Actas de las XXXIX Jornadas de Automática, Badajoz, 5-7 de Septiembre de 2018
https://doi.org/10.17979/spudc.9788497497565.0173

DESARROLLO DE UN EFECTO DE MOSAICO PARA DOCENCIA EN LA MATERIA DE ARQUITECTURAS PARALELAS

Alejandro González García
Departamento de Ingeniería de Sistemas y Automática, Universidad de Vigo, España, aggarcia3@esei.uvigo.es

Matías García Rivera
Departamento de Ingeniería de Sistemas y Automática, Universidad de Vigo, España, mgrivera@uvigo.es

Miguel Díaz-Cacho Medina
Departamento de Ingeniería de Sistemas y Automática, Universidad de Vigo, España, mcacho@uvigo.es

Resumen

En este artículo presentamos el desarrollo de un efecto artístico de mosaico entre dos imágenes usado en la enseñanza de técnicas de paralelismo y visión por computador. La implementación del efecto con el uso de paralelismo a nivel de datos e instrucción, el lenguaje de programación C y la biblioteca OpenCV permite que el alumnado descubra las ventajas y limitaciones de este paradigma en el contexto de una aplicación de procesado de imágenes, que consideramos atractiva para el alumnado, brindándole conocimientos prácticos en ambos campos y contribuyendo a afianzar los teóricos.

Palabras clave: educación, computación paralela, procesamiento vectorial, procesamiento de imagen.

1 INTRODUCCIÓN

El Departamento de Ingeniería de Sistemas y Automática además de su tradicional docencia en control, robótica y autómatas programables, es responsable también de materias de Arquitectura de Computadoras. Este es el caso en la Universidad de Vigo, donde este departamento imparte clases de Arquitectura de Computadores y Arquitecturas Paralelas en la Escuela Superior de Ingeniería Informática de Ourense.

En la materia de Arquitecturas Paralelas [1] los alumnos y alumnas completan su formación relativa a la Arquitectura de Computadores, estudiando los avances que han sido los principales responsables de las mejoras de rendimiento experimentadas a lo largo de las últimas décadas: paralelismo a nivel de instrucción y datos, multiprocesamiento, segmentación del cauce de ejecución, arquitecturas RISC y procesamiento vectorial [5]. Se trata de una materia presente en numerosos planes de estudios de universidades españolas y extranjeras.

Con esta propuesta hemos logrado diversos objetivos relacionados con la materia:

- Repaso de conceptos básicos de programación y algoritmia, como prerrequisitos para implementar este y otros problemas similares.
- Aplicar los conocimientos teóricos adquiridos a un problema real.
- Introducir técnicas de evaluación de rendimiento de un proyecto software.
- Estimular la creatividad y el interés de los estudiantes por las aplicaciones del paralelismo y multiprocesamiento.

El procesado de imágenes es además una técnica ampliamente utilizada en la localización de defectos en materiales y en la identificación de desgastes en diferentes elementos de maquinaria industrial y de transportes. Por ello, las técnicas de paralelismo en el procesado de imágenes sirven también para el desarrollo de complementos para la ayuda al mantenimiento industrial acorde a los nuevos paradigmas de la industria 4.0.

2 DESCRIPCIÓN DEL EFECTO Y DEL ALGORITMO

Uno de los primeros pasos para crear con éxito software es definir sin ambigüedad qué tiene que hacer y cómo. Para ello, hay que descomponer el comportamiento deseado de la aplicación en bloques de instrucciones lo suficientemente detalladas como para implementarlas en un computador; es decir, definir el problema que resuelve (o, equivalentemente, sus requisitos) y esbozar algoritmos que los satisfagan.

Primeramente, el usuario debe de seleccionar dos imágenes ya existentes, en formato RGB y TrueColor (24 bits de información de color por píxel, 8 por cada una de las tres componentes). Una manera de hacerlo es leyendo los argumentos de la línea de comandos. Si

las imágenes son válidas y se pudieron cargar en memoria, el efecto comienza. En caso contrario, la aplicación finaliza inmediatamente, mostrando un mensaje y devolviendo un código de error al sistema operativo.

