



UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

Advance on Loop Scheduling algorithms

Jose Aguilar

CEMISID, Dpto. de Computación

Esc. de Sistemas, Facultad de Ingeniería

Universidad de los Andes

Mérida, Venezuela

aguilar@ula.ve



UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

Outline

- Introduction
- Some Problems of Scheduling in Computational Systems
- The Loop Scheduling Problem



Introduction

- Choose the order in which a given number of activities should be carried out
- Many techniques have been used to solve this problem,
 - exhaustive search
 - heuristic techniques (SA, etc.)
 - intelligent techniques (GA, ANN, etc)

Introduction

- Set of resources and of clients that require services of those resources.
- Problem: *to find an efficient way to use this resources, in such a way of optimizing certain performance measures.*
- Key aspect: *policies used to assign the resources.*

Examples: “round-robin”, “first the shortest work”

Introduction

- The performance is the characteristic used to evaluate a scheduling system.
 - That so good the produced scheduling is?
 - As much as the scheduler lasts to achieve it?
- Scheduling Problem is a NP-complete Problem



UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

Introduction

- Aspects that characterize to the different scheduling algorithms
 - *Deterministic or Non Deterministic*
 - *Static or Dynamic*
 - *Centralized or Decentralized*
 - *Etc.*

Some Problems of Scheduling in Computational Systems

- Particularly for parallel/distributed system:
- Some problems:
 - Tasks Scheduling
 - I/O Scheduling
 - Disks Scheduling
 - Loops Scheduling
 - Parallel Loops
 - Loops with dependences among them.



The Loop Scheduling Problem

- Loops: *multiprocessing source in many parallel applications.*
- One form of exploiting this parallelism:
execute the iterations of the loops in parallel on the
different processors.

The Loop Scheduling Problem

- It should be exploited:
 - The parallelism that can exist among different iterations of loops and inside each iteration.
- The factors to consider in the loop scheduling are:
 - The dependence of data among iterations,
 - The load among the processors,
 - The cost due to the synchronization
 - The communication among the iterations
 - The execution of the scheduler, etc.

The Loop Scheduling Problem

- *Parallel Loops: A loop whose iterations don't have dependence of data among them*
- **Example:**

For (i=0; i<12; i++)

*A[i]=B[i]+C[i]*D[i]*

The parallel loops are perfect to parallel!!

The Loop Scheduling Problem

- Usually, the number of iterations is bigger than the number of available processors.
- If the execution times of the different iterations are similar, the loop is *uniform*
=> the N iterations can be distributed uniformly among the P processors.

The Loop Scheduling Problem

- Two examples of assignment of Parallel Loops

	Processors				Processors			
	①	②	③	④	①	②	③	④
Loop	1	2	3	4	1	4	7	10
Iterations	5	6	7	8	2	5	8	11
	9	10	11	12	3	6	9	12

- No-uniforms loops: *assign the same number of iterations on each processor is not always that each processor has the same quantity of work.*

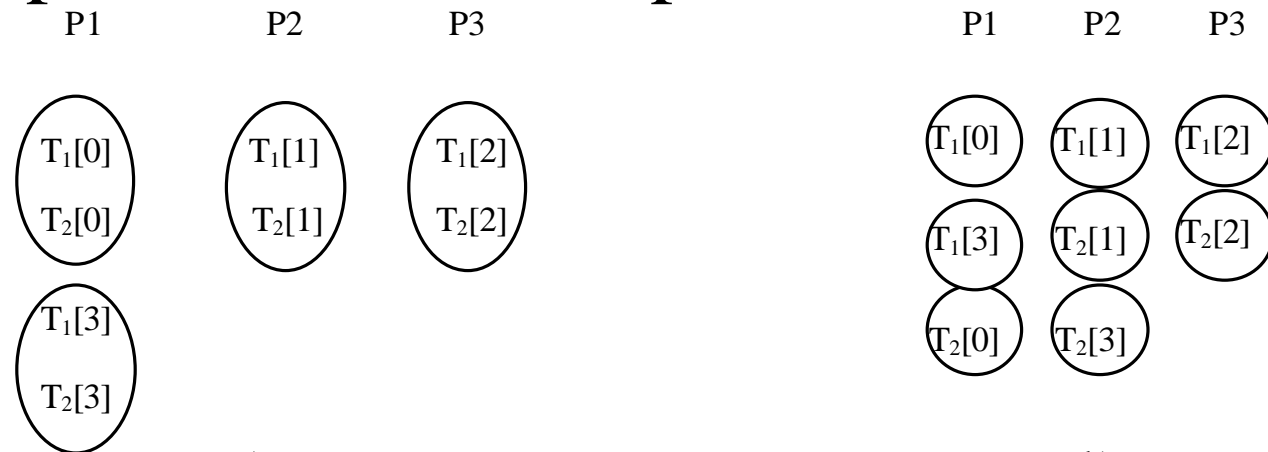
The Loop Scheduling Problem

For (i=0; i<3; i++)

