Mecánica Computacional Vol. 14, páginas 50-59
compilado por Sergio Idelsohn y Victorio Sonzogni
Editado por AMCA, Santa Fe, 1994

CALCULO PARALELO EN ANALISIS ESTRUCTURAL

Victorio E. Sonzogni
Grupo de Tecnología Mecánica - INTEC
CONICET - Universidad Nacional del Litoral
Güemes 3450, 3000 Santa Fe, Argentina

RESUMEN

Se realiza una presentación introductoria del cálculo paralelo aplicado a problemas de ingeniería estructural. El objetivo de la misma es proporcionar los conceptos básicos subyacentes en este tipo de cálculos. Para ello se brinda una noción somera acerca de las diferentes arquitecturas de hardware, acerca de los distintos modelos de programación y acerca de los lenguajes computacionales específicos. Se analiza también la estructura de un programa de elementos finitos y sus posibilidades de paralelización. Finalmente se menciona la aplicación de dos modelos de programación diferentes adecuados a distintos tipos de computadoras.

ABSTRACT

An introductory presentation on parallel computing in engineering is given. The aim is to bring basic ideas about hardware architecture, programming models and programming lenguajes issues related to parallel computing. The structure of finite element codes for structural analysis is discussed in connection with the parallelism. Two programming models are applied to non linear structural codes.

INTRODUCCION

Los grandes problemas de cálculo científico se resuelven en la actualidad mediante cálculos realizados en paralelo. Esta posibilidad se dió a partir de la disposición de computadoras capaces de operar con varios procesadores simultáneamente. El esquema físico del hardware evolucionó según diversos caminos, todos ellos conducen a computadoras denominadas genéricamente *paralelas* pero con características completamente distintas entre sí. Esta diversidad se traslada a la manera de programarlas y de utilizarlas.

Los desarrollos de software de aplicación ya no poseen la portabilidad de que gozaban en las computadoras secuenciales. Productos más o menos portables pueden obtenerse según la elección que se haga de arquitectura básica de hardware, de modelo de programación, de algoritmo de solución y de estándares de lenguaje y software utilitario. La eficiencia de un programa y su portabilidad marchan por caminos divergentes. Debe ponderarse cada una de ellas y realizar una elección que puede a veces resultar de un compromiso entre ambos requerimientos.

Existe por otro lado interés en aprovechar en la medida de lo posible el software específico desarrollado en computadoras secuenciales. La tarea de adaptar este software al nuevo ámbito de cálculo paralelo representa un emprendimiento de gran utilidad en lo inmediato. La complejidad de tal adaptación puede ser un elemento a tener en cuenta en la elección del modelo de trabajo.

En esta presentación se trata de brindar las nociones básicas para el cálculo en paralelo. Para ello se presenta, en un primer lugar, los distintos tipos de computadoras con procesadores en paralelo y se esbozan principios de clasificación de las mismas. A continuación se mencionan los niveles en que pueden realizarse los cálculo paralelos, como así también los modelos de programación que pueden adoptarse. Se incluye luego una referencia muy breve a los lenguajes para programación paralela. Luego se analiza la estructura

global de un programa computacional para análisis no lineal de estructuras por elementos finitos desde el punto de vista del cálculo concurrente. Finalmente se delineará la aplicación de dos modelos diferentes de programación.

COMPUTADORAS PARALELAS

Las computadoras tradicionales están basadas en la arquitectura denominada de von Neumann: en cada instante se está realizando *una* operación. Es decir hay un solo flujo de instrucciones, que se procesan secuencialmente. El desarrollo tecnológico permitió aumentar continuamente la potencia de cálculo desde el origen de las computadoras, pero esta línea de crecimiento se encuentra acotada por límites físicos como el de la velocidad de la luz. En efecto en un nanosegundo (este es el orden de magnitud del ciclo interno de reloj de las computadoras más veloces) la electricidad recorre una distancia de unos 30 cm. La distancia que media entre el procesador y la memoria puede ser una fracción importante de aquella. El desarrollo de las computadoras comenzo a orientarse, abandonando el esquema de von Neumann, hacia el procesamiento en paralelo.

