PACO Laboratory Assignment

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

Pau Alcázar Perdomo, Francesco Oncins Spedo, paco1101

06/11/2023



**1**

**Laboratory 3 notebook**

You have to add comments and observations following the questions that you have found along the docu- ment indicating **For the deliverable:** in each section (in this laboratory we have included those **For the deliverable** questions in the notebook with setences in gray under each **Comments/Observations**). We have also included a cross-reference to the practice document. **The length of the document is expected to be approximately 8-10 pages (including figures and tables) with a font size of at least 11pt**. Only PDF format for this document will be accepted. Deliverable assignment will be opened in *Atenea* and set the appropriate dates for the delivery. You also have to deliver the complete C source codes for Tareador instrumentation and all the OpenMP parallelisation strategies that you have done. Your lab professor should be able to re-execute the parallel codes based on the files you deliver. DON’T include code in the document except for the fragment modified with respect to the original one: just provide a code except. You will have to **deliver TWO files**, one with the document in PDF format and one compressed file (tgz, .gz or .zip) with the requested C source codes.

As you know, this course contributes to the **transversal competence ”Tercera llengua”**. Deliver your material in English if you want this competence to be evaluated. Please refer to the ”Rubrics for the third language competence evaluation” document to know the *R*ubric that will be used.

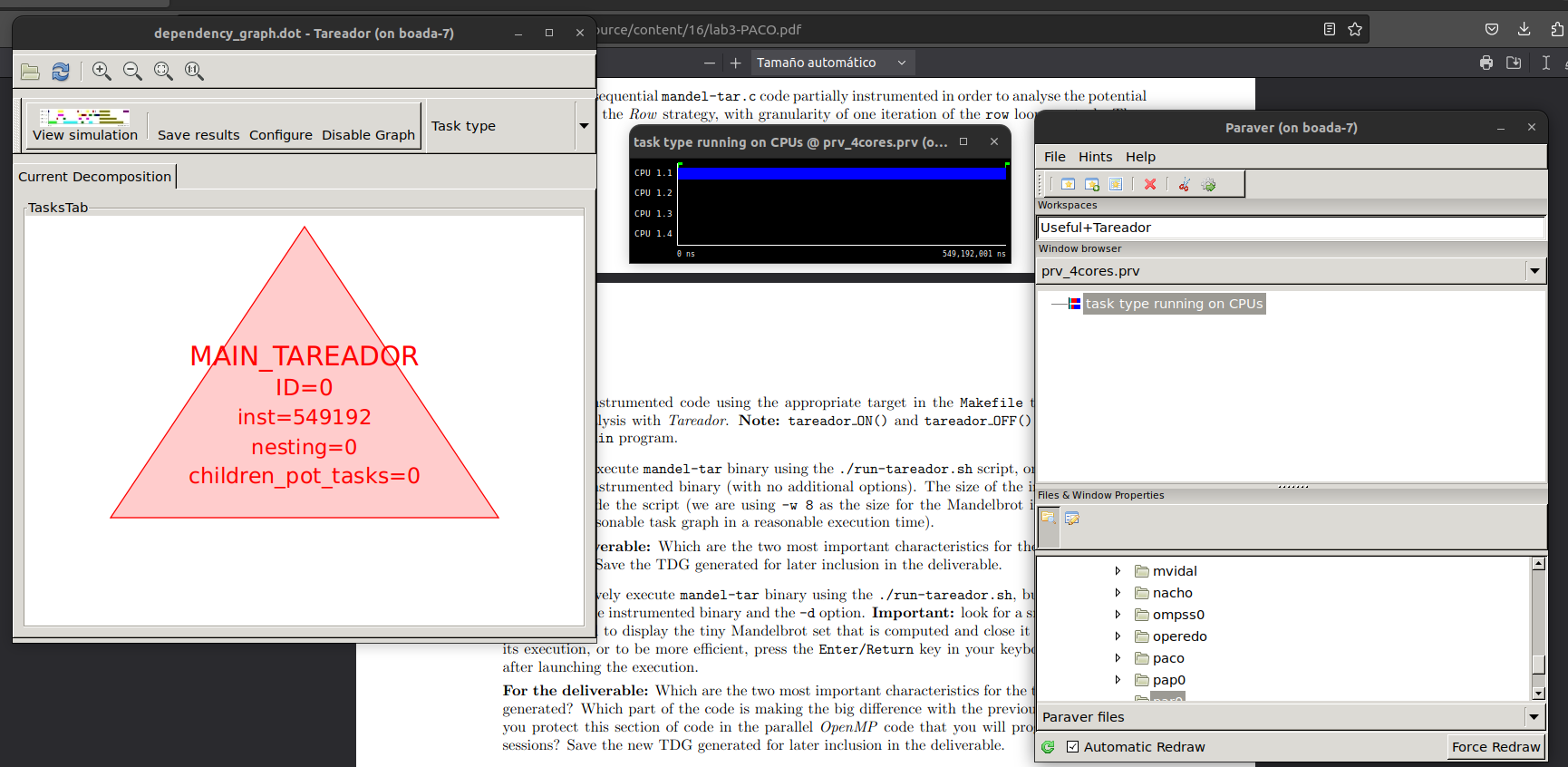
# Task decomposition analysis with *Tareador*

