84

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE QUIMICA

"APLICACION DEL PROCESAMIENTO EN PARALELO Y LA PROGRAMACION NO LINEAL EN LA INGENIERIA QUIMICA"

T E S I S

QUE PARA OBTENER EL GRADO DE:

INGENIERO QUIMICO

P R E S E N T A :

GERARDO MARIN FLORES

MEXICO, D. F. 1995

FALLA DE ORIGEN

Jurado asignado :

Presidente	**Prof. Carlos Escobar Toledo**
Vocal	**Prof. Manuel Vazquez Isias**
Secretario	**Prof. Celestino Montiel Maldonado**
1er. Suplente	**Juan Carlos Jimenez Bedolla**
2do. Suplente	**Victor Manuel Vargas Chavez**

La presente tesis se desarrolló en el Depto. de Ingeniería Química de la Facultad de Química Edif. E , U.N.A.M., C.U.

Ing. Celestino Montiel Maldonado
Asesor

Gerardo Marín Flores
Sustentante

Con todo cariño
A mis padres Silvia y Guillermo,
a mi hermana,
a la familia Marín por el apoyo y
amor que siempre me han brindado.

"Ante todo , es preciso conocer el fin hacia el que debemos dirigir nuestras acciones ; es necesario descubrir nuestro destino para poder tomar la firme determinación de dirigirnos hacia él. "

Confucio

Agradecimientos.

Agradezco al Ing. Celestino Montiel Maldonado la guía , la comprensión y el apoyo brindado para la realización del proyecto . Reconociendo que su promoción hacia trabajos novedosos impulsa el desarrollo de la Facultad de Química de nuestra Máxima Casa de Estudios.

También quisiera agradecer al Ing. Manuel Vazquez Islas por su valiosa ayuda y amistad a lo largo de mis estudios.

Mi sincero agradecimiento al Dr. Mike Bartholomew Biggs de Hatfield Polytechnic por sus valiosos comentarios y aportaciones al proyecto.

A la Lic. Marcela Peñaloza Baez de la Dirección General de Cómputo Académico mi gratitud por todo el apoyo recibido durante mi estancia como becario de la institución.

Índice

Introducción.

El procesamiento y la programación en paralelo es una técnica computacional que se ha venido desarrollando con la innovación de los equipos de cómputo . En este trabajo se describe los fundamentos y las aplicaciones posibles en la ingeniería química . Las mayores ventajas de esta técnica es que incrementa la velocidad de ejecución de los algoritmos y es posible ejecutar varias tareas independientes en una misma unidad de tiempo . En especial los algoritmos con bastante carga numérica han sido los más favorecidos con el uso de esta técnica .

La programación no- lineal es una técnica matemática para resolver modelos con función objetivo y restricciones (de igualdad y de desigualdad) no lineales . Su aplicación como método de optimización para problemas relacionados a la ingeniería química es amplia . Este tipo de rutinas , pueden llegar a tener una gran carga númerica dependiendo del problema en particular , lo que los convierte en candidatos ideales para procesarse en paralelo .

La estructura de este trabajo es la siguiente :

- En el capítulo 1 se describen los fundamentos del procesamiento en paralelo.

- En el capítulo 2 se presentan las aplicaciones generales de esta técnica a la ingeniería química.

- En el capítulo 3 se da la base y descripción de la programación no - lineal y sus diferentes métodos .

- En el capítulo 4 se desarrollan 3 problemas de ingeniería química , los cuales se resuelven con un algoritmo de programación cuadrática sucesiva previamente paralelizado y ejecutado en un equipo CRAY Y- MP4 .

En el anexo del trabajo se encuentra una descripción de los recursos de la supercomputadora y la forma de implementar una subrutina paralelizada en la misma .

La subrutina de programación cuadrática sucesiva en sus versiones secuencial y paralela estan a disposición de cualquier persona interesada en el departamento de Ingeniería Química de la Facultad de Química .

Capitulo 1

El Procesamiento en Paralelo.

Muchos de los problemas actuales de Investigación necesitan un mayor poder de cómputo y altas velocidades. Ejemplos de esta clase incluyen inteligencia artificial , robótica , procesamiento de señales , mecánica de fluidos, física cuántica, etc.

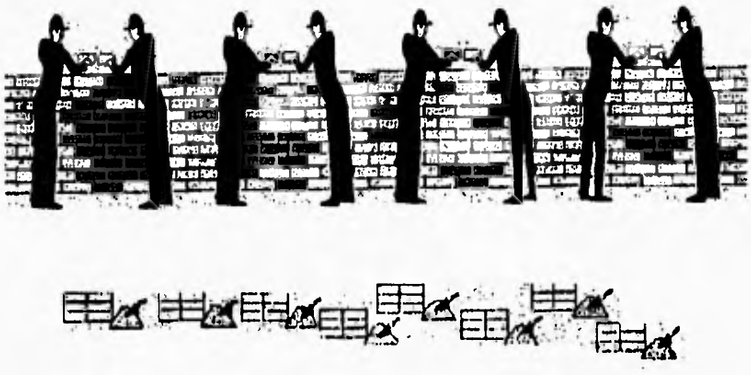
Debido a que los cálculos de los procesos químicos han llegado a ser más complejos , y el comportamiento del proceso es mejor comprendido , la simulación de procesos en estado estacionario se ha convertido en una importante herramienta para los cálculos de ingeniería . Un proceso químico simple puede fácilmente requerir la solución de 10000 o más ecuaciones . Una operación unitaria complicada tal como la destilación por platos agrega por lo menos 10000 más. Cerca del 80 % de estas ecuaciones describen relaciones termodinámicas . Para propiedades altamente no ideales donde métodos tales como SRK o UNIFAC son necesarios , los cálculos de las propiedades físicas usarán el mayor tiempo de cómputo [12]. Los argumentos anteriores permiten visualizar que la simulación y optimización de procesos se ha convertido en una rama que necesita cada vez más un mayor poder de cómputo y velocidad de procesamiento .

Teniendo en cuenta al procesamiento en paralelo como una alternativa para implementar en software de aplicación a esta rama.

Una aproximación para ganar velocidad , al igual que poder de cómputo, se encuentra a través del **paralelismo** ; aquí varios procesadores trabajan juntos , todos simultáneamente ejecutan porciones de un procedimiento usado en la resolución de un problema .

La programación paralela permite expresar en forma explícita , el comportamiento simultáneo e interdependiente de un conjunto de actividades o tareas las cuales tienen como finalidad la realización de un objetivo común fig. 1.1 [5] . Los objetivos de la programación paralela son básicamente

FIG. 1.1 EJEMPLO DE LA CONSTRUCCION CONCURRENTE.



1) Aumentar la eficiencia de procesamiento.

2) Aumentar la velocidad de respuesta.

3) Expresar la solución a problemas de naturaleza paralela.

Un programa paralelo se puede definir a través de :

1) Un conjunto de dos o más procesos que se ejecutan en forma simultánea.

2) La interacción entre los procesos por medio de la comunicación y la sincronización.

El procesamiento paralelo es la ejecución de procesos paralelos corriendo en procesadores separados , los cuales pueden tener cada uno su propia memoria local o tener acceso a una memoria compartida . De esta manera el procesamiento paralelo se categoriza como procesamiento distribuido o multiprocesamiento respectivamente.

Acoplamiento de los procesadores.

La tarea de ensamblar los procesadores para trabajar juntos es conocida como **acoplamiento.**

Dos métodos están disponibles para este propósito:

- Método de acoplamiento riguroso.
- Método de acoplamiento perdido.

Una red de procesadores acoplados rigurosamente comparte una memoria común o global y tiene un control centralizado. Los procesadores están físicamente unidos por un circuito o por un mecanismo de cableado.

Los procesadores con acoplamiento perdido usualmente no comparten memoria y no tienen un control centralizado . Usualmente consiste en una red de computadoras completamente separadas con su propia memoria , unidad de control y unidad central de proceso (UCP) conectadas vía un sistema de transferencia de mensajes [1,2,5].

Naturaleza de las tareas.

Los programas paralelos contienen varias partes que pueden ejecutarse paralelamente . Cada una de esas partes es en sí misma secuencial estrictamente y es llamada **tarea** o **proceso** [5] .

De esta manera un proceso es una unidad de código secuencial la cual se ejecuta en forma autónoma y paralelamente , junto con otros procesos en el mismo ambiente.

Hay dos grandes categorías dentro de las cuales las tareas computacionales pueden ser clasificadas [1]:

- Tareas mutuamente independientes o no interactivas.
- Tareas mutuamente dependientes o interactivas.

En el caso de tareas mutuamente independientes , la entrada consiste varios conjuntos de datos no interactivos , sobre estos datos se desarrolla un conjunto común de operaciones. Para llevar a cabo estas operaciones cada procesador trabaja independientemente dividiendo el problema .

Cuando las tareas son mutuamente dependientes , las diferentes tareas pueden ser simultáneamente ejecutables , algunas veces sobre un conjunto de datos común . Sin embargo , es posible que una tarea subsiguiente sea dependiente de la salida generada por una o varias tareas precedentes. En tal caso hay una necesidad por un mecanismo que coordine el arribo de la entrada de las tareas siguientes desde aquellas que las preceden. Además , una o más de las entradas requeridas pueden llegar en diferentes intervalos de tiempo , algunos antes otros después , por tanto las tareas subsiguientes tienen que esperar antes de cualquier acción. Para coordinar tal situación dinámica , o variante en el tiempo , un mecanismo de sincronización de alguna clase tiene que ser introducido. Tal mecanismo debe tener las siguientes propiedades:

1. La habilidad para detectar acciones desarrolladas por procesos interactivos.

2. La habilidad para permitir la transferencia de datos en el momento apropiado.

Estas dos propiedades requieren que la comunicación entre procesos dependientes y su sincronización vayan mano a mano y son , en general , inseparables.

El diseño de una máquina en paralelo con muchos procesadores y el diseño de un programa con muchos procesadores son críticamente dependientes en la manera en que la sincronización y la comunicación toman lugar.

Diferentes tipos de paralelismo y terminología.[1]

Concurrencia.

La concurrencia es un término alternativo usado para describir el paralelismo a través de un conjunto de procesos mutuamente dependientes los cuales se comunican entre ellos usando mecanismos bien definidos.

Multiprocesamiento distribuido.

El multiprocesamiento es la ejecución de trabajos independientes en paralelo usando muchas computadoras .Usualmente un sistema de multiprocesamiento consiste en dos o más computadoras ejecutando diferentes programas y usando la misma memoria . Los procesadores y la memoria están interconectadas a través de puertos múltiples y usando un interruptor central.

Pipelining[2]

Muchas computadoras , especialmente las llamadas supercomputadoras . usan pipelines para incrementar su velocidad. Un **pipeline** es algo semejante a una línea de ensamblaje , en la cual hay varios estados de procesamiento. Cada procesador ejecuta repetidamente la misma instrucción en conjuntos de datos

[1] Ver ref. 1,2,3,4

[2] Pipelining , es una técnica que encuentra su analogía en una línea de ensamblaje de autos . Ciertas tareas computacionales requieren varios pasos pequeños , cada uno de los cuales es hecho por un componente diferente del procesador , mientras un componente ejecuta su tarea los otros están esperando . Se prefiere usar el término en inglés debido a que el término en castellano puede presentar algunas confusiones.

sucesivos que son recibidos desde el procesador precedente. Esta aproximación es útil cuando un procedimiento o una instrucción puede romperse en pequeños componentes , así que cada componente puede ser asignado a diferentes etapas del pipe . Así que los resultados de un procesador llegan de entrada al siguiente procesador .

Algoritmos de multipaso.

Un algoritmo multipaso consiste de varios subalgoritmos o tareas. Cada tarea puede comunicarse con su tarea adyacente . Por lo tanto , la comunicación entre las dos tareas toma lugar de manera simétrica. Esto nos permite la ejecución de dos o más tareas esta aproximación es llamada **cuasiparalelismo** [1].

En una computadora mono procesador , un conjunto de procesos se pueden ejecutar en pseudo- paralelismo , por ejemplo a través de la técnica de tiempo compartido , el cual es controlado por el sistema operativo [5].

En una red de procesadores , un conjunto de procesos se ejecutan en verdadero paralelismo (pensando que cada proceso se encuentra en un procesador) , sin embargo , cualquier aspecto relativo al tiempo de ejecución de los procesos es indeterminado .

Cuando una colección de procesos concurrentes tiene procesos en los cuales la salida de cada proceso es usado como entrada a otro el flujo de información re ensambla una pipeline. Por ejemplo , UNIX es particularmente popular para el procesamiento en paralelo . Mucho del poder de UNIX viene de su diseño de multiprocesamiento . UNIX le permite al programador construir aplicaciones de una colección de programas que se ejecutan como procesos y estos se encuentran interconectados [3]

Procesamiento en arreglos vectoriales.

En procesadores de arreglos vectoriales , muchos procesadores rigurosamente acoplados ejecutan la misma instrucción . Este tipo de procesador es usado en un tipo de supercomputadoras llamadas **front end.**

Niveles de Paralelismo.[3]

Nivel de trabajo

Se desarrolla por medio del sistema operativo y los medios como se lleva a cabo son :

Multiprogramación.

Puede llevar cabo la ejecución de más de un programa . Se controlan las operaciones de entrada y salida y las operaciones del procesador.

Multiprocesamiento.

Correr dos o más CPU's concurrentemente sobre diferentes aplicaciones o sobre trabajos independientes de la misma aplicación general.

Nivel de Programa.

Se desarrolla por medio del Software y los medios para llevarla a cabo son:

Multitareas.

Es la descomposición de un programa en dos o más tareas (segmentos de programas) que se pueden ejecutar en forma concurrente . Esto requiere que las tareas sean independientes.

Microtareas.

Permite que más de un CPU trabaje sobre un programa a nivel de ciclos DO.

Nivel de Interinstrucción.

Lo desempeña el compilador y se desarrolla por medio de :

---

[3] Ver 9, 11

Concurrencia sobre las instrucciones múltiples.

Esto requiere un análisis de la dependencia de la información y se logra dividiendo cada instrucción en suboperaciones y desarrollar las diferentes suboperaciones como instrucciones diferentes.

Nivel de Intrainstrucción

El Hardware se encarga de desarrollarlo y se lleva acabo por medio de :

Pipelining.

Dividir cada instrucción en una secuencia de operaciones , cada una de las cuales puede ser ejecutada por una sección de hardware especializada que opera en forma concurrente con otras secciones en una pipeline .

Palabra-Instrucción .[4]

Desarrolla operaciones múltiples por instrucción , cada una con su propio campo de dirección.

Descripción de procesos paralelos.

Tradicionalmente , se enseña a programar procesos que son esencialmente secuenciales. En este tipo de programas , una lista de instrucciones son ejecutadas secuencialmente . La naturaleza secuencial da lugar a un conjunto de operaciones . Por lo tanto , un programa secuencial es fácilmente descrito con la ayuda de diagramas de flujo que contienen estructuras determinísticas primitivas :

1) begin (comenzar)

2) assignment (asignación)

[4]Entiendase como una palabra a la que se asocian varias instrucciones a ejecutarse.

3) if B then s else T (decisión)

4) while B do S (ciclo indefinido)

5) for y= 1 to n (ciclo definido)

6) end (terminar)

En caso de programas en paralelo o no secuenciales , en adición a las construcciones arriba señaladas son necesarios los siguientes mecanismos :

1) Mecanismos paralelos de iniciación y terminación

2) Mecanismos de sincronización, protección y comunicación .

Mecanismos de iniciación y terminación.

Cuando un conjunto de procesos son no-secuenciales se necesitan tener construcciones que puedan comenzar una lista de procesos simultáneamente y terminar la lista de procesos cuando todos ellos terminen . Estos procesos son respectivamente llamados construcciones **cobegin y coend.**

Mecanismos de Sincronización.

También , cuando un conjunto de procesos son no-secuenciales , los procesos pueden tener varias entradas y salidas simultáneas y pueden necesitar pasar datos de unos a otros o comunicarse.

La sincronización se puede ver como el conjunto de restricciones sobre el orden de los eventos.

Un mecanismo de sincronización especifica una secuencia parcial de puntos en los procesos que permiten , por ejemplo , retrasar la ejecución de los procesos con el fin de satisfacer las restricciones sobre el orden de los eventos [5].

Como la velocidad a la cual cada proceso trabaja no se puede determinar , el no determinismo crece al unir los procesos. Por consiguiente , aún si varios

procesos comienzan juntos pueden no todos ellos terminar simultáneamente. Por tanto se dice que existe una **condición de carrera** y las partes de computación que contienen tales procesos se conocen como **tiempos críticos.** De lo anterior se observa que es necesario introducir un mecanismo que coordine y controle el orden temporal en el cual los procesos son ejecutados para realizar un algoritmo no- secuencial. Tal mecanismo es llamado **mecanismo de sincronización.**

El mecanismo de sincronización es esencialmente necesario para manejar el no-determinismo que crece en la ejecución de procesos diferentes da oportunidad a la interrupción entre los procesos para coordinarlos. Aquí se necesitan dos construcciones adicionales una para la elección del no-determinismo (**selection**) y otro para espera (**wait**) , ambos para describir la programación no-secuencial .

Mecanismos de protección.

El mecanismo de sincronización solo no es adecuado para llevar a cabo la programación no-secuencial. Con el fin de coordinar los procesos es necesario prevenir los errores entre ellos. Es decir, se necesita restringir el acceso de una variable compartida o de un recurso a un proceso en un tiempo dado.

Esto se lleva a cabo mediante el uso de otro mecanismo llamado **mecanismo de protección.** Estos mecanismos de protección previenen interferencias de otros procesos cuando un proceso en particular esta usando una variable compartida o un recurso.

La secuencia de enunciados con los cuales un proceso exclusivamente accesa a una variable compartida o a un recurso es llamado sección crítica o región crítica. Cuando un proceso esta cerca de ejecutar su sección crítica el mecanismo de protección se asegura que otro proceso no este ejecutando su región crítica en el mismo tiempo. Una vez que se da acceso a un proceso se bloquea el recurso con un arreglo de bloqueo. Cuando el proceso completa su trabajo y deja su región crítica , el bloqueo del recurso se desactiva y se permite la entrada de otro proceso. En otras palabras , el mecanismo de protección

debería proveer una facilidad de bloqueo para cada proceso mientras ejecuta su región crítica y desbloquear y permitir a otros procesos ejecutar su región crítica.

Métodos de Sincronización.

Existen dos métodos para la sincronización de los procesos :

1. Método de la variable compartida.

En este método , la sincronización y comunicación entre los procesos se lleva a cabo usando mecanismos compartidos bajo un control central.

2. Método del envío de mensajes.

En este método los procesos son autónomamente controlados mediante envío y recepción de mensajes , sin compartir información ; y son coordinados mediante operaciones de espera entre ellos.

Estos dos métodos han contribuido al desarrollo de nuevos estilos de lenguajes de programación para la programación concurrente.

El diseño de métodos para la descripción de procesos paralelos y los mecanismos de protección y sincronización han sido un área activa de investigación durante los años recientes.

Tipos de algoritmos paralelos.

Flemming (1994)[5] da una clasificación de los tipos de algoritmos paralelos que pueden haber :

Réplica de eventos.

[5] Fleming , P. J. *Parallel algorithms & Genetic algorithms* , véase referencia 11

Aplicaciones que requieren el mismo programa (algoritmo) con diferente condición inicial , i.e. , simulaciones Monte Carlo con diferente valor aleatorio de inicio.

- Cada procesador tiene el mismo programa pero diferente información.
- Hay poca comunicación entre los procesadores , excepto para filtrar y analizar información.
- Eficientes pero el costo de memoria es alto.

Geométricos.

Explota la estructura geométrica de sistemas físicos .

- Cada procesador es responsable de un área espacial del sistema físico.
- La información se distribuye uniformemente a través de los procesadores , solo una fracción del total en cada uno.
- Cada procesador tiene una copia completa del todo el programa.

Algoritmico.

Usualmente cada procesador tiene una pequeña parte del total del algoritmo.

- Hay flujo de información entre los elementos de procesamiento . Puede haber una carga excesiva en la comunicación.
- Cada procesador requiere poco espacio de información.

Los Algoritmos Paralelos y su implementación.

Los siguientes comentarios conciernen a los algoritmos en paralelo y su implementación :

1. Los algoritmos en paralelo eficientes no lo son necesariamente cuando se ejecutan en computadoras secuenciales. En efecto , algunos algoritmos en

paralelo incluyen operaciones de punto flotante adicionales que los hace ineficientes en las máquinas secuenciales. También , los algoritmos seriales reestructurados pueden no ser tan eficientes en máquinas con procesamiento en paralelo.

2. Las propiedades matemáticas de las implementaciones seriales y paralelas del mismo algoritmo pueden ser diferentes. Por ejemplo , el grado de convergencia y estabilidad numérica de técnicas iterativas seriales y paralelas pueden ser diferentes . En algunos casos la implementación en paralelo puede degradar el desempeño y en otros casos , la mejora.

3. Para alcanzar un alto desempeño ambos , el algoritmo numérico y su implementación debe ser cuidadosamente manejada para la máquina en particular que se este usando. Esto lleva a la disyuntiva de la portabilidad de los programas en paralelo . No es practico el desarrollo de programas y algoritmos para cada computadora . De acuerdo a Noor [9] hay una gran cantidad de estudios encaminados a que haya una mayor portabilidad de los programas . Entre estas aproximaciones se encuentran : a) reestructuración de algoritmos en términos de módulos de alto nivel. (i.e. operaciones matriz-matriz , matriz-vector), b) Desarrollando e implementando una abstracción del procesamiento en paralelo que sea independiente de la arquitectura.

4) En la mayoría de los sistemas de multiprocesamiento disponibles actualmente la vectorización ofrece un gran desempeño aún sobre la multitarea . Consecuentemente , si la multitarea entra en conflicto con una vectorización eficiente , entonces los algoritmos deberían ser vectorizados más que paralelizados.

Modelado computacional del poder y la complejidad del paralelismo.

La teoría de computación concierne con el estudio de los diferentes modelos formales tales como :

1. Modelos de máquinas (por ejemplo máquinas de estado finito , máquinas de Turing)

2. Modelos de función recursiva.

3. Modelos gramaticales o producción.

Estos modelos permiten el entendimiento del poder computacional de las diferentes máquinas , solución de algoritmos , lenguajes , capacidad de generación y el estudio de computaciones complejas.

Para extender estos modelos para la computación en paralelo , es necesario entender los efectos de agregar opciones no deterministicas y acciones acopladas en los modelos secuenciales arriba mencionados.

Modelado de mecanismos paralelos de control e información.

Son dos los mecanismos básicos para la organización de cualquier programa. Estos son:

1. El mecanismo de control : Este mecanismo es responsable para la ejecución de instrucciones;

2. El mecanismo de información : Este mecanismo es responsable para proveer la información requerida por una instrucción .

La arquitectura clásica , conocida como la arquitectura de Von Neumann , el mecanismo de control esta organizado secuencialmente . Este mecanismo ejecuta las instrucciones una a una . El mecanismo de información usa direcciones a celdas de memoria para pasar información entre las instrucciones. En otras palabras , un nombre común de una variable es compartido por las instrucciones , así el mecanismo actúa "por referencia" a ese nombre común. También , los pasos de control junto con los segmentos de información fluyen secuencialmente . En otras palabras hay un control de flujo unidimensional.

Cuando se realiza computo en paralelo , no hay un flujo de instrucciones ni de información secuencial . Primero , la información de una computación en paralelo es generada y consumida concurrentemente y en forma no deterministica. Por lo tanto , el mecanismo de control necesita ser

descentralizado. Además una instrucción será ejecutada solo cuando todos los segmentos de información de entrada estén disponibles. También el mecanismo de información en estos casos no necesita actuar por referencia. Puede ser preferible pasar la información 'por valor' directamente entre las instrucciones , en lugar de almacenarlo y después referenciarlo . Esta comunicación directa eliminaría la lectura y escritura en celdas de memoria , y se ahorra tiempo en la computación.

La clasificación de los diferentes modelos basados en los mecanismos de control y de información es la siguiente :

1) Modelo de control de flujo de Multilectura.

Este modelo esta basado sobre las instrucciones paralelas especiales explícitas dentro del programa para iniciación , terminación y sincronización de los procesos tales como fork , join y wait para controlar el orden de ejecución de las operaciones. Existe un control centralizado y la secuencia de las instrucciones es manejada por los contadores del programa . Los objetos de información se pasan de instrucción a instrucción vía referencias a celdas de memoria compartida. por ello , este tipo de computación es conocida como computación **control-driven.**

2) Modelo de Flujo de Información.

En este modelo hay un control descentralizado . Las instrucciones son ejecutadas tan pronto como todos los objetos de información de entrada requeridas estén disponibles . El programa esta explícitamente estructurado en términos de flujo de información entre varias instrucciones . En otras palabras , la comunicación entre instrucciones es directa , sin memoria compartida. Los contadores no son usados. Los valores de salida intermedia se consumen y no están disponibles para un reuso. La dependencia en la información restringe el orden de las computaciones. Por lo tanto esto se conoce como computación **data-driven.**

3) Modelo de Reducción.

Este modelo denota la organización de la computación donde las instrucciones son seleccionadas solamente cuando el valor que producen es necesario para otra instrucción previamente seleccionada.

Este tipo de modelo es conocido como **demand-driven.**

Asociados con cada uno de los modelos señalados uno puede diseñar una clase particular de lenguajes de programación con una clase particular de arquitectura de soporte debido a que existen diferentes formas de trabajo de los mecanismos de control y de información.

Arquitecturas eficientes para el paralelismo.[6]

Los aspectos prácticos para el diseño de arquitecturas eficientes para el paralelismo son :

1. Sistemas de propósito general o especial.

Esto concierne a sistemas paralelos diseñados para realizar tareas especializadas o para un propósito general (por ejemplo , los procesadores de arreglos y vectores son máquinas paralelas de propósito especial.

2. Tamaño de Grano (Granularidad)

La capacidad y el número de procesadores básicos usados en un sistema determinan el tamaño del grano .

Los sistemas Multiprocesadores se pueden dividir en tres grupos :

a) Las Máquinas con unas decenas de procesadores grandes (tamaños grandes de palabra y gran capacidad de memoria) son llamadas de grano áspero Ejemplos de estas máquinas son la CRAY -2 , CRAY Y-MP y ETA-10.

---

[6] Las referencias 1,2 muestran un esquema general de las arquitecturas . Un estudio más completo sobre el tema se puede localizar en 3,4,6,7 y 9 .

b) Las máquinas con algunos cientos de procesadores más pequeños (tamaño de palabra pequeño y menor capacidad de memoria) son conocidas como de grano medio . Ejemplos de este tipo son Encore Multimax , FLEX/32 , iPSc de Intel .

c) Aquellas con varios miles de procesadores (tamaño de palabra pequeña y muy pequeña capacidad de memoria) son llamadas de grano fino . Cuando los procesadores trabajan en concierto pueden formar máquinas muy poderosas. Ejemplos de estas máquinas están la Goodyear Aerospace Massively Parallel Processor con 16384 procesadores , la Connection Machine C-2 de Thinking Machine Inc. con más e 65536 procesadores.

