**4. Паралелизација**

**Зашто паралелизовати?**

Оптимизациони проблеми су у великом броју случаја комплексни ( зато су и алгоритми за њихово решавање толико битни ), и захтевају обраду огромног броја података. Пошто је *PSO* по природи алгоритам који уме да прерано конвергира према локалном оптимуму, јако нам је битно ојачати ефикасност са разним могућим стратегијама, урачунајући и паралелизацију. Сви алгоритми базирани на некаквој популацији имају бројне делове, који су суштински паралелни. Пошто се у *PSO* алгоритму све честице крећу независно од осталих честица у роју, и једино зависе од дотад најбољег глобалног решења и *PSO* је природно паралелан алгоритам. Компоненте које могу да раде задатке у паралели су процесор и графичка картица. Паралелизација је процес поделе задатака на делове који могу да иду у паралели, притом да се програм очекивано понаша приликом сваког покретања. Општи проблеми паралелизације су оперативни систем, синхронизација задатака, програмски језици, функције и библиотеке. Такође постоје такозвани *cloud computing* сервиси који нам пружају велики број огромних сервера са најновијим компонента, за што бољу експлоатацију паралелног кода. У наредним деловима следи опис неких најпознатијих стратегија за *PPSO* ( паралелни *PSO*) алгоритам.

**4.1. Паралелизационе стратегије**

Када се паралелизује неки алгоритам, треба имати у обзир и циљану компоненту која ће њу извршити. Није свеједно да ли ће се извршити на једном или више рачунара, а и компонента може да буде процесор са више језгара, или чак и графичка картица са више хиљада језгара. Следи опис имплементација по циљаној компоненти.

**Базиране на процесору**

Користимо процесоре са више физичких и виртуелних језгара. Овде имамо избор да користимо један од следећих имплементација:

***Hadoop MapReduce***

Овај модел је развијен од компаније *Google* за обраду огромног броја података Имплементације нам скоро све сама уради (подела података, паралелизација, опоравак од грешака итд.), на нама је само неке основне операције да дефинишемо. Захтева се и један пар кључа и бредности коју ће библиотека да нам обради.

***MATLAB* библиотеке за паралелизацију**

*MATLAB* нам нуди један од најлакших начина за паралелизацију. Јако је лак за коришћење, али није бесплатан софтвер. На нама је само да дефинишемо паралелну *for* петљу.

***R Parallel package***

*R* је бесплатан софтвер, који је дизајниран за статистичка рачунања. Постоје библиотеке као *foreach* и doParallel, које нам омогућују паралелизацију и у овом језику.

***Julia: Parallel for and MapReduce***

*Julia* је модеран и функционалан језик изграђен баш за сврху јаке паралелизације. Овде имамо *@parallel for* и *MapReduce* као опције.

***Python* библиотеке**

*Python* је флексибилан интерпретирани језик. Постоје разне библиотеке за паралелизацију као на пример *Joblib.*

***OpenMP with C++***

*С++* је компајлирани језик познат по својој брзини и робустности. *OpenMP* је једна од најлакших библиотека које можемо да користимо за наше сврхе у овом језику.

***MPI***

Служи за паралелизацију у системима са више рачунара.

**Базиране на графичкој картици**

Прошлих година паралелизација коришћењем графичке картице постаје све популарнија. Она може да има више хиљада језгара. Све претходно поменуте имплементације се могу и овде користити, али постоје и имплементације које су направљене само за графичке картице, од којих су најпопуларнији:

***CUDA***

Овај модел је изградила компанија *nVIDIA,* за коришћење на њиховим компатибилним картицама. Корисник мора да дефинише функције које ће картица да изврши, затим да алоцира меморију за променљиве.

***OpenACC***

Има архитектуру *OpenMP,* код се гради тако што се серијском коду додају неке кључне речи. Код може да се пребаци и на процесор, чак и на њихову мешавину.

**4.2. Конвенционални *PPSO* алгоритми**

Када се паралелизује неки алгоритам, један од највећих проблема нам представља синхронизација то јест комуникација међу задацима. У овом случају задаци су нам под-ројеви или саме честице. Четири најкоришћенијих алгоритама за комуникацију су: ***Star PPSO***(звезда), ***Migration PPSO*** (миграција познато и под кружни или прстен), ***Diffusion PPSO***, ***Broadcast PPSO***. Следи опис ових алгоритама.

**Звезда**

Овај алгоритам има *master-slave* топологију, што значи да имамо један под-рој или једну честицу која је надређена, и која шаље свим осталим подређеним под-ројевима. Не постоји директна комуникација међу подређеним под-ројевима. Кораци у овом алгоритму су:

*Корак 1*: Надређени одлучи какве параметре за алгоритам ће користити, и подели их са подређенима. Ови параметри су углавном број итерација, тежина инерције, период комуникације, величина популације и коефицијенти убрзања.

*Корак 2*: Изврши се померанје роја, сваки под-рој паралелно мења или добија своју досад најбољу вредност, и глобалну најболју вредност.

*Корак 3*: Кад је свако извршио своје померање, подређени шаљу надређеном своју досад најбољу вредност.

*Корак 4*: Надређени из свих досад примлљених вредности бира најбољу.

*Корак 5*: Свима се промени коефицијент убрзања и позиција.

*Корак 6*: Опет се шаљу информације о персоналним оптимума надређеном и опет се мења глобални оптимум.

*Корак 7*: Ово понављамо док није пронађен оптимум, или по неком другом критеријуму терминације.

**Миграција**

У овом алгоритму су под-ројеви повезани у једном кругу, и под-ројеви могу само да комуницирају са комшијама. Један под-рој може да комуницира само са под-ројем који је са његове леве или десне стране. Кораци овог алгоритма су:

*Корак 1*: Сви параметри су познати одмах на почетку.

*Корак 2*: Сви подројеви се померају у паралели, и паралелно дођу до својих персоналних и глобалних оптимума.

*Корак 3*: Најбоља честица се замени са најгором честицом у комшијском под-роју. Током сваке комуникације међу под-ројевима се и глобални оптимум измењује.

*Корак 4*: Под-ројевима се измени позиција и коефицијент убрзанја са новим персоналним и глобалним оптимумима.

*Корак 5*: Понављамо корак 3.

*Корак 6*: Ове кораке понављамо док не буде задовољен критеријум заустављања.

***Broadcast***

Принцип овог алгоритма је да сваки под-рој може да комуницира са сваким другим под-ројем. Сви под-ројеви се извршавају у паралели, и све информације шаљу свим осталим под-ројевима. Први и други корак овог алгоритма су идентични миграционом, а остали су:

*Корак 3*: Сви под-ројеви шаљу своју дотад најбољу позицију да би се сазнало која је сада најбоља позиција.

*Корак 4*: После измене оптимума, под-ројеви ажурирају своје позиције и убрзања.

*Корак 5*: Понављамо корак 3.

*Корак 6*: Ове кораке понављамо док не буде задовољен критеријум заустављања.

***Diffusion***

Овај алгоритам је јако сличан миграционом алгоритму, са разликом да сада сваки под-рој има 4 комшије. Под-ројеви има левог и десног комшију, али и горњег и доњег, па су под-ројеви распоређени налик неке матрице, са разликом да су под-ројеви на угловима, где не би имали 4 комшије избачени. Кораци који треба да се имплементирају су исти као кораци из миграционог алгоритма.