Scheduling Jobs with Work-*Inefficient* Parallel Solutions

William Kuszmaul[1], Alek Westover[2]

Harvard[1], MIT[2]

SPAA 2024

Problem Motivation

Engineer wants to perform tasks on a parallel machine.
Needs to choose an implementation for each task.

Problem Motivation

Engineer wants to perform tasks on a parallel machine.
Needs to choose an implementation for each task.

1. Parallel implementation:
   work inefficient
   parallelizable

Problem Motivation

Engineer wants to perform tasks on a parallel machine.
Needs to choose an implementation for each task.

1. Parallel implementation:
   work inefficient
   parallelizable
2. Serial implementation:
   work efficient, parallelism across tasks
   not parallelizable

Problem Motivation

Engineer wants to perform tasks on a parallel machine.
Needs to choose an implementation for each task.

1. Parallel implementation:
   work inefficient
   parallelizable
2. Serial implementation:
   work efficient, parallelism across tasks
   not parallelizable

Which implementations should the engineer use?

Problem Motivation

Engineer wants to perform tasks on a parallel machine.
Needs to choose an implementation for each task.

1. Parallel implementation:
   work inefficient
   parallelizable
2. Serial implementation:
   work efficient, parallelism across tasks
   not parallelizable

Which implementations should the engineer use?

Our answer:
Engineer writes a serial and parallel implementation for each task and lets the *scheduler* decide which implementations to use.

Problem Motivation

Engineer wants to perform tasks on a parallel machine.
Needs to choose an implementation for each task.

1. Parallel implementation:
   work inefficient
   parallelizable
2. Serial implementation:
   work efficient, parallelism across tasks
   not parallelizable

Which implementations should the engineer use?
Our answer:
Engineer writes a serial and parallel implementation for each task and lets the *scheduler* decide which implementations to use.

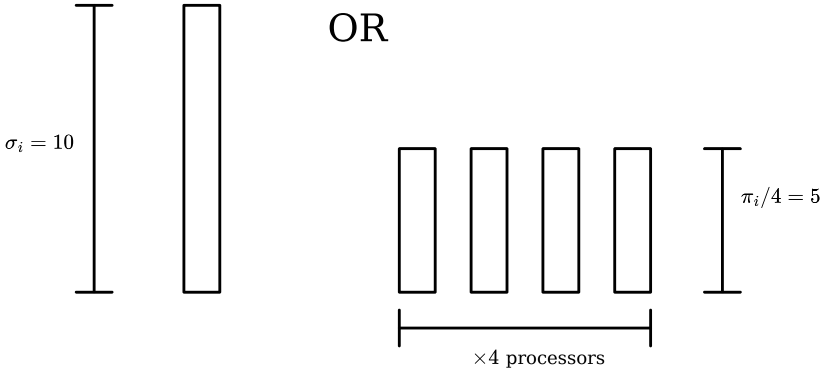
This motivates the algorithmic problem that we consider.

Defining The Serial Parallel Decision Problem

- Input: Set of n tasks (σ_i, π_i, t_i)
- σ_i = serial work, π_i = parallel work, t_i = arrival time.
- Output: serial/parallel decisions and job schedule.

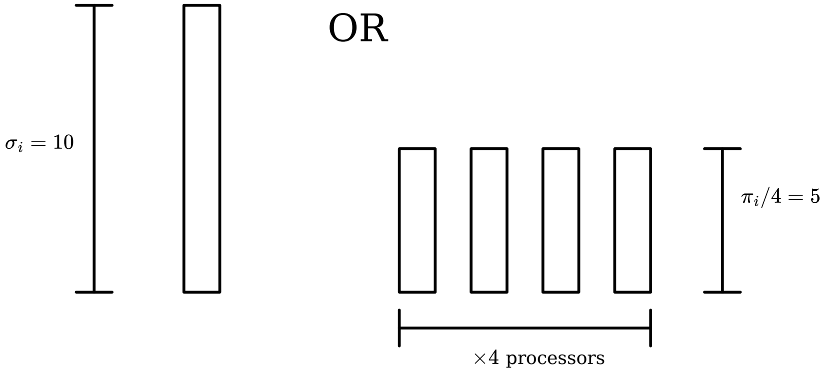
Defining The Serial Parallel Decision Problem

- Input: Set of n tasks (σ_i, π_i, t_i)
- σ_i = serial work, π_i = parallel work, t_i = arrival time.
- Output: serial/parallel decisions and job schedule.



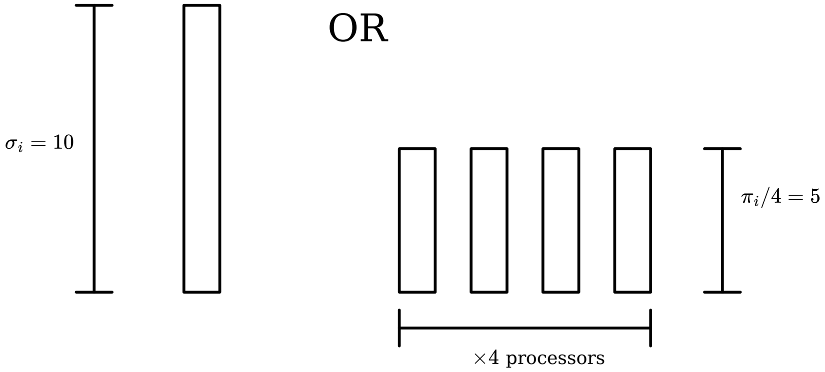
Defining The Serial Parallel Decision Problem

At each time step: allocate *p* processors to jobs.
(Serial job $\implies$ at most one processor at a time.)



