Universidad Politécnica de Cataluña

Paralelismo y Sistemas Distribuidos

Grado en Ciencia e Ingeniería de Datos

Proyecto High Performance Computing (HPC)

Autores
Enric Millán
y J.P. Zaldivar

Supervisora
Julita Corbalán

14th April, 2023

Contents

1 Introducción

El presente proyecto se enfocará en el análisis tres aplicaciones NASPB (NASA Advanced Supercomputing Parallel Benchmarks) distintas; **bt**, **sp**, **lu**. Para las cuales se realizarán una serie de pruebas y estudios, especificados a continuación.

1.1 Visión general

Cada aplicación se puede configurar con diferentes clases, es decir, tamaño de datos y número de procesos. Para el estudio se configurarán con la **clase D** para la versión MZ-MPI (NPB3.3.1-MZ/NPB3.3-MZ-MPI) que contiene los kernels de MPI+OpenMP. Una de las distintas versiones disponibles. El número de procesos y threads serán definidos más adelante con el propósito de entender el impacto de diferentes factores de múltiples combinaciones.

1.2 Objetivo

El objetivo del trabajo es elaborar un análisis de rendimiento de las aplicaciones, así como de sus principales componentes en cuanto a tiempo de ejecución y tipo de llamadas MPI y **schedule** OpenMP.

1.3 Metodología

Principalmente el trabajo se divide en tres grandes partes. La primera consiste en la ejecución de las tres aplicaciones con diferentes configuraciones. Con lo que se realizará un estudio de los resultados, principalmente del **tiempo de ejecución**, del **speedup** y de la **eficiencia**. Cada una con su debida justificación, con el objetivo de determinar el nivel de **paralelismo** más eficiente. Ya sea el nivel de **MPI**, con muchos procesos y pocos threads por proceso, o el nivel de **OpenMP**, con pocos procesos y muchos threads por proceso.

Para el segundo apartado del estudio, se explorará más a profundidad la información obtenida de las ejecuciones mediante el uso de comandos en la terminal. De lo que derivará en un análisis del **consumo energético** de las ejecuciones y de su **potencia media**.

El último apartado consiste en la comparación del schedule de OpenMP para la aplicación de lu.

1.3.1 Generación de códigos

Para realizar la compilación de los programas se ha creado un script de shell (para cada aplicación) con el comando:

```
make <app>-mz CLASS=D NPROCS=<tasks>
```

Tanto el script mencionado, como el resto de implementaciones de soporte se encuentran en el siguiente repositorio habilitado.

Además de los casos propuestos, hemos buscado añadir experimentos intermedios que nos permitiesen analizar los resultados de forma más gradual pero sin prescindir del *compute power*, es decir, de manera que el número de threads asignados a cada proceso estuviera balanceado o, en otras palabras, que la fracción entre los 48 threads disponibles por nodo y el número de procesos asignados a cada nodo sea entera.

Con este criterio, se han realizado los experimentos que se pueden consultar en el anexo A. Aquí se muestra un ejemplo para el caso de 4 nodos de la aplicación bt.

Nodos	Procesos (MPI)	Threads por proceso
4	8	24
4	12	16
4	16	12
4	24	8
4	32	6
4	48	4
4	96	2
4	192	1

1.3.2 Parámetros y Speedup teórico

Para cada ejecución se han realizado un total de tres repeticiones. Por lo que tras la obtención de

los datos muestrales, se procedió a hacer un computo de la media del tiempo de ejecución, además de la desviación estándar de las observaciones.

En cuanto al speedup teórico,

$$Speedup(p) = \frac{T(1)}{T(p)} \quad (1)$$

este se obtuvo de la salida para cada ejecución realizada, el cual esta calculado para el caso ideal. Es decir, que es prácticamente una aceleración casi de orden lineal.

1.3.3 Speedup y eficiencia empíricos

Para la obtención del speedup empírico se tuvo que calcular primeramente el tiempo secuencial de cada aplicación por separado. Como, por razones de configuración, no es posible calcular el tiempo secuencial con un único proceso y un único thread, se realizaron las medidas con 8 procesos y 1 thread. De modo que se obtuvieron las siguientes medias haciendo la media de las 3 repeticiones:

Aplicación	Tiempo medio (s)
bt	2206
sp	1151.58
lu	1818

El análisis a detalle del speedup obtenido para cada aplicación en función del número de nodos será discutido en el apartado 2. El estudio de la eficiencia,

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

que corresponde al speedup escalado por el número de recursos empleados, será evaluado a la par.

