A Verified Parallel Scheduler for OCaml 5

Clément Allain

Gabriel Scherer

Task Parallelism

Parabs: A Verified Realistic Parallel Scheduler

Chase-Lev Work-Stealing Deque

Future work

Parallelism in OCaml 5

- Domains
  - `Domain.spawn : (unit -> 'a) -> 'a Domain.t`
  - `Domain.join : 'a Domain.t -> 'a`
- Atomic primitives
  - Atomic references (`'a Atomic.t`)
  - Atomic record fields (`'a Atomic.Loc.t`)

Fibonacci Using Domains

```
let rec fibonacci n =
  if n <= 1 then
    1
  else
    let dom1 = Domain.spawn (fun () -> fibonacci (n - 1)) in
    let dom2 = Domain.spawn (fun () -> fibonacci (n - 2)) in
    Domain.join dom1 + Domain.join dom2
```

Fibonacci Using Domains

```
let rec fibonacci n =
  if n
    1
  else
    let
    let
    Dom
```

This implementation is *wrong*:

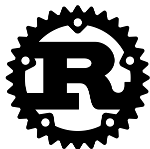
- the number of active domains is limited;
- performance collapses when spawning more domains than available cores.

Fibonacci Using a Parallel Scheduler

```
let rec fibonacci ctx n =
  if n <= 1 then
    1
  else
    let fut1 = Future.async ctx (fun ctx -> fibonacci ctx (n - 1)) in
    let fut2 = Future.async ctx (fun ctx -> fibonacci ctx (n - 2)) in
    Future.wait ctx fut1 + Future.wait ctx fut2

let fibonacci num_domain n =
  let pool = Pool.create num_domain in
  let res = Pool.run_on pool (fun ctx -> fibonacci ctx n) in
  Pool.close pool ;
  res
```