3. Interconexión (topología)

Una red de comunicación general , a través de la cual los procesadores con interconectados juega un papel muy importante . Por ejemplo , los procesadores podrían ser conectadas por anillos o como elementos en árboles , etc. Esto necesariamente determina la existencia de un mejor patrón de interconexión para un algoritmo dado[1] .

La Topología afecta la clase de problemas que pueden resolverse eficientemente en una máquina . Las topologías más comúnmente usada son :

Conexión tipo anillo.

Varios procesadores , sistemas de memoria , y sistemas de entrada y salida, residen en comunicación formando un anillo o un conjunto de anillos . La mayoría de las computadoras en este tipo incorporan memoria global , y compartida por todos los procesadores , y accesados vía la comunicación en el anillo . Ejemplos de máquinas con conexión de anillo son Encore Multimax , FLEX/32 , y Elxsi 6400 .

Conexión de Hipercubo.

Cada procesador esa conectado directamente a sus n vecinos . El número de conexiones por procesador resulta en un cubo multidimensional con 2^n nodos .

Esto tipo de conexión es usualmente usado con máquinas de memoria distribuida. Ejemplos de máquinas con conexión hipercubo son Intel iPSC .

Conexión por Interruptores.

Este mecanismo se basa en colocar un interruptor entre los procesadores y los bancos de memoria , y por tanto se remueve la propiedad entre la memoria y su procesador . Para diferentes colocaciones de los interruptores , un procesador puede conectarse a diferentes bancos de memoria.

4. Grado de acoplamiento.

Este factor determina si los procesadores comparten una memoria global y un reloj o no. La naturaleza de los mecanismos de control y de información esta determinada por el grado de acoplamiento.

5. Mecanismos de Control y de Información.

Existen tres mecanismos de control y de información diferentes : control-driven , data-driven y demand-driven. El uso de cada uno de estos mecanismos es dependiente del lenguaje de programación usado y de la arquitectura soporte.

6. Alojamiento de Tareas y ruteo.

Esta es una de los aspectos más difíciles . Habiendo escogido el patrón de interconexión y los mecanismos de control y de información , se debe de encontrar la forma más efectiva de alojar las tareas a los diferentes procesadores para la implementación de un algoritmo dado. Este problema es conocido como el **mapeo** del programa sobre la arquitectura de la máquina.

Existen dos estrategias para alojar las tareas : estática y dinámica. En el alojamiento estático , las tareas son alojadas desde la primera vez en un solo procesador , esto determinado por la topología y acoplamiento del sistema . En el alojamiento dinámico , las tareas varían de procesador a procesador para balancear la carga . Esto es análogo a la 'programación dinámica ' . También uno puede utilizar una combinación de estrategias dinámicas y estáticas.

Otro aspecto importante es si el alojamiento de las tareas se debe realizar cuando el programa esta corriendo o cuando se compila. Si el programa no esta bien acoplado a una arquitectura específica , el alojamiento de las tareas puede llevarse mucho tiempo. Alternativamente , uno puede alojar las tareas durante la compilación . En este caso los mejores resultados se llevan a cabo si la compilación produce un programa mejor acoplado a la arquitectura dada. Para determinar si un proceso esta bien acoplado a la arquitectura se debe de comparar las actividades del flujo del programa contra las actividades de comunicación de la máquina . Para este propósito las actividades del programa son representadas en un gráfica al igual que las actividades de la máquina . Un programa estará mejor acoplado a la arquitectura mientras mayor sea el **isomorfismo** existente entre ambas gráficas.

7. Lenguajes de programación.

Existen varios tipos de lenguajes de programación que pueden usarse - los imperativos (por ejemplo FORTRAN , Pascal) , los funcionales (por ejemplo LISP) y los lógicos (por ejemplo PROLOG).

8. Naturaleza de la Tecnología .

La tecnología VLSI es actualmente la más usada para la construcción de un gran número de procesadores . Los nuevos elementos de computación con tecnología de óptica holográfica seguramente causarán un nuevo impacto en la arquitectura de las nuevas computadoras.

Clasificación de los lenguajes de programación en la programación en paralelo.[7]

Lenguajes Control- Driven

Los lenguajes con orientación control-driven son clasificados de la siguiente manera :

1) Lenguajes Paralelos Síncronos.

[7]Ver ref. 1,2,10

2) Lenguajes Paralelos Asíncronos.

Lenguajes Paralelos Síncronos . Los lenguajes paralelos síncronos son aquellos que manejan tareas mutuamente independientes o no interactivas. Estos son usados para el desarrollo de los mismos cálculos sobre diferentes conjuntos de información . También son conocidos como 'lenguajes de supercomputadora ' ya que son usados para el procesamiento de grandes cantidades de información en paralelo en las supercomputadoras .

La mayoría de los lenguajes de supercómputo están basados en FORTRAN. Las modificaciones son hechas a FORTRAN para la ejecución de expresiones en paralelo (también llamado 'vectorización'). Aquí no hay facilidades especiales para la comunicación entre los procesos . La sincronización es realizada mediante el uso de expresiones paralelas para los ciclos DO y otras asignaciones.

Lenguajes paralelos -concurrentes Asíncronos . Los lenguajes Asíncronos son usados para la programación concurrente y distribuida . Por lo tanto se necesitan agregar más instrucciones que las disponibles para los lenguajes de programación secuenciales:

1) Una forma para separar secciones de código que pueda ser ejecutado en paralelo.

2) Un método que diga si un proceso puede o no ser ejecutado.

3) Un medio que garantice la exclusión mutua de dos procesos para compartir información al mismo tiempo.

Este mecanismo previene el uso de una variable compartida por más de dos procesos al mismo tiempo.

4) Sincronización de procesos usando tiempos de espera.

5) Mecanismos para detener un proceso por un determinado tiempo.

6) Asignación de prioridades a los procesos.

Los lenguajes asíncronos caen dentro de las siguientes tres clases:

1) Lenguajes orientados a procedimientos ;

2) Lenguajes orientados a mensajes;

3) Lenguajes orientados a operaciones;

Los lenguajes orientados a procedimientos son extensiones de los lenguajes secuenciales orientados a procedimientos , tal como Pascal. En estos lenguajes la interacción de los procesos esta basado en variables compartidas . Entre los lenguajes de esta clase se encuentran Pascal , Modula-2 y Ada .

Los lenguajes orientados a mensajes proveen la comunicación entre procesos mediante el uso de instrucciones *send* y *receive* para mensajes. A esta clase pertenecen los lenguajes occam y CSP.

Los lenguajes orientados a operaciones proveen llamadas a procedimientos remotos como un medio primario para la interacción de procesos. Estos lenguajes combinan aspectos de los lenguajes orientados a procedimientos y mensajes. A este tipo pertenecen Ada y StarMod.

Lenguajes Data-Driven.

Ejemplos de este tipo de lenguajes son : PARLOG , PROLOG concurrente y VAL (Value Algorithmic Language) .

Semántica de los programas paralelos.

La definición completa de un lenguaje de programación esta dividida en sintaxis , semántica y pragmática. La sintaxis define las oraciones gramaticalmente válidas ; La semántica asigna el significado de estas oraciones a los respectivos dominios. La pragmática es concerniente al uso del lenguaje.

Un aspecto importante del lenguaje de programación es proveer a los usuarios con un método conciso y mecánico para interpretar el significado del programa y establecer su corrección . El decir que un programa esta correcto

significa que cumple con las especificaciones. En otras palabras , nosotros estamos interesados en propiedades tales como ,

1) ¿El programa calcula los valores correctos para todos los posibles valores de entrada?

2) ¿El programa se ejecuta y termina bien para todos los posibles valores de entrada ?

La corrección de programas secuenciales.

Existen dos métodos importantes desarrollados para la corrección de los programas secuenciales , estos son :

1) El método de la aserción inductiva.

2) El método funcional.

El método de la aserción inductiva.

El método de la aserción inductiva trabaja como sigue . Se revisa si los datos de entrada al programa son correctos ; también se revisa el resultado que el programa supuestamente debe dar . Un programa puede ejecutar un conjunto de datos D y puede obtener un conjunto de resultados P o terminar abruptamente su ejecución . Entonces un programa estará totalmente correcto con respecto al conjunto de datos D si este termina satisfactoriamente su ejecución obteniéndose un conjunto válido de resultados P.

El método Funcional.

El método funcional usa la definición recursiva para las funciones calculadas del programa. La recursión es una forma compacta y natural para probar funciones que en su mayoría de las veces llegaran a un punto fijo.

Corrección de programas paralelos.

En el caso de programas secuenciales , solo se necesita probar que termina bien el programa y arroja resultados correctos . Pero para el caso de programas paralelos , se necesitan considerar otras propiedades :

1) Programas Continuos.

Algunos programas concurrentes como los sistemas operativos y los sistemas de tiempo real como los sistemas de reservación de las aerolíneas supuestamente deben de correr siempre . Es decir una salida o interrupción en estos sistemas es un error . En estos casos debemos modificar la definición de un programa correcto , ya que estos programas no deben terminar.

2) Exclusión Mutua.

Necesitamos asegurarnos que en las regiones críticas haya una exclusión mutua entre los procesos.

3) Ausencia de Bloqueo muerto[8]

Cuando hay varios procesos concurrentes , todos estos procesos pueden entrar en una competencia por los mismos recursos . Entonces todos los procesadores pueden bloquearse.

4) Ausencia de Bloqueo vivo[9]

En los bloqueos muertos todos los procesos son suspendidos indefinidamente . Otra condición que puede ocurrir es que un proceso no termine de su procesamiento y continúe ejecutándose . Esto se llama **bloqueo vivo .**

---

[8]Deadlock
[9]Livelock

5) Accesibilidad.

Una propiedad complementaria para la exclusión mutua es la accesibilidad. La accesibilidad garantiza que todo proceso eventualmente ejecutará su región crítica.

6) Cerradura[10]

Algunas veces puede ocurrir que ciertos procesos no pueden ejecutarse y otros si esto se debe a que si se asignan prioridades a los procesos un procesos de baja prioridad nunca ganará el acceso a un recurso ocupado . Esto se conoce como **cerradura .** Uno debe asegurar la ausencia de cerradura.

7) Fairness[11]

Cuando hay una elección repetitiva sobre varias alternativas , fairness significa que ninguna alternativa puede posponerse eternamente.

Computación en Paralelo . Aspectos Complejos.[12]

Una de las consideraciones más importantes en la computación es el total de los recursos necesarios para solucionar un problema . Dos medidas comunes para los recursos computacionales usados por un algoritmo son el tiempo , relacionados con el numero de pasos ejecutados por el algoritmo , y el espacio , la cantidad de memoria usada por el algoritmo.

La complejidad computacional (i.e. el número de operaciones aritméticas de punto flotante) ha sido usada como medida del desempeño de los algoritmos seriales , pero esto no es apropiado para los algoritmos paralelos. Esto se debe a que las computadoras paralelas pueden soportar computación extra sin un costo adicional si la computación se organiza adecuadamente , las computadoras en paralelo esta sujeta a nuevos costos como podrían ser por comunicación y sincronización , y no se reflejan necesariamente en la complejidad computacional.

---

[10]Lockout

[11]Puede traducirse como imparcialidad , sin embargo el término en inglés es el más ocupado.

[12]En 1,9 y 7 se hace un análisis de algunos aspectos importantes de la programación en paralelo , tales como las limitaciones a las que se puede enfrentr un algoritmo durante su ejecución en paralelo.

Uno de las medidas más comunes para el desempeño de los algoritmos numéricos en paralelo es la velocidad , S , la cual se define como :

S = tiempo de ejecución usando un procesador / tiempo de ejecución usando n procesadores.

Esta medida S tiene la ventaja que se usa el tiempo de ejecución y , por lo tanto , incorpora la pérdida por sincronización y comunicación .

El desempeño de las computadoras secuenciales usualmente se mide por:

a) la velocidad computacional medida en millones de operaciones de punto flotante por segundo (MFLOPS) , y

b) el grado de ejecución de instrucciones de control , aritméticas y lógicas , medidas en millones de instrucciones por segundo (MIPS) .

En el caso de máquinas SIMD y MIMD esta velocidad computacional puede ser estimada . Sin embargo , el desempeño computacional real es difícil de estimar ya que varía de acuerdo al nivel de paralelismo que se ejecute y esto a su vez de la aplicación en particular , es decir es función de la formulación usada , de los algoritmos numéricos seleccionados , de la implementación en la computadora, del compilador y del sistema operativo usado , tan bien como de la arquitectura del hardware. Para máquinas con multiprocesadores vectoriales su desempeño se estima como la suma del desempeño de todos los procesadores , el nivel más bajo es un procesador sencillo.

La parte central del procesamiento en paralelo que usa un gran número de procesadores es el diseño de algoritmos que de alguna manera se pueda comparar su desempeño con el tiempo T_1 de ejecución de un algoritmo secuencial para un solo procesador. Idealmente , nosotros requerimos que un algoritmo paralelo para un problema que usa N procesadores un tiempo de T_N que estará relacionado con T_1 de la siguiente forma $T_N = T_1/N$. En otras palabras , basados en el sentido común . que un procesador múltiple con N procesadores independientes debería ser capaz de obtener la solución de un problema N veces más rápido que un procesador sencillo. Esto se llama '

velocidad ideal ' . En la práctica la razón de tiempos T_1/ T_N esta lejos de ser N por las siguientes razones :

1) Los procesadores compiten por las mismas rutas de comunicación con otros procesadores o por el acceso a memoria compartida.

2) Como la lectura y escritura simultánea de un archivo puede causar conflictos , los procesadores son forzados a esperar mediante una exclusión mutua.

3) Los procesadores necesita ser sincronizados condicionalmente cuando las diferentes tareas están siendo coordinadas.

4) El componente secuencial de un algoritmo limita la velocidad del proceso total; en otras palabras , si Ts y Tp son respectivamente el tiempo gastado por los componentes seriales y paralelos de un algoritmo en un procesador simple , entonces la velocidad máxima S_N a la que se puede llevar a cabo usando N procesadores en paralelo para el componente paralelo esta dado por :

Eficiencia de velocidad y complejidad de comunicación.[13]

Para determinar la eficiencia en la utilización del procesador , la siguiente medida puede ocuparse :

$E = T_1/ N T_N$.

donde T_1 es el tiempo de ejecución del algoritmo secuencial , N es el numero de procesadores usados y T_N es el tiempo del algoritmo en paralelo.

Sin embargo , esta medida puede llegar a ser poco practica , ya que todo algoritmo en paralelo tiene una fuerte dependencia en la arquitectura , la cual finalmente determina la comunicación entre los procesadores. Esto significa que para analizar la eficiencia de los algoritmos paralelos , la arquitectura del modelo computacional también tiene que ser especificada :

[13]Sun [7] da un conjunto de definiciones y formulas que pueden ayudar a calcular la eficiencia de los programas en paralelo , todo en base a tiempos de ejecución del mismo . Este artículo lo recomiendo para profundizar en aspectos de medición de desempeño de los algoritmos en máquinas paralelas , pero sale del alcance del presente trabajo.

Uno puede introducir las siguientes especificaciones de la arquitectura :

1) El sistema de procesadores en paralelo consiste de N procesadores individuales.

2) El problema computacional a ser resuelto por el sistema de procesadores en paralelo puede ser representado como una secuencia de operaciones aritméticas y lógicas.

3) Cada uno de los N procesadores puede ejecutar cualquier operación en cualquier instante.

4) Toda operación requiere la misma unidad de tiempo.

5) No hay conflictos en el acceso a la información.

La ley de Amdahl

En 1967 , Amdhal hizo la observación que actualmente es ampliamente conocida como la ley de Amdahl [14] : Si una computadora tiene dos modos de operación uno de alta velocidad y otro de baja velocidad , entonces el desempeño general será dominado por el modo más lento . La ley de Amdahl es fundamental al entendimiento del desempeño de las computadoras , en el capítulo 3 se tratará más a fondo .

Problemas de clase N (Nick).

Una pregunta básica que nos preguntamos en la computación en paralelo es:

¿Cuales problemas pueden resolverse substancialmente más rápido usando muchos procesadores en lugar de uno solo ?

---

[14]Ver Amdahl, G *The Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities*, Vol. 30 , pp. 483-485 , Proceedings of the American Federation of Information Processing Societies , Washington , D.C. , 1967 . Referencia citada en varios artículos y libros del tema.

Nicholas Pippenger (Pippenger ,1979) formalizó esta pregunta definiendo los tipos de problemas conocidos como de Clase N (Clase Nick) . Este tipo de problemas pueden ser resueltos mucho más rápidos en máquinas de Turing.

Los siguientes problemas caen dentro de esta clase :

1) inversión de matrices.

2) evaluación de determinantes.

3) operaciones aritméticas (+ , - , * , /) de números binarios.

4) búsquedas.

5) conectividad gráfica.

6) Encontrar el máximo común divisor de polinomios.

En [8] se describe claramente el algoritmo vectorizado para la multiplicación de matrices , es decir para un problema de clase N.

Clasificación de los sistemas de Computación en Paralelo.

De acuerdo a la clasificación de Flynn citado en [9] se tiene la siguiente descripción , la cual se basa en la concurrencia en el control de la instrucción y la concurrencia en la ejecución. Una corriente (stream) se define como una secuencia de instrucciones que es ejecutada por el procesador .

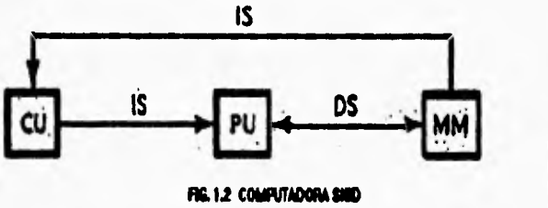
1. Máquinas de Flujo de Instrucción Sencilla , Flujo de Datos Sencillo (SISD).[15]

Estas máquinas incluyen a las computadoras seriales convencionales , en donde se ejecutan las instrucciones secuencialmente , una a la vez . Ver fig. 1.2.

[15]Single Instruction Stream , Single Data Stream Machines

2. Máquinas de Flujo de Instrucción Sencilla , Flujo de Datos Múltiple (SIMD).[16]

Estas son computadoras que tienen una unidad de control , una colección de procesadores idénticos , una memoria , y una red de interconexión que permite a los procesadores intercambiar datos . Durante la ejecución de un programa la unidad de control busca y descodifica las instrucciones para que se ejecuten en los procesadores . Cada procesador desarrolla la misma secuencia de instrucciones , pero usando diferentes datos. Ver fig. 1.3.



CU : UNIDAD DE CONTROL
PU : UNIDAD PROCESADOR
MM : MODULO DE MEMORIA
SM :MEMORIA COMPARTIDA
IS : FLUJO DE INSTRUCCIONES
DS : FLUJO DE INFORMACION

FIG. 1.2 COMPUTADORA SIMD

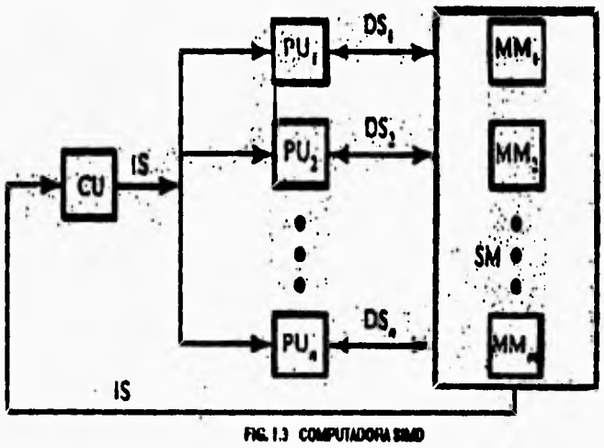


FIG. 1.3 COMPUTADORA SIMD

[16]Single Instruction Stream , Multiple Data Stream Machines

3. Máquinas de Flujo de Instrucción Múltiple , Flujo de Datos Sencillo (MISD).[17]

Esta categoría es únicamente formal para completar esta clasificación , pero no existe ninguna máquina con estas características. Ver fig. 1.4.

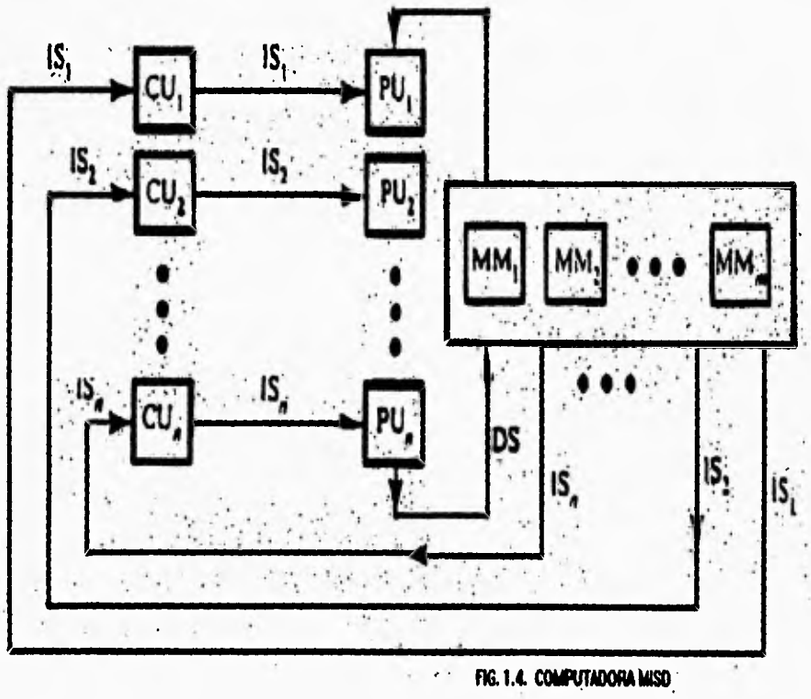


FIG. 1.4. COMPUTADORA MISD

[17]Multiple Instruction Stream , Single Data Stream Machines

4. Máquinas de Flujo de Instrucción Múltiple , Flujo de Datos Múltiple (MIMD).[18]

Estas son computadoras que contienen un determinado número de procesadores , cada uno de los cuales es programable y puede ejecutar sus propias instrucciones . Las instrucciones para cada procesador puede ser las mismas o diferentes . Los procesadores operan sobre una memoria compartida, generalmente en una manera asíncrona. Ver fig. 1.5 .

Estas máquinas generalmente se dividen , de acuerdo al nivel de interacción entre los procesadores y su localización física , en multiprocesadores y redes de multicomputadoras .

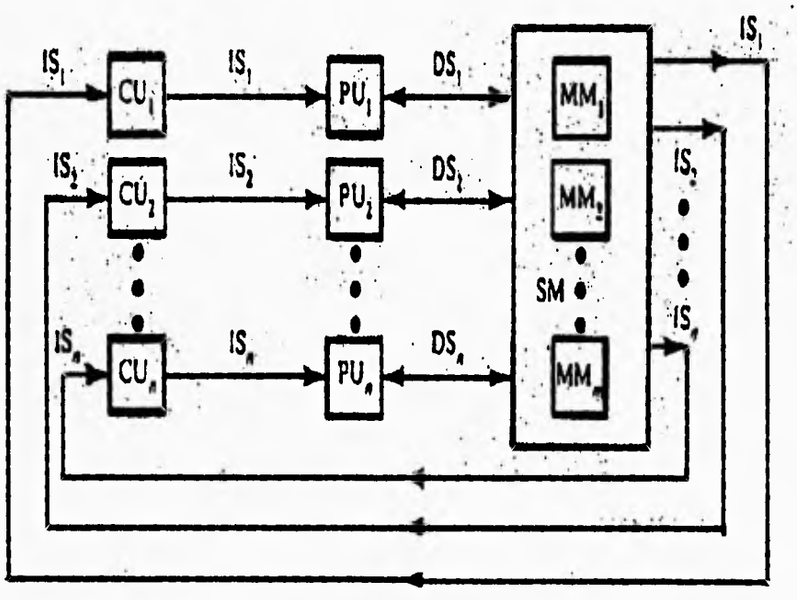


FIG. 1.5. COMPUTADORA MIMD

[18]Multiple Instruction Stream , Mulliple Data Stream Machines.

Referencias.

1. Krishnamurthy ,E.V. Parallel Processing Principles and Practice; Addison Wesley;1988

2. Perrot , Ronald H. Parallel programming / R.H. Perrot ;Workingham , England : Adison Wesley, 1987

3. Treleaven , Philip C."Parallel architecture overview " en Parallel Computing 8 (1988) 59-70 North - Holland

4. Basu , A."Parallel processing systems : a nomenclature based on their characteristics " en IEE Proceedings Vol. 134 , No. 3 May 1987.

5. Romero Salcedo , Manuel." Programación paralela en OCCAM " IIMAS UNAM

6. Worlton , Jack ." Toward a taxonomy of performance metrics " en Parallel Computing 17 (1991) 1073-1092 North-Holland

7. Sun , Xian-He y Gustafson , John. " Toward a better parallel performance metric" en Parallel Computing 17 (1991) 1093 - 1109 North- Holland

8. Hayes , Linda y Devloo , Phillippe. "A vectorized version of a sparse matrix vector multiply " en International Journal for Numerical Methods in Engineering , Vol. 23 1043 - 1056 (1986)

9. Noor , Ahmed. "New computing systems and their impact on structural analysis and design " en Supercomputing in Engineering Structures . Springer - Verlag,1989

10. Programming parallel processors / ed by Robert G, Babb. --Reading Massachusetts:Adison Wesley , 1988

11.Memoria del Taller Internacional de Procesamiento en Paralelo 1994 IIMAS , UNAM

12. Biegler , L. Theodor , Ph. D. Thesis , University of Wisconsin.

Capitulo 2

Aplicaciones del procesamiento en paralelo.

Como se mencionó en el capítulo anterior existen problemas de aplicación susceptibles para su resolución por medio del procesamiento en paralelo en las cuales se necesita un gran poder de cómputo y alta velocidad . Las áreas en las que ha tenido una gran aplicación son robótica , inteligencia artificial , procesamiento de señales , mecánica de fluidos , simulación y control de procesos químicos. En muchas de estas áreas se trabaja con **sistemas en tiempo real** . En este capítulo se explicará este concepto y se darán algunos ejemplos concretos de aplicaciones trabajadas con procesamiento en paralelo .

Definición de un sistema en tiempo real.

El diccionario Oxford de Computación da la siguiente definición de un sistema en tiempo real :

Cualquier sistema en el cual el tiempo en el que la salida se produce es importante. Esto se debe a que la entrada corresponde a algún movimiento en el mundo físico , y la salida esta relacionada con el mismo movimiento . El lapso de tiempo entre la entrada y la salida debe ser lo suficientemente pequeña .

Young (1982)[1] define el sistema en tiempo real como :

Cualquier actividad de procesamiento de información o sistema el cual tiene que responder a una entrada generada externamente por un estimulo dentro de un período finito y específico.

