What advantages does a set-associative cache have over a direct-mapped cache of the same total capacity?

A Fewer cold misses.

B Fewer capacity misses.

C Less false sharing.

D Fewer conflict misses.

E Lower latency.

Which phenomena could cause the following code snippet to use the L1-cache ineffectively on a 12-core machine?

```
#define NUM_CHUNKS 12

int sums[NUM_CHUNKS];

void compute_sum(int chunk_id) {
  sums[chunk_id] = 0;
  for (int i = 0; i < 5000; i++) {
    sums[chunk_id] += i;
  }
}

int main() {
  cilk_for (int i = 0; i < NUM_CHUNKS; i++) {
    compute_sum(i);
  }
  return 0;
}
```

A True sharing.

B False sharing.

C Capacity misses.

D Conflict misses.

E TLB misses.

Consider the following code for multiplying two matrices:

```
// Assume that n is an exact power of 2.

void Rec_Mult(double *C, double *A, double *B,
              int n, int rowsize) {
  if (n == 1) {
    C[0] += A[0] * B[0];
  } else {
    int d11 = 0;
    int d12 = n/2;
    int d21 = (n/2) * rowsize;
    int d22 = (n/2) * (rowsize+1);

    Rec_Mult(C+d11, A+d11, B+d11, n/2, rowsize);
    Rec_Mult(C+d11, A+d12, B+d21, n/2, rowsize);
    Rec_Mult(C+d12, A+d11, B+d12, n/2, rowsize);
    Rec_Mult(C+d12, A+d12, B+d22, n/2, rowsize);
    Rec_Mult(C+d21, A+d21, B+d11, n/2, rowsize);
    Rec_Mult(C+d21, A+d22, B+d21, n/2, rowsize);
    Rec_Mult(C+d22, A+d21, B+d12, n/2, rowsize);
    Rec_Mult(C+d22, A+d22, B+d22, n/2, rowsize);
  }
}
```

For the following questions, assume that we have a tall cache of size $\mathcal{M}$, filled with cache lines of size $\mathcal{B}$, and assume that $\mathcal{B}^2 \leq c\mathcal{M}$ for some sufficiently small constant $c \leq 1$.

1. What recurrence does the cache complexity (number of cache misses) $Q(n)$ satisfy?
2. What is the height of the recurrence tree for $Q(n)$?
3. What is the total number of cache misses that occur in the leaves of the tree?
4. What is the total number of cache misses for the algorithm?