Monadic Composition for Deterministic, Parallel Batch Processing

Ryan Scott\textsuperscript{1}  
Ryan Newton\textsuperscript{1}  
Omar Navarro Leija\textsuperscript{2}  
Joe Devietti\textsuperscript{2}

\textsuperscript{1}Indiana University  
\textsuperscript{2}University of Pennsylvania

rgscott@indiana.edu  
github.com/RyanGlScott
Nondeterminism

Ideal Program

Arguments
Input files

Output files
Nondeterminism

Nondeterministic Program

Arguments

Input files

Output files v1

Output files v2

...
Nondeterminism

Nondeterministic Program

Arguments → Input files

Thread scheduling

Output files v1 → Output files v2 → ...
Nondeterminism

Arguments → Nondeterministic Program → Output files v1
Input files → Nondeterministic Program → Output files v2

Thread scheduling → Nondeterministic Program → Environment leaks → ...

Nondeterminism

Arguments
Input files

Nondeterministic Program

Output files v1
Output files v2
...

Thread scheduling
Environment leaks
Nondet. system calls/CPU instructions
Where nondeterminism hurts
Where nondeterminism hurts

Continuous integration

Same commit, different results #770

amitaibu opened this issue on Nov 19, 2012 · 2 comments

amitaibu commented on Nov 19, 2012

I have a build that was successful, but after "rebuilding" it fails.

Success VS Fail -- so I assume it's related to the environment?
Where nondeterminism hurts

Parallel workflows

```
all: a b c
da b c:
    @for n in 1 2 ; do \
        echo $@-$$n && sleep 1 ; \
    done
```
Serial execution

$ make
a-1
a-2
b-1
b-2
c-1
c-2

Parallel execution
Serial execution

$ make
  a-1
  a-2
  b-1
  b-2
  c-1
  c-2

Parallel execution

$ make -j2
  a-1
  a-2
  b-1
  b-2
  c-1
  c-2
Serial execution

$ make
a-1
a-2
b-1
b-2
c-1
c-2

Parallel execution

$ make -j2
a-1
b-1
b-2
a-2
c-1
c-2
Serial execution

$ make
a-1
a-2
b-1
b-2
c-1
c-2

Parallel execution

$ make -j2
a-1
b-1
a-2
b-2
c-1
c-2
detflow

- Static determinism enforcement
- Dynamic runtime sandboxing
- Low overall overhead
detflow

Static determinism enforcement

Dynamic runtime sandboxing

Low overall overhead
Entrypoints

main :: IO ()
Entrypoints

Haskell

```haskell
main :: IO ()
main = do
    Parallel.mapM_ putStrLn [1..10]
```
main :: IO ()
main = do
    Parallel.mapM_ putStrLn [1..10]
    -- Already nondeterministic!
newtype DetIO a = MkDetIO (IO a)
newtype DetIO a = MkDetIO (IO a)
-- Expose only deterministic API calls
getLine :: DetIO String
putStrLn :: String -> DetIO ()
-- etc.
newtype DetIO a = MkDetIO (IO a)
-- Expose only deterministic API calls
getLine :: DetIO String
putStrLn :: String -> DetIO ()
-- etc.

Key idea: Only expose deterministic operations that can be composed in a deterministic fashion
newtype DetIO a = MkDetIO (IO a)
-- Expose only deterministic API calls
getLine :: DetIO String
putStrLn :: String -> DetIO ()
-- etc.

main :: DetIO ()
main = do
    x <- getLine
    putStrLn x
Parallelism

- `detflow` uses the filesystem for shared-memory parallelism
- Should this be allowed?

```hs
readFile :: FilePath -> DetIO String
writeFile :: FilePath -> String
          -> DetIO String
```
Parallelism

Thread 1

\[
\text{do } \text{writeFile "foo.txt" "Hello, World"}
\]

Thread 2

\[
\text{do } \text{foo <- readFile "foo.txt"}
\text{if foo == "Hello, World"}
\text{then ...}
\text{else ...}
\]
Parallelism

Thread 1

do writeFile "foo.txt"
   "Hello, World"

Thread 2

do foo <- readFile "foo.txt"
   if foo == "Hello, World"
      then ...
      else ...
Solution: permissions

- Every thread holds separate permissions on system filepaths
Solution: permissions

- Every thread holds separate permissions on system filepaths

```
/abcdef/ghijkl/mnopqr
```

<table>
<thead>
<tr>
<th></th>
<th>R 0.5</th>
<th>R 0.5</th>
<th>RW 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parallelism, revisited

\begin{verbatim}
data Perm -- (R/RW) + path + fraction

forkWPerms :: [PathPerm] -> DetIO a
             -> DetIO (Thread a)
joinThread :: Thread a -> DetIO ()
\end{verbatim}

