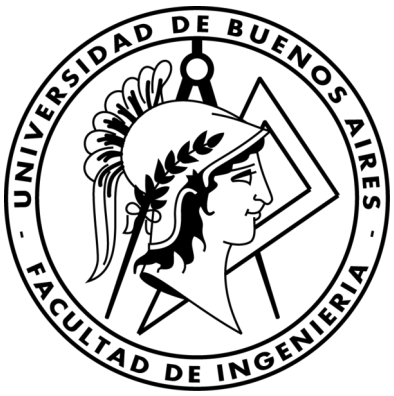
66.26 - Arquitecturas Paralelas
Trabajo final

Arturi, Augusto(#97498)
turitoh@gmail.com

Rozanec, Matias (#97404)
rozanecm@gmail.com

Diciembre 2018



Facultad de Ingeniería, Universidad de Buenos Aires

Índice

1. Introducción

La Dinámica de Fluidos (CFD por sus siglas en inglés) se usa para simular el flujo de fluidos en aplicaciones industriales. A medida que los simuladores se vuelven más complejos, el poder de cálculo que se requiere aumenta significativamente. En casos en que el tiempo de cómputo puede llevar días o meses, aún cambios relativamente pequeños en la eficiencia de cálculo pueden representar grandes mejoras en el cómputo.

El código usado para correr las simulaciones es OpenFOAM, un paquete CFD open source. Este software es diseñado para correr en paralelo, pudiendo ser configurado para correr efectivamente en cualquier cantidad de cores distribuidos a lo largo de cualquier número de computadoras presentes en una misma red. Idealmente los casos que se puedan correr en paralelo serán divididos en partes iguales distribuidas en cuantas unidades de procesamiento haya disponibles. Si cada proceso es capaz de correr independientemente, entonces el speedup será (en teoría) linealmente proporcional al incremento del hardware de cómputo.

Debido a la naturaleza de los cálculos de CFD, OpenFOAM requiere un nivel significativo de comunicaciones inter-proceso para asegurar resultados consistentes en el dominio del caso. Esto significa que aunque el software sea en teoría infinitamente paralelizable, cada proceso adicional agrega costo comunicacional que reduce el speedup. Adicionalmente, a medida que el tamaño del hardware crece, varios cuellos de botella del sistema, como la latencia de la red, pueden obstaculizar posibles mejoras y llegar a un límite en que ya no convenga agregar más hardware. Estas limitaciones que se encuentran en la práctica hay que encararlas con especial cuidado, ya que es lo que define el nivel del grano con que se va a trabajar en la paralelización: de tener un problema no demasiado grande, es probable que recurrir a un grano fino, o sea, subdividir el problema en partes muy pequeñas, no termine rindiendo tan bien como se esperaba debido a que el costo comunicacional probablemente supere la ventaja de paralelizar el problema, quedando esta última muy disminuida o incluso anulada.

En el presente trabajo se realizaron varias mediciones que permiten apreciar el desempeño en varios escenarios.

2. Herramientas

2.1. Hardware

PC1 Intel Core i5-7200U 2.5 GHz with Turbo Boost up to 3.1 Ghz
2 núcleos, 4 subprocesos.
2 canales de memoria
6 GB DDR4 memoria RAM 2400 MHz

PC2 AMD Ryzen 5 2400G 3.6GHz
4 nucleos, 8 subprocesos.
2 canales de memoria
16 GB DDR4 memoria RAM 2400 MHz

PC3 Intel Core i5-5200U CPU @ 2.20GHz 3M Cache, up to 2.70 GHz
2 nucleos, 4 subprocesos.
2 canales de memoria
8 GB DDR4 memoria RAM 2400 MHz

2.2. Software

2.2.1. Software de resolución numérica

El software utilizado para la resolución de problemas numéricos es OpenFOAM, un software de código abierto desarrollado por OpenCFD Ltd. desde 2004. Éste es utilizado en muchas áreas de ciencia e ingeniería, tanto en ámbitos comerciales como educativos. OpenFOAM presenta una amplia gama de características que le permite resolver una amplísima gama de problemas de fluidos, incluyendo pero no limitándose a reacciones químicas, turbulencias, transferencia de calor, acústica, mecánica de sólidos y electromagnetismo entre otros.

2.2.2. Software de paralelización

Para poder correr el solver en paralelo, se hace uso de mpirun (Open MPI) versión 1.10.2.

2.2.3. Software de medición de tiempo

Debido a que el programa puede ser operado desde una terminal, las mediciones de las corridas se han podido realizar directamente mediante el comando time, obteniendo las siguientes mediciones:

Real: mide el tiempo desde que arrancó a correr el programa hasta que terminó. Este tiempo incluye los time slices que el procesador está ocupado con otros procesos no relacionados con el programa de interés. Sin embargo, como esto siempre sucede en un sistema real[1], al haber hecho las mediciones corriendo solamente el software de interés sin otros procesos importantes de fondo, se puede considerar esta medición como buena.

User: mide tiempo en que la CPU está ocupada en modo usuario. Notar que este tiempo en teoría no podría ser mayor al tiempo Real, sin embargo esto no es cierto si se corre un proceso en varios procesadores: en tal caso, User indica el total de tiempo contemplando a todas las CPUs, por lo que se esperan tiempos mayores al Real.

Sys: mide tiempo en que la CPU está ocupada en modo kernel.

[1] Un sistema operativo siempre distribuye el tiempo de procesador de la mejor forma posible, y si bien puede haber tareas más importantes que otras, siempre habrá algunos procesos que requerirán algún tiempo de CPU, aunque el mismo sea mínimo, por lo que habrá ciclos de clock dedicados a tareas varias distintas a la medida.

