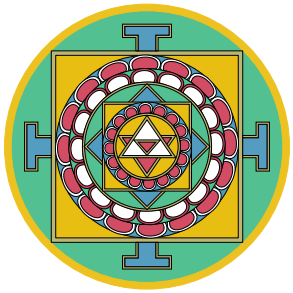
Optimisation of the first principle code OCTOPUS for massive parallel architectures: application to light harvesting complexes

Thesis Dissertation

April 23, 2015

Joseba Alberdi-Rodriguez
(joseba.alberdi@ehu.eus)



Supervisors:

Dr. Javier Muguerza Rivero
Department of Computer Architecture and Technology
Faculty of Computer Science

Prof. Angel Rubio Secades
Department of Materials Physics
Faculty of Chemistry

Universidad del País Vasco — Euskal Herriko Unibertsitatea

Servicio Editorial de la Universidad del País Vasco (UPV/EHU)
- *Euskal Herriko Unibertsitateko (UPV/EHU) Argitalpen Zerbitzua*
- University of the Basque Country (UPV/EHU) Press
- ISBN: 978-84-9082-371-2

Knows no end
Jakintzak ez duelako inoiz mugarik

Familiari

Abstract

Computer simulation has become a powerful technique for assisting scientists in developing novel insights into the basic phenomena underlying a wide variety of complex physical systems. The work reported in this thesis is concerned with the use of massively parallel computers to simulate the fundamental features at the electronic structure level that control the initial stages of harvesting and transfer of solar energy in green plants which initiate the photosynthetic process.

Currently available supercomputer facilities offer the possibility of using hundred of thousands of computing cores. However, obtaining a linear speed-up from HPC systems is far from trivial. Thus, great efforts must be devoted to understand the nature of the scientific code, the methods of parallel execution, data communication requirements in multi-process calculations, the efficient use of available memory, etc. This thesis deals with all of these themes, with a clear objective in mind: the electronic structure simulation of complete macro-molecular complexes, namely the Light Harvesting Complex II, with the aim of understanding its physical behaviour.

In order to simulate this complex, we have used (with the assistance of the PRACE consortium) some of the most powerful supercomputers in Europe to run Octopus, a scientific software package for Density Functional Theory and Time-Dependent Density Functional Theory calculations. Results obtained with Octopus have been analysed in depth in order to identify the main obstacles to optimal scaling using thousands of cores. Many problems have emerged, mainly the poor performance of the Poisson solver, high memory requirements, the transfer of high quantities of complex data structures among processes, and so on. Finally, all of these problems have been overcome, and the new version reaches a very high performance in massively parallel systems. Tests run efficiently up to 128K processors and thus we have been able to complete the largest TDDFT calculations performed to date. At the conclusion of this work it has been possible to study the Light Harvesting Complex II as originally envisioned.

Laburpena

Konputagailu bidezko simulazioa da, gaur egun, zientzialariek eskura duten tresnarik ahaltsuenetako bat sistema fisiko konplexuen portaera ulertzen saiatzeko. Oinarrizko fenomeno fisiko horiek simulatzeko superkonputagailuak erabili dira tesi honetan aurkezten den lanean. Konkretuki, punta-puntako konputagailuak erabili dira fotosintesiaren lehen urratsak ulertzeko, landare berdeetan eguzki-energiaren xurgatze-prozesua kontrolatzen duen molekula simulatuz.

Superkonputazio-zentroek ehunka milaka prozesatze-nukleo dituzten makinak erabiltzeko aukera eskaintzen dute, baina ez da batere erraza azelerazio-faktore linealak lortzea halako konputagailuetan. Hori dela eta, ahalegin handiak egin behar dira, informatikaren ikuspegitik, sistema osoaren ezagutza ahalik eta sakonena lortzeko: kode zientifikoen izaera, beraren exekuzio paraleloen aukerak, prozesuen arteko datu-transmisioaren beharrak, sistemaren memoriaren erabilera eraginkorrena, eta abar. Tesi honek aurreko arazo guztiei aurre egiten die, helburu argi batekin: konplexu makromolekular osoen simulazioa, konkretuki Light Harvesting Complex II sistemaren egitura elektronikoaren simulazioa, beraren portaera fisikoa ulertu ahal izateko.

Sistema hori simulatu ahal izateko bidean, Europako superkonputagailu azkarrenak erabili dira (PRACE partzuergoari esker) Octopus software-paketea exekutatzeko, zeina Density Functional Theory eta Time-Dependent Density Functional Theory izeneko teorien araberako simulazio elektronikoak egiten baititu. Lortutako emaitzak sakonki analizatu dira, milaka konputazio-nukleo eraginkorki erabiltzea oztopatzen zuten arazoak aurkitzeko. Problema ugari azaldu dira bide horretan, nagusiki Poisson ebazlearen errendimendu baxua, memoria eskaera handiak, datu-egitura konplexuen kopuru handiko transferentziak, eta abar. Azkenean, problema horiek guztiak ebatzi dira, eta bertsio berriak errendimendu handia lortu du superkonputagailu paraleloetan. Exekuzio eraginkorrak frogatu ahal izan ditugu 128K prozesadorera arte eta, ondorioz, inoizko TDDFT simulaziorik handienak egin ahal izan ditugu. Hala, lan honen amaieran, hasierako helburua bete ahal izan da: Light Harvesting Complex II sistema molekularraren azterketa egitea.

Contents

Chapter 0

Laburpena (Summary in Basque)

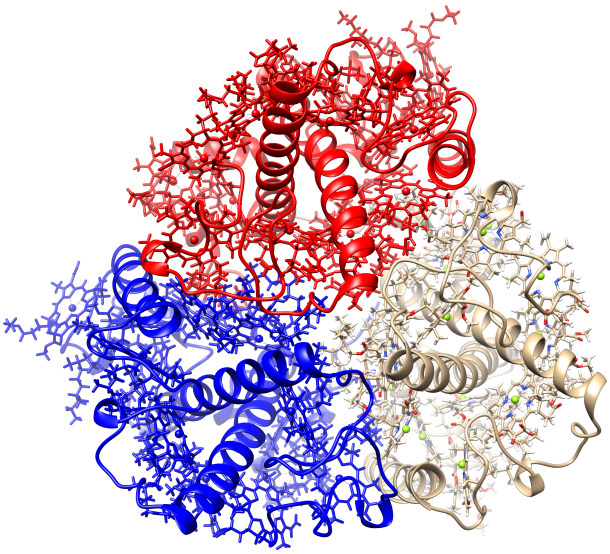
Contents

Lan honek informatikako doktore-tesi bat laburbiltzen du, zeina informatikarien, fisikarien eta kimikarien arteko lankidetzan garatu den. DFT eta TDDFT teoriak gauzatzen dituen Octopus software-paketea erabili da landareetan fotosintesia egiten duen molekularen simulazioak egiteko, Europako superkonputagailu bizkorrenetakoak erabiliz. Helburu hori lortzeko, ordea, konputazioko hainbat arazori aurre egin behar izan zaizkio, azelerazio-faktore egokiak eskuratzeko. Arazo horiek gainditzea lortu da, eta, ondorioz, inoizko TDDFT simulaziorik handienak egin ahal izan dira, milaka prozesadore eraginkorki erabiliz.

0.1 Sarrera eta motibazioa

Informatikaren aurrerapenak medio, zientzialariak eta ingeniariak gauza dira prozesu fisiko oso konplexuen portaera simulazio-tekniken bidez analizatzeko. Izan ere, konputagailu bidezko simulazioak errealitatea aurreikusteko tresna ezinbestekoak dira gaur egungo industria eta ikerketa-laborategietan, problema berrientzat soluzio berritzaileak sortzeko. Eta horretarako prest ditugu superkonputagailuen konputazio-ahalmen itzela, zeinetan ohikoa baita 100.000 prozesadore baino gehiago erabiltzea aplikazio oso konplexuak ebazteko. Zehazki, artikulu honetan fisika kuantikoko simulazioei buruz arituko gara. Teoria jakin batzuetan oinarrituta (aurrerago azalduko ditugun DFT eta TDDFT), fotosintesiaren prozesua aztertu dugu; izan ere, oraindik ez da erabat ulertzen nola xurgatzen duten argia landareek. Hala, lan honen helburua jakintza horretan pauso bat aurrera egitea da, eta, horretarako, Octopus [1, 2, 3, 4] software-paketea erabili dugu.

Landareen fotosintesiaren arduraduna Light Harvesting Complex II (LHC–II) molekula da (ikus 0.1 irudia). Molekula erraldoi horrek hiru aldiz errepikatzen den egitura simetrikoa du eta, orotara, 17.000 atomo baino gehiagoz osatua da. Hiru aldeko simetria horrengatik, trimero deitzen zaio eta bere ataletako bakoitzari, monomero. Monomero bakoitzaren zatirik garrantzitsuenak proteinak, karotenoideak (xantofilak eta klorofilak) eta ur molekulak dira.



Irudia 0.1: LHC–II molekula. Landareetan fotosintesia egiten duen molekula. Monomero bakoitza kolore batez adierazita dago, eta hiru monomeroek trimero bat osatzen dute. Azken helburua mekanika kuantikoa erabiliz molekula erraldoi hau simulatzea da.

LHC–II bezalako molekula handiak simulatu ahal izateko, bai konputagailu konplexuak eta programa aurreratuak behar dira, eta hori dena biltzen duen terminoa High Performance Computing (HPC) da. HPC arloan, azelerazio-faktorea da kontuan izan beharreko neurri garrantzitsuenetariko bat. Oro har, konputazio-baliabideak ugaldu ahala, programen exekuzio-denborak murrizten dira; idealki, prozesadore kopurua bikoiztuta exekuzio-denbora erdira jaitsi beharko litzateke, hau da, programak bi aldiz azkarrago exekutatzea lortu beharko genuke. Programa bat zenbat aldiz azkarrago exekutatzen den zehazten duen neurri horri azelerazio-faktore

edo *speed-up* deritzo, eta azelerazio-faktore ideala 0.2 irudiko marra beltz jarraituan ikus daiteke.

Programa baten exekuzioa hainbat prozesutan banatu daiteke, zeinek programa osoaren zati bana exekutatzen duten. Prozesu horiek aldi berean exekutatzen badira, programa paraleloa dela esaten da. Baina, zoritxarrez, programa guztietan badaude paraleloan exekutatu ezin daitezkeen zatiak, nahitaez seriean (prozesadore bakarrean) exekutatu behar direnak. Serieko zatiek errendimendua mugatzen dute, Amdahl-en legeak adierazten duen moduan [5]. 0.2 irudiko beheko marra berdeak adierazten du programa batek lortuko lukeen errendimendua %90 paraleloan eta %10 seriean exekutatuko balitz, benetan urria. Baina, zorionez, Gustafson-en legeak muga hori gaindigarria dela iragartzen du [6]. Zientzialariak beti daude sistema geroz eta handiagoak simulatzeko gogoz, eta, problemen tamaina hazten dugunean, serieko atala bere horretan mantendu ohi da, paraleloan egin daitekeen lan-karga askoz ere handiagoa izanik. Era horretan, baliabide berriei etekin hobea atera dakieke, 2.1 irudiko tarteko marra gorriak adierazten duen moduan. Problema errealak bi hurbilpen horien artean egon ohi dira.

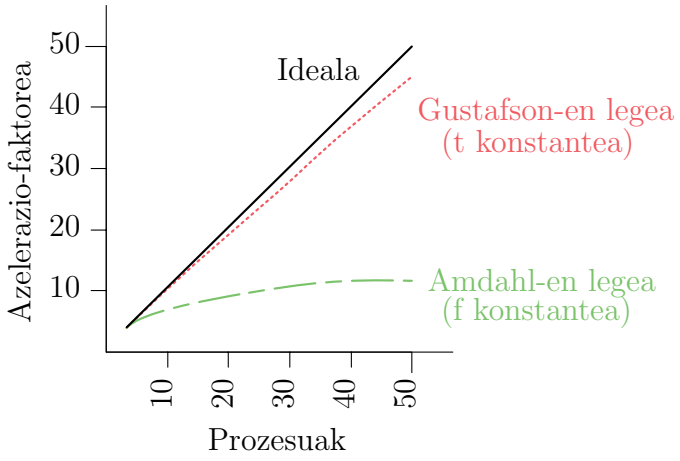
Oro har, beraz, prozesadore asko dituzten konputagailuak problema konplexuak eraginkorki ebazteko erabil daitezke; esaterako, 64.000 prozesadoreko makina bat problema jakin bat 64.000 aldiz azkarrago ebazteko (edo gauza bera dena, prozesadore batek zazpi urtetan egingo lituzkeen kalkuluak ordu bakar batean egitea). Lan honetan munduko superkonputagailu azkarrenetako batzuk erabili ditugu; besteak beste: Alemaniako Juqueen konputagailua (458.752 prozesatze-nukleo edo *core*), Italiako Fermi konputagailua (163.840 nukleo), Alemaniako Hydra (65.320 nukleo), Frantziako Curie konputagailua (11.520 nukleo), eta, Espainian, Bartzelonako MareNostrum konputagailua (48.896 nukleo). Juqueen eta Fermi IBMren Blue Gene/Q arkitekturako superkonputagailuak dira, hau da, propio diseinatutako makinak konputazio-abiadura handienak lortzeko. Hydra, Curie eta MareNostrum, aldiz, cluster motakoak dira, hau da, konputagailu "arrunt" ugari sare bizkor batekin lotutako makinak. Punta-puntako superkonputagailuei buruzko informazio gehigarria E. Strohmaier, J. Dongarra, H. Simon eta M. Meuer-ek urtean bitan argitaratzen duten TOP500[1] zerrendan aurki daiteke, non, LINPACK [7] proba-bankua exekutatzen munduko superkonputagailu azkarrenak ageri baitira.

0.2 Arloko egoera eta ikerketaren helburuak

Octopus software-paketea simulazioak *ab-initio* egiteko erabili da, hots, beste inolako hurbilpenik erabili gabe, teoria fisikoetan bakarrik oinarrituta. Funtsean, Octopus kodean Density Functional Theory (DFT) [8, 9] eta Time-Dependent Density Functional Theory (TDDFT) [10, 11, 12] teoriak erabiltzen dira atomo eta molekulen simulazioak egiteko. Teoria horiek mekanika kuantikoaren birformulazio bat dira, eta bai denborarekiko independente (DFT) nahiz menpeko (TDDFT) izan, dentsitate elektronikoan oinarritzen dira, frogatu ahal izan baita sistema atomiko baten propietate guztiak erator daitezkeela bere dentsitate elektronikoa ebatzita.

Kodean, orokorrean, elektroiak mekanika kuantikoari jarraiki simulatzen dira, eta nukleoa klasikoki tratatzen da. Teoria horiek mekanika kuantikoaren konputazioa ahalbideratu dute, bestela zuzenean ebatzi ezin diren ekuazio konplexuegiak sortzen

[1] http://www.top500.org



Irudia 0.2: Programa paralelo baten aurreikusitako errendimendu teorikoa: Ideala, Amdahl-en legea eta Gustafson-en legea. Zehazki, Amdahl eta Gustafson-en legeen arteko diferentzia ageri da, $f = 0.9$ kasurako (paraleloan exekutatzen den programaren zatia). Argi dago Gustafson-en legearen arabera azelerazio-faktore handiak lor daitezkeela, Amdahl-en aurreikuspen ezkorrarekiko.

baitira, baita gaur egungo superkonputagailu bizkorrenarentzat ere. Bai DFT eta TDDFT metodoak iteratiboak dira, hau da, sistema iterazioz iterazio ebazten da, konbergentzia lortu arte. Iterazio bakoitzean funtzio berak exekutatzen dira, eta bukaeran erabakitzen da simulazioko beste iterazio bati ekitea ala ez. Exekutatu behar den funtzio horietako bat Poisson ebazlea da, zeinak potentzial elektrostatikoa kalkulatzen duen dentsitate elektronikotik abiatuta.

Milaka atomoko sistemekin maila kuantikoan simulazioak egiteko gaitasuna izan ezkero, fisikako, kimikako eta biologiako fenomeno asko ulertzea posible izango litzateke. Nahiz eta azken urteetan HPC arloan egin diren hobekuntzak handiak izan, TDDFTrekin egin daitezkeen simulazioen tamainak mugatua izaten jarraitzen du, eta erronka handia dira oraindik. Tamalez, hainbat arazo aurkitu dira milaka atomo simulatu ahal izateko, eta arazo horiei nola aurre egin diegun azalduko dugu hurrengo atalean.