* + 1. **First analysis with *Tareador***

This part refers to point 2 of subsection 2.2 of the practice document.

For the deliverable: Which are the two most important characteristics for the task graph that

is generated? Save the TDG generated for later inclusion in the deliverable.



Granularitat de la tasca: Això fa referència a la mida de les tasques que s'han creat i com es relacionen amb el càlcul global. És crucial perquè pot afectar significativament l'eficiència de l'execució paral·lela. Les tasques massa fines poden portar a un sobrecost a causa de la creació i gestió de tasques, mentre que les tasques massa gruixudes podrien no utilitzar tot el paral·lelisme disponible.

Dependències de la tasca: Aquestes són les vores del TDG que indiquen quines tasques han d'acabar abans que altres puguin començar. Entendre les dependències és fonamental per a la paral·lelització perquè determina quines tasques es poden executar en paral·lel i quines s'han de serialitzar. Les dependències poden afectar l'equilibri de càrregues i el potencial per escalar el càlcul a través de múltiples processadors.

En la teva anàlisi, voldries mirar el TDG i identificar aquestes característiques. La forma del gràfic, la densitat de les vores i la distribució de les tasques poden donar-vos una idea dels possibles colls d'ampolla i oportunitats per optimitzar l'execució paral·lela.

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?(és la mateixa pregunta)

* + 1. **Second analysis with *Tareador***

This part refers to point 3 of subsection 2.2 of the practice document.

For the deliverable: 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 that you will program in the next

sessions? Save the new TDG generated for later inclusion in the deliverable.

Camí crític: Aquest és el camí més llarg a través del gràfic de tasques, el qual determina el temps mínim possible per a completar tot el conjunt de tasques. El camí crític ajuda a identificar quines tasques o seqüències de tasques estan limitant el rendiment paral·lel global.

Dependències i concurrència de tasques: És essencial identificar les dependències entre tasques perquè dicten l'ordre en què s'han d'executar les tasques i defineixen oportunitats de paral·lelisme. Un gràfic de tasques amb moltes tasques independents suggereix un major potencial per a l'execució paral·lela, mentre que un amb moltes dependències pot indicar paral·lelisme limitat.

Pel que fa a la protecció de seccions de codi en codi OpenMP paral·lel:

Seccions crítiques: Si el gràfic de tasques indica recursos compartits o dependències de dades, aquestes seccions necessitaran protecció en un codi OpenMP paral·lel per evitar condicions de carrera. OpenMP proporciona la directiva crítica per assegurar que només un fil alhora executa una secció particular de codi.

Operacions atòmiques: Per a actualitzacions simples de variables compartides, podeu utilitzar la directiva atòmica, que de vegades pot oferir un millor rendiment que una secció crítica, reduint l'excés.

Variables Privades: Per evitar les carreres de dades, les variables que s'utilitzen dins de seccions paral·leles es poden declarar com a privades, de manera que cada fil té la seva pròpia còpia, eliminant la necessitat d'exclusió mútua.

Operacions de reducció: En realitzar reduccions (com resums o trobar un valor mínim/màxim entre tasques), OpenMP té una clàusula de reducció que combina eficientment els resultats de tots els fils.