En los futuros apartados que mostrarán los gráficos de la eficiencia, es importante tener en cuenta que la eficiencia se ha calculado sobre 1 cpu, por lo que se ha escalado el speedup empírico para ajustar un futuro criterio de eficiencia mínima del 70%. Como el speedup se ha calculado sobre la base de 8 procesos MPIs, a la hora de calcular la eficiencia se ha multiplicado por 8 el speedup para encajar con dicha fórmula de la eficiencia.

2 Evaluación inicial

La decisión de las gráficas a continuación vino de la mano de poder visualizar los puntos correspondientes a cada configuración experimental dependiendo de la aplicación, pero al mismo tiempo de poder considerar la evolución de una posible tendencia mediante las rectas que unen los puntos.

Igualmente la distinción de dentro de cada gráfica para especificar cada tipo de nodo de una diferente tonalidad (en la escala "*Viridis*") la consideramos fundamental y mucho más entendible para cualquier tipo de gráfica a realizar. Si bien el número de puntos no es elevado, se consideró que una visión más a detalle de la tendencia supondría un mejor análisis que no únicamente de los valores representados de otra manera. Aunque es verdad que para la aplicación lu, una tendencia muy clara no se llega a apreciar, se decidió mantener el formato que se llevaba hasta al momento en lugar de decantarse por otro tipo de figura.

2.1 Análisis aplicación bt

2.1.1 Tiempo de ejecución

En primer lugar, se puede comprobar que en términos generales, cuanto más nodos disponibles, menor tiempo de ejecución. Esto se debe al hecho de que al haber un mayor número de nodos, se dispone de una mayor cantidad de threads con los que paralelizar.

Si nos fijamos en el número de MPIs (procesos), observamos una tendencia similar independientemente de los nodos. A partir de 72-96 procesos aproximadamente, el overhead empieza a influir notoriamente en el tiempo de ejecución, haciendo que los puntos tomen una tendencia creciente, dejando atrás los mínimos.

Encontramos también unos picos para cualquier número de nodos, aproximadamente en el mismo número de procesos MPI, lo que puede sugerir que estén posiblemente relacionados con un desbalanceo de trabajo, lo que ocasiona un empeoramiento del rendimiento.

Una posible causa del overhead, en este y los futuros razonamientos es por motivo de la creación de los nodos, la cual implica un incremento en el tiempo de ejecución.

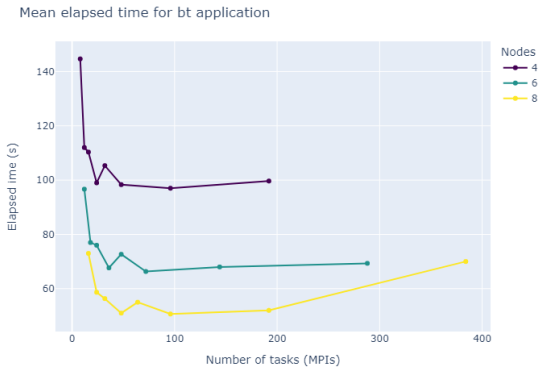


Figure 1: Mean elapsed time for bt

Teniendo esto en cuenta, la configuración ideal en función del tiempo de ejecución es en general de 8 nodos con 96 procesos MPI. Si se clasifica por número de nodos, para 4 nodos se obtiene el mínimo tiempo de ejecución con 96 procesos MPI y para 6 nodos con 72.

2.1.2 Speedup

Una vez más, en términos generales, las configuraciones con 8 nodos obtienen un mayor speedup, como es lógico y por la misma razón que antes se ha explicado.

Se observa un notorio efecto de overhead, que disminuye el speedup de las ejecuciones a partir de un cierto número de procesos. Es más evidente en el caso de 8 nodos, en el que decrece más rápidamente, debido al overhead asociado a crear más threads.

Al haber dependencia con la gráfica 1, se observan de nuevo los picos, invertidos a causa de la fórmula para el cálculo del speedup.

