INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

Department of Computer Science and Engineering

CS528 (High Performance Computing) Mid Semester Examination

Date: 29th Feb 2020 **Timing: 2 Hours** **Full Marks: 40** **Answer all the questions**

==========

Q1 [6 Marks] [Topic: Code Optimization]

```
double s, A[N], B[N], C[N];
for(int i=0;i<N; i++)//N is Large
      A[i]=B[i]+s*C[i];
```

A. [**2 marks**] Calculate code balance (b_c) of the shown code.

Sol: Code balance b_c is amount data by required per arithmetic operations. So for the above code b_c= 4 *8 Byte/2 F= 32B/2F = **16 B/F**. Assuming writing of A[i] to be buffered than b_c= 3 *8 Byte/2 F= 24B/2F = **12 B/F.**

B. [**2 marks**] Assuming the same code execute on processor having 32KB L1 data cache with 64B line size (or block size), Re-calculate the code balance.

Sol: As code to be executed on processor with 32KB L1 cache with 64B line size (or block). Number data can fit to block of cache is 64B/8B=8 and there will be one per 8 data access, which is number of miss per access =1/8. So miss rate is 1/8. Hence for the above code b_c= 1/8 (3 *8 Byte/2 F)= (1/8) 24B/2F = **1.5 B/F**.

C. [**2 Marks**] Assuming the same code execute on processor having 32KB L1 data cache with 64B line size (or block size) and L1 cache uses **No-Write Allocation Policy**, Re-calculate the code balance.

Sol: As code to be executed on processor with 32KB L1 cache with 64B line size (or block). So miss rate is 1/8. The cache use No-Write Allocation Policy, means cache block will not be allocated for array A, write access to array will be buffered and data access for array will not be counted. Hence for the above code b_c= 1/8 (2 *8 Byte/2 F)= (1/8) 16B/2F = **1 B/F**.

==========

Q2 [14 Marks] [Topic: Implicit Threading Programming]

A. [**3+3 marks**] Calculate the expected execution time for the following OpenMP code on 2 processors for the given code in term of N and R for two scheduling policy (a) schedule (static,1) and (b) schedule (dynamic,1). Assume FindScaledSumX(X,s) calls are independent for all the i^{th} iteration. (Chunk size =1)

```
#omp parallel for schedule (static/dynamic, 1)
{
      for(int i=0; i<N; i++) {
            X=rand()%R; s=rand()%200;
            Sum=FindScaledSum(X,s);
      }
}
```

```
int Data[R];
int FindScaledSum(int X,int s){
   int i, T=0;
   for(i=0; i<X; i++)
      T = T+s*Data[i];
   return T;
}
```

Sol: The above code generate N number of independent tasks indScaledSum(X,s). FindScaledSum(X,s) will take time proportional to X and X is randomly distributed between 0 and R, say **random(0, R)** .

In case of schedule (static, 1): the scheduler takes 2 tasks at time and executes one on each processor. Once execution time of both the tasks finished, it picks next two tasks and this continue till execute all the tasks executed. Every phase two tasks will be picked up and there will be **ceil (N/2)** phases of execution. Execution time of two tasks in a phases will maximum of execution time of both the tasks which is **max (random (0, R), random (0, R)) and the expected value of this will be (2/3)R (look at basic probability for explanation or intuition or google "expected value of the maximum of n random variable"→n/(n+1))**. So the total execution time will be N/2 Phases * (2/3)R = **N.R/3**.

Another explanation : Expected value of any RV Z is $E(Z \geq z)=\int_0^{+\infty} (1-P(Z \leq z)^2 dz$. Max of two independent uniform random variable $P(\max(X,Y) \leq z) = P(X \leq z).P(Y \leq z)$. Suppose distribution is uniform between [0-1] then $E(\max(X,Y))=)=\int_0^1 (1-z)^2 dz = 2/3$.

In case of schedule (dynamic, 1): the scheduler takes one task at time and executes one processor where execution of previous task is finished. If a processor get a shorter task then it will be get another task immediately. Both the processors will get almost equal number of task (=N/2) and amount work probabilistically if N->infinity. So expected execution will be N/2 * E(**random(0,R)) = (N/2) * (R/2) = N.R/4**.

