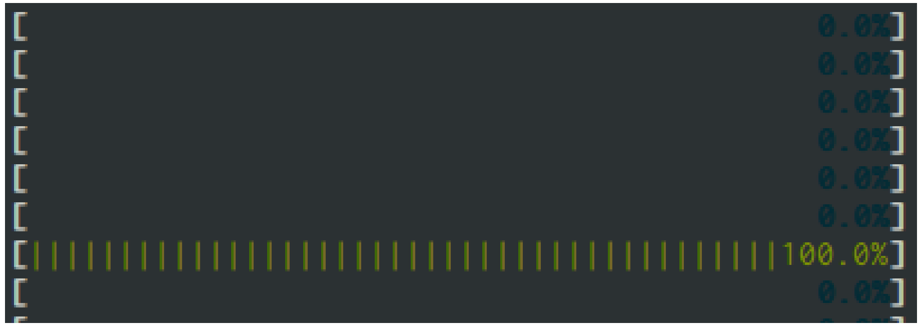
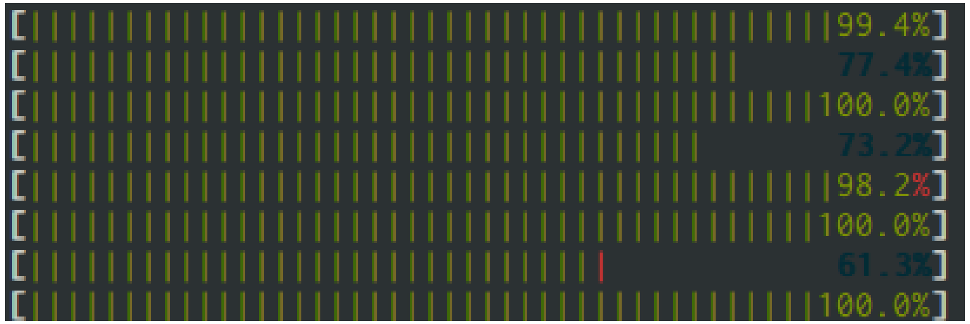
PaSh: A parallelizing shell

or how to get from this:



to this:



github.com/andromeda/pash

Joint work with:

Nikos Vasilakis

And many others (in alphabetical order):

Achilles Benetopoulos

Lazar Cvetkovic

Thurston Dang

Michael Greenberg

Shivam Handa

Kostas Mamouras

Tammam Mustafa

Radha Patel

Martin Rinard

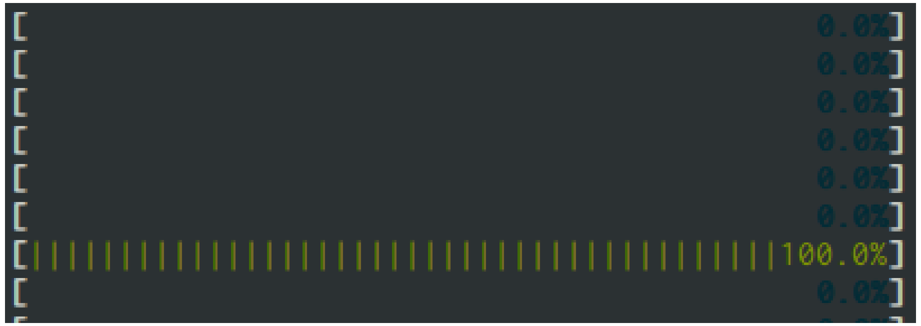
shell

Use

```
echo "Executing test: $microbenchmark"
# Execute the sequential script on the first run only
exec_seq="-s"
for n_in in "${n_inputs[@]}"; do
  echo "Number of inputs: ${n_in}"
  ## Generate the intermediary script
  python3 generate_microbenchmark_intermediary_scripts.py \
    $microbenchmarks_dir $microbenchmark $n_in $intermediary_dir "test"
  for flag in "${flags[@]:1}"; do
    echo "Flag: ${flag}"
    ## Execute the intermediary script
    ./execute_compile_evaluation_script.sh $assert_correctness $exec_seq $flag \
      "${microbenchmark}" "${n_in}" "test_results" "test_" > /dev/null 2>&1
    rm -f /tmp/eager*
    ## Only run the sequential the first time around
    exec_seq=""
  done
done
done
}

execute_tests "" "${script_microbenchmarks[@]}"
execute_tests "-c" "${pipeline_microbenchmarks[@]}"
echo "Below follow the identical outputs:"
grep --files-with-match "are identical" ../evaluation/results/test_result
echo "Below follow the non-identical outputs:"
grep -L "are identical" ../evaluation/results/test_res
TOTAL_TESTS=$(ls -la ../evaluation/result
PASSED_TESTS=$(grep --files-with-ma
echo "Summary: ${PASSED_TESTS
```

But ...



