6.895 Theory of Parallel Systems Lecture 15

Snoopy caching and the Spin-Block Problem

Lecturer: Bradley C. Kuszmaul and Michael Bender *Scribe: Sharad Ganesh and Neelkamal*

Lecture Summary

1. *Competitive snoopy cache proof*
   This section discusses the proof by induction of snoopy bus-based caching algorithm.

2. *Directory-based caching systems*
   This section gives a brief overview of the algorithm and its scalability as compared to the snoopy cache algorithm.

3. *Randomized algorithm for the Ski-rental Problem*
   This section introduces the Spin-Block problem, which is a continuous version of the Ski-buy problem.

1 Strategy for the proof (Snoopy caching)

A competitive algorithm that is within a factor of 2. The idea is to keep these buckets such that:

The economics of the algorithm:

$W_i(B) = p$; whenever we have to fetch block B from memory(M).

$W_j(B)--$; when i writes and j has money in its cookie-jar. Whenever I do a write, I decrement somebody else's cookie jar. At some point of time they are going to put money into the cookie-jar, so I am actually going to charge them something. If nobody has any money left, then they all have to invalidate their copies in their caches and then I will not have to do a broadcast in future.

$W_i(B) = 0$; when I drop a block. A reason I could drop a block is the possiblility of a conflict with another block or maybe somebody else decremented my cookie-jar count to 0.

$W_i(B)++$; We increment the count upto a point(only upto p), when we read a block.

Some facts about the algorithm

It does tell us when there is going to be a switch from a write-through to a write-back. Basically, you do a switch from pack-rat, where everybody tries to hang onto everything, to the exclusive write where I can do writes to my local cache and not tell anybody. Basically, I have to broadcast(keep updating everybody) as long as everyone is reading.

We are not too far off from the optimal, because everytime I did one of these operations that add money, there was already doing an operation that cost me that much (anyway $W_i(B) = p$ cost p cycles). So i'm putting money into the cookie-jar(i.e. p).
So, anybody who is going to do work and steal money from my cookie-jar, that work just charge it to me. Otherwise, anytime I increment it, because I did a read, its saying if it weren't in the cache, I'd be happy to pay something to have it gotten into the cache, because the read would really have cost me a lot more.

1.1 Strategy

Define a potential function, which function that tells us how far off our algorithm is from the offline algorithm A.
This is a function of the cache states of algorithm A(which is the one we are trying to beat - to come up within a factor of 2 of A). Those cache states after processing i steps in sequence of operations(reads and writes). The sequence we are talking about is:

- τ: which is the ops for A for the first read/write (tagged "A")
- ops for dsc for the first read / write (tagged "dsc")
- supply / update at the end of the first read / write. (tagged "both" - because both have to do it)

Basically both the algorithms are constrained in the sense that, when they see the first read, they can do whatever they want to, but you have to end with the supply which gives the value to the processor.

We will be taking all the operations for A and tag them as specified in the bracket as tagged "A". It is important to know the order in which the ops happened.

1.2 What is this potential function ?

$$\phi(t) = \sum_{(i,B) \in S_A} (w_i(B) - 2p) + \sum_{(i,B) \notin S_A} -w_i(B). \tag{1}$$

where: $(i, B) \in S_A =¿$ A has B in cache i at step t. Basically, the set of things that A caches.

Recall $w_i(B) = 0$ iff it is not in cache i. So, no matter how big the memory is, there is only a certain amount of non-zero values in the sum in equation (1).

1.3 Proof by induction of the following statement

$$COST_{dsc}(t) - 2COST_A(t) \leq \phi(t) - \phi(0). \tag{2}$$

where:
$COST_{dsc}(t)$: cost of doing dsc upto time t
$COST_{dsc}(t)$: cost of doing A upto time t
RHS : Amount the potential function has increased since we started.

Theorem 1

$$COST_{dsc}(t) \leq 2COST_A(t) + k$$

where: k is essentially the difference of the potential functions.

If cache started out with 0, i.e. completely empty, then k = 0. But, if cache started out with some value, that may favor A, k covers for that.

By construction:

$$\phi_t \leq 0. \tag{3}$$

Basically, the $W_{i's}(B)$ are between 0 and p. $(0 \leq w_i(B) \leq p)$.
Then we take some point of those numbers, subtract 2p and then $-w_i(B)$ makes it more negative.
So, we are taking a negative number + something having to do with the initial state.
Just let,

$$k = -\phi(0)$$

If the caches are completely empty at the beginning of the algorithm, $\phi(0)$ will turn out be be 0. So, we will later see that this will actually prove $k = 0$, if the cache starts out empty.

Proof
Base case for induction (t=0)

LHS:Neither algorithm has done anything yet, so their costs are both 0.
RHS: $\phi(0) - \phi(0) = 0$.

Both sides are 0.

Inductive case: Basically, we want to show something about the changes in the costs and how they are related to the change in potential function. We would want to show:

$$\Delta COST_{dsc} - 2\Delta COST_A \leq \phi$$

where:

$\Delta COST_{dsc}$ denotes the change in cost of dsc.
$\Delta COST_A$ denotes the change in cost of A.

We present a case analysis for the above:
Idea of the analysis:

We have the sequences, that we'd discussed earlier, and for everything in the sequence the property holds. i.e. We have to show that, if one of them drops a block or does a fetch or an update, the changes should have this property.

We will go through some of the cases, but for the rest you can refer to the Goodman's paper [1].

Case 1:
Step: A does a fetchblock(i, B), which is fetching over the bus to get a block into cache i of block B.

$$\Delta COST_A = p$$

$$\Delta COST_{dsc} = 0$$

This is because, dsc did'nt do anything

Now we have to show:

$$\Delta\phi \geq 2p$$

Before the action: $(i, B) \notin S_A$
This is because, none of these snoopy caches do a fetch unless they do a read / write. Moreover, they don't a fetch if they already got it cached. So, before the action it was'nt in the cache.