3. Análisis preliminar

3.1. Hardware

Hay una variedad de factores de hardware que pueden afectar la eficiencia en el cálculo de CFD.

Velocidad de CPU/RAM: se listan la velocidad de cores y velocidad de RAM de las PC utilizadas para las simulaciones, notando que OpenFoam es una aplicación de alto consumo de memoria, por ende se espera una mejor perfomance en aquellas RAM con mayor frecuencia de clock.

Canales de memoria RAM: tanto la velocidad de la memoria RAM como el número de canales entre la CPU y la memoria compartida puede afectar la velocidad de una simulación CFD. Dado que los procesadores que se utilizan tienen 2 canales de memoria, cuando la cantidad de cores activos sea mayor a 2 y el canal de memoria esté saturado dado el límite de ancho de banda, algunos cores quedarán en idle hasta que se transfiera toda la información a memoria, este proceso puede enlentecer la simulación.

Turbo boost: la capacidad de overclockear el procesador con el fin de aumentar su frecuencia, es una opción que puede maximizar la performance a un alto riesgo de dañar el CPU. Otra forma es utilizar el turbo, capacidad que solo tienen los chips Intel, que es una función designada a operar solo cuando unos pocos cores del CPU están siendo utilizados y el calor excedente producido por la disipacion de energia puede ser distribuido a lo largo del mecanismo de cooling del chip. Teóricamente, si todos los cores están siendo utilizados simultáneamente, el turbo boost no se activa.

Hyper-threading: esta característica le permite a cada núcleo del CPU presentarse al sistema operativo como si fuesen dos, de los que uno será real y el otro virtual. Luego el SO puede asignar trabajos a los cores virtuales y asignarles trabajo cuando los cores reales están en idle (como por ejemplo esperando una lectura/escritura en memoria), de esta forma se maximiza la utilización del CPU. Sin embargo, OpenFoam es generalmente una aplicacion de uso intensivo de memoria: en raros casos durante una simulación CFD, la utilización de la CPU baja del 100 %, por lo tanto incrementar el número de procesos incrementará la comunicación, generando más overhead, en conclusión a priori se podría decir que este factor afectaría a cualquier beneficio de la utilización de cores virtuales, se espera demostrarlo en el presente trabajo.

3.2. Software

Binding and distribution: el método por el cual los procesos que se ejecutan son alocados y limitados en los cores, tienen un impacto muy significativo en la velocidad del paralelismo, la proximidad de los procesos en la arquitectura de hardware afecta la velocidad y eficiencia del proceso de intercomunicación. En una computadora de escritorio la asignación de los procesos es provista por el scheduler del sistema operativo en uso, el cual hará lo necesario para balancear la carga y ejecutar los procesos de la mejor manera posible, a su vez esta tarea podría implicar mover procesos entre distintos cores, siempre con la intención de mejorar la performance. Las aplicaciones CFD asignan los datos discretos a cada proceso en paralelo que es guardado en la memoria de cada núcleo, si por alguna razón el SO decide mover el proceso a un core el cual en memoria no tiene los datos necesarios, la información debe ser reescrita en la memoria de este para que el trabajo pueda continuar; esto produce un efecto perjudicial a la performance.

Sin embargo, OpenMPI provee algunas primitivas para que en su ejecución se eviten este tipo de reasignaciones antes de que el trabajo sea terminado mediante el binding de un proceso a un core particular.

Case size and decomposition: cuando una simulación es preparada para correr en paralelo, el dominio total es repartido en piezas del mismo tamaño, las cuales se asignan acorde a la cantidad de cores disponibles. Así como el número de procesos incrementa, cada pieza se torna más pequeña y el cálculo se completa más rápidamente, pero tal como se comentó anteriormente, incrementar el paralelismo implica incrementar las intercomunicaciones en orden de mantener el resultado esperado consistente. Como resultado, el número de procesos a paralelizar alcanza un máximo práctico, donde el speedup no se puede mejorar dado el aumento de tiempo en comunicación. Para un caso dado, este punto máximo es importante de encontrar para asegurar que los recursos computacionales no son desperdiciados en vano.

El método por el cual es descompuesto el dominio afecta el tamaño y la forma de cada pieza de este, así como también el orden en el que estas son numeradas, lo cual afecta a las posiciones relativas dentro del dominio (distancia dentro del procesador).

OpenFoam provee de 4 métodos de descomposición:

Simple: la descomposición es geométrica en la cual el dominio se divide en partes a partir de la dirección en los ejes x,y,z. Esta forma se suele utilizar en problemas en los cuales la naturaleza del problema es simétrico.

Jerárquica: es igual a la simple con la salvedad de que el usuario puede especificar el orden en el cual quiere se divida por eje, es decir, primero por el eje Y, luego X y finalmente Z por ejemplo.

Scotch: esta descomposición es un poco más interesante ya que no divide geométricamente por igual, por lo cual suele ser la más apropiada en problemas de la vida real. Esta forma intenta minimizar las comunicaciones entre los procesadores haciendo que los límites entre estos sean los menores posibles; más adelante se lo explicará con un ejemplo práctico. A su vez se pueden asignar pesos a los trabajos por procesador, característica útil en caso de tener a disposición chips de distinta frecuencia para realizar la simulación.

Manual: el usuario especifica directamente la locación del área que se le asigna a cada procesador. Sin embargo, en un ejemplo en el cual se tienen que asignar una alta cantidad (millones) de celdas el proceso puede ser tedioso o impráctico.

