OpenMP

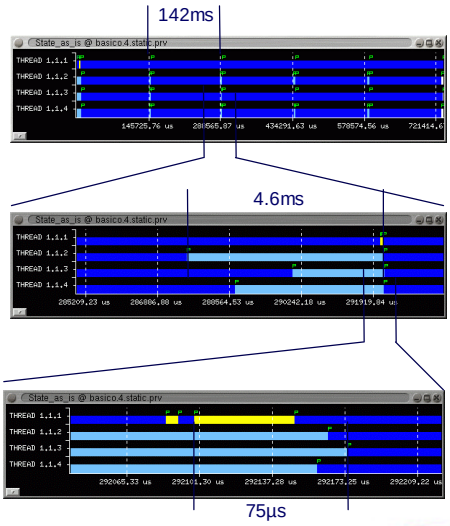
- **Introducción**
- **Directivas**
  - ✓ Regiones paralelas
  - ✓ Worksharing
  - ✓ sincronizaciones
  - ✓ Visibilidad datos
- **Implementación**

OpenMP: introduction

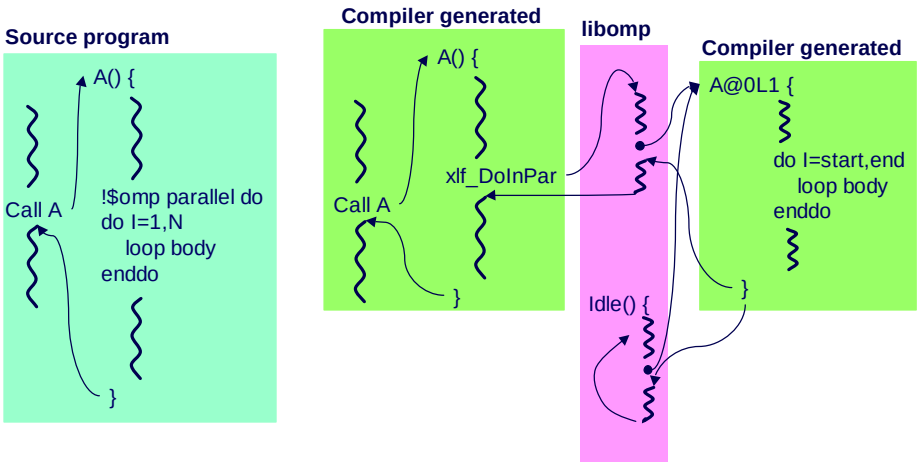
- **Standard promoted by main manufacturers**
  - http://www.openmp.org
  - Fortran
    - ✓ V1.0: Oct. 1997
    - ✓ V2.0: Nov. 2000
  - C
    - ✓ V1.0: Oct. 1998
    - ✓ V2.0: March 2002
  - Structure: Directives, clauses and run time calls
- **Manged by the ARB (Architectural Review Board)**
  - Companies: Intel, SGI, KAI, IBM, Compaq, HP, Fujitsu, …
  - Committees Fortan, C, Futures,…
  - Teleconferences

Basic example

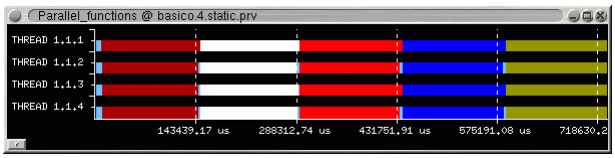
```
      PROGRAM test
      PARAMETER (N=1024)
      REAL dummy(N), factor, var
      REAL a(64000)
      INTEGER i,j,time
      common /varios/a

      factor=1/1.0000001
      time = 10

      DO 150 iter=1,5
C$OMP PARALLELDO SCHEDULE(STATIC)
C$OMP SHARED(dummy) PRIVATE(I,J)
      DO 200 i=0,N
         dummy(i)= dummy(i)*factor
         call delay(time)
200   CONTINUE
150   CONTINUE

      END
```



OpenMP compilation and Run Time



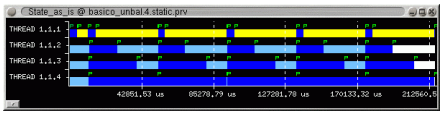
OpenMP outlined routines

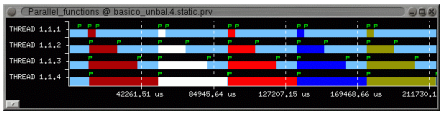


- Display of the identifier (color encoded) of the outline routine being executed
- The compiler unrolls the iter loop !!

