PACO Laboratory Assignment

Lab 3: Iterative task decomposition with OpenMP: the computation of the Mandelbrot set

---

Oriol Amate Sabat, Adrián García Campillo, 5011, 18 october, paco1106

1.1 Task decomposition analysis with Tareador..... 3
1.1.1 First analysis with Tareador 3
1.1.2 Second analysis with Tareador.....4
1.2 Point decomposition strategy.....6
1.2.1 Explicit tasks without taskloops.....6
1.2.2 Taskloops without using neither num tasks nor grainsize..... 9
1.2.3 Taskloops without nogroup.....13

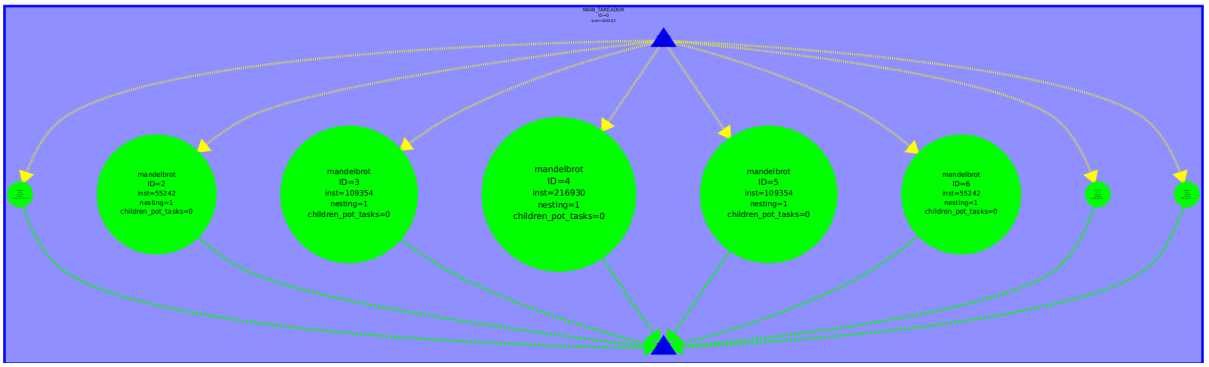
1.1 Task decomposition analysis with Tareador

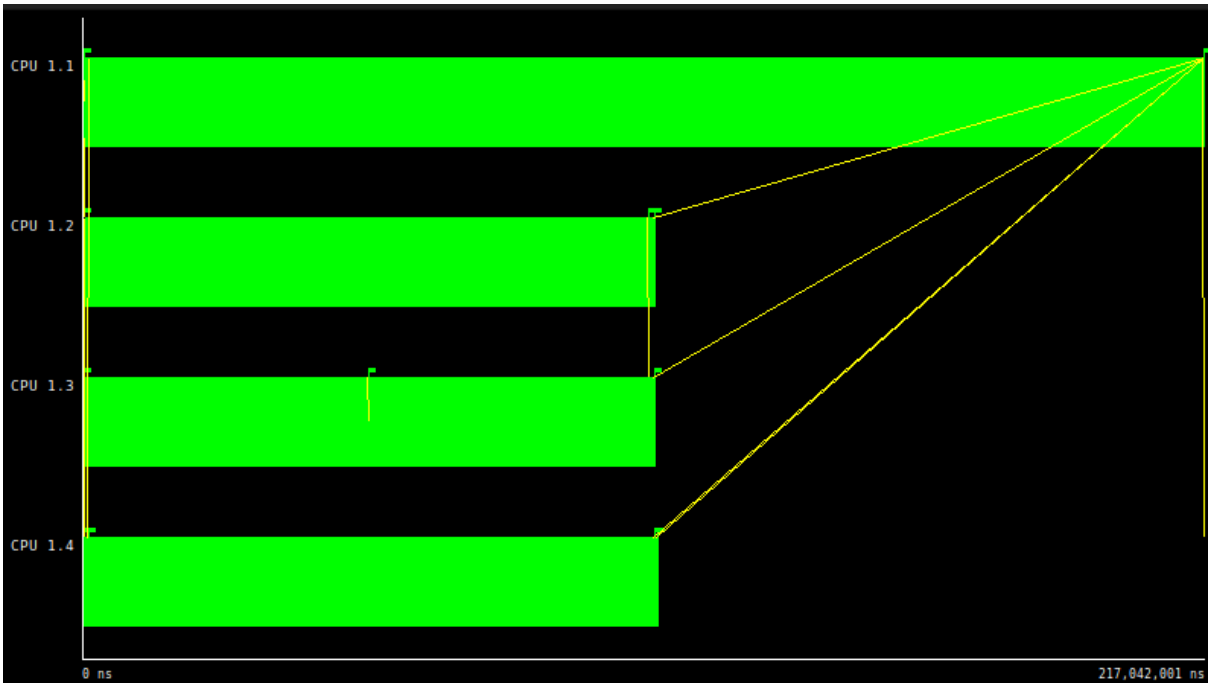
1.1.1 First analysis with Tareador .

