Algoritmos paralelos sobre Clusters de Multicores. Análisis de aplicaciones con descomposición funcional y de datos.

Tesina de grado

Alumna: Fabiana Yael Leibovich
Director: Dr. Marcelo Naiouf
Codirector: Dra. Laura De Giusti

Objetivos

- Analizar, investigar y desarrollar algoritmos de scheduling para ejecutar eficientemente aplicaciones de procesamiento paralelo sobre arquitecturas *multicore* y *clusters* de *multicore y* estudiar el comportamiento de las aplicaciones con paralelismo funcional y de datos en este tipo de arquitecturas.

- Manejar la distribución de procesos en los *cores* desde la aplicación para obtener ganancia de *performance*.

- Comparación de las técnicas de asignación automática (Sistema Operativo) y manual de hilos/procesos a *cores*/procesadores.

Agenda

- Contexto
  - Antecedentes de la arquitectura *multicore*
  - Arquitectura *multicore*
  - Arquitectura *Cluster* de *multicore*
  - Jerarquía de memoria
  - Modelos de programación paralela
  - Estrategias de descomposición
- Asignación de tareas a núcleos (automática y manual)
- Caso de estudio:BASIZ
- Soluciones implementadas
- Resultados experimentales
- Conclusiones y líneas futuras de investigación

Contexto: Antecedentes de la arquitectura *multicore*

Desde los orígenes del primer procesador se ha buscado **incrementar la capacidad de cómputo y el rendimiento** de los mismos **aumentando la potencia de cómputo** y al mismo tiempo **disminuyendo el tamaño de los circuitos integrados**.

Problemas: la mejora no es ilimitada

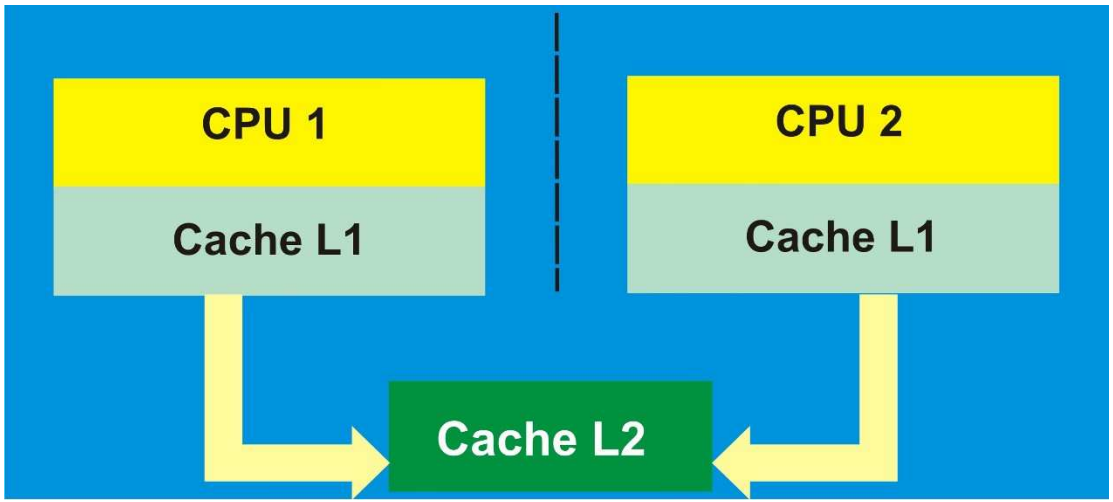
- Límite de construcción del silicio, material con el cual están construidos los procesadores.
- Temperatura imposible de disipar.

Consecuencias: **Surgimiento de la arquitectura *multicore***

Contexto: Arquitectura *multicore*

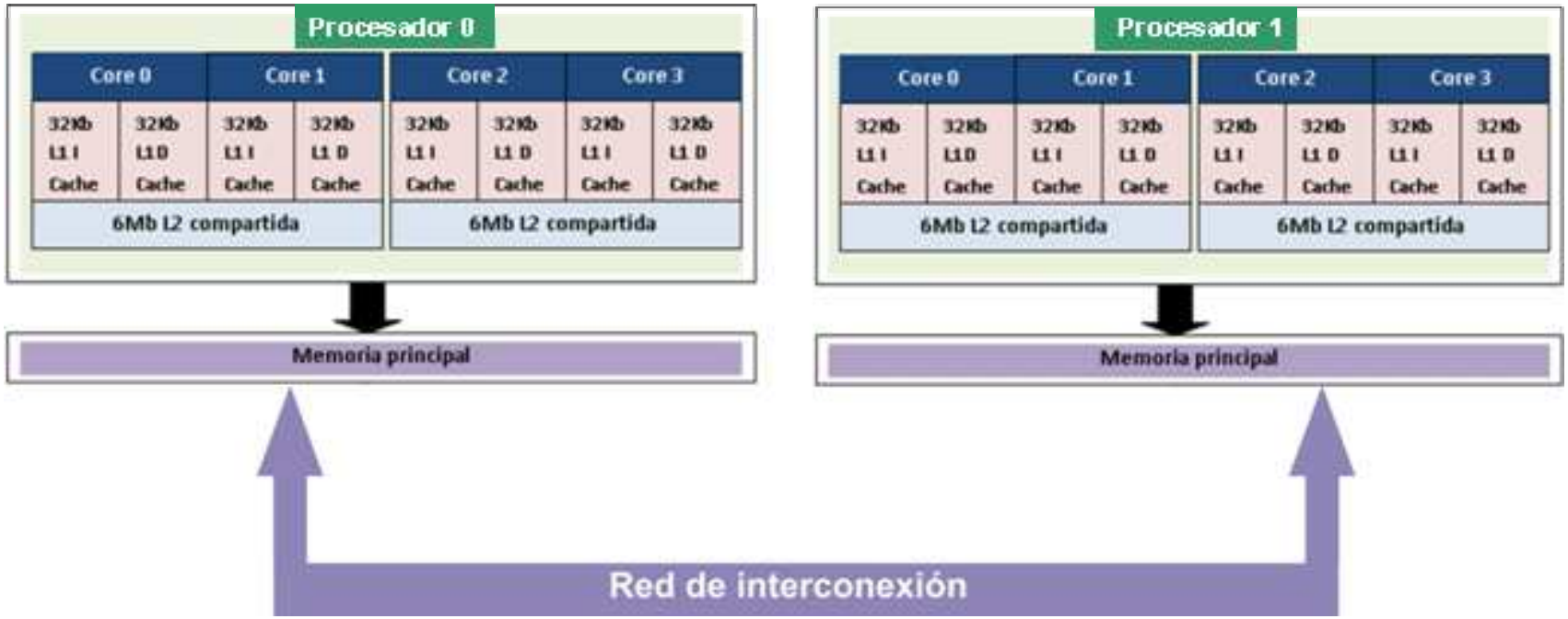
Consiste en un conjunto de núcleos de procesamiento integrados en un único chip.

Normalmente, cada uno de los núcleos (core) posee su propio nivel L1 de cache y de a pares comparten el nivel L2 de cache (incluido generalmente en la placa madre).



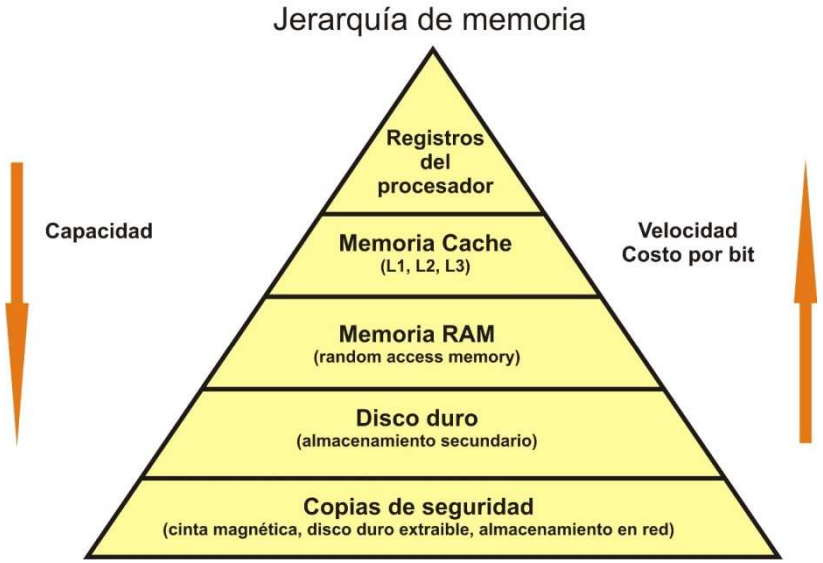
Contexto: Arquitectura *Cluster* de *multicore*

Es una colección de procesadores *multicore* interconectados mediante una red, en la que trabajan cooperativamente juntos como un único recurso de cómputo. Es decir, es similar a un *cluster* tradicional pero con la diferencia de que cada nodo posee un procesador *multicore* en lugar de un monoprocesador.



Contexto: Jerarquía de memoria

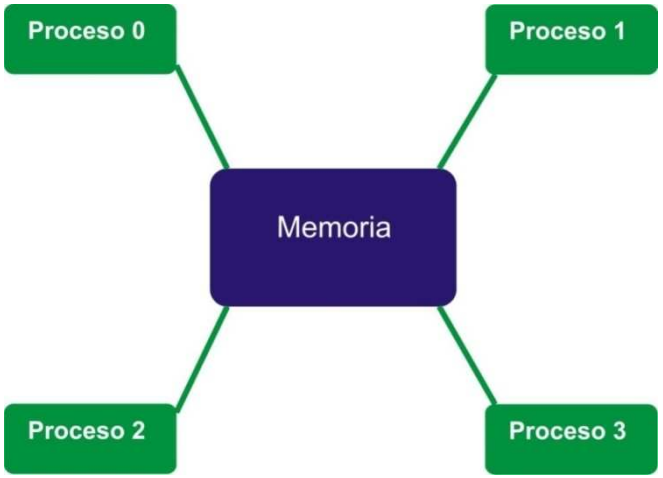
La *performance* de la jerarquía de memoria está determinada por dos parámetros de *hardware*: latencia de la memoria y el ancho de banda de la misma.