4. Casos de estudio

El software OpenFOAM incluye en su instalación una enorme cantidad de tutoriales a modo de ejemplo para los usuarios con el fin de demostrar la amplísima gama de problemas que se pueden resolver con el mismo. Debido a la complejidad de este tipo de problemas y por no ser el foco del presente trabajo, se decide trabajar con uno de los tutoriales que utiliza el *compressibleInterFoam solver*, que es un solver para dos fluidos compresibles no isotérmicos e immiscibles usando el volumen del fluido.

Entre las posibilidades de usar el caso 2D y el 3D, se decidió realizar pruebas con el 2D por un tema práctico: el caso 3D es computacionalmente muchísimo más complejo que el 2D, lo que extendería muchísimo el tiempo dedicado a correr el solver en sus distintas configuraciones posibles. Muy probablemente el estudio de los tiempo de ejecución y sus comparaciones serían más vistosos, pero dado el alcance del trabajo, se ha decidido que dicho límite es aceptable.

5. Preparación del entorno

El tutorial con el que se trabajó se encuentra en el directorio `$FOAM_RUN/tutorials/multiphase/compressibleInterFoam/laminar/depthCharge2D`.

Para el trabajo se han realizado leves cambios en los archivos originales, los cuales se encuentran todos debidamente documentados y adjuntos en el apéndice del trabajo.

6. Caso distribuido

Para poder resolver el problema en paralelo aprovechando los varios núcleos presentes en una PC, es necesario preparar el dominio de acuerdo a la cantidad de procesadores disponibles o que se quieran utilizar.

Como primer paso, hay que correr el comando `decomposePar`, que subdivide el problema en n partes iguales. n se especifica en el archivo `system/decomposeParaDict` en la línea 17. Al ejecutar el comando, se obtiene una salida similar a la siguiente, que puede variar en función de n:

```
1  Decomposing mesh region0
2
3  Create mesh
4
5  Calculating distribution of cells
6  Selecting decompositionMethod simple [4]
7
8  Finished decomposition in 0 s
9  Calculating original mesh data
10 Distributing cells to processors
11 Distributing faces to processors
12 Distributing points to processors
13 Constructing processor meshes
14
15 Processor 0
16         Number of cells = 3200
17         Number of faces shared with processor 1 = 80
18         Number of faces shared with processor 2 = 40
19         Number of processor patches = 2
20         Number of processor faces = 120
21         Number of boundary faces = 6520
22
23 Processor 1
24         Number of cells = 3200
25         Number of faces shared with processor 0 = 80
26         Number of faces shared with processor 3 = 40
27         Number of processor patches = 2
28         Number of processor faces = 120
29         Number of boundary faces = 6520
30
31 Processor 2
32         Number of cells = 3200
33         Number of faces shared with processor 0 = 40
34         Number of faces shared with processor 3 = 80
35         Number of processor patches = 2
36         Number of processor faces = 120
37         Number of boundary faces = 6520
38
39 Processor 3
40         Number of cells = 3200
41         Number of faces shared with processor 1 = 40
42         Number of faces shared with processor 2 = 80
43         Number of processor patches = 2
44         Number of processor faces = 120
45         Number of boundary faces = 6520
46
47 Number of processor faces = 240
48 Max number of cells = 3200 (0% above average 3200)
49 Max number of processor patches = 2 (0% above average 2)
50 Max number of faces between processors = 120 (0% above average 120)
51
52 Time = 0
53
```

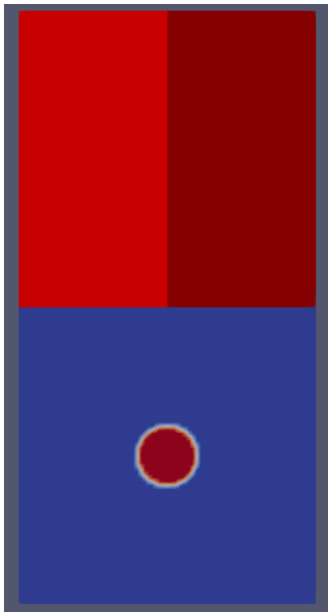


Figura 1: **El problema elegido** En esta imagen se puede apreciar la geometría del problema elegido: dos líquidos immiscibles en un recipiente, y una burbuja en la parte inferior del mismo.

```
Processor 0: field transfer
Processor 1: field transfer
Processor 2: field transfer
Processor 3: field transfer

End
```

En la salida se puede ver cómo el total de celdas queda dividido en 4, ya que se tienen disponibles los 4 procesadores y a su vez cómo se comparte la información de las caras de la figura entre cada uno de los procesadores.
Además se han creado 4 directorios nuevos, cada uno representando a un core distinto.

La información a continuación es importante en el caso de querer analizar visualmente la información del problema y su respectiva subdivisión en subproblemas. Es una práctica muy recomendable.
A continuación se ejecutará el comando `foamToVTK` en cada uno de los directorios de los procesadores, lo cual creará un directorio `VTK` que contendrá información de los subdominios para ser analizada con `paraView`.

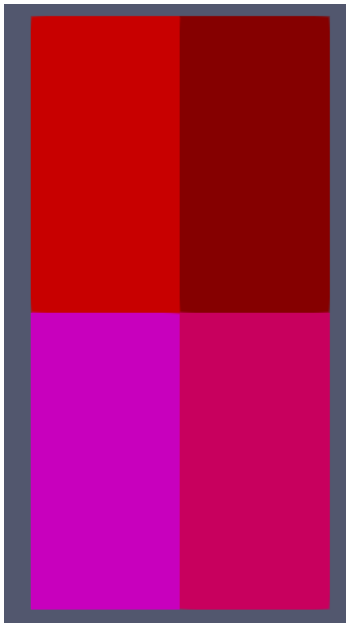


Figura 2: **Subdivisión del problema** En esta figura se puede cómo se asigna un núcleo para cada parte de la figura.