This part refers to point 2 of subsection 2.2 of the practice document. Include the TDG of the first Tareador analysis (without -d nor -h option). Comments/Observations

Which are the two most important characteristics for the task graph that is generated?

Una de las particularidades del código original es que algunas de las tareas de color verde varían en tamaño, siendo algunas de ellas más grandes que otras. Además, otra característica distintiva es que las tareas no tienen dependencias entre sí, excepto por su única dependencia con la tarea inicial, identificada por su color azul.





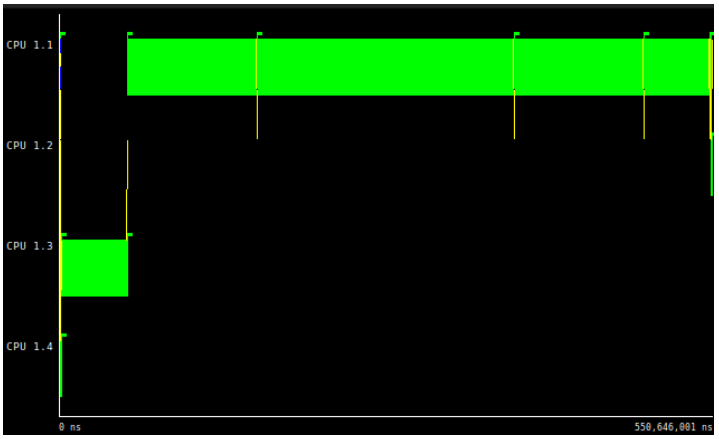
1.1.2 Second analysis with Tareador.

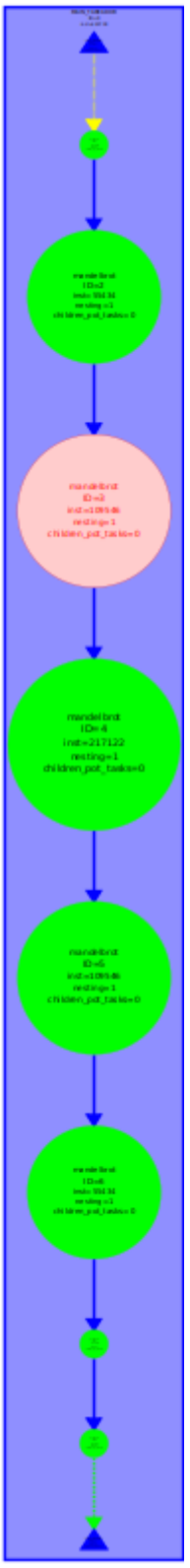
This part refers to point 3 of subsection 2.2 of the practice document. Include the TDG of the second Tareador analisis (with -d option).

Which are the two most important characteristics for the task graph that is generated? Which part of the code is making the big difference with the previous case? How will you protect this section of code in the parallel OpenMP code?

Una de las características del grafo es que, aunque algunas de las tareas de color verde siguen teniendo diferentes tamaños, la otra característica principal es que ahora existen dependencias entre las tareas. Estas dependencias generan una serialización, lo que significa que las tareas deben ejecutarse en un orden específico.

Es esencial proteger esta sección del código para evitar que múltiples hilos accedan a la misma variable y cambien su estado de color. De lo contrario, la variable podría ser modificada de manera repetida cuando en realidad debe estar protegida.

