Paralelismo y Concurrencia en Sistemas

Work Pool en arquitecturas multicomputadoras

Dr. Alejandro J. García
e-mail: agarcia@cs.uns.edu.ar
http:// cs.uns.edu.ar /~ajg

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca - Argentina

Distributed Workers

- El paradigma de "replicated workers" permite a un programa generar tareas dinámicamente mientras se ejecuta el programa.
- Estas tareas son almacenadas en un "work pool" de donde son tomadas por los "trabajadores"
- La técnica de "replicated workers" puede emplearse tanto en arquitecturas Multiprocesadores como en Multicomputadoras.

- Lo que sigue es de:

The Art of Parallel Programming. Capítulo 11
Bruce P. Lester. 1993. Prentice Hall

Distributed Workers

- En multicomputadoras el work pool ya no puede considerarse un área de memoria compartida accedida por trabajadores idénticos.
- Pero los datos del problema tampoco podrán estar en memoria compartida.
- Al programar el sistema, cada procesador tendrá una parte de los datos (Ej: un grafo deberá estar repartido entre las memorias de los procesadores)

Distributed Workers in Multicomputers

- El work pool debe particionarse y distribuirse en las memorias locales de cada procesador.
- Los trabajadores en general no serán idénticos y ciertas tareas irán a trabajadores específicos.
- Esto se debe a que una tarea podrá referirse a un partición específica de los datos, y por lo tanto, para evitar sobrecarga en la comunicación, se le asignará al trabajador que tiene acceso local a esa partición.
- Debe hacerse una detección de terminación distribuida.

Caminos mínimos con Distributed Workers

- Cada worker tendrá su propio channel (pool)
- Si P es el número de procesadores, entonces
  workpool: array [1..P] of channel of …
- Worker(i) leerá sólo del canal workpool[i]

Detección de terminación en Multicomputers

- Los retardos en la transmisión de los mensajes entre procesadores presentan un problema para detectar la terminación.

- El workpool puede parecer vacío pero puede haber datos "en viaje"

- Supongamos que en algún procesador hay un monitor con un contador global C que controla cuantos workers están ociosos.
- Si C = cantidad de workers esto implicaría que están todos ociosos

Detección de terminación en Multicomputers

- El contador global C se actualiza mediante mensajes:
  - si get_work encuentra el canal vacío envía al monitor un (+1)
  - al despertar get_work envia un (-1)

Existe un problema: suponga que C = P-1
w(i) termina, envía un trabajo a w(j), y (+1) al monitor
w(j) recibe el trabajo y envía (-1) al monitor

- Si el (-1) llega primero, entonces todo bien,
- pero si el (+1) llega primero, el monitor suma 1 a C, y queda C=P indicando que están todos ociosos.

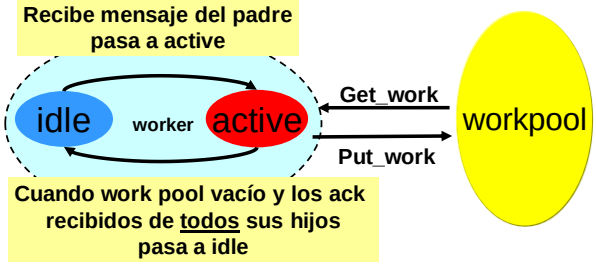
Condiciones de terminación en Multicomputers

La tarea en paralelo termina cuando:
1) no hay mas tareas en el work pool, y
2) ningún worker está trabajando, y
3) no hay items "en viaje"

- El algoritmo a presentar funciona para cualquier posible retardo
- Se basa en "message acknowledgments": cada mensaje enviado recibe un ack del proceso remoto
- El mismo channel (port de comunicación) se usa para items y acknowledges. item

Algoritmo para detectar terminación

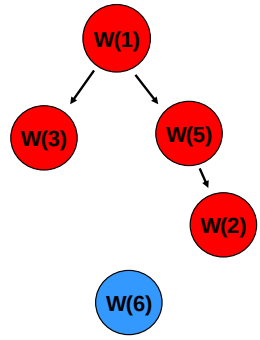
- Cada worker estará: active or idle
- El primero que lo activa se convierte en su padre.




```
procedure monitor;
(* espera a que el primer worker (raíz) avise que está
   idle y luego hacer terminar a todos los demás  *)
var i: integer; workitem: worktype;
Begin
   (* espera el "ack" del worker inicial*)
workitem:= workpool[0];
   (* arma el mensaje de terminación para todos *)
workitem.source:=0; workitem.distance:=done;
   (* envia el mensaje de terminación a todos *)
for i:=1 to numworkers do workpool[i] := workitem;
end; (*monitor*)
```


Árbol de trabajadores activos

```
Procedure worker(i);
   active:=false; (* idle*)
   get_work(); (* active/idle *)
   while not done do;
     if (...) then put_work(…);
     get_work();
   end;
end worker;
```



