6.895 Theory of Parallel Systems Lecture 7

Memory Contention

Lecturer: Charles Leiserson *Scribe: C. Scott Ananian and Barbara Mack*

Lecture Summary

1. *Introduction to Memory Contention*
   This section introduces a mechanism for sharing M memories among P processors.
2. *The* (P, M, T) *Recycling Game*
   We present a game played by an adversary which models concurrent random accesses to memory.
3. *Bounds on the Expected Total Delay*
   We prove that the expected total delay is bounded.

1 Introduction to Memory Contention

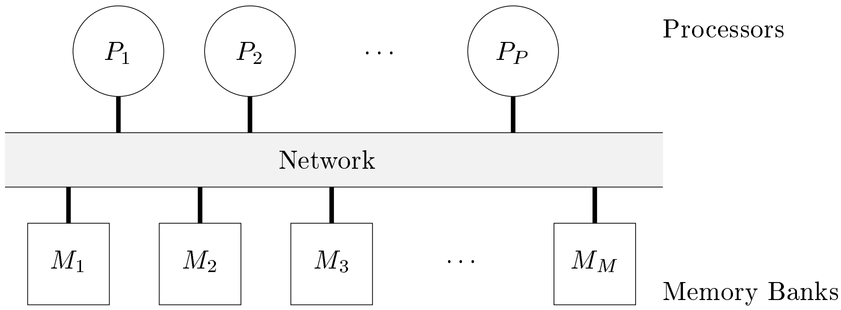

Figure 1: A parallel computer with P processors (circles) and M memory banks (squares) connected by a network (shaded).

Our model, illustrated in Figure 1, shows P processors and M memories distributed along an interconnect. P processors concurrently access M memory banks. We assume that concurrent accesses to the same memory bank (M_i) are queued. Requests are served one at a time, with the losers waiting.

When more than one request is made of a particular bank, we pessimistically allow the adversary to choose which access is served, although it can't choose to ignore everybody. That is, if 1 or more requests are queued, it much choose one. The adversary chooses the *worst possible* request to service in banks where there is a choice.

To clarify, all P processors access the memory banks at once, but they do not necessarily all access the same memory bank. Requests get queued only when more than one processor wants data *from the same bank*. In our model the network can carry an unlimited number of requests and replies at once.

How much are accesses delayed by contention? The length of the delay depends on the pattern of the accesses to the memory banks: if all P processors access the same memory bank (say, M_1), the delay is larger than if the accesses are distributed (even randomly) across the banks. As a trivial example, no access is ever delayed if P_i always accesses M_i.

Since the pattern matters, we will analyze a *random* pattern. This approach is similar to the one we will use for Cilk work-stealing in the next lecture. Our results with this assumption are useful enough that there have even been proposals to hash memory addresses to make the location access more uniformly random.

2 The (P, M, T) Recycling Game

The (P, M, T) recycling game [Blumofe 1995] is a "balls and bins" game with P balls (representing processors), M bins (representing memory banks), and T tosses (representing memory accesses).

The game is played by an adversary, with all the balls resting initially in a reservoir. On each turn of the game:

(a) The adversary chooses a subset of the balls in the reservoir and tosses each ball into a (uniform, independent) random bin. Each ball tossed counts towards T. A "toss" is *not* a turn.

(b) For each bin that contains ≥ 1 ball, the adversary returns one ball to the reservoir. The rest remain in the bin, and thus can't be tossed on the next turn.

The combination of (a) and (b) is a "turn."

The adversary's goal is to make the delay as large as possible. (We will define "delay" in a moment.) Intuitively, the adversary's best strategy is always to choose in part (a) *all* balls in the reservoir as their subset to be tossed, which is equivalent to all processors accessing memory whenever they are able.

Note that the adversary doesn't know the outcome of future random events, and therefore cannot exploit any foreknowledge of the future.

Note also that the adversary can leave a ball in a bin indefinitely, as long as there is at least one other ball in that bin. Equivalently, the adversary can make a given processor wait "forever" for its access to M_i as long there is always at least one other processor contending for M_i. The chance of this scenario occurring depends on the sizes of P and M. For example, if $M = 1$ and $P \geq 2$ than the adversary can always starve a given processor.

