View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by El Servicio de Difusión de la Creación Intelectual

Reconocimiento de objetos en video utilizando SIFT Paralelo

Bernarda Albanesi, Nadia Funes, Franco Chichizola, Laura Lanzarini

III-LIDI (Instituto de Investigación en Informática LIDI)
Facultad de Informática. Universidad Nacional de La Plata
La Plata, Buenos Aires, Argentina
balbanesi@lidi.info.unlp.edu.ar, funesnadia@hotmail.com,
{francoch, laural}@lidi.info.unlp.edu.ar

Resumen. El reconocimiento de objetos en video es una tarea computacionalmente costosa y de sumo interés en distintas áreas. Existen métodos que permiten caracterizar una imagen a través de la representación vectorial de sus pixels más relevantes entre los cuales SIFT es uno de los más utilizados. Este artículo propone una implementación paralela de este método con el objetivo de realizar tracking de video. Se trata de una solución de grano grueso sobre una arquitectura multicore que permite llegar a un rendimiento cercano al óptimo al usar un esquema *Bag of Task* para balancear el trabajo. Se logra incrementar la cantidad de frames resueltos por segundo para el procesamiento de video en Tiempo Real.

Palabras Claves: Tracking de Video, Descriptores SIFT, Arquitectura Multicore

1 Introducción

El reconocimiento de objetos en video es un problema de sumo interés en distintas áreas como seguridad, edición de video, entretenimientos e incluso control de accesos. Si bien se espera que cuadros sucesivos contengan objetos similares suelen producirse numerosos cambios en lo que respecta al punto de vista, la escala, la iluminación e incluso la oclusión parcial de dicho objeto.

Existen distintos métodos que permiten caracterizar una imagen a través de una representación vectorial de sus pixels más relevantes. En todos los casos se busca expresar la información referida a la vecindad de dichos puntos de la forma más invariante posible. El método SIFT definido por Lowe en [1] es, sin duda, uno de los más utilizados. Sin embargo, su costo computacional es alto si se piensa en su aplicación en tareas tales como tracking de video.

Este artículo propone una implementación paralela del método de SIFT sobre una arquitectura paralela de memoria compartida, aprovechando el amplio desarrollo de los *multicores*. Este tipo de máquina permite obtener menores tiempos de respuesta para una aplicación al dividir el trabajo entre los cores que lo forman [2][3]. Para lograr tiempos de respuesta acordes a los requeridos para el procesamiento de videos

en tiempo real se utiliza un esquema *Bag of Task* para lograr de esta manera un balance de carga que asegure un buen rendimiento de la aplicación [4].

Este trabajo está organizado de la siguiente forma: la sección 2 describe los trabajos relacionados existentes, la sección 3 explica brevemente el método de Lowe en el cual está basado este artículo, la sección 4 detalla la estrategia de paralelización propuesta, la sección 5 presenta los resultados obtenidos y la sección 6 presenta las conclusiones así como algunas línea de trabajo futuras.

2 Trabajos Relacionados

Lowe definió en [1] un método para extraer características de una imagen las cuales pueden utilizarse para hallar correspondencia entre dos vistas distintas de un mismo objeto. Dichas características, denominadas características SIFT (Scale Invariant Feature Transform), son invariantes a la escala y rotación de la imagen y bastante invariantes a cambios del punto de vista y de iluminación. Además poseen la capacidad de ser muy distintivas. Consiste en la identificación de los puntos más relevantes de la imagen y la determinación para cada uno de ellos de un vector numérico asociado. Este vector posee longitud 128 y se calcula a partir del histograma determinado por la información del gradiente en 8 direcciones diferentes. SIFT es un método de caracterización muy robusto aunque su mayor problema es su alto índice de falsos positivos.

