Informe DLX

Hola, somos Alberto Lornzo Castro y Raúl Melgosa Salvador. En este documento vamos a desarrollar las mejoras introducidas en nuestra práctica con el objetivo de reducir el número de ciclos del procesador utilizados para realizar el cálculo al mínimo posible y establecer una comparación de rendimiento entre la versión optimizada y la versión no optimizada del cálculo a realizar.

De esta manera, el documento va a estar dividido en dos partes. En la primera parte del documento se va a comentar todas las mejoras introducidas para optimizar el cálculo y en la segunda y en la segunda parte se va a hacer una comparación de rendimiento entre la versión optimizada y la versión no optimizada del cálculo para distintos tamaños de lista.

Mejoras introducidas

En este apartado vamos a comentar las técnicas y mejoras introducidas para obtener la versión del cálculo con la mayor optimización posible partiendo del desarrollo del cálculo de manera no optimizada.

Mejoras en el cálculo matemático

Lo primero que vamos a hacer para optimizar el cálculo es simplificar al máximo posible el número de operaciones a realizar. Esta parte de la optimización es muy importante ya que reducir las costosas operaciones de división y multiplicación permite ahora una gran cantidad de ciclos

En primer lugar, vamos a desarrollar el factor que multiplica a la matriz de Vandermonde con el objetivo de eliminar costas operaciones de división y de reducir al máximo las cuentas a realizar para calcular este factor. El desarrollo queda de la siguiente manera:

$$\frac{(a_2/a_5)\ +\ (a_4/a_5)}{(a_1/a_5)\ +\ (a_3/a_5)} = \frac{\frac{s_2}{t}/\frac{s_5}{20} + \frac{s_4}{t}/\frac{s_5}{20}}{\frac{s_1}{t}/\frac{s_5}{20} + \frac{s_3}{t}/\frac{s_5}{20}} = \frac{\frac{s_2}{s_5} + \frac{s_4}{s_5}}{\frac{s_1}{s_5} + \frac{s_3}{s_5}}$$

$$= \frac{\frac{s_2 + s_4}{s_5}}{\frac{s_1 + s_3}{s_5}} = \frac{s_2 + s_4}{s_1 + s_3}$$

Con este desarrollo podemos sacar varias conclusiones, las cuales nos van a ayudar enormemente en la optimización del cálculo:

- La primera conclusión consiste en que no es necesario calcular a5 para calcular el factor, por lo que las 4 divisiones entre a5 no son necesarias.
- La segunda conclusión que podemos observar consiste en que no es necesario calcular la media de los elementos de las listas 1, 2, 3 y 4, basta con obtener la suma de todos los elementos de estas listas. De esta manera, podemos calcular el factor antes de tener calculadas las media de las listas 1, 2, 3 y 4.

En segundo lugar, podemos observar que la matriz de Vandermonde, independientemente del resultado de a1, a2, a3 y a4, tiene 4 elementos que valen 1. De esta manera, no será necesario multiplicar el factor obtenido anteriormente por estos 4 elementos de la matriz de Vandermonde, ya que un número multiplicado por 1 da como resultado ese mismo número.

En tercer lugar, se calculará la inversa del tamaño de la palabra a través de una división al principio del programa, lo cual permitirá calcular las medias de las listas a través de un multiplicación cuya etapa de ejecución dura 5 ciclos y no una división cuya etapa de ejecución dura 19 ciclos.

Mejoras en la lógica de control

Estas mejoras tienen como enfoque minimizar el uso de operaciones de control de flujo como las instrucciones condicionales y la gestión de los recorridos de las distintas variables del programa.

En primer lugar, se ha eliminado cualquier tipo de bucle para evitar repetitivas instrucciones de salto condicional. Estas instrucciones son muy caras ya que DLX predice el salto como no tomado y carga la instrucción que precede a la instrucción de salto condicional. En el caso de que se produzca el salto, esta instrucción cargada se tiene que desechar y se tiene que cargar la instrucción correspondiente.

Para evitar los bucles, se ha optado por evaluar el tamaño de la lista al principio del programa y en función del tamaño saltar a un bloque de código u otro a través de una instrucción de salto condicional. De esta manera, se ha conseguido reducir en gran manera el número de instrucciones de salto condicional.