0.3 Lan-ildoak eta emaitzak

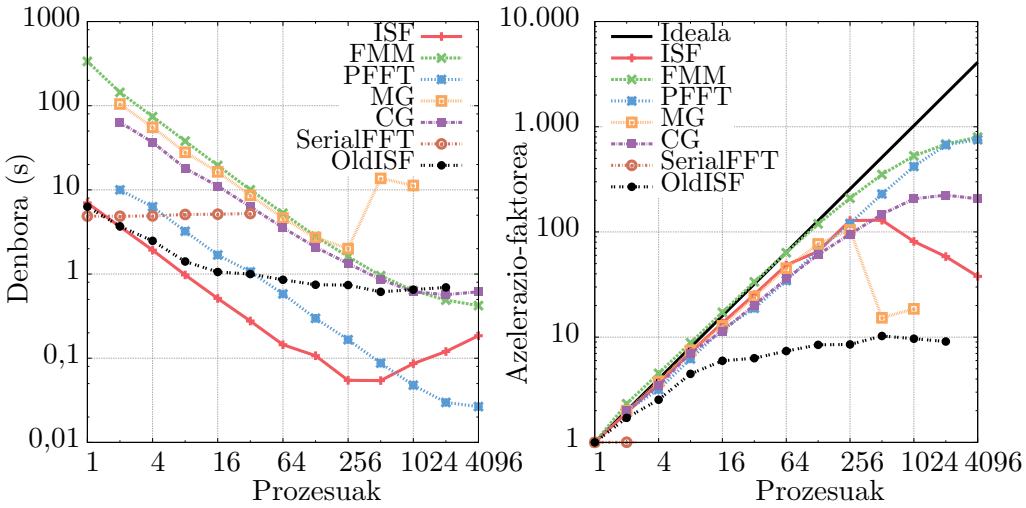
Octopus kodearen errendimendua hobetze aldera, hainbat azterketa egin dira. Hasierako azterketa horien ondorioz, exekuzio paraleloan hainbat hutsune aurkitu ziren simulazioan; aipagarriena, Poisson ebazlearen exekuzio ez optimoa izan zen [13]. Funtzio hori ia seriean exekutatzen zen eta, ondorioz, exekuzio-denborak ez ziren murrizten prozesu kopurua handitu ahala. 0.1 taulan ikus daiteke Poisson funtzioa exekutatzeko behar zen denborak gero eta pisu handiagoa hartzen zuela prozesu kopurua handitzen zenean (0.3 irudiak portaera bera berresten du; bai ezkerrean zein eskuinean “OldISF” marra horizontal bihurtzen da). Horregatik, Poisson ebazleak galarazten zuen errendimendu optimoa lortzea prozesu asko erabiltzen zirenean.

Hori horrela, hainbat aukera aztertu dira Poisson funtzio optimizatuagoa lortzeko. Orotara, zazpi metodo aztertu dira, 0.3 irudian ikus daitekeen bezala. Lau metodo aurrez inplementatuta zeuden: Interpolating Scaling Function zaharra (OldISF), Conjugate Gradients (CG), Multigrid (MG) eta serieko Fast Fourier Trans-

Prozesuak	512	1024	2048	4096	8192
Iterazioaren exekuzio-denbora (s)	44,89	34,03	19,61	11,74	8,81
Poisson funtzioaren exekuzio-denbora (s)	1,14	1,27	2,72	3,21	4,25
Poisson funtzioaren ehunekoa (%)	3	4	14	27	48

Taula 0.1: Poisson funtzioaren exekuzio-denbora, iterazio-denboraren ehuneko gisa, 1365 atomoko sistema baterako Blue Gene/P makinan exekutatuta.

forms (serialFFT); eta 3 metodo berri gauzatu dira Poisson ebazle gisa; aurrez lehenetsia zen ISF metodoaren bertsio berria [14], Parallel Fast Fourier Transforms (PFFT) [15] eta Fast Multipole Method (FMM) [16]. Lortutako errendimenduari dagokionez, FMM metodoak paralelizazio maila ona erakutsi duen arren, exekuzio-denbora altuak behar ditu. PFFT metodoa, aldiz, magnitude-ordena bat azkarragoa da aurrekoa baino, pareko paralelismo maila lortuz. Hots, bi metodo horiek azelerazio-faktore parekoa dute. Bestalde, ISF metodoa aurreko bi metodoak baino azkarragoa da, baina, prozesu asko erabili behar direnean, errendimendua jaitsi egiten da.



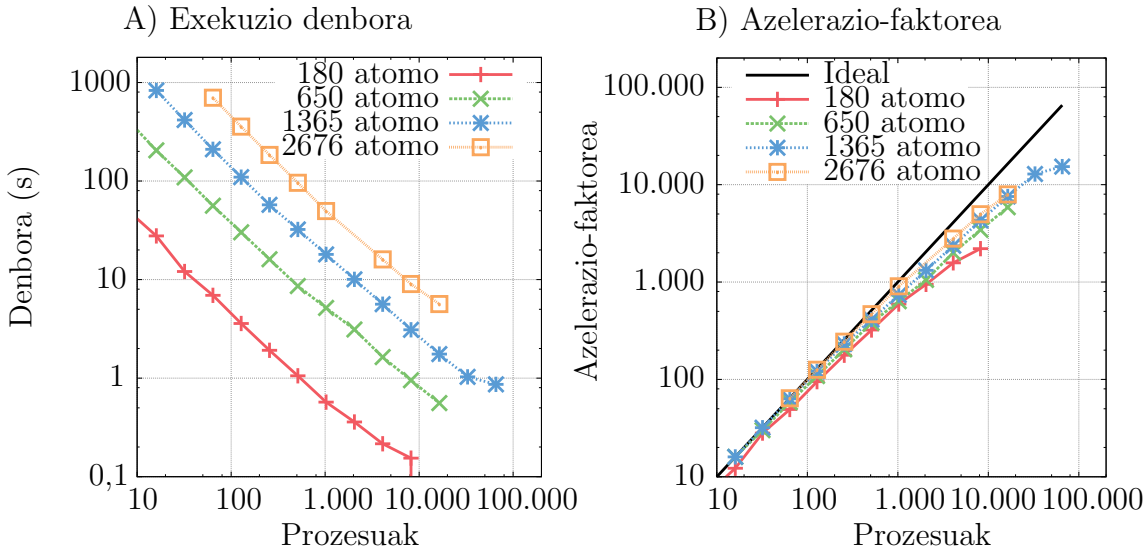
Irudia 0.3: Poisson ebazlea 4.019.679 sareta-puntuko sistema bat Blue Gene/P superkonputagailuan exekutatuta. Prozesu gutxitan ISF metodoa da azkarrena. Aldiz, eskalagarritasun hobea dela eta, Parallel Fast Fourier Transforms azkarrena da prozesu askotan.

Esfortzu berezia egin da FMM metodoa gure aplikaziora egokitzen. Modu diskretuan lan egiteko pentsatuta dago, eta nolabait jarraitua den dentsitatera egokitu behar izan dugu. Hau da, FMM metodoak sareta bateko puntuetan balio zehatzak jorratzen ditu; aldiz, dentsitate elektronikoa banatuago dago, eta hortik moldaketaren beharra. Alde teknikoari dagokionez, Scafacos [17, 18] liburutegiari deia egiten zaio eta lortutako emaitzari geuk garatutako zuzenketa aplikatu. Bestalde, ISF eta Parallel Fast Fourier Transforms metodoak gauzatzerakoan, datu-transferentzia eraginkorrak lortzeko ahalegin berezia egin dugu, sarrerako dentsitate elektronikoa eta irteerako potentzial elektrostatikoa modu desberdinean irudikatuak baitaude liburutegi horietan eta jatorrizko kodean. Bi sareta desberdin horien artean (eta gainera, puntuak modu desberdinean banatuak prozesuen artean) datu-mugimendua era azkar eta eskalagarrian gauzatu dugu. ISF metodoa, berez, Poisson ebazlea da eta liburutegiari deia egitea nahikoa da emaitza lortzeko (kontuan izanda aipatu berri dugun

datu-transferentzia). Parallel Fast Fourier Transforms aldiz, Fast Fourier Transforms liburutegi eraginkor bat besterik ez da, eta Poisson ebazle bezala funtziona dezan bi dei egin behar dira eta, tartean, datuak eskalatu behar dira (aurrez kalkulatutako balioekin biderketa bat aplikatuz).

Azterketa horren ondorioa garbia da, ISF metodoa aukeratu behar da prozesu gutxi erabili behar direnean, eta Parallel Fast Fourier Transforms metodoa, ordea, prozesu askoren kasuan. Gainera, metodo berriek guztiek aurreko errore maila berdindu edo hobetzen dute: hortaz, errendimenduarekin batera, zehaztasuna mantendu edo irabazi da. Beraz, emaitza garrantzitsua lortu dugu, Poisson ebazleak azelerazio-faktore handiak lortzeko oztopoa ez izatea. [19] egileen lanean sakonago azaltzen dira xehetasun guztiak.

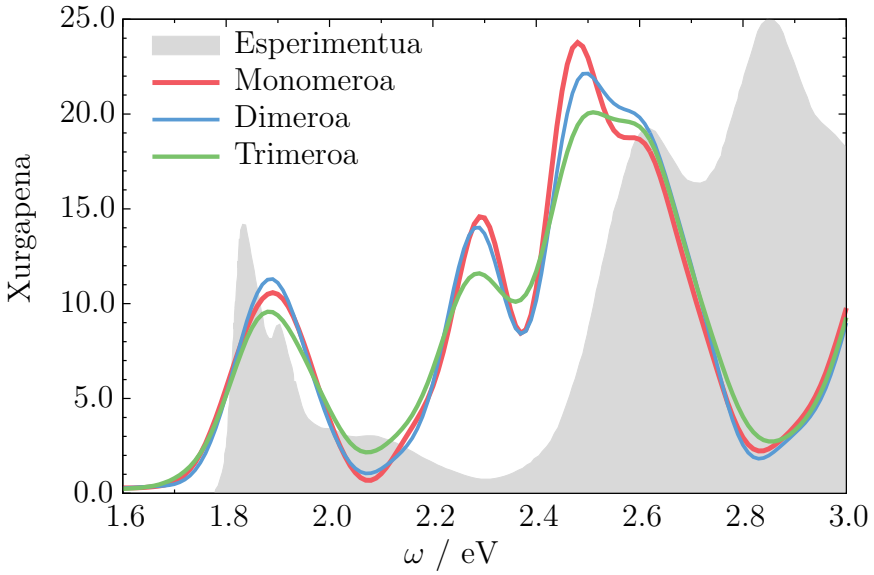
Prozesadoreen erabilera eraginkorraz gain, programek erabiltzen duten memoria kopurua ere oso aintzat hartzeko arloa da. Izan ere, gaur egungo superkonputagailuek milaka edota milioika prozesadore dituzten arren, prozesadore bakoitzak eskura duen memoria kopurua urria izan ohi da (1-8 GB inguru besterik ez). Programak behar duen memoria kopurua simulatu behar den sistema atomikoaren araberakoa da eta independentea behar luke erabilitako prozesu kopuruarekin. Errealitatean, ordea, posible da horrela ez izatea, eta memoria-beharrak gero eta handiagoak izatea prozesadore kopurua handitu ahala. Octopus kodea prozesadore kopuru handietan exekutatu ahal izateko, hainbat memoria- eta errendimendu-arazo konpondu behar izan dira. Hobekuntza horietako adibide gisa, geuk garatutako saretako puntuen banaketa berria aipatuko dugu. Octopus-ek espazio errealean egiten ditu simulazioak eta, horretarako, espazioa hainbat puntutan banatzen du, hiru dimentsiotako sareta osatuz. Paraleloan exekutatu nahi denean, sareta-puntuak prozesadoreen artean banatu behar dira. Gure lanaren aurretik, banaketa hori seriean egiten zen eta memoria asko eskatzen zuen. Orain, berriz, banaketa paraleloan egiten da, ParMETIS liburutegia erabiliz [20]. Exekuzioa azkarragoa izateaz gain, memoria gutxiago behar da. Hobekuntza horren eta beste hainbaten informazio zehatza [21] lanean aurkitu daiteke.



Irudia 0.4: 180, 650, 1365 eta 2676 atomoko sistemak Blue Gene/Q superkonputagailuan exekutatuta (prozesu bat PUZ bakoitzean exekutatuta). Octopus *kodeak eskalagarritasun handia erakusten du.*

Optimizazio guztien ondoren, frogatu ahal izan dugu errendimendu handia lor dezakegula prozesadore asko erabilita, 0.4 irudian ikus daitekeen moduan. Hainbat proba egin dira LHC–II molekularen zati desberdinekin, hain zuzen ere, 180, 650, 1365 eta 2676 atomoko molekulekin (452.878, 1.282.324, 2.371.809 eta 4.106.680 sareta-puntu hurrenez hurren). Sistema txikiena kodearen errendimendu handiaren erakusle da; 4 prozesutan banatuta exekutatu daiteke eta 8192 prozesura arte exekutatu daiteke eraginkor. Hau da, azelerazio-faktore edo *speed-up* handia lortzen du prozesu askorekin (2205eko azelerazio-faktorea du 8192 prozesurekin). Era berean, sistema handienak portaera bera erakusten du, nahiz eta, jakina, exekuzio-denbora handiagoak behar dituen, simulatu beharreko espazioa (sareta-puntu gehiago ditu) eta kontuan izan beharreko elektroi kopurua handiagoak baitira. Aipagarria da, halaber, 1365 atomoko sistema 65.536 prozesutan exekutatu ahal izan dugula, errendimendu handia mantenduz. Tarteko 650 atomoko sistemak ere aurrekoen portaera bera du. Erakusten diren emaitzak Blue Gene/Q superkonputagailuan lortu dira; bestelako arkitekturetan ere pareko errendimenduak lortu ditugu.

Octopus kodea optimizatu eta errendimendu handia frogatu ondoren, proiektuaren amaieran fisikaren ikuspuntutik interesgarriak diren simulazioak egin ahal izan ditugu. Nahiz eta LHC–II molekula osorik exekutatzea erronka bat izaten jarraitzen duen, monomero osatua, eta, dimero eta trimero sinplifikatuak (proteinarik gabekoak) simulatzea lortu dugu. Guk dakigula, sistema horiek dituzten 5759, 4050 eta 6075 atomo kopururekin sekula egin diren simulaziorik handienak dira. 0.5. irudian ikus daitekeen bezala, simulazioetan frogatu ahal izan dugu teoria bat datorrela errealitatearekin; hots, simulazioan eta errealitatean molekula hauen xurgapen-espektroak baliokideak direla.



Irudia 0.5: LHC–II konplexuaren xurgapen espektroa; monomeroa (gorriz), dimeroa (urdinez), trimeroa (berdez) eta esperimentua (grisez) [22] . Dimeroaren intentsitatea erdira jaitsi da, hobe ikus dadin. Irudian, TDDFT metodoaren zehaztasuna ikus daiteke: zehaztasun handiarekin ezkerreko tontorrean eta ezkerretara mugituta gainontzeko bietan.

0.4 Ondorioak

Proiektu honetan kode zientifiko baten optimizazioan lan egin dugu. Nagusiki alderdi informatiko batetik landu den arren, proiektua jakintza-arlo desberdinen arteko lankidetzaren emaitza da: informatikarien, fisikarien eta kimikarien artekoa hain zuzen ere. OCTOPUS software-paketea sistema atomikoen portaera simulatzeko tresna aurreratua da eta mekanika kuantikoko teoriak (DFT eta TDDFT) erabiltzen ditu. Zoritxarrez, tamaina errealeko sistemak simulatzeko, exekuzio-denbora oso handiak behar dira. Beraz, irtenbide bakarra superkonputagailuak erabiltzea da. Makina horiek dituzten ehunka milako prozesadoreetan errendimendu handia lortzea prozesu konplexua da, eta hori izan da proiektu honen helburua. Hori horrela, hainbat arlotan jorratu da optimizazioa. Batetik, Poisson ebazlearen errendimendu mugatu gainditu dugu, aukera berri eta eraginkorrak gauzatuz. Bestetik, memoria erabilera, datu-banaketak eta beste hainbat atal ere optimizatu eta birmoldatu ditugu.

Hobekuntza horien guztien ondorioz, inoiz egin diren DFT eta TDDFT simulazio handienak egitea lortu dugu, tamaina errealeko simulazioak ahalbideratuz. Simulazio horiek aukera emango digute, lehen aldiz, fotosintesian gertatzen diren lehenengo femtosegundoetako erreakzioak ulertzeko.

0.5 Etorkizunerako planteatzen den norabidea

LHC–II molekularen simulazioa egiteko gai garela ikusita, simulazioen emaitzak, fisikako/kimikako alderditik aztertu behar dira orain. Oinarrizko egoera bakar batetik abiatuta, kitzikapen desberdinak aplikatuko dizkiogu sistemari, eta bere portaera ulertzen saiatuko gara. Lan honek guztiak denbora beharko du simulatu eta aztertu ahal izateko. Izan ere, simulazio horiek aurrera atera ahal izateko milaka prozesadore erabili behar dira, hainbat egunez gainera. Gerora, fisika berri bat aztertzen ari garela kontuan izanda, lan nekeza da zer aurkituko dugun aurreikusten eta aukera berrietara irekita egon beharko dugu.