Es decir , estas definiciones cubren una gran variedad de actividades computacionales. Consecuentemente , un buen sistema en tiempo real depende no solo de los resultados lógicos de la computación , sino también del tiempo en que estos resultados se producen . Los principales diseñadores en el campo de sistemas en tiempo real frecuentemente distinguen entre sistemas de tiempo real rígidos (hard) y flexibles (soft) . Los **sistemas en tiempo real rígidos** son aquellos donde es absolutamente imperativo que la respuesta ocurre dentro del tiempo especificado . Los **sistemas en tiempo real flexibles** son aquellos donde los tiempos de respuesta son importantes , pero el sistema funcionará correctamente aún si los tiempos de respuesta fallan ocasionalmente . Por ejemplo, un sistema de control de vuelo de un avión de combate es un sistema en tiempo real rígido porque una falla en la respuesta puede ocasionar un desastre , mientras que un sistema de adquisición de datos para una aplicación de control de procesos es flexible ya que puede funcionar con un sensor a intervalos regulares , pero puede tolerar retrasos intermitentes .

Aplicaciones de los sistemas en tiempo real.[2]

-Control de Procesos.

-Robótica

-Control de calidad.

-Manejo automatizado de materiales.

-Transacciones financieras.

---

[1] Citado por Burns [1]
[2] Ver 1,2,4 y 5

-Manejo de Energéticos.

-Análisis de fallas en los sistemas y diagnosis.

-Simulación de procesos para el diseño y entrenamiento.

-Adquisición y grabación de información.

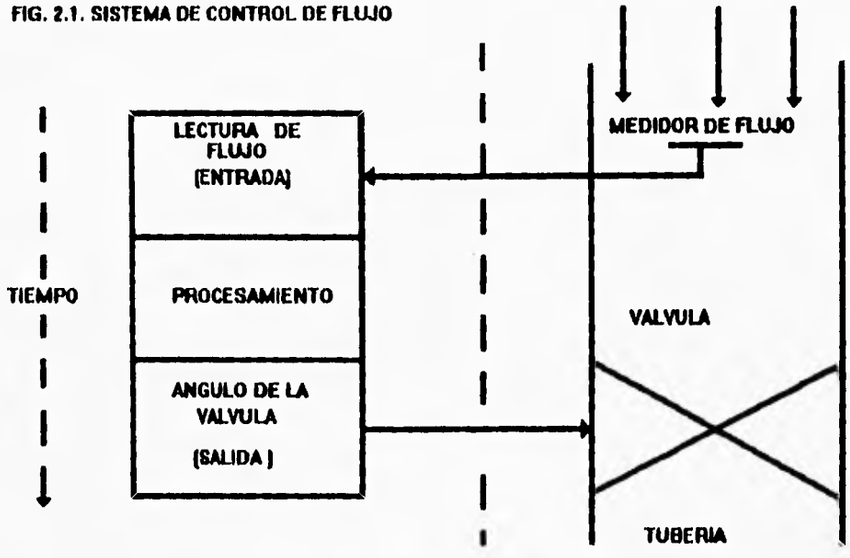
Ejemplos de sistemas en tiempo real .[3]

Control de procesos.

El primer uso de una computadora en un gran sistema de ingeniería ocurrió en la industria de control de procesos a comienzos de la década de 1960 . Hoy día el uso de microprocesadores es una norma . Considere el siguiente ejemplo ,que se muestra en la fig. 2.1 donde la computadora desarrolla una actividad sencilla : el asegurar el flujo regular de un líquido en una tubería controlando la válvula . En la detección de un incremento en el flujo la computadora debe responder alterando el ángulo de la válvula ; esta respuesta debe ocurrir dentro de un período finito si el equipo que recibe la señal no se encuentra sobrecargado . Es importante hacer notar que la respuesta puede envolver complejos cálculos y tiempo de computación antes de calcular el nuevo ángulo de la válvula.

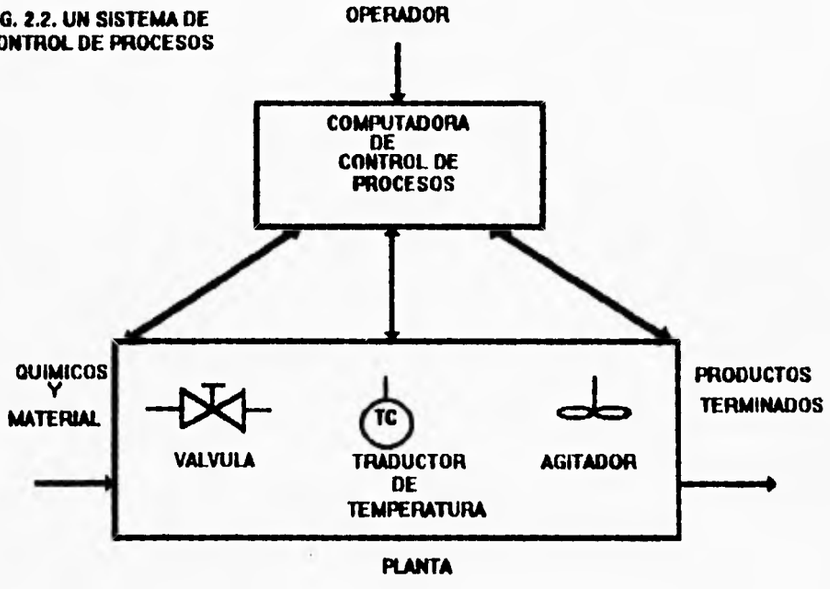
[3] Ver 1,4,5 y 10

FIG. 2.1. SISTEMA DE CONTROL DE FLUJO



Este ejemplo muestra solo un componente de un gran sistema de control . La figura 2.2 ilustra el papel de una computadora en tiempo real incrustada en un ambiente completo de control de procesos. La computadora interactua con el equipo usando sensores y elementos finales o ejecutores . Una válvula es un ejemplo de ejecutor y un traductor de temperatura es un ejemplo de sensor . (Un traductor es un dispositivo que genera una señal eléctrica que es proporcional a la cantidad física a ser medida.) La computadora controla la operación de los sensores y ejecutores para asegurar que la operación de la planta se desarrolle en los tiempos apropiados . Donde es necesario , que haya convertidores analógicos digitales y digitales analógicos entre el proceso que se controla y la computadora.

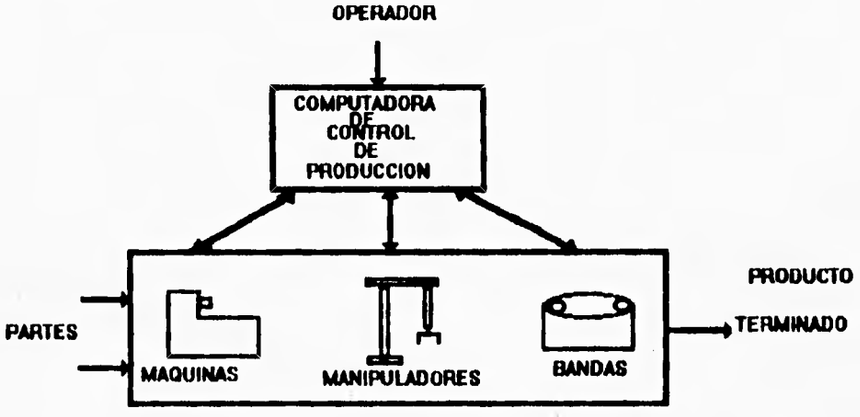
FIG. 2.2. UN SISTEMA DE CONTROL DE PROCESOS



Manufactura.

El uso de la computadora en la manufactura ha llegado a ser esencial en los últimos años para abatir costos de producción y elevar la productividad . Las computadoras han permitido la integración del proceso de manufactura completo desde el diseño del producto a la fabricación . Es en esta área del control de la producción donde los sistemas incrustados son ejemplificados de la mejor manera . La figura 2.3 representa , con un diagrama , el papel del control de la producción por computadora en el proceso de manufactura . El sistema físico consiste en una gran variedad de dispositivos mecánicos tales como máquinas herramientas , manipuladores y cintas transportadoras , todas ellas necesitan estar controladas y coordinadas por la computadora.

FIG. 2.3. UN SISTEMA DE CONTROL DE PRODUCCION



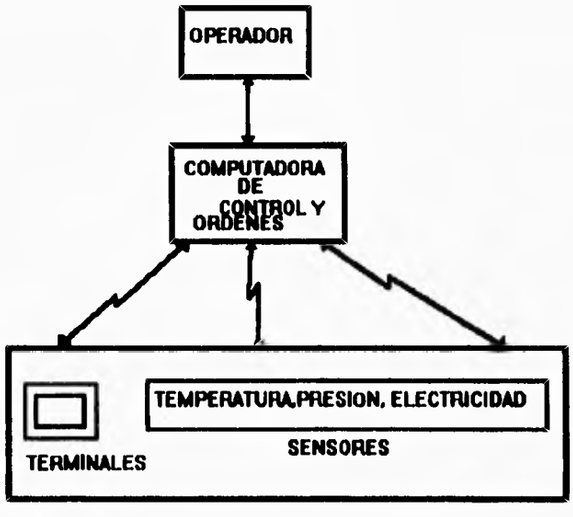
Comunicación , mando y control.

Aunque Comunicación , Mando y Control es un término de la industria militar estadounidense , hay un amplio rango de aplicaciones que exhiben características similares ; por ejemplo , las reservaciones de las aerolíneas , centros médicos para la automatización de los cuidados de los pacientes , control de tráfico aéreo y manejo de cuentas bancarias remotas . Todos estos sistemas consisten de un complejo conjunto de políticas , compartición de los dispositivos de información y procedimientos administrativos los cuales permiten la toma de decisiones , y proveen los medios para que puedan ser implementados . Frecuentemente , los dispositivos compartidos de información y los instrumentos requeridos para la implementación de decisiones están distribuidos sobre un amplia área geográfica . La figura 2.4 representa este sistema.

Sistemas computacionales incluidos o incrustados.

La computadora interactua directamente con el equipo físico en el mundo real . Para poder controlar estos dispositivos del mundo real la computadora necesitará muestrear las medidas que arrojen los diferentes dispositivos , por lo tanto , se requiere un reloj de tiempo real . Usualmente también hay una consola de operador que permite una intervención manual . El operador humano se mantiene constantemente informado del estado del sistema mediante el despliegue en pantalla de varios tipos de información incluyendo gráficas .

FIG. 2.4. UN SISTEMA DE CONTROL Y ORDEN

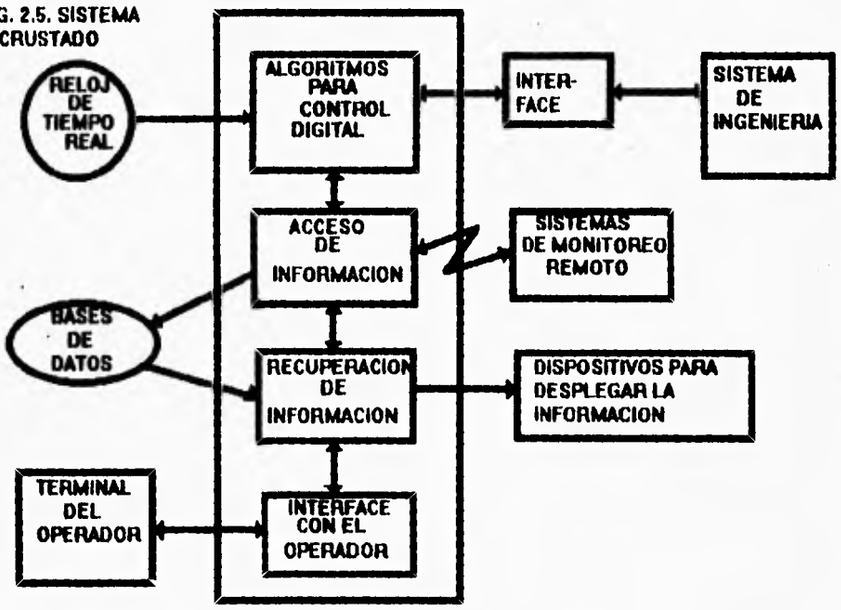


Los registros de los cambios en el estado del sistema también se almacenan en una base de información , que pueden consultar también los operadores . Esta información puede usarse para tomar decisiones en la corrección de fallas . Por ejemplo en las industrias químicas , el monitoreo de la planta es esencial para maximizar las ventajas económicas más que para maximizar la producción . Las decisiones que conciernen a la producción en una

planta pueden tener serias repercusiones para otras plantas en sitios remotos , particularmente cuando los productos de un proceso son usados como materia prima en otros.

Un típico ejemplo de sistema computacional incluido esta representado en la figura 2.5 . El software que controla las operaciones del sistema puede estar escrito en módulos que reflejen la naturaleza física del ambiente . Usualmente habrá un modulo que contenga los algoritmos para controlar físicamente los dispositivos , un modulo responsable para la grabación de los cambios del sistema , un modulo para recuperar y desplegar esos cambios y un modulo para interactuar con el operador.

FIG. 2.5. SISTEMA INCRUSTADO



Características de los sistemas en tiempo real .[4]

Un sistema en tiempo real posee muchas características especiales . Desde luego , no todos los sistemas en tiempo real exhibirán todas estas características .

Tamaño y complejidad.

Se dice que la mayoría de los problemas asociados con el desarrollo de software son aquellos que se relacionan al tamaño y la complejidad . Escribir programas pequeños no presenta un problema especial , ya que estos pueden ser diseñados , codificados , mantenidos y entendidos por una sola persona . Si esa persona deja la compañía o institución que esta usando el software , entonces alguien más puede aprender el programa en un periodo de tiempo relativamente corto .

Los sistemas en tiempo real incrustados , deben responder por definición a variaciones del mundo real . Esta propiedad intrínseca de estos sistemas los lleva a ser bastante grandes y generalmente están bajo cambios continuos . El costo de rediseñar o reescribir el software para satisfacer a la respuesta de los requisitos del constantemente cambiante mundo real se eleva a precios exorbitantes . Por lo tanto los sistemas en tiempo real están bajo constante mantenimiento durante su tiempo de vida .

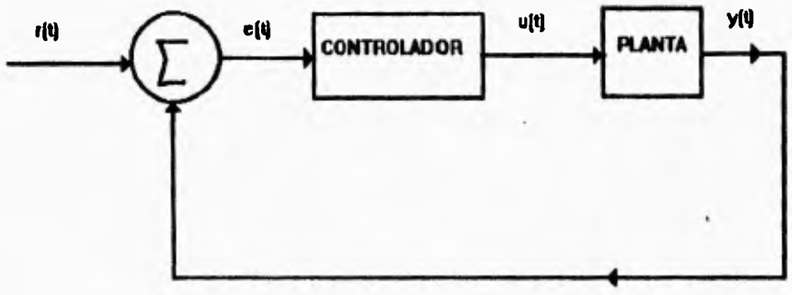
Manipulación de números reales.

Desde luego muchos sistemas en tiempo real pretenden realizar el control de alguna actividad de ingeniería . La figura 2.6 ejemplifica un sistema de control simple . La entidad controlada , la planta , tiene un vector de variables de salida , ***y***. que cambia con el tiempo , por lo tanto es ***y(t)*** . estas salidas se comparan con una señal de referencia ***r(t)*** para producir una señal de error ***e(t)*** . El controlador usa este vector para cambiar las variables de entrada a la planta ***u(t)*** . Para un sistema muy simple el controlador puede ser un dispositivo analógico trabajando en una señal continua.

---

[4]Ver 1,4 y 5

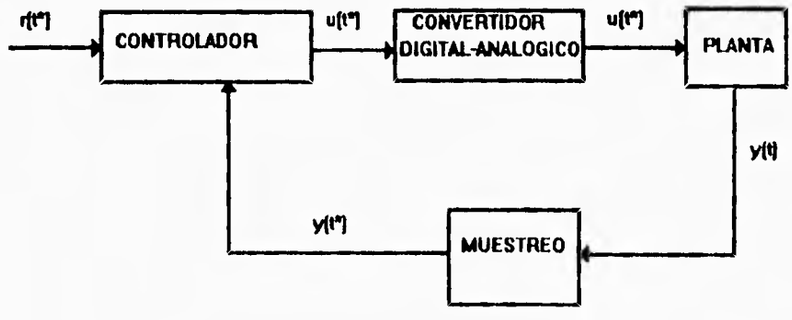
FIG. 2.6. CONTROLADOR SIMPLE



La figura 2.6 ilustra un controlador por retroalimentación . Para calcular que cambios deben hacerse en las variables de entrada , para que se realice el efecto deseado en las variables de salida , es necesario tener un modelo matemático de la planta . La derivación de estos modelos concierne a la disciplina de simulación de procesos . Frecuentemente una planta es modelada como un conjunto de ecuaciones diferenciales de primer orden . Esto liga la salida del sistema con el estado interno de la planta y sus variables de entrada . Cambiar la salida de la planta implica la resolución de estas ecuaciones para dar los valores de entrada requeridos . La mayoría de los sistemas físicos exhiben inercia así que el cambio no es instantáneo .El requisito del tiempo real de moverse a un nuevo punto (set point) dentro de un período de tiempo determinado agrega una mayor complejidad al modelo matemático y al sistema físico . El hecho de que en realidad , las ecuaciones diferenciales de primer orden son solo aproximaciones a las características reales del sistema también presenta complicaciones.

A causa de estas dificultades , la complejidad del modelo y el número de las distintas entradas y salidas , la mayoría de los controladores se implementan en computadoras . La introducción de un componente digital dentro de un sistema cambia la naturaleza del ciclo de control . La figura 2.7 es una adaptación del primer modelo mostrado en la figura 2.6 . Pero ahora las variables toman valores discretos ; el muestreo y el manejo de la operación se lleva a cabo por un convertidos analógico-a-digital , que están bajo el control de la computadora.

FIG. 2.7. CONTROLADOR SIMPLE COMPUTARIZADO



Dentro de la computadora las ecuaciones diferenciales pueden resolverse por técnicas numéricas , aunque los algoritmos necesitan ser adaptados para tomar en cuenta el hecho de que las salidas de la planta están ahora siendo muestreadas. No pretendo presentar como diseñar un algoritmo de control pero si remarcar como se implementa y que sus requerimientos de volumen de cálculos , naturaleza de las tareas y rápida velocidad los convierten en candidatos idóneos para la implementación del procesamiento en paralelo. Estos son matemáticamente complejos y requieren un alto grado de precisión.

Confiabilidad y seguridad.

Mientras la sociedad relega el control de funciones vitales a las computadoras , se vuelve más importante el hecho de que las computadoras y sus programas no fallen .

Por ejemplo la falla en un sistema bancario puede acarrear la pérdida de millones de nuevos pesos ; la falla de corriente eléctrica en una unidad de cuidados intensivos pude provocar la muerte del paciente y un paro prematuro de una planta química puede provocar severos daños al equipo o algún accidente. Estos ejemplos pueden lucir un poco dramáticos , pero ilustran el hecho de que el hardware y software debe ser confiable y seguro . Más aún cuando hay alguna interacción con un operador la interface debe de estar diseñada para minimizar la posibilidad y efecto de los errores humanos .

Control concurrente de los componentes de un sistema.

Un sistema incrustado consiste de computadoras y muchos elementos externos con los cuales los programas de la computadora deben interactuar . La naturaleza de los eventos del mundo real es su existencia en paralelo .

Afortunadamente , la velocidad de las computadoras modernas es tal que usualmente estas acciones pueden llevarse a cabo en forma secuencial , pero dando la ilusión de que ocurren simultáneamente .

En algunos sistemas incrustados , esto no es posible , por ejemplo donde la recolección de la información y su procesamiento se encuentran geográficamente en sitios alejados o simplemente donde el tiempo de respuesta de los componentes individuales no los pueda cubrir eficientemente una sola computadora. En este caso es necesario considerar sistemas incrustados con multiprocesamiento distribuido.

El mayor problema asociado con la producción de software para estos sistemas que exhiben concurrencia es como expresar esta concurrencia en la estructura de un programa. Una aproximación para la solución de este problema es dejar que un solo programador se encargue de construir este sistema , desde

luego esto es inimaginable ya que hay innumerables razones adversas que se pueden encontrar en los libros de diseño de sistemas , como podrían ser :

- Complica la ya difícil tarea del programador de hacer consideraciones adicionales de las estructuras del programa.
- El resultado son programas obscuros y difíciles de comprender .
- Hace que la corrección y la prueba del programa sea más difícil.
- La ejecución en paralelo del programa en más de un procesador es más difícil de llevarse a cabo.

Facilidades del tiempo real.

El tiempo de respuesta es crucial en cualquier sistema incrustado. Desafortunadamente , es difícil diseñar e implementar sistemas que garanticen que la salida apropiada será generada en los tiempos estipulados en todas las condiciones posibles. Por esta razón los sistemas en tiempo real son construidos con procesadores con capacidad reservada para asegurar que no se produzcan retrasos durante los períodos críticos de la operación del sistema .

El programador debe de contar con un poder de procesamiento , con lenguaje y con un soporte del tiempo real (sensores , ejecutores) adecuados además de una excelente comprensión del fenómeno físico esto para permitirle :

- Especificar tiempos dentro de los cuales las acciones tienen que desarrollarse
- Especificar tiempos dentro a los cuales las acciones deben completarse.
- Responder a situaciones donde *todos* los requisitos de tiempo no pueden cumplirse.
- Responder a situaciones donde los requisitos de tiempo cambian dinámicamente.

Lo anterior se le conoce como facilidades de control de un sistema en tiempo real.

Interacción con las interfaces del hardware.

La naturaleza de los sistemas incrustados requieren que la computadora interactue con el mundo externo . Ellos necesitan sensores y ejecutores para una gran variedad de dispositivos del mundo real . Estos dispositivos interactuan con la computadora vía registros de entrada y salida y sus requisitos para operar dependen de la computadora. Los dispositivos pueden también generar interrupciones para avisar al procesador que existe un error de operación .

En el pasado , la interface con los dispositivos los controlaba el sistema operativo , o los programadores los codificaban en lenguaje ensamblador . Hoy en día , debido a la variedad de dispositivos a la naturaleza crítica que juega el tiempo. su control debe ser directo a través del programa .

Implementación eficiente.

Como ya se ha mencionado , el tiempo es crítico en los sistemas de tiempo real , por lo tanto su eficiencia de implementación es más importante que en otros sistemas . Es aquí donde hay que señalar que uno de los beneficios de los lenguajes de alto nivel es que permite al programador concentrarse en la resolución del problema más que en los detalles de implementación . Por ello , la selección del lenguaje es también muy importante .

Inteligencia Artificial .[5]

Una de las áreas que mayor provecho pueden obtener del procesamiento en paralelo es sin duda aquella relacionada a la Inteligencia Artificial (IA) . Este campo es una subdivisión de las ciencias computacionales cuyo propósito es crear software y hardware que pueda producir resultados iguales o mejores que aquellos producidos por los humanos . El software de IA le permite a la

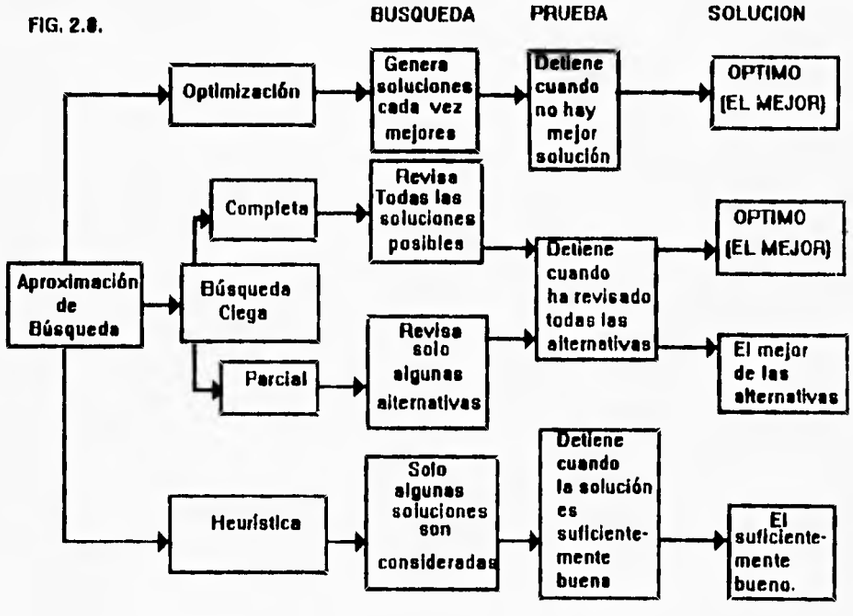
[5] Ver 2 y 6

computadora duplicar algunas funciones que realiza el cerebro humano , de una forma limitada .

Dentro de la tecnología de IA se encuentran los **sistemas expertos** , los cuales son un conjunto de programas que pretenden imitar el proceso de razonamiento y conocimiento de los expertos en la solución de problemas específicos . Los sistemas expertos son de gran interés para las organizaciones ya que ayudan a mejorar y aumentar la productividad .

Los sistemas expertos tratan de automatizar el proceso de **resolución de problemas** que se asocia con las criaturas capaces de pensar . La figura 2.8 muestra las aproximaciones de búsqueda formales que pueden existir para llegar a la solución de un problema.

FIG. 2.8.



Descripción de la figura 2.8 :

Optimización .

La Optimización pretende encontrar la mejor solución posible usando fórmulas matemáticas que modelen situaciones específicas . La optimización debe ser estructurada , y se realiza usando algún algoritmo o fórmula . Como generalmente el algoritmo genera paso a paso un proceso de búsqueda y evalúa si se puede mejorar esa solución.

Búsqueda ciega.

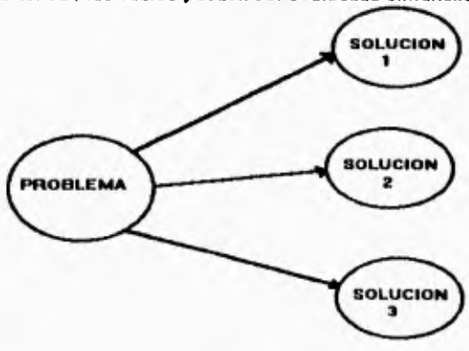
Se proponen varias soluciones posibles al problema y se evalúan cada una de ellas a fin de obtener la solución del conjunto de soluciones que se propone .

Búsqueda Heurística .

Para muchas aplicaciones , es posible encontrar información que nos guíe durante el proceso de búsqueda y así reducir el monto de computación.

El procesamiento en paralelo puede aplicarse en el proceso de búsqueda de la solución . Una búsqueda de la solución puede llevarnos a un árbol de soluciones como el que se muestra en la figura 2.9 .

FIG. 2.9. ARBOL DE SOLUCIONES. Muestra como un problema puede tener varias soluciones , las cuales pueden ser evaluadas simultáneamente.



Aquí cada nodo representa una posible solución al problema , mientras que un algoritmo secuencial evaluaría nodo por nodo de cada rama , un algoritmo paralelo evaluaría la mayor cantidad posible de soluciones al mismo tiempo .

Además del proceso de búsqueda el procesamiento en paralelo es particularmente importante en los sistemas expertos de tiempo real .

Sistemas Expertos en tiempo real .

A principios del capítulo se describieron los sistemas en tiempo real y sus características . Los sistemas expertos en tiempo real son la evolución natural de los sistemas tradicionales . Su grado de complejidad aumenta notablemente en tres aspectos :

1) El número de funciones controladas.

2) El grado en que estas funciones tienen que ser controladas .

3) El número de factores que deben considerarse antes de la decisión .

(Además deben considerarse las características de los sistemas en tiempo real que se vieron a principios del capítulo.)