Multicore	*Cluster* de *Multicore*
•Niveles de registros •L1 propio de cada núcleo •*Cache* compartida de a pares de núcleos (L2) •Memoria compartida entre los *cores* de un procesador *multicore*	•Niveles de registros •L1 propio de cada núcleo •*Cache* compartida de a pares de núcleos (L2) •Memoria compartida entre los *cores* de un procesador *multicore* •Memoria distribuida accesible vía red

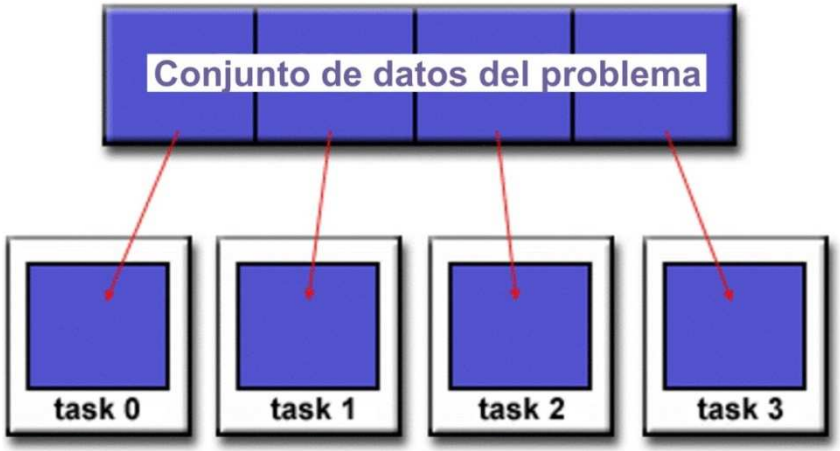
Contexto: Modelos de programación paralela

Memoria compartida

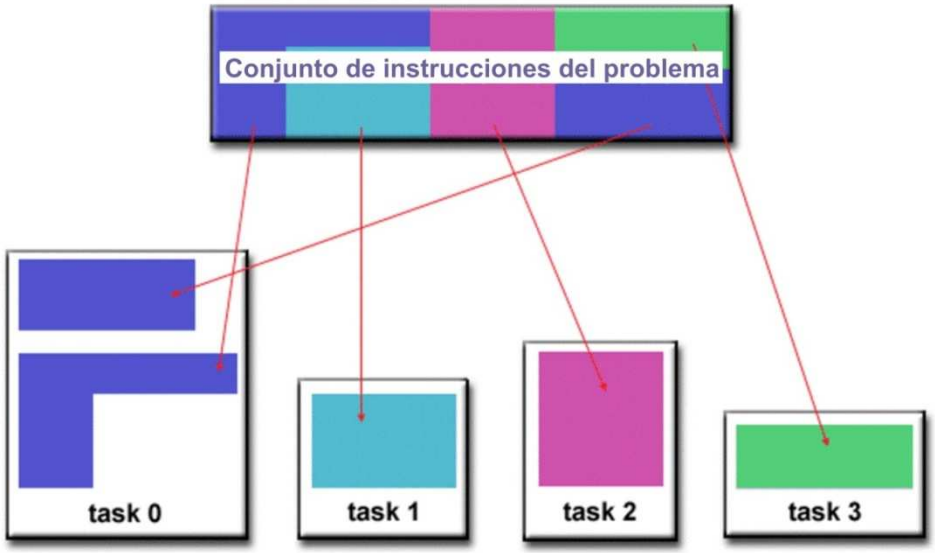


Pasaje de mensajes

Contexto: Estrategias de descomposición



Paralelismo de datos



Paralelismo funcional

Asignación de tareas a núcleos: *Mapping*

- **Mapping del sistema operativo (automático)**

Si bien el sistema operativo intenta no cambiar los procesos o hilos de *cpu* debido a los aciertos de *cache* (es decir, tomando en cuenta la jerarquía de memoria), utilizando la llamada al sistema *sched_getcpu()*, se llevaron a cabo pruebas en las cuales se pudo determinar:

- Los procesos o hilos cambian continuamente de *cpu* sobre la cual se ejecutan. Esto implica un *overhead* significativo ya que el contexto de los mismos debe migrarse a medida que cambian de núcleo de ejecución.
- Para mejorar la performance de una aplicación será necesario pensar en la manera de realizar el *mapping* manualmente desde la misma.

Asignación de tareas a núcleos (cont.)

- **Mapping manual**
  - **Memoria compartida (alternativa 1)**

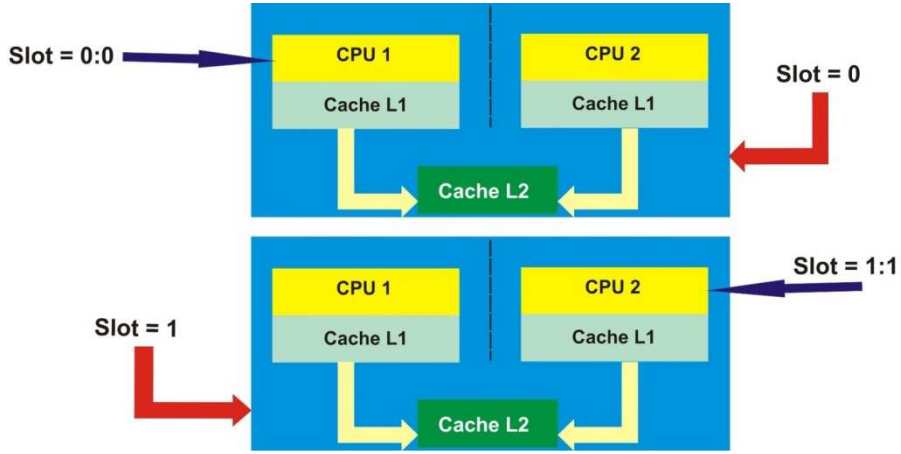
Los sistemas operativos basados en *Unix* proveen una llamada al sistema que permite definir la afinidad explícitamente. Ésta debe ser utilizada en ambientes con memoria compartida ya que puede administrar los núcleos y procesadores de la máquina sobre la cual está corriendo el sistema operativo.

*sched_setaffinity(pid_t pid, unsigned long cpusetsize, cpu_set_t *mask)*

Asignación de tareas a núcleos (cont.) *Mapping* manual

- **Memoria distribuida: Pasaje de mensajes (alternativa 2)**

- En el archivo ***hostfile*** se define el número de núcleos disponibles en el sistema y el nombre de la máquina dentro de la red. El formato del archivo es el siguiente:
  - *hostNameX slots = nro de núcleos*
  - *hostNameY slots = nro de núcleos*
- En el archivo ***rankfile*** se define una entrada por cada proceso de la siguiente manera:
  - *rank N = hostNameX slot =* ***nro de cpu***
  - *rank M = hostNameY slot =* ***nroSocket:nroNúcleo***



Ventajas y desventajas

Ventajas y desventajas alternativa 1 (system call)

- La ventaja de la utilización de la system call en memoria compartida es que podemos ir modificando el núcleo asignado a un proceso/hilo dinámicamente.
- La desventaja es que el código de nuestra aplicación debe ser modificado para agregar las directivas necesarias y la ejecución de ese código consumirá tiempo de *cpu*.

Ventajas y desventajas alternativa 2 (OPENMPI)

- La ventaja que ofrece OPENMPI es que no es necesario modificar el código de nuestra aplicación para realizar la asignación manual.
- La desventaja es que la asignación es estática y no puede modificarse durante la ejecución.

Caso de estudio: BASIZ

Es una aplicación de procesamiento de imágenes para la identificación y detección de las zonas con más brillo e intensidad de color de una imagen, zonas más perceptibles al ojo humano.

- Está dividida en 7 fases de procesamiento:
    1) **Separación** en los tres colores básicos (rojo, verde y azul).
    2) **Difuminado** de cada uno de los tres canales de color básico (bluring).
    3) **Suma** de las imágenes difuminadas para cada canal de color.
    4) **Unión** de los tres canales de color difuminados.
    5) **Conversión** de formato de representación de la imagen difuminada de RGB a HSV (matiz, saturación y brillo).
    6) **Umbralización**: obtener el umbral a partir del cual se determina la inclusión de los píxeles a un segmento sensitivo.
    7) **Detección** y marcado de las zonas más sensitivas al ojo humano.

Solución secuencial

Separación en los tres canales de color

Paralelización para cada canal de color ya que no hay dependencia de datos.

Difuminado de los tres canales de color

Suma de las imágenes difuminadas para cada canal de color

Unión de las tres imágenes difuminadas

Conversión formato representación (de RGB a HSV)

Umbralización

Detección

Soluciones implementadas

- **Solución secuencial**
- **Soluciones paralelas**

Soluciones que utilizan paralelismo funcional (pasaje de mensajes)

BASIZ-Pipe-5
BASIZ-Pipe-8
BASIZ-Gauss-subdividido

Soluciones que utilizan paralelismo de datos

BASIZ-Paralelismo datos MPI
BASIZ-Paralelismo datos OpenMP
BASIZ-Paralelismo datos OpenMP 2 fases

Soluciones que utilizan paralelismo funcional y de datos

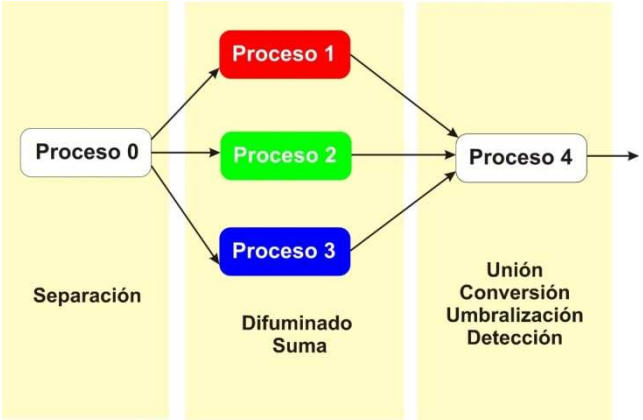
BASIZ-MW-Replicado

Soluciones que utilizan paralelismo funcional (memoria compartida y pasaje de mensajes)

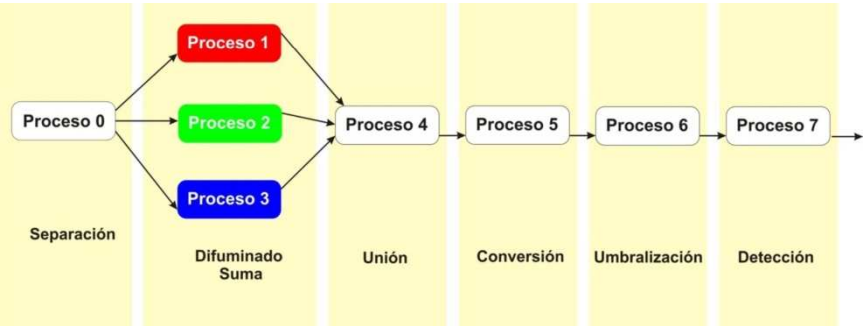
BASIZ-Pipe-5 MPI y Pthreads
BASIZ-Pipe-5 MPI y OpenMP

En todos los casos: Se utilizó el lenguaje C sin utilizar librerías destinadas al procesamiento de imágenes.