Unbalanced loop

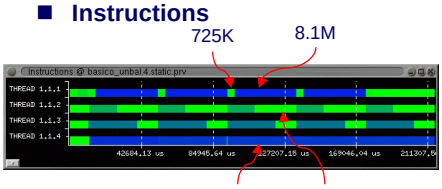
```
      PROGRAM test
      PARAMETER (N=1024)
      REAL dummy(N), factor, var
      REAL a(64000)
      INTEGER i,j,time
      common /varios/a

      factor=1/1.0000001

      DO 150 iter=1,5
C$OMP PARALLELDO SCHEDULE(STATIC)
C$OMP SHARED(dummy) PRIVATE(I,J,time)
      DO 200 i=0,N
         dummy(i)= dummy(i)*factor
         time = i/100
         call delay(time)
200   CONTINUE
150   CONTINUE

      END
```



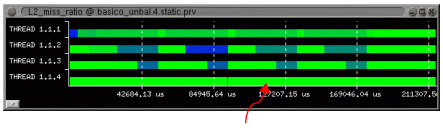


Performance indices

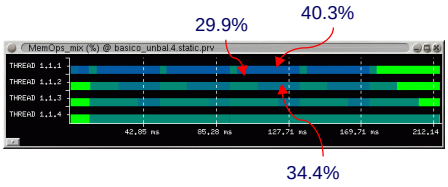
- **Instructions**



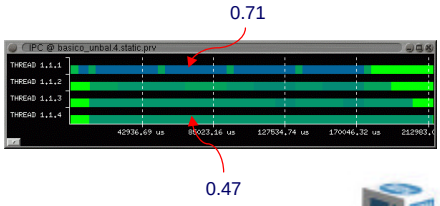
- **L2 miss ratio**



- **Mem Ops mix**

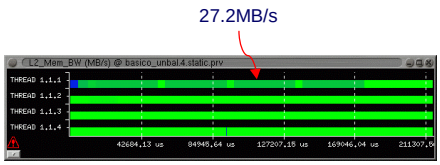


- **IPC**

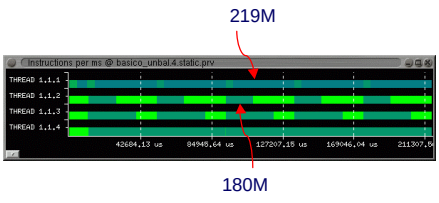


Performance indices

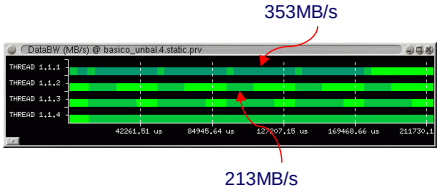
- **Memory Bandwidth**



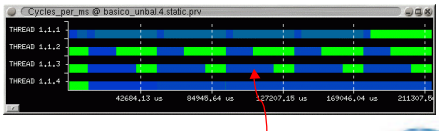
- **MIPS**



- **CPU Bandwidth**

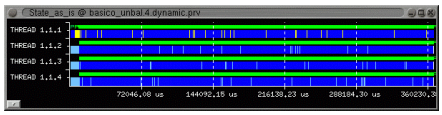


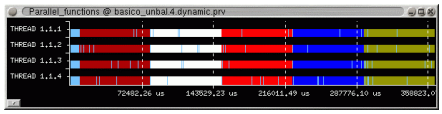
- **Cycles/s**

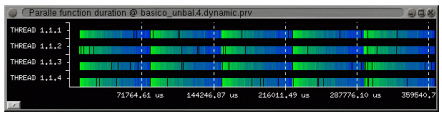


Dynamic scheduling

```
      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 150 iter=1,5
C$OMP PARALLELDO SCHEDULE(DYNAMIC)
C$OMP SHARED(dummy) PRIVATE(I,J,time)
      DO 200 i=0,N
         dummy(i)= dummy(i)*factor
         time = i/100
         call delay(time)
200   CONTINUE
150   CONTINUE

      END
```

