Lab 4: Parallel Task Decomposition Implementation and Analysis

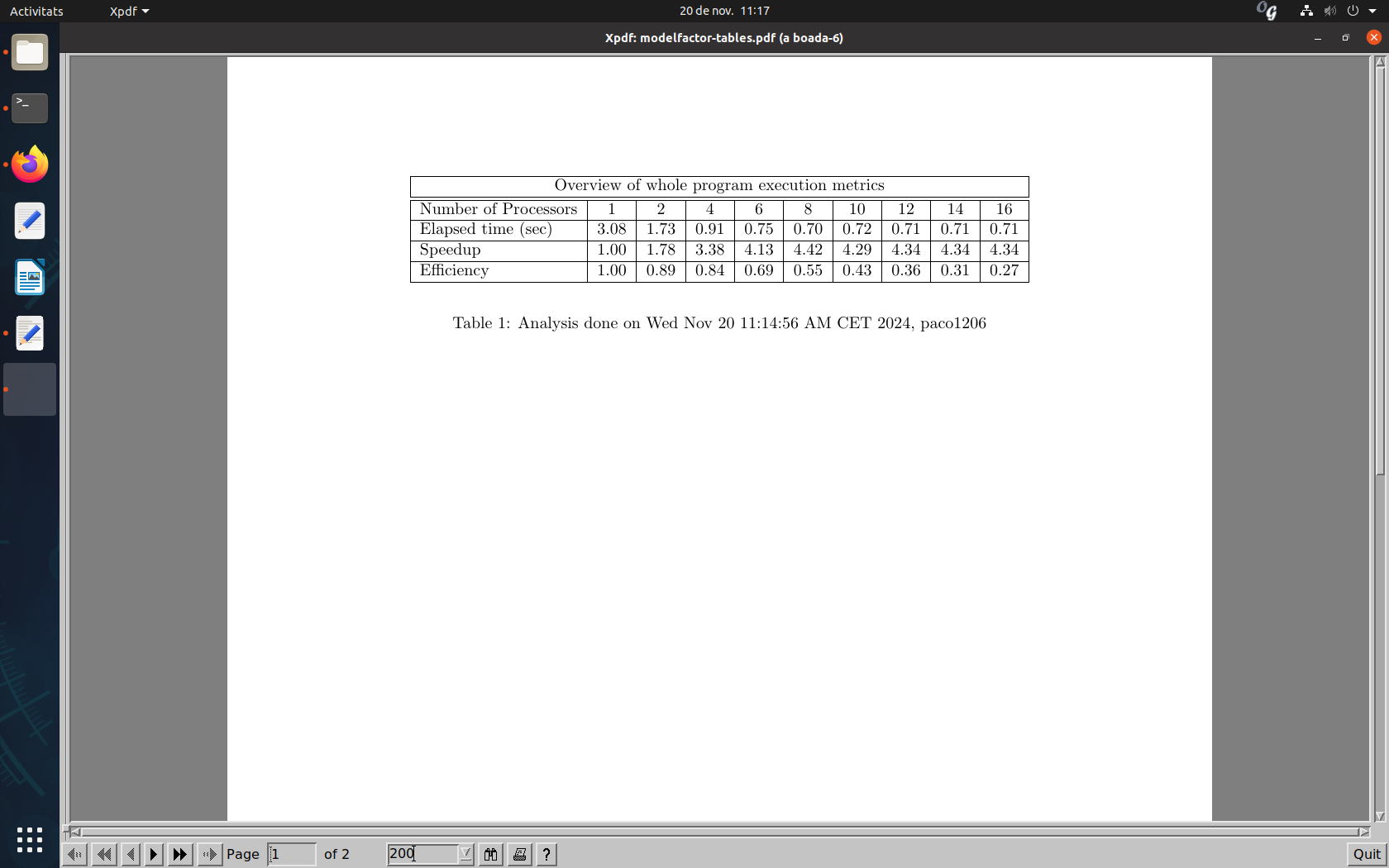
Number of threads

| version | 1 | 4 | 8 | 12 | 16 | 20 |
| --- | --- | --- | --- | --- | --- | --- |
| Iterative: Tile | 3.09s | 0.917s | 0.704s | 0.715s | 0.713s | 0.712s |
| Iterative: Fine-Grain | 3.8s | 0.966s | 0.52s | 0.36s | 0.284s | 0.23s |
| Recursive: Leaf | 1,612 | 1,258 | 1,335 | 1,374 | 1,377 | 1,379 |
| Recursive: Tree | 1,605 | 0,43 | 0,251 | 0,187 | 0,153 | 0,13 |