- readFile and writeFile must respect the permissions in a thread’s local state
Permissions checkout

\[
\text{pgm :: DetIO ()}
\]
\[
\text{pgm :: do -- Assume parent starts with R 1.0 on /a}
\]
\[
\text{th1 <- forkWPerms [R "a"]}
\]
\[
\text{computation1}
\]
\[
\text{th2 <- forkWPerms [R "b"]}
\]
\[
\text{computation2}
\]
\[
\text{joinThread t1}
\]
\[
\text{joinThread t2}
\]
Permissions checkout

```
pgm :: DetIO ()
pgm :: do -- Assume parent starts with R 1.0 on /a
  th1 <- forkWPerms [R "/a"]
  computation1
  -- Parent has R 0.5 on /a
  th2 <- forkWPerms [R "/a"]
  computation2

  joinThread t1

  joinThread t2
```
Permissions checkout

\[
\text{pgm} :: \text{DetIO} () \\
\text{pgm} :: \text{do} \quad \text{-- Assume parent starts with R 1.0 on /a} \\
\quad \text{th1} \leftarrow \text{forkWPerms} [\text{R } "/a"] \\
\quad \text{computation1} \\
\quad \text{-- Parent has R 0.5 on /a} \\
\quad \text{th2} \leftarrow \text{forkWPerms} [\text{R } "/a"] \\
\quad \text{computation2} \\
\quad \text{-- Parent has R 0.25 on /a} \\
\quad \text{joinThread } t1 \\
\quad \text{joinThread } t2
\]
Permissions checkout

pgm :: DetIO ()
pgm :: do -- Assume parent starts with R 1.0 on /a
   th1 <- forkWPerms [R “/a”]
      computation1
     -- Parent has R 0.5 on /a
   th2 <- forkWPerms [R “/a”]
      computation2
     -- Parent has R 0.25 on /a
   joinThread t1
     -- Parent has R 0.75 on /a
   joinThread t2
Permissions checkout

```
pgm :: DetIO ()
pgm :: do -- Assume parent starts with R 1.0 on /a
  th1 <- forkWPerms [R “/a”]
  computation1
  -- Parent has R 0.5 on /a
  th2 <- forkWPerms [R “/a”]
  computation2
  -- Parent has R 0.25 on /a
  joinThread t1
  -- Parent has R 0.75 on /a
  joinThread t2
  -- Parent has R 1.0 on /a
```
More detflow

- Replace nondeterministic IO operations with deterministic alternatives
  - Reading system time
  - putStrLn
- Full lattice of permissions, and formalization of permission checkout (see paper)
detflow

Static determinism enforcement

Low overall overhead

Dynamic runtime sandboxing
system calls

system :: String -> DetIO ()
main :: DetIO ()
main = system "gcc foo.c -o foo"
How can we make shelling out to arbitrary programs (not written in DetIO) deterministic?
System calls

\[
\text{system} :: \text{String} \rightarrow \text{DetIO}()
\]

\[
\text{main} :: \text{DetIO}()
\]

\[
\text{main} = \text{system} \ 	ext{"gcc foo.c -o foo"}
\]

- How can we make shelling out to arbitrary programs (not written in DetIO) deterministic?
- Answer: run them in a \textit{deterministic runtime}. 
libdet
libdet

libdet must intercept potential sources of nondeterminism at runtime.
libdet must intercept potential sources of nondeterminism at runtime.

Reading from “banned” directories

- /dev/urandom
- /proc
libdet must intercept potential sources of nondeterminism at runtime.

**Reading from “banned” directories**
- /dev/urandom
- /proc

**Solution**
- Intercept calls to `fopen()` (with `LD_PRELOAD`), error if they read anything blacklisted
Uncontrolled concurrency

- e.g., with pthreads
libdet

libdet must intercept potential sources of nondeterminism at runtime.

Uncontrolled concurrency

- e.g., with pthreads

Solution

- Intercept calls to pthread_create() (with LD_PRELOAD) to run everything sequentially
libdet

libdet must intercept potential sources of nondeterminism at runtime.

Nondeterministic OS properties

- e.g., reading addresses returned by `mmap()`
libdet

libdet must intercept potential sources of nondeterminism at runtime.

Nondeterministic OS properties

• e.g., reading addresses returned by `mmap()`

Solution

• Disable address-space layout randomization (ASLR)
detflow

- Static determinism enforcement
- Dynamic runtime sandboxing
- Low overall overhead
Case studies

• Ran a deterministic version of make against SPLASH2 benchmarks
  • Performance is essentially identical to that of GNU make
• Ported various bioinformatics scripts to deflow and measured parallel speedup
  • Overall performance overhead for determinism enforcement is less than 1%
Selected SPLASH2 benchmarks

(Lower is better)
Bioinformatics apps, parallel speedup

(Higher is better)
Future work

- Reach closer to catching all sources of nondeterminism in runtime
- Dynamic (at-runtime) checkout of permissions
- Make more programs feasible to determinize
detflow development:

https://github.com/iu-parfunc/detmonad

Any questions?
“Hello, World!” throughput

(Higher is better)