C++ GrPPI — HIGH LEVEL INTERFACE for PATTERNS

THR | OMP | TBB | THRUST | FF

C++ OpenMP Intel TBB Nvidia research framework } PISA

11-14-17

FastFlow

High level patterns

Core patterns

Communication primitives

→ Hw / Threads

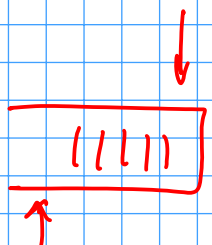
STHREN + threads

Header only library

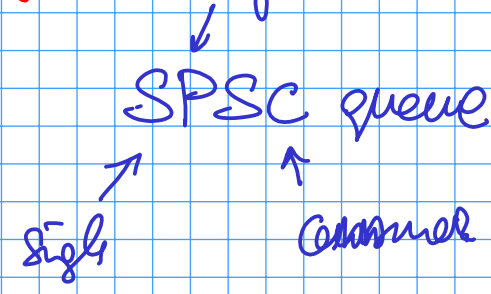
-O3

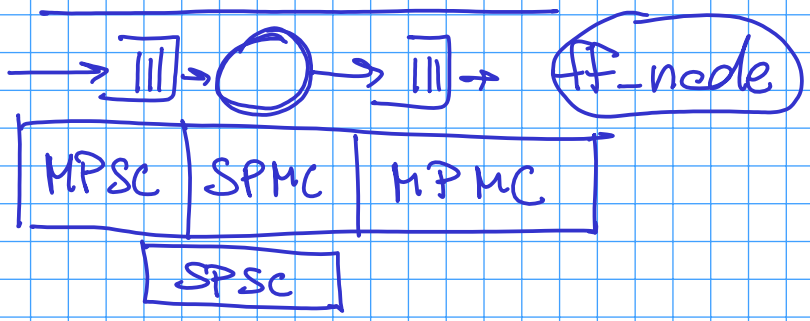
COMMUNICATION PRIMITIVES

- shared memory architectures
- efficient for fine grain
- lock free
- queues only carry pointers
- NULL is not an allowed message



"the" building block here is SPSC queue

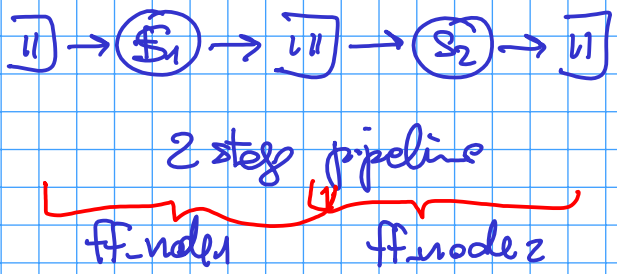




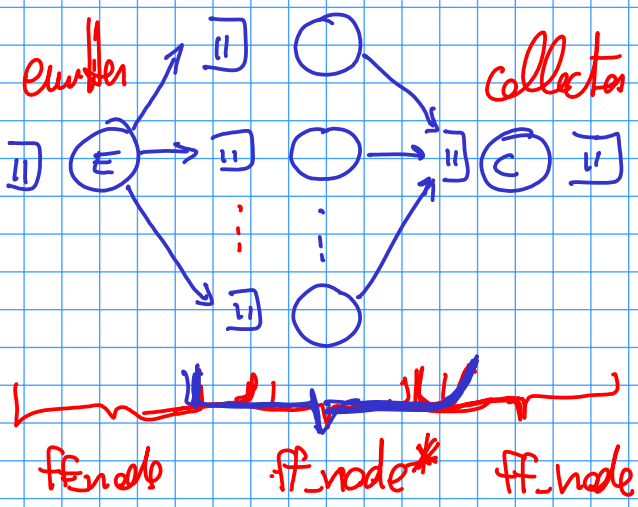
messages as "capabilities" (or pointers)

