PARALELISMO Y SISTEMAS DISTRIBUIDOS

PROYECTO HPC

Rebeca Torrecilla
46189534Z

Pau González
48102077S

Introducción

Este proyecto tiene como objetivo analizar el rendimiento de las aplicaciones incluidas en el paquete NASPB utilizando el software y el código que proporciona. Se estudian los principales factores que influyen en el rendimiento, como el tiempo de ejecución, los tipos de llamadas MPI y los esquemas de paralelización mediante OpenMP. Además, se evalúa cómo afecta la modificación del schedule de las funciones principales al rendimiento general del sistema.

Para ello, se generan los casos de clase D de tres aplicaciones concretas (bt-mz, sp-mz y lu-mz) utilizando la versión MZ-MPI (NPB3.3.1-MZ/NPB3.3-MZ-MPI). A partir de aquí, se estructuran varias secciones experimentales con distintos objetivos: identificar el nivel de paralelismo más eficiente, realizar un análisis preliminar del consumo energético de algunos de los jobs ejecutados, y comparar el efecto de variar el número de nodos utilizados durante la ejecución.

El proyecto también incluye un análisis gráfico de los distintos experimentos realizados, junto con tablas de resultados y el código (tanto original como modificado) empleado para llevarlos a cabo. Todos los gráficos, figuras y tablas se presentan en las secciones finales del documento.

1. Evaluación de escalabilidad

En el ámbito de la programación paralela, OpenMP y MPI representan dos enfoques distintos a la hora de gestionar tanto la memoria como la comunicación entre unidades de ejecución. OpenMP trabaja con hilos dentro de un mismo proceso que comparten un espacio de memoria común, siendo una opción eficiente para sistemas de memoria compartida. En cambio, MPI se basa en una arquitectura distribuida, en la que cada proceso dispone de su propia memoria privada y la coordinación entre ellos se realiza mediante el envío de mensajes.

El análisis inicial del proyecto se centra en medir el rendimiento de las aplicaciones BT-MZ, SP-MZ y LU-MZ, aplicando diferentes configuraciones que priorizan el uso de OpenMP o MPI. Estas pruebas se ejecutan utilizando 4, 6 y 8 nodos, con el objetivo de observar cómo influye el tipo de paralelismo en la eficiencia global del sistema.

¿Qué experimentos se llevarán a cabo?

Para los programas BT-MZ, SP-MZ y LU-MZ probaremos las siguientes opciones (para 4, 6 y 8 nodos):

- 1 CPU/task (caso MPI)
- 2 CPUs/task
- 4 CPUs/task
- 8 CPUs/task
- 16 CPUs/task
- 28 CPUs/task
- 56 CPUs/task
- 112 CPUs/task (caso OpenMP)

¿Por qué se han elegido estos recursos?

Cuando se trabaja con una carga fija de tareas, el objetivo de la programación paralela es completarlas en el menor tiempo posible, utilizando los recursos de forma eficiente. Para valorar el rendimiento, se emplean métricas como el tiempo de ejecución, el speedup o la eficiencia, que ayudan a determinar si un mayor número de recursos realmente acelera el proceso. En este contexto, también se busca comparar si es más conveniente emplear OpenMP o MPI según las características del entorno y la configuración del sistema.

No obstante, hay ciertos factores que condicionan los beneficios del paralelismo, como el overhead que introduce, el volumen de trabajo, el equilibrio en la distribución de las tareas o los propios límites del hardware disponible. Por ello, el análisis que se presenta considera diferentes configuraciones: dos experimentos extremos (con el mínimo y máximo de nodos) y seis configuraciones intermedias que reflejan tanto enfoques orientados a MPI como a OpenMP.

¿Cuáles son las medidas utilizadas para la evaluación del rendimiento?

TIEMPO DE EJECUCIÓN

El tiempo de ejecución representa la duración total que necesita un programa para completarse, y es una métrica clave a la hora de analizar el rendimiento en entornos paralelos. Comparar este valor entre distintas configuraciones —o frente a una ejecución secuencial— permite evaluar qué tan efectiva es una estrategia de paralelización. Para asegurar la fiabilidad de los resultados y evitar inconsistencias debidas a variaciones puntuales, cada combinación de parámetros se ejecuta hasta en tres ocasiones, y se toma como referencia la media de esos tiempos.

SPEEDUP

Al analizar los resultados obtenidos con distintas combinaciones de recursos, uno de los focos principales es determinar cuánto mejora el rendimiento de la aplicación al paralelizarla. Esta mejora se conoce como speedup y se calcula comparando el tiempo que tarda el programa en ejecutarse de forma secuencial con el tiempo que necesita cuando se aprovechan varios núcleos o procesadores. El valor resultante depende en gran medida de cómo se distribuyen las tareas entre los recursos disponibles, y sirve para cuantificar de forma clara la ganancia obtenida gracias a la paralelización. El Speedup se define como:

$$Speedup(p) = \frac{Tiempo(1)}{Tiempo(p)}$$

En nuestro caso, cogeremos como caso T(1) la ejecución con 112 tasks y 1 thread por task (un nodo completo).

EFICIENCIA

Junto al speedup, otra métrica importante que nos ayuda a decidir la cantidad óptima de recursos a utilizar es la eficiencia. Esta mide qué tan bien estamos aprovechando los recursos disponibles, y se calcula como el cociente entre el speedup y el número de núcleos utilizados, es decir, la cantidad de recursos empleados (p).

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

Es crucial tener en cuenta que tanto el speedup como la eficiencia deben evaluarse dentro de un contexto específico, ya que un speedup —por muy alto o bajo que sea— solo tiene sentido cuando se compara con la cantidad de recursos asignados. El overhead, que generalmente aumenta con el número de núcleos, provoca que la eficiencia disminuya a medida que se incrementa la cantidad de recursos utilizados. Este overhead se traduce en una ralentización, afectando negativamente al rendimiento. Más adelante,

analizaremos cómo estos factores influyen en el tiempo total de ejecución, así como en otros aspectos como el consumo energético y los costes asociados.

Resultados según kernels

BT-MZ

- Este kernel, que corresponde a las siglas de *Block Tri-diagonal solver, Multiple Zones*, se caracteriza por tener zonas de tamaños desiguales dentro de una misma clase de problema, con un número mayor de zonas conforme aumenta el tamaño de la clase. El objetivo principal del problema es resolver un sistema de ecuaciones tridiagonales por bloques.

En la Figura 1 podemos ver cómo el tiempo de ejecución decrece a medida que se ejecuta en más nodos. Esto ocurre en todos los casos, excepto en el caso MPI (1 CPUs/task), pues parece que el número de nodos no afecta al tiempo de ejecución. Esto puede deberse a la naturaleza del BT-MZ: al presentar zonas de tamaño desigual, es posible que alguna zona defina el camino crítico y limite la mejora del tiempo total, independientemente del número de cores utilizados. En cambio, el caso OpenMP (112 CPUs/task) es el que más mejora su tiempo de ejecución respecto al tiempo inicial. No obstante, aunque sea el que más mejora, sigue tardando más que el resto de configuraciones (exceptuando el caso MPI).

Según la Figura 2, el speedup del caso MPI es constante, lo cual era de esperar, ya que el tiempo no mejoraba con el número de cores. Además, el caso OpenMP presenta el segundo peor speedup (después del caso MPI). Los mejores resultados en cuanto a speedup los obtienen las configuraciones intermedias, concretamente las de 4 CPUs/task y 8 CPUs/task.

Si observamos el speedup en función del nombre de CPUs/task como en la Figura 10, podemos ver como la cantidad de cores utilizados resulta determinante para conseguir un valor de speedup mayor pese a que los tres casos (4, 6, y 8 cores) siguen un mismo patrón lineal, consiguiendo un máximo entre los 4 y 8 CPUs/task antes de decrecer proporcionalmente al aumento de recursos. El decrecimiento se puede asociar seguramente al overhead creado a partir de un, cada vez más, elevado coste de sincronización entre hilos, *race conditions*, partes del código no paralelizables, etc.

En cuanto a las eficiencias, observables en la Figura 3 y en la Figura 13, podemos decir que todos los casos mantienen siempre una eficiencia superior al 70% excepto tres casos (respecto al número de cores):

- El caso MPI:
  Era de esperar. Si a más recursos no mejora el tiempo de ejecución, pues no está siendo eficiente. Su eficiencia decrece conforme aumenta el número de nodos.

- El caso OpenMP:
  Se mantiene en torno al 52%. A pesar de ser el que más mejoras presenta en cuanto al tiempo de ejecución en función del número de núcleos, sigue siendo mucho más lento que muchos otros casos

y, como se ha mencionado antes, su speedup es de los peores.

- El caso de 2 CPUs/task:
  Se mantiene por encima del 70% hasta que se utilizan 8 núcleos. Se puede observar en la Figura 1 que no hay mejora en el tiempo de ejecución en este caso entre 6 nodos y 8 nodos, lo que explica la pérdida de eficiencia.

Si analizamos también la eficiencia respecto al número de CPUs/task, observamos en la Figura 13 que, a medida que aumentamos los recursos ésta métrica decrece en todos los casos de nodos, consiguiendo un máximo entre las 4 y 8 CPUs/task y dejando de ser eficiente según nuestros estándares a partir de las 70 CPUs/task aproximadamente.

En resumen, los casos que mejor eficiencia presentan son los que mejor speedup presentan, los de 4 CPUs/task y 8 CPUs/task.

SP-MZ

- Este kernel corresponde a las siglas de *Scalar Penta-diagonal, Multiple Zones* y se caracteriza por tener zonas de tamaño uniforme dentro de una clase de problema, con un número creciente de zonas a medida que aumenta el tamaño de la clase. El objetivo es resolver un sistema de ecuaciones pentadiagonales escalares.

En la Figura 4 se puede observar cómo para la gran mayoría de casos el tiempo de ejecución con 4 nodos está entre 200 y 100 segundos, mientras que para 8 está entre 150 y 50. El caso que destaca es el caso OpenMP, que empieza con 450 segundos para 4 nodos y acaba con unos 240 para 8 nodos. Esto se puede deber al overhead asociado a la creación y sincronización de los hilos. Además, para el caso MPI, no hay prácticamente diferencia entre usar 6 cores y 8 cores en cuanto al tiempo de ejecución.

Fijándonos en la información que nos da la Figura 5 sobre el speedup, podemos ver cómo la gran mayoría mejoran linealmente, menos el MPI, que se "estanca". Al no haber mejora respecto al tiempo de ejecución entre 6 nodos y 8 nodos, el speedup tampoco mejora. Es curioso ver cómo para este caso también los mejores speedups son los de los casos de 4 CPUs/task y 8 CPUs/task.

Observando ahora el speedup en función de las CPUs/task en la Figura 11, distinguimos un patrón similar al caso anterior del BT-MZ con un máximo en las 4 CPUs/task antes de decrecer.

En cuanto a la eficiencia, podemos mirar la Figura 6 y la 14. En la primera vemos cómo el caso OpenMP presenta la peor eficiencia, seguramente por el overhead dada la naturaleza de sp-mz, estando siempre por debajo del 70%. El caso MPI presenta una eficiencia decreciente del número de nodos de 6 a 8, pues al no mejorar al asignar más recursos, su eficiencia empeora y queda bajo el límite de eficiencia considerado. El resto de casos tienen una eficiencia aceptable.

En la segunda gráfica vemos como en los tres casos de cores utilizados se vuelven a conseguir los mejores resultados con 4 CPUs/task descendiendo a partir de ahí casi de forma lineal. Es curioso como, para esta aplicación, se tiene un valle mucho más agudo en las 16 CPUs/task del que se tuvo con BT-MZ cosa que puede deberse a la naturaleza o funcionamiento del SP-MZ.

También sucede algo anómalo, pues se pueden observar eficiencias mayores que 1. Esto puede deberse a que, al tener zonas de tamaño uniforme dentro de una clase de problema, entonces puede paralelizar extremadamente bien.

LU-MZ

- LU-MZ, que corresponde a *Lower-Upper de Lower-Upper Gauss-Seidel solver, multiple zones*, se caracteriza por tener zonas de tamaño uniforme dentro de una clase de problema, con un número constante de zonas para todas las clases de problemas. El objetivo es resolver un sistema de ecuaciones lineales mediante el algoritmo de factorización LU.

En la Figura 7 vemos cómo la gran mayoría de los casos decrecen su tiempo de ejecución conforme se aumenta el número de nodos. Para el caso OpenMP y el caso de 16 CPUs/task pasa algo diferente. Con 6 nodos tarda más que con 4, pero con 8 nodos tarda menos que con 4. Esto puede deberse a que con 6 nodos la paralelización no compensa el overhead, pero que para 8 nodos sí se compensa la paralelización con el overhead.

Fruto de esta peculiaridad, vemos que en la Figura 8 los speedups correspondientes a estos dos casos especiales no son siempre crecientes, sino que el speedup decrece de 4 a 6 nodos y luego sí crece de 6 a 8 nodos. En la Figura 12 también observamos que los tres casos no siguen un mismo patrón tal y como sucedía en las otras dos aplicaciones, mostrando un comportamiento más aleatorio pero que, en general, también se puede definir como decreciente a partir del máximo sobre las 4 u 8 CPUs/task.

En lo relacionado con la eficiencia, podemos ver en la Figura 9 que, aunque para algunos otros casos haya una caída de la eficiencia cuando se usan 6 nodos, la caída es mucho más significativa para los casos de 112 CPUs/task y 16 CPUs/task. Todos los casos se mantienen siempre por encima del 70%, excepto el caso OpenMP, que para 6 nodos queda por debajo del 50%.

Si miramos la eficiencia respecto a las CPUs/task (Figura 15), veremos también un patrón errático, con una diferencia importante entre las 56 y 112 CPUs/task en todos los casos.

Vuelve a suceder que hay eficiencias superiores a 1; de hecho, todas lo son, excepto la del caso OpenMP para 6 nodos. Esto puede pasar porque la referencia base para el cálculo de eficiencia no es del todo justa, pues es el tiempo de 1 nodo con 112 CPUs vs. 1 hilo, en vez de 1 nodo con 1 core.

Evaluación general

Viendo las Figuras 10, 11 y 12 podemos ver que, en general, el speedup según los CPUs por task siguen un patrón similar para todos los nodos, siendo el caso de 8 cores el que más tiene más speedup y el 4 cores el que menos. Mirando estas gráficas vemos que el speedup, para cualquier número de nodos, suele alcanzar el "peak" para los casos de 8 CPUs por task o 4 CPUs por task. Para los casos de BT-MZ y SP-MZ vemos como estos dos casos tienen un tiempo de ejecución menor (Figuras 1 y 4), mayores speedups (Figuras 2 y 5) y mayor eficiencia (Figuras 3 y 6). Aunque para los casos de LU-MZ el análisis puede parecer más enrevesado, no es difícil ver cómo, en promedio, estos dos mismo casos, 8 CPUs por task o 4 CPUs por task, son los que tienen menor runtime, speedup y eficiencia (Figuras 7, 8 y 9 respectivamente). Ahora haciendo una interpretación más concreta:

- BT-MZ: La configuración con menor tiempo de ejecución y mayor speedup es aquella que usa 8 nodos y 8 CPUs por task. Además, usar 8 CPUs por task es más eficiente cuando se usan 8 cores, como se puede observar en la Figura 3, 10 o 13.

- SP-MZ: La configuración que alcanza el menor runtime y el mayor speedup es la que utiliza 8 nodos con 4 CPUs por tarea. Además, al trabajar con 8 cores, la opción de 4 CPUs por task resulta ser la más eficiente, como se muestra en la Figura 6.

- LU-MZ: Aunque pueda haber empates en ciertos momentos, como se ha dicho anteriormente, las configuraciones que más destacan son las que tienen 4 CPUs por task u 8 CPUs por task. De entre estas dos, la que menor runtime tiene y mayor speedup tiene es la de 8 CPUs por task con 8 cores, siendo las más eficiente usando este número de cores.

En los tres casos analizados —BT-MZ, SP-MZ y LU-MZ—, ni la configuración puramente MPI ni la puramente OpenMP ofrecen el mejor rendimiento. De hecho, suelen situarse entre las opciones con peor comportamiento, tanto en tiempo de ejecución como en eficiencia.

Por tanto, las configuraciones híbridas, donde se equilibran el número de tareas y CPUs por tarea (como 4 o 8 CPUs/task), permiten un mejor aprovechamiento del paralelismo a varios niveles, reduciendo la comunicación innecesaria y al mismo tiempo evitando cuellos de botella internos.

2. Análisis de energía mediante "ear"

En este apartado vamos a analizamos el power relativo, el coste de los experimentos y los KW/h de los experimentos en función del número de cores y CPUs por task.

Power relativo

Cabe aclarar que esta medida resulta >1 para todos los casos debido a que los nodos puede consumir más gracias a la disponibilidad de otros componentes.

BT-MZ

Mirando las Figuras 16 y 17 vemos cómo la potencia relativa es menor cuantos más nodos usamos y cómo, en general, los mínimos se alcanzan con configuraciones que tengan un número relativamente pequeño de CPUs por task, excepto cuando se usan 4 nodos, que en este caso la curva decrece desde los 16 hasta los 56 para luego volver a crecer. Consecuentemente, lo óptimo sería usar 8 nodos y 1 CPU por task.

SP-MZ

En las Figuras 18 y 19 se observa lo contrario al BT-MZ, pues la potencia relativa es menor cuantos menos nodos usamos. Además, se ve un patrón similar para las curvas de la Figura 19, todas decrecen hasta los 56 CPUs por task y luego crecen. En la Figura 18 también se ve como este es siempre el caso con menor relative power. Por tanto, la mejor configuración para obtener un valor bajo es aquella que use 4 nodos y 56 CPUs por task.