Existing Parallel Schedulers



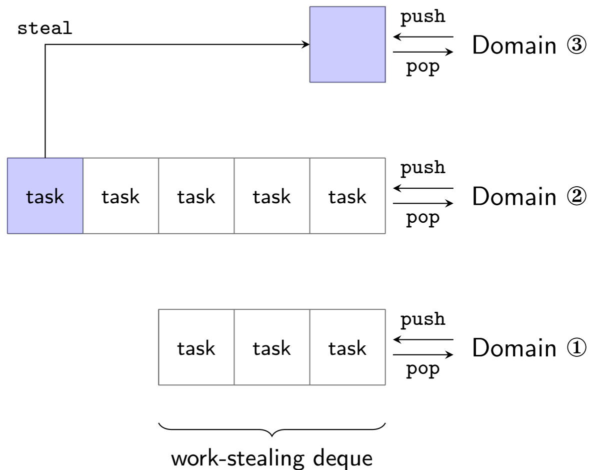
`Domainslib`
`Moonpool`

`Rayon`
`Tokio`

Cilk
TBB
`Taskflow`

Goroutines

Work-Stealing



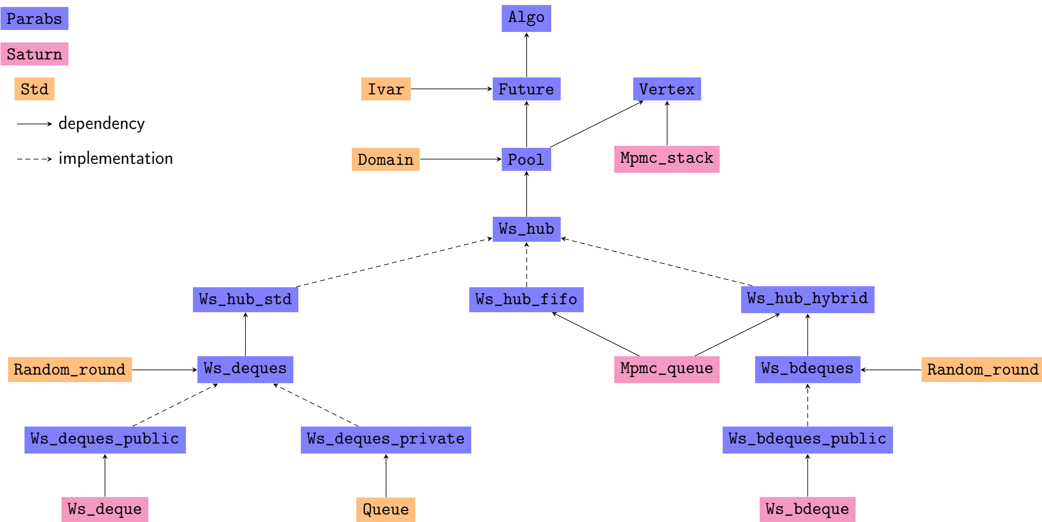
Task Parallelism

`Parabs`: A Verified Realistic Parallel Scheduler

Chase-Lev Work-Stealing Deque

Future work

Overview



Overview

Parabs
Saturn
Std
dependency
implementation

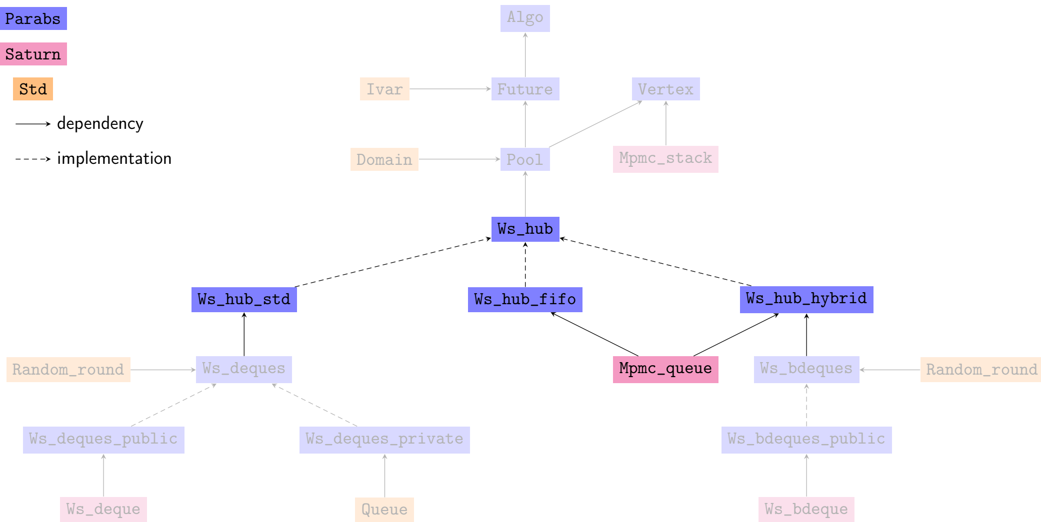
Algo
Ivar
Future
Vertex
Domain
Pool
Mpmc_stack
Ws_hub
Ws_hub_std
Ws_hub_fifo
Ws_hub_hybrid
Random_round
Ws_deques
Mpmc_queue
Ws_bdeques
Random_round
Ws_deques_public
Ws_deques_private
Ws_bdeques_public
Ws_deque
Queue
Ws_bdeque

Overview

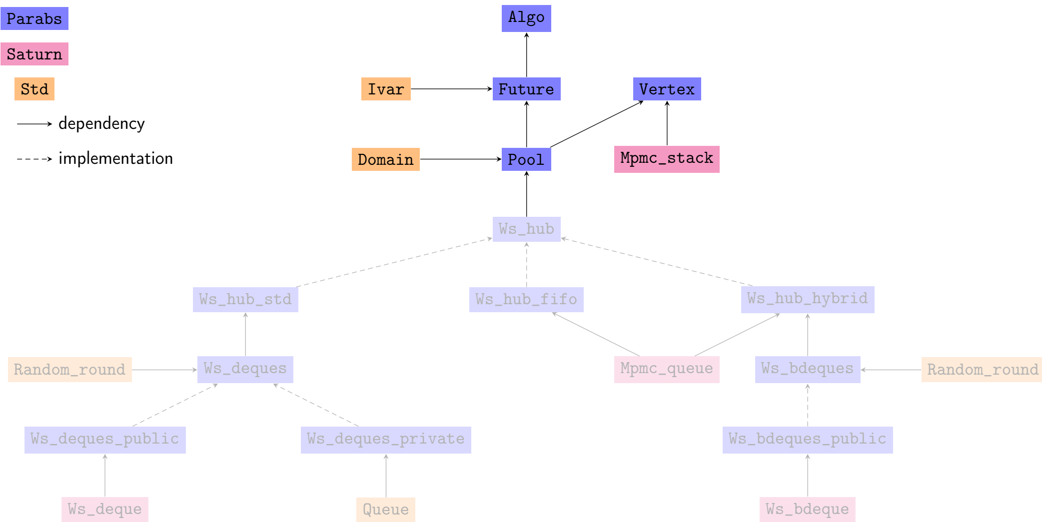
Parabs
Saturn
Std
⟶ dependency
- - -→ implementation