La ejecución simultánea de varias operaciones puede realizarse en distintos niveles y de diferentes maneras. En su ejemplificación más clara se encuentran las computadoras que poseen varias unidades de procesamiento: las *computadoras paralelas*. Pero aún en computadoras con un solo procesador hay varios mecanismos para efectuar operaciones simultáneas:

- *múltiples unidades funcionales*: varias unidades independientes para realizar distintas funciones. Operan simultáneamente sobre diferentes datos;
- *"pipelining"*: sigue el principio de la *producción en serie* o *línea de montaje*. La unidad de cálculo está dividida en segmentos y cada uno realiza una operación en un ciclo de reloj. Cuando una operación consta de varios pasos, una vez que la "cadena de montaje" está llena, sale un resultado en cada ciclo de reloj;
- *instrucciones vectoriales*: las instrucciones vectoriales especifican una operación que se realiza sobre un grupo de datos ordenados en un arreglo unidimensional (vectores);
- *encadenamiento de operaciones*: la salida de una operación es dirigida directamente a la entrada de otra, sin esperar que la primera haya terminado;
- etc.

Entre las computadoras con varios procesadores existe una gran variedad de arquitecturas [1,2,3]. Son denominadas *computadoras paralelas* y algunas de ellas combinan además capacidad de cálculo vectorial. Se han hecho varios intentos de clasificación de estas computadoras y en general resultan de interés aquellas basadas en las siguientes características: *granularidad del hardware*, *conexión entre los procesadores*, *organización de la memoria* y *control de la ejecución*.

• *Granularidad del hardware*: Pueden agruparse en computadoras denominadas de *granularidad gruesa*, *media* y *fina*. Las de granularidad *gruesa* consisten en un pequeño número de procesadores muy potentes (2 a 16, o hasta 32 procesadores). Por ejemplo las CRAY, las NEC, IBM 3090, Alliant FX/8, Silicon Graphics 4D/340, entre varias. Las computadoras de granularidad *fina* poseen cientos o miles de procesadores relativamente pequeños. Ejemplo de esto son: la Connection Machine CM2, con 65536 procesadores, la DAP-610 con hasta 4096 procesadores y la MasPar MP-1 de 1024 a 16384 procesadores. Las computadoras de *mediana* granulometría están comprendidas entre las dos clases anteriores y a ella pertenecen las denominadas *hipercubos*: iPSC/860 con 8 hasta 128 procesadores, NCUBE de 64 - 2048 procesadores y Meiko con 4 - 512 procesadores.

• *Conexión entre los procesadores*: Describe la topología del hardware. Esta conexión puede ser de diferentes tipos: llaves en puntos de cruce, "bus" de conexión (o bien anillos cerrados), grillas, hipercubos, o esquemas híbridos.

• *Organización de la memoria*: La forma en que los procesadores pueden acceder a la memoria es de fundamental importancia en la forma de programar para esa computadora paralela. La memoria puede ser *compartida* y en este caso todos los procesadores pueden acceder a cualquier posición en la memoria. La memoria también puede ser *local* y en este caso cada procesador tiene acceso solamente a su propia memoria. La comunicación de la información entre procesadores, en este último caso, se realiza por intercambio

de mensajes. La tendencia es utilizar una organización *jerárquica* de la memoria con una gran memoria compartida y con memorias locales de rápido acceso para cada procesador.

Un tópico diferente se refiere a la localización física de la memoria. Desde este punto de vista ella puede ser *centralizada* o *distribuida*. Puede hacerse una asociación directa entre la ubicación física y la forma de acceso a la memoria y generalmente las memorias *centralizadas* tiene acceso *compartido*, mientras que las memorias *distribuidas* son de acceso *local*. Sin embargo existen algunas excepciones (como en el caso de la CM-2, BBN y Myrias) donde la memoria es *distribuida* pero de acceso *compartido*.

• *Control de la ejecución*: La taxonomía de Flynn divide a las computadoras en cuatro grupos: *SISD*, *SIMD*, *MISD* and *MIMD*.

- *SISD*: *Single Instruction - Single Data*. Este es el caso de la arquitectura de von Neuman, secuencial, donde hay un solo flujo de instrucciones y un solo flujo de datos.
- *SIMD*: *Single Instruction - Multiple Data*. Una misma instrucción es ejecutada simultáneamente por varios procesadores con diferentes datos. Son ejemplos de estas máquinas la ILLIAC IV y algunas computadoras de grano fino como las Connection Machine CM-1 y CM-2, la DAP-610 y la MasPar MP-1.
- *MISD*: *Multiple Instruction - Single Data*. No es fácil encontrar ejemplos de este tipo de máquinas entre las computadoras comerciales. Pero los denominados "sistolic arrays" pueden ser vistos como computadoras MISD donde diferentes procesadores ejecutan diferentes instrucciones sobre un mismo flujo de datos.
- *MIMD*: *Multiple Instruction - Multiple Data*. Hay varias instrucciones que se operan simultáneamente sobre distintos flujos de datos. La mayoría de las supercomputadoras actuales pertenece a este tipo. Las hipercubos (Intel-iPSC/860, NCUBE) así como también las computadoras de grano grueso (CRAY, IBM 3090, Alliant, Convex, Nec, etc.) son máquinas MIMD.