LU-MZ

En la Figura 20 se ve como el relative power mínimo se alcanza con 112 CPUs por task, es decir, con el caso OpenMP, para 6 y 8 cores, mientras que para 4 cores es menor el caso de 56 CPUs por task, pero no por mucho. En la Figura 21 vemos algo curioso, esta vez el relative power no crece o decrece con el número de nodos, si no que se ve como las configuraciones que usan 6 nodos (el valor intermedio) acostumbran a tener un menor relative power. En consecuencia, la configuración óptima es aquella que use 6 nodos y 112 CPUs por task.

Coste de los experimentos y KW/h

Tiene sentido analizar estas dos medidas de forma conjunta, pues el coste de los experimentos no es más que los KW/h multiplicados por 0.2.

BT-MZ

En la Figura 22 se ve que las configuraciones que menos consumen son aquellas que usan 4 u 8 CPUs por task. De la Figura 23 no se puede decir que haya una configuración de nodos que haga ahorrar mucho, de hecho, la configuración con 8 cores es la más cara en general pero por muy poco. De la Figura 23 también podemos sacar la misma conclusión que de la 22, pues se ve como sobre las 4 a 8 CPUs/task tenemos el menor coste.

SP-MZ

En la figura 24 observamos que el caso 112 CPUs por task (OpenMP) es el más caro con diferencia, mientras que los de 4, 8 y 56 se mantienen por debajo del resto. La Figura 25 ayuda a entender que hay mínimos locales en esta configuraciones y luego el máximo para 112. También se ve como la de 8 cores es más cara para todos los casos menos para los de 28 y 4 CPUs por task.

LU-MZ

En la Figura 26 se ve como las configuraciones que usan 112 CPUs por task, igual que para SP-MZ, son más costosas, habiendo incluso un pico cuando se usan 6 nodos. Mientras que, mirando esta figura mencionada y la 27 vemos como las que usan 8 CPUs por task se mantienen menos costosas y, fijándonos sólo en la Figura 27 observamos como no acaba de estar claro que configuración de cores es la más barata, excepto para el caso OpenMP, que se ve que es la más cara si se usan 112 CPUs por task. Por tanto, la configuración que habíamos asumido óptima para la potencia relativa, es realmente la más cara.

Interpretación

El análisis del power relativo, coste en euros y kWh consumidos revela que no siempre existe una correlación directa entre eficiencia energética y número de nodos o CPUs por task. En muchos casos, la configuración que resulta óptima en términos de potencia relativa no lo es en coste total, como se ha visto especialmente en LU-MZ, donde el menor power relativo se daba en el caso de 112 CPUs por task (OpenMP), pero al mismo tiempo este resultó ser el más costoso.

Esto se puede explicar porque el power relativo mide cuán cerca estamos del máximo teórico de consumo (normalizado respecto a un nodo de referencia), mientras que el coste total tiene en cuenta cuánto tiempo ha estado corriendo el experimento y cuántos nodos se han utilizado. Es decir, una configuración que consume menos potencia de forma instantánea puede terminar siendo más cara si dura más tiempo o usa más nodos.

Otro punto interesante es que los patrones no son universales:

- BT-MZ se comporta de forma opuesta a SP-MZ en cuanto al número de nodos ideal para reducir el consumo.

- LU-MZ, por su parte, muestra comportamientos más impredecibles con picos puntuales y dependencias claras del número de CPUs por task.

Todo esto refuerza la idea de que no hay una configuración única que optimice simultáneamente todas las métricas, y que la elección óptima depende de qué se quiera priorizar: minimizar la potencia instantánea, el coste económico, o el tiempo de ejecución. Por tanto, es esencial evaluar varios indicadores de forma conjunta antes de decidir qué configuración adoptar para un entorno de computación eficiente y rentable.

3. Escalabilidad y métricas de performance

En este apartado vamos a comparar la evaluación de las métricas CPI y Memory bandwidth de las diferentes configuraciones para cada tamaño de número de nodos. Hay que saber que el CPI mide cuántos ciclos tarda en ejecutarse cada instrucción y el memory bandwidth mide la cantidad de datos que se mueven por segundo.

CPI

En el caso del kernel BT-MZ, como se observa en la Figura 34, el CPI (Cycles Per Instruction) crece de forma prácticamente lineal a medida que se incrementa el número de CPUs por task. Las configuraciones con 4 nodos tienden a presentar valores más altos de CPI en comparación con las de 8 nodos, que generalmente se mantienen por debajo. Todas las configuraciones alcanzan un valor máximo cercano a 0.45 cuando se llega a 112 CPUs por task.

Para el kernel SP-MZ, mostrado en la Figura 35, se mantiene esta tendencia creciente del CPI, aunque de manera menos lineal. De nuevo, las configuraciones con 4 nodos muestran valores superiores al resto, mientras que las de 8 nodos se sitúan por debajo en la mayoría de casos. En este escenario, el CPI máximo alcanza aproximadamente el valor de 1 con 112 CPUs por task.

En el kernel LU-MZ, representado en la Figura 36, se observa un comportamiento particular para la configuración con 4 cores. En lugar de empezar con un CPI bajo como en los otros dos casos (alrededor de 0.20), comienza con un valor relativamente alto, aproximadamente 0.47. A partir de ahí, se observan dos picos: uno en 4 CPUs por task (alrededor de 0.55) y otro en 28 CPUs por task (cerca de 0.60). Finalmente, con 112 CPUs por task, el CPI supera los 0.65. Por su parte, las configuraciones con 6 y 8 nodos siguen una evolución más regular y creciente, con pequeñas irregularidades en valores bajos de CPUs por task, alcanzando máximos en torno a 0.63.

Interpretación

El crecimiento del CPI con el aumento de CPUs por task puede deberse a una mayor contención por recursos compartidos, como la memoria o la interconexión entre procesos, especialmente en arquitecturas multinúcleo.

Además, en configuraciones con pocos nodos (como 4), al aumentar el número de CPUs por task, se tiende a saturar más rápidamente la capacidad de procesamiento compartido de cada nodo, lo que puede traducirse en un mayor CPI. Por otro lado, configuraciones con más nodos distribuyen mejor la carga de trabajo, reduciendo la presión sobre recursos compartidos, lo que explica que sus valores de CPI sean, en general, más bajos.

El comportamiento atípico en el caso de LU-MZ con 4 cores puede indicar una combinación de ineficiencia estructural del kernel y una mala asignación de recursos para ese tamaño de problema, lo que se agrava al aumentar las CPUs por task.

Memory bandwidth

En el caso del kernel BT-MZ, como se muestra en la Figura 37, el memory bandwidth alcanza sus valores máximos cuando se emplean 2 o 4 CPUs por task, mientras que el mínimo se observa en la configuración con 56 CPUs por task.

Para el kernel SP-MZ, ilustrado en la Figura 38, el comportamiento es similar al de BT-MZ. Sin embargo, el mínimo se produce con 28 CPUs por task, y, de forma destacable, en la configuración con 6 nodos, el máximo se alcanza con 1 CPU por task, es decir, en el caso puramente MPI.

En el caso del kernel LU-MZ, representado en la Figura 39, también se observan máximos en torno a 2 o 4 CPUs por task, y los mínimos se registran en la configuración con 112 CPUs por task. No obstante, se presenta un fenómeno interesante: en todas las configuraciones de número de nodos, al utilizar 28 CPUs por task, se produce un pico en el memory bandwidth. Esto rompe la tendencia general de disminución (aparentemente cuadrática), introduciendo una anomalía que se desvía del patrón esperable.

Interpretación

El memory bandwidth indica cuántos datos pueden transferirse entre la memoria y el procesador por segundo. Cuanto mayor es, mejor se está aprovechando la arquitectura del sistema.

En general, se observa que el bandwidth disminuye cuando se usan más CPUs por task. Esto puede pasar porque más hilos acceden a la memoria al mismo tiempo, lo que genera conflictos y cuellos de botella, reduciendo el rendimiento.

Por otro lado, configuraciones con menos CPUs por task, como el caso MPI puro (1 CPU por task), suelen acceder a la memoria de forma más ordenada y eficiente, sacando mayor partido al ancho de banda disponible.

El pico en 28 CPUs por task que aparece en LU-MZ es interesante: sugiere que en ese punto concreto hay un equilibrio entre paralelismo y eficiencia, tal vez por cómo se reparten los hilos o el tamaño del problema. Este tipo de comportamientos muestran que el rendimiento no siempre sigue una tendencia clara, y que probar diferentes configuraciones puede ayudar a encontrar la más eficiente.

Evaluación general

Primero hay que dejar claro cómo se relacionan estas métricas con el número de CPUs por task y de nodos. El CPI aumenta junto con el número de CPUs por task (excepto las irregularidades comentadas para para el LU-MZ). En cambio, el memory bandwidth se asemeja a una función cuadrática con un mínimo cerca de los 56 CPUs por task (excepto por lo comentado para el LU-MZ). También vemos en las figuras mencionadas anteriormente como con 4 cores el CPI es mayor pero también lo es la otra métrica. En cambio, para 8 cores ambas métricas son menores. De esta manera, es fácil justificar que la que mejor mantiene una relación entre un CPI bajo y un memory bandwidth alto es aquella configuración con 6 cores.

Teniendo en cuenta que el CPI mide cuántos ciclos tarda en ejecutarse cada instrucción y el memory bandwidth mide la cantidad de datos que se mueven por segundo, nos interesa que el CPI sea bajo y que el memory bandwidth sea alto. En consecuencia el procesador trabaja eficientemente (CPI bajo) y la memoria no es un cuello de botella (memory bandwidth alto). Por tanto, nos interesa tener un compromiso entre un memory bandwidth alto y un CPI bajo. Sabiendo las relaciones entre esta métricas y los CPUs por task podemos asumir que queremos un número de CPUs por task que esté entre el principio y antes de los 56. Para BT-MZ y SP-MZ se podría argumentar que las configuraciones que mejor mantienen este compromiso son, en su gran mayoría las que tienen entre 2 y 4 CPUs por task y 6 cores.

4. Algoritmos y métricas de GPUs

Tal y como se nos pide en el enunciado de esta última sección, repetimos el análisis de los experimentos con tensorflow de la tercera sesión de laboratorio añadiendo esta vez el análisis del power de la GPU teniendo en cuenta un máximo teórico de 700W.

Siguiendo la línea del trabajo en general, también mandamos a ejecutar todos los experimentos en una sola comanda y obtenemos todas las métricas que se muestran en la Tabla 1 (adjunta en el anexo).
En ésta podemos observar aplicaciones asociadas a diferentes modelos de redes neuronales (DenseNet, VGG...) con distintas políticas de ejecución y variantes de precisión (*mixed, disable*, etc.) con algunas de las métricas siendo el tiempo total, la potencia, energía total, uso de la GPU, etc. A continuación destacamos los aspectos más relevantes que hemos encontrado.

Consumo de energía y potencia GPU

Aplicación	G-POW Total (W)	% sobre 700W
DenseNet121 mixed	282,00	40,3%
DenseNet121 disable	342,91	49,0%
DenseNet121 (default)	344,12	49,16%
VGG19 mixed	414,11	59,2%
VGG19 disable	374.66	53,5%
VGG 19 (default)	369,43	52,8%
ResNet50 mixed	284,13	40,6%
ResNet50 disable	400,02	57,1%
ResNet50 (default)	351,43	50,2%

Un punto clave resulta el consumo de energía y potencia de la GPU por parte de las diferentes ejecuciones. En la tabla superior podemos visualizar más fácilmente el power total consumido por cada una (en watts) junto al porcentaje respecto al máximo teórico.

Lo primero que nos llama la atención es que contamos con tres redes neuronales con tres variantes de precisión cada una: *mixed, disable* y *default*. Sabemos que las diferentes variantes cuentan con sus respectivas ventajas y desventajas y, en lo relativo a la energía consumida, tenemos:

- *default*: es el estándar habitual de la mayoría de frameworks pese a consumir mayor memoria y realizar más operaciones por segundo (más lento) que puede llevar también a un mayor consumo de energía.
- *mixed*: las operaciones de esta variante mezclan los tipos *float16* y *float32* lo que conlleva un menor uso de memoria, mayor velocidad y, por ende, menor consumo energético (pese a contar con un riesgo leve de pérdida de precisión).
- *disable*: aquí se fuerza a usar únicamente *float32* lo que comporta más tiempo de ejecución y mayor consumo energético.

Teniendo esto presente nos es lógico ver que, tanto para DenseNet como para ResNet, la variante con menos consumo resulta la *mixed* al realizar operaciones más rápidamente mientras que, las otras dos variantes, presentan valores bastante similares (con más consumo en *disable* para la red ResNet y *default* para la red *DenseNet*). Vemos también que, para el caso de VGG, la energía consumida según la variante de precisión no presenta un cambio demasiado drástico entre ellas, siendo la *mixed*, a diferencia de los otros modelos, aquella con mayor consumición.

Otra cosa a destacar sobre este tópico es el porcentaje utilizado respecto al máximo teórico. Vemos que todas las ejecuciones rondan el intervalo de 40-60% lo que indica que la GPU está funcionando correctamente, manteniéndose en un régimen seguro y eficiente desde el punto de vista térmico y energético.

Tiempo de ejecución y eficiencia energética

Aplicación	Tiempo (s)	Energía (J)	Energía por segundo (W)
DenseNet121 mixed	344,21	332.158	964,99
DenseNet121 disable	359,80	342.598	1.031,26
DenseNet121 (default)	332,10	395.635	1.191,31
VGG19 mixed	93,57	98.571	1.053,42
VGG19 disable	279,44	296.788	1.062,08
VGG 19 (default)	188,05	194.366	1.033,60
ResNet50 mixed	251,88	244.248	969,71
ResNet50 disable	287,44	311.244	1.082,81
ResNet50 (default)	245,70	249.083	1.013,78

En la tabla superior mostramos la aplicación junto a su tiempo de ejecución, su energía (en joules) y una estimación de la eficiencia relativa, obtenida a partir de la energía entre el tiempo.

Vemos que las medidas varían según el modelo y la variante de precisión aunque, en todas, podemos observar que la variante *mixed* obtiene mejores tiempos que *disable,* ya sea con mucha diferencia (como en el modelo VGG) o por no demasiada (como en el modelo DenseNet). Destacamos pues, un mínimo en VGG *mixed*, que presenta un tiempo de ejecución de tan solo 93,57 segundos y un máximo de 359,80 segundos, en DenseNet *disable* que deja claro el drástico salto entre ambos casos.

Comentando ahora la estimación de la eficiencia de éstos, observamos que el consumo promedio de energía se mantiene en el rango de 960 a 1.080 W efectivos (no GPU solamente, sino el sistema completo). También, como sucedía al analizar el consumo de energía de cada ejecución, las variantes *mixed* de DenseNet y ResNet presentan valores menores respecto a sus otras variantes mientras que, para VGG, independientemente de la variante de precisión, no se consigue mucha diferencia. Respecto a estos datos, contamos con un mínimo de 964,99W de DenseNet *mixed* y un máximo por ResNet *disable*, con valor 1.082,81W.

Utilización de GPU y frecuencia

Aplicación	G-Util (%)	G-Freq (GHz)
DenseNet121 mixed	91% / 77%	1,95
DenseNet121 disable	94% / 85%	1,97
DenseNet121 (default)	95% / 86%	1,97
VGG19 mixed	91% / 54%	1,98
VGG19 disable	98% / 74%	1,97
VGG 19 (default)	96% / 78%	1,98
ResNet50 mixed	93% / 75%	1,96
ResNet50 disable	96% / 76%	1,97
ResNet50 (default)	93% / 87%	1,95

La segunda columna, correspondiente a G-UTIL, muestra los porcentajes de GPU y memoria utilizados por la aplicación.

La utilización de GPU (segunda columna) se mantiene en valores elevados en todas las ejecuciones, superando el 90% en todos los casos, lo cual es indicativo de una buena ocupación de los recursos computacionales. La utilización de memoria GPU, aunque algo más variable, también se mantiene en valores aceptables.

Las frecuencias de operación (tercera columna) ocupan un rango de 1,95-1,98 GHz lo cual evidencia un comportamiento térmico saludable.

Conclusión

El análisis completo muestra que, por lo general, las configuraciones con precisión mixta no solo presentan mejores tiempos de ejecución, sino que también reducen el consumo energético sin comprometer el uso de recursos de GPU. Además, todas las ejecuciones se mantienen dentro de márgenes seguros de potencia, muy por debajo del límite teórico de 700W, a la vez que utilizan una cantidad de energía considerable, demostrando eficiencia. Esto sugiere un buen aprovechamiento del hardware, con posibilidades de optimización adicional si se buscase reducir aún más el consumo sin pérdida significativa de rendimiento.

En resumen, el uso de precisión mixta parece ser la mejor opción para lograr una mejor eficiencia energética y un alto rendimiento en entornos HPC como el entrenamiento de redes neuronales profundas.

5. Trabajo en grupo

Dado que la realización del trabajo coincidió con las fechas de Semana Santa, los primeros días se dedicaron a ejecutar los diferentes programas con sus respectivas combinaciones de recursos. Una vez obtenidos los resultados, más o menos a mitad de la semana, se elaboraron documentos para organizar mejor la información. Gracias a esos documentos, pudimos generar las gráficas que facilitaron una mejor visualización de los datos, lo que nos permitió realizar, durante los últimos días de Semana Santa, el análisis adecuado para cada apartado.