Algo
Ivar
Future
Vertex
Domain
Pool
Mpmc_stack
Ws_hub
Ws_hub_std
Ws_hub_fifo
Ws_hub_hybrid
Random_round
Ws_deques
Mpmc_queue
Ws_bdeques
Random_round
Ws_deques_public
Ws_deques_private
Ws_bdeques_public
Ws_deque
Queue
Ws_bdeque

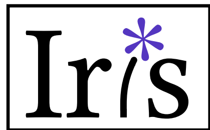
Overview



Overview



Verification



`Parabs` **has been fully verified using the Zoo framework**
(assuming sequential consistency).

Futures

ASYNC-SPEC

$$\frac{\left\{\begin{array}{l}\text{pool.context } \mathit{pool}\ \mathit{ctx}\ \mathit{scope} \ast \\ \quad \forall\ \mathit{ctx}\ \mathit{scope}. \\ \quad \text{pool.context } \mathit{pool}\ \mathit{ctx}\ \mathit{scope} \mathrel{-\!\!\ast} \\ \quad \text{wp } \mathit{task}\ \mathit{ctx} \left\{\begin{array}{l} v.\ \text{pool.context } \mathit{pool}\ \mathit{ctx}\ \mathit{scope} \ast \\ \quad \triangleright \Psi\ v \ast \\ \quad \triangleright \Box\, \Xi\ v \end{array}\right\} \end{array}\right\}}{\texttt{async}\ \mathit{ctx}\ \mathit{task}}\ \left\{\begin{array}{l} t.\ \text{pool.context } \mathit{pool}\ \mathit{ctx}\ \mathit{scope} \ast \\ \quad \text{inv } \mathit{pool}\ t\ \Psi\ \Xi \ast \\ \quad \text{consumer } t\ \Psi \end{array}\right\}$$

CONSUMER-DIVIDE

$$\frac{\begin{array}{c}\text{inv } \mathit{pool}\ t\ \Psi\ \Xi \\ \text{consumer } t \left(\lambda\, v.\ \mathop{\Large\ast}_{X \in Xs} X\ v\right)\end{array}}{\Rrightarrow \mathop{\Large\ast}_{X \in Xs} \text{consumer } t\ X}$$

Futures

WAIT-SPEC

$$\frac{\left\{\begin{array}{l}\textsf{pool.context}\ \mathit{pool}\ \mathit{ctx}\ \mathit{scope} * \\ \textsf{inv}\ \mathit{pool}\ t\ \Psi\ \Xi\end{array}\right\}}{\dfrac{\texttt{wait}\ \mathit{ctx}\ t}{\left\{\begin{array}{l}v.\ £\,2 * \\ \quad \textsf{pool.context}\ \mathit{pool}\ \mathit{ctx}\ \mathit{scope} * \\ \quad \textsf{result}\ t\ v\end{array}\right\}}}$$

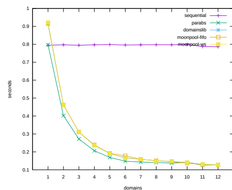
INV-RESULT-CONSUMER

$$\frac{\begin{array}{c}\textsf{inv}\ \mathit{pool}\ t\ \Psi\ \Xi \\ \textsf{result}\ t\ v \\ \textsf{consumer}\ t\ \mathrm{X}\end{array}}{\Rrightarrow \triangleright^2\, \mathrm{X}\ v}$$

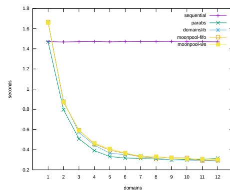
INV-RESULT

$$\frac{\begin{array}{c}\textsf{inv}\ \mathit{pool}\ t\ \Psi\ \Xi \\ \textsf{result}\ t\ v\end{array}}{\Rrightarrow \triangleright \Box\, \Xi\ v}$$