De las dos imágenes escogidas, se toma la primera como original y la otra segunda como patrón, un ejemplo de cada se puede ver en la Figura 1. Se mostrará por pantalla la imagen original, para poder compararla antes y después de aplicar el efecto. Este efecto consiste en que una vez divididas las dos imágenes en bloques de 16 x 16 pixels, se sustituye cada uno de los bloques de la imagen original por el bloque más parecido de la imagen patrón. Como es factible que el usuario seleccione dos imágenes de diferente tamaño, y por tanto puedan faltar bloques para sustituir, o bien que la resolución no sea múltiplo de 16 en ambos ejes y algún bloque no contenga 16 × 16 píxeles exactamente, hay que armonizar la resolución de ambas imágenes a la menor, asegurando en el proceso que sus dimensiones sean múltiplo de 16. Esto último no es difícil, pues los tamaños estándar de imágenes lo cumplen, 640 x 480, 1024 x 768, 1280 x 1024,…

Existen ya aplicaciones disponibles que realizan este mismo procesado de imagen, aunque en vez de una imagen patrón suelen usar una base de imágenes de tamaño icono [4]. Desconocemos si estos programas aprovechan el paralelismo que ofrecen los procesadores actuales.

Figura 1: Imágenes original y patrón de ejemplo.

Formalizando lo descrito en el anterior párrafo, se define cada bloque como un par ordenado (i, j), donde i representa la coordenada vertical del bloque y j su coordenada horizontal, estando ambas expresadas en píxeles. Un 0 se correspondería con el bloque que empieza en el píxel 0 en esa dirección, un 16 con el que empieza en el píxel 16, y así sucesivamente. De esta forma, toma forma un algoritmo que itera sobre cada bloque (i, j) que compone la imagen original, encontrando otro bloque (k, l) en la imagen patrón tal que su diferencia absoluta de color con (i, j) sea mínima, y copia el bloque (k, l) por encima del bloque (i, j), sustituyéndolo. La idea se plasma en la Figura 2 que viene a continuación, como pseudocódigo.

```
PROCEDIMIENTO efectoMosaico
INICIO
   DESDE i ← 0 HASTA alto PASO 16 HACER
      DESDE j ← 0 HASTA ancho PASO 16 HACER
         k, l ← encontrarBlq(i, j)
         copiarBlq(k, l, i, j)
      FIN_DESDE
   FIN_DESDE
FIN_PROCEDIMIENTO

FUNCIÓN encontrarBlq(Entero: i, Entero: j)
VARIABLES
menorDif : Entero
difActual : Entero
k, l : Par de Entero
INICIO
   menorDif ← ∞
   DESDE u ← 0 HASTA alto PASO 16 HACER
      DESDE v ← 0 HASTA ancho PASO 16 HACER
         difActual ← calcularDif(i, j, u, v)
         SI difActual < menorDif ENTONCES
            menorDif ← difActual
            k, l ← u, v
         FIN_SI
      FIN_DESDE
   FIN_DESDE
   DEVOLVER k, l
FIN_FUNCIÓN
```

Figura 2: Pseudocódigo del algoritmo del efecto.

Con el objetivo de computar la diferencia absoluta de color entre bloques, que en el pseudocódigo anterior sería tarea de la función `calcularDif`, resulta provechoso darse de cuenta de que un píxel en una imagen RGB está formado por tres componentes de color independientes: la roja, la verde y la azul. Posiblemente la forma más sencilla de comparar dos píxeles p_1 y p_2 en términos de similitud de color, con componentes de color r_1, g_1, b_1 y r_2, g_2, b_2, es tomando la suma de los valores absolutos de las diferencias de los componentes de color dos a dos, lo que se expresa matemáticamente en la ecuación 1.