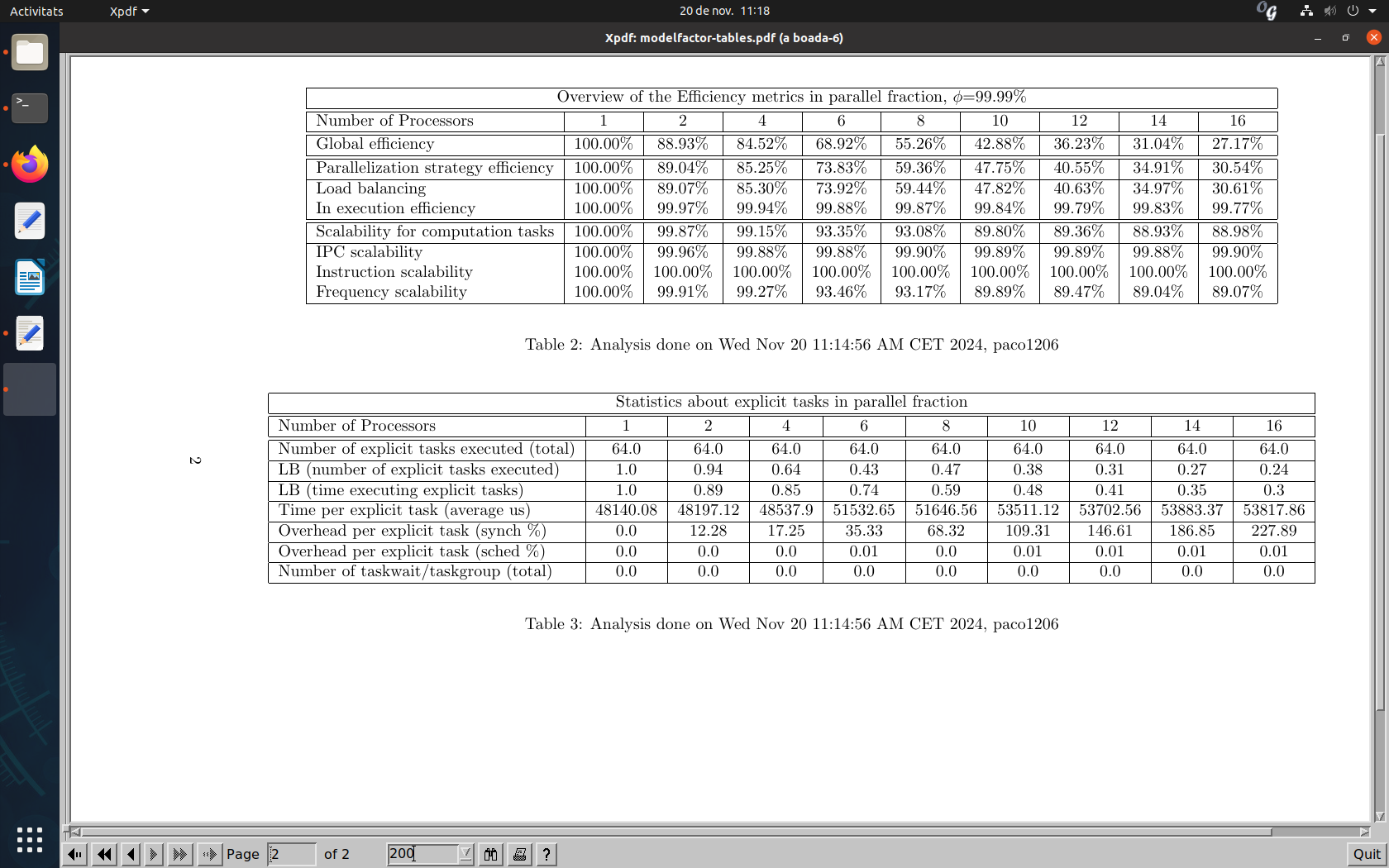
**Iterative task decomposition**

Iterative: Tile

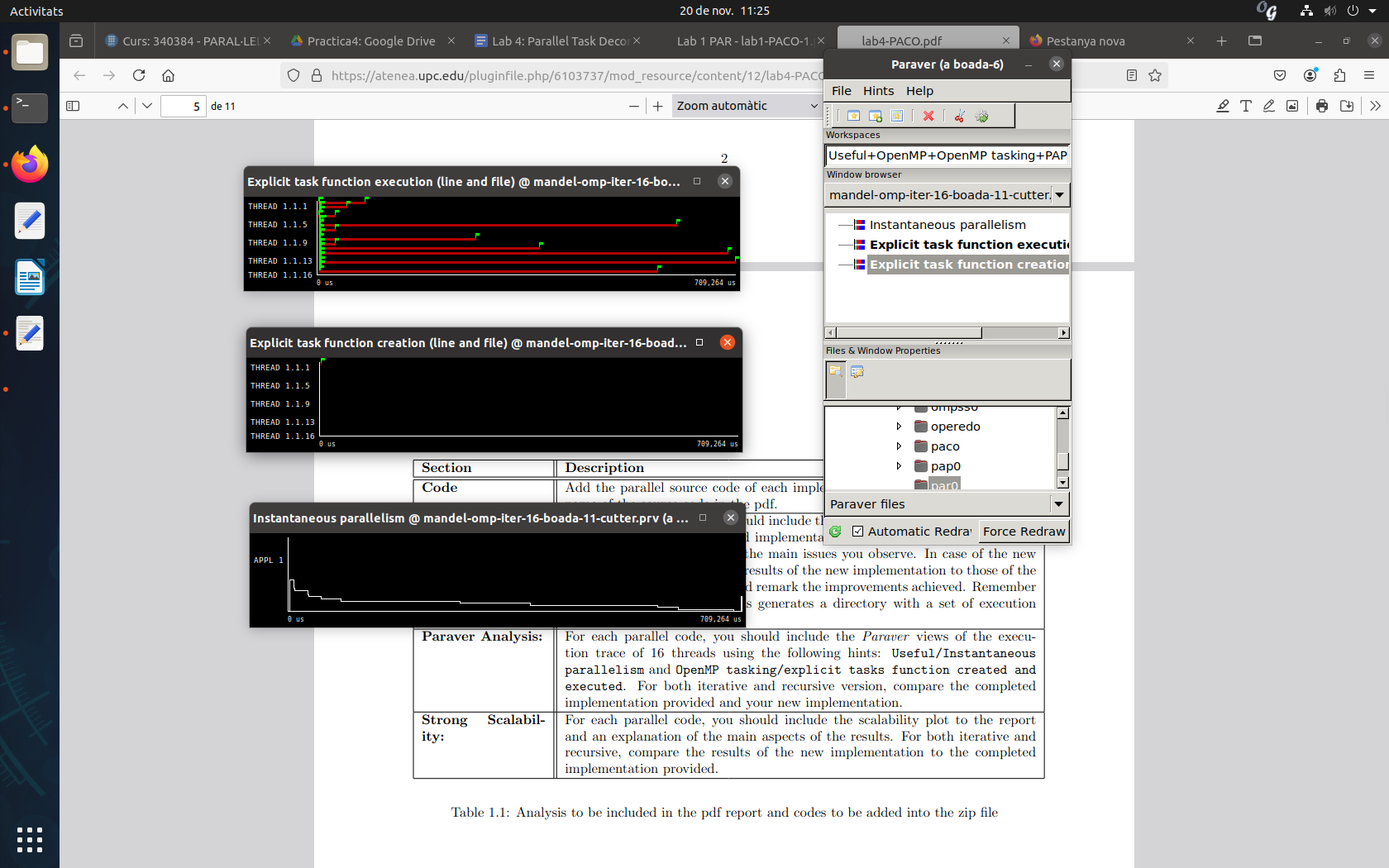
Modelfactor tables:



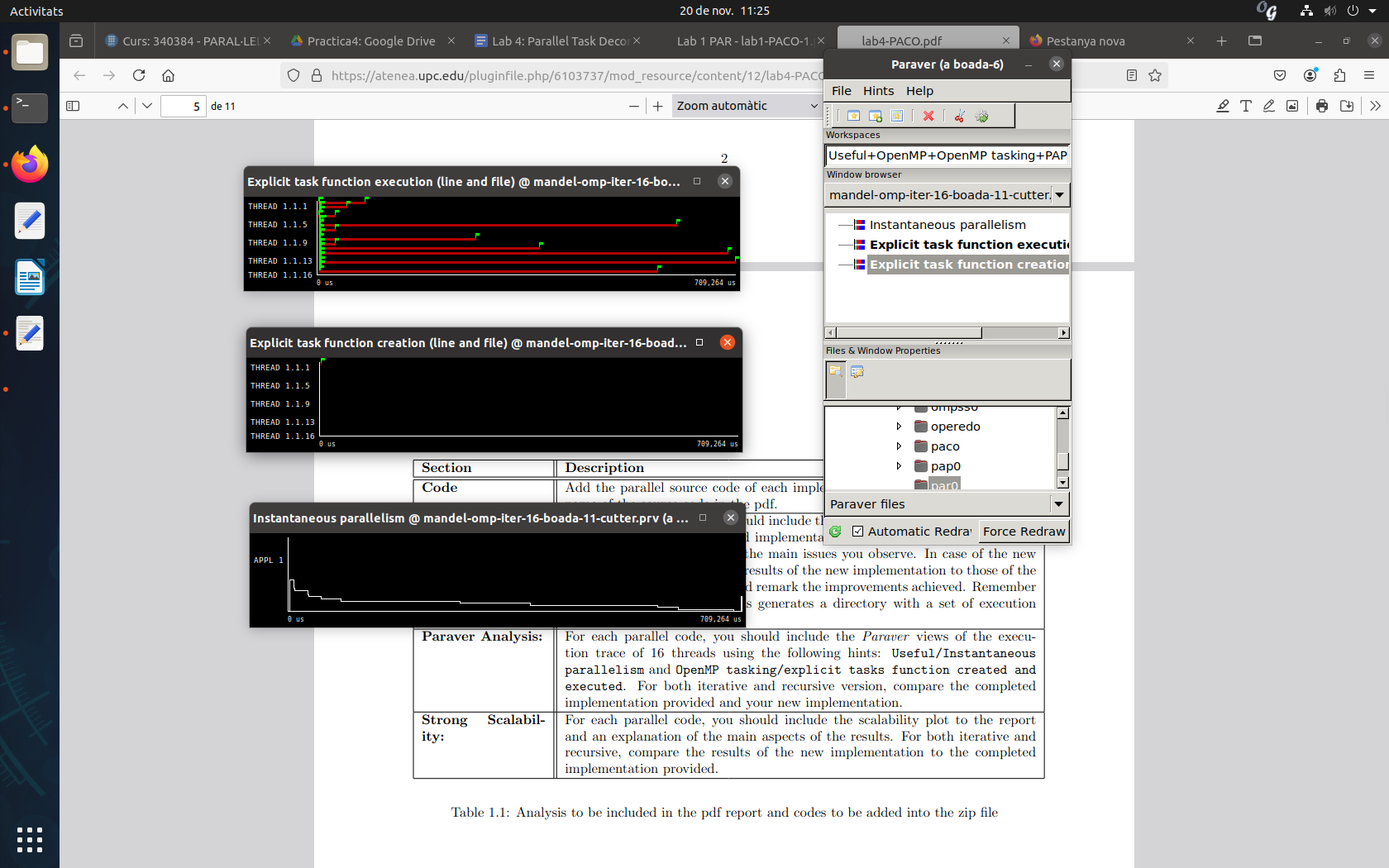
La primera taula ens mostra com la paral·lelització implementada no millora a partir de 8 threads. El temps no millora encara que utilitzem més threads, és a dir, T8 = Tinf. El speedup tampoc millora a partir de 8 threads. L'eficiència disminueix amb el número de threads, això pot ser degut a un desequilibri en la càrrega de treball, és a dir, alguns threads fan més treball que altres fent que no sigui eficient el paral·lelisme.



La taula 2 ens torna a mostrar com la càrrega de treball no està bé distribuïda, a la fila de "Load balancing" es pot observar com el nivell va disminuint amb el número de threads. La fila de "Parallelization strategy efficency" també ens mostra que l'estratègia de paral·lelització implementada no és gaire bona, ja que l'eficiència disminueix. En "Number of explicit tasks executed (total)" es poden veure el nombre de tasques totals que s'han executat, i 64 sembla no ser suficients per tenir una bona estratègia de paralització. Lo qual provoca que el temps per tasca sigui molt gran, pel fet que les tasques implicaran molt de treball, a l'haver-hi tan poques tasques.