Por ejemplo , los cuartos de control en plataformas petroleras pueden generar 20,000 señales para dos o tres operaciones , y esto desde luego dificulta a la gente interpretar y actuar de manera instantánea. Así los sistemas expertos están siendo diseñados para asistir a los operadores humanos [2].

Los sistemas Expertos en tiempo real , al igual que los sistemas en tiempo real , obtienen la información directa del proceso que ellos controlan . Ellos toman un completo control sobre el proceso , en un tiempo real , sin la intervención humana.

Aplicaciones prácticas.

Los Transputers.[6]

Transputer constituye un término genérico que describe a una familia de dispositivos programables que incluyen controladores de disco , procesadores de punto flotante , de gráficas , dispositivos de procesamiento de señales y procesadores de propósito general de 16 , 32 , y 64 bits.

El Transputer fue lanzado en 1985 por Inmos Inc. como el primer microprocesador disponible comercialmente para formular los principios del procesamiento en paralelo . Desde ese entonces , se han desarrollado diversas y numerosas aplicaciones que involucran el uso del lenguaje OCCAM y los Transputers.

Estas aplicaciones han surgido en Áreas tales como : inteligencia artificial , robótica , análisis numérico , reconocimiento de voz y señales , procesamiento en tiempo real , procesamiento de imágenes , simulación , sistemas operativos , control industrial , graficación , física , etc.

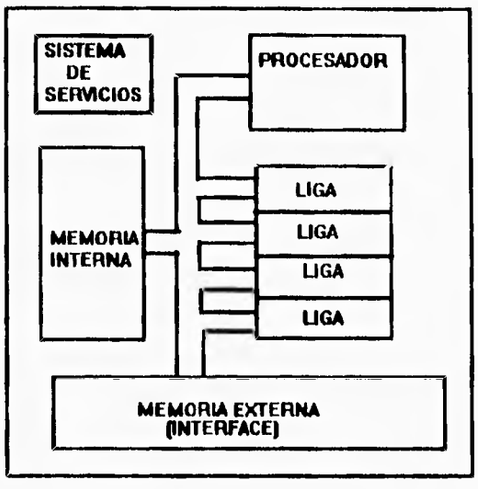
La arquitectura general de un Transputer esta compuesta de memoria interna de muy rápido acceso , un conjunto de ligas de comunicación para la conexión directa con otros Transputers (generalmente cuatro) , un procesador , una interface de aplicación específica para agregar más memoria externa y un módulo de servicios del sistema.

---

[6] Ver 3,4 y 7

Características del Transputer Inmos T8xx (fig. 2.10) :

FIG. 2.10. DIAGRAMA DE BLOQUES DE LA FAMILIA DE TRANSPUTER T8XX



-Espacio de direcciones 4Gb

-Unidad de punto flotante (3.3 Mflops)

-4 ligas de comunicación serial (20 Mbit / s)

-4 kbytes RAM

-es una máquina de procesamiento distribuido en un chip

-la familia completa consta de : T805 32 bit , T425 32 bit , T225 16 bit.

El Transputer Inmos T9000

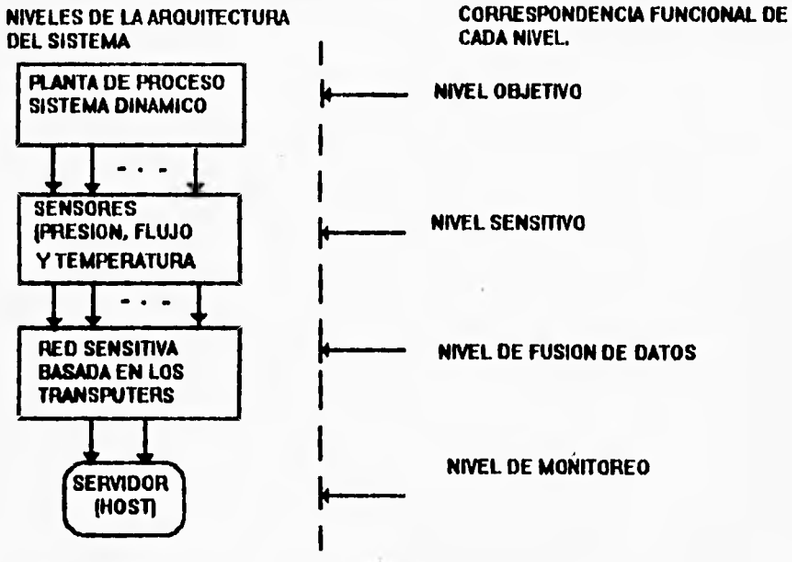
-Tiene un mejor desempeño (16 kbyte Cache y acepta instrucciones de tipo pipeline)

-Tiene un canal de comunicación Virtual .

Monitoreo de una Planta de Proceso basado en una red de sensores usando transputers.

Yang y Hugh (1990) [8] desarrollaron este sistema práctico de procesamiento en paralelo usando transputers en el monitoreo de una planta nuclear , que incorporaba 4 bombas , tres calentadores , dos intercambiadores y 56 válvulas de control . Se probó la eficacia de la implementación de algoritmos descentralizados para adquisición de datos con varios sensores en un sistema dinámico real . El sistema desarrollado tenía las siguientes características fig. 2.11:

FIG. 2.11. CONFIGURACION DEL SISTEMA



-Cada sensor recibía grandes cantidades de información.

- Cada nodo contenía un modelo del sistema parcial.

- Cada nodo podía ser conectado a una diferente topología.

- Cada nodo incorporaba funciones de procesamiento , por lo que no se requería un procesador central.

- La planta al proveer facilidades del control de fallas permitió evaluar los algoritmos descentralizados .

Aplicaciones Aeroespaciales.

Thompson (1994) [4] menciona que hay dos formas en las que el procesamiento en paralelo puede ser explotado en las aplicaciones aeroespaciales. El primero es usar el procesamiento en paralelo para mejorar la velocidad de ejecución de las tareas por medio de la división de las tareas y que estas se lleven a cabo simultáneamente en los elementos de procesamiento. La segunda es explotar el procesamiento en paralelo para introducir tareas de tolerancia de fallas y control en caso de emergencia (safety-critical control task).

La aplicación desarrollada fue el control de una turbina de gas . El control se llevó a cabo usando una aproximación heurística y reportó que las aproximaciones originales en el control de la turbina consumía el 61% del tiempo de ejecución del sistema solo en esta. Con la implementación del paralelismo este porcentaje se redujo a 35% . El sistema fue implementado sobre Transputers T800- 20 .

Simulación en Paralelo.

La simulación de sistemas continuos complejos ha tenido influencia en la evolución de los sistemas de cómputo en cuanto a su velocidad y desempeño. Para mejorar estos aspectos fue como se hicieron dos innovaciones importantes a la arquitectura de Von Neumann que es : paralelismo y pipelining . Esta implementación es característica de los sistemas de supercómputo .

El término "simulación en paralelo" es usado para dar significado a aquellas simulaciones hechas sobre una arquitectura en paralelo . Los usuarios de estas computadoras son motivados a usar las altas velocidades de procesamiento por dos razones : Se requiere resolver un modelo cada vez más complejo de sistemas que pueden incluir la solución de Ecuaciones Diferenciales

Parciales y la simulación de sistemas en tiempo real con un tiempo de respuesta mejor .

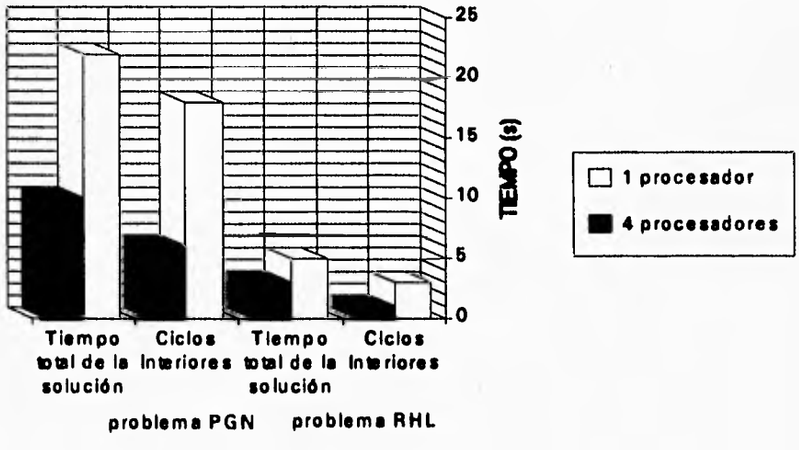
Camarda y Stadtherr (1993) [9] realizaron una comparación del efecto en el tiempo de ejecución del uso de múltiples procesadores en una Cray X-MP . Los problemas fueron los siguientes :

1) Planta de Recuperación de hidrocarburos ligeros (RHL) , sistema con 1477 variables.

2) Planta de procesamiento de Gas Natural (PGN) , sistema con 1235 variables .

Los resultados obtenidos en su trabajo se muestran en la gráfica 2.12

EFECTO DE PROCESADORES MULTIPLES SOBRE EL TIEMPO DE EJECUCION DE UN PROGRAMA EN UNA CRAY X-MP



Referencias.

1. Burns , Alan & Wellings, Andy . Real-Time systems and their programming languages . International Computer Science Series; Addison Wesley ,1992

2. Turban , Eraim. Expert Systems and Applied Artificial Intelligence ; Macmillan Publishing Company,1990

3. Romero Salcedo, Manuel. " Programación paralela en OCCAM " IIMAS UNAM

4. Memoria del Taller Internacional de Procesamiento en Paralelo 1994 IIMAS ,

5. Kerckhoffs , E.J. & Koppelaar , H..Toward Parallel Intelligent Simulation . Delft University of Technology . Delft , The Netherlands , 1989

6. Rice , J.P.Problems with Problem Solving in Parallel : The Poligon System . Stanford University , 1989

7. Graham , Ian .The Transputer handbook / New York : Prentince Hall , 1990

8. Yang Gao & Hugh , F. A Transputer-Based Sensing Network for Process Plant Monitoring en International Conference on Applications of Transputers (1989 : Liverpool , England) ed. por L. Freeman Amsterdam , 1990.

9. Camarda , Kyle & Stadherr , Mark. Exploiting Small-Grained Parallelism in Chemical Process Simulation on Massively Parallel Machines . AIChE Annual Meeting ,1993

10. Luyben , William. Process Modeling Simulation and Control for Chemical Engineers ; Mac Graw Hill , 1990.

11. Chemical Enginneering Progress Software Directory , 1994 .

Capitulo 3

Programación No- lineal.

I. INTRODUCCIÓN

El problema general de programación no lineal puede ser expresado como aquel donde se pretende encontrar la solución local x* de :

$$\begin{aligned} &\textit{minimizar} && f(\mathbf{x}) \quad \mathbf{x} \in \mathbf{R}^n \\ &\textit{sujeta a} && h_j(\mathbf{x}) = 0 \quad \mathrm{j} \in \mathbf{E} \\ & && g_j(\mathbf{x}) \geq 0 \quad \mathrm{j} \in \mathbf{I} \end{aligned} \qquad \text{f. 3.1}$$

donde **E** e **I** son conjuntos finitos de índices de las restricciones de igualdad y desigualdad respectivamente , y las funciones son en general no lineales. Este tema es de una gran importancia práctica y ha desarrollado una importante área de investigación en matemáticas.

Una observación importante es que para un problema dado , puede haber varias maneras de representar la forma 3.1 . Por ejemplo existen posibles pretransformaciones del problema para simplificar o mejorar el planteamiento

original . Estas ideas incluyen la eliminación de variables , agregar variables extra, y hacer transformaciones a las restricciones . En efecto la forma 3.1 es la representación más simplificada del problema[1].

Condiciones de Optimización .[2]

Considerando el problema 3.1.

Las siguientes son *las condiciones necesarias de optimización Fritz John (f. 3.2).* Si un punto **x** es una solución óptima local al problema anterior , entonces debe existir un vector diferente de cero $(u_o, \mathbf{u}, \mathbf{v})$ tal que

$$u_o \nabla f(\mathbf{x}) + \sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}) + \sum_{i=1}^{l} v_i \nabla h_i(\mathbf{x}) = 0$$
$$u_i g_i(\mathbf{x}) = 0 \quad \text{para i} \quad = 1, \ldots, m \qquad \text{f.3.2.}$$
$$u_o \geq 0, u_i \geq 0, \quad \text{para i} \quad = 1, \ldots, m$$

donde **u** y **v** son m y l vectores cuyos componentes i son ui y vi respectivamente. Aquí , uo , ui y vi se conocen como los *multiplicadores de Lagrange* asociados con la función objetivo las i restricciones de desigualdad gi(x) < 0 , y las i restricciones de igualdad hi(x)=0 . La condición $u_i g_i(\mathbf{x}) = 0$ es conocida como la condición complementaria de ajuste y establece que sea ui = 0 o gi(x)=0. Así si gi(x) < 0 , entonces ui= 0 . Siendo I el conjunto de restricciones de desigualdad obligatorias en **x** , esto es , I = {i: gi(**x**) = o } , las condiciones de Fritz John podrían escribirse en la siguiente forma equivalente. Si **x** es una solución óptima local al *problema 3.1* , entonces existe un vector diferente de 0 (uo,**ui** , **v**) que satisfacen la siguiente relación f.3.3. , donde ui es el vector de los multiplicadores de Lagrange asociados con gi(**x**)≤ 0 para i ∈ I .

[1] Fletcher , R. Methods for Nonlinear Constraints en [4]
[2] Las condiciones de Optimización son presentadas en casi toda la literatura consultada , si se quiere obtener referencias más amplias , consultar Bazaraa [3]

$$u_0 \nabla f(\mathbf{x}) + \sum_{i \in I} u_i \nabla g_i(\mathbf{x}) + \sum_{i=1}^{l} v_i \nabla h_i(\mathbf{x}) = 0$$
$$u_0 \geq 0, u_i \geq 0 \quad \text{para i} \quad \in I \qquad \text{f.3.3.}$$

Si uo= 0 , entonces las condiciones Fritz John son virtualmente inservibles , ya que esencialmente , ellos solo establecen que los gradientes de las restricciones de desigualdad estricta y los gradientes de las restricciones de igualdad son linealmente dependientes. Es posible referirse a ellas como , *requisitos de restricciones* , si se garantiza que uo es positivo , las condiciones de Fritz John se reducen a las condiciones de Kuhn -Tucker . Un requisito de restricción típico es que los gradientes de las restricciones de desigualdad para i∈ I y que los gradientes de igualdad sean linealmente independientes.

Las *condiciones necesarias de optimización de Kuhn-Tucker* f. 3.4.pueden establecerse como sigue. Si **x** es una solución local al Problema 3.1 , y tomando en cuenta los requisitos de restricción , existe un vector (**u,v**) tal que

$$\nabla f(\mathbf{x}) + \sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}) + \sum_{i=1}^{l} v_i \nabla h_i(\mathbf{x}) = 0$$
$$u_i g_i(\mathbf{x}) = 0 \quad \text{para i} \quad = 1, \ldots, m \qquad \text{f. 3.4.}$$
$$u_i \geq 0 \quad \text{para i} \quad = 1, \ldots, m$$

De nuevo , ui y vi son los *multiplicadores de Lagrange* asociados con las restricciones gi(**x**)≤0 y hi(**x**)=0 . A la expresión ui gi(**x**)=0 se conoce como *la condición complementaria de disminución* . Si I = {i:gi(**x**)=0} , entonces las condiciones de arriba pueden ser reescritas como f.3.5. :

$$\nabla f(\mathbf{x}) + \sum_{i \in I} u_i \nabla g_i(\mathbf{x}) + \sum_{i=1}^{l} v_i \nabla h_i(\mathbf{x}) = 0 \qquad \text{f. 3.5.}$$
$$u_i \geq 0 \quad \text{para i} \quad \in I$$

Bajo ciertas suposiciones de propiedades convexas , las condiciones de Kuhn-Tucker son también *suficientes* para la optimización . En particular ,

suponga que **x** es una solución viable al *Problema 3.1* y que se cumplen las condiciones de Kuhn-Tucker establecidas abajo f.3.6. :

$$\nabla f(\mathbf{x}) + \sum_{i \in I} u_i \nabla g_i(\mathbf{x}) + \sum_{i=1}^{l} v_i \nabla h_i(\mathbf{x}) = \mathbf{0}$$

$$u_i \geq 0 \qquad \text{para i} \qquad \in \text{ I}$$

f.3.6.

donde $I = \{i : g_i(\mathbf{x}) = 0\}$. Si *f* es pseudo convexa , gi es quasiconvexa para i ∈ I , y si hi es quasiconvexa si vi > 0 y quasiconcava si vi < 0 , entonces **x** es una solución óptima al Problema 3.1

Consideraciones Generales.

Históricamente (alrededor de 1960) , no había algoritmos numéricos efectivos para los problemas de optimización no lineales con restricciones , pero se empezaban a desarrollar buenas técnicas para la optimización sin restricciones. Por ejemplo , el método de la proyección del gradiente de Rosen (1960) (quizá la primera técnica efectiva para restricciones lineales) , el método de Davidon de la variable métrica fue formulado en 1959 y retomado en 1963 por Fletcher y Powell , y es todavía la base de muchos algoritmos de problemas sin restricciones.[3]

Los métodos para la solución de 3.1 son usualmente iterativos , generando una secuencia de soluciones x(1) , x(2), x(3) , ... a partir del punto x(1) y esperando que converja al punto óptimo x* . También son necesarias estimaciones de los multiplicadores de Lagrange en los métodos. En la concepción de un método para solucionar problemas de programación no lineal , son tres las ideas que se deben tener en mente : es decir como se tratarán las ecuaciones , las desigualdades , y los aspectos de minimización del problema .

No existe una elección que pueda considerarse la mejor para resolver este tipo de problema . Una primera aproximación es usar la eliminación de variables

[3] Conn ,A.R. Penalty Function Methods . en [4]

específicas . Otro es el método de *los multiplicadores de Lagrange* . Ambas aproximaciones requieren la solución de un sistema de ecuaciones no lineales . La linealización Sucesiva (i.e. algunas formas del método de Newton-Raphson) son usualmente las más eficientes aunque hay dificultades para garantizar la convergencia global. La aproximación alternativa es usar alguna forma de *función de penalidad* . Aquí el punto es reducir el problema a una optimización sin restricciones agregando a la función objetivo unos términos de peso de las restricciones. Existen dos tipos principales de función de penalidad ; *secuencial* , en la cual la solución se localiza como el punto límite de una secuencia de minimizaciones de una función de penalidad sin restricciones, y *exacta* en la cual x^* es el punto a minimizar de una función de penalidad simple. La eficiencia depende de los detalles de implementación y de como se use la función de penalidad , aunque una función de penalidad exacta es más eficiente . El uso de la linealización y de funciones de penalidad generan un tipo de métodos prácticos exitosos.

Los primeros métodos para manejar las restricciones de desigualdad incluían un término de barrera el cual restringía el dominio del problema al interior de una región viable. Estos métodos están en desuso , los métodos más modernos intentan identificar las restricciones activas en la solución y tratarlas como ecuaciones. Así el problema se trata como un problema combinatorio en donde se corrige el conjunto de índices discreto. En algunos métodos la selección de las restricciones activas se obtiene tomando las restricciones activas de un subproblema de desigualdad que estima el problema principal (la aproximación IQP) . En otros métodos el conjunto de índices de restricciones se obtiene de acuerdo a un subproblema de igualdad resuelto en cada iteración (la aproximación EQP). El conjunto de trabajo se cambia siempre que se tenga en claro que una restricción pueda borrarse o agregarse ; usualmente esto se lleva a cabo en cada iteración.

Existen muchos métodos buenos para resolver problemas de minimización sin restricciones y muchas de estas técnicas se ocupan en la programación no lineal, con métodos de Newton o cuasiNewton . De cualquier forma , a causa de la interacción de los efectos de curvatura de las restricciones , no es muy claro como

actualizar en cada iteración la información de dicha curvatura , aunque existen algunas buenas posibilidades , como los métodos del gradiente conjugado y los gradientes reducidos.

Para escoger un buen método de propósito general en programación no lineal , es necesario saber que tan bien puede este método resolver problemas de carácter general . Un aspecto de esto se realiza examinando las propiedades de convergencia del método . Sin embargo . cuando se analiza la convergencia global , hay situaciones típicas que pueden causar dificultad como son :

a) el último valor singular de la matriz Jacobiana del conjunto activo de restricciones tiende a cero.

b) los multiplicadores de Lagrange estimados tienden a infinito,

c) el problema no tiene puntos viables.

d) no se satisfacen las condiciones suficientes de primer y segundo orden.

Por lo anteriormente explicado , se debe escoger métodos con un alto desempeño en la solución de problemas prácticos.

Métodos para la programación no lineal.[4]

II. Métodos de funciones de penalidad y Lagrangianos aumentados.

De acuerdo con Sargent (1980)[5] estos métodos fueron las primeras aproximaciones para el problema de optimización con restricciones (i.e. , Frisch(1955) , Carroll (1961)) . Estas primeras ideas para la solución de problemas restringidos por medio de funciones de penalidad consideraban una secuencia de minimizaciones sin restricción en la cual un parámetro de penalidad se ajusta de una minimización a otra así que la secuencia de mínimos sin

[4] La clasificación de los métodos de programación no lineal en la literatura consultada varía de acuerdo al autor y al año de la obra , la clasificación que se usó en este capítulo es la que da Fletcher (1981) en [4] cada sección es un breve resumen de los métodos que dan las diferentes obras.
[5] Citado en [4]

restricciones converge a un punto viable del problema restringido que satisface las condiciones suficientes de optimización.

Típicamente , la reformulación de un problema por este método nos da la facilidad de movernos lejos de los límites de las restricciones , especialmente cuando no se encuentra en la vecindad un punto estacionario .

Su principal desventaja es la elección de los parámetros de penalidad . En otras palabras , los factores de peso que le dan importancia a la función objetivo y a sus restricciones.

Estas primeras ideas son actualmente obsoletas aunque han servido de base a algoritmos más eficientes .Es necesario remarcar que en varios aspectos los métodos de función de penalidad están cercanamente relacionados a las técnicas que usan Lagrangianos aumentados , programación cuadrática sucesiva, y métodos del gradiente. Las funciones de penalidad proveen una interpretación natural a estas técnicas . Con estos argumentos es legítimo establecer que algunos de los mejores algoritmos disponibles son , en efecto , técnicas de funciones de penalidad.

La idea intuitiva que gira alrededor de los métodos de función penalidad es simple. Supongamos que nosotros buscamos un mínimo de una función real f sobre su apropiado subconjunto X de R^n . Esto es , un problema de minimización con restricciones que puede ser transformado a un problema de optimización sin restricciones después de algunas modificaciones a la función objetivo. Sea

$$P(x) = \begin{cases} 0 & x \in X \\ +\infty & x \notin X \end{cases} \qquad \text{f.3.7.}$$

y considerando la minimización sin restricciones de la **función objetivo aumentada** F , dada por

$$\min \; F(\mathbf{x})_{x \in R^n} = f(\mathbf{x}) + P(\mathbf{x}) \qquad \text{f.3.8.}$$

donde *f* esta definida para R^n . Un punto x* minimiza a F si y solo si también minimiza a *f* sobre X. La función P se llama **función de penalidad** . En la practica, la optimización sin restricciones 3.8. no puede llevarse a cabo a causa de la discontinuidad de F en los límites de X y los valores infinitos fuera de X .

Funciones de penalidad exteriores.

Los métodos de función de penalidad exteriores usualmente resuelven (P) como una secuencia de problemas de minimización sin restricciones cuyas soluciones se aproximan a la solución de (P) desde afuera del conjunto viable.

Los métodos de función de penalidad exteriores consisten en resolver una secuencia de optimizaciones sin restricciones para k=0,1,2,..., dado por :

$$min \quad F(\mathbf{x}, p^k) = f(\mathbf{x}) + \frac{1}{p^k}\left\{ \sum_{i=1}^{m} \left| min\ [0, g_i(\mathbf{x})] \right|^{\alpha} + \sum_{j=1}^{p} \left| h_j(\mathbf{x}) \right|^{\beta} \right\}$$

f.3.9.

usando una secuencia decreciente de números positivos p^k . Definiendo x^{k*} como la solución óptima de 3.7. , se construye una secuencia de puntos $\{x^{k*}\}$, la cual bajo ciertas condiciones de (P) , tiene una convergencia al óptimo de (F).

Funciones de penalidad interiores.

Aquí los programas no lineales con restricciones de desigualdad se resuelven a través de una secuencia de problemas de optimización sin restricciones cuyos mínimos son puntos que satisfacen estrictamente las restricciones , esto es , en el interior del conjunto viable . Como el algoritmo requiere que el conjunto de puntos viables interiores no sea vacío, no se puede manejar restricciones de igualdad por este método.

El método de penalidad interior , se establece como sigue :

Para k=0,1,..., se define una función G que es la función objetivo aumentada a minimizar en una secuencia de problemas de optimización sin restricciones dado por :

$$min \quad G(\mathbf{x}, p^k) = f(\mathbf{x}) + t(p^k)q(x)$$

Las funciones más comunes para las funciones t y q son :

$$t_1(p) = p$$

$$t_2(p) = (p)^2$$

$$q_1(\mathbf{x}) = -\sum_{i=1}^{m} \log \quad g_i(\mathbf{x})$$

$$q_2(\mathbf{x}) = \sum_{i=1}^{m} \frac{1}{g_i(\mathbf{x})}$$

$$q_3(\mathbf{x}) = \sum_{i=1}^{m} \frac{1}{[g_i(\mathbf{x})]^2}$$

$$q_4(\mathbf{x}) = \sum_{i=1}^{m} \frac{1}{max\ [0, g_i(\mathbf{x})]}$$

f.3.10.

Funciones de penalidad exactas

En los métodos anteriores la solución se obtiene mediante la solución de una secuencia de problemas de optimización sin restricciones . Si existe una función real con la propiedad de que en una simple minimización sin restricciones se obtenga una solución óptima de (P) . Este tipo de función se llama una **función de penalidad exacta.**

Uno de los primeros trabajos con funciones de penalidad exactas se debe a Zangwill(1967) y Pietrzykowki(1969) . Esencialmente , estas funciones son más simples y están más relacionadas al problema original y su función Lagrangiana que las funciones de penalidad que no son exactas .

Una función de penalidad exacta para una función objetivo convexa y para una g cóncava , es:

$$P\ (\ x\ ,\ p\ )\ =\ f\ (\ x\ )\ -\ \frac{1}{p}\sum_{i=1}^{m}\ min\quad (\ 0\ ,\ g\ (\ x\ ))\qquad \text{f.3.11.}$$

Métodos del Lagrangiano Aumentado.

