Simulación distribuída de arquitecturas multi-DSP.

A.C. L. Acosta Burllaile[1] , Lic. M. De Andrea[2], Lic. A. Cosentino[3], Ing. A.De Giusti[4]

Laboratorio de Investigación y Desarrollo en Informática[5]
Departamento de Informática - Facultad de Ciencias Exactas
Universidad Nacional de La Plata

Resumen

Se analiza la extensión de un simulador DSP TMS320C3x para poder estudiar el comportamiento de una arquitectura multi-DSP basada en el mismo procesador. Este trabajo forma parte de un proyecto conjunto en el desarrollo de arquitecturas y software de base para procesamiento multi-DSP, especialmente orientado a imágenes del que participan dos Universidades argentinas (UNLP y UNCPBA) y la Universidad de la República (Uruguay).

Se explica el desarrollo de un soporte de comunicaciones entre las múltiples sesiones del simulador mono-DSP, de modo de ejecutar el proceso en forma distribuída sobre una red de microcomputadoras heterogéneas.

Por otra parte fue necesario modificar el ambiente de ejecución del simulador mono-DSP de modo de (mediante u programa residente) interceptar una clase de comunicaciones de Entrada-Salida y vincularlas con mensajes a procesadores remotos donde se están ejecutando sesiones simultáneas de simulación mono-DSP.

Por último, se analiza la ejecución de algoritmos distribuídos sobre el simulador y algunas medidas de performance, en algoritmos clásicos de tratamiento de imágenes.

[1] Becario de entrenamiento CIC. Ayte. Dip. Dpto. de Informática, Facultad de Ciencias Exactas, UNLP.
E-mail lacostab@ada.info.unlp.edu.ar
[2]Becario de Iniciación UNLP. Ayte. Dip., Facultad de Ciencias Exactas, UNLP.
E-mail mdeandrea@ada.info.unlp.edu.ar
[3]Ayte. Dip., Facultad de Ciencias Exactas, UNLP.
E-mail acosenti@ada.info.unlp.edu.ar
[4] Inv. Principal CONICET. Profesor Tit. Ded. Excl., Dpto. de Informática, Facultad de Cs. Exactas, UNLP.
E-mail degiusti@ada.info.unlp.edu.ar
[5] Calle 50 y 115 Primer Piso, (1900) La Plata, Argentina, Teléfono 54-21-227707
E-mail lidi@ada.info.unlp.edu.ar

Introducción:

Continuando con la línea de trabajo en el área de procesamiento paralelo sobre arquitecturas DSP y multi-DSP iniciada el año anterior, habiendo ya estudiado y evaluado el simulador del DSP TMS320C3x y comparado los resultados obtenidos en él con los obtenidos en un transputer, surge la necesidad de contar con en ambiente multi-DSP a fin de poder evaluar el paralelismo a nivel de proceso, comparando la performance obtenida en un multi-DSP con la obtenida en una red de transputers. Teniendo en cuenta que el simulador del DSP TMS320C3x no permite simular un ambiente multi-DSP, se analiza la posibilidad de realizar la extensión de este ambiente para que pueda simular una máquina multi-DSP. Además se efectúa la implementación de esta simulación en un ambiente distribuido y se evalúa dicha implementación utilizando un algoritmo paralelo de reconocimiento de patrones en imágenes digitales.

Arquitecturas para el procesamiento de señales:

DSP: el dsp es un procesador CISC dedicado diseñado específicamente para el procesamiento digital de señales. Las características salientes de este tipo de procesador son: operaciones adaptadas al procesamiento de señales implementadas en hardware, modos de direccionamiento adaptados al procesamiento de señales, gran espacio de direccionamiento, interface multi-procesador, memorias y buses independientes, paralelismo interno. Además, gracias a los dos ports de interface externa, se pueden conectar varios dsps para formar una máquina multi-dsp.

El DSP TMS320C3x :

La arquitectura del dsp TMS320C3x responde a las demandas de sistemas que están basados sobre algoritmos de aritméticas sofisticadas que requieren soluciones tanto desde el punto de vista del hardware como del software. La alta performance de este tipo de arquitectura de 32 bits está lograda a través de las siguientes características: la alta precisión y el amplio rango dinámico de las unidades de punto flotante, memoria on-chip, un alto grado de paralelismo y un controlador de acceso directo a memoria.