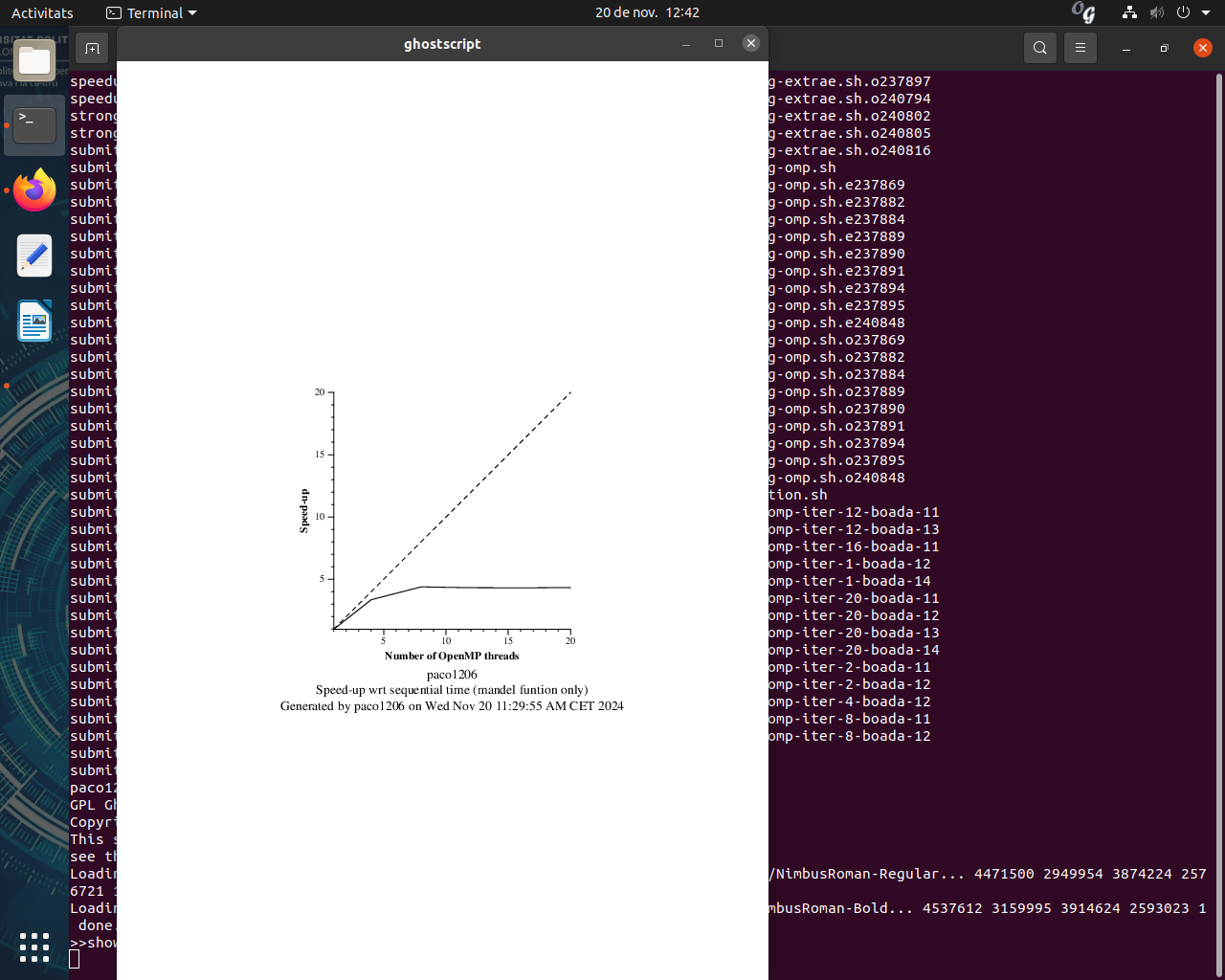
Paraver views

Es pot observar com al principi de l’execució és quan més threads treballen, però amb el temps disminueix, ja que cada un acaba la seva tasca. Aquest gràfic ens mostra el desequilibri de la càrrega de treball i com només al principi treballen tots els threads alhora.



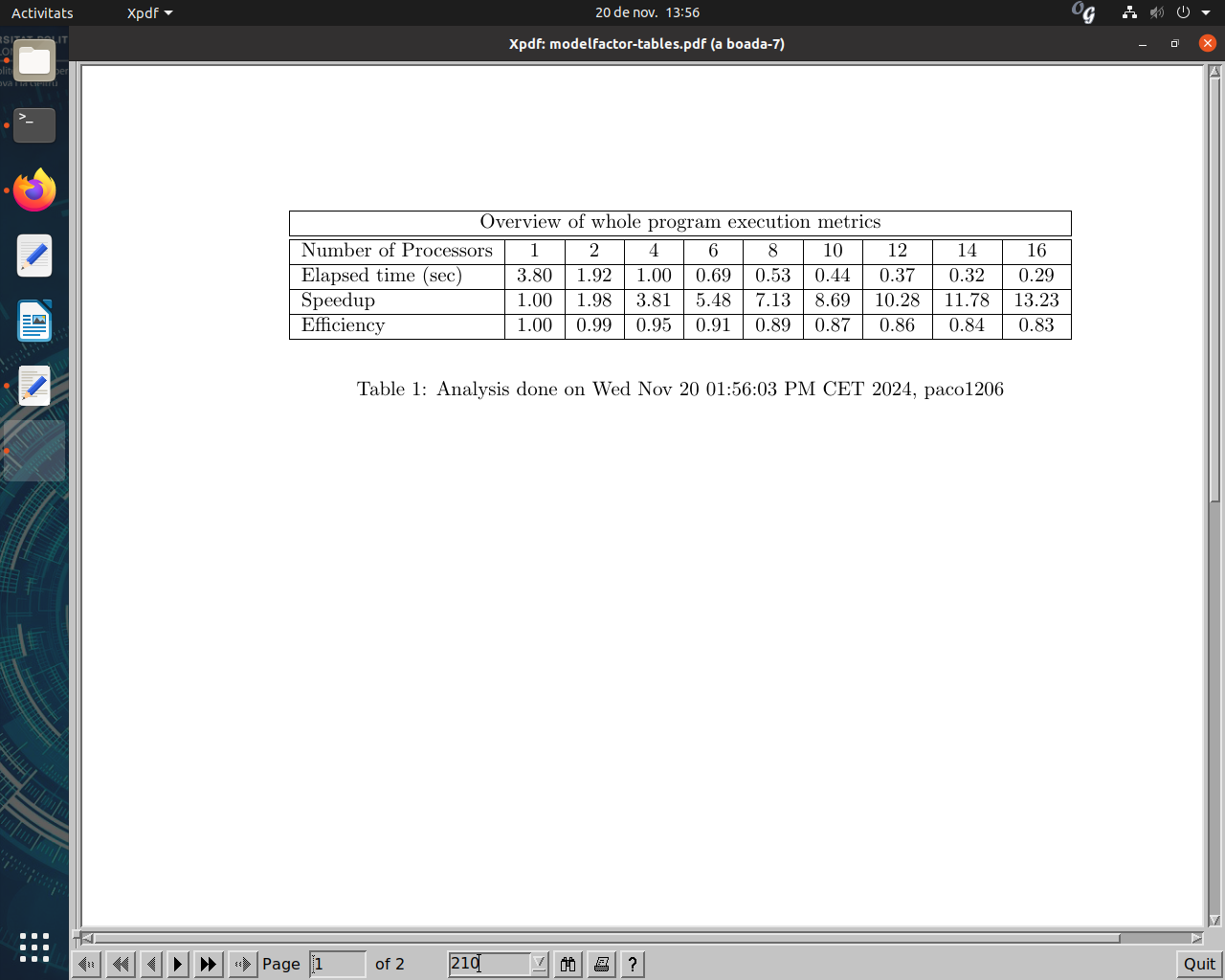
En aquesta segona captura de paraver, es pot veure com només 1 node crea les tasques, a més que totes les tasques es creen al principi de l’execució. En el gràfic de l’execució de les tasques es pot veure que la càrrega de treball és molt diferent. Alguns nodes fan molt més treball que altres i això no és eficient, ja que hi ha molts nodes que no fan res de treball quan poden fer-ho.

