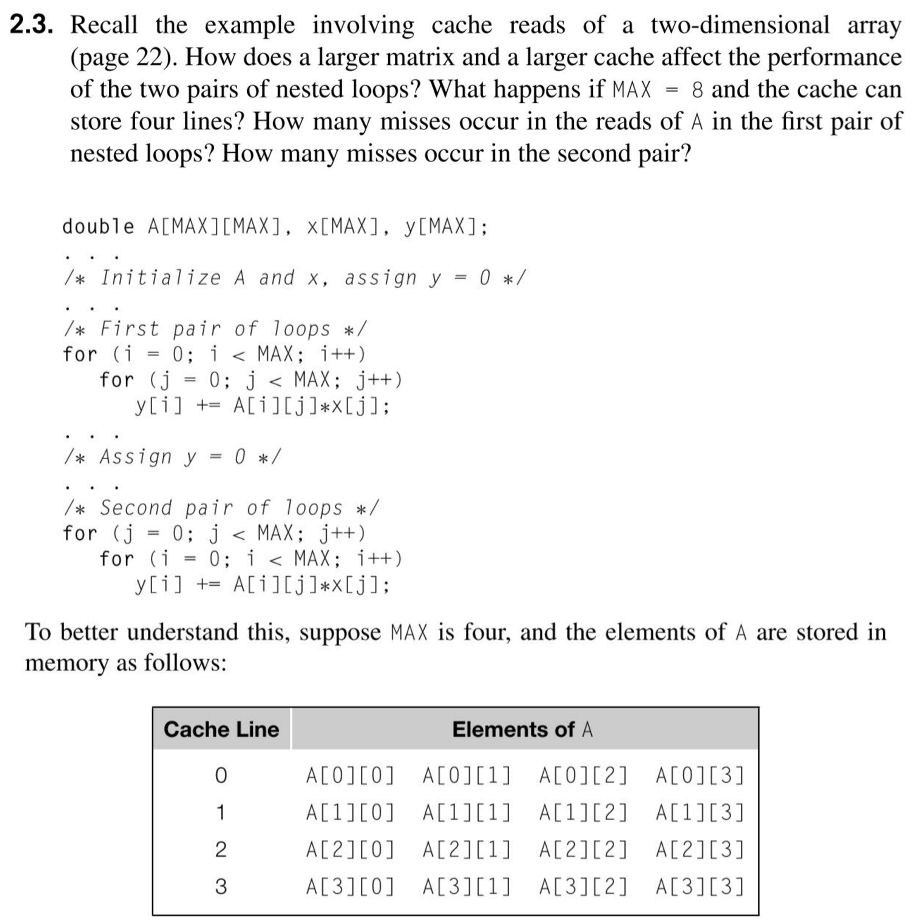
Homework of Chapter 2

Exercise 2.3

Problem



2.3. Recall the example involving cache reads of a two-dimensional array (page 22). How does a larger matrix and a larger cache affect the performance of the two pairs of nested loops? What happens if `MAX = 8` and the cache can store four lines? How many misses occur in the reads of `A` in the first pair of nested loops? How many misses occur in the second pair?

```
double A[MAX][MAX], x[MAX], y[MAX];
. . .
/* Initialize A and x, assign y = 0 */
. . .
/* First pair of loops */
for (i = 0; i < MAX; i++)
   for (j = 0; j < MAX; j++)
      y[i] += A[i][j]*x[j];
. . .
/* Assign y = 0 */
. . .
/* Second pair of loops */
for (j = 0; j < MAX; j++)
   for (i = 0; i < MAX; i++)
      y[i] += A[i][j]*x[j];
```

To better understand this, suppose `MAX` is four, and the elements of `A` are stored in memory as follows:

Cache Line	Elements of `A`			
0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
2	A[2][0]	A[2][1]	A[2][2]	A[2][3]
3	A[3][0]	A[3][1]	A[3][2]	A[3][3]

Answer

a) If the size of array increased, the program will run slower because of the increasing of cache miss rate caused by the lack of cached space. And when cache increase at the same time, the program can run faster because of the decreasing of cache miss rate.

b) If MAX = 8 and the cache can store only four lines, the program will slow down due to many cache miss error.

c) Assuming that the cache can store four lines and MAX = 8, the first pair of nested loops will meet 16 cache misses. For each line at the matrix A, two misses will happen. And as for the second pair of nested loops, 32 cache misses will happened. Because for each line 4 misses will happen.

Exercise 2.10

Problem

2.10. Suppose a program must execute 10^{12} instructions in order to solve a particular problem. Suppose further that a single processor system can solve the problem in 10^6 seconds (about 11.6 days). So, on average, the single processor system executes 10^6 or a million instructions per second. Now suppose that the program has been parallelized for execution on a distributed-memory system. Suppose also that if the parallel program uses p processors, each processor will execute $10^{12}/p$ instructions and each processor must send $10^9(p-1)$ messages. Finally, suppose that there is no additional overhead in executing the parallel program. That is, the program will complete after each processor has executed all of its instructions and sent all of its messages, and there won't be any delays due to things such as waiting for messages.

a. Suppose it takes 10^{-9} seconds to send a message. How long will it take the program to run with 1000 processors, if each processor is as fast as the single processor on which the serial program was run?

b. Suppose it takes 10^{-3} seconds to send a message. How long will it take the program to run with 1000 processors?

