Aplicaciones Multimedia de Técnicas de Paralelismo

Valentina Romero, Matias García Rivera, Miguel Diaz-Cacho,
Esc. Ing. Informática, Universidad de Vigo, vromero18@esei.uvigo.es, mgrivera@uvigo.es, mcacho@uvigo.es

René Lastra
Oficina de Proyectos Internacionales, Universidad de Vigo, renelastracid@hotmail.com

Resumen

La visión artificial es una de las técnicas mas utilizadas para la sensorización de datos de retorno en sistemas de control. Este artículo muestra el empleo de la técnica del paralelismo enfocado a procesamiento de imagen y video, tanto con instrucciones de tipo SIMD como con threads (hilos), como método docente. La finalidad del empleo de estas técnicas es conseguir aprovechar al máximo, en la medida de lo posible, los recursos de la computadora, así como conseguir un mejor rendimiento y velocidad de procesamiento. Las técnicas de paralelismo tienen muy diversas aplicaciones, resaltándose en este caso las aplicaciones en la industria y particularmente en el paradigma de la Industria 4.0 y el mantenimiento. Este artículo muestra cómo utilizar estas técnicas utilizando el lenguaje de programación C y la librería OpenCv.

Palabras clave: Paralelismo, OpenCV, IIoT.

1 Introducción

Una de las formas de crear sensores de defectos visuales para la industria y en particular para la ingeniería del mantenimiento es la de dividir la imagen en secciones y comparar todas las secciones con las secciones de un patrón visual del defecto. La división de la imagen en secciones y su comparación con otra tiene los mismos fundamentos que la creación de un "*efecto mosaico*" tomando como base una imagen cualquiera. El "*efecto mosaico*" permite crear en entornos docentes una práctica amena y visual sin perder en ningún momento la perspectiva de la detección de patrones mediante comparación.

En la materia de Arquitecturas Paralelas de la Escuela de Ingeniería Informática de la Universidad de Vigo [6], se plantea el uso de técnicas de paralelismo del tipo SIMD (Singel Instruction Multiple Data) para procesar imagen y vídeo, centrándose en el empleo de instrucciones SSE2 (Streaming SIMD Extensions 2), threads e instrucciones AVX2 (Advanced Vector Extensions 2). Existen precedentes en la misma materia utilizando el paralelismo en video [1] e incluso utilizando el audio como ejemplo de paralelismo empleando la librería Portaudio [12]. El objetivo esencial de esta materia es el de aprovechar las posibilidades que ofrecen las arquitecturas monoprocesador, procesadores vectoriales, sistemas multiprocesador, sistemas multicomputadora, clusters y los procesadores multi-core. Para la consecución de este objetivo es necesario introducirse en las prácticas del paralelismo, demostrando que existe una mejora bastante notable entre aquellos programas que hacen uso de estas técnicas y aquellos que no las aplican y observando cómo afecta a la velocidad de ejecución de los mismos y al rendimiento del sistema en general.

Las competencias de la materia de Arquitecturas Paralelas son:

- Utilizar los conocimientos adquiridos para conseguir programar con un alto rendimiento.
- Ser capaz de diseñar o modificar algoritmos para que apliquen paralelismo, comprendiendo cómo se puede descomponer un problema y cómo afectará el utilizar estas técnicas a la hora de implementarlo.

Los trabajos asociados al "*efecto mosaico*" comparan una imagen patrón (imagen P) con cualquier otra imagen (imagen O), dividiendo ambas imágenes en sectores del mismo tamaño y asignando pesos a cada uno en función de los parámetros que se quieran comparar. Una vez hecho se sustituyen los sectores de la imagen O por los sectores "mas parecidos" de la imagen P. Esta técnica se hace habitualmente para imágenes estáticas, pero en la industria, y en particular en la ingeniería de mantenimiento es necesario conocer el comportamiento de un sistema de forma dinámica, por lo que se hace necesario comparar un movimiento con un patrón. La realización del "*efecto mosaico*" no es conocida para imágenes en movimiento.

