SKANDIUM

JAVA BASED SKELETON LIB

LITHIUM → CALCIUM → ProActive

Tobi — M. LEYTON — INRIA – UNIV. NICE

$\Delta ::= seq(f) \mid farm(\Delta) \mid pipe(\Delta_1, \Delta_2) \mid while(f_c, \Delta) \mid$ (pipe: $X \to Y$, $Y \to Z$)

$if(f_c, \Delta_1, \Delta_2) \mid for(i, \Delta) \mid map(f_s, \Delta, f_m) \mid$ (Δ: I)

$fork(f_s, \{\Delta_i\}, f_m) \mid d\&c(f_c, f_s, \Delta, f_m)$ (Δ: ↑)

(f_s: I, f_m: I)

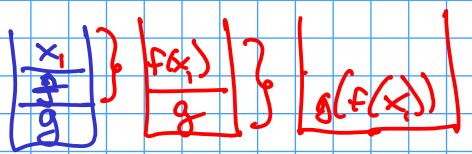
then — else

$pipe(\Delta, \Delta) = pipe(seq(f), farm(seq(g)))$

f_x = muscles

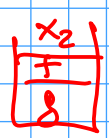
SkelExpr → ASM instruction that operate on stacks

$pipe(seq(f), seq(g))$



x_1

x_2



muscles → f : worker stage

↳ Execute<Tin, Tout>

Tin = input type, Tout = output type

→ Tout execute(Tin param) {

}

class StageIncr implements Execute<Integer, Integer> {

public Integer execute(Integer x) {

return (x + 1);

}

}

class StageDecr ...

Skeleton<Integer, Integer> pipe = new Pipe<Integer, Integer>(

new (StageIncr()),

new (StageDecr));

// associate an "interpreter"

5 → (parallelism degree)

Skandium interp = new Skandium(parallelism);

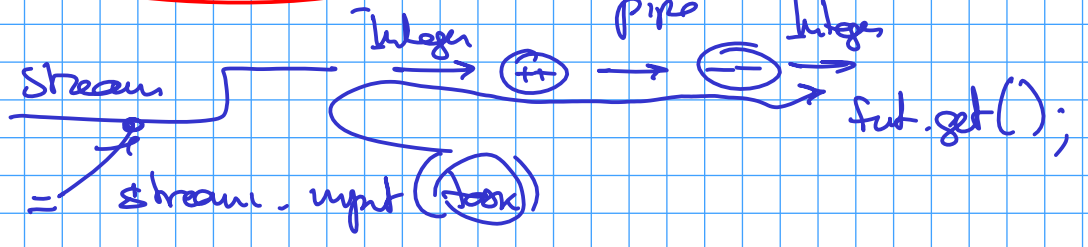
// associate a stream with the skeleton

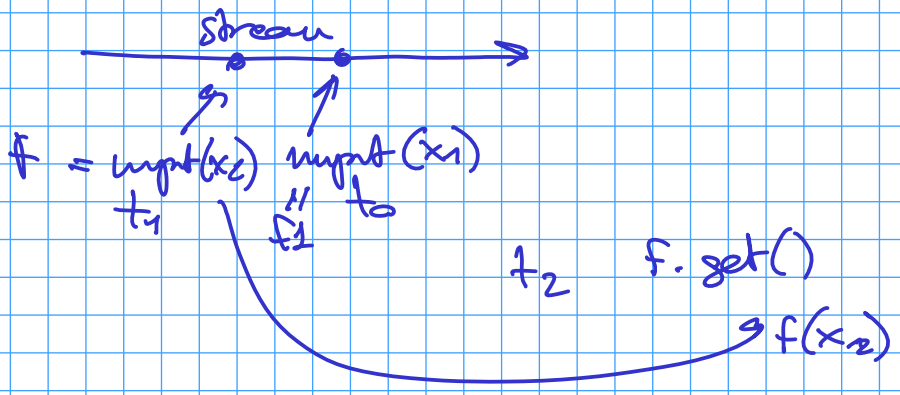
Stream<Integer, Integer> stream =

interp.newStream(pipe);

Future<Integer> fut = stream.input(task1);

Integer result = fut.get()





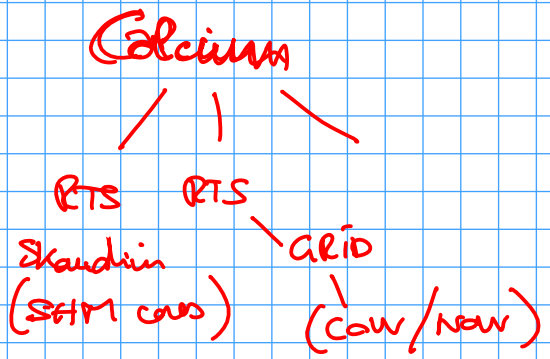
ordering

Hp1

f1.get(); f2.get();

