Análisis de rendimiento de la vectorización automática, guiada y explícita en un kernel de detección de bordes mediante Intel Advisor

Juan Pablo Palmero Valdés[1], Mario Rossainz López[2]

Resumen— **La detección de bordes es un kernel numérico fundamental en visión por computador y procesamiento de imágenes. En este trabajo se implementa el detector de bordes por magnitud del gradiente $G[i] = \sqrt{G_x[i]^2 + G_y[i]^2}$ en C++ moderno en cuatro variantes: línea base escalar, vectorización automática del compilador, vectorización guiada (OpenMP SIMD) y vectorización explícita (Intel AVX2). Se analiza el rendimiento y la eficiencia de vectorización con Intel Advisor, reportando tiempos de ejecución y porcentajes de vectorización. Las tres variantes vectorizadas (auto, guiada y explícita) alcanzan 100 % de tiempo en bucle vectorizado según Intel Advisor; la explícita (AVX2) es la más rápida en ejecución directa (aprox. 2,2× frente a la escalar) y la guiada la sigue.**

Palabras clave— **Detección de bordes, magnitud del gradiente, kernel Sobel, SIMD, vectorización automática, OpenMP SIMD, Intel AVX2, Intel Advisor, análisis de rendimiento.**

I. Introducción

La detección de bordes es una operación fundamental en visión por computador, procesamiento de imágenes médicas, percepción robótica y como etapa de preprocesado para redes neuronales convolucionales. Los bordes marcan cambios bruscos de intensidad que suelen corresponder a contornos de objetos y permiten reducir la imagen a información estructural: formas, contornos y regiones. Desde el punto de vista computacional, los bordes se caracterizan por una magnitud del gradiente elevada, por lo que los enfoques estándar utilizan operadores como Sobel, Prewitt o Canny [**?**].

El kernel numérico en el que nos centramos es el cómputo de la magnitud del gradiente: en cada píxel i, la fuerza del borde viene dada por

$$G[i] = \sqrt{G_x[i]^2 + G_y[i]^2}, \tag{1}$$

donde G_x y G_y son los gradientes horizontal y vertical obtenidos convolucionando la imagen con kernels pequeños (p.ej. Sobel 3×3). Este kernel está limitado por cómputo y es muy regular, lo que lo convierte en un buen candidato para la vectorización SIMD en CPUs modernas.

El objetivo de este trabajo es comparar tres estrategias de vectorización—automática (solo compilador), guiada (directivas como OpenMP SIMD) y explícita (intrinsics SIMD escritas a mano)—sobre una implementación en C++ de este kernel, y analizar su rendimiento y eficiencia de vectorización con Intel Advisor. Implementamos una línea base escalar más las tres versiones vectorizadas, las ejecutamos bajo Intel Advisor para obtener tiempos y métricas de vectorización, y presentamos los resultados en este artículo. El documento se organiza así: la Sección II repasa el kernel y los conceptos de vectorización; la Sección III describe la implementación y la metodología basada en Advisor; la Sección IV presenta los resultados experimentales; la Sección V resume las conclusiones y líneas futuras.

II. Antecedentes y trabajo relacionado

A. Detección de bordes por magnitud del gradiente

Un borde se define como un cambio local fuerte en la intensidad de la imagen. Para una imagen 2D discreta, el gradiente se aproxima convolucionando la imagen con kernels derivados. El operador de Sobel utiliza dos kernels 3×3 para calcular G_x (gradiente horizontal) y G_y (gradiente vertical). La magnitud G en la Ec. (1) es entonces una medida escalar de la fuerza del borde en cada píxel. La convolución se implementa como un doble bucle sobre la imagen con una ventana 3×3; la etapa de magnitud es un único recorrido sobre los píxeles. Ambas fases son paralelas en datos y presentan acceso a memoria regular, lo que favorece la SIMD y la auto-vectorización del compilador [**?**].