Algunos tópicos privilegian items de otra clasificación. Por ejemplo:

* Computadoras de *grano grueso* (CRAY, Encore, IBM, Alliant,etc.) acceden a una memoria *compartida*, mientras que aquellas de *grano medio* o *fino* son generalmente de memoria *local*;
* Asimismo las computadoras de *grano grueso y memoria compartida* en general están conectadas por un "bus", mientras que aquellas de *memoria local* poseen conexiones de tipo hipercubos (*grano medio*: iPSC 860, NCUBE, etc.) o de "vecinos próximos" (*grano fino*: CM-2, MP-1, etc.)
* Las computadoras *SIMD* tienen *memoria distribuida* (CM-2, MasPar MP-1, DAP-610). Aquellas *MIMD* pueden a su vez tener memoria *compartida* (CRAY, Alliant, IBM, Convex, NEC, etc.) o *local* (iPSC/860, Meiko, NCUBE, etc.)

NIVELES DE PARALELISMO

El procesamiento en paralelo puede realizarse en diferentes niveles: a nivel de ejecución de programas ("job level"), a nivel de programa, a nivel inter-instrucción y a nivel intra-instrucción. En el nivel inferior (intra-instrucción) resulta esencial el papel del hardware, mientras que el del software aumenta hacia los niveles más altos:

- *nivel de ejecución de programas* : es llevado a cabo principalmente por el sistema operativo. Puede tomar la forma de *multiprograma* o bien de *multiproceso*.
- *nivel de programa* : realizado por el software, puede ser interpretado a su vez en distintos niveles: desde tareas de *fina granularidad* (también denominadas "microtasking" en las computadoras CRAY), tales como la subdivisión de un ciclo DO entre diferentes procesadores, hasta tareas de *granularidad gruesa* ("macrotasking", o "multitasking" de CRAY), que implican procesar en forma concurrente secciones enteras del programa.
- *nivel inter-instrucción* : realizada por el compilador, requiere que éste analise las dependencias entre datos de diferentes instrucciones y superponga sub-operaciones de diferentes instrucciones.
- *nivel intra-instrucción* : realizada por el hardware, como en el caso de *"pipelining"*.

Desde el punto de vista del ingeniero, como utilizador de máquinas paralelas y en la elaboración y/o adaptación de programas computacionales, la clase de paralelismo que resulta de mayor importancia es la asociada al nivel del programa. En este contexto, como ya se ha mencionado, pueden introducirse diferentes niveles de paralelismo dentro de la parte de programación:

1. Un paralelismo de granularidad fina*: Esto significa, para una computadora de memoria compartida, que la cantidad de operaciones realizadas por cada procesador antes de comunicarse con los otros es pequeña. Para computadoras de memoria distribuida la granularidad esta definida como la relación entre la cantidad de operaciones y la cantidad de datos a ser transmitidos en cada tarea. Y el caso típico de granularidad fina es cuando un procesador ejecuta operaciones sobre un índice de un vector, por ejemplo, en un ciclo DO. A veces la posibilidad de aprovechar este tipo de paralelismo fino puede ser detectada automaticamente por el compilador (ej. Alliant) o por un pre-compilador (como el "autotasking" de CRAY)
2. Un paralelismo de granularidad gruesa: En este caso porciones enteras del programa se ejecutan en paralelo. El disparo de las tareas en paralelo insume mucho tiempo de ejecución y por lo tanto se requiere un alto grado de granularidad para que resulte conveniente. Para este tipo de paralelismo se requiere una arquitectura de tipo MIMD.

