Nested parallelism can address this problem.

Note that if the number of sections is less than the number of threads, full parallelism is not achieved.

These are called sections.

Rather than have each thread get a set of iteration indices and execute the same iteration code body, it is possible to specify distinct code for each parallel task.

It is also possible to specify a simple form of inhomogeneous parallelism.
a significant difference code and independent as well.

function of temperature, density, enthalpy, entropy. 

• Evaluation the equation of state e.g. pressure as a

access the data structure appropriately.

• Independent of the others but requires distinct code to

• Evaluating a derivative in each axis direction is

parameters e.g. velocities, temperatures, densities.

Suppose we have a 3-D field of values of physical
i$omp end parallel sections

call eastate(field)

i$omp section

call zderiy(field)

i$omp section

call xderiy(field)

i$omp section

call xderiy(field)

i$omp parallel sections shared(field)
A SCHEDULE clause cannot be used.

• All of the data specification fields can also be used.

parallel execution.

• An IF clause may be specified to control conditional

• A join is assumed with the master thread continuing:

• Each section is executed by a thread in the team of

- threads consuming parallel work.
A barrier is assumed after the **END PARALLEL**.

All threads get the same code so it is homogeneous.

There are no iterations as in a do loop.

**PARALLEL** pair is executed by each thread.

The code specified by a **PARALLEL/END** section.

Version of the parallel do loop or a homogeneous form of

The final basic construct can be viewed as a noniterative

...
The following uses the static aspect of the assumed property.

- If clause can be specified.
- Data specification clauses can be used.
- No `SCHEDULE` clause is needed.

Evenly schedule and other calls to divide updates to an array.
$omp parallel

forall (i.am, i.inputs)

iomp_set_num_threads()

iomp_get_num_threads()

iomp_set_num_threads()

iomp_get_num_threads()

$$omp parallel default( privately(x, inputs) )$$

$$omp end parallel$$
master thread after the barrier is undone.
each thread is undone and their final value in the
type as declared earlier in the code. Their initial value in
private copy of the variable created. They have the same
PRIVATE(list) - This lists variables that are to have a
earlier in the code.
by all threads. They have the same type as declared
SHARED(list) - This lists variables that are to be shared

Insight Document: We review some basic ones here.
These are detailed in Section A.7.2 of the Fortran 90
control data layout and other aspects of the code.
The three constructs all have optional clauses that
threads' private to each thread, or none.
variables in the block of code to be shared between
PRIVATE or NONE. This sets the default scope of the
DEFUNLT((option) – The option can be SHARED'

PARALLEL pair.
PARALLEL/END
section. Not defined for a
would have executed last sequentially, or the last lexical
value after the join is set by the iteration of the loop that
LASTPRIVATE(Init) – Private variables whose final
that was operating sequentially.
taken to be the value of the variable in the master thread
PRIVATE except that the initial value in each thread is
FIRSTPRIVATE(Init) – Variables are the same as
OMP EBD PARALLEL DO
  enddo

  !$OMP PARALLEL DO
  j = j + 1

  !$OMP PARALLEL DO
  i = i + 1
  !$OMP PARALLEL DO
  x(k) = x(k) + j
  !$OMP END PARALLEL DO

  !$OMP PARALLEL DO
  k = k + 1
  !$OMP END PARALLEL DO

  do k=1,10
    !$OMP PARALLEL DO
    j = j + 1
    !$OMP END PARALLEL DO

    !$OMP PARALLEL DO
    i = i + 1
    !$OMP END PARALLEL DO

    integer i, j
    real x(10)
\begin{itemize}
  \item Value after do is unknown.
  \item \texttt{LASTPRIVATE}.
  \item Value after do is known to be 13 due to
  \item \texttt{FIRSTPRIVATE}.
  \item \texttt{i} initialized to 2 due to \texttt{FIRSTPRIVATE}.
\end{itemize}
values of master thread.

- \texttt{COPYIN} initializes selected variables and commons to
  
  - Initial state is undefined unless \texttt{COPYIN} clause used.

- \texttt{THREADPRIVATE} directive.
  
  - Separate copies of common blocks are indicated by

  - Initialization of commons therefore becomes a problem.

- Common blocks can be shared or private to threads.
i$OMP END PARALLEL

code segment

i$OMP PARALLEL DEFAULT(PRIVATE), COPYIN(blk1,z)

i$OMP THREADPRIVATE(blk1,fields)

common /fields/x,y,z

common /blk1/scratch
reductions –
locks –
barriers –
temporary sequential execution –
critical sections –

Synchronization must also be discussed •

When using multiple levels of granularity,
Nested parallelism must also be considered for efficiency •

more complicated worksharing:
used in concert with forms of the do and sections to get

The PARALLEL/END PARALLEL •
The constructs discussed so far are all single fork/joins. •