B. [**3+3+2 Marks**] Given the Cilk like Pseudo code for the Heat Flow Simulation (HFS) Application in the two text boxes. This code does HFS for M time iterations and in each iteration it spawn HFS function. Calculate the amount of Work (T_1), Span (T_∞) and available Parallelism (P) for the Heat Flow Simulation Application using order notation using M and N. Assume value of BASE is constant and base case takes O(N) computation time.

```
double P[N][N], C[N][N];
cilk HFS(double **P, double **C,
                int l, int u, int TS){
if( (l-u)< BASE)
    BASECaseF(){..}//do Base*N Operations.
else { m=(l+u)/2;
       spawn HFS(P, C, l, m, TS);
       spawn HFS(P, C, m+1, u, TS);
       sync;
  }
 }
```

```
cilk main(){ //TS is TimeStamp
 for(int TS=0; TS<M; TS++){
   spawn HFS(P, C, 0, N, TS);
   sync;
 }
}
```

Sol: In main(), HFS() is repeated serially M times.

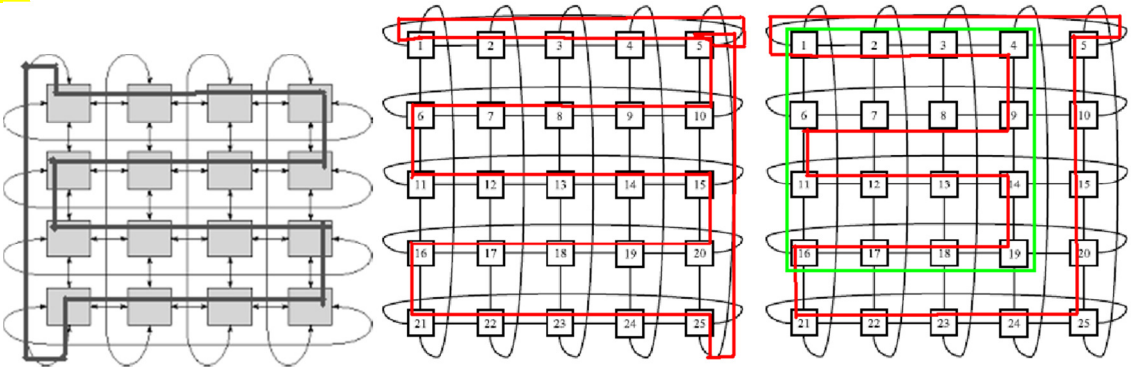
- Work of one iteration HFS will be multiplication of number of base case call $T_{1b}(N)=[2T_{1b}(N/2)+c]$ and BaseCase time. Number of base case call will be $T_{1b}(N)=O(N)$ and base case time will be O(N), so the work for one iteration will be $T_1(N) = T_{1b}(N) * O(N)=O(N^2)$ and total work will be $\mathbf{T_1=O(M*N^2)}$.
- Span of one iteration will be $T_{1b}(N)=[1T_{1b}(N/2)+c]$ and BaseCase time. So the span for one iteration will be $\log_2 N$ +N. The span of all the iteration will be $\boldsymbol{T_\infty = O(M * (N + Log_2 N))}$.
- Parallelism available in the application will be $Tp=M*O(N^2)/MN\log_2 N$, $\mathbf{Tp=O(N^2/(N+Log_2 N))}$

==========

Q3 [10 Marks] [Topic: Static Network and Embedding]

A. [**4+3 Marks**] Given a parallel application written and optimized for ***Ring Network*** (of size N nodes, $N=k^2$), the same parallel application needed to be re-implemented on a ***2D-Torus Network*** (of size N nodes, $N=k^2$). Provide an efficient network embedding (mapping function of i^{th} node of Ring to $(j, k)^{th}$ node of 2D Torus) and calculate dilation, congestion and load factor for your embedding.

Sol: Number of nodes for both Ring and 2D torus are same. The embedding is fairly simple: convert 2D torus to ring by virtually removing extra edges of 2D torus and keep the essential edges of 2D torus. The approach is demonstrated in the figures



Assuming 1st row is 0th row (even row) Mapping of R(i) to 2DT(l,m) is

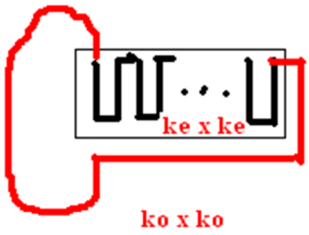
if i/k is even i_{th} node of Ring R(i) mapped to 2DT(i/l, i%k) node

if i/k is odd i_{th} node of Ring R(i) mapped to 2DT(i/l, (k-1)-i%k) node

If k is even : dilation, load factor and congestion is 1 (depicted in left figure)

If k is odd : load factor is 1, congestion is 1 but dilation is 2 (one edge of ring need to be mapped to two edges of torus) (depicted in right figure)

Another: Elegant solution in converting k_{1_even} x k_{2_even} 2D torus is easy using above method, but it is possible to do k_{1_odd} x k_{2_odd}, k_{1_oven} x k_{2_odd} and k_{1_even} x k_{2_odd} 2D torus to ring by doing it for lower one. A 4x4 torus ring can be part of 5x5, 5x4 and 4x5 torus ring. This is depicted in bellow figure where $k_o=k_{e+1}$. In this case, dilation, congestion and load factor will be one.