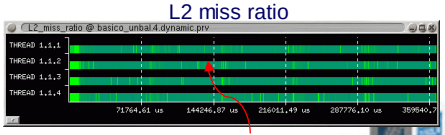






Outline routine duration

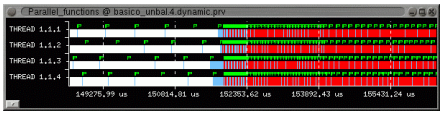
L2 miss ratio

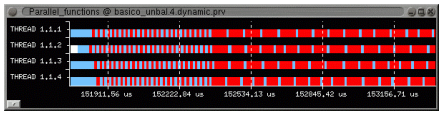


Avg. 2.3%

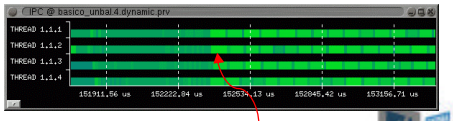
Dynamic scheduling

```
      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 150 iter=1,5
C$OMP PARALLELDO SCHEDULE(DYNAMIC)
C$OMP SHARED(dummy) PRIVATE(I,J,time)
      DO 200 i=0,N
         dummy(i)= dummy(i)*factor
         time = i/100
         call delay(time)
200   CONTINUE
150   CONTINUE

      END
```





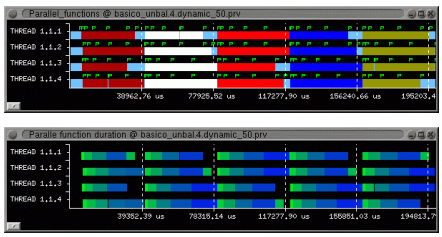
IPC



0.25

Coarser Gain Dynamic

```
      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 150 iter=1,5
C$OMP PARALLELDO SCHEDULE(DYNAMIC,50)
C$OMP SHARED(dummy) PRIVATE(I,J,time)
      DO 200 i=0,N
         dummy(i)= dummy(i)*factor
         time = i/100
         call delay(time)
200   CONTINUE
150   CONTINUE

      END
```

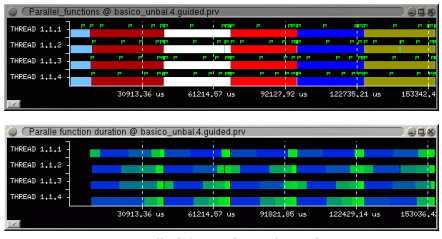


Parallel function duration

- **Less overhead**
- **Some unbalance:**
  - Heavy chunks towards the end

Guided Scheduling

```
      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 150 iter=1,5
C$OMP PARALLELDO SCHEDULE(GUIDED)
C$OMP SHARED(dummy) PRIVATE(I,J,time)
      DO 200 i=0,N
         dummy(i)= dummy(i)*factor
         time = i/100
         call delay(time)
200   CONTINUE
150   CONTINUE

      END
```

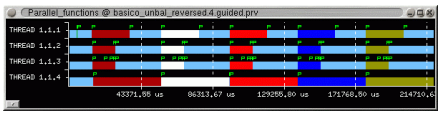


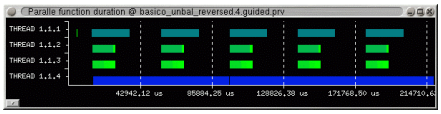
Parallel function duration

- **Less overhead**
- **Good load balance:**
  - Heavy chunks towards the beginning
- **Dynamic:**
  - Non repetitive pattern

Always possible to fool a schedule

```
      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 150 iter=1,5
C$OMP PARALLELDO SCHEDULE(GUIDED)
C$OMP SHARED(dummy) PRIVATE(I,J,time)
      DO 200 i=0,N
         dummy(i)= dummy(i)*factor
         time = (N-i)/100
         call delay(time)
200   CONTINUE
150   CONTINUE

      END
```