$$D_{p_1,p_2} := |r_1 - r_2| + |g_1 - g_2| + |b_1 - b_2| \quad (1)$$

Bajo esta igualdad, que es la que se usará en la implementación, píxeles iguales tendrán un valor D igual a 0, ocurriendo la diferencia máxima posible entre un píxel totalmente blanco y un píxel totalmente negro, que es igual a 255 × 3 = 765. Siguiendo la misma lógica, se extrapola el resultado final de comparar un bloque B_1 con un bloque B_2 como la suma de los resultados parciales de comparar sus píxeles dos a dos:

$$D_{B_1,B_2} := \sum_{p_1,p_2=0}^{255} D_{p_1,p_2} \quad (2)$$

En la fórmula 2 anterior, p_1 y p_2 representa el número ordinal del par de píxeles dentro de los bloques, que debido a que un bloque contiene 256 píxeles está en el rango de 0 a 255. El primer par de píxeles, de número ordinal 0, se corresponde con los píxeles más arriba y a la izquierda de cada bloque; el segundo, con los píxeles más arriba y un poco más a la derecha de cada bloque, y así hasta terminar.

Combinando adecuadamente todos los algoritmos y definiciones explicadas ya es viable codificar el efecto. El aspecto esperado del efecto se muestra en la Figura 3. Se ha generado una imagen muy parecida a la original, pero sin emplear ningún bloque de esa misma imagen, los bloques empleados son de la imagen patrón. Cabe destacar que hay interés en realizarlo lo más eficientemente posible, en lo que a velocidad de procesamiento se refiere. Las motivaciones e implementación de este objetivo se discuten con detalle en la sección 3.2 del presente artículo.

Figura 3: Resultado del efecto usando como entradas las imágenes de la Figura 1.

3 IMPLEMENTACIÓN

3.1 LENGUAJES DE PROGRAMACIÓN, HERRAMIENTAS DE DESARROLLO Y BIBLIOTECAS

Antes de empezar a programar una aplicación, es importante decidir adecuadamente el lenguaje de programación que se va a usar, pues ello condiciona la elección de entornos de desarrollo y recursos a emplear. Para la implementación de este efecto por el alumnado se usará el lenguaje de programación C, debido a su casi omnipresencia en contextos académicos y profesionales de la Ingeniería Informática; la gran disponibilidad de componentes reutilizables; el amplio número de editores de código, compiladores y computadores soportados; la posibilidad de usar instrucciones SIMD mediante funciones intrínsecas [8]; la destacable predictibilidad algorítmica del rendimiento obtenido por un programa (exceptuando factores concernientes al estado del SO que lo ejecuta y el hardware) y la simplicidad de su sintaxis.

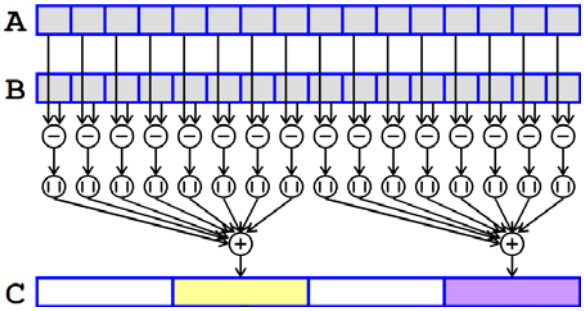
Por otro lado, un lenguaje de programación estaría incompleto sin herramientas de compilación, un editor de código o incluso un IDE. En la Escuela Superior de Ingeniería Informática de Ourense se viene utilizando por convenio el IDE NetBeans de Oracle en su versión 8.2, que soporta la edición, compilación y depuración de código, entre otras funcionalidades. Para compilar el programa en C se usará GCC, un compilador libre, de código abierto y robusto del proyecto GNU comúnmente usado bajo entornos Unix, compatible con NetBeans. No obstante, como la mayoría de ordenadores personales usan sistemas operativos Windows, incluyendo los presentes en los laboratorios de las prácticas, se instalará el conjunto de herramientas Cygwin, que porta GCC y otras utilidades Unix a Windows.