We now define ***delay***. Let b_j be the number of balls in bins immediately after turn j. Then the ***total delay***, D, is given by

$$D = \sum_j b_j \ . \tag{1}$$

Figure 2 illustrates the actions in a 3-turn sample game where $P = 5$, $M = 4$, and $T = 8$. The total delay D for this game is 3, which is the sum of $b_1 = 1$, $b_2 = 2$, and $b_3 = 0$. The adversary's strategy for this game is not optimal: balls are left in the reservoir after part (a) of turns 1 and 3. Tossing these balls may have allowed the adversary to increase the total delay.

3 Bounds for the Expected Total Delay

We can bound the expected total delay in terms of the number of balls (processors) P, bins (memory banks) M, and tosses (total accesses) T.

Theorem 1 *The expected total delay* $\mathrm{E}[D]$ *is bounded by*

$$\mathrm{E}[D] \leq \frac{PT}{M} \ .$$

In particular, if $P = M$*, then since* $\mathrm{E}[D] \leq T$*, the expected total delay is, at most, the number of tosses.*

If we perform the tosses over a period of time, we will find that a constant fraction of bins has 0 balls and a constant fraction of bins has 1 ball. The fractional amounts will fall off exponentially as the balls pile up.

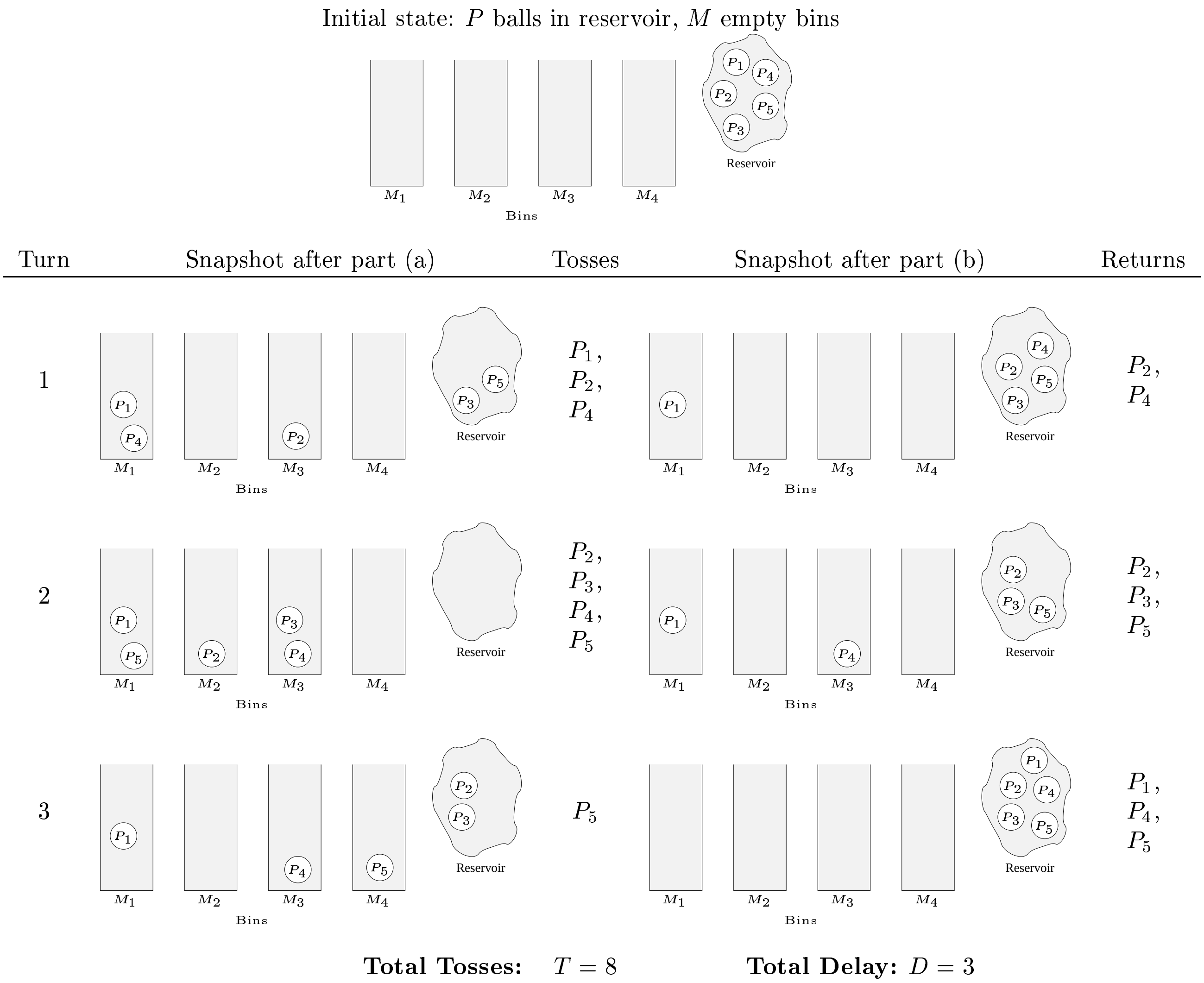

Figure 2: Three turns of the (P, M, T) recycling game with $P = 5$ balls, $M = 4$ bins, and $T = 8$ tosses. Bins are on the left and the reservoir is on the right. Circles represent balls. Each snapshot represents the state of the game after part (a) of a turn (on the left) or part (b) of a turn (on the right). Time advances toward the bottom of the page.

CLRS contains an expanded discussion of the balls and bins game in section 5.4.2. A summary of Chapter 5 and Appendix C from CLRS is provided below in Appendix A.

Proof Without loss of generality, the adversary's strategy in part (b) of the game is FIFO: the returned ball is always the one which has been in the bin the longest. In terms of total delay, it doesn't matter which ball is chosen in part (b), because the number of balls in a bin is the same regardless of the identity of the ball which is removed.[1]

Let δ_r be the number of steps after which ball r finishes in a bin (not a reservoir). Then, the total delay is