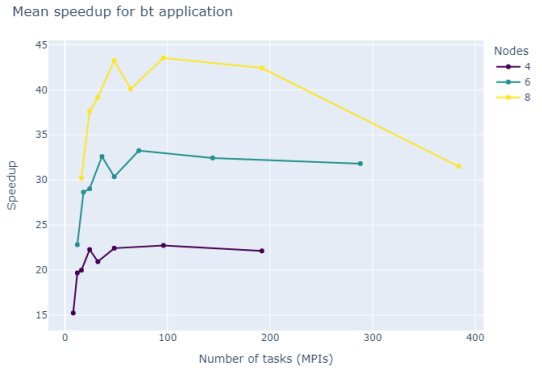


Figure 2: Mean speedup for bt

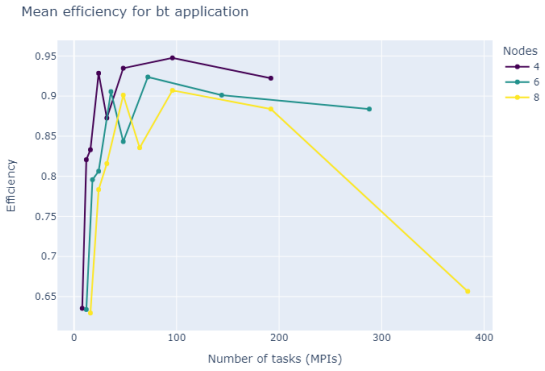


Figure 3: Mean efficiency for bt

Por la misma razón, se establece que las mejores configuraciones coinciden con las del mejor tiempo de ejecución. Esta configuración es la de 8 nodos y 96 procesos MPI.

2.1.3 Eficiencia

En primer lugar, llama la atención el hecho de que se ha invertido el orden de los nodos en términos de magnitud, es decir, respecto al speedup y al tiempo de ejecución, las configuraciones de 8 nodos siempre eran superiores a las de 4 nodos y en el caso de la eficiencia sucede lo contrario. Esto se debe a que cuantos más nodos haya, mayor será el denominador del computo de la eficiencia y este crecimiento es superior al decrecimiento del tiempo de ejecución.

Teniendo esto en cuenta, la mejor configuración respecto a la eficiencia es para 96 procesos MPI en el caso de 4 nodos. Subsiguientemente, para el caso de 6 nodos es con 72 procesos con los que se alcanza la mejor configuración. Finalmente, con 96 procesos se obtiene la mejor configuración para 8 nodos.

Cabe recalcar que para esta aplicación todas las configuraciones son aceptables en términos de eficiencia, puesto que están por encima del margen de 0.5. Sin embargo, aplicando un límite mínimo de 70% de eficiencia, las primeras configuraciones para cada número de nodos no son válidas, al igual que la última configuración de 384 procesos MPI para 8 nodos, pues tiene una eficiencia del 65%.

Nuevamente se nota la presencia del overhead entre los 72-96 procesos, siendo para los 8 nodos, un efecto mucho más pronunciado. Una vez más, aparecen unos sospechosos picos, que se discutirán en el apartado 2.4 de análisis global.

En este caso, la mejor configuración en términos de eficiencia es la de 4 nodos y 96 procesos MPI.

2.2 Análisis aplicación sp

2.2.1 Tiempo de ejecución

Para el caso de la aplicación sp, vemos que ambos extremos se ven acentuados respecto al rendimiento de la aplicación bt. Es decir, la mejoría con el aumento de MPIs antes de llegar al mejor tiempo de ejecución es mucho más pronunciada.

Sin embargo, el aumento por el overhead también toma una pendiente mayor. Como es el caso de los resultados para esta aplicación, el overhead tiene un mayor impacto sobre todo en las ejecuciones con 4 nodos.

A excepción del caso de 8 nodos, también hay presencia de picos. Si bien tiene una altura menor en comparación con la aplicación anterior, seguramente son ocasionados por motivos similares.

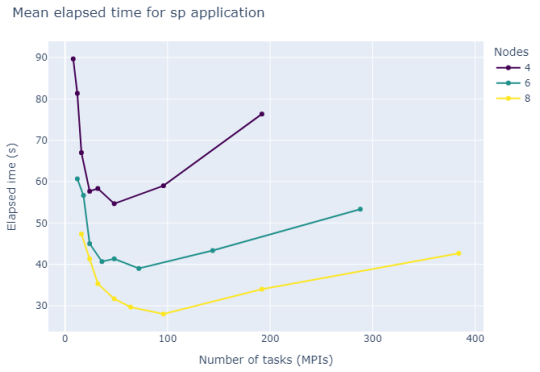


Figure 4: Mean elapsed time for sp

Como en el programa anterior, el número de nodos afecta positivamente en términos generales y la mejoría empieza a disiparse a partir de 48, 72 y 96 procesos MPIs respectivamente para 4, 6 y 8 nodos. Estas son entonces las mejores configuraciones para cada nodo y la mejor configuración es una vez más para el caso de 8 nodos y 96 MPIs. Cabe mencionar que en esta segunda aplicación, las tres mejores configuraciones coinciden en que el número de threads por proceso es 4.

2.2.2 Speedup

Como el speedup es inversamente proporcional al tiempo de ejecución, se obtienen las mismas conclusiones. Siendo la mejor configuración también con 8 nodos y 96 procesos MPIs, manteniéndose también las otras configuraciones.

