Profiling the Memory Usage of OCaml Applications without Changing their Behavior

OCaml 2013

Çagdas Bozman\textsuperscript{1,2,3}
Thomas Gazagnaire\textsuperscript{1}
Fabrice Le Fessant\textsuperscript{2}
Michel Mauny\textsuperscript{3}

OCamlPro\textsuperscript{1} | INRIA\textsuperscript{2} | ENSTA-ParisTech\textsuperscript{3}

Sept. 24 2013 - Boston (MA)
Memory Problems

- **What?**
  - Study the memory behavior of OCaml programs
  - Memory profiling tools

- **Why?**
  - To decrease memory footprint
  - To fix memory leaks
  - To spend less time in memory management
Real World Example – Why3¹ (1/2)

¹Why3 is a platform for deductive program verification (http://why3.lri.fr/)
With locations precision
How do we do that?

$ opam switch 4.00.1+ocp-memprof
$ opam install why3

$ OCAMLRUNPARAM=m why3replayer.opt -C why3.conf p9_16
this step will generate a lot of snapshots of heap image

No need to change your code nor the compilation options.
No impact on execution time.

$ opam install ocp-memprof
$ ocp-memprof -loc -sizes PID
this step analyzes all these snapshots

Look at the graphs.
Snapshots

What is a snapshot?
- Compressed version of the heap
- Location identifiers, graph with pointers, etc.
- Save globals (toplevel modules)

How do we obtain these snapshots?
- Computed by a linear scan of all chunks\(^2\) which contain sets of consecutive blocks.

\(^2\) huge block of memory
Generate A Snapshot

Two ways to trigger the generation of snapshots

- Use OCAMLRUNPARAM=m force a program to generate a snapshot after every GC
- Request explicitly the program to generate a snapshot
  - by sending a HUP signal (very useful for server-like application, cf mldonkey)
  - in module GC, use the following function
    ```ocaml
    val dump_heap : string -> unit
    ```
OCaml memory block:

<table>
<thead>
<tr>
<th>header</th>
<th>word [0]</th>
<th>word [1]</th>
<th>...</th>
</tr>
</thead>
</table>
OCaml memory block:

```
header  word [0]  word [1]  ...
```

OCaml block's header (one word) on 64-bit machines:

- **WOSIZE**: 54 bits
- **COLOR**: 2 bits
- **TAG**: 8 bits
Header after our modification:

```
<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>WOSIZE</th>
<th>COLOR</th>
<th>TAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>21 bits</td>
<td>33 bits</td>
<td>2 bits</td>
<td>8 bits</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>WOSIZE</th>
<th>COLOR</th>
<th>TAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>54 bits</td>
<td>2 bits</td>
<td>8 bits</td>
</tr>
</tbody>
</table>
```
Patched Compiler (3/3)

- Minimal impact on performance (only when generating snapshots)

- Only on 64-bit platforms

- Location identifiers are limited (2^{21} ∼ 2 million allocation sites)

- Maximum block size is now 64GB
Patched Compiler (3/3)

- Minimal impact on performance (only when generating snapshots)
- No space overhead
Patched Compiler (3/3)

- Minimal impact on performance (only when generating snapshots)
- No space overhead
- No impact on GC (its behavior is not changed)

- Only on 64-bit platforms
- Location identifiers are limited (2^21 ∼ 2 million allocation sites)
- Maximum block size is now 64GB
Patched Compiler (3/3)

+ Minimal impact on performance (only when generating snapshots)
  - No space overhead
  - No impact on GC (its behavior is not changed)

- Only on 64-bit platforms
Patched Compiler (3/3)

+ Minimal impact on performance (only when generating snapshots)
+ No space overhead
+ No impact on GC (its behavior is not changed)

− Only on 64-bit platforms
− Location identifiers are limited ($2^{21} \sim 2$ million allocation sites)
Patched Compiler (3/3)

有利点:
- Minimal impact on performance (only when generating snapshots)
- No space overhead
- No impact on GC (its behavior is not changed)

不利点:
- Only on 64-bit platforms
- Location identifiers are limited ($2^{21} \sim 2$ million allocation sites)
- Maximum block size is now 64GB
One Tool Based On Identifiers

memprof.PID.DUMPNAME.COUNT.dump
memprof.PID.locations

STEP 1

Get all identifiers from heap and the corresponding location from memprof.PID.locations

identifier -> location

STEP 2

Find the right type from a specific location using cmt files

cmt  cmt  cmt  cmt

* A cmt file is a binary file containing the typed AST
Conclusion

Future Work:

- Improve the current framework
  - Aggregate information by type and location (work in progress)
  - Recover more types (e.g. using G.Henry’s work)
  - Display life span of values (number of GC for example)
- More tools based to analyzed snapshots:
  - a graphical assistant to explore snapshots
  - a tool which use pointers to see which root retains some specific values

Questions ?