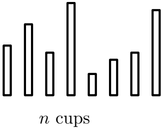
The Variable-Processor Cup Game

Alek Westover

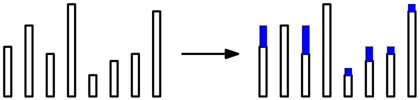
Belmont High School

June 7, 2020

p-PROCESSOR CUP GAME ON n CUPS



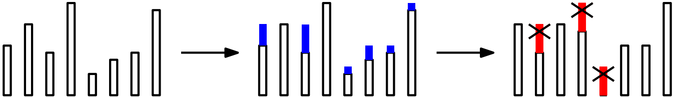
p-PROCESSOR CUP GAME ON n CUPS



n cups

filler adds p units of water
(with at most 1 unit per cup)

p-PROCESSOR CUP GAME ON n CUPS

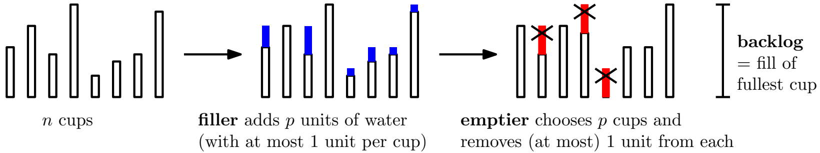


n cups

filler adds p units of water (with at most 1 unit per cup)

emptier chooses p cups and removes (at most) 1 unit from each

p-PROCESSOR CUP GAME ON n CUPS



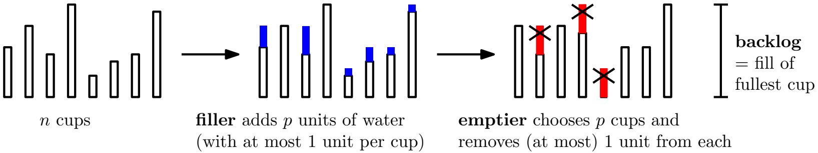
n cups

filler adds p units of water (with at most 1 unit per cup)

emptier chooses p cups and removes (at most) 1 unit from each

backlog = fill of fullest cup

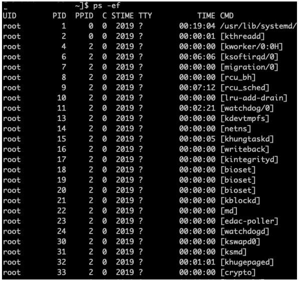
p-PROCESSOR CUP GAME ON n CUPS


n cups
filler adds p units of water (with at most 1 unit per cup)
emptier chooses p cups and removes (at most) 1 unit from each
backlog = fill of fullest cup

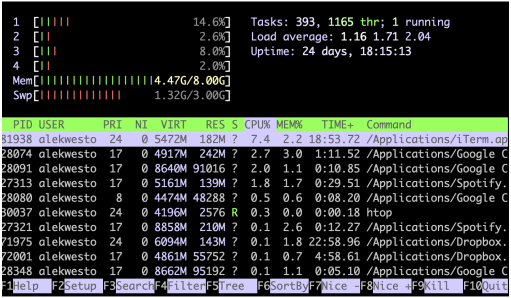
- filler: wants high backlog
- emptier: wants low backlog

In this talk we take the side of the filler (so high backlog is good)