Shell scripts are mostly sequential!

Parallelizing requires a lot of manual effort:

- Using specific command flags (e.g., sort -p, make -jN)
- Using semi-automatic restricted parallelization tools (e.g., GNU parallel)
- Rewriting parts of a script in languages that support parallelism (e.g. Erlang)

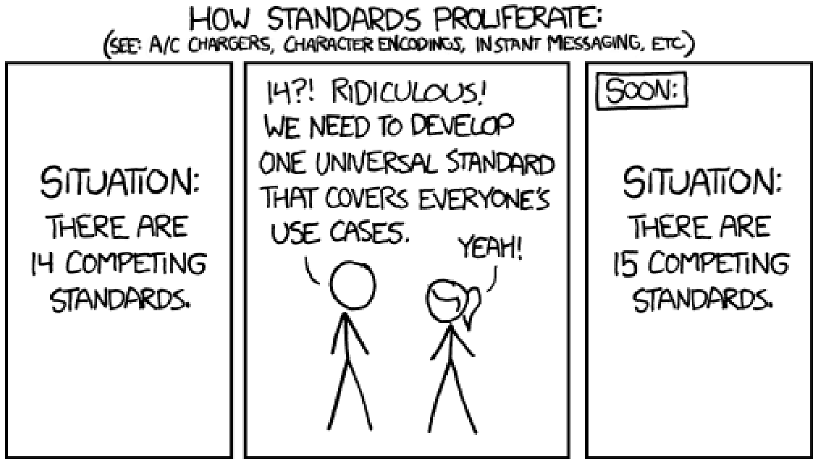
What did we do to deserve this??? :'(

PaSh

PaSh

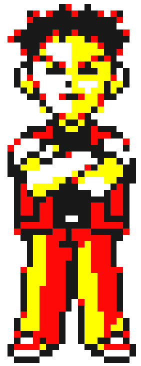
A JiT shell2shell compiler that:

- exposes latent data parallelism in shell scripts
- is a lightweight layer on top of bash
    - Executes the optimized version of the script on bash
    - Negligible slowdown for non parallelizable scripts
    - Correctness w.r.t. bash without implementing a new shell

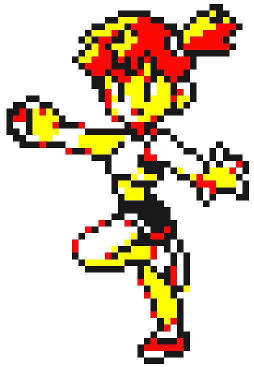


Not so fast!!!

We have to go through them!!!



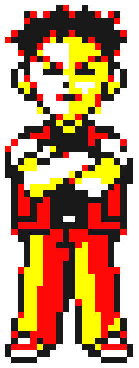
Subtle Parallelism



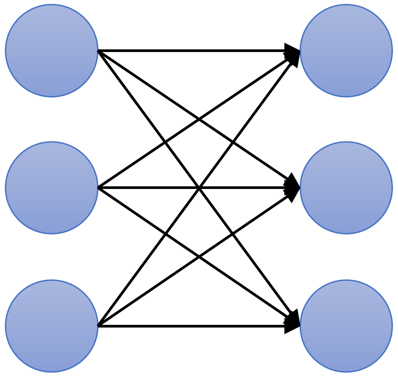
Arbitrary black-box commands

Lack of static information

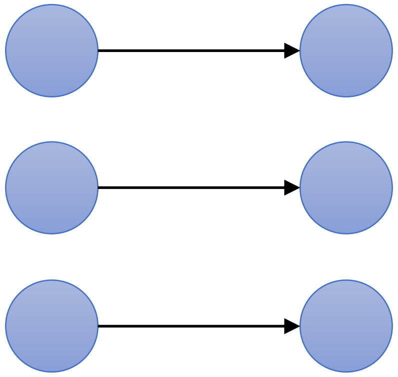
Challenge: Shell Data Parallelism is Subtle



- Parallel frameworks such as MapReduce or Spark
  - Either require commutativity
  - Or key-by independence to achieve parallelism