Pel que fa a la secció de codi que està marcant una gran diferència respecte al cas anterior, sense més informació, és difícil dir-ho definitivament. Normalment, els canvis en el gràfic de tasques poden ocórrer a causa de modificacions en com es generen o sincronitzen les tasques. Per exemple, introduir tasques de gra més fi, alterar l'estructura dels bucles o canviar com es comparteixen o es parteixen les dades pot tenir un impacte significatiu.

Include the TDG of the second *Tareador* analisis (with -d option).

## Comments/Observations

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?

Granularitat de la tasca: es refereix a la mida de les tasques individuals dins del gràfic. La granularitat de la tasca és crucial perquè influeix en la despesa general de la gestió de tasques enfront del treball realitzat per cada tasca. Si les tasques són massa fines, la despesa en la gestió d'aquestes tasques pot superar els beneficis del paral·lelisme. Per contra, si són massa gruixuts, pot haver-hi un paral·lelisme insuficient per utilitzar eficaçment els recursos del sistema.

Dependències de tasca: Les dependències entre tasques indiquen quines tasques s'han de completar abans que altres puguin començar. Això impacta directament en el potencial paral·lelisme del programa. Un gran nombre de dependències poden suggerir oportunitats limitades per a l'execució paral·lela, mentre que menys dependències poden indicar que més parts del programa es poden executar en paral·lel.

Pel que fa a la part del codi que està marcant una gran diferència en el gràfic de tasques del cas anterior, normalment es pot identificar per canvis en els patrons de dependència de tasques o en la granularitat de les tasques. En el codi de Mandelbrot que heu proporcionat, la part significativa probablement seria el doble per al bucle en la funció de mandelbrot. Aquí és on es realitzen els càlculs de píxels, i és l'àrea principal on es produiria la descomposició de tasques.

En el codi OpenMP paral·lel, per protegir seccions de codi que no s'han d'executar per múltiples fils simultàniament (com ara actualitzar variables compartides o realitzar operacions d'E/S), podeu utilitzar:

Seccions crítiques: La directiva .pragma omp critical garanteix que una secció de codi s'executa només amb un fil alhora.

Operacions atòmiques: La directiva atòmica .pragma omp es pot utilitzar per protegir l'actualització d'una variable compartida assegurant-se que una ubicació de memòria específica s'actualitza de manera atòmica.

Variables Privades: Podeu declarar les variables com a privades en una regió paral·lela per a donar a cada fil la seva pròpia còpia, evitant la necessitat d'exclusió mútua per a aquestes variables.

Clàusula de reducció: Quan necessiteu realitzar una operació de reducció entre tasques (com resumir valors), la clàusula de reducció en OpenMP es pot utilitzar per combinar els valors de cada fil d'una manera segura.

Per protegir adequadament les seccions pertinents a l'hora de convertir el codi seqüencial a codi OpenMP paral·lel, analitzareu el gràfic de tasques per entendre on pot ocórrer l'accés concurrent a recursos compartits i després aplicareu les construccions OpenMP apropiades per gestionar aquesta concurrència.

* + 1. **Third analysis with *Tareador***

This part refers to point 4 of subsection 2.2 of the practice document.

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?

Què representa cada cadena de tasques en el gràfic de tasques?

Cada cadena de tasques en un TDG representa una seqüència de càlculs on cada tasca pot dependre de la compleció de la tasca anterior en la cadena. Aquestes cadenes il·lustren les dependències i el flux d'execució a través de diferents parts del programa.

Quina part del codi marca la gran diferència amb els dos casos anteriors?

En el context del codi de Mandelbrot, els bucles niats que realitzen el càlcul del conjunt de Mandelbrot són probablement responsables de les principals diferències observades en el TDG. Els canvis en com s'estructuren o executen aquests bucles alterarien significativament el gràfic de tasques.

Com protegiràs aquesta secció de codi en el codi paral·lel OpenMP que programaràs en les properes sessions?

Per protegir les seccions de codi que són crítiques o impliquen recursos compartits a l'hora de paral·lelitzar amb OpenMP, es poden utilitzar diverses estratègies com seccions crítiques ( criticalpragma omp critical), operacions atòmiques (.pragma omp atomic), i variables privades.

For the deliverable: Which is the main change that you observe with respect to the Row strategy?