La forma básica del método del Lagrangiano aumentado requiere la solución de una secuencia de problemas de la forma :

$$minimizar \qquad \mathrm{L}_{\ cK}\ (\mathbf{x}, \lambda_k) = f(\mathbf{x}) + \lambda^T h(\mathbf{x}) + \frac{1}{2} c\left|h(\mathbf{x})\right|^2$$

f.3.12.

donde la función a minimizar se denomina Lagrangiano aumentado y la secuencia de parámetros de penalidad {c } satisfacen $0 < c_k \le c_{k+1}$ para toda k . El vector inicial de multiplicadores λ o se define al inicio para crear los vectores subsecuentes de multiplicadores λ k , k ≥ 1 se generan por alguna fórmula de actualización , tal como la iteración de primer orden :

$$\lambda_{\ k+1}\ =\ \lambda_{\ k}\ +\ c_{\ k}\, h\ (\ \mathbf{x}_{\ k}\ )\qquad \text{f.3.13.}$$

donde xk resuelve el problema 3.12 . Puede haber una iteración de 2o. orden:

$$\lambda_{k+1} = \begin{array}{l} \lambda_{k+1} = \bar{\lambda} + \Delta\lambda_k \\ donde \\ \bar{\lambda}_k = \lambda_k + c_k h(\mathbf{x}_k) \\ \text{la iteración de 1er. orden y donde} \\ \Delta\lambda_k \text{ junto con algun vector } \Delta x_k \\ \text{resuelve el sistema:} \end{array}$$

$$\begin{pmatrix} \mathbf{H}_k & \nabla h(\mathbf{x}_k) \\ \nabla h(\mathbf{x}_k) & 0 \end{pmatrix} \begin{pmatrix} \Delta \mathbf{x}_k \\ \Delta \lambda_k \end{pmatrix} = - \begin{pmatrix} \nabla_x L_o(\mathbf{x}_k, \overline{\lambda}) \\ h(\mathbf{x}_k) \end{pmatrix}$$

f.3.14.

Aquí Hk es la matriz Hessiana $\nabla^2_{xx} L_o(\mathbf{x}_k, \overline{\lambda_k})$ de la función Lagrangiana Lo evaluada en $(\mathbf{x}_k, \overline{\lambda_k})$, o alguna aproximación por cuasi - Newton . Hay que notar que el sistema 5.9 es también la base de los métodos PCR , lo que resalta las relaciones entre ambos métodos.

Existe un gran número de variaciones y extensiones de esta idea del método del Lagrangiano aumentado. Hay una rica teoría asociada con los métodos de Lagrangianos aumentados , combinando dualidad , problemas convexos y no convexos , y esto forma una base importante en el desarrollo de nuevos métodos.

Las ventajas típicas citadas en favor de la aproximación de Lagrangianos aumentados es que son métodos robustos , y relativamente fáciles de programar . Sus desventajas frente a otros métodos son principalmente en dos áreas. Primero, la viabilidad de las iteraciones generadas no se mantiene , así , si el algoritmo termina prematuramente , no dará una solución factible. La segunda desventaja se manifiesta en si misma principalmente en problemas pequeños y se basa en la comparación de la eficiencia respecto a las técnicas PCS .

Sumario.

Entre los primeros métodos que se desarrollaron para aplicarlos en códigos computacionales , se encuentran los métodos de transformación que se describieron en este capítulo . Las funciones de penalidad son para problemas que contienen igualdades o desigualdades . Formalmente , es prácticamente inevitable que se generen algunos puntos no viables , y esto es su principal desventaja. La motivación para desarrollar estos métodos fue reemplazar un problema que no se podía resolver como una secuencia de problemas por uno que si se pudiera. Sin embargo , algunos problemas son más difíciles de resolver a medida que se aproximan al mínimo restringido , lo cual es lo opuesto a lo que

uno idealmente podría pensar . Por estas razones , estos métodos ya no se ocupan para problemas de propósitos generales . Por otro lado , si la precisión no es muy importante y se tiene que programar el código , entonces las técnicas de funciones de penalidad pueden ser consideradas una opción . A pesar de sus desventajas , estos métodos se siguen utilizando para problemas particulares , aun en problemas muy grandes , pero las razones para ello según Scales (1985) son de carácter histórico . Después de todo si no se tiene el código de las nuevas técnicas puede ser poco razonable gastar recursos en resolver un problema ya resuelto por las técnicas de función de penalidad. Finalmente hay que resaltar que el estudió de estos métodos es importante para comprender los desarrollos actuales.

Los métodos del Lagrangiano aumentado son un desarrollo de las aproximaciones por función de penalidad en la cual las minimizaciones sin restricciones son capaces de localizar un mínimo restringido . Fundamentalmente, el término de penalidad se agrega a una función de Lagrange, en lugar de a la función objetivo . Porque la función de Lagrange tiene un punto estacionario en el mínimo restringido . Los estimados de los multiplicadores de la función de Lagrange pueden variarse más que el término de penalidad . Tan pronto como se encuentran los valores precisos de los multiplicadores la función tiene un mínimo en la solución que cumple con los criterios de optimización . En los métodos de Lagrangiano aumentado exactos, a los multiplicadores se les permite variar continuamente como parte de la minimización sin restricciones.

La desventaja de esta aproximación es que las primeras derivadas de las funciones del problema están incluidas en la función del Lagrangiano aumentado, y esto crea problemas para los algoritmos de minimización sin restricciones. En los métodos secuenciales , se desarrollan una serie de minimizaciones sin restricciones cada una fijando estimados de los multiplicadores que tienden a los multiplicadores exactos conforme las minimizaciones proceden y nos aproximamos al mínimo restringido . La razón de convergencia del mínimo sin restricciones al mínimo restringido está limitada por la razón de convergencia de los estimados de los multiplicadores.

Se necesita una mayor cantidad de tiempo computacional comparado con los métodos PCS . Pero para problemas grandes cada iteración de este método requiere menos cálculos globales . Así que para problemas de grandes dimensiones este tipo de métodos puede llegar a ser más eficiente computacionalmente hablando que los métodos PCS.

III. Métodos del Gradiente Reducido.

Ciertamente , la aproximación más directa del problema general de programación no lineal sería la linealización del problema y la aplicación de las técnicas de programación lineal.

1. Formular un modelo con un punto operativo nominal y linealizar todas las restricciones y la función objetivo cerca del punto de operación asumiendo un formato de programación lineal.

Usar programación lineal para resolver el problema linealizado.

2. Sucesivamente linealizar las restricciones y la función objetivo de un problema no lineal y sucesivamente proponer soluciones. Una vez que se obtiene la soluciónde1 problema lineal se prueba el punto óptimo .

3. Linealizar las funciones en una sección mediante segmentos de lineas rectas .

No existe garantía de convergencia para ninguno de estos métodos.

En la práctica el mejor algoritmo general que usa la linealización iterativa es el algoritmo del Gradiente Reducido Generalizado (GRG) . Se han propuesto e implementado muchas variantes del algoritmo GR . Wolfe (1967) fue el primero en proponer la idea para problemas con restricciones lineales . Las extensiones a esta clase de problemas han sido desarrolladas por Murtagh y Saunders (1978) y Shanno y Marsten (1979) . Ambos procedimientos resuelven problemas grandes y dispersos , basados en métodos de programación lineal . Beck (1981) ha especializado esta idea a problemas de conservación de flujo en redes . Para el caso de restricciones no lineales fue pionero Abadie y Carpenter , (1969) , quienes llamaron al procedimiento Gradiente Reducido Generalizado (GRG) . Se

desarrollaron después variantes de este método por Lasdon y Waren (1978) , Gabriele y Ragsdell (1977) , y Helne y Littschwager (1975) .[6] Estas versiones resuelven sistemas grandes y dispersos . En esencia el método emplea restricciones linealizadas , define nuevas variables que son normales a las restricciones , y expresa el gradiente en términos de la base normal. (Wolfe ha demostrado que el método del Gradiente Reducido esta relacionado con el método Simplex de la programación lineal). Aunque el problema que resuelve el método GRG es

Minimizar $f(\mathbf{x})$ $\mathbf{x} = [x_1, x_2, x_3, x_n]$ f.3.15.

Sujeta a $h(\mathbf{x}) = 0$ $j = 1, \ldots, m$

$$L_i \leq x_i \leq U_i \qquad i = 1, \ldots, n$$

donde L_i y U_i son los límites inferior y superior sobre x_i , respectivamente, los límites superior e inferior son tratados como vectores separados más que ser clasificados como restricciones de desigualdad porque son tratados en forma diferente para determinar la longitud de paso en la dirección de búsqueda.

Las restricciones de desigualdad no lineales pueden ser acomodadas mediante substracción o adición del cuadrado de variables auxiliares[7] de las restricciones :

$$h_j(\mathbf{x}) = g_j(\mathbf{x}) - \sigma^2_j = 0$$

y permitiendo que los límites de σ_j ser $-\infty \leq \sigma_j \leq \infty$.

Los métodos del gradiente reducido buscan mantener viabilidad sobre un conjunto de las restricciones no lineales mientras se reduce el valor de la función objetivo . Así , la dimensionalidad de la búsqueda es reducido al número total de variables menos el número de restricciones independientes . Una dificultad con este procedimiento es regresar al punto viable si la solución de un subproblema

[6] Lasdon , L.S. " Reduced Gradient Methods " . University of Texas , Austin , Texas .
[7] slack variable

con restricciones activas linealizadas lleva a un punto inviable con respecto a las restricciones originales.

Concepto del Gradiente Reducido.

Suponga que las *m* restricciones en el problema 3.15.son lineales o son aproximaciones linealizadas a $h_j(\mathbf{x}) = 0$. La imposición de estas restricciones reduce el número de grados de libertad asociados con $\mathbf{x}$ desde *n* hasta (*n-m*). En la búsqueda de la solución del problema , solo (*n-m*) variables serán las variables independientes ; *m* variables dependientes.

$$\mathbf{h} \equiv \begin{bmatrix} h_1(\mathbf{x}) \\ \cdot\cdot \\ h_2(\mathbf{x}) \end{bmatrix}$$ un vector de m X 1

$$\frac{\partial \mathbf{h}}{\partial \mathbf{x}_D} \equiv \begin{bmatrix} \frac{\partial h_1(\mathbf{x})}{\partial x_1} & \frac{\partial h_2(\mathbf{x})}{\partial x_m} \\ \cdots\cdots\cdots & \cdots\cdots\cdots \\ \frac{\partial h_m(\mathbf{x})}{\partial x_2} & \frac{\partial h_m(\mathbf{x})}{\partial x_m} \end{bmatrix}$$ una matriz de m X m

$$\left[\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_I}\right]^T \equiv \left[\nabla^T_{\mathbf{x}_I} f\right] = \left[\frac{\partial f(\mathbf{x})}{\partial x_{m+1}} \quad \frac{\partial f(\mathbf{x})}{\partial x_n}\right]$$

$$\left[\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_D}\right]^T = \left[\nabla^T_{\mathbf{x}_D} f\right] = \left[\frac{\partial f(\mathbf{x})}{\partial x_1} \cdots \frac{\partial f(\mathbf{x})}{\partial x_m}\right]$$

$$\frac{d\mathbf{x}_D}{d\mathbf{x}_I} \equiv \begin{bmatrix} \frac{dx_1}{dx_{m+1}} & \cdots & \frac{dx_1}{dx_n} \\ \cdots & \cdots & \cdots \\ \frac{dx_m}{dx_{m+1}} & \cdots & \frac{dx_m}{dx_n} \end{bmatrix}$$

$$\frac{d\mathbf{h}}{d\mathbf{x}_I} \equiv \begin{bmatrix} \frac{dh_1(\mathbf{x})}{dx_{m+1}} & \cdots & \frac{dh_1(\mathbf{x})}{dx_n} \\ \cdots & \cdots & \cdots \\ \frac{dh_m(\mathbf{x})}{dx_{m+1}} & \cdots & \frac{dh_m(\mathbf{x})}{dx_n} \end{bmatrix}$$

Los componentes del gradiente reducido son :

$$\mathbf{g}_R^T = \text{gradiente reducido} = \left[\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_I}\right]^T - \left[\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_D}\right]^T \left[\frac{\partial \mathbf{h}}{\partial \mathbf{x}_D}\right]^{-1} \left[\frac{\partial \mathbf{h}}{\partial \mathbf{x}_I}\right]$$

Algoritmo del Gradiente Reducido Generalizado.

Las fases esenciales del algoritmo del gradiente reducido generalizado se muestran a continuación . Cada código contiene variaciones particulares a las fases aquí mostradas :

Fase 1. Determinar los componentes de búsqueda para las variables independientes.

A cada paso k linealizar las restricciones en el punto viable **x**k y computar el gradiente reducido .

Los componentes de la dirección de búsqueda en el espacio de las variables independientes se establecen a partir de los elementos del gradiente reducido . como sigue :

a) Si x_i esta en uno de los límites , el componente de la dirección de búsqueda es $\Delta_i^k = 0$

Si $x_i^k = U_i \quad g_{Ri} < 0$

$x_i^k = L_i \quad g_{Ri} > 0$

b) Si Li < x_i < Ui , la dirección es la negativa del elemento correspondiente del gradiente reducido.

Fase 2. Determinar los componentes de búsqueda para las variables dependientes. Para mantener la viabilidad con respecto a las restricciones linealizadas , se calcula los componentes de la dirección debúsqueda en el espacio de variables dependientes como sigue :

$$\Delta_D^k = -\left[\frac{\partial \mathbf{h}^k}{\partial \mathbf{x}_D}\right]^{-1}\left[\frac{\partial \mathbf{h}^k}{\partial \mathbf{x}_I}\right]\Delta_I^k$$

Fase 3 . Mejorar el valor de la función objetivo.

Minimizar $f(\mathbf{x}_I^k + \lambda\Delta_I^k, \mathbf{x}_D^k + \lambda\Delta_D^k)$

con respecto a λ (λ>0) por medio de una búsqueda unidimensional.

La fase 3 comprende :

$$\mathbf{x}_I^{k+1} = \mathbf{x}_I^k + \lambda^k \Delta_I^k$$

$$\tilde{\mathbf{x}}_D^{k+1} = \mathbf{x}_D^k + \lambda^k \Delta_D^k$$

donde $\tilde{\mathbf{x}}_D$ es un punto tentativo

Fase 4. Usar método de Newton para tener viabilidad en las variables dependientes.

Ya que algunos de los componentes de $\tilde{\mathbf{x}}_D^{k+1}$ no será viable , $\tilde{\mathbf{x}}_D^{k+1}$ se modifica por medio de un método de Newton para regresar a un punto que pueda satisfacer las restricciones.

$$\mathbf{x}_D^{k+1} = \tilde{\mathbf{x}}_D^{k+1} - \left[\frac{\partial \mathbf{h}(\mathbf{x}_I^{k+1}, \tilde{\mathbf{x}}_D^{k+1})}{\partial \mathbf{x}_D^k}\right]^{-1} \mathbf{h}(\mathbf{x}_I^{k+1}, \tilde{\mathbf{x}}_D^{k+1})$$

Fase 5. Procedimiento de convergencia del método de Newton.

a) Si $\mathbf{x}^{k+1}$ es un punto viable y $f(\mathbf{x}_I^{k+1}, \mathbf{x}_D^{k+1}) < f(\mathbf{x}_I^k, \mathbf{x}_D^k)$, adoptar $\mathbf{x}^{k+1}$, y comenzar la fase 1 de nuevo.

b) Si $\mathbf{x}^{k+1}$ es un punto viable pero si $f(\mathbf{x}_I^{k+1}, \mathbf{x}_D^{k+1}) > f(\mathbf{x}_I^k, \mathbf{x}_D^k)$, reducir λ en un factor y comenzar fase 4.

c) Si el método de Newton falla después de determinado número de iteraciones , se reduce λ en un factor y se comienza la fase 4.

d) Si ninguno de los pasos anteriores ocurre después de dos o tres intentos , cambiar la base intercambiando una variable dependiente con una variable independiente.

El criterio de finalización del algoritmo puede ser cuando $|\Delta_i^k| < \varepsilon_i$

Ventajas y desventajas de los algoritmos de Gradiente Reducido.

Ventajas.

a) Por medio de la eliminación de algunas variables , los algoritmos GRG operan en un espacio de dimensión reducida . Frecuentemente , se localiza el óptimo rapidamente.

b) Las extensiones a problemas grandes y dispersos es conceptualmente simple.

c) Las excursiones fuera de la región viable pequeñas comparadas con otros algoritmos . Esto es importante en la resolución de problemas donde algunas funciones no están definidas para ciertas valores de sus argumentos.

d) Los conceptos que se manejan en el método son simples y directos , lo que permite entender con facilidad el método.

Desventajas.

Para restricciones no lineales , la mayoría del tiempo computacional se ocupa en buscar satisfacer estas restricciones en cada paso . Por ello el algoritmo se vuelve lento . Para restricciones lineales , esta desventaja desaparece .

IV. Métodos de programación cuadrática sucesiva.

Programación Cuadrática.

La programación cuadrática (PC) es el nombre dado al procedimiento que minimiza una función cuadrática de *n* variables sujetas a *m* restricciones lineales de igualdad , desigualdad , o ambos tipos. Un problema de programación cuadrática es la forma más simple de un problema de programación no lineal con restricciones de desigualdad. La PC puede ser tratada como un subproblema para resolver problemas más generales de programación no lineal. Un gran número de problemas prácticos de optimización son propuestos como problemas de PC , tal como mínimos cuadrados con restricciones , control óptimo de sistemas lineales con funciones cuadráticas de costos , y la solución de ecuaciones diferenciales y algebraicas . Las técnicas propuestas para la solución de un problema de programación cuadrática tiene muchas similaridades con aquellas usadas en la solución de problemas de programación lineal. Especificamente , cada restricción de desigualdad , debe poder ser satisfecha como una igualdad o no es incluida en la solución del problema de PC, así que una vez que las restricciones de desigualdad son identificadas, la técnica de PC puede reducirse a un procedimiento de búsqueda de vértice , examinando la intersección de las ecuaciones lineales como en PL . Esta idea es el vínculo fundamental entre ambos métodos.

En notación compacta , el problema de programación cuadrática es

Minimizar $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x}$

Sujeto a $\mathbf{A}\mathbf{x} \geq \mathbf{b}$

$\mathbf{x} \geq \mathbf{0}$

donde **c** es un vector de coeficientes constantes , **A** es una matriz de ($m \times n$) y se asume generalmente que **Q** es una matriz simétrica.

Debido a que las restricciones son lineales y presumiblemente independientes , las condiciones necesarias para encontrar un mínimo local son satisfechas , y por lo tanto las condiciones Kuhn-Tucker son condiciones necesarias para obtener la solución óptima de un problema de PC. Además si **Q** es positiva , las condiciones Kuhn- Tucker son también condiciones suficientes para un extremo , y una solución que cumpla estas condiciones será el óptimo global. Si **Q** no es positivo , el problema puede tener soluciones ilimitadas y un mínimo local.

Comenzando con la función de Lagrange

$$L = \mathbf{x}^T\mathbf{c} + \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} - \mathbf{v}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) - \mathbf{u}^T\mathbf{x}$$

e igualando el gradiente de L (con respecto a $\mathbf{x}^T$) a cero (notar que $\mathbf{v}^T(\mathbf{A}\mathbf{x}-\mathbf{b}) =$

$(\mathbf{A}\mathbf{x} - \mathbf{b})^T\mathbf{v} = (\mathbf{x}^T\mathbf{A}^T - \mathbf{b}^T)\mathbf{v}$ y $\mathbf{u}^T\mathbf{x} = \mathbf{x}^T\mathbf{u}$)

$$\nabla L = \mathbf{c} + \mathbf{Q}\mathbf{x} - \mathbf{A}^T\mathbf{v} - \mathbf{u} = \mathbf{0}$$

Las variables auxiliares (slack) no negativas **y** pueden ser insertadas dentro de las restricciones $\mathbf{A}\mathbf{x} - \mathbf{b} \geq \mathbf{0}$ así que $\mathbf{A}\mathbf{x} - \mathbf{b} - \mathbf{y} = \mathbf{0}$. Entonces las condiciones de Kuhn - Tucker se reduce al siguiente conjunto de ecuaciones lineales :

$$\mathbf{c} + \mathbf{Q}\mathbf{x} + \mathbf{A}^t\mathbf{v} - \mathbf{u} = \mathbf{0}$$

$$Ax - b - = 0$$

$$x \geq 0 \quad u \geq 0 \quad v \geq 0 \quad y \geq 0$$

$$u^T x = 0 \qquad v^T x = 0$$

donde u_j y v_j son los multiplicadores de Lagrange y los y_j son las variables. El conjunto de variables (x^* ,u^* , v^*, y^*) son las soluciones óptimas.

Se puede demostrar que si (**x** ,**u** , **v**, **y**) es la solución del problema QP , entonces **f (x)** es equivalente :

$$f = \tfrac{1}{2} (c^T x + b^T y)$$

La minimización de la función anterior sujeta a sus restricciones , forma lo que es llamado el *problema lineal complementario* , la solución de esta es la solución del problema original.

Un problema en el cual las restricciones lineales **Ax > b** son remplazados con **Ax = b** es llamado también problema de programación cuadrática.

Pang (1983) revisó varios métodos que han sido usados para resolver problemas de programación cuadrática., y ha identificado cuatro familias de métodos :

1. Relacionados al LP Simplex

2. Estrategia del conjunto activo.

3. Conjunto de representación interna.

Programación Cuadrática Sucesiva.

El método más reciente es el de programación cuadrática sucesiva . Wilson (1963) describe un programa llamado SOLVER que aproximaba la función objetivo localmente por una función cuadrática , y las restricciones por funciones lineales , así que esa programación cuadrática podía ser recursiva . Al comienzo de una iteración **x** , un estimado de **x*** , es conocido . Una dirección de búsqueda **s** , se usa para tener un mejor estimado de **x*** , se obtiene de resolver un problema cuya forma general es :

Minimizar {una función cuadrática en términos de **s**

Sujeta a {restricciones linealizadas de igualdad y desigualdad } como funciones de **s**

Forma del Subproblema de programación cuadrática.

Una solución óptima al problema PCS debe satisfacer las condiciones de Kuhn- Tucker . Si se aplica el método de Newton (para resolución de ecuaciones) a las condiciones necesarias Kuhn -Tucker para un problema de PNL conteniendo solo restricciones de igualdad , la función de Lagrange es

$$L (\mathbf{x}, \omega) = f (\mathbf{x}) + \sum_j \omega_j \, h_j (\mathbf{x})$$

y las condiciones necesarias de Kuhn Tucker son

$$\nabla L = \nabla f (\mathbf{x}) + \sum_j \omega_j \nabla h_j (\mathbf{x}) = 0$$

$$h_j (\mathbf{x}) = 0$$

El método de Newton aplicado a las dos ecuaciones anteriores lleva a :

$$\begin{bmatrix} \nabla^2 L & -\mathbf{J} \\ -\mathbf{J}^T & 0 \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \omega \end{bmatrix} = \begin{bmatrix} -\nabla L \\ \mathbf{h} \end{bmatrix}$$

donde **J** es la matriz Jacobiana para las restricciones de igualdad . Este sistema de ecuaciones lineales se resuelve para $\Delta \mathbf{x}$ y $\Delta \omega$. Puede demostrarse que si $\Delta \mathbf{x}$ y $\Delta \omega$ satisfacen las dos ecuaciones lineales , entonces satisfacen también las condiciones necesarias para el problema de programación cuadrática para determinar **s** (Powell , 1978) .

Minimizar $F(\mathbf{s}) = \mathbf{s}^T \nabla f(\mathbf{x}) + \frac{1}{2}\mathbf{s}^T \mathbf{B}\mathbf{s}$

Sujeta a

$$g_j(\mathbf{x}) + \mathbf{s}^T \nabla g_j(\mathbf{x}) \geq 0$$
$$j = m+1, \ldots, p$$
$$h_j(\mathbf{x}) + \mathbf{s}^T \nabla h_j(\mathbf{x}) = 0$$
$$j = 1, \ldots, m$$

donde **s** es la dirección de búsqueda y **B** es una aproximación positiva y definida del Hessiano. Las restricciones son linealizadas y la función objetivo es cuadrática. Si **B** se utiliza en lugar de $\nabla^2 f(\mathbf{x})$, la convergencia es superlineal.

Las restricciones de desigualdad pueden ser tratadas si se incluyen en la función de Lagrange. Ya que la matriz Hessiana de la función de Lagrange puede no ser positiva y definida , el problema puede no tener solución , por ello el uso de una matriz **B** que se actualiza de acuerdo a la fórmula BFGS , una fórmula que solo requiere las primeras derivadas.

Desarrollo del Método.

Estos métodos fueron desarrollados mediante la aplicación del método de Newton sobre el Lagrangiano :

$$L\ (\ \mathbf{x},\mathbf{u},\mathbf{v}\ )\ =\ f\ (\ \mathbf{x}\ )\ +\ \sum_{j} v_j\ \mathrm{h}_j(\ \mathbf{x}\ )+\sum_{j}\mathbf{u}_j\ \mathrm{g}_j(\ \mathbf{x}\ )$$

tal y como uno lo haría sobre f(x) para una minimización sin restricciones. Encontrando un punto estacionario para L que satisface las condiciones de Kuhn Tucker si u > 0 .

Si se expande la función del Lagrangiano con respecto a x , u , y V a los primeros tres términos de la serie de Taylor , tenemos :

$$L(\mathbf{x},\mathbf{u},\mathbf{v}) = L(\mathbf{x}^i,\mathbf{u}^i,\mathbf{v}^i)+\nabla L^{T\mathrm{i}}\begin{bmatrix}(\mathbf{x}-\mathbf{x}^i)\\(\mathbf{u}-\mathbf{u}^i)\\(\mathbf{v}-\mathbf{v}^i)\end{bmatrix}+\frac{1}{2}\begin{bmatrix}(\mathbf{x}-\mathbf{x}^i)\\(\mathbf{u}-\mathbf{u}^i)\\(\mathbf{v}-\mathbf{v}^i)\end{bmatrix}^T\nabla^2 L^i$$

$$L(\mathbf{x},\mathbf{u},\mathbf{v})=L(\mathbf{x}^i,\mathbf{u}^i,\mathbf{v}^i)+\nabla L^{T\mathrm{i}}\begin{bmatrix}(\mathbf{x}-\mathbf{x}^i)\\(\mathbf{u}-\mathbf{u}^i)\\(\mathbf{v}-\mathbf{v}^i)\end{bmatrix}+\frac{1}{2}\begin{bmatrix}(\mathbf{x}-\mathbf{x}^i)\\(\mathbf{u}-\mathbf{u}^i)\\(\mathbf{v}-\mathbf{v}^i)\end{bmatrix}^T\nabla^2 L^i\begin{bmatrix}(\mathbf{x}-\mathbf{x}^i)\\(\mathbf{u}-\mathbf{u}^i)\\(\mathbf{v}-\mathbf{v}^i)\end{bmatrix}$$

Como L es lineal en u y v , expandiendo y cancelando términos en la ecuación anterior lleva sólo a una serie en (x-xi) :

$$L(\mathbf{x},\mathbf{u},\mathbf{v})=L(\mathbf{x}^i,\mathbf{u},\mathbf{v})+\nabla_x L(\mathbf{x}^i,\mathbf{u},\mathbf{v})^T(\mathbf{x}-\mathbf{x}^i)+\frac{1}{2}(\mathbf{x}-\mathbf{x}^i)^T\nabla L(\mathbf{x}^i,\mathbf{u}^i,\mathbf{v}^i)(\mathbf{x}-\mathbf{x}^i)$$

un punto estacionario a esta aproximación satisface :

$$\nabla L(\mathbf{x},\mathbf{u},\mathbf{v}) = 0 = \nabla_x L(\mathbf{x}^t,\mathbf{u},\mathbf{v}) + \nabla_{xx} L(\mathbf{x}^t,\mathbf{u}^t,\mathbf{v}^t)(\mathbf{x}-\mathbf{x}^t)$$
$$0 = g(\mathbf{x}^t) + \nabla g(\mathbf{x}^t)^T(\mathbf{x}-\mathbf{x}^t) \text{ , si u } > 0$$
$$0 = h(\mathbf{x}^t) + \nabla h(\mathbf{x}^t)^T(\mathbf{x}-\mathbf{x}^t)$$

Esta es parte de las condiciones Kuhn Tucker para el siguiente programa cuadrático en donde d= x-xi:

$$\begin{aligned} \textit{Min} \quad & \nabla f^T d + \frac{1}{2} d^T \nabla_{xx} L(\mathbf{x}^t,\mathbf{u}^t,\mathbf{v}^t) d \\ \textit{sujeto a} \quad & g(\mathbf{x}^t) + \nabla \mathbf{g}(\mathbf{x}^t)^T d \le 0 \\ & h(\mathbf{x}^t) + \nabla h(\mathbf{x}^t)^T d = 0 \end{aligned}$$

La solución del programa cuadrático forza a las restricciones linealizadas activas a satisfacer la condición de Kuhn Tucker. Mas aún , mientras d se aproxima a cero , las condiciones KT del programa cuadrático se aproximan a las condiciones del programa no lineal original.