Un ejemplo de efecto mosaico, orientado a la detección de irregularidades en una yanta se presenta en la figura 1, donde la imagen patrón P es un

https://doi.org/10.17979/spudc.9788497498043.226

cuadro de materiales, y la imagen original O es una yanta estándar de un automóvil. El resultado sería la imagen R.

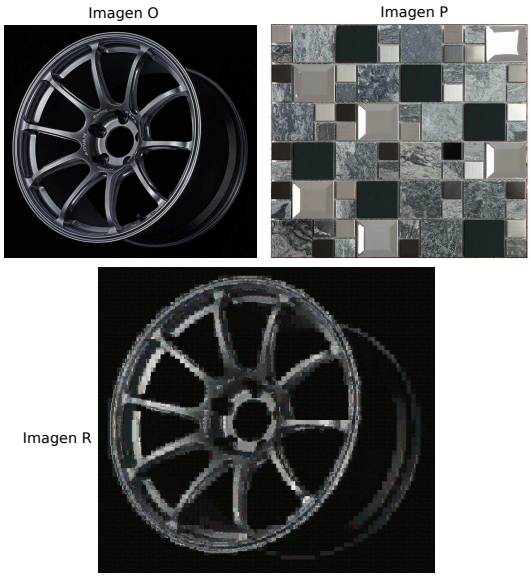

Figura 1: Ejemplo efecto mosaico aplicado a una yanta

Este trabajo muestra cómo desarrollar la forma de crear un mosaico de un video partiendo de una imagen base utilizando técnicas de paralelismo. De esa forma se consiguen proyectar las competencias a adquirir por el alumnado que curse la materia con los requerimientos presentes y futuros de la producción y mantenimiento industrial.

Los objetivos y la motivación de la realización de *efectos mosaicos* para la industria y el entorno académico se adelantan en esta introducción. La sección II presenta las técnicas de paralelismo y su aplicación a situaciones reales. La sección III describe la creación de un mosaico entre un vídeo y una imagen con técnicas específicas de paralelismo SSE2 y división en hilos o Threads. Finalmente las Conclusiones se resumen en la sección IV.

2 DESARROLLO DE LAS TÉCNICAS DE PARALELISMO APLICADO A CASOS PRÁCTICOS

El lenguaje C está demostrado ser el mas eficaz para el aprovechamiento de las cualidades de un procesador, al ser prácticamente un lenguaje ensamblador de alto nivel. No obstante requiere de una curva de aprendizaje mucho mas atenuada que otros lenguajes de programación. Para la programación utilizando técnicas de paralelismo en procesamiento de imagen, este lenguaje es el mas recomendado, a pesar de que otros lenguajes como python disponen de las librerías OpenCV [10]. Uno de los motivos es la existencia de funciones nativas como "intrinsics", SSE2 e instrucciones AVX2.

2.1 INTRINSICS Y C

Un intrinsics es un conjunto de funciones que se compilan directamente a una secuencia de una o más instrucciones en lenguaje ensamblador [9]. Los intrinsics de Intel fueron pensados desde un principio para ser utilizadas en lenguaje C. Se pueden emplear con otros lenguajes, pero eso implicaría la traducción de esas instrucciones en C a esos lenguajes.

2.2 INSTRUCCIONES SSE2

El conjunto de funciones intrinsics son las instrucciones que se emplean al trabajar con SIMD (Single Instruction Multiple Data). Esta técnica es la que se utiliza para conseguir paralelismo a nivel de datos. Dentro de esta técnica, se encuentran algunas de las funciones "intrinsics". Uno de los conjuntos de funciones "intrinsics" es el conjunto de instrucciones SSE2 (Streaming Single instruction multiple data Extensions 2). SSE2 reemplazó a la versión anterior (SSE1) y a MMX. SSE2 no tiene ningún registro en común con la FPU x87 [11] (registros reservados para instrucciones de coma flotante) y además añade 8 nuevos registros no compartidos. En cambio, MMX y SSE1 sí tienen registros compartidos con la FPU x87, lo cual genera conflictos. Además, los últimos cambios que se hicieron en SSE2 fueron en el año 2004, por lo tanto, cualquier persona con un ordenador de 64 bits podrá ejecutar este tipo de instrucciones.