El principal canvi en canviar de l'estratègia Row a l'estratègia Point és l'augment en el nombre de tasques a causa d'una granularitat més fina i un canvi potencial en el patró de dependència, resultant en un TDG més complex amb un major paral·lelisme potencial, però també possiblement més sobrecàrrega en la gestió d'un major nombre de tasques.

Include the TDG of the third *Tareador* analysis (with -h option).

## Comments/Observations

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?

En un Task Dependency Graph (TDG), cada cadena de tasques representa una seqüència de càlculs dependents. Les dependències dicten que certs càlculs (tasques) s'han de completar abans que puguin començar els càlculs posteriors. La longitud de cada cadena i les dependències dins i entre cadenes proporcionen informació sobre el potencial paral·lelisme del programa.

En el context del càlcul de conjunts de Mandelbrot, la part del codi que probablement causa una diferència significativa en el gràfic de tasques entre casos serien les estructures de bucles que calculen el conjunt. Si el cas anterior implicava calcular cada fila de la imatge de sortida com una tasca (estratègia de la fila), i el cas actual implica calcular cada punt com una tasca (estratègia de punt), llavors el bucle més interior, que irradia sobre columnes (punts), és on està la diferència.

Per protegir aquesta secció de codi en una implementació paral·lela d'OpenMP, utilitzareu construccions OpenMP per assegurar que les tasques estan correctament sincronitzades i que les dependències de dades són respectades. Si les tasques escriuen a estructures de dades compartides, és possible que hàgiu d'utilitzar mecanismes com ara:

- pragma omp critical or #pragma omp atomic to per a protegir les actualitzacions de les variables compartides,

- private or firstprivate per donar a cada tasca la seva pròpia còpia de variables per evitar condicions de carrera,

- #pragma omp barrier per a sincronitzar tasques en determinats punts,

- #pragma omp taskwait per a fer que una tasca esperi per a completar les seves tasques filles.

Aquestes construccions ajuden a gestionar l'accés concurrent als recursos compartits, assegurant que el programa s'executi correctament en un entorn paral·lel.

* + 1. **Point Strategy analysis with *Tareador***

This part refers to the last question of subsection 2.2 of the practice document.

Include the part of mandel-tar.c you have modified to analyse the Point strategy and the TDG of the

*Tareador* analysis.

## Comments/Observations

Which is the main change that you observe with respect to the *Row* strategy? (resposta abans)

* 1. ***Point* decomposition strategy**
     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? 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?

És la velocitat-dalt apropiat?

Velocitat apropiada-dalt és generalment jutjat per comparar el temps d'execució del programa paral·lel amb el temps d'execució de la versió seqüencial. Velocitat-dalt és considerat apropiat si escala raonablement amb el número dels processadors utilitzats. Que és, doblant el número de processadors idealment halve el temps d'execució, encara que això és sovint no aconseguit a causa de overheads i no-parallelizable parts del codi.

És l'escalabilitat apropiada?

L'escalabilitat és considerada apropiada si el programa continua beneficiar des de recursos computacionals addicionals. Si afegint més rosques porta a retorns minvants o fins i tot actuació pitjor, llavors l'escalabilitat pot ser un assumpte. Això podria ser a causa de diversos factors com elevat des de gestor un número excessiu de fi-tasques granulades, contesa per recursos compartits, o porcions significatives del codi que no pot ser parallelized.

Quines rosques estan creant i executant les tasques explícites?

Dins OpenMP, la rosca que entra el #pragma omp el bloc sol és responsable per crear tasques. Les altres rosques en l'equip seran executar tasques. Si acates que u roscar predominantment està creant tasques i no fent molt de la feina d'execució, podria portar per carregar desequilibri des d'aquella rosca no està contribuint al computacional workload mentre es poden sobrecarregar les altres rosques.

És un de les rosques va dedicar a crear, i per tant executant molt menys?

Si l'anàlisi de perfils o traçant les eines indiquen que u roscar principalment està creant assigna i fent poca execució, llavors sí, això podria ser un col·laborador clar per carregar desequilibri. En un ideal execució paral·lela, la feina seria uniformement distribuït entre les rosques per assegurar que tots processadors són amb eficiència utilitzat.

pensar la granularitat de les tasques és apropiada per això parallelization estratègia?

