using It

- MPI Library and standard algorithms parallelized
- Message passing communication
- Influence of distinct address spaces

Distributed Memory Programming Paradigm

- Performance Limits
- Performance Prediction
- Programming Paradigm and SW support
- Architecture
- Vector/Data Parallel processors

So far,
Parallel algorithms

Parallel performance analysis

and library design

Parallel architectures and their influence on algorithm

OPENMP approach to shared memory loop-based code

Shared memory programming paradigm concepts

What's Next?
shared and internal variables

- Typically requires explicit specification of externally

- used at the highest level of parallelism

- has a large variable context

  large-grain unstructured macrotasks

  loops plus directives

  - explicit parallelism

  loop-based

- paradigm of shared memory machines:

There are four basic approaches to the programming

Shared Memory Programming Paradigm
independent data parallel computing, e.g., HPF, Fortran
aggregate data structure parallelism (essentially platform

- limited variable context – like loops
- unstructured control typically queue-based
medium-to-fine-structured microtasks
most often based on Posix threads.

Restructuring compiler translates code — recently this is

parallel version of the language into which the

Such systems almost always have a proprietary explicitly

...to indicate parallelism to a restructuring compiler

Most systems use Fortran 90 plus user-supplied directives

There is no accepted standard for loop-based parallel

Loop-based parallel programming


directive approach will then be discussed.

The main characteristics will be identified using a

constructs and related synchronization.

We will discuss some general parallel loop control

four SM approaches

Fortran dialects, e.g., Cedar Fortran (it contains all

There have been some attempts at explicitly parallel
Cedar team members.

(OPENMP was designed by a team that included

It cannot be used verbatim on any current commercial

These are based on the form that appeared in Cedar Fortran.

We will show forms of the DOALL that bring out its function.

(referred to as a pardo.

Typically referred to as a doall loop. Sometimes this is

The basic parallel loop encountered in the literature is
iterations.

do anything with respect to scheduling of the

dependences. The compiler and runtime systems can

completely independent – no loop-carried

In a DOALL the iterations are assumed to be

END DOALL

iteration body

LOOP

declarations and preambles

DOALL i = low, high, increment
any order.

- The iterations are independent and can be executed in any order.
- The output sets of the iterations are disjoint.
- The input sets of the iterations are disjoint.

Example 1

```plaintext
end do

(\tau) \lambda + (\tau) x = (\tau) x

u \in \tau \in \{0, 1, \ldots, 1000\}

double precision x(1000).\lambda(1000)

integer \tau, u

Sequential code:
```
end doall

\begin{align*}
(y^\top + (t)_x) x &= (t)_x \\
\text{do}\text{loop} \\
\text{doall } i = 1, \ldots, n \\
\text{u}=1000 \\
\text{integer } i, \text{u} \\
\text{parallel doall code:}
\end{align*}
such variables is a key task in a parallel implementation.
always implicitly done by the software. Identifying other
processor must have a local copy of i. This is
as a loop index. This is a special situation and each
Note the variable i is declared as an integer and is used
in.

any element of x and y. They all can access the variable
Shared memory is assumed so all processors can access
assertions assigned to a particular processor are executed.

• No assumptions are made about the order in which the iterations are executed on which processors.

• No assumptions are made about which iterations are

• Setting the local value of the iteration variable

• Assigning to processors (processes), i.e., scheduling and

• The system is assumed to take care of assigning
two different iterations.

There may not be a dependence between statements in

An iteration in a doall need not be so simple.

The basic parallel unit of computation in Example 1 was
Example 2

Sequential code:

```plaintext
DO I = 1, 100
  IF C(I) > 0 THEN
    B(I) = C(I) / (D(I-1) + A(I))
    A(I) = B(I) * C(I) + D(I)
  END IF
END DO
```
and iteration $j$ where $I \neq j$.

- There is no dependence between statements in iteration $I$ based on $I$.

- Available this dependence statement could be removed.

- If information on the values of the elements of $C$ were conservatively.

- Dependences for loop iterations are interpreted.

- Particular value of $I$ there may in fact be none.

- This is viewed as a dependence even though for a $2$ to statement $3$ for a fixed value of $I$.

- There is a how (true) dependence from statements $1$ and statement $2$ for a fixed value of $I$.

- There is a how (true) dependence from statement $1$ to
END DOALL

END IF

C(I) = A(I) * B(I)

THEN

IF C(I) > 0

B(I) = C(I) / (A(I) * D(I-I) + A(I))

A(I) = B(I) * C(I) * D(I)

LOOP

DOALL I = 1, 100

Parallel code:
dependencies within a single iteration (fixed index value)

- All three statements of iteration I are executed on the same processor. In an order that respects the
- All variables shared except the implicitly local index
• We have not discussed what happens before or after the doall.

• Unless otherwise indicated, loop-based programming typically assumes sequential execution of the statements before and after the doall.

• This implies that some thought must be given to how the processors start and complete the loop.
is assumed. (Although it may be suppressed at times.)

At the end of the loop a form of barrier synchronization

schedule decision.

to delay in their joining the parallel loop or due to a
in some cases, not all processors may be given work due
to processors.

begins the scheduling process to assign work to
processors that there is work to do in parallel and then
synchronization construct that informs the idle

The first is accomplished by some form of

•
until the next parallel loop construct reactsivate them. 

that do not take over the sequential execution become idle 

synchronization point. The processors that were active but 

synchronization point once all processors have reached the 

Excatly one processor can proceed beyond the

sometimes referred to as a Join.

The end of the doall is a special case of the barrier.

once all processors have reached the synchronization point.

all processors can proceed beyond the synchronization point.

A barrier synchronization point in a code is such that all
We have made no assumption here as to how the barrier is implemented.

Synchronization strategies in fairly complicated codes are expensive but, they can be used effectively to simplify both shared and distributed memory machines.

Barriers and joins can be implemented in several ways on
get to the location first and therefore read incorrect data.

congestion and the subsequent sequential section read
the write issued in parallel stalled in the network due to
distinct processors to a given memory location to have
different paths from

It is possible in networks that have multiple paths from

accesses the data location.

must complete before the sequential section read
was updated during the parallel section that update

If the sequential portion is going to access a variable that

during the parallel portion of the code.

The barrier must also consider memory requests issued


remain to be discussed.

- More control over scheduling and synchronization forms.

  distributed across the distinct address spaces.

  passing form with the barrier and independent data

data parallel computing that we also saw in messages

equivalent of the shared memory loop-based equivalent of

- Note that the deall and barrier synchronization is

  completed before proceeding through the barrier or join.

  Memory system that all of its outstanding writes have

  parallel processor to wait for acknowledgments from the

  To solve this problem it is typically required for the