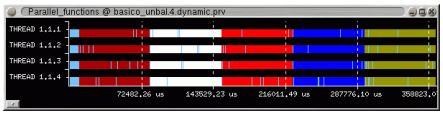


Parallel function duration

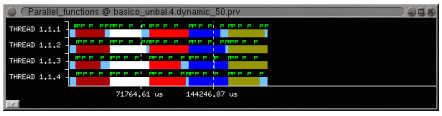
- **Dynamic:**
  - Repetitive pattern (not enforced)

Comparison

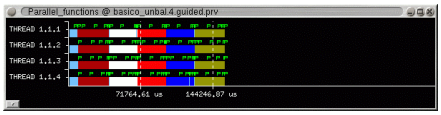
- **Dynamic**



- **Dynamic,50**



- **Guided**

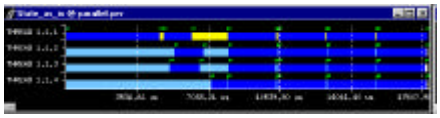


Same scale

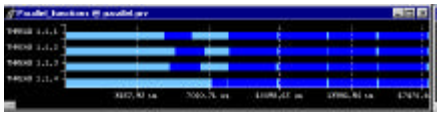
Back to basics: Parallel directive

```
      DO iter=1,5

C$OMP PARALLEL PRIVATE(time)
      time = 10
      call delay(time)
C$OMP ENDPARALLEL

      ENDDO
```

State



Parallel functions



- **Team**
  - Threads to work on a parallel
- **Parallel**
  - All threads execute the body

Worksharings

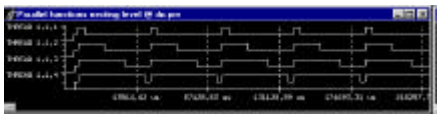
- **Directives within parallel**
- **Switch from replicated work to partitioned work**
- **Work sharing constructs**
  - loops
    - ✓ `C$OMP [END] DO [clause[[,] clause]…]`
    - ✓ `C$OMP [END] PARALLEL DO …`
  - sections
    - ✓ `C$OMP [END] SECTIONS [clause[[,] clause]…]`
    - ✓ `C$OMP SECTION`
    - ✓ `C$OMP [END] PARALLEL SECTIONS …`

Do

```
      DO iter=1,5

C$OMP PARALLEL PRIVATE(time)
      time = 50
      call delay(time)
C$OMP DO SCHEDULE (STATIC)
      DO I=1,N
         time = i/100
         call delay(time)
      ENDDO
C$OMP ENDPARALLEL

      ENDDO
```

Parallel function

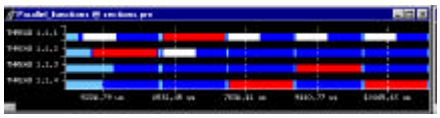
Parallel function nesting level



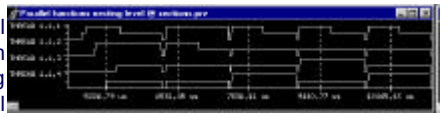
Sections

```
      DO iter=1,5

C$OMP PARALLEL PRIVATE(time)
C$OMP SECTIONS
C$OMP SECTION
      time = 30
      call delay(time)
C$OMP SECTION
      time = 60
      call delay(time)
C$OMP END SECTIONS
C$OMP ENDPARALLEL

      ENDDO
```

Parallel function



Parallel function nesting level



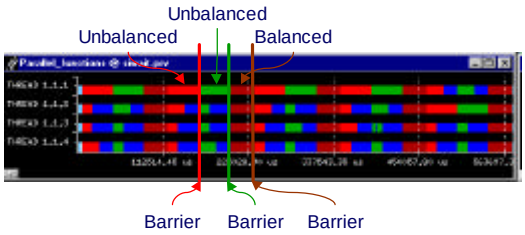
- **Dynamic:**
  - Sections randomly taken by threads

Shortcuts

- **Parallel do**
  - Shortcut for do within parallel
  - Allows more efficient implementation
- **Parallel sections**
  - Shortcut for sections within parallel
  - Allows more efficient implementation