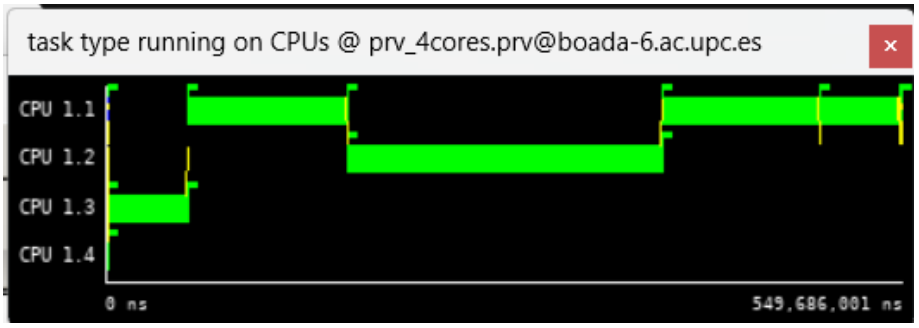




4. Finally, interactively execute mandel-tar binary using the ./run-tareador.sh, but now indicating the name of the instrumented binary and only the -h option. For the deliverable:
What does each chain of tasks in the task graph represents? Which part of the code is making the big difference with the two previous cases? How will you protect this section of code in the parallel OpenMP code that you will program in the next sessions? Don't forget to save the new TDG generated for later inclusion in the deliverable.

Similar a los dos gráficos anteriores, algunas de las tareas verdes varían en tamaño, con algunas siendo más grandes que otras. Además, en el grafo actual, hay dependencias que generan una serialización entre todas las tareas, y estas dependencias se han vuelto más complejas en comparación con versiones anteriores.

En este código, los hilos acceden a las posiciones k-1 en el histograma, lo que hace que el código se vuelva secuencial, ya que no pueden acceder a un elemento que aún no ha sido creado. También se repite el problema del código anterior, donde los hilos acceden simultáneamente a una ubicación de memoria, lo que resulta en resultados incorrectos. Solo un hilo debe acceder a esa zona, no todos al mismo tiempo, lo que puede provocar errores en la solución.



Once the study for the Row strategy is completed, modify the instrumented mandel-tar.c code to analyse the potential parallelism for the Point strategy. Repeat the analysis for the three previous executions.
For the deliverable: Which is the main change that you observe with respect to the Row strategy?

La principal diferencia entre las dos estrategias radica en la cantidad de tareas generadas. En la estrategia "Row," se crean significativamente más tareas debido a la tarea dentro del bucle. En ambos enfoques, la estrategia "Row" resulta más adecuada en este escenario, ya que algunas tareas son considerablemente más grandes que otras. Además, en cada ejecución, dado que se ejecutan secuencialmente, la estrategia "Row" involucra muchas menos tareas para calcular. Por otro lado, la estrategia "Point" presenta problemas con la cantidad de tareas al crear nuevas tareas en cada iteración en ambos bucles, lo que la hace menos eficiente.

1.2 Point decomposition strategy

1.2.1 Explicit tasks without taskloops

This part refers to the overall analysis with modelfactors and detailed analysis with Paraver of subsection 3.1 of the practice document. Include here the tables generated by modelfactors after executing the submit-strong-extrae.sh script and the PostScript file generated by submit-strong-omp.sh script for the OpenMP version with explicit tasks and with no taskloops. Also, include some captions of the Paraver analysis to visualise when the explicit tasks are created and executed.
Comments/Observations
Is the speed–up appropriate? Is the scalability appropriate?

La velocidad de aceleración se calcula como la relación entre el tiempo de ejecución en un solo procesador y el tiempo de ejecución en múltiples procesadores. Los valores proporcionados en la Tabla 1 indican lo siguiente:

Con 4 procesadores, el speedup es de 1.71, lo cual no es ideal (debajo de 4, que sería el speedup lineal), pero es común en la práctica debido a los costos de coordinación y comunicación entre procesadores.
Con 8 procesadores, el speedup es solo de 1.94, lo que indica que la eficiencia se reduce aún más a medida que aumenta el número de procesadores.

A medida que aumentamos a 12 y luego a 16 procesadores, el speedup aumenta muy marginalmente (1.95 y 2.01, respectivamente), lo que muestra claramente un problema de escalabilidad: el sistema no está utilizando eficazmente los procesadores adicionales.
La escalabilidad está directamente relacionada con la eficiencia. Idealmente, cuando se incrementa el número de procesadores, esperamos que la eficiencia se mantenga constante si el programa es perfectamente escalable. Sin embargo, la eficiencia disminuye significativamente de 1.00 con un solo procesador a 0.13 con 16 procesadores, lo cual es una clara indicación de que el programa no escala bien. Los problemas pueden incluir la sobrecarga de sincronización y comunicación, el balanceo de carga inadecuado, y la falta de suficiente trabajo paralelo para justificar los procesadores adicionales.