Se ha optado no por una división del trabajo, sino por la realización en paralelo y conjunta de las secciones, lo que ha permitido compartir observaciones en tiempo real y agilizar tanto la redacción como la validación de los resultados. Esta forma de trabajo colaborativa ha favorecido una mejor comprensión global del proyecto por parte de todos los miembros del equipo.

6. Anexo

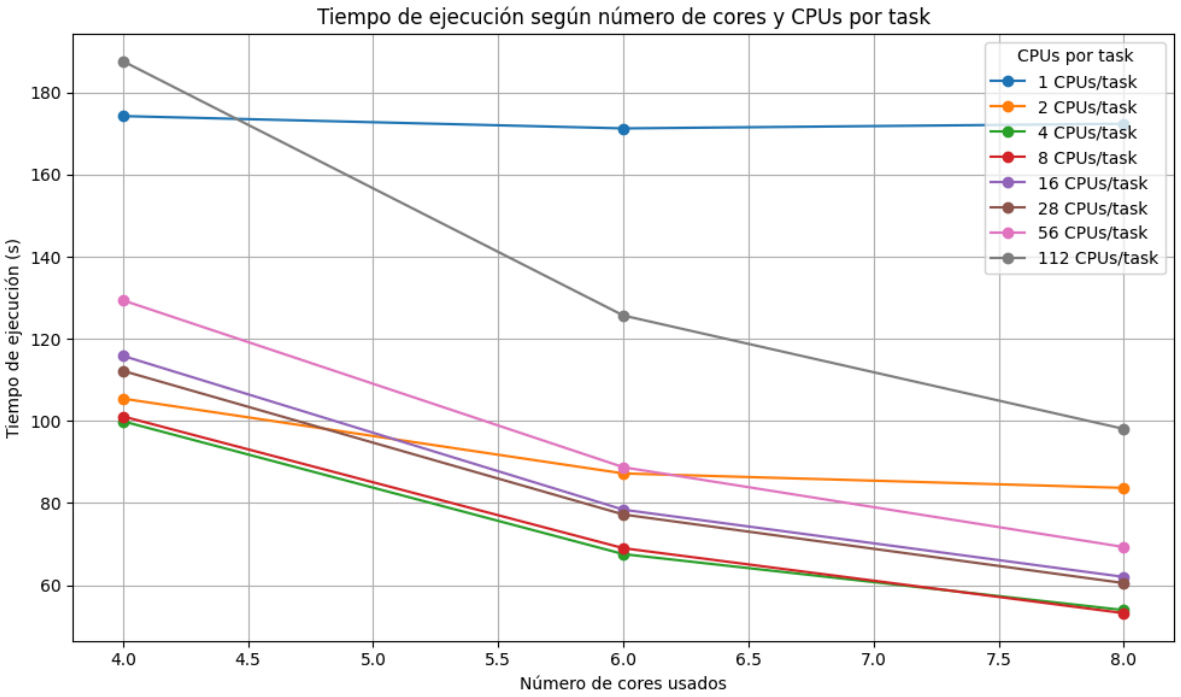

Figura 1: Tiempo de ejecución según número de cores y CPUs por task para la aplicación bt-mz.

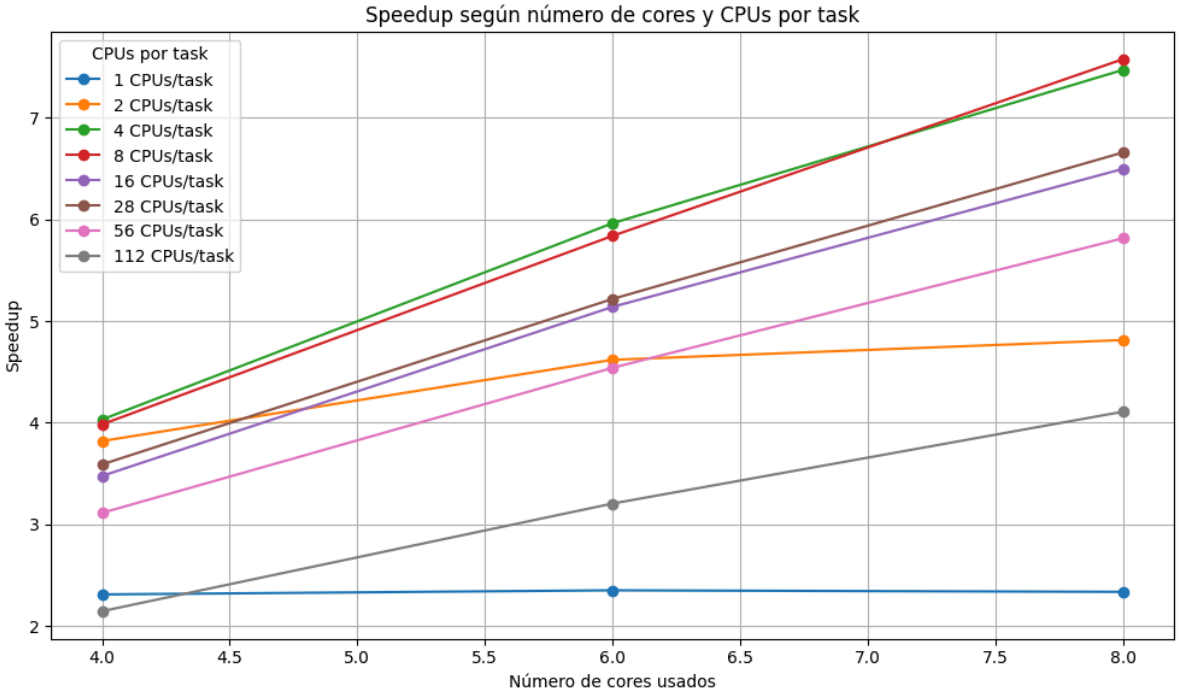

Figura 2: Speedup según número de cores y CPUs por task para la aplicación bt-mz.

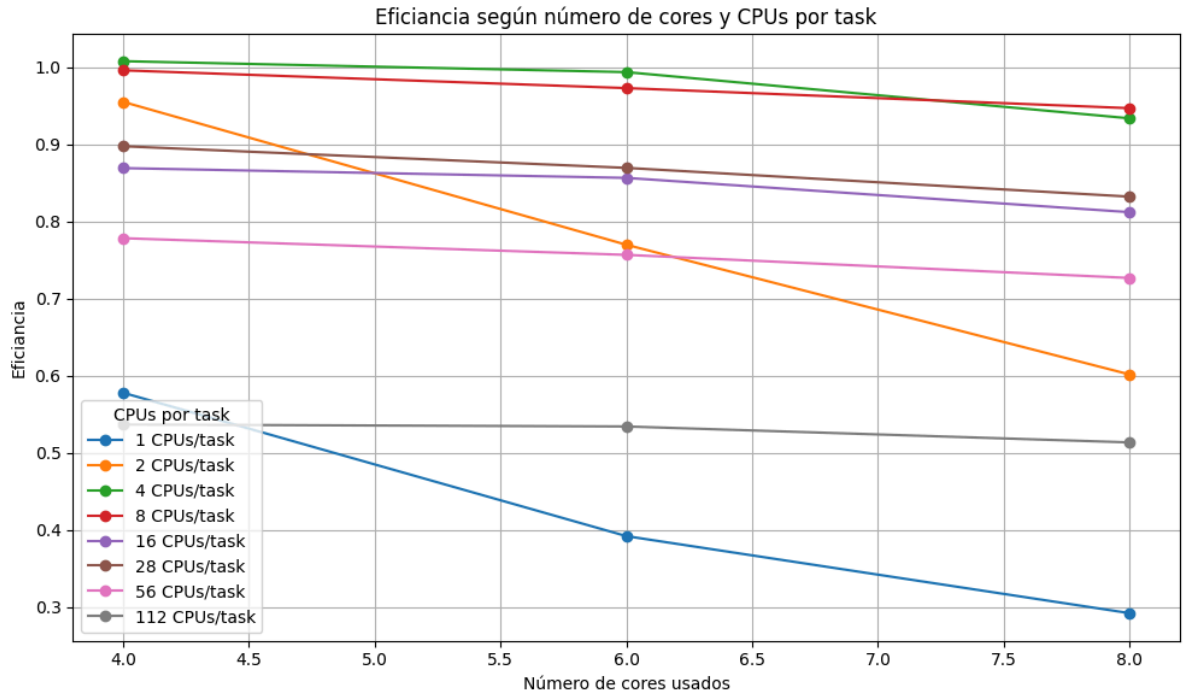

Figura 3: Eficiencia según número de cores y CPUs por task para la aplicación bt-mz.

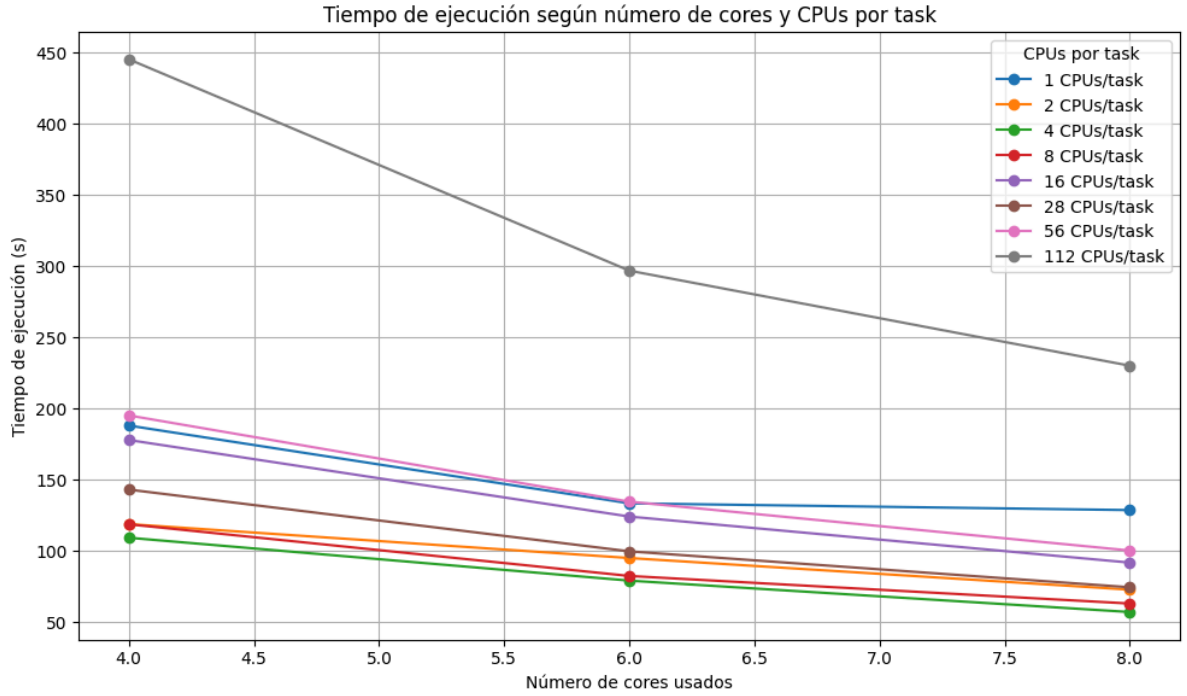

Figura 4: Tiempo de ejecución según número de cores y CPUs por task para la aplicación sp-mz.

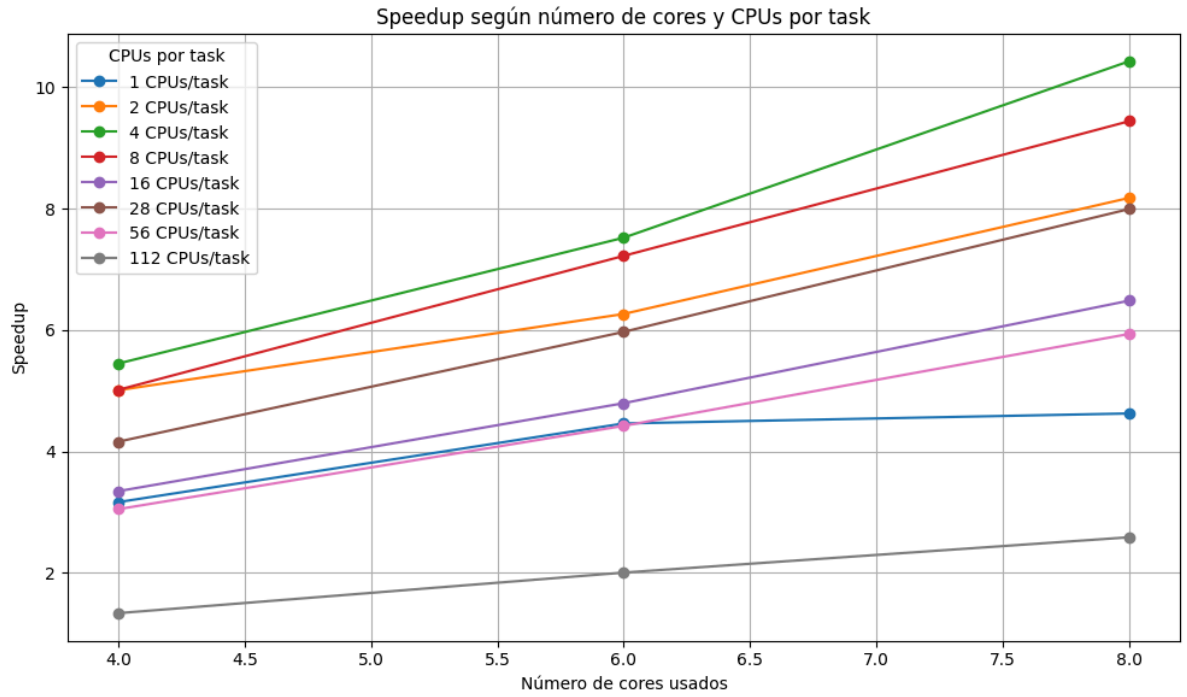

Figura 5: Speedup según número de cores y CPUs por task para la aplicación sp-mz.

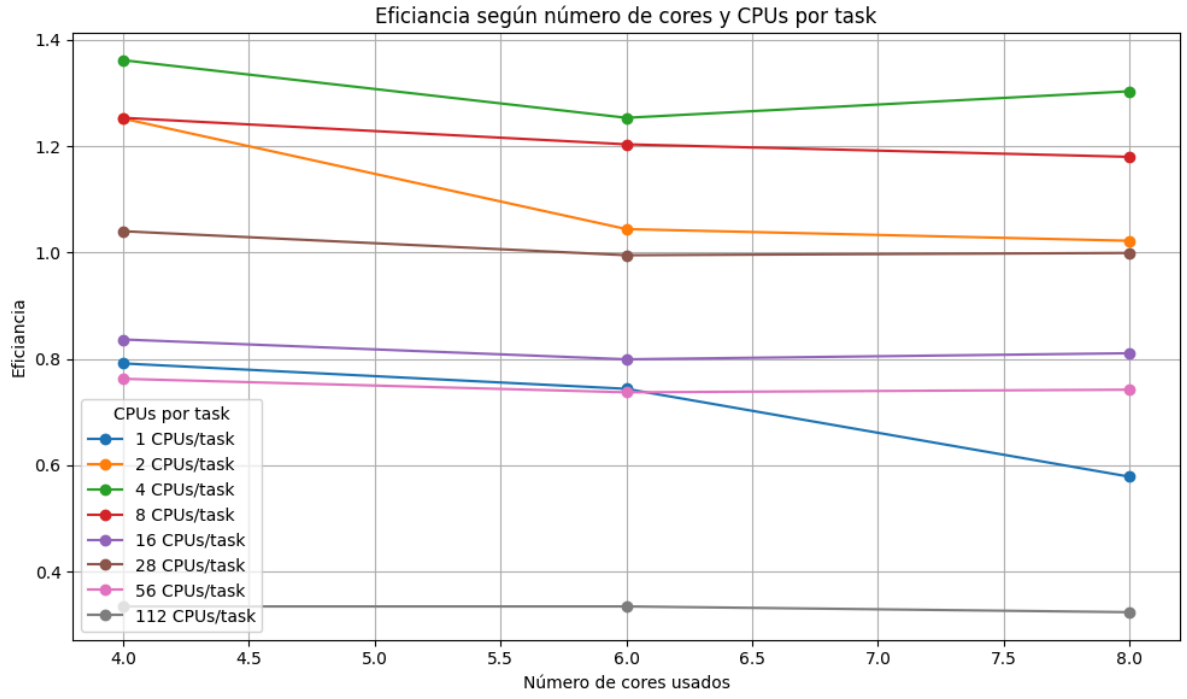

Figura 6: Eficiencia según número de cores y CPUs por task para la aplicación sp-mz.

