Open Problems from my perspective
Our group (last 3 years)
Our group (last 3 years)

High-level parallel prog.
Our group (last 3 years)

High-level parallel prog.
Our group (last 3 years)

- Deterministic parallelism
- Parallel effects
- Profiling
- GPUs
- Concurrent data structs
- Mem mgmt.
- Irregular apps
- Compiler construction

High-level parallel prog.
Our group (last 3 years)

High-level parallel prog.

- Deterministic parallelism
- Parallel effects
- Profiling
- Concurrent data structs
- Mem mgmt.
- Compiler construction
- Irregular apps
- GPUs

[POPL'14] [PLDI'14] [PLDI'16] [ICFP'15A] [ICFP'15B]

[OOPSLA'14] [Haskell'15] [PPoPP'16] [ICFP'16]
1. Archival-quality computation
1. Archival-quality computation
1. Archival-quality computation
1. Archival-quality computation
1. Archival-quality computation
p
input-data $\rightarrow p \rightarrow$ chart
Determinism Enforcement
Determinism
Enforcement
input-data $\rightarrow p \rightarrow$ Determinism Enforcement

control
- Environment
- Execution
Determinism Enforcement

control

- Environment
- Execution
  - dynamic
Determinism

Enforcement

- Environment
- Execution
  - dynamic
  - static

control

\[ p \]
Environment:

Deterministic base image
Environment:

Deterministic base image

input-data

p

docker

NixOS
input-data $p$ Deterministic base image

Execution:

docker  NixOS
Execution:
Execution:

$p$

Dynamic determinism enforcement
Dynamic determinism enforcement
Dynamic determinism enforcement
Dynamic determinism enforcement

- Kendo
- DThreads
- ...

Dynamic determinism enforcement
\( p \) : (Inputs, \textbf{NumThreads}) \rightarrow \text{Outputs}
Reproducibility

\[ p : (\text{Inputs}, \text{NumThreads}) \rightarrow \text{Outputs} \]
Reproducibility

Portability
Reproducing a function $p$:

$\text{Inputs} \rightarrow \text{Outputs}$

- **Reproducibility**
- **Portability**
: \((\text{Inputs}, \ \text{OS Version}) \rightarrow \text{Outputs}\)

- Reproducibility
- Portability
Reproducibility

Portability
\[ p : (\text{Inputs}, \text{Sched. Trace}) \rightarrow \text{Outputs} \]

- Reproducibility
- Portability
Reproducibility

Portability

- Still open problems for dynamic determinism enforcement:
  user-space process tree determinism, file systems
Reproducibility
Portability
Static checking

- Reproducibility
- Portability
TOWARDS A THEORY OF PARALLEL PROGRAMMING

C. A. R. HOARE

(1971)

OBJECTIVES

The objectives in the construction of a theory of parallel programming as a basis for a high-level programming language feature are:
Tony Hoare, 1971, on the prospect of **Static checking**

"It is therefore very important that a high-level language designed for parallel programming should provide complete security against time-dependent errors by means of a *compile-time check.*"
Determinism as a Safety property

{-# LANGUAGE Safe #-}
import Control.LVish
x = <your untrusted code>
main :: IO ()
main = print x
Determinism as a Safety property

{-# LANGUAGE Safe #-}
import Control.LVish
x = <... runPar _ ...>
main :: IO ()
main = print x
Determinism as a Safety property

{-# LANGUAGE Safe #-}

import Control.LVish

x = (... runPar _ ...)  
main :: IO ()
main = print x

{-# LANGUAGE Safe #-}

...
main :: Det ()
main = print x
Relationship to program generation

- Archiving programs in source form requires precise representations of the:
  - code
  - compiler
  - environment
  - (A “perfect name” or hash for a program.)
2. Composable Autotuning

• Typically:
  • Global parameters
  • Hacky Scripts
  • Fixed K-dimensional search spaces

$ cat "1,2,3" > params.txt
$ ./run_stuff.sh
2. Composable Autotuning

- Typically:
  - Global parameters
  - Hacky Scripts
  - Fixed K-dimensional search spaces

```bash
$ cat "1,2,3" > params.txt
$ ./run_stuff.sh
```
Ideal

import A (a)
import B (b)
...

do a — may use auto-tuning
  b — may use auto-tuning
  ...

