The Lazy Task Creation Machine

Umut Acar, Arthur Charguéraud, Adrien Guatto, Mike Rainey, and Filip Sieczkowski

ERC Deepsea, Inria Gallium

Gallium Seminar - July 28th 2016
Parallel algorithmics

- Design algorithms with good parallel complexity.
- Difficult, or even impossible for some problems.
- Use low-level machine models or informal arguments.

Parallel programming

- Implement parallel algorithms on real-world computers.
- Many low- and high-level abstractions and systems.
- Parallel programming has traditionally been done by experts.
Parallel algorithmics

- Design algorithms with good parallel complexity.
- Difficult, or even impossible for some problems.
- Use low-level machine models or informal arguments.

Parallel programming

- Implement parallel algorithms on real-world computers.
- Many low- and high-level abstractions and systems.
- Parallel programming has traditionally been done by experts.

*Translating a scalable algorithm into an efficient parallel program is difficult with today’s mainstream hardware architectures.*
Scheduling parallel programs

- **Static**: compile time; needs program and machine models in general.
- **Dynamic**: run time; few hypotheses; adds overhead.
From Logical to Physical Parallelism on Multicores

Scheduling parallel programs

- **Static**: compile time; needs program and machine models in general.
- **Dynamic**: run time; few hypotheses; adds overhead.
Key Ingredients of Dynamic Scheduling

Load balancing
- Dispatch ready tasks over cores in order to maximize utilization.
- Intensely studied, from both theoretical and practical standpoints.
- Task handling incurs large overheads w.r.t. sequential execution.
- A popular technique: *Work Stealing*.

Granularity control
- Tries to minimize the creation of small tasks.
- Used in industrial runtime systems such as Cilk and Intel® TBB.
- Much less understood than load balancing.
- A popular technique: *Lazy Task Creation*.
Key Ingredients of Dynamic Scheduling

Load balancing

- Dispatch ready tasks over cores in order to maximize utilization.
- Intensely studied, from both theoretical and practical standpoints.
- Task handling incurs large overheads w.r.t. sequential execution.
- A popular technique: Work Stealing.

Granularity control

- Tries to minimize the creation of small tasks.
- Used in industrial runtime systems such as Cilk and Intel® TBB.
- Much less understood than load balancing.
- A popular technique: Lazy Task Creation.
Investigating Lazy Task Creation

The Lazy Task Creation granularity control policy

- Trade potential parallelism for lower overhead.
  - Run newly created parallel tasks sequentially by default.
  - Create actual parallelism only when enough sequential work is done.
- Usually described in terms of concrete implementations.
- No analytic performance study.
Investigating Lazy Task Creation

The Lazy Task Creation granularity control policy

- Trade potential parallelism for lower overhead.
  - Run newly created parallel tasks sequentially by default.
  - Create actual parallelism only when enough sequential work is done.
- Usually described in terms of concrete implementations.
- No analytic performance study.

This Talk

- Lazy Task Creation as a non-standard cost semantics.
  - Big-step semantics for a strict λ-calculus with parallel pairs.
  - Extract parallelism from the stack in a lazy manner.
- Compare it to the fully sequential and fully parallel semantics.
  - Prove its soundness: does it compute the same thing?
  - Bound its overheads: what is its effect on performance?
Outline

1. Introduction

2. The Semantics of Lazy Task Creation

3. Efficiency and Soundness Results

4. Perspectives
A standard $\lambda$-calculus with a non-standard notation for pairs.

$$\text{Exp} \ni e ::= x \mid \lambda x.e \mid (e \; e) \mid (e \; || \; e) \mid \text{fst} \; e \mid \text{snd} \; e$$

Values are closures or pairs of values.

$$\text{Val} \ni v ::= (\lambda x.e)\{\sigma\} \mid (v, v)$$

$$\text{Env} \ni \sigma = \text{Var} \rightarrow_{\text{fin}} \text{Val}$$
Syntax of the Sequential Abstract Machine

Frames are expression constructors with formal holes and partial results.

