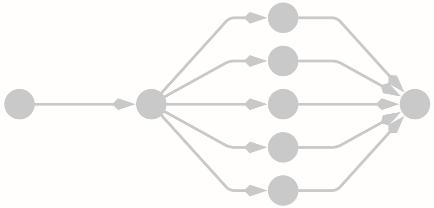
GNU PARALLEL

Parallelizing and
Distributing programs with the Shell

Felix Rieg and Florian Sihler



July 3, 2022
CCPDP, Ulm University

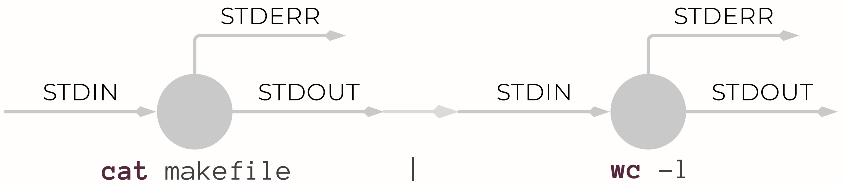
Motivation

Doing stuff parallel.

With the commandline!

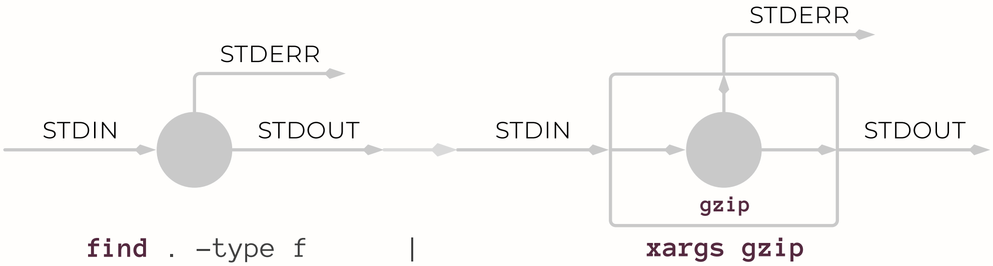
[3]: *GNU Parallel 20210822 ('Kabul')* Tange, 2021

Pipes



```
cat makefile | wc -l
```

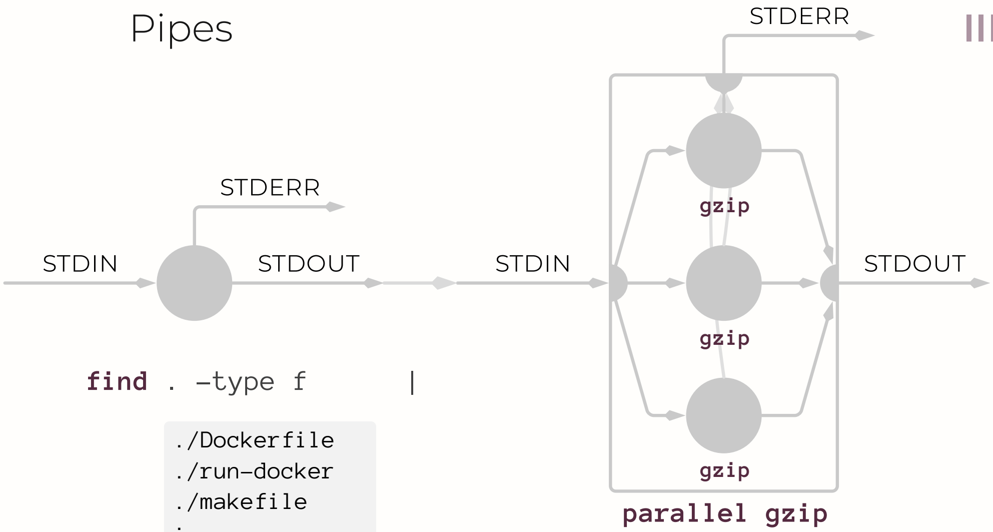
Pipes



```
find . -type f | xargs gzip
```

```
./Dockerfile
./run-docker
./makefile
⋮
```