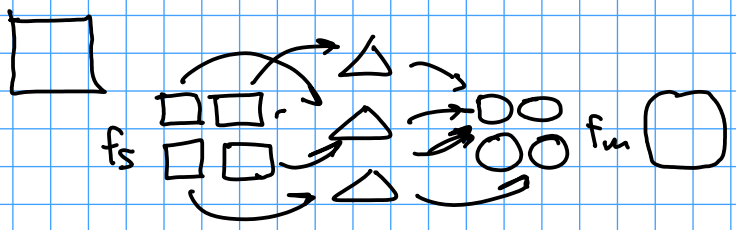
f2.get(); f1.get();

blocking calls!

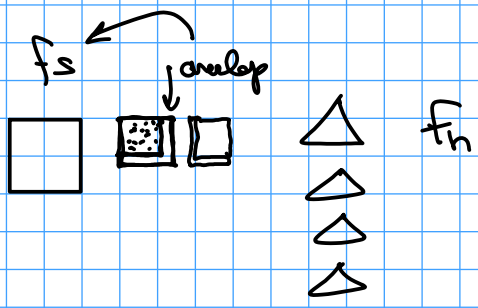


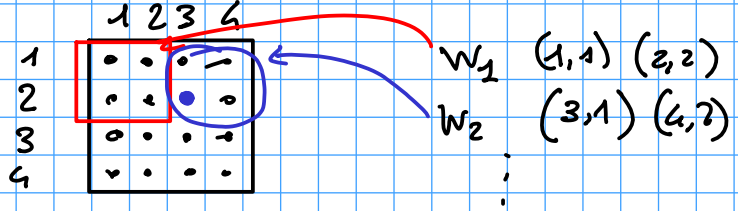
map(f_s, , f_m)

user user



stencil





SKEPU

targets GPUs → CUDA / OPENCL

data parallel skeletons only

CONTAINERS (vectors & arrays)

DATA PARALLEL OPERATIONS on containers

PGM

CPU GPU

Linkoping
↳ Christoph Kessler

16 items

```
skepu::Vector<T>
skepu::Matrix<T>
skepu::Vector<int> v(16, 0)
```

initialized to 0

```
#define SKEPU-CUDA
        SKEPU-OMP
        SKEPU-OPENCL
```

$x_1 \cdots x_n$

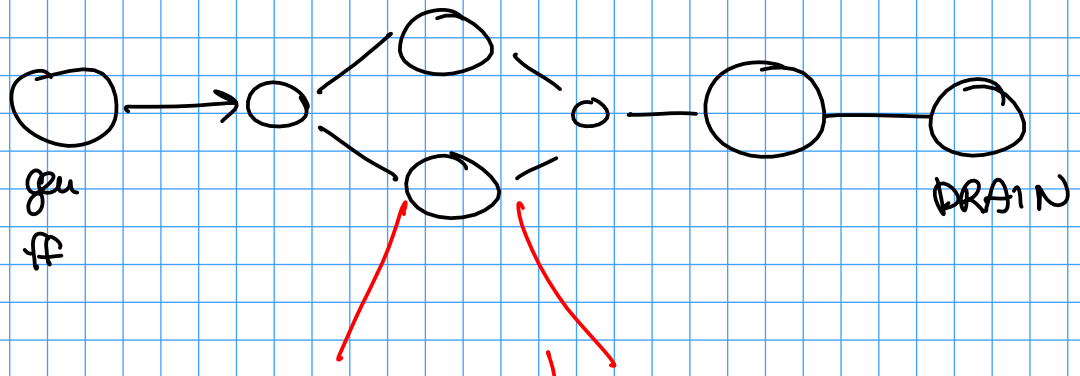
$y_1 \cdots y_n$

$y_i = f(x_i)$

```
int inc(int x) {
  return(x+1);
}
```

```
skepu::Vector<float> x(N, 0.0);
skepu::Vector<float> y(N, 0.0);
UNARY_FUNC(inc, int, x, return(x+1);)
skepu::Map<inc> mymap(new inc);
mymap(x, y);
BINARY_FUNC(add, int, a, b, return(a+b);)
skepu::Reduce<add> myreduce(new add);
int sum = myreduce(y);
OVERLAP_FUNC(avg, float, 1, 2, return((a[-1]+a[0]+a[1])/3);)
skepu::MapOverlap<avg> mystencil(new avg);
.....
```

(name, type, parameter name, body)



ff_node {
— svc(—) {
svcpu::vector
svcpu::map
}

#define SKEPU_CUDA