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 x(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

Source program

Call A

Compiler generated

A() {
  !$omp parallel do
do l=1,N
  loop body
enddo
}

Compiler generated

ibomp

Compiler generated

A@0L1 {
  do l=start,end
  loop body
enddo
}

Jesus Labarta, SC, 2002
OpenMP outlined routines

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

Unbalanced loop

```fortran
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
```

Jesús Labarta, SC, 2002
Performance indices

- **Instructions**
  - 725K
  - 8.1M
  - 7.3M
  - 28K

- **Mem Ops mix**
  - 29.9%
  - 40.3%
  - 34.4%

- **L2 miss ratio**
  - <0.1%

- **IPC**
  - 0.71
  - 0.47

Performance indices

- **Memory Bandwidth**
  - 27.2MB/s

- **MIPS**
  - 219M
  - 180M

- **CPU Bandwidth**
  - 353MB/s
  - 213MB/s

- **Cycles/s**
  - 375M
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$OMP PARALLEL DO SCHEDULE(DYNAMIC)
C$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

Jesús Labarta, SC, 2002

Outline routine duration
L2 miss ratio
Avg. 2.3%

IPC
0.25

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$OMP PARALLEL DO SCHEDULE(DYNAMIC)
C$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

Jesús Labarta, SC, 2002

IPC
0.25
### Coarser Gain Dynamic

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

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

### Guided Scheduling

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

- **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/
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

```fortran
DO iter=1,5
   C$OMP PARALLEL PRIVATE(time)
   time = 10
   call delay(time)
   C$OMP ENDPARALLEL
ENDDO
```

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

Work sharings

- **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

\[
\text{DO iter=1,5}
\]
\[
\text{C$OMP PARALLEL PRIVATE(time)}
\]
\[
\text{time} = 50
\]
\[
\text{call delay(time)}
\]
\[
\text{C$OMP DO SCHEDULE (STATIC)}
\]
\[
\text{DO i=1,N}
\]
\[
\text{time} = i/100
\]
\[
\text{call delay(time)}
\]
\[
\text{ENDDO}
\]
\[
\text{C$OMP END PARALLEL}
\]
\[
\text{ENDDO}
\]

Sections

\[
\text{DO iter=1,5}
\]
\[
\text{C$OMP PARALLEL PRIVATE(time)}
\]
\[
\text{C$OMP SECTIONS}
\]
\[
\text{C$OMP SECTION}
\]
\[
\text{time} = 30
\]
\[
\text{call delay(time)}
\]
\[
\text{C$OMP SECTION}
\]
\[
\text{time} = 60
\]
\[
\text{call delay(time)}
\]
\[
\text{C$OMP END SECTIONS}
\]
\[
\text{C$OMP END PARALLEL}
\]
\[
\text{ENDDO}
\]

- **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

```fortran
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$OMP END DO
  C$OMP DO SCHEDULE(GUIDED)
  DO i=0,N
    time = (N-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$OMP ENDDO NOWAIT
C$OMP DO SCHEDULE(GUIDED)
   DO i=0,N
      time = i/100
      call delay(time)
   ENDDO
C$OMP END PARALLEL

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

```
myBlock ∈ [1,#processors] ← f(resources)
Block ∈ [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[,...]]
  - 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

- 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)
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 END PARALLEL
ENDDO

- Implementation
  - 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

- **REDDUCTION(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**

![MPIRE screenshot](image-url)
Loop scheduling: MPIRE

NANOS OS scheduling environment
CPU scheduling & memory

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

Page migration

reported execution time on 32 processors

- NAS BT, Class A
- NAS SP, Class A