Binary Rewriting at Runtime for Efficient Dynamic Domain Map Implementations

3rd CHIUW Workshop, Chicago, May 27, 2016

Josef Weidendorfer, Jens Breitbart

Chair for Computer Architecture
Department of Informatics, Technical University of Munich

The beginning...

We were looking for an abstraction of data distribution that

- allows for automatic load balancing
- could handle nodes failure
- and is transparent to the user

But performance implications of our concepts were unsatisfactory.

Our solution: binary rewriting at runtime

- Language / programming model independent
  - Directly parse instructions in binary form
  - ISA dependent, but there are far less ISAs

- Use runtime information to optimize code
  - Data distribution among nodes
  - Memory layout

Our API

- Configuration based on C calling convention (ABI)
  - E.g.: „rewrite f into version with parameter 2 == 100“
- Returns a function pointer usable as drop-in-replacement
  - If the condition is true
  - Otherwise use the original function
- In case rewriting fails we return the original function
  - No error handling required

Our API

```
#include "dbrew.h"

void mm_kernel(int s,
               double a[][s], double b[][s], double c[][s],
               int i, int j, int k);


...

dbrew_set_function(r, (uint64_t) mm_kernel);
dbrew_config_staticpar(r, 0); // size is constant
mmf = (mm_t) dbrew_rewrite(r, s, a, b, c, 0, 0, 0);
```

- Rewrite function mm_kernel() for a constant size

Initial Chapel Experiments

- We manually modified the generated C code
- Specialized accesses to data distributed with cyclic compiled for multiple locales

Specialized for a single locale:

→ 54% of instructions removed for array accesses

Available

- Currently in prototyping phase
  - Only parts of the x86_64 ISA
  - We add new instructions as they are required

- Source code is available on GitHub:

https://github.com/lrr-tum/dbrew

Please give it a try and report any issues you find

Feedback welcome!

- Our experiments by itself is obviously not very useful…
- Do you need a component to specialize code at runtime?
- Should something like that be a language feature?