Finalmente, a causa de que C es un lenguaje de aplicación general que no proporciona herramientas especialmente diseñadas para la manipulación y visualización de imágenes, e implementarlas excedería el ámbito de la asignatura, se utilizará la biblioteca OpenCV 3.0.0. De licencia de código abierto BSD, esta biblioteca cuenta con más de 47.000 usuarios en el mundo repartidos entre diferentes disciplinas, como la robótica o el arte interactivo, y soporta diversos tipos de dispositivos y lenguajes de programación.

3.2 USO DE TÉCNICAS DE PARALELISMO. RENDIMIENTO

Recordando la sección 2 de este artículo, uno de los objetivos de implementación es conseguir procesar el efecto en el menor tiempo que permita el hardware. Este objetivo tiene un propósito docente dual: por una parte, introduce el uso de técnicas de medición de rendimiento como método para comparar diferentes soluciones, que el alumnado debe de conocer y saber aplicar. Por otra parte, justifica considerar la aplicación de técnicas de paralelismo a nivel de datos y multiprocesamiento, ya enseñadas en teoría, bajo el marco de la arquitectura x86-64 usada en computadores contemporáneos.

De hecho, este efecto se trata de un buen candidato para ser optimizado con técnicas de paralelismo. Observando cuidadosamente el pseudocódigo de la Figura 2, el lector puede pensar acertadamente que modificando el paso y el valor de inicio de las variables de los bucles `DESDE` del procedimiento principal se logra un reparto lo más equitativo posible de las operaciones de búsqueda y comparación entre diferentes hilos de ejecución, lo que se traduce en un factor de mejora igual al número de núcleos de CPU de los que disponga el sistema, n (el máximo factible según la ley de Amdahl). Una manera simple de llevar a cabo este reparto es por filas de bloques, de forma que cada hilo empiece en la fila h, siendo h un identificador consecutivo del hilo, y avance de n en n filas. Asimismo, la función `calcularDif` se beneficia de la presencia de conjuntos de instrucciones SIMD, que en una sola instrucción operan sobre vectores de datos. Para entender esta última afirmación, hay que destacar que OpenCV almacena en memoria los canales de color entrelazados, siguiendo el orden BGR, BGR,…, BGR (Little-Endian). Por su extensión se usará la tecnología SSE2 de Intel® [2], que ofrece la función `_mm_sad_epu8` [7]. Ésta es idónea para el caso de uso que nos concierne, pues realiza la suma de los valores absolutos de las diferencias de un total de 16 pares de componentes de color a la vez, reduciendo la incidencia de los cuellos de botella de la predicción de saltos, obtención y decodificación de instrucciones y accesos independientes a memoria principal notoriamente. El funcionamiento de la instrucción se visualiza en la Figura 4.



Description

Compute the absolute differences of packed unsigned 8-bit integers in a and b, then horizontally sum each consecutive 8 differences to produce two unsigned 16-bit integers, and pack these unsigned 16-bit integers in the low 16 bits of 64-bit elements in dst.

Operation

```
FOR j := 0 to 15
        i := j*8
        tmp[i+7:i] := ABS(a[i+7:i] - b[i+7:i])
ENDFOR
FOR j := 0 to 1
        i := j*64
        dst[i+15:i] := tmp[i+7:i] + tmp[i+15:i+8] +
                       tmp[i+23:i+16] + tmp[i+31:i+24] +
                       tmp[i+39:i+32] + tmp[i+47:i+40] +
                       tmp[i+55:i+48] + tmp[i+63:i+56]
        dst[i+63:i+16] := 0
ENDFOR
```

Figura 4: Descripción de `_mm_sad_epu8`.

En la Figura 5 puede verse el código de la función que calcula la diferencia entre 2 bloques de pixels. Un par bucles, el primero para recorrer todas las filas de pixels de la imagen, y el segundo para recorrer todas las columnas de pixels de una fila. Seleccionado el pixel, se calcula y se añade la diferencia de sus 3 componentes de color.

