Problemas

Jesús Labarta

MPI + OpenMP

- Parallelize with OpenMP, MPI and Nested (MPI + OpenMP)

```
#define N 1000000
double A[N],B[N],s;
for (iter=0; iter<100; iter++) {
   for (i=0;i<N;i++) {
      A[i]= A[i]+ B[i];
   }
   for (i=1;i<N;i++) {
      B[i]+= g(A[i-1]);
   }
   for (i=0;i<N;i++) {
      s+= f(A[i]);
   }
}
```

OpenMP

```
#define N 1000000
double A[N],B[N],s;
for (iter=0; iter<100; iter++) {
   #pragma omp parallel for reduction (+:s)
   for (i=0;i<N;i++) {
     A[i]= A[i]+ B[i];
     s+= f(A[i]);
   }
   #pragma omp parallel for
   for (i=1;i<N;i++) B[i]+= g(A[i-1]);
}
```

Loop fusion: locality

MPI

```
#define N 1000000
double *A,*B, s; integer quisoc,somos,next,prev,Mychunk;
double Aanterior;
MPI_comm_rank(COMMWORLD, &quisoc); MPI_comm_size(COMMWORLD, &somos);
Mychunk = N/somos; /* Ajustar si N no es multiplo de P */
malloc: A y B. Size = Mychunk
next=quisoc<somos-1 ? quisoc+1:MPI_PROC_NULL;
prev=quisoc>0 ? quisoc-1:MPI_PROC_NULL;
for (iter=0; iter<100; iter++) {
   A[Mychunk-1]= A[Mychunk-1]+ B[Mychunk-1]
   MPI_send(&A[Mychunk-1],1,DOUBLE, next,…);
   for (i=0;i<Mychunk-1;i++) A[i]= A[i]+ B[i];
   for (i=0;i<Mychunk;i++)   s+= f(A[i]);
   MPI_recv(&Aanterior,1,DOUBLE, prev,…);
   B[0]+= g(Aanterior);
   for (i=1;i<Mychunk;i++) B[i]+= g(A[i-1]);
}
MPI_allreduce(&s,&s, 1,DOUBLE, MPI_SUM, MPI_COMM_WORLD);
```

Advance transfer

Independent

Reception

Dependent

Just once

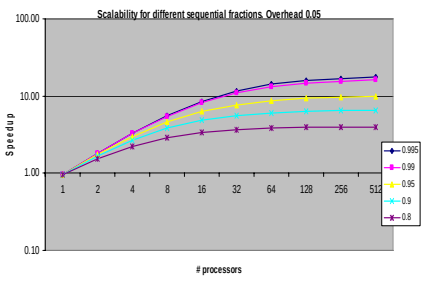
MPI + OpenMP

```
#define N 1000000
double *A,*B, s; integer quisoc,somos,next,prev,Mychunk;
double aanterior;
MPI_comm_rank(COMMWORLD, &quisoc); MPI_comm_size(COMMWORLD, &somos);
Mychunk = N/somos; /* Ajustar si N no es multiplo de P */
malloc: A y B. Tamaño = Mychunk
next= … ;prev= …;
for (iter=0; iter<100; iter++) {
   A[Mychunk-1]= A[Mychunk-1]+ B[Mychunk-1]; s+= A[Mychunk-1];
   MPI_send(&A[Mychunk-1],1,DOUBLE, next,…);
   #pragma omp parallel for reduction (+:s)
   for (i=0;i<Mychunk-1;i++) {
     A[i]= A[i]+ B[i];
     s+= f(A[i]);
   }
   MPI_recv(&aanterior,1,DOUBLE, prev,…);
   B[0]+= g(aanterior);
   #pragma omp parallel for
   for (i=1;i<Mychunk;i++) B[i]+= g(A[i-1]);
}
MPI_allreduce(&s,&s, 1,DOUBLE, MPI_SUM, MPI_COMM_WORLD);
```

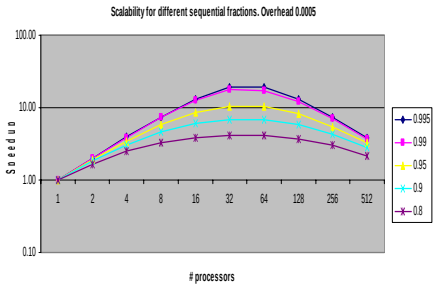

Preguntas

- Donde se declara iter? Cuantas copias hay ?
- Que pasaría en la versión MPI si el bucle interno fuera de 0 a N/(iter+1) ?
- Si el original hubiera tenido sólo el segundo bucle en i, se podrían haber distribuido los datos de alguna forma que evitara la comunicación? Que implicaría en el fuente?
- Cuantos fallos de cache se generaran en un sistema con 4 procesadores para la versión OpenMP si el tamaño de la línea es 128B y un double 8 bytes y N fuera: N=10240? N=10248? N=512 ?
- Un proceso MPI tiene un vector de 1024 elementos. Escribir el código para distribuir bloques contiguos de elementos entre todos los procesos. Lo mismo para distribuir los elementos de forma entrelazada.

Speed-up: Overheads



S(p) = 1 / (1 - f + (f / p)+o)



S(p) = 1 / (1 - f + (f / p)+o*p)

Speed-up: Overheads

- **Modelo 1** **S(p) = 1 / (1 - f + (f / p)+o)**
  - T(P)=Tseq + Tpar/P + o*T(1)
  - o: overhead propocional al tiempo secuencial total (tanto por uno)
- **Modelo 2**
  - T(P)=Tseq + Tpar/P + o*P*T(1) **S(p) = 1 / (1 - f + (f / p)+o*p)**
  - o: overhead proporcional al tiempo secuencial total y al numero de procesadores
- **Razonable???**
  - Proporcional al número de veces que se abre y cierra?
  - Proporcional al

Balanceo de carga

- En el siguiente programa, delay es una rutina no paralelizable internamente que tarda un tiempo proporcional al argumento. Las distintas iteraciones de bucle si son paralelizables.
- Que eficiencia se puede conseguir con N=8 si P= 4? Y si P=5?
- Que eficiencia se conseguiría para N=P?
- Particularizar la ley de Amdahl para este caso en función de N y P.

```
      PROGRAM test
      PARAMETER (N=1024)
      INTEGER i

      DO i=0,N
         call delay(i)
      ENDDO

      END
```

Amdahl´s Law

Program XXXXX. Problem size S0

Time XXXXX results

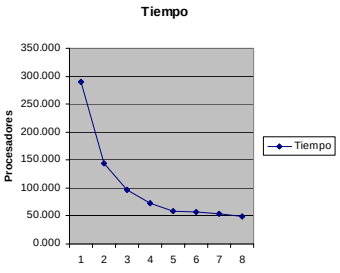
```
OMP_NUM_THREADS=1:   289.080u 0.220s 4:49.42 99.9%   3+32725k 0+0io 2pf+0w
OMP_NUM_THREADS=2:   289.170u 0.300s 2:24.95 199.7%  3+32733k 0+0io 0pf+0w
OMP_NUM_THREADS=3:   288.400u 0.400s 1:37.06 297.5%  3+32728k 0+0io 0pf+0w
OMP_NUM_THREADS=4:   287.430u 0.470s 1:12.54 396.8%  3+32740k 0+0io 0pf+0w
OMP_NUM_THREADS=5:   286.490u 0.730s 0:58.92 487.4%  3+32718k 0+0io 0pf+0w
OMP_NUM_THREADS=6:   282.610u 1.580s 0:57.58 493.5%  3+32638k 0+0io 0pf+0w
OMP_NUM_THREADS=7:   285.230u 4.800s 0:53.54 541.7%  3+32299k 0+0io 0pf+0w
OMP_NUM_THREADS=8:   280.000u 3.440s 0:49.47 572.9%  3+32456k 0+0io 0pf+0w
```

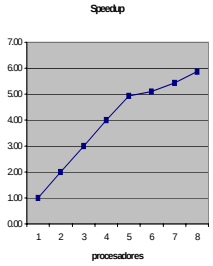
Internal measurement report:

```
iterations: 1018 time to compute = 289.089877
iterations: 1018 time to compute = 144.639161
iterations: 1018 time to compute = 96.734308
iterations: 1018 time to compute = 72.174009
iterations: 1018 time to compute = 58.590414
iterations: 1018 time to compute = 56.851213
iterations: 1018 time to compute = 53.128441
iterations: 1018 time to compute = 49.153017
```