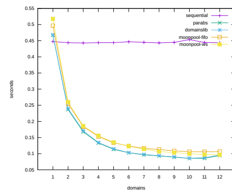
Benchmarks



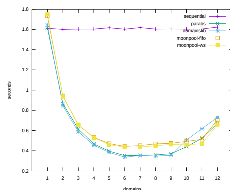
(a) `fibonacci`



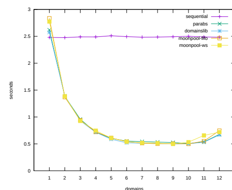
(b) `for_irregular`



(c) `iota`



(d) `lu`



(e) `matmul`

`Parabs` **has equal or better performance than** `Domainslib`**.**

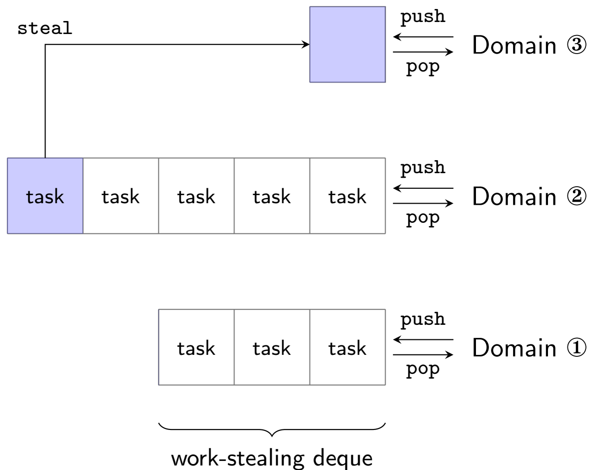
Task Parallelism

Parabs: A Verified Realistic Parallel Scheduler

Chase-Lev Work-Stealing Deque

Future work

Work-Stealing Deque



Chase-Lev Work-Stealing Deque

Dynamic Circular Work-Stealing Deque

David Chase
Sun Microsystems Laboratories
Mailstop UBUR02-311
1 Network Drive
Burlington, MA 01803, USA
dr2chase@sun.com

Yossi Lev
Brown University & Sun Microsystems Laboratories
Mailstop UBUR02-311
1 Network Drive
Burlington, MA 01803, USA
yosef.lev@sun.com

ABSTRACT

The non-blocking work-stealing algorithm of Arora, Blumofe, and Plaxton (henceforth *ABP work-stealing*) is on its way to becoming the multiprocessor load balancing technology of choice in both industry and academia. This highly efficient scheme is based on a collection of array-based double-ended queues (deques) with low cost synchronization among local and stealing processes. Unfortunately, the algorithm's synchronization protocol is strongly based on the use of fixed size arrays, which are prone to overflows, especially in the multiprogrammed environments for which they are designed. We present a work-stealing deque that does not have the overflow problem.

The only ABP-style work-stealing algorithm that eliminates the overflow problem is the list-based one presented by Hendler, Lev and Shavit. Their algorithm indeed deals with the overflow problem, but it is complicated, and introduces a trade-off between the space and time complexity, due to the extra work required to maintain the list.

Our new algorithm presents a simple lock-free work-stealing deque, which stores the elements in a cyclic array that can grow when it overflows. The algorithm has no limit other than integer overflow (and the system's memory size) on the

1. INTRODUCTION

The ABP work-stealing algorithm of Arora, Blumofe, and Plaxton [2] has been gaining popularity as the multiprocessor load-balancing technology of choice in both industry and academia [2, 1, 4, 9]. The scheme implements a provably efficient work-stealing paradigm due to Blumofe and Leiserson [3] that allows each process to maintain a local work deque,[1] and steal an item from others if its deque becomes empty. The deque's owner process pushes and pops local work to and from the deque's bottom end. To minimize synchronization overhead for the deque's owner, stolen elements are taken from the top end of the deque. No elements are added to the top end of the deque. An ABP deque thus presents three methods in its interface:

- `pushBottom(Object o)`:
  Pushes *o* onto the bottom of the deque.
- `Object popBottom()`:
  Pops an object from the bottom of the deque if the deque is not empty, otherwise returns *Empty*.
- `Object steal()`:
  If the deque is empty, returns *Empty*. Otherwise, re-

Déjà-Vu

Verification of Chase-Lev work-stealing deque
(work in progress)