MODELOS DE PROGRAMACION PARALELA

Tal como ha sido mencionado no existe una única arquitectura de computadoras paralelas. Por el contrario, hay una diversidad grande de ellas y la manera de programarlas presenta aproximadamente la misma diversidad de opciones. En términos prácticos resulta útil referirse a dos o tres grandes clases de arquitectura, para las cuales pueden establecerse modelos adecuados de programación. Un tipo de computadoras puede ser resumido como de *memoria compartida*. Aquí se incluyen las MIMD con pocos procesadores y de acceso a una memoria común tales como las CRAY, la Alliant, IBM 3090, Convex, Silicon Graphics , etc. Un segundo grupo podría denominarse genericamente las *hipercubo*. Representante típica es la Intel iPSC 860, o la NCUBE. Poseen procesadores de memoria local que funcionan como *nodos* de cálculo, y un procesador ("host " o "front end") a través del cual el usuario se comunica con la computadora. El tercer tipo de arquitectura podría ser el de las denominadas computadoras *masivamente paralelas* cuya representante típica es la Connection Machine CM-2, y donde trabajan miles de procesadores. En este caso también hay un procesador "front end" por el cual el usuario se comunica con el sistema.

Con respecto a los modelos de programación puede hacerse referencia a dos tipos principales: *memoria compartida* e *intercambio de mensajes*.

El modelo de *memoria compartida*, evidentemente orientado al primer grupo de procesadores descriptos anteriormente, se basa en descomponer una tarea que encuentra el programa entre varios procesadores. El flujo del programa es llevado a cabo por un único procesador. Cuando hay una tarea que requiere cálculo paralelo, se convoca a otros procesadores para emprender conjuntamente esa porcion del cálculo. Una vez cumplida la tarea en paralelo, un único procesador continua con la parte secuencial del programa. El caso típico es la descomposición de los ciclos indexados de un programa (los ciclos DO de FORTRAN, por ejemplo). Esta es la tarea paralela básica en este modelo de programación. Si hay que hacer operaciones sobre un vector con índice que varia entre 1 y 100, por ejemplo, y se dispone de 4 procesadores, el primer procesador realizará las operaciones sobre los índices 1,5,9,... ; el segundo procesador lo hará sobre los índices 2,6,10, ...; etc.

Este primer modelo se apoya en dos capacidades:

- la existencia de zonas de memoria compartida. Todos los procesadores tiene acceso a cualquier parte de la memoria. En particular se declaran variables *privadas* que pueden ser accedidas sólo por un procesador, y variables *compartidas* por todos los procesadores.
- la sincronización entre procesadores. Hay maneras de iniciar y terminar la tarea de un procesador desde el programa, y permitir que sus tareas parciales se realicen en forma sincronizada.

El segundo modelo de programación podría denominarse de *intercambio de mensajes*. Es el utilizado en las computadoras de tipo *hipercubo* y en las *masivas*. En este caso en cada procesador hay una copia del mismo programa y cada uno comienza a ejecutarlo. En determinadas instancias del programa es necesario intercambiar la información. Esto se hace por envíos de mensajes de un procesador a otro. La característica sobresaliente es en este caso la *comunicación* y existen funciones que permiten, entre otras cosas, enviar (recibir) un mensaje a (de) otro procesador. Este sistema de mensajería adquiere aspectos más restrictivos

* *No hay que confundir el término granularidad utilizado aquí para las tareas paralelas, con el mismo término utilizado en la sección anterior para la arquitectura del hardware*

en el caso de las computadoras *masivas* ya que a fin de reducir los tiempos invertidos en comunicación es importante que se realice una "proyección" del problema físico sobre la red de procesadores.

Este modelo de programación de intercambio de mensajes puede utilizarse también sobre un sistema de computadoras interconectadas entre si. Una red de grandes computadoras, de estaciones de trabajo o aún de PCs tiene las mismas características de las computadoras paralelas de memoria local. Hay sistemas de comunicación (PVM, EXPRESS, etc.) que permiten utilizarlas de igual manera que una computadora de tipo hipercubo, por ejemplo.

LENGUAJES DE PROGRAMACION

Los lenguajes para programar en computadoras paralelas presentan ciertas especificidades. Hay formas de trabajo asociadas a los modelos de programación. Otras, como las extensiones de lenguajes secuenciales, son específicas de cada familia comercial de computadoras. En general el desarrollo de lenguajes para computadoras paralelas ha seguido una de las siguientes líneas:

• La creación de un nuevo lenguaje. Esto permitiría explotar completamente las posibilidades de cada máquina paralela. Ejemplos pueden ser: ADA, introducido por el Departamento de Defensa de los Estados Unidos; o el lenguaje OCCAM, desarrollado para los transputers. A pesar de las ventajas técnicas de este procedimiento, esto implica severos cambios en la forma de trabajo de los programadores e introduce problemas de compatibilidad con los códigos existentes. Este hecho, por otra parte, fundamenta la gran inercia que presentan los programas computacionales tradicionales y explica la supervivencia de uno de los pioneros: FORTRAN.