Strong Scalability

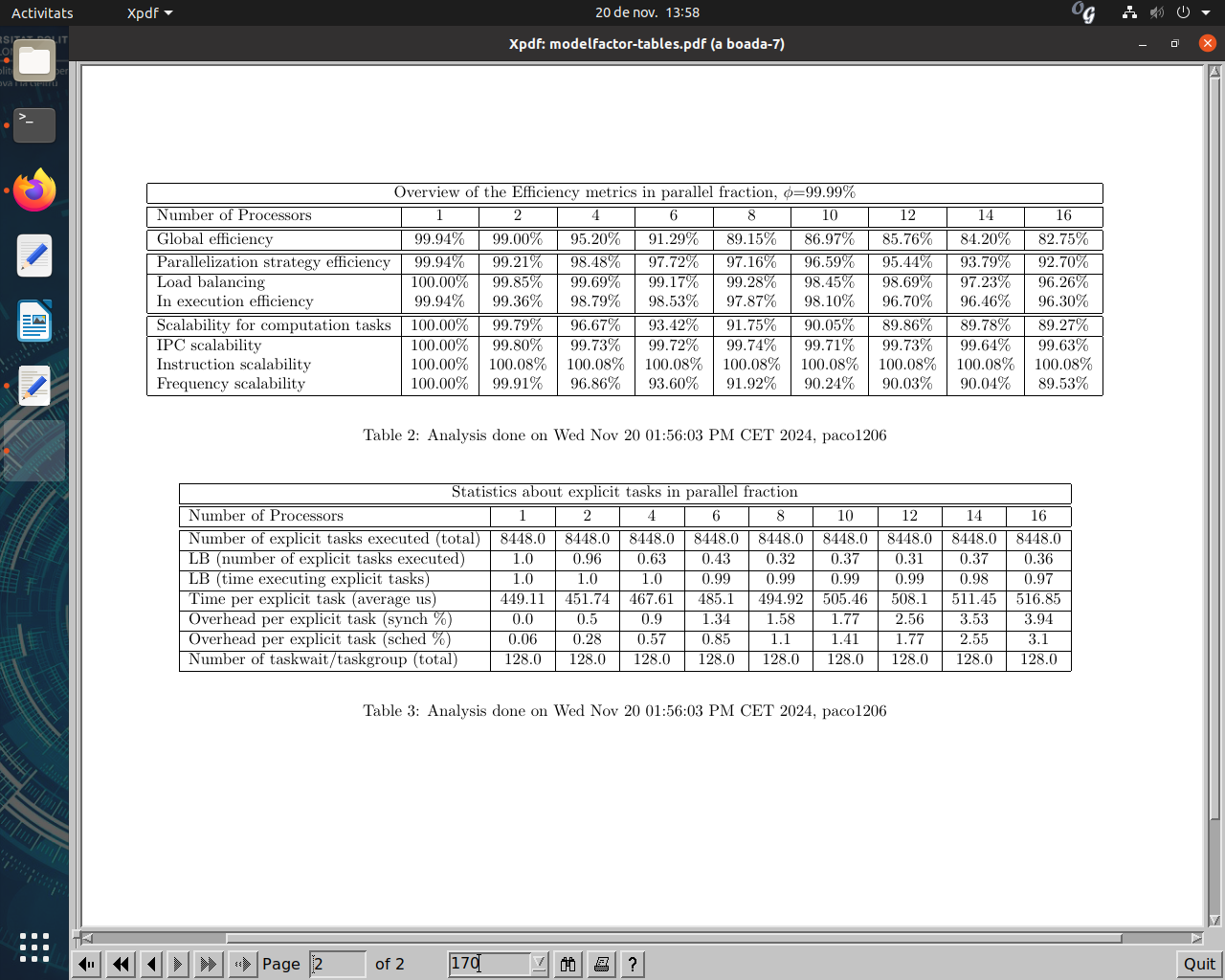


En aquest gràfic es torna a observar com amb 8 threads ja no es pot millorar el speed-up ni el temps, igual que hem vist a la primera taula del Model-Factor. A més també es pot veure que a partir de 4 threads l’escabilitat empitjora bastant, fins a 8 que arriba al seu màxim.

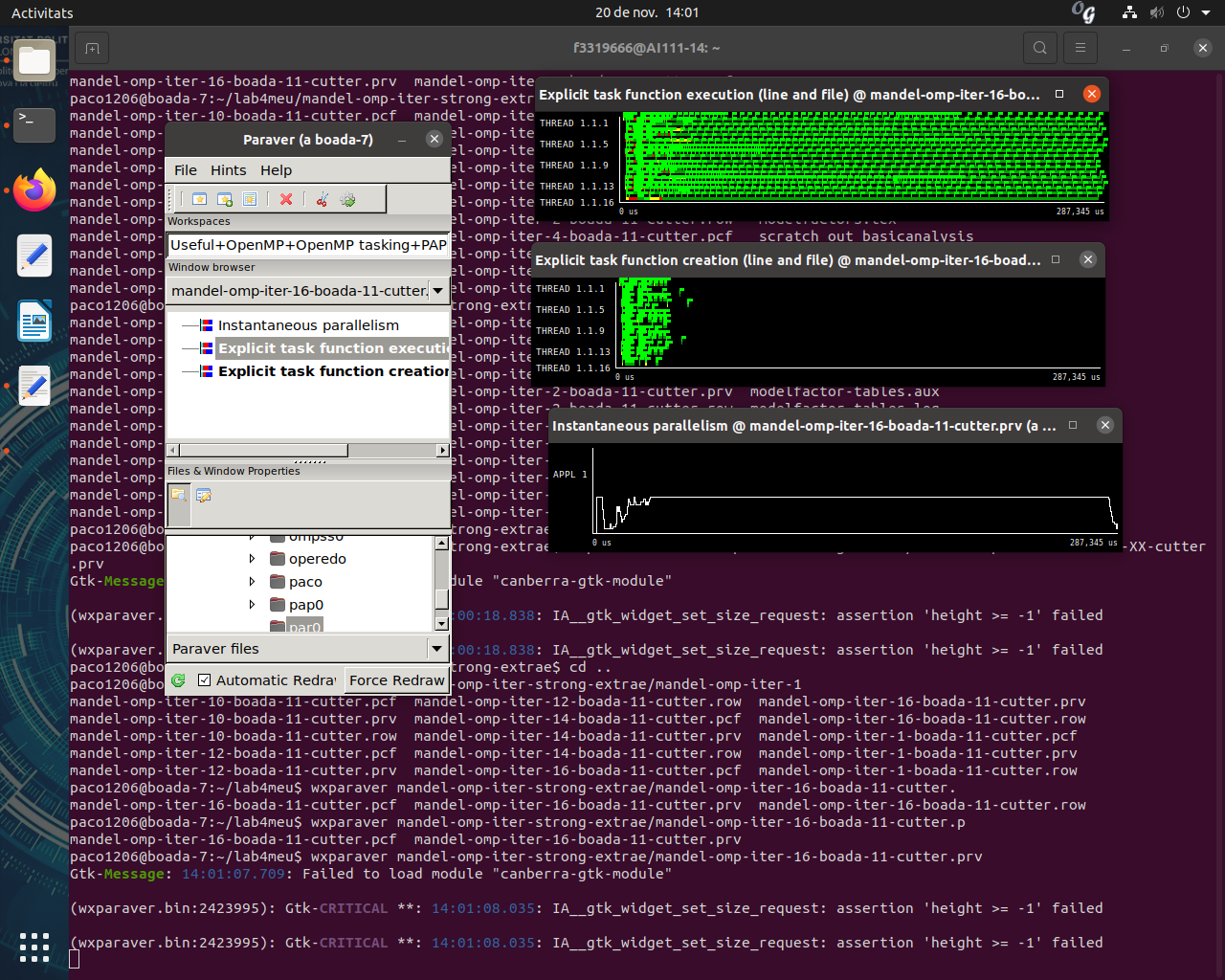
**Fine-Grain:**

ModelFactor Tables:

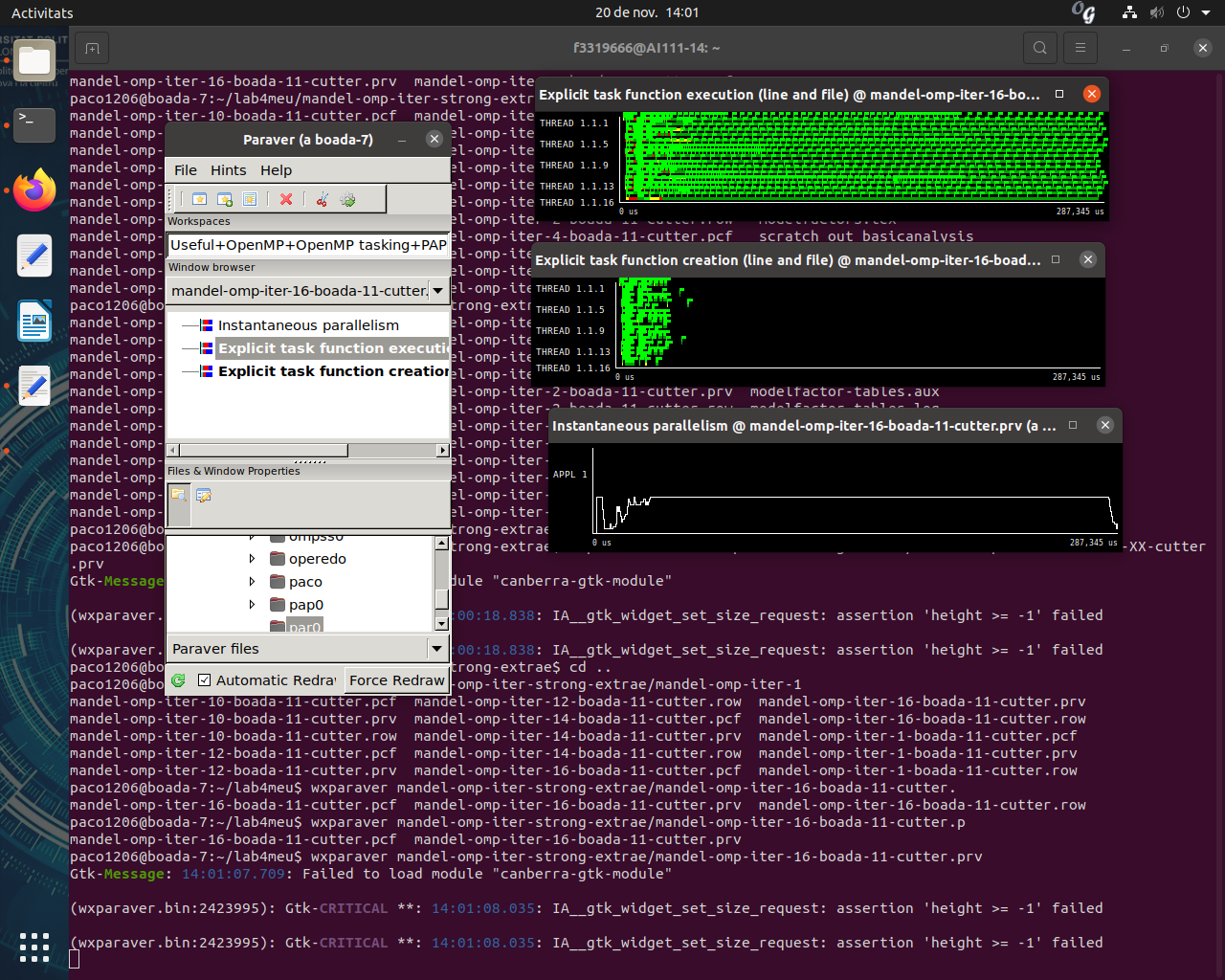
Amb el codi del Fine-grain podem veure com, a diferència de l'anterior, aquest sí que a mesura que utilitzem més threads el temps disminueix significativament, sense la barrera dels 8 threads, com ho fa també el speed up que sense cap límit, fins als 20 threads, augmenta significativament. En l'eficiència sí que veiem, a comparació de l'altre, una gran millora, on es va reduint menys a mesura que augmentem els threads.