Para estos casos, el efecto del overhead parece ser similar independientemente del número de nodos. Siendo ligeramente más acentuado para el experimento de 8 nodos. Al igual que la creación de nodos, la comunicación y sincronización de nodos también requiere de tiempo y recursos. Por lo que a medida que se aumentan el número de nodos, el overhead se vuelve cada vez más considerable.

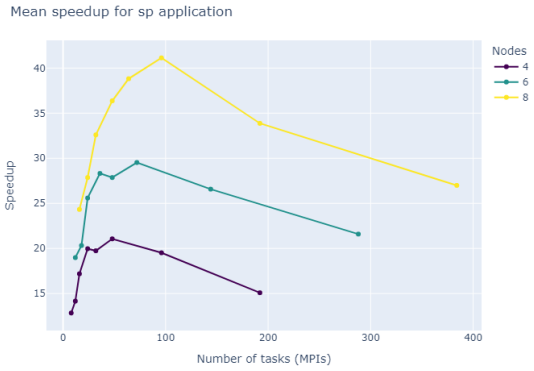


Figure 5: Mean speedup for sp

2.2.3 Eficiecnia

Para la aplicación sp, se observa un comportamiento similar a la eficiencia de bt para las primeras muestras, pero en este caso, el overhead se hace notar más pronto y tiene un mayor efecto sobre todo para el caso de 4 nodos.

Los picos en este caso aparecen para 4 y 6 nodos únicamente, pero parece que su efecto no es tan notorio para esta aplicación.

Tal y como sucede con el programa bt, se puede considerar que todas las configuraciones son aceptables al superar todas el valor de 0.5. En cambio, fijando el límite inferior sobre una eficiencia mínima de 70%, solo un número reducido de configuraciones pueden ser aceptadas como válidas. En definitiva, menos que en la aplicación anterior, puesto que para 4 y 6 nodos, dentro de un rango de 16 a 144 procesos MPI, solo estos cumplen con la condición impuesta. Así mismo, de 48 a 192 MPIs son aceptables para las configuraciones con 8 nodos.

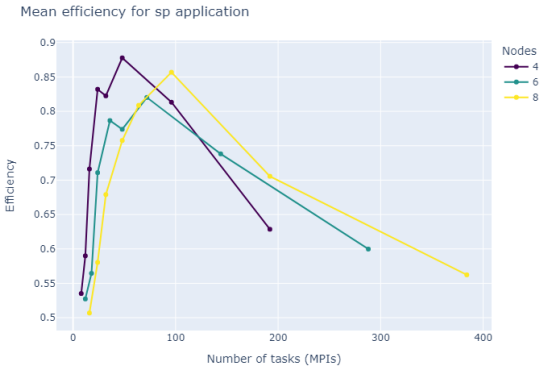


Figure 6: Mean efficiency for sp

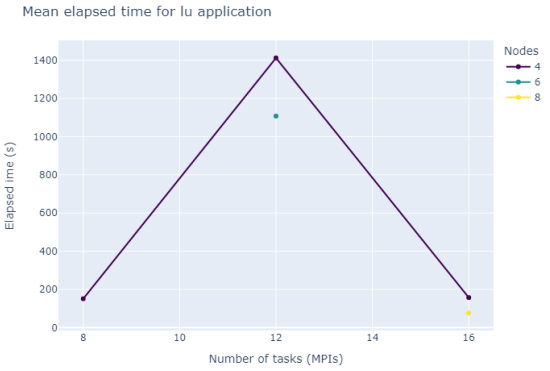


Figure 7: Mean elapsed time for lu

La mejor configuración se sigue manteniendo para los 4 nodos, esta vez con 48 procesos MPI, pero a diferencia de la aplicación anterior, la segunda mejor configuración corresponde a los 8 nodos con 96 procesos.

2.3 Análisis aplicación lu

Tal como explica el enunciado del proyecto, el caso de la aplicación lu es especial y el número de configuraciones experimentales apropiadas son reducidas, por lo que tanto las gráficas como su consecuente análisis serán más simples que los casos anteriores.

2.3.1 Tiempo de ejecución

A grandes rasgos, la aplicación lu consume mucho más tiempo que el resto de aplicaciones. Sin embargo, se mantiene el hecho de que con 8 nodos se obtiene el mejor tiempo.

También se debe mencionar que los casos con 8 y 16 procesos para el caso de 4 nodos, son más rápidos que el único caso para 6 nodos. Hecho que desentona con las aplicaciones anteriores, en las que el aumento de nodos implicaba una reducción general y considerable del tiempo de ejecución.

Basándonos en la brevedad de la gráfica, la mejor configuración es como ya se ha explicado la de 8 nodos y 16 procesos MPI. Para el caso de 4 nodos, la mejor configuración es con 8 procesos.