Los kernels de Sobel para G_x y G_y son:

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \quad K_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}.$$

El Algoritmo 1 resume el cálculo de la magnitud del gradiente a partir de una imagen de entrada I: convolución con K_x y K_y (con replicación de borde o salida de mismo tamaño) y luego cómputo de $G[i] = \sqrt{G_x[i]^2 + G_y[i]^2}$ para cada píxel interior. Tanto la convolución como el bucle de magnitud son inherentemente paralelos en datos: cada píxel de salida se puede calcular de forma independiente, lo que permite aplicar SIMD procesando varios píxeles por instrucción (p.ej. cuatro `double` con AVX2).

B. Estrategias de vectorización

Vectorización automática: El compilador (p.ej. Intel oneAPI `icpx` o GCC con `-O3 -march=native`) intenta mapear operaciones escalares a instrucciones

[1]Lic. en Ciencias de la Computación. Benemérita Universidad Autónoma de Puebla. E-mail: palmerovaldes99@gmail.com.

[2]Doctor en Ciencias de la Computación. Benemérita Universidad Autónoma de Puebla. E-mail: mario.rossainz@correo.buap.mx.

Algoritmo 1 Magnitud del gradiente (Sobel)

Imagen de entrada I de tamaño $H \times W$ Imagen de magnitud G (mismo tamaño en píxeles interiores)

1: Calcular $G_x \leftarrow I * K_x$ (convolución 3×3, borde replicado)
2: Calcular $G_y \leftarrow I * K_y$ cada píxel i en el interior de la imagen
3: $G[i] \leftarrow \sqrt{G_x[i]^2 + G_y[i]^2}$
4: G

SIMD sin modificar el código fuente. La efectividad depende de la estructura del bucle, la alineación y la ausencia de dependencias que impidan la vectorización.

Vectorización guiada (implícita): El programador añade directivas como `#pragma omp simd` para marcar bucles como vectorizables, ayudando al compilador y permitiendo optimizaciones específicas de OpenMP SIMD [**?**].

Vectorización explícita: El programador utiliza intrinsics SIMD (p.ej. Intel AVX/AVX2 `_mm256_*` para vectores de 256 bits) para implementar el kernel a mano [**?**]. Esto da control total y puede alcanzar una utilización alta cuando el compilador o el enfoque guiado no lo logran, a costa de portabilidad y mantenibilidad.

Intel Advisor [**?**] proporciona análisis tipo roofline, asesoramiento sobre vectorización y métricas como cobertura de vectorización y eficiencia del cuerpo del bucle, que utilizamos para interpretar el rendimiento de cada variante.

III. Metodología

A. Implementación escalar

Implementamos el kernel de detección de bordes en C++ moderno (C++17). La versión escalar consta de:

1. Carga de una imagen en escala de grises (formato PGM) en un buffer contiguo de `double`.
2. Convolución de la imagen con los kernels de Sobel G_x y G_y 3×3 con salida de mismo tamaño (los píxeles de borde se omiten o replican). Dos bucles anidados sobre filas y columnas calculan cada píxel de salida con una ventana 3×3 (esténcil de 9 puntos).
3. Cálculo de la magnitud del gradiente $G[i] = \sqrt{G_x[i]^2 + G_y[i]^2}$ en un único bucle sobre todos los píxeles.

No se usan directivas SIMD ni intrinsics; sirve como línea base y como referencia de "sin vectorización" para Intel Advisor.

El Algoritmo 2 detalla el bucle principal de la convolución (un píxel por iteración) y el bucle de magnitud. En la variante escalar ambos bucles son puramente escalares; en las variantes guiada y explícita estos mismos bucles son el objetivo de la vectorización.

