Applicazioni
in
Orca e Linda

Riferimenti

- Niholas Carriero, David Gelertner "How to Write Parallel Programs: A guide to the Perplex" from ACM Computing Surveys, vol. 21, No. 3, Sept. 1989, pp. 323-357, Copyright 1989, Association for Computing Machinery, Inc.
- ORCA: A LANGUAGE FOR PARALLEL

PROGRAMMING OF DISTRIBUTED SYSTEMS

Henri E. Bal M. Frans Kaashoek Andrew S. Tanenbaum Dept. of Mathematics and Computer Science Vrije Universiteit Amsterdam, The Netherlands

Programma per il calcolo dei numeri primi

```
Imain()
{
  int i, ok;
  for(i = 2; i < LIMIT; ++i) {
     eval("primms", i, is_prime(i));
  }
  for(i = 2; i <= LIMIT; ++i) {
     rd("primms", i, ? ok);
     if (ok) printf("%d\n", i);
  }
}
```

Finzione **`is-prime(...)`**

```
is-prime(me)
  int me;
{
  int=i, limit, ok;
  double sqrt();
  limit= sqrt((double) me) + 1;
  for (i=2; i < limit; ++i) {
      rd("primes", i, ? ok);
      if (ok && (me%i == 0)) return 0;
  }
  return 1;
}
```

Calcolo parallelo

```
#include "linda.h"
#define GRAIN 2000
#define LIMIT 1000000
#define NUM_INIT_PRIME 15
long primes[LIMIT/10+1] =
(2,3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47); long
    p2[LIMIT/10+1) =
{4,9,25,49,121,169,289,361,529,841,961,1369,
    1681,1849,2209};
lmain(argc, argo)
    int argc;
    char *argo[];
```

Calcolo parallelo

```
{
   int eot, first_num, i, nun, num_primes, num_workers;
   long new_primes[GRAIN], np2;
   num_workers = atoi(argv[1]);
   for (i = 0;   i < num_workers; ++i)
         eval(“worker", worker());
   num_primes = NUM_INIT_PRIME;
   first nun = primes[num_primes-1] + 2;
   out("next task", first nun);
   eot = 0; /* becomes 1 at "end of table" -- i.e., table
   complete*/
```

Calcolo parallelo

```
for (num = firstnum; num < LIMIT; num += GRAIN) {
        in("result", num, ? new_primes: size);
        for (i =0; i < size; ++i, ++num_primes) {
                primes[num_primes] = new_primes[i];
                if(!eot)
                        np2 = new_primes[i]*new_primes[i];
                        if (np2 > LIMIT) {
                                eot = 1;
                                np2 = -1;
                        }
                out("primes", num_primes, new_primes[i], np2);
        }
}
}
```

Calcolo parallelo

```
/* " ? int" means "match any int; throw out
   the value"*/
    for (i = 0; i < nun_workers; ++i)
        in("workers",?int)
   printf("'%d: %d\n", num_primes,
   primes[num_primes-1]);
    }
}
```

Calcolo parallelo

```
worker() {
   long count, eot, i, limit, num, num_primes, ok,
   start;
   long my_primes[GRAIN];
   num_primes = NUM_INIT_PRIME;
   eot = 0;
   while(1) {
      in("next task", ? num);
      if (num == -1) {out("next task", -1);return;}
```

Calcolo parallelo

```
limit = num + GRAIN;
out("next task", (limit > LIMIT) ? -1 : limit);
if (limit > LIMIT) limit = LIMIT;
start =num;
for (count = 0; num < limit; num +- 2) {
    while (!eot && nun > p2[num-primes-1]) {
        rd("primes", num primes, ? primes [nun-primes], ?
p2[num-primes]);
        if (p2[num-primes] < 0)
            eot = 1;
        else
            ++num_primes;
}
```

Calcolo parallelo

```
    for (i = 1, ok = 1; i < num_primes; ++i)
            if (!(num%primes[i])) {
                    ok = 0;
                    break;
            }
            if (num < p2[i]) bleak;
            }
    if (ok) {
            my_primes[count]=num;
            ++count;
    }
}
    /• Send the control process any primes found.
    out("result', start, my-primes: count);
}/*While(1)*/
} /*classe*/
```

Orca

generic (**type** T)

object specification GenericJobQueue;

operation AddJob(job: T); # add a job to the tail of the queue

operation NoMoreJobs(); # invoked when no more jobs will be added

operation GetJob(job: **out** T): boolean;

Fetch a job from the head of the queue. This operation

fails if the queue is empty and NoMoreJobs has been invoked.

end generic;