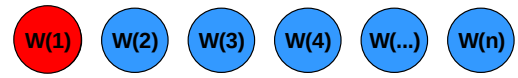

```
procedure get_work(…);
    repeat (* hasta que una tarea real sea recibida *)
        if active and (ackcount=0) and ("pool vacio…")
        then  (*cambiar a  "idle" y enivar "ack" al padre*)
              begin active:=false; workpool[parent]:=ack  ;end;
        inwork:= leer del pool … ; (* read msg or wait for it*)
        if inwork "es un ack…" then ackcount:=ackcount -1;
    until inwork "no sea ack";
    if active (*acknowledge inmeadiatly*)
      then Responde inmediatamente
           porque no será su padre
    else (* change to active state *)
        begin active:= true; parent:= "sender del work"; end;
…
end;
```

Caminos mínimos con Distributed Workers

- El algoritmo para el ejemplo de caminos mínimos debe adaptarse para Multicomputers: el worker(i) trabajará sobre el vértice i, en el procesador i, (asumimos que hay suficientes procesadores)
- El grafo G y MINDIST[ ] estarán particionados en las memorias de cada procesador.
- El worker(i) recibirá, mindist[i] y toda la fila G[i], esto es, todos los adyacentes del vertice i

Creación de los trabajadores

```
type item = record source, distance: integer; end;
var workpool : array of channel of item;
    start:item;
begin
start.source:=0; start.distance:=0;
workpool[1]:= start; (* esto activará al worker 1 *)
FORALL i:=1 to numworkers
 do (@i port workpool[i]) worker(i, G [i], finaldist[i]);
```




```
procedure worker (me: integer; myweight:weightrow;
                  var answer:integer);
var mindistance,w,trialdistance,ackcount,
    parent:integer;   active:boolean;

 procedure put_work(vertex,outputdistance:integer);
  …
 procedure get_work(var inputdistance: integer);
  …

begin
ackcount:=0;   active:=false;   mindistance:= infinity;
        …
end;
```

Variables usadas por put y getwork

Put_work

- Put_work envía la tarea directamente al canal del trabajador que la resolverá

```
procedure put_work(vertex,outputdistance:integer);
 var outwork: worktype;
 begin
   ackcount := ackcount + 1;
   outwork.source := me;
   outwork.distance := outputdistance;
   workpool[vertex] := outwork; (* envia al canal del worker*)
 end;
```

Variables no locales

- El mismo channel (port de comunicación) se usa para items y acknowledges

Creación de los trabajadores

```
type item = record source, distance: integer; end;
var workpool : array of channel of item;  start:item;
begin
FORK(@0 PORT workpool[0]) monitor;
start.source:=0; start.distance:=0;
workpool[1]:= start;
FORALL i:=1 to numworkers
 do (@i port workpool[i]) worker(i, G [i], finaldist[i]);
```

Quién lo envía y que distancia hay hasta allí

Esto activará worker 1 con padre 0 (el monitor)


```
procedure worker (me: integer; myweight:weightrow;
                  var answer:integer);
… procedimientos get_work y put_work……
begin
ackcount:=0;   active:=false;  mindistance:= infinity;
get_work(trialdistance);  (*lo saca de saca de workpool[me]*)
while trialdistance <> done do (* mientras haya tareas*)
begin if trialdistance < mindistance then
          begin
              mindistance:=trialdistance;
              for w:=1 to n do
              if myweight[w] < infinity
                then put_work(w,mindistance+myweight[w]);
          end;
         get_work(trialdistance); (*saca otro de workpool[me]*)
end;
answer:=mindistance;
end;
```

```
procedure get_work(var inputdistance: integer);
   repeat (* hasta que una tarea real sea recibida *)
      if active and (ackcount=0) and (not workpool[me]?)
      then  (*cambiar a  "idle" y enivar "ack" al padre*)
            begin active:=false; workpool[parent]:=ack  ;end;
      inwork:=workpool[me]; (* read msg or wait for it*)
      if inwork.source = ack then ackcount:=ackcount -1;
   until inwork.source <> ack;
   if active (*acknowledge inmeadiatly*)
   then workpool[inwork.source]:=ackwork
   else (* change to active state *)
         begin active:= true; parent:=inwork.source; end;
   inputdistance:= inwork.distance;
end;
```

Variables no locales

Responde inmediatamente porque no será su padre


```
ARCHITECTURE HYPERCUBE(3);
CONST numworkers = 7;      n = numworkers;
          done = -1;  ack = -2;  infinity = 32000;
TYPE worktype = record   source, distance:integer; end;
        weightrow = array [1..n] of integer;
VAR  startwork: worktype; i,j: integer;
      workpool:ARRAY[0..numworkers] OFchannel OF worktype;
      weight: array [1..n] of weightrow;
      finaldist: array [1..n] of integer;
.....
FORK (@ 0 PORT workpool[0]) monitor; (*creates monitor*)
startwork.source:=0;
startwork.distance:=0;
workpool[1]:= startwork; (* this msg will activate worker 1 *)
FORALL i:=1 to numworkers
        do (@i port workpool[i]) worker(i, weight[i], finaldist[i]);
JOIN; (*monitor*)
```