\[
Frame \ni f ::= \text{APPL}(\square, e, \sigma) \mid \text{APPR}((\lambda x. e)\{\sigma\}, \square) \\
\mid \text{PAIRL}(\square, e, \sigma) \mid \text{PAIRR}(v, \square) \\
\mid \text{FST}(\square) \mid \text{SND}(\square)
\]

Stacks are list of frames.

\[
Kont \ni k ::= \text{TOP} \mid f :: k
\]

Heads are either an expression in its local environment or a final value.

\[
Head \ni h ::= (e \mid \sigma) \mid (v \mid -)
\]

Machines are formed by heads in front of stacks.

\[
Mach \ni m ::= \langle h \mid k \rangle
\]
Transitions of the Sequential Abstract Machine

$$m \rightarrow m$$

<table>
<thead>
<tr>
<th>Expression</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x) (\sigma) (k)</td>
<td>(\sigma(x)) (\sigma) (k)</td>
</tr>
<tr>
<td>(\lambda x. e) (\sigma) (k)</td>
<td>((\lambda x. e){\sigma}) (-) (k)</td>
</tr>
<tr>
<td>((e_1 \ e_2)) (\sigma) (k)</td>
<td>(e_1) (\sigma) (k)</td>
</tr>
<tr>
<td>((\lambda x. e){\sigma}) (-) (\text{APPL}(\□, e_2, \sigma')::k)</td>
<td>(e_2) (\sigma') (\text{APPR}((\lambda x. e){\sigma}, \□)::k)</td>
</tr>
<tr>
<td>(\nu) (-) (\text{APPR}((\lambda x. e){\sigma}, \□)::k)</td>
<td>(e) (\sigma[x \mapsto \nu])</td>
</tr>
<tr>
<td>((e_1 \</td>
<td></td>
</tr>
<tr>
<td>(\nu_1) (-) (\text{PAIRL}(\□, e_2, \sigma)::k)</td>
<td>(e_2) (\sigma) (\text{PAIRR}(\nu_1, \□)::k)</td>
</tr>
<tr>
<td>(\nu_2) (-) (\text{PAIRR}(\nu_1, \□)::k)</td>
<td>((\nu_1, \nu_2)) (-)</td>
</tr>
<tr>
<td>(\text{fst} e) (\sigma) (k)</td>
<td>(e) (\sigma) (\text{FST}(\□)::k)</td>
</tr>
<tr>
<td>((\nu_1, \nu_2)) (-) (\text{FST}(\□)::k)</td>
<td>(\nu_1) (-)</td>
</tr>
<tr>
<td>(\text{snd} e) (\sigma) (k)</td>
<td>(e) (\sigma) (\text{SND}(\□)::k)</td>
</tr>
<tr>
<td>((\nu_1, \nu_2)) (-) (\text{SND}(\□)::k)</td>
<td>(\nu_2) (-)</td>
</tr>
</tbody>
</table>
The Fully Sequential Evaluation Judgment

- We assume that all the transitions of a machine are constant time.
- Thus we take the cost of a sequence of transitions to be its length.
- It will turn out to be convenient to have an explicit big-step judgment.

\[
\text{SeqVal} \langle v \mid - \mid \text{TOP} \rangle \Rightarrow \text{seq} v ; 0
\]

\[
\text{SeqSimpleStep} \langle h \mid k \rangle \rightarrow \langle h' \mid k' \rangle \langle h' \mid k' \rangle \Rightarrow \text{seq} v ; n
\]
We assume that all the transitions of a machine are constant time.
Thus we take the cost of a sequence of transitions to be its length.
It will turn out to be convenient to have an explicit big-step judgment.

\[ \mathit{SeqVal} \]

\[ \langle v \mid - \mid \mathit{TOP} \rangle \Rightarrow_{\mathit{seq}} v; 0 \]
The Fully Sequential Evaluation Judgment

- We assume that all the transitions of a machine are constant time.
- Thus we take the cost of a sequence of transitions to be its length.
- It will turn out to be convenient to have an explicit big-step judgment.

\[
\begin{array}{c}
\text{SEQVAL} \\
\langle v \mid - \mid \text{TOP} \rangle \Rightarrow_{\text{seq}} v; 0
\end{array}
\]

\[
\begin{array}{c}
\text{SEQSIMPLESTEP} \\
\langle h \mid k \rangle \rightarrow \langle h' \mid k' \rangle \\
\langle h' \mid k' \rangle \Rightarrow_{\text{seq}} v; n \\
\langle h \mid k \rangle \Rightarrow_{\text{seq}} v; 1 + n
\end{array}
\]
Assigning costs to parallel executions require two distinct notions.