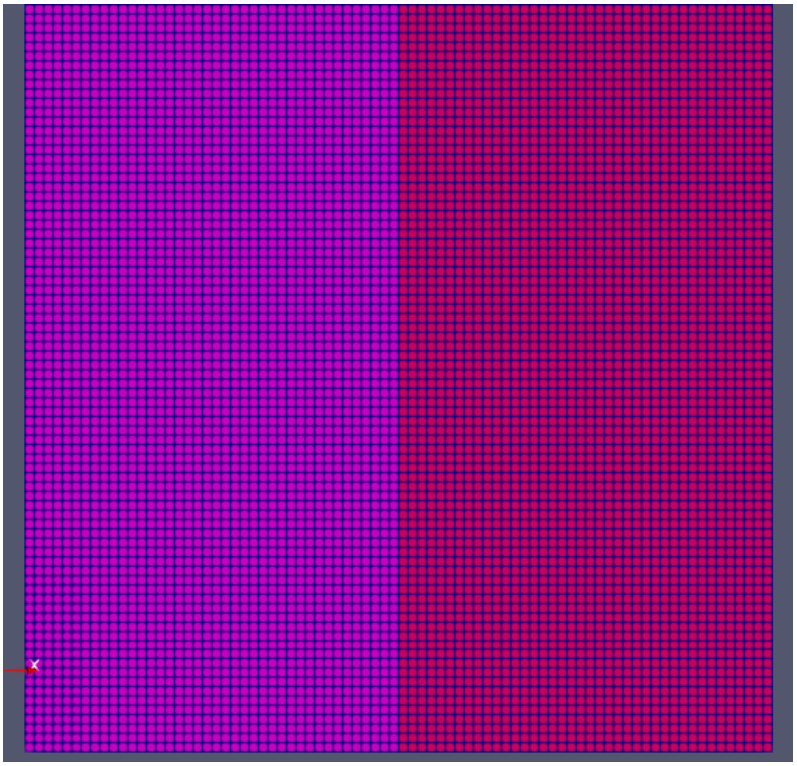


Figura 3: **Detalle de la discretización** En esta imagen se ve en detalle la discretización, cómo cada core se subdivide en 3200 celdas (eje y dividido en 40, el x en 30). Notar que para realizar las operaciones en el límite de las caras se necesita información de varios procesadores distintos: es importante no pasar por alto este detalle, ya que es la variable que juega en contra de la paralelización por traer consigo un costo comunicacional extra. En el hipotético caso de tener 3200 cores para resolver este problema, claramente la situación no escala, debido a que el costo comunicacional terminaría ocupando la mayor porción del tiempo.

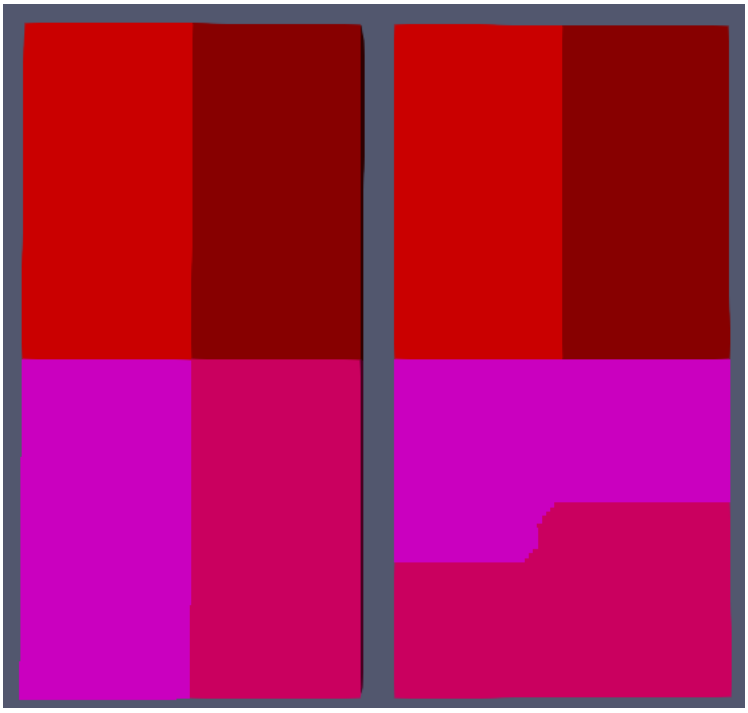


Figura 4: **Descomposición Jerárquica vs. Scotch** En esta descomposición se puede ver que existe una ligera modificación en cómo se asignan las caras y la distribución de celdas en cada uno de los cores. Este tipo de descomposición es ideal para usar cuando la geometría del problema no es simétrica como en el caso del ejemplo, donde le trata de asignar a cada core la misma cantidad de celdas y minimizar las caras entre estos en base a un algoritmo u estrategia.

7. Ejecución

7.1. Monocore

Para ejecutar el solver sin paralelismo, es suficiente con correr el comando `./Allrun`. El mismo se encarga de crear la malla de discretización y de correr el solver.

7.2. Multicore

Para la resolución con paralelismo se hace uso del MPI (Message Passing Interface) mediante el comando mpirun. Al mismo se le pasan flags indicando cantidad de procesadores y tipo de binding, tema que se cubrirá a continuación. Por un tema de prolijidad se sugiere redireccionar la salida del programa a un archivo log.

```
1 mpirun -np 4  compressibleInterFoam -parallel > log.6626
```

Listing 1: Ejemplo de comando a ejecutar para resolver con paralelismo.