T₁[i]

T₂[i]

- Planning on 3 processors based on a decomposition of the loop



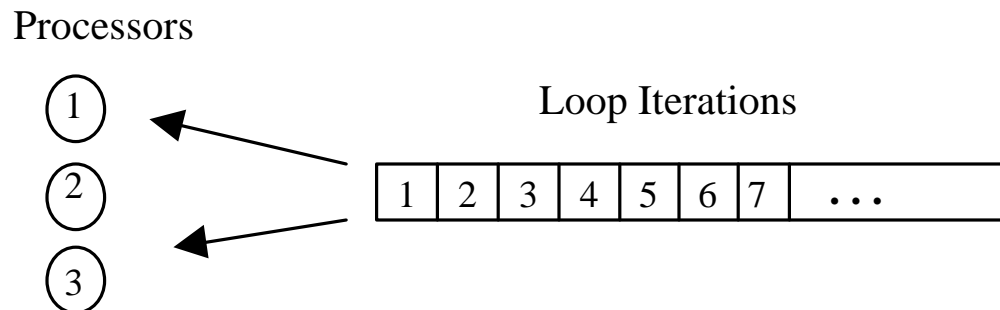
a)

J. Aguilar

b)

Classification of the Parallel Loop Scheduling Algorithms

- *Static Parallel Loop Scheduling Algorithms*
- *Dynamics Parallel Loop Scheduling Algorithms*
 - *Based on a Central Queue*



- *Based on Distributed Queues*

General Adaptive Parallel Loop Scheduling for Distributed and Shared Memory Systems

- *Central Queue and Dynamic Partition (for both distributed and shared memory systems)*

Repeat until central queue is empty

 Select idle processor j to receive iterations

 Lock central queue

 Remove chunk _{g} of the remaining iterations and send to processor i .

 Unlock central queue

 Execute these loops

 Modify CS

 Update state

J. Aguilar

Data Dependent Loop Scheduling

- Dependence among different iterations
- Types of Dependences:
 - *Loop carried*: when data are passed between different iterations
 - *Loop independent*: when data are passed from one task to another within the same iteration
- The second dependence type can be represented using task graph.

Data Dependent Loop Scheduling

- *Iteration vector*: each instance of execution of a n -nested loop is described by a iterations vector $\{I_1, \dots, I_n\}$.
- *Distance*: suppose two tasks v and w in a l -nested loop. If the task w executed in the iteration I_w depends on the task v executed in the iteration I_v , the distances is $D = I_w - I_v$

Data Dependent Loop Scheduling

- *Dependence Pair*: exists between two tasks v and w , where w is executed in the iteration I_w and v in the iteration I_v . The nomenclature is (D, W) , where D is the distances and W is the size of the message that w receives from v .
- *Dependence Set*: The set of all dependence pairs between two tasks.

Data Dependent Loop Scheduling

- *Upper bound vector of a n -nested loop:* is $\{b_1, b_2, \dots, b_n\}$, where b_i is the superior or upper bound of the loop of the level i .
- *Unrolling vector:* If it is equal to $\{u_1, \dots, u_n\}$, the loop i is decomposed (unrolling) u_i times.

Data Dependent Loop Scheduling

For i= 1 to 2

For j=1 to 2

T1(i, j)

T2(i, j)

Task T1(i,j):

$X[i, j] = F1(V[i-1, j], X[i-1, j-1])$

$Z[i, j] = \text{constant1}$

$V[i, j] = \text{constant2}$

Task T2(i, j):

$Y[i, j] = F2(Z[i, j], Y[i-1, j])$

Loop Task Graph

Suppose:

F2=8 units of computation

X=10 units of storage

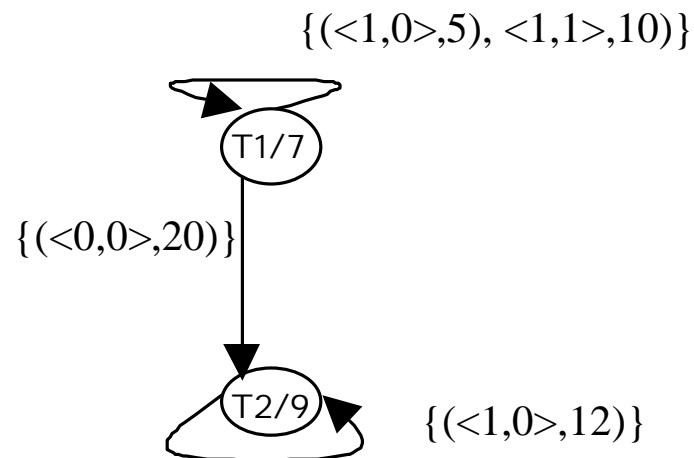
Z=20 units of storage

F1= 4 units of computation

Assignment=1 units of computation.

Y=12 units of storage

V=5 units of storage



J. Aguilar

Data Dependent Loop Scheduling

- *Loop Unrolling*: the iterations of a loop are replaced by their code.
- *The idea*: unroll the loop in order to uncover loop-carried dependencies that allow several iterations to overlap in execution.

For i= 1 to 4

$$X[i+2]= X[i+1]+X[i]$$

For i= 1 to 4 step 2

$$X[i+2]= X[i+1]+ X[i]$$

$$X[i+3]= X[i+2]+ X[i+1]$$

Data Dependent Loop Scheduling

- When a loop is unrolled u times:
 - $u+1$ copies of the body are replicated,
 - the variable of control of the loop is adjusted by each copy
 - the step of the loop is multiplied by $u+1$.

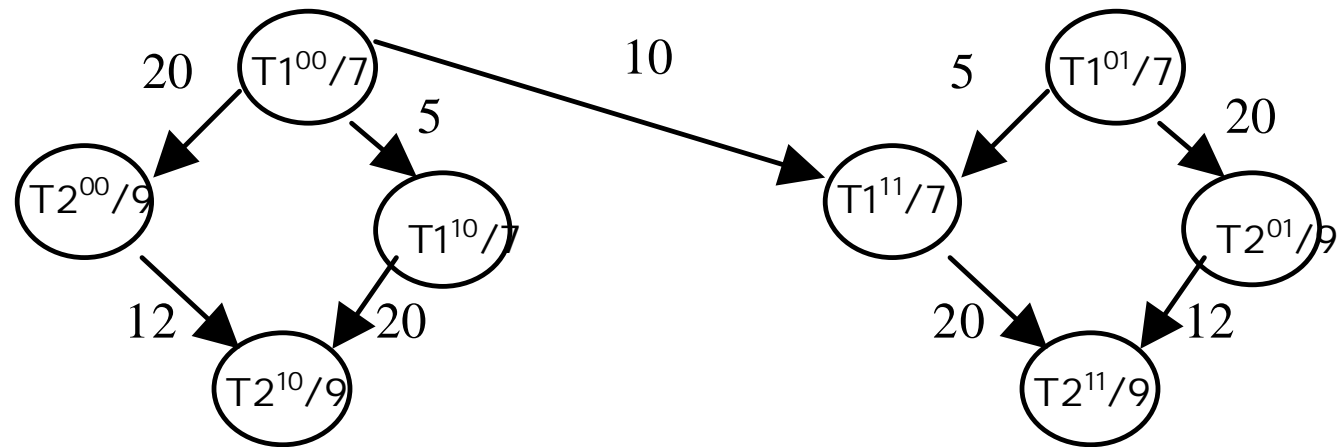
$\prod_{i=1}^n (u_i + 1)$ copy of the body are created !!

Data Dependent Loop Scheduling

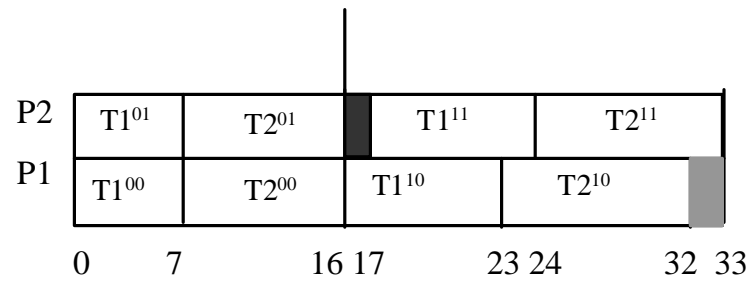
- *replicated tasks graph* ($G_u = V_u, A_u$): is an acyclic graph that represents the body of the loop after being unrolled using $u = \{u_1, \dots, u_n\}$.
 - The group of nodes V_u is the group of replicated tasks ($|V_u| = |V| * \prod_{i=1}^n (u_i + 1)$).
 - Set of arcs: arcs that represents the original loop-independent dependencies and loop-carried dependencies that have become loop-independent as a result of the unrolling.
 - The weight of the nodes and replicated arcs are similar to that of the original graph