Algoritmo 2 Bucle escalar: convolución y magnitud

Imagen I, kernels K_x, K_y 3×3; dimensiones H, W Imagen de bordes G

1: Inicializar G_x, G_y del tamaño de I $y \leftarrow 1$ **to** $H-2$ $x \leftarrow 1$ **to** $W-2$
2: $gx \leftarrow 0$; $gy \leftarrow 0$ $dy \leftarrow -1$ **to** 1 $dx \leftarrow -1$ **to** 1
3: $gx \leftarrow gx + I[y+dy, x+dx] \cdot K_x[dy+1, dx+1]$
4: $gy \leftarrow gy + I[y+dy, x+dx] \cdot K_y[dy+1, dx+1]$
5: $G_x[y,x] \leftarrow gx$; $G_y[y,x] \leftarrow gy$ cada píxel i en el interior
6: $G[i] \leftarrow \sqrt{G_x[i]^2 + G_y[i]^2}$
7: G

B. Versión auto-vectorizada

El mismo código fuente se compila con optimización agresiva y flags específicos de arquitectura: `-O3 -march=native` (con Intel oneAPI se usa `icpx`). No se añaden pragmas ni intrinsics. Se espera que el compilador auto-vectorice los bucles de convolución y magnitud cuando sea legal. Esta variante se denomina "auto" en los resultados.

C. Vectorización guiada (implícita)

Una segunda variante se construye con la misma estructura algorítmica pero con `#pragma omp simd` aplicado al bucle interno de la convolución y al bucle de magnitud. El proyecto se enlaza con OpenMP (`-fiopenmp` en Intel oneAPI). Así se guía al compilador para generar código SIMD en esos bucles y Advisor puede reportar la vectorización guiada.

D. Vectorización explícita

Una tercera variante implementa la convolución y el cómputo de la magnitud usando intrinsics Intel AVX2 (`_mm256_*`). Se procesan cuatro valores `double` por iteración (registros de 256 bits). El bucle interno de la convolución se desenrolla para calcular cuatro píxeles consecutivos cada vez; el resto se trata con un epílogo escalar. La etapa de magnitud usa `_mm256_loadu_pd`, `_mm256_mul_pd`, `_mm256_add_pd` y `_mm256_sqrt_pd`, con cola escalar. Esta variante se compila con `-O3 -march=native` para disponer de FMA y otras instrucciones.

E. Análisis de rendimiento con Intel Advisor

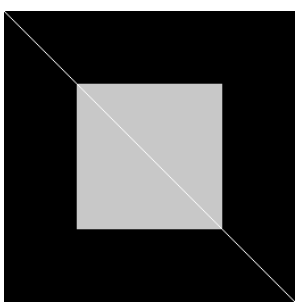
La compilación se realiza con `icpx` y los parámetros necesarios en cada caso: escalar sin auto-vectorización; auto con `-O3 -march=native`; guiada con `-O3 -march=native -fiopenmp`; explícita con `-O3 -march=native`. Para cada variante se ejecutó el binario bajo Intel Advisor (Survey) sobre una imagen de entrada fija y se utilizaron los informes de vectorización para obtener métricas como porcentaje de código vectorizado y eficiencia del bucle. Las imágenes de entrada son PGM en escala de grises; el tamaño y la ruta se mantienen fijos en todas las ejecuciones para que tiempos y métricas sean comparables.

IV. Resultados experimentales

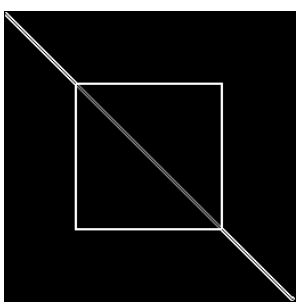
Los experimentos se ejecutaron en una máquina con **1 hilo de CPU** y conjunto de instrucciones **AVX**. La imagen de entrada utilizada en todas las ejecuciones es de **256×256** píxeles (PGM en escala de grises). Los tiempos de la Tabla I se obtuvieron por **ejecución directa** de cada binario (cronómetro alrededor del kernel, mismo criterio en las cuatro variantes). Adicionalmente, cada variante se ejecutó bajo **Intel Advisor** (Survey) para obtener las métricas de vectorización y las capturas de la subsección IV-D.

