Homework 1 Solutions

1. (5 points) Briefly state what benefits remained to decreasing features size (i.e. making smaller transistors) when decreasing clock period (i.e. increasing clock frequency) was no longer feasible because of heat dissipation issues? In other words, why did the engineers keep decreasing feature size after 2003?

SOLUTION:

- Smaller transistors makes it possible to fit more hardware (cores, ALUs, FPUs, cache, etc.) onto a chip of the same size.
- Allows for cheaper hardware, more powerful hardware, and ultimately multiple cores that allow for parallel programming, which is how machine performance has continued to increase despite the infeasibility of decreasing clock frequencies.
- It becomes possible to reduce energy requirements for the same hardware

2. (5 points) In your own words, explain the benefits of a cache. What are some common characteristics of programs does it take advantage of to improve memory access speeds?

SOLUTION:

Caches sit between the CPU and DRAM and take advantage of temporal and spatial locality in a program. The idea is that if a program accesses a specific location in memory, it is likely to access it again in the near future (temporal locality) or access nearby locations (spatial locality), so by storing more than just that one location's worth of data during the load from main memory, the cache can help speed up future memory accesses. Programs that access data in these ways will benefit the most from caching.

3. (20 points) Consider the following code:

```
for (int i = 0; i < N-1; i++)
  for (int j = 0; j < N; j++)
    A[i][j] += A[i+1][j];
```

a. (10 points) Compute the number of misses in terms of N and w (words in a cache line, where a word is 8 bytes) if the cache can only store 1 entire row of matrix A (cache size = N*sizeof(double)). You can assume that A starts at the beginning of a cache line, that A is an array of doubles, and that the cache is fully associative and is an LRU cache. How many compulsory, capacity, and conflict misses are there?

SOLUTION:

Every row gets read from at least once, and will incur the $\frac{N}{w}$ compulsory misses. So there are $\frac{N^2}{w}$ compulsory misses. When the N-2 inner rows (`i = 2...N-1`) get read the second time however, they will incur $\frac{N}{w}$ capacity misses. This is because at the completion of iteration `i`, the cache will contain data from `A[i][N/2...N-1]` and `A[i+1][N/2...N-1]`, so the load of `A[i+1][0]` at the next iteration `i+1` will be a capacity miss as it has been evicted out of the cache in the previous iteration `i`. So there

are $(N-2)\frac{N}{w}$ capacity misses. Because it's a fully associative cache, there are 0 conflict misses. So the total miss count is $\frac{N^2}{w} + \frac{N}{w}(N-2) = \frac{2N^2-2N}{w} = \frac{2N(N-1)}{w}$.

b. (10 points) Compute the number of misses in terms of N and w if the cache can store 2 entire rows of matrix A (cache size >= 2*N*sizeof(double)). You can make the same assumptions as part (a). How many compulsory, capacity, and conflict misses are there?
SOLUTION:
All of the N-2 inner rows (2 to N-1) that are accessed twice remain in the cache the second time they are accessed, so each row access only incurs compulsory misses and no capacity misses. Therefore there are $\frac{N^2}{w}$ compulsory misses. Because it's a fully associative cache, there are 0 conflict misses. So the total miss count is $\frac{N^2}{w}$.

4. (10 points) Consider the following code:

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    A[i][j] += B[i][j] + C[i][j];
```

a. (4 points) Will 2D-tiling (blocking) reduce the number of cache misses for the above code? Justify your answer. Assume the cache is empty, fully-associative and the arrays do not alias.
SOLUTION:
No, tiling does not optimize this code any further because all accesses are row-major and of stride 1 and there is no reuse of any data.

b. (2 points) Can you think of any optimizations (aside from tiling/blocking) that could improve the performance of this code? Rewrite the code with the optimizations. State your assumptions clearly.
SOLUTION:
- Vectorization
- Prefetching
- Loop unrolling
- Parallelism via multithreading/OpenMP

c. (4 points) What is the arithmetic intensity of this code? You can assume all arrays store double precision data.
SOLUTION:
There are 2 floating point operations (the assignment operator '=' does not count as one), 3 loads, and 1 store.
So 2 FLOPs / (3 word loads + 1 word store) = 2/4 = 0.5 FLOPs/word = 0.0625 FLOPs/byte