Para poder trabajar con SSE2 se debe de incluir la librería *emmintrin* al principio del programa ($\#include < emmintrin.h >$) y también es necesario indicarle al compilador que se utiliza SSE2 (-msse2 en las opciones del compilador). El tipo de dato empleado es el de un registro de 128 bits que almacena integers (las imágenes trabajan con integers de 8 bits). Este tipo de dato es $\_\_m128i$.

A continuación se describirán las instrucciones utilizadas en las prácticas de Arquitecturas Paralelas, [4] que sirven tanto para crear un mosaico entre 2 imágenes como para crear un mosaico entre un vídeo y una imagen.

- __m128i _mm_loadu_si128 (__m128i const* mem_addr)

  Carga 128 bits de datos en destino. A diferencia de la instrucción __m128i _mm_load_si128 (__m128i const* mem_addr), no necesita estar "acotada" bajo ningún límite particular como

https://doi.org/10.17979/spudc.9788497498043.226

pasa con la última, que debe estar acotada en un límite de 16 bytes (si lo supera, da lugar a una excepción). Ocurre lo mismo con todas las demás instrucciones de ese estilo.

- void _mm_storeu_si128 (__m128i* mem_addr, __m128i a)

  Guarda 128 bits de datos del registro 'a' en destino.

- __m128i _mm_add_epi32 (__m128i a, __m128i b)

  Añade (suma) 32 bits de datos empaquetados en los registros a y b y guarda los resultados en destino.

- __m128i _mm_srli_si128 (__m128i a, int imm8)

  Desplaza a la derecha imm8 bytes mientras se desplaza en ceros y guarda los resultados en destino.

- __m128i _mm_sad_epu8 (__m128i a, __m128i b)

  Calcula diferencias absolutas (resta y hace el valor absoluto del resultado) entre los enteros sin signo de 8 bits de los registros 'a' y 'b', a continuación, realiza la suma horizontal cada 8 diferencias consecutivas, con el fin de generar enteros de 16 bits sin signo y luego empaqueta estos últimos en los 16 bits bajos de los elementos de 64 bits en destino. Esta instrucción es muy útil, ya que permite calcular las diferencias entre 2 bloques (bloques de pixel) a la vez, es decir, en paralelo. Esto quiere decir que la carga computacional se verá reducida, lo cual implica una mejora de rendimiento.

- int _mm_cvtsi128_si32 (__m128i a)

  Copia el entero bajo de 32 bits en el registro 'a' en destino.

2.3 INSTRUCCIONES AVX2

AVX2 [6] es un conjunto de instrucciones que emplean paralelismo que surge como mejora de AVX y de SSE2. La novedad de AVX2 fue el aumento de tamaño de los registros, que pasaron de ser de 128 bits a ser de 256 bits. Además, añade 32 nuevos registros no compartidos. Como consecuencia de su mayor tamaño, se puede almacenar más información (el doble que en SSE2) y tratarla simultáneamente, es decir, de forma paralela. Otra ventaja es que no solo admite instrucciones de 256 bits, sino que también admite las de 128 bits. El tipo de dato que se usa para AVX2 correspondiente con __m128i es __m256i. Además, es necesario incluir la libreía immintrin (#include ¡immintrin.h¿) y hay que indicarle al compilador que se están empleando instrucciones AVX2 (-mavx2 en las opciones del compilador). El funcionamiento de estas instrucciones es exactamente el mismo que el de las de SSE2. Algunas de las instrucciones AVX2 que se han utilizado en este trabajo son las siguientes:

- __m256i _mm256_loadu_si256 (__m256i const* mem_addr)
- void _mm_storeu_si256 (__m256i* mem_addr, __m256i a)
- __m256i _mm256_sad_epu8 (__m256i a, __m256i b)

2.4 Threads

Los threads, también conocidos como Hilos, se definen como la agrupación de un trozo de programa junto con el conjunto de registros del procesador que utiliza y una pila de máquina. [2] El uso de threads tiene la finalidad de repartir el trabajo para conseguir unas mejores prestaciones, consiguiendo que varias tareas se ejecuten de forma paralela [5]. En este trabajo se utilizaron threads para distribuir la tarea de creación del mosaico de cada frame de un vídeo. Además, utilizándose de forma conjunta con paralelismo, SSE2 en este caso, los resultados fueron superiores.

