GNU PARALLEL

Parallelizing and
Distributing programs with the Shell

Felix Rieg and Florian Sihler

July 3, 2022
CCPDP, Ulm University

2.1 Motivation

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

2.2 Motivation

Doing stuff parallel.

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

2.3 Motivation

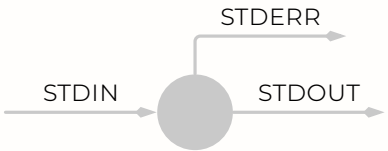
Doing stuff parallel.

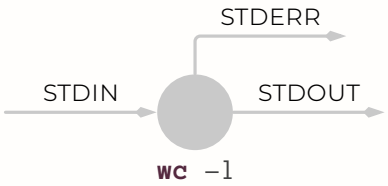
With the commandline!

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


STDIN

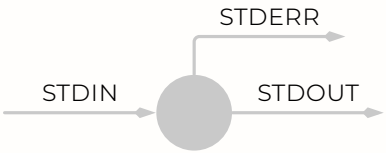

STDIN
STDOUT

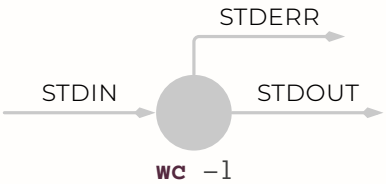

STDERR
STDIN
STDOUT


STDERR
STDIN
STDOUT
wc -l

Pipes

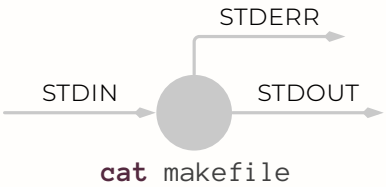
Unix Pipelines

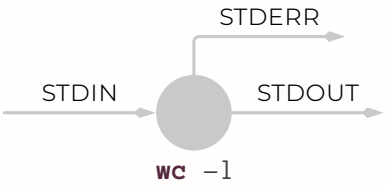




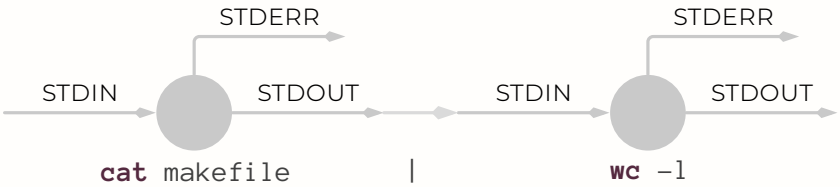
Pipes

Unix Pipelines





Pipes



Pipes

Pipes

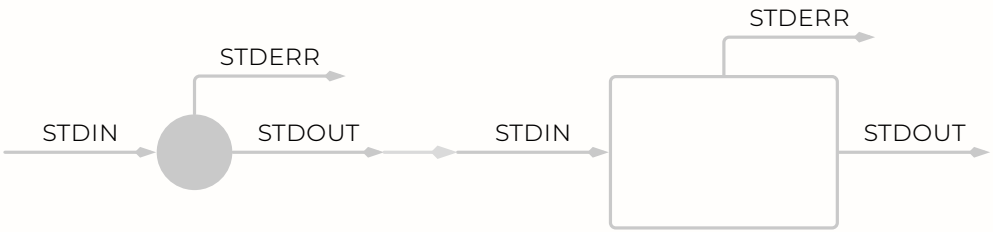
Pipes


STDERR
STDERR
STDIN
STDOUT
STDIN
STDOUT
find . -type f |
./Dockerfile
./run-docker
./makefile
⋮