• La modificación de lenguajes existentes para manejar herramientas de paralelismo. Entre los lenguajes secuenciales más utilizados en ingeniería estructural se cuentan FORTRAN y C. Los distintos fabricantes han desarrollado extensiones del lenguaje original que permite manejar paralelismo de granularidad fina (ciclos DO de FORTRAN), o incluso paralelismo de granularidad gruesa. Estas extensiones son en realidad directivas al compilador, que facilitan el trabajo del usuario. De otra manera éste debería utilizar las funciones de bajo nivel de cada máquina. También se han desarrollado "pre-lenguajes" que permiten la fácil construcción de tareas paralelas y que tienen como característica muy interesante su portabilidad. Algunos ejemplos de estos son: FORCE [4], PCF [5], y EPF [6].

• La detección automática de paralelismo por el compilador. En este caso se mantiene inalterado el lenguaje y se habilita al compilador para detectar automáticamente el paralelismo y traducir las instrucciones empleando las funciones de bajo nivel. También en este caso la mayoría de los desarrollos ha sido realizada en FORTRAN. Esto puede manejar solamente paralelismo de granularidad fina. El caso de los ciclos DO de FORTRAN es el ejemplo típico y el que aparece en todas las computadoras de memoria compartida. La detección automática de paralelismo (denominada "autotasking" en la jerga de las computadoras CRAY) es, logicamente, la más simple para el usuario, sin embargo es limitada. Un programa eficiente debería contar con paralelismo introducido explícitamente por el programador.

En el caso de computadoras de memoria distribuida, por ejemplo de tipo hipercubo, se requieren dos programas diferentes: uno en el procesador "host" y otro en cada uno de los nodos de cálculo. Existen funciones y subrutinas que permiten realizar comunicación y sincronización de tareas. El programa "host" realiza en primer lugar la carga de los ejecutables en cada nodo y dispara su ejecución. Envía y recibe mensajes a los nodos de cálculo y finalmente puede detener la ejecución de éstos.

El programa en cada nodo, a su vez, utiliza subrutinas para enviar y recibir mensajes, básicamente, y realiza los cálculos en paralelo.

De manera totalmente similar a estas subrutinas de las hipercubos funcionan los sistemas de comunicación, como el PVM [7], que permiten trabajar sobre redes de computadoras.

PARALELISMO EN PROGRAMAS NO LINEALES DE ELEMENTOS FINITOS

El Cuadro I muestra el esquema de un programa computacional típico de elementos finitos para resolver problemas transitorios nolineales.

Cuando se va a escribir un programa con cálculo en paralelo evidentemente resulta de interés comenzar atacando las partes que insumen mayor tiempo de procesamiento. En el esquema del Cuadro I aquellas son

Cuadro I: Esquema de un programa no lineal de elementos finitos

1. Lectura de los datos.
2. Inicialización
3. Ciclo sobre los pasos en el tiempo (n←n+1)
   - 3.1 Predicción de los desplazamientos
   - 3.2 Ciclo de iteraciones (i←i+1)
     - 3.2.1 Evaluación de fuerzas residuales
     - 3.2.2 Cálculo del error y verificación de convergencia:
       si converge va a 3.3
       si no, va a 3.2.3
     - 3.2.3 Cálculo de las matrices del sistema
       - 3.2.3.1 Cálculo de matrices elementales
       - 3.2.3.2 Ensamble
     - 3.2.4 Solución del problema linealizado
     - 3.2.5 Corrección de los desplazamientos
     - 3.2.6 Cálculo de tensiones y deformaciones
     - 3.2.7 Va a la próxima iteración (3.2)
   - 3.3 Actualiza las variables
   - 3.4 Impresión y almacenamiento de los resultados
   - 3.5 Termina el análisis ? Si: va a 4. ; NO: va a 3.
4. Impresion de mensajes finales y terminación del programa.

la solución del sistema lineal de ecuaciones (3.2.4) y, dependiendo del algoritmo de solución, la evaluación de las tensiones en cada elemento (3.2.6) y la actualización de matrices y/o vectores elementales (3.2.3).