3 CREACIÓN DE UN MOSAICO ENTRE UN VÍDEO Y UNA IMAGEN CON SSE2 Y THREADS

Este trabajo consiste, en síntesis, en crear el mosaico de un vídeo. Para ello, se partirá de un vídeo y una imagen base. El objetivo es ver en una ventana los frames del vídeo original y en otra ventana los frames en mosaico utilizando SSE2 [8] y threads [7]. Además, es importante prestar atención al tiempo de ejecución, ya que se intentará que el vídeo se vea fluido.

Para la creación del mosaico, se precisan 3 funciones:

```
void copiarBloqueSSE2(int i, int j, IplImage * imagenOrigen, int k, int l, IplImage * imagenDestino) {

  int fila, columna;

  for (fila = 0; fila < ALTOBLOQUE; fila++) {
    __m128i *origen = (__m128i*) (imagenOrigen->imageData + (i + fila) * imagenOrigen->widthStep
        + j * imagenOrigen->nChannels);
    __m128i *destino = (__m128i*) (imagenDestino->imageData + (k + fila) * imagenDestino->widthStep
        + l * imagenDestino->nChannels);
    for (columna = 0; columna < (ANCHOBLOQUE * imagenDestino->nChannels); columna += ANCHOBLOQUE) {
      __m128i reg = _mm_loadu_si128(origen++);
      _mm_storeu_si128(destino++, reg);
    }
  }
}
```

Figura 2: Función copiar bloque con SSE2

https://doi.org/10.17979/spudc.9788497498043.226

```
int compararBloqueSSE2(int filaBloque1, int columnaBloque1, IplImage * img1,
    int filaBloque2, int columnaBloque2, IplImage * img2) {
  int fila, columna,
      sad, suma = 0;
  __m128i reg1, reg2, reg3, reg4, reg5;

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

    __m128i *origen = (__m128i*) (img1->imageData +
        ((fila + filaBloque1) * img1->widthStep)
        +(columnaBloque1 * img1->nChannels));
    __m128i *destino = (__m128i*) (img2->imageData +
        ((fila + filaBloque2) * img2->widthStep)
        +(columnaBloque2 * img2->nChannels));
    for (columna = 0; columna < (ANCHOBLOQUE *
        img2->nChannels); columna += ANCHOBLOQUE){
      reg1 = _mm_loadu_si128(origen++);
      reg2 = _mm_loadu_si128(destino++);
      reg3 = _mm_sad_epu8(reg1, reg2);
      reg4 = _mm_srli_si128(reg3, 8);
      reg5 = _mm_add_epi32(reg3, reg4);
      sad = _mm_cvtsi128_si32(reg5);
      suma += sad;
    }
  }
  return suma;
}
```

Figura 3: Función comparar bloque con SSE2

Las 2 funciones anteriores, en conjunto, sirven para buscar en una imagen 'A' el bloque de pixel más semejante al bloque de pixel correspondiente en una imagen 'B' y a continuación, copiarlo.

