Functional programming interface for parallel and distributed computing

I. Petriakov I. Gankevich

Saint Petersburg State University

September 2021

Motivation

- There is no universal low-level representation of distributed computations.
- There is no high-level interface for distributed computing in functional languages.
- Existing solutions do not provide automatic fault tolerance for both slave and master nodes.

Parallel — several processor cores of single cluster node.
Distributed — several cluster nodes.

From sync. call stack to async. call stack (kernels)

Kernel = data + code + result of the computation.

```
int nested(int a) {
  return 123 + a;
}
```

```
struct Nested: public Kernel {
  int result;
  int a;
  Nested(int a): a(a) {}
  void act() override {
    result = a + 123;
    async_return();
  }
};
```

async_call	push child kernel to the queue
async_return	push current kernel to the queue
async_message	send a kernel to another one via the queue

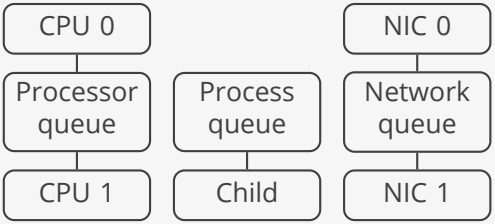
From sync. call stack to async. call stack (kernels)

```
void main() {
  // code before
  int result = nested();
  // code after
  print(result);
}
```

```
struct Main: public Kernel {
  void act() override {
    // code before
    async_call(new Nested);
  }
  void react(Kernel* child) override {
    int result = ((Nested*)child)->result;
    // code after
    print(result);
    async_return();
  }
};

void main() {
  async_call(new Main);
  wait();
}
```

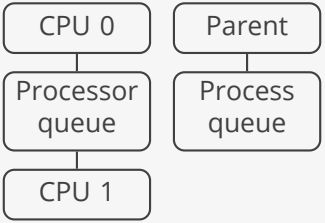
Cluster scheduler architecture

Daemon process:



- Run applications in child processes.
- Route kernels between application processes running on different cluster nodes.
- Maintain a list of available cluster nodes.

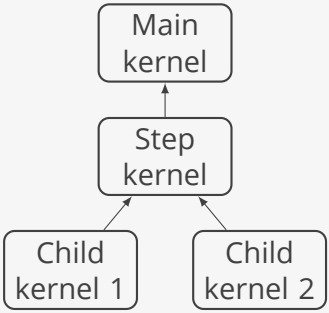
Application process:



`async_call`	push child kernel to the queue
`async_return`	push current kernel to the queue
`async_message`	send a kernel to another one via the queue

Fault tolerance

- Assumption: *main* kernel has only one child kernel at a time.
- Every *step* kernel (a child of *main*) has a copy of the *main*.
- Scheduler ensures that *main* and *step* are on different cluster nodes.
- Every *step* is also appended to the local log file.



Failure	Resolution
Child 1	resend Child 1 to the remaining nodes
Child 2	resend Child 2 to the remaining nodes
Step	resend Step to the remaining nodes
Main	restore Main from the copy
Main and Step	restore Main and Step from the log

- I. Gankevich, Yu. Tipikin, V. Korkhov Subordination: Providing resilience to simultaneous failure of multiple cluster nodes, HPCS'17, 2017.
- I. Gankevich, Yu. Tipikin, V. Korkhov, V. Gaiduchok, A. Degtyarev, A. Bogdanov Master node fault tolerance in distributed big data processing clusters, International Journal of Business Intelligence and Data Mining, 2017.

Kernel and queue definition

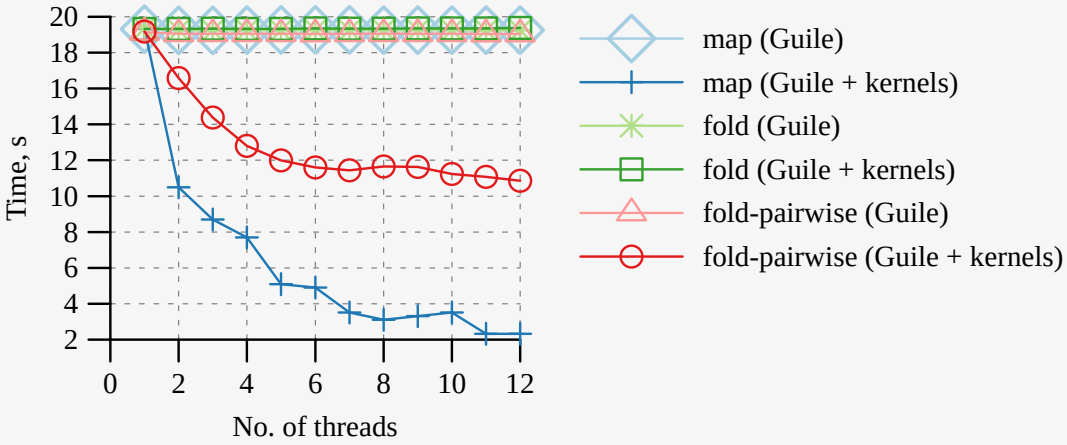
```
enum class states {upstream, downstream, point_to_point};

class kernel {
public:
  virtual void act();
  virtual void react(kernel* child);
  virtual void write(buffer& out) const;
  virtual void read(buffer& in);
  kernel* parent = nullptr;
  kernel* target = nullptr;
  states state = states::upstream;
};

class queue {
public:
  void push(kernel* k);
};
```

Automatic parallelism

The idea: evaluate arguments in parallel (one kernel for each argument).

```
(define (map proc lst) "Parallel map."
  (if (null? lst) lst
    (cons (proc (car lst)) (map proc (cdr lst)))))
(define (fold proc init lst) "Sequential fold."
  (if (null? lst) init
    (fold proc (proc (car lst) init) (cdr lst))))
(define (do-fold-pairwise proc lst)
  (if (null? lst) '()
    (if (null? (cdr lst)) lst
      (do-fold-pairwise proc
        (cons (proc (car lst) (car (cdr lst)))
              (do-fold-pairwise proc (cdr (cdr lst))))))))
(define (fold-pairwise proc lst) "Parallel pairwise fold."
  (car (do-fold-pairwise proc lst)))
```

Guile with automatic parallelism (synthetic benchmark)



Conclusion and future work

Kernels provide

- standard way of expressing parallel and distributed programme parts,
- automatic fault tolerance for master and worker nodes and
- automatic load balancing via cluster scheduler.

Arguments-based parallelism provide

- high-level programming interface for clusters and single nodes,
- conveniently hides the shortcomings of parallel and distributed computations.