En segundo lugar, se ha decidido usar los registros enteros para almacenar distintos múltiplos de 4. De esta manera, cuando hagamos el recorrido de una lista o una matriz en vez de utilizar una variable índice a la que se le incrementa o decrementa 4 unidades cada vez que se accede a una celda del vector o matriz se utilizará en la propia instrucción de lectura el desplazamiento correspondiente a esa celda dentro del vector o matriz. Esto también ofrece una gran flexibilidad ya que se pueden hacer tantos recorridos como uno quiera sin ningún gasto adicional.

Mejoras en el orden de ejecución de las instrucciones

Aunque pueda parecer poco importante una buena organización de las instrucciones y de los cálculos a realizar puede marcar una diferencia muy importante en el rendimiento.

Por un lado, la idea principal que hemos seguido para optimizar el cálculo es calcular lo antes posible la suma total de los elementos de las listas 1, 2, 3 y 4. Una vez se ha realizado esto, ya tenemos la información necesaria para calcular el factor y empezar a calcular los términos de la matriz M mientras se recorren los elementos de la lista 5. Esto es importante ya que hay que realizar una gran cantidad de multiplicaciones y cuanto antes se empiece a realizarlas mejor.

Por otro lado, el recorrido de las listas 1, 2, 3 y 4 se realizará de 1 en 1. Esto tiene como objetivo poder calcular a1, a1^2 y a1^3 mientras se recorre la lista 2, calcular a2, a2^2 y a2^3 mientras se recorre la lista 3 y calcular a3, a3^2 y a3^3 mientras se recorre la lista 3. Por tanto, este recorrido 1 a 1 permite adelantar parte de las multiplicaciones, para que así no se concentren todas al final y así poder intercalar el mayor número de instrucciones entre las multiplicaciones. Durante la ejecución de estas multiplicaciones en la parte final, se irán calculando y almacenando en memoria los distintos resultados buscando la mayor paralelización posible.

Hay que tener en cuenta que esto es posible gracias al uso de los registros enteros como desplazamiento y haber eliminado los gastos asociados al índice que recorre la lista. De no ser así, recorrer las listas 1 a 1 sería poco eficiente.

Mejoras en la reducción de paradas

Una vez se han introducido las mejoras anteriores, el último paso es intentar reducir lo máximo posible la parada que introduce el procesador debido a diferentes tipos de conflictos (estructurales o de dependencia de datos).

En primer lugar, para reducir los conflictos estructurales se va a buscar en todo momento alternar instrucciones de lectura cuya etapa de ejecución dura 1 ciclo y se ejecuta en la unidad intEX con instrucciones de suma en punto flotante cuya etapa de ejecución dura 2 ciclos y se ejecuta en la unidad faddEX. Esto es posible gracias al disponer de varios entornos de ejecución para distintas operaciones.

En segundo lugar, se va a intentar intercalar 3 instrucciones de suma en punto flotante y 2 instrucciones de lectura entre dos instrucciones de multiplicación cuya etapa de ejecución dura 5 ciclos y se ejecuta en la unidad fmulEX. De esta manera, conseguimos que ningún tipo de instrucción se atasque, haciendo para las demás. Eso solo va a ser posible en las primeras fases del programa (antes de calcular el factor) ya que más adelante no va a haber las instrucciones necesarias para poder intercalar sin producir pérdida.

Para garantizar la alternancia de operaciones de suma y lectura y la intercalación de 3 instrucciones de suma en punto flotante y 2 instrucciones de lectura entre dos instrucciones de multiplicación, se va a adelantar la lectura de algunos valores de la lista. El objetivo es realizar estas lecturas antes de realizar la primera instrucción de suma y así conseguimos no romper la alternancia suma-lectura.