Con respecto a la viabilidad de ejecutarlas en paralelo, las tareas del algoritmo pueden ser agrupadas en:

a) tareas naturalmente paralelas: como lo son los cálculos de matrices o tensiones a nivel de elemento (3.2.3.1 y 3.2.6) o la actualización de vectores globales (3.1, 3.2.5, 3.3) donde no hay acoplamiento entre sus elementos. Estas tareas son fácilmente paralelizables. La manera más simple de hacerlo es asignar un elemento o grado de libertad a cada procesador.

b) tareas poco acopladas*: como las operaciones de ensamble (3.2.3.2) o de cálculo de parámetros globales de error (3.2.2). Aquí debe evitarse que dos procesadores intenten escribir simultáneamente sobre la misma posición de memoria. Para sincronizar esta actualización de variables globales puede recurrirse a técnicas de reordenado de variables (por ejemplo el "coloreado" [8]).

c) tareas muy acopladas*: la solución del sistema de ecuaciones. Es la de mayor complejidad con respecto a su paralelización debido al acoplamiento entre las variables. En este caso puede recurrirse a un paralelismo de tipo *algebraico* que consiste en descomponer entre los procesadores las tareas individuales del algorítmo de solución [9], o a procedimientos de *descomposición de dominio*, donde se asigna a cada procesador un subdominio físico del problema y se trata en particular el problema acoplado de los grados de libertad de interface [10,11,12].

A continuación se hará referencia a los dos modelos de programación mencionados anteriormente para ser aplicados a programas no lineales de elementos finitos. En el análisis se tendrán en cuenta no sólo consideraciones de *eficiencia*, sino también *portabilidad* y posibilidad de aprovechar software secuencial existente y el trabajo de *adaptarlo* al nuevo ámbito de cálculo paralelo.

Modelo de memoria compartida

Este modelo se orienta a computadoras de pocos procesadores y de memoria compartida, como las CRAY, IBM 3090, Alliant FX 8, entre otras.

La herramienta principal de este modelo es la descomposición de los ciclos DO. Esto implica la reorganización de algunos cálculos de modo de poder colocar en evidencia esta descomposición.

La paralelización de tareas naturalmente desacopladas, como los cálculos a nivel de elemento (tareas de tipo *a*), se realiza fácilmente en este modelo (con las salvedades a nivel de programación discutidas más abajo). Las tareas de tipo *b*, tales como el ensamble de vectores globales, pueden resolverse convenientemente reagrupando las variables y tratando concurrentemente grupos que no posean grados de libertad que los acoplen [8]. Las tareas de tipo *c* ya involucran algún algoritmo especial para la solución del sistema de ecuaciones. Este modelo se presta bien para efectuar un paralelismo de tipo algebraico, así como también utilizar un método de descomposición de dominio [9].

Aplicaciones de este modelo a problemas no lineales de respuesta transitoria de estructuras aporticadas muestran que en la ejecución concurrente de tareas del tipo *a* puede obtenerse una alta eficiencia paralela [13]. Esto se logra tanto con el lenguaje FORTRAN de la máquina (en el caso de ref 13 se utilizó una Silicon Graphics 4D/340 con 4 procesadores) y sus directivas paralelas al compilador, como con los macros de Force [4]. La ventaja de este último es que es *portable* y funciona en una gran gama de computadoras de memoria compartida.

El cálculo de fuerzas no lineales a nivel de elemento es una tarea que puede insumir gran parte del tiempo de computación en el caso de respuesta elasto-plástica mencionado, así como en otros tipos de respuesta como es el caso de materiales visco-plásticos, para los que se obtienen resultados similares [15].

Es importante reducir al mínimo posible la parte del programa que será ejecutada secuencialmente. De lo contrario la eficiencia computacional disminuye comparativamente al aumentar el número de procesadores. Este hecho se conoce como la ley de Amdahl e indica que la aceleración de cálculo paralelo ("speedup") está acotada por

$$S_p \leq \frac{1}{f_s}$$

* *"poco" y "muy" son solamente términos usados aquí para expresar la complejidad de su tratamiento en paralelo*

cuando el número de procesadores p tiende a ∞. En esa expresión f_s representa la fracción secuencial o fracción del código que no puede ser ejecutada en paralelo. La aceleración ("speedup") para p procesadores en paralelo está definida como el cociente entre el tiempo computacional de ejecución secuencial y el tiempo de ejecución concurrente con p procesadores. En condiciones ideales S_p debería tender a p. Pero puede verse de la expresión anterior que con una fracción secuencial del 10% S_p está acotada en 10 para $p \rightarrow \infty$, mientras que para $f_s = 50\%$ $S_p \leq 2$ (!).