El TMS320C3x tiene una arquitectura de cpu basada en registros (ver gráfico adjunto). La cpu consta de los siguientes componentes: multiplicador de enteros/punto flotante, ALU, shifter barrer de 32 bit (usado para desplazamientos de hasta 32 bits en un ciclo simple de máquina), buses internos, registros auxiliares de las unidades aritméticas y el CPU file register que es un conjunto de 28 registros de propósito general que pueden ser usados tanto por el multiplicador como por la ALU. Además el dsp posee una cache de programa, memorias internas de acceso dual, un canal DMA que soporta entrada/salida concurrente y un ciclo de máquina corto. [ACO96]

El TMS320C3X puede ejecutar en paralelo operaciones de multiplicación y operaciones de la ALU en un ciclo simple de máquina, tanto sobre enteros como sobre punto flotante. Además, los buses separados de programa, datos y DMA permiten a los programas realizar en paralelo búsquedas, accesos a datos y accesos a DMA. El tiempo de ciclo simple de 40 ns permite ejecutar operaciones a una performance promedio de hasta 60 millones de instrucciones de punto flotante por segundo (MIPFS) y de 30 millones de instrucciones por segundo (MIPS). [TMS94]

El espacio total de memoria disponible del TMS32C3x es de 16 millones de palabras de 32 bits. Programas, datos y el espacio de entrada/salida están incluídos en esos 16 millones. Cada bloque de RAM y ROM es capaz de soportar dos accesos de CPU en un ciclo simple de máquina.

El ambiente de simulación:

En el laboratorio se cuenta con un simulador del dsp TMS320C3x, el cual permite realizar el debugging del código desarrollado para el dsp, ya sea en lenguaje C o en lenguaje assembler.

Además del simulador se cuenta con un compilador del lenguaje C, el cual nos permite obtener a partir del código fuente escrito en el C particular del DSP (compatible con el ANSI C, pero más restringido), el código que será utilizado por el linker. A su vez se cuenta con un optimizador del compilador C, el cual permite realizar optimización en dos niveles: optimización general y optimización específica de la arquitectura. El código resultante es tomado por el linker, que se encargará de generar el código a ejecutarse en el simulador. [DEA96]

El simulador cuenta con dos ambientes de debugging: un ambiente básico de propósito general y un ambiente de comportamiento. El primero permite seguir la ejecución del programa línea por línea, pudiendo observar el contenido de las variables, de los registros de la cpu y de la memoria en cada punto de la ejecución. El segundo permite recolectar estadísticas acerca de la ejecución del código, como por ejemplo el tiempo de ejecución de una función en particular, o de una porción del programa, la cantidad de veces que es invocada una función, la dirección de memoria de una línea en particular, y otros.

El ambiente de simulación es fácil de usar y de aprender, con una interfaz orientada a ventanas; permite crear ventanas para visualizar variables, estructuras, arreglos, etc. Permite además direccionar posiciones de memoria y posee actualización continua.

El simulador que describimos puede ejecutarse tanto en un ambiente windows, como así también en DOS. Si bien tiene una interface amigable, tiene la desventaja de ser lento.

El ambiente de simulación para DOS no posee el ambiente de debugging de comportamiento descripto anteriormente. [DEB93]

Generación de código

El compilador C acepta código fuente C y produce código fuente assembler TMS320C3x. El compilador es un paquete compuesto por un shell, un optimizador y un listador. El shell permite compilar, ensamblar y linkear los módulos fuentes. El listador permite ver el código assembler generado por cada sentencia C original. El optimizador usa fases de optimización sofisticadas que emplean varias técnicas avanzadas para generar código más eficiente y compacto a partir del fuente C. Las optimizaciones generales pueden ser aplicadas a cualquier código C, existiendo además las optimizaciones específicas de la arquitectura del dsp TMS320C3x. [COM95] Estas optimizaciones específicas de la arquitectura lo que hacen es detectar, en el código fuente original, las secciones que pueden ser ejecutadas en paralelo, y traducir esas secciones a las instrucciones paralelas provistas en el conjunto de instrucciones del procesador (paralelismo interno). El assembler traduce archivos fuente en lenguaje assembler a archivos objeto en lenguaje de máquina COFF.