CORE PATTERNS

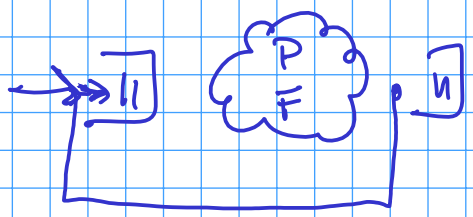
PIPELINES



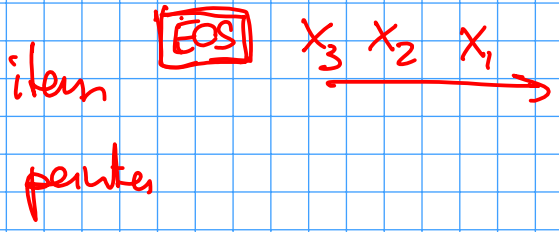
FARM



FEEDBACK



STREAMS



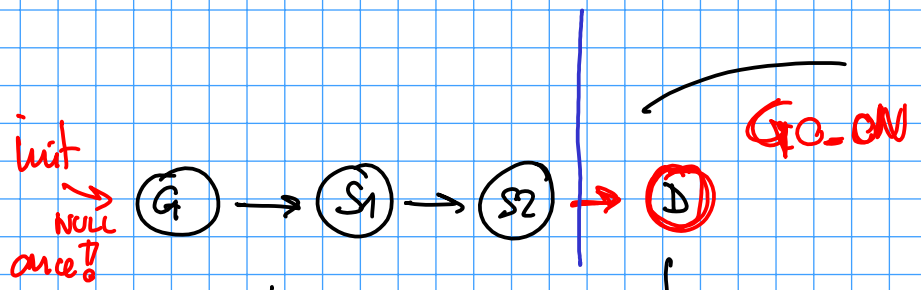
ff_node

```
ff_node {

    void * svc(void *t) = 0;

}
```

```
class S1 : public ff_node {
    void * svc(void *t) {
        int *task = ((int *) t);
        (*task)++;
        return(t);
    }
};
```

S2

```
(*task)*2;
cout << (*task) <<
return(GO_ON);
```




```
class Gen : public ff_node {
    void * svc(void *) {
        for(int i=0; i<10; i++) {
            int * task = new int(i);
            ff_send_out((void *) task);
        }
        ff_send_out(EOS);
        return
    }
};
class Drain : public ff_node {
    void * svc(void * task) {
        // process task ...
        return (GO_ON);
    }
};
```

main ( ) {

ff_Pipe P(S1 S2 S3)

ff_pipeline P;

P. add_stage (Gen);
P. add_stage (S1);
P. add_stage (S2);

builds the program but does not run it (yet)

P. run_and_wait_end ();

---

ff_farm<> farm;

-> Farm. add_emitter ( )
Farm. add_workers ( )
-> Farm. add_collector ( )
NULL

void* SVC (void * t) {
return (t);
}

std::vector<ff_node *>

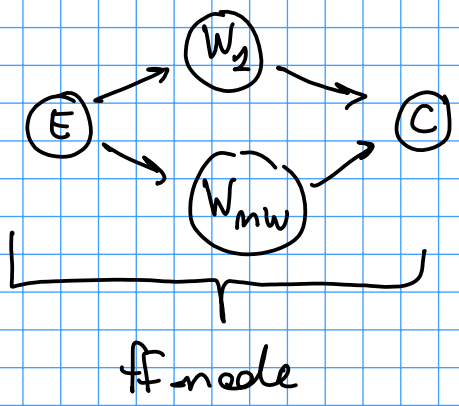
Farm. run_and_wait_end()

E SVC S1 S2 : ff send out

Farm
void set_scheduling_ondemand(const int inbufferentries=1)

auto scheduling mode

5 3 W1
E
42 W2



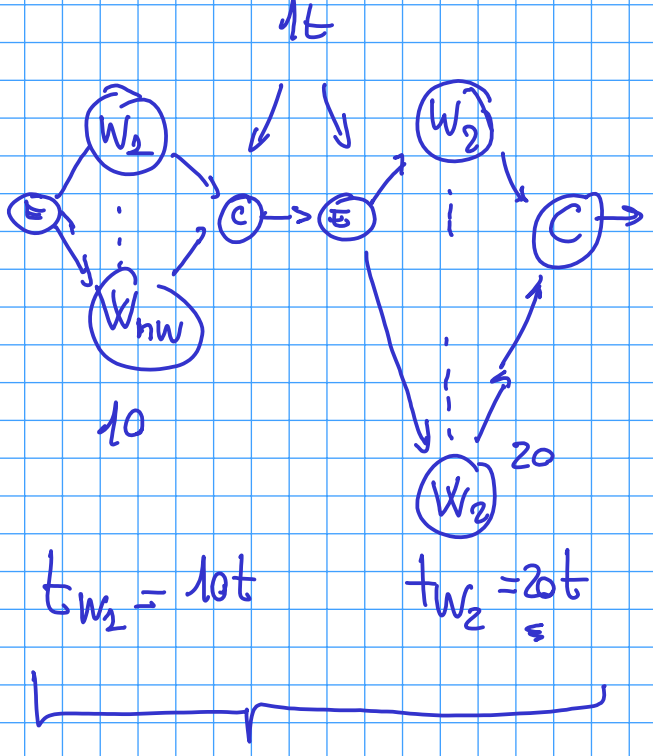
W1 svc
f()