- The *work* is the total number of elementary steps performed.
- The *span* is the length of the longest sequential subexecution.
Assigning costs to parallel executions require two distinct notions.

- The *work* is the total number of elementary steps performed.
- The *span* is the length of the longest sequential subexecution.

We thus define costs to be series/parallel directed acyclic graphs.

\[
\text{Cost } \ni \ g \ ::= \ 0 \mid 1 \mid (g \cdot g) \mid (g \parallel g)
\]
Assigning costs to parallel executions require two distinct notions.

- The *work* is the total number of elementary steps performed.
- The *span* is the length of the longest sequential subexecution.

We thus define costs to be series/parallel directed acyclic graphs.

\[ \text{Cost} \ni g ::= \quad 0 \mid 1 \mid (g \cdot g) \mid (g \parallel g) \]

Work and span are defined inductively on the structure of graphs.

\[
\begin{align*}
\text{work}(0) & = 0 \\
\text{work}(1) & = 1 \\
\text{work}(g_1 \cdot g_2) & = \text{work}(g_1) + \text{work}(g_2) \\
\text{work}(g_1 \parallel g_2) & = \text{work}(g_1) + \text{work}(g_2) \\
\text{span}(0) & = 0 \\
\text{span}(1) & = 1 \\
\text{span}(g_1 \cdot g_2) & = \text{span}(g_1) + \text{span}(g_2) \\
\text{span}(g_1 \parallel g_2) & = \max(\text{span}(g_1), \text{span}(g_2))
\end{align*}
\]
Assigning costs to parallel executions require two distinct notions.

- The work is the total number of elementary steps performed.
- The span is the length of the longest sequential subexecution.

We thus define costs to be series/parallel directed acyclic graphs.

$$\text{Cost} \ni g ::= 0 \mid 1 \mid (g \cdot g) \mid (g \parallel g)$$

Work and span are defined inductively on the structure of graphs.

$$\begin{align*}
\text{work}(0) &= 0 \\
\text{work}(1) &= 1 \\
\text{work}(g_1 \cdot g_2) &= \text{work}(g_1) + \text{work}(g_2) \\
\text{work}(g_1 \parallel g_2) &= \tau + \text{work}(g_1) + \text{work}(g_2)
\end{align*}$$

$$\begin{align*}
\text{span}(0) &= 0 \\
\text{span}(1) &= 1 \\
\text{span}(g_1 \cdot g_2) &= \text{span}(g_1) + \text{span}(g_2) \\
\text{span}(g_1 \parallel g_2) &= \tau + \max(\text{span}(g_1), \text{span}(g_2))
\end{align*}$$

We assume that the overhead of synchronization is a fixed constant $\tau \in \mathbb{N}$. 
The Fully Sequential Evaluation Judgment Revisited

**SEQVal**

\[
\begin{align*}
\langle v \mid - \mid \text{TOP} \rangle & \Rightarrow_{\text{seq}} v; 0 \\
\end{align*}
\]

**SEQSimpleStep**

\[
\begin{align*}
\langle h \mid k \rangle & \Rightarrow \langle h' \mid k' \rangle \\
\langle h' \mid k' \rangle & \Rightarrow_{\text{seq}} v; n \\
\langle h \mid k \rangle & \Rightarrow_{\text{seq}} v; 1 \cdot g \\
\end{align*}
\]
The Fully Sequential Evaluation Judgment Revisited

\[ m \Rightarrow_{\text{seq}} v; g \]

**SEQVAL**

\[ \langle v \mid - \mid \text{TOP} \rangle \Rightarrow_{\text{seq}} v; 0 \]