A partir de este trabajo surgieron otras propuestas. El método RIFT es una generalización de SIFT que hace énfasis en la invarianza a la rotación. La construcción del vector asociado se realiza normalizando un área circular alrededor del punto de interés, dividiéndola en anillos concéntricos de igual ancho para luego calcular un histograma de orientación de gradientes [5]. G-RIF (Generalized Robust Invariant Feature) es un descriptor de contexto general que combina un detector de simetría radial con un detector de estructura corner-like. Codifica la orientación y densidad de borde y la información de color de manera unificada combinando la información sensorial con la codificación espacial [6]. El método SURF fue presentado por Herbert Bay et al. en 2006 y está parcialmente inspirado en SIFT. Su funcionamiento utiliza funciones 2D Haar wavelet en el vecindario de cada punto de interés. Utiliza sólo 64 dimensiones y hace uso del signo del laplaciano lo que redunda en un menor tiempo de cómputo y una mayor robustez del descriptor [7][8]. PCA_SIFT fue definido por Ke et al en 2004 [9]. Es una variante de SIFT que busca reducir la detección de falsos positivos. Los puntos de interés para los cuales se construyen los descriptores se obtienen como en SIFT. Luego, para cada uno de ellos, se utiliza una vecindad de 39x39 y se codifican los gradiente en las dirección x e y por separado. Esto da como resultado un vector de dimensión 3042 que luego se reduce a dimensión 32 utilizando PCA (Principal Component Analysis).

Los trabajos anteriores sólo pretenden mostrar la importancia del método SIFT en el estado del arte. Sin embargo, su costo computacional es alto. Su uso en reconocimiento de objetos en video requiere la extracción de características en tiempo real. Las investigaciones realizadas se centran principalmente en la ejecución y la

aceleración de la extracción de características SIFT en unidades de procesamiento gráfico (GPU). Sinha et al. ha aplicado SIFT en una GPU [10]. Debido a las limitaciones de hardware y OpenGL es necesario realizar transferencias de datos entre la GPU y la CPU, lo cual consume una parte importante del tiempo de cómputo. Como resultado, su aplicación puede extraer alrededor de 800 punto relevantes (keypoints) de un vídeo de 640x480 a 10 cuadros por segundo (FPS). Recientemente, Heymann et al. propusieron otra aplicación SIFT en las GPU, lo que se puede aplicar a secuencias de imágenes con 640x480 píxeles a 20 fps. [11]. Actualmente, puede utilizarse la potencia de cálculo de los sistemas multi-core para acelerar la aplicación del método de SIFT al reconocimiento de objetos en video.

3 Descriptores SIFT

El proceso definido en [1] para determinar las características SIFT de una imagen consiste de cuatro pasos:

- En primer lugar se calcula la ubicación de potenciales puntos de interés dentro de la imagen, los que corresponden a los puntos extremos calculados a partir de subconjuntos de planos de diferencias de filtros gaussianos (DoG) aplicados sobre la imagen a distintas escalas.
- Luego, se descartan los puntos de interés que poseen bajo contraste. Esto es una mejora con respecto a lo definido en [12].
- En tercer lugar, se calculan las orientaciones de los puntos de interés relevantes.
- Utilizando las orientaciones anteriores, se analiza el entorno de cada punto y se determina el vector de características correspondientes.

Como resultado de este proceso se obtiene un conjunto de vectores de características de longitud 128 que pueden ser comparados con los correspondientes a otra imagen del mismo objeto con diferente escala, orientación y/o punto de vista.

Dicha comparación puede realizarse directamente a través de una medida de distancia estableciendo un umbral de similitud.

4 Paralelización del método de SIFT

La extracción de características utilizando los descriptores SIFT es un proceso costoso en tiempo al aplicarlo para reconocer objetos en los diferentes frames de un video. Esto dificulta su aplicación en procesos que requieren obtener respuesta en tiempo real.

Esta necesidad de reducir el tiempo de cómputo se puede resolver incrementando la potencia de cómputo de las máquinas. Se ha llegado a un punto en el cual escalar la velocidad de los procesadores convencionales (un núcleo), trae aparejado problemas de consumo de energía y generación de calor.

Esta limitación ha llevado a desarrollo y utilización masiva de arquitecturas paralelas de memoria compartida, como lo son los actuales *multicores*. Un procesador multicore surge a partir de integrar dos o más núcleos computacionales dentro de un mismo "chip" [13]. Este tipo de máquina permite incrementar el rendimiento de una aplicación al dividir el trabajo de cómputo entre los núcleos disponibles [2][3].

Dada la necesidad de lograr tiempos de respuesta acordes a los requeridos para el procesamiento de videos en tiempo real, sumado a la disponibilidad de estas arquitecturas paralelas, se desarrolla una versión paralela del algoritmo.