Clément Allain
François Pottier

Inria Paris

October 15, 2025

1 / 19

Déjà-Vu

Conclusion

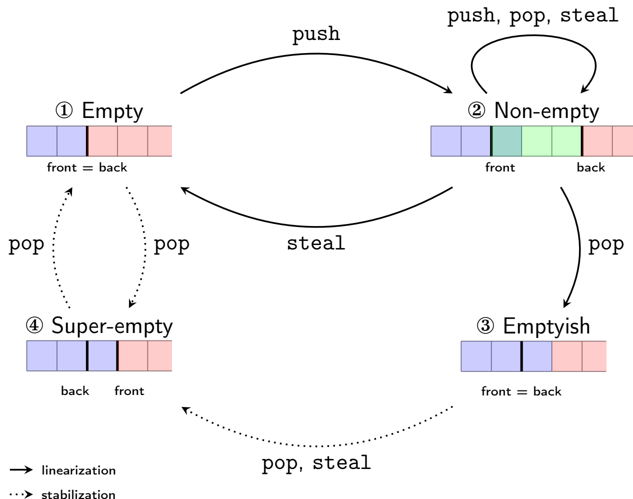
- Coq mechanization is available on **`github`** :
  `https://github.com/clef-men/caml5`
- Simplified Chase-Lev deque (one infinite array) ✓
  Real-life Chase-Lev deque (multiple circular arrays) ⚠
- Proof looks more complex than the sketch. In particular, transitions between logical states are not really formalized.
- We plan to verify more primitives (Domainslib, Taskflow) based on Chase-Lev deque. This is thanks to modularity of IRIS specifications.

My First Interaction With Robbert Krebbers

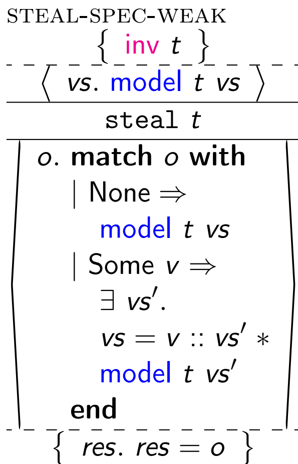
Already verified in `iris/examples`
without prophecy variables!

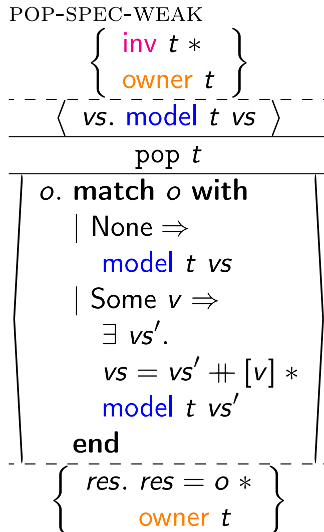
Robbert Krebbers, Iris Workshop 2023

Future-Dependent Linearization Point



Weak Specification





Weak Specification

STEAL-SPEC-WEAK

$\{ \textsf{inv}\ t \}$

POP-SPEC-WEAK

$\{ \textsf{inv}\ t * \textsf{owner}\ t \}$

This specification is *too weak* to prove completion :
Once the scheduler terminates, all submitted tasks have finished.

OBLIGATION-FINISHED

$$\frac{\textsf{obligation}\ \mathit{pool}\ P \qquad \textsf{finished}\ \mathit{pool}}{\triangleright \Box P}$$

CONSUMER-FINISHED

$$\frac{\textsf{consumer}\ \mathit{pool}\ P \qquad \textsf{finished}\ \mathit{pool}}{\Rrightarrow P}$$

model t vs

end

$\{ \mathit{res}.\ \mathit{res} = o \}$

model t vs

end

$\{ \mathit{res}.\ \mathit{res} = o * \textsf{owner}\ t \}$

Strong Specification

STEAL-SPEC

$$\{\ \mathsf{inv}\ t\ \}$$
$$\langle\ vs.\ \mathsf{model}\ t\ vs\ \rangle$$
$$\texttt{steal}\ t$$
$$\langle\ \mathsf{model}\ t\ (\mathsf{tail}\ vs)\ \rangle$$
$$\{\ res.\ res = \mathsf{head}\ vs\ \}$$