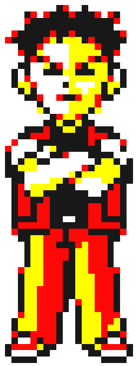


Round Robin



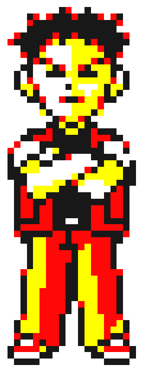
Partition By Key

Challenge: Shell Data Parallelism is Subtle



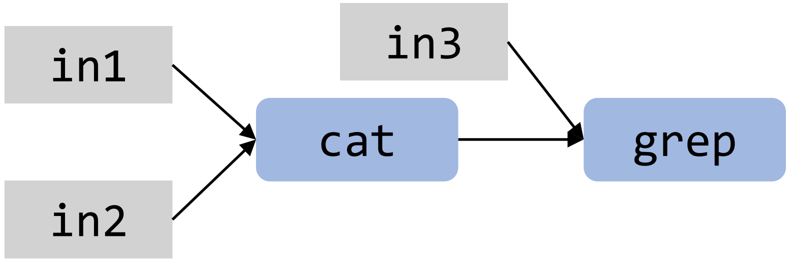
- Data parallelism in the shell is trickier
  - Commutativity and independence based on key is rare
- Commands most often read from their inputs in sequence
  - This order matters for the output
  - For example, `grep "foo" in1 - in2` reads in1, its stdin, and then in2
- We need a model that captures a parallelizable subset of the shell
  - That also captures order of command input consumption

Solution: Order-aware dataflow model

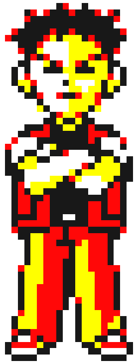


- The following pipeline would be translated to:

```
cat in1 in2 | grep "foo" in3 -
```



- Model defines a shell fragment with no scheduling constraints
  - Intuitively: commands composed with &, |
- The expressiveness allows us to define a bidirectional correspondence between:
  - Shell programs in this fragment
  - Dataflow graphs in our model



Solution: Order-aware dataflow model

- On the graph we have defined semantics preserving transformations.
- M

$$\frac{\text{cmd2node}(w, \overline{x}_o \leftarrow f(\overline{x}_i)) \quad \text{add\_metadata}(f, \overline{as}, \overline{r}) = f' \quad \text{redir}(\overline{x}_o, \overline{x}_i, \overline{r}, \overline{x}'_o, \overline{x}'_i)}{\overline{as} w \overline{r} \Uparrow \langle \text{input}\overline{x}'_i; \text{output}\overline{x}'_o; \overline{x}'_o \leftarrow f'(\overline{x}'_i), \text{fg}\rangle} \ \textsc{CommandTrans}$$

$$\frac{\text{cmd2node}(w, \bot)}{\overline{as} w \overline{r} \Uparrow \overline{as} w \overline{r}} \ \textsc{CommandId} \qquad \frac{c \Uparrow \langle p, b\rangle}{c\& \Uparrow \langle p, \text{bg}\rangle} \ \textsc{BackgroundDfg} \qquad \frac{c \Uparrow c'}{c\& \Uparrow c'\&}$$

$$\frac{c_1 \Uparrow \langle p_1, \text{bg}\rangle \quad c_2 \Uparrow \langle p_2, b\rangle}{c_1; c_2 \Uparrow \langle \text{compose}(p_1, p_2), b\rangle} \ \textsc{SeqBothBg} \qquad \frac{c_1 \Uparrow c'_1 \quad c_2 \Uparrow \langle p_2, \text{bg}\rangle \quad \text{opt}(p_2) \Downarrow c'_2}{c_1; c_2 \Uparrow c'_1; (c'_2\&)}$$

$$\frac{c_1 \Uparrow \langle p_1, \text{fg}\rangle \quad c_2 \Uparrow \langle p_2, \text{bg}\rangle \quad \text{opt}(p_1) \Downarrow c'_1 \quad \text{opt}(p_2) \Downarrow c'_2}{c_1; c_2 \Uparrow c'_1; (c'_2\&)} \ \textsc{SeqBothFgBg}$$

$$\frac{c_1 \Uparrow c'_1 \quad c_2 \Uparrow \langle p_2, \text{fg}\rangle \quad \text{opt}(p_2) \Downarrow c'_2}{c_1; c_2 \Uparrow c'_1; c'_2} \ \textsc{SeqRightFg} \qquad \frac{c_1 \Uparrow c'_1 \quad c_2 \Uparrow c'_2}{c_1; c_2 \Uparrow c'_1; c'_2} \ \textsc{SeqNone}$$