En la Figura 6 se presenta el código que calcula la diferencia entre 2 bloques de pixels con SSE2. Ahora el segundo bucle recorre todas las componentes de color de una fila en grupos de 16. Seleccionado un grupo de 16 componentes de color, se calcula la diferencia entre esos 2 grupos de 16 componentes en una única operación `_mm_sad_epu8`.

```
unsigned int calcularDif (IplImage* img1, int img1f, int img1c, IplImage* img2, int
    img2f, int img2c, int alto, int ancho) {

    unsigned int diferencia = 0;

    int fila, columna;

    for (fila = 0; fila < alto; fila++) {

        unsigned char* pImg1 = (unsigned char*) (img1->imageData + (img1f + fila) *
            img1->widthStep + img1c * img1->nChannels);
        unsigned char* pImg2 = (unsigned char*) (img2->imageData + (img2f + fila) *
            img2->widthStep + img2c * img2->nChannels);

        for (columna = 0; columna < ancho; columna++) {

          diferencia += abs(*pImg1++ - *pImg2++);
          diferencia += abs(*pImg1++ - *pImg2++);
          diferencia += abs(*pImg1++ - *pImg2++);


        }
    }
    return (diferencia);
}
```

Figura 5: Función `calcularDif` con SISD.

```
unsigned int calcularDif (IplImage* img1, int img1f, int img1c, IplImage* img2, int
    img2f, int img2c, int alto, int ancho) {

    int diferencia = 0;

    __m128i resultadoProvisional, img1CC16, img2CC16, sadCC16;
    resultadoProvisional = _mm_xor_si128(resultadoProvisional, resultadoProvisional);

    int fila, cc;

    for (fila = 0; fila < alto; fila++) {

        __m128i* pImg1 = (__m128i*) (img1->imageData + (img1f + fila) * img1->
            widthStep + img1c * img1->nChannels);
        __m128i* pImg2 = (__m128i*) (img2->imageData + (img2f + fila) * img2->
            widthStep + img2c * img2->nChannels);

        for (cc = 0; cc < ancho * 3; cc += 16) {

            img1CC16 = *pImg1++;
            img2CC16 = *pImg2++;

            sadCC16 = _mm_sad_epu8(img1CC16, img2CC16);

            resultadoProvisional = _mm_add_epi32(resultadoProvisional, sadCC16);

        }
    }

    int *enteros = (int *) &resultadoProvisional;

    diferencia = enteros[0] + enteros[2];

    return (diferencia);
}
```

Figura 6: Función `calcularDif` con SIMD SSE2.

En la Figura 7, se puede ver el uso de hilos. Cada hilo es el responsable del cálculo de una parte de la imagen mosaico. Con 2 hilos, el primero realizaría los cálculos del mosaico para la mitad superior, y el segundo para la mitad inferior.

```
    pthread_t threads[NTHREADS];

    int i;
    int indice[NTHREADS];
    for (i = 0; i < NTHREADS; i++) {
        indice[i] = i;
        pthread_create(&threads[i], NULL, (void *) &mosaico_thread, (void *) &indice[
            i]);
    }

    for (i = 0; i < NTHREADS; i++) {
        pthread_join(threads[i], NULL);
    }
```

Figura 7: Empleo de hilos

En vista de todo lo anterior, hemos implementado y medido el rendimiento del efecto, bajo diferentes combinaciones de uso de herramientas de paralelismo, con la finalidad de comparar el margen de mejora que proporciona cada una de ellas, tanto en presencia de otras como solas. Dicha implementación de referencia está disponible en GitHub, bajo la licencia de código abierto MIT, en el repositorio accesible desde el URL https://github.com/aggarcia3/EfectoMosaico. Las marcas de tiempo recogidas en un portátil con un procesador Inter Core i7 2670QM figuran en la Tabla 1.

Tabla 1: Comparación de rendimiento del efecto usando diferentes técnicas de paralelismo.

Técnicas	**Tiempo**
SISD, 1 hilo	8,5 s
SISD 4 hilos	3,1 s
SIMD, 1 hilo	1,2 s
SIMD, 4 hilos	0,35 s

Se puede ver en la tabla que la mejora es notable, desde los 8,5 s sin ninguna técnica de paralelismo, hasta los 0,35 s aplicando SIMD SSE2 e hilos.