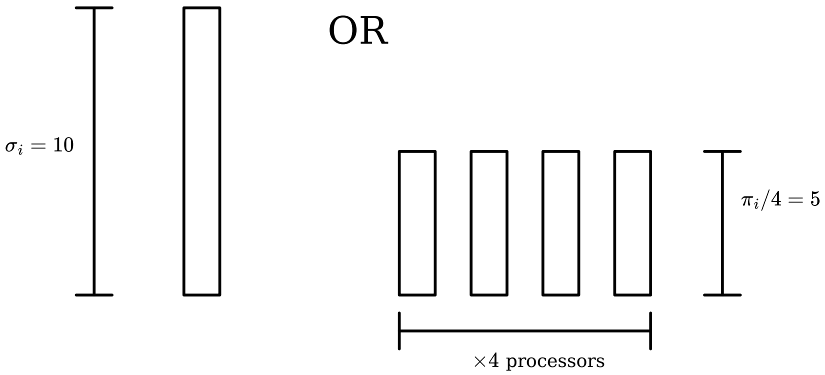
Defining The Serial Parallel Decision Problem

Completion criterion: Suppose job i has work $w \in \{\pi_i, \sigma_i\}$. Let $x_i(t)$ denote the number of processors allocated to job i at time t. Job i is completed once $\int_0^T x_i(t)dt = w$.



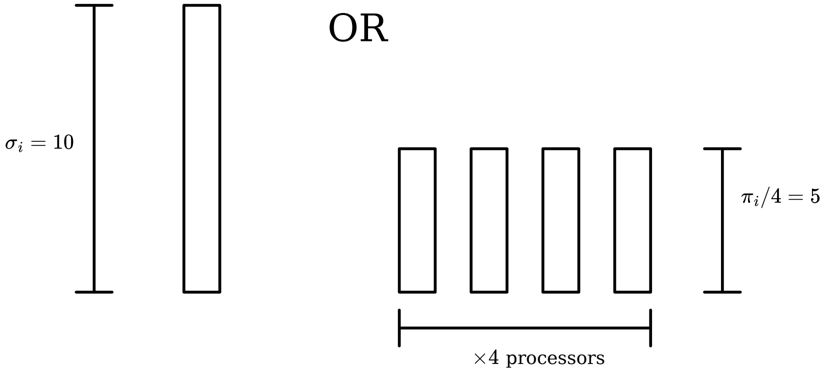
Defining The Serial Parallel Decision Problem

We require $\pi_i/p \leq \sigma_i \leq \pi_i$.



Defining The Serial Parallel Decision Problem

We've now described the model.
Next: discuss the scheduler's objectives.

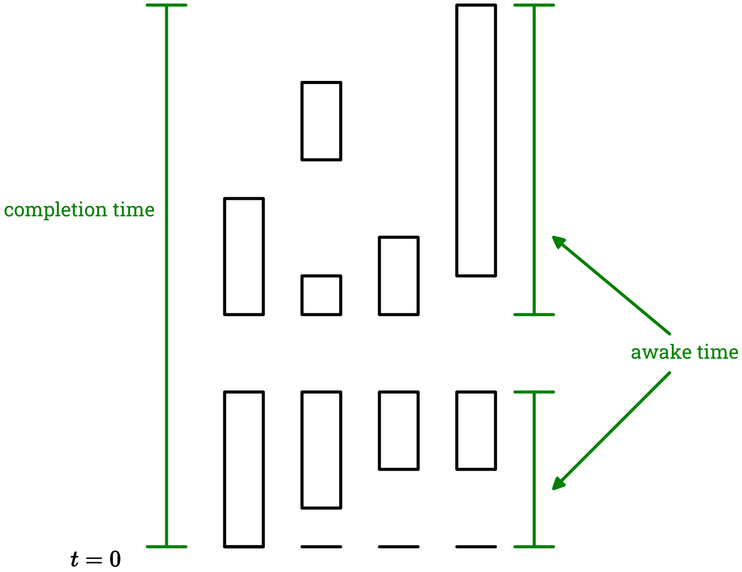


Metric 1: Awake Time

Amount of time when there are uncompleted tasks.

Metric 1: Awake Time

Amount of time when there are uncompleted tasks.

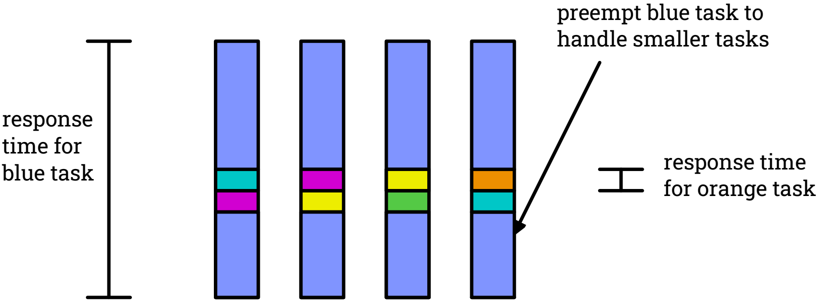


Metric 2: Mean Response Time (MRT)

Average time between receiving a task and completing it.

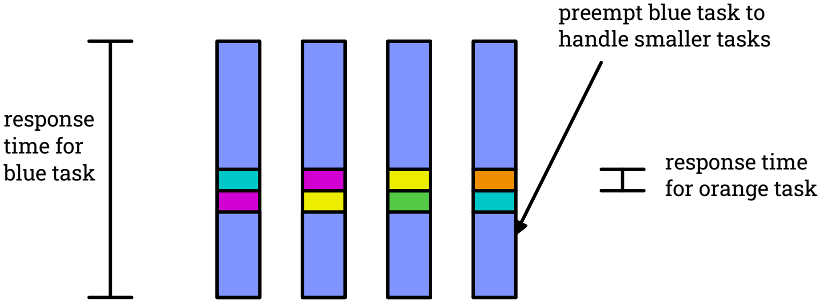
Metric 2: Mean Response Time (MRT)

Average time between receiving a task and completing it.



Metric 2: Mean Response Time (MRT)

Average time between receiving a task and completing it.



We've now described the metrics.