$$\frac{\begin{array}{c} c_1 \Uparrow \langle p_1, b_1\rangle, \dots c_n \Uparrow \langle p_n, b_n\rangle, \\ p'_1 \dots p'_{n-1} = \text{map}(\text{connectpipe}, p_1 \dots p_{n-1}) \\ p = \text{fold\_left}(\text{compose}, p'_1\ p'_2 \dots p'_{n-1}\ p_n) \end{array}}{pipe_{|} c_1\ c_2 \dots c_n\ \& \Uparrow \langle p, bg\rangle} \ \textsc{PipeBG} \qquad \frac{\begin{array}{c} c_1 \Uparrow \langle p_1, b_1\rangle, \dots c_n \Uparrow \langle p_n, b_n\rangle, \\ p'_1 \dots p'_{n-1} = \text{map}(\text{connectpipe}, p_1 \dots p_{n-} \\ p = \text{fold\_left}(\text{compose}, p'_1\ p'_2 \dots p'_{n-1}\ p_n) \end{array}}{pipe_{|} c_1\ c_2 \dots c_n \Uparrow \langle p, fg\rangle}$$

Figure 5. A subset of the compilation rules.

$$\frac{x_j \leftarrow \text{Unused}(\mathcal{I}, \mathcal{O}, \mathcal{E}),\ \mathcal{E}' = \mathcal{E}[x_j/x_i]}{\mathcal{I}, \mathcal{O}, \mathcal{E} \Longleftrightarrow \mathcal{I}, \mathcal{O}, \mathcal{E}' \cup \{x_j \leftarrow \text{relay}(x_i)\}} \ \textsc{Relay}$$

$$\frac{x_s, x'_s \leftarrow \text{Unused}(\mathcal{I}, \mathcal{O}, \mathcal{E}), E = \{\langle x_s, x'_s\rangle \leftarrow \text{split}(x), \langle x_1, \dots x_k\rangle \leftarrow \text{split}(x_s), \langle x_{k+1}, \dots x_m\rangle \leftarrow \text{split}(x'_s)\}}{\mathcal{I}, \mathcal{O}, \mathcal{E} \cup \{\langle x_1, \dots x_m\rangle \leftarrow \text{split}(x)\} \Longleftrightarrow \mathcal{I}, \mathcal{O}, \mathcal{E} \cup E} \ \textsc{Split-Split}$$

$$\frac{x_c, x'_c \leftarrow \text{Unused}(\mathcal{I}, \mathcal{O}, \mathcal{E}),\ E = \{x_c \leftarrow \text{cat}(\langle x_1, \dots x_k\rangle), x'_c \leftarrow \text{cat}(\langle x_{k+1}, \dots x_m\rangle), x \leftarrow \text{cat}(\langle x_c, x'_c\rangle)\}}{\mathcal{I}, \mathcal{O}, \mathcal{E} \cup \{x \leftarrow \text{cat}(\langle x_1, \dots x_m\rangle)\} \Longleftrightarrow \mathcal{I}, \mathcal{O}, \mathcal{E} \cup E} \ \textsc{Concat-Concat}$$

$$\frac{\overline{x} \leftarrow \text{Unused}(\mathcal{I}, \mathcal{O}, \mathcal{E}),\ E = \{\overline{x} \leftarrow \text{split}(x_j), x_i \leftarrow \text{cat}(\overline{x})\}}{\mathcal{I}, \mathcal{O}, \mathcal{E} \cup \{x_i \leftarrow \text{relay}(x_j)\} \Longleftrightarrow \mathcal{I}, \mathcal{O}, \mathcal{E} \cup E} \ \textsc{Split-Concat}$$

$$\frac{\begin{array}{c} x^u_1, x^d_1, x^u_2, x^d_2, \dots x^u_n, x^d_n \leftarrow \text{Unused}(\mathcal{I}, \mathcal{O}, \mathcal{E}), \\ E = \{\langle x^u_1, x^d_1\rangle \leftarrow \text{tee}(x_1), \langle x^u_2, x^d_2\rangle \leftarrow \text{tee}(x_2), \dots \langle x^u_n, x^d_n\rangle \leftarrow \text{tee}(x_n), \\ x_o \leftarrow \text{cat}(x^u_1, x^u_2, \dots x^u_n), x'_o \leftarrow \text{cat}(x^d_1, x^d_2, \dots x^d_n)\} \end{array}}{\mathcal{I}, \mathcal{O}, \mathcal{E} \cup \{x \leftarrow \text{cat}(x_1, x_2, \dots x_n), \langle x_o, x'_o\rangle \leftarrow \text{tee}(x)\} \Longleftrightarrow \mathcal{I}, \mathcal{O}, \mathcal{E} \cup E} \ \textsc{Tee-Concat}$$