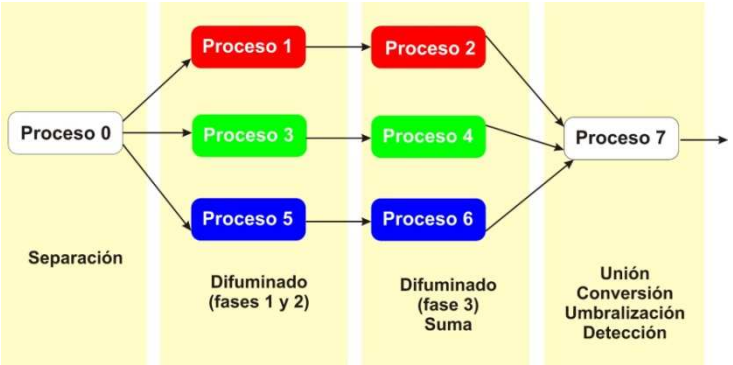
Soluciones que utilizan paralelismo funcional
(pasaje de mensajes)



BASIZ-Pipe-5

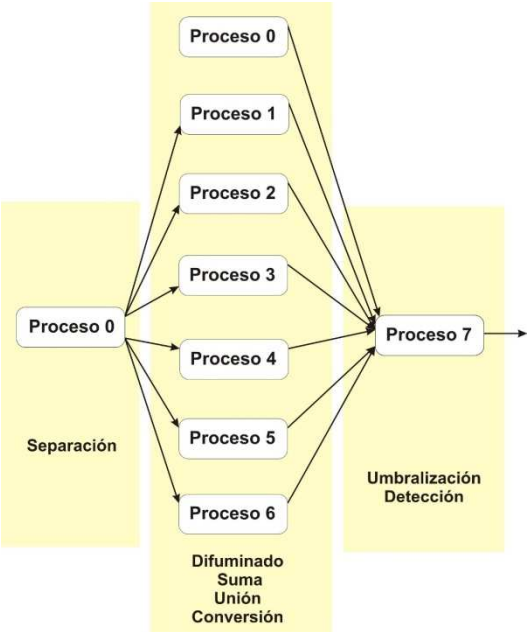


BASIZ-Pipe-8

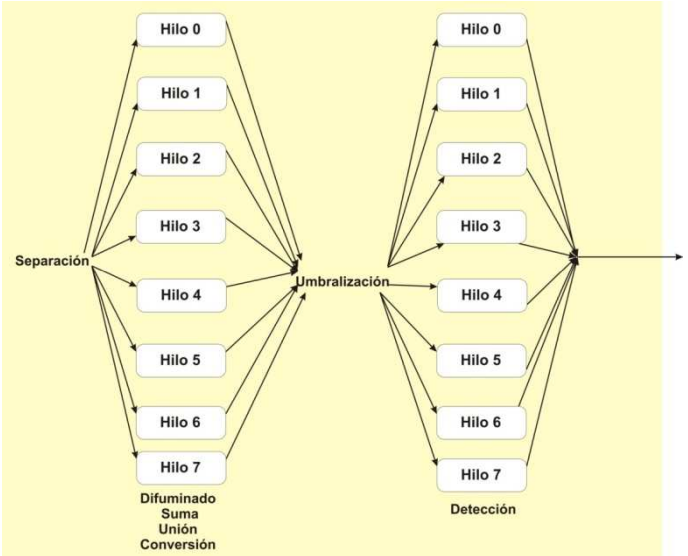


BASIZ-Gauss-subdividido

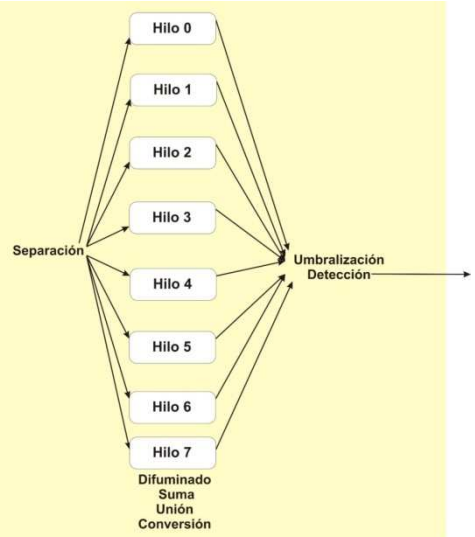
Soluciones que utilizan paralelismo de datos



BASIZ-Paralelismo datos MPI



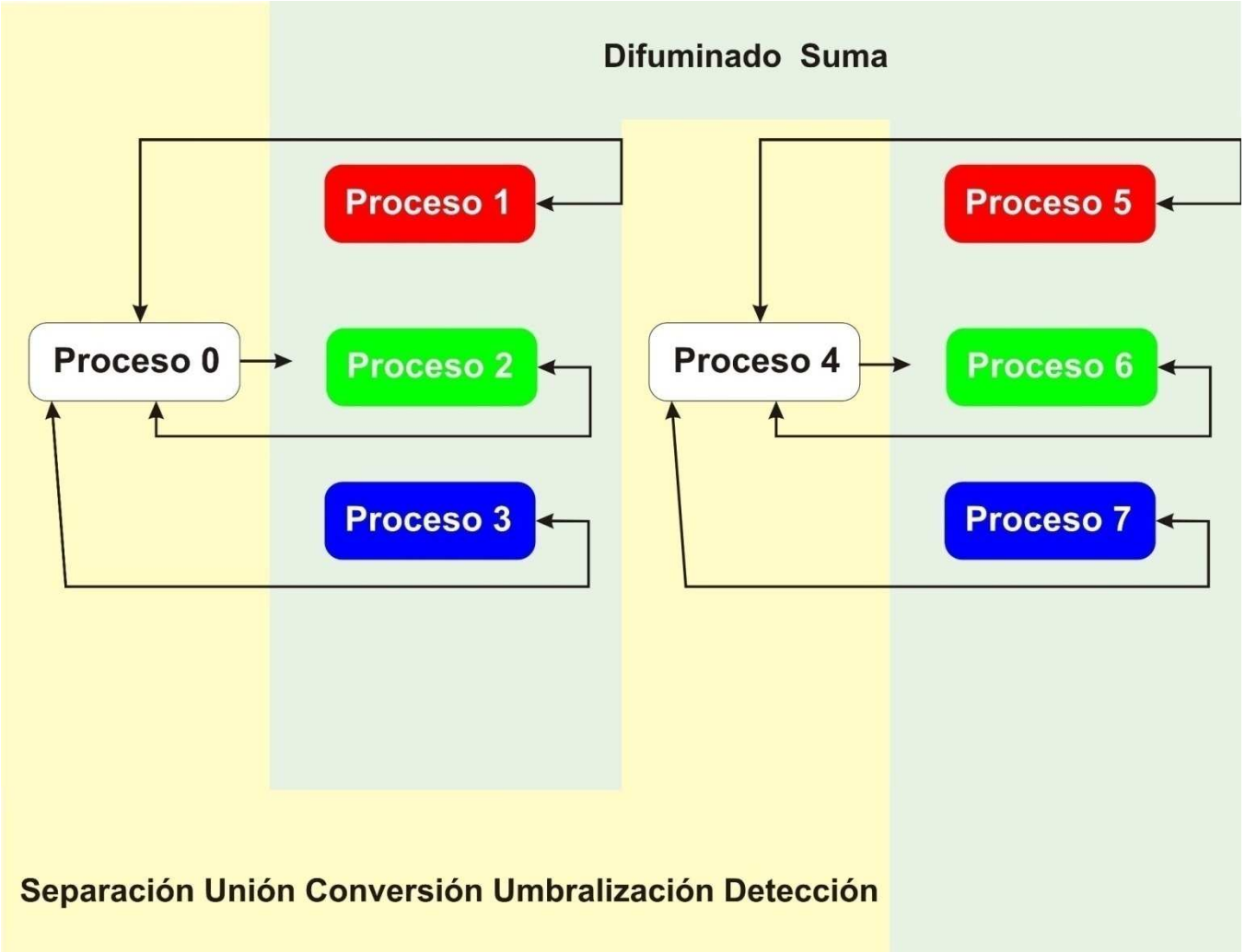
BASIZ-Paralelismo datos OpenMP 2 fases



BASIZ-Paralelismo datos OpenMP

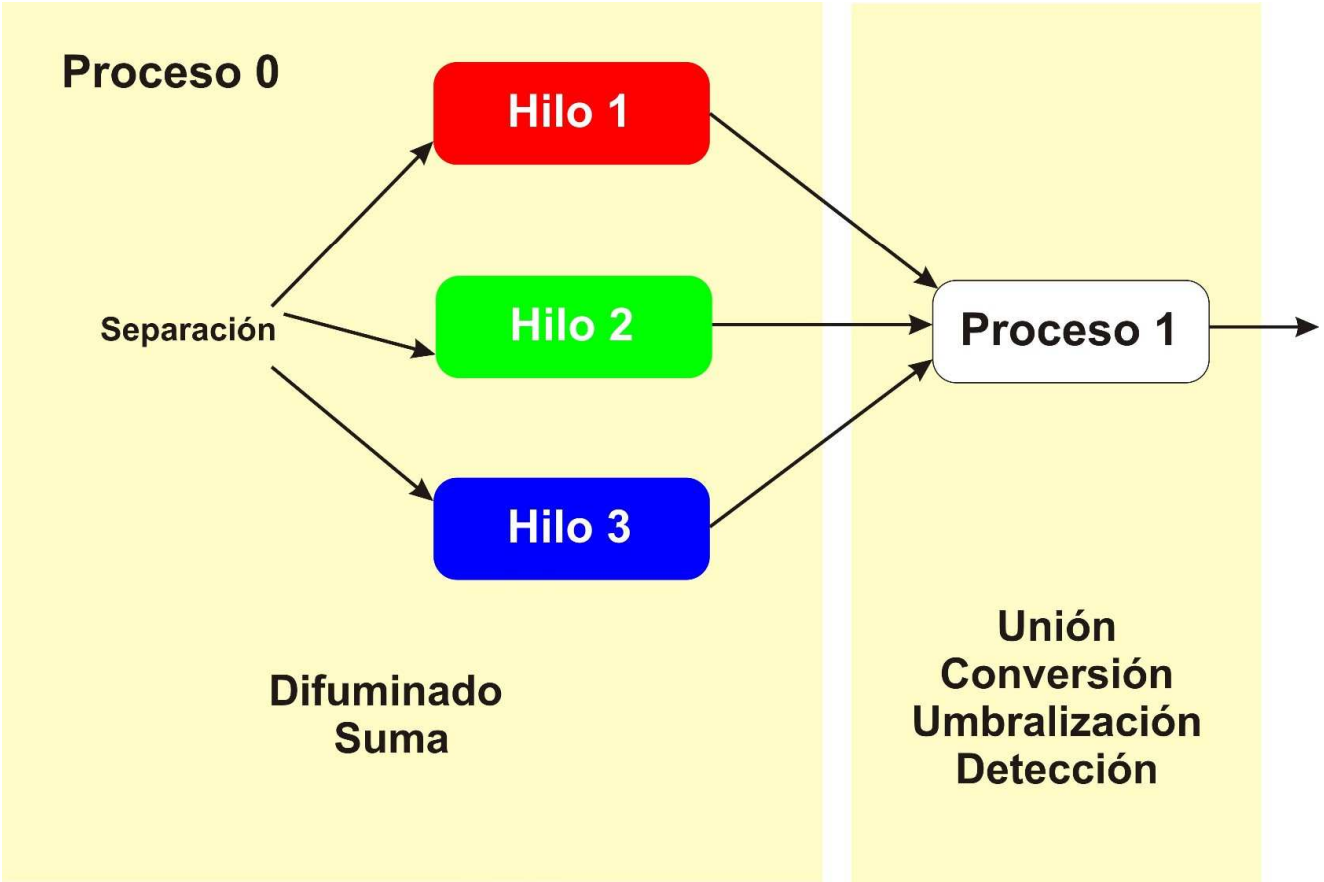
Soluciones que utilizan paralelismo funcional y de datos