POP-SPEC

$$\left\{\begin{array}{l}\mathsf{inv}\ t\ * \\ \mathsf{owner}\ t\ ws\end{array}\right\}$$
$$\langle\ vs.\ \mathsf{model}\ t\ vs\ \rangle$$
$$\texttt{pop}\ t$$
$$\left\langle\begin{array}{l} o\ ws'.\ \textbf{match}\ o\ \textbf{with} \\ \quad \mid \mathsf{None} \Rightarrow \\ \quad\quad vs = [] * ws' = [] * \\ \quad\quad \mathsf{model}\ t\ [] \\ \quad \mid \mathsf{Some}\ v \Rightarrow \\ \quad\quad \exists\ vs'. \\ \quad\quad vs = vs' +\!\!+ [v] * ws' = vs' * \\ \quad\quad \mathsf{model}\ t\ vs' \\ \quad \textbf{end}\end{array}\right\rangle$$
$$\left\{\begin{array}{l} res.\ res = o\ * \\ \quad \mathsf{owner}\ t\ ws'\end{array}\right\}$$

Strong Specification

POP-SPEC

$$\left\{ \begin{array}{l} \mathsf{inv}\ t\ * \\ \mathsf{owner}\ t\ ws \end{array} \right\}$$

$$\langle\, vs.\ \mathsf{model}\ t\ vs \,\rangle$$

$$\mathrm{pop}\ t$$

$o\ ws'.$ **match** o **with**

| Some $v \Rightarrow$
  $\exists\, vs'.$
  $vs = vs' +\!\!+ [v] * ws' = vs' *$
  $\mathsf{model}\ t\ vs'$

end

$$\left\{ \begin{array}{l} res.\ res = o\ * \\ \quad \mathsf{owner}\ t\ ws' \end{array} \right\}$$

STEAL-SPEC

$$\langle\, \mathsf{model}\ t\ (\mathrm{tail}\ vs) \,\rangle$$

$$\{\, res.\ res = \mathrm{head}\ vs \,\}$$

This specification is *sufficient* to prove completion :
Once the scheduler terminates, all submitted tasks have finished.

Prophecy Variables / Prophets

- **Primitive prophets**
  Predict the future of the execution.
- **Typed prophets**
  Prediction belongs to a user-supplied semantic type.
- **Wise prophets**
  Remember past predictions.
  Eliminate inconsistent branches.
- **Multiplexed prophets**
  Separate logically independent predictions on a single prophet.

Multiplexed Prophets

SNAPSHOT-GET

$$\frac{\textsf{model}\ \mathit{pid}\ \gamma\ \mathit{pasts}\ \mathit{prophss}}{\textsf{snapshot}\ \gamma\ k\ (\mathit{pasts}\ k)\ (\mathit{prophss}\ k)}$$

SNAPSHOT-VALID

$$\frac{\textsf{model}\ \mathit{pid}\ \gamma\ \mathit{pasts}\ \mathit{prophss} \qquad \textsf{snapshot}\ \gamma\ k\ \mathit{past}\ \mathit{prophs}}{\exists\ \mathit{past}'.\ \mathit{pasts}\ k = \mathit{past} +\!\!+\ \mathit{past}' * \mathit{prophs} = \mathit{past}' +\!\!+\ \mathit{prophss}\ k}$$

RESOLVE-SPEC

$$\frac{\begin{array}{c}\textsf{atomic}\ e \\ \textsf{to-val}\ e = \textsf{None} \qquad v = \mathit{prophet}.\textsf{to-val}\ \mathit{proph} \qquad \textsf{model}\ \mathit{pid}\ \gamma\ \mathit{pasts}\ \mathit{prophss} \\ \textsf{wp}_{\mathcal{E}}\ e \left\{ \begin{array}{l} w.\ \forall\ \mathit{prophs}. \\ \quad \mathit{prophss}\ k = \mathit{proph} :: \mathit{prophs} -\!\!* \\ \quad \textsf{model}\ \mathit{pid}\ \gamma\ (\textsf{alter}\ (\cdot +\!\!+ [\mathit{proph}])\ k\ \mathit{pasts})\ (\mathit{prophss}\,[k \mapsto \mathit{prophs}]) -\!\!* \\ \quad \Phi\ w \end{array} \right\}\end{array}}{\textsf{wp}_{\mathcal{E}}\ \textsf{Resolve}\ e\ \mathit{pid}\ (k, v)\ \{\ \Phi\ \}}$$

Task Parallelism

Parabs: A Verified Realistic Parallel Scheduler

Chase-Lev Work-Stealing Deque

Future work

Future work

- Functors
- Relaxed memory
- Other scheduling strategies
- Heterogeneous scheduling (CPUs & GPUs)

Thank you for your attention!