Era berean, aplikazioan egin diren hobekuntzek atea ireki dute tamaina horretako beste hainbat sistema simulatu ahal izateko. Software librea izaki, fisikarien eskura dago kodea, eta ziur gaude beste hainbat fenomeno ulertzeko baliagarria izango dela.

Chapter 1

Context

Contents

This doctoral thesis is the work done in the last five years at the University of the Basque Country UPV/EHU, in collaboration between groups of Informatics Faculty (ALDAPA) and Chemistry Faculty (Nano-bio Spectroscopy). A short stage at the University of Coimbra also was carried out. Obviously, this is not a personal work, but the result of a collaboration with many international scientist of the areas of Informatics, Physics, Chemistry and Biology.

This thesis is mainly focused in the computer simulations of physical phenomena from the point of view of a computer engineer. Computer simulation has enabled the possibility to do a vast variety of tests using few resources. Computer simulation is also a great tool to predict the behaviour of any kind of experiment, before it is actually done, saving a lot money and time. For that to be true, on the one hand, the simulation code has to be written by mainly engineers and scientists. On the other hand, due to always increasing demand of computational resources, big computational centres have to be created. Those centres are managed nationally by the Red Española de Supercomputación (RES) and internationally by Partnership for Advanced Computing in Europe (PRACE). Nowadays, both requirements are met and, therefore, in this project we can do a step forward in computer simulation.

1.1 Introduction

The importance of the simulation using computer has grow in recent years, until reaching a vital importance in many research areas such as Chemistry, Physics, Biology, Astrophysics, Applied Mathematics and different industries. Specially, the understanding of phenomena at nanoscopic scale is a major goal of science since the very moment of the appearance of quantum physics. Computer simulation is an invaluable tool to study these types of phenomena, which otherwise can not be understood. Currently, High Performance Computing (HPC) [23] enables the simulation of atomic and molecular systems according to the fundamental equations of quantum mechanics, opening a new world of opportunities.

The final aim of this project will be to improve a scientific code, which will enable, at the end of the project, the simulation of a really complex molecular system. More precisely, we want to simulate the absorption of the light, or in other words, how the first steps of the photosynthesis are. The main objective of this research is to obtain a highly efficient parallel version of a real scientific code.

The main phases of the project will be the following: (a) introduction of the project and overview of the problem, (b) description of the working environment or facilities, (c) understanding of the physical phenomena on the approach we chose, (d) introduction to the actual scientific code Octopus, that we are going to use in the following, (e) first executions and identification of the problems, (f) solving of detected problems, (g) demonstration of the achieved performance, (h) real-case run to do a scientifically relevant huge simulation, which was not possible before, (i) conclusions.

The computational speed of supercomputers has grown significantly and steadily in recent years, having already more than 30 system passed the petaflop/s barrier. The main reason for the increase in performance lies in the use of massively parallel systems with hundreds of thousands of processors, because improvements in individual processors are already very low. This is owning to the fact that producing ever-faster processors at this time implies that the cost and consumption grows non-linearly, while the use of multiple processors increase computing speed in order to maintain a linear scaling whole system. This trend is already well established and will continue to expand in the future to other areas of computing, as, for instance, with personal multicore computers.

Much of the scientific software that runs on supercomputers is not being able to keep pace with the increasingly capabilities that offer machines with a large number

of processors. There are several reasons for this, but certainly one of them is because developing efficient parallel code is a very complex task that requires dedication, knowledge and skills beyond those that are to be expected in people whose interest is focused mainly on the scientific dimensions of his research. The massively parallel code development must therefore be a multidisciplinary effort, involving researchers from both areas as basic science as computers. The benefit of such collaboration is twofold: on the one hand, the development of high performance scientific code, and secondly, the development and improvement of techniques for parallelisation, which can then be applied to other technical and scientific areas. This thesis is a clear example of such synergistic collaboration between two scientific fields, in which parallel programming experts will collaborate with researchers in physics for the parallelisation of numerical code for simulating electron dynamics.

At the beginning of the project a deep analysis of that code must be done. Firstly, we will evaluate the current state of the code and we will search for the main problems that could prevent for the use of a high level of parallelism, before starting to modify it.

To begin with, it is essential to learn how the scientific software works. We have to be able to easily use and understand the software and all the related tools (operating system, remote tools, queue system...). The atomic systems simulations are not an easy task, and we have to involve in them.

So, we will measure the executions of the code and assess the performance, searching for possible bottlenecks. Once problems have been identified we will propose a way to solve them. All the problems and solutions proposed are discussed in detail in the next chapters.

1.2 Research environment

This work is the result of the collaboration of two research groups of the University of the Basque Country, UPV/EHU: the Nano-bio Spectroscopy group (Fac. of Chemistry), composed mainly by physicists; and ALDAPA (Fac. of Computer Science), whose members are computer engineers. Computer engineers are concerned with the performance of parallel supercomputers, trying to optimise compilations processes, computations, processor communications, the use of the memory hierarchy, etc. On the other hand, physicists care about solving problems of increasingly complexity, that require virtually unlimited computing time. Therefore, joining both expertises we have the possibility of facing up to complex real problems that, otherwise, can not be efficiently solved.

Computer engineers aim is to use the supercomputers to the limit of their maximum performance. The physicists care about the possibility to solve systems of increase complexity, i.e. real problems, but while they are not able to take advantage of all performance they need computer engineers help. In this manner, interesting problems for the physicist could be solved effectively and efficiently.

1.2.1 Nano-bio Spectroscopy group

The Nano-bio Spectroscopy[1] group activities are focused on the field of theory and modelling of electronic and structural properties in condensed matter and on devel-

[1] http://nano-bio.ehu.es/

oping novel theoretical tools and computational codes to investigate the electronic response of solids and nanostructures to external electromagnetic fields. Present research activities are: new developments within many-body theory and TDDFT, including *ab-initio* description of electron excitations, optical spectroscopy, time-resolved spectroscopies, and lifetimes; novel techniques to calculate total energies and assessment and development of Exchange-Correlation functionals for TDDFT calculations; improvements on transport theory within the real-time TDDFT formalism; characterisation of the electronic and optical properties of solids, nanostructures (in particular nanotubes, nanowires and semiconducting -clusters-) and biomolecules. The director of the group as well as ETSF[2] vice-president is Prof. Angel Rubio.

The group is active in the development of the ETSF and hosts the Vicepresidency for Scientific Development.

1.2.2 ALDAPA group

ALDAPA[3] group focuses its research activity in the context of data mining and parallelism. The team trajectory covers, on the one hand, supervised and unsupervised learning paradigms, optimisation, prediction and diagnosis methods. And, on the other, the development and tuning of high performance solutions using massively parallel computers (MPP, GPU...) to simulate complex systems in science and engineering. Nowadays, a new research line is devoted to the study of Physiological Computing Systems and Brain-Computing Interfaces. Research works have been funded by the UPV/EHU, the Basque and Spanish Governments, and the European Community; currently, ALDAPA belongs to a consolidate research group recognised by the Basque Government.

1.2.3 Partnership for Advanced Computing in Europe

Partnership for Advanced Computing in Europe (PRACE) is an initiative that aims to provide European scientists with world-class leadership supercomputing infrastructures. It is an international network that connects supercomputer facilities with the research groups. The main tasks of PRACE is to decide who and how uses the supercomputers, and, thus, resources are efficiently used by final researchers. Every 6 month PRACE grants access to high-end European HPC resources to the best evaluated projects, to different research groups. It is established as an international not-for-profit association. The biggest available systems are:

1.2.4 Red Española de Supercomputación (RES)

The Spanish Supercomputing Network (RES, Red Española de Supercomputación) is a national supercomputer centres network. It was founded in March 2007 after the first upgrade of the MareNostrum machine. The old machine was split among different Universities, creating the network. Besides those HPC centres, the machines on the network are the new MareNostrum III and Magerit at Universidad Politéctina de Madrid (UPM). The usage is awarded every 4 months to the best submitted projects of the scientific community.

[2]ETSF: European Theoretical Spectroscopy Facility. http://www.etsf.eu

[3]http://www.sc.ehu.es/acwaldap/

System Name	Hosting Centre	Launch date	Architecture	Pflop/s
CURIE	GENCI@CEA, France	Q2 2011	Bullx cluster	2
Hermit	GCS@HLRS, Germany	Q3 2011	Cray XE6	5
FERMI	CINECA, Italy	Q2 2012	BlueGene/Q	2
SuperMUC	GCS@LRZ, Germany	Q2 2012	iDataPlex	3
JUQUEEN	GCS@FZJ, Germany	Q4 2012	BlueGene/Q	5
MareNostrum	BSC, Spain	Q4 2012	iDataPlex	1

Table 1.1: The most powerful computers in Europe can be accessed by PRACE.

1.3 Physical problem

Several aspects of the photosynthetic processes have been identified in recent years as being of great potential value to the sustainable development of modern society in the 21st century. A report commissioned by the US Department of Energy outlined several grand challenges facing basic energy research, and highlighted, amongst these the importance of understanding the quantum mechanical effects involved in energy capture and transfer in photosynthetic systems [24, 25]. Similarly, the European Science Foundation highlighted in the field of photonics in its report on Key Enabling Technologies (KETs) for the coming century improvements in understanding of photosynthesis as being vital for potential energy applications and developments [26, 27]. Consequently, photosynthesis is a widely studied phenomena, both experimentally and through simulations. One of the main aims of this project will be to quantum mechanically simulate the very first steps of this complex process.

Central to the continued existence of life on Earth, the conversion of solar energy into a chemical form that can be stored until needed, transported to different parts of the organism and released as required in order to drive the fundamental processes of biology is one of the most highly studied of all biochemical phenomena. The importance of developing a detailed understanding of the microscopic physical phenomena that contribute to the processes of photosynthesis, however, goes far beyond purely academic interest in biological function and has the potential to directly impact upon a wide range of technological applications from optimisation of food crop production [28, 29] to aiding attempts to utilise aspects of photosynthetic light harvesting to generate usable energy either directly in conventional solar cell apparatus [30, 31] or alternatively through secondary processes such as combustion of hydrogen produced by photosynthesis-driven (bio)reactor systems [32, 33, 34, 35].

More than 50% of the light captured by green plants for use in photosynthesis is absorbed by the major Light Harvesting Complex II (LHC–II) [36]. LHC–II is a trimeric protein assembly with three–fold symmetry (Figure 1.1). Despite the fact that LHC–II has been the focus of sustained research activity for some time, many important details such as how interaction with the protein environment modulates the light absorption characteristics of individual chromophores within the complex and how excitation energy is transferred between chromophore sites remain poorly understood. An important barrier to this understanding is that in complex systems such as LHC–II it is very difficult to obtain this information using experimental methods. In cases like this, computational electronic structure methods such as (TD)DFT become particularly valuable since calculation schemes can be devised

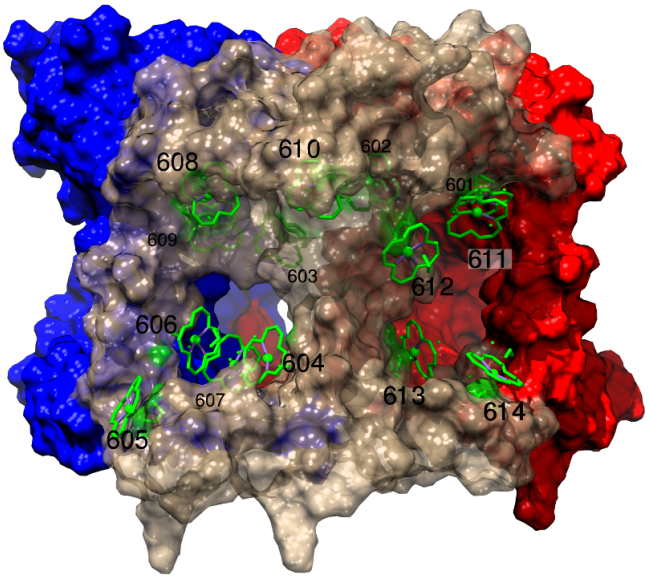


Figure 1.1: Light Harvesting Complex II (LHC–II) molecule, responsible of doing the photosynthesis. It has a three-fold symmetry and each of the parts is a monomer. Each monomer has 14 chlorophylls, coloured in green and labelled from 601 to 614.

with which to investigate well defined aspects of the relevant photophysics. However, although these methods may be suitable in principle for the purpose of studying large systems like LHC–II there remains the problem of prohibitive computational cost due to poor scaling behaviour of the available computer codes.

For testing purposes we have used different portions of the LHC–II molecule. Our test systems consisted of 180, 441, 650, 1365 and 2676 atoms, and contained several chlorophyll units (see Figure 1.2). Those systems have from 452,878 mesh points and 250 electronic states to 4,106,680 points and 3,656 states. The space is represented with cubic meshes with edge length $2L_e$ containing these molecules, where L_e is the half of the edge of the parallelepiped mesh and the used values are 15.8, 22.1, 25.9 and 31.7, respectively.

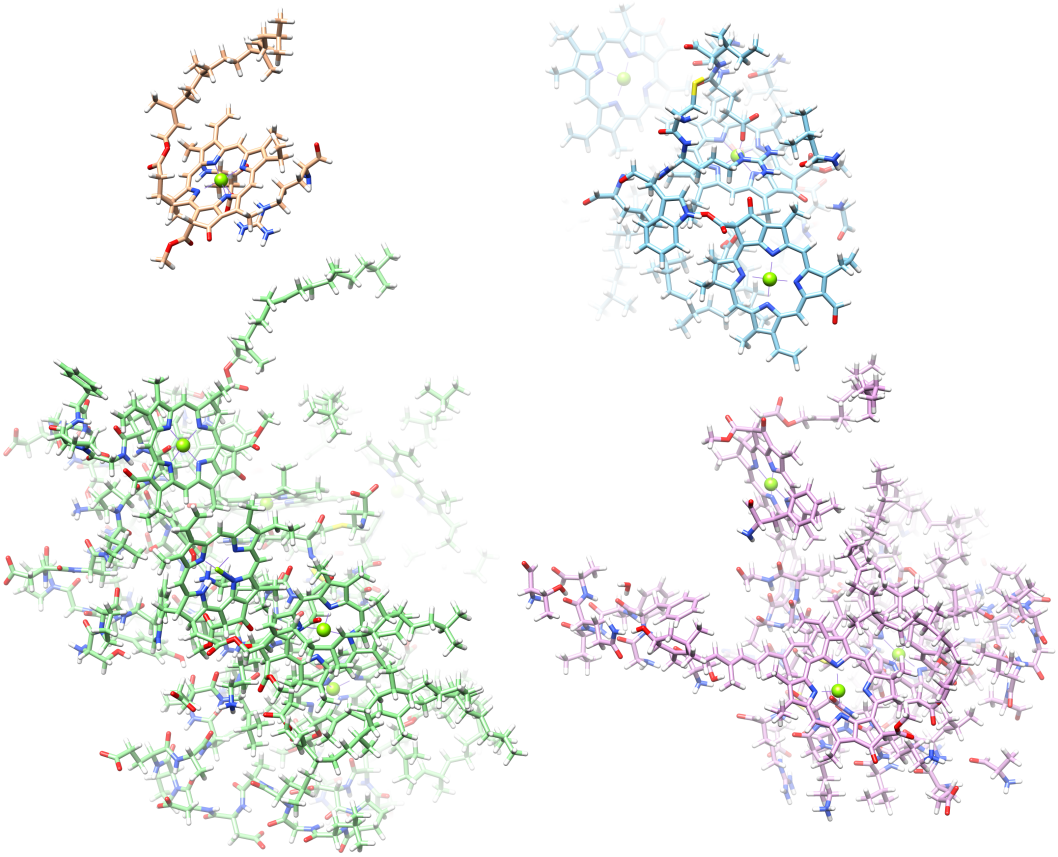


Figure 1.2: Different chunks of the LHC–II molecule, shown systems are composed by 180, 650, 1365 and 2676 atoms (brown, blue, green and pink respectively). Oxygen atoms are coloured in red, nitrogen atoms in blue and hydrogen atoms in white. Remaining atoms are carbon atoms.

Chapter 2

Technical background

Contents

In this chapter we will cover the fundamental technical concepts concerning this PhD project, regarding to the computers. We will introduce all technical background that will be used later in this document.

2.1 Parallelism (main concepts)

Scientific investigations and developments are ones of the most computing time consuming applications. Some of them —climate and weather prediction, geological analysis, atomic physics, astrophysics, engineering simulations... — are pushing ahead the limits of computer's hardware and software every day. Despite modern processors become more and more powerful, the only way to meet those needs is by means of massive parallelism.