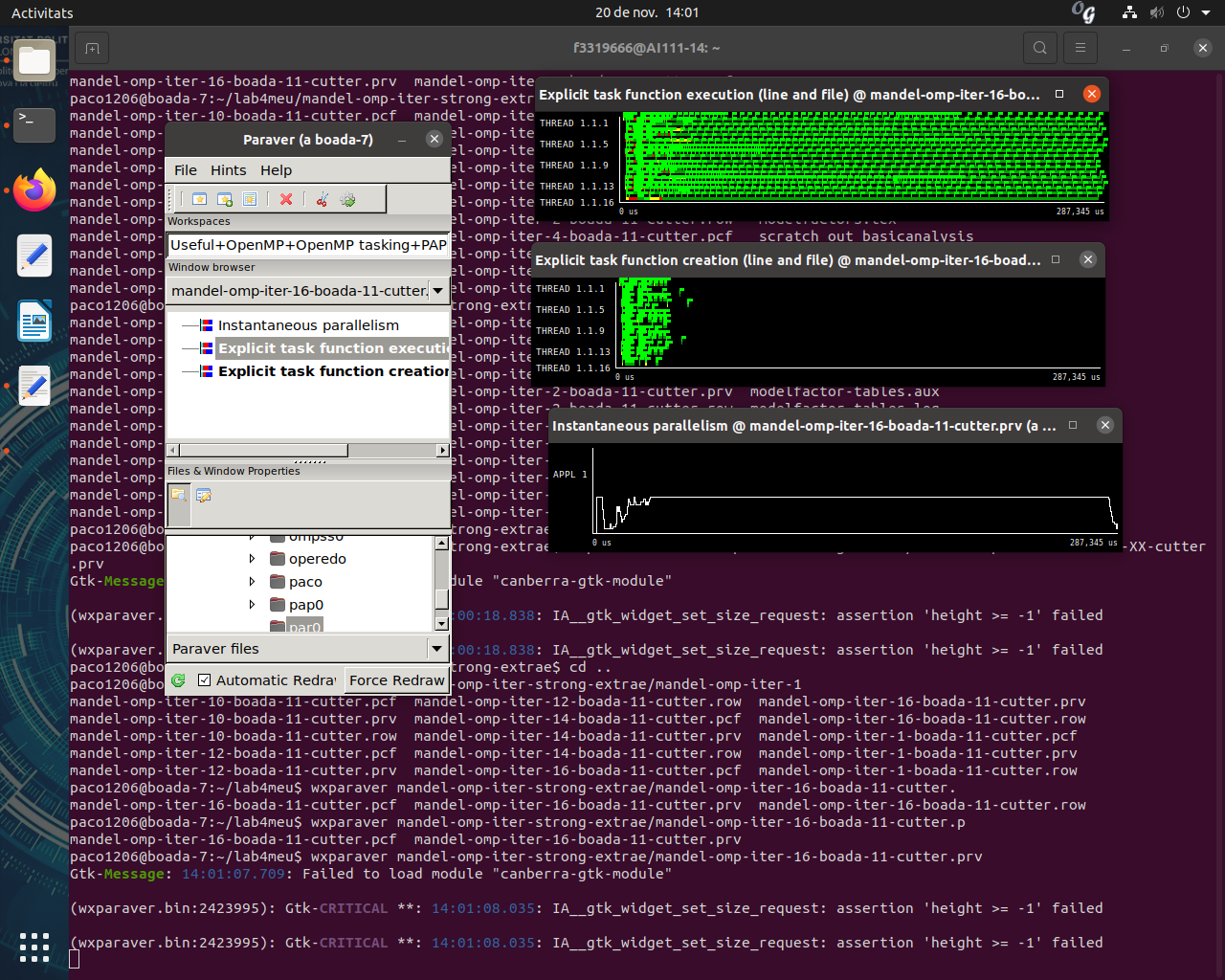
Amb aquest codi, sobretot hem millorat el balanceig de les tasques, on amb 16 threads hem passat d'un 30% a més d’un 96% el qual implica que tots els threads tenen quasi la mateixa càrrega de treball. En la “Parallelization strategy efficency” també l'hem augmentat significativament, s’ha incrementat el percentatge prop de 3 vegades el del codi anterior, sent molt més proper al 100%. Aquests augments, apropant-nos al 100% ens provoquen un increment de l'eficiència global del codi, com hem vist a la taula d’abans. D'altra banda, i fixant-nos en l'última taula veiem com el nombre de tasques executades és molt més gran en comparació amb el codi anterior, amb el qual provoquem que es redueixi el temps per tasca en un 99%, a més de tenir un millor balanceig de les taques entre els threads com hem vist a la taula d’abans.

Paraver Views:

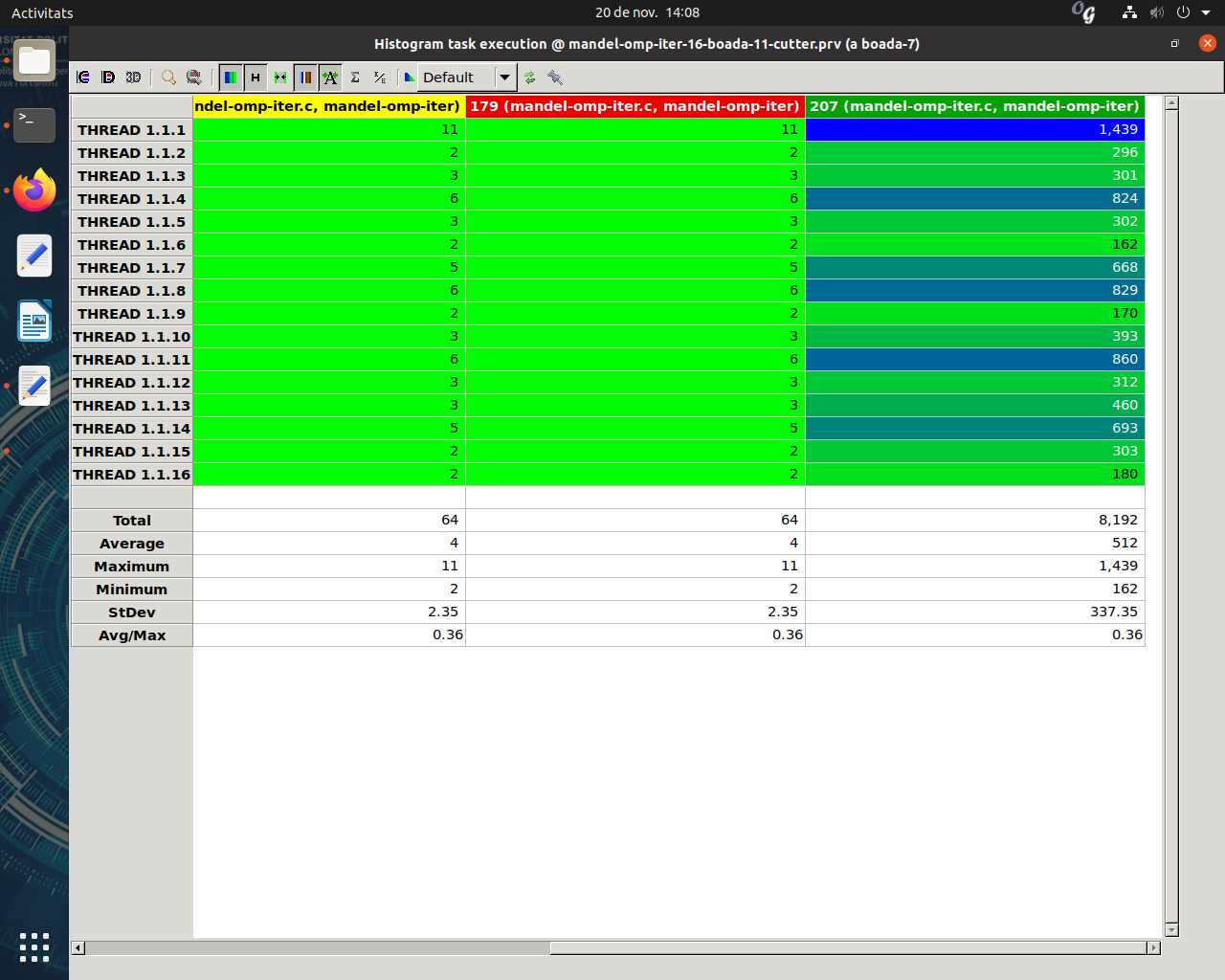


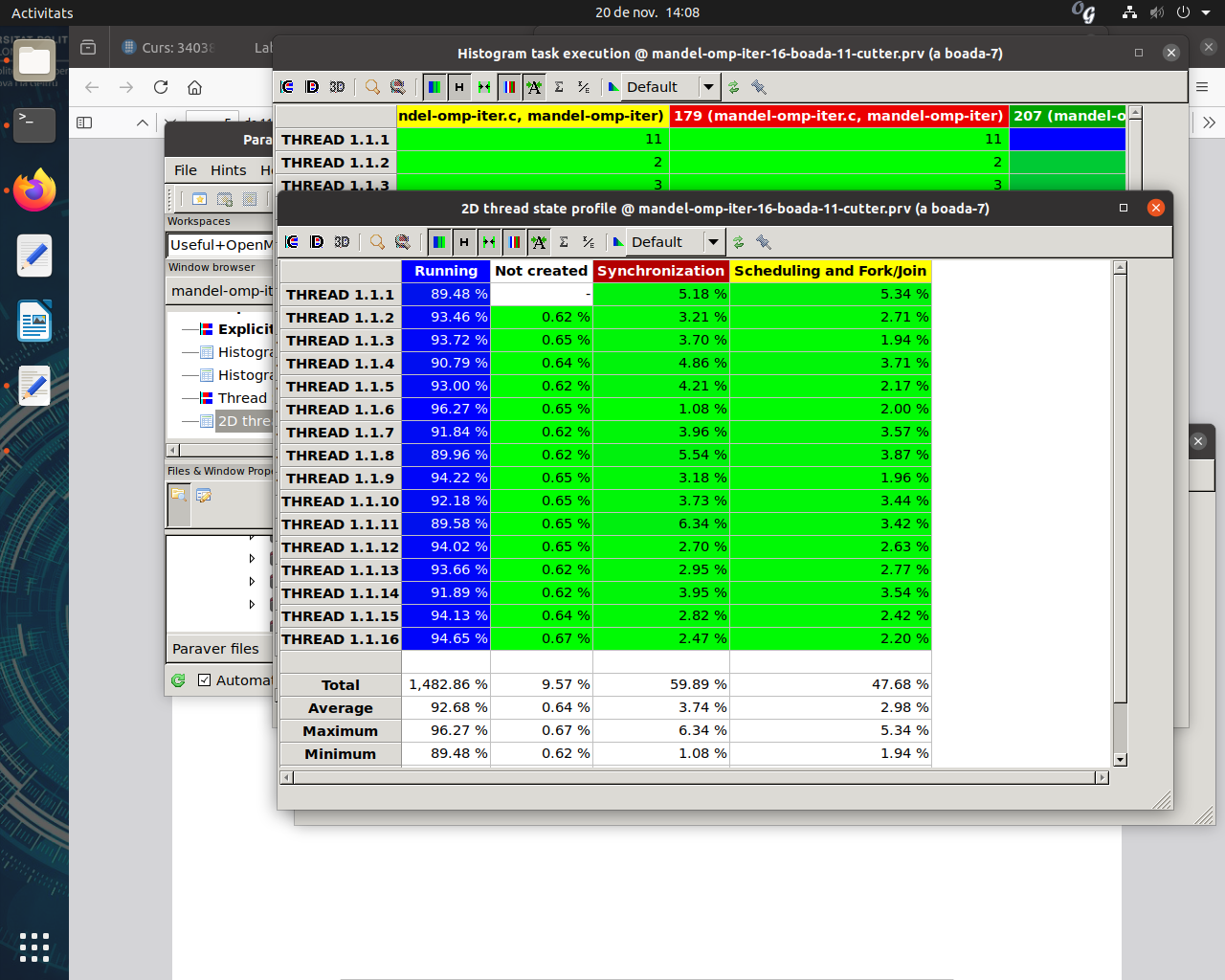
En el codi anterior, en el gràfic podíem apreciar el desbalanceig de les taques amb un gran descens en el gràfic a l'inici, ja que alguns threads acaben les seves tasques, però d’altres els implicava molt més temps. En aquest gràfic del codi fine-grain podem veure com hem millorat el balanceig de les tasques fent que la càrrega de treball sigui similar per a tots els threads i, per tant, durant la mateixa quantitat de temps aquests estiguin treballant, i acabin quasi al mateix temps.





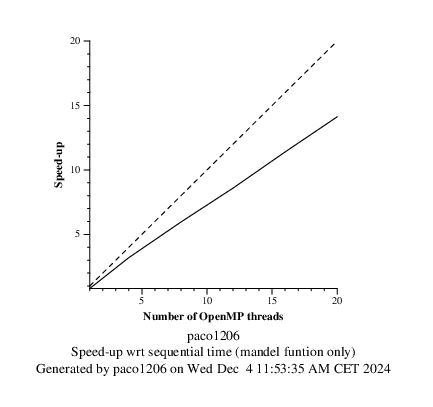
En aquest gràfic podem veure un altre cop el gran balanceig que hem obtingut en aquest codi, si comparem amb el d'abans hi havia nodes que feien una gran quantitat de les tasques, aquí podem veure com tots tenen la mateixa quantitat de tasques i a més podem veure com hi ha moltes més tasques que impliquen menys temps a diferència d’abans que tenim poques tasques amb gran quantitat de temps.





En aquesta taula veiem, a la primera columna, un quasi perfecte balanceig de les tasques, on tots els nodes fan un percentatge de temps executant tasques similars, també veiem que el temps que els threads esperen a d’altres a què acabin les tasques (taula synchronization) no és molt elevat i és similar entre ells.

Strong Scalability:

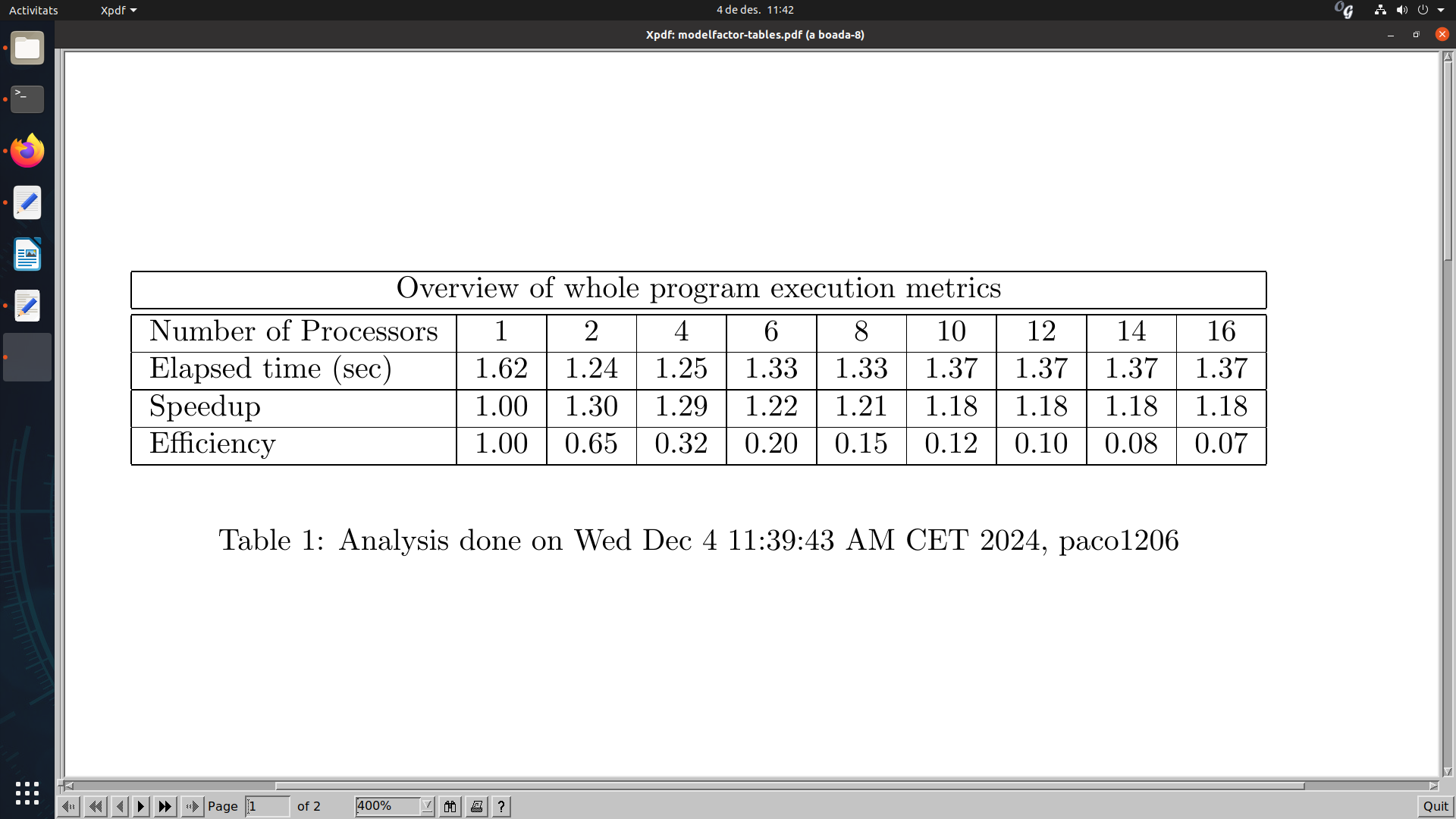


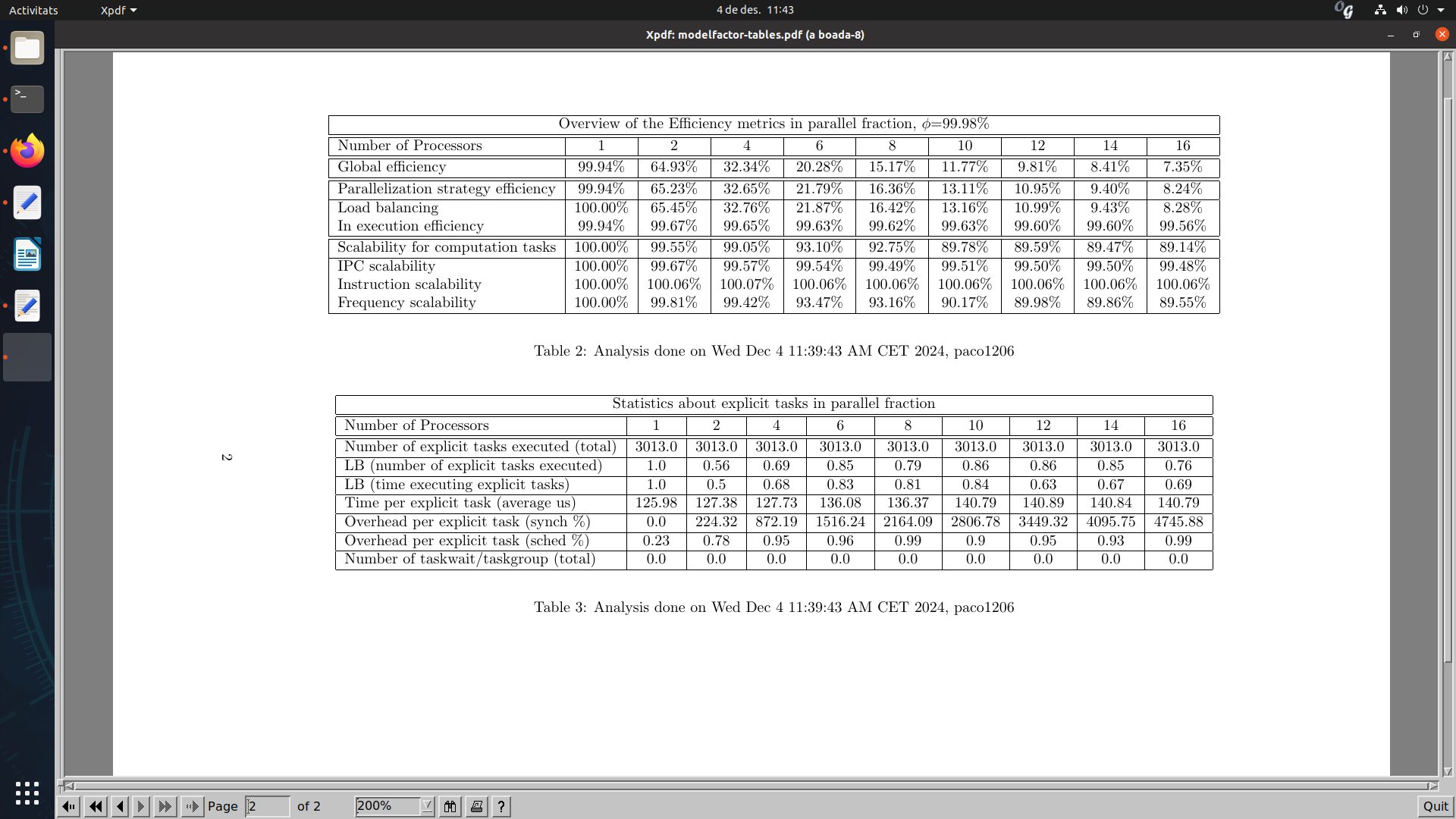
En el gràfic podem veure com a mesura que hi ha més threads el speed-up augmenta, cosa que dona valor al codi, ja que, a més està proper a la relació threads-speed-up ideal.

**Recursive task decomposition**

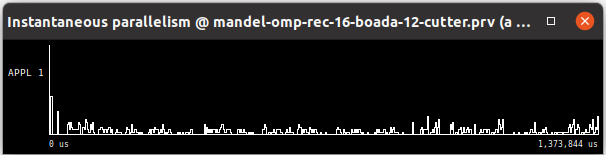
Recursive: Leaf

Modelfactor tables:

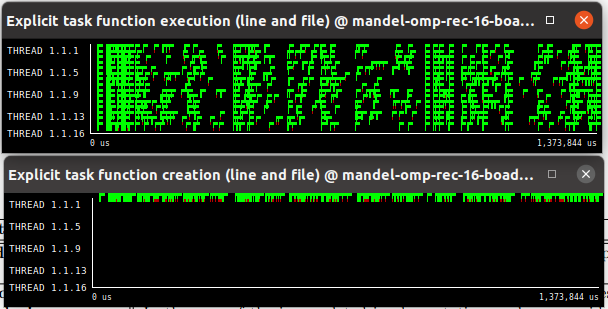
En aquesta taula podem veure primer de tot com es disminueix el temps d'execució si incrementem els threads d'un a dos, però a partir dels dos threads, incrementen els threads no obtenim cap reducció de temps, sinó que, inclús, s’incrementa una mica el temps. A més l'eficiència es redueix a uns nivells que amb 16 threads sent quasi 0.

En la primera taula d’aquestes dues veiem un greu problema amb el balanceig de les tasques on amb 16 threads el balanceig és del 8% quan perquè estiguin ben balancejades hauria d’estar proper al 100%, això provoca que l'eficiència de l'estratègia de paral·lelització no sigui gens bona i es disminueixi a mesura que incrementem els threads. I és d’on ens surt el baix percentatge d'eficiència de la taula anterior. En la segona taula podem veure que hi ha una gran quantitat de tasques creades, i que el temps que s’inverteix en cadascuna no és massa gran, seria un bon codi si el balanceig de les tasques sigues millor, proper al 100%.

Paraver Views:

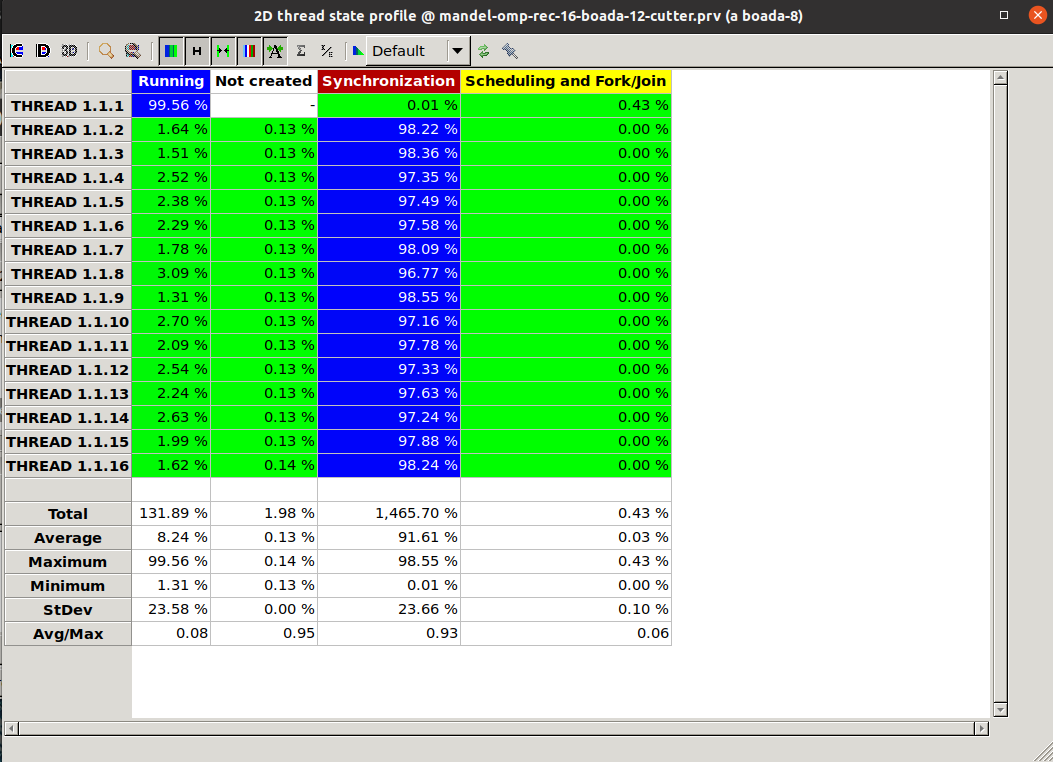


Aquí podem apreciar el desbalanceig i la mala eficiència de l'estratègia de paral·lelització, on veiem que treballen un o pocs nodes alhora, per tant, a més que tardem més a acabar-hi, veiem que tenim un problema amb la paral·lelització.