A. Imágenes procesadas

La Figura 1 muestra la imagen de entrada original y el resultado de la detección de bordes producido por la variante explícita (AVX2). La magnitud del gradiente destaca claramente los contornos principales. Salidas adicionales para las variantes auto, guiada y explícita, así como otras imágenes de prueba (p.ej. `guitar.pgm`, `bee.pgm`), están disponibles en la carpeta `img/` del repositorio.



(a) Entrada original (256×256)



(b) Resultado detección de bordes (AVX2 explícito)

Fig. 1: Ejemplo: imagen original y salida de detección de bordes (variante explícita).

B. Tiempo de ejecución

La Tabla I recoge el tiempo de ejecución en **milisegundos (ms)** para la línea base escalar y las tres versiones vectorizadas. Todas las medidas se tomaron por ejecución directa (sin instrumentación): `high_resolution_clock` alrededor del kernel, igual que en `edge_scalar`; el resultado se convierte a ms (1 ms = 1000 µs).

Tabla I: Tiempo de ejecución por variante (ejecución directa, en ms).

Variante	Tiempo (ms)
Escalar	0,52
Auto	0,34
Guiada (OpenMP SIMD)	0,27
Explícita (AVX2)	0,24

La versión escalar tarda 0,52 ms; la auto, 0,34 ms; la guiada, 0,27 ms; la explícita, 0,24 ms. Las variantes vectorizadas son más rápidas que la escalar; la explícita (AVX2) es la más rápida y mejora respecto de la guiada.

C. Métricas de vectorización de Intel Advisor

Se utilizó Intel Advisor para recoger datos de Survey en cada compilación. La Tabla II resume las métricas de vectorización según el resumen "Vectorization and Code Insights": tiempo de CPU en bucles vectorizados e instrucciones AVX.

Tabla II: Métricas de vectorización de Intel Advisor (tiempo en código vectorizado).

Variante	Vectorización (tiempo en bucle vectorizado)
Escalar	0 %
Auto	100 %
Guiada	100 %
Explícita	100 %

Para la versión escalar, Advisor reporta 0 % de tiempo en código vectorizado. Para las variantes auto, guiada y explícita, el 100 % del tiempo de CPU medido se invirtió en un único bucle vectorizado (AVX). Las capturas del Advisor confirman que las tres variantes vectorizadas alcanzan la vectorización del bucle crítico.

D. Capturas de Intel Advisor

Las Figuras 2–4 muestran el resumen "Vectorization and Code Insights" de Intel Advisor para cada variante. En las tres (auto, guiada y explícita) el 100 % del tiempo de CPU corresponde a un único bucle vectorizado (AVX); solo la variante escalar tiene 0 % en código vectorizado.

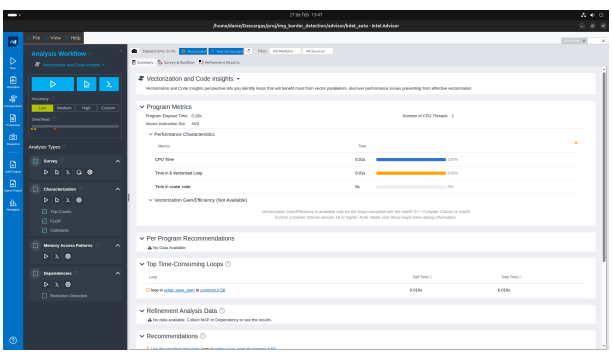


Fig. 2: Resumen Intel Advisor: variante auto-vectorizada (`bdet_auto`).

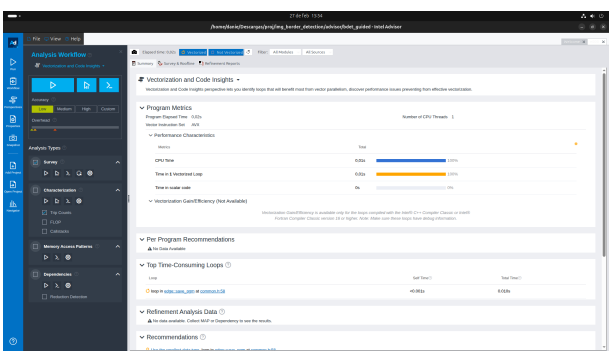


Fig. 3: Resumen Intel Advisor: variante guiada (OpenMP SIMD) (`bdet_guided`).