In supercomputers, a high number of processors are used in parallel to solve a unique problem in a cooperative way. Parallel computers can be classified using the Flynn's taxonomy [37] according to two main parameters: the number of data and instructions streams. Two cases are considered: only one stream or multiple streams. Parallelism could be achieved applying the same instruction to different data. Processor-arrays (*distributed memory*), vector-computers (*shared memory*) and GPGPU [38] (General-purpose computing on graphics processing units) are of this type. All modern processors have also some kind of vectorial instructions in its machine language, like SSE or AVX in Intel [39] and 3DNow! instructions in AMD [40]).

However, in the general case of the parallelism, p processors collaborate to solve the same problem. There are lots of processes executing over different data, executing each one a different instruction. Two main architectures can be distinguished, depending on the memory arrangement: shared or distributed. In the first category, all the processors share a unique address space. In the nowadays common multicore architectures, the shared memory is centralised, being accessed using the usual data buses. Those systems are denoted as Simetric Multiprocessors (SMP) and because of the memory organization, are limited to a reduced number of processors/cores.

To overcome this problem, the memory system must be physically distributed among the processors, now connected by means of a high throughput network. If the overall memory space remains virtually shared, we have a Distributed Shared Memory (DSM) system, and the interconnection network is used for memory operations; to load/store data in remote memory modules, to maintain data coherence, to synchronise processes, and so.

But if tens of thousands of processes must be used, the common way is to use a private memory space for each process. As memory data are no longer shared, interprocess communication must be done using explicit communication functions (send/receive). These computers are denoted as Massively Parallel Processors (MPP) or clusters. MPP systems are computers made by design to build a supercomputer, composed by tens of thousands of processors (or even millions), usually with a proprietary high performance communication network. They are the most expensive computers. Clusters are made by commodity components and, therefore, they are more affordable, with a better price-performance ratio. They can be composed from few tens up to millions of processors and of-the-shelve communication hardware (10 Gbit Ethernet, Infiniband, Myrinet...).

The above mentioned architectures are the simplest ones. But modern computers use more complex architectures, which can be considered as a mixture of all of them. For instance, the distributed nodes of any supercomputer are not simple processors but small SMP multicore systems, where all the cores (usually from 8 to 32) shared a centralised or distributed memory system.

Moreover, nodes also use coprocessors to accelerate specific computation. As above mentioned, coprocessors use a type of SIMD parallelism. The two main streams include, on the one hand, the use of GPU devices, and, on the other, the use of multicore vector units (like in the Xeon Phi coprocessors). A high performance GPU includes thousand of simple processing units that use a sophisticated memory system. Fine grain parallelism is exploited using thousand of independent threads in parallel, to overcome memory latencies and to maintain occupied all the processing units. The Xeon Phi coprocessor uses another alternative, inherited from the traditional vector processing. For instance, a common PHI coprocessor uses 60 cores each one with a vector processing unit of 512 bits. These 512 bits are processed, for instance, as 16 words of 32 bits, being able to process simultaneously 960 of those words.

Although the implementations of these two approaches are quite different, the underlying ideas are no so far. To have a clear idea about the computation power of these systems, they can offer today up to 1 Tflop/s for some type of computation (mainly, operations with very structured data sets, without dependences among them).

The computers that we have used in this project are MPP machines, —like Fermi, Juqueen and Jugene—, or clusters —Curie, MareNostrum, Vargas and Ganbo—. Data level parallelism is used at both levels: inside the multicore chip and between nodes with message passing model. The detailed description of the machines will be shown below.

2.1.1 Performance of a parallel system

There are several ways to specify the performance of a parallel computer. The simplest one is to measure the number of flop/s (floating point operations per second) the computer can execute. If a superscalar processor can achieve today, for instance, 10-15 Gflop/s, a parallel computer with up to 10.000 processors could achieve 100-150 Tflop/s (tera = 10^{12}).

In addition to the peak performance of the system, three relative measures are also used to qualify a parallel execution: the speed-up, the efficiency and the weak-scaling.

Speed-up

The speed-up is the measurement of how much faster is a parallel program than the serial one. The speed-up compares two execution times, usually the serial time (with only one process) and the parallel time: $S_p = {}^{T_s}/_{T_p}$ where, S_p: speed-up; T_s: serial time; T_p: parallel time. Using a system with p processes, the speed-up should range from 0 to p. Of course, the objective is to use p processes to execute our application p times faster; unfortunately, there are several problems that must be solved in order to obtain this result.

Efficiency

Efficiency is the achieved percentage of the maximum possible acceleration factor. It measures how far we are from the ideal case: efficiency = ${}^{S_p}/_{p}$ where, S_p: speed-up; p: number of processes. So, this parameter will range from 0 to 1, and will tell us how much of the theoretical parallelism of the computer system are we obtaining.

Weak-scaling

Weak-scaling measures the ability of a program to scale with the number of parallel processes at a fixed computing load, comparing the execution time of a problem of size N using p processes with the execution time of the analogous problem of size kN using kp processes. In this way, each process will use the same amount of computational resources, regardless of the number of used processors. If the program is mainly constricted by computational needs, then the execution times should be very similar in both cases. On the other hand, if communication needs dominate the parallel execution, then the execution times will grow with p, usually faster than linearly.

2.1.2 Performance limits

When a program executed in parallel, we must take into account that, in any realistic case, several problems that will limit the expected maximum speed-up or peak performance of the parallel system must be overcome. Unfortunately, parallel execution times will be always higher than the serial time divided by the number of processes, because of some limits and overheads that must be *paid*. Limits are related mainly to the existence of serial parts in the code and with the amount of memory, and overheads include the needed to send and receive data among processes, inefficient load balancing of the computation, replication of some operations, synchronisation of the processes, the input/output processes, and so on.

The serial part of the code

Although the main part of a concrete application could be done in parallel, there is always part of the code that is intrinsically serial or limited to a reduced level of parallelism. The effect of these parts of a parallel code in the overall performance is well established by the famous Amdahl's law [41]. Considering that f is the fraction of the code that can be perfectly executed in parallel (thus, the fraction $1-f$ must be executed in serial), the obtained performance or speed-up can be expressed as: $p/[(1-f)p+f]$

Thus, it will be limited by the serial part to a maximum of $1/(1-f)$. In other words, if only a 1% of the code can not be executed in parallel, then the maximum speed-up will be limited to 100, regardless of the number of processes we use (lower green line in Figure 2.1).

This limit to the speed-up is very disappointing, because seems to limit the use of parallelism to a reduced number of processes or only to "all-parallel" applications. But massively parallel systems are used not only to speed the execution, but usually to execute more accurately scientific and engineering applications, i.e. to process a higher amount of data in a realistic time. For these cases, although the computation amount grows (processing more data) the time devoted to serial processes does not increase (at least not in the same amount). The effect is clear: for those cases, the serial execution time fraction reduces as the problem size increase. This behaviour is reflected in the Gustafson's law [6], where the speed-up can be expressed as: $fp+(1-f)$ (red upper line in Figure 2.1). Amdahl's and Gustafson's laws represent the two extremes of a parallel behaviour. Real applications will be between both of them, allowing in most cases the use of massive parallelism.

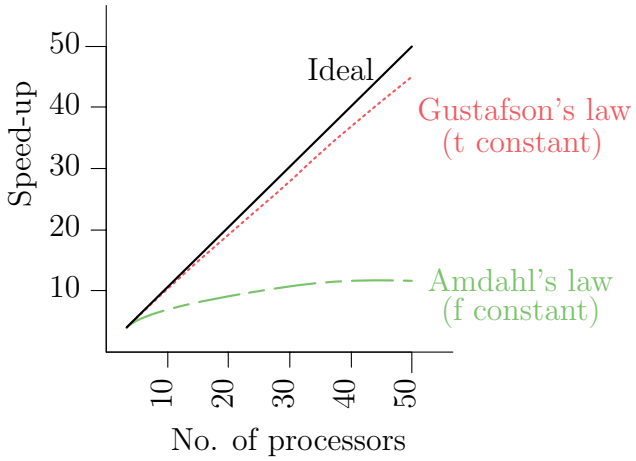


Figure 2.1: Differences between Amdahl's and Gustafson's laws, with a parallel portion of $f = 0.9$. Clearly, there is a big improvement in the speed-up in Gustafson's approximation. Real problems are between both.

Communication among the processes

The main overhead added when executing in parallel are the explicit communication functions that must be added to send data from process to process. Communication time is negligible only for very simple cases, where it is limited to the initial and final phases of the computation; Thus, the parallel execution time must be expressed as $T_p = {}^{T_s}/_{T_p} + T_{comm}$. While the execution time decreases with the number of processes, communication time, needed to send and receive messages, goes up (in many cases more than linearly). So a compromise in the number of processes must be found to achieve the lowest possible time.

Load balancing

Processes to be executed in parallel must be equally distributed among the processors; otherwise, some processors will be idle while other are working and working. The time a processor is idle is not used to do effective calculation, so parallel overall execution time increases. Different assignments are used to schedule processes in a balanced way. The scheduling can be static (tasks are distributed at compile time) or dynamic (tasks are distributed during the execution of the application). Synchronization between processes (point to point, or global) is another source of load imbalancing, where some processes wait for the others to obtain their results.

Input/output

Usually, parallel applications use and produce huge amounts of data, that must be read and stored in external devices (discs) using the so called input/output operations. Although parallel storage systems allow many simultaneous read/write operations, they are very time consuming, so the performance of the input/output system becomes crucial, mainly for those applications that must use external discs very often. At any rate, parallel input/output operations must be used in almost all real cases to save intermediate data for recovery purposes.

When different computing nodes share the same storage device, a shared interface has to be created to access the data. The basic implementation is NFS, which saturates quite soon when few tens of users are using it at the same time. A known alternative is LUSTRE, which scales much better to many users and many

computing nodes. Other proprietary alternatives also exist.

Amount of available memory

Whereas the peak performance of parallel systems is still increasing following the Moore's law [42], the RAM memory available for each parallel process fails to keep pace with this evolution. Moreover, when we increase the accuracy of a physical system simulation, memory needs per core grow usually more than linearly. Therefore, memory becomes often the limiting factor when very big amounts of data must be executed in parallel. If not enough memory is available, data must be retrieved and saved in the external memory system, incurring in serious performance losses. Consequently, many supercomputing centres has the policy to forbid this kind of executions.

2.2 Supercomputers

Supercomputers are massively parallel computers, i.e. a combination of a set of CPU/memory/HD (optional) repeated many times and connected with a high speed and low latency network. Supercomputers are infrastructures that evolves rapidly.

2.2.1 TOP500

The TOP500 list is a manner to arrange the most powerful supercomputers in the world. The performance obtained executing the LINPACK benchmark [7] is used to rank the computers twice a year.

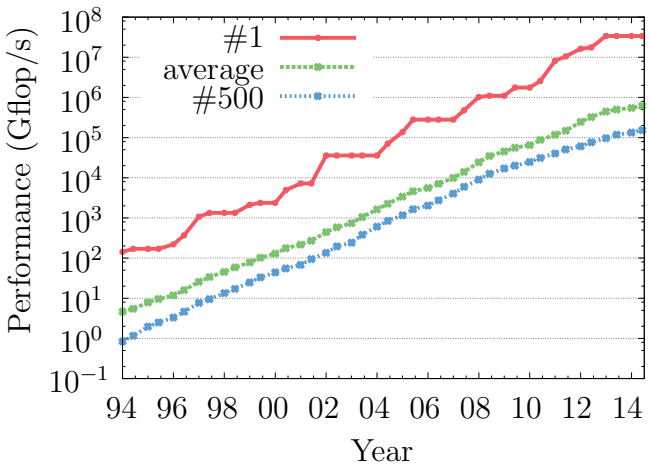


Figure 2.2: Performance in Gflop/s for the computers ranked first, the average and the last.

In Figure 2.2 can be seen the evolution over time of the performance obtained by fastest computers in the world. As Moore's law predicts, performance increases exponentially (for the first ranked, the last ranked and the average of the 500 systems). Nowadays (November 2014), the fastest one is the Tianhe-2 (N.S.C., Guangzhou, China), which obtains 33.9 Pflop/s (3.4×1015) using 3,120,000 cores and Intel Xeon Phi coprocessors. The second is Titan (Oak Ridge N.L., USA), a Cray XK7 system which reaches 17.6 Pflop/s with 560,640 cores and NVIDIA K20 GPUs as accelerators.

In the last list, four systems overcome 10 Pflop/s and 50 go over the Pflop/s. For comparison's sake, the first supercomputer ever to reach the milestone of 1 Pflop/s was the IBM Roadrunner (Los Alamos, USA) in 2008. More than 80% of the computers in the list are cluster, all the rest are MPP systems. They have in average almost 50,000 CPU cores; the smallest has 3,036 cores and reaches 154 Tflop/s. Practically, all the computers run Linux. The 15% of the machines have some kind of accelerator (mainly Nvidia GPU or Intel Xeon Phi).

2.2.2 Green500

Whereas the TOP500 takes only into account the performance (in flop/s) of the system, the Green500 list also considers its energy consumption. Power consumption is crucial for building an HPC system. There exist technology to build up systems with more processors than existing ones, but the electricity power they consume is preventing them from building. Not only the power that the processors require, but also the need to cool them has to be taken into account (roughly, the bill is the half for each them; machines consumption and its refrigeration). Consequently, more efficient processors appear and better ways to cool them. A lower energy requirement will not only reduce the electric bill, but also will help in the reliability, availability, and usability [43]. Reductions on the maintenance and cooling systems will also be benefited, allowing to create more powerful computers.

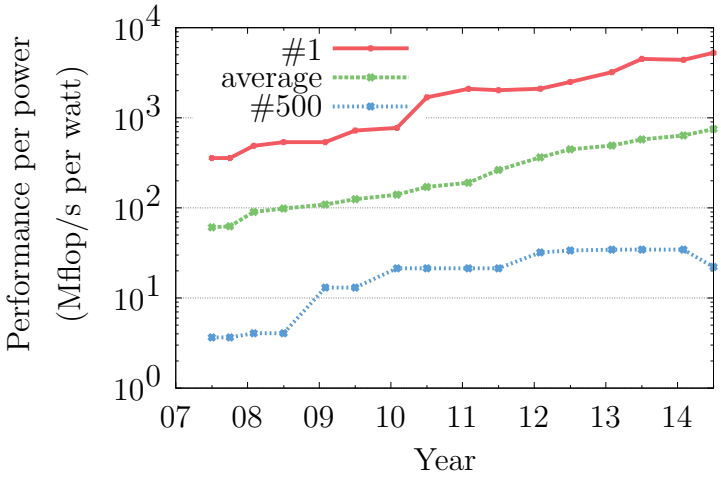


Figure 2.3: Performance per power in Mflop/s for the computers ranked first, the average and the last.

The base of the Green500 list is the TOP500 list, where computers are ranked according to the obtained Gflop/s per watt. Currently (November 2014), the most efficient system, at the GSI Helmholtz Center in Germany, is able to compute 5.3 Gflop/s consuming a watt; using 10,976 cores it reaches 316 Tflop/s and it is ranked 168 in the TOP500 list. As can be seen in Figure 2.3, the grow is also exponential; for instance, the most efficient system in 2008, when Roadrunner bet the Pflop/s, was able to compute only 0.48 Gflop/s per watt (and would be ranked 233 in the last green list). Only the last systems of the TOP500 list are very inefficient, but they are being rapidly replaced.

2.2.3 Parallel Computers used in this project

In this project we have used the most advanced computers in Europe, indeed, the fastest ones: Jugene (in 2009 and June 2010), Curie (November 2010) and Juqueen (November 2012 and June 2013). The fastest of Italy: Fermi (in 2012 and 2013). And also the Spanish fastest ones: MareNostrum II (from 2006 to 2010), Magerit (June 2011), and MareNostrum III (fastest since 2012). Moreover, we have used a local cluster (Ganbo), and a cluster in France; Vargas at IDRIS.

All the computers presented here as in all supercomputers facilities, inside a room designed to cool the computer and managed by dedicated personal. Computing nodes are accessed through common front-end or login nodes, which are used to compile applications, submission of batch jobs, management of hard disks, etc. For the batch submission LoadLeveler and SLURM [44, 45] software utilities are mainly used. The scheduler is responsible for managing jobs on the machine by allocating partitions for the user on the compute nodes, returning job output and error files to the users.

Fermi // Juqueen (Blue Gene/Q MPP)

Both Fermi and Juqueen machines share the IBM Blue Gene/Q architecture [46]. It is a high-end architecture, nowadays 4 machines with this architecture are in the top10 of the TOP500. Fermi machine is located at CINECA and it is the most powerful system there. CINECA's hardware resources are the most powerful available in Italy and among the most powerful available in the world. Fermi is an IBM Blue Gene/Q system composed of 10.240 PowerA2, totalling 163.840 compute cores and a system peak performance of 2.1 Pflop/s. Fermi is one of the most powerful machine in the world, and had ranked #7 in the TOP500 list in June 2012; in November 2014 occupies the 23 position.