$$D = \sum_{r=1}^{P} \delta_r \ , \tag{2}$$

which is just another way of looking at the total-delay summation from Equation (1).

Let the ***ith cycle*** of a ball r be those steps in which ball r remains in a bin from its ith toss to its removal. Then we say that the ***ith delay*** of a ball is the number of steps in the ith cycle. This quantity may be 0 if the ball went into a bin and was immediately removed.

All balls are symmetric, so let's focus on the delay of the *first* ball. Let $\delta = \delta_1$ (delay of ball 1), and let t_1 be the number times ball 1 is tossed.[2] Let $d_i \geq 0$ denote the ith delay of ball 1; i.e. how long this ball stays in a bin during the ith cycle. Therefore, we have

$$\delta = \sum_{i=1}^{t_1} d_i \ ,$$

which is to say that the sum of all the cycle delays of ball 1 is the total delay of ball 1.

We say the ith cycle of ball 1 is ***delayed*** by ball r if ball 1 and ball r are in the same bin when r is removed. Recall that we're assuming FIFO order, so balls are delayed *once* or *not at all*.

This binary choice means we can use ***indicator random variables***, which are random variables that only take on two values, typically 0 and 1. Indicator random variables have many nice properties that make them easy to deal with [CLRS, 5.2]. We'll be seeing more IRVs throughout the semester.

We define an indicator random variable x_{ir} as follows:

$$x_{ir} = \begin{cases} 1 \text{ if } i\text{th cycle of ball 1 is delayed by } r, \\ 0 \text{ otherwise.} \end{cases}$$

Then, we have

$$d_i = \sum_{r=2}^{P} x_{ir} \ ,$$

and

$$\delta = \sum_{i=1}^{t_1} \sum_{r=2}^{P} x_{ir} \ . \tag{3}$$

By the definition of expectation [see Appendix A.3 below],

$$\begin{aligned} \mathrm{E}[x] &= 1 \cdot \Pr[x = 1] + 0 \cdot \Pr[x = 0] \\ &= \Pr[x = 1] \ , \end{aligned}$$

when x is an indicator random variable. Consequently, we have

$$\begin{aligned} \mathrm{E}[x_{ir}] &= \Pr[x_{ir} = 1] \\ &= \Pr[i\text{th cycle of ball 1 is delayed by ball } r] \ . \end{aligned}$$

[1] The choice of FIFO order produces the same statistics as any other queuing discipline (adversarial strategy).

