Introducción a la Computación

Taller Flash Nº 5

Fecha de entrega: Lunes 4 de noviembre de 2019, 13:30 hs.

En este taller compararemos el rendimiento (*performance*) alcanzado por `Python` y por `C++` al ejecutar un algoritmo muy utilizado en las aplicaciones de cálculo numérico, como lo es el producto entre matrices. La medida de performance que utilizaremos es el tiempo de ejecución de dicho algoritmo.

La multiplicación de matrices es un algoritmo muy estudiado en computación, y existen distintas formas de implementarlo. Elegiremos, en este caso, la forma más sencilla que se deriva directamente de la forma en la que se realiza la operación manualmente.

Además del algoritmo elegido, diversos aspectos de cómo está programado pueden tener incidencia en su velocidad de ejecución, tales como el nivel de abstracción que utiliza el lenguaje en el que está implementado (qué tan cerca de las instrucciones de la máquina está dicho lenguaje) o la forma de almacenamiento de los datos en memoria.

Analizaremos las siguientes variantes entre los dos lenguajes:

- Utilizando *listas de listas* en `Python`.
- Utilizando el soporte matricial del módulo `NumPy` para cálculo numérico en `Python` (aquí evaluaremos el algoritmo que provee internamente la implementación de `NumPy`).
- Utilizando un arreglo unidimensional de elementos en `C++` (en esta representación, cada fila de la matriz se encuentra consecutiva en memoria).

1. Python

Utilizaremos como base el archivo `t5.py`, que se encuentra entre los archivos adicionales del taller. El script se encuentra incompleto y, para completarlo, deben resolver los siguientes ejercicios:

1) Implementar `matrizCeros()`, que devuelva una lista de N listas de tamaño N con `0.0` en sus posiciones.

2) Implementar `matrizAleatoria(a)`, que complete la lista de listas `a` con números aleatorios[1]. La función no devuelve nada, sino que debe modificar `a`.

3) Implementar `multMatrices(a, b, res)`, que complete los elementos de `res` con el resultado de multiplicar las matrices representadas por las listas de listas `a` y `b`, que debe cumplir la siguiente especificación:

problema multMatrices($[[a, b, res : \mathbb{R}]]$){
requiere : $esMatriz(a) \land esMatriz(b) \land esMatriz(res) \land dimMultOk(a, b, res)$;
modifica res;
asegura : $esMatriz(res) \land dimMultOk(a, b, res) \land (\forall i, j : \mathbb{Z})(0 \leq i < cantFilas(res) \land 0 \leq j < cantCols(res) \rightarrow multFilaCol(a, i, b, j, res)$;
pred $esMatriz([[x : \mathbb{T}]])\{(\forall i, j : \mathbb{Z})(0 \leq i < j < |x| \rightarrow |x[i]| == |x[j]|) \land cantFilas(x) > 0 \land cantCols(x) > 0\}$;
pred $dimMultOk([[a, b, c : \mathbb{T}]])\{cantCols(a) == cantFilas(b) \land cantFilas(c) == cantFilas(a) \land cantCols(c) = cantCols(b)\}$;
pred $multFilaCol([[a : \mathbb{T}]], f : \mathbb{Z}, [[b : \mathbb{T}]], c : \mathbb{Z}, [[res : \mathbb{T}]])\{res[f][c] == \sum_{i=0}^{cantCols(a)-1} a[f][i] \cdot b[i][c]\}$;
aux $cantFilas([[x : \mathbb{T}]]) : \mathbb{Z} = |x|$;
aux $cantCols([[x : \mathbb{T}]]) : \mathbb{Z} = |x[0]|$;
}

[1] Utilizar `random.random()`

4) Implementar `medirTiempos(fn, *args)`, que calcule cuánto demora en ejecutarse el llamado a `fn(*args)`. Utilizar `time.time()`, del módulo `time`, para medir el valor del tiempo actual. Devolver la diferencia entre los valores de tiempo inicial y final expresada como un número de punto flotante en segundos.

5) Modificar la función `multMatrices(a, b, res)` para que acepte, además de listas de listas, matrices de `NumPy` (y las multiplique).

   Ayuda: para que la función acepte tanto listas de listas como matrices de `NumPy`, debe preguntar utilizando `isinstance(a, np.matrix)` si `a` es una matriz de `NumPy`, y en caso afirmativo modificar `res` con el resultado de `np.matmul(a, b)` (ver nota[2]). Si `isinstance` devolviese `False`, la función debe hacer lo mismo que en su versión original.

6) Implementar `realizarExperimento()`, que realice el siguiente experimento:

   a) Generar con la función `matrizCeros` tres matrices `A`, `B` y `C` de tamaño $N \times N$.

   b) Completar las matrices `A` y `B` con números aleatorios.

   c) Evaluar con `medirTiempos` el tiempo que demora `multMatrices` en multiplicar `A` por `B` y guardar el resultado en `C`.

   d) Imprimir en pantalla la leyenda: `'Tiempo total listas: '`, acompañada del tiempo obtenido.

   e) Generar a partir de las matrices `A` y `B` sus equivalentes en `NumPy` (utilizar `np.matrix`).

   f) Reinicializar `C` como una matriz de `NumPy` con ceros.

   g) Repetir la evaluación de `medirTiempos` para estas nuevas matrices.

   h) Imprimir en pantalla la leyenda: `'Tiempo total numpy: '`, acompañada del tiempo obtenido.