Relaxing synchronizations

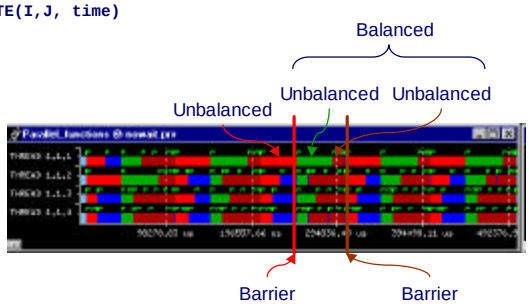
```
      DO iter=1,5
C$OMP PARALLEL SHARED(dummy) PRIVATE(I,J, time)
C$OMP DO SCHEDULE(GUIDED)
      DO i=0,N
         time = (N-i)/100
         call delay(time)
      ENDDO
C$OMP DO SCHEDULE(GUIDED)
      DO i=0,N
         time = (N-i)/100
         call delay(time)
      ENDDO
C$OMP ENDDO
C$OMP DO SCHEDULE(GUIDED)
      DO i=0,N
         time = i/100
         call delay(time)
      ENDDO
C$OMP END PARALLEL
      ENDDO
```



- **Globally**
  - 2 unbalanced loops

Relaxing synchronizations: Nowait

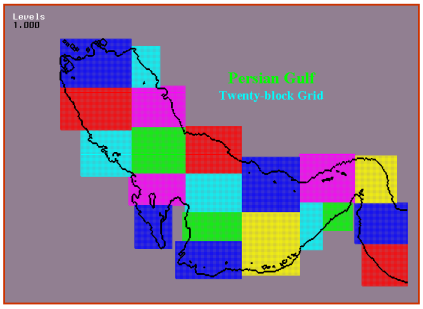
```
      DO iter=1,5
C$OMP PARALLEL SHARED(dummy) PRIVATE(I,J, time)
C$OMP DO SCHEDULE(GUIDED)
      DO i=0,N
         time = (N-i)/100
         call delay(time)
      ENDDO
C$OMP DO SCHEDULE(GUIDED)
      DO i=0,N
         time = (N-i)/100
         call delay(time)
      ENDDO
C$OMP ENDDO NOWAIT
C$OMP DO SCHEDULE(GUIDED)
      DO i=0,N
         time = i/100
         call delay(time)
      ENDDO
C$OMP END PARALLEL
      ENDDO
```

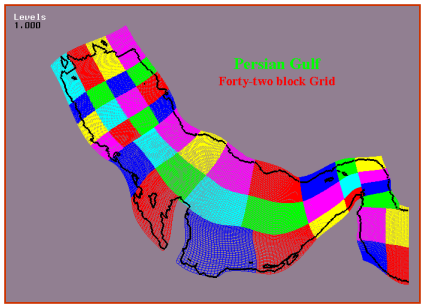


- **Globally**
  - Achieved global balance

SPMD parallelization

- **Very coarse grain**
  - Similarity to MPI
- **Need for Barrier**





Malleability

Scheduling decisions: Once for all

Explicit code only related to parallelism

```
C$OMP PARALLEL
WhoAmI=RunTimeCall()
myBlock=f(WhoAmI)
…
Call Compute1(myBlock)
…
DO iters=1, #iters
   Call Compute2(myBlock)
END DO
…
C$OMP END PARALLEL
```

```
C$OMP PARALLEL DO
DO Block=1, #blocks
 Call Compute1(Block)
ENDO
C$OMP END PARALLEL
…
DO iter=1,#iters
   C$OMP PARALLEL DO
   DO Block=1, #blocks
      Call Compute1(Block)
   ENDO
   C$OMP END PARALLEL
END DO
…
C$OMP END PARALLEL
```

myBlock Î [1,#processors] ¬ f(resources)
Block Î [1,#blocks] ¬ f(algorithm)

Other computation patterns

- **Master**
  - C$OMP [END] MASTER
  - Only master threads executes. All other wait
- **Single**
  - C$OMP [END] SINGLE [clause[[,] clause]…]
  - One thread executes
  - Barrier at end
- **Ordered**
  - C$OMP [END] ORDERED
  - Ensures execution in sequential order

