Advanced Topics in Compilers Task

Simone Campanoni
simone.campanoni@northwestern.edu
Outline

• What is a Task in NOELLE

• Creation of a Task

• Invoking a Task
Task in NOELLE

• Sources:
src/core/task

• Header:
install/noelle/core/Task.hpp
A task \( t \) is a wrapper of
1. A set of instructions \( S \) organized in basic blocks cloned from the original code
2. \( S \) is wrapped into a new function \( f \) and
3. An environment \( e \) that includes live-in and live-out variables of \( S \)

\[ i: \ldots \]
\[ j: \%v1 = \%v0 + 1 \]
\[ k: \%v2 = \%v1 \times 4 \]
\[ z: \text{call } @g(\%v2) \]

**Get live-in values**
\[ j': \%v1' = \%v0' + 1 \]
\[ k': \%v2' = \%v1' \times 4 \]

**Propagate live-out values**

**Code mapping**

<table>
<thead>
<tr>
<th>Value</th>
<th>Live-In ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>%v0</td>
<td>True</td>
</tr>
<tr>
<td>%v2</td>
<td>False</td>
</tr>
</tbody>
</table>

**Table**

<table>
<thead>
<tr>
<th>Original</th>
<th>Clone</th>
</tr>
</thead>
<tbody>
<tr>
<td>%v0</td>
<td>%v0'</td>
</tr>
<tr>
<td>%v1</td>
<td>%v1'</td>
</tr>
<tr>
<td>%v2</td>
<td>%v2'</td>
</tr>
</tbody>
</table>
Task in NOELLE

A task \( t \) is a wrapper of

1. A set of instructions \( S \) organized in basic blocks cloned from the original code
2. \( S \) is wrapped into a new function \( f \) and
3. An environment \( e \) that includes live-in and live-out variables of \( S \)

- \( f \) is called “task body”
- \( e \) is called “task environment”
- \( t \) has a static unique ID (\( \text{uint64}_t \)) and a dynamic instance ID
  - The static ID is set by the Task abstraction automatically
  - The instance ID is a Value * and whoever defines \( t \) is responsible to create it and register it to Task
A task $t$ is a wrapper of

1. A set of instructions $S$ organized in basic blocks cloned from the original code
2. $S$ is wrapped into a new function $f$ and
3. An environment $e$ that includes live-in and live-out variables of $S$
Task in NOELLE: task signature

A task $t$ is a wrapper of
1. A set of instructions $S$ organized in basic blocks cloned from the original code
2. $S$ is wrapped into a new function $f$ and
3. An environment $e$ includes pointers to all live-in and live-out variables of $S$

- Whoever creates $t$ is responsible to define the signature of $f$
  - $f$ needs to obtain as inputs everything that it needs to execute
  - An instance of $e$ (of some shape/form) needs to be an input of $f$
  - The return type of the signature of $f$ can only be `void`
  - The signature is an input to the Task constructor
Task in NOELLE: body definition

A task $t$ is a wrapper of
1. A set of instructions $S$ organized in basic blocks cloned from the original code
2. $S$ is wrapped into a new function $f$ and
3. An environment $e$ that includes live-in and live-out variables of $S$
Task in NOELLE: task signature

A task $t$ is a wrapper of
1. A set of instructions $S$ organized in basic blocks cloned from the original code
2. $S$ is wrapped into a new function $f$ and
3. An environment $e$ includes pointers to all live-in and live-out variables of $S$

```c++
/*
 * Define the signature of the task.
 */
auto tm = noelle.getTypesManager();
auto funcArgTypes = ArrayRef<Type *>{ tm->getVoidPointerType() };
auto taskSignature = FunctionType::get(tm->getVoidType(), funcArgTypes, false);
```
A task $t$ is a wrapper of
1. A set of instructions $S$ organized in basic blocks cloned from the original code
2. $S$ is wrapped into a new function $f$ and
3. An environment $e$ that includes live-in and live-out variables of $S$