Amdahl´s Law

- **Tiempo**
- **Speedup**





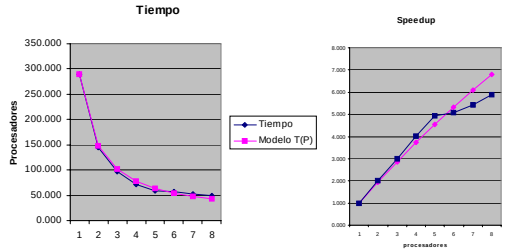
Amdahl´s Law

- **Fit**

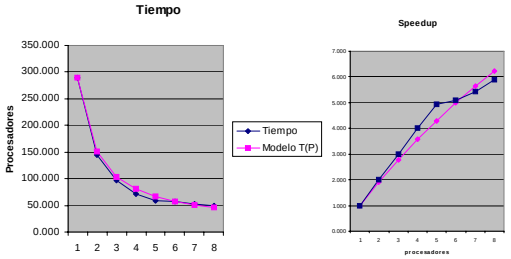
$$T(P) = Tseq + Tpar / P$$

$$pf = Tpar / T(1)$$

  - Speedup
    - ✓ pf=0.9594

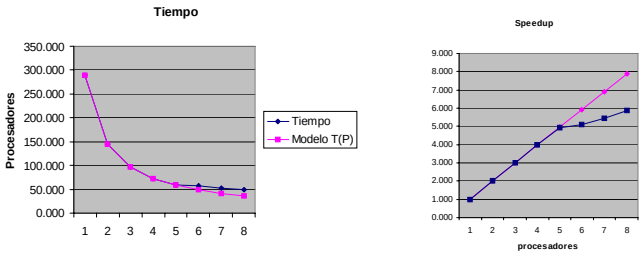


  - Time
    - ✓ pf=0.9750



Amdahl´s Law

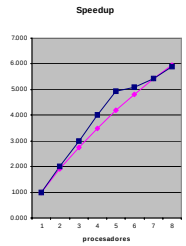
- **Fit only the first 5 values**



  - ✓ pf=0.9977

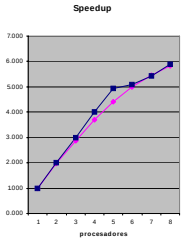
Amdahl´s Law

- **Fit only the last 3 values**



$T(P) = Tseq + Tpar / P$

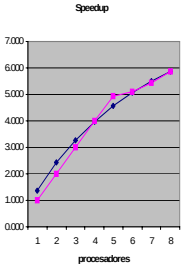
$pf = 0.9513$



$T(P) = Tseq + Tpar / P + o * (P - 1)$

$pf = 0.9982$

$o = 1.86$



$T(P) = (Tseq + Tpar / P) / S'(1)$

$pf = 0.8773$

$S'(1) = 1.361$

- **Interpretation?**

Amdahl´s Law

Program XXXXX. Problem size S1

Time XXXXX results

```
      OMP_NUM_THREADS=1:   24.000u 0.040s 0:24.06 99.9%    11+1615k 0+0io 0pf+0w
      OMP_NUM_THREADS=2:   24.390u 0.210s 0:12.32 199.6%   11+1619k 0+0io 0pf+0w
      OMP_NUM_THREADS=3:   24.810u 0.710s 0:08.53 299.1%   11+1599k 0+0io 0pf+0w
      OMP_NUM_THREADS=4:   25.230u 0.780s 0:07.15 363.7%   11+1612k 0+0io 0pf+0w
      OMP_NUM_THREADS=5:   25.540u 1.290s 0:06.62 405.2%   11+1592k 0+0io 0pf+0w
      OMP_NUM_THREADS=6:   26.390u 1.300s 0:06.75 410.2%   11+1608k 0+0io 0pf+0w
      OMP_NUM_THREADS=7:   26.260u 2.960s 0:10.26 284.7%   10+1534k 0+0io 0pf+0w
      OMP_NUM_THREADS=8:   27.230u 3.080s 0:10.24 295.9%   10+1549k 0+0io 0pf+0w
```

Internal measurement report:

```
iterations: 15574 time to compute = 24.004162
iterations: 15575 time to compute = 12.287694
iterations: 15575 time to compute = 8.503046
iterations: 15575 time to compute = 7.126867
iterations: 15575 time to compute = 6.593301
iterations: 15576 time to compute = 6.722184
iterations: 15576 time to compute = 10.231101
iterations: 15576 time to compute = 10.214918
```

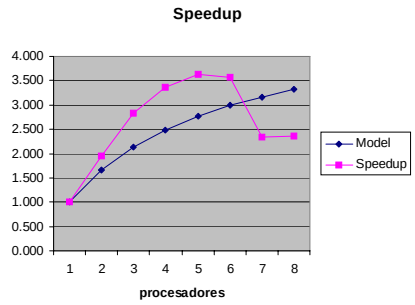
Amdahl´s Law

- **Fit**

$$T(P) = Tseq + Tpar / P$$

$$pf = Tpar / T(1)$$

$$pf = 0.798$$

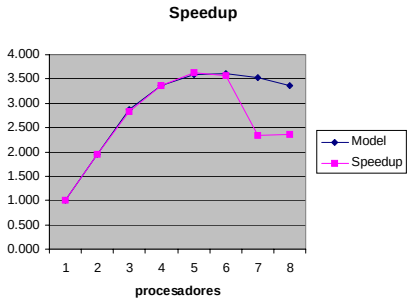


Amdahl´s Law

- **Fit**
  - First 6 points
  - $$T(P) = Tseq + Tpar / P + o * (P - 1)$$

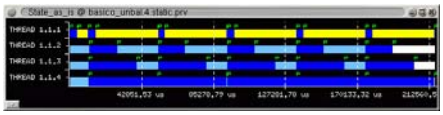
$$pf = 0.946$$

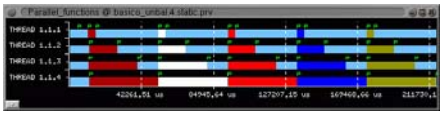
$$o = 0.725$$



“Reading” performance counters

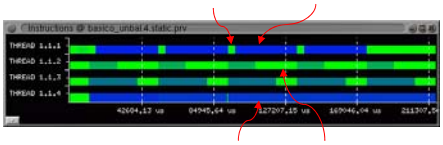
```
      PROGRAM test
      PARAMETER (N=1024)
      REAL dummy(N), factor, var
      REAL a(64000)
      INTEGER i,j,time
      common /varios/a

      factor=1/1.0000001

      DO iter=1,5
C$OMP PARALLELDO SCHEDULE(STATIC)
C$OMP SHARED(dummy) PRIVATE(I,time)
         DO i=0,N
            dummy(i)= dummy(i)*factor
            time = i/100
            call delay(time)
         ENDDO
      ENDDO

      END
```





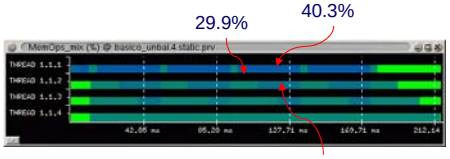
“Reading” performance counters

- **Instructions**

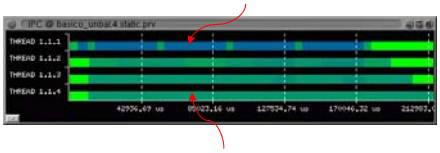


- **L2 miss ratio**

- **Mem Ops mix**



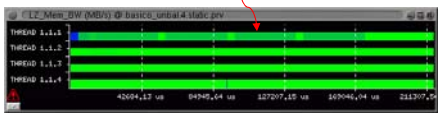
- **IPC**



“Reading” performance counters

- **Memory Bandwidth**

27.2MB/s



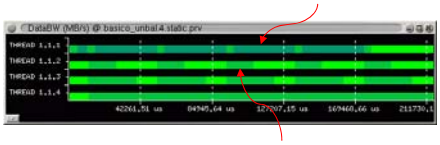
- **MIPS**

219M

180M

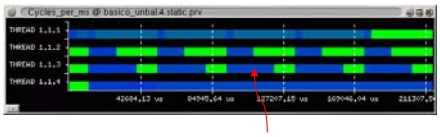
- **CPU Bandwidth**

353MB/s



213MB/s

- **Cycles/s**



375M

Traspuesta

- **Escribir el programa para trasponer una matriz de 1000x1000 con 10 procesadores en OpenMP y en MPI.**

Variables scope

```
! The computation of forces and energies is fully parallel.
!$omp parallel do private(?????)
!$omp& reduction(+ : ?????)
do i=1,np
    f(1:nd,i) = 0.0
    do j=1,np
    if (i .ne. j) then
          call dist(nd,box,pos(1,i),pos(1,j),rij,d)
          pot = pot + 0.5*v(d)
          do k=1,nd
                    f(k,i) = f(k,i) - rij(k)*dv(d)/d
          enddo
    endif enddo
    kin = kin + dotr8(nd,vel(1,i),vel(1,i))
enddo
!$omp end parallel do
kin = kin*0.5*mass
```