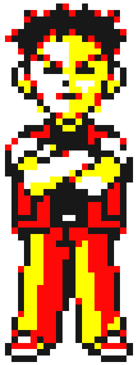
$$\frac{\begin{array}{c} x \leftarrow \text{Unused}(\mathcal{I}, \mathcal{O}, \mathcal{E}), \\ \overline{x}_i = \langle x_1, x_2, \dots x_n\rangle, \overline{x}_j = \langle x'_1, x'_2, \dots x'_n\rangle, E = \{x'_1 \leftarrow \text{relay}(x_1), x'_2 \leftarrow \text{relay}(x_2), \dots x'_n \leftarrow \text{relay}(x_n)\} \end{array}}{\mathcal{I}, \mathcal{O}, \mathcal{E} \cup \{x \leftarrow \text{cat}(\overline{x}_i), \overline{x}_j \leftarrow \text{split}(x)\} \Longrightarrow \mathcal{I}, \mathcal{O}, \mathcal{E} \cup E} \ \textsc{Concat-Split}$$

$$\frac{}{\mathcal{I}, \mathcal{O}, \mathcal{E} \cup \{x_j \leftarrow \text{cat}(x_i)\} \Longrightarrow \mathcal{I}', \mathcal{O}, \mathcal{E}' \cup \{x_j \leftarrow \text{relay}(x_i)\}} \ \textsc{One-Concat}$$

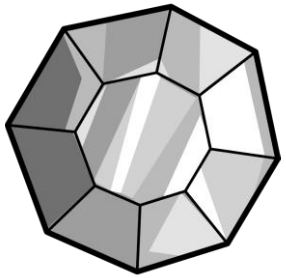
$$\frac{}{\mathcal{I}, \mathcal{O}, \mathcal{E} \cup \{x_j \leftarrow \text{split}(x_i)\} \Longrightarrow \mathcal{I}', \mathcal{O}, \mathcal{E}' \cup \{x_j \leftarrow \text{relay}(x_i)\}} \ \textsc{One-Split}$$

- Int … ggregate
  - Can be parallelized by applying the map on
  - And then applying the aggregate
- Auxiliary transformations enable parallelization by inserting cat + split.

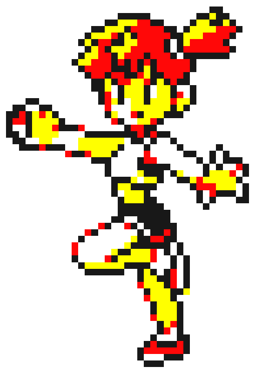
Proofs in the paper!



Subtle Parallelism
was defeated !!!

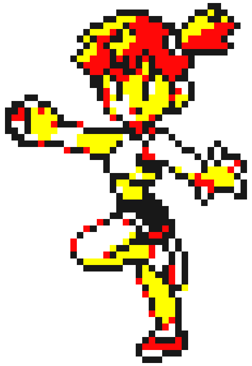


Challenge: Arbitrary black-box commands



- Restricted programming frameworks (MapReduce, Spark, etc)
  - offer a limited set of constructs
  - can be easily mapped to a dataflow abstraction
- The shell is used to compose:
  - arbitrary commands
  - written in arbitrary languages
  - and are updated (or modified) over time
- This makes automated analysis infeasible
- Any one-time effort quickly obsolete and useless.