[2] Note that the number of tosses is $T = \sum_{r=1}^{P} t_r$.

The probability that the ith cycle of ball 1 is delayed by ball r is either $1/M$ or 0: if r is in a bin when ball 1 is tossed, the probability that we get tossed in that same bin is $1/M$; and if r is in the reservoir instead, the probability of landing in a bin with it is 0. Although a tighter bound on $\mathrm{E}[x_{ir}]$ is possible, the loose bound

$$\mathrm{E}[x_{ir}] \leq \frac{1}{M}$$

is adequate for our purposes.

Using this bound and Equation (3), we obtain

$$\begin{aligned} \mathrm{E}[\delta] &\leq \sum_{i=1}^{t_1}\sum_{r=2}^{P} x_{ir} \\ &= \sum_{i=1}^{t_1}\sum_{r=2}^{P} \frac{1}{M} \\ &\leq t_1 \frac{(P-1)}{M} \\ &\leq t_1 \frac{P}{M} \end{aligned}$$

By symmetry, if t_r is the number of tosses of ball r, then $\mathrm{E}[\delta_r] \leq t_r P/M$. Then, using Equation (2), we have

$$\begin{aligned} \mathrm{E}[D] &= \sum_{r=1}^{P} \mathrm{E}[\delta_r] \\ &\leq \sum_{r=1}^{P} t_r \frac{P}{M} \\ &= \frac{PT}{M} \end{aligned} \qquad (4)$$

□

In summary: the good news is that delay doesn't hurt you much; memory contention is not the biggest problem that we face in parallel processing. It is more important to worry about load-balancing, which we will cover later this semester.

Alternative proof: The expected ith delay of ball one, $\mathrm{E}[d_i]$, is the expected number of balls in a bin. By linearity of expectation,[3] $\mathrm{E}[\text{number of balls in bin}] \leq P/M$. Then, since we have $\delta = \sum_{i=1}^{t_1} d_i$ and $\mathrm{E}[d_i] \leq P/M$, we obtain

$$\mathrm{E}[\delta] \leq t_1 \frac{P}{M} \ ,$$

and Equation (4) immediately follows.

References

[1] Robert D. Blumofe, "Executing Multithreaded Programs Efficiently", Ph.D. thesis, Department of Electrical and Computer Science, Massachusetts Institute of Technology, September 1995.

[3] See CLRS, Appendix C, p 1108.

A Summmary of CLRS Chapter 5: Probabilistic Analysis and Randomized Algorithms.

As a background for our consideration of the Memory Contention Problem, Professor Leiserson recommended that we take a look at CLRS Chapter 5 and Appendix C. Chapter 5, Probablistic Analysis and Randomized Algorithms, outlines the general process for analyzing quantities, such as the cost of a particular endeavor, or the run time of an algorithm. We are shown how to use prior knowledge of, or make assumptions about, the distribution of the inputs. The expectation is then taken over the distribution of the possible inputs. In the end, we average the quantity (e.g. in terms of cost or run time) over all possible inputs.

A.1 Example 1: The Hiring Problem

CLRS describes a situation where your company needs to hire a new office assistant. Management has decided to use an employment agency for the candidate search. The agency will send one candidate for an interview every day. There is a small cost associated with conducting the interview and there is a larger cost for firing your current assistant and hiring a new assistant through the agency. You are looking for the best possible candidate, so you are willing to absorb these costs. Still, you want to gain a better understanding of what the costs will be.