Al estar ejecutándose la simulación se puede ver el uso de cpu mediante comandos como `top` o `htop`: de acuerdo a la cantidad de procesadores que se haya decidido utilizar, se observa un uso cercano o igual al 100 % en las cpus en uso.

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
13224 acer      20   0  593904  91360  77764 R  97,7  1,5   1:16.46 compressibleInt
13226 acer      20   0  593968  93576  79988 R  97,7  1,6   1:16.11 compressibleInt
13225 acer      20   0  593968  90924  77336 R  96,0  1,5   1:17.81 compressibleInt
13223 acer      20   0  594000  93456  79760 R  92,0  1,6   1:15.21 compressibleInt
```

Figura 5: Detalle de ejecución de `top`

7.2.1. Bindings

El binding se setea mediante el flag –bind-to. Por defecto se hace –bind-to core, lo que asegura que un mismo proceso se corra siempre en el mismo procesador. Esto es positivo, ya que si el proceso se va cambiando de procesador, se pierde toda la información que el procesador haya guardado en sus registros o cache. Para desactivar esto, se usa –bind-to none.

El flag –bycore asegura que procesos secuenciales de MPI corran en procesadores adyacentes.

Lo esperado es que el binding incremente la performance, previniendo al sistema operativo mover procesos de un core a otro con el fin de balancear la carga.

8. Análisis de las ejecuciones

En esta sección se realizará un análisis basado en gráficos que permitan apreciar los distintos speedups obtenidos con las distintas configuraciones.

Para el análisis se usará el tiempo real, ya que si bien contempla también tiempos no usados exclusivamente para el procesamiento, se la considera una medida más real debido al tiempo que se usa para comunicación.

Los speedup fueron calculados tomando como tiempo viejo el tiempo necesario para correr el solver en un único núcleo, ya que es el único caso que no hace uso del paralelismo y se quiere ver cómo se puede mejorar a partir de ese caso paralelizando el problema.

8.1. PC #2

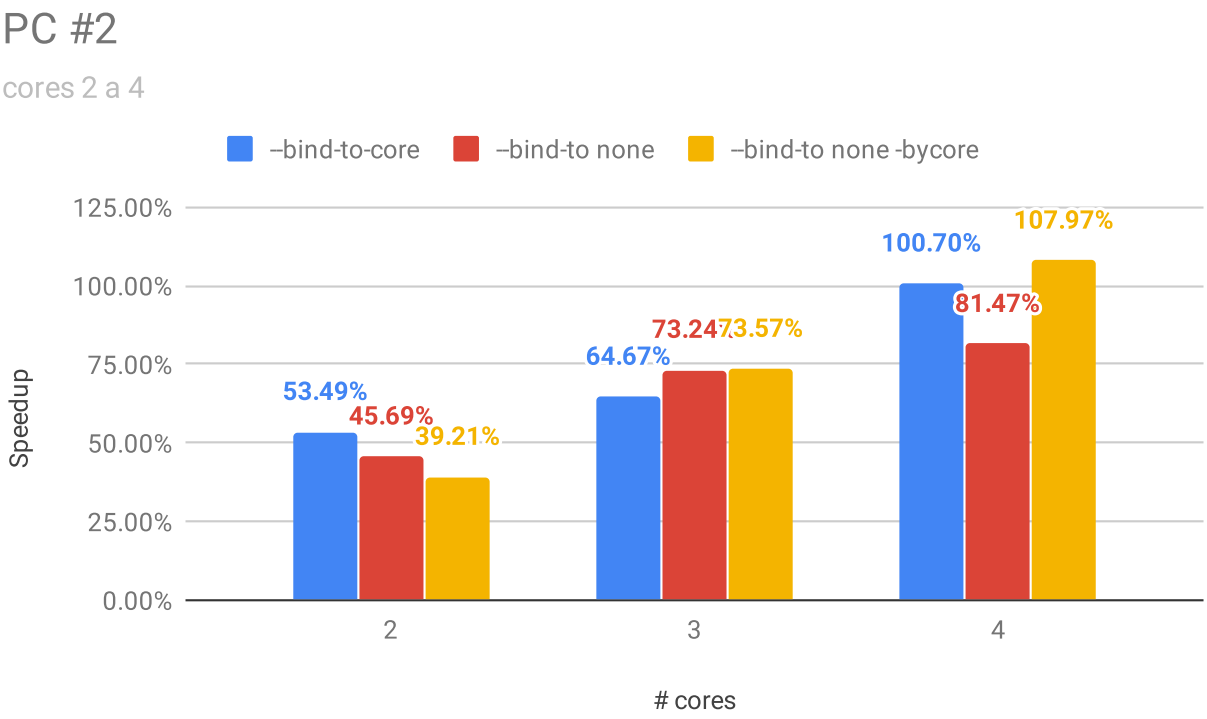


Figura 6: Speedups de PC 2: cantidad de núcleos entre 2 y 4.