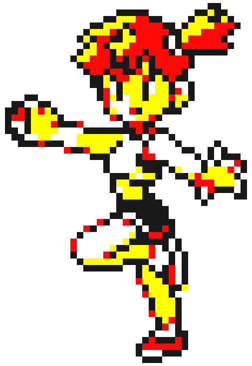
Solution: Node Correspondence Framework



Users describe how to:

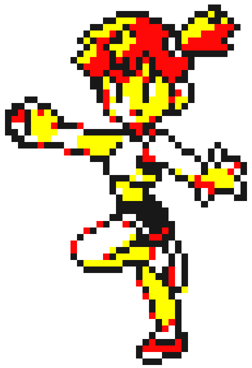
- Map a command to a dataflow node (if possible)
  - Inputs, outputs, parallelizability from arguments
- How to map a dataflow node to a command
  - Instantiating command arguments from inputs, outputs, metadata
- This is achieved by defining two python functions
- Developer instantiates correspondence once for each command
  - The goal is for this to be used by command developers or other experts
  - Library of correspondence can be inspected and shared

Solution: Node Correspondence Framework

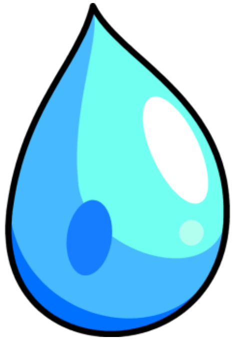


- Many commands have restricted, well-defined behavior
- Designed an annotation language
  - Annotation uniquely defines the two correspondence functions
  - Language guided by study of POSIX and GNU Coreutils
- Part of annotation for cat:
- Defined annotations for 53 commands
- More details in our EuroSys 21 paper

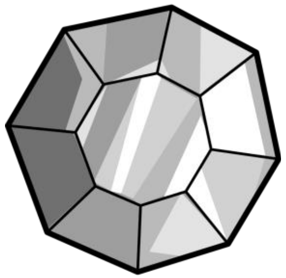
```
{
   "command": "cat",
   "cases": [
     ...,
     {
       "predicate": "default",
       "class": "parallelizable",
       "aggregator": "cat",
       "inputs": ["args[:]"],
       "outputs": ["stdout"]
     }]
}
```

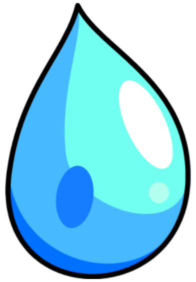


Arbitrary black-box
commands
were defeated !!!



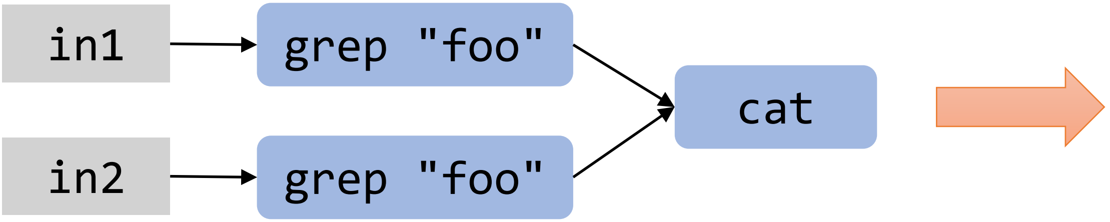
Combining the first 2 badges





- Compiler:
  - Given a shell script
  - Compiles it to a dataflow graph if possible
  - Applies parallelizing transformations
  - Compiles it back to a shell script
- Piggybacking on the shell to execute the parallel script

For more details see our talk on EuroSys 21 next week!




```
mkfifo /tmp/t1 /tmp/t2
grep "foo" in1 > /tmp/t1 &
grep "foo" in2 > /tmp/t2 &
cat /tmp/t1 /tmp/t2 &
wait
rm -f /tmp/t1 /tmp/t2
```

Challenge: Lack of Static information

- Shell execution depends on several dynamic components:
  - File system
  - Current directory
  - Environment variables
  - Unexpanded strings

```
cat $DIR/* | tr A-Z a-z |  tr -cs A-Za-z '\n' |                # (spell)
      sort | uniq | comm -13 $DICT -
```

- Very difficult to have a static parallelization procedure that is both:
  - Sound
  - *Somewhat* effective

Solution: JiT compilation process

- PaSh switches between interpretation and compilation
  - Calling the compiler as late as possible
- Provides critical information to the compiler:
  - State of shell
  - Variables
  - Directory
  - Files and even their contents(!)