Este algoritmo , el primero del tipo de programación cuadrática sucesiva , fue propuesto por Wilson (1963) .

García-Palomares y Mangasarian (1976) probó las propiedades de convergencia local de este algoritmo y mostró que el Hessiano , no necesita ser calculado sino puede ser aproximado por una actualización tipo cuasi- Newton. Han (1977) mostró que la convergencia global a un punto KT podría ser obtenida si se usa una búsqueda lineal junto con una función de penalidad en el subproblema cuadrático.

Powell (1978) propuso una búsqueda lineal sobre la función del Lagrangiano. Biggs (1975) también desarrolló un método de programación cuadrática sucesiva.

Powell (1977) refinó las ideas de Han (1977) y propuso un algoritmo usando actualizaciones del Hessiano mediante la fórmula BFGS y una búsqueda

lineal sobre la función de penalidad. Sus resultados en muchos problemas de prueba muestra que este algoritmo es dos veces más rápido que otros algoritmos. Powell (1978) también probó algunas propiedades teóricas del método.

Mejoras al algoritmo de Programación Cuadrática Sucesiva.

Un método propuesto por Chamberlin et.al. (1979) es la técnica del sabueso (watchdog) . Aquí las funciones del Lagrangiano o la función de penalidad exacta son satisfechas en la búsqueda lineal .

Biegler (1981) propone un algoritmo de programación cuadrática sucesiva . Biggs (1989) mejora su algoritmo anterior , ambos utilizando como base el algoritmo de Wilson-Han-Powell :

El algoritmo WHP es el siguiente :

1. Inicializar todas las constantes .

$\mathbf{B}^1 = \mathbf{I}$

$\mu^0 = \upsilon^0 = 0$

$\delta = \frac{1}{2}$

$\mathbf{x}^1 = \mathbf{x}^1$

2. Evaluar

$$f(\mathbf{x}^i), g(\mathbf{x}^i), h(\mathbf{x}^i), \nabla f(\mathbf{x}^i), \nabla g(\mathbf{x}^i), \nabla h(\mathbf{x}^i)$$

3. Si i > 1 , se actualiza Bi usando la ecuación de BFGS.

$$\mathbf{B}^i = \mathbf{B}^{i-1} - \frac{\mathbf{B}^{i-1}\sigma\sigma^T\mathbf{B}^{i-1}}{\sigma^T\mathbf{B}^{i-1}\sigma} + \frac{\eta\eta^T}{\sigma^T\eta}$$

donde $\sigma = \mathbf{x}^i - \mathbf{x}^{i-1}$

$$\eta = \Theta\,\gamma + (1 - \Theta)\,b^{i-1}$$

$$\Theta = \begin{cases} 1 & \sigma^T \gamma > 0.2\,\sigma^T \mathbf{B}^{i-1}\sigma \\ \dfrac{0.8\,\sigma^T \mathbf{B}^{i-1}\sigma}{\sigma^T \mathbf{B}^{i-1}\sigma - \sigma^T \gamma} & \sigma^T \gamma < 0.2\,\sigma^T \mathbf{B}^{i-1}\sigma \end{cases}$$

$$\gamma = \nabla_x L(\mathbf{x}^i, \mathbf{u}^{i-1}, \mathbf{v}^{i-1}) - \nabla_x L(\mathbf{x}^{i-1}, \mathbf{u}^{i-1}, \mathbf{v}^{i-1})$$

Esto asegura que Bi es positiva y definida.

4. Resolver el siguiente programa cuadrático.

$$\nabla f(\mathbf{x}^i)^T d^i + \frac{1}{2} d^{iT} \mathbf{B}^i d^i$$

sujeto a :

$$g(\mathbf{x}^i) + \nabla g(\mathbf{x}^i)^T d^i \leq 0$$

$$h(\mathbf{x}^i) + \nabla h(\mathbf{x}^i)^T d^i = 0$$

para obtener di , ui, vi.

5. Si

$$\left|\nabla f(\mathbf{x}^i)^T d^i\right| + \left|\mathbf{u}^i h(\mathbf{x}^i)\right| + \left|\mathbf{v}^i h(\mathbf{x}^i)\right| \leq \varepsilon$$

para alguna preespecificada ε > 0 , detener . xi es un punto KT dentro de la tolerancia de ε.

6. sino , calcular el Lagrangiano ,

$$L(\mathbf{x}^i, \mathbf{u}^i, \mathbf{v}^i) = f(\mathbf{x}^i) + \mathbf{u}^i\, \mathbf{g}(\mathbf{x}^i) + v^i\, \mathbf{h}(\mathbf{x}^i)$$

la función de penalidad exacta ,

$$P(\mathbf{x}^i, \mu^i, \upsilon^i) = f(\mathbf{x}^i) + \mu^i\, \mathbf{g}(\mathbf{x}^i) + v^i \left|\, \mathbf{h}(\mathbf{x}^i)\right|$$

y sus derivadas

$$\nabla_x L\ (\ \mathbf{x}^i, \mathbf{u}^i, \mathbf{v}^i\ )\ =\ \nabla f\ (\ \mathbf{x}^i\ )\ +\mathrm{u}^i\,\nabla \mathrm{g}\ (\ \mathbf{x}^i\ )+\ v^i\,\nabla\ \mathrm{h}\ (\ \mathbf{x}^i\ )$$

$$P'(\ \mathbf{x}^i, \mu^i, \upsilon^i\ )\ =\ \nabla f\ (\ \mathbf{x}^i\ )d^i\ -\mu^i\ \mathrm{g}\ (\ \mathbf{x}^i\ )-\ v^i\ |\ \mathrm{h}\ (\ \mathbf{x}^i\ )\ |$$

Aquí

$\mu^{i}\ =\ max\quad (2\,|u^{i}_{j}|, \mu^{i-1}_{j})$

$v^{ij}_{j}\ =\ max\quad (2\,|v^{i}_{j}|, v^{i-1}_{j})$

$|h\ (\mathbf{x}^{i})|_{j}\ =\ |h_{j}\ (\mathbf{x}^{i})|$

P' es la derivada direccional de la función exacta de penalidad .

7. Actualiza valor de λ

8. Calcular

$$f(\mathbf{x}^i + \lambda^i d^i), g(\mathbf{x}^i + \lambda^i d^i), h(\mathbf{x}^i + \lambda^i d^i)$$

9.

$$P(\mathbf{x}^i + \lambda^i d^i) = f(\mathbf{x}^i) + \mu^i g(\mathbf{x}^i + \lambda^i d^i) + v^i |\ h(\mathbf{x}^i + \lambda^i d^i)\ |$$

$$L^*(\mathbf{x}^i + \lambda^i d^i) = f(\mathbf{x}^i + \lambda^i d^i) + u^i g(\mathbf{x}^i + \lambda^i d^i) + v^i h(\mathbf{x}^i + \lambda^i d^i)$$

si

$L^{*}\ (\mathbf{x}^{i} + \lambda^{i} d^{i}) < L\ (\mathbf{x}^{i} + \lambda^{i} d^{i})$, ir a 12.

10. Si

$$P(\mathbf{x}^l + \lambda^l d)^l \leq P(\mathbf{x}^l, \mu^l, \nu^l) + \delta\, \lambda^l P'(\mathbf{x}^l, \mu^l, \nu^l)$$

entonces ir a 12.

11.

$$\lambda^l = \lambda^l \left[max \;\; 0.1, \frac{0.5\, \lambda^l P(x^l, \mu^l, \nu^l)}{(\lambda^l P'(x^l, \mu^l, \nu^l) + P(x^l, \mu^l, \nu^l) - P(x^l + \lambda^l d^l)} \right]$$

ir a 8.

12.

$$x^{l+1} = x^l + \lambda^l d^l$$

$$i = i + 1$$

ir a 2.

En cada caso la matriz B es actualizada a la matriz identidad y el programa cuadrático es resuelto.

Esto ocurre cuando.

a) no hay solución para el programa cuadrático

b) λi es reducida más de 5 veces (la búsqueda lineal falla)

c) P' (xi,μi) > 0 , la dirección de búsqueda es ascendente.

Bibliografía consultada.

1. Luenberger , David . Introduction to linear and nonlinear programming. Reading, Mass : Adison Wesley,1973

2. Himmelblau, David . Applied nonlinear programming. New York: Mac Graw Hill , 1972.

3. Bazaraa , Mokhtar . Nonlinear programming :theory and algorithms. New York : Academic Press.1979

4. Powell , M.J.D. Nonlinear Optimization 1981 , London Academic , 1982.

5. Avriel , M. Nonlinear Programming , Prentince Hall :New Jersey ,1976.

6. Biegler L.T. Optimization Methods for Sequential Modular Simulators . Ph. D. University of Wisconsin , 1981.

7. Gill & Murray . Numerical methods for constrained Optimization. 1982

8. Bertsekas, Dimitri . Augmented Lagrangian and Differentiable Exact Penalty Methods en Powell Nonlinear Optimization 1981.

9. Scales , L.E. Introduction to Non- linear Optimization. Macmillan ,1985

10. Fletcher , R. Methods for Nonlinear Constraints en Powell , M.J.D. Nonlinear Optimization 1981 , London Academic , 1982.

11. Bartholomew - Biggs , M.C. Recursive Quadratic Programming Methods for Nonlinear Constraints . en Powell , M.J.D. Nonlinear Optimization 1981 , London Academic , 1982.

12. Fukushima , Masao . A succesive quadratic programming algorithm with global and superlinear convergence properties , en Mathematical Programming 35 (1986) 253- 264 : North Holland.

13. Himmelblau , D. M. Optimization for Chemical Processes. MacGraw Hill , 1988.

14. Lasdon , L. Modeling and Optimization with GINO , The Scientific Press: California.

15. Cuthbert , Thomas . Optimization using personal computers ; John Wiley & Sons, 1987.

Capitulo 4

Aplicación a la Ingeniería Química de la programación no lineal y el procesamiento en paralelo.

Problema 1. Planta de Williams - Otto[1]

Este problema se refiere a la planta de Williams- Otto .

La figura 4.1 muestra un diagrama simplificado del proceso . La planta consiste en un reactor de tanque agitado , un decantador y una columna de destilación en serie . Hay una recirculación del reboiler de la columna al reactor.

---

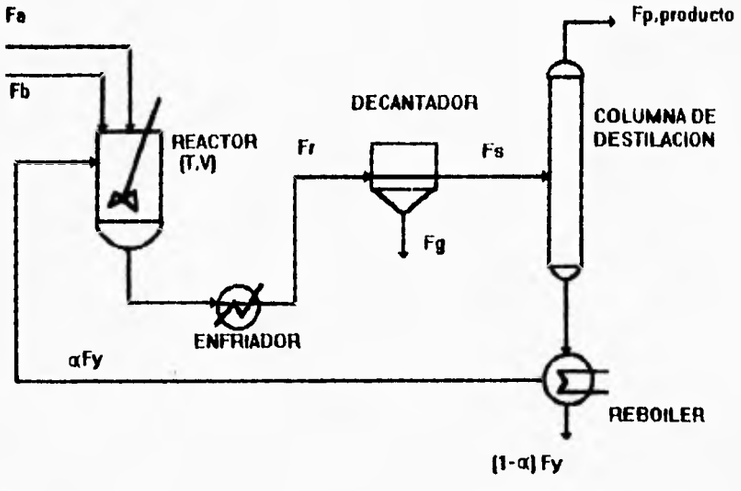
[1] Veasc :
Mirosh &Jung "Large Scale Process Optimization Techniques applied to Chemical and Petrolcum Processes." The Canadian Journal Chemical Engineering Vol 49, December 1971.
Biegler & Vasantharajan "Large Scale Decomposition for Succesive Quadratic Programming " Comput. chem Engng, Vol 12. , No 11 , pp. 1087-1101,1988.

El problema tiene 10 variables y 7 restricciones , dejando 3 grados de libertad . Encontrar el máximo de la tasa interna de retorno anual sobre la inversión que esta expresada por f.8.1

Maximizar :

$$Q = \frac{100}{600V\rho}[8400(0.3F_P + 0.0068F_D - 0.02F_A - 0.03F_B - 0.01F_G) -$$
$$- 2.22F_R - (0.124)(8400)(0.3F_P - 0.0068F_D) - 60V\rho\ ]$$

Fig. 8.1 Planta de Williams-Otto



Restricciones:

$g_1 = F_A + F_{TA} - F_{RA} - P_1 = 0$
$g_2 = F_B + F_{TB} - F_{RB} - P_1 - P_2 = 0$
$g_3 = F_{TC} + 2P_1 - F_{RC} - 2P_2 - P_3 = 0$
$g_4 = F_{TE} + 2P_2 - F_{RE} = 0$
$g_5 = F_{TG} + P_2 - F_{RG} - 0.5P_3 = 0$
$g_6 =$ **Balance de materia global**
$g_6 = F_A + F_B - F_D - F_G = 0$
$g_7 =$ **Requisitos de producción**
$g_7 = F_P - 4763 = 0$
$0 \le \alpha \le 1$
$500 \le T \le 1000$

Modelos matemáticos por unidad.

Decantador

$F_{Si} = F_{Ri}$
$F_{SC} = 0$
$F_{Cb} = F_{RC}$
$i = A, B, C, E, P$

Columna de Destilación

$F_{Ri} = F_{Si}$
$F_P = F_{SP} - 0.1F_{SE}$
$F_{YP} = F_{SP} - F_P$
$i = A, B, C, E, P$

Reboiler

$F_{Ti} = \alpha F_i$

$F_{Di} = (1-\alpha) F_i$

$i = A, B, C, E, P$

Reactor

$V = 0.0002964 F_R$

$K_i = a_i \exp(-b_i / T) / F_R^2$

	i	a_i	b_i
$P_1 = K_1 F_{RA} F_{RB}$	1	5.9755E9	12000
$P_2 = K_2 F_{RB} F_{RC}$	2	2.5962E12	15000
$P_3 = K_3 F_{RC} F_{RG}$	3	9.6283E15	20000

Nomenclatura del problema :

F_A, F_B	Alimentación de los componentes A,B (lb/h)
F_R	Salida total del reactor
F_{Ri}	Flujo de salida del reactor del componente i
F_{Si}	Flujo de salida del decantador del componente i
F_G	Flujo de salida del fondo del componente G
F_P	Flujo de salida del producto de la columna

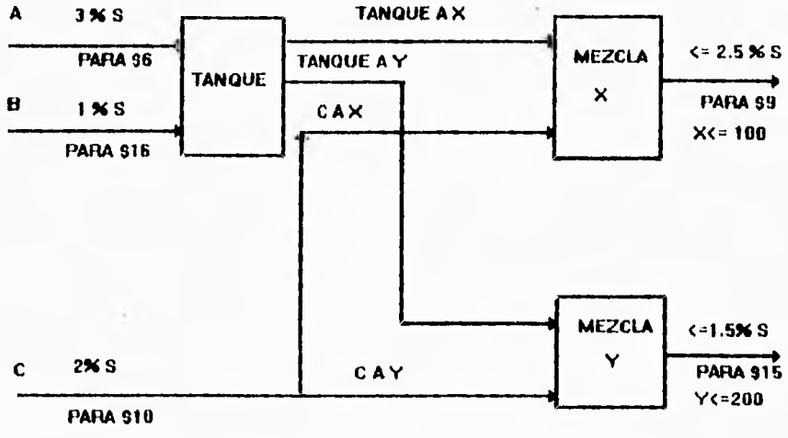
F_{Bi}	Flujo de salida del fondo de la columna del componente i
F_B	Flujo total del fondo de la columna
F_{Bi}	Flujo del fondo de columna retirado del componente i
F_T	Flujo del fondo de la columna recirculado
F_{Ti}	Flujo del fondo de la columna recirculado del componente i
α	Fracción de fondos recirculados al reactor
V	Volumen del reactor (ft3)
ρ	Densidad de la mezcla de reacción (se asume constante 50 lb/ft3)

Problema 2. Mezclado de Crudos.[2]

El mezclado de productos químicos líquidos (sin reacción química) que tengan diferentes concentraciones de impureza puede llevarnos a un modelo no lineal . Este problema es típico en la industria petrolera.

Considere el modelo simple propuesto por Haverly (1978) como se muestra en la fig.4.2. A,B,C son químicos de entrada que contienen azufre como impureza. Estos compuestos tiene que mezclarse para proveer dos productos de salida , X y Y , los cuales deben tener ciertos contenidos de azufre como se muestra en la figura. Los clientes comprarán todo el producto X y Y , hasta un máximo de 100 unidades de X y 200 unidades de Y , a los precios establecidos . El problema es operar el proceso para maximizar las ganancias.

PROBLEMA 2. MEZCLA DE PRODUCTOS.



[2] Haverly , *Studies of the Behavior of Recursion for the Pooling Problem* ,Association for Computing Machinery , 1978

Modelo :

Las ecuaciones de restricción definidas en este sistema incluye balances de materia y restricciones de azufre para los productos de salida.

El balance de materia para el tanque (asumiendo que no hay acumulación) es simplemente:

Cantidad A + Cantidad B = Cantidad del tanque a X + Cantidad del tanque a Y

Para los productos de salida , las ecuaciones de balance son :

Tanque a X + C a X = Cantidad de X

Tanque a Y + C a Y = Cantidad de Y

y para C :

C a X + C a Y = Cantidad C

Introduciendo el porcentaje de azufre en el Tanque , Tanque S , como una nueva variable , hace más fácil escribir las restricciones de azufre para X y Y . Si dejamos que Tanque S , tome valores de 0 a 100 , y expresar todos los otros porcentajes en la misma escala , las restricciones son :

Tanque S * Cantidad deTanque a X + 2 * C a X <= 2.5 * Cantidad de X

Tanque S * Cantidad de Tanque a Y + 2*C a Y <= 1.5* Cantidad de Y

El lado izquierdo de cada desigualdad representa el contenido de azufre del producto y el lado derecho es el valor maximo permitido de azufre en el producto. La ecuación de balance de azufre en el Tanque es :

3 * A + 1 * B = Tanque S * (A + B)

Esto define a Tanque S como la cantidad de azufre en el tanque dividido por la cantidad total de materia en el tanque.

Como se mencionó anteriormente , la demanda de producto es :

X <= 100

Y <= 200

y las consideraciones físicas restringen las variables a que sean positivas.

Claramente el azufre en el tanque estará entre 1% y 3%.

1<= Tanque S <= 3

Finalmente la función ganancia puede formularse de la siguiente forma :

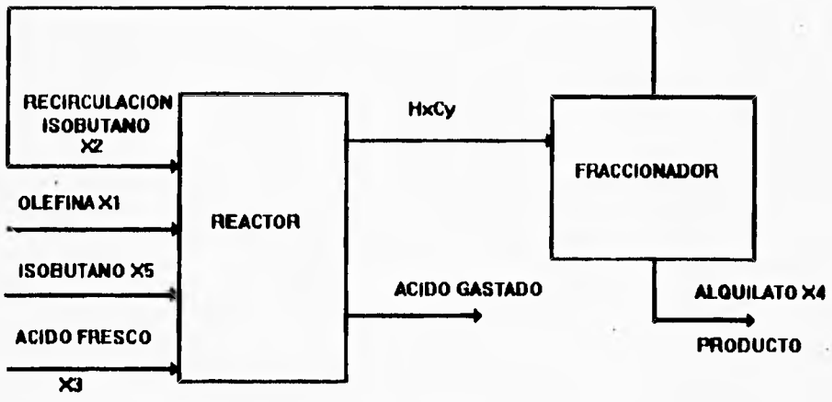
Costo X * X + Costo Y * Y - Costo A * A - Costo B * B - Costo C * C

Problema 3. Optimización de un proceso de alquilación.[3]

La figura 4.3 muestra un proceso de alquilación simplificado similar en esencia a los que se encuentran en la industria petrolera . El modelo contiene un reactor y un fraccionador . Varias olefinas e isobutano se alimentan al reactor , junto con ácido fresco . El acido se usa como catalizador en la reacción química . La salida del reactor se alimenta al fraccionador .

La columna fraccionadora separa el producto alquilado , el cual sale por fondos del fraccionador , el isobutano que no reacciona sale por domos y es recirculado al reactor . El total de isobutano que entra al reactor consiste de isobutano recirculado y de isobutano nuevo que se agrega para mantener la proporción adecuada con la olefina en la entrada del reactor.

PROBLEMA 3 . PROCESO DE ALQUILACION.



[3] Liebman , et al , *Modeling and Optimization with GINO,The Scientific Press , 1980*

Las siguientes son simplificaciones hechas para la construcción del modelo:

1. La alimentación de la olefina es butileno puro.
2. El isobutano nuevo y su recirculación son isobutano puro.
3. El ácido fresco tiene 98 % en peso.
4. El ácido gastado se descarga del reactor y no se recircula.

Las variables del problema son:

X1 = Alimentación de Olefina.

X2 = Reciclo de Isobutano.

X3 = Adición de Acido

X4 = Alquilato (Producto)

X5 = Entrada de Isobutano

X6 = Pureza del Acido (porcentaje en peso)

X7 = Número de Octano del alquilato.

X8 = Razon de Isobutano externo/ olefina

X9 = Factor de dilución del ácido.

X10 = Eficiencia del fraccionador para obtención de alquilato.

Hay 10 variables de interés ; tres de las cuales (X1,X2,X3) son consideradas variables independientes. Estas tres variables pueden ser ajustadas por el operador y sus cambios afectan a las otras variables . El objetivo es determinar los valores de las variables independientes para maximizar las ganancias.

Para completar el modelo de este sistema se debe de determinar las relaciones entre las variables dependientes e independientes .

El alquilato , X4 , es una función de la olefina , X1 , y la razon Isobutano / olefina , X8 . Por un analisis de regresión no lineal en el reactor con temperaturas entre 80 y 90 F y una pureza del acido entre 85 y 93 % en peso , da :

X4 = X1(1.12+0.1316* X8 - 0.00667*X8^2)

El numero de octano del alquilato , X7 , es una función de la razón isobutano/olefina, X8 , y de la pureza del ácido , X6 . Su ecuación de regresión no lineal , determinada bajo las mismas condiciones de operación como las del alquilato es :

X7 = 86.35+1.098*X8 - 0.038*X8^2+0.325(X6-89)

El análisis muestra que el factor de dilución , X9 , puede ser expresado como una función lineal de X10 . La ecuación de regresión lineal es :

X9 = 35.82 - 0.222* X10

Similarmente , x10 , puede expresarse como una función lineal del número de octano , X7 , por la ecuación :

X10 = 3*X7-133

La razón isobutano /olefina es :

X8 = (X2+X5)/X1

La entrada de isobutano , X5 , puede ser determinada por un balance de volúmen del sistema. La entrada (en barriles por día) es X1+X5 y la salida es X4 barriles por día . El volumen resultante de la mezcla puede ser menos que la suma de los volumenes de las entrads . Esta perdida volumetrica es constante en el sistema , y el porcentaje de perdida de volumen de la salida es 22 % del volumen de salida del alquilato . Así ,

X4 = X1 + X5 -0.22 * X4

o

X5 = 1.22 * X4 -X1

La pureza del ácido esta relacionada a la razón de adición de ácido nuevo , al factor de dilución , y la salida del alquilato por la ecuación :

1000*X3 = X4*X6*X9/(98-X6)

donde se asume que el % de pureza del ácido es de 98 , rearreglando la ecuación queda :

X6 = 98000*X3/(X4*X9+1000*X3)

El paso final es determinar la función de ganancia. Asumimos que la ganancia depende del valor de los productos de salida menos el costo de las entradas , y que todos los costos de operación son independientes o pueden despreciarse. Los valores que se usan son los siguientes :

Valor del producto = $0.063/barril

Costo de la olefina de alimentación = $ 5.04/barril

Costo del Isobutano de alimentación = $3.36 / barril

Costo del isobutano de recirculación = $0.035/barril

Acido nuevo = $10.00/barril

Por lo tanto la función de ganancia es :

0.063*X4*X7-5.04*X1-0.035*X2-10*X3-3.36*X5

Resultados y análisis.

PROBLEMA 1.[4]

Variable	Jung	Stevens	Obtenidos
Fa	12537	13338	12409
Fb	28379	30759	28442
Fr	324065	360538	328421
Fra	38251	45977	38453
Frb	120818	142784	121746
Frc	7266	7725	7250
Fre	136416	141784	137549
Frp	18405	18955	18518
Fg	2911	3130	2903
Fp	4763	4763	4763
Fd	33242	36159	33184
T(R)	654	672.61	633
I (%)	98.68	121.66	134.58

Las comparaciones entre los resultados obtenidos y los reportados en trabajos anteriores muestra ligeras variaciones en cuanto a la tasa interna de

[4] Los resultados de los trabajos de Jung y Stevens , estan reportados en Jung op. cit.

retorno esto se debe principalmente a las suposiciones que cada autor pudo haber hecho , por ejemplo si el reactor tenía un volumen constante o no , o en los rangos dados a las variables, también puede influir en menor forma el punto de inicio de las iteraciones y el % de error considerado para cada planteamiento. Los métodos que utilizaron los autores Jung y Stevensen fueron el de proyección del gradiente y de función de penalidad respectivamente.

PROBLEMA 2.

Variable	Haverly	Obtenidos
Costo X	9	11.4
Monto X	0	38.78
Costo Y	15	21.22
Monto Y	200	44.25
Costo A	6	4.18
Monto A	0	28.33
Costo B	16	11.03
Monto B	100	26.4
Costo C	10	6.67
Monto C	100	20.88
Tanque a X	0	27.61
Tanque a Y	100	24.33
C a X	0	12.22
C a Y	100	11.15
Azufre en T.	1	2.11
Función	400	1070

Este problema es un caso típico de existencia de multiples óptimos locales, Haverly[5] menciona que dependiendo de nuestra suposición inicial el algoritmo puede converger a uno de los multiples puntos óptimos . En este caso la solución emcontrada corresponde a una que nos da un mayor beneficio que la mencionada por Haverly .