4.1 Algoritmo paralelo desarrollado

Para este tipo de aplicación en la que se debe realizar un procesamiento costoso sobre un conjunto de frames de forma independiente, resulta claro que la forma más simple y eficiente de paralelizar es distribuir las imágenes que forman el video entre los *threads* (o hilos) generados.

Un punto a tener en cuenta es la forma en que se distribuye el trabajo entre los hilos dado que el procesamiento a realizar en cada frame depende de los datos (distribución de los *pixels*) y no sólo del tamaño de la imagen.

Para obtener un buen rendimiento del algoritmo se debe balancear la carga de trabajo entre los diferentes *threads*. Para cumplir este objetivo se debe pensar en un esquema de distribución *Bag of Task* en el cual cada hilo se encarga de sacar una nueva tarea a realizar (frame a procesar) de un repositorio común a todos ellos [4].

Lo interesante de esta solución paralela es que no debe hacer ningún procesamiento extra al realizado por el algoritmo secuencial. Excepto aquellas acciones implícitas del compilador como son la creación o activación de los hilos, y la sincronización para acceder al repositorio de tareas.

Para implementar el algoritmo paralelo se usó OpenMP sobre el lenguaje C. Este es un modelo basado en primitivas que permite manejar con un alto nivel de abstracción la administración, comunicación y sincronización de *therads* por medio de Memoria Compartida.

Un programa en OpenMP comienza con la ejecución de un hilo principal, el cual al llegar a un *Bloque Paralelo* genera o activa una cantidad previamente definida de *threads* para trabajar en paralelo. Al terminar todos los *threads* se destruyen o desactivan continuando únicamente el hilo principal hasta que se llegue a otro *Bloque Paralelo* o al final del programa. Dentro de los *Bloques Paralelos* se pueden usar directivas que permiten especificar concurrencia entre iteraciones y tareas (constructores de trabajo compartido). Uno de estos constructores es la directiva *For* que permite distribuir las iteraciones entre los *threads* que se están ejecutando, especificando la forma de realizar el *Scheduling*.

La Figura 1 muestra el pseudocódigo de la solución paralela. El hilo principal calcula el vector de características de la imagen que se desea buscar en el video (*Imagen a buscar*) por medio de la función *SIFT* y lo almacena en la variable compartida *descriptor*. Al llegar a la directiva ***#pragma omp parallel for***, genera el conjunto de *threads* para realizar en forma concurrente las iteraciones del *for i*. Los hilos sólo comparten la variable *descriptor*, el resto son variables privadas (o locales) a cada uno. En cada iteración se calculan los descriptores del frame correspondiente

(*SIFT*), y se lo compara con el de la *imagen a buscar* para determinar si el *puntaje* supera el umbral establecido, y en caso de ser así obtener la *posición* dentro del frame donde se ubica el objeto (para esto utiliza la función *Matching*SIFT).

```
main () {
   descriptor = SIFT (Imagen a buscar)
   #pragma omp parallel for default (private) shared (descriptor) schedule (dynamic,1)
      for (i = 0; i < N; i++) {
            descriptorFrame = SIFT (Frame_i)
            (posición, puntaje) = MatchingSIFT (descriptor, descriptorFrame)
      }
}
```

Fig. 1. Pseudocódigo de la solución paralela.

4.2 Descripción de las pruebas realizadas

Para realizar las pruebas se usa un multicore Dell Poweredge 1950, cuyas características son las siguientes: 2 procesadores quad core Intel Xeón e5410 de 2.33 GHz, 4 Gb de memoria RAM (compartida entre ambos procesadores) y memoria cache L2 de 6Mb entre cada par de núcleos de los procesadores.

Para poder analizar el comportamiento del algoritmo paralelo se utiliza un video en tonos de gris cuyos frames tienen un tamaño de 480x640. Las pruebas realizadas varían en la cantidad *N* de frames utilizados (*N* = 250, 500, 1000, 1250, 1500, 1750, 2000, 2250 y 2500). Por cada valor de *N* se hicieron 15 corridas del algoritmo para obtener un tiempo paralelo promedio. En todos los casos se utilizan 8 *threads* (1 por cada núcleo de la arquitectura).