Example Application: Work Scheduling

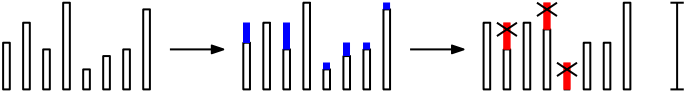


```
                 ~]$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0  2019 ?        00:19:04 /usr/lib/systemd/
root         2     0  0  2019 ?        00:00:01 [kthreadd]
root         4     2  0  2019 ?        00:00:00 [kworker/0:0H]
root         6     2  0  2019 ?        00:06:06 [ksoftirqd/0]
root         7     2  0  2019 ?        00:00:00 [migration/0]
root         8     2  0  2019 ?        00:00:00 [rcu_bh]
root         9     2  0  2019 ?        00:07:12 [rcu_sched]
root        10     2  0  2019 ?        00:00:00 [lru-add-drain]
root        11     2  0  2019 ?        00:02:21 [watchdog/0]
root        13     2  0  2019 ?        00:00:00 [kdevtmpfs]
root        14     2  0  2019 ?        00:00:00 [netns]
root        15     2  0  2019 ?        00:00:05 [khungtaskd]
root        16     2  0  2019 ?        00:00:00 [writeback]
root        17     2  0  2019 ?        00:00:00 [kintegrityd]
root        18     2  0  2019 ?        00:00:00 [bioset]
root        19     2  0  2019 ?        00:00:00 [bioset]
root        20     2  0  2019 ?        00:00:00 [bioset]
root        21     2  0  2019 ?        00:00:00 [kblockd]
root        22     2  0  2019 ?        00:00:00 [md]
root        23     2  0  2019 ?        00:00:00 [edac-poller]
root        24     2  0  2019 ?        00:00:00 [watchdogd]
root        30     2  0  2019 ?        00:00:00 [kswapd0]
root        31     2  0  2019 ?        00:00:00 [ksmd]
root        32     2  0  2019 ?        00:01:01 [khugepaged]
root        33     2  0  2019 ?        00:00:00 [crypto]
```



```
  1  [|||||                    14.6%]   Tasks: 393, 1165 thr; 1 running
  2  [||                        2.6%]   Load average: 1.16 1.71 2.04
  3  [|||                       8.0%]   Uptime: 24 days, 18:15:13
  4  [||                        2.0%]
  Mem[|||||||||||||||||4.47G/8.00G]
  Swp[|||||||||||||     1.32G/3.00G]

  PID USER      PRI  NI  VIRT   RES S CPU% MEM%   TIME+  Command
81938 alekwesto  24   0 5472M  182M ?  7.4  2.2 18:53.72 /Applications/iTerm.ap
28074 alekwesto  17   0 4917M  242M ?  2.7  3.0  1:11.52 /Applications/Google C
28091 alekwesto  17   0 8640M 91016 ?  2.0  1.1  0:10.85 /Applications/Google C
27313 alekwesto  17   0 5161M  139M ?  1.8  1.7  0:29.51 /Applications/Spotify.
28080 alekwesto   8   0 4474M 48288 ?  0.5  0.6  0:08.20 /Applications/Google C
30037 alekwesto  24   0 4196M  2576 R  0.3  0.0  0:00.18 htop