En esta PC se puede observar que ya usando dos núcleos se obtiene un speedup de 53 % en el mejor de los casos. Este speedup que supera el 50 % es muy bueno considerando que duplicando la cantidad de núcleos el óptimo teórico tiene un tope de 100 %, el cual se sabe que es inalcanzable en la práctica. Notar de todas formas que hay un 14 % de diferencia con el speedup obtenido en el peor caso. Si bien un speedup de 39 % es de todas formas muy bueno, es evidente ya desde el principio que los distintos tipos de binding juegan un papel clave en el manejo de la paralelización.

En el caso de la corrida con 3 núcleos se observa que el mejor speedup fue alcanzado por dos de las tres técnicas de binding, logrando un speedup de 73 %. En este caso la diferencia con el peor speedup logrado es mucho menor, de tan solo 9 %. Sigue siendo, de todas formas, una diferencia significativa. A diferencia del caso anterior, se observa que se logra un cuarto del speedup óptimo teórico (300 %).

Fue necesario ocupar 4 núcleos para alcanzar un speedup del 100 %. La relación entre el óptimo teórico y el mejor real es igual al caso anterior: se logra solamente un cuarto del máximo teórico

(400 %). Además se puede ver nuevamente que el binding juega un papel no menor, ya que la diferencia entre mejor y peor caso utilizando 4 núcleos es del 27 %.

Como la PC en análisis cuenta con 8 subprocesos pero solamente 4 núcleos, será interesante ver los resultados de las mediciones subsiguientes. ¿Seguirá aumentando el speedup, o fue el máximo posible ya alcanzado?

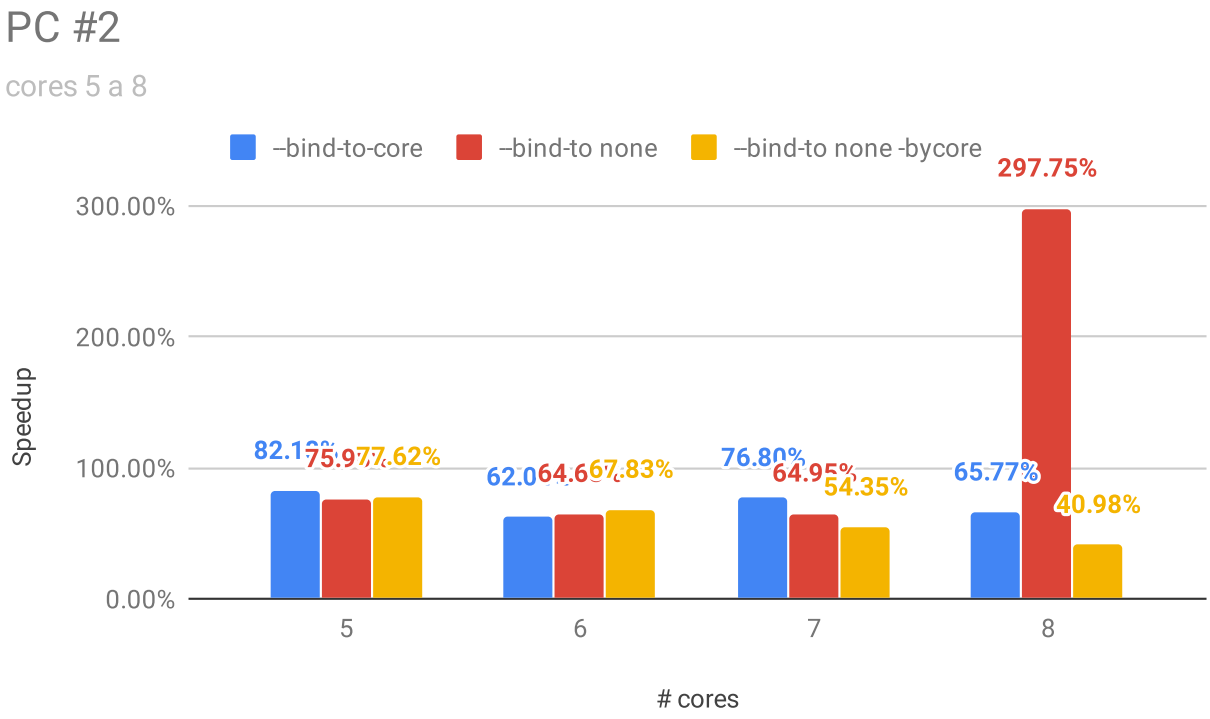


Figura 7: Speedups de PC 2: cantidad de núcleos entre 5 y 8.

A primera vista se puede ver que los speedups obtenidos son todos inferiores al mejor speedup obtenido con hasta 4 núcleos.

En los casos de 5 y 6 núcleos casi no hay diferencia entre los distintos tipos de binding, mientras que en el caso de 7 núcleos, dependiendo del tipo de binding se puede obtener hasta un 21

En el caso de los 5 núcleos se observa un speedup de entre el 76