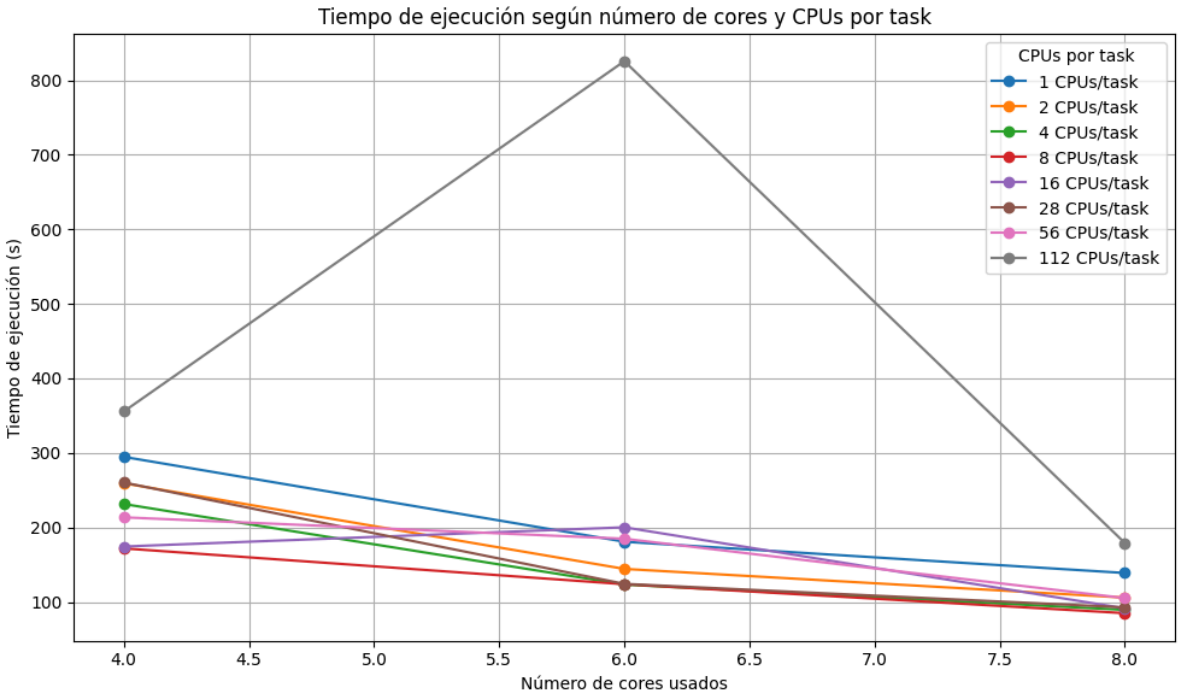

Figura 7: Tiempo de ejecución según número de cores y CPUs por task para la aplicación lu-mz.

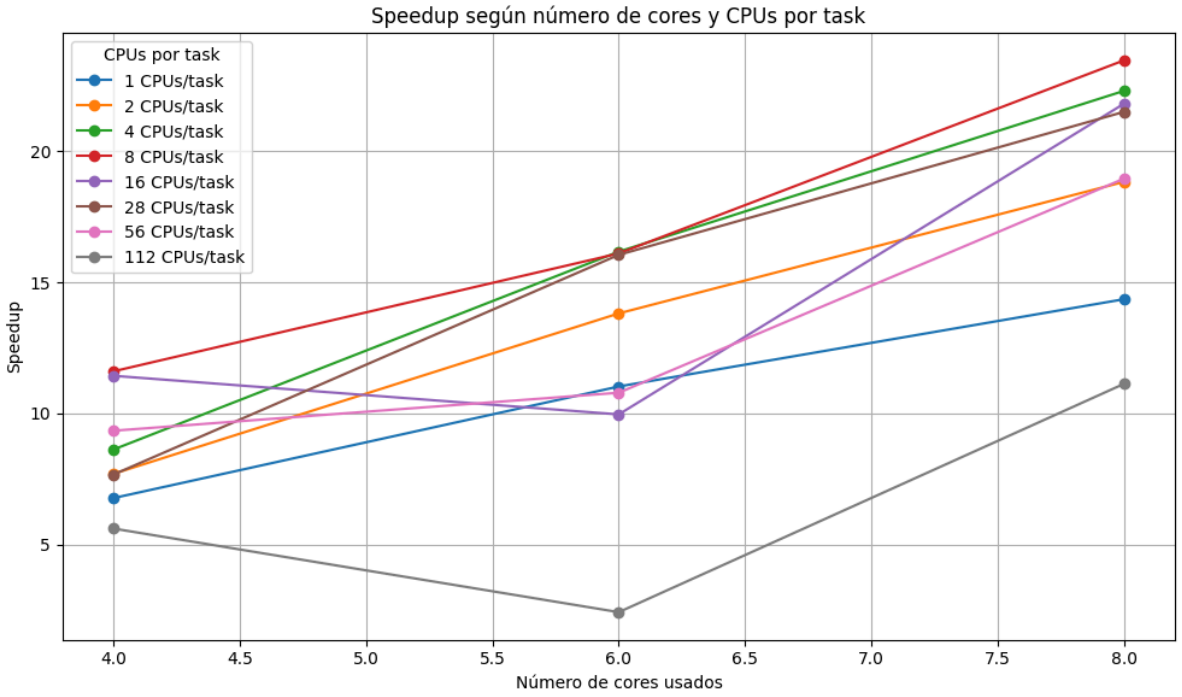

Figura 8: Speedup según número de cores y CPUs por task para la aplicación lu-mz.

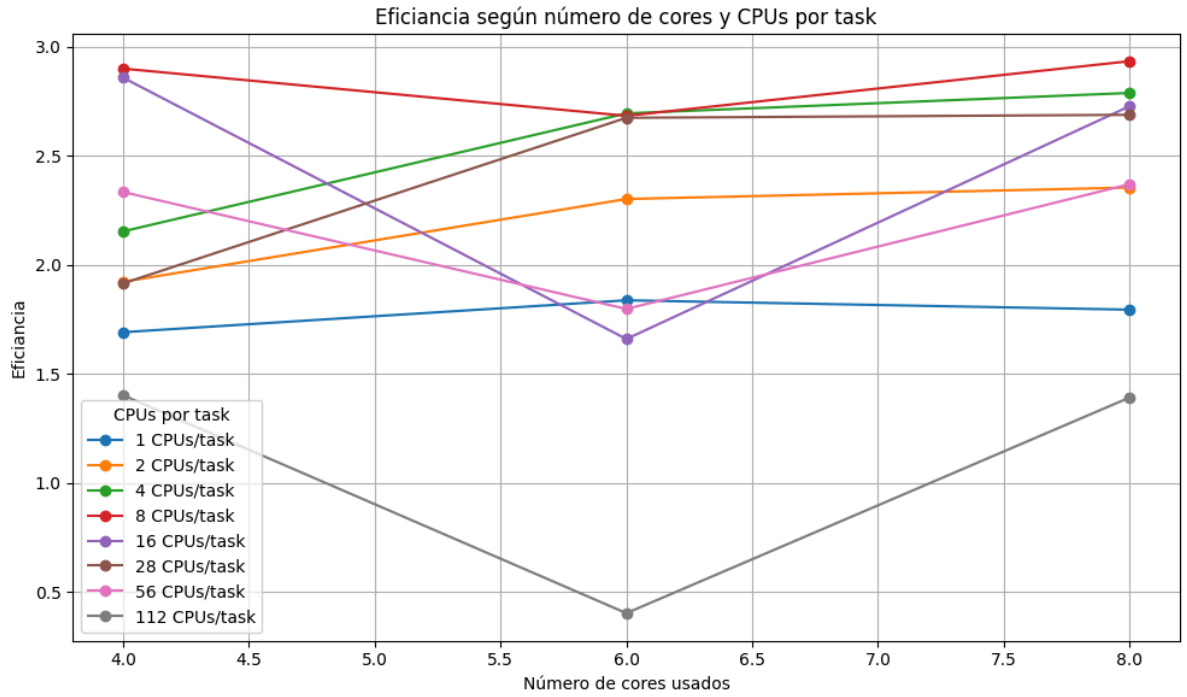

Figura 9: Eficiencia según número de cores y CPUs por task para la aplicación lu-mz.

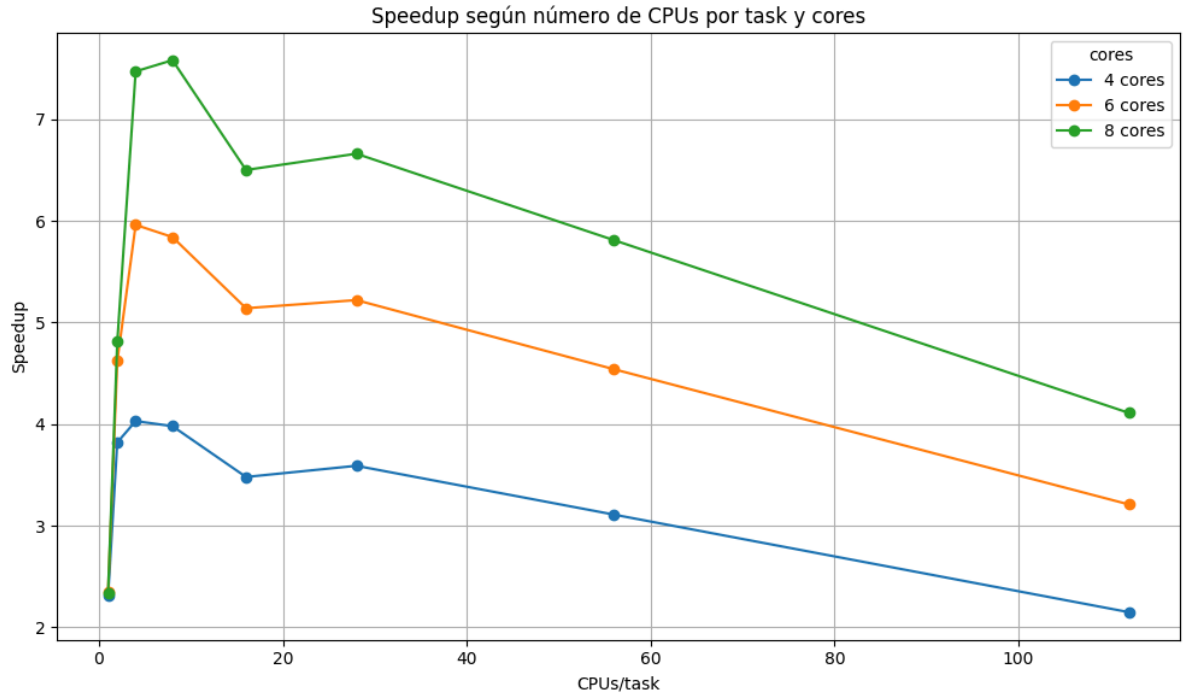

Figura 10: Speedup según número de CPUs por task y cores para la aplicación bt-mz.

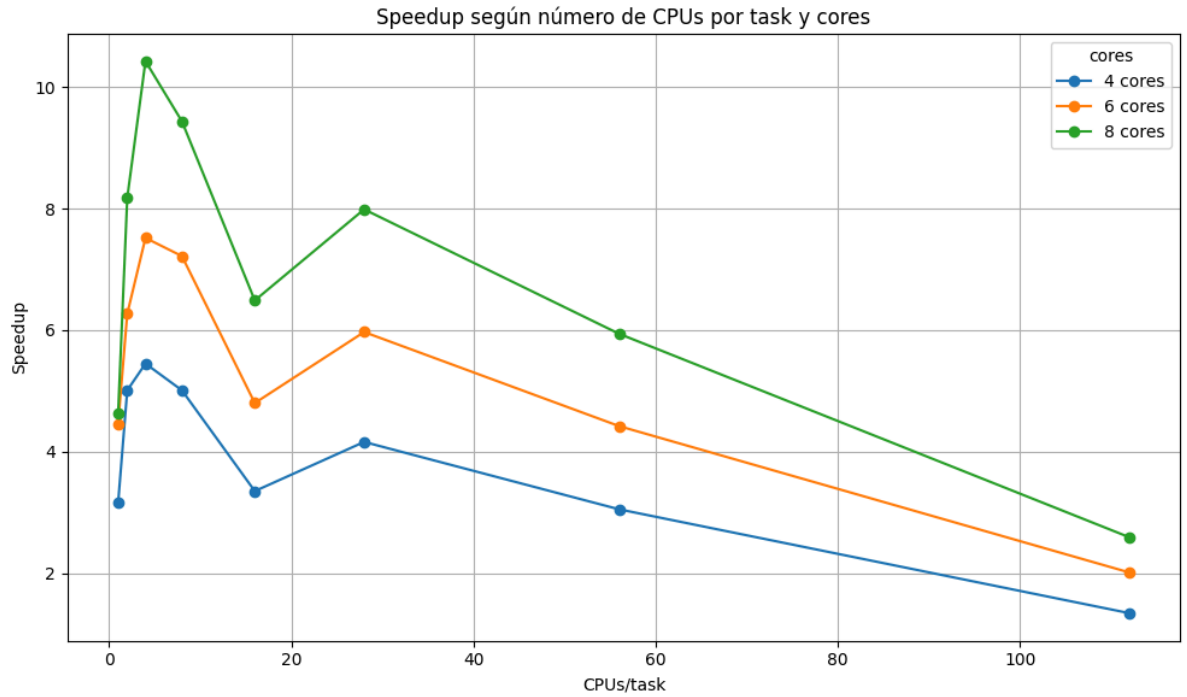

Figura 11: Speedup según número de CPUs por task y cores para la aplicación sp-mz.

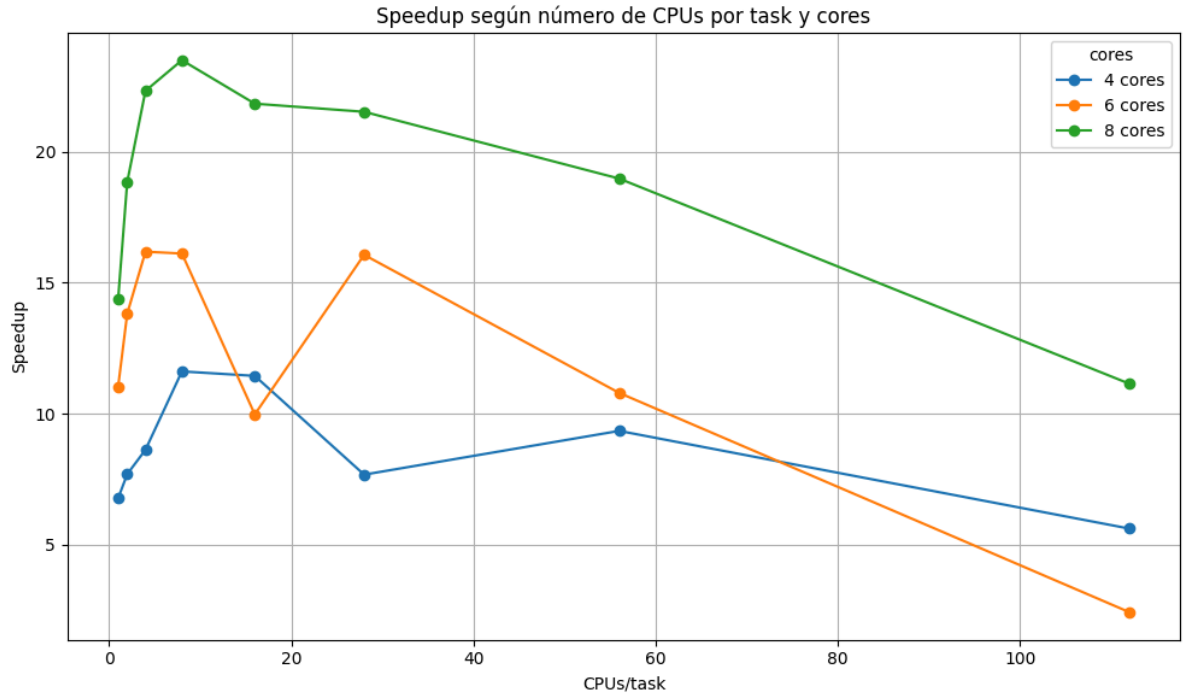

Figura 12: Speedup según número de CPUs por task y cores para la aplicación lu-mz.

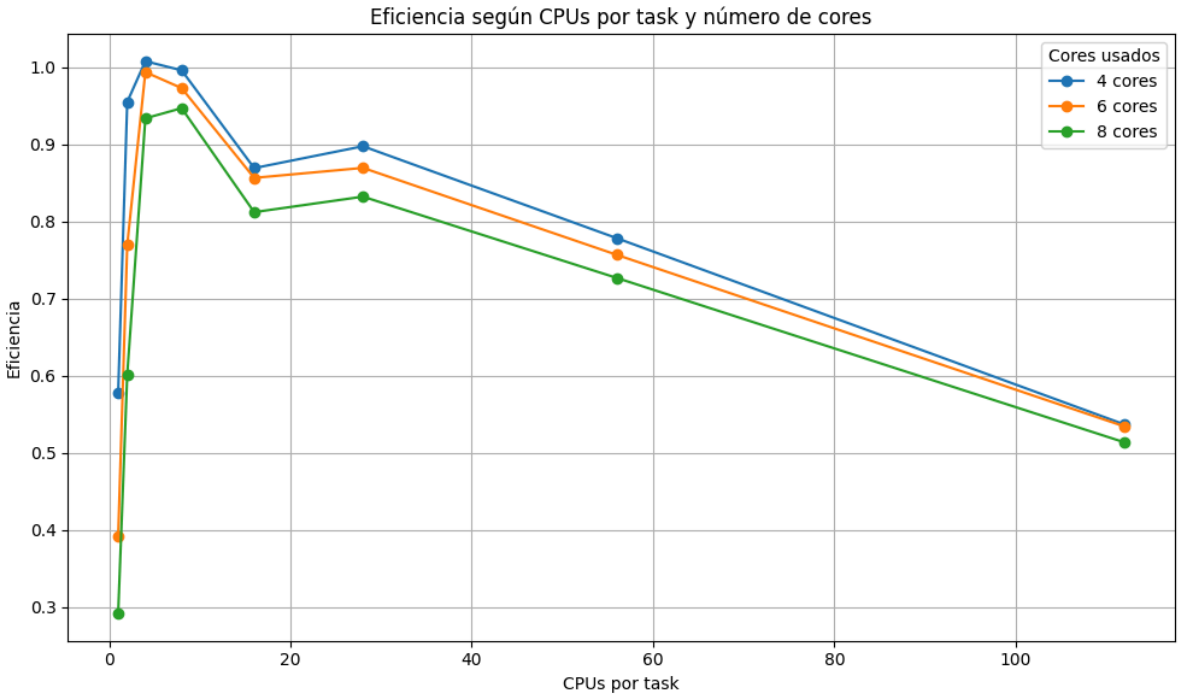

Figura 13: Eficiencia según las CPUs por task y el número de cores para la aplicación bt-mz.