Clearly, the overall costs will be related to how many interviews are conducted and how many replacements are made. If c_1 is taken as the cost of interviewing, n the number of candidates interviewed, and m the number of people hired, then

$$O(nc_1 + mc_h) \ ,$$

This equation represents the total cost associated with the interviewing and hiring process.

In the worst case, we decide to hire every candidate that we interview, for a total hiring cost of $O(nc_h)$. This assumes that each new candidate is better than the last and that we are never satisfied with our current assistant. It is more likely that applicants will be interviewed in more or less random order (e.g. that a reputable agency will not send them to us for the interview in worst to best order). So the queue of candidates will form a uniform random permutation, spreading the probability of any given permutation across the total range of possible permutations, $n!$.

A.2 Example 2: A More Complex Hiring Problem

Instead of office assistants, consider the case of a university (let's say Harvard) that is planning to hire a new professor. As with the office assistant, there will be a small cost to interview candidates and a larger cost to replace the existing professor. However, we also know that the most qualified candidates will be more aggressive about negotiating their compensation packages, so now there is some variability in the total cost structure. How would you account for this variability (within certain reasonable bounds) based on the initial equation of $O(nc_1 = mc_h)$ as above? Assume that if you retain your existing professor, he will not negotiate a raise successfully.

A.3 Indicator Random Variables

Indicator Random Variables (IRVs) provide a convenient method for converting between probabilities and expectations. Suppose that we are given a sample space S and an event A,

$$I\{A\} = \begin{cases} 1 \text{ if A occurs,} \\ 0 \text{ if A does not occur.} \end{cases}$$

Taking the results of a coin toss as an example, let's say that $I\{A\} = 1$ if Y = Heads (H) and 0 if Y = Tails (T). Then,

$$
\begin{aligned}
\mathrm{E}[X_h] &= \mathrm{E}[I\{Y = H\}] \\
&= 1 * \Pr[Y = H] + 0 * \Pr[Y = T] \\
&= 1 * \frac{1}{2} + 0 * \frac{1}{2} \\
&= \frac{1}{2}
\end{aligned}
$$

If we continue to toss the coin and are interested in the number of heads obtained, we will find that multiple coin toss trials are expressed by

$$X = \sum_{i=1}^{n} X_i$$

where X is the random variable denoting the total number of heads in the n coin tosses.

If you wish to calculate the expected number of heads, take the expectation of both sides of the equation:

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{n} X_i\right],$$

We will find that the expected number of heads is $n/2$.

Later in Chapter 5, we see how the probability of streaks (consecutive heads in the coin toss, for example) may be calculated. See CLRS Section 5.4.3.

In conclusion, linearity of expectations makes the use of IRVs a powerful analytical technique; it applies even when there is dependence among the random variables.

B Further Summary of Chapter 5 and Appendix C

Table of additional topics covered in CLRS, Chapter 5 and Appendix C.

5.3 *Randomized Algorithms.* This section introduces a method to analyze problems where we do not know the distribution of the inputs and therefore cannot analyze the average-case behavior of an algorithm. In one method, we permute the given input of an array to produce a randomly permuted array.

5.4 *Probabilistic analysis and further uses of indicator random variables.* This section introduces more advanced concepts in probabilistic analysis, particularly analysis where arrays are used to predict frequency of a given attribute in a closed set or population. The Birthday Paradox provides a clear example of this operation and leads to a discussion of the Balls and Bins game which we have outlined in the notes above.

C *Counting and Probability.* Provides a review of many basic concepts including:

- C.1 Counting
  - i. Strings
  - ii. Permutations
  - iii. Combinations
  - iv. Binomial coefficients
  - v. Binomial bounds
- C.2 Probability

i. Axioms of probability
ii. Discrete probability distributions
iii. Continuous uniform probability distribution
iv. Conditional probability and independence
v. Baye's Theorem

C.3 Discrete Random Variables

i. Probability density function
ii. Joint probability density function
iii. Expected value of a random variable
iv. Variance and standard deviation

C.4 The Geometric and Binomial Distributions

i. The geometric distribution of a Bernoulli Trial
ii. The binomial distribution

C.5 The Tails of the Binomial Distribution

i. More Bernoulli Trials