2.3.2 Speedup

Tal y como sucede con las demás aplicaciones, las conclusiones para el speedup son las mismas que nos ofrecen el tiempo de ejecución. Se observa que, nuevamente, el caso de 8 nodos (y 16 procesos MPI) es notablemente mejor y que para 4 nodos los 8 MPIs obtienen el mejor resultado.

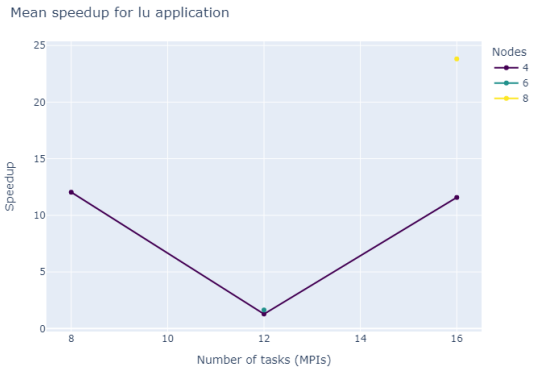


Figure 8: Mean lu speedup for lu

2.3.3 Eficiecnia

Con un primer vistazo, nos damos cuenta que esta es la aplicación con la que se obtiene una

menor eficiencia, independientemente de la configuración. Es más, los valores más altos apenas superan el 0.5, que marca que se trata de una eficiencia aceptable.

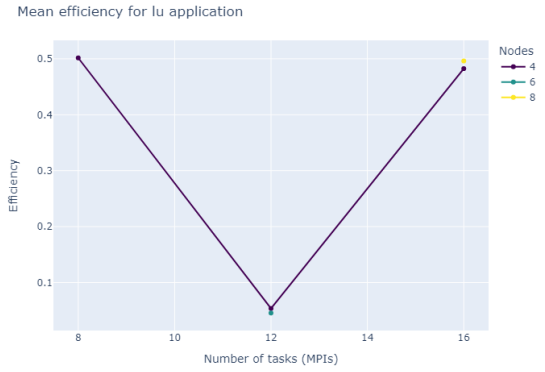


Figure 9: Mean efficiency for lu

De hecho, la única configuración que a duras penas supera el valor de 0.5 es la de 4 nodos y 8 procesos MPI. El resto no pueden considerarse eficientes bajo estos criterios, siendo las peores configuraciones, con diferencia, las de 12 procesos para 4 y 6 nodos. Para el criterio mínimo de 70% de eficiencia, no se puede decir que ninguna configuración sea eficiente.

Al haber tan pocas muestras, no se puede establecer con claridad una tendencia con la que analizar el efecto del overhead.

2.4 Análsis global

Con respecto a la escalabilidad de cada aplicación, hemos concluido que bt es la mejor, ya que sufre menos por el overhead a medida que se aumenta el número de procesos MPI, que son los que permiten mejorar la escalabilidad del programa.

Tras analizar más a fondo los picos que han ido apareciendo en las distintas gráficas de las distintas aplicaciones, hemos podido observar que en todos los casos estos repentinos picos aparecían en configuraciones en las que había 6 threads por proceso MPI. Esto nos lleva a pensar que estas configuraciones podrían estar generando un desbalanceo de los datos a nivel OpenMP, lo que conllevaría a una perdida de rendimiento. Esto se ve también reflejado en un incremento súbito del tiempo de ejecución y un decremento proporcional en el speedup y la eficiencia.

A continuación se discutirá si para estas aplicaciones resulta recomendable usar más MPIs por nodo o más threads por proceso y menos procesos. En el caso de la aplicación bt, es conveniente emplear entre 12 y 24 procesos MPI por nodo, lo que equivale a emplear una configuración con un mayor numero de procesos MPI y menos threads por proceso. Respecto a sp, se mantiene la misma conclusión que en la aplicación anterior, ya que para cualquier número de nodos las mejores configuraciones se obtienen al emplear más procesos MPI (12-24) por nodo. Para lu, en parte por la propia limitación de la aplicación, en todos los casos el número de threads por proceso es superior al número de procesos por nodo, con lo que para este caso no se tienen suficientes datos como para comprobar si realmente es mejor aumentar el número de threads por proceso.

Globalmente, es recomendable usar un mayor número de procesos MPI por nodo, por encima de threads por proceso. No obstante, como se ha observado, manteniendo un cierto equilibrio entre los 12 y 24 procesos por nodo.