Vector distancia y norma

Variables scope

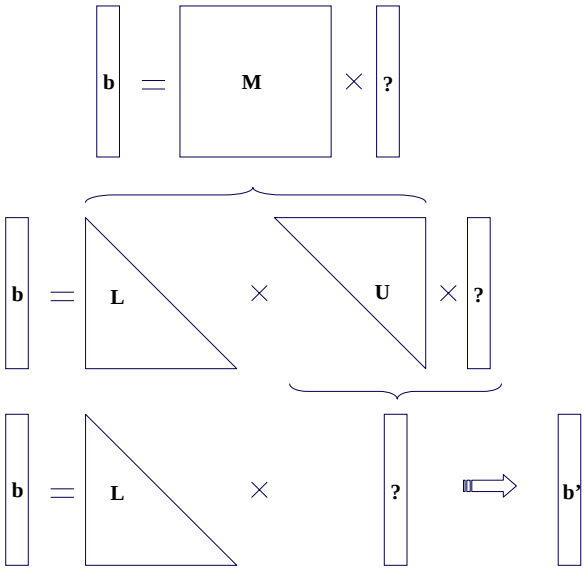
- **http://www.openmp.org → Sample Programs → md.f**
  - **i,j,k,rij,d**
  - **pot, kin**

OpenMP / MPI

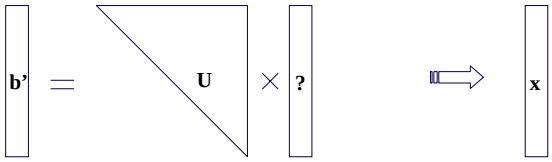
- Paralelizar con OpenMP y con MPI

```
#define VECES 100
double A[1024,1024],B[1024,1204];
for (ext=0; ext<VECES; ext++)
   for (i=1;i<1024;i++)
      for (j=1;j<1024;j++) {
         A[i,j]= f(A[i-1,j-1],A[i,j],B[i,j]);
}
```


LU

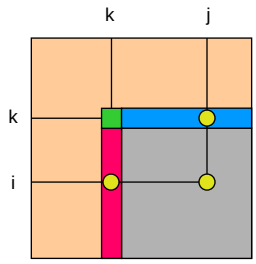


LU



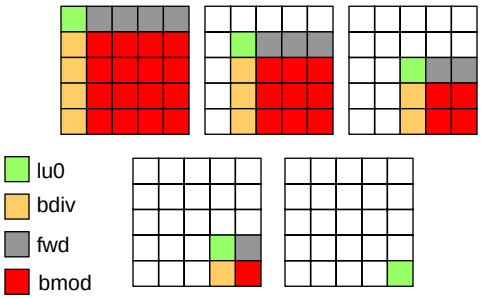
LU

```
program LU
integer i, j, k
double precision A(N, N)

do k = 1, N
  do i = k + 1, N
    A(i, k) = A(i, k) / A(k, k)
  enddo
  do i = k + 1, N
    do j = k + 1, N
      A(i, j) = A(i, j) - A(i, k) * A(k, j)
    enddo
  enddo
enddo

end
```



Blocked LU

- **Multiple levels of parallelism:**
  - Intra-block loop-level parallelism
  - Inter-block parallelism




```
do kk=1,NB
  call lu0(A,kk)
  do jj=kk+1,NB
    call fwd(A,kk,jj)
  enddo
  do ii=kk+1,NB
    call bdiv(A,ii,kk)
    do jj=kk+1,NB
      call bmod(A,ii,jj,kk)
    enddo
  enddo
enddo
```


Blocked LU

```
subroutine lu0(a,ii)
      implicit none
      include 'parameter.inc'
      double precision a(N,N)
      integer i,j,k
      integer ii
      integer kinf,ksup

ccc traducciones
      kinf = (ii-1)*B + 1
      ksup = min(ii*B, N)

ccc calculo sobre espacio de indices global a
nivel elemento

      do k=kinf,ksup
       do i=k+1,ksup
        A(i,k) = A(i,k) / A(k,k)
        do j=k+1,ksup
          A(i,j) = A(i,j) - A(i,k) * A(k,j)
        enddo
       enddo
      enddo

      end
```

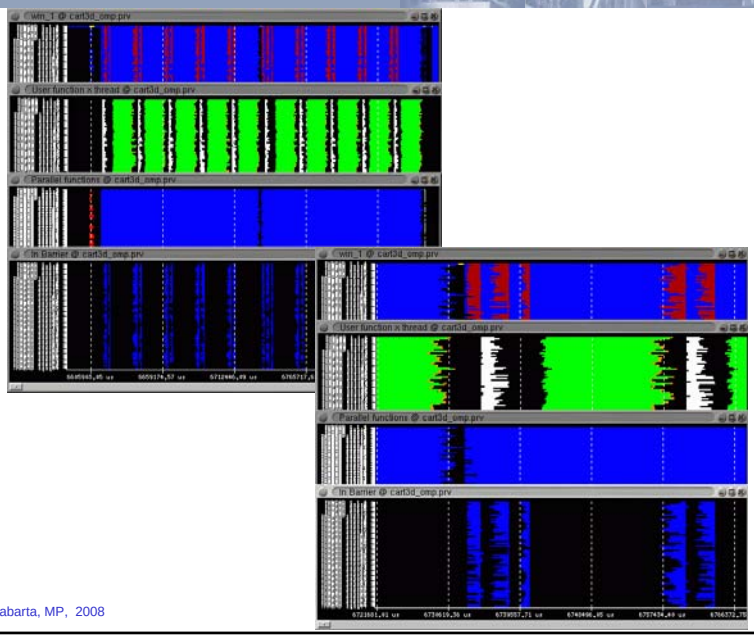
```
subroutine fwd(A,ii,jj)
      implicit none
      include 'parameter.inc'
      double precision a(N,N)
      integer i,j,k
      integer ii,jj,kk
      integer kinf,ksup, iinf,isup, jinf,jsup

ccc traducciones
      kinf = (ii-1)*B+1
      ksup = min(ii*B, N)
      isup = min(ii*B, N)
      jinf = (jj-1)*B+1
      jsup = min(jj*B, N)

ccc calculo sobre espacio de indices global a
nivel elemento

      do k=kinf,ksup
        do i=k+1,isup
          do j=jinf,jsup
            A(i,j) = A(i,j) - A(i,k) * A(k,j)
          enddo
        enddo
```

Blocked LU

```
subroutine bdiv(a,ii,kk)
      implicit none
      include 'parameter.inc'
      double precision a(N,N)
      integer i,j,k
      integer ii,kk
      integer kinf,ksup, iinf,isup

      kinf = (kk-1)*B+1
      ksup = min(kk*B, N)
      iinf = (ii-1)*B+1
      isup = min(ii*B, N)

      do k=kinf,ksup
        do i=iinf,isup
          A(i,k) = A(i,k) / A(k,k)
          do j=k+1,ksup
            A(i,j) = A(i,j) - A(i,k)*A(K,j)
          enddo
        enddo
      enddo

      end
```

```
subroutine bmod(A,ii,jj,kk)
      implicit none
      include 'parameter.inc'
      double precision a(N,N)
      integer i,j,k
      integer ii,jj,kk
      integer kinf,ksup, iinf,isup, jinf,jsup

      kinf = (kk-1)*B+1
      ksup = min(kk*B, N)
      iinf = (ii-1)*B+1
      isup = min(ii*B, N)
      jinf = (jj-1)*B+1
      jsup = min(jj*B, N)

      do k=kinf,ksup
        do i=iinf,isup
          do j=jinf,jsup
            A(i,j) = A(i,j) - A(i,k) * A(k,j)
          enddo
        enddo
      enddo

      end
```

