

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	3
2. Paradigmas	3
3. Gestión de memoria	3
4. Sistema de tipos	3
5. Tipos de dato	3
5.1. Básicos	3
5.2. Complejos	3
5.3. Modificadores de Tipos	3
6. Operadores	4
6.1. Tabla de operadores	4
6.2. Sobrecarga operadores	4
6.2.1. Introducción	4
6.2.2. Operadores vectoriales	5
7. Visibilidad	5
7.1. Visibilidad Local	5
7.2. Visibilidad Global	5
8. Tratamiento de errores	5
8.1. Matemáticos	5
8.2. Overflow	5
8.3. Segmentación	5
9. Programas o trozos de código	6
10. Características	6
10.1. Características	6
10.2. Inserción de código OpenCL	6
10.3. Optimizaciones en cálculos	6
11. Necesidades especiales	6
11.1. IDE	6
11.2. Puntos de interrupción	6
11.3. Gráfica	6
12. Arquitectura	6
12.1. Compilador	6
12.2. Errores	7

1. Introducción

En este documento se propone un lenguaje de programación orientado al cálculo numérico. La sintaxis es idéntica a C, con nuevas funcionalidades.

El objetivo consiste en explotar la capacidad de las máquinas modernas de realizar computaciones en paralelo. Tanto de las tarjetas gráficas (GPU) como de los procesadores multinúcleo (CPU). Y emplear esta capacidad para acelerar los procesos de cómputo que sean paralelizables y así hacer una ejecución más eficiente.

En concreto, se introducirán operaciones como el cálculo de productos matriciales, o operaciones vectoriales, realizadas en paralelo implícitamente.

2. Paradigmas

Programación imperativa procedural utilizando funciones y estructurado.

3. Gestión de memoria

Manual, se especifican tamaños o se inicializan las variables con un valor determinado. El uso y la limpieza de la memoria corren a cargo del programador.

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
- void
- Punteros a tipos básicos (como en C).

5.2. Complejos

- vector
- matrix

- complex

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	+
subtract	-
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	^
bitwise not	~
logical and	&&
logical or	
logical exclusive or	^^
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 ser 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. 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.

10.2. Suma de vectores

Ejemplo sin paralelizar:

```
#define N 100

float A[N], B[N], C[N];
int i;

for(i = 0; i < N; i++)
    C[i] = A[i] + B[i];
```

Ejemplo paralelizado:

```
#define N 100

float A[N], B[N], C[N];

C = A + B;
```

Ejemplo de kernel para la suma de vectores en OpenCL:

```
__kernel void vector_add(__global const int *A, __global const int *B,
    __global int *C)
{
    int i = get_global_id(0);

    C[i] = A[i] + B[i];
}
```

10.3. Multiplicación de matrices

Ejemplo sin paralelizar:

```
#define N 100

float A[N][N], B[N][N], C[N][N];
float sum = 0.0;
int i, j, k;

for(i = 0; i < N; i++) {
    for(j = 0; j < N; j++) {
        sum = 0;
        for(k = 0; k < widthB; k++)
            sum += A[i][k] * B[k][j];
        C[i][j] = sum;
    }
}
```

Ejemplo paralelizado:

```
#define N 100

float A[N][N], B[N][N], C[N][N];

C = A * B;
```

Ejemplo de kernel para la multiplicación en OpenCL:

```
__kernel void matrix_mul(__global float* A, __global float* B, __global
    float* C, int widthA, int widthB )
{
    int i = get_global_id(0);
    int j = get_global_id(1);
    float value=0;
    for(int k = 0; k < widthA; k++)
    {
        value = value + A[k + j * widthA] * B[k*widthB + i];
    }
    C[i + widthA * j] = value;
}
```

11. Pruebas de ejecución

A continuación se muestran algunos resultados, al ejecutarse en secuencia, o en paralelo:

Tamano de matriz: 1024x1024
Tamano de grupo: 128x128

Paralelo:

GPU envio:	5.546000E+03 us
GPU comp.:	2.000000E+00 us
GPU recep.:	6.047000E+03 us
GPU total:	1.187700E+04 us

```
GPU total:      11877.000000 us = 11.8 ms

Secuencial:
CPU comp.:      1.192175E+07 us
CPU comp.:      11921751.000000 us = 11.9 s

Aceleracion:    1003.767871
Fallos comp.:   0
```

```
Tamano de matriz: 2048x2048
Tamano de grupo: 512x512
```

```
Paralelo:
GPU envio:      2.188400E+04 us
GPU comp.:      1.000000E+01 us
GPU recep.:     1.793400E+04 us
GPU total:      4.010200E+04 us
GPU total:      40102.000000 us = 40.1 ms
```

```
Secuencial:
CPU comp.:      1.114093E+08 us
CPU comp.:      111409292.000000 us = 111.4 s = 1 min 25 s

Aceleracion:    2778.148023
Fallos comp.:   0
```

Cuanto más grande sea la matriz o el vector, más se aprecia la aceleración en paralelo.

12. Necesidades especiales

12.1. IDE

12.2. 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.

12.3. Gráfica

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

13. Arquitectura

13.1. 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 código paralelo es especificado por el programador a través de los operadores específicos (los operadores vectoriales). En caso de hacer una operación entre tipos diferentes, el compilador transformará el dato en el tipo necesario (si es posible) solo si el dato de donde se va a almacenar es el correcto.

13.2. 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 salida del programa es civilizada.