Next: main results.

Main Results

Theorem 1

There is an $O(1)$-competitive scheduler for awake time.

Main Results

Theorem 1

There is an $O(1)$-competitive scheduler for awake time.

Theorem 2

There is an $O(1)$-competitive scheduler for MRT, with $O(1)$-speed augmentation.

Main Results

Theorem 1

There is an $O(1)$-competitive scheduler for awake time.

Theorem 2

There is an $O(1)$-competitive scheduler for MRT, with $O(1)$-speed augmentation.

Next: Awake time specific results.

Optimizing Awake Time with Additional Restrictions

Theorem 3

*There is a 3-competitive **decide on arrival** scheduler for awake time.*

Optimizing Awake Time with Additional Restrictions

Theorem 3

*There is a 3-competitive **decide on arrival** scheduler for awake time.*

Theorem 4

*There is a 6-competitive **parallel work oblivious** scheduler for awake time.*

Optimizing Awake Time with Additional Restrictions

Theorem 3

*There is a 3-competitive **decide on arrival** scheduler for awake time.*

Theorem 4

*There is a 6-competitive **parallel work oblivious** scheduler for awake time.*

Remark 1

Any scheduler that is both decide on arrival and parallel work oblivious is not $o(\sqrt{p})$-competitive for awake time.

Optimizing Awake Time with Additional Restrictions

Theorem 3

*There is a 3-competitive **decide on arrival** scheduler for awake time.*

Theorem 4

*There is a 6-competitive **parallel work oblivious** scheduler for awake time.*

Remark 1

Any scheduler that is both decide on arrival and parallel work oblivious is not $o(\sqrt{p})$-competitive for awake time.

Remainder of Talk:
Description and analysis of parallel work oblivious scheduler.

Defining the Scheduler

Scheduler PRO (procrastinator) chooses its jobs as follows:

Defining the Scheduler

Scheduler PRO (procrastinator) chooses its jobs as follows:

- If the time since some task i arrived is larger than task i's serial work, but task i hasn't been started yet, start task i in serial.

Defining the Scheduler

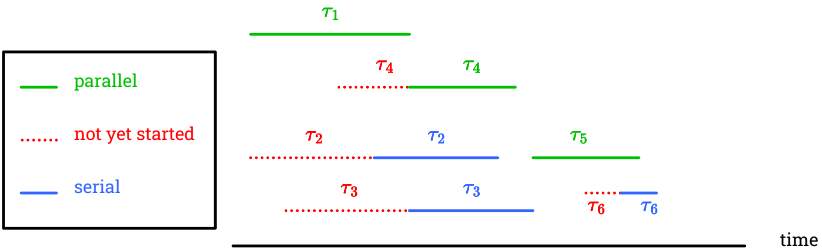
Scheduler PRO (procrastinator) chooses its jobs as follows:

- If the time since some task i arrived is larger than task i's serial work, but task i hasn't been started yet, start task i in serial.
- If there are idle processors and unstarted tasks, choose an arbitrary task to start in parallel.

Defining the Scheduler

Scheduler PRO (procrastinator) chooses its jobs as follows:

- If the time since some task *i* arrived is larger than task *i*'s serial work, but task *i* hasn't been started yet, start task *i* in serial.
- If there are idle processors and unstarted tasks, choose an arbitrary task to start in parallel.



Defining the Scheduler

At each time step, PRO allocates processors to its chosen jobs as follows:

Defining the Scheduler

At each time step, PRO allocates processors to its chosen jobs as follows:

- Allocate a processor to all serial jobs, or the p serial jobs with the most remaining work if there are more than p serial jobs.

Defining the Scheduler

At each time step, PRO allocates processors to its chosen jobs as follows:

- Allocate a processor to all serial jobs, or the p serial jobs with the most remaining work if there are more than p serial jobs.
- Allocate any remaining processors to the single running parallel job, if there is any such job.

Defining the Scheduler

At each time step, PRO allocates processors to its chosen jobs as follows:

- Allocate a processor to all serial jobs, or the p serial jobs with the most remaining work if there are more than p serial jobs.
- Allocate any remaining processors to the single running parallel job, if there is any such job.

Next: Proof outline.

Proof Outline

Theorem 5

PRO *is* 6*-competitive for awake time.*

WLOG: consider task sequences where PRO always has at least one uncompleted task present.

Proof Outline

Theorem 5

PRO *is* 6*-competitive for awake time.*

WLOG: consider task sequences where PRO always has at least one uncompleted task present.

Proof outline:

Proof Outline

Theorem 5

PRO *is* 6*-competitive for awake time.*

WLOG: consider task sequences where PRO always has at least one uncompleted task present.

Proof outline:

1. Show that at PRO has no idle processors at least half of the time.

Proof Outline

Theorem 5

PRO *is 6-competitive for awake time.*

WLOG: consider task sequences where PRO always has at least one uncompleted task present.

Proof outline:

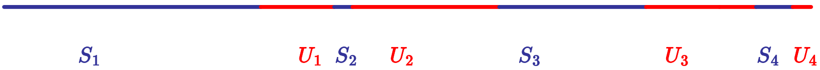
1. Show that at PRO has no idle processors at least half of the time.
2. Bound the amount of work that PRO takes.

Analysis of PRO

Saturated time step: no idle processors.

