PARALELISMO A NIVEL DE PROCESO

Práctica 4

Paralelización a nivel de proceso de la solución aportada al problema presentado en la práctica2. Análisis del algoritmo planteado y propuesta de las partes a paralelizar. Comparación de resultados con la paralelización a nivel de hilo.

Vicente Martín Rueda

Marcos González Verdú

Pablo Requena González

25/12/2015

PARALELISMO A NIVEL DE PROCESO

Práctica 4

CONOZCAMOS MPI

Es una implementación de la interfaz de paso de mensajes MPI. OpenMPI se caracteriza por su alta eficiencia y prestaciones para la ejecución en entornos distribuidos (clústeres de ordenadores).

Sus principales características son:

- Implementa el estándar MPI.
- Permite la distribución de procesos de forma dinámica.
- Alto rendimiento.
- Tolerancia a fallos: capacidad de recuperarse de forma transparente de los fallos de los componentes (errores en el envío o recepción de mensajes, fallo de un procesador o nodo).
- Soporta redes heterogéneas: permite la ejecución de programas en redes cuyos ordenadores presenten distinto número de nodos y de procesadores.
- Una única librería soporta todas las redes.
- Portable: funciona en los sistemas operativos Linux, OS-X, Solaris y en un futuro próximo en Windows.
- Modificable por los instaladores y usuarios finales: presenta opciones de configuración durante la instalación de la API, la compilación de programas y su ejecución.

Comandos

Vamos a explicar qué hacen los comandos más importantes y usados.

mpdboot

Lanza el demonio encargado de crear el anillo o clúster entre las máquinas especificadas en el archivo "*mpd.hosts*".

mpdtrace

Comprueba el estado del anillo o clúster con el parámetro "*–l*". Si todo funciona correctamente, se listarán las máquinas encargadas de llevar a cabo el anillo. Este comando se ejecuta en la máquina donde se haya ejecutado el demonio "*mpdboot*".

mpdallexit

Se usa para destruir el anillo, es decir, finaliza el demonio en todos los nodos del anillo.

Código

Los cambios realizados para la realización de esta práctica, en un primer lugar, crear los flujos de control del ordenador en el cual se está trabajando, para que el proceso padre haga un trabajo, y los procesos esclavos o hijos hagan otro distinto. El proceso padre, iniciará la comunicación MPI, además, separará la imagen en sectores, igual que hacíamos en prácticas anteriores, solo que en este caso, en vez de llamar al módulo de procesar la imagen para pixelizarla y realizar todo el proceso de las prácticas anteriores, cogerá los segmentos, que se almacenarán en un array de imágenes, y ese será su papel principal, por ahora.

El otro flujo de control es el que controla a los procesos esclavos o hijos, que básicamente reciben el array correspondiente, e irán procesando las imágenes con el módulo que se ha utilizado en las anteriores prácticas, aunque ha sido modificado, ya que ahora trabajamos con las imágenes de una forma distinta.

Antes el módulo recibía la imagen, el ancho, el largo, el tamaño del píxeles por pixel grande que queremos crear, además de posición en la cual estamos trabajando, ya que antes necesitábamos saber cuál era la primera posición del sector, para no salirnos de él. Ahora, el modulo solo recibe la imagen, y el tamaño del píxeles por pixel grande que queremos crear, donde los controles se hacen ahora con este último dato, que será el rango, hará de igual manera de ancho y alto de la imagen, puesto que ahora trabajamos con subsectores de la imagen, y no con la imagen original.

```
Mat imagenfinal;
imagenfinal.create(superAncho, superAlto, superImagen.depth())
int auxAn=0, auxAl=0;
for(int i=0; i<aux1; i++)
{
    for(j=0; j<tamBigPix; j++)
    {
        for(k=0; k<tamBigPix; k++)
        {
            imagenfinal.at<cv::Vec3b>(auxAn, auxAl)=imagenesEsclavo1[i].at<cv::Vec3b>(j, k);
            auxAn++;
        }
        auxAl++;
        auxAn=0;
    }
}

for(int i=0; i<aux2; i++)
{
    for(j=0; j<tamBigPix; j++)
    {
        for(k=0; k<tamBigPix; k++)
        {
            imagenfinal.at<cv::Vec3b>(auxAn, auxAl)=imagenesEsclavo2[i].at<cv::Vec3b>(j, k);
            auxAn++;
        }
        auxAl++;
        auxAn=0;
    }
}

for(int i=0; i<aux3; i++)
{
    for(j=0; j<tamBigPix; j++)
    {
        for(k=0; k<tamBigPix; k++)
        {
            imagenfinal.at<cv::Vec3b>(auxAn, auxAl)=imagenesEsclavo3[i].at<cv::Vec3b>(j, k);
            auxAn++;
        }
        auxAl++;
        auxAn=0;
    }
}
imwrite(argv[i+1], imagenfinal); // Escribimos la imagen que queremos guardar.
cout << "Procesada la imagen: " << argv[i] << endl; // Informamos al usuario.
```

Aquí una prueba de como rellenamos la imagen nueva, copiando cada posición de cada imagen del array correspondiente.