B. [**3 Marks**] Write benefit and demerits of Hypercube network as compared to 2D Mesh network.

Sol: Benefit: Hypercube diameter is Log N, All pair communication is easy, Bisection is N/2 and Bisection BW = N/2*link BW, robust as compared to Mesh, but require (N*LogN)/2 links to connect.
2D mesh diameter is 2*sqrt (N) which is higher than Log(N) of HC. BS is sqrt(N) which is less than N/2. But require N*2 (bi-dir) links to make 2D mesh which is less costly as compared to HC.

==========

Q4 [10 Marks] [Topic: Scheduling]

[**2+2+3+3 Marks**] Describe the meaning of these two scheduling problems given bellow. Propose efficient solution scheme for them, and if possible provide prove idea for the proposed solution schemes.

A. $P \mid p_j=1 \mid \Sigma w_jU_j$ **B. $Q \mid ptmn \mid \Sigma C_j$**

Sol:

- Meaning of **$P|pi=1| \Sigma w_jU_j$** is there m homogenous processor in the system and there are n independent tasks with execution time of task is 1, every task have deadline d_j and each task have some weight w_j, all the tasks arrived at time 0 and pre-emption is not allowed. The tasks are needed to be scheduled before their deadline. If a task missed its deadline (completion time $C_j>d_j$) then $U_j=1$ otherwise $U_j=0$. We need to minimize the weighted sum of missed task.

Model Scheduling Approach: [[Ref Page 118, Sec 5.1 of Bruker Book]. **//Greedy Approach**

1. Sort all the jobs with $d_1 \le d_2 \le \ldots \le d_n$
2. Set S=Φ
3. For i=1 to n do
4. If (the i_{th} task is late when scheduled in the earliest time slot on a machine)
   a. Find a task i* with w_i* = min weight of tasks in the already scheduled tasks of the set S
   b. If (w_i* < w_i) replace i* with i_{th} task in the schedule and in S.
5. else add i_{th} task to S and schedule the task in the earliest time slot

This approach takes O(nlogn) time and produce optimal result. Optimality can be prove by contradiction by assuming the some other set is optimal.

- Meaning of **$Q|ptmn|\ \Sigma C_j$** is there m uniform processor (processor with different speed) in the system and there are n independent tasks with different execution time of task (p_j), all the tasks arrived at time 0 and pre-emption is allowed. We need to minimize the sum of completion time of all the tasks.

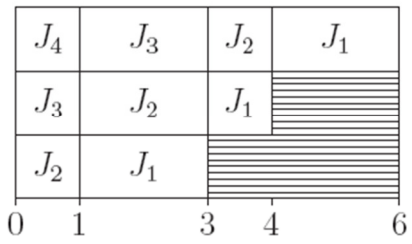
Model Scheduling Approach: [[Ref Page 134, Sec 5.1 of Bruker Book].
The problem can be solved using **modified version of SPT** (shortest remaining time) rule. Order the tasks according to non-decreasing processing time. Schedule task 1 on available highest speed machine up to time $t_1=p_1/s_1$. Schedule 2^{nd} task on M2 for t_1 time and then on M_1 from time t1 to time $t_2 \geq t_1$ until it is completed and same process continues.

1. Sort all the jobs with $p_1 \leq p_2 \leq \ldots \leq p_n$
2. a=0
3. while ($p_n>0$) // largest task remaining execution time is not zero
4. Find the smallest index i with $p_i>0$// the current shortest task
    a. dt =p_i/s_1;
    b. For (k=i to k=min{n, i+m-1}
        i. Schedule task k on machine M_{1+i-k} during [a, a+dt];
        ii. Pk=pk-dt.$s_{1+i-k;}$
5. a= a+dt

This approach takes O(nlogn+mn) time and produce optimal result. Simple interchange argument use to prove the optimality.

Example m=3, s1=3, s2=2, s3=1 and n=4, p1=10, p2=8, p3=8, p4=3

SRT Job J_4 get scheduled on M1 with speed s1 for 1 time unit. Job 3 get scheduled on M_2 upto time 1 and then shifted to M1. Gant chat is given bellow with $\Sigma C_j=14$



==========