S_i saturated intervals

U_i unsaturated intervals

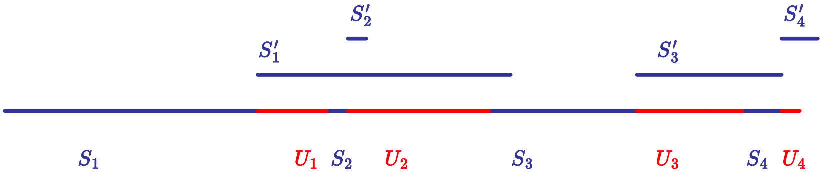


Analysis of PRO

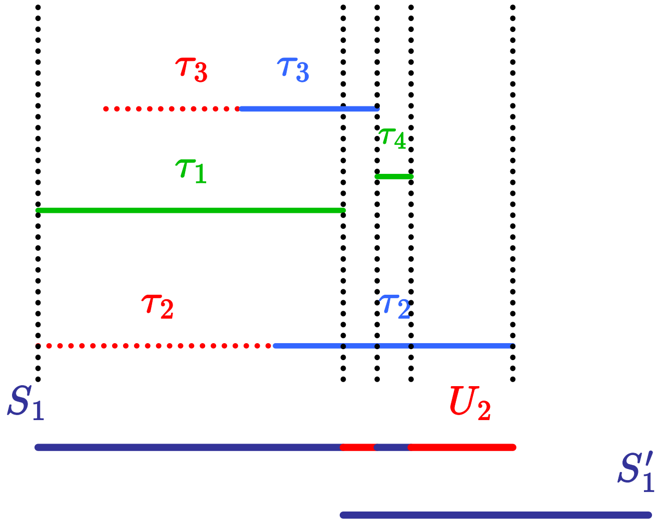
Lemma 6

PRO *is saturated at least* $1/2$ *of the time.*

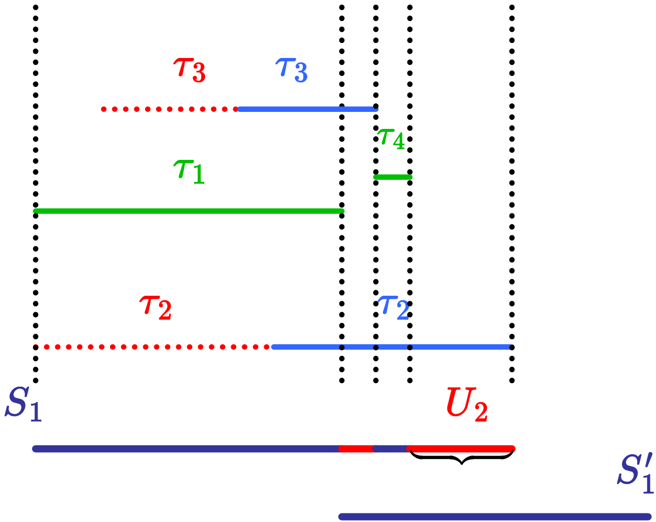
Proof: Let S_i' be a copy of S_i, shifted to start at the end of S_i. We claim that $\bigcup_j U_j \subseteq \bigcup_k S_k'$.



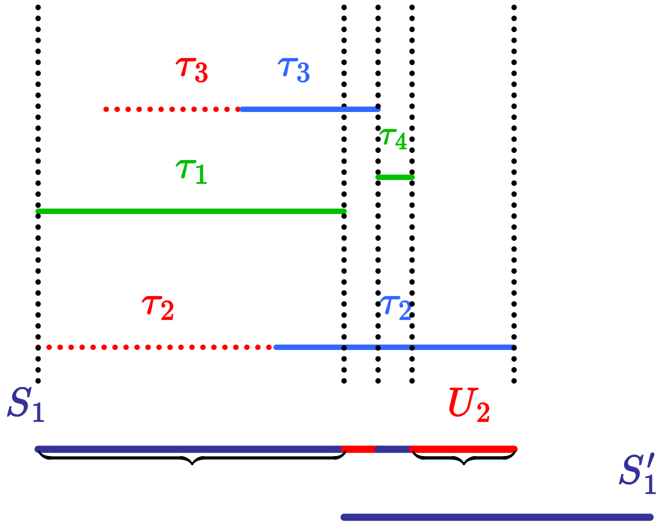
Lemma Proof Sketch



Lemma Proof Sketch



Lemma Proof Sketch



Lemma: PRO's saturated time is at most $3T_{OPT}$

T_{OPT}: optimal awake time on the tasks.

Lemma 7

The amount of time that PRO *is saturated is at most* $3T_{OPT}$.

Lemma: PRO's saturated time is at most $3T_{OPT}$

T_{OPT}: optimal awake time on the tasks.

Lemma 7

The amount of time that PRO *is saturated is at most* $3T_{OPT}$.

Proof idea:

Bound work on each of four (non-exclusive) categories of tasks τ.
Proof omitted due to time.

PRO Analysis: Combining the Lemmas

Theorem 8

PRO *is a 6-competitive parallel work oblivious scheduler for awake time.*

PRO Analysis: Combining the Lemmas

Theorem 8

PRO *is a 6-competitive parallel work oblivious scheduler for awake time.*

Proof: PRO is saturated for at least $1/2$ of its time steps, and has at most $3T_{OPT}$ saturated time steps. □

Open Questions

Awake Time

model	lower bound	best algorithm
vanilla	$1.618 - O(1/p)$	2
decide on arrival	$2 - O(1/p)$	3
parallel work oblivious	$2 - O(1/p)$	6
randomized	$1.18 - O(1/p)$	2

Mean Response Time

model	lower bound	best algorithm
$O(1)$ speed augmentation	??	$O(1)$
decide on arrival	??	
parallel work oblivious with $O(1)$ speed augmentation	$\Omega(p^{1/4})$	
non-preemptive	∞	
no speed augmentation	??	

Extra slides

Decide on Arrival Scheduler

Decide on Arrival Scheduler Definition

Fix TAP $\tau_1, \tau_2, \ldots, \tau_n$.

Definition 9

C^i_{ALG}: completion time of scheduler ALG on tasks $\tau_1, \tau_2, \ldots, \tau_i$.

Scheduler BAL: When task τ_i arrives,

- If $\sigma_i + t_i \geq C^i_{\mathsf{BAL}}$ run τ_i in serial.
- Else, run τ_i in parallel.

