Problem 1

1 a

my_first_i = 0
my_last_i = p

1 b

my_first_i = 0
my_last_i = $p - n\%p$ essentially use only what u need to maximize parallelization.

Problem 2

2 a

core0_adds = $p - 1$
core0_receives = $p - 1$

2 b

core0_adds = $\log_2 p$
core0_receives = $\log_2 p$

2 c

2	1	1
4	3	2
8	7	3
16	15	4
32	31	5
64	63	6
128	127	7
256	255	8
512	511	9
1024	1023	10

2 d

Doing the receive is much slower, because computation is much faster than reading from cache.

Problem 3

You should try to speedup the parts of the program which take the most amount of time, and this will make sure that the program will gain the most speedup rather than optimizing small for loops.

Problem 4

Data Level Parallelism

- Manipulates the amount of data that can be processed by the processor, like word size. Contrasts with Task Parallelism

Instruction Parallelism

- Instructions can be rerun and run in a different order in order to achieve parallelism.

Task Parallelism

- Different calculations can be done on separate and independent parts of the data at the same time.

Problem 5

This is because with shared memory, if computation does not depend on the computed value of another core, if each core has its own discrete memory, the lookup time will be faster.

Problem 6

6 a

Yes, we can, as we increment the first half of the array with the second half, none of the values are dependent on each other.

6 b

$n/2$, this is because there are no more immediate subtasks which can be parallelized besides possible sequential tasks after and before, so this is the fastest possible speedup