Today we analyze an algorithm that uses a MESI-like protocol.

Setup:

- Processes and memory can see all messages
- Only one (for M) can send a message at a time
- Processes (P) talk to caches (S) which talk to bus.

Locality of reference: Spatial locality of a program

Spatial locality is the property that if a processor requests location \( v \), it is likely to operate on \( v+1 \) or other locations near \( v \).

Programs exhibit spatial locality => transmit blocks of size \( p \).

\[ p = 6/0.64 \text{ size} \]

If \( b = 1 \), fetch costs 2 cycles

\[ \text{read address} \rightarrow \text{read} \]

If \( 6 \) bytes, \( b = 6 + 1 \) cycles, saving \( 2x \) for roundly \( b \).

Blocks reside at \( > 1 \) processor copies. Reduces read cost.

To update a replicated block requires communication. Either invalidate the other copy or update them.

Two common strategies:

- Exclusive Write: Invalidate everyone else
- Peer-to-peer: Update every copy else

Worst-case behavior of exclusive-write:

\[ \begin{align*}
\text{Peers:} & \quad v \quad \text{Po write } v \\
\text{Peer:} & \quad v \quad \text{Po write } v \\
& \vdots \quad \vdots \\
& \text{local } v
\end{align*} \]

\[ \text{read} \rightarrow \text{local} \rightarrow \text{K}(u,v) \cdot p \]

\[ \text{peer} \rightarrow \text{update} \rightarrow \text{K}(u,v) \cdot p \]

Strategy: For this sequence is peer write.

\[ \text{Cost} = \frac{K + p + K + \text{all }} {K + p} + \frac{K + p} {K + p} \]
\[ \lim_{K \to 0} \frac{K \cdot (n-1)p}{n+np} = \frac{(n-1)p}{(n-1)p} \]

(Review:
\[ \lim_{a \to 0} \frac{a \cdot b}{a + c} = \lim_{a \to 0} \frac{(a+c) \cdot b - cb}{a + c} \]
\[ = \lim_{a \to 0} \frac{a \cdot cb}{ac} - \lim_{a \to 0} \frac{cb}{ac} \]
\[ = b - 0 \]
\[ = b \]

\[ \text{Beckers} \]
\[ B2 \text{ (Worst case analysis for packet):} \]
\[ Po \ w/d \]
\[ li \ w/d \]
\[ repd w/trx \]
\[ Po \ w/d \]
\[ Tots \ cost \geq w \]

Optimal is exist and can be achieved
Stop after waiting for \( c + d \) packets

\[ \lim_{w \to c} \frac{w}{c} \text{ is unbounded.} \]

\[ \text{Strategy: Competitive algorithm} \]

Def \( A_0 \): an online algorithm which \( c \)-compares \( A \)

\[ c \leq \text{cost} \left( A \right) \leq c \cdot \text{opt} + 1 \]

\[ \text{Optimal cost} \leq \text{cost} \left( A \right) \leq \text{opt} \cdot c + \text{constant on overhead} \text{ and to some} \]

\[ \text{Def: Strongly } \left( c \right) \text{-competitive } \Rightarrow \text{ no better } c. \]
Another algorithm: Gooden's algorithm

On first write, update everything else
On second write, invalidate existing cache

Analysis: Why is it hard to do something once?
Why not twice?

Strategy: Competitive Algorithm

Define: An online algorithm is $c$-competitive (or competitive) if

For any sequence of inputs

\[ \text{cost}(A) \leq c \times \text{cost}(\text{optimal offline algorithm}) \]

Define: An algorithm is an online algorithm if it must make its
behavior depend only on the inputs.
Define an algorithm that takes a sequence of inputs and produces a sequence of outputs.

Def. Such an algo is **online** if it can produce its i'th output now, having seen only the first i inputs, otherwise **offline**.

Intuitively, offline algo *looks into the future*.

Intuitively, offline algo *looks into the future*.

**Strategy:** Competitive Algorithm

**Def.** An online algorithm $A$ is $c$-competitive (or "competitive") if there exists a sequence of inputs $I$ such that

$$\text{cost}(A) \leq c \cdot \text{cost}(\text{optimal offline algorithm}) + d$$

Note: $c,d$ independent of sequence.

Def. **Strongly** $c$-competitive if no $d$ better $c$ can be found.
Example of a competitive algorithm: (Due to Jonn Rudolph)
Skiing: Ski rental = $10 to rent, $100 to buy.
What should I do?

- Buy ski? =) call so once and $100 too
- Rent skis? =) call and rental $1.