Figure 2.4 shows the overall structure of the Blue Gene/Q system. Each compute card (or compute node) features an IBM PowerA2 chip with 16 cores working at a frequency of 1.6 GHz, with 16 GB of RAM and the required network connections. A total of 32 compute nodes are plugged into a so-called node card, and 16 node cards are assembled in one midplane, which is combined with another midplane and two I/O drawers to give a rack with a total of 16K cores. Fermi is composed by 10 of those racks, amounting a total of 160K cores.

Compute nodes use a light Linux-like kernel called Compute Node Kernel (CNK); they are diskless, and I/O functionalities are provided by dedicated I/O nodes, that provide a more complete range of OS services, e.g.: files, sockets, process launch, signalling, debugging, and termination. Two of the 10 racks have 16 I/O nodes each, implying a minimum job allocation of 64 nodes - 1024 cores (because of the fact that a job must have at least one I/O node allocated to it). The other 8 racks have 8 I/O nodes each (and, thus, a minimum allocation of 128 nodes – 2048 cores).

In addition to common login nodes, some service nodes perform system management services (e.g., partitioning, heart beating, monitoring errors) and can be used only by system administrators. Access to the compute nodes are mediated by I/O nodes, since only they are able to interact with the file systems. The scheduler of the FERMI system is LoadLeveler.

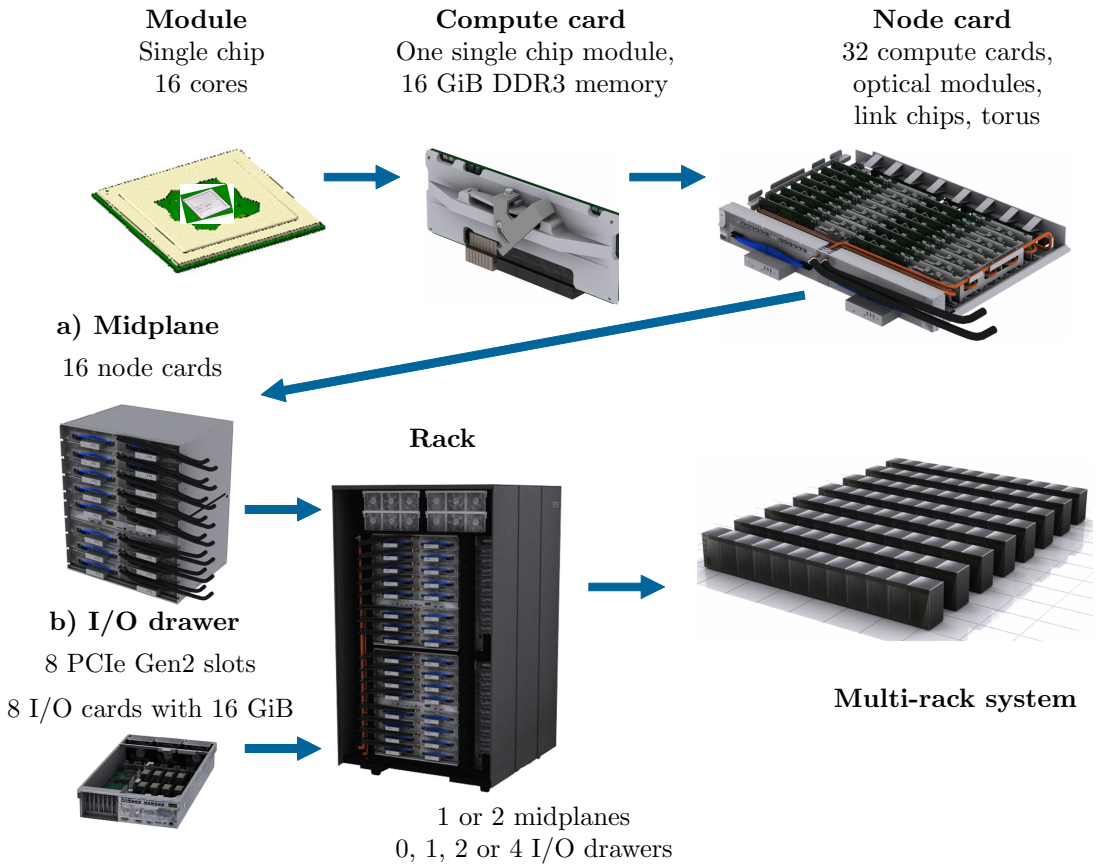


Figure 2.4: Blue Gene/Q system overview.

Juqueen machine is at Jülich Supercomputing Centre (JSC),[1] substituting the next explained Jugene. The configuration of Juqueen is the same as Fermi, but instead of having 163,840 CPU cores, it has 458,752.

Jugene // Genius (Blue Gene/P MPP)

Jugene (Jülich Blue Gene/P) [47] was a supercomputer located at Jülich Supercomputing Centre (JSC)[2]. It had 294,912 cores in 73,728 chips, each one with four IBM PowerPC processor cores at 850 MHz [48]. Each chip was capable of 13.6 Gflop/s and had 2 GB of RAM memory. In total, the machine had a peak performance of 1 Pflop/s and 144 TB of RAM memory.

The machine was composed in the following way: the base part of the machine was the mentioned 4 cores chip, also referred as a processor. One chip was soldered to a small motherboard, together with memory (DRAM), to create a compute card (one node). The amount of RAM per compute card was 2 GB and they do not had any type of hard disk. All this compute cards were plugged to a node card. There were two rows of sixteen compute cards on the node card. A rack hold a total of 32 node cards, and in total were 72 racks. This structure is similar to the one illustrated in Figure 2.4.

Jugene machine had 5 communication networks and each compute node had 4

[1] http://www.fz-juelich.de/jsc/juqueen
[2] http://www.fz-juelich.de/jsc/jugene

network interfaces: (a) connections to the general point-to-point 3D torus network, 2 per dimension; (b) 3 connections to the collective communication network; (c) 4 connections to the barrier network; and (d) 1 connection to the control network. I/O nodes, in addition, were connected to the a 10 gigabit Ethernet network.

Jugene used LoadLeveler queueing system [48]. There were small queues with a minimum of 32 nodes to reserve and a maximum of 256, and a maximum wall clock time of 30 minutes. There were also bigger queues, until 32,768, that they have the maximum clock time in 24 hours [49].

A similar system, named Genius, with 16K cores was available at the Rechenzentrum Garching of the Max Planck Society, in Germany.

Hydra (x86-64 cluster)

This cluster is substituting the old Genius machine in the Rechenzentrum Garching of the Max Planck Society. It is an IBM iDataPlex DX360M4, with Intel Xeon 2680 processor (20 cores per node) and 64 GB of main memory. In total, it has 65,320 cores and a theoretical performance of 1.5 Pflop/s (1.3 Pflop/s running LINPACK). It has a consumption of 1,260.00 kW.

Curie (x86-64 cluster)

Curie is a supercomputer that belongs to the French Commissariat à l'Energie Atomique (CEA) and that consists of Intel Xeon processors interconnected by an Infiniband network. Each compute node has 32 cores (4 chips of 8 cores each) and 128 GB of RAM. In total 11,520 cores and almost 46 TB of RAM are available. The shared file system is a General Parallel File System (GPFS) and it has a capacity of 4 PB.

MareNostrum II & III (x86-64 cluster)

MareNostrum II was a supercomputer located in Barcelona, at the Barcelona Supercomputing Center. It was the most powerful until 2010 in Spain and it was the fifth in the world in November 2006 [50]. It had 10,240 IBM PowerPC 970 (2.2 GHz) processors connected with a low latency Myrinet network. The peak performance was 94.2 Tflop/s, and running LINPACK it reached 63.8 Tflop/s.

The supercomputer was based on the IBM eServer BladeCenter JS20 [51] and it was structured in this way:

- *Blades*: contains the basic components of a computer: processors (one or two chips), hard disks and memory. Each one runs its own copy of the operating system.

- *Midplanes*: it is a base that contains several blades and share common components: power supplies, fans, CD-ROM, and floppy drives.

The MareNostrum III has replaced the old MareNostrum II at the Barcelona Supercomputing Center, and it is the fastest supercomputer in Spain since 2012. The new system has 3,056 nodes, each one with a 2 Intel Xeon E5-2670 (x 8 cores) and 32 GB of RAM, connected with Infiniband FDR. It has a LINPACK performance of 925.1 Tflop/s.

Vargas (Power6 cluster)

Vargas was an IBM SP Power6 server located in France, at IDRIS, Institut du Développement et des Ressources en Informatique Scientifique. It had 3,584 Power6 cores connected with an Infiniband network. The Power6 processor is a dual core with a clock frequency of 4.7 GHz, and 3 levels of cache. The total amount of memory was 17.5 TB. It had a peak performance of 67.3 Tflop/s.

Magerit (Power7 cluster)

Magerit is a IBM Power7 computer located in Cesvima, at the Universidad Politécnica de Madrid. It is composed by 245 eServer BladeCenter PS702, connected with a Infiniband DDR network. Each of the nodes has 2 Power7 of 8 cores at 3.3 GHz, and 32 GB of RAM, performing 422.4 Gflop/s. It has SUSE Linux Enterprise Server SP1 for the operating system. In total it has almost 3920 CPU cores and a maximum performance of 72 Tflop/s.

Ganbo (x86-64 cluster)

Ganbo is the computational cluster of the Nano-bio Spectroscopy group. This cluster is structured as shown in Figure 2.5, with a management node and several computing nodes.

Due to several upgrades of the system, it is composed by several islands of computing nodes, listed below:

Nodes	CPU Cores per node	Processor	Freq. (GHz)	RAM (GB)	SSD (GB)	HD (GB)
8	16	Xeon E5-2620	2.10	64	120	-
80	12	Xeon E5645	2.40	48	120	-
42	8	Xeon E5520	2.27	24	80	-
10	8	Xeon E5345	2.33	16	-	200
9	12 512 (GPU)	Xeon E5645 & 2 × Nvidia M2090	2.40 1.3	24 6	120	-

All the computing nodes are connected through a low latency Infiniband network, with a tree topology. In total the cluster has 1612 CPU cores, 9216 GPU cores and 5.7 TB of RAM memory. For the storage the machine has, on the one hand, two cabinets of NFS (with 10 and 16 TB) and, on the other, a fast I/O LUSTRE system with 8.8 TB.

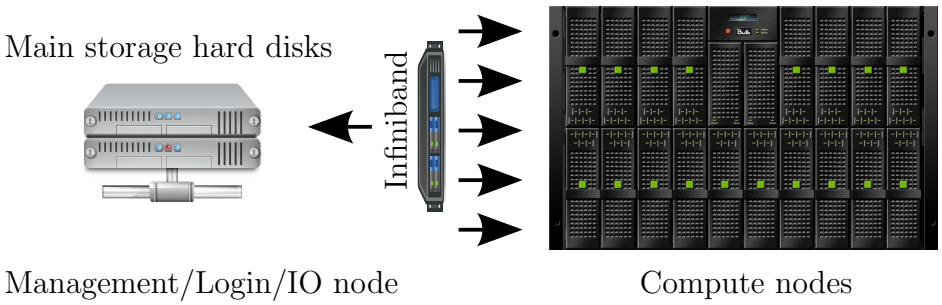


Figure 2.5: Overview of Ganbo cluster: one service node and 149 compute nodes.

2.3 Parallel programming tools

Up to day, efficiently parallel programming is a complex task that must be carried out by the programmers. Automatic parallelisation at compiling time is limited to elementary codes. This is the reason why they exist different types of Application Programming Interfaces (API) to take advantage of the parallel environment, just as MPI, OpenMP, Cuda, OpenCL and others.

Scientific libraries are also widely used in HPC. Furthermore, profiling techniques are used to get an idea of how is the execution of a program.

2.3.1 Parallel programming models

Although there are other options, OpenMP and Message Passing Interface (MPI) has become the *de facto* standards to develop parallel applications, the former for shared memory systems and the later for distributed memory systems. This classification is not strict; for instance MPI can also be used for shared memory applications, and, UPC (Unified Parallel C) is a known alternative [52] for programming distributed shared memory applications. Besides, the best known approaches to write parallel programs in the GPUs are OpenCL and CUDA.

OpenMP

OpenMP is a shared-memory API that "implements" the shared-memory parallel programming paradigm, based on parallel threads. OpenMP is mainly devoted to obtain loop parallelism. Specific pragmas must be introduced previous to loop, to indicate the compiler that the loop can be safely executed in parallel. Several clauses of the pragma indicate the scope of the variables (private, shared...), the scheduling of loop iterations, etc. Using shared variables implies that threads must be synchronized to ensure a correct access to these variables; hence, typical synchronization functions, like critical sections, locks, barriers... are include in the API [53].

MPI

Message Passing Interface (MPI) is an standard API specification developed by the Message Passing Interface Forum (MPIF) that implements message-passing model between computers [54], and is widely used in supercomputers and clusters.

The processes that are in different nodes have separate memory addresses and they have to communicate each other. In fact, MPI is mainly a library of communication functions. Briefly, communication can be point-to-point or global. Point-to-point communication is carried out using send and receive functions; on the other hand, global communication functions include broadcast, scatter and gather operations, to be able to distribute and collect data among all the processes.

UPC

Unified Parallel C (UPC) is an extension of the C programming language designed for high performance computing on large-scale parallel machines with a common global address space. The programmer is presented with a single shared, partitioned address space, where variables may be directly read and written by any processor, although each variable is physically associated with a single processor [55].

Cuda // OpenCL

Cuda and OpenCL are the best known alternatives to write programs that will run in accelerators (GPUs and coprocessors). Whereas Cuda is only valid for Nvidia GPUs, the OpenCL standard could run in both AMD and Nvidia GPUs, and also in Intel Xeon Phi coprocessors. Alternatively, Xeon Phi vector coprocessors uses special vector instruction that can be used by the program or the compiler.

2.3.2 Scientific libraries

Some mathematical functions are used in practically all scientific applications. These functions are implemented and optimized in parallel libraries, so the programmer does not need to write code for them [56]. These are the most used ones:

Basic Linear Algebra Subprograms (BLAS) library is a sort of routines of basic vector and matrix operations. Linear Algebra PACKage (LAPACK) is a library to solve and analyse linear equations, linear lest-square problems and to compute eigenvalues and eigenvectors for several types of matrices. ScaLAPACK is the parallel portable version of some of those routines. The Fast Fourier Transform in the West (FFTW) is a C library to compute multidimensional, complex and discrete Fourier transforms. Also, there are other efficient FFT implementations; Goedecker, Boulet and Deutsch [57] is one of them.

Intel Math Kernel Library (MKL) is an example of a library of highly optimized, extensively threaded math routines for science, engineering, and financial applications, that integrates BLAS, LAPACK, ScaLAPACK, Sparse Solvers, Fast Fourier Transforms, Vector Math, and more libraries in only one package.

2.3.3 Profiling

When scientific software becomes a complex net of code —thousand of lines and lots of files or subroutines calling each other even inside loops— it is quite hard to analyse programs behaviour: what part of the code is the responsible of the execution time? are we efficiently using the memory hierarchies?

This is specially true if the program is being executed in parallel, because the execution of the program is not controlled by a unique control unit, communication between processes introduces non deterministic effects, and the load balancing is not assured.

To clarify the mess, the user can introduce instructions to trace the program (printing some variables, collecting execution times...), but the most efficient way is to use profilers, specific tools to thoroughly profile the execution of programs.

Profiling a program we obtain specific data about the program execution itself: execution times, memory usage, communication patterns and time, etc. These data can be used to tune the program in order to obtain faster executions, a better usage of the memory, etc.; in other words, to execute programs more efficiently.

Profling can be “quite simple”, like Gprof (profiling of time in serial), Massif tool of Valgrind package (it measures the memory usage per each process) or Jumshop (profiling of MPI calls in parallel); or very complex and complete like TotalView or Paraver (profilers for parallel programs).

2.3.4 Other tools

Although next tools presented below are not inherit of a parallel programming, these additional tools make easier every-day use of the working environment. Some example are: Subversion, Buildbot, Trac and Doxygen

Subversion (also known as SVN or svn) is a revision control software. It is aimed to be a repository of any kind of files within a project, having documented all the versions. Whole repository has an unique identifier and any change can be reverted. For every file it is stored the creation date, change time, author and the description of the change, among other information. It is mainly used in the code development or to maintain a web page.

Subversion has a client-server architecture. There are different clients to access to the server, such as command lines, Windows explorer extensions (TortoiseSVN), Eclipse plug-ins (Subclipse)...