27321 alekwesto  17   0 8858M  210M ?  0.1  2.6  0:12.27 /Applications/Spotify.
71975 alekwesto  24   0 6094M  143M ?  0.1  1.8 22:58.96 /Applications/Dropbox.
72001 alekwesto  17   0 4861M 55752 ?  0.1  0.7  4:58.61 /Applications/Dropbox.
28348 alekwesto  17   0 8662M 95192 ?  0.1  1.1  0:05.10 /Applications/Google C
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```



n tasks
(cups)

new work arrives
(filler)

allocate p processors to tasks
(emptier)

backlog
= farthest behind
on any task

Previous Work [1,2]

Adaptive filler: can see emptier's actions

Theorem

With an adaptive filler optimal backlog is $\Theta(\log n)$.

Oblivious filler: can **not** see emptier's actions

Theorem

With an oblivious filler optimal backlog is between $\Omega(\log \log n)$ *and* $O(\log \log n + \log p)$ *(with high probability in short games).*

[1] [William Kuszmaul. Achieving optimal backlog in the vanilla multi-processor cup game. SODA, 2020.]

[2] [M. Bender, M. Farach-Colton, and W. Kuszmaul. Achieving optimal backlog in multi-processor cup games. In Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC), 2019.]

This Talk

Our Question: What if p can change?

Variable-Processor Cup Game:
Each round the filler can change p

Modification seems small...

OUR RESULT

The variable-processor cup game is *fundamentally different* than the p-processor cup game!

Adaptive Filler Lower Bound on Backlog

Theorem

There is an adaptive filling strategy that achieves backlog

$$\Omega(n^{1-\epsilon})$$

for any constant $\epsilon > 0$ *in running-time*

$$2^{O(\log^2 n)}.$$

Adaptive Filler Lower Bound on Backlog

Theorem

There is an adaptive filling strategy that achieves backlog

$$\Omega(n)$$

in running-time

$$O(n!).$$

Upper Bound on Backlog

Corollary

A greedy emptier never lets backlog exceed

$$O(n).$$

This matches our lower bound!

Corollary follows from more general theorem:

Theorem

A greedy emptier maintains the invariant:

$$\textit{Average fill of k fullest cups } \leq 2n - k.$$

Oblivious Filler Lower Bound on Backlog

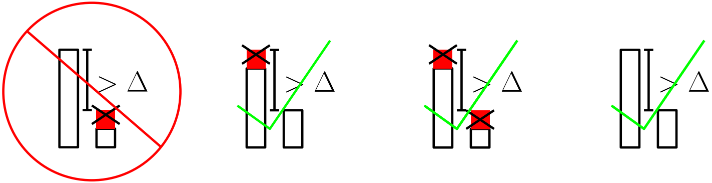
Theorem

There is an oblivious filling strategy that achieves backlog

$$\Omega(n^{1-\epsilon})$$

for constant $\epsilon > 0$ with probability at least $1 - 2^{-\operatorname{polylog}(n)}$ in running time $2^{O(\log^2 n)}$ against a greedy-like emptier.

Δ-greedy-like emptier:



Adaptive Filler Lower Bound Proof Sketch

Amplification Lemma

Lemma

Given a strategy f for achieving backlog $f(n)$ on n cups, we can construct a new strategy f' that achieves backlog