**SEQSIMPLESTEP**

\[ \langle h \mid k \rangle \rightarrow \langle h' \mid k' \rangle \quad \langle h' \mid k' \rangle \Rightarrow_{\text{seq}} v; n \]

\[ \langle h \mid k \rangle \Rightarrow_{\text{seq}} v; 1 \cdot g \]

Work and span are always equal and do not depend on \( \tau \).
The Fully Parallel Evaluation Judgment

\[ m \Rightarrow_{\text{par}} v; g \]
The Fully Parallel Evaluation Judgment

\[ m \Rightarrow_{\text{par}} v; g \]

\[ \text{PARVAL} \]

\[ \langle v \mid - \mid \text{TOP} \rangle \Rightarrow_{\text{par}} v; 0 \]
The Fully Parallel Evaluation Judgment

\[ m \Rightarrow_{\text{par}} v; g \]

**ParVal**

\[ \langle v \mid - \mid \text{TOP} \rangle \Rightarrow_{\text{par}} v; 0 \]

**ParSimpleStep**

\[ h \neq (\_ \| \_ \mid _) \mid - \quad \langle h \mid k \rangle \rightarrow \langle h' \mid k' \rangle \quad \langle h' \mid k' \rangle \Rightarrow_{\text{par}} v; g \]

\[ \langle h \mid k \rangle \Rightarrow_{\text{par}} v; (1 \cdot g) \]
The Fully Parallel Evaluation Judgment

\[ m \Rightarrow_{\text{par}} v; g \]

**ParVal**

\[ \langle v \mid - \mid \text{TOP} \rangle \Rightarrow_{\text{par}} v; 0 \]

**ParSimpleStep**

\[ h \neq (- \parallel -) \mid - \quad \langle h \mid k \rangle \rightarrow \langle h' \mid k' \rangle \quad \langle h' \mid k' \rangle \Rightarrow_{\text{par}} v; g \]

\[ \langle h \mid k \rangle \Rightarrow_{\text{par}} v; (1 \cdot g) \]

**ParPair**

\[ \langle e_1 \mid \sigma \mid \text{TOP} \rangle \Rightarrow_{\text{par}} v_1; g_1 \]

\[ \langle e_2 \mid \sigma \mid \text{TOP} \rangle \Rightarrow_{\text{par}} v_2; g_2 \]

\[ \langle (v_1, v_2) \mid - \mid k \rangle \Rightarrow_{\text{par}} v; g_3 \]

\[ \langle (e_1 \parallel e_2) \mid \sigma \mid k \rangle \Rightarrow_{\text{par}} v; ((g_1 \parallel g_2) \cdot g_3) \]
Defers task creation until enough sequential work has been done.

Assume "enough" is a fixed constant $N \in \mathbb{N}$.

Maintain a work counter $n \in \mathbb{N}$ incremented at sequential steps.

Look for \text{PAIRL}($\Box, e_2, \sigma$) frames to extract parallelism when $n \geq N$.

Rest of left branch

@ \text{PAIRL}($\Box, e_2, \sigma$)

Right branch

$\Box$ Final continuation
Defer task creation until enough sequential work has been done.

Formally:

\[
\text{Assume “enough” is a fixed constant } N \in \mathbb{N}.
\]

\[
\text{Maintain a work counter } n \in \mathbb{N} \text{ incremented at sequential steps.}
\]

\[
m/n, N \Rightarrow a m o v \text{;}
\]

\[
\text{Look for PAIRL(□, e^2, σ) frames to extract parallelism when } n \geq N.
\]
Towards Lazy Task Creation

Deferring task creation until enough sequential work has been done.

Formally:

- Assume “enough” is a fixed constant $N \in \mathbb{N}$.
Towards Lazy Task Creation

Defer task creation until enough sequential work has been done.