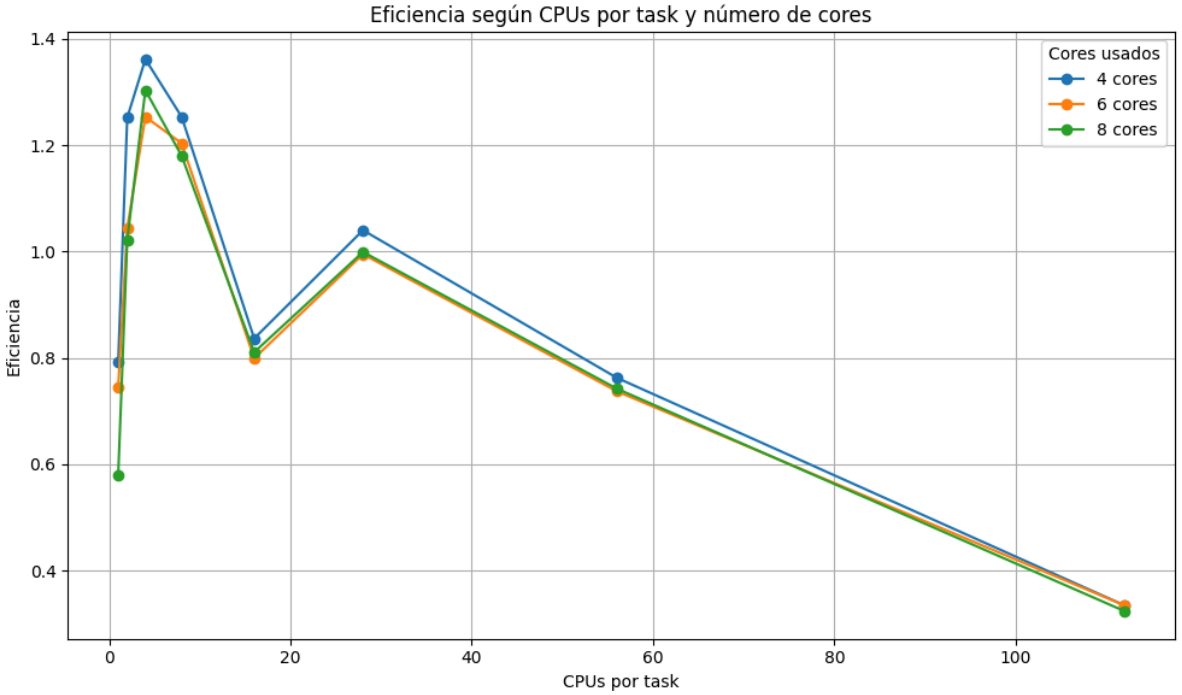

Figura 14: Eficiencia según las CPUs por task y el número de cores para la aplicación sp-mz.

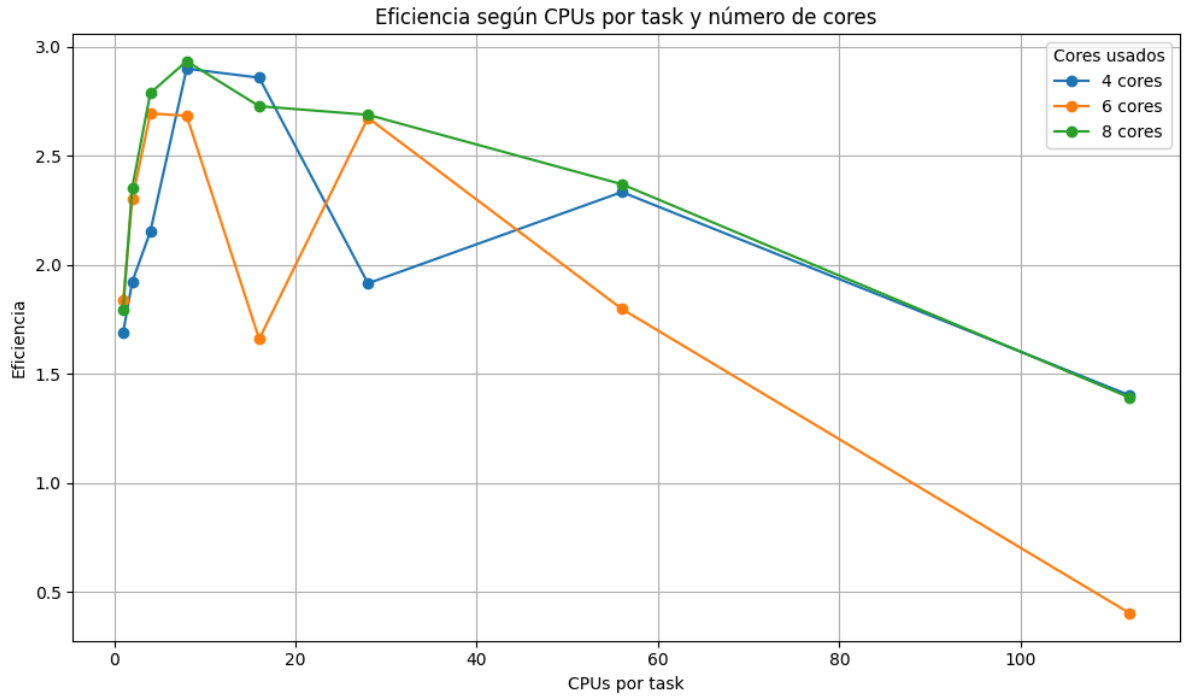

Figura 15: Eficiencia según las CPUs por task y el número de cores para la aplicación lu-mz.

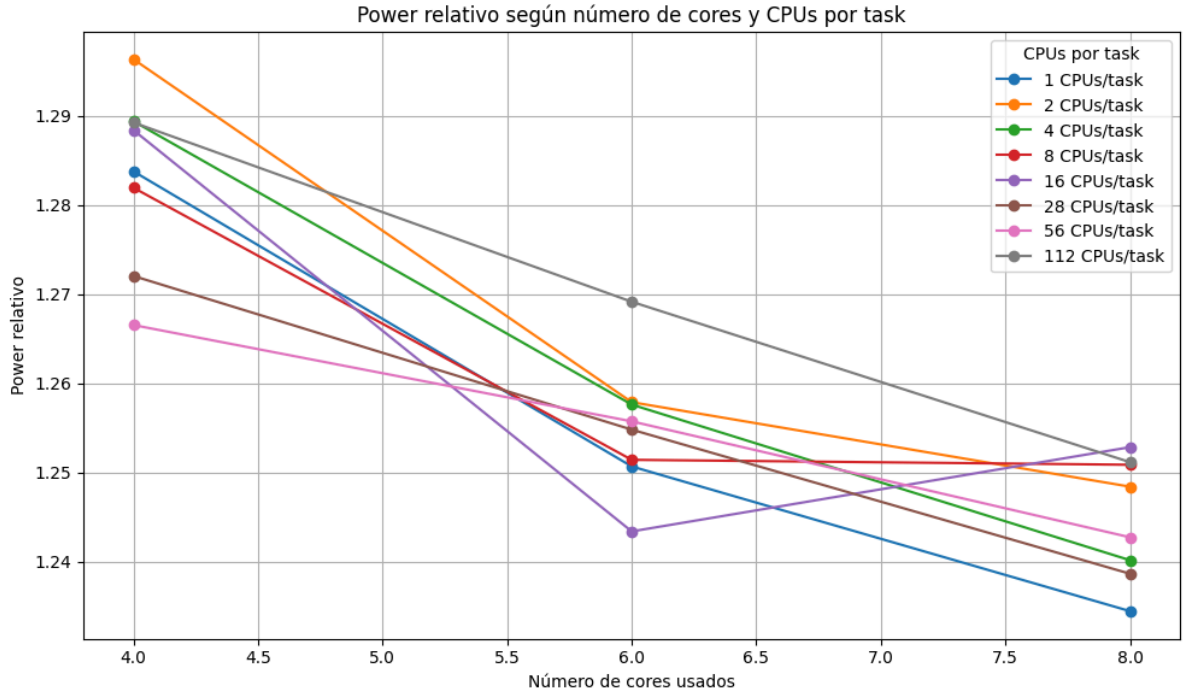

Figura 16: Power relativo según número de cores y CPUs por task para la aplicación bt-mz.

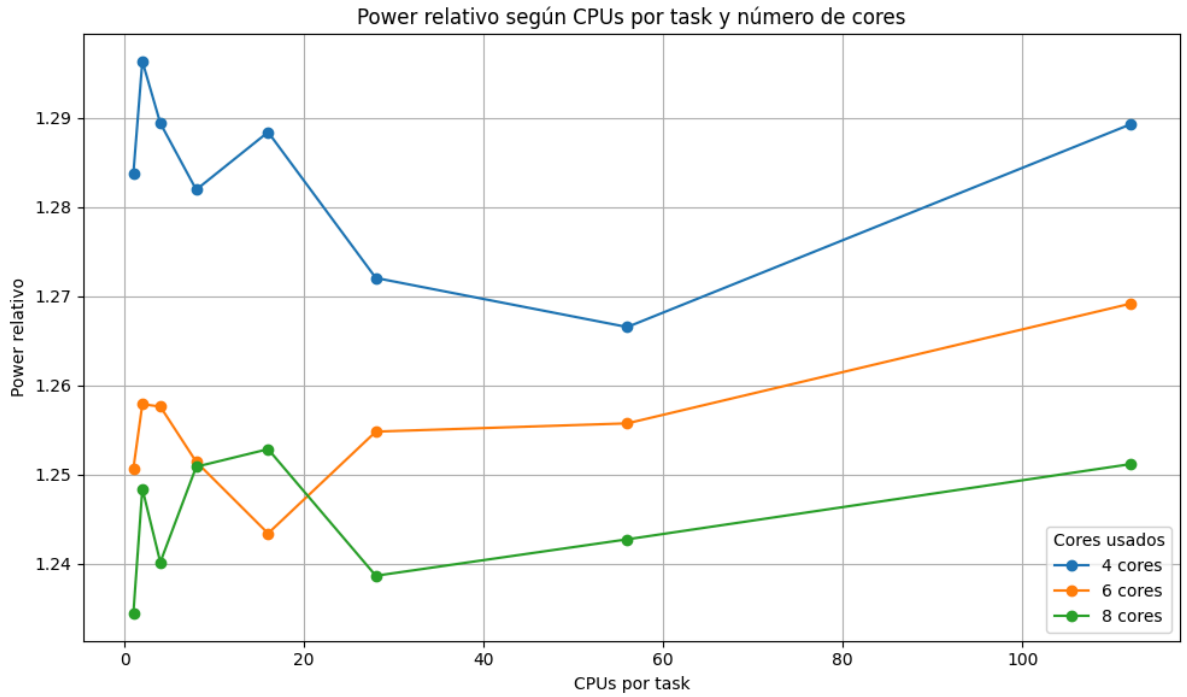

Figura 17: Power relativo según CPUs por task y número de cores para la aplicación bt-mz.

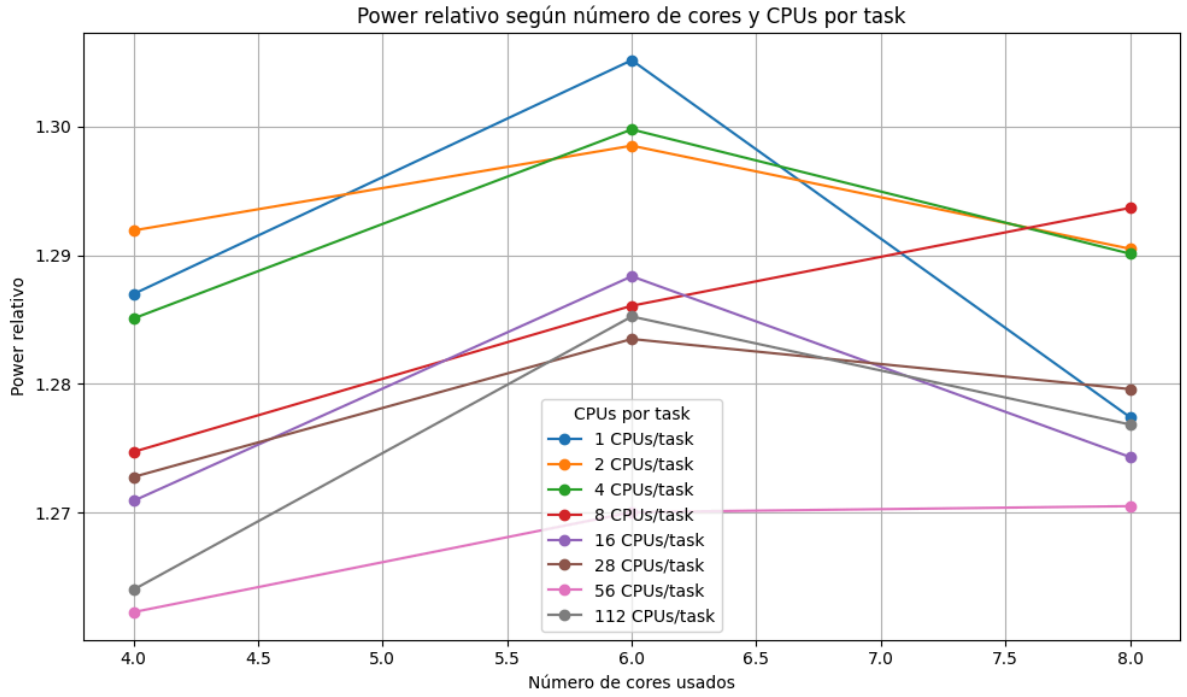

Figura 18: Power relativo según número de cores y CPUs por task para la aplicación sp-mz.

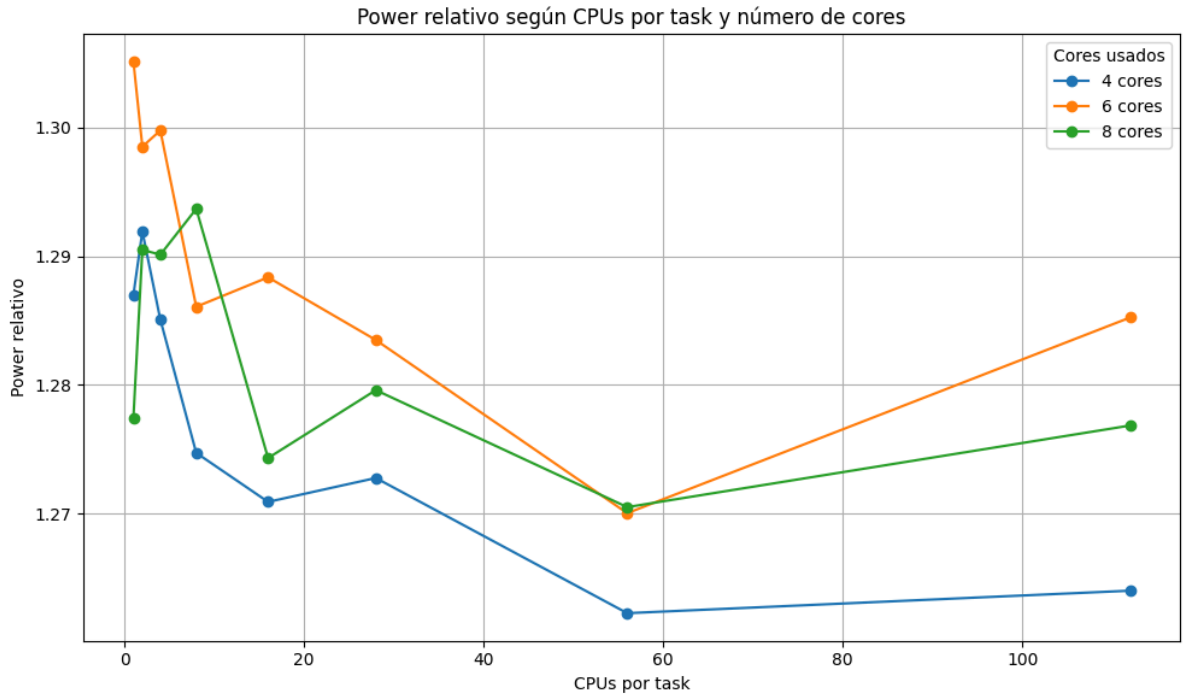

Figura 19: Power relativo según CPUs por task y número de cores para la aplicación sp-mz.