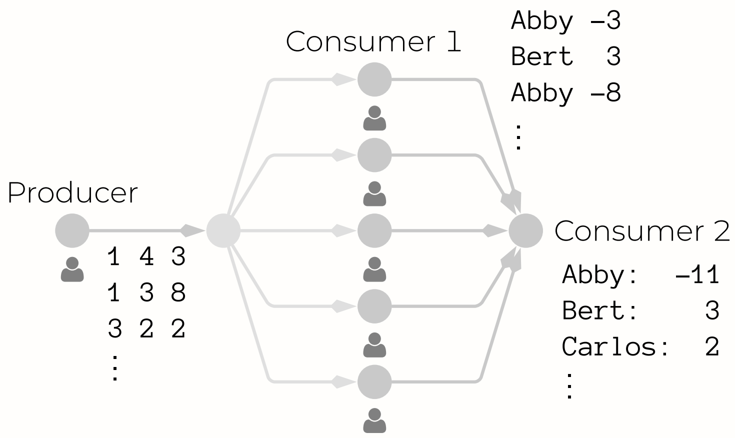
Pipes



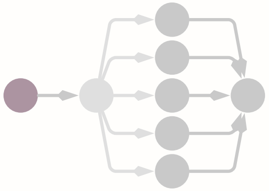
```
find . -type f        |
```

```
./Dockerfile
./run-docker
./makefile
⋮
```

A simple Bank



A simple Bank



Producer.java

```
final var rand = new Random();
```

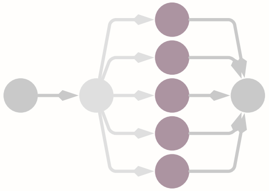
number of transactions

known account names

```
for (int i = 0; i < n; i++) {
   int from = rand.nextInt(NAMES.length);
   int to = rand.nextInt(NAMES.length);
   System.out.format("%d␣%d␣%d%n", from, to, rand.nextInt(100) * 10);
}
```

value

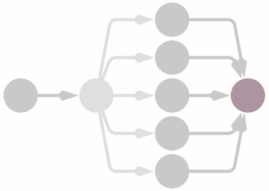
A simple Bank



Consumer.java

```
final var scanner = new Scanner(System.in);
while (scanner.hasNextLine()) {
   String[] s = scanner.nextLine().split("␣");
                                                  0
   String from = NAMES[Integer.parseInt(s[FROM])];
                                                                2
   System.out.printf("%s␣-%d%n", from, Integer.parseInt(s[VALUE]));
                                              1
   String to = NAMES[Integer.parseInt(s[TO])];
   System.out.printf("%s␣%d%n", to, Integer.parseInt(s[VALUE]));
}
scanner.close();
```

A simple Bank



Accountant.java

```
final var scanner = new Scanner(System.in);
final var accounts = new HashMap<String, Integer>();
while (scanner.hasNextLine()) {
   String[] trans = scanner.nextLine().split("␣");

   final var old = accounts.getOrDefault(trans[0], 0);
   accounts.put(trans[0], old + Integer.parseInt(trans[1]));
}
scanner.close();
System.out.println(accounts);
```

Arbitrary initialization

AbstractMap.toString()

Running the Example

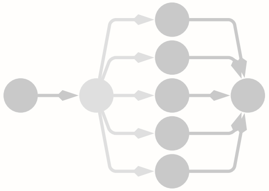
```
java -jar producer.jar 1000000\
         | java -jar consumer.jar\
         | java -jar accountant.jar
```

```
  pid     ppid   cpuid   cmd
25621    25618       7   /bin/bash -c java -jar producer.jar 1000000 | java -jar consumer.jar | java -jar accountant.jar
25622    25621       2   java -jar producer.jar 1000000
25623    25621      14   java -jar consumer.jar
25624    25621      13   java -jar accountant.jar
25738     5113       9   ps -o pid,ppid,cpuid,cmd
```