Overview of whole program execution metrics					
Number of processors	1	4	8	12	16
Elapsed time (sec)	0.63	0.37	0.33	0.33	0.32
Speedup	1.00	1.71	1.94	1.95	2.01
Efficiency	1.00	0.43	0.24	0.16	0.13

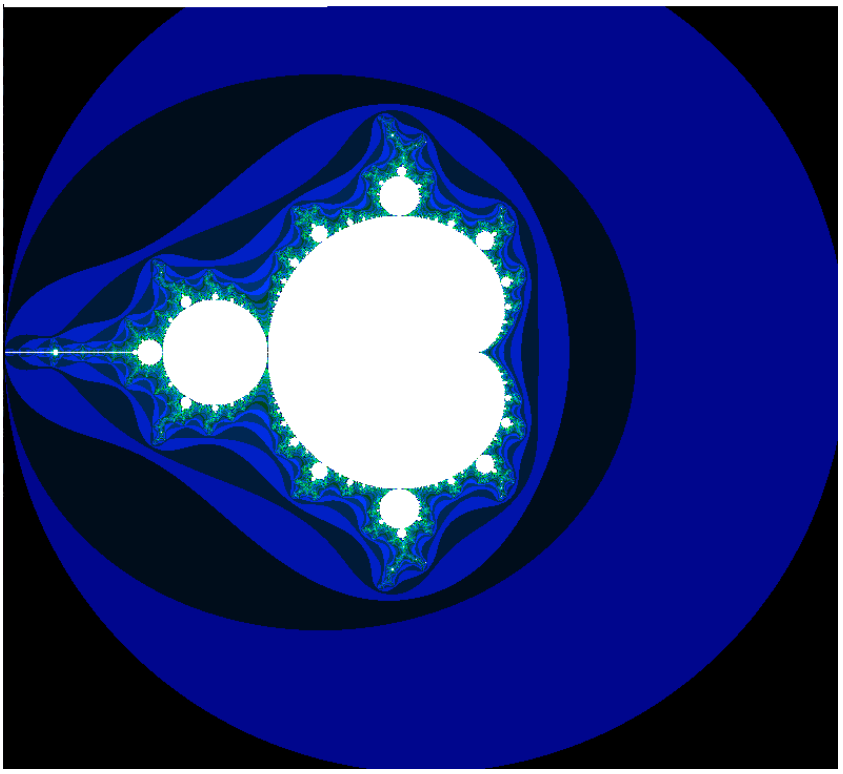
Table 1: Analysis done on Mon Nov 6 08:16:46 PM CET 2023, paco1106

Overview of the Efficiency metrics in parallel fraction, ϕ=99.94%					
Number of processors	1	4	8	12	16
Global efficiency	84.55%	36.05%	20.52%	13.73%	10.62%
Parallelization strategy efficiency	84.55%	48.18%	29.06%	20.39%	16.16%
Load balancing	100.00%	95.70%	68.02%	46.89%	33.89%
In execution efficiency	84.55%	50.35%	42.72%	43.48%	47.68%
Scalability for computation tasks	100.00%	74.83%	70.60%	67.37%	65.70%
IPC scalability	100.00%	69.88%	69.00%	68.58%	67.98%
Instruction scalability	100.00%	104.86%	105.82%	105.47%	105.60%
Frequency scalability	100.00%	102.11%	96.69%	93.15%	91.54%