5 Resultados

Para evaluar el comportamiento de los algoritmos desarrollados se analiza el *speedup y* la *eficiencia*, en este caso sobre arquitecturas homogéneas dado que todos los núcleos son iguales [4][14][15].

La Ecuación 1 indica la forma de calcular el *speedup*, donde *TS* es el tiempo de ejecución del mejor algoritmo secuencial en uno de los núcleos de una hoja, y *TP* en tiempo paralelo.

$$Speedup = \frac{TS}{TP}. \quad (1)$$

Para evaluar el rendimiento logrado por el algoritmo en la arquitectura utilizada se calcula la *eficiencia* como se indica en la Ecuación 2, donde *p* es la cantidad total de núcleos usados. En el caso de las pruebas desarrolladas en este trabajo *p* = 8.

$$Eficiencia = \frac{Speedup}{p} \quad (2)$$

En la Tabla 1 se muestra un resumen de los resultados obtenidos en las pruebas realizadas. Para cada cantidad de frames (*N*) se detalla el tiempo secuencial de la ejecución más rápida en segundos (*TS*), el tiempo paralelo promedio en segundos (*TP*) de las 15 ejecuciones repetidas, el *speedup* y la *eficiencia*.

Tabla 1. Resumen con los resultados promedio de las pruebas.

Cantidad de Frames	*TS*	*TP*	*Speedup*	*Eficiencia*
250	277,82	38,06	7,30	0,91
500	552,83	73,82	7,49	0,94
750	832,63	109,81	7,58	0,95
1000	1103,67	145,40	7,59	0,95
1250	1390,55	181,24	7,67	0,96
1500	1655,61	216,90	7,63	0,95
1750	1938,21	252,82	7,67	0,96
2000	2226,80	288,78	7,71	0,96
2250	2490,00	324,43	7,68	0,96
2500	2780,49	360,68	7,71	0,96

Como se puede calcular a partir de los tiempos paralelos promedios, la cantidad de frames que se pueden procesar por segundo pasa de ser menor a 1 (al usar el algoritmo secuencial) a casi 7 en el algoritmo paralelo. Cuanto mayor es la cantidad de frames utilizados, más se acerca a ese límite.

En la Figura 2 se muestra en forma gráfica la eficiencia del algoritmo paralelo para las pruebas con las distintas cantidades de frames. Para observar con más detalle las diferencias el eje vertical se grafica a partir del valor 0,90.

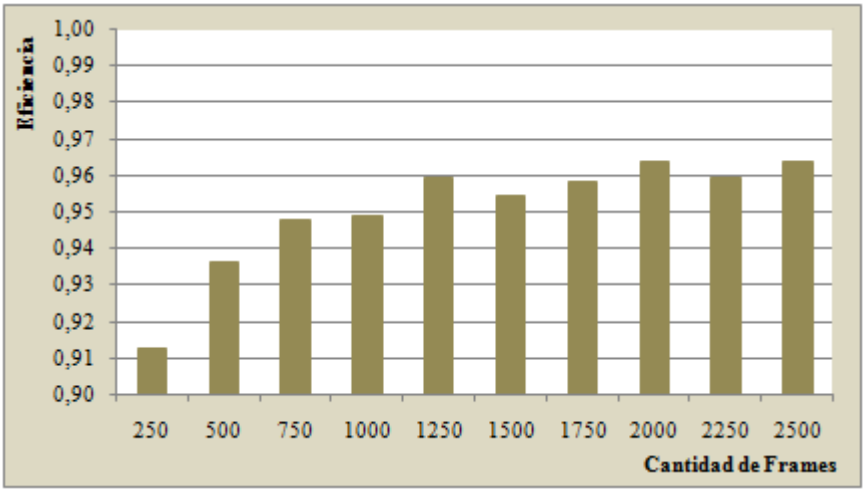

Fig. 2. Eficiencia promedio alcanzada para las diferentes cantidades de frames.

En este gráfico se puede observar que el algoritmo paralelo logra un rendimiento cercano al óptimo (*eficiencia = 1*). La eficiencia crece al incrementar la cantidad de frames utilizados tendiendo a estacionarse a partir de las 1200 imágenes, este comportamiento es de esperar de acuerdo a la Ley de Amdahl [4][14].