En el caso de los 6 núcleos el speedup fue menor: en el mejor caso se alcanzó el 67

Con 7 núcleos la variedad de resultados desorienta bastante: en el mejor de los casos, el speedup es casi igual al peor caso de los 5 núcleos (!).

. . . . . .

De todas estas mediciones con 5 o más núcleos, se pueden rescatar las siguientes observaciones contraintuitivas: en primer lugar, el máximo speedup fue logrado con 5 núcleos, que es la menor cantidad de la muestra observada. Por otro lado, este speedup es muy similar al speedup obtenido con 3 núcleos, lo que significa que todos los speedups con 5 o más están muy por detrás del speedup obtenido con 4 núcleos.

8.2. PC #3

8.2.1. Descomposicion Jerarquica

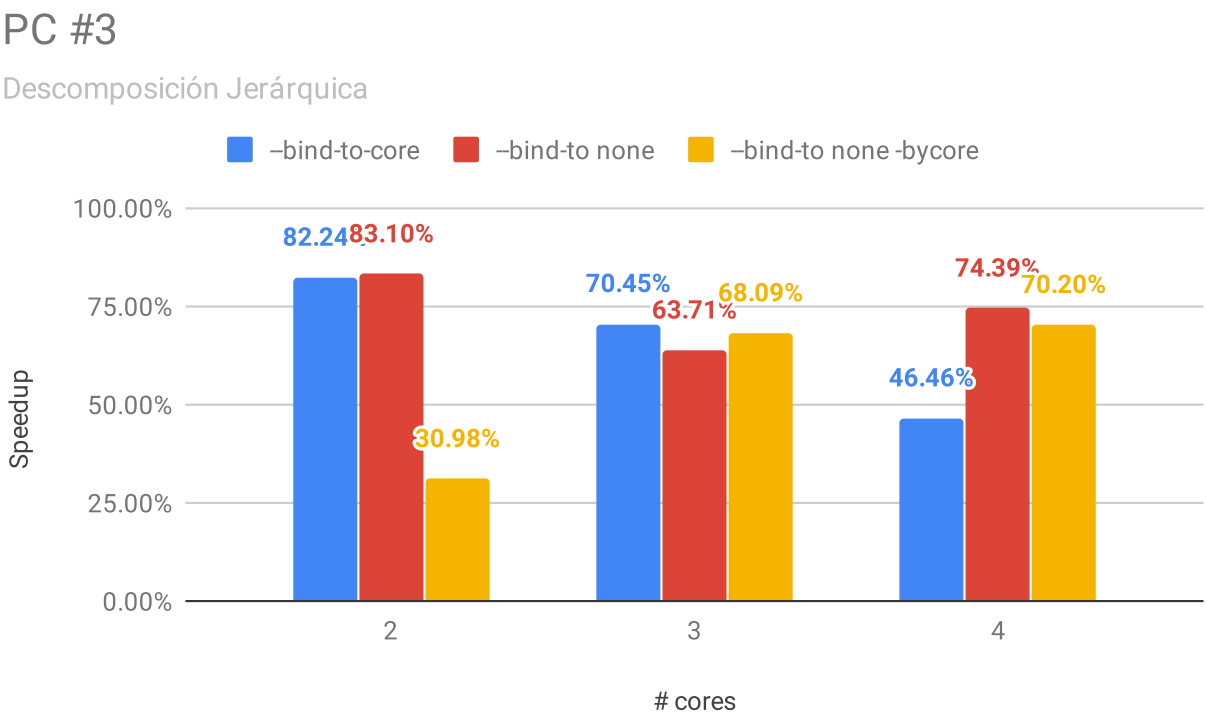


Figura 8: Speedups de PC 3 con descomposición jerárquica

Con esta pc se puede ver que utilizando la descomposición jerárquica, ya usando solamente 2 núcleos se obtiene un excelente speedup, un poco mejor al 80 %, que es el 80 % del óptimo teórico posible: llama la atención, ya que no es común acercarse tanto al óptimo teórico. Notar, de todas formas, que si bien con dos tipos de binding se obtienen excelentes resultados, al usar bind to none by core, el speedup es muchísimo menor: solamente es del 30 %, teniendo un 50 % de diferencia con los otros dos tipos de binding (!). Indudablemente no es un detalle menor.

Con 3 núcleos se observa un desempeño relativamente parejo entre los tres tipos de bindings, habiendo solamente un 7 % de diferencia entre los casos extremos. Sorprende un poco que el speedup no sólo no haya mejorado con respecto al caso de los dos núcleos, sino que ha empeorado alrededor del 10 %.

En el caso de los 4 núcleos, los resultados son un tanto variables: en el mejor de los casos se obtiene un speedup del 75 %, mientras que en el peor de los casos el speedup obtenido es de 46 %, lo que hace una diferencia de 30 % entre los extremos. Nuevamente se observa que el speedup no supera al del caso de los 2 núcleos, y además es bastante parecido al obtenido con 3 núcleos.

