CS320/CSE 302/ECE 392

Introduction to Parallel Programming for Scientists and Engineers

Spring 2000

i. Course organization

Chapter 1. Introduction

Applications

- Traditionally, highly parallel computers have been used for numerical simulations of complex systems such as the weather, mechanical devices, electronic circuits, manufacturing processes, chemical reactions, etc.

  ~~See the 1991 report from the General Accounting Office on Industry uses of supercomputers and high-speed networks.~~ (no longer available)

- Perhaps the most important government program in parallel computing today is the

  Accelerated Strategic Computing Initiative (ASCI).

  Its main objective is to accurately simulate nuclear weapons in order to verify safety, reliability, and performance of the US nuclear stockpile. Several highly-parallel computers (1000s of processors) from Intel, IBM, and SGI are now being used to develop these simulation

- Commercial applications are also important today.

  Examples include: transaction processing systems, web servers, junk-mail generators, etc. These applications will

2.1 The Von Neumann computational model

Discussion taken from Almasi and Gottlieb: Highly Parallel Computing. Benjamin Cummings, 1988.

- Designed by John Von Neumann about fifty years ago.
- All widely used “conventional” machines follow this model. It is repres4es fnext:

previous assignment statement would be translated into a sequence of the form:

```
LD  1,B   (load B from memory into register 1)
LD  2,C   (load C from memory into register 2)
ADD 3,1,2 (add registers 1 and 2 and put the result into register 3)
ST  3,A   (store register 3's contents into variable A's address in memory)
```

- It is said that the compiler creates a "virtual machine" with its own language and computational model.
- Virtual machines represented by conventional languages, such as Fortran 77 and C, also follow the Von Neumann model.

2.2 Multicomputers

ˇrs

- Another example is the workstation cluster at Fermilab, which consists of about 400 Silicon Graphics and IBM workstations. The system is used to analyze accelerator events. Analyzing any one of those events has nothing to do with analyzing any of the others. Each machine runs a

2.3 Shared-memory multiprocessors

- VLIW (Very Long Instruction Word) processors are an

Chapter 3. Parallel Programming Models

- There are many different parallel programming paradigms. Mostial prerostonly.a

Tasks

Tasks are a central concept in parallel programming.

A task is a sequential program under execution. The programmer may assume that there is a processor devoted to each task. (There may not be a physical processor, but the operating system will time-share the real processors to give the illusion of one processor per task. It is said tper AM [(i1e)m-1.1133i1m

Shared-Memory Parallel Programming

The program starts as a single task program and then a second task is initiated. The second task proceeds by

Notice that because h is private to the task, a new copy of h will be created every time `start_task sub` is executed.

Consider the following program:

```
read b,c,e,g
start_task sub(b,c,a)
call sub(e,g,d)
wait_for_all_tasks_to_complete
q=f4(d,a)
print q
end


subroutine sub(e,g,d)
local h
h=f2(e)
d=f3(h, g)
end sub
```

Two copies of sub will be executing simultaneously, and each will have its own copy of h.

Parallel loops

One of the most popuar constructs for shared-memory programming is the parallel loop.

Parallel l /Flu0O ,1ong ing i6eratg i

Parallel loops are implemented using tasks. For example, the previous program could be translated by the compiler into something similar to the following program:

```
c = sin (d)
start_task sub(a,b,c,1,10)
start_task sub(a,b,c,11,20)
call sub(a,b,c,21,30)
wait_for_all_tasks_to_complete
e = a(20)+ a(15)
...

subroutine sub(a,b,c,k,l)
   ...
   for i=k to l
      a(i) = b(i) + c
   end for
```

SPMD model and parallel loops

Starting a task is usually very time consuming.

For that reason, current implementations of extensions

```
c = sin (d)
i=my_proc()*10
call sub(a,b,c,i+1,i+10)
call barrier()
if my_proc() .eq. 0 then
   e = a(20)+ a(15)
end if
...

subroutine sub(a,b,c,k,l)
   ...
   for i=k to l
      a(i) = b(i) + c
   end for
end sub
```

express parallelism since array operations can be easily executed in parallel.

Thus, all the arithmetic operations (+, -, * /, **) involved in a vector expression can be performed in parallel. Intrinsic reduction functions ,such as `sum`

Vector programs are easily translated for execution on shared-memory machines. The code segment:

```
c = sin(d)
a(1:30)=b(2:31) + c
e=a(20)+a(15)
```

is equivalhe 5st Thfollowinghe code seegment:

Going in the other direction, it is not always simple to transform forall loops into vector operations. For example, how would you transform the following loop into vector form?

```
parallel do i=1 to n
   if c(i) .eq. 1 then
      while a(i) .gt. eps do
         a(i) = a(i) - a(i) / c
      end while
   else
      while a(i) .lt. upper do
         a(i) = a(i) + a(i) * d
      end while
   end if
end parallel do
```