Diseño de un lenguaje de Programación

Andrés Baamonde Lozano (andres.baamonde@udc.es)
Rodrigo Arias Mallo (rodrigo.arias@udc.es)

11 de diciembre de 2014

Índice

1. Introducción

Este lenguaje, que está enfocado a las operaciones matemáticas pretende utilizar al máximo la capacidad de la GPU que tenga el ordenador que lo ejecute para así liberar de cálculos a la CPU y así hacer una ejecución más eficiente. A grandes rasgos, vamos a hacer un lenguaje como C pero añadiendole funcionalidades como multiplicación de matrices.

2. Paradigmas

Programación imperativa utilizando funciones.

3. Gestión de memoria

Manual, se especifican tamaños o se inicializan las variables con un valor determinado.A la hora de liberar el espacio de las variables se operará

4. Sistema de tipos

Tipado estático, deben especificarse todos los tipos de las variables. En las matrices se deben especificar sus dimensiones y su tipo, que sólo puede ser uno no se permiten matrices heterogeneas.

5. Tipos de dato

5.1. Básicos

- Int.
- Double.
- Float.
- Char.
- Punteros a tipos básicos(como en C).

5.2. Complejos

- Vectores.
- Matrices.
- Números imaginarios.

5.3. Modificadores de Tipos

Existirán unos modificadores que permitirán compartir un determinado dato con la GPU para así, en caso de que sea frecuentemente usado, evitar el tiempo de enviarlo reiteradamente.

6. Operadores

6.1. Tabla de operadores

Para utilizar la GPU al máximo, lo más adecuado sería utilizar una tabla de operadores como la de openCL. A mayores se incorporarán las funciones típicas de imágenes, como convoluciones y filtros. En estos últimos se podrán aprovechar al máximo las propiedades de las gráficas como el acceso a la memoria de los núcleos vecinos.

add	+
substract	-
multiply	*
divide	/
remainder	%
unary plus	+
unary minus	-
post and pre increment	++
post and pre decrement	–
relational greater than	>
relational less than	<
relational greater-than or equal-to	>=
relational less-than or equal-to	<=
equal	==
not equal	!=
bitwise and	&
bitwise or	—
bitwise not	ˆ
bitwise not	˜
logical and	&&
logical or	——
logical exclusive or	ˆˆ
logical unary not	!
ternary selection	?:
right shift	>>
left shift	<<
size of	sizeof
comma	,
dereference	*
address-of	&
assignment	=

6.2. Sobrecarga operadores

6.2.1. Introducción

Se añaden (sobrecargando los operadores) las operaciones con vectores y matrices. El compilador podrá detectar posibles errores gracias al tipado estático.

6.2.2. Operadores vectoriales

Suma, multiplicación y resta. Estos operadores funcionarían con los tipos vector y matriz, pero también para su multiplicación por un escalar.

7. Visibilidad

7.1. Visibilidad Local

A nivel de función, la variable es accesible y puede manipulada sólo en la función que es declarada.

7.2. Visibilidad Global

A nivel de programa,la variable es accesible y puede ser manipulada en todo el programa.

8. Tratamiento de errores

8.1. Matemáticos

Se lanza una excepción que interrumpe el programa automáticamente (Lo hace el compilador). Errores :

- División entre 0.
- Raíz de un número negativo(sin ser el destino un número imaginario).

8.2. Overflow

Se lanza una excepción que interrumpe el programa automáticamente (Lo hace el compilador). Errores :

- algo

8.3. Segmentación

Se lanza una excepción que interrumpe el programa automáticamente (Lo hace el compilador). Errores :

- Desreferenciación punteros NULL.
- Intento de acceder a memoria que el programa no tiene permisos.
- Intentar acceder a una dirección de memoria inexistente.
- Intentar escribir memoria de sólo lectura.
- Un desbordamiento de búfer.
- Usando punteros no inicializados.

9. Programas o trozos de código

10. Características

10.1. Características

10.2. Inserción de código OpenCL

Además de la programación normal del lenguaje, se permitirá añadir trozos de código directamente en OpenCL, que nos permiten manejar la GPU manualmente y así optimizar más nuestro código. También se incluyen operadores para la sincronización entre procesos y bloqueos para la posible concurrencia a una misma variable.Tarea que el planificador en las operaciones implementadas hará automáticamente.

10.3. Optimizaciones en cálculos

Añadimos el paralelismo de la GPU para agilizar los cálculos de matrices y vectores.

11. Necesidades especiales

11.1. IDE

11.2. Puntos de interrupción

Se puede compilar de forma secuencial en la CPU para así poder depurar sin problemas y después poder optimizar el cálculo haciéndolo de forma paralela.

11.3. Gráfica

Necesidad de una targeta gráfica que soporte OpenCL para poder utilizar la ejecución en paralelo.

12. Arquitectura

12.1. Compilador

Se dispondría de una especie de planificador, que detectaría el código paralelo, lo dispondría en forma de grafo de ejecución.Una vez identificadas las partes que pueden ir en paralelo, con esas partes se dividirían en bloques en función al hardware disponible y posteriormente se enviarían al a GPU.
El codigo paralélo es especificado por el programador a través de los operadores específicos(los operadores vectorialess).
En caso de hacer una operación entre tipos diferentes, el compilador transformará el dato en el tipo necesario(si es posible) solo sí el dato de donde se va a almacenar es el correcto.

12.2. Errores

A la hora de un error en la CPU es de fácil gestión, pero en la GPU nos tendremos que asegurar de que la salída del programa es civilizada.