La granularitat de tasques és apropiada si l'elevat de creació de tasca, gerència, i la sincronització no eclipsa els profits d'execució paral·lela. Si les tasques són també fi-granulat, l'elevat podria ser massa alt, i si són també tosc-granulat, no podries explotar tot paral·lelisme disponible. La granularitat dreta hauria d'equilibrar l'elevat de gerència de tasca amb l'esforç computacional de cada tasca, maximitzant l'ús de recursos computacionals disponibles sense causar significatiu elevat.

* + 1. **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.

## Comments/Observations

Do you think the granularity of the tasks is appropriate for this parallelization strategy? Is the version with taskloop performing better than the last version based on the use of task? Do you notice a relevant change in the two metrics that contribute to the parallelization strategy efficiency? 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 %. Why there are now task synchronizations in the execution? Where are these task synchronisations happening?

La tasca és avaluada basada damunt tant si l'elevat associat amb gerència de tasca és outweighed per la feina computacional cada tasca actua. Si acates que sincronització de tasca està consumint un percentatge significatiu de temps d'execució, això podria suggerir que les tasques són també fi-granulat.

La versió amb taskloop podria ser actuar més ben si mostra la càrrega millorada equilibrar i va reduir elevat, els quals són profits comuns d'utilitzar taskloop a causa de la seva tasca més eficaç planificar i potencial per millor workload distribució.

Si acates un canvi pertinent en la mètrica allò contribueix a parallelization eficiència d'estratègia, típicament seria plasmat velocitat millorada dins-dalt i va reduir temps d'execució. Si l'eficiència d'execució és quieta baix, podries atribuir aquest a qualsevol equilibratge de càrrega pobre (si la feina no és uniformement distribuït a través de rosques) o a sincronització alta i planificant overheads (si aquestes operacions estan dominant el temps d'execució de tasca).

Sincronitzacions de tasca ara podrien ser presents a causa de la introducció de punts de sincronització nova dins el taskloop versió, potencialment al remat de cadascú taskloop on una barrera implícita assegura tot assigna en el bucle és completat abans que emotiu damunt. Això pot passar si el nogroup la clàusula no és utilitzada, el qual altrament permetria tasques per executar més independentment i potencialment redueix la sincronització elevada.

For the deliverable: Is the speed–up appropriate? Is the scalability appropriate? (resposta)

For the deliverable: Based on the modelfactors tables, performance plots, Paraver timelines and

profiles, do you think the granularity of the tasks is appropriate for this parallelisation strategy? Impor-

tant: Take note of all the relevant information in order to draw some conclusions about the behaviour

of this code version.

Temps d'execució i Velocitat-Dalt: granularitat de tasca Idònia hauria de portar a una disminució en temps d'execució i un augment en velocitat-dalt com més rosques són utilitzades. Si els temps d'execució no redueixen com esperat amb rosques addicionals, o si la velocitat-cap amunt de planells o disminucions, això podria indicar suboptimal granularitat de tasca.

La càrrega que Equilibra: El Paraver les cronologies poden proporcionar idea a com uniformement la feina és distribuïda a través de rosques. Si algunes rosques coherentment tenen més feina que altres, la granularitat de tasca pot necessitar ser ajustada per aconseguir equilibratge de càrrega millor.

Sincronització i Planificant Overheads: Si aquests overheads és un percentatge significatiu del temps d'execució, pot suggerir que les tasques són també fi-granulat. El gol és per tenir la majoria del temps d'execució del programa va gastar en computació productiva força que damunt tasques paral·leles gestores.

Perfil d'Execució de la tasca: Els perfils poden mostrar la distribució de temps d'execució de tasca. Una variància ampla podria suggerir mides de tasca inconsistent, mentre que una distribució estreta podria indicar una tasca més uniforme mida, el qual és sovint desitjable.

Sincronitzacions de tasca: Si hi ha moltes sincronitzacions, examina on i per què ocorren. Les sincronitzacions freqüents podrien ser un signe aquelles tasques són també petit o no bé estructurat.

Durada de tasca: Revisa la durada mitjana de tasques. Si les tasques són completades molt de pressa relatiu a l'elevat de crear i gestor els, això suggeriria que les tasques són també fi-granulat.

* + 1. **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?