BASIZ-MW-Replicado



Soluciones que utilizan paralelismo funcional
(memoria compartida y pasaje de mensajes)

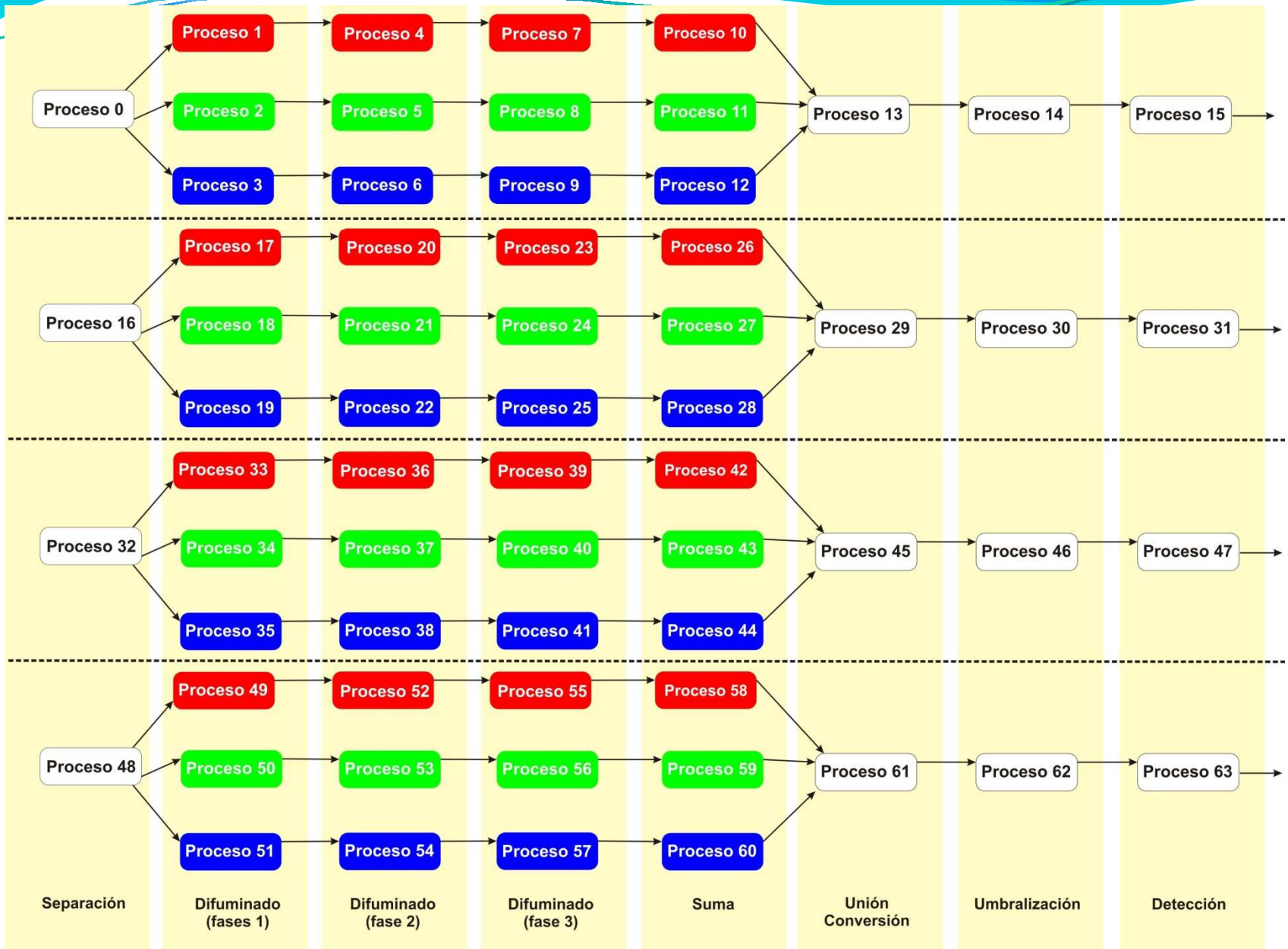
BASIZ-Pipe-5 MPI y OpenMP / BASIZ-Pipe-5 MPI y Pthreads



Soluciones implementadas en arquitectura *multicore*

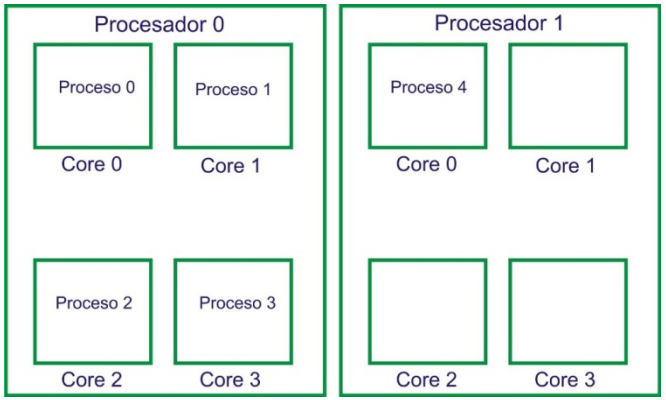
Paralelismo funcional	Paralelismo de datos	Paralelismo funcional y de datos (híbrido)
BASIZ-Pipe-5	BASIZ-Paralelismo datos MPI	BASIZ-MW-Replicado
BASIZ-Pipe-8	BASIZ-Paralelismo datos OpenMP	
BASIZ-Gauss-subdividido	BASIZ-Paralelismo datos OpenMP 2 fases	
BASIZ-Pipe-5 MPI y Pthreads		
BASIZ-Pipe-5 MPI y OpenMP		

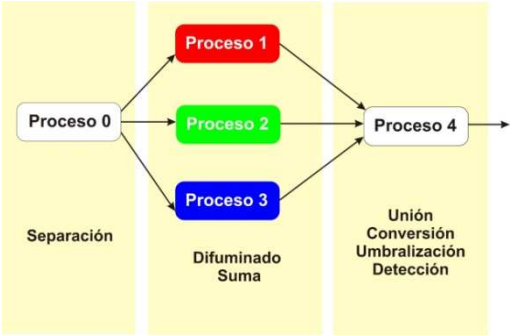
Filtro gaussiano subdividido utilizando 64 núcleos



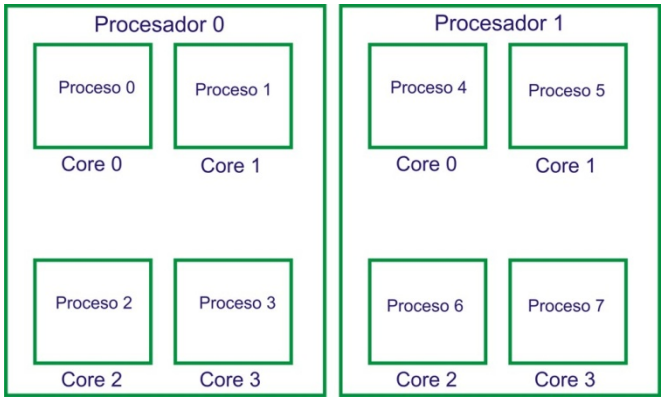
Soluciones que utilizan paralelismo funcional (pasaje de mensajes)