Pipes


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

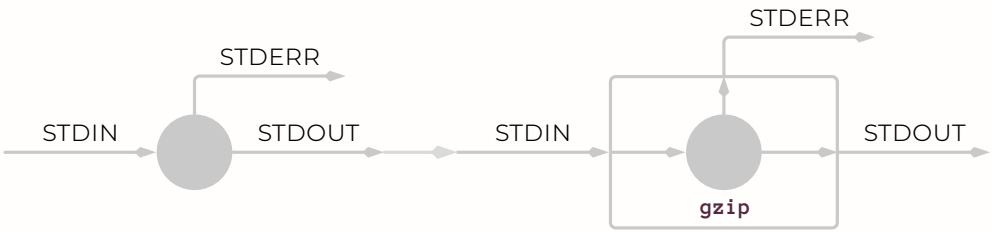
Pipes




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

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

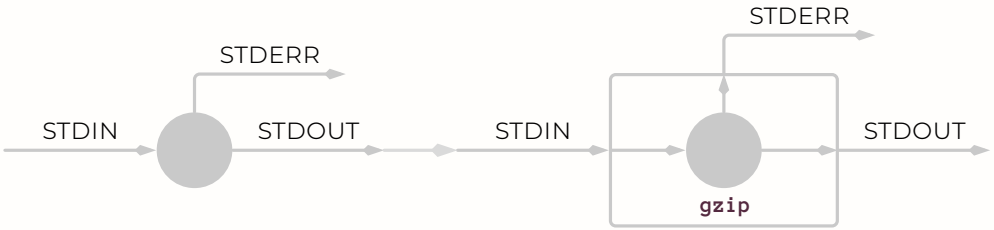
Pipes




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

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

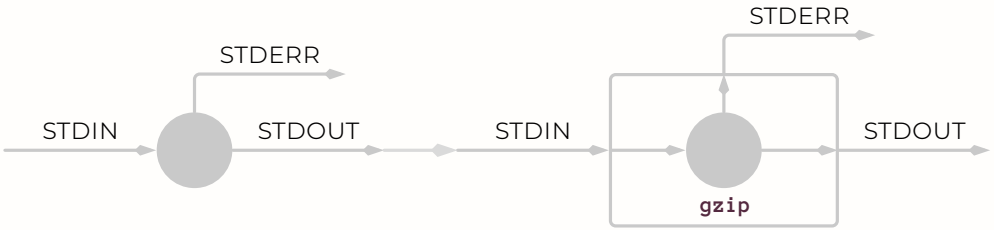
Pipes




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

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

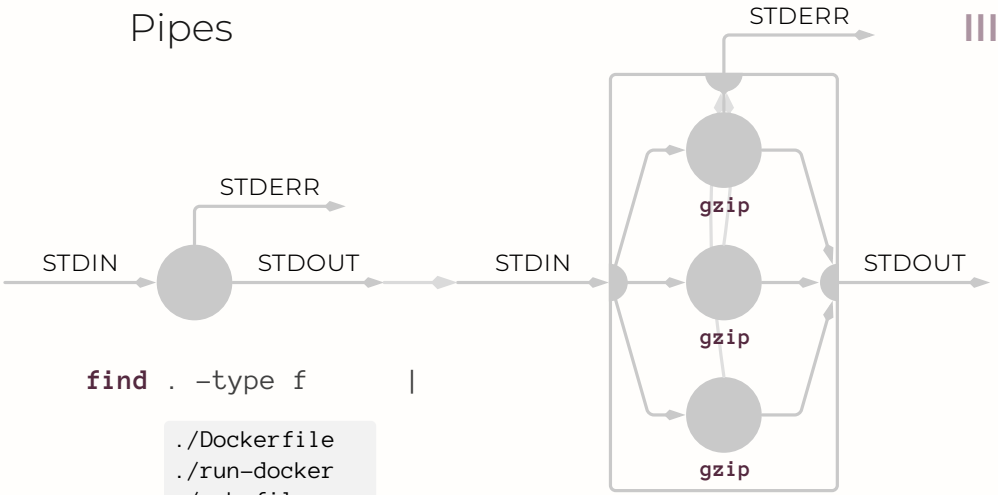
Pipes




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

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

Pipes



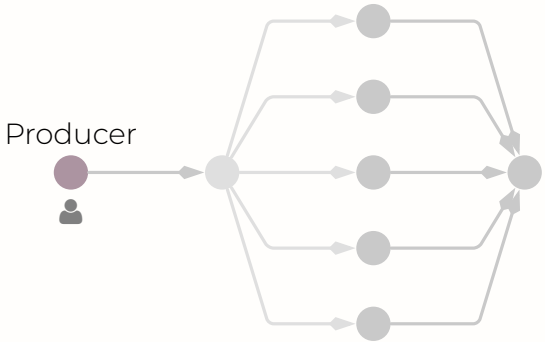

```
find . -type f |
```

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