Per adreçar tant si l'escalabilitat ha millorat i per valorar granularitat de tasca i overheads, típicament revisaries la mètrica d'actuació abans que i després de fer canvis al parallelization estratègia, com introduir taskloop o ajustant mides de tasca. Aquí és els factors consideraries:

Millorament d'Escalabilitat: Compara la velocitat-dalt i mètrica d'eficiència des del modelfactors taules abans que i després dels vostres canvis. Si l'escalabilitat ha millorat, veuries una relació més lineal entre el número de rosques i la velocitat-dalt, significant que afegint més resultats de rosques dins proporcionalment actuació millor.

Granularitat de tasca: Caire al temps d'execució de tasca mitjà. Si la granularitat és optimitzada, les tasques haurien de ser prou grans per minimitzar overheads mentre quiet prou petit per permetre per a equilibratge de càrrega eficaç. La granularitat és probablement millorat si veus una reducció en la variabilitat de temps d'execució de tasca a través de rosques.

Overheads: Comprova els percentatges elevats van relacionar per assignar sincronització i planificar. Si aquests percentatges han reduït, suggereix que el overheads està esdevenint una part menys significativa del temps d'execució total, el qual és un signe de granularitat millorada.

For the deliverable: Do you notice a relevant change in the two metrics that contribute to

the parallelisation strategy efficiency? Should you blame to ”load balancing” or to ”in exe-

cution efficiency”? 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 gener-

ated 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 %. (resposta)

For the deliverable: Include modelfactor tables in the deliverable. Why there are now task

synchronizations in the execution? Where are these task synchronisations happening? Continue

with the following section to be able to answer these questions

Raó per a Sincronitzacions de Tasca: sincronitzacions de Tasca són típicament introduït per assegurar que tot assigna ha completat un conjunt segur de les computacions abans del programa va a un punt que depèn en aquelles computacions que són complet. Són un mecanisme per aplicar dependències i assegurar execució de programa correcte.

Ubicació de Sincronitzacions de Tasca: les sincronitzacions sovint ocorren al remat de regions paral·leles, feina-la compartició construeix, o a punts de sincronització explícita com taskwait o taskgroup. En el cas de taskloop, una sincronització implícita pot ocórrer al remat del bucle llevat que el nogroup la clàusula és utilitzada, el qual permet les tasques per ser més independents i potencialment pot millorar actuació per reduir la sincronització elevada.

For the deliverable: Do you think these task barriers are necessary? If your answer was negative,

you can add the nogroup clause to the taskloop clause in order to eliminate the implicit taskgroup. Edit

the code and submit again the submit-strong-extrae.sh script. Has the scalability improved? What

about task granularity and overheads?

Necessitat de Barreres de Tasca: barreres de Tasca són essencials quan les tasques tenen interdependencies que requereix sincronització per mantenir la correcció del programa. Si les tasques són independents de cadascú altre i hi ha #cap dependències de dada allò necessita sincronització, llavors aquestes barreres poden introduir innecessàries elevat i podria ser tret a potencialment millorar actuació.

Utilitzant el nogroup Clàusula: Si determines que barreres de tasca no són necessàries, pots utilitzar el nogroup clàusula amb el taskloop construir per impedir la creació d'un implícit taskgroup. Això pot reduir la sincronització elevada des de les rosques no esperaran al remat del taskloop per tot assigna per completar.

Avaluant Escalabilitat: Després de modificar el codi per incloure el nogroup clàusula, estrenaries els submit-strong-extrae.sh dicten i analitza els resultats. L'escalabilitat millorada seria indicada per una velocitat més significativa-dalt amb l'augment en el número de rosques, mostrant utilització millor dels recursos computacionals.

Valorant Granularitat de Tasca i Overheads: Amb el nogroup clàusula, també examinaries la granularitat de tasca — tant si les tasques són ara massa gran o petit — i com que afecta overheads. Idealment, veuries una disminució dins overheads relacionat per assignar gerència, suggerint que la granularitat ha assolit un nivell més òptim.

# 1.3 *Row* decomposition strategy

This part refers to subsection 3.2 of the practice document.

Include here the *modelfactors* tables generated by the submit-strong-extrae.sh script.

## Comments/Observations

Which are the main differences that you observe when comparing the *Row* and *Point* strategies? What is the number of tasks created for *Row* and *Point* strategies? Is there any load unbalance? What is the task synchronisation cost now?