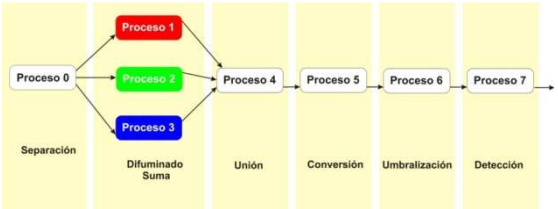
BASIZ-Pipe-5



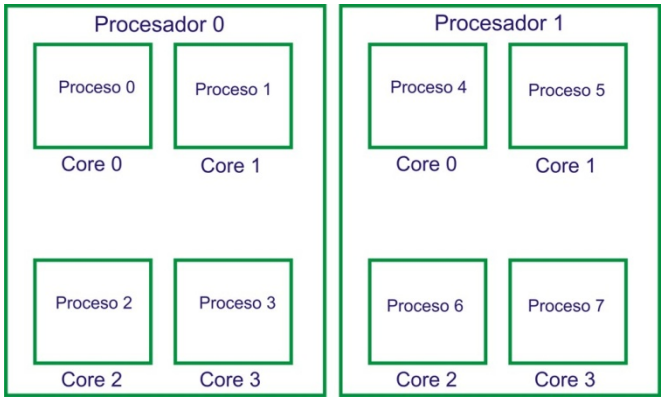


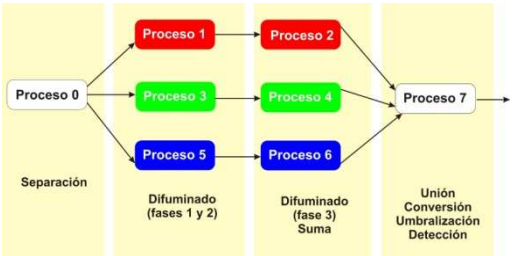
BASIZ-Pipe-8





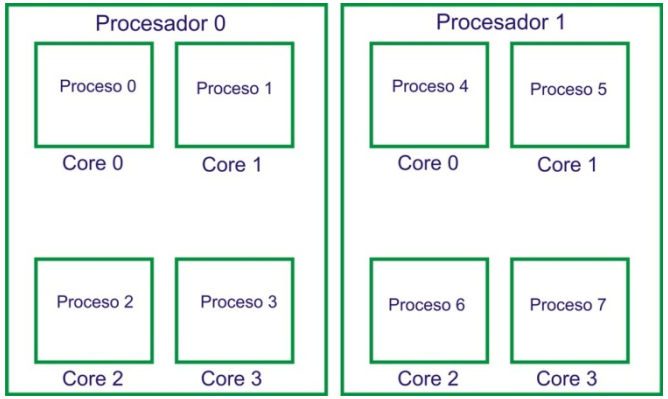
BASIZ-Gauss-subdividido

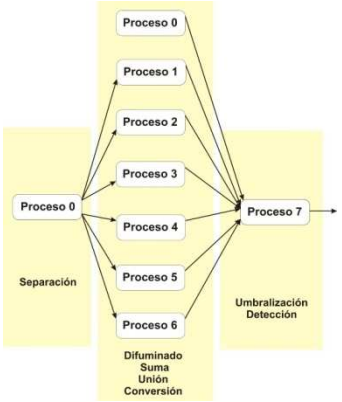




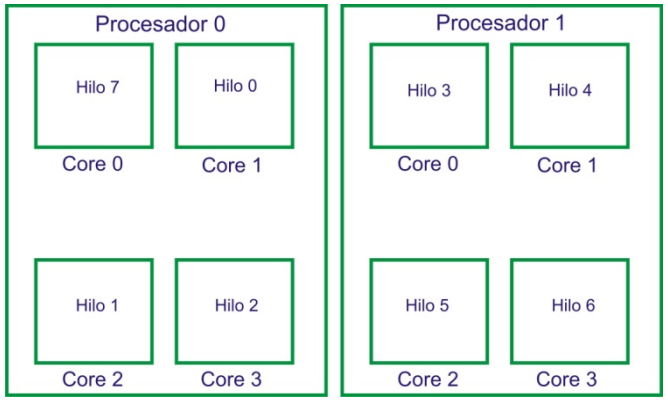
Soluciones que utilizan paralelismo de datos

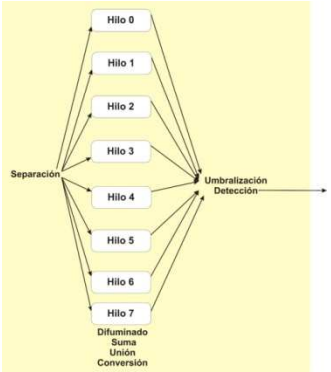
BASIZ-Paralelismo datos MPI



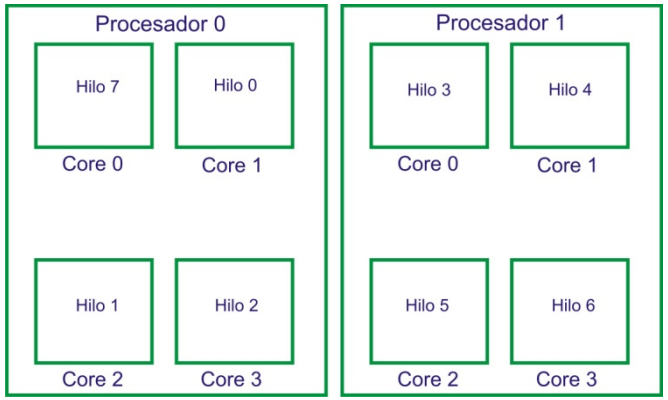


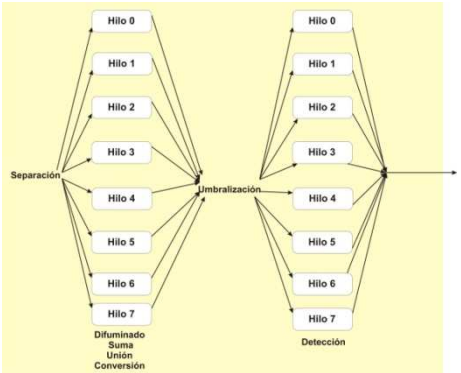
BASIZ-Paralelismo datos OpenMP





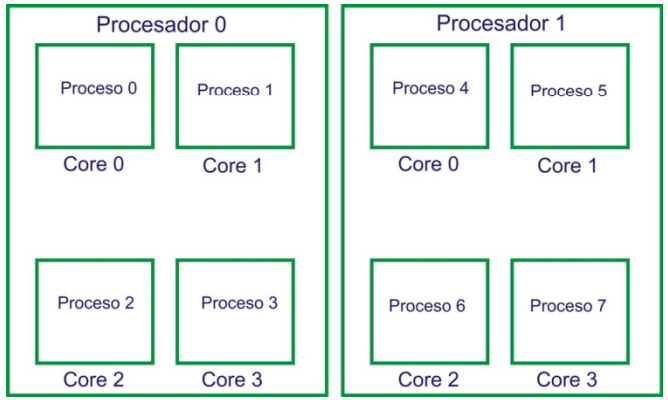
BASIZ-Paralelismo datos OpenMP 2 fases

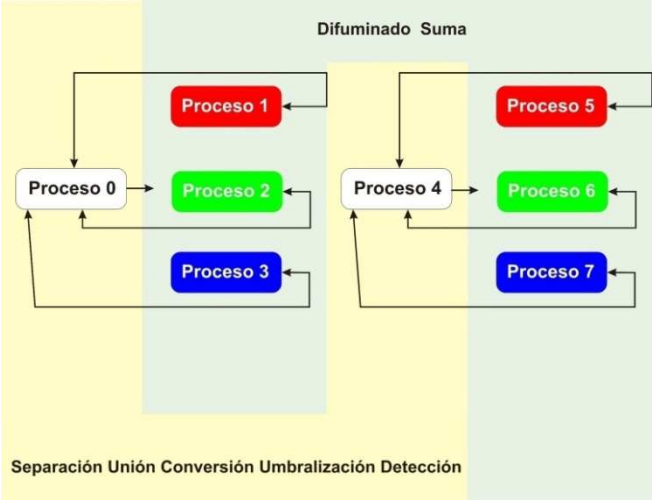




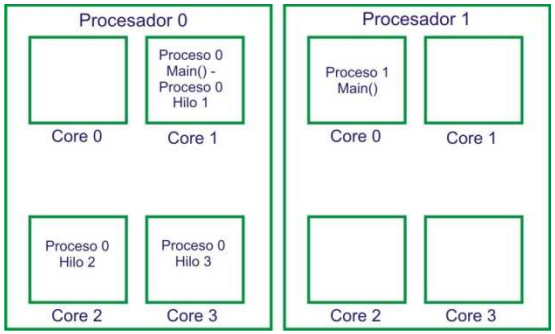
Soluciones que utilizan paralelismo funcional y de datos

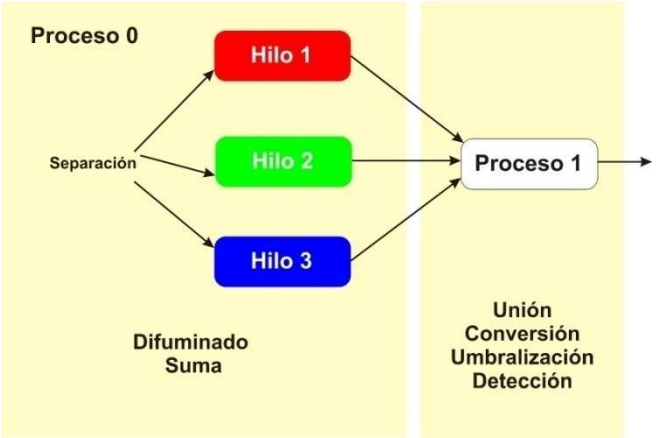
BASIZ-MW-Replicado





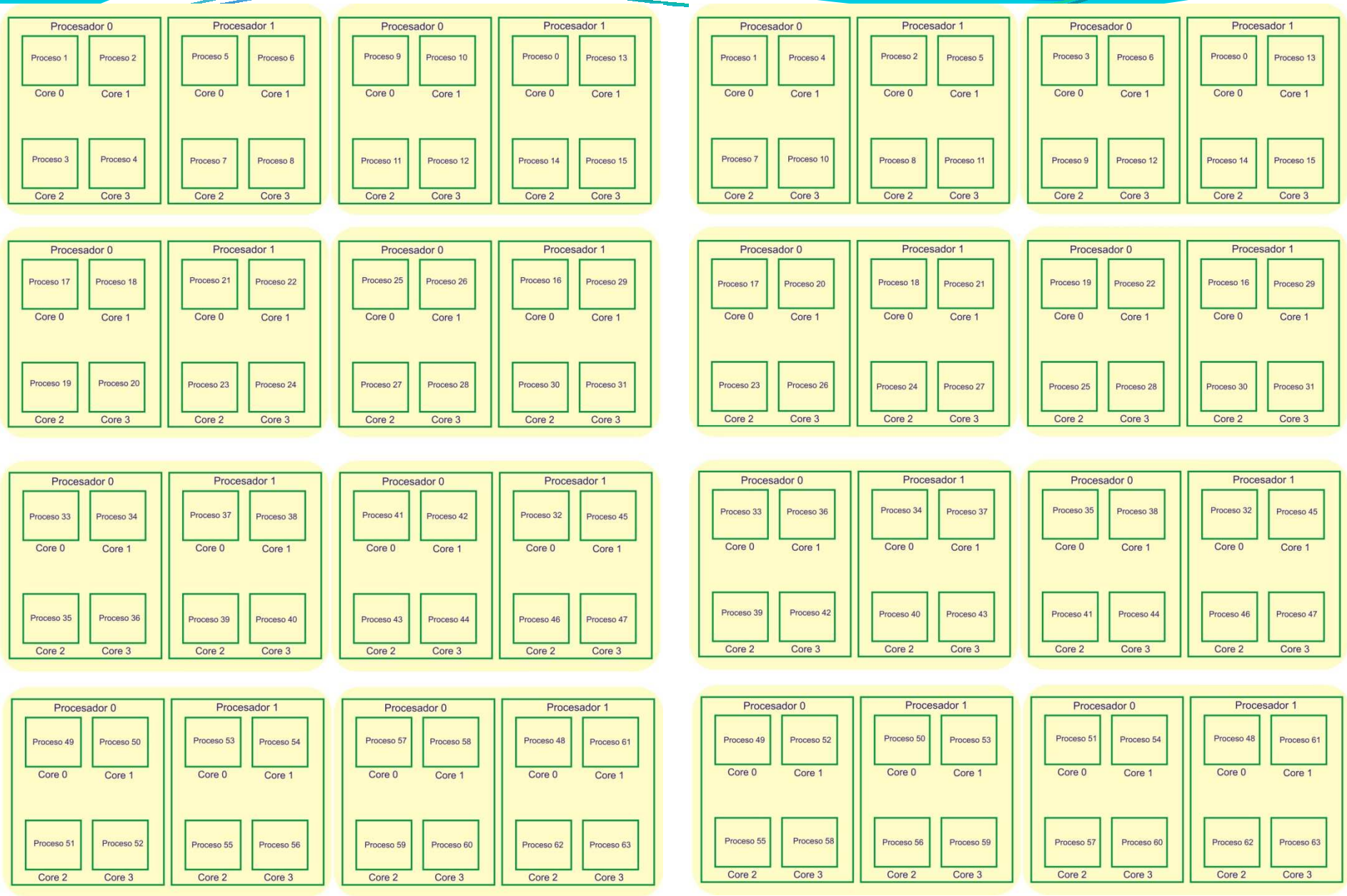
Soluciones que utilizan paralelismo funcional
(memoria compartida y pasaje de mensajes)