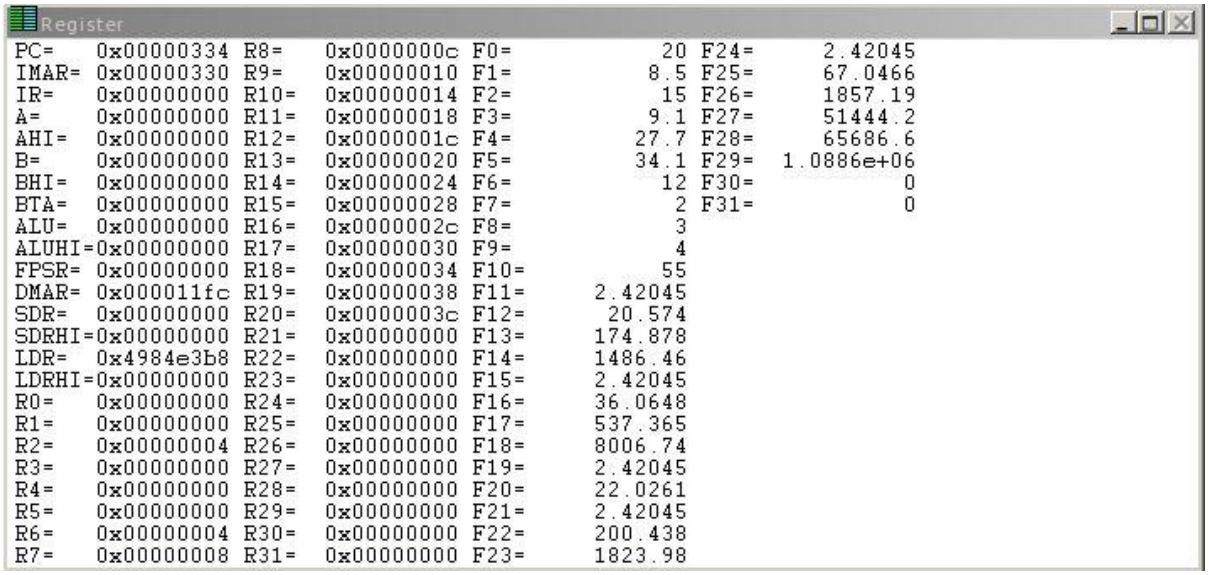
Todo esto tiene como efecto colateral que la lectura y suma de los elementos de una lista no se va a realizar estrictamente en orden pero esto no es relevante en ningún momento ya

que el objetivo es calcular la suma de todos los elementos de una lista mientras se calcula la media de la lista anterior.

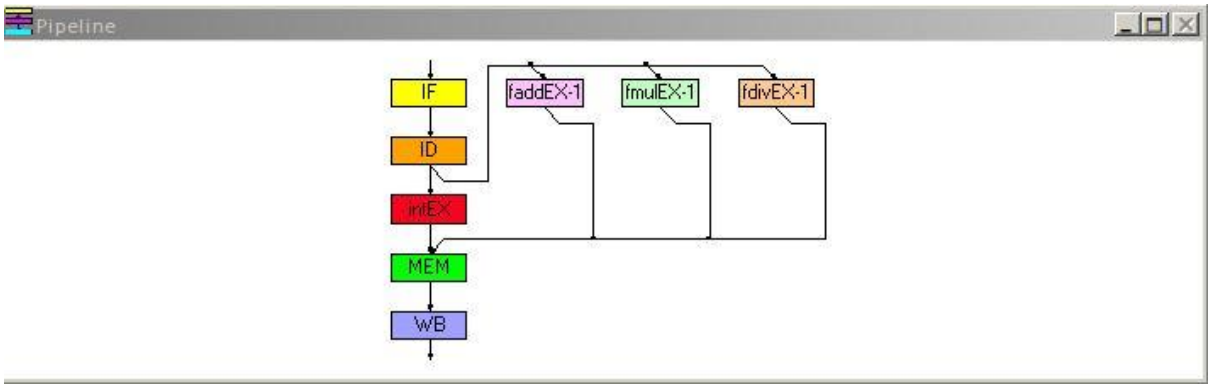
COMPARACIÓN DE DATOS

En este apartado vamos a comparar los valores de salida de la versión no optimizada con los de la versión optimizada, así como los ciclos e instrucciones realizadas en ambas versiones. Se realizarán comentarios de los datos, todo sobre tamaño 20.

Versión no optimizada:



Valor de los registros de la versión no optimizada

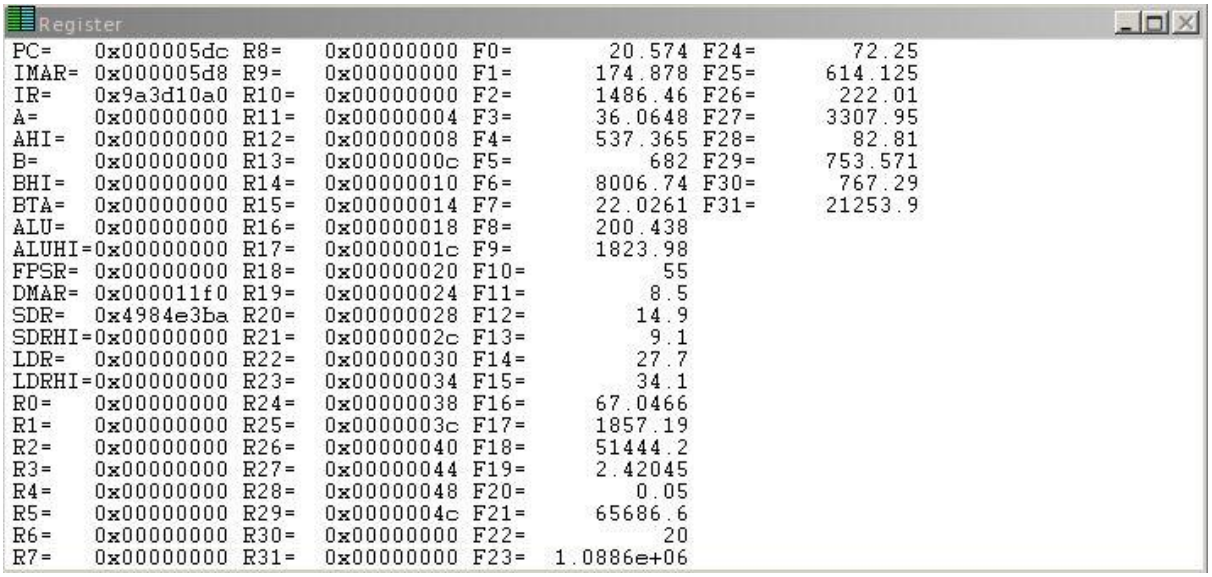


Los registros F1,F2,F3,F4 y F5 contienen los valores a1,a2,a3,a4 y a5 respectivamente.
Los registros de F11 a F27 tienen los valores de cada celda de la matriz M
F28 contiene CheckM
F29 contiene CheckA
El resto de registros son usados para operaciones, por ejemplo F24 contiene el valor del factor de la derecha de la operación.

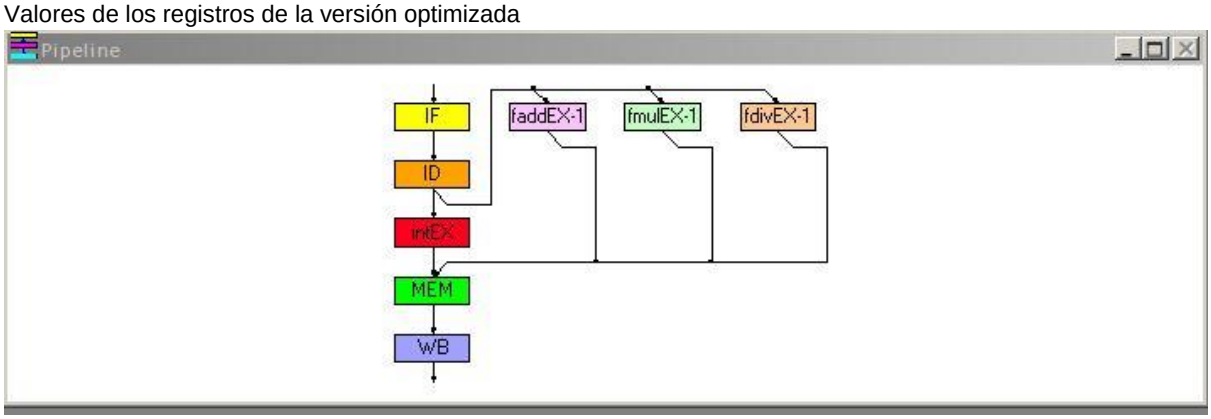
ESTADÍSTICAS VERSIÓN NO OPTIMIZADA	
Total	
Nº de ciclos	754
Nº de instrucciones ejecutadas	368
Stalls	
RAW stalls	156 (20,70%) de todos los ciclos
LD stalls	1 (0,64%) de RAW stalls
Branch/Jump stalls	1 (0,64%) de RAW stalls
Floating point stalls	154 (100%) de RAW stalls
WAW stalls	0 (0,00%) de todos los ciclos
Structural stalls	206 (27,32%) de todos los ciclos
Control stalls	19 (2,52%) de todos los ciclos
Trap stalls	3 (0,40%) de todos los ciclos
Total	384 (50,93%) de todos los ciclos
Conditional branches	
Total	23 (6,25%) de todas las instrucciones
Tomados	19(82.61%) de las Cond.Branches
No tomados	4(17,39%) de las Cond.Branches
Instrucciones Load/Store	
Total	147 (39,94%) de todas las instrucciones
Loads	124(84,35%) de los loads/stores
Stores	23(15,65%) de los loads/stores
Instrucciones de punto flotante	
Total	156 (42,39%) de todas las instrucciones
Sumas	117(75%) de todas las operaciones
Multiplicaciones	29(18,60%) de todas las operaciones
Divisiones	10(6,41%) de todas las operaciones