Diferències majors Entre Fila i Estratègies de Punt:

Granularitat: L'estratègia de Punt té granularitat més fina, creant una tasca per a cada punt (píxel), mentre l'estratègia de Fila té granularitat més tosca, amb u assigna per fila de punts.

La tasca Elevada: A causa de granularitat més fina, l'estratègia de Punt probablement incorre més elevat des de crear i gestor un número més gran de tasques.

La càrrega que Equilibra: L'estratègia de Fila podria oferir equilibratge de càrrega millor si cada fila duu aproximadament el temps mateix per computar. L'estratègia de Punt podria portar per carregar desequilibri si els punts varien significativament en temps de computació.

Localitat de dada: L'estratègia de Fila podria beneficiar des de localitat de dades millors punts tan consecutius en una fila són probables de ser processat per la mateixa rosca, mentre l'estratègia de Punt no pot utilitzar memòria cau tan amb eficiència.

Número de les tasques Creades:

Per a l'estratègia de Fila, el número de tasques és igual al número de files.

Per a l'estratègia de Punt, el número de tasques és igual al número total de punts, el qual és el producte del número de files i columnes.

La càrrega Desquadrar:

L'estratègia de Fila pot tenir menys desequilibri de càrrega si cada fila és similar en termes de computació.

L'estratègia de Punt podria tenir més desequilibri de càrrega, especialment prop de les vores del Mandelbrot conjunt on el número d'iteracions pot variar significativament des de punt a punt.

Sincronització de tasca va Costar:

El cost de sincronització per a l'estratègia de Fila podria ser més baix de llavors ençà hi ha menys tasques i, per això, potencialment menys punts de sincronització.

L'estratègia de Punt podria tenir un cost de sincronització més alt a causa d'un número més gran de tasques, tot i que això també dependrà damunt que les tasques són planificades i la presència de qualssevol dependències de tasca.

Per fer observacions concretes sobre les diferències en balanç de càrrega, recompte de tasca, i costos de sincronització, necessitaries analitzar les traces d'execució des d'eines com Paraver, revisar dades d'anàlisi de perfils, i compara la mètrica d'actuació abans que i després de l'aplicació de cada estratègia. Les troballes serien concretes a com el Mandelbrot va posar la computació és implementada i podria variar basat en factors com la distribució del conjunt a través del pla complex i les optimitzacions particulars van aplicar al parallelization estratègies.

For the deliverable: Support your reasoning and draw the attention to the main differences that

you observe when comparing the Row and Point granularities. Analyse the number of tasks created

vs. the number of tasks executed and their granularities. Is there any load unbalance? Any task

synchronisation to eliminate?

Fila i granularitats de Punt:

Granularitat de Tasca:

Granularitat de Punt: Típicament crea un número gran de tasques petites. Això pot portar a alt elevat a causa de creació de tasca i gerència.

Granularitat de fila: Genera menys, les tasques més grans correspondre a cada fila. Això sovint resulta dins més baix elevat però podria potencialment underutilize el paral·lelisme disponible si no tot els nuclis són seguits ocupats.

Número de les tasques Creades en contra. Executat:

Amb Granularitat de Punt: Espera per veure una tasca per a cada punt en l'espai de computació. s'haurien d'executar totes aquestes tasques, donat recursos paral·lels suficients.

Amb Granularitat de Fila: El número de tasques és igual al número de files. Si les tasques són massa grans, hi podria haver menys tasques que rosques disponibles, punter a alguns nuclis que són idle.

La càrrega que Equilibra:

Estratègia de Punt: Pot sofrir des de desequilibri de càrrega si el temps de computació varia significativament a través del dataset. Algunes rosques poden acabar les seves tasques més ràpides que altres.

Estratègia de fila: Tendeix per ser més equilibrat si cada fila té una càrrega computacional similar. Tanmateix, variacions en computació a través de parts diferents del Mandelbrot va posar encara pot causar desequilibri.

Sincronització de tasca:

Examina tant si punts de sincronització són necessaris. Si hi ha cap dependència de dades entre tasques, es podria reduir la sincronització.

Per a Granularitat de Punt: hi pot haver més punts de sincronització freqüents a causa del número més alt de tasques.

Per a Granularitat de Fila: la sincronització podria ser menys freqüent però podria causar demores més llargues si moltes rosques estan esperant per uns quants per acabar.