One idea
One idea

- Use an Applicative transformer (Tune m a)
One idea

• Use an Applicative transformer (Tune m a)

• Gather params before running `m`
One idea

• Use an Applicative transformer (Tune m a)
• Gather params before running `m`
• But individual runs are still anonymous
One idea

• Use an Applicative transformer (Tune m a)

• Gather params before running `m`

• But individual runs are still *anonymous*

Avoid:

```haskell
x = runSearch $
  setParam (Proxy::Proxy "a") (0,10) $
  setParam (Proxy::Proxy "b") (10,20) $
  go
```
Problems:

• *Persistence* is awkward

• Store learned results

• Comes back to naming programs:
  • What is the *same* program?

• And naming *choices*:
  • choice structure is a (un)labeled tree?
3. Fusion for nested, irregular data
3. Fusion for nested, irregular data

fold1(setUnion,
map(nbrs, states))
3. Fusion for nested, irregular data

```
fold1(setUnion, map(nbrs, states))
```

```
do acc ← newEmptySet()
   forEach x ∈ states:
      forEach n ∈ nbrs(x):
         insert(n, acc)
```
4. Data structure representation rewrites
4. Data structure representation rewrites

\[
\text{Set (Maybe } a\text{)} \Rightarrow (\text{Bool, Set } a)
\]
4. Data structure representation rewrites

\[
\text{Set} \ (\text{Maybe} \ a) \Rightarrow (\text{Bool}, \ \text{Set} \ a)
\]

\[
\text{from } s = (\text{Nothing} \in s, \\
\quad \text{map fromJust} \\
\quad (\text{filter isJust } s))
\]
4. Data structure representation rewrites

Set (Maybe a) ⇒ (Bool, Set a)

from s = (Nothing ∈ s,
           map fromJust
           (filter isJust s))

- (Closed type families are sufficient for this example.)
5. User-friendly metaprogramming
5. User-friendly metaprogramming

- Should I need to know whether a particular expression happens at a particular stage?
5. User-friendly metaprogramming

- Should I need to know whether a particular expression happens at a particular stage?

- Staging goals
5. User-friendly metaprogramming

• Should I need to know whether a particular expression happens at a particular stage?

• Staging goals

  • this function isn’t called at runtime (INLINE)
5. User-friendly metaprogramming

- Should I need to know whether a particular expression happens at a particular stage?

- Staging *goals*
  - this function isn’t called at runtime (INLINE)
  - this datatype doesn’t appear at runtime
5. User-friendly metaprogramming

- Should I need to know whether a particular expression happens at a particular stage?

- Staging goals
  - this function isn’t called at runtime (INLINE)
  - this datatype doesn’t appear at runtime
  - this type class creates no dictionaries at runtime
6. Deterministic par. without loose ends
6. Deterministic par. without loose ends

- Don’t assume associativity, commutativity
6. Deterministic par. without loose ends

- Don’t assume associativity, commutativity
- Accelerate, DPJ, etc…
6. Deterministic par. without loose ends

• Don’t assume associativity, commutativity
  • Accelerate, DPJ, etc…

• Prove it!
6. Deterministic par. without loose ends

• Don’t assume associativity, commutativity
  • Accelerate, DPJ, etc…

• Prove it!
  • (... and integrate with static compiler checks like -XSafe)
6. Deterministic par. without loose ends

• Don’t assume associativity, commutativity
  • Accelerate, DPJ, etc…
  • *Prove it!*
  • (... and integrate with static compiler checks like -XSafe)
• *(WIP: w/ Ranjit Jhala)*