end do
(\tau) x + s = s
\nu \tau_1 = \tau
0 = s
u = 1000

double precision x(1000)'
s integer \tau, u

Sequential code: scalar on 1 processor

\forall x \in x \exists u \quad \bigwedge_{\tau=1}^{u} = 0

Just using the D0ATL.

We can get some control over synchronization and scheduling.

Example 3
\[(\forall x)(\exists u)(\sum_{n=1000}^{u} x(n)) = s \]

\[\sum_{n=1000}^{u} x(n)\] is double precision vector at \[n \in \text{integer}\]

Vector code on \[1\] processor
end do

end-critical-section

s = s + slocal

start-critical-section

do do

local = local + x(i)

do j = 1, n

loop

0 = local

double precision local

integer j

do all i = 1, numproc

0 = s

Parallel scalar, static scheduling
to execute any iterations.

processor involved in executing the loop before it begins

Code in the preamble is executed exactly once by each

The preamble is used to initialize the local variable `slocaI`.

•

local copy for workspace.

private to the processor, i.e., each processor has its own

All variables that are declared in the preamble are

•

explicitly (f and slocaI).

Local variables `private to each processor` are defined

the computation of the dot product.

The `doall` is used only to get all processors involved in

•
change from loop to loop.

change with virtual to physical assignment or may even

It is not an invariant processor number however. It may

The index number in this case provides the latter.

identification number for each processor.

number of processors involved in the loop and an

The interleaved scheduling requires knowledge of the

dotproduct.

the user to define the subvector(s) involved in the local

Simple static interleaved scheduling is done explicitly by

•
handled by a critical section synchronization strategy.

The local summation of the partial dot products is

argued that it is therefore not static.

parameter for this static scheduling. (Parvatiis would

parameter being used. This is the only runtime

The runtime number() is assumed to return the number

•
Relative to the amount done in a critical section; and the amount of parallel work to be done
number of processors that must complete the critical
section a processor must enter a critical section; the
time it takes to execute a critical section; the number of
Criticl performance considerations include: the length of
they may be idle waiting for a synchronization event
Other processors may be executing noncritical sections or
i.e. a code may have more than one critical section,
Other processors may execute different critical sections,
system is active if a critical section is being executed.
This does not mean that only one processor in the
By exactly one processor at any given time,
A critical section or code is one that can be executed
a lock variable.

some dialects the variable i must be specifically declared

lock i, locks it, and proceeds with the critical section. (In

it is "unlocked" the processor succeeds in seizing the

A call to lock() checks the state of the lock variable. If

the critical section of code.

A lock/unlock pair can be used to control admission to

around a critical section.

There are many ways to implement the synchronization
now proceed with the locking procedure.

scheme the next processor waiting on the lock that it can

unlocked and may not be based on some priority

A call to unlock changes the state of the lock to

A processor can succeed at any given time.

This locking is guaranteed to be indivisible, i.e., only one

-
busy-waiting.

declare when to try again, i.e., the user-controlled
an indication of failure and the main code could
-- The processor could return from the lock routine with

queue-based busy-waiting approach.

informed that the lock has been given to it, i.e., the
processors waiting for this lock and become idle until
-- The processor could add its id number to a list of

i.e., the system-controlled busy-waiting approach.
-- The processor could wait for a while and try again,

depending on the implementation.

If lock's state is "locked", there are multiple options •
and memory system. To implement this primitive depending on the network location can be done at a time. There are various ways.

Individually, i.e., only one reference to the memory condition and if the condition is not found it is set – TEST-AND-SET. A memory location is tested for an indivisible memory operation referred to as a

The special memory hardware typically exploits an

Special memory hardware to operate the operating system software.

Lock/unlock can be implemented at several levels from
failed TALS requests in the memory system.
set then a TALS is done. This reduces the number of
first before attempting the TALS. If the condition is not
read/write it is often the case that one tests the location
system and is therefore more expensive than a normal
Since it requires a special operation in the memory •
often used.

On distributed architectures a token-based strategy is

distributed architectures.

He developed these strategies for both shared and

uses coordinated reads and writes.

processors and a generalization for p processors that only

Lamport developed a mutual exclusion strategy for two

Lamport.

Perhaps the most famous efforts are by Dijkstra and

mutual exclusion required by a critical section.

There has been extensive work on how to implement the

implement a critical section.

It is not necessary to have a TAS or lock/unlock pair to
Parallel vector, system scheduled

1k = 0
s = 0
doall i = 1, n/32
integer j
double precision local
loop
j = (i-1)*32 + 1
local = sum(x(j:j+31))
call lock(1k)
s = s + local
call unlock(1k)
end doall
codes.

the subvectors, i.e., more than the p used in the previous

- There is one execution of the critical section for each of

- Subvector of length 32.

- The basic iteration is the partial dot product on a

- The system handles the scheduling.

-
barrier synchronization handling.

After completing the postamble the processor enters the parallel loop. Further work to do in the parallel loop.

After the processor determines it has no loop after the processor involved in the parallel.

A postamble is like a preamble in that it is executed processor and still have system scheduling.

We need postamble to dump local variables once per the system-scheduled form.

One lock/unlock pair per iteration \( n/32 \) is performed in the static user-scheduled form.

Note one lock/unlock pair per iteration (processor) is
end doAll

  call unLock(tk)

  s = s + Stocal

  call Lock(tk)
  end loop

end doAll

) ( )

sum + Stocal = Stocal

j = (i-1)*32 + 1

loop

Stocal = 0.0

double precision Stocal

integer j

doAll j = 1, n/32

0 = s

0 = tk
We have used explicit stack allocation here. (Stack)

We must define workspace per processor outside doall 

In that case, we need another way to maintain and dump 

Postamble construct on doall loops.
((sum(stocal(1:nuhproc)))

end doall

((sum(x)p) + (stocal(p)) stocal(p) = (\(j: j + 1\))

(1 = (1) * j + 1

loop

stocal(p) = 0.0

1 = extract-vpu() + 1

integer j,p

doall j = 1, n/32

allocate(stocal(nuhproc))

mark(stack)

0 = s

double precision stocal()

integer stack