6 Conclusiones y líneas de trabajo futuras

Se ha presentado una versión paralela del método SIFT que permite su aplicación para realizar tracking de video. Los resultados obtenidos al procesar un video en tonos de grises han resultado satisfactorios.

El algoritmo paralelo no obtiene una eficiencia óptima (pero si muy cercano a él) por tres motivos:

- El segmento de código que se debe resolver en forma secuencial (obtener los descriptores del objeto a buscar).
- El overhead generado al crear/activar y sincronizar los threads.
- El tamaño de las cache compartida por cada par de cores, es suficientemente grande para trabajar con una imagen (algoritmo secuencial), pero no para procesar 2 frames al mismo tiempo (dos hilos del algoritmo paralelo), lo que genera más fallos de cache. Esto afecta en mayor grado a medida que se aumenta el tamaño de los frames.

Actualmente se está trabajando en incrementar el tamaño de los frames (produce reducción importante en la eficiencia por el problema mencionado anteriormente). Esto lleva a analizar una solución de grano fino para poder realizar la paralelización a nivel de obtención de puntos de interés en cada frame. Al distribuir el procesamiento de cada imagen entre los distintos cores, se reduce la cantidad de datos que debe almacenar cada hilo, decrementando los fallos de cache.

Poder representar objetos de un video a través de un conjunto de vectores característicos no sólo identifica a dichos objetos sino también al video que los contiene. Este enfoque podría permitir caracterizar videos a partir de sus objetos más relevantes haciendo factible el uso de técnicas de Minería de Textos para recuperar y catalogar dichos videos automáticamente. Este último enfoque es una de las líneas actuales de investigación.

7 Referencias

1. Lowe D. G., “Object recognition from local scale-invariant features”. In International Conference on Computer Vision, Corfu, Greece, pp. 1150-1157. 1999.
2. Mc Cool M., “Programming models for scalable multicore programming”. 2007. http://www.hpcwire.com/features/17902939.html.

3. Siddha S., Pallipadi V., Mallick A., “Process Scheduling Challenges in the Era of Multicore Processors”, Intel Technology Journal, Vol. 11, No. 4. 2007.

4. Grama A., Gupta A., Karypis G., Kumar V., “An Introduction to Parallel Computing. Design and Analysis of Algorithms. 2nd Edition”. Pearson Addison Wesley. 2003.

5. Lazebnik S., Schmid C., Ponce, J., “Semi-Local Affine Parts for Object Recognition”. Proceedings of the British Machine Vision Conference. 2004.

6. Sungho Kim, Kuk-Jin Yoon, In So Kweon, “Object Recognition Using a Generalized Robust Invariant Feature and Gestalt’s Law of Proximity and Similarity”. Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06). 2006.

7. Bay H., Tuytelaars T., Gool L.V., “SURF: Speeded Up Robust Features”. Proceedings of the ninth European Conference on Computer Vision. 2006.

8. Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, “SURF: Speeded Up Robust Features”. Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346-359. 2008.

9. Ke Y., Sukthankar R., “PCA-SIFT: A More Distinctive Representation for Local Image Descriptors”. Computer Vision and Pattern Recognition. 2004.

10. Sinha S. N., Frahm J. M., Pollefeys M., Genc Y., “Feature Tracking and Matching in Video Using Programmable Graphics Hardware”. Machine Vision and Applications. 2007.

11. Heymann S., Müller K., Smolic A., Froehlich B., Wiegand T., “SIFT Implementation and Optimization for General-Purpose GPU”. *In Proc. of the WSCG'07*, Plzen, Czech Republic. 2007.

12. Lowe D. G., “Distinctive image features from scale-invariant keypoints”. International. journal of computer vision, 60. 2004.

13. AMD, “Evolución de la tecnología de múltiple núcleo”. 2009. http://multicore.amd.com/es-ES/AMD-Multi-Core/resources/Technology-Evolution.

14. Leopold C., “Parallel and Distributed Computing. A survey of Models, Paradigms, and Approaches”. Wiley, New York. 2001.

15. Andrews G., “Foundations of Multithreaded, Parallel, and Distributed Programming”. Addison Wesley. 2000.