We have saved a repository for the developed source code, and also, for article editing and sharing.

Trac is mainly a web interface of the Subversion repository. But not only that, it also creates a to-do list, manages tickets with the pending tasks, sends email in every change in the repository, etc. http://www.tddft.org/trac/octopus

Buildbot is a tool to check the correction of a source code [58]. Checks can be scheduled automatically. In our project, Buildbot can be found at http://www.tddft.org/programs/octopus/buildbot/.

Doxygen is a tool to create documentation from the comments in the source. It is a powerful tool: it is able to create complex diagram for calling and called functions, for data-structure hierarchies, and so on. The documentation created for this project can be found in the http://www.tddft.org/programs/octopus/doc/doxygen_doc/index.html

Chapter 3

Physical background

Contents

Due to the interdisciplinary nature of this PhD thesis, an introduction to the used physical theories is needed. In this chapter we provide some basic definitions of the quantum mechanics, namely, about DFT, TDDFT, LCAO and Poisson solvers.

Understanding the behaviour of the Nature to be able to predict it, is one of the major goals of the humans. Physics is one of such example, which studies the matter, and its motions and reactions over the time and space. In this dissertation we will focus in the study of the matter in a region where the motion is really fast and the size is really small. More precisely, we will introduce the reader with some basic concepts in first principle theories (which means that they do not depend on other theories, called *ab-initio*) of the quantum mechanics.

In this small introduction we will conceptually introduce some basic theories that apply to the matter in the quantum Physics approach. Even if the presented theories are in principle exact, in order to be able to compute it, some approximations might be needed. For example, it is negligible the error that appear treating classically the nucleus, compared with the fast and lightweight electron.

After a general introduction of the main concepts, we will present the Density Functional Theory (DFT) and Time-Dependent Density Functional Theory (TDDFT), with an special emphasis in the LCAO technique and Poisson solver. The LCAO is required to properly initialise a DFT calculation. The Poisson solver, denoted also as the electrostatic interaction between charges, is present inside both DFT and TDDFT methods. Nonetheless, this interaction is one of the most common problems in scientific computing, therefore, it is present in atomic and molecular simulations, and also in fields like quantum chemistry, solid state physics, fluid dynamics, plasma physics and astronomy, among others.

The reader has a great introduction to quantum mechanics in the chapter 1 of the book of Harrison [59].

3.1 Basic definitions

Before giving any further detail, we shall introduce some main concepts of physics in quantum mechanics that we are going to use later on.

Functional: a function of a function. The input variable of functional G is $f(x)$, which is another function (f a function depending on x), so $G(f(x))$ and the result is either a real or a complex number. So, $G(f(x)) \to \mathbb{K}$ ($\mathbb{K} \in \mathbb{R}|\mathbb{C}$).

An example of a functional is the integral. The input parameter of the integral is a function, and the result is a number.

Electronic density: a measure of the probability of an electron occupying an infinitesimal element of space surrounding any given point. It is a scalar quantity depending upon three spatial variables, and it is denoted as $\rho(\vec{r})$. The electronic density can be calculated from the system wavefunction:

$$\rho(\vec{r}) = N \int |\Psi(\vec{r}, \vec{r}_2, .., \vec{r}_N)|^2 d^3r_2, .., d^3r_N \tag{3.1}$$

were N is the number of elements (electrons) of the system, $\vec{r}$ is the position of each element and Ψ is the system wavefunction.

Operator: mathematically, an operator is a mapping from one vector space or module to another. In physics, an operator is a function over the space of physical states.

Hamiltonian: is the operator corresponding to the total energy of the system. The Hamiltonian sums up all the energies involved in the system in a unique term.

In general is written as

$$\hat{H} = \hat{T} + \hat{V} \tag{3.2}$$

where $\hat{T}$ is the kinetic energy and $\hat{V}$ is the potential energy [60]. A general form of the Hamiltonian is build taken into account the general five contributions [61], shown in Figure 3.1: electron and nuclei kinetic energies, electron nuclei attraction, electron electron repulsion, and nuclei nuclei repulsion.

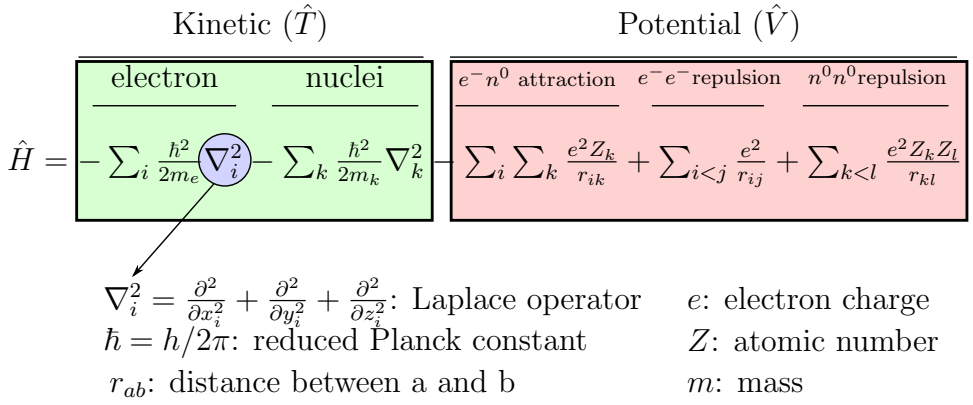


Figure 3.1: Hamiltonian example

Expectation value: the real number obtained from an operator, applying the conjugate transposed wavefunction from the left and the wavefunction from the right. For example the expectation value of operator $\hat{O}$ can be found as follow:

$$\langle \hat{O} \rangle = \int \Psi^*(r_1, .., r_N) \hat{O} \Psi(r_1, .., r_N) dr_1, ..dr_N \tag{3.3}$$

Energy: the energy of a quantum system can be found from the expectation value of the Hamiltonian:

$$E = \langle \hat{H} \rangle = \int \Psi^*(r_1, r_2, .., r_N) \hat{H} \Psi(r_1, r_2, .., r_N) dr_1, dr_2, .., dr_N \tag{3.4}$$

Observable: it is a hermitian operator which its expectation value gives the value for a measurable physical quantity. For example, observable is: any type energy, potential, expectation value of the energy, electronic density.

State: the solution of the stationary Schrödinger equation are the eigenstates (besides the eigenvalue (Energy)), which correspond to the states of the system. Those states are a discrete set: only some permitted energy exists. The resulting energy states are related to the frequency.

Orbital: the eigenstate of the system, when applied to atoms is equivalent to atomic state.

Wavefunction: the state of a system is described by its wavefunction. Thereby, any value or characteristic (observable or not) can be obtained from the wavefunction; or, to simplify it, the wavefunction can be thought as a black box to ask whatever characteristic of the system, with the proper operator. More precisely, the wavefunction represents the probability amplitude of finding the system in one state. Ψ is in general a function that depends on space (x,y,z) and time (t) [62] and it is depicted with complex numbers. $|\Psi|^2(\vec{r})$ is a real number, the probability of finding

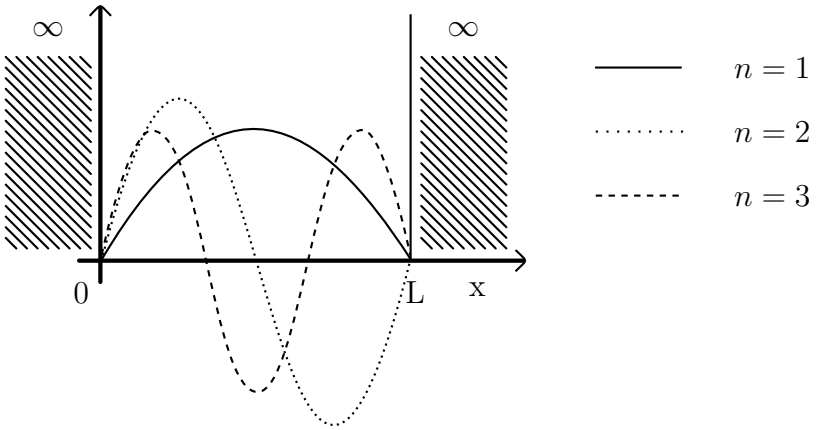


Figure 3.2: Three first solutions (stationary states) for one single particle in 1D (from 0 to L). $n = 1$ solution has the lowest energy. Different number of oscillations, depending on n, for the Stationary Schrödinger equation (3.5).

an electron in a given position at a given moment, thus, the electronic density. If we assume a single-electron system and integrate all over space, the probability will be 1, so: $\int |\Psi(x, y, z, t)|^2 dxdydz = 1$. Generalising, a N element system will be integrated to N [63] $\Psi : \mathbb{R}^3 \to \mathbb{C}$. The wavefunction is not a physical observable.

Stationary Schrödinger equation: The Time-Independent Schrödinger equation describes the behaviour of a stationary system and it is denoted as;

$$\hat{H}\Psi = E\Psi \tag{3.5}$$

where E is the energy (eigenvalue) which has to be minimised, $\hat{H}$ is the Hamiltonian of the system and Ψ is the set of wavefunctions, written also $\Psi = \Psi(\vec{r_1}, \vec{r_2}...\vec{r_N})$ where N is the number of elements of the system and $\vec{r} = (x, y, z)$ spatial Cartesian coordinates. One can say that the solution of the stationary Schrödinger equation is a set of wavefunctions. This solution will give the Ground State of the system, or, e.i. the lowest possible energy of the system.

Schrödinger equation: In quantum mechanics, the time evolution of a system can be described by the Schrödinger equation,

$$\hat{H}\Psi = i\hbar\frac{\partial}{\partial t}\Psi \tag{3.6}$$

where $\hat{H}$ is the Hamiltonian of the system, i is the imaginary number, $\hbar$ is the Planck constant divided by 2π (which is known as the reduced Planck constant), t is the time and Ψ are the wavefunctions of the system. The Hamiltonian and the wavefunctions depend on the time and space, so: $\hat{H} = \hat{H}(\vec{r}, t)$ and $\Psi = \Psi(\vec{r}, t)$. Although the solution of the Schrödinger equation (3.6) is exact, it can only be solved for really small number of particles [11]. This is because many-particle wavefunctions scale exponentially: $\Phi(\vec{r}_1, \vec{r}_2, .., \vec{r}_N)$.

3.2 Density Functional Theory

The need of the Density Functional Theory (DFT) [8, 9] comes from the fact that the stationary Schrödinger equation (3.5) can not be solved in a real environment for a many-body system. In the real world, each element of the system interacts with all the rest, creating a $\mathcal{O}(N^3)$ complexity problem.

To illustrate the complexity of a many-body system, we choose a simple system of a single nitrogen atom, which has only 7 electrons. So, from equation (3.7) can be seen that the system wavefunction has 21 coordinates.

$$\Psi(x_1, y_1, z_1, x_2, y_2, z_2 ... x_7, y_7, z_7) \tag{3.7}$$

Solving the Schrödinger equation in this many-body problem is too complex. In this simple example, we discretise the wavefunction to 10 mesh points, and if we create a rough table of 10 entries for each coordinate, it will have 10^{21} entries in total. Assuming only 1 byte per entry, it will occupy one zettabyte, only for one single nitrogen atom. Clearly, unaffordable.

The DFT comes to solve this problem and can be summarised with the next main ideas:

- First theorem of Hohenberg-Kohn (HK): all observable properties of a N-electrons system can be obtained from its Ground State electronic density (ρ). In particular, the energy of this system can be expressed as

$$E_{gs}[\rho] \equiv F_{\text{HK}}[\rho] + \int V_{ne}\rho(r)dr \tag{3.8}$$

  where the first term contains the electronic kinetic energy and the electronic interaction of the real system. The second term describes the nucleus-electron electrostatic interaction.

- Second theorem of Hohenberg-Kohn (HK variational principle): the minimum value of the energy of a N-electron system is achieved only if the test density $\tilde{\rho}$ corresponds to the real Ground State density (ρ). This means that any guessed density's ($\tilde{\rho}$) energy will be greater or equal to the GS, i.e. $E_{GS} \leqslant E[\tilde{\rho}]$.

- Levy-Lieb reformulation of the DFT: although HK formulation is strictly *ab-initio*, universal expressions of the density and the external potential are unknown. Levy-Lieb reformulate DFT finding the connection between the variational principle of quantum mechanics, which uses wavefunctions as variables, and HK postulates. Levy-Lieb proposed a way to obtain the solution of equation (3.8): on the first step, between all possible wavefunction for a given density, it minimises the energy. Once having them, in the space of densities has to be found the minimum one. Levy and Lieb proposed a way to compute the unknown HK function (F_{HK}). However, this formulation loose the attractiveness of original Density Functional Theory needing the calculation of many-body wavefunctions (Ψ).

  Next equation (3.9) represents that:

$$E_{gs} = Min_{\rho}\left[Min_{\Psi\to\rho}\left\langle\Psi|\hat{T} + \hat{V}_{ee}|\Psi\right\rangle + \int V_{ne}(r)\rho(r)dr\right] \tag{3.9}$$

- Kohn-Sham theorem: finally, Kohn-Sham (KS) proved that the Ground State energy can be obtained from the electronic density of an equivalent non-interacting system, avoiding then the problem to solve the calculation of the many-body interacting system. Instead, a non-interacting wavefunction (i.e. Slater determinant) should be construct on the basis of (non-interacting) KS orbitals. Then, the KS potential, used to solve the Schrödinger equation for the N-non interacting system, can be expressed as:

$$v^{KS}(r_1) = V_{ext} + \int \frac{\rho(r_2)}{r_{12}} dr + V_{xc} \tag{3.10}$$

where V_{ext} is the external potential, usually the electrostatic interaction between nucleus and electrons. The second term is the so called Hartree potential, that can be computed using Poisson solver. And the last term refers to the Exchange-Correlation potential. This potential includes all the differences between the real system and the proposed (non-interacting) KS system. This term is computed as:

$$V_{xc} = \left(\frac{\partial E_{xc}[\rho]}{\partial \rho} \right)_{\rho=\rho_0} \tag{3.11}$$

where, ρ_0 is Ground State electronic density for an interacting N-electron system. However, a universal expression for the Exchange-Correlation functional ($E_{xc}[\rho]$) is still unknown, and many approximations are reported on the literature.

So, the essence of the DFT is that every observable value can be calculated from a non-interacting GS electronic density, without having to calculate the many-body wavefunctions. In other words, the solution of a non-interacting system has the same properties as an interactive one [60]. In theory, this approach is exact. But, in reality, the Hamiltonian has to be approximated; whereas kinetic, potential and Coulombic energies can be exactly calculated, it is not the case for the exchange and correlation.

The solution of the DFT is solved self-consistently; it is an iterative process, shown in Figure 3.3, and it is known as Self Consistent Field (SCF) cycle. An initial guess for the density is obtained from the molecular orbitals calculated with LCAO. Molecular orbitals are used to construct the density, using the Slater determinant. Having the density, the electrostatic potential is obtained after solving the Poisson equation (more details about the Poisson solver are given in the next Section 3.5). This potential is used to evaluate the convergence; if the system is not converged, KS wavefunctions are constructed again and the process starts over.

3.3 Time-Dependent Density Functional Theory

DFT describes well the system that they do not change over time, but more interesting phenomena happen over time. So, Time-Dependent Density Functional Theory (TDDFT) [10, 11, 12] is the variant of the DFT considering also the time. We will consider a system with N particles, which is involved over time and it reacts to a external perturbation, such as a laser.

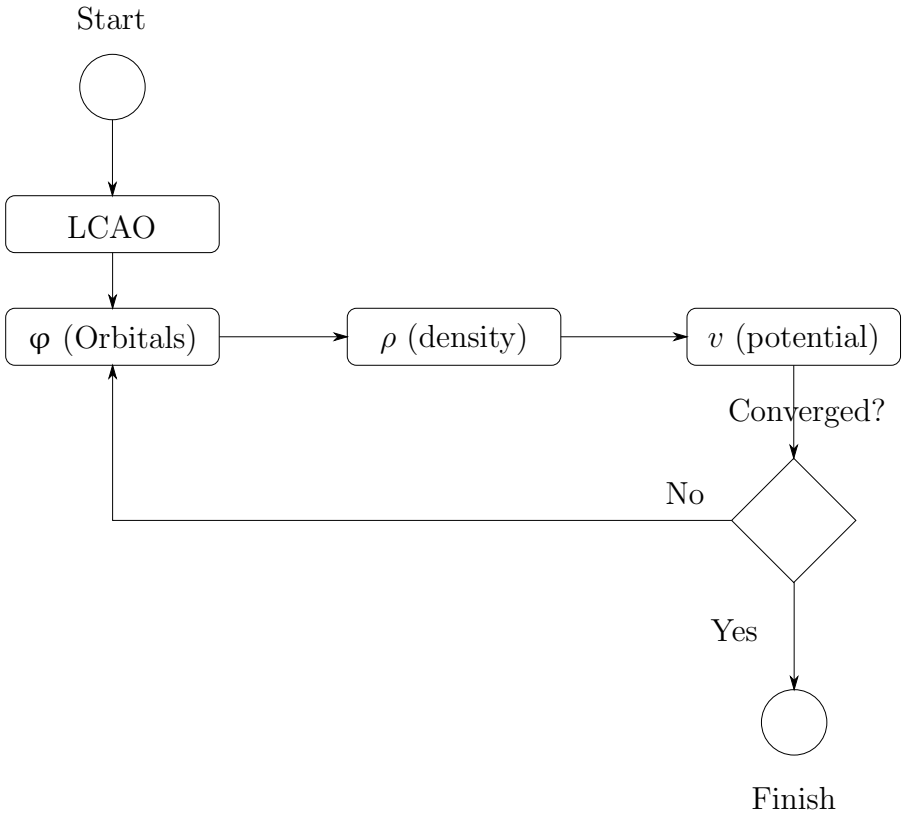


Figure 3.3: Simplified scheme of a Self Consistent Field (SCF) cycle.