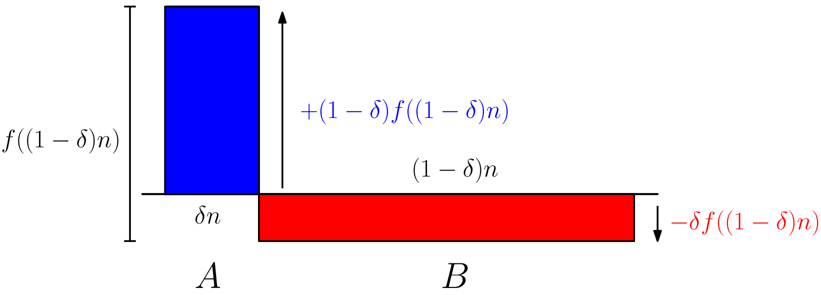
$$f'(n) \geq (1-\delta)\sum_{\ell=0}^{L} f(n\delta^{\ell}(1-\delta))$$

for appropriate parameters $L \in \mathbb{N}, 0 < \delta \ll 1/2$.
If the running time of $f(n)$ is $T(n)$ the running time of $f'(n)$ satisfies

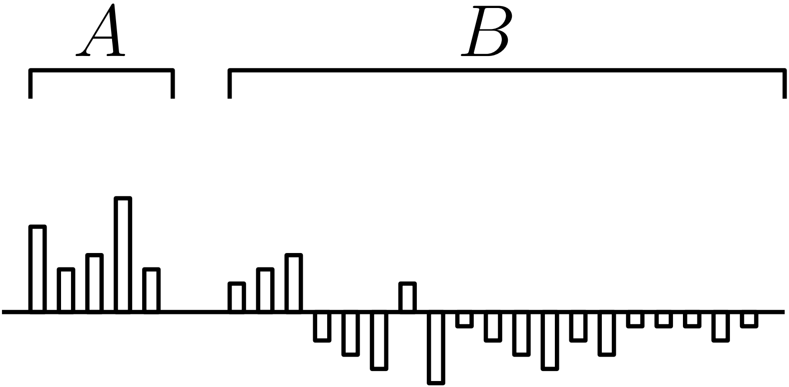
$$T'(n) \leq n \sum_{\ell=0}^{L} n\delta^{\ell} T(n\delta^{\ell}(1-\delta)).$$

Proof Meta-Structure

- A starts as the δn fullest cups, B as the $(1-\delta)n$ other cups.
- Repeatedly apply f to B and swap generated cup into A.
- Decrease p, recurse on A.

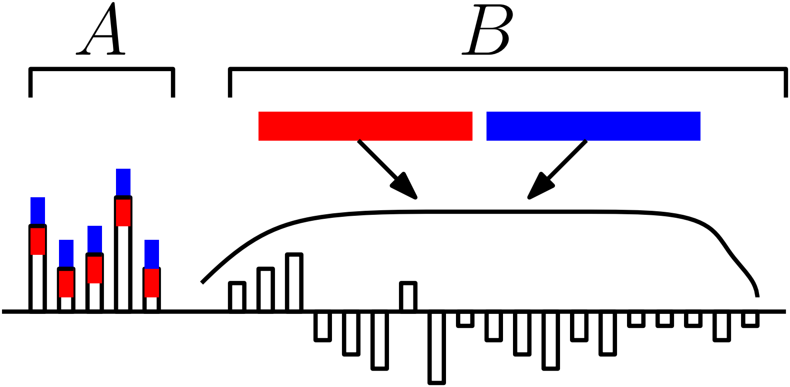


Amplification Lemma Proof Sketch



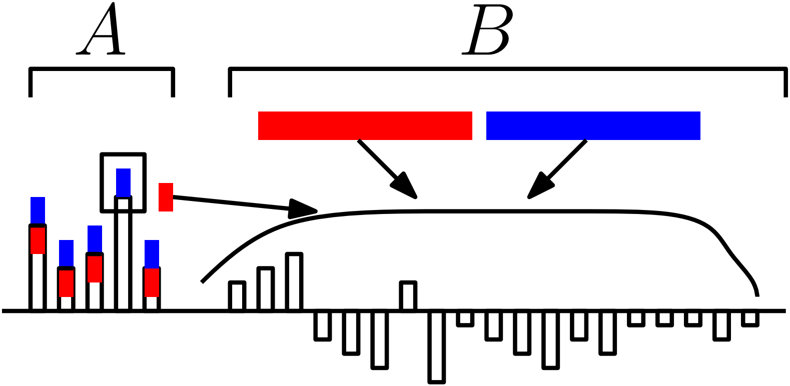
Instantiate *A* and *B*

Amplification Lemma Proof Sketch



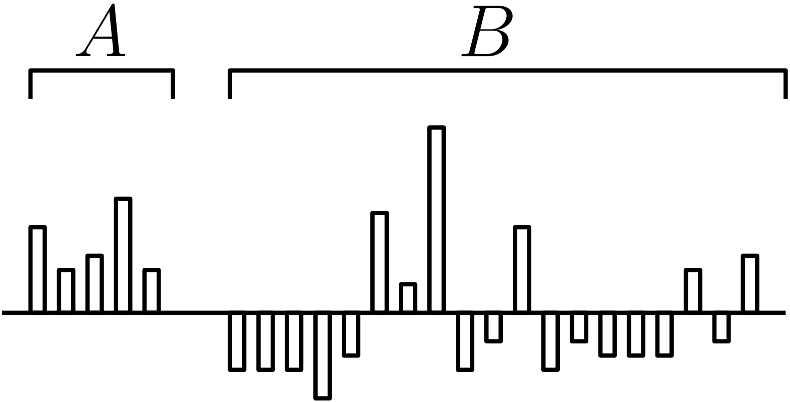
Filling Strategy: Place 1 fill in each cup in A, try to apply f to B.

Amplification Lemma Proof Sketch



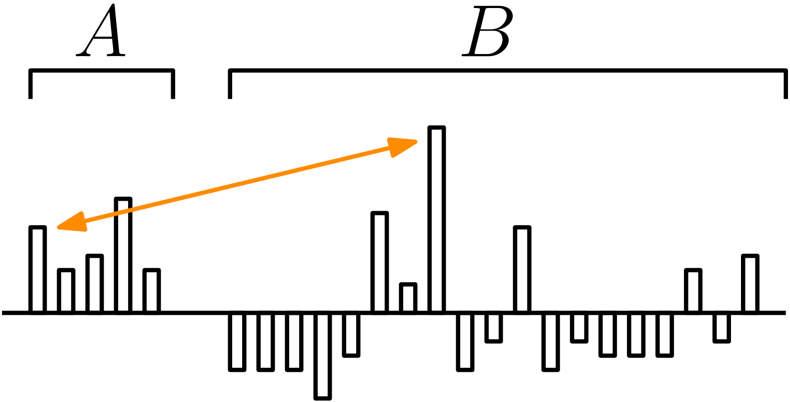
If the emptier ***neglects*** A then the average fill of A rises!
We repeat our strategy many times; if the emptier neglects A too many times we get the desired backlog in A.

Amplification Lemma Proof Sketch



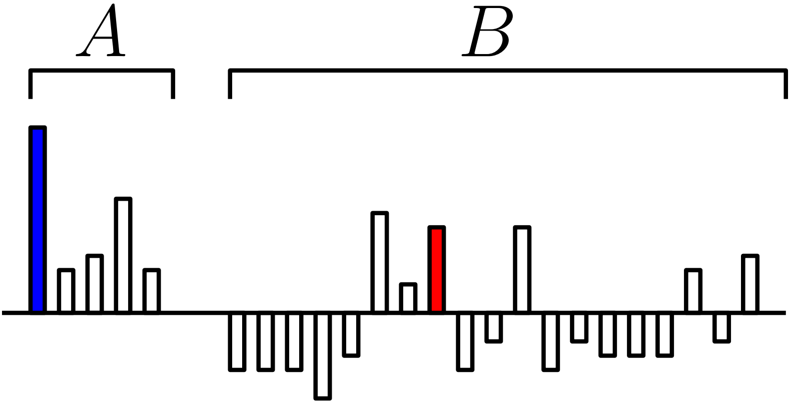
If emptier doesn't neglect A filler can apply f to B

Amplification Lemma Proof Sketch



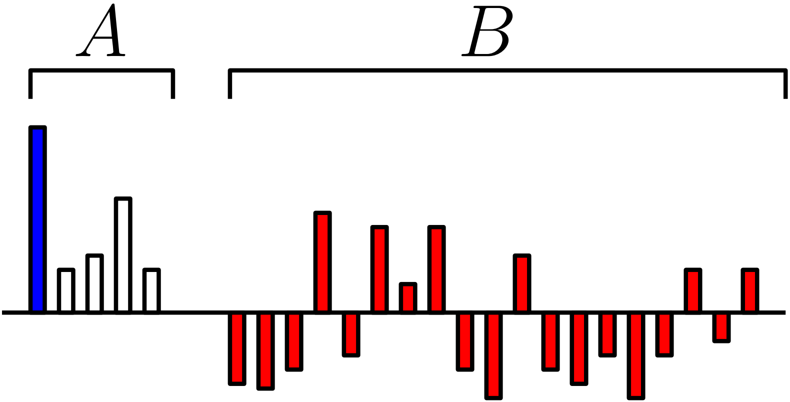
Get a cup with high fill in *B*, swap it into *A*

Amplification Lemma Proof Sketch



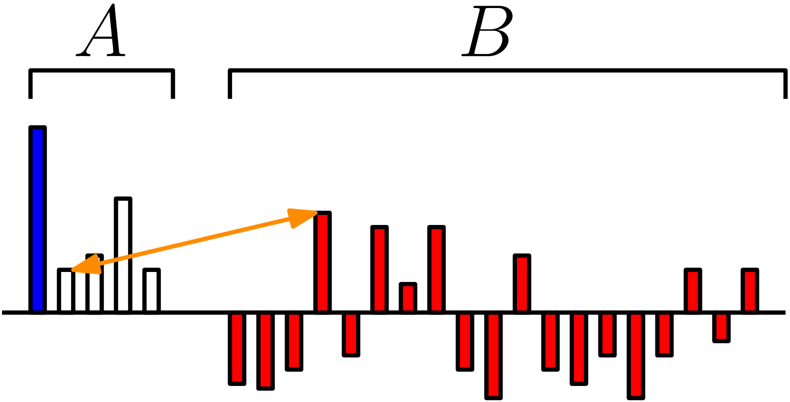
Note: swaps increase average fill of A, decrease average fill of B.

Amplification Lemma Proof Sketch



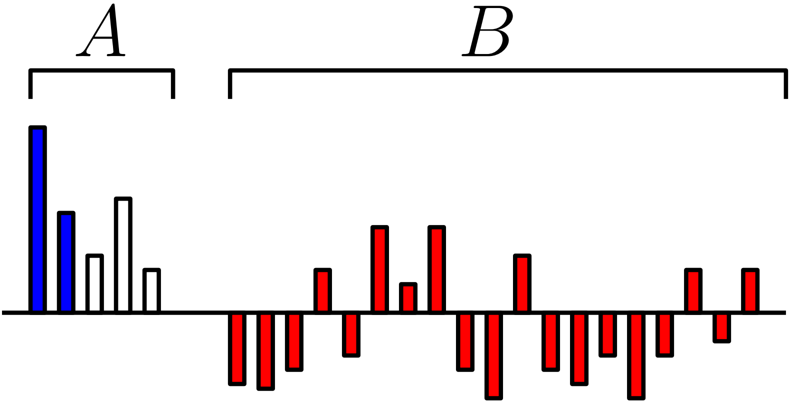
Apply f to B again

Amplification Lemma Proof Sketch



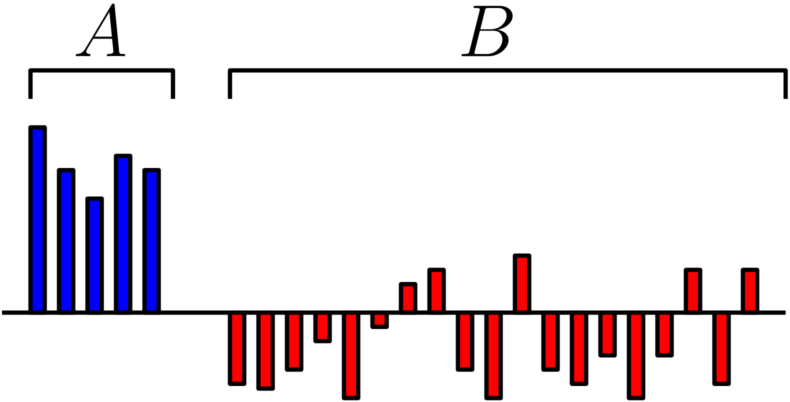
Swap cup into A again

Amplification Lemma Proof Sketch



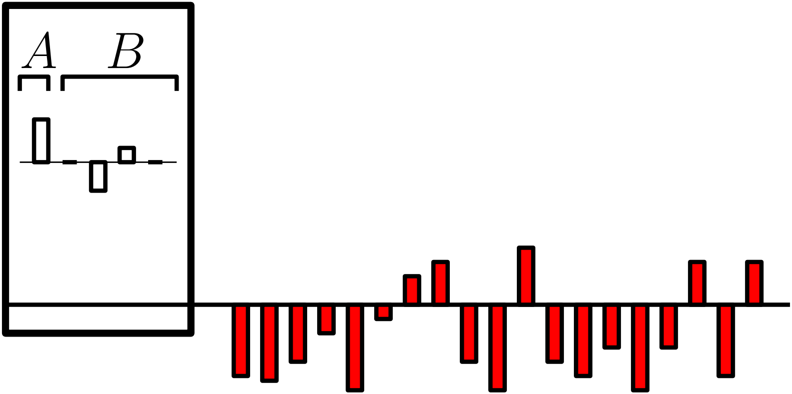
Swap this cup into *A*.

Amplification Lemma Proof Sketch



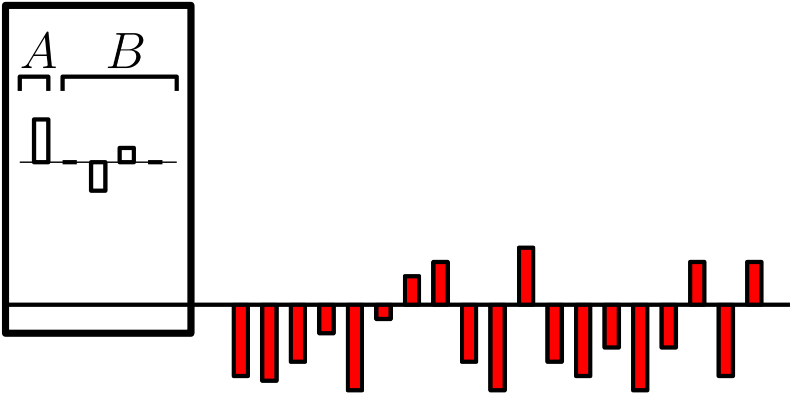
Eventually average fill of A is at least $(1-\delta)f(n(1-\delta))$.

Amplification Lemma Proof Sketch



Recurse on A for L levels of recursion.
Problem size shrinks by a factor of δ each time.

Amplification Lemma Proof Sketch



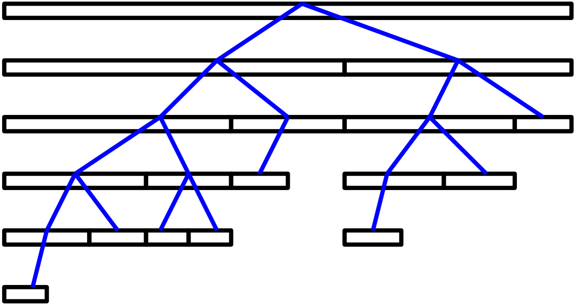
$$f'(n) \geq (1-\delta)\sum_{\ell=0}^{L} f(n\delta^{\ell}(1-\delta))$$

Adaptive Filler Lower Bound

Let $\epsilon > 0$ be any constant. There exists an appropriate $\delta = \Theta(1)$ such that by repeated amplification we get:

Theorem

There is an adaptive filling strategy that achieves backlog $\Omega(n^{1-\epsilon})$ in running-time $2^{O(\log^2 n)}$.