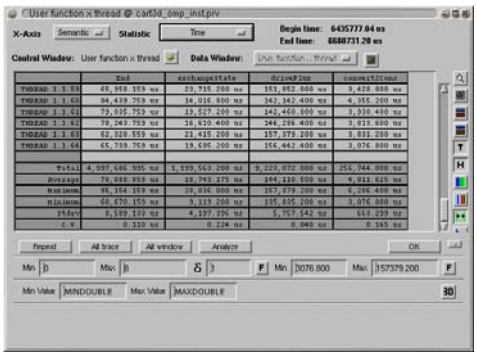

Estructura del programa

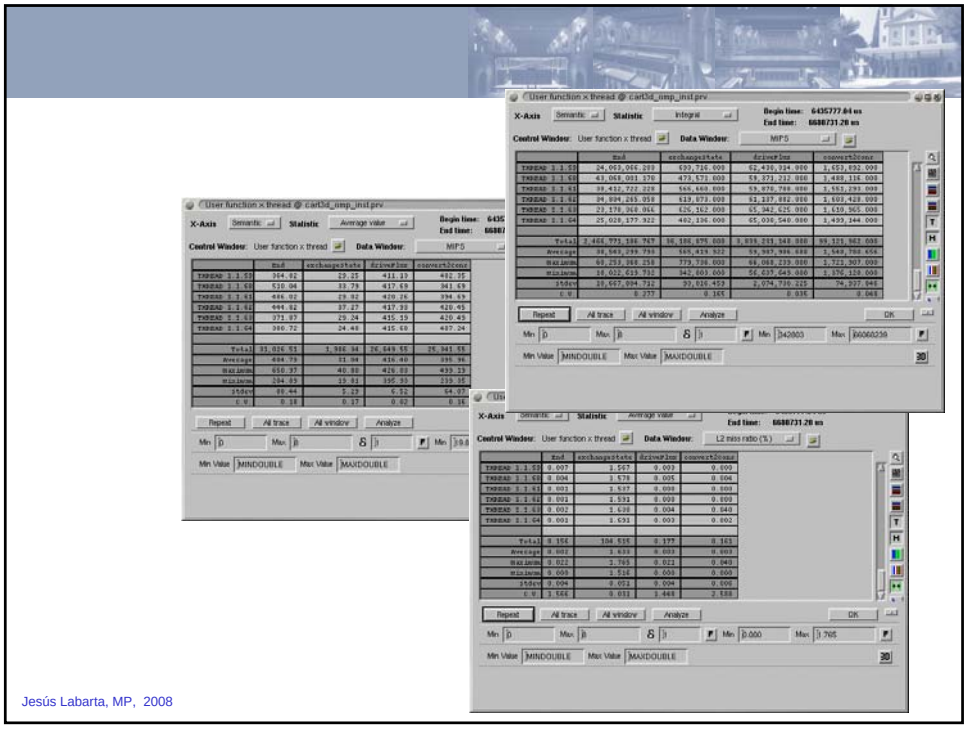


Estructura del programa?

```
Do outer=1,2
C$OMP Parallel
   do i=1,5
       If (i==1) {
          Call rutina_no_instrumentada
          C$OMP barrier
       }
       call rutina_blanca
       C$OMP barrier
       If (i==1) {
           Call rutina_no_instrumentada
           C$OMP barrier
       }
       Call runtina_verde
       C$OMP barrier
    end do
C$OMP end parallel
end do
```

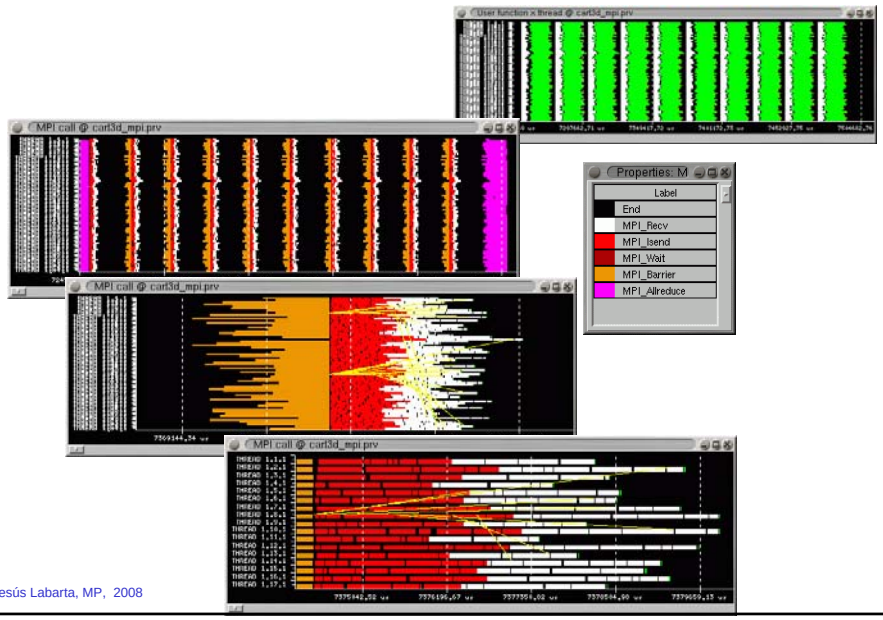

- **Blanca/verde?**





Jesús Labarta, MP, 2008

En MPI



Jesús Labarta, MP, 2008

Array lock

- **Reuse positions**

```
lock:   f&i   r1, counter;
        if (r1==MAXSIZE){
           r1=0;
           counter = 0;
        }
        while (array[r1]==WAIT);
        array[r1] = WAIT;
```

```
Unlock: r1= (r1+1)%MAXSIZE;
        array[r1]=CONTINUE;
```

- **Correcto ??**
- **Implementarlo con instrucción swap (v1,v2) en lugar de f&a ??**

Array lock

- **Reuse positions**

```
lock:   f&i   r1, counter;
        if (r1==MAXSIZE){
           r1=0;
           counter = 0;
        }
        while (array[r1]==WAIT);
        array[r1] = WAIT;
```

```
Unlock: r1= (r1+1)%MAXSIZE;
        array[r1]=CONTINUE;
```

Array lock

- **Reuse positions**

```
lock:   f&i   r1, counter;
        r1 = r1%MAXSIZE;
        while (array[r1]==WAIT);
        array[r1] = WAIT;
```

```
Unlock: r1= (r1+1)%MAXSIZE;
        array[r1]=CONTINUE;
```


Array lock

- **Reuse positions**

```
lock:   f&i   r1, counter;
        r1 = r1%MAXSIZE;
        while (array[r1]==WAIT);
        array[r1] = WAIT;
```

```
Unlock: r1= (r1+1)%MAXSIZE;
        array[r1]=CONTINUE;
```

- 4 threads hacen un bucle infinito de (lock,unlock) en un sistema con bus. Cuanto tarda por iteración? Suponer arbitro del bus fair y duración nula de las instrucciones aritméticas y de control de flujo. Generalizarlo a P procesadores.

Array lock

- **Reuse positions**

```
lock:   f&i   r1, counter;
        r1 = r1%MAXSIZE;
        while (array[r1]==WAIT);
        array[r1] = WAIT;
```

```
Unlock: r1= (r1+1)%MAXSIZE;
        array[r1]=CONTINUE;
```

  - Coste:

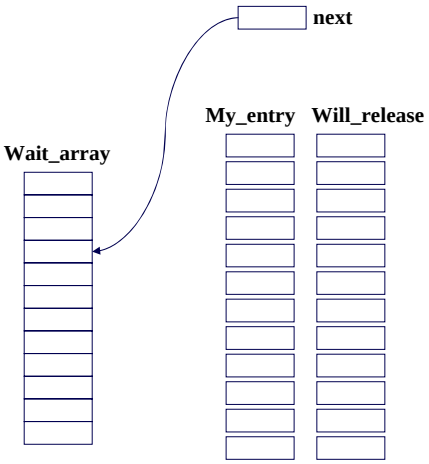
P1 RdX C	P2 RdX C	P3 RdX C	P4 RdX C	P1 Rd A	P2 Rd A	P3 Rd A	P4 Rd A	P1 RdX A	P1 RdX A'

P2 Rd A	P1 RdX C	P2 RdX A	P1 Rd A	P2 RdX A'

●●●

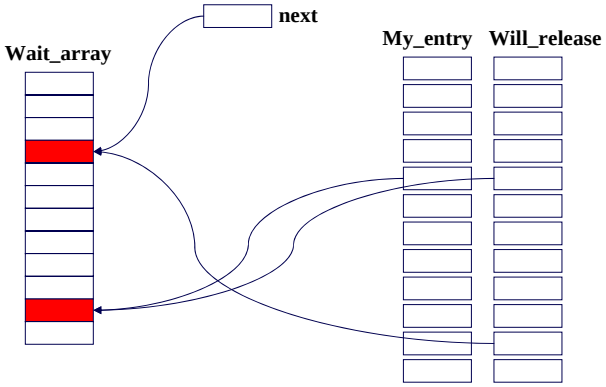
Array lock (swap)

- **Implementarlo con instrucción swap (v1,v2)**
  - Next:
    - ✓ Siguiente posición donde esperar
      - – El ultimo en entrar indicara allí cuando termina
  - My_entry:
    - ✓ yo se que esta libre.
  - Will_release:
    - ✓ Copia temporal de My_entry.
    - ✓ Indica a quien liberare.