El archiver permite extraer módulos para modificar las librerías existentes.

El library build utility permite construir nuevas librerías.

El linker combina archivos objeto en un módulo objeto ejecutable.

Extensión del ambiente de simulación del dsp TMS320C3x:

Introducción:

Dado que el simulador descripto anteriormente no permite simular una arquitectura multi-DSP, lo que se propone es tener varias sesiones del simulador ejecutándose simultáneamente en distintas PCs con algún mecanismo que les permita a dichas sesiones comunicarse entre sí a fin de que puedan sincronizarse así como también intercambiar información.

Interface del simulador con el exterior:

La única manera en la cual el simulador permite comunicarse con el exterior para realizar la entrada/salida de datos es a través de archivos. Los mismos, para poder permitir una comunicación elemental, deberían poder compartirse entre las diferentes sesiones que se están ejecutando; pero el simulador abre los archivos que utiliza para escribir en forma exclusiva, con lo cual hay que descartar esta forma comunicación elemental, ya que otra sesión no podría leer dichos archivos.

Se plantea entonces utilizar algunos archivos como canales de comunicación, implementando un programa residente que intercepte la escritura y la lectura sobre estos archivos, realizando en realidad una comunicación e intercambio de datos entre las sesiones que los utilizan como canales.

Surge también la necesidad de implementar una librería en el lenguaje C del dsp TMS320C3x, la cual provee las funciones de comunicación a nivel de programa entre los procesos (sesiones del simulador) que necesitan comunicarse utilizando los canales.

Comunicación entre diferentes sesiones de simulación:

Como se dijo antes, el objetivo es implementar un programa residente que intercepte la escritura y la lectura que realiza el simulador en determinados archivos (que en realidad son utilizados como canales), y que realice la comunicación real entre las distintas sesiones de simulación.

Soporte de comunicaciones:

NetBios:

El programa residente, implementado en C, utiliza como soporte de comunicación las rutinas provistas por NetBIOS, ajustándose al paradigma de pasaje de mensajes sincrónico.

NetBIOS (Sistema Básico de Entrada y Salida de Redes de IBM) es una interfase de comunicación "peer to peer" que provee una conexión virtual entre computadoras en una red. Es un software de interfase para aplicaciones que requieran intercambio de información en una red de computadoras. No existe el concepto de Server o la relación Amo/Esclavo. Netbios brinda una serie de servicios que permiten a las

máquinas conectarse y comunicarse sobre una red LAN. No es necesario un Server, ni archivos compartidos ni espacio de disco. La función principal de Netbios es establecer una conexión virtual entre computadores en una red y pasar datos a través de dicha conexión.

NetBIOS es la más ampliamente implementada y emulada interfase entre programas de aplicación LANs en el mundo de las PCs.

Realizando llamadas a NetBIOS se puede transmitir instantáneamente información a una aplicación en otra máquina de la red.

Las llamadas a NetBIOS son independientes del hardware; los mismos comandos que trabajaron originalmente con la red PC LAN de IBM trabajan sin cambios en Ethernet, Token Ring, etc.

Los comandos de NetBIOS son, además, independientes del software, dado que se pueden ejecutar sobre Netware de Novell, eComs's+, OS/2, etc.

Hay dos formas de transferir datos utilizando NetBios:

- transferencia de datos confiables provista por el soporte se sesión. Si hay pérdida de datos NetBios retorna un código de error. Se pueden enviar hasta 64K de datos en una sesión, y hay comprobación de errores.
- soporte de datagrama. Se utiliza para enviar mensajes cortos (de hasta 512 bytes) y no hay comprobación de errores.

La mayoría de las órdenes de NetBios se pueden ejecutar en modo espera o en modo no espera.

- Modo espera: el programa que llama espera a que se ejecute la orden antes de continuar con la próxima instrucción.
- Modo no espera: el programa sigue con la próxima instrucción antes de finalizar con la orden. Se usan para obtener un rendimiento máximo y cuando se desea que NetBios ejecute varias órdenes, una después de otra.