Traps	
Traps	1 (0,27%) de todas las instrucciones

Versión optimizada



Valores de los registros de la versión optimizada



El registro F21 contiene CheckM
El registro F23 contiene CheckA

Los registros de F11 a F15 contienen a1,a2,a3,a4 y a5 respectivamente.
En el registro F19 se encuentra el cálculo del valor del factor de la derecha de la operación

ESTADÍSTICAS VERSIÓN NO OPTIMIZADA	
Total	
Nº de ciclos	315
Nº de instrucciones ejecutadas	300
Stalls	
RAW stalls	4 (1,27%) de todos los ciclos
LD stalls	0 (0,00%) de RAW stalls
Branch/Jump stalls	0 (0,00%) de RAW stalls
Floating point stalls	4 (100%) de RAW stalls
WAW stalls	1 (0,32%) de todos los ciclos
Structural stalls	0 (0,00%) de todos los ciclos
Control stalls	1 (0,32%) de todos los ciclos
Trap stalls	4 (1,27%) de todos los ciclos
Total	10 (3,18%) de todos los ciclos
Conditional branches	
Total	3 (1,00%) de todas las instrucciones
Tomados	1 (33,33%) de las Cond.Branches
No tomados	2 (66,67%) de las Cond.Branches
Instrucciones Load/Store	
Total	126 (42,00%) de todas las instrucciones
Loads	103 (81,75%) de los loads/stores
Stores	23(18,25%) de los loads/stores
Instrucciones de punto flotante	
Total	146 (48,83%) de todas las instrucciones
Sumas	114 (78,08%) de todas las operaciones
Multiplicaciones	29 (19,86%) de todas las operaciones
Divisiones	3 (2,05%) de todas las operaciones

Traps	
Traps	1 (0,33%) de todas las instrucciones

Comentarios:

Como podemos observar y es lógico, la versión no optimizada realiza mucho más ciclos que la versión optimizada, de 754 pasamos a 315 que es menos de la mitad de ciclos, por lo tanto el rendimiento es mucho mejor. Si lo comparamos con las instrucciones de cada versión nos queda que en la primera tenemos 754 ciclos en 369 instrucciones y en la segunda 314 ciclos en 300 instrucciones. Las instrucciones se reducen en 62, cosa que reduce los ciclos, aunque no tenemos la mitad de instrucciones como sucedía con los ciclos totales, esto se debe también a:

-La reducción muy destacada de las paradas. De los riesgos de dependencia de datos pasamos de 154 RAW a 4, se ve mejor en el porcentaje, donde pasamos de un 20% de los ciclos a apenas un 1%. Por eso se evita usar el mismo registro dos veces consecutivas. Las 10 paradas totales de la versión optimizada no se pueden eliminar debido a las condiciones del problema. Algunas paradas son forzadas para evitar más stalls.

-Reducción de los saltos condicionales, de 20 pasamos a sólo 3 que es un 1% de las instrucciones totales. Estos riesgos reducen el rendimiento durante los ciclos, ya que puede provocar que se detengan las siguientes instrucciones también.

-Se reducen el número de instrucciones de guardar y cargar debido a la reducción de instrucciones, pero éstas sólo consumen un ciclo, así que no hay mucho que comentar aquí.

-Reducción de instrucciones de operación, aunque no en gran medida, podemos observar los porcentajes, donde las sumas tienen más peso que multiplicaciones ya que consumen muchos menos ciclos. En la versión optimizada el porcentaje de sumas es mayor que en la no optimizada. Las divisiones en la versión optimizada solo representan el 2% de las operaciones.

Todas estas mejoras en la implementación de la operación ayudan a que el rendimiento sea mayor, por lo que el número de ciclos se reduce a 315.