Ran from within a makefile (directly done by shell otherwise)

Produces (a superset of) this table

Running the Example



```
java -jar producer.jar 1000000\
      | parallel --pipe java -jar consumer.jar\
      | java -jar accountant.jar
```

```
  pid     ppid    cpuid   cmd
 38279   38276      3    /bin/bash -c java -jar producer.jar 1000000 | parallel --pipe java -jar consumer.jar | java -jar a...
 38280   38279      6    java -jar producer.jar 1000000
 38281   38279     15    perl /usr/bin/parallel --pipe java -jar consumer.jar
 38282   38279      5    java -jar accountant.jar
 38344   38281     10    perl -e if(sysread(STDIN,$buf,1)){open($fh[...]} /usr/bin/bash -c java -jar consumer.jar
 38345   38281      4    perl -e if(sysread(STDIN,$buf,1)){open($fh[...]} /usr/bin/bash -c java -jar consumer.jar
  ...     ...      ...   ...
 38363   38281     10    perl -e if(sysread(STDIN,$buf,1)){open($fh[...]} /usr/bin/bash -c java -jar consumer.jar
 38383   38281      4    perl /usr/bin/parallel --pipe java -jar consumer.jar
 38384   38344      2    java -jar consumer.jar
 38438   38136     12    ps -o pid,ppid,cpuid,cmd
```

Used for generic splitting magic (e.g., not done with `--roundrobin`*)*

History

- Originally two tools: xxargs and parallel
  - Parallel was originally a wrapper that generated a makefile and used `make -j` to do the parallelization
  - xxargs and parallel got merged into parallel
  - Two objectives:
    - replace xargs
    - run commands in parallel
- In 2010, parallel was adopted as an official GNU tool, named GNU parallel

[2]: *GNU Parallel - The Command-Line Power Tool* Tange, 2011

[4]: *History of GNU Parallel - GNU Project — gnu.org* Tange, 2021

Distributed

- GNU parallel can run jobs on remote servers
  - It uses ssh to communicate with the remote machines

```
parallel -S $SERVER echo running on ::: $SERVER
```

- Transfer Files using rsync:
  - Long version:

```
parallel -S 1/"sshpass␣-p␣'$SECRET_PW'␣↩
    ssh␣limerent@localhost" --transferfile {}\
   --return {}.gz --cleanup gzip ::: README.txt
```

  - Shorthands like `-trc` (transferfile, return, cleanup)

parallel tutorial [06/30/22]

Useful Flags

- Prevent `sshd` overloading:

```
parallel -S $SERVER --sshdelay 0.2 echo ::: 1 2 3
```

- Multiplex connections with `--controlmaster`
- Transfer Files using rsync:
  - `--basefile`, copy this file to each sshlogin
  - `--workdir`, change from login directory
  - `--onall`, run job on all sshlogins

parallel tutorial [06/30/22]

Other Languages

consumer.ts

```
const accMap = new Map<string, number>();
const rl = createInterface({ input: process.stdin });

function handleLine(input: string): void {
  const s = input.split("␣");
  accMap.set(s[0], accMap.get(s[0]) ?? 0 + Number(s[1]));
}

rl.on("line", handleLine);
rl.on("close", () => console.log(accMap));
```

Read from STDIN

Use 0 as initial value

Register callback for 'line' event

Anonymous callback for 'closed' event (arrow style)

12

Integrating TypeScript

```
java -jar producer.jar 1000000\
    | parallel --pipe --roundrobin -j4 java -jar consumer.jar\
    | yarn --silent start
```

--recend "\n"
Distributes records amongst all jobs.
No longer guarantees order.