Formally:

- Assume “enough” is a fixed constant $N \in \mathbb{N}$.
- Maintain a work counter $n \in \mathbb{N}$ incremented at sequential steps.

\[
m/n, N \Rightarrow_{amo} v; g
\]
Towards Lazy Task Creation

*Defer task creation until enough sequential work has been done.*

Formally:

- Assume “enough” is a fixed constant \( N \in \mathbb{N} \).
- Maintain a work counter \( n \in \mathbb{N} \) incremented at sequential steps.

\[
m/n, N \Rightarrow_{amo} v; g
\]

- Look for \( \text{PAIRL}(\Box, e_2, \sigma) \) frames to extract parallelism when \( n \geq N \).

\[
\langle h \mid k_1 \rangle \ @ \ PAIRL(\Box, e_2, \sigma) :: k_2
\]
Towards Lazy Task Creation

Defer task creation until enough sequential work has been done.

Formally:

- Assume “enough” is a fixed constant $N \in \mathbb{N}$.
- Maintain a work counter $n \in \mathbb{N}$ incremented at sequential steps.

$$m/n, N \Rightarrow^\text{amo} v; g$$

- Look for $\text{PAIRL}(\Box, e_2, \sigma)$ frames to extract parallelism when $n \geq N$.

$$\langle h \mid k_1 \rangle \@ \text{PAIRL}(\Box, e_2, \sigma) :: k_2$$

Rest of left branch
Towards Lazy Task Creation

Defer task creation until enough sequential work has been done.

Formally:

- Assume “enough” is a fixed constant \( N \in \mathbb{N} \).
- Maintain a work counter \( n \in \mathbb{N} \) incremented at sequential steps.

\[
m/n, N \Rightarrow_{amo} v; g
\]

- Look for \( \text{PAIRL}(\Box, e_2, \sigma) \) frames to extract parallelism when \( n \geq N \).

\[
\langle h \mid k_1 \rangle \quad @ \quad \text{PAIRL}(\Box, e_2, \sigma) \quad ::= \quad k_2
\]

Rest of left branch

Right branch
Defer task creation until enough sequential work has been done.

Formally:

- Assume “enough” is a fixed constant $N \in \mathbb{N}$.
- Maintain a work counter $n \in \mathbb{N}$ incremented at sequential steps.

$$\frac{m}{n}, N \Rightarrow_{\text{amo}} v; g$$

- Look for $\text{PAIRL}(\Box, e_2, \sigma)$ frames to extract parallelism when $n \geq N$.

\[
\left\langle h \left| k_1 \right. \right. \right. \atop \text{Rest of left branch} \quad \left. \text{PAIRL}(\Box, e_2, \sigma) \right. \atop \text{Right branch} \quad \left. \vdots \right. \atop \text{Final continuation} \left. k_2 \right. \right. 
\]
The Amortized Evaluation Judgment

\[ \frac{m}{n}, N \Rightarrow_{amo} v; g \]
The Amortized Evaluation Judgment

\[ \frac{m/n, N \Rightarrow_{\text{amo}} v; g}{\text{AmoVal}} \]

\[ \langle v \mid - \mid \text{TOP} \rangle_{/n, N} \Rightarrow_{\text{amo}} v; 0 \]
The Amortized Evaluation Judgment

\[ \frac{m/n, N \Rightarrow \text{amo } v; g}{\text{AmoVal}} \]

\[ \langle v \mid - \mid \text{TOP} \rangle_{/n, N} \Rightarrow \text{amo } v; 0 \]

\[ \text{AmoSimpleStep} \]

\[ \frac{n < N \lor \text{PAIRL}(\square, _, _) \not\in k}{\langle h \mid k \rangle \rightarrow \langle h' \mid k' \rangle \langle h' \mid k' \rangle_{/n+1, N} \Rightarrow \text{amo } v; g}{\langle h \mid k \rangle_{/n, N} \Rightarrow \text{amo } v; (1 \cdot g)} \]
The Amortized Evaluation Judgment

\[
m/n, N \Rightarrow_{amo} v; g
\]

**AmoVal**

\[
\langle v \mid - \mid TOP \rangle/n, N \Rightarrow_{amo} v; 0
\]

**AmoSimpleStep**

\[
n < N \lor \text{PAIRL}(\Box, -, -) \notin k
\]

\[
\langle h \mid k \rangle \rightarrow \langle h' \mid k' \rangle \quad \langle h' \mid k' \rangle/n+1, N \Rightarrow_{amo} v; g
\]

\[
\langle h \mid k \rangle/n, N \Rightarrow_{amo} v; (1 \cdot g)
\]

