Part 1

The cache program is design in this manner, first you allocate a large chunk of memory into an array and then you gradually increase your rate of access from one element all the way to every element. In order, to determine a cache miss while compensating for the increase in array size a change if access time of 1ms is omitted. After multiple runs the size of cache block is determined to be 4KB and the cache size to be around 128K and this was tested on various machine with 3.4GHz intel quad core CPU with similar results such as null.cs.rutgers.edu, prototype.cs.rutgers.edu. After doing random accesses of various parts of array in intervals of power of 2 to determine that the associativity of the processor is 4-way. The average penalty for miss cache hits is about 15 to 20 microseconds which is a really big penalty. The main method for determining block size is checking the speed of a subset of cache size and if it switches from one block to another there is a slight delay of time. For Cache misses I just gradually increase size of memory accesses and look for a sudden peak in cost.

Part 2

In fork() the average time increase with memory size because the kernel copies the heap information exactly for the children as if it became a separated entity. A standalone process so that no memory are shared. To make sure that the processes executes in the right order and that time doesn’t get confused the child will exit and only the parent will keep track of time of fork and then calculate average time.

Part 3

Then communication cost of pipes seems to be constant because the information that seems to pass over is memory the memory address of the information so no matter what size the time for communication stays constant. Which seems to be fair because only one processes can write to the pipe area at a time else there would be an error.