En aquests gràfics podem veure com un threads és l'encarregat de crear totes les tasques, i després en l'execució es divideix entre els diferents nodes, fent tots una quantitat de tasques similars.

Podem veure també altres coses interessants com:

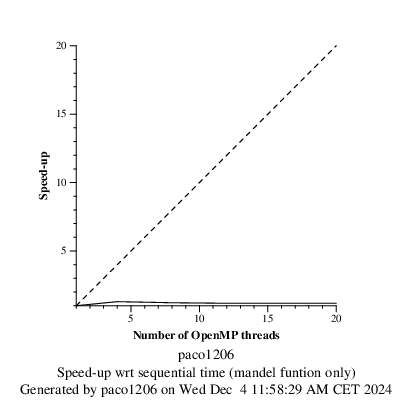


La primera columna “Running” ens diu el percentatge de temps que cada thread ha estat executant una tasca. Veiem com el primer node és el que més temps, per molta diferència amb els altres, implica en les tasques, això segurament es deu a què és l’encarregat de la creació de totes les tasques.

La columna “Synchronization” ens diu el percentatge de temps que el thread ha dedicat a esperar que acabin altres tasques. El percentatge del primer node es casi nul, ja que no ha d’esperar que acabin les tasques els altres, perquè ell tarda més i són els altres nodes els que ho han d’anar esperant i, per tant, aquests tenen un percentatge en aquesta columna molt elevat.

L’última columna ens indica el temps invertit en la gestió de tasques. i això només l’implica al primer node.

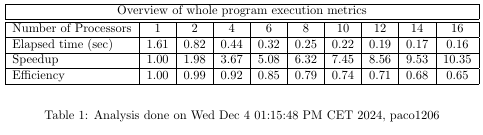
Strong Scalability:



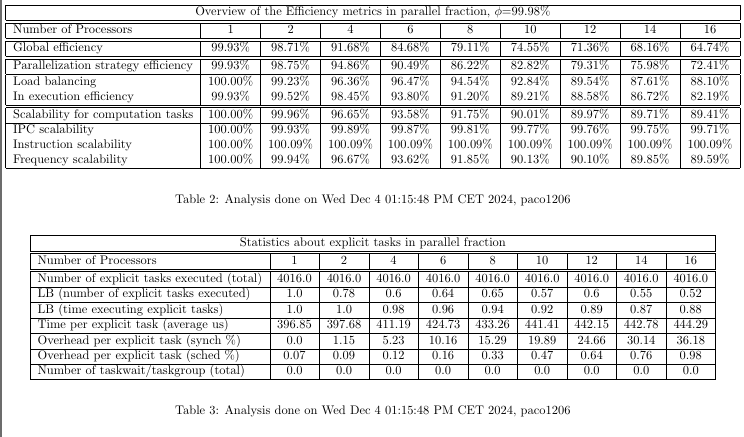
En aquest gràfic observem el que hem parlat amb la primera taula, un petit increment del speed-up de tenir un threads a tenir-ne dos i a partir de tenir 4 threads tornem a baixar cap a un speed-up de 1, estant molt lluny de l'ideal.

Recursive: Tree

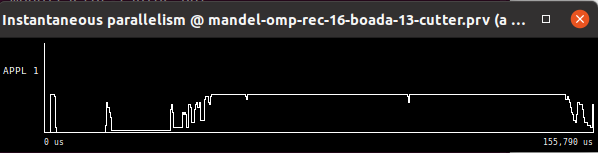
Modelfactor tables



Amb la implementació tree, el speed-up ha millorat bastant, i els temps també. El motiu es podrà veure en els gràfics de paraver, on es veurà com hem millorat la distribució de càrrega entre altres coses.



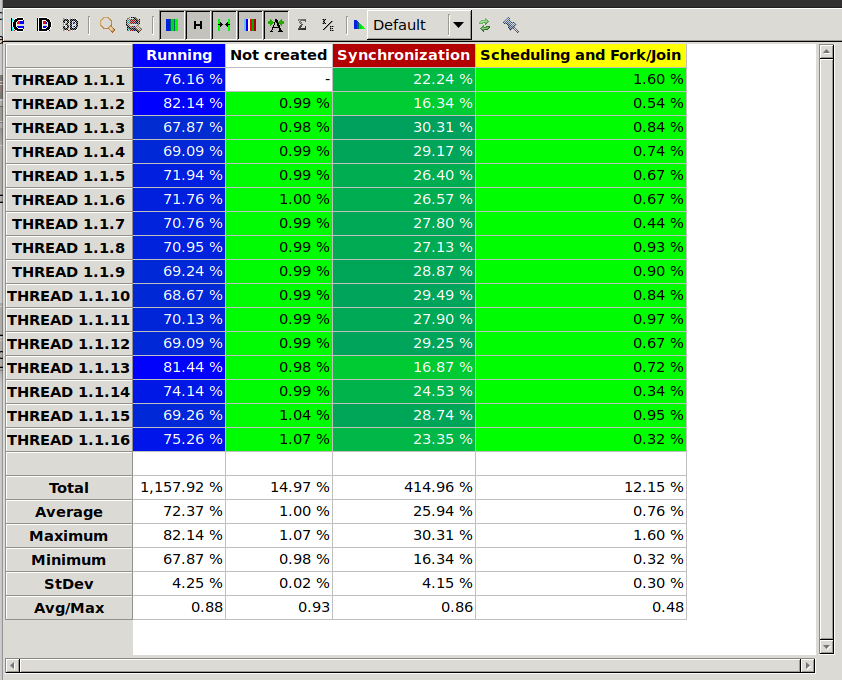
En aquestes dues taules, també es pot veure com hem millorat la distribució de càrrega, encara que podria ser millor. També es creen més tasques, ja que amb la implementació tree anem creant tasques dins de tasques fins a arribar a les fulles.

Paraver views:

Aquest gràfic mostra que el paral·lelisme no es manté constant al llarg de tota l'execució. Aquest comportament és lògic, ja que al començament només es crea una tasca per node. A mesura que avança l'execució, el codi recursiu genera noves tasques de manera progressiva. Aquestes tasques, al seu torn, creen altres tasques, incrementant així el grau de paral·lelisme amb el pas del temps.



Amb aquests gràfics, podem veure que tots els nodes creen tasques i les exectuen, millorant l'eficiència i el paral·lelisme. Tots els nodes executen les tasques encara que hi ha algunes que costen més treball que altres, és a dir, la càrrega de treball ha millorat considerablement, però podria ser millor.



Aquesta taula ens mostra informació sobre diferents aspectes.

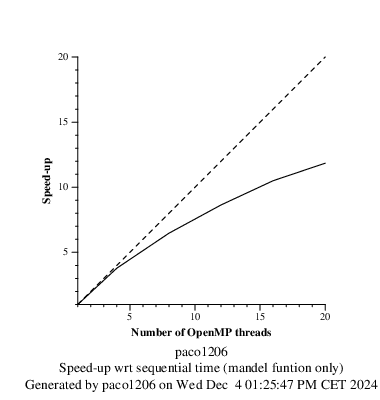
La primera columna "Running" ens diu el percentatge de temps que cada thread ha estat executant una tasca. És prou equitatiu que ens diu que el paral·lelisme és correcte, però encara hi ha threads que executen tasques més temps com el thread 2, o threads que fan menys tasques com el tercer. Això vol dir que el paral·lelisme encara es podria millorar.

La columna "Synchronization" ens diu el percentatge de temps que el thread ha dedicat a esperar que acabin altres tasques. El percentatge és prou alt considerant que intentem resoldre les dependències per evitar aquest temps. Una possible millora és la d'utilitzar 2 equals en comptes d'un al principi de la funció per evitar aquesta dependència.

L'última columna ens indica el temps invertit en la gestió de tasques.

Tots els threads tenen temps molt baixos i molt semblants que indica que es gestionen els fils de forma eficaç i de forma equitativa.

Strong Sacalability:



Aquest gràfic ens indica que el speed-up augmenta a mesura que creix el nombre de fils utilitzats. Tot i això, a partir d'un cert nombre de fils, el speed-up deixa de millorar tan significativament com al principi. Per exemple, passar d'utilitzar 1 fil a 4 provoca una millora considerable en el speed-up, però, a partir de 8 fils, la millora ja no és tan notable, i aquesta tendència es redueix encara més amb 16 fils. Això ens porta a suposar que, en algun moment, no s'assolirà cap millora, i aquest nombre de fils serà el màxim amb el qual es pot aconseguir el millor nivell de paral·lelisme.