**AmoPromote**

\[
n \geq N \quad \text{PAIRL}(\Box, -, -) \notin k_2
\]

\[
\langle e_2 \mid \sigma \mid TOP \rangle/0, N \Rightarrow_{amo} v_2; g_2 \quad \langle (v_1, v_2) \mid - \mid k_2 \rangle/0, N \Rightarrow_{amo} v; g_3
\]

\[
\langle h \mid k_1 \circ \text{PAIRL}(\Box, e_2, \sigma) :: k_2 \rangle/n, N \Rightarrow_{amo} v; ((g_1 \parallel g_2) \cdot g_3)
\]
Outline

1. Introduction
2. The Semantics of Lazy Task Creation
3. Efficiency and Soundness Results
4. Perspectives
### Statement

If $\langle e \mid \emptyset \mid \text{TOP} \rangle_{0,N} \Rightarrow_{\text{amo}} \nu; g_a$, there is $g_s$ s.t. $\langle e \mid \emptyset \mid \text{TOP} \rangle \Rightarrow_{\text{seq}} \nu; g_s$.

Moreover, we have:

$$W_a \leq (1 + \frac{\tau}{N}) \cdot W_s$$

with $W_a$ and $W_s$ the work of $g_a$ and $g_s$, respectively.
If $\langle e \mid \emptyset \mid \text{TOP} \rangle_{/0, N} \Rightarrow_{amo} v; g_a$, there is $g_s$ s.t. $\langle e \mid \emptyset \mid \text{TOP} \rangle \Rightarrow_{seq} v; g_s$.

Moreover, we have:

$$W_a \leq (1 + \frac{\tau}{N}) \cdot W_s$$

with $W_a$ and $W_s$ the work of $g_a$ and $g_s$, respectively.

Proof ingredients:

- The “inductive” bound is $W_a \leq (1 + \frac{\tau}{N}) \cdot W_s + n \cdot \frac{\tau}{N}$. 
Work Bound

Statement

If \( \langle e \mid \emptyset \mid \text{TOP} \rangle_{/0,N} \Rightarrow_{amo} v; g_a \), there is \( g_s \) s.t. \( \langle e \mid \emptyset \mid \text{TOP} \rangle \Rightarrow_{seq} v; g_s \).

Moreover, we have:

\[
W_a \leq (1 + \frac{\tau}{N}) \cdot W_s
\]

with \( W_a \) and \( W_s \) the work of \( g_a \) and \( g_s \), respectively.

Proof ingredients:

- The “inductive” bound is \( W_a \leq (1 + \frac{\tau}{N}) \cdot W_s + n \cdot \frac{\tau}{N} \).
- If \( \langle h \mid k_1 \rangle \rightarrow \langle h' \mid k'_1 \rangle \) then \( \langle h \mid k_1 @ k_2 \rangle \rightarrow \langle h' \mid k'_1 @ k_2 \rangle \).
Statement

If \( \langle e \mid \emptyset \mid \text{TOP} \rangle_{0,N} \Rightarrow_{\text{amo}} v; g_a \), there is \( g_s \) s.t. \( \langle e \mid \emptyset \mid \text{TOP} \rangle \Rightarrow_{\text{seq}} v; g_s \).
Moreover, we have:

\[
W_a \leq (1 + \frac{\tau}{N}) \cdot W_s
\]

with \( W_a \) and \( W_s \) the work of \( g_a \) and \( g_s \), respectively.