Data Dependent Loop Scheduling

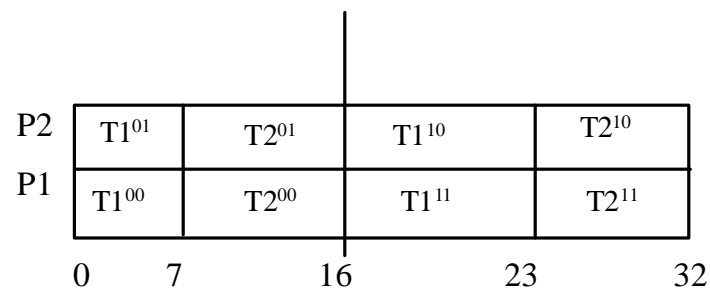
Replicated Task Graph for $u=\{1, 1\}$



Data Dependent Loop Scheduling



a)



b)

- Idle processor
- Communication delay among tasks

Data Dependent Loop Scheduling

1. Calculate the complete loop unrolling $u = \{b_1 - 1, \dots, b_n - 1\}$ for the Loop Task Graph G in order to determine the set of M tasks on the loop.
2. Each task on the loop is assigned to a ready queue.
3. As long as the ready queue is not empty (M times):
 - 3.1 Obtain a task from the front of the queue.
 - 3.2 Select an idle processor to run the task.
 - 3.3 When all the immediate predecessors of a particular task are executed, their successor is ready to be inserted into the ready queue.

Data Dependent Loop Scheduling

- The sequential execution time of a nest of n loops is

$$T_s = \sum_{I^1=0}^{b_1-1} \dots \sum_{I^n=0}^{b_n-1} \sum_{j=1}^N T_j^{I^1 \dots I^n}$$

- The parallel execution time (T_p) of the loop on K processors, according to a given assignment $\tilde{O}(g)$ of the tasks of the loop

$$T_p(\Pi(g)) = \max_{T_j^{I^1 \dots I^n} \in TS} \left\{ T_j^{I^1 \dots I^n}(\Pi(g)) \right\}$$

Data Dependent Loop Scheduling

The instant at which $T_j^{I^1 \dots I^n}$ will terminate, according to the current assignment $\tilde{O}(g)$, can be defined as follows:

$$T_j^{I^1 \dots I^n}(\Pi(g)) = T_j^{I^1 \dots I^n} + \max \left\{ X_i^{I^1 \dots I^n}(\Pi(g)) \right\} +$$

$$\left| \max_{T_i \in \mathbb{R}} \left\{ T_i^{I^1 \dots I^n}(\Pi(g)) + \lambda_{ij}^{I^1 \dots I^n, I^1 \dots I^n}(\Pi(g)) \right\} - \max \left\{ X_i^{I^1 \dots I^n}(\Pi(g)) \right\} \right|_{\text{if (COND=true)}}$$

$$\text{COND} = \max_{T_i^{I^1 \dots I^n} \in \mathbb{R}} \left\{ T_i^{I^1 \dots I^n}(\Pi(g)) + \lambda_{ij}^{I^1 \dots I^n, I^1 \dots I^n}(\Pi(g)) \right\} > \max \left\{ X_i^{I^1 \dots I^n}(\Pi(g)) \right\}$$

Data Dependent Loop Scheduling

- Ideally, we want to choose an assignment $\tilde{O}(g)$ which minimizes T_p . For our problem, we define the objective function as follows

$$\text{Cost Function} = \min_g \{T_p(\Pi(g))\} = \min_g \left\{ \max_{T_j^{I^1 \dots I^n} \in ST} \left\{ T_j^{I^1 \dots I^n}(\Pi(g)) \right\} \right\} \quad \forall g = 1, \dots, L$$

- The data dependent loop scheduling problem is a *min-max problem* where L is the number of the possible different assignments of the tasks ($L=K^M$).



UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

ADDITIONAL INFORMATION

- <http://www/cemisid.ing.ula.ve/~aguilar>
- "Task Scheduling in Parallel and Distributed Systems", H. The-Rewini, T. Lewis, H. Ali, Prentice-Hall, 1994.
- "Introducción a la Computación Paralela", J. Aguilar, E. Leiss, Edit. Universidad de los Andes, June 2003