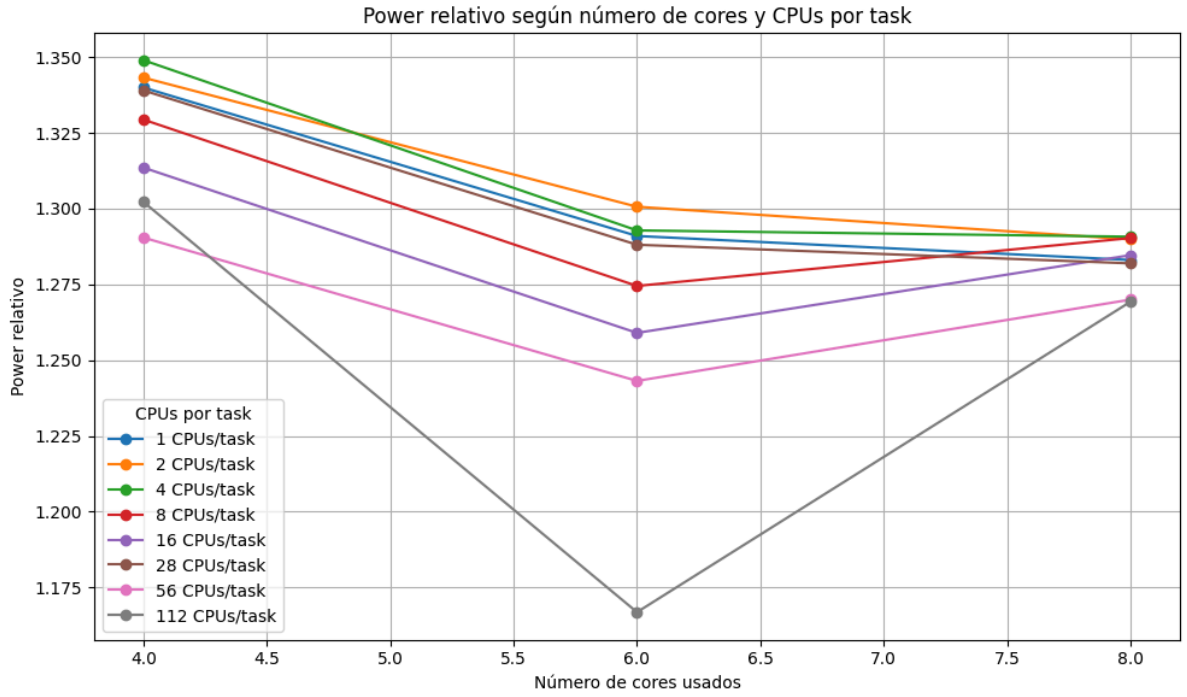

Figura 20: Power relativo según número de cores y CPUs por task para la aplicación lu-mz.

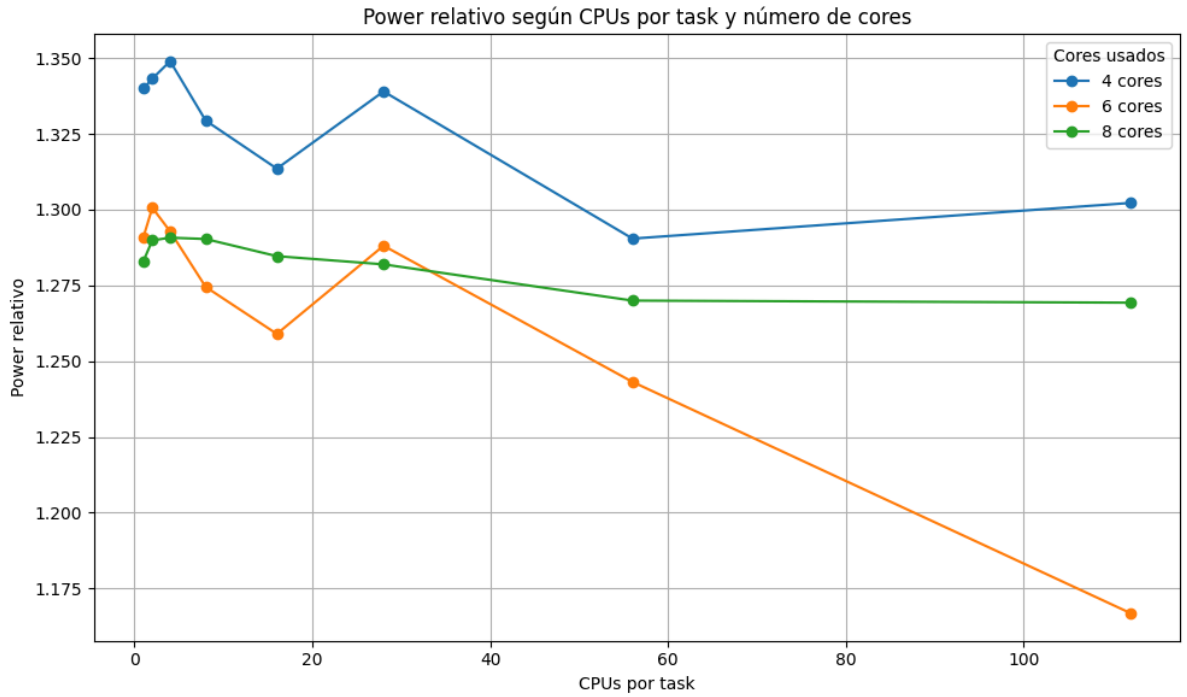

Figura 21: Power relativo según CPUs por task y número de cores para la aplicación lu-mz.

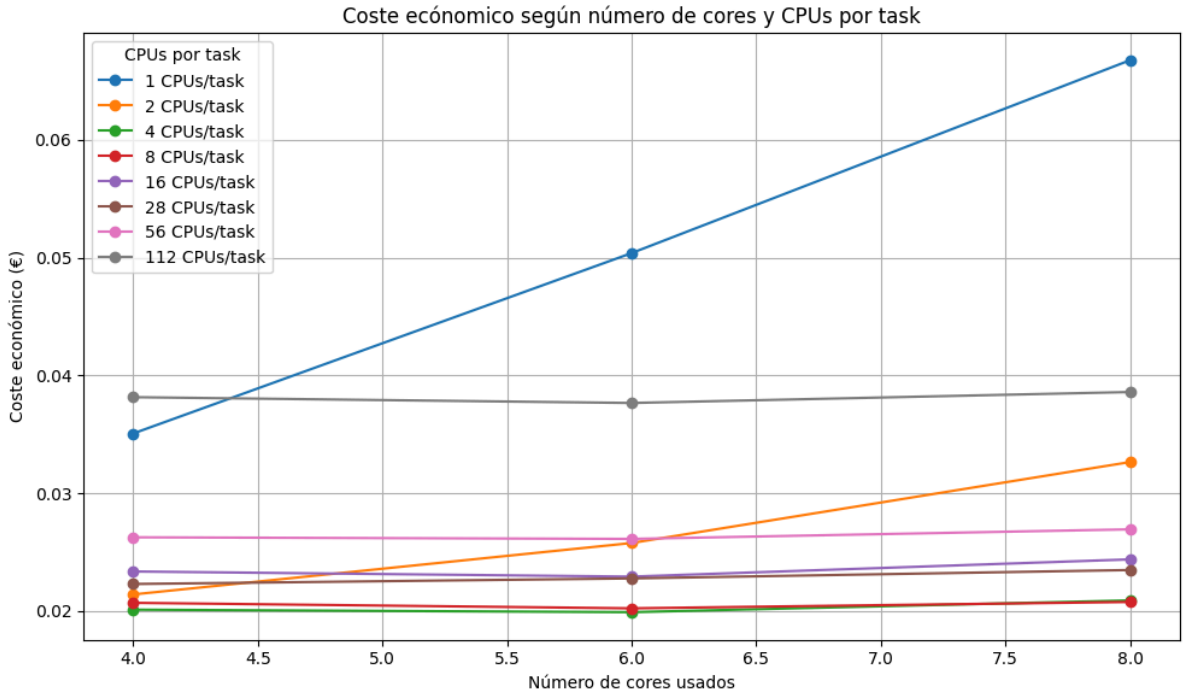

Figura 22: Coste económico según número de cores y CPUs por task para la aplicación bt-mz.

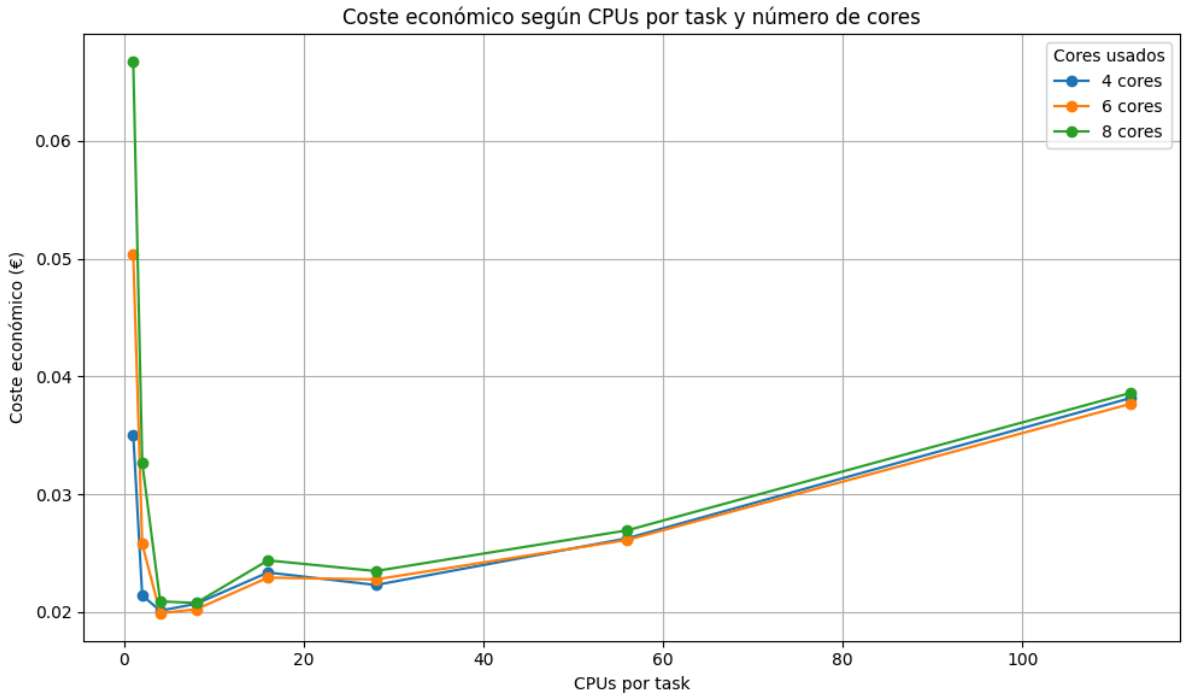

Figura 23: Coste económico según CPUs por task y número de cores para la aplicación bt-mz.

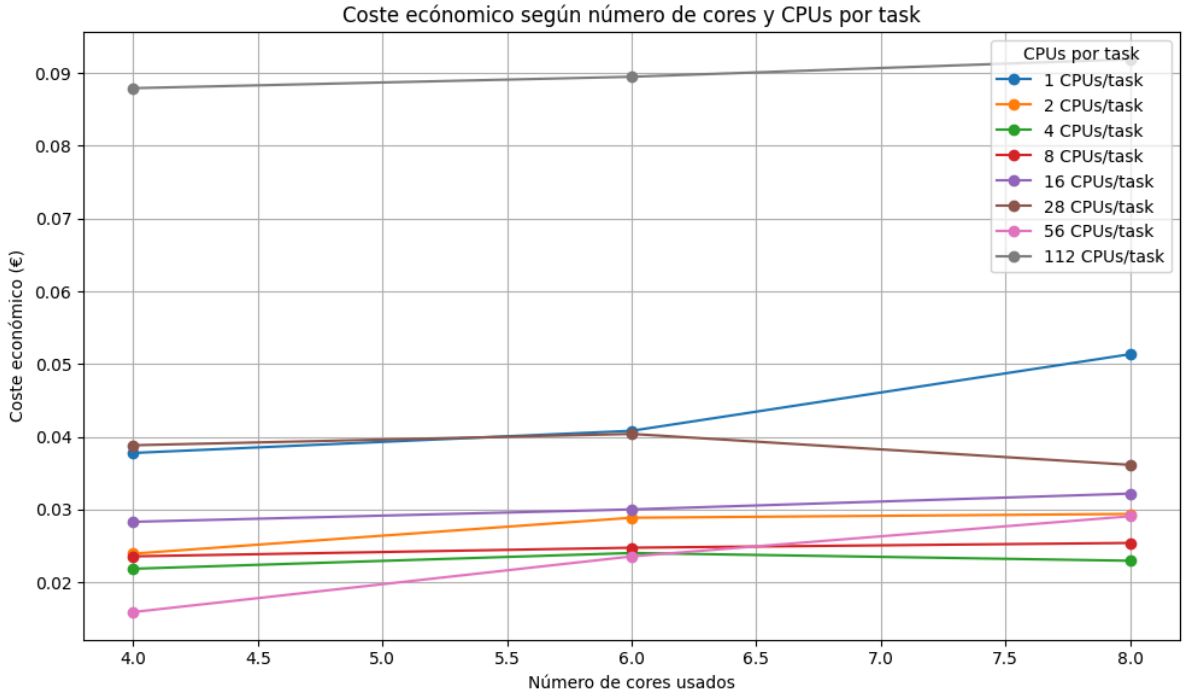

Figura 24: Coste económico según número de cores y CPUs por task para la aplicación sp-mz.

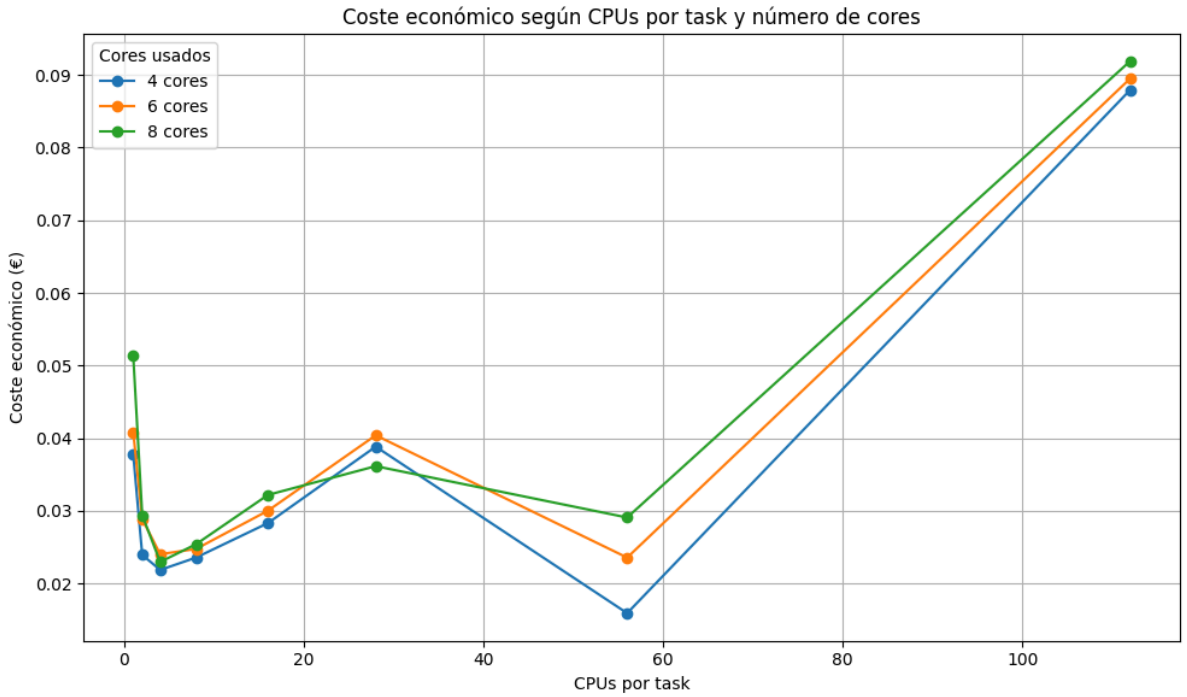

Figura 25: Coste económico según CPUs por task y número de cores para la aplicación sp-mz.

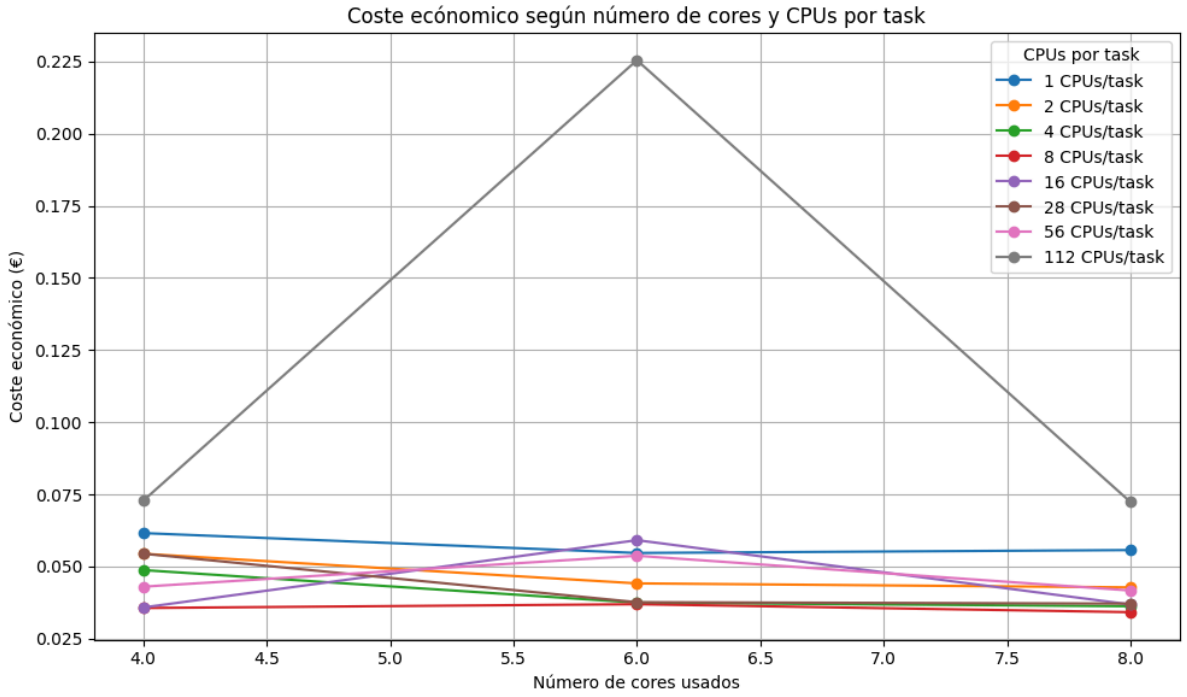

Figura 26: Coste económico según número de cores y CPUs por task para la aplicación lu-mz.