Table 2: Analysis done on Mon Nov 6 08:16:46 PM CET 2023, paco1106

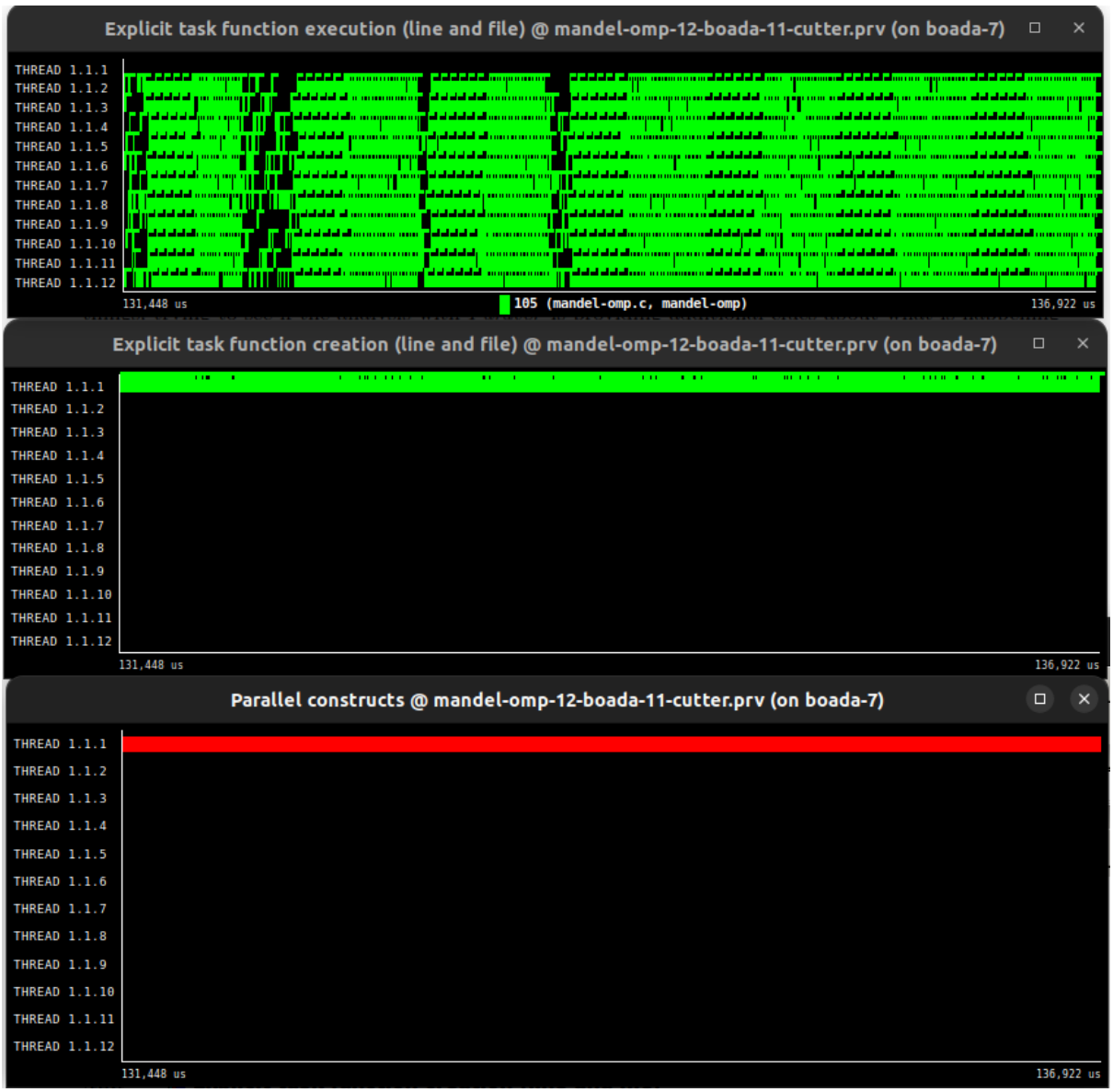
Statistics about explicit tasks in parallel fraction					
Number of processors	1	4	8	12	16
Number of explicit tasks executed (total)	102400.0	102400.0	102400.0	102400.0	102400.0
LB (number of explicit tasks executed)	1.0	0.84	0.82	0.82	0.82
LB (time executing explicit tasks)	1.0	0.82	0.88	0.89	0.91
Time per explicit task (average us)	5.18	7.63	8.83	10.15	11.07
Overhead per explicit task (synch %)	8.13	74.68	183.8	280.28	357.08
Overhead per explicit task (sched %)	4.65	23.74	20.37	17.54	14.44
Number of taskwait/taskgroup (total)	0.0	0.0	0.0	0.0	0.0

Table 3: Analysis done on Mon Nov 6 08:16:46 PM CET 2023, paco1106



Which threads are creating and executing the explicit tasks? Is one of the threads devoted to create them, and therefore executing much less? If yes, this should be a clear contributor to load unbalance, right? Do you think the granularity of the tasks is appropriate for this parallelization strategy?