As alternative to the Scrödinger equation (3.6), the approximation given by TDDFT is good enough.

Even if it is difficult to achieve the electronic density of an interactive system, Kohn and Sham approximated it with a density of non-interactive system. Thus, they create much simpler description that can be easily solved numerically.

DFT can be viewed as a re-writing of the Schrödinger equation for a many-electron system in terms of its electronic density. It is therefore a fully *ab-initio* approach. TDDFT is thus formally equivalent to the Time-Dependent Schrödinger equation, and is used to model the evolution in time of any quantum system. TDDFT has proven to be quite accurate even for biological chromophores, as the results obtained for fluorescent proteins [2-4], the Genji-Botaru luciferin [5], and other biological systems (flavins, porphyrins, DNA, etc. [6]) show. Low-energy transitions between bound states are usually quite well described with TDDFT, the error being usually smaller than a few tenths of an eV. A few problems remain, most notably charge-transfer excitations, and transitions to weakly bound states like Rydberg states. Nevertheless, when properly validated, TDDFT calculations can be quite reliable, and are increasingly used by non-experts to support and interpret experimental results. Important reasons for the success of TDDFT in photochemistry are its cost/performance ratio which is unmatched by traditional methods and the relatively wide applicability range.

The process of light absorption (and emission) is quantum mechanical, and it is not therefore accessible using the classical mechanics techniques usually applied in biochemistry. This implies that the calculation of the absorption spectrum of a par-

ticular molecule has to be performed with *ab-initio* methods. For small molecules, these methods can be quantum-chemistry based or one can use TDDFT. For large molecules the only viable option is the latter.

3.4 LCAO

The nature of the wavefunctions is in principle unknown. For that reason, we have to approximate them. One such technique is the Linear Combination of Atomic Orbitals (LCAO). LCAO is the assumption that the systems solution is the sum of the solutions of its parts. One can argue that the approximation made by solving independent particles (electrons in our case) is good enough to get an idea of the atomic system. Molecular ortitals describe the "behaviour" of a molecule, exactly the same way the atomic orbitals for atoms. Starting with atomic wavefunctions, the molecular wavefunctions are constructed. So, the construction of a ϕ as a linear combination of atomic wavefunctions φ, is knows as Linear Combination of Atomic Orbitals (LCAO) [61], and it is denoted as:

$$\phi = \sum_{i=1}^{N} a_i \varphi_i \tag{3.12}$$

These guesses of the wavefunctions is used to initialise a DFT run.

3.5 Poisson solver

The electrostatic interaction is a key part in the DFT and TDDFT formulations of quantum mechanics. In DFT the many-body Schrödinger equation is replaced by a set of single particle equations, the Kohn-Sham equations, which include an effective potential that contains the electronic interaction. Such effective potential is usually divided into three terms: the Hartree potential, the Exchange-Correlation (XC) potential, and the external potential. The Hartree term corresponds to the classical electrostatic potential generated by the electronic charge distribution.

In the context of quantum mechanics, the electrons and their electric charge are delocalised over space forming a continuous charge distribution $\rho(\vec{r})$. Such a charge density creates an electrostatic (Hartree) potential $v(\vec{r})$, which is given by [64]

$$v(\vec{r}) = \int \mathrm{d}\vec{r}\,' \frac{1}{4\pi\epsilon_0} \frac{\rho(\vec{r}\,')}{|\vec{r} - \vec{r}\,'|} \ , \tag{3.13}$$

where ϵ_0 is the electrical permittivity of the vacuum, $1/4\pi$ in atomic units. When considering the electric interaction in a medium, it is possible to approximate the polarization effects by replacing ϵ_0 by an effective permittivity, ϵ. In 3D, it is simple to show that equation (3.13) is equivalent to the Poisson equation [65, 66]

$$\nabla^2 v(\vec{r}) + \frac{\rho(\vec{r})}{\epsilon_0} = 0 \ . \tag{3.14}$$

In fact, this equation provides a convenient general expression that is valid for different dimensions and boundary conditions. For example, to study crystalline systems, when periodic boundary conditions are usually imposed. It is also possible to simulate a molecular system interacting with ideal metallic surfaces by choosing the appropriate boundary conditions [67, 68]. Both formulations of the problem,

i.e. equations (3.13) and (3.14), are quite useful: while some methods to calculate the Hartree potential are based on the former, others rely on the latter.

In order to numerically calculate the electrostatic potential we need to discretise the problem. To this end, we use a mesh representation, which changes the charge density and the electrostatic potential to discrete functions, with values defined over a finite number of points distributed over a mesh. Such an approach is used in many electronic structure codes, even when another type of representation is used for the orbitals. The direct calculation of the potential would take $\mathcal{O}(N^2)$ operations, N being the total number of mesh points. This is prohibitive for systems beyond a given size. Fortunately, there exist a variety of methods — either using equation (3.13) or (3.14) — that by exploiting the properties of the problem, reduce the cost to a linear or quasi-linear dependency with a negligible accuracy drop. In this work, we have selected several parallel implementations of some of the most popular of these methods (Fast Fourier Transforms, Interpolating Scaling Function, Fast Multipole Method, Conjugate Gradients, and Multigrid). In the next subsections we introduce these methods and give a brief account of their theoretical foundations and properties.

3.5.1 Fast Fourier Transforms

The Fourier Transform (FT) is a powerful mathematical tool both for analytical and numerical calculations, and certainly it can be used to calculate the electrostatic potential created by a charge distribution represented in an equispaced mesh by operating as follows. Let $f(\vec{r})$ be a function whose Fourier transform $\hat{f}(\vec{k})$ exists. By construction $f = f(\vec{r})$, and $\hat{f}^{-1}\big(\hat{f}(\vec{k})\big)(\vec{r}) = f(\vec{r})$. This expression, plus the convolution property of the Fourier transform, applied to equation (3.13), imply that

$$v(\vec{r}) = \hat{v}^{-1}\big(\hat{v}(\vec{k})\big)(\vec{r}) = \frac{1}{4\pi\epsilon_0}\hat{v}^{-1}\big(\hat{\rho}(\vec{k})/|\vec{k}|^2\big) \ , \tag{3.15}$$

where we have used that the Fourier transform of the function $1/|\vec{r}|$ is $\widehat{(1/|\vec{r}|)}(\vec{k}) = 1/|\vec{k}|^2 = 1/(k_x^2 + k_y^2 + k_z^2)$ [66].

Since $\rho(\vec{r})$ is represented in discrete equispaced points $(r_{j,k,l})$ at the centre of meshes whose volume equals Ω, the Fourier transform of $\rho(\vec{r})$, i.e. $\hat{\rho}(\vec{k})$, can be calculated using its discrete Fourier transform

$$\hat{\rho}(\vec{k}) := (2\pi)^{-3/2} \int \mathrm{d}\vec{r}\, \exp(-i\vec{k}\cdot\vec{r})\rho(\vec{r}) \tag{3.16}$$

$$\simeq (2\pi)^{-3/2}\Omega \sum_{j,k,l} \rho(r_{j,k,l}) \exp(-i(k_x j + k_y k + k_z l)) \ . \tag{3.17}$$

The use of equation (3.16) in equation (3.15) results in a discretised problem, which requires the use of a discrete Fourier transform plus an inverse discrete Fourier transform. The expression of the potential in terms of discrete Fourier transforms allows the application of the efficient FFT technique [69], so the problem can be solved in $\mathcal{O}(N \log_2 N)$ steps.

It is to be stressed that the use of the FT automatically imposes periodic boundary conditions on the density, and therefore to the potential. When finite systems

are studied, some scheme is required to avoid the interaction between periodic images. A simple way to solve this is to increase the size of the real-space simulation mesh and set the charge density to zero in the new points. This moves the periodic replicas of the density away, thus decreasing their effect on the potential [66, 70]. Another strategy is to replace the $1/(\epsilon_0 \vec{k}^2)$ factor of equation (3.15), known as the kernel of the Poisson equation, by a quantity that gives the free space potential in the simulation region. This modified kernel has been presented in reference [71] for molecules, one-dimensional systems, and slabs. This Coulomb cut-off technique is very efficient and easy to implement. Therefore, we will consider fast Fourier transform method that combines the two approaches, doubling the size of the mesh and using a modified kernel [66, 71]. This results in a potential that accurately reproduces the free space results. On the other hand, this method also imposes some constrains. For example, in the case of molecules, the cut-off is spherical and, therefore, it requires the enlarged mesh to be always cubic, regardless of the shape of the original mesh.

The power of the FFT method has made it part of many algorithms for the calculations of pairwise interactions. In the method summarised in this subsection, we treat the whole contribution to the potential with FFT.

In one dimension, the discrete Fourier transform of a set of n complex numbers f_k (f_k can be, for example, the values of ρ in a set of discrete points) is given by

$$F_l = \sum_{k=0}^{n-1} f_k \exp\left(-2\pi i \tfrac{kl}{n}\right) \qquad \text{for } l = 0, \ldots, n-1 \ , \tag{3.18}$$

with i being the imaginary unit. The inverse discrete Fourier transform is given by

$$f_k = \frac{1}{n} \sum_{l=0}^{n-1} F_l \exp\left(+2\pi i \tfrac{kl}{n}\right) \qquad \text{for } k = 0, \ldots, n-1 \ . \tag{3.19}$$

The definitions above enable computational savings using the fact that both the input and output data sets ($\rho(\vec{r})$ and $v(\vec{r})$) are real. Thus $F_{n-l} = F_l^*$, and we just need to calculate one half of the n discrete Fourier transforms.

Solving the Poisson problem using equations (3.18) and (3.19) would require $\mathcal{O}(N^2)$ arithmetic operations (with e.g. $N = n^3$), which would not represent any improvement over the cost of evaluating the potential directly. However, in 1965, J. W. Cooley and J. W. Tukey published an algorithm called Fast Fourier Transforms (FFT) [69] that exploits the special structure of equation (3.18) in order to reduce the arithmetic complexity. The basic idea of the radix-2 Cooley-Tukey FFT is to split a discrete FT of even size $n = 2m$ into two discrete FTs of half the length; e.g.,

for $l = 0, \ldots, m-1$ we have

$$\begin{aligned} F_{2l} &= \sum_{k=0}^{m-1} f_k \exp\left(-2\pi i \tfrac{k2l}{n}\right) + f_{k+m} \exp\left(-2\pi i \tfrac{(k+m)2l}{n}\right) \\ &= \sum_{k=0}^{m-1} (f_k + f_{k+m}) \exp\left(-2\pi i \tfrac{kl}{m}\right) \ ; \end{aligned} \tag{3.20}$$

$$\begin{aligned} F_{2l+1} &= \sum_{k=0}^{m-1} f_k \exp\left(-2\pi i \tfrac{k(2l+1)}{n}\right) + f_{k+m} \exp\left(-2\pi i \tfrac{(k+m)(2l+1)}{n}\right) \\ &= \sum_{k=0}^{m-1} \exp\left(-2\pi i \tfrac{k}{n}\right) (f_k - f_{k+m}) \exp\left(-2\pi i \tfrac{kl}{m}\right) \ . \end{aligned} \tag{3.21}$$

If we assume n to be a power of two, we can apply this splitting recursively $\log_2 n$ times, which leads to $\mathcal{O}(n \log_2 n)$ arithmetic operations for the calculation of equation (3.18). There exist analogous splitting for every divisible sizes [72], even for prime sizes [73].

3.5.2 Interpolating Scaling Function

This method was developed by Genovese *et al.* [14] for the BigDFT code [74]. Formally it is based on representing the density in a basis of Interpolating Scaling Functions (ISF) that arise in the context of wavelet theory [75]. A representation of ρ from $\rho_{j,k,l}$ can be efficiently built by using wavelets, and efficient iterative solvers for the Hartree potential v can be tailored, e.g. from ordinary steepest descent or conjugate gradient techniques [76]. A non-iterative and accurate way to calculate v [14] is to use the Fast Fourier Transforms (FFT) in addition to wavelets. If we represent

$$\rho(\vec{r}) = \sum_{j,k,l} \rho_{j,k,l} \phi(x-j) \phi(y-k) \phi(z-l) \ , \tag{3.22}$$

where $\vec{r} = x, y, z$ and ϕ are Interpolating Scaling Function in one dimension, then equation (3.13) becomes

$$v_{m,n,o} := v(\vec{r}_{m,n,o}) = \sum_{j,k,l} \rho_{j,k,l} K(j,m;k,n;l,o) \ , \tag{3.23}$$

where

$$K(j,m;k,n;l,o) := \int_V \mathrm{d}\vec{r}\,' \frac{\phi_j(x') \phi_k(y') \phi_l(z')}{|\vec{r}_{m,n,o} - \vec{r}\,'|} \ , \tag{3.24}$$

and V indicates the total volume of the system. Due to the definition of the ISF, the discrete kernel K satisfies $K(j,m;k,n;l,o) = K(j-m;k-n;l-o)$, and therefore equation (3.23) is a convolution, which can be efficiently treated using FFTs (see the previous subsection). The evaluation of equation (3.24) can be approximated by expressing the inverse of r in a Gaussian basis ($1/r \simeq \sum_k \omega_k \exp(-p_k r^2)$), which also enables efficient treatment. All this makes the order of this method $N \log_2 N$. Another important characteristic of the method is that it uses a kernel in equation (3.15) that yields an accurate free space potential without having to enlarge the mesh.

3.5.3 Fast Multipole Method

The Fast Multipole Method (FMM) was first proposed in 1987 for 2D systems [77], and it was soon extended to 3D problems [78]. Although only $\mathcal{O}(N)$ operations are necessary to calculate the electrostatic potential, the first FMM versions had big prefactors that in practice made the method competitive only for huge systems or low accuracy calculations [79]. After thorough research, it was possible to develop signficantly more accurate and efficient versions of the FMM [80, 81], making it a largely celebrated method [82].

The original FMM was devised to calculate the potential generated by a set of discrete point-like particles

$$v(\vec{r}_i) = \frac{1}{4\pi\epsilon_0} \sum_{j=1, j\neq i}^{N} \frac{q_j}{|\vec{r}_i - \vec{r}_j|} \ , \tag{3.25}$$

which is different from the charge-distribution problem that we are studying in this work. While extensions of the FMM to the continuous problem exist [83, 84, 85], in order to profit from the efficient parallel FMM implementations we have devised a simple scheme to recast the continuous problem into a discrete charge one without losing precision.

We assume that each mesh point $r_{j,k,l}$ corresponds to a charge of magnitude $\Omega\rho_{j,k,l}$, where $\Omega = h^3$ (h being the mesh spacing) is the volume of the space associated to each mesh point (mesh volume). Using the FMM we calculate at each point the potential generated by this set of charges, $v^{\mathrm{FMM}}_{j,k,l}$. However, this is not equivalent to the potential generated by the charge distribution, and some correction terms need to be included (see Appendix D.IV for the derivation of these corrections). The first correcting term (self-interaction term) comes from the potential generated at each point by the charge contained in the same mesh

$$v^{\mathrm{SI}}_{j,k,l} = 2\pi \left(\frac{3}{4\pi}\right)^{2/3} h^2 \rho_{j,k,l} \ . \tag{3.26}$$

Additionally, we apply a correction to improve the accuracy of the interaction between neighbouring points, which has the largest error in the point-charge approximation. This correction term is derived using a formal cubic interpolation of the density to a finer mesh, obtaining a simple finite-differences-like term

$$\begin{aligned} v^{\mathrm{corr.}}_{j,k,l} &= h^2\big(27/32 + (\alpha)2\pi(3/4\pi)^{2/3}\big)\rho_{j,k,l} \\ &+ (h^2/16)\big(\rho_{j-1,k,l} + \rho_{j+1,k,l} + \rho_{j,k-1,l} + \rho_{j,k+1,l} + \rho_{j,k,l-1} + \rho_{j,k,l+1}\big) \\ &- (h^2/64)\big(\rho_{j-2,k,l} + \rho_{j+2,k,l} + \rho_{j,k-2,l} + \rho_{j,k+2,l} + \rho_{j,k,l-2} + \rho_{j,k,l+2}\big) \ . \end{aligned} \tag{3.27}$$

Here α is a parameter to compensate the charge within the mesh (j, k, l) that is counted twice. The final expression for the potential is

$$v_{j,k,l} = v^{\mathrm{FMM}}_{j,k,l} + v^{\mathrm{SI}}_{j,k,l} + v^{\mathrm{corr.}}_{j,k,l} \ . \tag{3.28}$$

Now we give a brief introduction of the FMM algorithm. More detailed explanations on the FMM can be found in Refs. [77, 78, 86, 80]. To introduce the method we use spherical coordinates in what remains of this subsection.