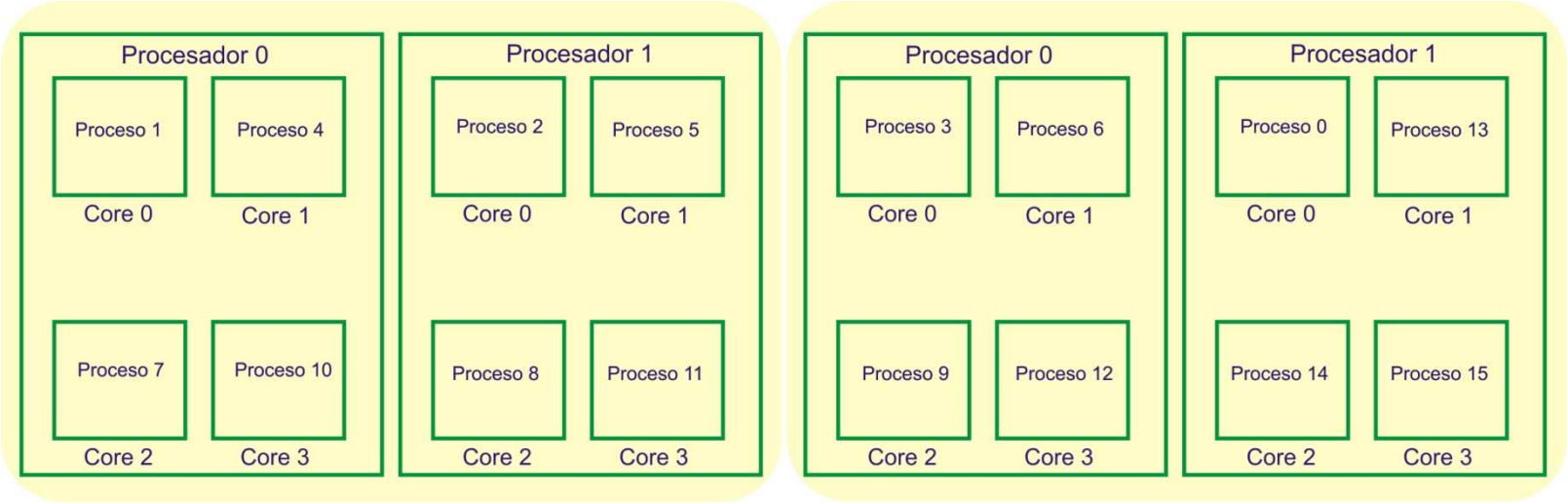


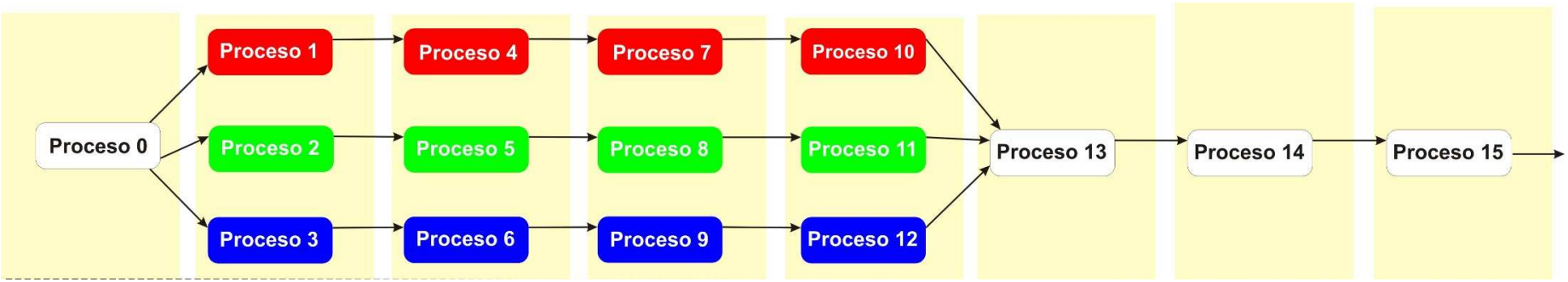
BASIZ-Pipe-5 MPI y OpenMP / BASIZ-Pipe-5 MPI y Pthreads

Filtro gaussiano subdividido utilizando 64 núcleos



Filtro gaussiano subdividido utilizando 64 núcleos





Resultados experimentales

Multicore

- Las pruebas fueron realizadas en un *Multicore Dell Poweredge* 1950, que posee 2 procesadores *quad core Intel Ceón* e5410 de 2.33 *GHz*; 4 *Gb* de memoria *RAM* (compartida entre ambos procesadores); *cache* L2 de *6Mb* entre cada par de núcleos de los procesadores. El sistema operativo que se utiliza es *Fedora* 11 de 32 *bits*.

- El tamaño de las imágenes procesadas es de 4096 X 4096 pixels

- Las pruebas fueron realizadas para 16, 18, 20, 22, 24, 26, 28 y 30 imágenes.

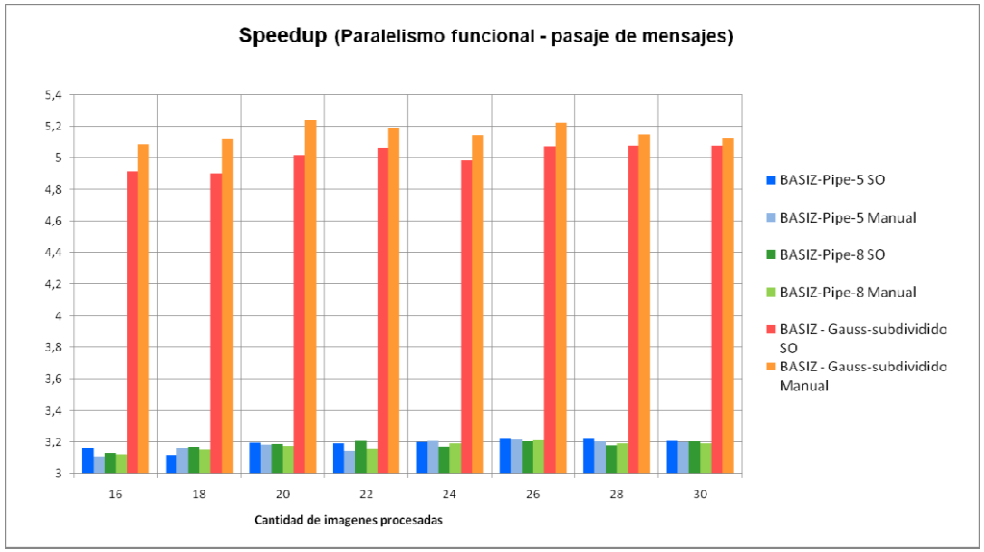
Cluster* de *multicore

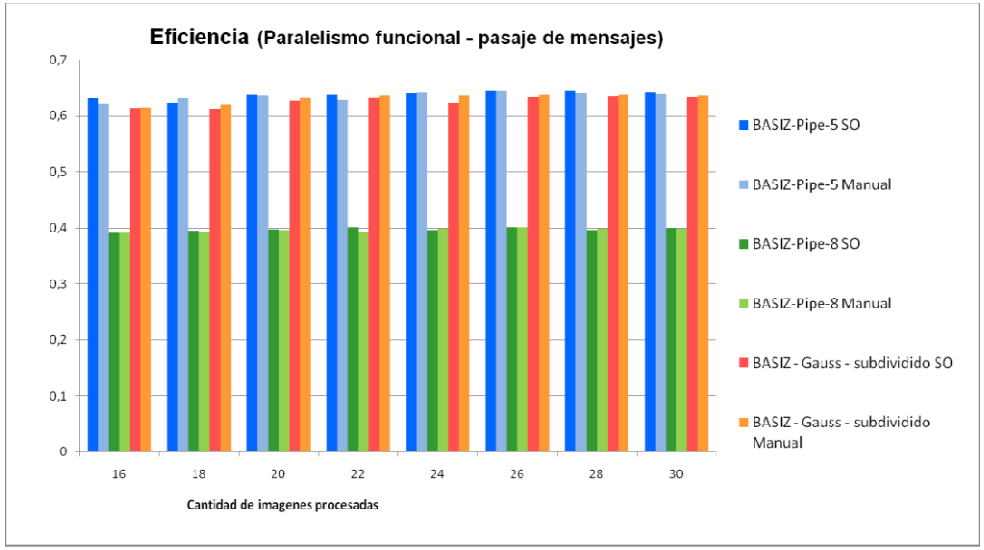
- Blade de 8 servidores (hojas). Cada hoja posee 2 procesadores quad core Intel Xeón e5405 de 2.0 GHz; 2 Gb de memoria RAM (compartida entre ambos procesadores); cache L2 de 2 X 6Mb compartida entre cada par de cores por procesador. El sistema operativo que se utiliza es Fedora 12 de 64 bits.