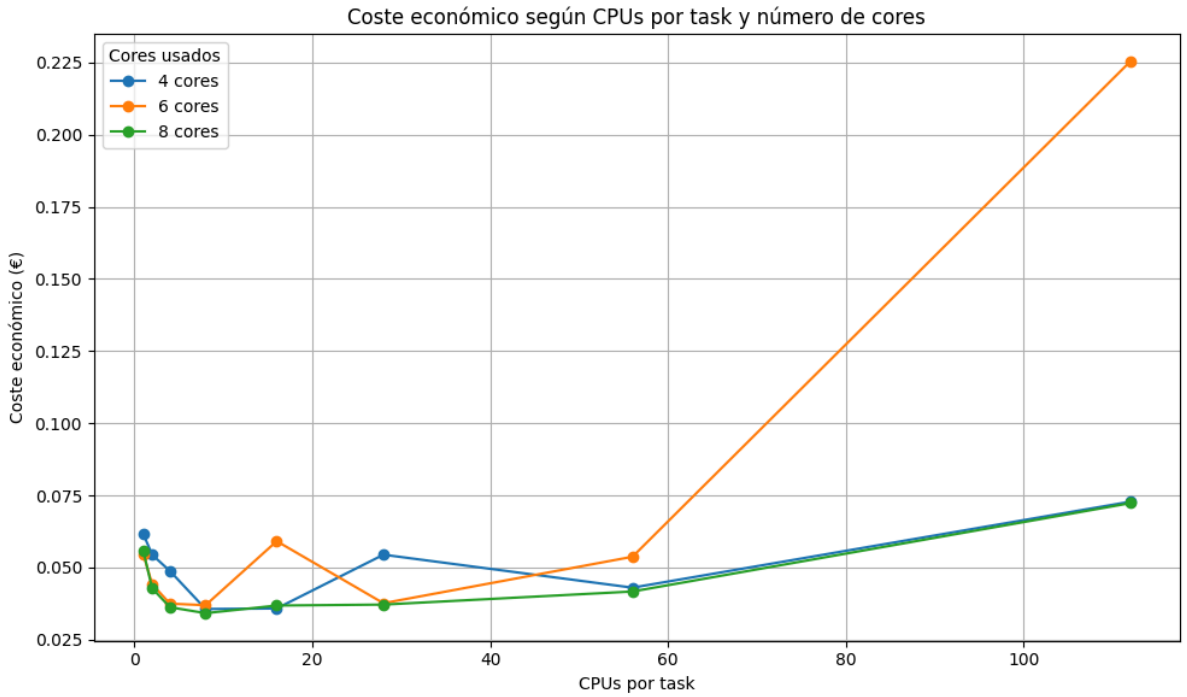

Figura 27: Coste económico según CPUs por task y número de cores para la aplicación lu-mz.

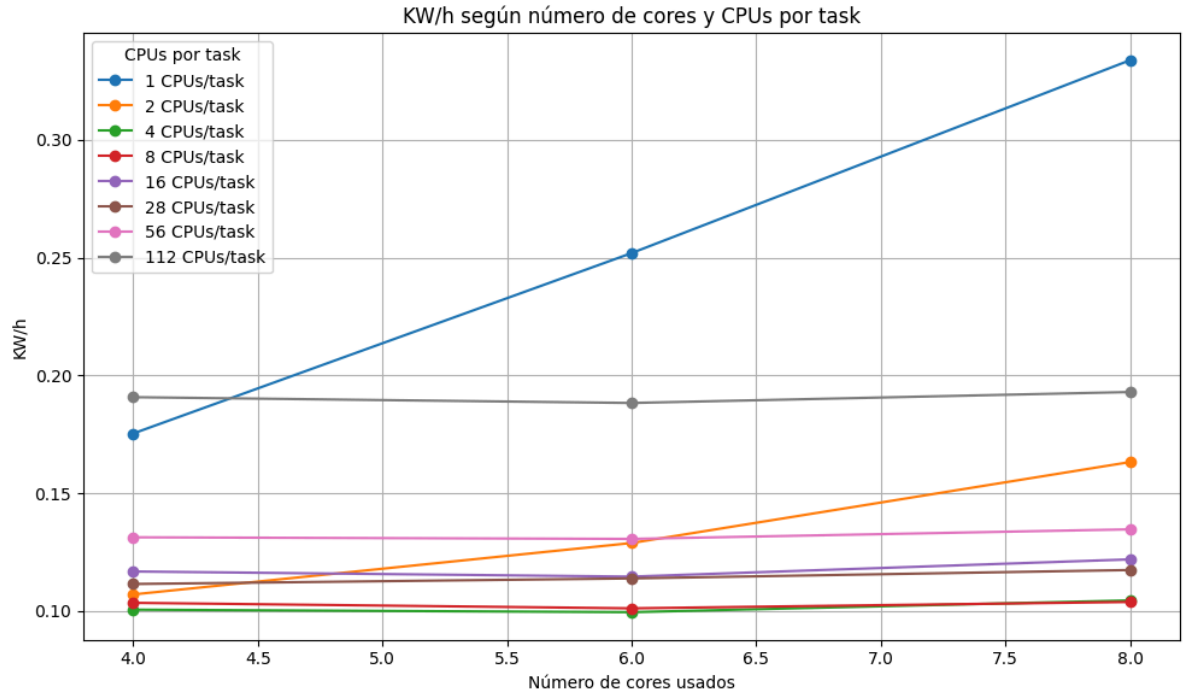

Figura 28: KW/h según número de cores y CPUs por task para la aplicación bt-mz.

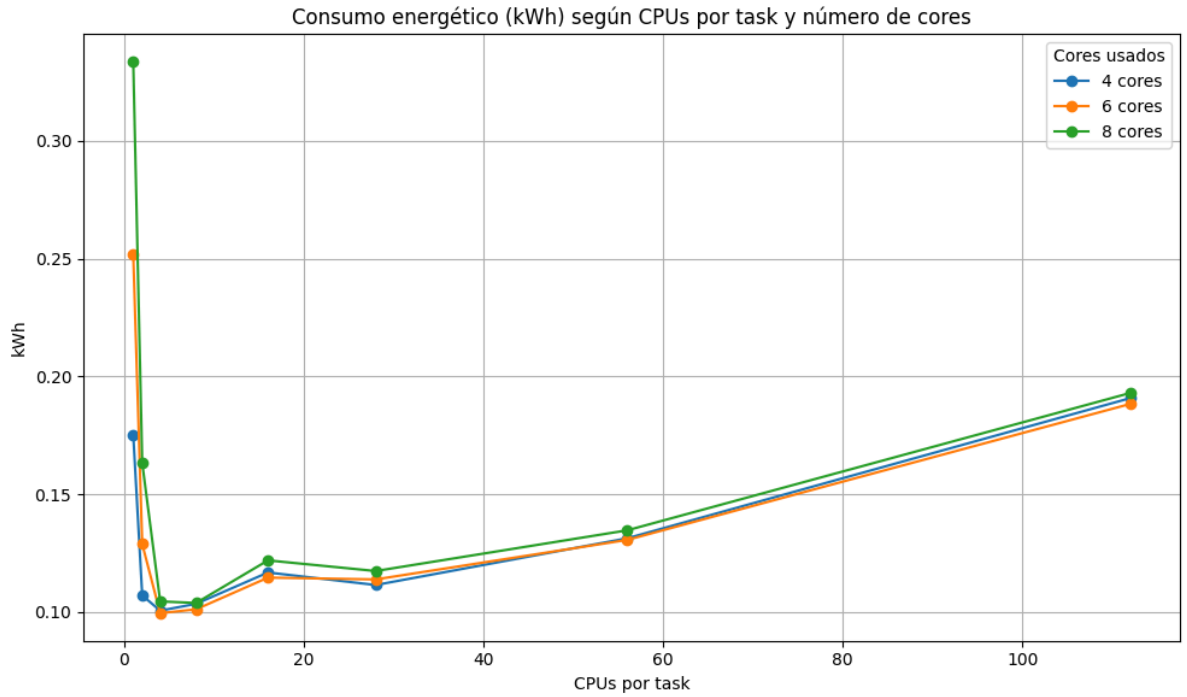

Figura 29: KW/h según CPUs por task y número de cores para la aplicación bt-mz.

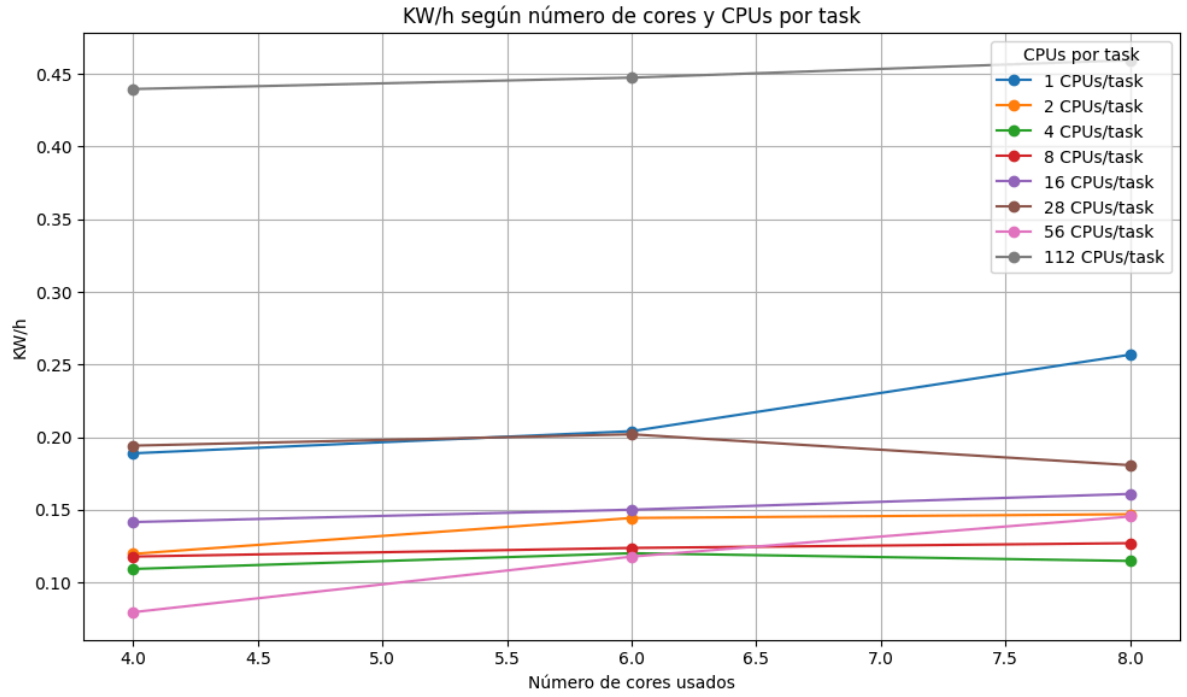

Figura 30: KW/h según número de cores y CPUs por task para la aplicación sp-mz.

Figura 31: KW/h según CPUs por task y número de cores para la aplicación sp-mz.

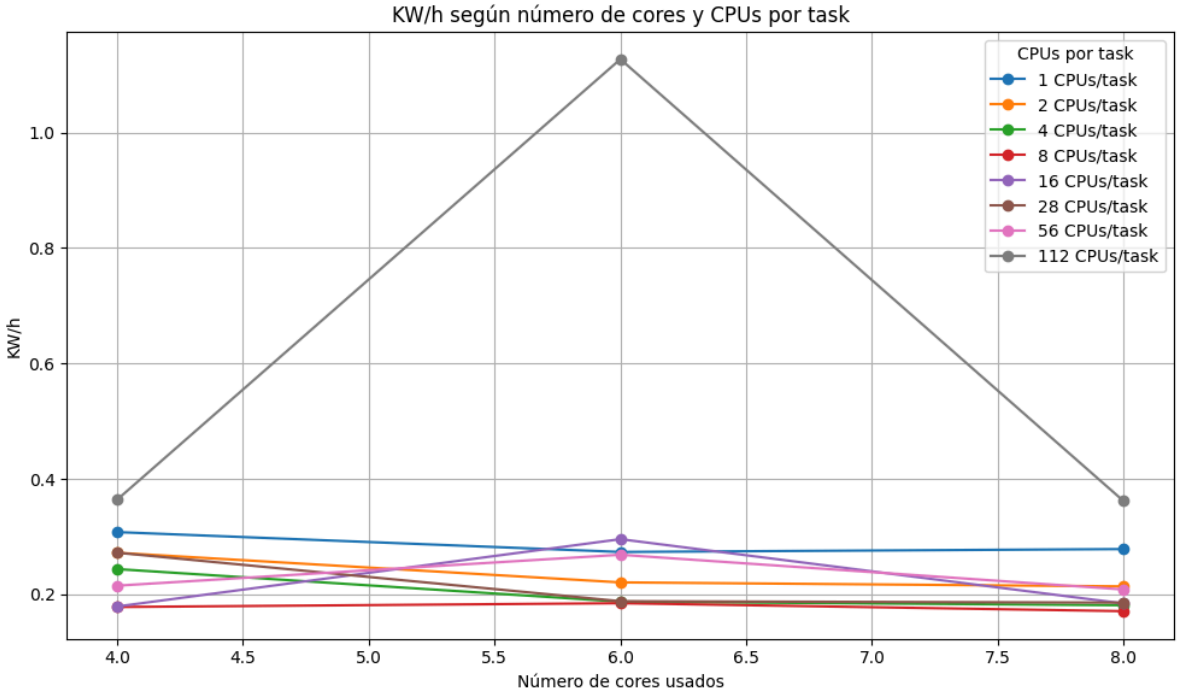

Figura 32: KW/h según número de cores y CPUs por task para la aplicación lu-mz.

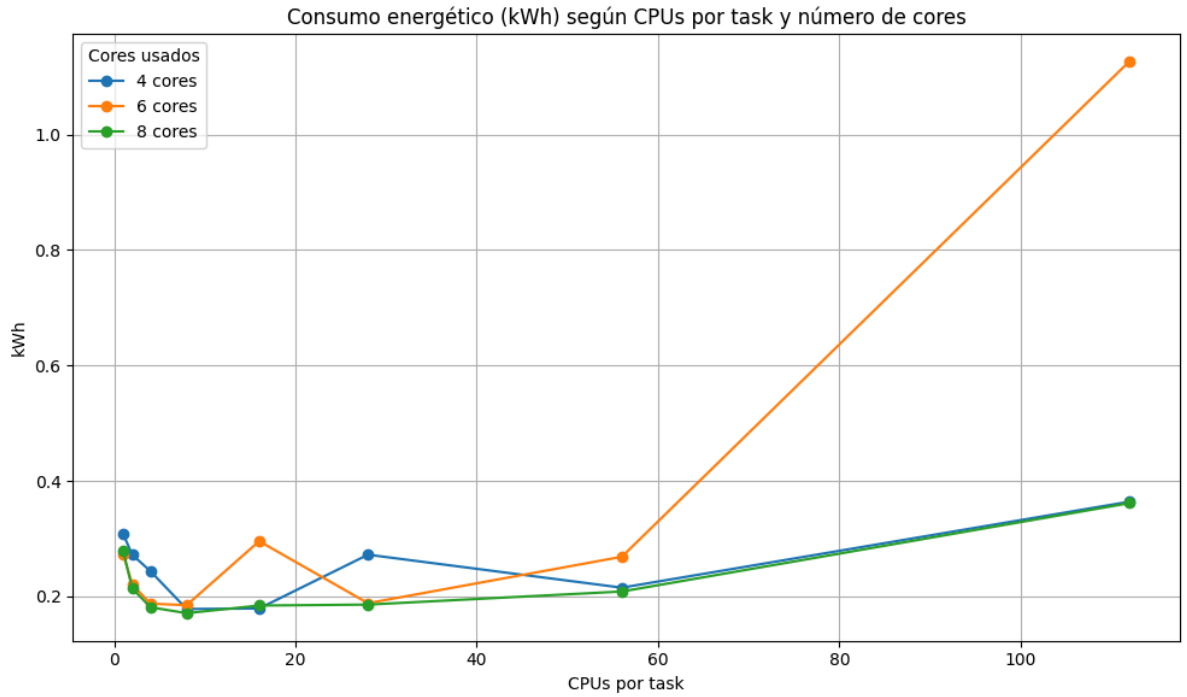

Figura 33: KW/h según CPUs por task y número de cores para la aplicación lu-mz.