[5] Op.cit.

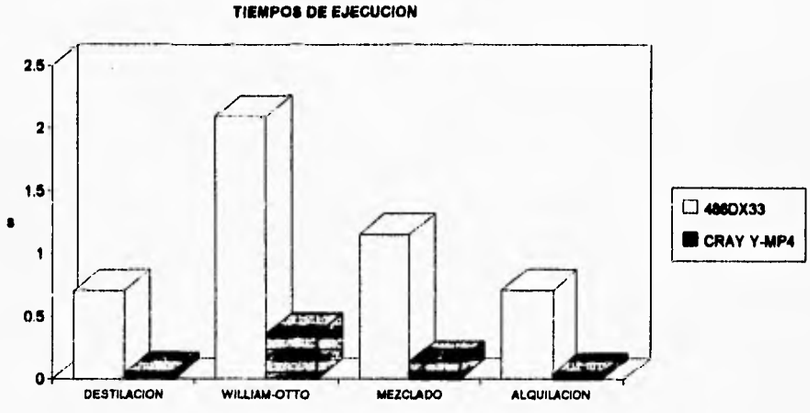
PROBLEMA 3.

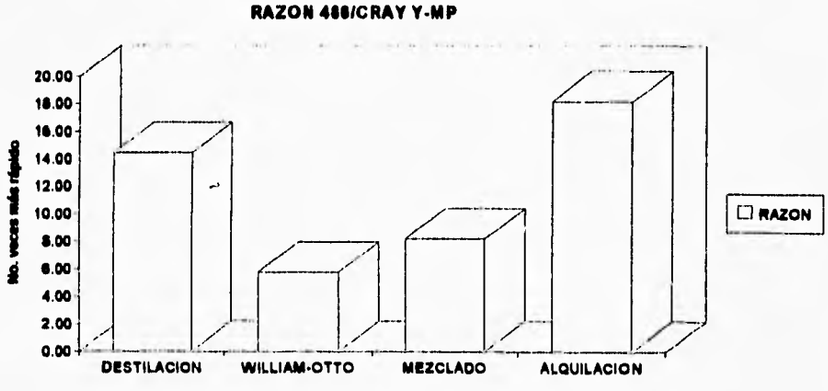
Variable	Liebman	Obtenidos
X1	1728.92	1790.55
X2	16000	12999.8
X3	98.15	107.97
X4	3056.46	3052.36
X5	2000	1977.04
X6	90.61	92.21
X7	94.18	96.22
X8	10.41	8.1
X9	2.61	2.21
X10	149.56	153.42
función	1161.33	1300.97

Liebman usó el método del Gradiente Reducido para llegar a la solución del problema ,los resultados obtenidos en este trabajo son muy similares y realmente las diferencias pueden deberse a suposiciones adicionales que el autor pudo hacer.

En cuanto a los tiempos de ejecución se refiere se comparan las corridas de los tres problemas junto con la simulación de destilación multicomponente que se hicieron en una máquina con procesador Intel 486DX/33 con sus contrapartes en la CRAY Y-MP 4.

Los resultados muestran que la corrida del código vectorizado y la capacidad de la CRAY Y MP 4 reduce significativamente el tiempo de ejecución del programa, como se aprecia en las gráficas siguientes.


TIEMPOS DE EJECUCION
2.5
2
1.5
1
0.5
0
s
DESTILACION
WILLIAM-OTTO
MEZCLADO
ALQUILACION
486DX33
CRAY Y-MP4


RAZON 486/CRAY Y-MP
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
No. veces más rápido
DESTILACION
WILLIAM-OTTO
MEZCLADO
ALQUILACION
RAZON

Conclusiones y Recomendaciones

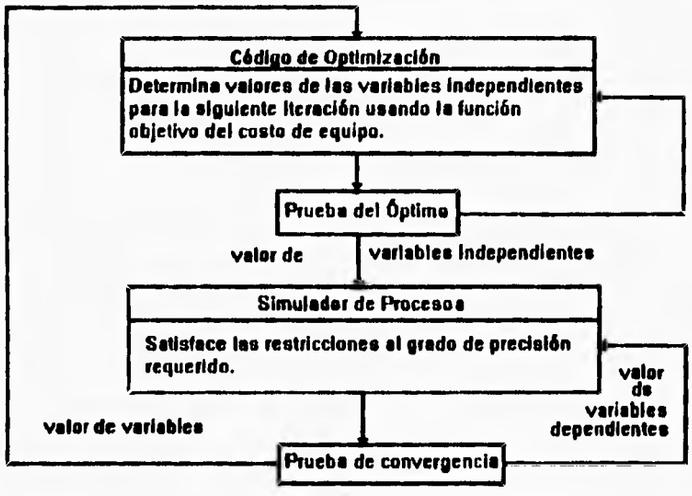
Los resultados en cuanto al tiempo de ejecución de la subrutina de optimización muestran que efectivamente el procesamiento en paralelo aumenta la velocidad de la rutina numérica entre 6 y 18 veces dependiendo de la carga computacional del problema comparandola con los tiempos de las corridas de una máquina secuencial.La razón de que el aumento en la velocidad haya superado al máximo teórico de 4 se debe a que por diseño un solo procesador de la supercomputadora es más poderoso que un procesador para PC 486 a 33 Mhz , ya que este solo alcanza los 40 MIPS como máximo , mientras que el procesador de la CRAY YMP llega a alcanzar 166 MIPS.

Lo importante del uso de subrutinas paralelizadas en la supercomputadora CRAY Y-MP es que estas interactuen con los paquetes comerciales disponibles.

De tal manera que una forma de aprovechar los recursos de la supercomputadora es en este caso haciendo la subrutina de optimización más eficiente y con capacidad para problemas de gran tamaño.

Sería recomendable acoplar el código de optimización[1] con simuladores de proceso modulares secuenciales tal como lo es ASPEN - PLUS . Cabe señalar que el acoplamiento con simuladores basados en ecuaciones sería mucho más eficiente que un simulador secuencial , pero no se puede negar la amplia aceptación de estos últimos en la industria ya que son mucho más fáciles de usar que la programación directa de simuladores basados en ecuaciones. La estructura que combinaría ambos es la siguiente :

1 IMPORTANTE : LA SUBRUTINA DE PROGRAMACIÓN CUADRÁTICA SUCESIVA SE ENCUENTRA DISPONIBLE EN SUS VERSIONES SECUENCIAL Y PARALELA EN EL DEPTO. DE INGENIERÍA QUÍMICA DE LA FACULTAD DE QUIMICA , UNAM.



Cabe enfatizar que en este trabajo se probó solo un nivel del paralelismo , ya que , como se mencionó a lo largo del trabajo , este tiene varios niveles que potencialmente se pueden aplicar a la ingeniería química.

Las desventajas de esta técnica es que es dependiente de la arquitectura de la máquina , por lo tanto no es transportable . Los sistemas son más costosos que los convencionales.

Anexo 1

La supercomputadora CRAY Y-MP 4/432.

Breve reseña histórica.

La primera supercomputadora que comenzó a operar en 1975 fue la ILLIAC-IV , la cual consistía de 64 elementos de procesamiento , cada uno conteniendo una unidad aritmético - lógica y una memoria local . Durante los mediados de los 1970's se enfocó el desarrollo hacia sistemas pipelining . Las dos grandes supercomputadoras que aparecieron en esta década con procesadores vectoriales fueron la STAR-100 de Control Data Corporation y la ASC (Advanced Scientific Computer) de Texas Instrument . La primera supercomputadora que ganó aceptación fue la CRAY -1 desarrollada por Cray Research Inc. y fue instalada en 1976 . Esta máquina fue el patrón a seguir por otros diseños desarrollados posteriormente por Hitachi , Fujitsu , Nippon Electric en Japón e incluso por la misma Cray Research Inc.

A finales de la década de 1970's aparecieron una nueva clase de computadoras llamadas de "procesamiento de arreglos periféricos " . Estas realmente eran maquinas de procesadores vectoriales que funcionaban como dispositivos periféricos para computadoras secuenciales digitales .

Aparecieron algunas máquinas enfocadas a aplicaciones especializadas . Un ejemplo es la AD10 y AD100 de Applied Dynamics International , principalmente útil en la simulación de sistemas dinámicos descritos por EDO (Ecuaciones Diferenciales Ordinarias) Un fenómeno particularmente importante en la evolución de sistemas computacionales orientados a la simulación fue la aparición de las minisupercomputadoras en los mediados de la década de 1980's. Un subconjunto importante de estas minisupercomputadoras son aquellas en las que se implementó el concepto de arquitectura MIMD , implicando una red de elementos de procesamiento idénticos [1].

La mayoría de estos sistemas emplean un sistema operativo basado en UNIX y cuentan con compiladores de FORTRAN 77 , C y algunas veces Pascal.

En cuanto a los sistemas Cray , después de la aparición de la CRAY - 1 A en 1976 siguieron los desarrollos de la familia CRAY X- MP en 1982 y CRAY - 2 en 1985 presentando la primera maquina de la familia CRAY Y - MP en 1988 .

Hardware de la Cray Y- MP 4/32.

Físicamente la supercomputadora CRAY se compone de varios gabinetes : el CPU , las unidades de disco , la unidad de refrigeración y la unidad de control de energía y refrigeración.

El CPU ocupa un área de 1.5 m2 , pesa 2450 Kg y mide 1.9 m de altura.

Esta computadora consume 200 KVA de potencia proporcionados por un sistema de potencia ininterrumpida que cuenta con un banco de baterías y una planta de emergencia . Opera con corriente de 400 Hz por lo que se requiere de un motor -generador para convertir la corriente de 60 Hz que se usa normalmente a la frecuencia de operación.

El CPU y las unidades de disco tienen un sistema de refrigeración estructural por líquido con propiedades dieléctricas especiales (Fluorinert) , y una unidad intercambiadora de calor con agua.

Requiere para su funcionamiento de un ambiente de temperatura controlada con aire limpio (sin partículas de polvo) y seco para evitar condensaciones . Esto se obtiene con sistemas de aire acondicionado, dehumidificadores y filtros.

Características.

El equipo instalado tiene 4 procesadores diseñados con capacidad de trabajar en paralelo y realizar operaciones matemáticas escalares o vectoriales. Dichos procesadores son capaces de direccionar en su totalidad a la memoria principal que es de 512 Mb. El reloj del sistema tiene un ciclo de 6 ns lo que da una frecuencia máxima aproximada de 167 MHz.

Cada procesador puede rendir teóricamente 166 millones de instrucciones por segundo , para un total de 664 MIPS como rendimiento teórico pico . Cabe hacer notar que este rendimiento se obtiene solamente si trabajan los 4 procesadores simultáneamente a toda su capacidad , lo cual no es su condición cotidiana de trabajo.

Las operaciones matemáticas de punto flotante pueden ser de precisión sencilla o doble (64 o 128 bits) . La memoria central del CPU es de 32 Megapalabras (1 Palabra = 64 bits) . Esta cantidad es equivalente a 256 Mbytes, aunque la CRAY opera con palabras completas de 64 bits, por eso no se usa el término "byte " en la literatura CRAY . La memoria tiene un sistema de corrección de errores en un solo bit y detección de errores dobles (SECDEC) .

Además de la memoria central existe una memoria temporal (Buffer Memory) de 4 Mp (32 MB) , esta sirve como interfaz entre los procesadores y los subsistemas restantes del CPU . Estos subsistemas controlan funciones de E/S con memoria central , periféricos de comunicaciones , de almacenamiento secundario y otros.

También cuenta con un banco de memoria auxiliar RAM de 128 MP (1 GB). Este dispositivo se conoce como SSD o "Solid State Storage Device" . Gracias a este dispositivo no es usada ninguna memoria de tipo cache . La velocidad de

transferencia entre esta memoria y la memoria central del CPU es de 1000 MBytes/s a través de canales especiales . No existe el concepto de memoria virtual ya que la memoria disponible bajo este mecanismo es demasiado lenta.

Una parte de la CRAY Y-MP es el subsistema de entrada/salida (IOS) , que es el punto de distribución de datos del CPU . El IOS los reúne y distribuye por una variedad de redes internas que comunican a los procesadores con los discos , memoria central y SSD , periféricos de comunicaciones y otras máquinas o periféricos múltiples.

Existen 10 canales de E/S ; 4 de ellos son de baja velocidad (6MB/s) , 4 son de mediana velocidad (100 MB/s) y 2 son de alta velocidad (1000 MB/s) . El equipo instalado posee 8 unidades de disco modelo DD-41 con capacidad de 38.4 GB , con una velocidad de transferencia de 9.6 MB/s y el tiempo de acceso promedio de 16 ms.

Para conectar a la CRAY a la red universitaria existen 2 ruteadores con interfaces Ethernet y FDDI cada uno . El protocolo de comunicación que usa el sistema operativo es TCP/IP (a nivel de red y transporte según el modelo de referencia OSI) [2].

Software

El sistema operativo de CRAY es UNICOS 7.0 el cual esta basado en UNIX (System V Release 4 de los laboratorios AT&T , con extensiones de Berkeley) . Dicho sistema operativo tiene capacidad de multiproceso y multiprogramación así como un sistema de archivos distribuidos en diferentes unidades físicas. La supercomputadora de la U.N.A.M. cuenta con software de aplicación especializada de diferentes áreas , bibliotecas de rutinas matemáticas ya paralelizadas así como dos compiladores de los lenguajes de FORTRAN 77 y C .

El Compilador CF77[1]

Las capacidades del procesamiento en paralelo existe en diferentes niveles y puede ser usada de varias maneras en los sistemas de cómputo de la Cray Research , Inc. (CRI)

La evolución del software de procesamiento en paralelo de CRI consiste en tres implementaciones: macrotarea , microtarea y Autotarea . La forma de Macrotarea requiere que los programadores modifiquen sus códigos para explotar el paralelismo haciendo uso de varias llamadas a librerías especificas de CRI. El modo de Microtarea es más versátil que las Macrotarea , ya que las llamadas a librerías son reemplazadas con directivas de compilación. La más reciente implementación , la Autotarea , combina los mejores aspectos de la Microtarea con dos mejoras fundamentales :

- La Autotarea puede ser completamente automática ; esto quiere decir , que no requiere la intervención del programador , aunque el programador es libre de interactuar con el sistema de Autotarea para una mejor ejecución.
- La Autotarea puede explotar el paralelismo al nivel de ciclos DO .

El sistema de compilación CF77 , se compone de los subsistemas FPP , FMP , y del compilador CFT77.

[1] IMPORTANTE : LA EXPLICACION DETALLADA DE CADA UNA DE LAS OPCIONES SE ENCUENTAN EN LOS MANUALES CRAY LISTADOS EN LA BIBLIOGRAFIA . POR TANTO SE RECOMIENDA SU CONSULTA AL LECTOR INTERESADO .

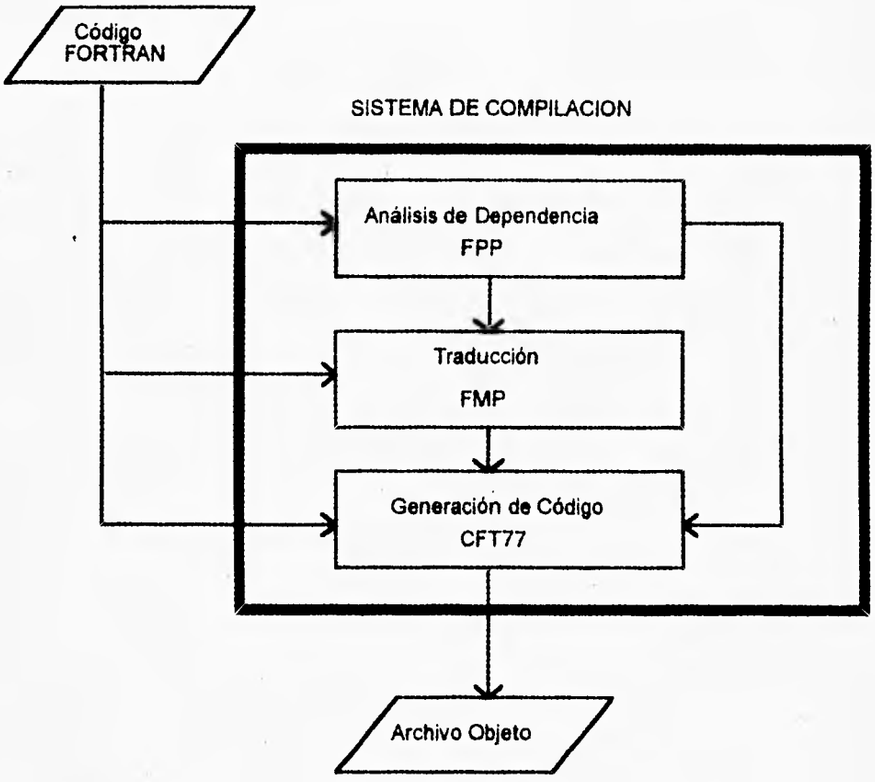
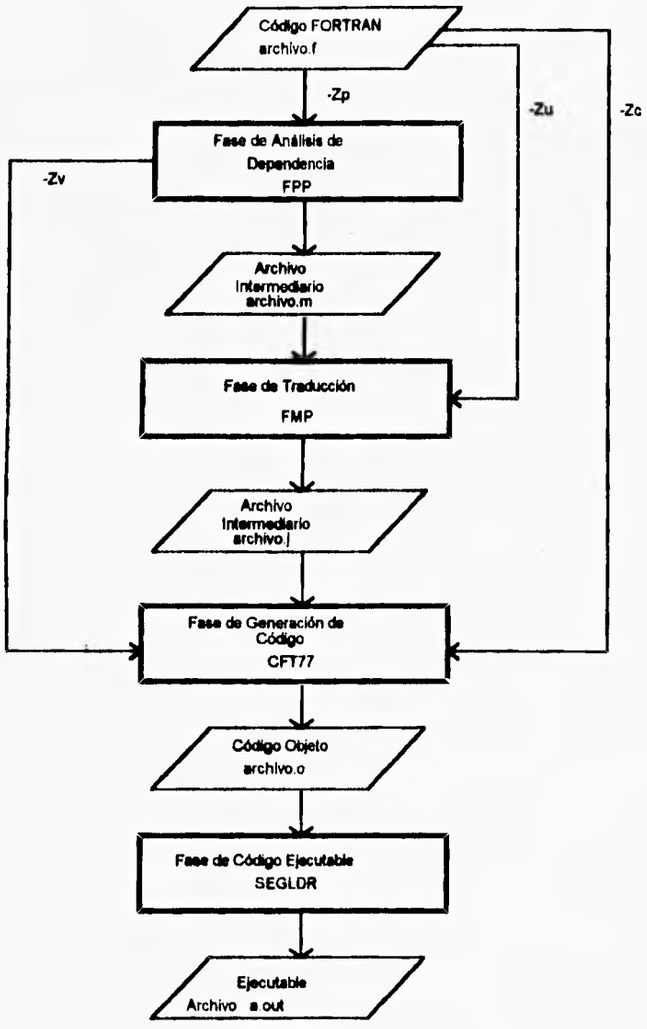

Código
FORTRAN
SISTEMA DE COMPILACION
Análisis de Dependencia
FPP
Traducción
FMP
Generación de Código
CFT77
Archivo Objeto

fig. a1.1.

fig.a1.2.

FASES DE AUTOTAREA PARA UNICOS USANDO cf77



FPP.

FPP es la fase de *análisis de dependencia* , toma el código fuente original de Fortran , revisa el paralelismo existente en las unidades del programa y produce un archivo con código fuente transformado como salida . Algunas de las transformaciones que ejecuta FPP son las siguientes :

- Agrega directivas de Autotarea con la lista de variables privadas y compartidas donde la ejecución en paralelo es posible.
- Agrega directivas CDIR@ IVDEP antes de los ciclos que pueden ser vectorizados.
- Expande los procedimientos externos , si es posible.
- Reestructura los ciclos anidados.
- Reemplaza ciertas partes del código con llamadas a rutinas de librería altamente optimizadas .
- Genera código vectorial o escalar.
- Convierte ciclos IF a ciclos DO , donde es posible , para mejorar la vectorización y la paralelización.
- Reordena los enunciados para remover la dependencia de información.

FMP.

La función primaria de la fase *de Traducción* , FMP , es transformar un código fuente de Fortran para la multitarea. Las directivas de Autotarea son traducidas a funciones intrínsecas de Autotarea o a llamadas a librerías.

La salida del traductor de Fortran es un archivo de código fuente con llamadas a rutinas de librería que dependen de la arquitectura de la máquina incluye funciones intrínsecas del compilador para controlar la ejecución en paralelo.

CFT77.

La fase de generación de código es el compilador CFT77 , que toma la salida del traductor y produce código máquina ejecutable .

Cada una de estas fases del sistema de Autotarea contribuye al tiempo total de compilación para la Autotarea. El tiempo de compilación es generalmente una función del número de lineas del código fuente procesado en cada fase. Las transformaciones producidas por FPP usualmente resultan en un incremento en el número de lineas del código fuente. De cualquier modo , FMP puede incrementar substancialmente el tamaño del código fuente.

Ventajas del sistema de compilación.

Las tareas que se realizan en estas tres fases puede parecer redundante . Después de todo , cada fase toma el código de Fortran , lo analiza , y lo modifica. Parecería más simple ejecutar este trabajo en una sola vez , las razones por las cuales el sistema de Autotarea fue implementado de esta forma son :

Primero , las fases separadas permiten mayor flexibilidad para cambiar una fase sin afectar otras . Segundo , manteniendo fases separadas se tiene un impacto más pequeño en las funciones del compilador CFT77 , el cual tiene muchas otras funciones a parte del procesamiento en paralelo.

Estas tres fases crean opcionalmente los archivos de salida del código fuente , y permiten ver que es lo que la Autotarea realiza a nuestro programa , y nos permite alimentar directivas adicionales .

Podemos ver el código fuente generado como salida y comparar las diferencias con el original , y detectar paralelismo que el análisis de dependencia no encontró.

Niveles para la intervención del usuario con el sistema de Autotarea.

Como se explicó previamente , la Autotarea representa la tercera fase de desarrollo del software multitarea de CRI . De tal forma que permite al programador varias opciones de interacción con la Autotarea.

- Ninguna intervención (Sistema Automático)
- Insertar directivas de Autotarea para identificar el paralelismo donde no fue detectado automáticamente.
- Procesar código en Microtarea.
- Insertar directivas de Autotarea uno mismo en lugar de usar FPP.

Para muchos programas , el uso de este "modo automático " da un incremento de velocidad substancial. De cualquier modo, algunas veces uno sabe información acerca de la estructura del programa y acerca de los datos que no están disponibles para FPP . Por esta razón , las directivas proporcionan un camino para guiar a FPP . Todas las directivas son tratadas como comentarios por otros compiladores de FORTRAN, así se preserva su transportabilidad a otros sistemas.

Directivas.[2]

Uno puede pasar información a todas las fases del sistema de compilación insertando directivas dentro del código fuente . FPP , FMP y CFT77 tienen su propio conjunto de directivas. Por ejemplo , uno puede insertar directivas para instruir a FPP donde ejecutar o no el análisis de dependencia vectorial y paralela. Los siguientes tipos de directivas están disponibles para usarse con el sistema de Autotarea :

- Directivas FPP

[2] EXISTEN GRAN NUMERO DE DIRECTIVAS QUE EL SISTEMA O EL PROGRAMADOR PUEDE AGREGAR AL PROGRAMA , LA DESCRIPCION DE CADA UNA DE ELLAS SE ENCUENTRA EN EL MANUAL DE PROCESAMIENTO EN PARALELO DE CRAY [4] DE LA BIBLIOGRAFIA CONSULTADA.

- Directivas FMP
- Directivas de Compilación.

Las directivas FPP son lineas especiales de código comenzando con CFPP$, las directivas FMP son CMIC$, mientras que las directivas de compilación son CDIR$ y CDIR@.

Desempeño de la autotarea.

La intención de la Autotarea es proveer un mecanismo para el multiprocesamiento automático.

El *Multiprocesamiento* implica el uso de múltiples procesadores para llevar a cabo el trabajo : El objetivo de la Autotarea es que el programa desempeñe el mismo trabajo que el programa original en menos tiempo.

Expectativa de desempeño para la vectorización.

Cuando el concepto de vectorización automática fue introducido , había una tendencia a esperar que, comparado a un código escalar , habría mucho mayor velocidad del programa de Fortran , con poco costo de desarrollo . En efecto , una gran velocidad se obtuvo para ciclos de pequeños programas que podrían ser vectorizados (10 a 20 veces).

Esta velocidad no se observaba , en programas de aplicación reales , y esto se debía primero a que estos programas contenían ciclos que eran tan grandes , complejos , que contienen mucha dependencia de datos , o tienen un flujo de control que es difícil de seguir por el compilador , por lo que no los vectorizaba. Otra razón era que la mayoría de los programas reales casi siempre contienen algoritmos que son intrínsecamente escalares . Los algoritmos escalares dominan el desempeño , es lo que la Ley de Amdahl establece : el desempeño es dominado por el componente más lento.

Con el tiempo los compiladores llegaron a ser más eficientes . Los usuarios ahora pueden vectorizar los programas rompiendo los ciclos en ciclos más

pequeños que puedan ser vectorizados . Los compiladores vectorizan ciertos tipos de dependencias de datos e imprimen mensajes informativos indicando porque otros ciclos no pueden ser vectorizados. El flujo de control dentro de los ciclos se mejora. El resultado de este trabajo es que muchos programas llegan a ser vectorizados entre el 70% y el 80% . Esto es , de 70 % a 80% del tiempo de ejecución se gasta en la ejecución de instrucciones vectoriales . Estos programas se ejecutan 2 a 4 veces más rápidos que el código escalar original.

Ley de Amdahl para la vectorización.

La razón para que el código se ejecute sólo de 2 a 4 veces más rápido que su código escalar equivalente y no 10 o 20 veces se explica por la Ley de Amdhal la cual establece que el desempeño del programa es dominado por su componente más lento . Para la vectorización el componente más lento es el código escalar . Para la multitarea el componente más lento es el código serial. Una formulación de esta ley para código vectorial, que es R veces más rápido que el código escalar , se muestra en la siguiente ecuación :

$$S_v = \frac{1}{f_s + \frac{f_v}{R_v}}$$

S_v = máxima velocidad esperada de
de la vectorización
f_v = fracción del programa que esta
vectorizado
f_s = fracción del programa que es escalar.
R_v = razón de tiempo entre el procesamiento
escalar y vectorial

Para los sistemas CRI , Rv se encuentra entre 10 y 20 . Para un programa que es 50% vectorizado , esta fórmula da una velocidad de solo 1.8 . Para un programa que esta 80% vectorizado , la velocidad es solo de 3.6 . Solo con 100 %

de vectorización el programa alcanzaría una velocidad de 10 . No siempre es fácil alcanzar entre el 70% y el 80% de vectorización en un programa .