Other computation patterns

- **Critical**
  - `C$OMP [END] CRITICAL [(name)]`
  - Mutual exclusion
- **Atomic**
  - `C$OMP ATOMIC`
  - Atomicity of single assignment statement
    - ✓ Optimizable implementation
- **Barrier**
  - `C$OMP BARRIER`
  - All threads must reach it before continuing

Other computation patterns

- **Flush**
  - `C$OMP FLUSH [(list)]`
  - Enforce consistency
- **Lock routines:**
  - `OMP_INIT_LOCK (var)`
  - `OMP_DESTROY_LOCK (var)`
  - `OMP_SET_LOCK (var)`
  - `OMP_UNSET_LOCK (var)`
  - `OMP_TEST_LOCK (var)`

Single

```
      DO iter=1,5

C$OMP PARALLEL PRIVATE(time)
      time = 50
      call delay(time)
C$OMP SINGLE
      time = 25
      call delay(time)
C$OMP END SINGLE
      time = 10
      call delay(time)

C$OMP ENDPARALLEL

      ENDDO
```

Parallel function

Parallel function nesting level

- **Implementaion**
  - outlined call
  - Other alternatives possible
- **Duration**
  - Not proportional to time !!!
  - ???

Who/how many?

- **Run time calls to find out**
  - `OMP_GET_NUM_THREADS`
    - ✓ How many are we?
  - `OMP_GET_THREAD_NUM`
    - ✓ Who am I
- **… or set**
  - `OMP_SET_NUM_THREADS (expr)`
- **Advice: minimize their use if possible**
  - Avoid the MPI/SPMD approach
  - Write malleable codes

Data scope

- **`PRIVATE(list)`**
  - ˜ allocate local space on entry to outlined routine
  - Scalars / vectors
- **`FIRSTPRIVATE(list)`**
  - Allocate and initialize
- **`LASTPRIVATE(list)`**
  - Copy value of last iteration to global
- **`THREADPRIVATE(list)`**
  - Privatization of common blocks

Data scope

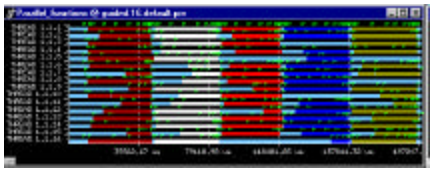
- **`REDUCTION(list)`**
  - Perform reduction operation

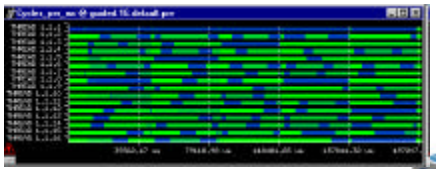
Run time library

- **Compute schedules**
- **Wait modes**
  - Overhead
  - Behavior under overcommitted multiprogrammed workload
  - Busy wait, yield, block
    - ✓ When to change mode?
- **Locks**
- **Synchronizations**

Run time library

- **Multiprogramming effects**
  - OMP_NUM_THREADS=16
  - Guided
  - Machine
    - ✓ 16 way SMP
    - ✓ load ≅ 9

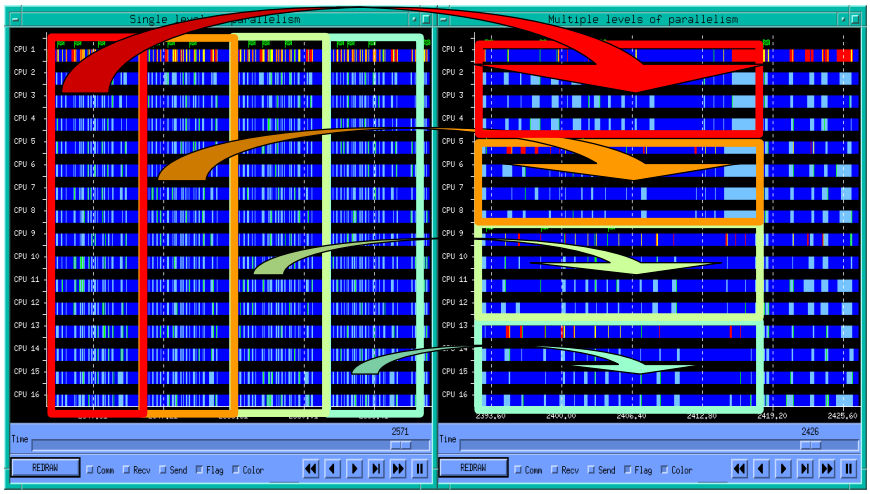




Other topics

- **Array worksharings**
- **Future topics**
  - Nested parallelism
  - Task queues
  - Precedences
  - Further schedules
  - Multiprogrammed workload management