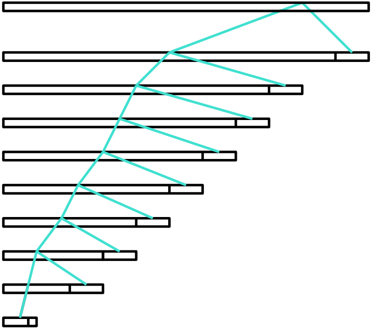


Adaptive Filler Lower Bound

Extremal strategy:
By repeated amplification using $\delta = \Theta(1/n)$ we get:

Theorem

There is an adaptive filling strategy that achieves backlog $\Omega(n)$ in running-time $O(n!)$.



OPEN QUESTIONS

- Can we extend the oblivious lower bound construction to work with arbitrary emptiers?
- Are there shorter more simple constructions?

ACKNOWLEDGEMENTS

- My mentor William Kuszmaul
- MIT PRIMES
- My Parents

Question Slides

What is the Cup Game?

Definition

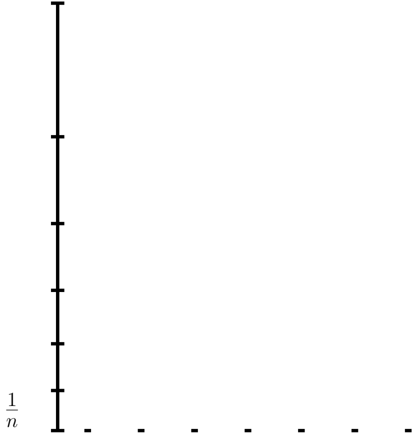
***p*-processor cup-game** on n cups:
multi-round game. every round:

- ***filler*** adds water
- ***emptier*** removes water

Note:
Emptier must allocate resources discretely
Filler can allocate resources continuously

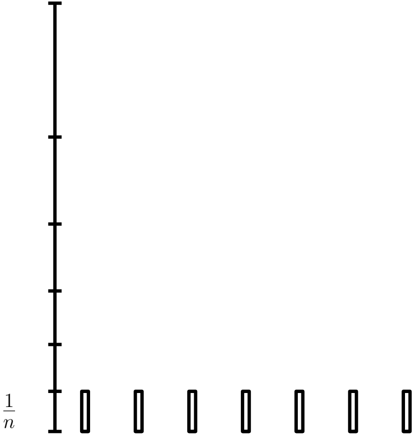
Single-Processor Lower Bound

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



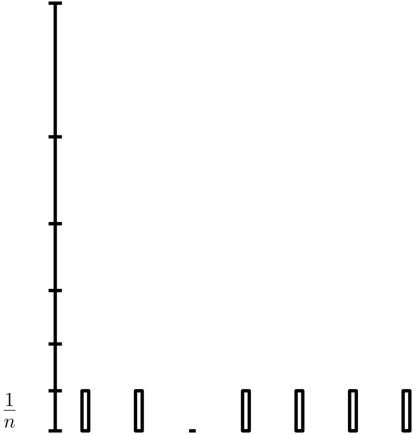
SINGLE-PROCESSOR LOWER BOUND

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



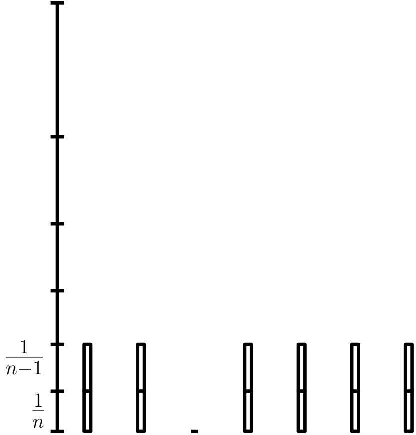
SINGLE-PROCESSOR LOWER BOUND

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



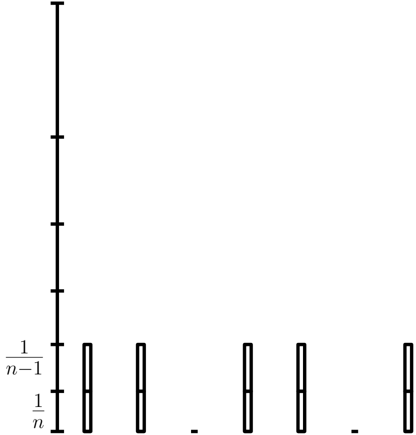
Single-Processor Lower Bound

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



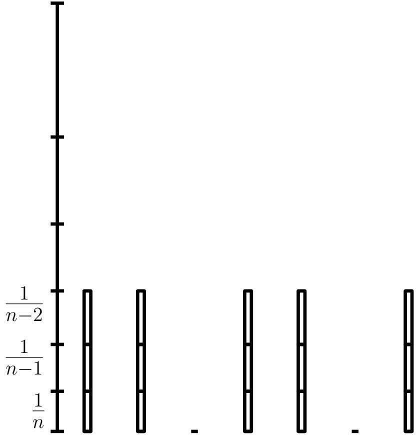
Single-Processor Lower Bound

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



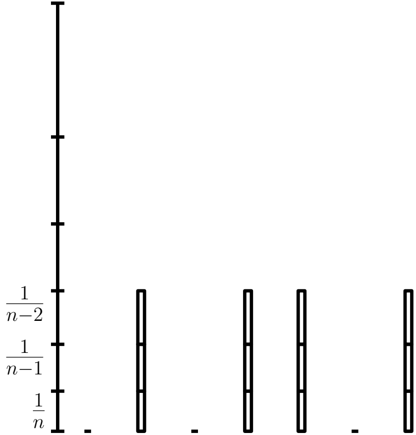
Single-Processor Lower Bound

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



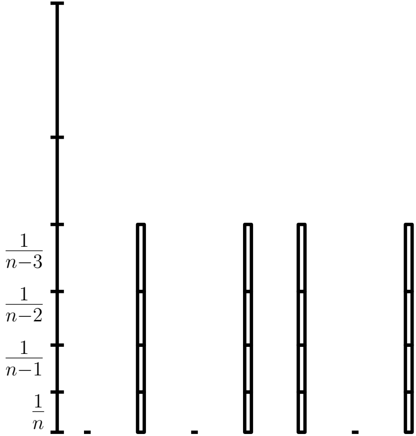
SINGLE-PROCESSOR LOWER BOUND

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



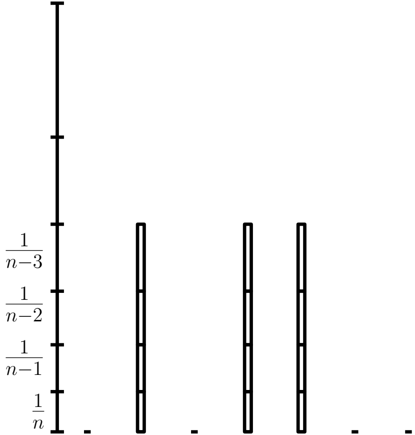
Single-Processor Lower Bound

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



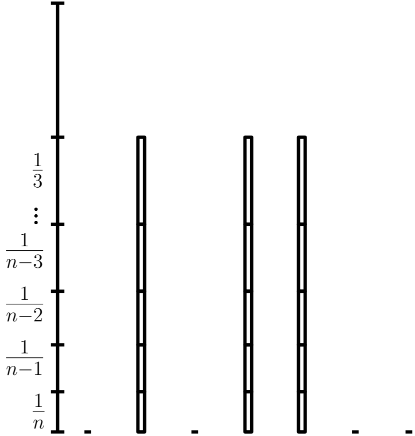
Single-Processor Lower Bound

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



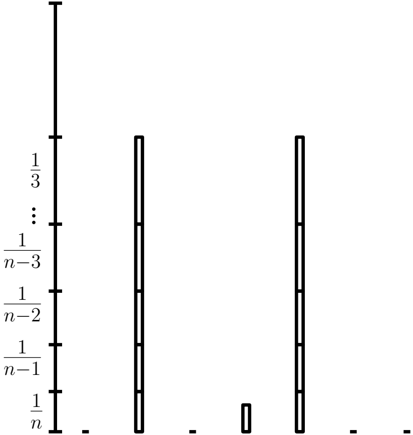
SINGLE-PROCESSOR LOWER BOUND

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



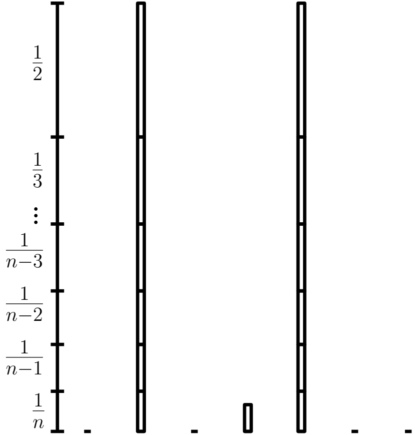
SINGLE-PROCESSOR LOWER BOUND

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



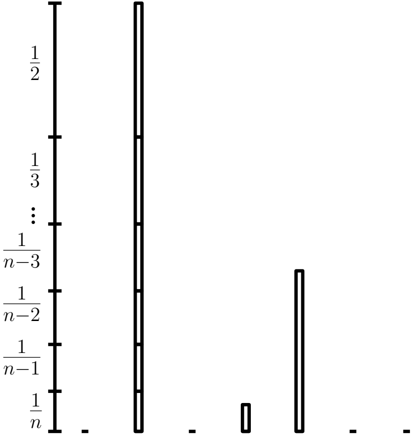
Single-Processor Lower Bound

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



Single-Processor Lower Bound

Filling strategy:
Distribute water equally amongst cups not yet emptied from.



Single-Processor Lower Bound