Array lock (swap)

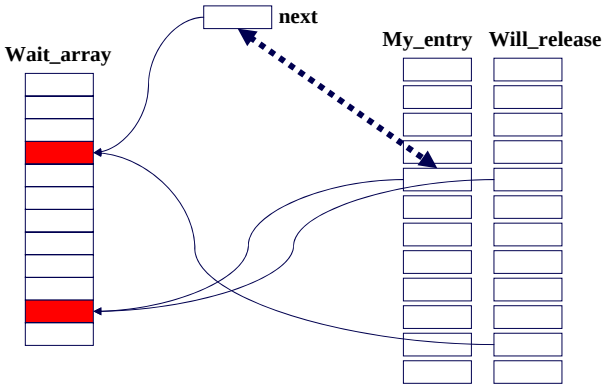
- **Implementarlo con instrucción swap (v1,v2)**



✓ Tomar nota de qué liberar

Array lock (swap)

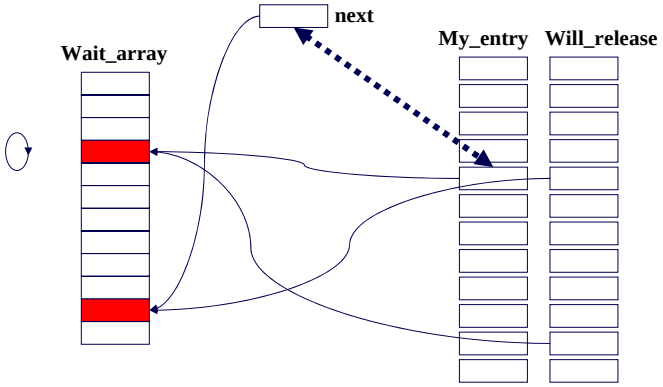
- **Implementarlo con instrucción swap (v1,v2)**



✓ Saber donde me toca esperar (Swap) …

Array lock (swap)

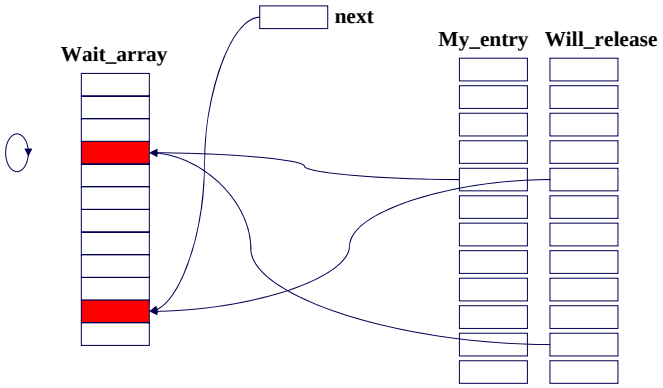
- **Implementarlo con instrucción swap (v1,v2)**



✓ …y esperar

Array lock (swap)

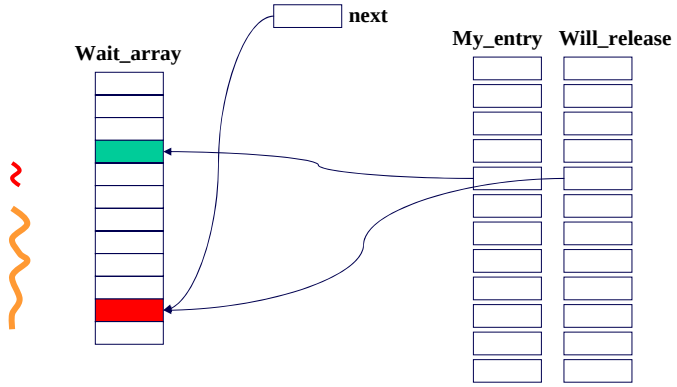
- **Implementarlo con instrucción swap (v1,v2)**



✓ Termina el anterior y paso …

Array lock (swap)

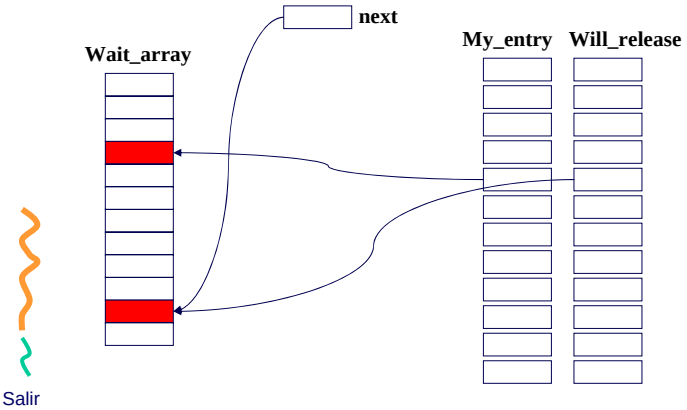
- **Implementarlo con instrucción swap (v1,v2)**



- ✓ Dejar a rojo para la siguiente vez
- ✓ y entrar en la exclusión mutua

Array lock (swap)

- **Implementarlo con instrucción swap (v1,v2)**



- ✓ Salir

Array lock (swap)

- **Implementarlo con instrucción swap (v1,v2)**

```
lock: will_release=My_entry;
      swap (next, My_entry);
      while(wait_array[My_entry]==WAIT);
      wait_array[My_entry] = WAIT;
```

```
Unlock: wait_array[will_release]=CONTINUE;
```


Barrier algorithms

- **Problema??**

```
Barrier(barr) {
   local_sense = !(local_sense);
   lock (barr.lock);
   mycount =  barr.counter++;
   unlock (barr.lock);
   if (barr.counter == P) {
      barr.counter =0;
      barr.flag = local_sense;
   } else
      while (barr.flag != local_sense);
}
```

- **Una vez arreglado, cuantas transferencias de líneas de cache tienen lugar si lo ejecutan cuatro procesos que**
  - entran en orden y con bastante diferencia de tiempo entre ellos
  - entran “simultáneamente”

Barrier algorithms

- **Problema ??**

```
Barrier(N) {
   lock (barr.lock);
   barr.counter++;
   if (barr.counter == P) {
      barr.releasing = 1;
      barr.counter--;
   }else{
      unlock (barr.lock);
      while (!barr.releasing);
      lock (barr.lock);
      barr.counter--;
      if (barr.counter == 0)
          barr.releasing = 0;
   }
   unlock (barr.lock);
}
```


Barrier algorithms

- **Problema ??**

```
Barrier(N) {
   if (f&a(barr.counter,1) == P-1) {
      barr.counter=0;
   }else{
      while (barr.counter !=0);
}
```

Cuantificación de rendimiento

	Instructions	FLOPS	References	Total Reads	Total Writes	Shared Reads	Shared Writes
Barnes-Hut	2002.74	239.24	720.13	406.84	313.29	225.04	93.23
LU	489.52	92.2	151.07	103.09	47.99	92.79	44.74
Raytrace	833.35	-	290.35	210.03	80.31	161.1	22.35
Ocean	376.51	101.54	99.7	81.16	18.54	76.95	16.97
Radix	14.02	-	5.27	2.9	2.37	1.34	0.81

Cuantificación de rendimiento

- **Cache: 1MB**
- **Set associative LRU: 4**
- **Line: 64B**
- **Transiciones /1000 refs**
- **Σ may be ≥ 1000**

		NP	I	E	S	M
Barnes-Hut	NP	0	0	0.0011	0.0362	0.0035
	I	0.0201	0	0.0001	0.1856	0.001
	E	0	0	0.0153	0.0002	0.001
	S	0.0029	0.213	0	97.1712	0.1253
	M	0.0013	0.001	0	0.1277	902.782
LU	NP	0	0	0	0.6593	0.0011
	I	0	0	0	0.0002	0.0003
	E	0	0	0.4454	0.0004	0.2164
	S	0.0339	0.0001	0	302.702	0
	M	0.0001	0.0007	0	0.2164	697.129
Raytrace	NP	0	0	1.3358	1.5486	0.0026
	I	0.0242	0	0	0.3403	0
	E	0.8663	0	29.0187	0.3639	0.0175
	S	1.1181	0.374	0	310.949	0.2898
	M	0.0559	0.0001	0	0.297	661.011

Cuantificación de rendimiento

- **Cache: 1MB**
- **Set associative LRU: 4**
- **Line: 64B**
- **Transiciones /1000 refs**

		NP	I	E	S	M
Barnes-Hut	NP	0	0	0.0011	0.0362	0.0035
	I	0.0201	0	0.0001	0.1856	0.001
	E	0	0	0.0153	0.0002	0.001
	S	0.0029	0.213	0	97.1712	0.1253
	M	0.0013	0.001	0	0.1277	902.782
LU	NP	0	0	0	0.6593	0.0011
	I	0	0	0	0.0002	0.0003
	E	0	0	0.4454	0.0004	0.2164
	S	0.0339	0.0001	0	302.702	0
	M	0.0001	0.0007	0	0.2164	697.129
Raytrace	NP	0	0	1.3358	1.5486	0.0026
	I	0.0242	0	0	0.3403	0
	E	0.8663	0	29.0187	0.3639	0.0175
	S	1.1181	0.374	0	310.949	0.2898
	M	0.0559	0.0001	0	0.297	661.011

Cuantificación de rendimiento

- **Cache: 1MB**
- **Set associative LRU: 4**
- **Line: 64B**
- **Transiciones /1000 refs**

		NP	I	E	S	M
Ocean	NP	0	0	1.2484	0.9565	1.6787
	I	0.6362	0	0	1.8676	0.0015
	E	0.204	0	14.004	0.024	0.9955
	S	0.41715	2.4994	0	134.716	2.2392
	M	2.6259	0.0015	0	2.2996	843.565
Radix	NP	0	0	0.004746	3.524705	11.41111
	I	0.130988	0	0	1.108079	4.57868
	E	0.000759	0.002848	0.080301	0	0.00019
	S	0.029804	1.120988	0	178.1932	0.817818
	M	0.044232	11.53127	0	4.03157	802.282