El programa residente que utiliza NetBios como soporte de comunicación utiliza la transferencia de datos a través de sesiones, debido a que es un modo de comunicación seguro. Además, los comandos de NetBios son utilizados en modo espera, ya que de esta manera se ajusta al modelo de pasaje de mensajes sincrónico.

El programa residente:

Se debe tener un programa residente por cada sesión de simulación.

El simulador trabaja con los archivos a través de los servicios que brinda la interrupción 21. Las funciones que utiliza son las siguientes:

- ***INT 21, 3C:*** crear un archivo utilizando un handler
- ***INT 21, 3D:*** abrir un archivo utilizando un handler
- ***INT 21, 3E:*** cerrar un archivo utilizando un handler
- ***INT 21, EF:*** leer un archivo o dispositivo utilizando un handler
- ***INT 21, 40:*** escribir un archivo o dispositivo utilizando un handler

Para poder interceptar los llamados que el simulador realiza a estas funciones, lo primero que realiza el programa residente es cambiar la rutina de atención de la interrupción 21, de forma tal que si el simulador llama a estas funciones para alguno de los archivos que van a utilizarse como canales, no se ejecute la rutina de atención original sino la implementada por nosotros (es decir, la que realiza la comunicación entre las diferentes sesiones); en otro caso (llamado a estas funciones con archivos que no son canales), se llama a la rutina de atención original. También se deben definir los nombres de los archivos que serán utilizados como canales, de forma tal que el programa pueda identificarlos. Los "canales" son unidireccionales, por lo tanto en los archivos correspondientes se efectuará un sólo tipo de operación: lectura (canal de entrada) o escritura (canal de salida).

Además, se debe establecer la sesión de comunicación entre los programas residentes de las sesiones de simulación que desean comunicarse (utilizando las funciones *call* y *listen* de NetBios).

Luego se implementan las nuevas rutinas de atención de las funciones de la INT21 citadas antes. A saber:

- ***apertura:*** cuando el simulador llama a la función de apertura para un archivo definido como un canal, lo que se hace es obtener y almacenar el handler que el sistema operativo le asignó a dicho archivo al realizar su apertura (la apertura se realiza através de la rutina original) .
- ***creación:*** idem apertura
- ***lectura:*** cuando el simulador llama a esta función para un archivo definido como canal de entrada

(utilizando el handler del archivo), el programa residente espera recibir datos de otra sesión cuyo canal de salida esté asociado con este canal de entrada. La comunicación con la otra sesión se realiza utilizando la función *receive* de NetBios, y los datos recibidos son colocados en el buffer de lectura correspondiente, el cual será leído por el simulador.

- ***escritura:*** cuando el simulador llama a esta función para un archivo definido como canal de salida, el programa residente envía los datos que el simulador intenta escribir en el archivo (utilizando el handler del archivo) a la sesión que tenga asociado su canal de entrada con este canal de salida. La comunicación con la otra sesión se realiza utilizando la función *send* de NetBios; los datos a enviar se obtienen del buffer de escritura correspondiente, en el cual son colocados cuando el simulador intenta escribir utilizando esta operación.
- ***cierre:*** el programa residente finaliza la sesión de comunicación.

Dado que el simulador realiza la lectura y escritura a través de buffers de longitud fija, se debió implementar una librería (en el lenguaje C del dsp TMS320C3x) que provee las funciones de comunicación a nivel de programa entre los procesos (sesiones del simulador) que necesitan comunicarse utilizando los canales. Es decir, si un proceso quiere enviar o recibir un solo dato, se debe rellenar el espacio restante del buffer con información inválida, de forma tal que al escribir en el archivo definido como canal el dato que se intenta enviar, el envío sea realizado en ese instante y no se deba esperar a que se complete el buffer. Una situación análoga se da con la lectura.

Además se debió implementar un protocolo de comunicación en el programa residente, de forma tal que al recibir el buffer a ser transmitido, descarte la información inválida y envíe sólo los datos útiles y la información de control que le permita al receptor determinar cuándo finaliza el envío. Asimismo la función del programa residente que realiza la recepción de los datos, debe poder establecer cuando el envío ha finalizado en base a esa información de control, y en ese momento debe devolver los datos y el control al programa que está efectuando la lectura.

