1.

> There might be several reasons:
>
> 1. One core has higher performance than the other, so executing on a single two-way hyperthreading core will get better performance.
> 2. Frequent cache consistency in two cores increases the overhead of executing the program in two cores.

2.

> 1. The overhead of creating and terminating threads is less efficient than the concurrency.
> 2. Resource contention of the threads affects the performance of the application.

3.

> 1. Not fully utilizing the resource such as the CPU: lead to wasting the rest of the computation resources.
> 2. More time consumption: the problem is not paralleled enough to reduce the running time.
> 3. Less responsive in the application: some tasks in the application may be delayed because of the time consumption then affects the following functions.

4.

> ```
> a[0] = 0;
> for(i = 1; i < n; i++)
>   a[i] = (1+i)i/2
> ```

5. a

> Sequential time is $3+4=7$ms, parallelizable time is $16*5=80$ms, so $F=7/(7+80)=7/87$.
>
> Amdahl's Law: $1/[F+(1-F)/P]=1/[7/87+(1-7/87)/P]=1/[7/87+80/87P]$
>
> As we can only simultaneously 5 tasks here, so more cores will not affect the speed. If $P=5$, the result goes to around 3.78.
>
> Hence, the maximum speed-up is 3.78.

b.

> As we can see, the maximum speed is when the number of cores goes to 5, then the time consumption of then the parallelizable part will be 16ms.
>
> Hence, the largest number of cores is 5.

c.

> When using one core, the running time is $3+4+5*16=87$

> When using five cores, the running time is $3 + 4 + 16 = 23$
>
> The speedup is $87/23 = 3.78$

d.

> No difference.
>
> Because a is using percent of the sequential and parallelizable part, c is actually using time as a metric. They are the same in essence.

6. a

> The minimum number of cores is 2.
>
> Total time taken in one core: $50 + 10 + 10 + 5 + 10 + 100 + 10 + 100 = 295$
>
> Total time taken in two cores: $50 + 10 + 5 + 100 + 100 = 265$
>
> Speedup: $295/265 = 1.113$

b.

> Span: $50 + 10 + 5 + 100 + 100 = 265$
> Work: $50 + 10 + 10 + 5 + 10 + 100 + 10 + 100 = 295$

c.

> Parallelism: work/span = 1.113

d.

> They are the same.
>
> a is calculating the total work runtime and calculating the shortest time of all strategies, then checking if another core can handle the rest of the tasks.
>
> c is actually the same idea, as checking how many extra cores are needed except for the work in the longest path including the original one.