El modelo de memoria compartida posee las siguientes características:

- Está asociado a una arquitectura de hardware: funciona sobre computadoras de memoria compartida;
- El uso del macro-lenguaje Force lo hace portable, dentro de esta arquitectura de hardware;
- La ejecución de los ciclos DO en paralelo es algorítmicamente simple;

Si se trata de la modificación de un código secuencial para adaptarlo al proceso en paralelo, es necesario realizar algunas consideraciones. La paralelización directa de las operaciones naturalmente desacopladas, como lo son los cálculos a nivel de elemento, si bien conceptualmente resulta muy sencilla ("solamente" procesar un ciclo DO sobre los elementos, en paralelo), puede significar una tarea formidable de modificación, según sea la estructura del programa secuencial de base. Esto se produce en virtud de tener que declarar al comienzo del programa todas las variables intervinientes ya sea como *compartidas* o como *privadas*. Además debe prestarse particular atención al caso en el cual se requiere actualizar el valor de una variable global y prever la necesaria sincronización entre los procesadores. La lectura/escritura de archivos en disco puede requerir, según sea la computadora y el lenguaje utilizados, también sincronización.

Modelo de intercambio de mensajes

Este modelo se utiliza sobre computadoras de memoria local (por ejemplo hipercubos), o bien sobre redes de computadoras interconectadas. A este último soporte físico está orientado en primer lugar el desarrollo iniciado en el INTEC. La descripción que se realizará corresponde al procedimiento en particular que se ha seguido en este trabajo. Es de hacer notar que el modelo de programación por intercambio de mensajes si bien es la elección natural para este tipo de arquitectura, es adaptable a otros tipos como por ejemplo a computadoras de memoria compartida.

Se utilizarán procedimientos de descomposición de dominio, que son técnicas de resolución de sistemas de ecuaciones que permiten a su vez considerar tambien el paralelismo natural y aquel poco acoplado (por ejemplo, ensamble de matrices globales).

Para manejar la comunicación se empleará el paquete de software PVM [7], de libre distribución, y que presenta la interesante característica de su portabilidad sobre un amplia gama de arquitectura ded hardware. Alternativamente se tratará de utilizar en, nuestro caso, una red local de PCs para prueba de los desarrollos en cálculo distribuido. PVM funciona bajo el sistema operativo UNIX. Por este motivo, se está desarrollando un simulador de comunicaciones "host/node" para funcionamiento bajo el sistema operativo DOS [16].

La idea global del procedimiento con este modelo de programación es la siguiente:

- Descomponer el dominio del problema asignando un subdominio (o más) a cada procesador. La idea es trabajar con un bajo número de procesadores;
- Ejecutar separadamente un programa en cada procesador;
- Utilizar un método apropiado para tratar el problema de interface. Hay diversos procedimientos para ello referidos en la literatura. Entre los que pueden resultar de interés para este caso puede mencionarse:
  - Dirichlet-Neumann [14], donde se recurre a soluciones a nivel de los subdominios con condiciones de contorno de Dirichlet o de Neumann, alternadamente;
  - Balancing Domain Decomposition [12], donde se resuelve en cada iteración, además de los problemas en cada subdominio, un problema global con un número reducido de incógnitas, lo cual mejora la convergencia del método;
  - FETI (Finite Element Tearing and Interconnecting) [10], en el cual los grados de libertad de interface se tratan a través de variables duales (multiplicadores de Lagrange) que en un modelo de elementos finitos de desplazamiento tienen el significado de fuerzas de interface.
- Se busca realizar tareas en paralelo con la mayor granularidad posible.

Si se considera la paralelización de códigos secuenciales, este modelo si bien es de mayor complejidad algorítmica que el anterior, puede requerir modificaciones mayores, pero más localizadas. En algunos casos esto puede resultar en un menor trabajo del programador.

Entre las ventajas de un modelo de envío de mensajes puede mencionarse su relativa portabilidad frente a distintas arquitecturas de hardware y frente a lenguajes de programación. También presenta cierta flexibilidad para preservar las características de los programas de base en cada nodo de cálculo (gestión de memoria, algoritmos de solución interna, etc.). Entre los inconvenientes de este modelo puede destacarse una mayor complejidad para equilibrar tareas entre los procesadores. El equilibrado de tareas es un punto importante ya que procesadores momentáneamente ociosos disminuyen la eficiencia paralela.