Todos los hilos participan tanto en la creación como en la ejecución de tareas



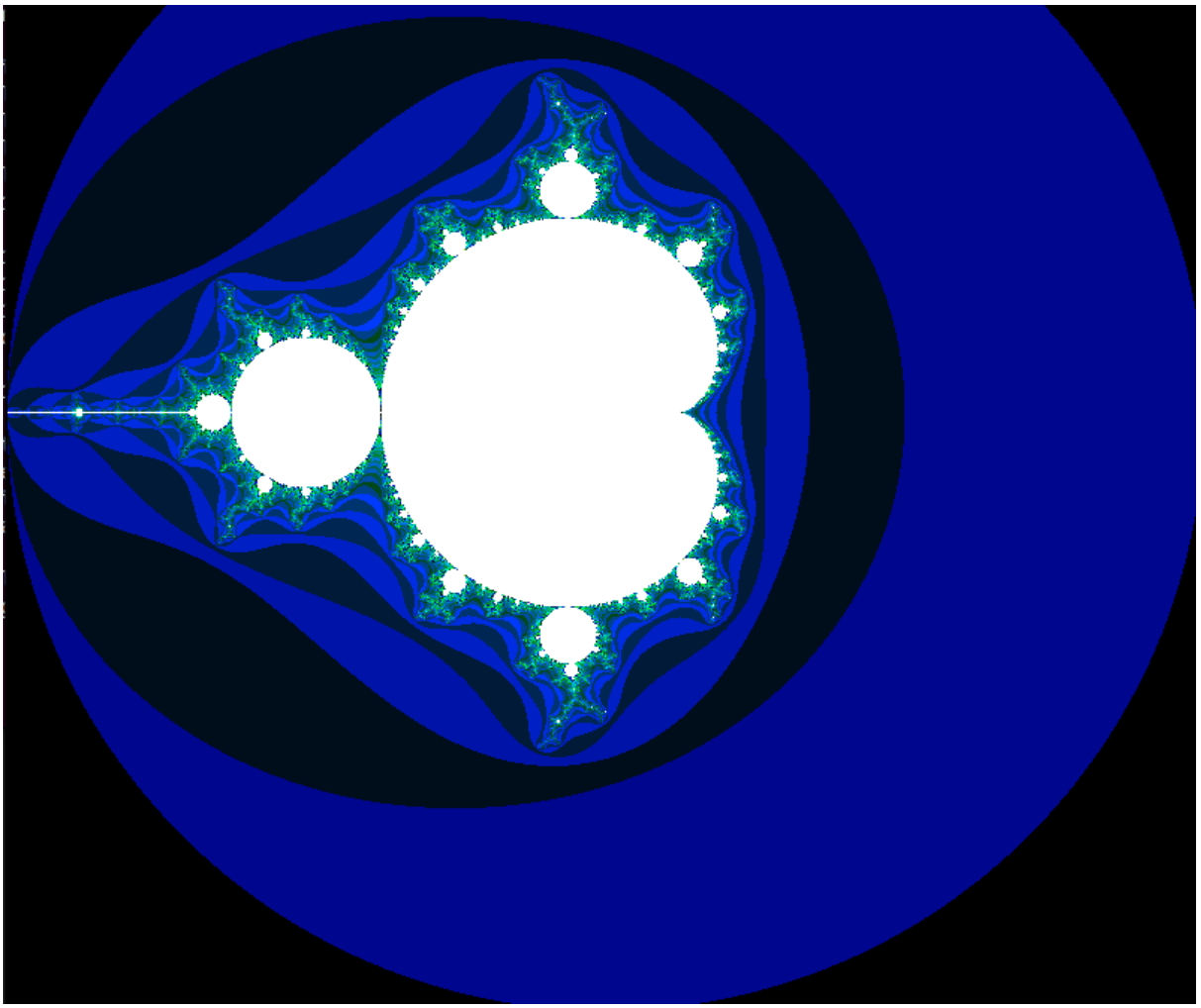
1.2.2 Taskloops without using neither num tasks nor grainsize

This part refers to the optimization: granularity control using taskloop, overall analysis part of subsec-tion 3.1 of the practice document.

Include here the modelfactors tables generated by the submit-strong-extrae.sh script and the PostScript file generated by submit-strong-omp.sh script for the OpenMP version with taskloops without using neither num tasks nor grainsize.