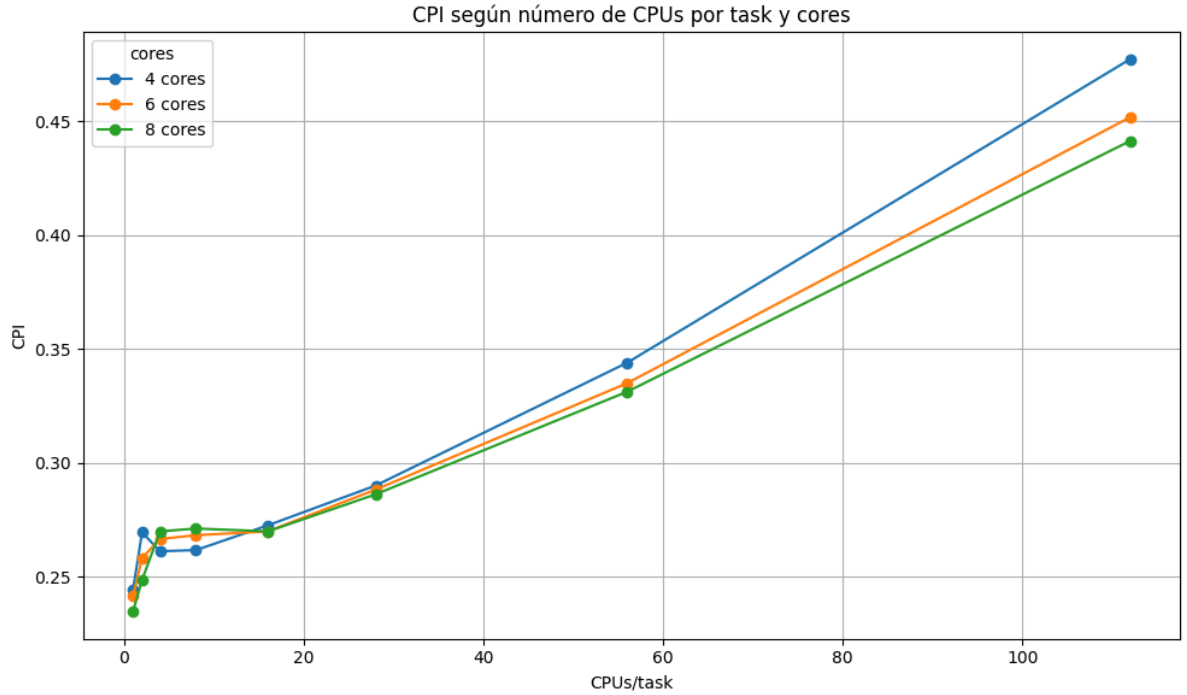

Figura 34: CPI según número de CPUs por task y cores para la aplicación bt-mz.

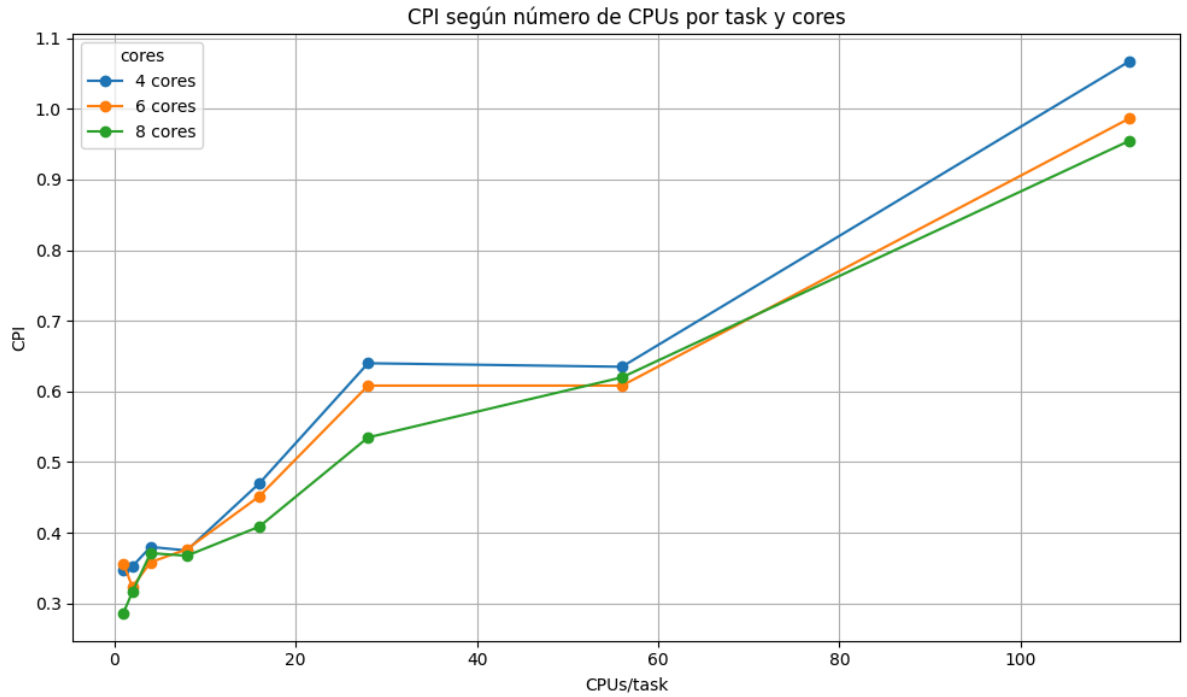

Figura 35: CPI según CPUs por task y número de cores para la aplicación sp-mz.

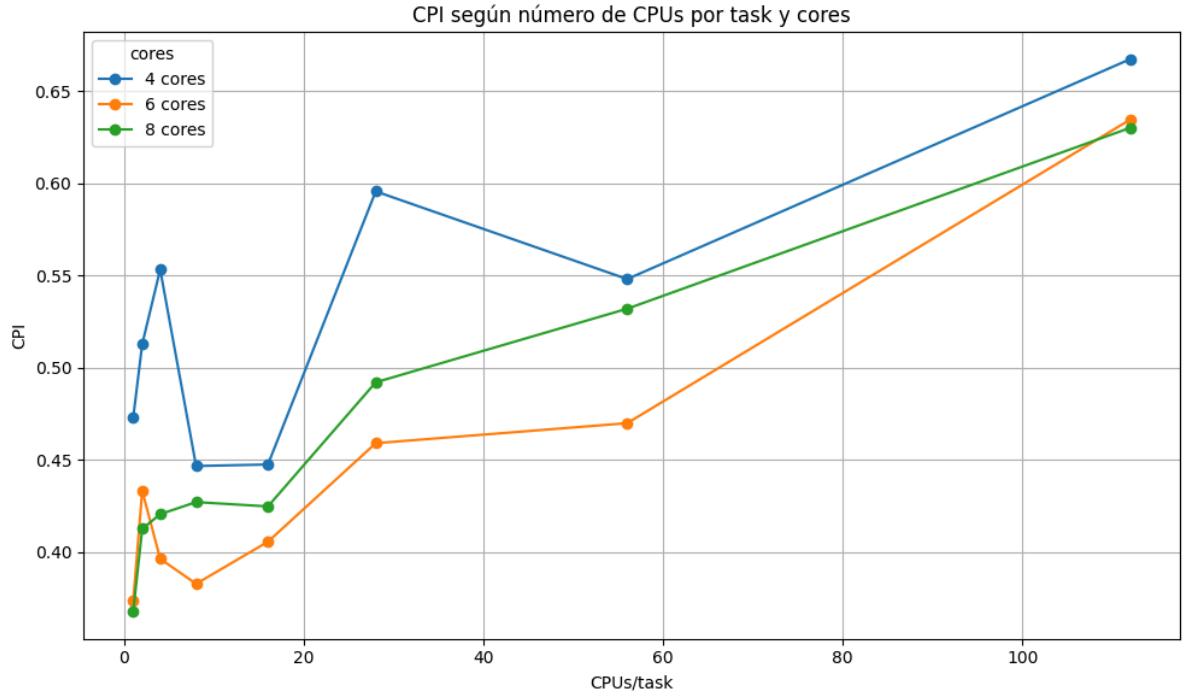

Figura 36: CPI según número de CPUs por task y cores para la aplicación lu-mz.

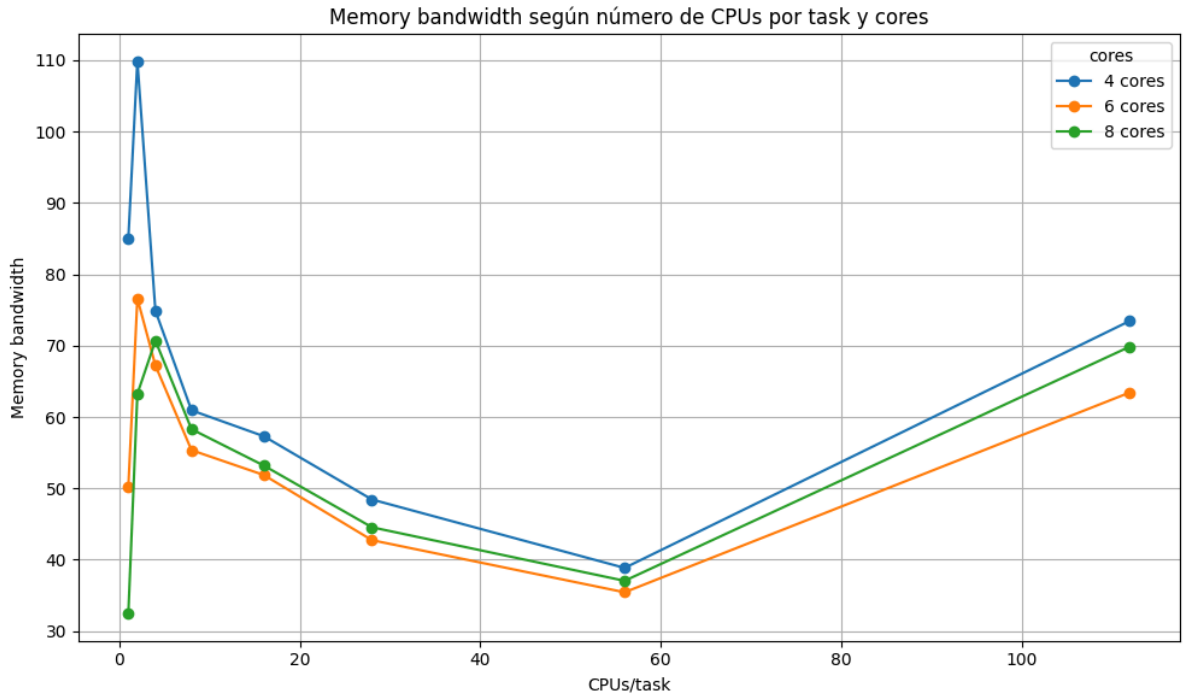

Figura 37: Memory bandwidth según CPUs por task y número de cores para la aplicación bt-mz.

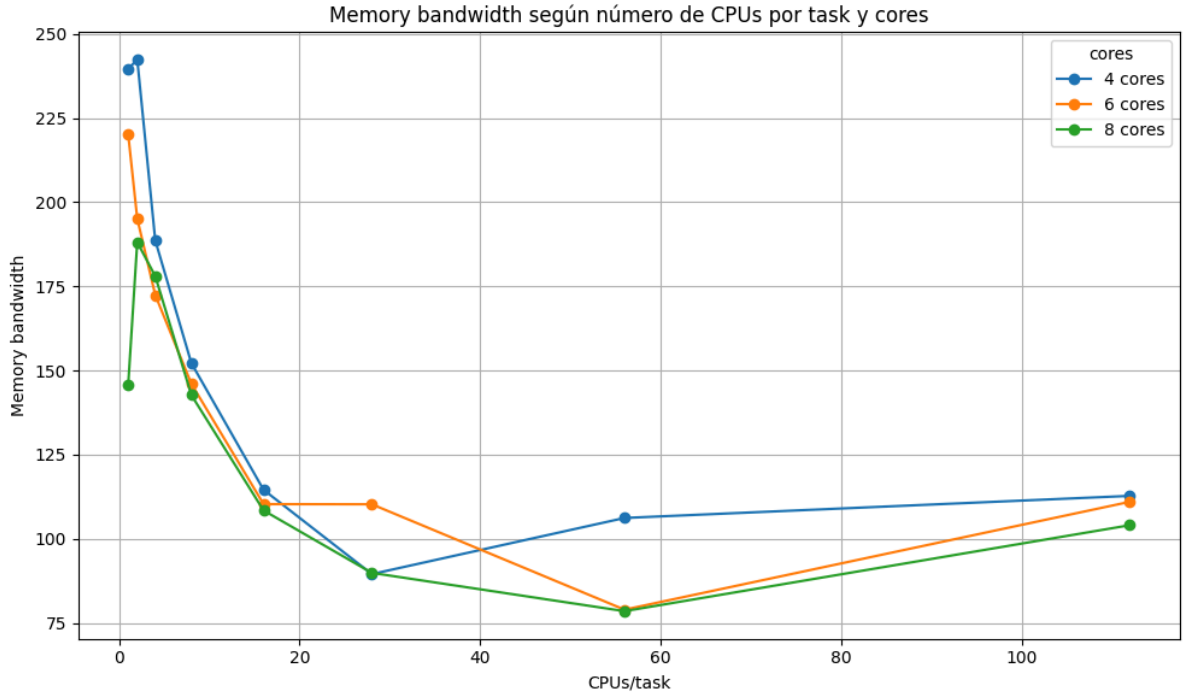

Figura 38: Memory bandwidth según número de CPUs por task y cores para la aplicación sp-mz.

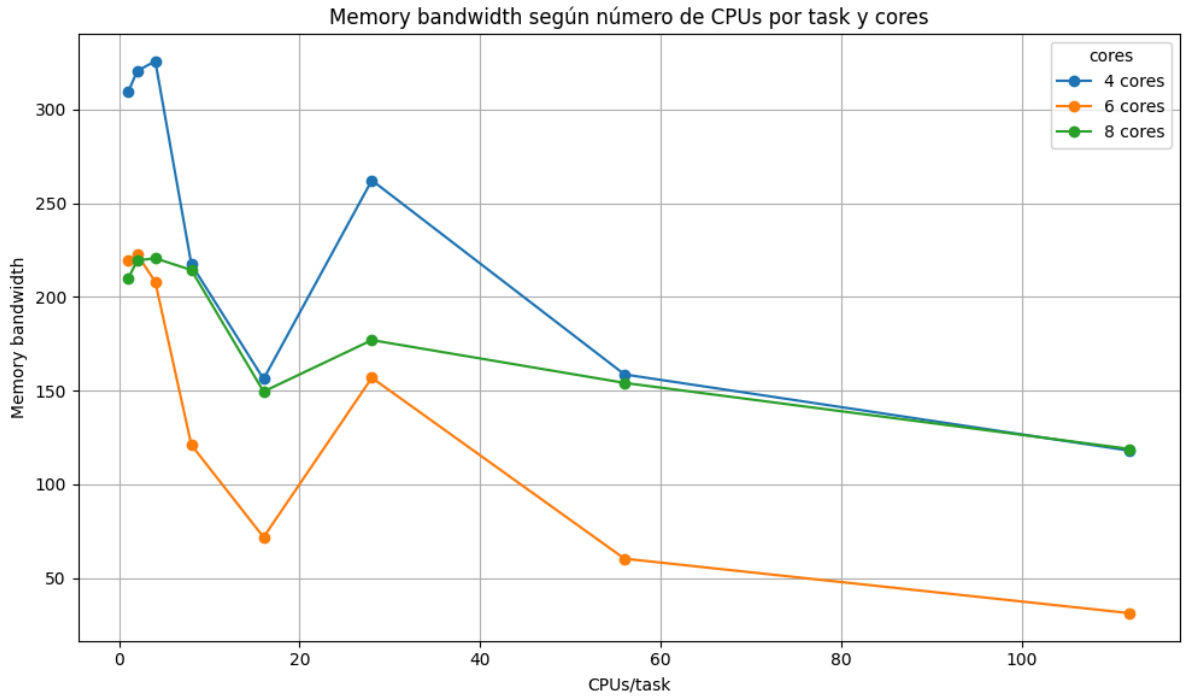

Figura 39: Memory bandwidth según número de CPUs por task y cores para la aplicación lu-mz.

APPLICATION	POLICY	# NODES	AVG/DEF/IMC (GHz)	TIME	# POWER	GBS	CPI	# ENERGY(J)	GFLOPS/W	IO(MBS)	MPI%	G-POW(T/U)	G-FREQ	G-UTIL(G/MEM)
tensorflow	NP	1	3.56/2.00/---	1741	1043,14	-	-	1816102						
DenseNet121	MO	1	3.68/2.00/2.36	332.10	1.022,68	0.31	0.58	395635	0	0,2	0	344.12/344.12	1.971	95%/86%
DenseNet121_mixe	NP	1	3.66/2.00/2.36	344.21	964,99	0.37	0.60	332158	0	0,2	0	282.00/282.00	1.950	91%/77%
DenseNet121_disa	MO	1	3.68/2.00/2.36	359.80	1.031,26	0.32	0.58	342598	0	0,2	0	342.91 /342.91	1.971	94%/85%
VGG19_disable	MO	1	3.68/2.00/2.36	279.44	1.062,08	0.22	0.51	296788	0	0,2	0	374.66 /374.66	1.970	98%/74%
VGG19_mixed	MO	1	3.50/2.00/2.33	93.57	1.053,42	0.33	0.66	98571	0	0,7	0	414.11 /414.11	1.980	91%/54%
VGG19	MO	1	3.68/2.00/2.36	188.05	1.033,60	0.25	0.60	194366	0	0,3	0	369.43 /369.43	1.980	96%/78%
ResNet50_disable	MO	1	3.68/2.00/2.36	287.44	1.082,81	3.34	0.58	311244	0	0,2	0	400.02 /400.02	1.970	96%/76%
ResNet50_mixed	MO	1	3.66/2.00/2.34	251.88	969,71	0.31	0.59	244248	0	0,3	0	284.13 /284.13	1.963	93%/75%
ResNet50	MO	1	3.68/2.00/2.36	245.70	1.013,78	0.29	0.56	249083	0	0,3	0	351.43 /351.43	1.951	93%/87%

Tabla 1: Métricas de cada experimento de tensorflow obtenidas mediante la comanda eacc