RESULTADOS DE RENDIMIENTO

Los resultados obtenidos mediante la paralelización a nivel de proceso son los siguientes:

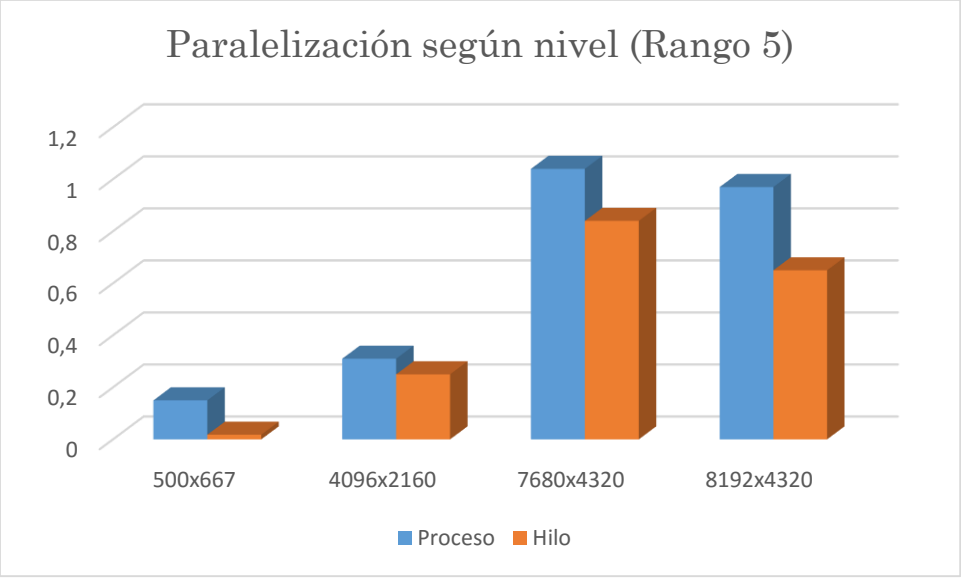
PC-2	Tamaño	5 (s)	25 (s)	50 (s)	100 (s)
fea.jpg	500x667	0.15	0.13	0.11	0.09
pajaro4k	4096x2160	0.31	0.27	0.25	0.21
car8k	7680x4320	1.04	1.02	0.99	0.96
earth8k	8192x4320	0.97	0.96	0.91	0.85
Total Programa		2.47	2.38	2.26	2.11

Podemos ver los resultados obtenidos según tamaño de imagen y el rango de pixels seleccionado, ya que los datos entre diferentes máquinas eran muy cercanos.

COMPARATIVA

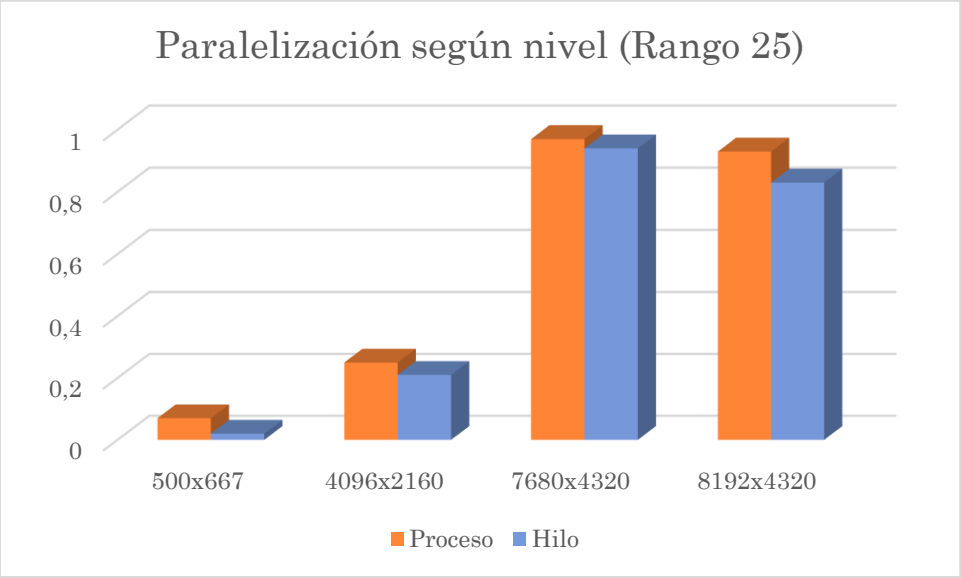
Vamos a realizar las comparativas acordes a nuestros diferentes rangos de pixeles que son 5, 25, 50 y 100.

