1. a.

> Work: $T_1 = 370$
>
> Span: $T_\infty = 170$
>
> Parallelism: $T_1/T_\infty = 2.18$
>
> To check that, as $T_p \geq T_\infty$, so the smallest time is at least the consumption for $A \to D \to F \to G$. The rest of the time is $370 - 170 = 200$, which means it is still larger than 170 running by another core.
>
> Hence, the smallest number of cores needed to get the best performance is 1+2=3.

b.

> The total run time is the longest duration, which is 170.

c.

> Span: $T_\infty = 10 + 100 + 50 + 10 = 170$
>
> Work: $T_1 = 10 + 50 + 100 + 100 + 50 + 50 + 10 = 370$

d.

> Parallelism: $T_1/T_\infty = 2.18$

e.

> The number means an estimation of the smallest number of cores that gives the best speedup.

2.

> 1. SIMD is cheaper, and smaller, only one single instruction decoder is needed and it requires less memory, while MIMD needs more than one decoder, which determines its higher expense.
> 2. The MIMD is complex and not easily programmed for a specific task, which may get less ROI on the performance.

3.

> 1. The overhead of parallelizing this part into pieces is large, so it would be better not to waste resources to parallelize them.
> 2. According to Amdahl's Law, the maximum speed-up that can be achieved by using P processors is $1/[F + 1(1 - F)P]$. If the sequential part is large enough, the parallel part is small. It is not worth parallelizing them.
> 3. Resources such as memory are not enough for parallelizing.

4.

> 1. Time consumption. Choose the program that takes less time to finish the problem.
> 2. Computing resources. Choose the one with less hardware usage.

3. Portability. The ability to be used in other operating systems.
4. Scalability. If the program can be improved in the future or be used for other tasks.

5.

No.

1. Parallelism. When it comes to multi-core, the problem may be parallelizable for the whole part, so the time consumption can be largely decreased.
2. Resource usage. The sequential algorithm for a single core may not efficiently use the computing resources such as memory. These can be optimized by the multicore platform.

6. a

Yes. The algorithm is used to add the element from the second half to the first half index by index.

Therefore, each addition can be considered an atom operation, so we can parallelize the algorithm.

b.

The maximum number can be $N/2$.

As there are $N/2$ additions, each of which is atomic and can be executed by one core, so the number is $N/2$.

7.

According to Amdahl's Law, the speed-up $3 = 1/[F + (1 - F)/P] = 1/[0.2 + (1 - 0.2)/P]$, so $P = 6$. Therefore 6 cores are required for three-fold acceleration.

If $5 = 1/[F + (1 - F)/P] = 1/[0.2 + (1 - 0.2)/P]$, $P = +\infty$. Hence, it cannot be speedup five-fold but can approach it when the number of cores tends to infinity.

8.

Cathe hit rate is more important.

If we go from L2 to memory, the latency will largely increase. It can be accepted if more cache access latency from L1 to L2. But generally, cache access latency and cache hit rate should be in an appropriate range, here we tend not to go to the memory.