Coda di Job

```
generic
object implementation GenericJobQueue;
    type ItemName = nodename of queue;
    type queue =
        graph # a queue is represented as a linear list
            first, last: ItemName; # first/last element of queue
        nodes
            next: ItemName; # next element in queue
            data: T; # data contained by this element
        end;
    done: boolean; # set to true if NoMoreJobs has been invoked.
    Q: queue; # the queue itself

begin # Initialization code for JobQueues ; executed on object creation.
    done := false; # initialize done to false
end generic;
```

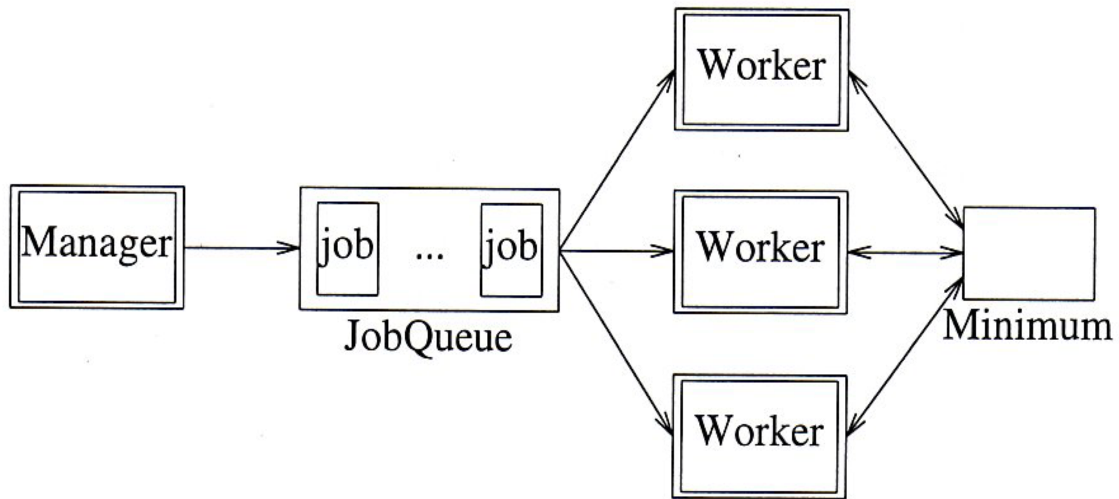
Coda di Job

```
operation AddJob(job: T);
        p: ItemName;
   begin # add a job to the tail of the queue
        p := addnode(Q); # add a new node to Q, return its name
   in p
        Q[p].data := job; # fill in data field of the new node; next
   field is NIL
        if Q.first = NIL then # Is it the first node?
                Q.first := p; # yes; assign it to global data field
        else
                Q[Q.last].next := p; # no; set predecessor's next field
        fi;
        Q.last := p; # Assign to "last" global data field
   end;
```

Coda di Job

```
operation NoMoreJobs();
    begin # Invoked to indicate that no more jobs will be added
        done := true;
    end;
    operation GetJob(job: out T): boolean;
        p: ItemName;
    begin # Try to fetch a job from the queue
        guard Q.first /= NIL do # A job is available
            p := Q.first; # Remove it from the queue
            Q.first := Q[p].next;
            if Q.first = NIL then Q.last := NIL; fi;
            job := Q[p].data; # assign to output parameter
            deletenode(Q,p); # delete the node from the queue
            return true; # succeeded in fetching a job
        od;
        guard done and (Q.first = NIL) do
            return false; # All jobs have been done
        od;
    end;
```

TSP



TSP Orca

type PathType = **array**[integer] **of** integer;
type JobType =
record
len: integer; # length of partial route
path: PathType;# the partial route itself
end;
type DistTab = ...; # distances table
object TspQueue = **new**
GenericJobQueue(JobType);
# Instantiation of the GenericJobQueue type

TSP Orca

```
process master();
minimum: IntObject; # length of current best path (shared
    object)
q: TspQueue; # the job queue (shared object)
i: integer;
distance: DistTab; # table with distances between cities
begin
    minimum$assign(MAX(integer)); # initialize minimum to infinity
    for i in 1.. NCPUS() - 1 do
        # fork one worker per processor, except current processor
        fork worker(minimum, q, distance) on(i);
    od;
    GenerateJobs(q, distance); # main thread generates the jobs
    q$NoMoreJobs(); # all jobs have been generated now
    fork worker(minimum, q, distance) on(0);
    # jobs have been generated; fork a worker on this cpu too
end;
```

TSP Orca

```
process worker(
    minimum: shared IntObject; # length of current best path
    q: shared TspQueue; # job queue
    distance: DistTab) # distances between cities
    job: JobType;
begin
    while q$GetJob(job) do # while there are jobs to do:
    tsp(job.len, job.path, minimum, distance);
    # do sequential tsp
    od;
end;
```