Depiction of BAL

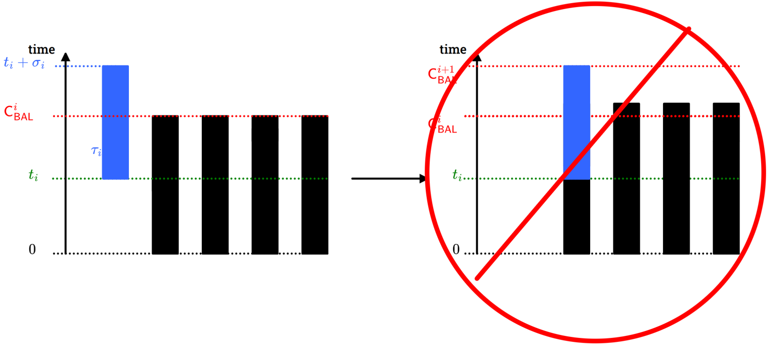


Figure: Serial job is too large: BAL chooses parallel job

Depiction of BAL

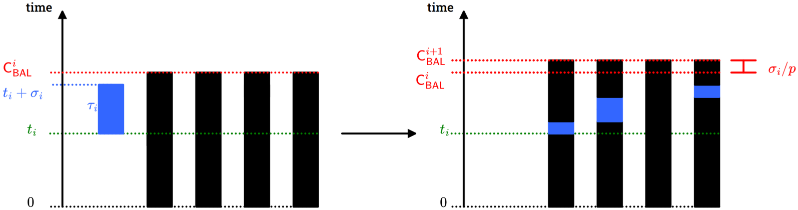


Figure: BAL chooses a serial job

Observe BAL is always "***balanced***": never has idle processors.

Key Invariant

Let OPT denote the optimal schedule of $\tau_1, \tau_2, \ldots, \tau_n$.
Important: OPT is not optimal on the first i tasks, is optimal overall.
Let K^i_{OPT} denote the work of OPT on the first i tasks.

Key Invariant

Let OPT denote the optimal schedule of $\tau_1, \tau_2, \ldots, \tau_n$.
Important: OPT is not optimal on the first i tasks, is optimal overall.
Let K^i_{OPT} denote the work of OPT on the first i tasks.

Lemma 10

$$\mathsf{C}^i_{\text{BAL}} \leq 2\mathsf{C}^i_{\text{OPT}} + \mathsf{K}^i_{\text{OPT}}/p.$$

Key Invariant

Let OPT denote the optimal schedule of $\tau_1, \tau_2, \ldots, \tau_n$.
Important: OPT is not optimal on the first i tasks, is optimal overall.
Let K^i_{OPT} denote the work of OPT on the first i tasks.

Lemma 10

$$\mathrm{C}^i_{\mathrm{BAL}} \leq 2\mathrm{C}^i_{\mathrm{OPT}} + \mathrm{K}^i_{\mathrm{OPT}}/p.$$

Immediate corollary: BAL is 3-competitive for completion time.
(Later: extend to awake time.)

Proof of Key Invariant

Assume

$$\mathsf{C}_{\mathsf{BAL}}^{i-1} \leq 2\mathsf{C}_{\mathsf{OPT}}^{i-1} + \mathsf{K}_{\mathsf{OPT}}^{i-1}/p.$$

Case 1: BAL **runs** τ_i **in serial.**

$$\begin{aligned} \mathsf{C}_{\mathsf{BAL}}^{i} &= \mathsf{C}_{\mathsf{BAL}}^{i-1} + \sigma_i/p \\ &\leq 2\mathsf{C}_{\mathsf{OPT}}^{i-1} + (\mathsf{K}_{\mathsf{OPT}}^{i-1} + \sigma_i)/p \\ &\leq 2\mathsf{C}_{\mathsf{OPT}}^{i} + \mathsf{K}_{\mathsf{OPT}}^{i}/p. \end{aligned}$$

Proof of Key Invariant

Assume

$$C_{\mathsf{BAL}}^{i-1} \le 2C_{\mathsf{OPT}}^{i-1} + K_{\mathsf{OPT}}^{i-1}/p.$$

Case 2: BAL **and** OPT **both run** τ_i **in parallel.**

$$\begin{aligned} C_{\mathsf{BAL}}^{i} &= C_{\mathsf{BAL}}^{i-1} + \pi_i/p \\ &\le 2C_{\mathsf{OPT}}^{i-1} + (K_{\mathsf{OPT}}^{i-1} + \pi_i)/p \\ &\le 2C_{\mathsf{OPT}}^{i} + K_{\mathsf{OPT}}^{i}/p. \end{aligned}$$

Proof of Key Invariant

Assume

$$C_{BAL}^{i-1} \le 2C_{OPT}^{i-1} + K_{OPT}^{i-1}/p.$$

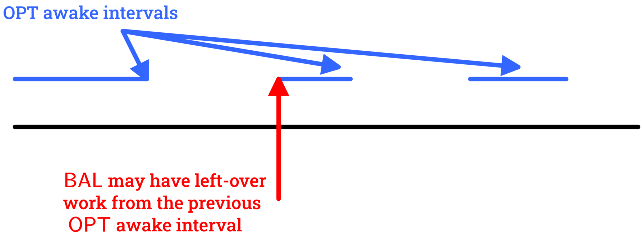
Case 3: BAL **runs** τ_i **in parallel,** OPT **runs** τ_i **in serial.**
τ_i was too large for BAL to run in serial, but OPT ran τ_i in serial:

$$C_{OPT}^{i} \ge \sigma_i + t_i \ge C_{BAL}^{i-1}.$$

Thus,

$$\begin{aligned} C_{BAL}^{i} &= C_{BAL}^{i-1} + \pi_i/p \\ &\le C_{OPT}^{i} + \sigma_i \\ &\le 2C_{OPT}^{i}. \end{aligned}$$

Extending To Awake Time



Solution: if BAL starts an awake interval with more work BAL wont get further behind on this extra work.