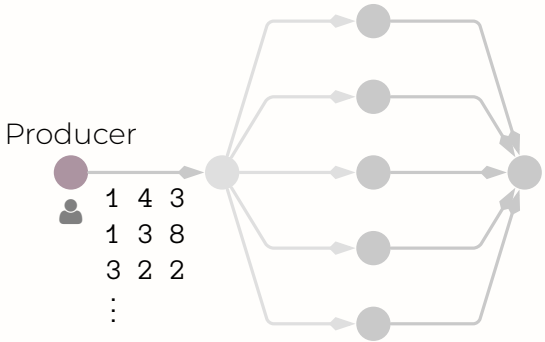
6.1 A simple Bank

6.2 A simple Bank

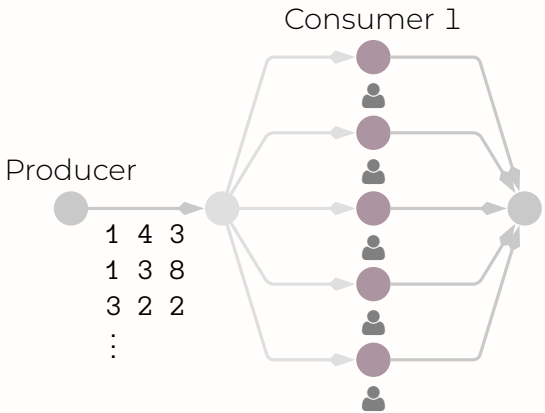
6.3 A simple Bank



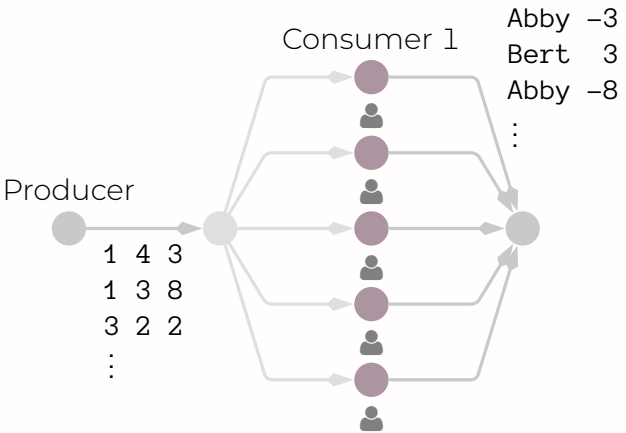
6.4 A simple Bank



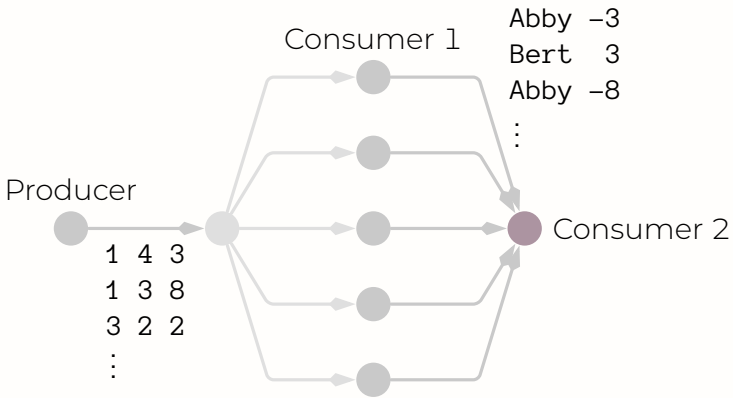
6.5 A simple Bank



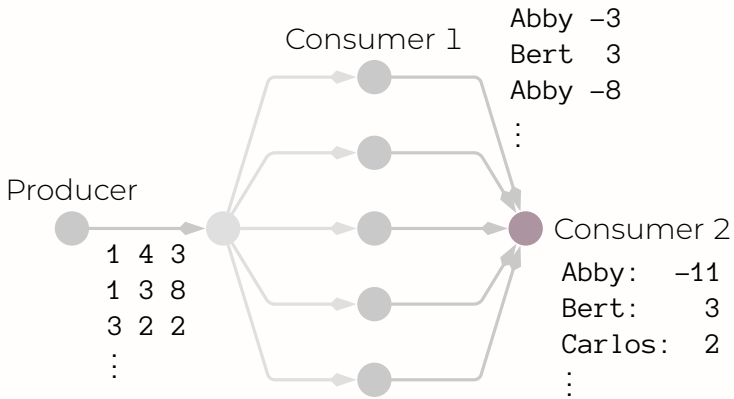
6.6 A simple Bank



6.7 A simple Bank

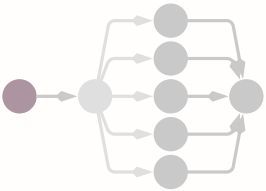


6.8 A simple Bank



6.9 A simple Bank

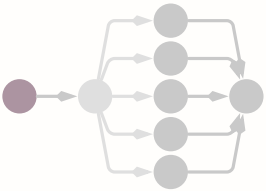
6.10 A simple Bank



Producer.java

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

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);
}
```

6.11 A simple Bank

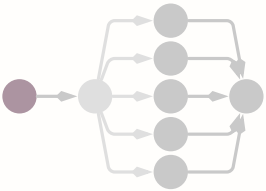


Producer.java

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

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);
}
```