4 CONCLUSIONES

El software desarrollado puede ser programado sobre cualquier ordenador personal y comprobar las capacidades de paralelismo que ofrece. El alumno puede trabajar en su propio equipo sin necesidad de un hardware costoso.

El alumno puede ver que ha desarrollado una aplicación con un llamativo resultado visual, que es un producto comercial, y que ha reducido de forma notable su tiempo de ejecución, mostrando que todas las mejoras que proporcionan los procesadores actuales a nivel de paralelismos, no son puros desarrollos teóricos difíciles de llevar a la práctica.

5 LÍNEAS FUTURAS

Quedan abiertas incorporar mejoras al programa. La más inmediata es añadir el conjunto de instrucciones AVX2 [3]. De esta forma, se pasará de procesar en paralelo 16 componentes de color en SSE2, a procesar el doble en AVX2.

En procesadores segmentados como los actuales, el desenrollado de bucles puede ser otra forma de mejora [5]. Haciendo el bucle más grande el microprocesador puede planificar mejor las instrucciones para su reordenación y su cauce de ejecución poder estar ocupado durante más tiempo.

Un análisis más detallado de la ejecución del programa empleando un profiler [8], es una buena herramienta para buscar las partes del código a optimizar, detección de cuellos de botella, fallos y aciertos de la memoria caché, etc.

En cuanto a una utilidad más allá de la enseñanza del paralelismo, se está trabajando en la comparación de imágenes de piezas en tiempo real con modelos de desgaste de esas piezas para la identificación de averías en maquinaria y la posibilidad de realización de

un mantenimiento predictivo y automatizado acorde a posibles funcionalidades de la Industria 4.0.

El empleo de otra aplicación como la estabilización de video [6], puede también atraer el interés del alumno.

Agradecimientos

This work has been partially supported by the European Project "Argelian National Laboratory for Maintenance Education". Project 586035-EPP-1-2017-1-DZ-EPPKA2-CBHE-JP.

English summary

DEVELOPMENT OF A MOSAIC EFFECT FOR TEACHING IN THE COURSE OF PARALLEL ARCHITECTURES

In this article we present the development of an artistic mosaic effect between two images used in the teaching of parallelism and computer vision techniques. The implementation of the effect with the use of parallelism, the programming language C and the OpenCV library allows students to discover the advantages and limitations of this paradigm in the context of an image processing application, which we consider attractive for students. , providing practical knowledge and contributing to improve the theorists.

Keywords: lecturing, parallel computing, vector processing, image processing.

Referencias

[1] Guía docente Arquitecturas Paralelas, https://secretaria.uvigo.gal/docnet-nuevo/guia_docent/index.php?centre=106&ensenyament=O06G150V01&assignatura=O06G150V01401

[2] Intrinsics for Intel® Streaming SIMD Extensions (Intel® SSE), https://software.intel.com/en-us/node/524252

[3] Intel® Intrinsics Guide, https://software.intel.com/sites/landingpage/IntrinsicsGuide/#

[4] Photoshop Tutorial, Mosaic Effect, https://www.youtube.com/watch?v=d0KP57dTRc0

[5] Stallings, William, Organización y arquitectura de computadores, 7ª edición, Prentice Hall, 2006, Madrid

[6] Estabilización de video, https://filmora.wondershare.com/es/consejos-avanzados-edicion-video/como-estabilizar-videos-tembleorosos.html

[7] Instrucción SSE2 suma de diferencias en valor absoluto `_mm_sad_epu8`, https://software.intel.com/sites/landingpage/IntrinsicsGuide/#text=sad&techs=SSE2&expand=4562

[8] The Software Vectorization Handbook. Applying Multimedia Extensions for Maximum Performance, A.J.C. Bik. Intel Press, June, 2004.

[9] Intel® VTune™ Amplifier, https://software.intel.com/en-us/intel-vtune-amplifier-xe