- Whoever creates $t$ is responsible to define the body of $f$
  - The body is first defined by the creation of two basic blocks
    - Entry basic block: first code executed when $f$ is invoked
    - Exit basic block: last code executed before leaving $f$
    - Both basic blocks are empty
Task in NOELLE: task signature

A task $t$ is a wrapper of
1. A set of instructions $S$ organized in basic blocks cloned from the original code
2. $S$ is wrapped into a new function $f$ and
3. An environment $e$ includes pointers to all live-in and live-out variables of $S$

```c
/*
 * Define the signature of the task.
 */
auto tm = noelle.getTypesManager();
auto funcArgTypes = ArrayRef<Type *>(tm->getVoidPointerType());
auto taskSignature = FunctionType::get(tm->getVoidType(), funcArgTypes, false);

/*
 * Create an empty task.
 */
auto t = new Task(taskSignature, M);
```
## Task in NOELLE: body definition

A task $t$ is a wrapper of

1. A set of instructions $S$ organized in basic blocks cloned from the original code
2. $S$ is wrapped into a new function $f$ and
3. An environment $e$ that includes live-in and live-out variables of $S$

- Whoever creates $t$ is responsible to define the body of $f$
  - The body is first defined by the creation of two basic blocks
  - New basic blocks are then created by cloning $S$

```cpp
void Task::cloneAndAddBasicBlocks(
    const std::unordered_set<BasicBlock *> &bbs,
    std::function<bool(Instruction *origInst)> filter);
```
Task in NOELLE: body definition

A task \( t \) is a wrapper of
1. A set of instructions \( S \) organized in basic blocks cloned from the original code
2. \( S \) is wrapped into a new function \( f \) and
3. An environment \( e \) that includes live-in and live-out variables of \( S \)

• Whoever creates \( t \) is responsible to define the body of \( f \)
  • The body is first defined by the creation of two basic blocks
  • New basic blocks are then created by cloning \( S \)

```c
/*
 * Define the body.
 */
auto filter = [] (Instruction *i) -> bool {
    return true;
};
t->cloneAndAddBasicBlocks(hottestLoop->getBasicBlocks(), filter);
```
A task \( t \) is a wrapper of
1. A set of instructions \( S \) organized in basic blocks cloned from the original code
2. \( S \) is wrapped into a new function \( f \) and
3. An environment \( e \) that includes live-in and live-out variables of \( S \)
A task $t$ is a wrapper of
1. A set of instructions $S$ organized in basic blocks cloned from the original code
2. $S$ is wrapped into a new function $f$ and
3. An environment $e$ that includes live-in and live-out variables of $S$

- Whoever creates $t$ is responsible to identify and instantiate $e$ correctly
  - $t$ sees $e$ as a `Value *` to be the pointer from which you can reach all live-in and live-out variables of the code wrapped into $f$
  - The data layout of the object pointed by $e$ is decided by whoever designs a task (rather than Task itself)
  - In other words, Task ignores the details about how $e$ looks in memory
Task in NOELLE: environment

A task \( t \) is a wrapper of

1. A set of instructions \( S \) organized in basic blocks cloned from the original code
2. \( S \) is wrapped into a new function \( f \) and
3. An environment \( e \) that includes live-in and live-out variables of \( S \)

\[
\begin{align*}
  i: & \ldots \\
  j: & %v1 = %v0 + 1 \\
  k: & %v2 = %v1 * 4 \\
  z: & \text{call } @g(%v2)
\end{align*}
\]

\[
\begin{align*}
  %p0 &= \text{GEP}(%e, \ldots) \\
  %v0p &= \text{load } %p0 \\
  %v0' &= \text{load } %v0p \\
  %p1 &= \text{GEP}(%e, \ldots) \\
  %v2p &= \text{load } %p1 \\
  \text{store } %v2', %v2p \\
  \text{ret}
\end{align*}
\]