--jobs 4

Runs `tsc && node consumer.js`

```
    pid       ppid   cpuid    cmd
 258484     258481       3    /bin/bash -c java -jar producer.jar 1000000 | parallel --pipe [...] | yarn --silent start-consumer
 258485     258484       2    java -jar producer.jar 1000000
 258486     258484       9    /usr/bin/perl /usr/bin/parallel --pipe --roundrobin -j4 java -jar consumer.jar
 258487     258484       3    node /home/lord-waddle/.nvm/versions/no:c:de/v16.15.1/bin/yarn -silent start-consumer
 258519     258486      11    java -jar consumer.jar
 258520     258486       3    java -jar consumer.jar
 258522     258486       7    java -jar consumer.jar
 258524     258486      10    java -jar consumer.jar
 258604     258487      11    /bin/sh -c tsc && node consumer.js
 258605     258604       4    /home/lord-waddle/.nvm/versions/node/v16.15.1/bin/node consumer.js
 258618     258480      11    ps -o pid,ppid,cpuid,cmd
```

Much simpler (reuses workers for records)

Compilation (tsc) already completed

Path on Florian's system

Data Encoding

- Pipes can be used to pass any data (encoded as binary)
- For example, we can use JSON to encode objects
- In Java, we can use Gson
  - Serialization with `gson.toJson(obj)`
  - Deserialization with `gson.fromJson(input, ⟨Class⟩)`
  - Common (de-)serialization problems (cf. `MarshalException` with Java RMI)
- Exemplified with Consumer.java

[1]: *Java virtual machine support for object serialization* Breg, 2003

Java RMI [07/01/22]

google/gson [07/01/22]

Recap

- Stream-based communication
  - Cf. Javas functional streams
  - Serialization and deserialization
  - Decoupled programs (e.g., no shared memory)
- Allows distribution individual operators in the pipeline
- Easy combination of different languages
  - Comparable with gRPC
  - But: no message standardization (cf. protocol buffers)
- Programs don’t know anything of the parallelization

gRPC [07/01/22]

protocol buffers [07/01/22]

About Pipelines

- Executes each program in own subshell
- Buffer provided by the kernel
  - Works on bytes (no known boundaries except max-size)
  - Limited capacity (`/proc/sys/fs/pipe-max-size`)
  - By default blocking read and write
  - Can be changed with `O_NONBLOCK` flag (`pipe2`, `fnctl`)
  - This sets `errno` to `EWOULDBLOCK` or `EAGAIN`
- By default unidirectional
  - Named pipes (like `fifo`) allow half duplex data flow

posix/pipe [07/01/22]

unix pipeline [07/01/22]

bash pipelines [07/01/22]

named pipes [07/01/22]

GNU Parallel

- Mixes tabs and spaces for padding
- Supplied as a single file (object-oriented Perl)
  - Runs wherever there is a Perl interpreter
  - Rather slow, 3–10 ms per job and 1 ms/MB output
  - Uses busy wait (with exponential sleeping times)
  - A lot of support for the hosting shell
- Buffers output on disk for distinction
- GNU parallel parses processes everything from stdin
  - E.g., this limits the throughput of `--pipe`

parallel design [07/01/22]

unbuffered output [07/01/22]

It can do more!

- Different spreading strategies (`--shard`, `--bin`, ...) for `--pipe`
- Replacement strings (`{}`, `{%}`, ...)
- Compression of buffer data (`--compress`)
- Comfort-Support for named pipes (`--fifo`)
- Support for unfair counting semaphore with timeout
  - With options (`--semaphore`)
  - As alternative Program (`sem`)
- Load Balancing (`--limit`, `--load`, ...)
- And so much more (Tables, SQL, Shebang, ...)

parallel tutorial [06/30/22]

parallel alternatives [06/30/22]

[1] Fabian Breg and Constantine D. Polychronopoulos. “Java virtual machine support for object serialization”. 2003

[2] Ole Tange. “GNU Parallel - The Command-Line Power Tool”. Feb. 2011

[3] Ole Tange. *GNU Parallel 20210822 (’Kabul’).* 2021

[4] Ole Tange. *History of GNU Parallel - GNU Project — gnu.org.* 2021

Felix R. & Florian S.

Ulm July 3, 2022