Filling strategy:
Distribute water equally amongst cups not yet emptied from.

Achieves backlog:

$$\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{2} = \Omega(\log n).$$

Single-Processor Upper Bound

A ***greedy emptier*** – an emptier that always empties from the fullest cup – never lets backlog exceed $O(\log n)$.

Definitions

- S_t: state at start of round t
- I_t: state after the filler adds water on round t, but before the emptier removes water
- $\mu_k(S)$: average fill of k fullest cups at state S.

Single-Processor Upper Bound Proof

Proof: Inductively prove a set of invariants:

$$\mu_k(S_t) \leq \frac{1}{k+1} + \ldots + \frac{1}{n}.$$

Let a be the cup that the emptier empties from on round t

If a is one of the k fullest cups in S_{t+1}:

$$\mu_k(S_{t+1}) \leq \mu_k(S_t).$$

Otherwise:

$$\mu_k(S_{t+1}) \leq \mu_{k+1}(I_t) \leq \mu_{k+1}(S_t) + \frac{1}{k+1}.$$

Previous work on Cup Games

- The Single-Processor cup game ($p = 1$) has been tightly analyzed with ***oblivious*** and ***adaptive*** fillers (i.e. fillers that can't and can observe the emptier's actions).
- The Multi-Processor cup game ($p > 1$) is substantially more difficult. With an adaptive filler:
  - Kuszmaul established upper bound of $O(\log n)$.[3]
  - We established a matching lower bound of $\Omega(\log n)$.
- The multi-processor cup game with an oblivious filler has not yet been tightly analyzed.
- Variants where valid moves depend on a graph have been studied.
- Variants with resource augmentation have been studied.
- Variants with semi-clairvoyance have been studied.

[3] William Kuszmaul. Achieving optimal backlog in the vanilla multi-processor cup game. SIAM, 2020.

PREVIOUS WORK — $p = 1$

Single-processor cup game
Adaptive filler:

- $\Omega(\log n)$ lower bound
- $O(\log n)$ upper bound

Oblivious filler (can't see emptier's actions): [4]

- $\Omega(\log \log n)$ lower bound
- $O(\log \log n)$ upper bound (with good probability in short games)

[4] [M. Bender, M. Farach-Colton, and W. Kuszmaul. Achieving optimal backlog in multi-processor cup games. In Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC), 2019.]

PREVIOUS WORK — RESTRICTED VERSIONS

Cup flushing game (emptier can completely empty cups):[5]

- $\Omega(\log\log n)$ lower bound
- $O(\log\log n)$ upper bound

Bamboo Garden Trimming (filler always adds same amount):[6]

- 2 lower bound
- 2 upper bound

Cups are nodes in a graph, moves restricted based on graph structure. D is the diameter of the graph.