Solution: JiT compilation process

- Preprocessor:
  - Parses script
  - Performs analysis to find potential dataflow regions
  - Replaces potential DFG regions with calls to runtime
  - Unparses script
  - Executes it with bash

```
cat $DIR/* | tr A-Z a-z | tr -cs A-Za-z '\n' |
      sort | uniq | comm -13 $DICT - > out ;
cat out | wc -l | sed 's/$/ mispelled words!/'
```

```
source pash_runtime.sh /tmp/pash_ast.TZDAyhVaFr ;
source pash_runtime.sh /tmp/pash_ast.PDmnT7PUug
```

Solution: JiT compilation process

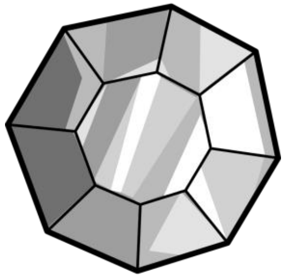
Runtime is just a shell script:

1) Save shell state and set pash default state
   - E.g., variables, previous exit code, etc
2) Call the parallelizing compiler
   - Providing information about the current state
3) Revert the shell state
4) If the compiler has succeeded:
   - Run the produced parallel script
   - Else run the original script
5) Save shell state and set pash default state
6) Finish up pash work
   - E.g., measure (4) exec time
7) Revert shell state

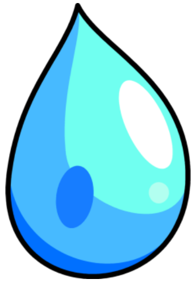
```
-- bash -- | -- pash --
   ...     |
    \----(1)----\
           |       ...
           |       (2)
           |       ...
    /----(3)----/
   ...     |
   (4)     |
   ...     |
    \----(5)----\
           |       ...
           |       (6)
           |       ...
    /----(7)----/
   ...     |
```

Lack of static
information
was defeated !!!

Combining all 3 badges



Order-aware
Dataflow model



Node Correspondence
Framework

Just in Time
compilation

Demo Time

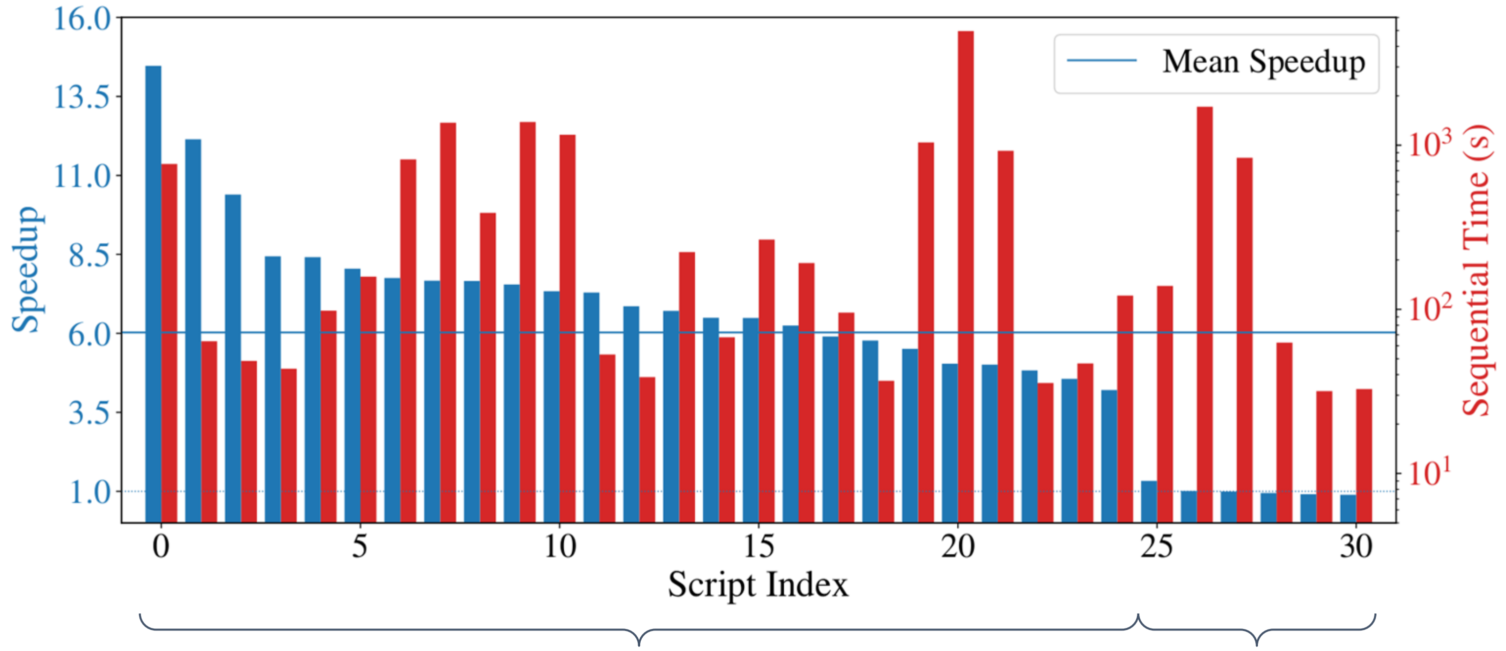
Evaluation

Evaluation

Two aspects:

- Performance Evaluation
- Preliminary Correctness Evaluation

Pipelines in the wild



Parallelizable

Non parallelizable

+ PaSh awareness goes a long way!

e.g. #26

```
cat $IN6 | awk '{print $2, $0}' | sort -nr | cut -d ' ' -f 2    (1.01×)
cat $IN6 |                        sort -nr -k2 | cut -d ' ' -f 1 (8.1× !!1!1)
```

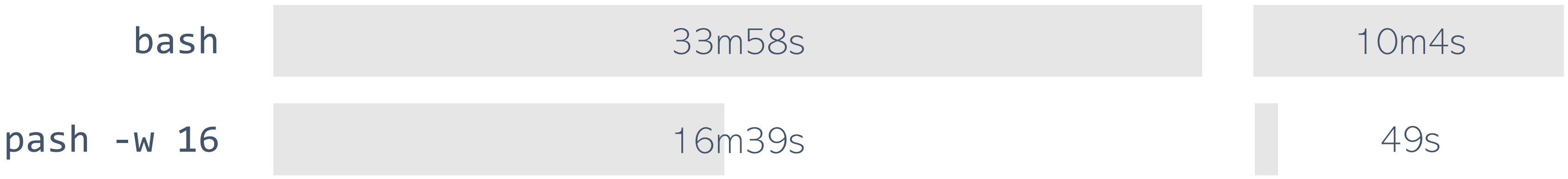
Case Study: NOAA Weather Analysis

82GB (5y weather data)

Hadoop only focuses on this part

fetch, preprocess, cleanup, filter

calculate



2.04×

speedup for preprocessing

12.31×

speedup for preprocessing

2.52×

combined speedup for the full program

This part is not the focus of traditional parallelization frameworks but parallelizing it has the biggest impact

Preliminary correctness evaluation (WIP)

- Smoosh [2] test suite
  - Comprehensive POSIX shell test suite
- Started from the bottom:

```
xxxxxxxxxx.xxxxxxxxxx.xx..xxxxxxxxxx.x.xxx.xxxxxxxx.xxxx...xx.xxxxxxxx
xx.xxxx.xx..xxxxxxxxxxxxxxxxxxxx.xxxxxxxxxxx.xxxxxxx.xxxxxxx.xxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
shell_tests.sh: 20/174 tests passed
```

- Now we are here:

```
xxx.x....x.xxxx...x...x...x..x.xxxx.....xx..xx.x.....x......x.xx.xx.x.
.xx.x.x......xx.xx...xx........xxxxx..xx.x....x.x......x.......x...xxx
xxxx.....x.xx.x..xxxxx..xxxx.xxxxx
shell_tests.sh: 98/174 tests passed
```

- Meanwhile bash:

```
..x.x....x.xxxx...x.......x..x..xxx......x..xx.x.....x......x.xx.xx.x.
.x..x.x..........................x.......x......x...........x........x
.xx......x..x................x..x.
shell_tests.sh: 136/174 tests passed
```

[2] Greenberg, Michael, and Austin J. Blatt. "Executable formal semantics for the POSIX shell." POPL. 2019.

Conclusion

Discussion

- Shell scripts have mostly escaped the PL community attention
  - Some notable exceptions: Smoosh [2], dgsh [3], Shark [4]
- This is mostly because:
  1) Commands have arbitrary behaviors and cannot be easily analyzed
  2) Shell's dynamic nature makes static analysis incorrect or ineffective
  3) Shell semantics is **BLACK MAGIC**
- Recent work [2] addressed (3)
- PaSh makes a step towards addressing (1) and (2)
  - Enabling further study of the performance and correctness of shell scripts

[2] Greenberg, Michael, and Austin J. Blatt. "Executable formal semantics for the POSIX shell." POPL. 2019.
[3] D. Spinellis and M. Fragkoulis, "Extending Unix Pipelines to DAGs," in *IEEE Transactions on Computers*. 2017.
[4] Berger, Emery D. "Optimizing Shell Scripting Languages." 2003.

Thank you :)

- PaSh is open source
- Upcoming talk at EuroSys next week (ask me for preprint)
- Upcoming HotOS paper and panel on the future of the shell
- More exciting research on the shell on its way!