CISC 811: High Performance Computing, Assignment 3

Set Feb 12th due March 5th
Office: 308A Stirling Hall. Please email if you wish to set a time for an appointment.

For this assignment you will need an account on the HPCVL systems. I have set up a master account name (hpcg1142). To get your account, go to the HPCVL website (www.hpcvl.org) and fill out the account request form. You will need to fax a user agreement to them directly. I have allowed an extra week for the assignment in case the account applications take a few days to process.

Q1. Which of the following loops can be parallelized via an OpenMP parallel do, and which cannot? If they do not parallelize state why.

(a)
```
do i=1,n,3
a(i)=a(i-2)*3.
end do
```

(b)
```
      do i=2,n-2
      a(i+1)=a(i)+b(i+1)
      end do
```

(c)
```
do i=1,n
a(i)=a(i+1)+c(i)
d(i)=a(i)**2
end do
```

(d)
```
    do i=1,n
     j=j+1
     k=1-k
    a(i)=b(i*k+j,l)+c(i)
    end do
```

(e)
```
do i=1,n
c=a(i)
b(i)=a(i-1)*c
end do
```

(f)
```
    do i=1,n
     if (a(i).gt.1.0) then
       b(i)=b(i)+a(i)
     else
       b(i)=b(i)+1.0
    end if
    end do
```

Q2. Parallelize the following loop using `PARALLEL DO` loops (auxiliary storage is allowed):
```
do i=1,n-1
a(i)=a(i+1)
end do
```

Q3. Write an OpenMP parallel C/FORTRAN version of the following pseudo-code *without* using a `REDUCTION` statement on `iters` or a critical section/lock/atomic update to control access to the addition on `iters`. You may turn `iters` into an array of values which can be summed at the end. Note the optimal array access in C/FORTRAN should be used.
```
n=4096
val(n,n)
iters=0
do i=1,n
do j=1,n
val(j,i)=func(j,i)
iters=iters+1
end do
```

```
end do
write(*,*) 'Iters=',iters
end
function func(j,i)
func=sin(sqrt(float(i)))*cos(sqrt(float(j)))
return
```

(a) Test the speed of this code on 1,2,4,8 processors on HPCVL, and plot a graph of the results.
(b) Write a second version of the code that does use the `REDUCTION` intrinsic. Again, run on 1,2,4,8 processors and plot these results on your earlier graph. If you observe a significant performance difference between the two codes explain the origin of the difference in terms of the machine architecture. Submit both versions of the code with your graphs and explanation, and email me a copy of both codes.

Q4. The recursive loop

```
do i=2,n
a(i)=a(i)+a(i-1)
end do
```

is typically considered a barrier to parallelization. However, this operation can be parallelized to a certain degree by using partial sums (*i.e.* you sum isolated pieces of the array first and then bring them together in a specific order). Write an algorithm for the above operation that while taking O(n log n) operations can still be run in parallel using the `PARALLEL DO` in OpenMP. Auxiliary storage will be necessary, and for simplicity you may assume that n and the number of threads are both powers of 2. Again provide both a written and electronic version (via email) of your solution.

Q5. The sieve of Eratosthenes is a well known method for determining the number of prime numbers below a given number. The algorithm works by eliminating all the multiples of all primes less than or equal to the square root of the given number. The remaining numbers are then primes. Pseudo-code for the algorithm:

```
Eratosthenes(n) {
a[1] := 0
for i := 2 to n do a[i] := 1
p := 2
while (p^2 ≤ n) do {
j := p^2
while (j ≤ n) do {
a[j] := 0
j := j+p}
repeat p := p+1 until a[p] = 1
}
return(a)}
```

Implement this algorithm in a code that uses a task queue within an OpenMP parallel region to achieve parallel execution. Compare speed up for n=1000 on 1,2,4 processors. Do the same for n=1000000. Explain the scaling results you achieve.