- $\Omega(D)$ lower bound
- $O(D)$ upper bound

[5] [P. F. Dietz and R. Raman. Persistence, amortization and randomization. In Proceedings of the Second An- nual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 78–88, 1991.]

[6] [Bilò, Davide, Luciano Gualà, Stefano Leucci, Guido Proietti, and Giacomo Scornavacca. "Cutting Bamboo Down to Size." arXiv preprint arXiv:2005.00168 (2020).]

Our Variant

Definition

Variable-Processor Cup Game:
Each round filler can change p

Modification may seem small, but it drastically alters the game!

Adaptive Filler
Lower Bound

NEGATIVE FILL

In lower bound proofs we allow ***negative fill***

- Measure fill relative to average fill
- Important for recursion
- Strictly easier for the filler if cups can zero out

Amplification Lemma

Lemma

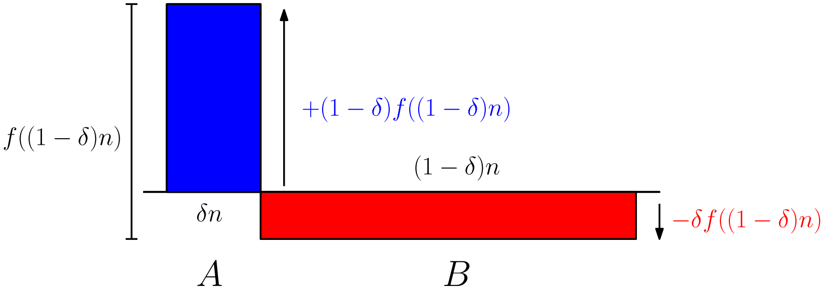
Given a strategy for achieving backlog $f(n)$ on n cups, we can construct a new strategy that achieves backlog

$$f'(n) \geq (1-\delta)\sum_{\ell=0}^{L} f((1-\delta)\delta^{\ell} n)$$

for appropriate parameters $L \in \mathbb{N}, 0 < \delta \ll 1/2$.

Amplification Lemma Proof Sketch

- A starts as the δn fullest cups, B as the $(1-\delta)n$ other cups.
- Repeatedly apply f to B and swap generated cup into A.
- Decrease p, recurse on A.

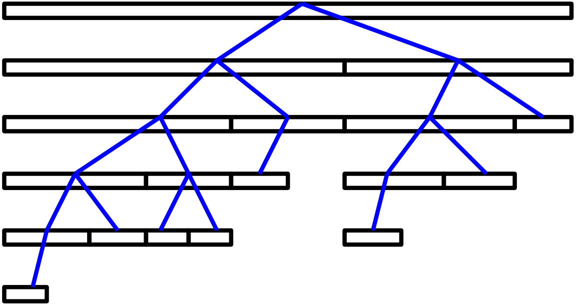


Adaptive Filler Lower Bound

Let $\epsilon > 0$ be any constant. Then there is some $\delta = \Theta(1)$ such that by repeated amplification we get:

Theorem

There is an adaptive filling strategy that achieves backlog $\Omega(n^{1-\epsilon})$ in running-time $2^{O(\log^2 n)}$.

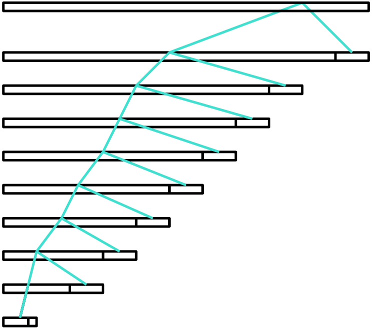


Adaptive Filler Lower Bound

Extremal strategy:
By repeated amplification using $\delta = \Theta(1/n)$ we get:

Theorem

There is an adaptive filling strategy that achieves backlog $\Omega(n)$ in running-time $O(n!)$.



Upper Bound

Upper Bound

We prove a novel set of invariants:

Theorem

A greedy emptier maintains the invariant:

$$\mu_k(S_t) \leq 2n - k.$$

Corollary

A greedy emptier never lets backlog exceed

$$O(n).$$

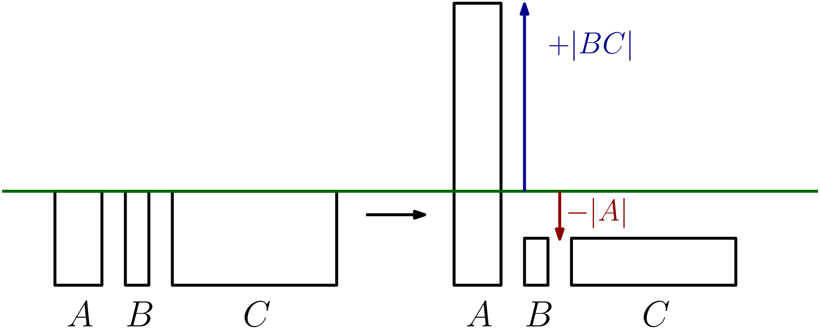
Note: this matches our lower bound!

Upper Bound Proof Sketch

Induct on t. Fix k. Define sets of cups:

- A: (emptied from) $\cap$ (k fullest in S_t) $\cap$ (k fullest in S_{t+1})
- B: (emptied from) $\cap$ (k fullest in S_t) $\cap$ (**not** k fullest in S_{t+1})
- C: AC is the k fullest cups in S_{t+1}

$\mu_k(S_{t+1})$ is largest if fill from BC is pushed into A