Jesús Labarta, MP, 2008

Cuantificación de rendimiento

- **Cache: 1MB**
- **Set associative LRU: 4**
- **Line: 64B**
- **Transiciones /1000 refs**

		NP	I	E	S	M
Ocean	NP	0	0	1.2484	0.9565	1.6787
	I	0.6362	0	0	1.8676	0.0015
	E	0.204	0	14.004	0.024	0.9955
	S	0.41715	2.4994	0	134.716	2.2392
	M	2.6259	0.0015	0	2.2996	843.565
Radix	NP	0	0	0.004746	3.524705	11.41111
	I	0.130988	0	0	1.108079	4.57868
	E	0.000759	0.002848	0.080301	0	0.00019
	S	0.029804	1.120988	0	178.1932	0.817818
	M	0.044232	11.53127	0	4.03157	802.282

Cuantificación de rendimiento

- **Memory bandwidth used**
  - 200 MIPS
  - Transfers:
    - ✓ Command: 6B
    - ✓ Data: 64B

Cuantificación de rendimiento

- **Motivo de la transición**

	NP	I	E	S	M
NP			Miss	Miss	Miss
I			Miss	Miss	Miss
E	Reemplazo	Invalidación	Hit	Downgrade	Hit
S	Reemplazo	Invalidación		Hit	Miss
M	Reemplazo	Invalidación		Downgrade	Hit

- **Transactions per transition**

	NP	I	E	S	M
NP	-	-	BusRd	BusRd	BusRdX
I	-	-	BusRd	BusRd	BusRdX
E	-	-	-	-	-
S	-	-	Not possible	-	BusUpgr
M	BusWB	BusWB	Not possible	BusWB	-

Cuantificación de rendimiento

- **Bytes per transition**

	NP	I	E	S	M
NP	0	0	70	70	70
I	0	0	70	70	70
E	0	0	0	0	0
S	0	0	0	0	6
M	70	70	0	70	0

Cuantificación de rendimiento

- **Bytes per Kref**

		NP	I	E	S	M	
Barnes-Hut	NP	0.00	0.00	0.08	2.53	0.25	
	I	0.00	0.00	0.01	12.99	0.07	
	E	0.00	0.00	0.00	0.00	0.00	
	S	0.00	0.00	0.00	0.00	0.75	
	M	0.09	0.07	0.00	8.94	0.00	Total
	Total	0.09	0.07	0.08	24.47	1.07	25.78
LU	NP	0.00	0.00	0.00	46.15	0.08	
	I	0.00	0.00	0.00	0.01	0.02	
	E	0.00	0.00	0.00	0.00	0.00	
	S	0.00	0.00	0.00	0.00	0.00	
	M	0.01	0.05	0.00	15.15	0.00	Total
	Total	0.01	0.05	0.00	61.31	0.10	61.47
Raytrace	NP	0.00	0.00	93.51	108.40	0.18	
	I	0.00	0.00	0.00	23.82	0.00	
	E	0.00	0.00	0.00	0.00	0.00	
	S	0.00	0.00	0.00	0.00	1.74	
	M	3.91	0.01	0.00	20.79	0.00	Total
	Total	3.91	0.01	93.51	153.01	1.92	252.36

Cuantificación de rendimiento

- **Bytes per Kref**

		NP	I	E	S	M	
Ocean	NP	0.00	0.00	87.39	66.96	117.51	
	I	0.00	0.00	0.00	130.73	0.11	
	E	0.00	0.00	0.00	0.00	0.00	
	S	0.00	0.00	0.00	0.00	13.44	
	M	183.81	0.11	0.00	160.97	0.00	Total
	Total	183.81	0.11	87.39	358.66	131.05	761.01
Radix	NP	0.00	0.00	0.33	246.73	798.78	
	I	0.00	0.00	0.00	77.57	320.51	
	E	0.00	0.00	0.00	0.00	0.00	
	S	0.00	0.00	0.00	0.00	4.91	
	M	3.10	807.19	0.00	282.21	0.00	Total
	Total	3.10	807.19	0.33	606.50	1124.19	2541.31

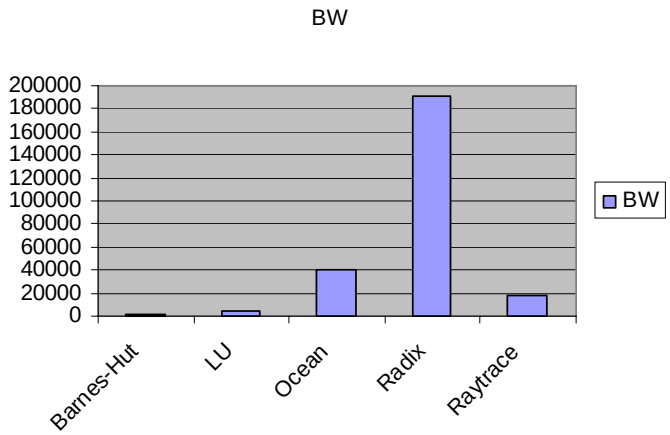
Cuantificación de rendimiento

- **Bandwidth**

	Instructions (M)	FLOPS (M)	Refs (M)	Reads (M)	Writes (M)	Bytes per Krefs	BW (KB/s)
Barnes-Hut	2002.74	239.24	720.13	406.84	313.29	25.78	1853.7
LU	489.52	92.20	151.07	103.09	47.99	61.47	3793.8
Ocean	376.51	101.54	99.70	81.16	18.54	761.01	40303.4
Radix	14.02	-	5.27	2.90	2.37	2541.31	191051.7
Raytrace	833.35	-	290.35	210.03	80.31	252.36	17585.1

$$BW = MIPS * \frac{Rfs}{Instrs} * BytesPerRf$$

Cuantificación de rendimiento



Cuantificación de rendimiento

- **How much bandwidth saves the MESI vs MSI?**
  - Transitions

	NP	I	S	M
NP	=	=	NP->E + NP->S	=
I	=	=	I->E + I->S	=
S	E->NP + S->NP	E->I + S->I	E->E + E->S + S->S	E->M + S-> M
M	=	=	=	=

  - Transactions per transition

	NP	I	S	M
NP	-	-	BusRd	BusRdX
I	-	-	BusRd	BusRdX
S	-	-	-	BusUpgr
M	BusWB	BusWB	BusWB	-

Cuantificación de rendimiento

- **Cuantos procesadores pueden ponerse en un bus de 1GB/s?**
- **Y si el procesador hace 1000MIPS?**

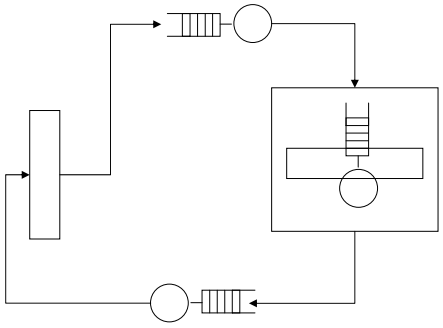
Cuantificación de rendimiento

- **Que rendimiento puede esperarse si:**
  - Procesador 1GHz , IPC 1 con memoria ideal
  - Ciclo de bus 10 ns; BW: 1.6 GB/s
  - Servicio de memoria 100ns
  - 1, 4 procesadores

- **1 procesador**
  - Sin contención
    - ✓ T_acceso_memoria: 10 + 100 + 10
    - ✓ T_instruccion: 1 + %refs * miss_ratio * T_acceso_memoria
    - ✓ MIPS = 1 / T_instruccion

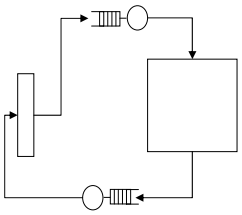
Modelización

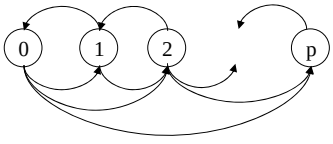
- **N procesadores**
  - Contención
    - ✓ Bus
    - ✓ Memoria



Modelización

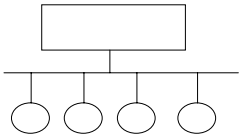
- **Contención**
  - Tiempos de servicio: ~ constantes !!
    - ✓ Scpu = #Instr / #fallos * T_ciclo
    - ✓ Sbus = 10
    - ✓ Smem = 100
  - Alternativas:
    - ✓ Simulación
    - ✓ Modelo de colas analítico considerando exponencial
    - ✓ Cadena de Markov
    - ✓ Modelo aproximado