CONCLUSIONES

Las computadoras paralelas representan un ámbito de cálculo que, a pesar de su ya alto grado de avance, es relativamente nuevo para la ingeniería aplicada en nuestro medio. La forma de trabajo con ellas es diferente a la usual en computadoras secuenciales. Ya la arquitectura del hardware presenta una gran diversidad de opciones. Los modelos de programación, así como los lenguajes y el soporte de software, también exigen una elección particular en cada caso. Por otra parte eficiencia y portabilidad no siguen el mismo camino y es necesario llegar a un nivel de compromiso entre ambas.

En esta presentación se ha tratado de exponer sintéticamente los principales ingredientes que entran en juego. En tal sentido se ha mostrado brevemente la diversidad de arquitectura de las computadoras, se han mencionado los niveles en que se puede realizar el cálculo paralelo y los principales modelos de programación. Se ha hecho breve referencia a los lenguajes para programación paralela. Se ha mostrado la estructura de un programa no lineal de elementos finitos y sus características desde el punto de vista de su paralelización.

Se ha delineado la aplicación de dos modelos de programación paralela: uno para computadoras de memoria compartida y otro para cálculo distribuido en procesadores de memoria local. En esta última arquitectura, se está comenzando a trabajar en el Grupo de Tecnología Mecánica del INTEC.

AGRADECIMIENTOS

Este trabajo se realiza con apoyo del CONICET y de la Universidad Nacional del Litoral.

REFERENCIAS

1. Dongarra, J.J., Duff, I.S., Sorensen, D.C. and van der Vorst, H.A., "Solving Linear Systems on Vector and Shared Memory Computers", SIAM, Philadelphia, 1991.
2. Farhat, Ch., "An introduction to parallel scientific computations", *Postgraduate studies in supercomputing*, Universidad de Lieja, Bélgica, 1991.
3. Noor, A.K., "New Computing Systems and their Impact on Structural Analysis and Design", *Supercomputing in Engineering Structures*, Eds: P. Melli, C.A. Brebbia, Springer-Verlag, Heidelberg,, 1989.
4. Jordan, H.F., "The Force on the flex: global parallelism and portability", *ICASE Report No. 86-54*, NASA Langley Research Center, Hampton,VA, August, 1986.
5. Perrot, R.H., "Languages for Supercomputers", *Proc. PACTA'92*, Barcelona, 21-24 sept.1992, pp.29-38, 1992.
6. Brawer, S., "Introduction to Parallel Programming", Academic Press, San Diego, CA, 1989.
7. Beguelin, A., Dongarra, J., Geist, A., Manchek, R. and Sunderam, V., "A User's Guide to PVM: Parallel Virtual Machine", *Report*, ORNL/TM-11826, Oak Ridge National Lab., Oak Ridge, TN, July, 1991.
8. Farhat, Ch. and Crivelli, L., "A General Approach to Nonlinear FE Computations on Shared Memory Multiprocessors", *Computer Methods in Applied Mechanics and Engineering*, Vol. 72, No. 2, pp. 153-172, 1989.
9. Coulon, D., Piret, G. and Géradin, M., "Parallel Design of Linear Equations Solvers for Finite Element Systems", *1st European Conference on Numerical Methods in Engineering*, Brussels, Sept.7-11, 1992.
10. Farhat, Ch., Roux, F.-X., "A Method of Finite Element Tearing and Interconnection and its Parallel Solution Algorithm", Int.J.Numerical Methods in Engineering, Vol.32, pp. 1205-1227, 1991.

11. De Roeck, Y.-H., Le Tallec, P., "Analysis and Test of a Local Domain Decomposition Preconditioner", IV Intl.Symp. on Domain Decomposition Meth. for Partial Differential Equations, R. Glowinsky, Y. Kuznetsov, G. Meurant, J. Périaux and O. Widlund, Eds. SIAM, Philadelphia. 1991.
12. Mandel J., "Balancing Domain Decomposition", Communications in Numerical Methods in Engineering, Vol. 9, pp 233-241, 1993.
13 Sonzogni, V.E., "Parallelization of sequential finite element programs". *Report SA-165*, LTAS. Universidad de Lieja, Bélgica, 1992.
14. Bjørstad, P.E. and Widlund, O.B., "Iterative Methods for the Solution of Elliptic Problems on Regions Partitioned into Substructures", SIAM J.Num.Analysis. Vol. 23. pp. 1097-1120, 1986.
15. Sonzogni, V.E., "Domain decomposition on a nonlinear finite element program", ,, Comunicación en el *5ta. Reunión del Proyecto BRITE-EURAM*, Paris, setiembre. 1993
16. Sonzogni, V.E., "A System to Simulate Message Passing Programming Models under DOS", *Internal Report*, INTEC (in preparation), 1994.