En el lenguaje a nivel de proceso, podemos definir entonces las siguientes operaciones basadas en el paradigma de pasaje de mensajes sincrónico:[AND91]

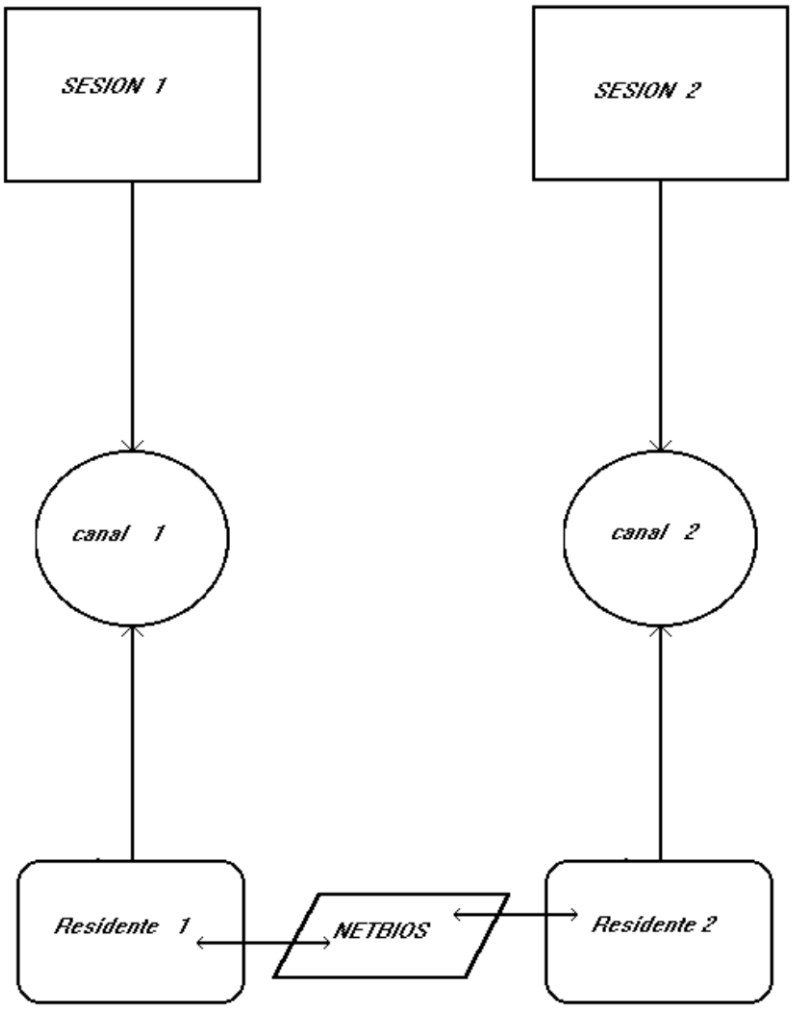
- enviar(cant, buffer): se traduce como la escritura de los datos que están en buffer en un archivo definido como un canal de salida. Para que esta escritura se realice en forma inmediata, aquí se efectúa

el agregado de información inválida a fin de completar el tamaño de buffer requerido por el simulador para efectuar la escritura, además de agregar la información de control correspondiente (para que el programa residente envíe sólo los datos válidos). Al realizarse esta escritura, los datos son interceptados por el programa residente por medio de la función de escritura descripta antes.

- int recibir(buffer): se traduce como la lectura de los datos del archivo definido como un canal de entrada. Esta operación llama a la función lectura descripta antes, la cual se queda bloqueada hasta que se establezca la comunicación con la sesión que debe enviar los datos. Una vez que el programa residente los recibe y retorna el control al simulador, esta operación lee los datos hasta que encuentra la información de control que indica que el mensaje ha terminado y coloca esos datos en buffer.

