of the corresponding structure and (iv) *StartWord* indicates with which word of the structure the page starts .

0.1.2. The MetaData-Buffer The `MetaData-Buffer` stores the quantization parameters. Figure 1(b) shows the entries in each `MetaData-Set`. Each `MetaData-Set` keeps track of two different types of information.

Quantization Parameters. There are two types of parameters. First, it stores the quantization parameters for each variable: *Bit-width*, *Step-size*, and *Mid*. We decided to have one entry per each word in the structure as the floating point variables are word-aligned. We keep another field per word, *AccLen*, which is the accumulated quantized length of variables.This filed simplifies finding the location of quantized bits corresponding to an L1 block within a L2 blocks. The second type of parameters are boolean values. The *Conv?* field is a 1-bit field that indicates whether the variables in that word should be quantized or not. Note that we only convert floating-point and boolean variables. This field is set to zero for integers, characters or any other types indicating that no conversion is required. The *IsBool* field is a bit-map field which has one bit per each of the four bytes of the word. Each bit determines whether the corresponding byte in the word contains a boolean variable or not. For example, if this field is set to "TTFF" (1100), it indicates that the first two bytes are booleans. Accordingly, the two first bits of the quantized data corresponding to this word will be converted to boolean values.

Bookkeeping Parameters. These parameters are stored in the `MetaData-Set` to locate and track additional information, simplifying address translation and conversion. There are seven fields at the beginning of each `MetaData-Set`: (i) The *MetaDataID* and (ii) the *ASID* are used to locate this specific entry for a structure. The *MetaDataID* is also duplicated in PTE and TLB entry so that it is possible to find the entry during the address translation (PTE and TLB entry traditionally have *ASID*). (iii) The *OutlierBlk* field specifies how many blocks per page should be reserved for outliers and (iv) *QntStructBits* determines the number of bits that a structure needs in quantized format. These two pieces of information is used to determine the appropriate page size for the quantized structures during the page allocation. (v) *DeQWordCount* is the number of de-quantized words that fits in a quantized block, used in address translation. (vi) *NumWords* stores the number of words within the structure in order to determine the number of valid entries in a `MetaData-Set`. (vii) *OutlierOffset* stores the page offset of the reserved space for outliers.

0.2. Storage Overhead

The major storage overhead of our proposed method stems from six sources: *(i)* extended TLB entries, *(ii)* extra bits in cache tags, *(iii)* `MetaData-Buffer`, *(iv)* outlier buffer in the memory controller*(v)*, and *(iv)* valid bits in prefetch buffers.

- **Extended TLB Entries.** Existing systems, such as Intel x86-64 systems [1] have up to 15 unused bits in their TLB entries. Our proposed method requires 18 bits (Figure 1(a)). Accordingly, we only add three bits to the original TLB entry and the total overhead per core is 216 bytes (in a system with 64-entry first level TLB and 512-entry second level TLB).
- **Extended Cache Tags.**
  We add 16 bits to the cache block tags (eight bits for ASID, one bit for *IsQuant*, three bits for *MetaDataID*, and five bits

for *StartWord*). Therefore, in a system with 32 KB L1, the overhead is 1 KB bytes per core.

- **MetaData-Buffer.**
  We have observed that applications often have few different quantizable structures (less than four). Accordingly, we designed a `MetaData-Buffer` with eight sets, to support having up to eight different quantizable structure. Each set `MetaData-Set` (Figure 1(b)) have up to 32 entries to support structures that have up to 32 words. Hence, the size of the `MetaData-Buffer` is 1972 bytes per core.
- **Outlier buffer.**
  We assigned 256 Bytes per core to store the recently accessed outlier blocks. The total storage overhead is less than 3.5 KB bytes.
- **Valid bits in prefetch buffers.**
  We assumed a 4-entry prefetch buffer for partial blocks [2]. As our method is word-aligned, we only add one valid bit per word to the prefetch buffer (total overhead of 64 bits).

References

[1] Intel 64 and IA-32 Architectures Developerś Manual . https://https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-manual-325462.html.

[2] Norman P Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Computer Architecture, 1990. Proceedings., 17th Annual International Symposium on*, pages 364–373. IEEE, 1990.