Extending to Awake Time

Lemma 11

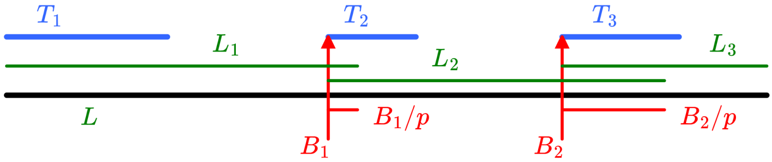
If BAL *starts (balanced) with* B *extra work and then handles the same TAP as* OPT *then*

$$\mathrm{C}_{\mathrm{BAL}} \leq 3\mathrm{C}_{\mathrm{OPT}} + B/p.$$

Extending to Awake Time

Theorem 12

BAL *is a 3-competitive decide on arrival scheduler for awake time.*



T_1, T_2, T_3: OPT completion times
L_1, L_2, L_3: BAL completion times
L: BAL total completion time
B_1, B_2: extra work

$L_1 \leq 3T_1 + 0$
$L_2 \leq 3T_2 + B_1/p$
$L_3 \leq 3T_3 + B_2/p$

$$L = L_1 - B_1/p + L_2 - B_2/p + L_3 \leq 3(T_1 + T_2 + T_3)$$

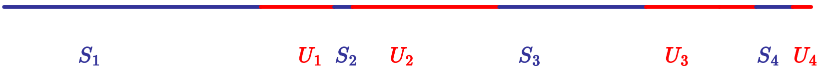
Parallel Work Oblivious Scheduler – Analysis

Analysis of PRO

Saturated time step: no idle processors.

S_i saturated intervals

U_i unsaturated intervals

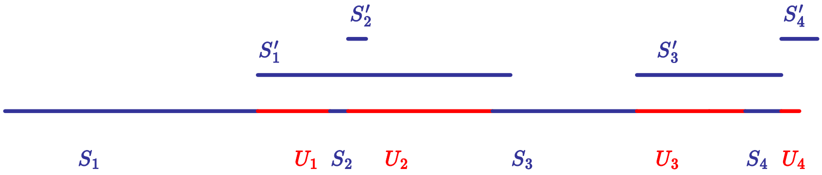


Analysis of PRO

Lemma 13

PRO *is saturated at least* $1/2$ *of the time.*

Proof: Let S_i' be a copy of S_i, shifted to start at the end of S_i. We claim that $\bigcup_j U_j \subseteq \bigcup_k S_k'$.

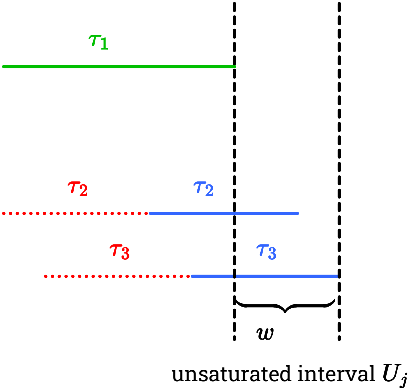


Lemma: PRO is saturated at least 1/2 the time

Claim 1

Let w be maximum over tasks i present at the start of U_j of the serial work remaining on task i. Then, $|U_j| \leq w$.

Proof:

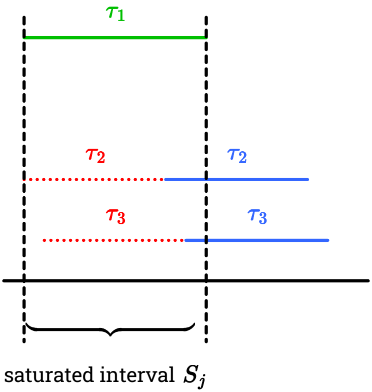


Lemma: PRO is saturated at least 1/2 the time

Claim 2 (2)

Suppose task i is started in serial during saturated interval S_j. Then, $|S_j| \geq \sigma_i$.

Proof:



Lemma: PRO is saturated at least $1/2$ the time

Claim 3 (3)

Suppose that task i is started in serial at time t and runs during an unsaturated interval $U_j = [a, b]$. Then task i is allocated a processor at each step in $[t, a]$.

Proof: If serial task i gets work stolen from it at some time t, then PRO must have p serial tasks with at least as much remaining work as task i at time t. Then, PRO will remain saturated (at least) until task i is finished.

Lemma: PRO is saturated at least 1/2 the time

Corollary 14

For each unsaturated interval U_j, there is a saturated interval S_k such that $U_j \subseteq S'_k$.

Lemma: PRO is saturated at least 1/2 the time

Corollary 14

For each unsaturated interval U_j, there is a saturated interval S_k such that $U_j \subseteq S'_k$.

Proof:
Task $i =$ serial job with largest remaining work at beginning of U_j.
$S_k =$ the saturated interval when task i was started.
Let $U_j = [a, b]$, let $t \in S_k$ be the time when task i is started.

Lemma: PRO is saturated at least $1/2$ the time

Corollary 14

For each unsaturated interval U_j, there is a saturated interval S_k such that $U_j \subseteq S'_k$.

Proof:
Task $i =$ serial job with largest remaining work at beginning of U_j.
$S_k =$ the saturated interval when task i was started.
Let $U_j = [a, b]$, let $t \in S_k$ be the time when task i is started.

- Claim 3 $\implies$ task i runs on every time step in $[t, b]$.

Lemma: PRO is saturated at least 1/2 the time

Corollary 14

For each unsaturated interval U_j, there is a saturated interval S_k such that $U_j \subseteq S'_k$.

Proof:
Task i = serial job with largest remaining work at beginning of U_j.
S_k = the saturated interval when task i was started.
Let $U_j = [a, b]$, let $t \in S_k$ be the time when task i is started.

- Claim 3 $\implies$ task i runs on every time step in $[t, b]$.
- So task i has at most $\sigma_i - (a - t)$ work left at the start of U_j.