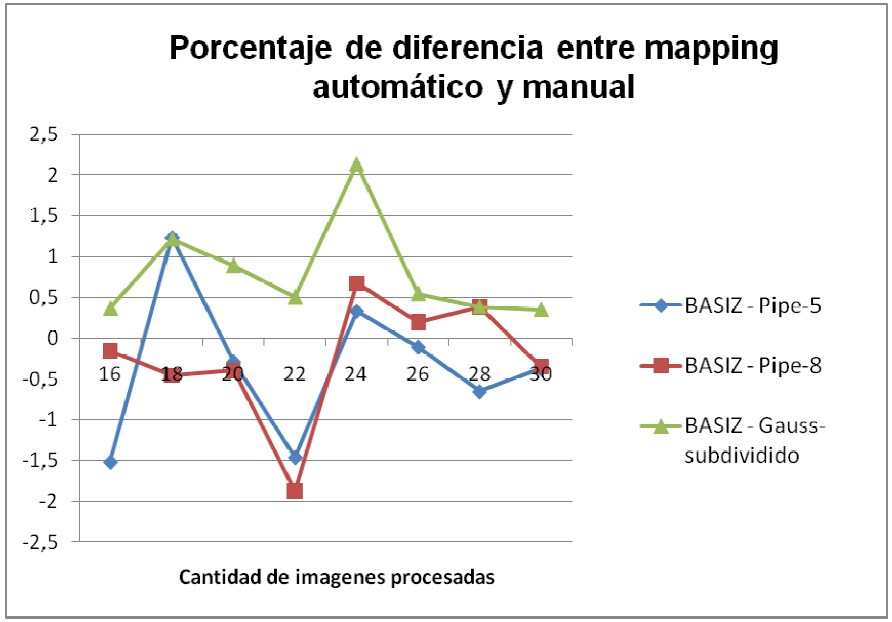
- El tamaño de las imágenes procesadas es de 4096 X 4096 pixels

- Las pruebas fueron realizadas para 16, 20, 24, 28 y 32 imágenes.