3 Análisis coste energético

3.1 Comandos extras

Para la obtención de más información a la salida, utilizará el mismo comando `sacct` con los siguientes datos: Jobid, Tiempo CPU, Número de CPUS utilizadas, Número de CPUS pedidas en el script, Energía utilizada, frecuencia media de la CPU, Nombre del job, y Elapsed time.

El comando en cuestión es el siguiente:

```
sacct -j <jobid>
--format=JobID,CPUTime,NCPUS,ReqCPUS,
ConsumedEnergy,AveCPUFreq,JobName,Elapsed -p
```

Las ejecuciones para este apartado se realizaron con el script de shell `cost.sh`. Solo con un par de ejecuciones experimentales ya predefinidas al incio del proyecto. Aunque todas las ejecuciones se han realizado individualmente en este caso por motivos del comando `sacct`, se han reunido en un mismo archivo por motivos de comodidad a la hora de la entrega.

Para el cálculo de la potencia media P *(kW)* se ha empleado la siguiente fórmula:

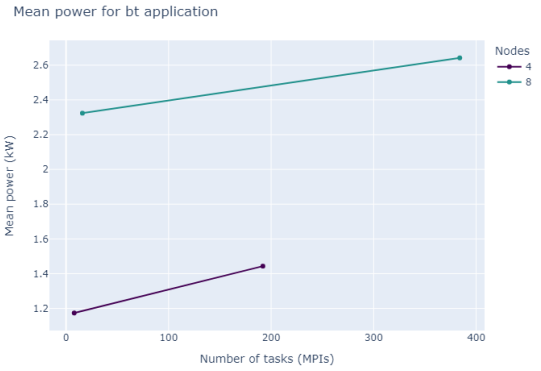
$$E = P \times T \tag{3}$$

donde el tiempo T corresponde al tiempo elapsed en segundos y la energía consumida E tiene unidades en *kJoules*. Bajo un criterio de referencia de $0.2kW/h$, el coste de cada ejecución es inmediato.

3.2 Aplicación bt

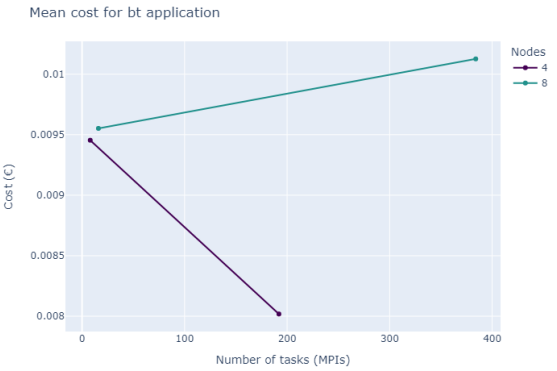
3.2.1 Potencia media

Respecto a la potencia media para la aplicación bt, observamos lo que parece un aumento lineal, teniendo en cuenta las reducidas muestras disponibles, con respecto al número de MPIs. De igual manera, una configuración con más nodos tendrá un mayor consumo energético.



3.2.2 Coste

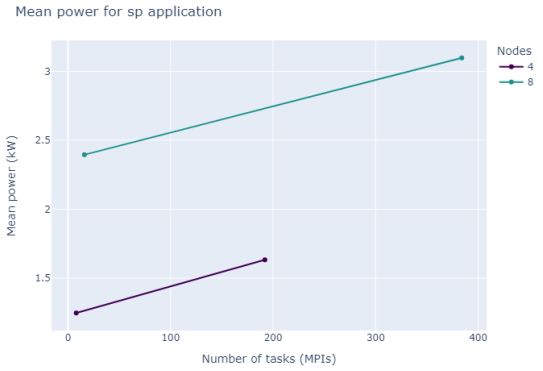
Si bien para una configuración con 8 nodos, más procesos MPI implica un aumento en el coste de ejecución, como este coste depende únicamente de la energía consumida y no del tiempo de ejecución, se observa que el segundo experimento para 4 nodos decae en coste al consumir menos energía.



3.3 Aplicación sp

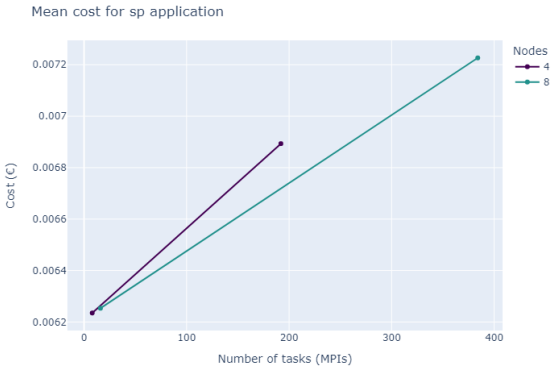
3.3.1 Potencia media

En la aplicación sp, hay una tendencia similar a la anterior aplicación, pero la diferencia subyace en el hecho de que la energía para los casos iniciales (menos MPIs) es menor que en la aplicación bt, en cambio el aumento tiene una mayor pendiente, haciendo que los casos con más MPIs tengan un mayor consumo energético.