Lemma: PRO is saturated at least 1/2 the time

Corollary 14

For each unsaturated interval U_j, there is a saturated interval S_k such that $U_j \subseteq S'_k$.

Proof:

Task $i =$ serial job with largest remaining work at beginning of U_j.
$S_k =$ the saturated interval when task i was started.
Let $U_j = [a, b]$, let $t \in S_k$ be the time when task i is started.

- Claim 3 $\implies$ task i runs on every time step in $[t, b]$.
- So task i has at most $\sigma_i - (a - t)$ work left at the start of U_j.
- Then, Claim 1 $\implies$ $|U_j| \leq \sigma_i - (a - t)$.

Lemma: PRO is saturated at least 1/2 the time

Corollary 14

For each unsaturated interval U_j, there is a saturated interval S_k such that $U_j \subseteq S'_k$.

Proof:
Task i = serial job with largest remaining work at beginning of U_j.
S_k = the saturated interval when task i was started.
Let $U_j = [a, b]$, let $t \in S_k$ be the time when task i is started.

- Claim 3 $\implies$ task i runs on every time step in $[t, b]$.
- So task i has at most $\sigma_i - (a - t)$ work left at the start of U_j.
- Then, Claim 1 $\implies$ $|U_j| \leq \sigma_i - (a - t)$.
- So $U_j \subseteq [a, a + \sigma_i - (a - t)] = [a, t + \sigma_i] \subseteq [t, t + \sigma_i]$.

Lemma: PRO is saturated at least 1/2 the time

Corollary 14

For each unsaturated interval U_j, there is a saturated interval S_k such that $U_j \subseteq S'_k$.

Proof:
Task i = serial job with largest remaining work at beginning of U_j.
S_k = the saturated interval when task i was started.
Let $U_j = [a, b]$, let $t \in S_k$ be the time when task i is started.

- Claim 3 $\implies$ task i runs on every time step in $[t, b]$.
- So task i has at most $\sigma_i - (a - t)$ work left at the start of U_j.
- Then, Claim 1 $\implies$ $|U_j| \leq \sigma_i - (a - t)$.
- So $U_j \subseteq [a, a + \sigma_i - (a - t)] = [a, t + \sigma_i] \subseteq [t, t + \sigma_i]$.
- Claim 2 $\implies$ $|S_k| \geq \sigma_i$

Lemma: PRO is saturated at least 1/2 the time

Corollary 14

For each unsaturated interval U_j, there is a saturated interval S_k such that $U_j \subseteq S'_k$.

Proof:
Task $i =$ serial job with largest remaining work at beginning of U_j.
$S_k =$ the saturated interval when task i was started.
Let $U_j = [a, b]$, let $t \in S_k$ be the time when task i is started.

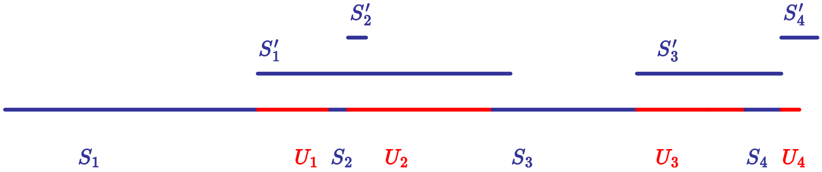
- Claim 3 $\implies$ task i runs on every time step in $[t, b]$.
- So task i has at most $\sigma_i - (a - t)$ work left at the start of U_j.
- Then, Claim 1 $\implies |U_j| \leq \sigma_i - (a - t)$.
- So $U_j \subseteq [a, a + \sigma_i - (a - t)] = [a, t + \sigma_i] \subseteq [t, t + \sigma_i]$.
- Claim 2 $\implies |S_k| \geq \sigma_i$
- So $U_j \subseteq [t, t + |S_k|]$. □

Lemma: PRO is saturated at least 1/2 the time

We have shown $\bigcup_j U_j \subseteq \bigcup_k S'_k$, which gives:

Lemma 15

PRO *is saturated at least* 1/2 *of the time.*

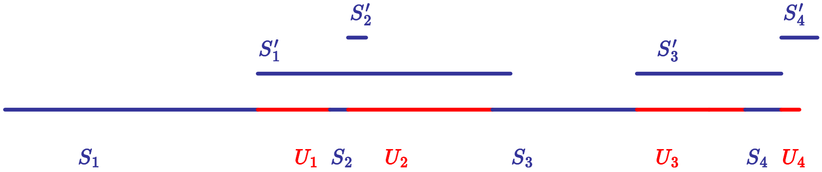


Lemma: PRO is saturated at least 1/2 the time

We have shown $\bigcup_j U_j \subseteq \bigcup_k S'_k$, which gives:

Lemma 15

PRO *is saturated at least* 1/2 *of the time.*



Next: bound saturated time by analyzing PRO's work.

Lemma: PRO's saturated time is at most $3T_{OPT}$

T_{OPT}: optimal awake time on the tasks.

Lemma 16

The amount of time that PRO *is saturated is at most* $3T_{OPT}$.

Lemma: PRO's saturated time is at most $3T_{OPT}$

T_{OPT}: optimal awake time on the tasks.

Lemma 16

The amount of time that PRO *is saturated is at most* $3T_{OPT}$.

Proof idea:

Bound work on each of four (non-exclusive) categories of tasks τ:

Lemma: PRO's saturated time is at most $3T_{OPT}$

T_{OPT}: optimal awake time on the tasks.

Lemma 16

The amount of time that PRO *is saturated is at most* $3T_{OPT}$.

Proof idea:

Bound work on each of four (non-exclusive) categories of tasks τ:

1. PRO runs τ is serial.

Lemma: PRO's saturated time is at most $3T_{OPT}$

T_{OPT}: optimal awake time on the tasks.

Lemma 16

The amount of time that PRO *is saturated is at most* $3T_{OPT}$.