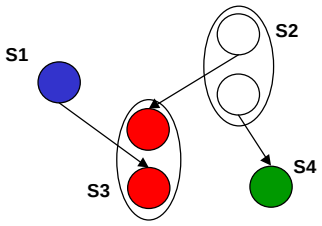
Nested parallelism



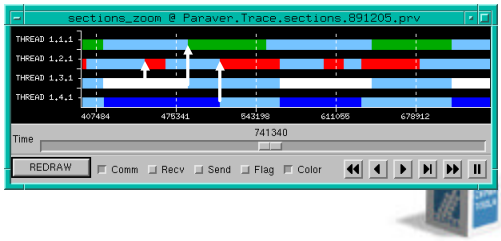
SPEC95 hydro2d

Single level precedences

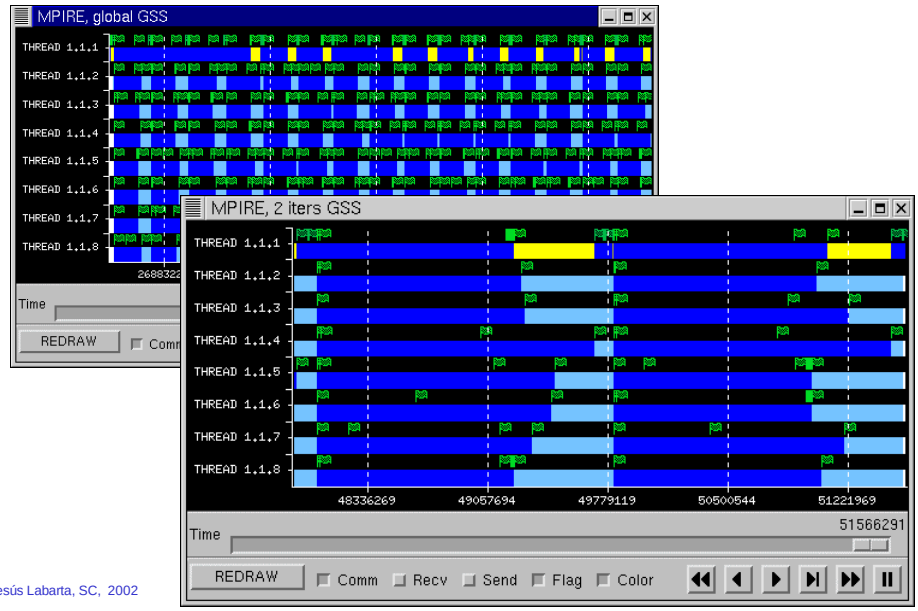
- **Sections**
  - PRED/SUCC at any point



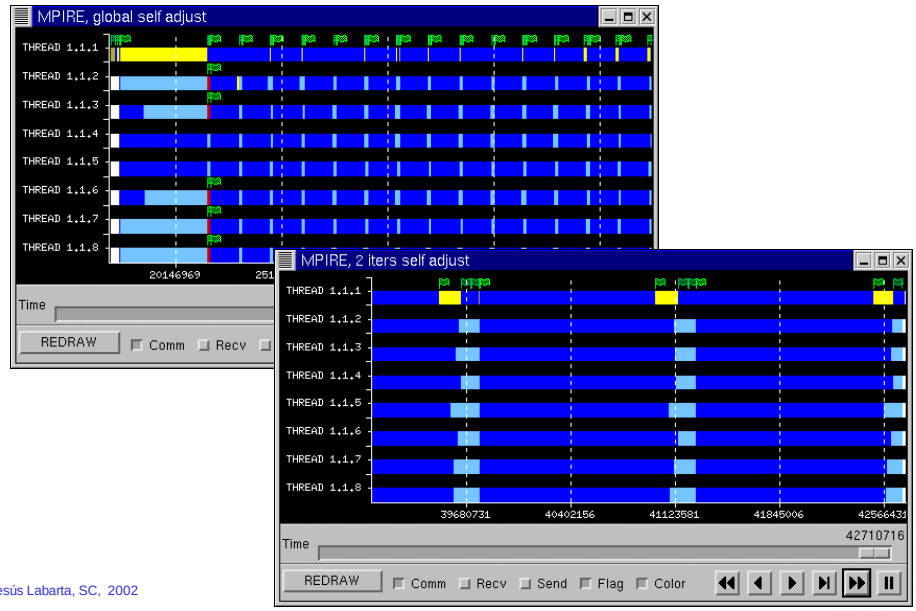

```
!$omp parallel sections
   !$omp section name (S1) succ (S3)
   !$omp section name (S2) succ (S4)
      !$omp succ (S3)
   !$omp section name (S3) pred (S2)
      !$omp pred (S1)
   !$omp section name (S4) pred (S2)
!$omp end parallel sections
```