2-competitive algorithm:
Rent until 10 times < 10 times, then buy.
Analysis: if I ski > 10 times, the optimal alg is rent, which I did.
if I ski <= 10 times, the optimal alg is buy.

- Rent cost is $100
- $100 rent = $100, a total of 2 off.

Competitive Swoosh Cache, by Kuhn, Mcruise, Rudolph, Sleator
(No online copy to be found)

Examples from kids of caches.

I'll focus on direct-mapped swoosh cache.

- Direct mapped. i.e. if a future 1 request is the
  block #5 to a single slot in each cache.

- Caches swap on variables.
  
  That's the upper limit on
  
  On read, if not in cache, must fetch a
  whole cache line of 64 bytes.

On write, if we communicate, we send the new

Value for V over the

bus. This takes 1 cycle.
Only freedom: when shall a processor drop a block from its cache?

\[ \begin{array}{c}
\text{P}_0 \text{ with } b_{100} \text{ over } \\
\text{P}_1 \text{ may decide to drop the block so } \text{P}_1 \text{ is left. }
\end{array} \]

The only way for data to get into a cache is if the processor reads or writes it.

Once it is there, it must remain up-to-date.

\underline{Algorithm: Direct-Mapped - Sequential - Caching (dsc)}

\[ \begin{align*}
&\text{A data structure} \\
&\quad \begin{array}{c}
W_i[\text{B}] \quad \text{any type, } \\
\text{i is processor #} \\
\text{B is block #}
\end{array}
\end{align*} \]

\underline{Invariant:}

\[ \begin{array}{c}
\quad W_i[\text{B}] = 0 \quad \text{iff Block B is not in cache i.}
\end{array} \]

\underline{Algorithm:}

\[ \text{To read V on processor i:} \]

1. Let \( B = iV \)
2. If \( W_i(B) = 0 \) then GETBLOCK(i, B)
3. Else \( W_i(B) \leftarrow W_i(B) + 1 \quad \text{(actually can be incremented up to any value in memory)} \)
4. \( W_i(B) = \min(p, W_i(B)+1) \quad \text{(or any value in between)} \)
5. Provide B to cache.
To write \( v \) from processor \( i \):

```plaintext
let \( B = \{ v \} \)
if \( \omega_i(B) = 0 \) then GET\&LOC(i, B)
else \( \omega_i(B) = p \)

if \( \exists j \) s.t. \( j \neq i \) and \( \omega_j(B) \neq 0 \) then
- Update \( B \) in some other clock \( j \)
  - Broadcast the update on the bus.
  - \( \omega_j(B) \leftarrow j \)
  - If \( (\omega_j(B) = 0) \) then drop \( B \) from clock \( j \).
else
  Update local coy (now \( B \) is done)
```

Procedure GET\&LOC(i, B):

```plaintext
if \( \exists B' \) s.t. \( h(B) = h(B') \) and \( \omega_i(B) \neq 0 \)
- \( B \) collides with \( B' \)!
  - Any \( B' \)
if \( B' \) is done then
  Write \( B' \) into target
  \( C++ \)
  \( \omega_i(B') = 0 \)
  Drop \( (i, B') \)
Fetch \( B \) into copy
\( C++ \)
\( \omega_i(B) = p \)
```

**Two choices:** increment on \( B \) or \( B' \)
- Choice of \( j \): \( i \) or \( \omega_j \)
Theorem: Algorithm DSC is strongly $2$-competitive.

That is, for any sequence of read and write, and any (online or offline) algorithm $A$

$$\text{cost}_{\text{DSC}}(\sigma) \leq 2 \cdot \text{cost}_{A}(\sigma) + k$$

where $k = 0$ if all reads are initially exact.

We use a potential function $\Phi$

$$\Phi(t) = \sum_{(i, \theta) \in S}\left(\omega_{i}(\theta) - 2p\right) + \sum_{(i, \theta) \in S_{A}} -\omega_{i}(\theta)$$

where $S_{A}$ is the set of pairs $(i, \theta)$ such that $A$ has blue $\theta$ in color $i$ at step $t$.

Note $\Phi(t) \leq 0$

We'll prove inductively

$$\text{cost}_{\text{DSC}}(t) - 2 \cdot \text{cost}_{A}(t) \leq \Phi(t) - \Phi(0)$$

for $t = 0$, both sides are 0.