Proof ingredients:

- The “inductive” bound is \( W_a \leq (1 + \frac{\tau}{N}) \cdot W_s + n \cdot \frac{\tau}{N} \).
- If \( \langle h \mid k_1 \rangle \rightarrow \langle h' \mid k_1' \rangle \) then \( \langle h \mid k_1 \@ k_2 \rangle \rightarrow \langle h' \mid k_1' \@ k_2 \rangle \).
- The following rule is derivable.

\[
\begin{array}{c}
\text{SEQ Cut} \\
\hline
\langle h \mid k_1 \rangle \Rightarrow_{\text{seq}} v_1; g_1 \\
\langle v_1 \mid - \mid k_2 \rangle \Rightarrow_{\text{seq}} v_2; g_2
\end{array}
\]

\[
\Rightarrow_{\text{seq}} v_2; g_1 \cdot g_2
\]
Span Bound

Statement

If $\langle e \mid \emptyset \mid \text{TOP} \rangle /_0, N \Rightarrow_{\text{amo}} v; g_a$, there is $g_p$ s.t. $\langle e \mid \emptyset \mid \text{TOP} \rangle \Rightarrow_{\text{par}} v; g_p$.

Moreover, we have:

$$S_a \leq (1 + \frac{N}{\tau}) \cdot S_p$$

with $S_a$ and $S_p$ the span of $g_a$ and $g_p$, respectively.
Span Bound

Statement

If \( \langle e \mid \emptyset \mid \text{TOP} \rangle_{/0,N} \Rightarrow_{\text{amo}} v; g_a \), there is \( g_p \) s.t. \( \langle e \mid \emptyset \mid \text{TOP} \rangle \Rightarrow_{\text{par}} v; g_p \).

Moreover, we have:

\[
S_a \leq (1 + \frac{N}{\tau}) \cdot S_p
\]

with \( S_a \) and \( S_p \) the span of \( g_a \) and \( g_p \), respectively.

Proof ingredients:

- The “inductive” bound is \( S_a \leq (1 + \frac{N}{\tau}) \cdot S_p - (p \ ? \ min(n, N) : 0) \cdot \tau \)
  with \( p \) true iff \( \text{PAIRL}(\square, -, -) \in k \).
Statement

If \( \langle e \mid \emptyset \mid \text{TOP} \rangle_{0,N} \Rightarrow_{\text{amo}} v; g_a \), there is \( g_p \) s.t. \( \langle e \mid \emptyset \mid \text{TOP} \rangle \Rightarrow_{\text{par}} v; g_p \).
Moreover, we have:
\[
S_a \leq (1 + \frac{N}{\tau}) \cdot S_p
\]
with \( S_a \) and \( S_p \) the span of \( g_a \) and \( g_p \), respectively.

Proof ingredients:

- The “inductive” bound is \( S_a \leq (1 + \frac{N}{\tau}) \cdot S_p - (p \land \min(n, N) : 0) \cdot \tau \) with \( p \) true iff \( \text{PAIRL}(\square, _, _) \in k \).
- The proof uses an extended parallel judgment which eagerly evaluates frames of shape \( \text{PAIRL}(\square, e_2, \sigma) \) in parallel.
Span Bound

Statement

If $\langle e \mid \emptyset \mid \text{TOP} \rangle_{/0,N} \Rightarrow \text{amo} \ \nu; g_a$, there is $g_p$ s.t. $\langle e \mid \emptyset \mid \text{TOP} \rangle \Rightarrow \text{par} \ \nu; g_p$.

Moreover, we have:

$$S_a \leq (1 + \frac{N}{\tau}) \cdot S_p$$

with $S_a$ and $S_p$ the span of $g_a$ and $g_p$, respectively.

Proof ingredients:

- The “inductive” bound is $S_a \leq (1 + \frac{N}{\tau}) \cdot S_p - (\nu \ ? \ min(n, N) : 0) \cdot \tau$ with $\nu$ true iff $\text{PAIRL}(\Box, _, _) \in k$.

- The proof uses an extended parallel judgment which eagerly evaluates frames of shape $\text{PAIRL}(\Box, e_2, \sigma)$ in parallel.

- If $\langle h \mid k_1 \otimes k_2 \rangle \Rightarrow \text{par} \ \nu; g$ with $\text{PAIRL}(\Box, _, _) \not\in k_2$, there is a value $\nu_1$ s.t. $\langle h \mid k_1 \rangle \Rightarrow \text{par} \ \nu_1; g_1$ and $\langle \nu_1 \mid - \mid k_2 \rangle \Rightarrow \text{par} \ \nu; g_2$ with $g = g_1 \cdot g_2$. 
Soundness

Sequential-to-amortized
If $m \Rightarrow_{\text{seq}} v; g_s$ then for any $n, N$ there is $g_a$ s.t. $m/n, N \Rightarrow_{\text{amo}} v; g_a$.