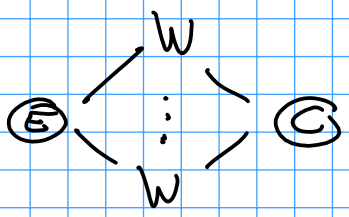
W2 svc
g()

```
ff_pipeline P;

ff_farm<> F1

std::vector<ff_node*> workers1;
for(     nw     )
    workers1.push_back(new W2());
F1.add_workers(workers1);

ff_farm<> F2
```

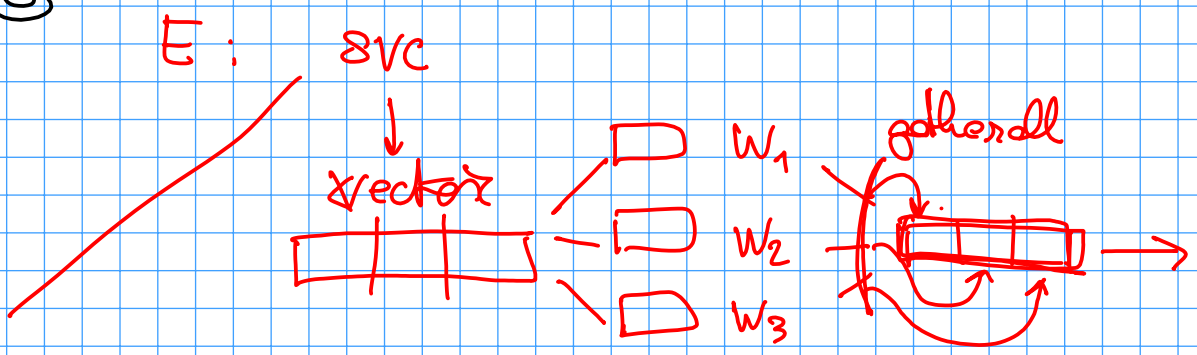
new W2

```
P.add_stage(F1)
P.add_stage(F2)

P.run_and_wait_end()
```





E: svc ≡ id ⇒ FARM



} known nw

splits v in nw pieces (Vi)

for (i < nw)

ff_send_out (Vi)

return (GO_ON);

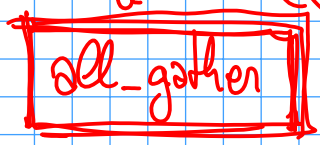
C: svc

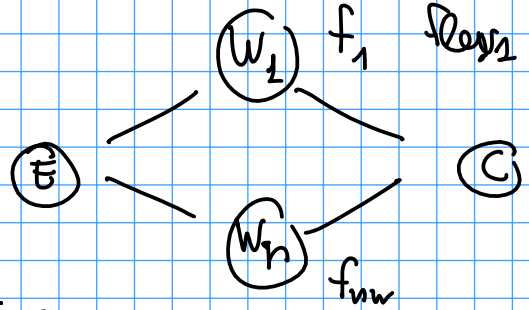
svc ← task

from Wi

vector of nw pointers

gatherall (task, vector)