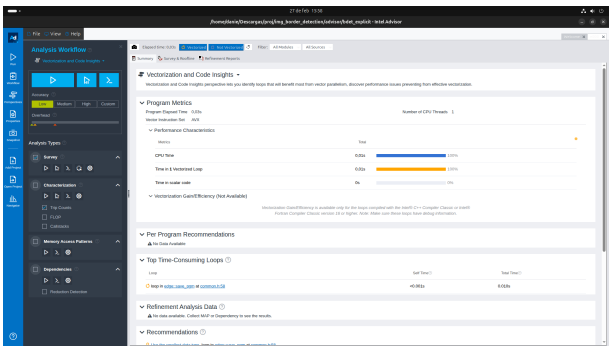


Fig. 4: Resumen Intel Advisor: variante explícita (AVX2) (`bdet_explicit`).

E. Aceleración y comparación

- Tiempos por ejecución directa (ms): escalar 0,52; auto 0,34; guiada 0,27; explícita 0,24.
- Aceleración frente a escalar: auto 1,5×; guiada 1,9×; explícita 2,2×.
- Mejora de la explícita respecto de la guiada: aproximadamente 1,1× (0,27 ms frente a 0,24 ms).

Además de estas mediciones directas, cada variante se analizó con Intel Advisor para las métricas de vectorización (Tabla II y figuras anteriores). Advisor reporta 100 % de tiempo en un bucle vectorizado (AVX) para las variantes auto, guiada y explícita; la escalar queda en 0 %. Las diferencias de tiempo (0,34 ms, 0,27 ms, 0,24 ms) reflejan el mayor grado de control y optimización de la guiada y la explícita frente a la auto-vectorización del compilador.

V. Conclusiones

En este trabajo se ha implementado el kernel de detección de bordes por magnitud del gradiente $G[i] = \sqrt{G_x[i]^2 + G_y[i]^2}$ en C++ moderno en cuatro variantes: línea base escalar, vectorización automática del compilador, vectorización guiada (OpenMP SIMD) y vectorización explícita (Intel AVX2). Se ha analizado el rendimiento y la eficiencia de vectorización con Intel Advisor.

Los tiempos de ejecución directa (en ms) son: escalar 0,52; auto 0,34; guiada 0,27; explícita 0,24. La variante explícita (AVX2) es la más rápida, con una aceleración de aproximadamente 2,2× frente a la escalar y 1,1× frente a la guiada. Con Intel Advisor se confirmó que las tres variantes vectorizadas (auto, guiada y explícita) alcanzan 100 % de tiempo en un bucle vectorizado (AVX); solo la escalar queda en 0 %.

El porcentaje de vectorización reportado por Intel Advisor es 0 % para escalar y 100 % para auto, guiada y explícita. Las capturas de Advisor confirman que las tres variantes vectorizadas dedican el 100 % del tiempo de CPU medido a un único bucle vectorizado; las diferencias de rendimiento entre ellas (0,34 ms, 0,27 ms, 0,24 ms) se deben al mayor grado de optimización de la guiada (OpenMP SIMD) y la explícita (AVX2) frente a la auto-vectorización del compilador.

En resumen, las tres variantes vectorizadas (auto, guiada y explícita) mejoran frente a la línea base escalar; la explícita AVX2 es la más rápida y ofrece el máximo control a costa de portabilidad. Como trabajo futuro se podría considerar el uso de imágenes de mayor tamaño, otros kernels (p.ej. Canny completo) o la comparación con implementaciones en GPU.

Agradecimientos

Agradecemos a la Benemérita Universidad Autónoma de Puebla (BUAP) y al CONACYT por el apoyo institucional. Asimismo, nuestro agradecimiento a los profesores de la Facultad de Ciencias de la Computación de la BUAP por su orientación y apoyo durante la realización de este trabajo.