Is the version with taskloop performing better than the last version based on the use of task?

La versión con taskloop es considerablemente más eficiente, ejecuta tareas más rápidamente, tiene mejor escalabilidad, un balanceo de carga perfecto, y menos sobrecarga en comparación con la versión que utiliza solo task. Esto significa que la versión con taskloop está efectivamente desempeñando mejor que la versión basada en task.



Overview of whole program execution metrics					
Number of processors	1	4	8	12	16
Elapsed time (sec)	0.41	0.12	0.06	0.05	0.04
Speedup	1.00	3.42	6.48	8.62	11.18
Efficiency	1.00	0.86	0.81	0.72	0.70

Table 1: Analysis done on Mon Nov 6 08:58:46 PM CET 2023, paco1106

Overview of the Efficiency metrics in parallel fraction, ϕ=99.93%					
Number of processors	1	4	8	12	16
Global efficiency	99.99%	85.67%	81.67%	72.67%	70.92%
Parallelization strategy efficiency	99.99%	88.92%	91.66%	85.49%	84.52%
Load balancing	100.00%	89.30%	93.24%	88.69%	89.27%
In execution efficiency	99.99%	99.58%	98.31%	96.39%	94.68%
Scalability for computation tasks	100.00%	96.34%	89.10%	85.01%	83.92%
IPC scalability	100.00%	98.02%	97.33%	96.41%	96.63%
Instruction scalability	100.00%	100.00%	99.99%	99.99%	99.98%
Frequency scalability	100.00%	98.29%	91.54%	88.18%	86.86%

Table 2: Analysis done on Mon Nov 6 08:58:46 PM CET 2023, paco1106

Statistics about explicit tasks in parallel fraction					
Number of processors	1	4	8	12	16
Number of explicit tasks executed (total)	10.0	40.0	80.0	120.0	160.0
LB (number of explicit tasks executed)	1.0	0.59	0.32	0.22	0.17
LB (time executing explicit tasks)	1.0	0.89	0.93	0.89	0.88
Time per explicit task (average us)	41159.42	10651.75	5703.42	3923.98	2938.6
Overhead per explicit task (synch %)	0.0	12.02	7.15	13.59	12.33
Overhead per explicit task (sched %)	0.01	0.07	0.19	0.32	0.42
Number of taskwait/taskgroup (total)	1.0	1.0	1.0	1.0	1.0

Table 3: Analysis done on Mon Nov 6 08:58:46 PM CET 2023, paco1106

Should you blame to ”load balancing” or to ”in execution efficiency”? Remenber to scan the information provided in the third table that is included in the modelfactor-tables.pdf file. What can you tell now from the total number of tasks generated and the new average execution time for explicit tasks and the synchronization and scheduling overheads percentage? How many tasks are executed per taskloop? Is the task granularity better? But notice that task synchronization still takes a big %.

Basándonos en las dos métricas que contribuyen a la eficiencia de la estrategia de paralelización, es decir, la eficiencia del balanceo de carga y la eficiencia en ejecución, podemos observar lo siguiente entre las versiones con `task` y `taskloop`:

1. **Balanceo de carga (Load balancing)**:
    - Con `taskloop`: Se mantiene en 100%, lo que indica un balanceo de carga perfecto.
    - Con `task`: Varía entre 85.70% y 89.27%, lo que sugiere que hay una menor eficiencia en el balanceo de carga.

2. **Eficiencia en ejecución (In execution efficiency)**:
    - Con `taskloop`: Varía entre 99.90% y 84.27%, lo cual es bastante alto.
    - Con `task`: Varía entre 84.55% y 47.68%, lo que indica que hay una mayor caída de la eficiencia en ejecución con el aumento del número de procesadores.

De estos datos se desprende que la eficiencia de la estrategia de paralelización es mayor en la versión con `taskloop`. El "load balancing" no es un problema en esta versión, mientras que sí parece ser un factor limitante en la versión con `task`. Por otro lado, la eficiencia en ejecución también es mejor en la versión con `taskloop`.