Parallel-to-amortized
If $m \Rightarrow_{\text{par}} v; g_p$ then for any $n, N$ there is $g_a$ s.t. $m/n, N \Rightarrow_{\text{amo}} v; g_a$.

Amortized-to-amortized
If $m/n, N \Rightarrow_{\text{amo}} v; g_a$ then for any $n', N'$ there is $g_a'$ s.t. $m/n', N' \Rightarrow_{\text{amo}} v; g_a'$.

Remarks:
▶ We have not yet proved amortized-to-amortized soundness.
▶ The other results mostly rely on stack decomposition lemmas.
Soundness

Sequential-to-amortized
If \( m \Rightarrow_{\text{seq}} v; g_s \) then for any \( n, N \) there is \( g_a \) s.t. \( m/n, N \Rightarrow_{\text{amo}} v; g_a \).

Parallel-to-amortized
If \( m \Rightarrow_{\text{par}} v; g_p \) then for any \( n, N \) there is \( g_a \) s.t. \( m/n, N \Rightarrow_{\text{amo}} v; g_a \).

Amortized-to-amortized
If \( m/n, N \Rightarrow_{\text{amo}} v; g_a \) then for any \( n', N' \) there is \( g'_a \) s.t. \( m/n', N' \Rightarrow_{\text{amo}} v; g'_a \).

Remarks:
- We have not yet proved amortized-to-amortized soundness.
- The other results mostly rely on stack decomposition lemmas.
Definitions, statements, and proofs have been mechanized in Coq.

- The complete development spans 758 LoC at the moment.
- No big complication with respect to pen-and-paper proofs.
  - No binder issues since substitutions are not involved.
- All results were first proved in Coq except for the work bound.
Outline

1 Introduction

2 The Semantics of Lazy Task Creation

3 Efficiency and Soundness Results

4 Perspectives
A Library Implementation of Lazy Task Creation

- Written in C++ with a little bit of x86 assembly.
- Much less naive than what I presented.
  - Compiled rather than interpreted.
  - Relies on techniques used for implementing control operators.
  - Measure execution time much more concretely (wall-clock time).
- Offers parallel for loops in addition to parallel pairs.
  - Generalize binary fork/join to n-ary fork/join.
  - Expose more information to the runtime system.
- Performance looks at least as good as in comparable systems.
Future Work

In the short term:

▶ Finish and clean up the Coq development.
▶ Add and study n-ary fork/join, probably as a looping construct.
▶ Complement the theoretical study with more empirical data.

Longer-term possibilities:

▶ Understand the limitations of the technique beyond fork/join.
▶ Justify the cost semantics with respect to a lower-level semantics.
▶ Study the metatheory of cost semantics?
I presented a formal description of Lazy Task Creation, a granularity control technique used in parallel runtime systems.

- The description takes the form of a relatively simple mix of big-step and small-step operational semantics.
- Defining the cost of each execution makes it possible to study the (idealized) performance of programs in a high-level way.
I presented a formal description of Lazy Task Creation, a granularity control technique used in parallel runtime systems.

- The description takes the form of a relatively simple mix of big-step and small-step operational semantics.
- Defining the cost of each execution makes it possible to study the (idealized) performance of programs in a high-level way.

This formal description makes it possible to prove the first analytic bounds on the overhead of Lazy Task Creation.
I presented a formal description of Lazy Task Creation, a granularity control technique used in parallel runtime systems.

- The description takes the form of a relatively simple mix of big-step and small-step operational semantics.
- Defining the cost of each execution makes it possible to study the (idealized) performance of programs in a high-level way.

This formal description makes it possible to prove the first analytic bounds on the overhead of Lazy Task Creation.

In particular, we showed that with $N$ set to $\tau$, Lazy Task Creation properly amortizes scheduling overheads while still preserving parallelism.