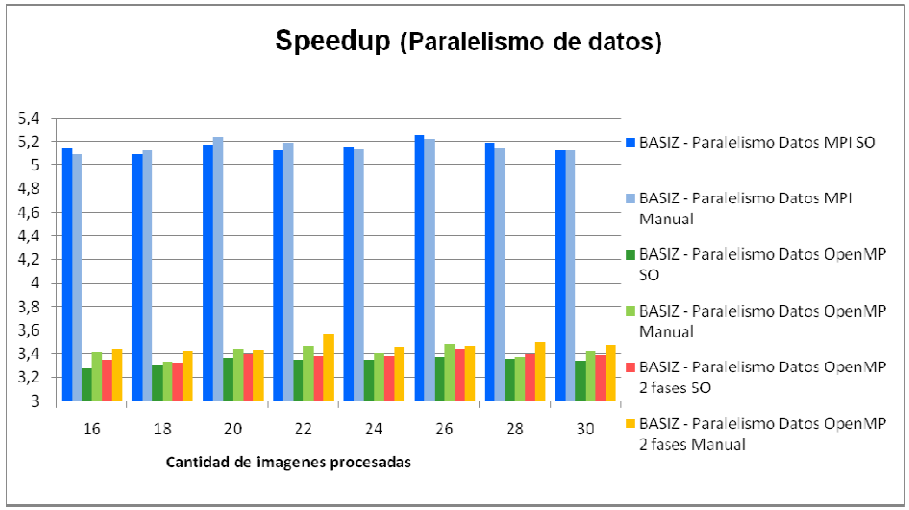
Resultados comparados

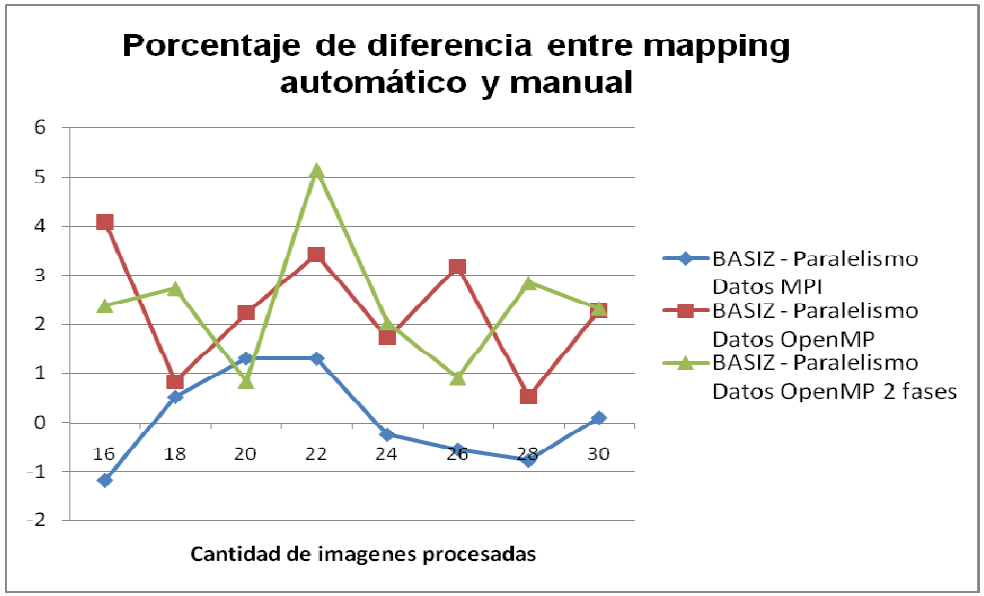


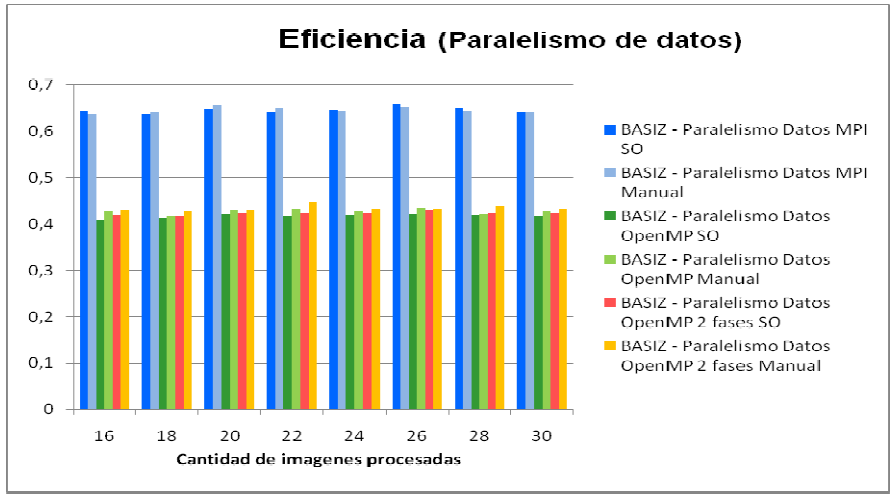




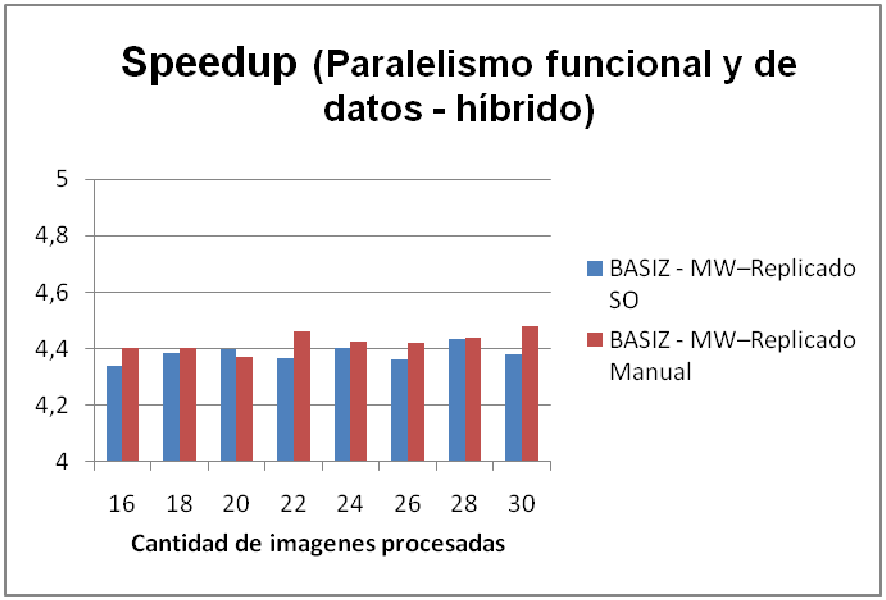
Resultados comparados

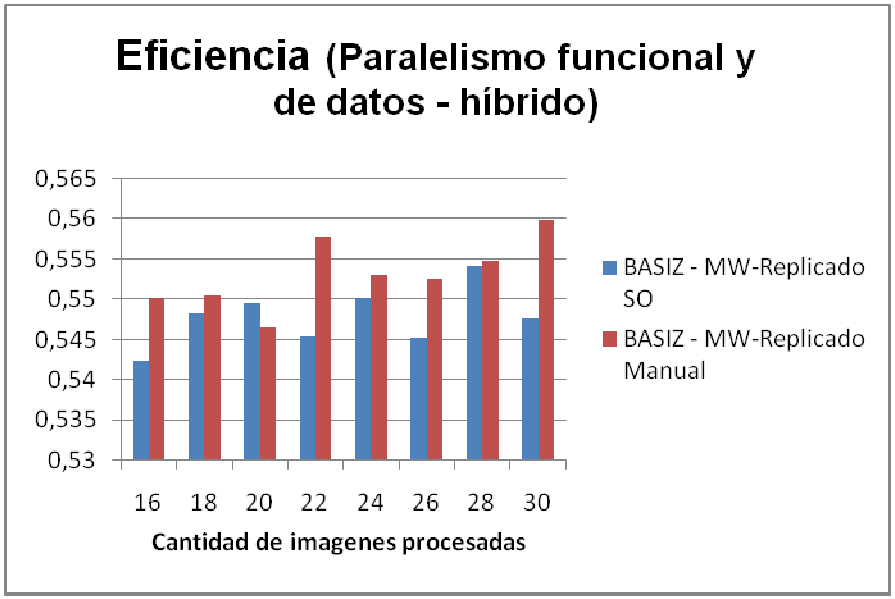


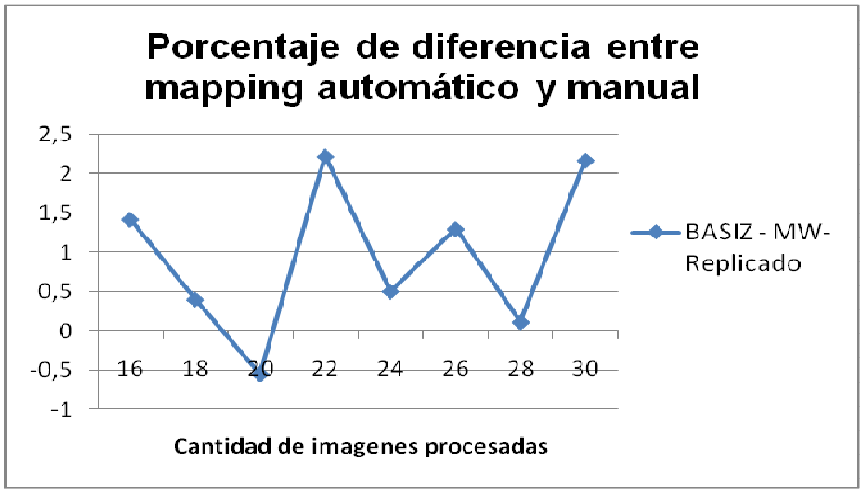




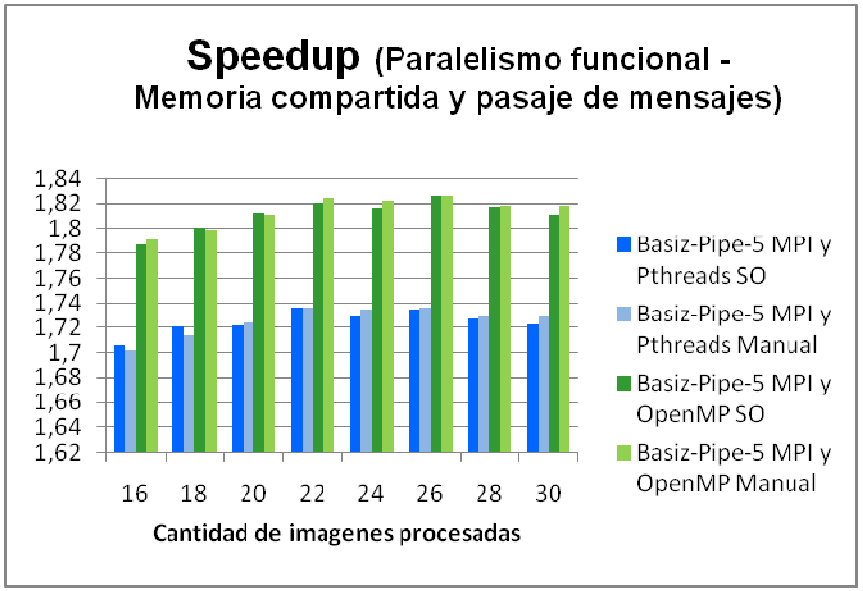
Resultados comparados

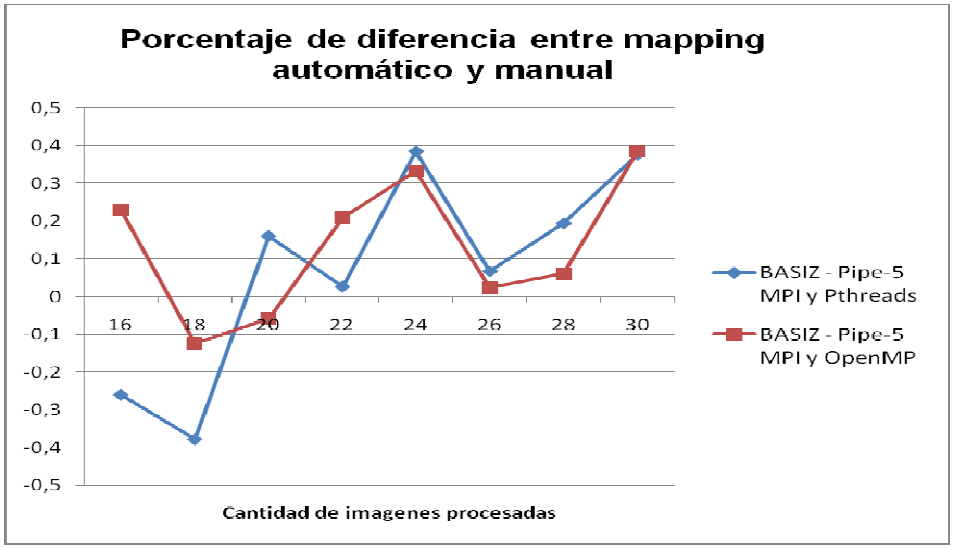


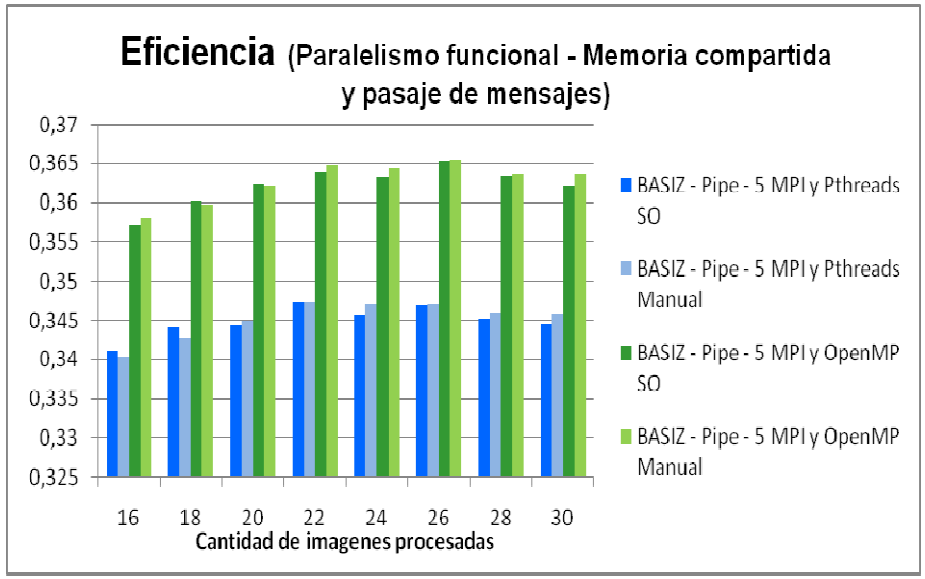




Resultados comparados

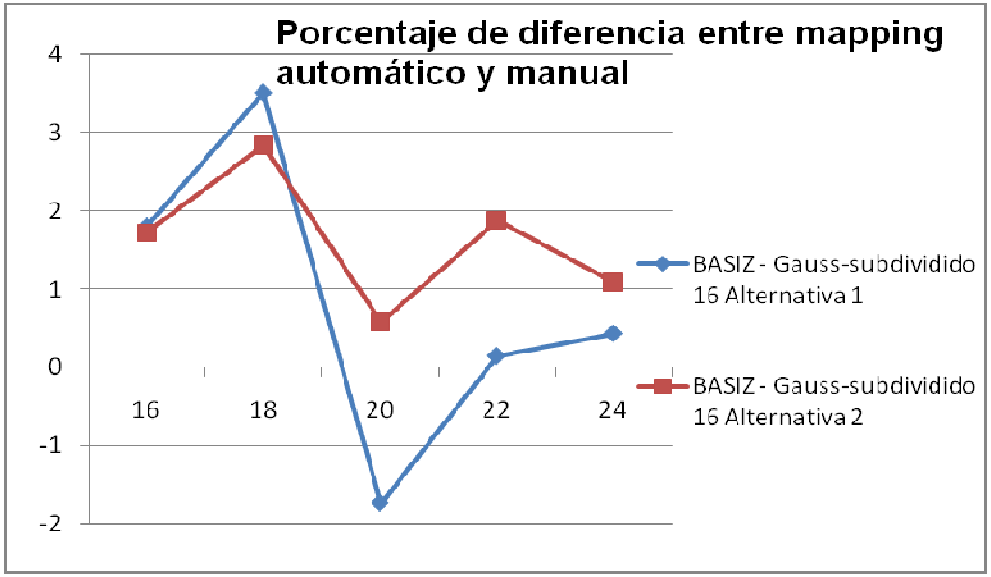






Resultados comparados

Cluster de *multicore*



Conclusiones

- El *speedup* del *Algoritmo BASIZ-Pipe-8* no mejora con respecto al *speedup* del *Algoritmo BASIZ-Pipe-5.*

- Los resultados de la solución Basiz-Gauss-subdividido muestran una mejora importante tanto en el *speedup* como en la eficiencia del mismo si se lo compara con las dos soluciones anteriores.

- En el algoritmo *BASIZ-MW* Replicado el *speedup* es mayor que en las soluciones *BASIZ-Pipe-5 y BASIZ-Pipe-8. E*l *mapping* manual de procesos mejora el *speedup* si se compara con el *mapping* realizado por el Sistema Operativo. Si se la compara con la solución BASIZ – Gauss subdividido, su *speedup* y eficiencia no logran superar a esta última.

- La solución *BASIZ* - Paralelismo Datos *OpenMPI* es la que mayor *speedup* y eficiencia alcanza si se la compara con *BASIZ* - Paralelismo Datos *OpenMP* y *BASIZ* - Paralelismo Datos *OpenMP* 2 fases. Si se comparan las tres soluciones anteriores, las que utilizan memoria compartida arrojan mejores resultados con respecto al porcentaje de diferencia entre *mapping* manual y automático.

Conclusiones (cont.)

- Las pruebas realizadas en el *cluster* de *multicore* arrojan resultados alentadores si se comparan las dos alternativas de *mapping* manual con la llevada a cabo por el sistema operativo.

- Utilizando el *mapping* manual (que toma en cuenta las jerarquías de memoria) en soluciones que utilizan diferentes modelos de programación paralela y diferentes estrategias de descomposición, se mejora la *performance* alcanzable de los mismos si se los compara con la obtenida con el *mapping* del sistema operativo.

- Los diferentes experimentos permitieron adquirir un *know-how* importante en cuanto a la asignación de procesos a núcleos, mostrando que lo que hace automáticamente el Sistema Operativo es mejorable.

Líneas futuras

- Investigar y analizar si la ganancia en *performance* obtenida por el *mapping* manual respecto al llevado a cabo por el sistema operativo se incrementa al aumentar el tamaño del problema.

- Implementar un *scheduler* que tenga en cuenta el tipo de aplicación, los recursos del sistema y las jerarquías de memoria existentes.

- Investigar y desarrollar un algoritmo (automático) que lleve a cabo la asignación de manera eficiente.

Muchas gracias!