Indudablemente, la descomposición jerárquica ha sorprendido un poco con los resultados obtenidos.

8.2.2. Descomposición Scotch

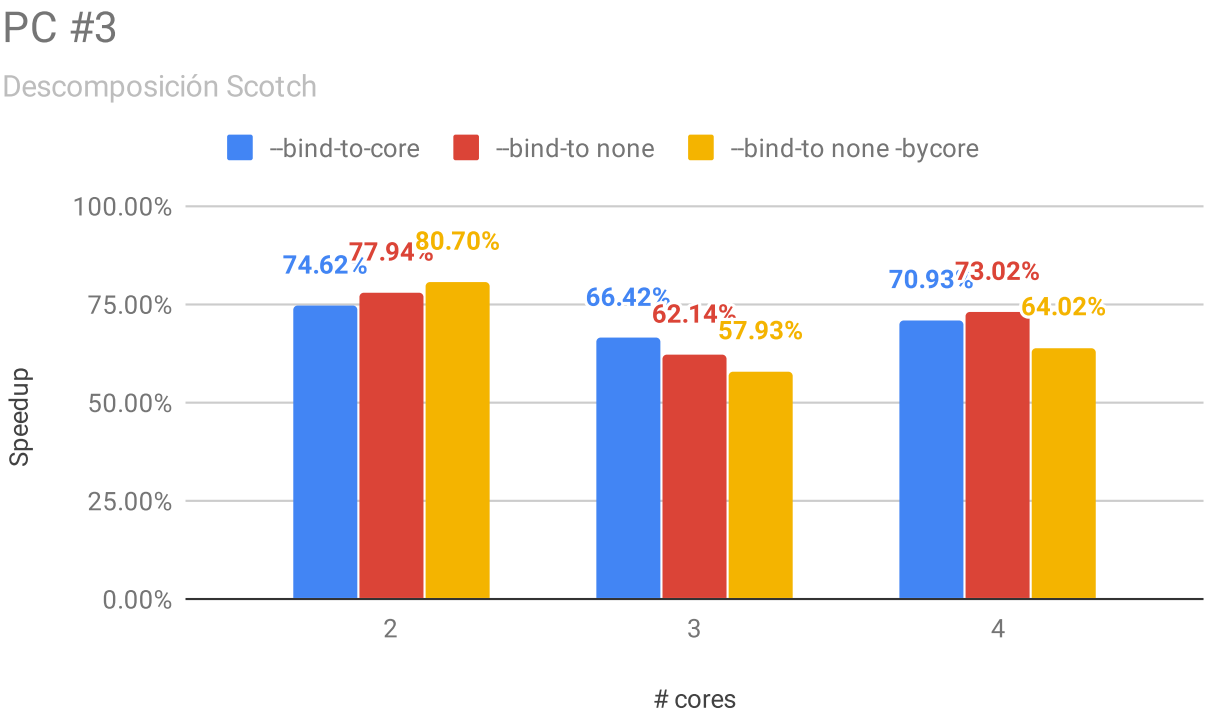


Figura 9: Speedups de PC 3 con descomposición scotch

En el caso de la descomposición scotch con dos núcleos se midió un speedup de entre 75 % y 80 %. Nuevamente, como se comentó en el caso de la descomposición jerárquica, dicho speedup es excelente considerando que el máximo teórico es del 100

Lamentablemente el speedup no sigue creciendo a medida que se agregan más núcleos: viendo los gráficos de 3 y 4 núcleos se puede ver que en ninguno de los casos se alcanza, con ninguno de los tres bindings posibles, el peor de los speedups obtenidos con solamente 2 núcleos.

8.2.3. Jerarquica vs Scotch

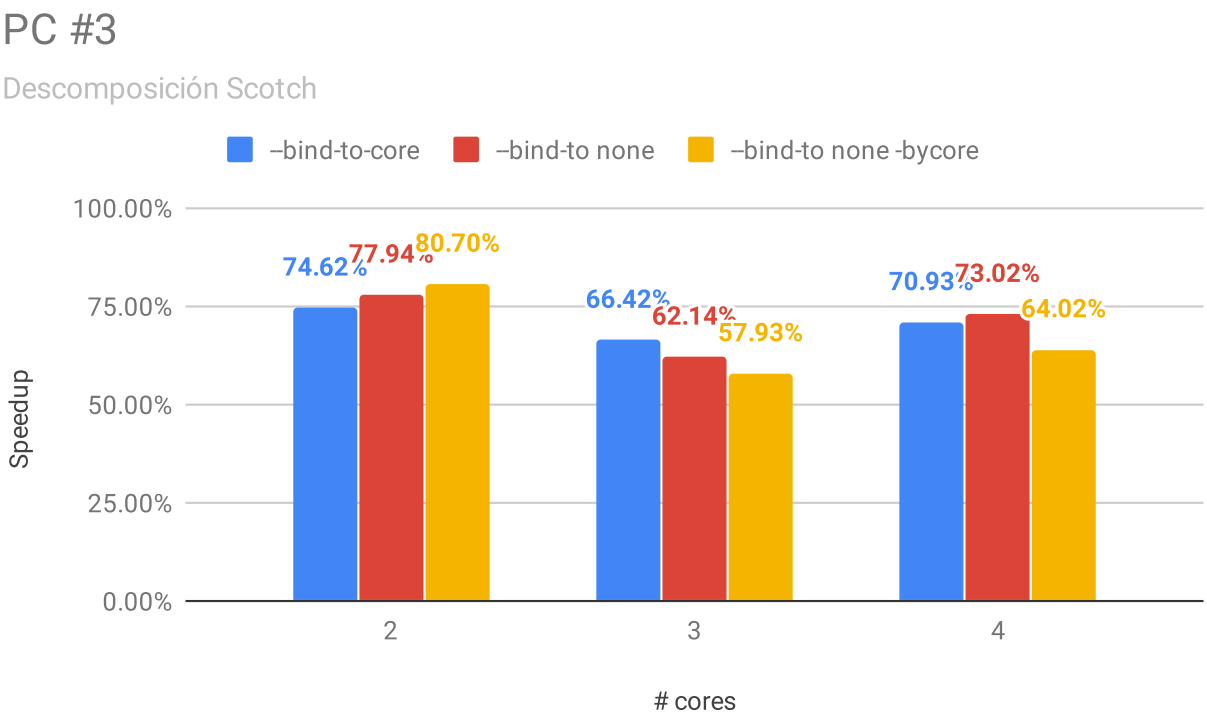


Figura 10: Speedups de PC 3 con descomposición scotch

Si bien se pueden observar pequeñas diferencias que en los tres casos favorecen a la descomposición Jerárquica, estas diferencias no son mayores. De todas formas toda la evidencia indica que es mejor trabajar con descomposición Jerárquica.

9. Conclusiones

A partir de las mediciones se puede ver que no siempre se cumple que al agregar más procesadores aumenta el speedup. Esto tiene que ver con cuestiones planteadas al principio del trabajo: el costo comunicacional consume tiempo, y si bien es subconscientemente subestimado debido a que el enfoque está en la parte computacional, termina siendo evidente que no es un tema menor.

Si bien en la PC 2 se pudo observar un speedup creciente a medida que se iba aumentando la cantidad de núcleos reales, no sucedió lo mismo con la PC 3, donde si bien no se esperaban valores exagerados e irreales que se acerquen demasiado al speedup óptimo teórico, sí se esperaba que aumente por lo menos un poco al agregar más núcleos para el cómputo.

10. Anexo

10.1. Mediciones

asdf. La hoja de datos con las mediciones junto a los gráficos correspondientes están disponibles para una mejor apreciación en este link.