Proof idea:

Bound work on each of four (non-exclusive) categories of tasks τ:

1. PRO runs τ is serial.
2. PRO runs τ in parallel starting after OPT finishes τ.

Lemma: PRO's saturated time is at most $3T_{OPT}$

T_{OPT}: optimal awake time on the tasks.

Lemma 16

The amount of time that PRO *is saturated is at most* $3T_{OPT}$.

Proof idea:

Bound work on each of four (non-exclusive) categories of tasks τ:

1. PRO runs τ is serial.
2. PRO runs τ in parallel starting after OPT finishes τ.
3. PRO runs τ in parallel completely during times when OPT has uncompleted tasks.

Lemma: PRO's saturated time is at most $3T_{OPT}$

T_{OPT}: optimal awake time on the tasks.

Lemma 16

The amount of time that PRO *is saturated is at most* $3T_{OPT}$.

Proof idea:

Bound work on each of four (non-exclusive) categories of tasks τ:

1. PRO runs τ is serial.
2. PRO runs τ in parallel starting after OPT finishes τ.
3. PRO runs τ in parallel completely during times when OPT has uncompleted tasks.
4. PRO runs τ in parallel starting before OPT finishes τ, but PRO's execution of τ overlaps with a time when OPT has no uncompleted tasks.

PRO Analysis — Type 1 and 2 Tasks

Type 1: PRO runs τ is serial.
Type 2: PRO runs τ in parallel starting after OPT finishes τ.

Claim 4 (1,2)

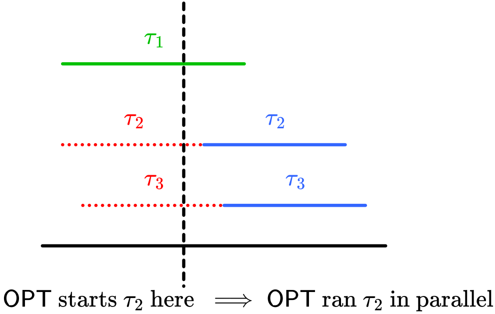
PRO spends at most $p\mathsf{T}_{\mathsf{OPT}}$ work on tasks of types (1) and (2).

PRO Analysis — Type 1 and 2 Tasks

Type 1: PRO runs τ is serial.

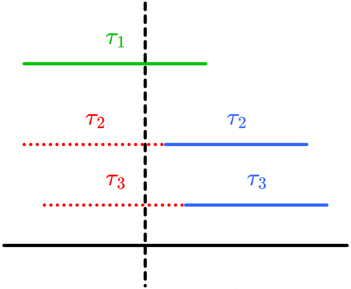
Type 2: PRO runs τ in parallel starting after OPT finishes τ.

Proof: If τ_i is a type (2) task then OPT finishes τ_i faster than σ_i, or else PRO would have started τ_i in serial. Thus, OPT must run type (2) tasks in parallel.



PRO Analysis — Type 1 and 2 Tasks

Type 1: PRO runs τ is serial.
Type 2: PRO runs τ in parallel starting after OPT finishes τ.
Proof: If τ_i is a type (2) task then OPT finishes τ_i faster than σ_i, or else PRO would have started τ_i in serial. Thus, OPT must run type (2) tasks in parallel.



OPT starts τ_2 here $\implies$ OPT ran τ_2 in parallel

Thus, the total work performed by OPT is at least the sum of π_i for type (2) tasks and σ_i for type (1) tasks.

PRO Analysis — Type 3 Tasks

Type 3: PRO runs τ in parallel completely during times when OPT has uncompleted tasks.

Claim 4 (3)

PRO spends at most $p\mathrm{T}_{\mathrm{OPT}}$ work on tasks of types (3).

Proof: Clear.

PRO Analysis — Type 4 Tasks

Type 4: PRO runs τ in parallel starting before OPT finishes τ, but PRO's execution of τ overlaps with a time when OPT has no uncompleted tasks.

Claim 5 (4)

PRO spends at most $p\mathsf{T}_{\mathsf{OPT}}$ work on tasks of types (4).

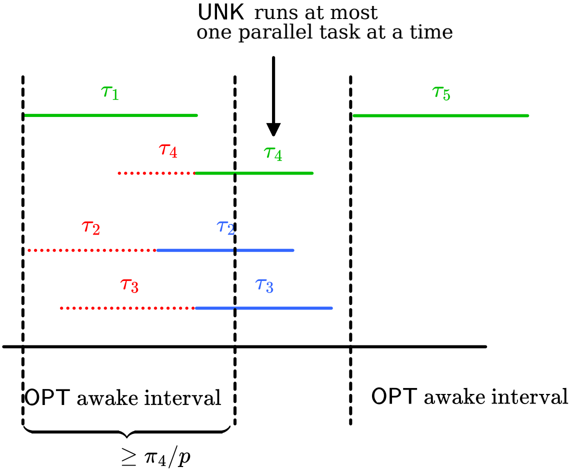
PRO Analysis — Type 4 Tasks

Type 4: PRO runs τ in parallel starting before OPT finishes τ, but PRO's execution of τ overlaps with a time when OPT has no uncompleted tasks.

Proof: For each OPT awake interval I there is at most one type (4) task that starts during I in parallel and runs past the end of I. The length of I is at least π_i/p for this type (4) task.

PRO Analysis — Type 4 Tasks

Type 4: PRO runs τ in parallel starting before OPT finishes τ, but PRO's execution of τ overlaps with a time when OPT has no uncompleted tasks.



PRO Analysis: Combining the Lemmas

Theorem 17

PRO *is a 6-competitive parallel work oblivious scheduler for awake time.*

PRO Analysis: Combining the Lemmas

Theorem 17

PRO *is a 6-competitive parallel work oblivious scheduler for awake time.*

Proof: PRO is saturated for at least $1/2$ of its time steps, and has at most $3T_{OPT}$ saturated time steps. □