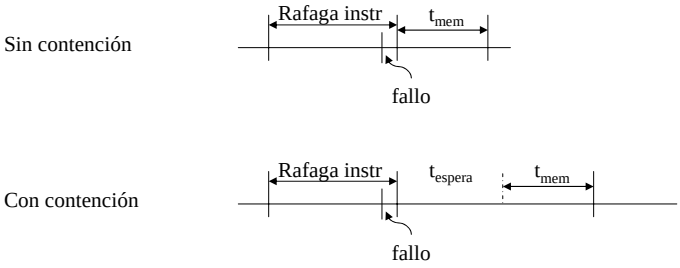




Modelos analíticos aproximados

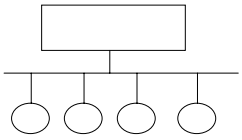
- **Contención**
  - Funcionamiento

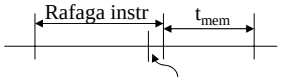




Modelos analíticos aproximados

- **Contención**
  - Modelos analíticos





✓ r = probabilidad petición en un ciclo = $t_{mem}/(1/m + t_{mem})$

Fallos/instruccion

✓ BW = $1 - (1-r)^p$

Ninguno pida

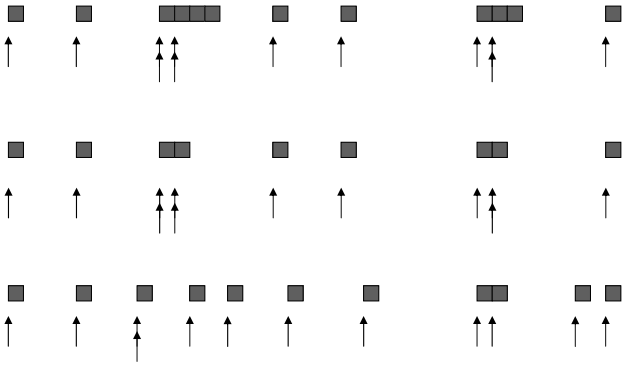
Uno no pida

Alguno pida ➔ uno consigue

Infravalora la demanda real
Cambia patrón petición

Modelos analíticos aproximados

- **Contención**
  - Modelos analiticos



Sistema Real

Inflavaloración demanda

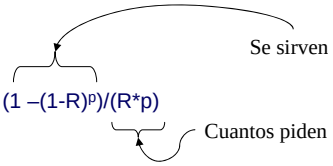
Patrón mas uniforme

Modelos analíticos aproximados

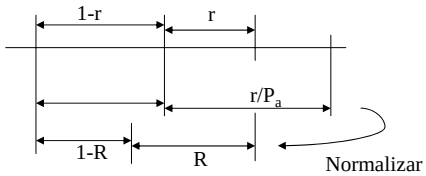
- **Contención**
  - Infravaloración demanda ➔
    - ✓ Usar r mayor. Cuanto?
      - – Modelos iterativos
- **BW ➔ rendimiento?**

Modelos analíticos aproximados

- **Contención**

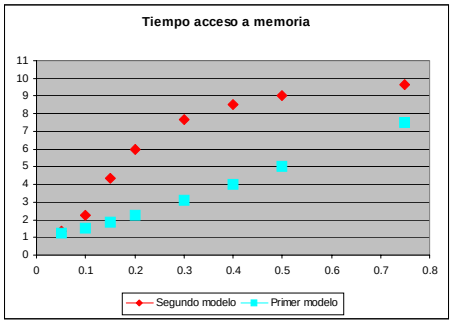


  - ✓ P_a = probabilidad aceptación = $(1 - (1-R)^p)/(R*p)$
  - ✓ T'_{mem}=tiempo respuesta memoria = $1/P_a$
  - ✓ Si no es aceptada se mantiene
    - R | $R = \frac{r/Pa}{(1-r)+r/Pa} = \frac{1}{1+Pa(1/r-1)}$
  - ✓ Iterar entre P_a y R



Modelos analíticos aproximados

- **10 procesadores**



Modelos analíticos aproximados

- Como se modificaría el modelo si hubiera 2 módulos de memoria?

Backfilling

- Que planificación generaría Backfilling en una máquina con 8 procesadores para la siguiente secuencia de peticiones?

  A: (4,3), B: (3,5), C: (4,2), D: (1,1), E: (2,1), F: (1,1), G: (3,8), H: (2,5), I: (4,3)

  donde los pares anteriores representan (numero de procesadores, tiempo de ejecución).

Backfilling

A: (4,3), B: (3,5), C: (4,2), D: (1,1), E: (2,1), F: (1,1), G: (3,8), H: (2,5), I: (4,3)

A	A	A	C	C	E					I	I	I
A	A	A	C	C	E					I	I	I
A	A	A	C	C	G	G	G	G	G	G	G	
A	A	A	C	C	G	G	G	G	G	G	G	
B	B	B	B	B	G	G	G	G	G	G	G	
B	B	B	B	B	H	H	H	H	H	I	I	I
B	B	B	B	B	H	H	H	H	H	I	I	I
D	F											

Computation-communication ratio

- **FFT**
  - Tamaño del vector: n
  - Algoritmo:
    - ✓ log n pasos. Cada paso totalmente paralelo. Secuencial entre pasos
    - ✓ Lee y escribe cada dato (8 bytes) log n veces en total
    - ✓ 5n log n operaciones coma flotante en total
- **SMP: Ancho de banda que consumen 16 procesadores a 250MFLOPS ?**
- **NUMA(p procesadores): log n/p de los pasos acceden a datos locales. El resto (log p) hacen la mitad de sus lecturas y escrituras remotas. BW consumido por 16 procesadores a 250MFLOPS ?**

Computation-communication ratio

- **Que rendimiento (MFLOPS) equivalente para toda la ejecución obtendremos de los 16 procesadores si BW=1.6GB/s). Suponer n=2^{14})**

Computation-communication ratio

- **SMP**
  - ✓ log n pasos, 5n log n ops ➔ 5n ops/paso ➔ 5 ops/dato
  - ✓ log n accesos lectura y escritura a dato ➔ 1 acceso R y W / paso
  - ✓ 250MFLOPS/5 ➔ 50Mloads y stores/s
  - ✓ 50M * 8 * 2 * 16 = 12800MB/s

Computation-communication ratio

- **NUMA:**
  - ✓ Los pasos con acceso local no consumen BW
  - ✓ Los pasos con acceso remoto la mitad ➔ 6400MB/s

Computation-communication ratio

- **Velocidad equivalente:**
  - ✓ Pasos locales: 250MFLOPS
  - ✓ Pasos remotos: 1.6GB/s = ¼ * 6400MB/s ➔ 62.5 MFLOPS
  - ✓ $n=2^{14}$, p=16 ➔10 pasos locales, 4 globales
  - ✓ Si t(paso local)=1 => t(paso global)=4
  - ✓ T(total) = 1*10 + 4*4 = 26
  - ✓ Velocidad equivalente: 250*14/26 = 134.6 MFLOPS

Link bandwidth

- **Formato paquete:**
  - 10 bytes routing y control
  - Payload: 64 bytes línea de cache + 8 bytes comando y @
  - 6 bytes CRC y trailer
- **Raw link BW: 500MB/s**
- **Effective BW?**
- **Si linea 32 bytes? 128 bytes? 4KB?**

Switching strategy

- **Red**
  - ✓ Link: 1 byte, 300 MHz
  - ✓ Distancia media: $\log_4 P$
  - ✓ Paquete 80 Bytes
  - ✓ Retardo por switch: 4 ciclos
- **Latencia con store and forward y cut-through**
  - ✓ P=16
  - ✓ P=1024

Traspuesta

- Matriz nxn por filas en P procesadores
- Cuantos datos cruzan la bisección al trasponer ?
- Escribir un programa de trasposición de una matriz en MPI.

MPI + OpenMP

- Paralelizar OpenMP, MPI, mixed

```
DO J=1,N
   X(J)=B(J)/A(J,J)
   DO I=J+1,N
      B(I)=B(I)-A(I,J)*B(J)
   ENDDO
ENDDO
```

MPI

- **Paralelizar MPI**

```
# define VECES 100
double A[1024,1024],B[1024,1204];
for (ext=0; ext<VECES; ext++)
   for (i=1;i<1024;i++)
      for (j=0;j<1024;j++) {
         A[i,j]= f(A[i-1,j-1],A[i,j],B[i,j]);
}
```

- **Modelo de rendimiento**
  - f: 1 us
  - BW: 8 MB/s
  - T(P) ? P=10,100,1000 sin contención
  - T(P) con bus?

OpenMP

- **Dado**

```
#define N 4096
double A[N],B[N];
for (ext=0; ext<N; ext++)
#pragma omp parallel for
   for (i=ext;i<N;i++) {
   A[i]= A[i]+B[i];
}
```

- **Modelo de rendimiento**
  - Sizeof(double)=8; línea 128B; cache 4MB
  - Invalidaciones? Inv(P=8) ?
  - Fallos de cache? M(P=8) ?

OS page mapping

- NUMA, 128 CPUS, 256MB memory modules
- First touch vs interleaved?