Answer

If we use 1000 processors to execute the task, each processor will execute $\frac{10^{12}}{10^3} = 10^9$ instructions during $10^9/10^6 = 10^3$ seconds. Define the time to send message is t, for each processor the overhead will be $999 * 10^9 * t$ (Suppose there is no other overhead).

a) The time of each processor will be $10^3 + 999 = 1999$ seconds.

b) The time will be $10^3 + 999 * 10^6 = 999001000$ seconds.

Exercise 2.15

Problem

2.15. a. Suppose a shared-memory system uses snooping cache coherence and write-back caches. Also suppose that core 0 has the variable `x` in its cache, and it executes the assignment `x = 5`. Finally suppose that core 1 doesn't have `x` in its cache, and after core 0's update to `x`, core 1 tries to execute `y = x`. What value will be assigned to `y`? Why?

b. Suppose that the shared-memory system in the previous part uses a directory-based protocol. What value will be assigned to `y`? Why?

c. Can you suggest how any problems you found in the first two parts might be solved?

Answer

a) In the snooping cache coherence, all cores will be informed when the cache contains variables been updated. And in the write-back caches, the data in memory will not be updated immediately. Instead, the updated data in cache will be marked as dirty. When the cache line is replaced by a new memory cache line, the dirty line is written to memory. So the value of y will be 5, because when core 1 retrieve value of x, the dirty cache will be updated.

b) In the directory-based cache, only cores using a particular variable in their local memories will be contacted. So the value of y will be what was stored in memory at the time core 1 retrieve the variable x, since it was not using the variable x.

c) In the first part, all cores will be informed when the variable updated. It's no need to do so. And in the second part, data adventure will happened when two core use one variable at the same time. To solve the problems, we can store the data in memory with a history log. Each update will be logged. By this method, all core can read the log to decide which variable should be used.

Exercise 2.16

Problem

2.16. a. Suppose the run-time of a serial program is given by $T_{\text{serial}} = n^2$, where the units of the run-time are in microseconds. Suppose that a parallelization of this program has run-time $T_{\text{parallel}} = n^2/p + \log_2(p)$. Write a program that finds the speedups and efficiencies of this program for various values of n and p. Run your program with $n = 10, 20, 40, \ldots, 320$, and $p = 1, 2, 4, \ldots, 128$. What happens to the speedups and efficiencies as p is increased and n is held fixed? What happens when p is fixed and n is increased?

b. Suppose that $T_{\text{parallel}} = T_{\text{serial}}/p + T_{\text{overhead}}$. Also suppose that we fix p and increase the problem size.

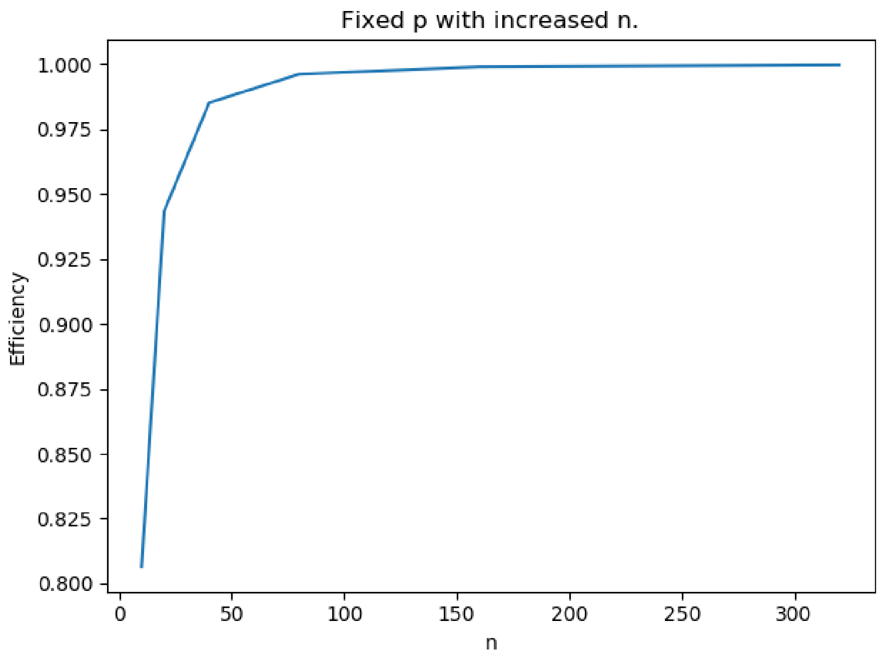
- Show that if T_{overhead} grows more slowly than T_{serial}, the parallel efficiency will increase as we increase the problem size.
- Show that if, on the other hand, T_{overhead} grows faster than T_{serial}, the parallel efficiency will decrease as we increase the problem size.