### Code mapping

<table>
<thead>
<tr>
<th>Original</th>
<th>Clone</th>
</tr>
</thead>
<tbody>
<tr>
<td>%v0</td>
<td>%v0'</td>
</tr>
<tr>
<td>%v1</td>
<td>%v1'</td>
</tr>
<tr>
<td>%v2</td>
<td>%v2'</td>
</tr>
</tbody>
</table>

### Value | Live-In ?
---|---
%v0 | True
%v2 | False
A task $t$ is a wrapper of
1. A set of instructions $S$ organized in basic blocks cloned from the original code
2. $S$ is wrapped into a new function $f$ and
3. An environment $e$ that includes live-in and live-out variables of $S$

```
\begin{align*}
&i: \ldots \\
&j: \%v1 = \%v0 + 1 \\
&k: \%v2 = \%v1 * 4 \\
&z: \text{call } \text{@g}(\%v2)
\end{align*}
```

```
\begin{align*}
&\%p = \text{GEP}(\%e, \ldots) \\
&\%v0' = \text{load } \%p \\
&\%v2p = \text{GEP}(\%e, \ldots)
\end{align*}
```

```
\begin{align*}
&j': \%v1' = \%v0' + 1 \\
&k': \%v2' = \%v1' * 4 \\
&\text{store } \%v2', \%v2p \\
&\text{ret}
\end{align*}
```

```
\begin{tabular}{|l|l|}
\hline
\text{Original} & \text{Clone} \\
\hline
\%v0 & \%v0' \\
\%v1 & \%v1' \\
\%v2 & \%v2' \\
\hline
\end{tabular}
```

```
\begin{tabular}{|l|l|}
\hline
\text{Value} & \text{Live-In ?} \\
\hline
\%v0 & True \\
\%v2 & False \\
\hline
\end{tabular}
```
Outline

• What is a Task in NOELLE

• Creation of a Task

• Invoking a Task
Task in NOELLE: task invocation

A task $t$ is a wrapper of
1. A set of instructions $S$ organized in basic blocks cloned from the original code
2. $S$ is wrapped into a new function $f$ and
3. An environment $e$ that includes live-in and live-out variables of $S$

- $t$ is invoked by calling $f$
  - The code that invokes $f$ needs to setup a memory instance of $e$ consistently with the data layout chosen by whoever defined the Task
Task in NOELLE: example0

%le = alloca ...
%v1s = alloca
%v2s = alloca
%v1sp = GEP(%le, 0)
store %v1s, %v1sp
%v2sp = GEP(%le, 1)
store %v2s, %v2sp

i: ...
  j: %v1 = %v0 + 1
  k: %v2 = %v1 * 4
  z: call @g(%v2)

store %v1, %v1s
call @f (%le)
%v2 = load %v2s

Original | Clone
---|---
%v0 | %v0'
%v1 | %v1'
%v2 | %v2'

Value | Live-In ?
---|---
%v0 | True
%v2 | False

%p = GEP(e, ...)
%v0p = load %p
%v0 = load %v0p
%v2p = GEP(e, ...)

j': %v1' = %v0' + 1
k': %v2' = %v1' * 4
store %v2', %v2p
ret
Task in NOELLE: example1

%le = alloca ...
%v1sp = GEP(%le, 0)
%v2sp = GEP(%le, 1)

i: ...

j: %v1 = %v0 + 1
k: %v2 = %v1 * 4
z: call @g(%v2)

store %v1, %v1sp
call @f (%le)
%v2 = load %v2sp

Value | Live-In ?
---|---
%v0 | True
%v2 | False

Original | Clone
---|---
%v0 | %v0'
%v1 | %v1'
%v2 | %v2'

%p = GEP(e, ...)
%v0' = load %p
%v2p = GEP(e, ...)
j': %v1' = %v0' + 1
k': %v2' = %v1' * 4
store %v2', %v2p
ret
Always have faith in your ability

Success will come your way eventually

Best of luck!