Adicionalmente , la fórmula anterior asume que todo el código vectorial es Rv veces más rápido que el código escalar equivalente . En realidad , no todo el código vectorial es Rv veces más rápido que el código escalar . Cuando el código vectorial tiene vectores pequeños de 2 o tres elementos , el código escalar es usualmente más rápido. Una velocidad real para un programa que esta 80% vectorizado puede ser 3.0 .

Expectativas de desempeño para la Autotarea.

La Multitarea se usa para disminuir el tiempo real de ejecución (wall- clock execution time) para un programa relativo al tiempo que se requiere cuando se ejecuta en un solo procesador.

Un programa diseñado para multitarea generalmente mantiene la misma carga de trabajo para los procesadores que el programa original pero cuando el trabajo se ejecuta entre varios procesadores , el tiempo requerido para completar el trabajo debe ser menor.

Esta es una distinción importante entre el procesamiento en paralelo y la vectorización . Mientras que la vectorización usualmente disminuye ambos , el tiempo de CPU y el tiempo real , la multitarea disminuye sólo el tiempo real . En efecto , la multitarea generalmente incrementa el tiempo de CPU porque se requiere código adicional , para la sincronización de los procesadores. En un sistema , la razón de velocidad , para un programa en multitarea puede ser calculado de la siguiente manera :

$$\text{Razon de Velocidad} = \frac{\text{Ejecución de tiempo real para un procesador}}{\text{Ejecución de tiempo real en multitarea}}$$

Con N CPU's , la razón de velocidad tiende a N.

La Ley de Amdahl para multitarea.

La formula de Amdahl para la multitarea es la siguiente :

$$S_M = \frac{1}{f_s + \frac{f_p}{N}}$$

S_M = máxima velocidad esperada de
de la Multitarea
f_p = fracción del programa que esta
paralelisado.
f_s = fracción del programa que es serial.
N = Número de procesadores disponibles
para el procesamiento en paralelo

Ley de Amdahl para multitarea.

Velocidad máxima teórica Sm , en N CPU's con % de paralelismo.

% de	Numero de CPU'S							
Paralelismo	2	4	8	16	32	64	128	256
0.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
10.00	1.05	1.08	1.10	1.10	1.11	1.11	1.11	1.11
20.00	1.11	1.18	1.21	1.23	1.24	1.25	1.25	1.25
30.00	1.18	1.29	1.36	1.39	1.41	1.42	1.42	1.43
40.00	1.25	1.43	1.54	1.60	1.63	1.65	1.66	1.66
50.00	1.33	1.60	1.76	1.88	1.94	1.97	1.98	1.99
55.00	1.38	1.70	1.93	2.06	2.14	2.18	2.20	2.21
60.00	1.43	1.82	2.11	2.29	2.39	2.44	2.47	2.49
65.00	1.48	1.95	2.32	2.56	2.70	2.78	2.82	2.84
70.00	1.54	2.11	2.56	2.91	3.11	3.22	3.27	3.30
75.00	1.60	2.29	2.91	3.37	3.66	3.82	3.91	3.95
80.00	1.67	2.50	3.33	4.00	4.44	4.71	4.85	4.92
82.00	1.69	2.60	3.54	4.32	4.86	5.19	5.36	5.46
84.00	1.72	2.70	3.77	4.71	5.37	5.78	6.00	6.12
86.00	1.75	2.82	4.04	5.16	5.99	6.52	6.82	6.98
88.00	1.79	2.94	4.35	5.71	6.78	7.48	7.88	8.10
90.00	1.82	3.08	4.71	6.40	7.80	8.77	9.34	9.66
91.00	1.83	3.15	4.91	6.81	8.44	9.60	10.30	10.69
92.00	1.85	3.23	5.13	7.27	9.20	10.60	11.47	11.96
93.00	1.87	3.31	5.37	7.80	10.09	11.83	12.94	13.58
94.00	1.89	3.39	5.63	8.42	11.19	13.39	14.85	15.71
95.00	1.90	3.48	5.93	9.14	12.55	15.42	17.41	18.62
96.00	1.92	3.57	6.25	10.00	14.29	18.18	21.05	22.86
97.00	1.94	3.67	6.61	11.03	16.58	22.15	26.61	29.60
98.00	1.96	3.77	7.02	12.31	19.75	28.32	36.16	41.97
99.00	1.98	3.68	7.48	13.91	24.43	39.26	56.39	72.11
99.10	1.98	3.89	7.53	14.10	25.02	40.84	59.73	77.69
99.20	1.98	3.91	7.58	14.29	25.64	42.55	63.49	84.21
99.30	1.99	3.92	7.63	14.48	26.29	44.41	67.76	91.92
99.40	1.99	3.93	7.68	14.68	26.98	46.44	72.64	101.19
99.50	1.99	3.94	7.73	14.88	27.71	48.67	78.29	112.53
99.60	1.99	3.95	7.78	15.09	28.47	51.12	84.88	126.73
99.70	1.99	3.96	7.84	15.31	29.28	53.83	92.69	145.04
99.80	2.00	3.98	7.89	15.53	30.13	56.84	102.07	169.54
99.90	2.00	3.99	7.94	15.76	31.04	60.21	113.58	203.98
100.00	2.00	4.00	8.00	16.00	32.00	64.00	128.00	256.00

Prerequisitos para un alto desempeño.

La vectorización es todavía la mejor forma de obtener un alto desempeño por las siguientes razones :

- La Vectorización disminuye el tiempo de CPU y el tiempo real.
- La Vectorización disminuye el tiempo de trabajo total.
- Finalmente , es más fácil escribir código que pueda ser vectorizado que escribir código para ejecutarse en paralelo.

Con un sistema CRI , la vectorización puede incrementar el desempeño de los ciclos por un factor de entre 10 y 20 , mientras que la multitarea esta limitada a 4 veces la velocidad serial.

Características de los programas en Paralelo.

La siguiente lista describe las características de los programas en paralelo. La lista no contiene toda las características , ni tampoco garantiza que exista paralelismo o pueda ser explotado en programas donde existan estas características.

Cada uno debe de determinar el grado de paralelismo en sus programas.

Los siguientes tipos de programas tienen un alto potencial para el paralelismo :

- Programas que están altamente vectorizados.
- Programas cuyo desempeño esta dominado por ciclos anidados que no contienen llamadas a subrutinas.
- Programas cuyo desempeño esta dominado por ciclos con llamadas a subrutinas propias (esto es , subrutinas que no hacen llamadas externas.) .
- Programas que contienen ciclos anidados cuyo ciclo más interno pueda ser vectorizado.

- Programas que tengan ciclos internos que puedan vectorizarse, estos ciclos deben tener un gran número de iteraciones y operaciones . Si el número de operaciones es pequeño , el número de iteraciones debe ser grande para que la Autotarea sea efectiva. El número de iteraciones para ciclos internos debe ser mayor a 64.

- Programas con ciclos que hagan trabajo con multiplicaciones de matrices , recurrencias lineales de primer o segundo orden . Estos ciclos pueden ser reemplazados por llamadas a rutinas científicas , las cuales ya fueron optimizadas para el desempeño en multiprocesamiento.. Este tipo de programa representa una fuente indirecta de paralelismo porque el paralelismo existe en una rutina de librería.

El paralelismo y los balances de carga.

El paralelismo en una región del código es determinado por el número de particiones o pedazos , de trabajo independiente que existe en cada región. Si un ciclo de Autotarea tiene N iteraciones , entonces N es la extensión de paralelismo de ese ciclo. Esto significa que el ciclo puede efectivamente romperse en N pedazos independientes de trabajo . Para los ciclos de Autotarea internos , el grado de paralelismo es el numero de iteraciones dividido entre 64.

El *balance de carga* es el proceso que trata de asegurar la cantidad de trabajo hecho por cada procesador disponible para la tarea sea aproximadamente igual. Para ilustrar lo que significa el balance de carga con respecto al grado de paralelismo y tamaño de grano , se presenta el siguiente ejemplo.

Ejemplo :

Se tiene un ciclo externo DO (de 10 iteraciones) que esta diseñado para Autotarea , si el total de tiempo de ejecución es de 500 segundos , el grado de paralelismo es de 10 , y cada tarea se llevaría 50 segundos de trabajo. La figura a1.3. ilustra un escenario posible para el ciclo. Para propósitos de simplificación , cada iteración en este escenario es ejecutada completamente sin interrupción ; de

cualquier forma , la Autotarea no asegura que el procesador se mantendrá disponible durante la ejecución de todo el trabajo.

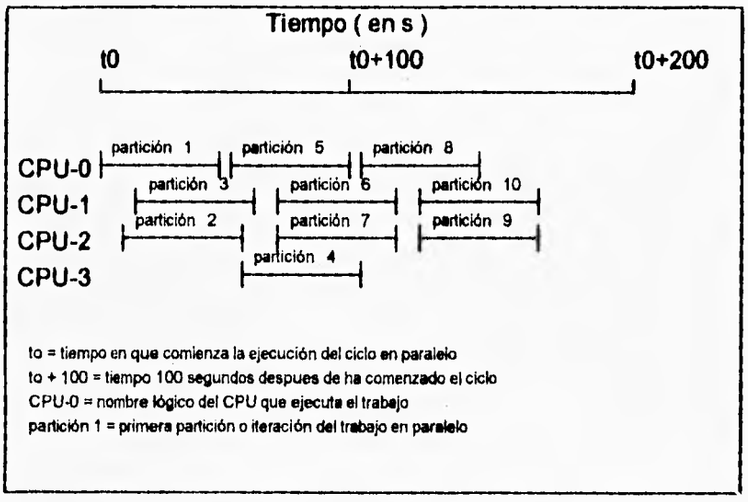


fig. a1.3

La secuencia de los eventos como los muestra la figura anterior es la siguiente :

- El CPU-0 comienza la ejecución de la primera iteración del ciclo al tiempo t0.
- La partición se completa 50 segundos después.
- Mientras otros procesadores están disponibles , ellos ejecutan una iteración en un tiempo. El número de CPU's disponibles durante el trabajo cambia durante la ejecución .
- Hay períodos de tiempo durante el cual un CPU ha terminado una iteración y es llamado para alguna otra requisición . Estos períodos se ilustran como espacios en blanco en la cadena de ejecución.

- El CPU-3 ejecuta una sola iteración (partición 4) y nunca regresa nuevamente.

Períodos de tiempo extra producidos por la Autotarea.

Existen períodos de tiempo extra a la ejecución que se crean debido al proceso de multiprocesamiento mismo . Hay cuatro tipos diferentes de estos períodos de tiempo introducidos por el procesamiento en paralelo:

tiempo gastado en la espera de semáforos. Se produce cuando los procesadores tienen que esperar cierto tiempo durante el proceso de sincronización.

tiempo gastado por la ejecución de código extra para Autotarea. Esto ocurre ya que el código preparado para Autotarea cuenta con un mayor número de instrucciones.

conflictos creados en el banco de memoria. Se pueden crear conflictos en el banco de memoria provocados por las referencias a memoria que hacen cada procesador.

menor desempeño de la vectorización. Esto se debe al uso de vectores de longitud corta que disminuyen la eficiencia de la Autotarea

Utilerías para optimización y monitoreo de programas.[3]

- Las utilerías para optimización y monitoreo de programas , son facilidades que ofrece el sistema Cray para evaluar el desempeño de nuestros programas e identificar y analizar las partes que requieren de optimización o donde se está ocupando la mayor parte del tiempo y recursos de máquina [3].

Existen utilerías específicas de acuerdo con el tipo de análisis que se va a realizar ; algunas requieren de directivas especiales al momento de compilación mientras otras se activan al momento de ejecución.

[3] SE RECOMIENDA CONSULTAR REFERENCIA 3 PARA EL USO DE ESTAS UTILERIAS . DE TAL MODO QUE SE OCUPEN EN LA DEPURACIÒN Y OPTIMIZACION DEL CODIGO EN FORTRAN.

Flowview

Flowview es una utilería que proporciona información sobre le tiempo de ejecución , número de llamadas y control de flujo de las subrutinas de un programa. Permite identificar cuales son las partes del código que están consumiendo mayor tiempo y que por lo tanto requieren de optimización. También genera un árbol de llamadas , es decir , un organigrama con la información acerca de cuales rutinas fueron llamadas por otras y evalúa que subrutinas son candidatos para "inlining" . El inlining es una técnica de optimización que consiste en inseertar el código de la subrutina dentro del programa y evitar la llamada . Rutinas típicas a las que se les aplica esta técnica son aquellas cuyo código es pequeño y que son llamadas muchas veces.

Para aplicarle Flowview a un programa en Fortran úsese la siguiente secuencia :

cf77 - F miprog.f

a.out

flowview

Donde miprog.f es el nombre del programa en Fortran a compilar. Flowview genera un archivo llamado flow.data donde se almacena la información . Mientras el programa no se vuelva a ejecutar , este archivo contiene la información de la última corrida.

HPM

El HPM (Hardware Performance Monitor) es un conjunto de registros que permiten monitorear los recursos en hardware que utiliza un programa al momento de ejecución . La información que proporciona el HPM se divide en los siguientes cuatro grupos:

GRUPO 0

0 Instrucciones ejecutadas

1 Tiempo de espera en periodos de reloj

2 Búsquedas del buffer de instrucciones a memoria.

3 Operaciones de suma en punto flotante

4 Operaciones de multiplicación en punto flotante

5 Operaciones de aproximación recíproca (división) en punto flotante

6 Accesos CPU - memoria

7 Acceso sistema E/S memoria

GRUPO 1

0 Semáforos

1 Registros compartidos

2 Registros y unidades funcionales tipo A (de dirección)

3 Registros y unidades funcionales tipo S (escalares)

4 Registros tipo V (vectoriales)

5 Unidades funcionales vectoriales

6 Referencias de tipo escalar a memoria

7 Accesos en bloques a memoria

GRUPO 2

0 Búsquedas del buffer de instrucciones a memoria

1 Conflictos en búsquedas a memoria

2 Accesos sistema E/S - memoria

3 Conflictos sistema E/S- memoria

4 Referencias de tipo escalar a memoria

5 Escrituras CPU-memoria

6 Accesos CPU-memoria

7 Conflictos CPU-memoria

GRUPO 3

0 Número de instrucciones especiales y de salto

1 Número de instrucciones a unidades funcionales.

2 Número de instrucciones escalares a memora

3 Númeero de instrucciones vectoriales enteras y lógicas

4 Número de instrucciones vectoriales en punto flotante

5 Número de instrucciones en punto flotante de carga y salvado a memoria

6 Número de operaciones vectoriales enteras y lógicas

7 Número de operaciones vectoriales en punto flotante.

Para utilizar HPM , no es necesario especificar ninguna directiva al momento de compilación sin embargo , sólo es posible obtener información respecto a uno de los grupos por corrida.

Para usar HPM la sintaxix es :

hpm -g# prog

Donde # es el número del grupo del que se desea obtener información y prog es el nombre del programa ejecutable que se obtuvo al compilar el programa en Fortran.

Perfview

Perfview es un programa que además de proporcionar información de flujo utiliza los registros del HPM para generar reportes de cada grupo a nivel subrutina.

Profview

Profview es una utilería que da información sobre la cantidad de ejecución en distintos segmentos del código de cada subrutina y por tanto nos permite identificar con exactitud las partes específicas de cada rutina que consumen mayor tiempo y recursos de máquina.

Job Accounting (ja)

La utilería ja proporciona información diversa de contabilidad referente a una sesión. El comando ja activa el sistema de contabilidad y todos los comandos que se ejecuten hasta que sea desactivado serán contabilizados para el reporte.

Implementación de un algoritmo en la CRAY Y-MP.

En el capítulo 1 se mencionaron algunos comentarios respecto a la implementación de los algoritmos en paralelo . En este anexo se remarcó la dependencia de los algoritmos en paralelo con la arquitectura de la máquina donde se desee implantar, en esta seción se describirán los pasos de la implementación en la supercomputadora CRAY Y-MP con un ejemplo .

El diseño de los algoritmos en paralelo , se enfrenta con varios problemas como son la manipulación de información , alojamiento de información , interferencias de memoria y comunicación entre procesadores . En general , los algoritmos numéricos reportados en la literatura caen dentro de dos clasificaciones:

-reformulación de algoritmos seriales en algoritmos concurrentes y

-algoritmos desarrollados para máquinas paralelas.

La mayoría del trabajo en algoritmos numéricos pertenecen a la primera categoría , i.e. , la descomposición de algoritmos seriales en tareas concurrentes. Ejemplos son las operaciones de matrices , los métodos iterativos para la solución de ecuaciones algebraicas y técnicas de extracción de eigenvalores.

La segunda categoría incluye los algoritmos implementados en determinadas máquinas de procesamiento en paralelo , muy pocos de ellos son reportados en la literatura , se les conoce como *algoritmos paralelos únicos* . En algunos casos el desempeño de algoritmos paralelos únicos es superior a sus contrapartes seriales.

Para el diseño de programas en paralelo de la segunda categoría no existe una metodología ya determinada , ya que todos los diseñadores se encuentran con la pregunta ¿cómo diseñar el paralelismo de una solución ? . Romero Salcedo [5] sugiere usar la técnica del análisis de sistemas estructurados conocida como diagrama de flujo de datos (DFD) . En el análisis y el diseño de una solución , se usa el DFD para crear una solución secuencial , en algunos lenguajes concurrentes como OCCAM se puede usar el DFD en forma directa , ya que se puede asociar los nodos de procesamiento del DFD con los procesos del programa OCCAM , similarmente , las flechas de flujo de datos se pueden asociar con los canales que unen a los procesos , y a través de las cuales fluyen los mensajes.

En cuanto a la reformulación de algoritmos secuenciales para su operación concurrente se pretende usar el paralelismo en forma de multitarea para mejorar la velocidad del mismo , pero no sus propiedades intrínsecas de convergencia o aproximación a la solución ya que estas seguirán estando planteadas por un lineamiento secuencial.

Las características de la CRAY Y MP y de su sistema de compilación CF77 hacen de ésta una máquina extremadamente eficiente para trabajar o producir algoritmos secuenciales reformulados , su uso en el planteamiento de algoritmos paralelos únicos no ha sido todavía explotado en la U.N.A.M. , ello es comprensible ya que presenta ciertas dificultades la arquitectura y el software como podrían ser los siguientes puntos :

1. El sistema de compilación aún con sus directivas esta diseñado para explotar la multitarea a nivel de ciclos internos DO del programa en FORTRAN 77 y no otro tipo de paralelismo.

2. FORTRAN 77 es un buen lenguaje de programación , pero fue pensado y desarrollado para máquinas secuenciales y nunca podrá superar a lenguajes concebidos para su uso exclusivo en máquinas paralelas como lo es OCCAM.

3. La CRAY Y MP fue diseñada para explotar la vectorización al máximo ya que la estructura de cada procesador así lo permite .

Habiendo mencionado estos puntos se puede comprender el porque la mayoría del software instalado en la CRAY Y MP son reformulaciones de algoritmos secuenciales , únicamente que se aprovecha la velocidad que ofrece la vectorización de sus programas y su posterior división en multitareas. Por lo tanto, para la implementación de un algoritmo en la CRAY Y MP funciona de manera adecuada la forma tradicional secuencial con algunas variantes , el procedimiento constaría de las siguientes etapas :

1. Planteamiento del modelo matemático del fenómeno o problema a resolver. Siempre esta es la etapa fundamental ya que los resultados obtenidos estarán en función de que tan bueno sea el modelo planteado.

2. Codificación del algoritmo en FORTRAN 77. Se recomienda usar rutinas de las librerías matemáticas para los sistemas CRAY tanto como sea posible , este tipo de rutinas se encuentran ya optimizadas en cuanto a su tiempo de ejecución . También es recomendable ejecutar un programa batch y dejar la interface del programa con el usuario a un equipo menor , como puede ser una workstation. Las pruebas de resultados pueden de igual forma ejecutarse en un equipo menor, de tal forma que el programa fuente que se mande a ejecutar al sistema CRAY este libre de errores en la producción de resultados o de sintaxis (bugs).

3. Ejecutar las fases FPP y FMP del sistema de compilación.

4. Revisar el programa fuente generado y agregar directivas faltantes donde se crea conveniente para la ejecución de la multitarea que el sistema pueda haber omitido en el modo automático.

5. Ejecutar la última fase de compilación obteniendo el programa en código ejecutable.

6. Probar el programa , si el tiempo de ejecución es satisfactorio fin , sino realizar una optimización del código , para ello se ocupan las utilerías de optimización y monitoreo de programas , ir al paso 3.

Ejemplo : Columna de destilación multicomponente no ideal.

Primero se debe plantear el modelo matemático de una columna de destilación multicomponente no ideal , para poder hacer la simulación de la misma.

Se puede comenzar haciendo algunas suposiciones que pueda delimitar el modelo :

1. El líquido en cada plato se mezcla perfectamente y es incompresible.

2. Las demoras del vapor por plato es despreciable.

3. La Dinámica del condensador y del reboiler no se simulará.

4. El vapor y el líquido están en equilibrio térmico , pero no en equilibrio de fase . Se usa una eficiencia para la fase de vapor para describir su desviación del equilibrio.

$$E_{nj} = \frac{y_{nj} - y^{T}_{n-1,j}}{y^{*}_{nj} - y^{T}_{n-1,j}}$$

donde

y^{*}_{nj} = composición del vapor en equilibrio con el líquido en el plato n con composición x_{nj}.

y_{nj} = composición real del vapor al dejar el plato n.

$y^{T}_{n-1,j}$ = composición real del vapor al entrar al plato n.

E_{nj} = Eficiencia del vapor para el componente j en el plato n.

Puede haber alimentaciones multiples , de líquido y vapor y también extracciones laterales. Tomando en cuenta a la fig. a1.4 se tiene:

donde

Número	Flujo	Composición	Temperatura
1	F_n^L	x_{nj}^F	T_n^F
2	F_{n-1}^V	$y_{n-1,j}^F$	T_{n-1}^F
3	L_{n-1}	$x_{n-1,j}$	T_{n-1}
4	V_n	y_{nj}	T_n
5	V_{n-1}	$y_{n-1,j}$	T_{n-1}
6	S_n^L	x_{nj}	T_n
7	L_n	x_{nj}	T_n
8	S_n^V	y_{nj}	T_n

Las ecuaciones que describen este plato son :

Balance de materia total

$$\frac{dM_n}{dt} = L_{n-1} + F_n^L + F_{n-1}^V + V_{n-1} - V_n - L_n - S_n^L - S_n^V$$

Balance de materia por componente (NC-1 por plato):

$$\frac{d(M_n x_{nj})}{dt} = L_{n-1}x_{n-1,j} + F_n^L x_{nj}^F + F_{n-1}^V y_{n-1,j}^F + V_{n-1}y_{n-1,j} - V_n y_{nj} - L_n x_{nj} - S_n^L x_{nj} - S_n^V y_{nj}$$

Balance de energía (una por plato)

$$\frac{d(M_n h_n)}{dt} = L_{n-1}h_{n-1} + F_n^L h_n^F + F_{n-1}^V H_{n-1}^F + V_{n-1}H_{n-1} - V_n H_n - L_n h_n - S_n^L h_n - S_n^V H_n$$

Equilibrio de fases (NC por plato)

$$y_{nj}^{*} = f(x_{nj}, P_n, T_n)$$

Se debe tener una buena relación para calcular el equilibrio liquido-vapor.

Otras ecuaciones adicionales al modelo serían las relaciones para calcular las propiedades físicas para obtener densidades y entalpías , una ecuación hidráulica del vapor para calcular el flujo del vapor a partir de las caídas de presión por plato , y una relación hidráulica para el líquido para saber el flujo del líquido .

Un algoritmo propuesto por Luyben (1989) [6] para este modelo es el siguiente :

1. Información de entrada sobre el tamaño de la columna , componentes , propiedades físicas , alimentaciones y condiciones iniciales (composiciones del líquido , flujo del líquido , suposiciones iniciales de temperatura en todos los platos)

2. Calcular el perfil de presión para los platos.

3. Calcular las temperaturas y composiciones de vapor de la información del equilibrio líquido-vapor . Para casos ideales se usaría la ley de Raoult en casos no ideales se agregaría la ecuación de los coeficientes de actividad.

4. Calcular entalpías del líquido y vapor.

5. Calcular flujo de vapor en todos los platos , comenzando en la base de la columna , usando las ecuaciones de balance de energía.

6. Evaluar todas las derivadas de las ecuaciones de balance por componente para todos los componentes en todos los platos más el reflujo y la base de la columna.

7. Integrar todas la ecuaciones diferenciales ordinarias.

8. Calcular las nuevas fracciones molares del líquido a partir de las demoras en cada plato.

9. Calcular el flujo del líquido a partir de las demoras en cada plato

10. Ir al paso 3 y repetir para el siguiente paso en el tiempo.

La siguiente etapa después de tener el algoritmo es codificarlo en FORTRAN 77 , para pasar a las etapas de vectorización y paralelización.

Desarrollo de aplicaciones en un ambiente de Supercómputo.

El mayor uso para las supercomputadoras CRI en cualquier parte del mundo es simular el comportamiento de sistemas técnicos y científicos complejos. Esto usualmente se hace en proyectos a largo plazo donde están involucrados todo un equipo bien estructurado de trabajo. Las diferentes computadoras disponibles en el mercado tienen diferentes áreas donde se posee un mejor desempeño . . Mientras que las supercomputadoras son primordialmente usadas para aplicaciones con un alto grado de vectorización , códigos y manejo de información muy grandes . otras máquinas de un equipo de trabajo deben tener hardware para aplicaciones gráficas, ya que la gran mayoría de las aplicaciones de supercómputo tienen una salida para visualización gráfica donde se representan los resultados obtenidos [4].

Debido a las características de hardware y software de estos sistemas descritos en el presente capítulo es necesario señalar que el máximo aprovechamiento se consigue cuando las aplicaciones tienen un código muy extenso, altamente vectorizado que requiera el procesamiento de gran cantidad de información. El desarrollo de software que no cumpla con estas características provoca un desperdicio de los recursos , ya que el paralelismo y la vectorización no es un gran beneficio a programas cortos con información que maneje vectores de pocos elementos, donde una PC con un solo procesador es bastante eficiente.

Referencias

1. Kerckhoffs , E.J. & Koppelaar , H. " Toward Parallel Intelligent Simulation " Delft University of Technology , Delft , The Netherlands.

2. Dirección de Cómputo para la investigación (D.G.S.C.A. - U.N.A.M.)Guía Introductoria a la computadora Cray Y-MP

3. Alvarez-Manilla, Mauricio Utilerias para optimización y monitoreo de programas D.G.S.C.A. , U.N.A.M.

4. Rühle R. Scientific Applications in a Supercomputer Environment University of Stuttgart Computer Center,West Germany Science and Engineering on Supercomputers 1990

5. Romero Salcedo, Manuel. " Programación paralela en OCCAM " IIMAS UNAM

6. Luyben , W. "Process modeling , simulation and control for chemical engineers" McGraw Hill Co , 1989.

Bibliografía Consultada.

1. Cray Research Inc.SQ-0138 UNICOS CFT77 Reference Card

2. Cray Research Inc. SR-0009 Fortran (CFT) Reference Manual

3. Cray Research Inc. SR-2011 UNICOS User Commands Reference Manual

4. Cray Research Inc. SG-3074 Compiling System , Volume 4 Parallel Processing Guide.