number of transactions

6.12 A simple Bank



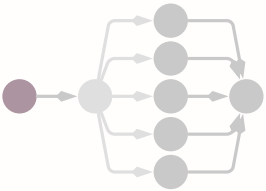
Producer.java

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

number of transactions

known account names

6.13 A simple Bank



Producer.java

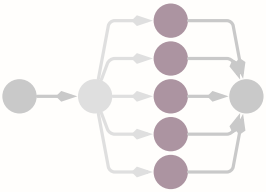
```
final var rand = new Random();

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);
}
```

number of transactions

known account names

value

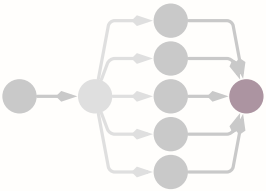
6.14 A simple Bank



Consumer.java

```
final var scanner = new Scanner(System.in);
while (scanner.hasNextLine()) {
   String[] s = scanner.nextLine().split("␣");

   String from = NAMES[Integer.parseInt(s[FROM])];
   System.out.printf("%s␣-%d%n", from, Integer.parseInt(s[VALUE]));

   String to = NAMES[Integer.parseInt(s[TO])];
   System.out.printf("%s␣%d%n", to, Integer.parseInt(s[VALUE]));
}
scanner.close();
```

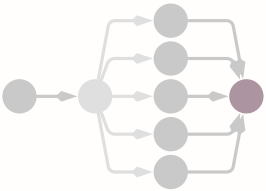
6.15 A simple Bank



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);
```

6.16 A simple Bank

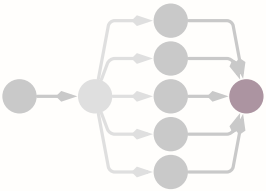


Accountant.java

```
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

6.17 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()

7.1 Running the Example

7.2 Running the Example

7.3 Running the Example

```
java -jar producer.jar 1000000
```

7.4 Running the Example

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

7.5 Running the Example

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

7.6 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

7.7 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

7.8 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

7.9 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

7.10 Running the Example


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

7.11 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...
...	...	...	...

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

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

9.1 Distributed

parallel tutorial [06/30/22]

9.2 Distributed

- GNU parallel can run jobs on remote servers

parallel tutorial [06/30/22]

Distributed

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

parallel tutorial [06/30/22]

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
```

parallel tutorial [06/30/22]

9.5 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:

parallel tutorial [06/30/22]

9.6 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
```

parallel tutorial [06/30/22]

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]

11.1 Other Languages

11.2 Other Languages

consumer.ts

```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));
```

11.3 Other Languages

consumer.ts

```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

Chapter E-30–32

11.4 Other Languages

consumer.ts

```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

Chapter E-30–32

11.5 Other Languages

consumer.ts

```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

Chapter E-30–32

11.6 Other Languages

consumer.ts

```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)

Chapter E-30–32

12.1 Integrating TypeScript

12.2 Integrating TypeScript

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

12.3 Integrating TypeScript

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

--jobs 4

12.4 Integrating TypeScript

Distributes records amongst all jobs.
No longer guarantees order.

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

--jobs 4

12.5 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`

12.6 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`

12.7 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

12.8 Integrating TypeScript

--recend "\n"

Distributes records amongst all jobs.
No longer guarantees order.