The inductive step is to show

$$\Delta \text{cost}_{\text{DSC}} - 2 \Delta \text{cost}_{A} \leq \Delta \Phi$$
Strategy: given $\mathcal{O}$ and $\mathcal{E}$, create a sequence $\mathcal{R}$

It is contained from $\mathcal{O}$ - $\mathcal{S}$

for each request in $\mathcal{O}$

Strategy: break up operations into

**Fetch block** $(i, B)$
- Block $B$ added to cache $i$, cost = $p$

**Drop** $(i, B)$
- Block $B$ is dropped from $i$. ($B + \lambda B$)
  cost = $0$

**Writeback** $(i, B)$
- Block $B$ is made clean. $B$ is added to $\mathcal{B}$
  cost = $p$

**Supply** $(i, V)$
- Variable $V$ is supplied to proc $i$ by its code.
  block $(V)$ must be in cache. cost = $0$

**Update loc** $(i, V)$
- Variable $V$ is updated in code $i$.
  block $(V)$ becomes dirty. cost = $0$

**Update global** $(i, V)$
- Variable $V$ is updated in code $i$ by block.
  block $(V)$ must be clean and in cache.
  cost = $1$

**Notes**

Algorithm for performing for $\mathcal{O}$

Any algorithm must perform these ops: for a read, end with supply $(i, V)$

for a write, end with update.

Consider $n$ a sequence of these steps.

$R = \text{subsequence of } \mathcal{R}$
A: Fetch(block(); Drop(); sup's(); ... \\
\text{disc: } \text{Fetch(block(); sup's();}

\text{Idea: line these smaller updates up, so we have a new scen}

\gamma =
\begin{cases}
\text{A fetch block} & \text{put A actions in first} \\
\text{A drop} & \\
\text{disc: fetch block} & \text{then disc actions} \\
\text{disc: sup's block} & \text{then to commit action}
\end{cases}

\text{Now define a potential function: spend w. each step of disc + A}
\text{after processing the first t steps of \gamma}

\phi(t) = \sum_{(i,b) \in A} \sum_{(i,b) \in A} (w_i(b) - 2p) + \sum_{(i,b)} -w_i(b)

\text{not: } \phi(t) \leq 0

\text{Proof (i) induction}
\text{cost}_{disc} (t) - 2 \text{cost}_A (t) \leq \phi(t) - \phi(0)

\text{Theorem will follow if } \quad k = \phi(0) \quad \text{since } \phi(t) \leq 0
CASE CASE: \( t = 0 \), both \( x_i = 0 \)

**Inductive Case:**

Show

\[
\Delta \text{cost}_{\text{disc}} - 2 \Delta \text{cur}_{A} \leq \Delta \Phi
\]

**Case Analysis:**

If step \( i \) is "A does fetch block \((i, B)\)"

\[
\Delta C_A = p
\]

must show \( \Delta \Phi \geq -2p \)

Before this \( A \) checks \((i, B)\) in case \( i \) in \( A \).

Afterward, it is in.

There is

\[
\Delta \Phi = \omega_i(B) - 2p - (\omega_i(B))
\]

\[
\geq -2p.
\]

Step \( i \) is "A drops \((i, B)\)"

\[
\Delta C_A = 0
\]

must show \( \Delta \Phi \geq 0 \)

Before \((i, B)\) in case \( i \), after \((i, B)\) in case \( i \).

because \((i, B)\)

\[
\Delta \Phi - \omega_i(B) \geq 0
\]

"A write block \((i, B)\)"

\[
\Delta C_A = p
\]

must show \( \Delta \Phi \geq -2p \)

but \( \Delta \Phi = 0 \).
Deficiency (s, t)

\[ \Delta_{stc} = P \]

must show \( \Delta \phi \geq P \)

\( w_{i}(B) \) changes from 0 to \( P \).

Because we did \( A \)'s first, \( A \) must have \( B \) in cell \( i \) now.

So \( \Delta \phi = P \)

Both supply \((i, r)\) to \( i \).

Cost to \( Gola = 0 \)

\[ \Delta \phi = (\Delta w_{i}(B)) = 0 \]

since it is a flow.

\( B \) in \( i \) so \( \Delta \phi = 0 \).
Directory-Based Cache

Problem with MESI: doesn't scale

Solution: Add onto top cache block in memory

Idea: Leave info at memsys to identify who needs to be updated when cache must be invalidated or updated

Example:

A
Nobody by 1

B
"Proc 5 by 1"

C
Proc 3, 5, 9 have S

To obtain E across for proc 2

case A: Fetch from mem, set M to "2 has it in M"

case B: Send message to proc 5

proc 5 sends back to mem, then case A

case C: Send message to 3, 5, 9 to invalidate (in parallel)

3, 5, 9 send to 2

2 then proceeds to write. NO ANALYSIS!