3.3.2 Coste

El coste general para esta aplicación es menor. Sin embargo, para 8 nodos, el aumento respecto a MPIs es mayor que para la aplicación bt. Además destaca el hecho de que la configuración con 4 nodos y 192 procesos MPI aumenta de coste, en lugar de disminuir respecto a la configuración con 4 nodos y 8 procesos. Es decir, sucede lo contrario que con la aplicación bt, en este aspecto.



3.4 Análisis global

Para tener una visión más general, el coste total de estos ocho experimentos ha sido de aproximadamente 0.06376 €. Con un total de 0.03715 € de la aplicación bt y 0.02660 € de sp.

Como el coste depende directamente de la energía, podemos concluir que la aplicación más eficiente energéticamente es la aplicación sp. Puesto que representa un porcentaje menor de coste, en comparación con la aplicación bt, a pesar de tener una pendiente similar.

4 Análisis OpenMP

4.1 Tipo de paralelismo bt

En esta aplicación se está explotando principalmente el paralelismo de bucles con la directiva "!$OMP DO" para Fortran. En todos los casos se le añade la clausula "SCHEDULE(STATIC)" especificando el schedule como estático. En la mayoría de estos bucles se elimina la barrera implícita con un "!$OMP END DO nowait". Además de los bucles, también se están declarando regiones paralelas con "SHARED" como default mediante la directiva "!$OMP PARALLEL DEFAULT(SHARED)" pero privatizando diversas variables en la propia cláusula. Finalmente también se usan en algunos casos directivas como "!$OMP MASTER" o !$OMP ATOMIC".

4.2 Tipo de paralelismo sp

En este caso se explotan principalmente las regiones paralelas con y sin bucles, pero en el caso de los bucles hay varios casos en los que se usa "!$OMP PARALLEL DO" con "SHARED" como default pero privatizando a continuación diversas variables. Para los casos con "!$OMP DO" también se usa el "SCHEDULE(STATIC)" y también se elimina en alguna ocasión la barrera implícita. También se usan en algunos casos las directivas "!$OMP MASTER" y "!$OMP ATOMIC".

4.3 Tipo de paralelismo lu

Para la última aplicación también se explotan los bucles con "!$OMP DO SCHEDULE(STATIC)" eliminando en algunos casos la barrera implícita. También se paralelizan bucles con "!$OMP PARALLEL DO" de la misma forma que para sp-mz y lo mismo sucede con las regiones paralelas de "!$OMP PARALLEL DEFAULT(SHARED)". Una cosa que destaca de este programa es que hay un uso muy elevado de la directiva "!$OMP

MASTER". También se usa la directiva "!$OMP ATOMIC" pero destaca el uso de directivas como "!$OMP SINGLE" y "!$OMP FLUSH" que no habíamos visto en las otras aplicaciones. También resalta el uso de barreras no implícitas con "!$OMP BARRIER".

4.4 Análisis del schedule

Para el análisis del impacto del modo del `schedule` se modificó únicamente el código de la aplicación lu.

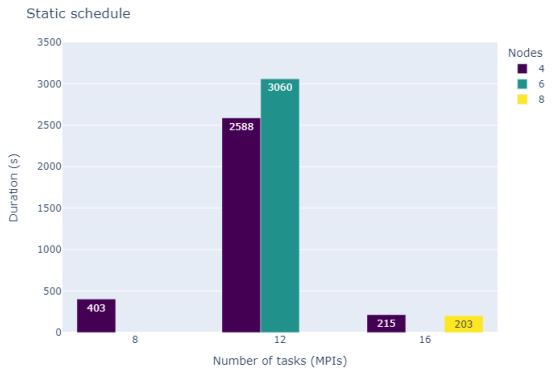


Figure 10: Tiempo de ejecución del schedule static

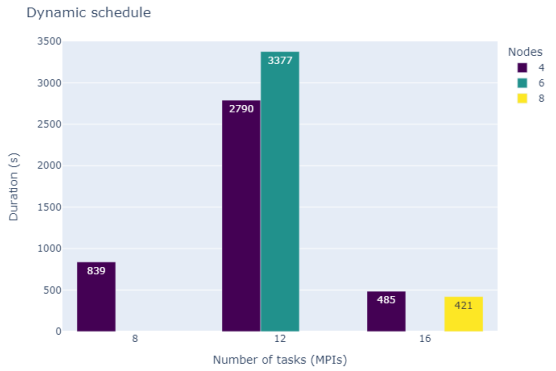


Figure 11: Tiempo de ejecución del schedule dynamic

Con una primera observación, salta a la vista el incremento general que supone configurar el schedule como `dynamic`. En el peor de los casos, que corresponde a la ejecución con 8 procesos MPI y 4 nodos, el tiempo llega a aumentar hasta 436 segundos. Esto nos indica claramente que para el programa lu es más adecuado usar un schedule `static` ya que la mejora que pudiese suponer un schedule `dynamic` se ve sobrepasada por el coste extra de aplicar tal cambio.