  Ver gráfico adjunto:

Simulación del multi-dsp



Resultados Obtenidos

Prueba del algoritmo:

Se probó el funcionamiento del sistema utilizando un conjunto de imágenes capturadas por medio de un *scanner* y generadas manualmente utilizando un graficador, retocándolas de modo tal que cumplieran con las hipótesis de que el borde no tuviera ramificaciones y que el cuerpo de la figura no tuviera huecos. Luego de ajustar apropiadamente las cotas del error admisibles para cada atributo de cada figura, se consiguió que el sistema reconociera cada una de las imágenes presentadas de la misma manera que lo hace la percepción humana.[ACO95]

El algoritmo paralelo:

Para poder probar la performance de las arquitecturas paralelas se tendría que disponer físicamente de las mismas. Además, para poder probar la escalabilidad de las diferentes arquitecturas tendríamos que contar con un alto número de arquitecturas idénticas (procesadores). Esto está limitado por el costo económico de las mismas.

En el LIDI se probó el algoritmo paralelo sobre una red con 2 placas transputer interconectadas sobre un bus de PC. Los resultados obtenidos utilizando el mismo conjunto de imágenes con el que se probó el algoritmo secuencial fueron, en promedio, de 650 milisegundos, resultando un speed-up promedio de 1,49.[DEA96]

Posteriormente se realizó la extensión del ambiente de simulación del dsp para que pudiera simular un multi-dsp. Una vez hecha esto, se probó el algoritmo paralelo sobre el simulador extendido con dos sesiones de simulación. Los resultados obtenidos, en promedio, fueron de 45 milisegundos, resultando un speed-up promedio de 1,77.

Conclusiones

Se ha analizado la evaluación del "speed-up" al paralelizar el algoritmo reconocedor, utilizando DSP tipo TM320C, comparando la solución equivalente sobre transputers.

Se han explicado las características del ambiente de simulación DSP disponible en el LIDI, mostrando los resultados de evaluación de perfomance obtenidos para esta aplicación. Además, se realizó la extensión de este ambiente y se la evaluó utilizando el algoritmo reconocedor paralelo.

Si bien se pudo realizar un procesamiento distribuído con múltiples sesiones del simulador ejecutándose concurrentemente, el tiempo requerido por las sesiones para comunicarse e intercambiar datos está condicionado por la performance que brinda Netbios como soporte de comunicación sobre una red Lan, con lo cual la performance del algoritmo paralelo es seguramente menor a la que se obtendría si dicho algoritmo se ejecutara sobre una máquina multi-dsp real, en la cual el tiempo requerido para la comunicación sería probablemente mucho menor. Además, se podrían aprovechar las ventajas del paralelismo interno del dsp pudiendo, por ejemplo, conectar un dsp con otros dos y realizar operaciones de entrada/salida en paralelo entre estos procesadores, obteniendo en tal caso mejores resultados.

Bibliografía:

[AND91] - "Concurrent Programming" - Gregory R. Andrews

The Benjamin/Cummings Publishing Company Inc. 1991

[INM90] - "Transputer Technical Specifications" - Inmos

CSA - Computer System Architects. 1990

[JAI89] - "Fundamentals of Digital Image Processing" - Anil K. Jain

Prentice-Hall. 1989

[MAY88] - "Occam2 language definition" - David May

Inmos. 1988

[HWA93] - "Advanced Computer Architecture: Parallelism, Scalability, Programmability"

Kai Hwang

Mc Graw-Hill 1993.

[BAX94] - "Digital image processing"

Gregory A. Baxes

John Wiley & Sons, Inc. 1994

[RAM96] - "Analysis and Extension of a Simulation Tool for Multi-Dsp parallel processing"

F. Tinetti - H. Ramón - C. Russo - A. De Giusti.

II Congreso Argentino de Ciencias de La Computación 1996 .

[DEB93] - "TMS320C3x C Source Debugger."

Texas Instruments 1993.

[TMS94] - "TMS320C3x User's Guide."

Texas Instruments 1994.

[COM95] - "TMS320C3x Floating Point Dsp Optimizing C Compiler."

Texas Instruments 1995.

[ACO95] - "Algoritmo paralelizable para el reconocimiento de patrones de figuras geométricas"

A. Cosentino - L. Acosta Burllaile - M. De Andrea - A. De Giusti

Second International Congress of Information Engineering 1995.

[DEA96] - "Evaluación de algoritmos de procesamiento paralelo sobre DSP y multi-DSP"

A. Cosentino - L. Acosta Burllaile - M. De Andrea - A. De Giusti

II Congreso Argentino de Ciencias de La Computación 1996 .