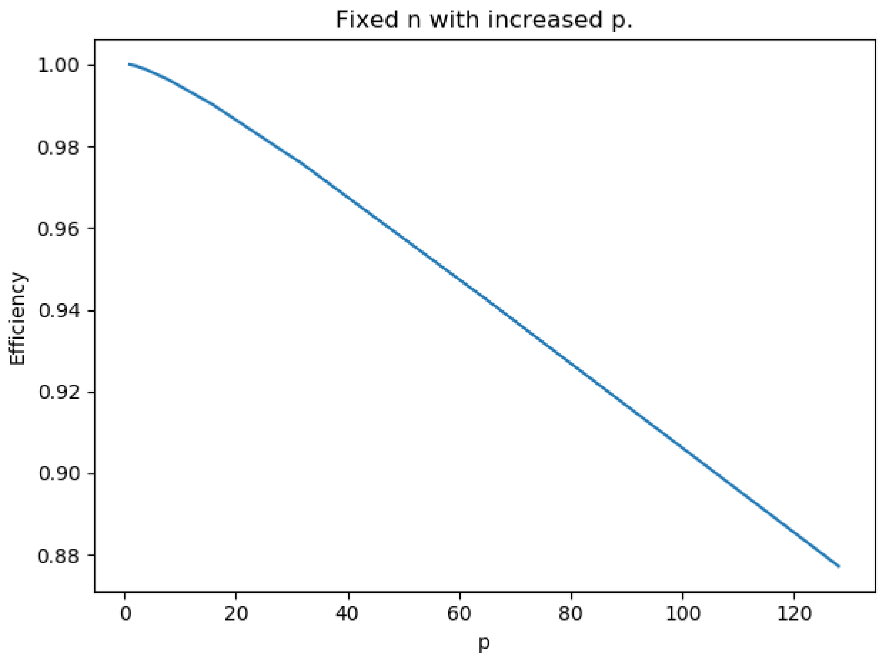
Answer

Code

```
import math
import matplotlib.pyplot as plt

n = 10
p = 8

ns = []
```

```python
ts = []
for i in range(6):
    _n = n * pow(2, i)
    t_serial = _n * _n
    t_parallel = _n * _n / p + math.log2(p)
    ns.append(_n)
    ts.append(t_serial / (p * t_parallel))

plt.plot(ns, ts)
plt.xlabel('n')
plt.ylabel('Efficiency')
plt.title('Fixed p with increased n.')
plt.show()
plt.close()


n = 80

ns = []
ts = []
for i in range(8):
    p = pow(2, i)
    t_serial = n * n
    t_parallel = n * n / p + math.log2(p)
    ns.append(p)
    ts.append(t_serial / (p * t_parallel))

plt.plot(ns, ts)
plt.xlabel('p')
plt.ylabel('Efficiency')
plt.title('Fixed n with increased p.')
plt.show()
```





a) When n is fixed and p increases, the serial execution time is kept constant and the parallel execution time decreases more rapidly when the value of n is greater than p. When p is held fixed and n increases, the serial execution time grows and the parallel execution time increases.

b) If we check the efficiency function, at the time that $T_{overhead}$ grows more slowly than the T_{serial}, the efficiency of the parallel executing time will be changed due to the serial time . Thus, the parallel time would be practically limited by the serial time.

Exercise 2.22

Problem

2.22. As we saw in the preceding problem, the Unix shell command `time` reports the user time, the system time, and the "real" time or total elapsed time. Suppose that Bob has defined the following functions that can be called in a C program:

```
double utime(void);
double stime(void);
double rtime(void);
```

The first returns the number of seconds of user time that have elapsed since the program started execution, the second returns the number of system seconds, and the third returns the total number of seconds. Roughly, user time is time spent in the user code and library functions that don't need to use the operating system—for example, `sin` and `cos`. System time is time spent in functions that do need to use the operating system—for example, `printf` and `scanf`.

a. What is the mathematical relation among the three function values? That is, suppose the program contains the following code:

```
u = double utime(void);
s = double stime(void);
r = double rtime(void);
```

Write a formula that expresses the relation between `u`, `s`, and `r`. (You can assume that the time it takes to call the functions is negligible.)

b. On Bob's system, any time that an MPI process spends waiting for messages isn't counted by either `utime` or `stime`, but the time *is* counted by `rtime`. Explain how Bob can use these facts to determine whether an MPI process is spending too much time waiting for messages.

c. Bob has given Sally his timing functions. However, Sally has discovered that on her system, the time an MPI process spends waiting for messages is counted as user time. Furthermore, sending messages doesn't use any system time. Can Sally use Bob's functions to determine whether an MPI process is spending too much time waiting for messages? Explain your answer.

Answer

a) r = u + s

b) waiting_time = r - u - s

c) Sally cannot use Bob's function to determine whether MPI process is spending too much time waiting for messages. Because in Sally's system the waiting time has been calculated and included in user's time.