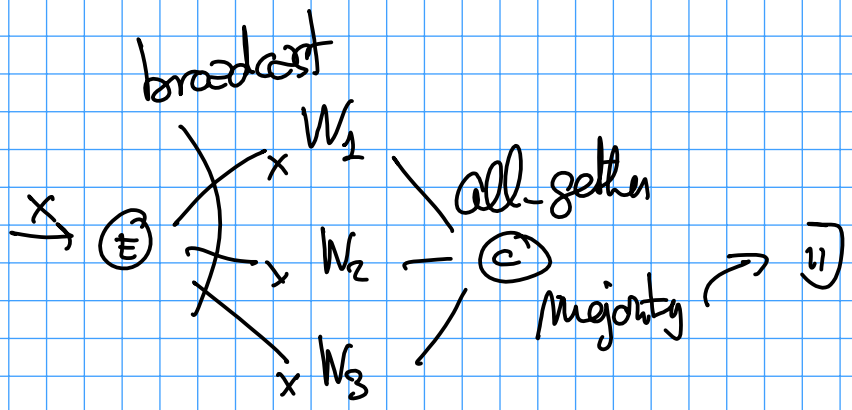


$f_i \neq f_j$

$f_i = f_j : S \times I \to O$

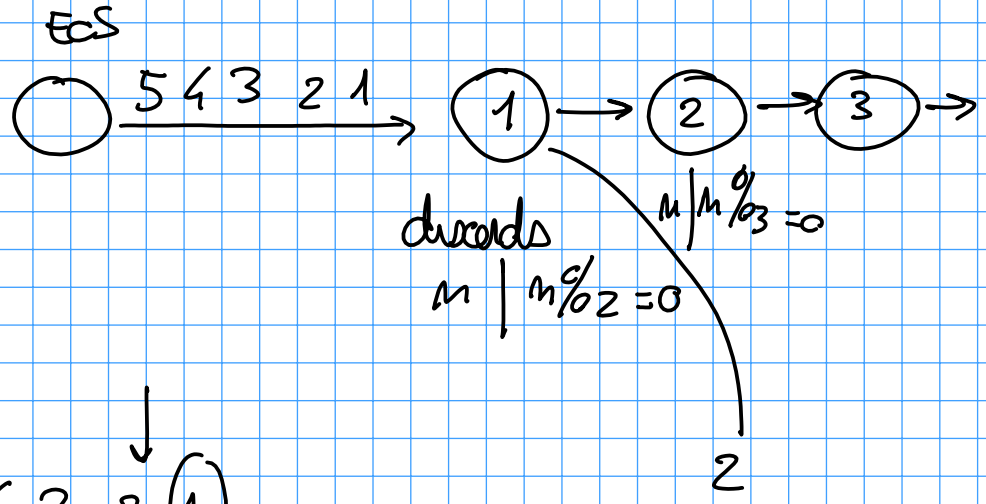
flow

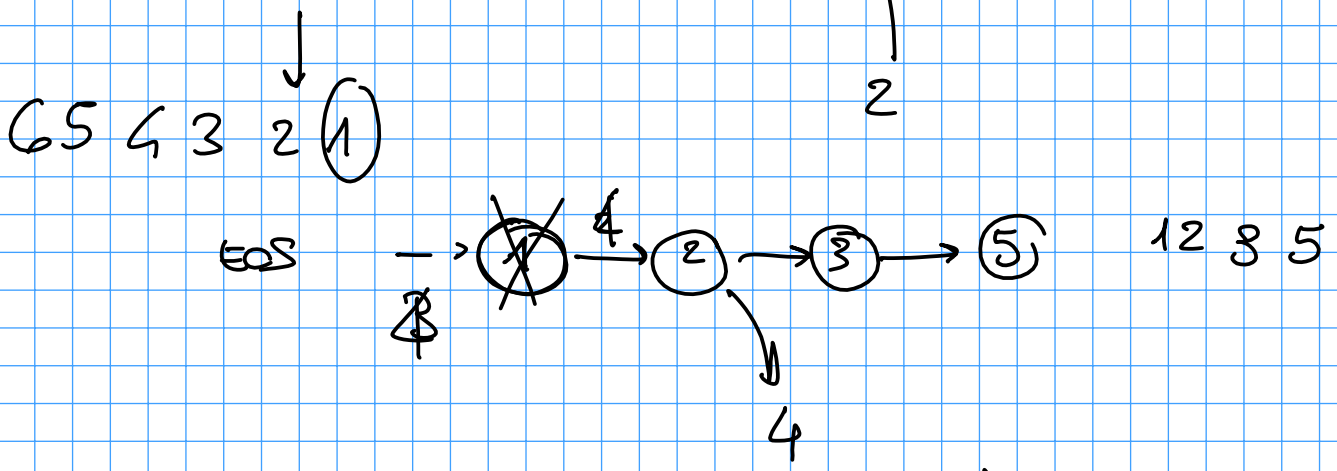
ff_send_out_to (void* pointer , #W)



W_i all compute (F)

W_i compute f_i pgr_i





class Stage : public ff_node {

Stage (int n)

svc ( )

if (task % n == 0)

≠ |

return ( )

n

COLON

svc_init

svc_end