Rango 5



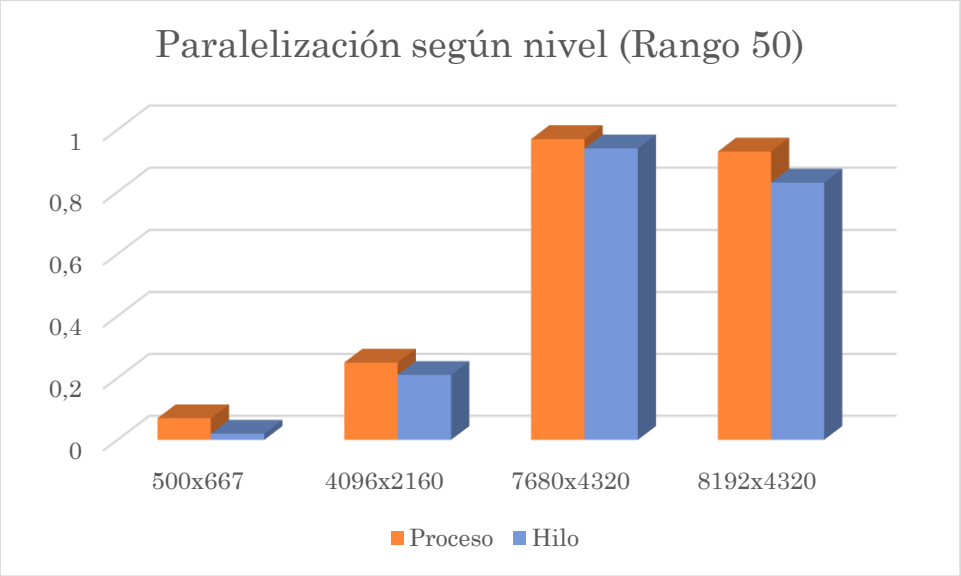
Como se puede ver, tenemos una gran pérdida de tiempo, debido a esta paralelización. Suponemos que la pérdida se deba al tiempo invertido en mover los datos de la aplicación por el anillo.

Rango 25



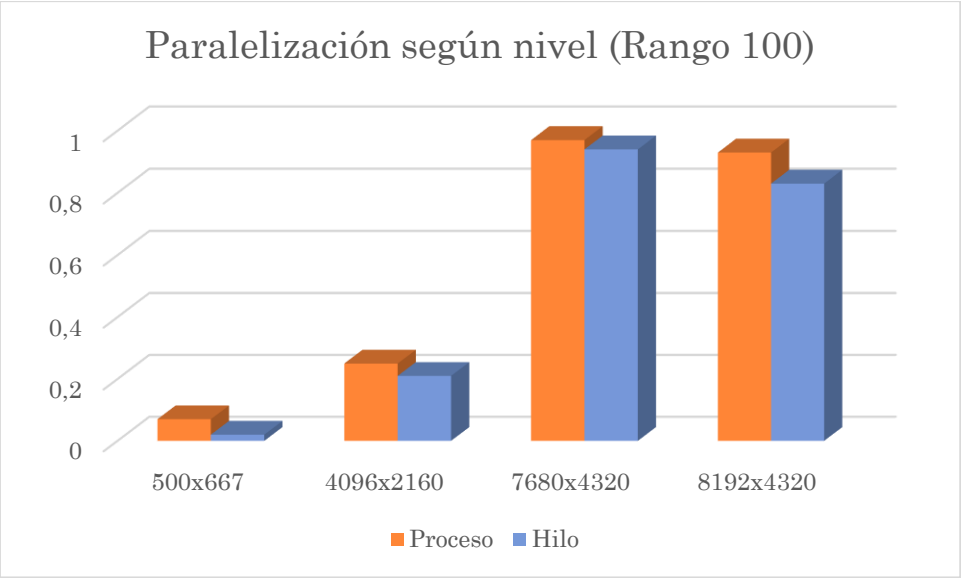
Al aumentar el rango, se ve que tenemos una pérdida algo menor, pero sigue siendo demasiado extensa y obviamente nos sigue sin ser beneficioso para nuestro proyecto.

Rango 50



En este rango la pérdida es un punto intermedio entre los dos rangos anteriores.

Rango 100



Y por último, el rango más grande que tenemos. En este, la pérdida es la mínima posible pero sigue teniendo una pérdida.

Conclusión

Una vez vistos los resultados y la comparativa con la paralelización a nivel de hilo, podemos observar que existe un proceso de mejora según el problema es cada vez mayor.

Por desgracia, nuestro proyecto si hiciésemos el problema mayor no tendría sentido, porque se verían pixeles demasiado grandes y no sería bonito ver una fotografía de alguien transformada en una fotografía con tan solo 40 cuadrados de colores.

Así que, en definitiva, pensamos que si existiesen fotografías mucho mayores, o paralelizásemos las fotografías en lugar de los rangos, podríamos hablar de éxito con este método de paralelización.