La siguiente función es la más importante en esta práctica de laboratorio.

```
void mosaicoThread(void *ptr) {
  int parte = *((int*) ptr);
  int filasP = Video->height / NTHREADS;
  unsigned int fila1 = 0, fila2,
      columna1, columna2,
      diferencia, nuevaDiferencia,
      i, j;
  for (fila1 = parte * filasP; fila1 < (parte + 1) * filasP; fila1 += ALTOBLOQUE) {
    for (columna1 = 0; columna1 < Video->width; columna1 += ANCHOBLOQUE) {
      diferencia = ANCHOBLOQUE * ANCHOBLOQUE * 255 * 3 + 1;
      for (fila2 = 0; fila2 < Video->height; fila2 += ALTOBLOQUE) {
        for (columna2 = 0; columna2 < Video->width; columna2 += ANCHOBLOQUE) {
          nuevaDiferencia = compararBloqueSSE2(fila1, columna1,
              Video, fila2, columna2, Frutas);
          if (nuevaDiferencia < diferencia) {
            diferencia = nuevaDiferencia;
            i = fila2;
            j = columna2;
          }
        }
      }
      copiarBloqueSSE2(i, j, Frutas, fila1, columna1, Mosaico);
    }
  }
}
```

Figura 4: Función mosaico Thread

La función mosaicoThread (figure 4) es la función a la que se llamará en el método main. Esa llamada se realizará para indicarle a cada thread que se invoque, que tiene que realizar el mosaico.

NTHREADS se corresponde con el número de threads que se van a utilizar. Es conveniente conocer las capacidades del procesador con el que se va a trabajar, ya que el número de threads que se puedan usar, depende directamente del procesador. Se puede trabajar hasta con 8 threads para aquellos procesadores que tienen 4 núcleos o incluso 12 threads, para aquellos procesadores con

```
pthread_t threads[NTHREADS];
int i, filas[NTHREADS];

while (cvGrabFrame(Capture)) {

  Video = cvQueryFrame(Capture);

  if (!Video) {
    break;
  }

  for (i = 0; i < NTHREADS; i++) {
    filas[i] = i;
    pthread_create(&threads[i], NULL,
        (void *) &mosaicoThread,
        (void *) &filas[i]);

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

  cvShowImage("Video Shrek", Video);
  cvShowImage("Mosaico", Mosaico);

  cvWaitKey(1);
  cvWriteFrame(pWriter, Mosaico);
}
```

Figura 5: Bucle para mostrar por pantalla el vídeo base y el mosaico del vídeo

6 núcleos. A veces, al ver las características del procesador, puede indicar que es capaz de trabajar con más threads de los que realmente puede. Esto se debe a la tecnología Hyperthreading. Esta tecnología consiste en que cada núcleo cambie la ejecución entre 2 threads a una velocidad muy alta. Esto no quiere decir que tenga el mismo rendimiento que un procesador que sí es capaz de trabajar normalmente con ese número de threads, de hecho es menor, pero mayor que si se empleasen menos threads de los posibles.

Se pueden comprobar las características de la CPU mediante la aplicación CPU Z para Windows [3].

En el método main, se creará un array, threads[NTHREADS], que contendrá tantos elementos como threads haya y será necesario a la hora de "llamar" a los threads y un array, filas[NTHREADS], encargado de definir la parte de la imagen que se le encargará a cada thread.

Lo que hace Capture es almacenar el vídeo base que se haya escogido. Video almacena cada frame del vídeo base. Es importante tener en cuenta que un vídeo es una sucesión de imágenes, por lo tanto nunca se deja de trabajar con imágenes como se hacía en prácticas anteriores.

Se podría decir que el bucle hace lo siguiente: mientras haya vídeo base, se guardará cada frame en Video, el cual también se mostrará por pantalla. Hay dos bucles for. El primero se encarga de crear los threads hijos (tantos como se haya in-

https://doi.org/10.17979/spudc.9788497498043.226

dicado en NTHREADS) y de invocar a la función mosaicoThread, indicando al thread invocado la parte del mosaico que tiene que crear. El segundo for es el encargado de esperar a que el thread que se le indique finalice y una vez este termine, parar la ejecución del thread padre.

Con cada frame, muestra en una ventana ("Video Shrek") los frames del vídeo base y en otra ventana ("Mosaico") el resultado de realizar el mosaico de cada frame del vídeo base, a tiempo real. Es decir, ambas ventanas se reproducen de manera fluida simultáneamente.