Oblivious Filler Lower Bound

Oblivious Filler Lower Bound

Definition

Oblivious Filler: Can't observe the emptier's actions

- Classically emptier does better in the randomized setting.
- But not in the variable-processor cup game!
- We get the same lower bound as with an adaptive filler in quasi-polynomial length games!

Oblivious Filler Lower Bound

Definition

Δ-***greedy-like*** emptier:
Let x, y be cups. If $\text{fill}(x) > \text{fill}(y) + \Delta$ then a Δ-greedy-like emptier empties from y *only if* it also empties from x.

Oblivious filler can achieve backlog $\Omega(n^{1-\epsilon})$ for $\epsilon > 0$ constant in running time $2^{\text{polylog}(n)}$ against a Δ-greedy-like emptier ($\Delta \leq O(1)$) with probability at least $1 - 2^{-\text{polylog}(n)}$.

Flattening

Definition

A cup configuration is R-flat if all cups have fills in $[-R, R]$.

Proposition

Oblivious filler can get a $2(2+\Delta)$-flat configuration from an R-flat configuration against a Δ-greedy-like emptier in running time $O(R)$.

Oblivious Filler: Constant Fill

Getting constant fill in a *known* cup is hard now. Strategy:

- Play many single-processor cup games on $\Theta(1)$ cups blindly. Each succeeds with constant probability.
- By a Chernoff Bound with probability $1 - 2^{-\Omega(n)}$ at least a constant fraction nc of these succeed.
- Set $p = nc$.
- Fill nc known cups; because emptier is greedy-like it must focus on the nc cups with high fill before these cups.
- Recurse on the nc known cups with high fill.

Oblivious Amplification Lemma

Almost identical to the Adaptive Amplification Lemma!

Lemma

Given a strategy f for achieving backlog $f(n)$ on n cups, we can construct a new strategy that achieves backlog

$$f'(n) \geq \phi \cdot (1-\delta) \sum_{\ell=0}^{L} f((1-\delta)\delta^{\ell} n)$$

for appropriate parameters $L \in \mathbb{N}, 0 < \delta \ll 1/2$ and constant $\phi \in (0,1)$ of our choice against a greedy-like emptier.

(Note: Lemma is actually more complicated than this.)

Oblivious Filler Lower Bound

Theorem

There is an oblivious filling strategy that achieves backlog

$$\Omega(n^{1-\epsilon})$$

for constant $\epsilon > 0$ with probability at least $1 - 2^{-\operatorname{polylog}(n)}$ in running time $2^{O(\log^2 n)}$ against a greedy-like emptier.

Achieve this probability by a union bound on $2^{\operatorname{polylog}(n)}$ events.

Proof notes:

- Similar to adaptive filler proof
- need larger base case for union bound to work; this doesn't harm backlog though