Parallelism

- Primitives for matrix manipulation hide all details of
  BLAS and LAPACK for linear algebra

- Primitives.

  Sequential code design with occasional calls to parallel
  and in concert.

There are several approaches which can be used individually
parallel codes on the SCI is fairly typical.
The current environment for implementing shared memory

200

Loop-based Parallel Programming on the SCI Origin
- Assumes the use of Fortran90
- No automatic dependence analysis assumed.
- Directive form used

standards such as OpenMP

(implicit/parallel programming via emerger

(semi)explicit parallel programming

codes

commercial/academic codes

- Paratrace, KAP, VAST and several other
- APO on SGI/CRAY machines
- Power Fortran and Power C on SGI's (now outmoded)

Directives plus an automatic parallelizing compiler
compiler.

They are specified as a command line parameter to the

possible: -00, -01, -02, -03, -0fast.
Several levels of code generation optimization are also

used by f90, f77, cc, & c89 compilers.

prefetch form, lock wait strategies, etc.

They include processor chip, cache miss penalties,

optimizing group

SGI machines describe these options

man into and the istinct online documentation on the

wants used to drive the code generation process

specific aspects of the architecture that the user

The SGI also allows several environment options to

A
i.e., obey the directives and use parallel libraries.

The `-mp` directive is required to generate parallel code.

- Effects of performance evaluation and controlling certain side

- 03 or 0Fast typically used. The others can be useful for
Chapter 5 is also important especially for Origin 2000

Chapter 3 gives a complete discussion of the various directives such as TNO.

Chapter 4 through 4.7, and 4.11 and 4.12.

Read all of the sections with particular attention to 4.3 directives.

Chapter 4 contains a good introduction to the OpenMP

Portran 90 Commands and Directives manual.

There are several manuals of interest but the descriptions of OpenMP and Portran 90 are taken from the MIPSpro
implemented using OPENMP and how.

We will now consider the concepts we discussed for

shared memory loop-based programming which are
A simple parallel do loop is specified as follows:

\[
q(\tau) = 3a(\tau)
\]

**do** \(\tau = 1, n\)**

\[
i$OMP END PARALLEL DO
\]

\[
i$OMP PARALLEL DO
\]
continuing sequentially.

A join is assumed after the loop with the master thread.

The setting of other environment or directive variables,
explicitly and therefore would have to be deduced from
above.

The scheduling mechanism has not been specified above
consume the iterations.

worker threads (think of them as processors) that
work in the parallel do is split across all of the
to the threads.

The iteration variable is implicitly assumed to be private

i$omp end parallel do

enddo

p(i) = 3*a(i)
do i=1,n

i$omp parallel do if(n .ge. 1000), private(i)

Several options can be specified explicitly.
variables that must have a private copy for each thread.

• _PRIVATE_ can be used to declare any number of variable private.

• The _PRIVATE_ clause explicitly declares the iteration parallelism is invoked only if the condition is true.

• The _IF_ clause gives a runtime condition that is evaluated.
• The number of threads can be specified dynamically or statically.

• **man omp_threads** gives details of routines and environment variables controlling Fortran, C, and C++.

• The environment variable **OMP_NUM_THREADS** sets the number of threads to be used by a code. This can be superseded within a code or changed dynamically over the life of a code.
specificed number is used as a maximum value.
set the number of threads as it sees fit and any user
default. When true the runtine environment may also
declared. It is recommended that this be set to false as a
dynamically.
set to true if the code is allowed to change the number of
The environment variable **OMP_DYNAMIC** must be
This is not the same as the number of threads.

- CALL OMP_GET.getNumProcs() returns the number
  returned when operating sequentially. master thread that spawned the parallelism and is also between 0 and OMP_GET.getNumThreads() - 1. 0 is the execution.

- CALL OMP_GET_NUM_THREADS() returns an integer
  the parallel code. It returns 1 if called during sequential number of threads currently used in the team executing returns the

- CALL OMP_GET_NUM_THREADS() returns the
  the number of threads.

There are several Fortran routines that are relevant to
environment variable OMP_DYNAMIC.
interpreted differently depending on the value of the
environment variable OMP_NUM_THREADS. It is
called OMP_SET_NUM_THREADS() overrides the
overrides the environment variable OMP_DYNAMIC.
called OMP_SET_DYNAMIC(TRUE, or FALSE).
continuous group of iterations sent to each thread.

Standard \texttt{STATIC} scheduling is used, with one
an interleaved fashion. If no chunk is specified then
scheduled. The pieces are assigned to the threads in
scheduled together into the basic piece of work that is
that specifies how many continuous iterations are
\texttt{SCHEDULE}(\texttt{STATIC}, \text{chunk}) - chunk is an integer
characterized.

In the \texttt{PARALLEL} loop directive the schedule can be

associated with scheduling and threads.

main environment discussion the environment variables

run-time.

Schedule of a loop can be specified at compile-time or
Scheduling of Iterations.

It will execute, this is effectively a {	exttt{doacross}} type
was assigned it dynamically determines the next piece
scheduled, after each thread finishes the last piece it
Grouped together into the basic piece of work that is
that specifies how many contiguous iterations are

\verb|SCHEDULE(DYNAMIC,chunk)| - chunk is an integer.
continues until the minimum chunk size is reached. This
halved as each new request for a piece is serviced. The
number of threads (like static scheduling) then it is
constant. It starts at the number of iterations over the
piece it will execute. However the size of the piece is not
that is scheduled. After each thread finishes the last
that are grouped together into the basic piece of work
specifies the minimum number of continuous iterations.

\[ \text{SCHEDULED} \] (GUIDEDchunk) – chunk is an integer that
A third mechanism is also allowed.
smaller tasks at the end.

beginning and then fill in load balancing problems with
granularity and reduce synchronization points at the
It uses a hybrid of static and dynamic ideas to increase

•
the use of the environment variables for scheduling.

The clause `SCHEDULE(RUNTIME)` is used to indicate

This is done via environment variables.

determine the schedule at run-time.

It is also possible to compile the code once and

The previous examples are all scheduled at compile time.
setenv OMP_SCHEDULE "GUIDED"

setenv OMP_SCHEDULE "DYNAMIC"

setenv OMP_SCHEDULE "STATIC,2"

setenv OMP_SCHEDULE "STATIC"

- It is set with a type and chunk.

- Determines the run-time scheduling decision.

The environment variable OMP_SCHEDULE
Compile-time:

```c
!$OMP PARALLEL DO IF(n.ge.1000), SCHEDULE(DYNAMIC,4)
    do i=1,n
        b(i) = 3*a(i)
    enddo
!$OMP END PARALLEL DO
```

Run-time:

```c
!$OMP PARALLEL DO IF(n.ge.1000), SCHEDULE(RUNTIME)
    do i=1,n
        b(i) = 3*a(i)
    enddo
!$OMP END PARALLEL DO
```