El bucle principal termina cuando ya no hay más frames en el vídeo base, lo que indicaría que el vídeo habría terminado. El resultado [13] se puede comprobar en el vídeo al que se puede acceder escaneando el código QR. También cuenta con una breve explicación acerca del funcionamiento del programa.

Figura 6: Video efecto mosaico

4 Conclusiones

El uso de técnicas de paralelismo es especialmente útil a la hora de programar con altas prestaciones. En el caso práctico propuesto en este artículo, explicado en el apartado 3.2, se aprecia cómo utilizando threads combinado con SSE2, hay una mejora de rendimiento notable. En eso consiste utilizar técnicas de paralelismo. Con SIMD se consigue tratar varios datos al mismo tiempo con una sola instrucción y con threads se reparte la carga de trabajo de manera que la ejecución de la parte que le toca a cada thread se ejecuta en paralelo a las demás. En el proyecto descrito en este trabajo, se ha creado el *efecto mosaico* de un vídeo mediante el empleo de threads y SSE2.

Agradecimientos

Este trabajo ha sido realizado con el soporte de los proyectos ANL-MEd (586035-EPP-1-2017-1-DZ-EPPKA2-CBHE-JP) y SM-TMC (586035-EPP-1-2017-1-DZ-EPPKA2-CBHE-JP).

English summary

PARALLELISM TECHNIQUES APPLIED TO MULTIMEDIA

Abstract

Artificial vision is one of the most used techniques to collect feedback data in control systems. This article shows the use (as teaching method) of the technique of parallelism centered on image and video processing, both with SIMD-type instructions and threading. The purpose of using these techniques is to take advantage as much as possible of the resources of the computer, as well as to achieve better performance and to reduce processing time. These techniques has several applications, but highlighting the use in Industry 4.0 and Maintenance 4.0. This article shows how to use these techniques using the C programming language and the OpenCv library.

Keywords: Parallelism, IIoT, OpenCV.

Referencias

[1] C. Márquez, M. García, M. Díaz-Cacho, J. Camaño, "Paralell computing technologies in video stabilization for teaching purposes", artículo para 'Jornadas de Automática 2019'

[2] Concepto de Thread, [online] https:/www.rastersoft.com/OS2/CURSO/THREAD.HTM

[3] CPU Z, System Information Software, [online] https:/www.cpuid.com/softwares/cpu-z.html.

[4] Intel Intrinsics Guide, intel.com, [online] https://software.intel.com/sites/landingpage/IntrinsicsGuide/

[5] James Reinders, Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism, O'Reilly, 2007'.

[6] M.G.Rivera, (2019) Guía docente de Arquitecturas Paralelas, ESEI-Universidad de Vigo, España.

[7] M.G.Rivera, (2019), Mosaico y Paralelismo con Hilos, material de Arquitecturas Paralelas, Dpto. Ing. Sistemas y Automática, Universidad de Vigo, España.

[8] M.G.Rivera, (2019), Mosaico y Paralelismo SIMD, material de Arquitecturas Paralelas, Dpto. Ing. Sistemas y Automática, Universidad de Vigo, España.

https://doi.org/10.17979/spudc.9788497498043.226

[9] M.G.Rivera, (2019), Transiciones entre imágenes con SSE2, material de Arquitecturas Paralelas, Dpto. Ing. Sistemas y Automática, Universidad de Vigo, España.

[10] OpenCV, opencv.org, [online] https://opencv.org.

[11] Parhami, Behrooz, Arquitectura de computadoras: de los microprocesadores a las supercomputadoras, McGraw-Hill Interamericana, 2007, [online] http://www.apps4two.com/curso_dba/bibliografia/3-Arquitectura.de.computadoras.Behrooz.Parhami.PDF

[12] Portaudio, portaudio.com, [online] http://www.portaudio.com.

[13] Valentina Romero, Efecto mosaico de un video con threads y SIMD, 2020. [online],

https://doi.org/10.17979/spudc.9788497498043.226