5 Conclusiones

Iniciando con la evaluación inicial, las dos mejores aplicaciones en términos de tiempo de ejecución corresponden a bt y sp. Si bien bt presenta una peor rango de tiempo ante sp, sp sufre un mayor efecto del overhead a partir de un cierto número de procesos MPI mucho antes y mucho más pronunciado que bt. En cuanto al speedup de las aplicaciones ocurre un suceso similar. Por lo que se ha comentado ya en apartados anteriores que la mejor aplicación en términos de escalabilidad es la aplicación de bt.

En el caso de las configuraciones, para tanto el tiempo de ejecución como el speedup, los mejores se obtienen con 8 nodos para 96 y 16 procesos MPI (para los casos bt y sp, y el caso lu respectivamente). Con lo que configuraciones con un mayor número de nodos claramente obtuvieron una mejor calificación dentro de estos parámetros. Aunque es adecuado mencionar que en el caso de la aplicación lu, al haber tan pocas pruebas experimentales, las conclusiones obtenidas para ese caso no se pueden considerar determinantes.

Para la eficiencia, vemos una ganancia mucho mejor en los casos de ejecución para la aplicación de bt, tanto sobre sp como lu (pues lu no cumple con la condición de superar el 70% de eficiencia mínima).

En términos de coste, la aplicación bt resulta ser más costosa en cuanto a financiación, pues ocupa un porcentaje mayor del coste total de las ejecuciones, como se ha computado en la sección de análisis de coste energético. Por lo que se debería continuar a tomar una decisión sobre la prioridad que se le da tanto a la escalabilidad de la aplicación como su coste asociado, ya que la aplicación bt es más escalable, mientras que sp

es más económica.

Para el análisis de schedule en la aplicación lu, nos damos cuenta al final que es más adecuado utilizar un schedule static, ya que no se observa mejora alguna al utilizar el método dynamic.

En conclusión podemos determinar que un mayor número de procesos MPI por nodo es más conveniente para el caso de estas aplicaciones que utilizar más threads por proceso. Aunque se ha notado que es necesario tener un balance entre 12 y 24 procesos por nodo.

6 Repartición del trabajo

El informe en su totalidad ha sido creado y revisado por los dos autores de forma igualitaria. Para empezar, se hicieron todos los experimentos que se requerían para este proyecto a excepción de los experimentos para el análisis del schedule. Para esta tarea, ambos estudiantes se conectaban en la aplicación *Discord* en la que se pueden llevar a cabo llamadas y compartir la pantalla del ordenador y de esta manera se iban turnando para ejecutar los diferentes casos en el supercomputador *MareNostrum* mediante los scripts que previamente habían creado.

Para el análisis de los resultados y el resto del informe se llevó a cabo la misma dinámica salvando algunas excepciones en las que los autores decidieron hacer individualmente algún apartado para más tarde comentarlo y revisarlo conjuntamente por temas de disponibilidad horaria principalmente. El análisis de los schedules fue el último en hacerse y se hizo de la misma manera al igual que el apéndice.

Appendices

A Configuraciones experimentales

Configuraciones experimentales para el aparatdo de la evaluación inicial.

Las siguientes tres tablas corresponden a las configuraciones de 4, 6 y 8 nodos para las aplicaciones bt y sp. La última tabla está dedicada a la aplicación lu al tratarse de un caso diferente de las otras dos.

Nodos	Procesos (MPI)	Threads por proceso
4	8	24
4	12	16
4	16	12
4	24	8
4	32	6
4	48	4
4	96	2
4	192	1

Nodos	Procesos (MPI)	Threads por proceso
6	12	24
6	18	16
6	24	12
6	36	8
6	48	6
6	72	4
6	144	2
6	288	1

Nodos	Procesos (MPI)	Threads por proceso
8	16	24
8	24	16
8	32	12
8	48	8
8	64	6
8	96	4
8	192	2
8	384	1

Nodos	Procesos (MPI)	Threads por proceso
4	8	24
4	12	16
4	16	12
6	12	24
8	16	24