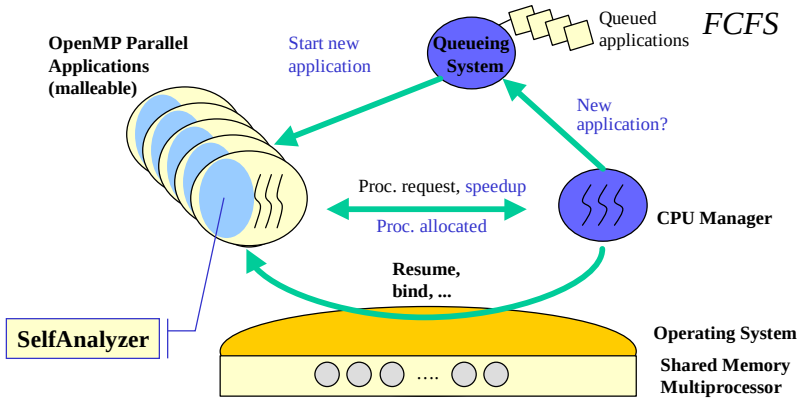
Loop scheduling: MPIRE



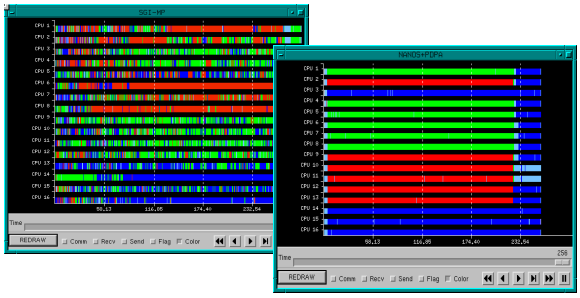
Loop scheduling: MPIRE



NANOS OS scheduling environment

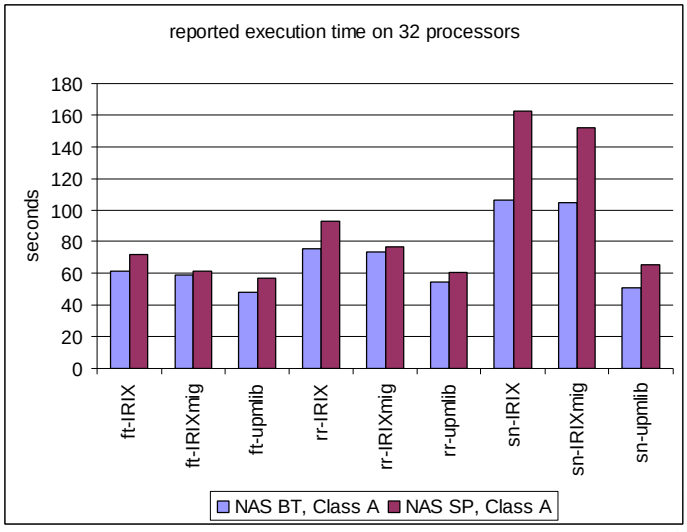


CPU scheduling & memory



- **What is the real difference ?**
  - Time
  - Context switch overhead
  - Migrations

Page migration



LHPCA