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

--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

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

Path on Florian's system

14.1 Recap

gRPC [07/01/22]

protocol buffers [07/01/22]

14.2 Recap

- Stream-based communication

gRPC [07/01/22]

protocol buffers [07/01/22]

14.3 Recap

- Stream-based communication
  - Cf. Javas functional streams

gRPC [07/01/22]

protocol buffers [07/01/22]

14.4 Recap

- Stream-based communication
  - Cf. Javas functional streams
  - Serialization and deserialization

gRPC [07/01/22]

protocol buffers [07/01/22]

Recap

- Stream-based communication
  - Cf. Javas functional streams
  - Serialization and deserialization
  - Decoupled programs (e.g., no shared memory)

gRPC [07/01/22]

protocol buffers [07/01/22]

14.6 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

gRPC [07/01/22]

protocol buffers [07/01/22]

14.7 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

gRPC [07/01/22]

protocol buffers [07/01/22]

14.8 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

gRPC [07/01/22]

protocol buffers [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)

gRPC [07/01/22]

protocol buffers [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

posix/pipe [07/01/22]

unix pipeline [07/01/22]

bash pipelines [07/01/22]

named pipes [07/01/22]

About Pipelines

- Executes each program in own subshell

posix/pipe [07/01/22]

unix pipeline [07/01/22]

bash pipelines [07/01/22]

named pipes [07/01/22]

About Pipelines

- Executes each program in own subshell
- Buffer provided by the kernel

posix/pipe [07/01/22]

unix pipeline [07/01/22]

bash pipelines [07/01/22]

named pipes [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)

posix/pipe [07/01/22]

unix pipeline [07/01/22]

bash pipelines [07/01/22]

named pipes [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`)

posix/pipe [07/01/22]

unix pipeline [07/01/22]

bash pipelines [07/01/22]

named pipes [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

posix/pipe [07/01/22]

unix pipeline [07/01/22]

bash pipelines [07/01/22]

named pipes [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`)

posix/pipe [07/01/22]

unix pipeline [07/01/22]

bash pipelines [07/01/22]

named pipes [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`

posix/pipe [07/01/22]

unix pipeline [07/01/22]

bash pipelines [07/01/22]

named pipes [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

posix/pipe [07/01/22]

unix pipeline [07/01/22]

bash pipelines [07/01/22]

named pipes [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]

16.1 GNU Parallel

parallel_design [07/01/22]

unbuffered output [07/01/22]

16.2 GNU Parallel

- Mixes tabs and spaces for padding 

parallel_design [07/01/22]

unbuffered output [07/01/22]

16.3 GNU Parallel

- Mixes tabs and spaces for padding 
- Supplied as a single file (object-oriented Perl)

parallel_design [07/01/22]

unbuffered output [07/01/22]

16.4 GNU Parallel

- Mixes tabs and spaces for padding 
- Supplied as a single file (object-oriented Perl)
  - Runs wherever there is a Perl interpreter

parallel_design [07/01/22]

unbuffered output [07/01/22]

16.5 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

parallel_design [07/01/22]

unbuffered output [07/01/22]

16.6 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)

parallel_design [07/01/22]

unbuffered output [07/01/22]

16.7 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

parallel_design [07/01/22]

unbuffered output [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

parallel_design [07/01/22]

unbuffered output [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

parallel_design [07/01/22]

unbuffered output [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]

17.1 It can do more!

parallel tutorial [06/30/22]

parallel alternatives [06/30/22]

17.2 It can do more!

- Different spreading strategies (`--shard`, `--bin`, …) for `--pipe`

parallel tutorial [06/30/22]

parallel alternatives [06/30/22]

17.3 It can do more!

- Different spreading strategies (`--shard`, `--bin`, …) for `--pipe`
- Replacement strings (`{}`, `{%}`, …)

parallel tutorial [06/30/22]

parallel alternatives [06/30/22]

17.4 It can do more!

- Different spreading strategies (`--shard`, `--bin`, …) for `--pipe`
- Replacement strings (`{}`, `{%}`, …)
- Compression of buffer data (`--compress`)

parallel tutorial [06/30/22]

parallel alternatives [06/30/22]

17.5 It can do more!

- Different spreading strategies (`--shard`, `--bin`, …) for `--pipe`
- Replacement strings (`{}`, `{%}`, …)
- Compression of buffer data (`--compress`)
- Comfort-Support for named pipes (`--fifo`)

parallel tutorial [06/30/22]

parallel alternatives [06/30/22]

17.6 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

parallel tutorial [06/30/22]

parallel alternatives [06/30/22]

17.7 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`)

parallel tutorial [06/30/22]

parallel alternatives [06/30/22]

17.8 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`)

parallel tutorial [06/30/22]

parallel alternatives [06/30/22]

17.9 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`, …)

parallel tutorial [06/30/22]

parallel alternatives [06/30/22]

17.10 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] Ole Tange. *GNU Parallel 20210822 ('Kabul').* 2021

Felix R. & Florian S.

Ulm July 3, 2022

Slides and LaTeX-sources on GitHub!