After the action: $(i, B) \in S_A$
After the action, it is in the cache.

So,

$$\Delta\phi = w_i(B) - 2p - wi(B)$$

Therefore, $\Delta\phi = -2p$
Therefore, whatever A does if it does a fetch, potential function does the right thing.

Case 2:
A drops (i, B)

At this point 'A' has more freedom to do something different than we did. 'A' might drop something that you didn't drop because we were still paying something and so could'nt drop it. A might decide to drop something and let the slot be occupied by something else. In that case, I won't be incurring any cost for update later.

$$\Delta COST_A = 0$$

$$\Delta COST_{dsc} = 0$$

Now we have to show:

$$\Delta\phi \geq 0$$

So we have,
$w_i(B) + 2p - w_i(B)$
The second $-w_i(B)$, because it is no longer in the cache.

$$\Delta\phi = -2w_i(B) + 2p$$

The above term is always greater than or equal to 0, because $0 \leq W_i(B) \leq p$.

Case 3:
dsc does a fetchblock(i, B)

$$\Delta COST_{dsc} = p$$

$$\Delta COST_A = 0$$

Now we have to show:

$$\Delta\phi \geq p$$

Therefore here, $wi(B)$ changes from 0 to p.
Before it was'nt in the cache($wi(B) = 0$), and then when we did a fetch we set($wi(B) =$ p).
Since we did A's operations first, we must have (i, B) in A's cache at the end of A's ops. Both algorithms can be supplied from the cache. The key point is since we did A's ops first, we know it is in the cache.

Therefore,

$$\Delta\phi = p$$

For the other cases, you can read the Goodman's[1] paper. □

2 Directory based caches

The problem with MESI is the bus.

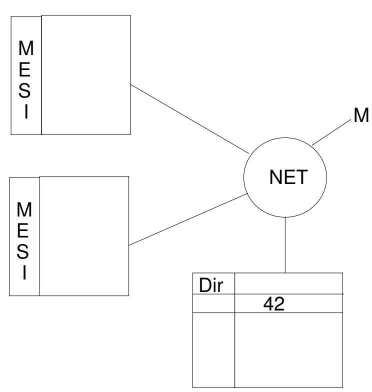

Figure 1: MESI network setup

Some processor wants to obtain E(exclusive) access, So send a message to M saying you want E access.

Case 1: Nobody has it. Memory says you have got it and changes the state.

No body has it in their cache
Processor k has it in E state
Processor 4, 7, 9 has it in S state

Figure 2: A typical directory entry

Case 2: Some other proc has it in E state. M sends message to the other processor. The other processor sends it to M. If it has been modified, it has to send it back to M. In any case M sends it to P. Real machines do a shortcut sending a message directly to the processor.

Directory based cache analysis
None.

3 Spin-Block Problem

There are basically two ways in which a process waits for a lock

1. Spin: keep on waiting in a nop loop till get the lock Cost = spin time
2. Block: You ask the lock manager to give u a lock and while he manages for the lock u go on and do your work and return after some time. Cost: fixed cost 'c'

4 Continuous ski rental problem

A simple algorithm based on randomized algorithm strategy.

4.1 Spin

```
1. until time c/2
2. With probability P, block
3. With probability 1-P, keep spinning until 'c' spins.
4. Then block
```

Definition 2 *f(t) = expected cost of waiting t units of time.*

$$f(t)=\begin{cases} t, & t < c/2; \\ p(c+c/2)+(1-p)t, & c/2 \le t < c; \\ p(c+c/2)+(1-p)*2c, & t \ge c. \end{cases}$$

Goal: Choose p to minimize competitive ratio (1+ α)
Setting the inequalities to equalities and solving:

$$f(t) = c/2 + pc = (1+\alpha)c/2 \tag{4}$$

$$f(t) = p(3/2c) + (1-p)*2c = (1+\alpha)c \tag{5}$$

$$p = 2/5 \tag{6}$$

$$1+\alpha = 9/5 \tag{7}$$

Good News: (1+ α) is less than 2.

Definition 3 ***Function*** $\Pi(t)$ ***= density function of time before a processor should block.***

Expected cost of waiting Q steps.

$$f(q) = \int_{t=0}^{q} dt \prod(t)(t+c) + q \int_{t=q}^{\inf} (dt\Pi(t)) \tag{8}$$

Solving the above equation:
Idea: choose $\Pi(t)$ **to minimum competitive ratio:**

$$f1(q) = (q+c)\Pi(q) + \int_{t=q}^{\inf} (dt\Pi(t)) - q\Pi(q) \tag{9}$$

$$f1(q) = c\Pi(q) + \int_{t=q}^{\inf} (dt\Pi(t)) - q\Pi(q) \tag{10}$$

$$f2(q) = c\Pi1(q) - c\Pi(q) = 0 \tag{11}$$

$$\Pi(q) = A \exp q/c \tag{12}$$

Calculating A

$$\int_{t=0}^{c} dt\Pi(t) = 1 \tag{13}$$

$$A \int_{t=0}^{c} \exp(t/c) dt \Rightarrow AC(e-1) = 1 \tag{14}$$

$$\Rightarrow A = 1/(C(e-1)) \tag{15}$$

Calculating 1+α

$$f(C)/C = A/C \times \int_{t=0}^{c} dt(t+C)\exp(t/c) \tag{16}$$

$$= A/C \times C^2 \tag{17}$$

$$= e/(e-1) \simeq 1.59 \tag{18}$$

Note: If you make this discrete space solution, then also u will get the same value

References

[1] **James R. Goodman. Using cache memory to reduce processor-memory traffic.**