2. C++

Para la versión en `C++` de este taller, en el archivo `t5.cpp` (en el código adicional que se pueden bajar de la página de descargas de la materia) van a encontrar el esqueleto de un programa equivalente al que realizaron en `Python` en el punto anterior.

En este caso, la representación de las matrices será un arreglo unidimensional comprendido por filas consecutivas. El tipo de datos correspondiente es `float *` (puntero a `float`).

En dicha representación, accederemos a cada elemento de la matriz calculando el *offset* con respecto al primer elemento de la matriz. Al encontrarse cada fila consecutiva en memoria, dados `i` (número de fila), `j` (número de columna) y `N` (cantidad de columnas), la forma de acceder al elemento (i, j) de la matriz `a` utilizando arreglos unidimensionales es: `a[i * N + j]`.

El archivo `t5.cpp` incluye además las siguientes funciones ya implementadas, equivalentes a las versiones en `Python` con el mismo nombre:

1) `float *matrizCeros();`

2) `void matrizAleatoria(float *a);`

3)
```
float medirTiempos(void (*fn)(const float *, const float *, float *),
    const float *a, const float *b, float *res);
```

4) `void realizarExperimento();` (en este caso, solamente evaluamos arreglos unidimensionales).

Además, se incluye la siguiente función:

5) `void eliminarMatriz(float *a);`

Esta función libera la memoria pedida por `matrizCeros` para la matriz `a`.

Completar la implementación de la siguiente función para que realice la operación especificada en la versión en Python homónima:

6) `void multMatrices(const float *a, const float *b, float *res);`

[2] Utilizar `np.copyto(destino, expresion)`

3. Comparación entre Python y C++

Resolver los siguientes ejercicios a partir de los códigos escritos en las secciones anteriores:

1) Ejecutar los programas en ambos lenguajes, utilizando matrices de 100×100[3]. Anotar los valores obtenidos en pantalla para las distintas variantes evaluadas.

2) Repetir el punto anterior. Los valores obtenidos, ¿son estables? Si no lo fueran, proponer una modificación que le permita realizar una comparación más adecuada.

3) Agregar `-O3` a las opciones de compilación de la versión en `C++`. Repetir la ejecución de dicho programa y comparar el resultado obtenido contra no agregar dicha opción.

4) Confeccionar un gráfico de barras que agrupe, para distintos tamaños de matriz, los valores agrupados obtenidos al ejecutar `realizarExperimento()` en ambos lenguajes. Incluir en la evaluación de C++ los resultados obtenidos con `-O3` y sin dicha opción. Utilizar los siguientes tamaños:

   a) 100×100

   b) 500×500

   c) 1000×1000

   d) 2000×2000

5) ¿Qué conclusiones puede sacar del gráfico anterior?

Condiciones de entrega:

- Tanto la versión impresa como la digital deben incluir las versiones modificadas finales de los archivos `t5.cpp` y `t5.py` y un informe que responda los ejercicios correspondientes a la sección 3 (Comparación entre `C++` y `Python`).
- Se evaluará la correctitud del código producido, su claridad y legibilidad; y el uso de la herramienta `git` (esto es, no debe subirse sólo la versión final, sino usar el repositorio para subir versiones intermedias, con descripciones de `commit` significativas).
- Asegurarse de que todos/as los/as docentes del curso tengan permisos de lectura (*Reporter*) en el repositorio de Gitlab (ver usuarios en el Campus). Se descargará la última versión de los archivos directamente de ahí, luego de ser informados de que el taller se encuentra listo.
- Para la materia de grado, el mail deberá tener el siguiente *subject*:
  "[Flash 5]: Gonzalez 666/21 (grado)".
- Para posgrado, el mail deberá tener el siguiente *subject*:
  "[Flash 5]: <apellido> <DNI, Pasaporte o Libreta> (posgrado)". Por ejemplo, "[Flash 5]: Trump 34509ABD (posgrado)".
- Se revisará que la forma del *subject* se corresponda **exactamente** con el formato pedido.
- Enviar dicho aviso por correo electrónico a la lista de los docentes de la materia:
  icb-doc@dc.uba.ar.
- Entregar **de forma impresa** el código del taller.

  Importante: Solo se admite la entrega por medio de `Gitlab`. Se buscarán el/los archivos en la carpeta correspondiente (en este caso, dentro de `Taller5`).
  No se aceptarán las entregas de código por mail o solo en forma impresa.

[3] En la versión en `Python`, indicar el valor de N como argumento por línea de comandos. En la versión en C++, puede agregar la opción de compilación `-DN=100` para indicar que las matrices se creen con el tamaño 100×100, y de esta forma realizar las distintas evaluaciones sin necesidad de modificar el código fuente.