Ben Shaker
ECE 2500
May 2019

Project 3

At first, this project seemed quite intimidating. The abstract nature of what we were asked to accomplish actually seem too ill-defined to be possible. I decided to charge in blindly and start with what I knew for certain: there were 128 possible configurations that I had to iterate through for my system's input and output. While I let me subconscious plan ahead for me, I spent a good deal of time setting up my program to simple handle I/O and this basic iteration. That alone required probably one hundred lines of code.

Next, I recalled my readings from the textbook which spoke of memory blocks as being like books in the library. I took this metaphor to its extreme and I started drawing bookshelves on scratch paper until I understood the concept without mistake (hopefully!). I decided to think of the system in this way: A Cache contains some Set(s) which contains some Block(s).

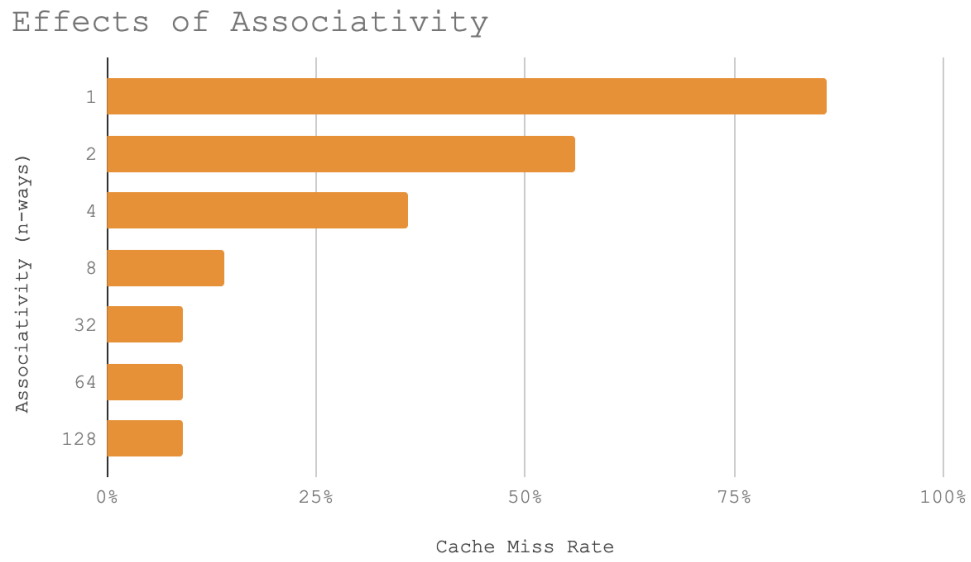
Likewise, the overall Cache is just a *Bookshelf*. The number of *Shelves* (Sets) on the Bookshelf is computed from this Cache's particular configuration settings. On each Shelf, there may exist some *Books* (Blocks), (i.e. in each Set there may exist some Blocks) but - this is where the metaphor seems complicated - if each Book represents one *byte* in memory then it may be best to to use Baskets (or Bins) to represent Blocks. In this way, each Shelf would hold multiple Baskets, and each Basket would contain multiple Blocks.

Now, to bring tie this back in to the data structures: The *Index* would correspond to a particular Shelf. The *Tag* would correspond to a particular Basket on that Shelf. And the *Offset* would correspond to a particular *Book* in that Basket on that Shelf.

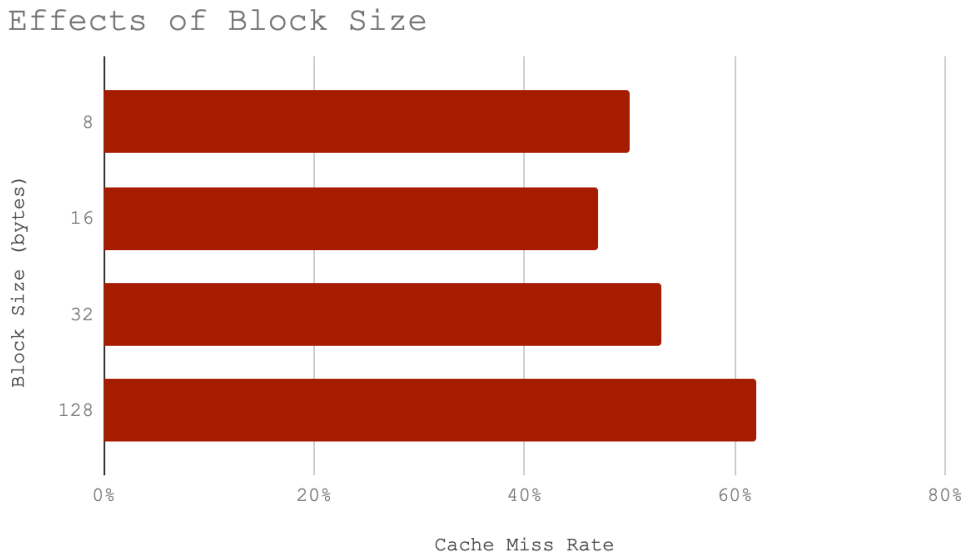
In a way, this arrangement makes it a bit challenging to relate to some figures in the textbook (such as Figure 5.18) because their representation of associativity *seems* unrepresentable in this world of Books, Baskets, and Bookshelves. But it does seem as simple as adding Baskets to individual Shelves. Whereas in DM, each Shelf is only allowed one Basket, in 4-way associativity there are 4 Baskets per Shelf, and given full associativity there is only one shelf which contains all possible Baskets.

I held this image in my mind as I create my List of Lists to house my Lists of Objects; this was my Pythonic representation of the same Bookshelf setup. To indicate that my Baskets were empty I simply gave them no books. But to indicate that my Shelves were empty I gave them empty baskets (so that I did not have to come up with the Baskets as I was using them). This metaphor held well until I encountered specific issues implementing the write-through as opposed to write-back of instructions, but even then I only had to modify a few lines of code. Overall, I first wrote only the 'read' instructions for this program, then afterward the 'write' instructions. Afterward I found so many similarities that I ended up combining the two together.

Ben Shaker
ECE 2500
May 2019



* as associativity increases, the miss rate typically decreases
* this is most applicable to programs working in a small memory range
* the miss rate shows a plateau in this example but even more-specific address access could show a benefit of the 128-way over the 64-way, etc.



* as block size increases, spatial locality is generally benefited, but..
* if a program's memory access addresses are *too* random or not-very-spatially located (e.g. forming groups but not all groups are close together), performance can suffer more than usual because of the higher costs of moving larger data blocks