Examinando la información proporcionada en la tercera tabla:

- **Número total de tareas generadas (Number of explicit tasks executed)**: Se mantiene constante en ambos casos, lo que significa que se generan la misma cantidad de tareas independientemente de la versión utilizada.
- **Nuevo tiempo promedio de ejecución para tareas explícitas (Time per explicit task (average us))**: El tiempo promedio disminuye considerablemente en la versión con `taskloop`, lo que indica que cada tarea se ejecuta más rápido.
- **Sobrecargas de sincronización y programación (Overhead per explicit task (sched %))**: Aunque hay una reducción en la versión con `taskloop`, el porcentaje sigue siendo significativo. Esto implica que, aunque la granularidad de las tareas puede ser mejor (dado el menor tiempo de ejecución promedio), la sincronización todavía representa un porcentaje considerable del tiempo.
- **Número de tareas ejecutadas por `taskloop`**: No se proporciona un número directo de tareas por `taskloop` en los datos, pero la mejora en el tiempo promedio de ejecución sugiere que las tareas son manejadas de manera más eficiente.

En resumen, aunque la versión con `taskloop` mejora significativamente la granularidad de las tareas y la eficiencia general, la sincronización sigue ocupando un porcentaje importante, lo cual puede ser un área para buscar optimizaciones adicionales.

Why there are now task synchronizations in the execution? Where are these task synchronisations happening?

Las sincronizaciones de tareas en la ejecución se deben a la necesidad de coordinar el trabajo entre múltiples tareas que se ejecutan en paralelo. Esto es fundamental para asegurar que las dependencias de datos se manejen correctamente y que las tareas no accedan ni modifiquen datos de forma concurrente de manera inapropiada, lo que podría llevar a condiciones de carrera o resultados inconsistentes.

1.2.3 Taskloops without nogroup

This part refers to the optimization: granularity control using taskloop, detailed analysis and possible optimization part of subsection 3.1 of the practice document.
Include here the modelfactors tables generated by the submit-strong-extrae.sh script for the OpenMP version with taskloops with nogroup clause.
Comments/Observations
Has the scalability improved? What about task granularity and overheads?

El uso de "nogroup" parece tener un efecto positivo en la escalabilidad, la granularidad de las tareas y los sobrecostos. Hay una mejora ligera en la aceleración (speedup) y la eficiencia, y una reducción más notable en el tiempo promedio por tarea y en el sobrecosto asociado con la gestión de tareas. En resumen, "nogroup" mejora el rendimiento al reducir el sobrecosto de la gestión de tareas en OpenMP, aunque las diferencias no son muy grandes.

Overview of whole program execution metrics					
Number of processors	1	4	8	12	16
Elapsed time (sec)	0.41	0.12	0.06	0.05	0.04
Speedup	1.00	3.50	6.59	8.67	11.23
Efficiency	1.00	0.87	0.82	0.72	0.70

Table 1: Analysis done on Mon Nov 6 09:26:20 PM CET 2023, paco1106

Overview of the Efficiency metrics in parallel fraction, ϕ=99.88%					
Number of processors	1	4	8	12	16
Global efficiency	99.99%	87.78%	82.82%	72.63%	70.73%
Parallelization strategy efficiency	99.99%	89.80%	91.51%	85.49%	85.25%
Load balancing	100.00%	90.38%	92.91%	87.93%	89.19%
In execution efficiency	99.99%	99.35%	98.49%	97.22%	95.58%
Scalability for computation tasks	100.00%	97.76%	90.51%	84.96%	82.97%
IPC scalability	100.00%	99.04%	98.34%	96.52%	96.24%
Instruction scalability	100.00%	100.00%	99.99%	99.99%	99.98%
Frequency scalability	100.00%	98.70%	92.05%	88.04%	86.23%

Table 2: Analysis done on Mon Nov 6 09:26:20 PM CET 2023, paco1106

Statistics about explicit tasks in parallel fraction					
Number of processors	1	4	8	12	16
Number of explicit tasks executed (total)	10.0	40.0	80.0	120.0	160.0
LB (number of explicit tasks executed)	1.0	0.59	0.32	0.22	0.17
LB (time executing explicit tasks)	1.0	0.9	0.93	0.88	0.9
Time per explicit task (average us)	41113.58	10462.23	5614.06	3923.42	2952.49
Overhead per explicit task (synch %)	0.0	10.87	7.54	13.66	12.19
Overhead per explicit task (sched %)	0.01	0.04	0.13	0.31	0.44
Number of taskwait/taskgroup (total)	0.0	0.0	0.0	0.0	0.0

Table 3: Analysis done on Mon Nov 6 09:26:20 PM CET 2023, paco1106