VERSION 1

```
#define N 16*1024*1024
int i,its;
double A[N];
#pragma omp parallel for
for (i=0;i<N;i++) A[i]=0;
for(its=0;its<1000;its++){
#pragma omp parallel for
   for (i=0; i<N; i++)
       A[i]= f(A[i]);
}
```

VERSION 2

```
#define N 16*1024*1024
int i,its;
double A[N];
for (i=0;i<N;i++) A[i]=0;
for(its=0;its<1000;its++){
#pragma omp parallel for
   for (i=0; i<N; i++)
       A[i]= f(A[i]);
}
```


OS page mapping

- Comportamiento esperable?

	V1	V2
First touch	Local accesses	Master concentrates data. Remote accesses Contention
Interleaved	Spread data Remote accesses Not much contention	Spread data Remote accesses Not much contention

- Mejorable Interleaved-V1?
  - Coordinar scheduling con placement.
    - ✓ # elementos por página ➔ STATIC(chunksize)
    - ✓ Alineamiento vector a página y en primer procesador

Bus based SMP

- **Modify State transition diagram for write through**
- **Transition M→I when receiving BusRd ?**

Microbenchmarks

- **Memoria: diseñar microbenchmarks para medir**
  - Tiempo de acceso a memoria de un procesador
  - Ancho de banda máximo
  - Efecto de contención en la red de interconexión
- **Comunicación: diseñar distintas versiones de microbenchmarks para medir el ancho de banda de MPI**
-

Microbenchmarks

- **Memoria**

```
for (i=0; I <N; i+=B)
   s+= a[i];
```

- BW … max?

Microbenchmarks

- **Memoria**

```
for (i=0; I <N; i+=B)
   s+= a[i];
```

- BW … max?

```
for (i=0; I <N; i++)
   p=p->next;
```

- Back-to-back latency

Microbenchmarks

- **Memoria**

```
for (i=0; I <N; i+=B)
   s+= a[i];
```

- BW … max?

```
for (i=0; I <N; i++)
   p=p->next;
```

- Back-to-back latency

```
for (i=0; I <N; i++){
   p=p->next;
   for (j=1;j<J;j++)
      s+=1;
}
```

- True unloaded latency

Microbenchmarks

- **Memoria**
  - Dependencia del estado de la línea
    - ✓ Ejemplo: medir coste acceso para escritura a una línea en estado Shared en todos los procesadores

Microbenchmarks

- **Comunicación**
  - Ping pong
  - Unidireccional
  - Contención ?

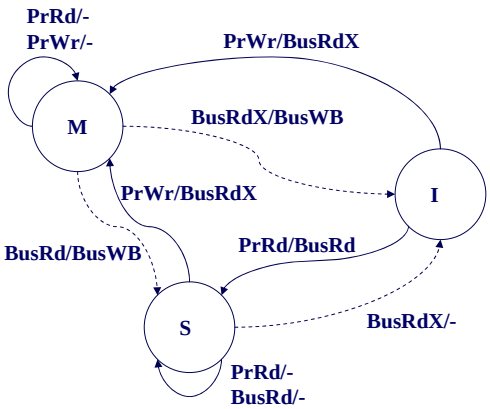
Sort

- **Hacer un algoritmo de ordenación paralelo**

- **Memoria**
  - Modulos independientes para soportar mayor ancho de banda
  - Mapeo:
    - ✓ Secuencial
    - ✓ Entrelazado
      - – Lineas
      - – Páginas, ….
    - ✓ K-entrelazado

MSI

- **Notation controller state transition**
  - command (form processor or bus) / Bus transaction
- **States**
  - Invalid
  - Shared
  - Modified



Another example: Radix sort

```
sweep(KP* VOrig, KP* VDest, int size, int offset)
 {
   int C[MAXBYTE+1], tmp, acum, valT, val;
   int i, j, pos;

   for (i=0; i<=MAXBYTE; i++) C[i] = 0;

   #pragma omp parallel private(val, acum, valT, pos)
   {
      int CLoc[MAXBYTE+1];
      for (i=0; i<=MAXBYTE; i++) CLoc[i] = 0;

      #pragma omp for schedule(static)
      for (i = 0; i < size; i++) {
         val = (VOrig[i].k>>offset) & MAXBYTE;
         CLoc[val]++;
      }

      #pragma omp critical
      for (i=0; i<=MAXBYTE; i++) C[i] += CLoc[i];
```

```
      #pragma omp master
         {
            tmp = C[0]; C[0] = 0;
            for (i=0; i<MAXBYTE; i++) {
               acum = tmp + C[i];
               tmp = C[i+1];
               C[i+1] = acum;
            }
         }
      }

      #pragma omp parallel private(valT, pos)
      {
         #pragma omp for ordered schedule(static)
         for (i = 0; i < size; i++) {
            #pragma omp ordered
            {
               valT = (VOrig[i].k >> offset) & MAXBYTE;
               pos = C[valT];
               C[valT]++;
               VDest[pos].k = VOrig[i].k;
            }
         }
      }
   }
```

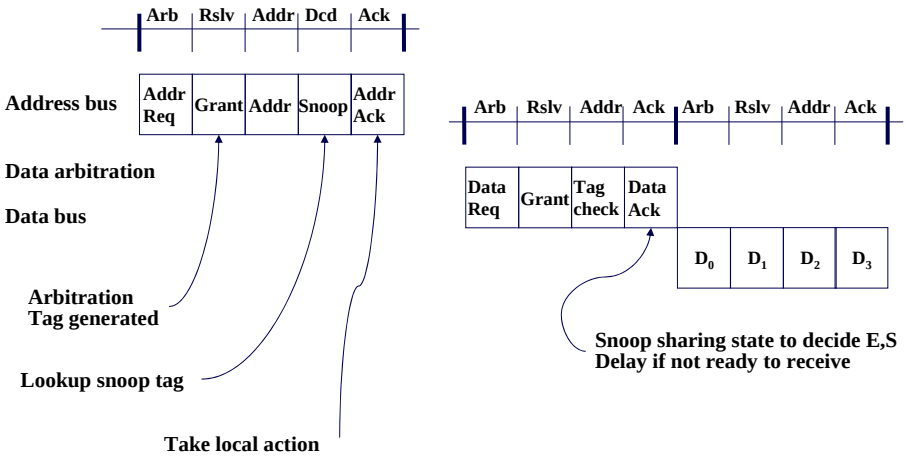

Another example: Radix sort

```
sweep(KP* VOrig, KP* VDest, int size, int offset)
 {
   int C[MAXBYTE+1], tmp, acum, valT, val;
   int i, j, pos;

   for (i=0; i<=MAXBYTE; i++) C[i] = 0;

   #pragma omp parallel private(val, acum, valT, pos)
   {
      int CLoc[MAXBYTE+1];
      for (i=0; i<=MAXBYTE; i++) CLoc[i] = 0;

      #pragma omp for schedule(static)
      for (i = 0; i < size; i++) {
         val = (VOrig[i].k>>offset) & MAXBYTE;
         CLoc[val]++;
      }

      #pragma omp critical
      for (i=0; i<=MAXBYTE; i++) C[i] += CLoc[i];
```

```
      #pragma omp master
         {
            tmp = C[0]; C[0] = 0;
            for (i=0; i<MAXBYTE; i++) {
               acum = tmp + C[i];
               tmp = C[i+1];
               C[i+1] = acum;
            }
         }
      }

      #pragma omp parallel private(valT, pos)
      {
         #pragma omp for ordered schedule(static)
         for (i = 0; i < size; i++) {
            #pragma omp ordered
            {
               valT = (VOrig[i].k >> offset) & MAXBYTE;
               pos = C[valT];
               C[valT]++;
               VDest[pos].k = VOrig[i].k;
            }
         }
      }
   }
```

- Earliest moment a write can proceed?
- Hardware support?

Snooping @ Split transaction bus



Take local action
Drive snoop lines: tell memory whether it has to respond or not
Delay if not time to check/NACK
Note if response needed
Withdraw request if conflicting with previous

- **Describir que tipos de hilos (y cuantos de cada tipo) pondrías en un bus para soportar la conexión de 8 procesadores a un máximo de 8 GB de memoria física con líneas de 128 bytes y el protocolo MESI de coherencia. Suponer que el bus no esta segmentado.**
  - Dirección: 26 (33 correpondiendo a 8GB – 7 de desplazamiento dentro de la línea). Activados por los procesadores.
  - Datos: 128 para la línea entera.
  - Arbitraje: 8 de petición (procesador ➔ árbitro del bus); 8 de respuesta.
  - Comando: 2 : Codificar ReqRd, ReqRdX, WriteBack.
  - Snoop:
    - ✓ 1 or cableada: cada cache indica si tiene la línea.
    - ✓ 1 or cableada: cada cache indica si la tiene en modo escritura.
  - Sincronización
  - 1 and cableada indicando si se ha podido comprobar el snoop o es necesario alargar el ciclo.