Consider a system of l charges $\{q_i,\ i = 1, \ldots, l\}$ located at points $\{(\tau_i, \alpha_i, \beta_i),\ i = 1, \ldots, l\}$ which lie in a sphere D of radius a and centre at $Q = (\tau, \alpha, \beta)$. It can be proved [80] that the electric field created by them at a point $P = (r, \theta, \phi)$ outside D is

$$v(P) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \frac{O_n^m}{(r')^{n+1}} Y_n^m(\theta', \phi') \ , \tag{3.29}$$

where $P - Q = (r', \theta', \phi')$ and

$$O_n^m = \sum_{i=1}^{l} q_i \tau_i^n Y_n^{-m}(\alpha_i, \beta_i) \ , \tag{3.30}$$

with $Y_n^m(\alpha, \beta)$ known functions (the spherical harmonics). If P lies outside of a sphere D_1 of radius $a + \tau$ (see Figure 3.4A) the potential of equation (3.29) can be re-expressed as

$$v(P) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} \frac{M_j^k}{r^{j+1}} Y_j^k(\theta, \phi) \ , \tag{3.31}$$

where

$$M_j^k = \sum_{n=0}^{j} \sum_{m=-n}^{n} \tag{3.32}$$
$$\frac{O_{j-n}^{k-m}\ i^{|k|-|m|-|k-m|} \sqrt{(j-n-k+m)!(j-n+k-m)!} \sqrt{(n-m)!(n+m)!}\ \tau^n\ Y_n^{-m}(\alpha, \beta)}{\sqrt{(j-k)!(j+k)!}} \ .$$

Note that the "entangled" expression of the potential of equation (3.25), in which the coordinates of the point where the potential is measured and the coordinates of the charge that creates the potential are together, has been converted to a "factorised" expression, in which the coordinates of the point where we measure the potential are in terms $(Y_j^k(\theta, \phi)/r^{j+1})$ that multiply terms (M_j^k) which depend on the coordinates of the charges that create the potential. It is this factorisation which enables efficient calculation of the potential that a set of charges creates at a given point by using the (previously calculated) expression of the potential created by this set of charges at other points.

If the set of l charges described above is located inside a sphere D_Q of radius a with centre at $Q = (\tau, \alpha, \beta)$, where $\tau > 2a$ (see Figure 3.4B), then equation (3.29) implies that the potential created by these charges inside a sphere D_0 of radius a centred at the origin is given by

$$v(P) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} L_j^k\ r^j\ Y_j^k(\theta, \phi) \ , \tag{3.33}$$

where

$$L_j^k = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \frac{O_n^m\ i^{|k-m|-|k|-|m|} \sqrt{(n-m)!(n+m)!} \sqrt{(j-k)!(j+k)!}\ Y_{j+n}^{m-k}(\alpha, \beta)}{(-1)^n\ \tau^{j+n+1}\ \sqrt{(j+n-m+k)!(j+n+m-k)!}} \ . \tag{3.34}$$

The evaluation of the equations above requires truncation of the infinite sums to a given order, which can be chosen to keep the error below a given threshold.

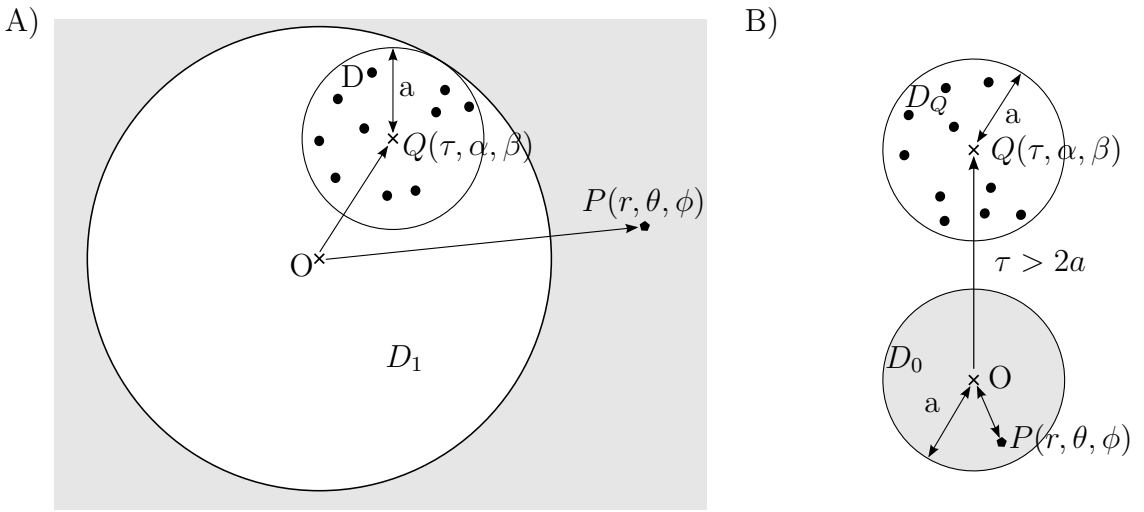


Figure 3.4: A) A set of point charges (black circles) inside a sphere D of radius a centred at $Q = (\tau, \alpha, \beta)$ creates a potential outside the sphere D_1 of radius $(a + \tau)$ and centred in the origin that can be expressed with equation (3.31)*. B) A set of point charges (black circles) inside a sphere D_Q of radius a centred at $Q = (\tau, \alpha, \beta)$ creates a potential inside the sphere D_0 of radius a and centred in the origin that can be expressed with equation* (3.33) *(provided that $\tau > 2a$). In both A) and B), O represents the origin of coordinates, and $P = (r, \theta, \phi)$ is the point where the potential is measured. In our systems, the charges lie in equispaced mesh points.*

The equations (3.31) and (3.33) enable the efficient calculation of the potential experienced by every charge of the system due to the influence of the other charges. In order to calculate it, the system is divided into a hierarchy of boxes. Level 0 is a single box containing the whole system; level 1 is a set of 8 boxes containing level 0; and so on (a box of level $\mathcal{L}$ consists of 8 boxes of level $\mathcal{L} + 1$). Different boxes at a given level do not contain common charges. The highest level ($\mathcal{N}_l$) contains several charges (in our case, each lying in a mesh point) in every box. The procedure to calculate the potential in all mesh points can be summarised as follows:

- For every box in the highest box level $\mathcal{N}_l$ (smallest boxes), we calculate the potential created by the charges in that box using equation (3.29).
- We gather 8 boxes of level $\mathcal{N}_l$ to form every box of level $\mathcal{N}_l$-1. We calculate the potential created by the charges of the $(\mathcal{N}_l - 1)$-box using the potentials created by the eight $(\mathcal{N}_l)$-boxes that form it. To this end, we use equation (3.31).
- We repeat this procedure (we use equation (3.31) to get the potentials created by box $\mathcal{L}$-1 by using those of box $\mathcal{L}$) until all the levels are swept.
- Then, the box hierarchy is swept in the opposite direction: from lower to higher levels, the equation (3.33) is used to calculate the potential created by the boxes (the potential given by equation (3.33) is valid in regions that are not equal to those where equation (3.31) is valid).
- Finally, the total potential in every mesh point ($v_{j,k,l}^{FMM}$) is calculated as an addition of three terms: the potentials due to nearby charges are directly

calculated with the pairwise formula of equation (3.25), and the potentials due to the rest of the charges are calculated either with equation (3.31) or with equation (3.33), depending on the relative position of the boxes which create the potential and the box where the potential is evaluated.

In the whole procedure above, the charge in the mesh point (j, k, l) is $\Omega\rho_{j,k,l}$. This scheme corresponds to the traditional version of FMM [80]. Later in this project, we will use a slight modification of it [16] which not only converts multipoles between consecutive levels, but also within every given level, which enables further computational savings.

3.5.4 Conjugate Gradients

In this subsection and in the following one we present two widely used iterative methods to calculate the electrostatic interaction: Conjugate Gradients (CG) and Multigrid [87]. These methods are based on finding a solution to the Poisson equation (3.14) by starting from an initial guess and systematically refining it so that it becomes arbitrarily closer to the exact solution. These two methods have the advantage that if a good initial approximation is known, only a few iterations are required.

When using a mesh representation, the Poisson equation can be discretised using finite differences. In this scheme the Laplacian at each mesh point is approximated by a sum over the values of neighbouring points multiplied by certain weights. High-order expressions that include several neighbours can be used to control the error in the approximation [88]. The finite-difference approximation turns equation (3.14) into a linear algebraic equation

$$\tilde{L}\vec{x} = \vec{y}\ , \tag{3.35}$$

where $\tilde{L}$ is a sparse matrix (taking advantage of the sparsity of a system of equations can greatly reduce the numerical complexity of its solution [89]), $\vec{y}$ is a known vector ($y = -\rho/\epsilon_0$, in this case) and $\vec{x}$ is the vector we are solving for, in this case the electrostatic potential. Equation (3.35) can be efficiently solved by iterative methods that are based on the application of the matrix-vector product without the need to store the matrix.

Since the matrix is symmetric and positive definite, we can use the standard CG [90] method. In the CG method, $\vec{x}$ is built as a linear combination of a set of orthogonal vectors $\vec{p_k}$. In every iteration, a new term is added

$$\vec{x}_{k+1} = \vec{x}_k + \alpha_k\vec{p}_k\ , \tag{3.36}$$

which attempts to minimise

$$f(\vec{x}) := \frac{1}{2}\vec{x}^T\tilde{L}\vec{x} - \vec{x}^T\vec{y}\ , \tag{3.37}$$

whose minimum is the solution of equation (3.35). The term added to the potential in iteration k is built so that $\vec{x}$ moves in the direction of the gradient of f, but being orthogonal to the previous terms. The gradient of $f(\vec{x})$ satisfies, $-\nabla f(\vec{x}) = \vec{y} - \tilde{L}\vec{x}$. Therefore the search direction in iteration $k+1$ is

$$\vec{p}_{k+1} = \vec{r}_{k+1} + \frac{|\vec{r}_{k+1}|^2}{|\vec{r}_k|^2}\vec{p}_k \ . \tag{3.38}$$

with $\vec{r}_{k+1} = \vec{r}_k - \alpha_k \tilde{L}\vec{p}_k$ and $\vec{p}_0 = \vec{r}_0 = \vec{y} - \tilde{L}\vec{x}_0$, being $\vec{x}_0$ arbitrary. The coefficient associated with each direction, α_k, is obtained from the minimization condition, yielding

$$\alpha_k = \frac{|\vec{r}_k|^2}{\vec{p}_k^T \tilde{L}\vec{p}_k} \ . \tag{3.39}$$

The equations above show that the CG method has a linear scaling ($\mathcal{O}(N)$) with the number of points N for a given number of iterations whenever the matrix $\tilde{L}$ has a number of non-zero entries per row that is much lower than N (as is the usual case for the Poisson equation). Bigger exponents in the scaling can appear, however, in problems where the number of performed iterations is chosen to keep the solution error below a given threshold that depends on N [91].

An issue that appears when using the finite-difference discretisation to solve the Poisson equation are boundary conditions: they must be given by setting the values of the points on the border of the domain. For free space boundary conditions we need a secondary method to obtain the value of the potential over these points; this additional method can represent a significant fraction of the computational cost and can introduce an approximation error. In our implementation we will use a multipole expansion. Such an expansion concerns not only the boundaries: all the charges of the system are decomposed into two contributions: the first contribution is obtained with using a multipole expansion, and the corresponding potential is analytically calculated [66]; the potential created by the rest of the charge is calculated numerically with either the Conjugate Gradients method or the Multigrid method. Splitting methods like this are commonly used in the literature [70].

3.5.5 Multigrid

Multigrid (MG) [92, 93, 94, 95, 96, 87] is a powerful method to solve elliptic partial differential equations, such as the Poisson problem [97], in an iterative fashion. MG is routinely used as a solver or preconditioner for electronic structure and other scientific applications [98, 99, 100, 101]. In this subsection we will make a brief introduction to a simple version of the MG approach that is adequate for the Poisson problem. MG, however, can also be generalised to more complex problems, like non-linear problems [96] and systems where there is no direct geometric interpretation, in what is known as algebraic Multigrid [102]. It has also been extended to solve eigenvalue problems [103, 100].

MG is based on iterative solvers like Jacobi or Gauss-Seidel [87]. These methods are based on a simple iteration formula, that for equation (3.35) reads

$$\vec{x} \leftarrow \vec{x} + M^{-1}(\vec{y} - \tilde{L}\vec{x}) \ . \tag{3.40}$$

The matrix M is an approximation to $\tilde{L}$ that is simple to invert. In the case of Jacobi, M is the diagonal of $\tilde{L}$, and for Gauss-Seidel, M is upper diagonal part of $\tilde{L}$. These methods are simple to implement, in particular in the case of the Laplacian operator, but are quite inefficient by themselves, so they are not practical

as linear solvers for high-performance applications. However, they have a particular property: they are very good in removing the high-frequency error of a solution approximation, where the frequency is defined in relation to the mesh spacing. In other words, given an approximation to the solution, a few iterations of Jacobi or Gauss-Seidel will make the solution smooth. In MG the smoothing property is exploited by using a hierarchy of meshes of different spacing, and hence changing the frequency that these smoothing operators can remove efficiently.

A fundamental concept in MG is the residual of a linear equation. If we have $\bar{x}$ as an approximate solution of equation (3.35), the associated residual, $\vec{b}$, is defined as

$$\vec{b} = \vec{y} - \tilde{L}\bar{\vec{x}} \; . \tag{3.41}$$

We can use the residual to define an alternative linear problem

$$\tilde{L}\vec{a} = \vec{b} \; . \tag{3.42}$$

Due to the linearity of the Laplacian operator, finding $\vec{a}$ is equivalent to solving the original linear problem, equation (3.35), as

$$\vec{x} = \bar{\vec{x}} + \vec{a}. \tag{3.43}$$

If a few iterations of a smoothing operator have been applied to the approximate solution, $\bar{x}$, we know that the high-frequency components of the corresponding residual $\vec{b}$ should be small. Then it is possible to represent $\vec{b}$ in a mesh that has, for example, two times the spacing without too much loss of accuracy. In this coarser mesh equation (3.42) can be solved with less computational cost. Once the solution $\vec{a}$ is found on the coarser mesh, it can be transferred back to the original mesh and used to improve the approximation to the solution using equation (3.43).

The concept of calculating a correction term in a coarser mesh is the basis of the MG approach. Instead of two meshes, as in our previous example, a hierarchy of meshes is used, at each level the residual is transferred to a coarser mesh where the correction term is calculated. This is done up to the coarsest level that only contains a few points. Then the correction is calculated and transferred back to the finer levels.

To properly define the MG algorithm it is necessary to specify the operators that transfer functions between meshes of different spacing. For transferring to a finer mesh, typically a linear interpolation is used. For transferring to a coarser mesh a so-called restriction operator is used. In a restriction operator, the value of the coarse mesh point is calculated as a weighted average of the values of the corresponding points in the fine mesh and its neighbours.

Now we introduce the Multigrid algorithm in detail. Each quantity is labelled by a super-index that identifies the associated mesh level, with 0 being the coarsest mesh and L the finest. We denote S^l as the smoothing operator at level l, which corresponds to a few steps (usually 2 or 3) of Gauss-Seidel or Jacobi. I_l^m represents the transference of a function from the level l to the level m by restriction or interpolation. Following these conventions, we introduce the algorithm of a MG iteration in Figure 3.5. Given an initial approximation for the solution, we perform a few steps of the smoothing operator, after which the residual is calculated and transferred

Multigrid *v*-cycle
Input: $\vec{y}, \bar{\vec{x}}$
Output: $\bar{\vec{x}}$

$\vec{y}^L \leftarrow \vec{y}$
$\vec{x}^L \leftarrow \bar{\vec{x}}$
for l from L to 0 **do**
  if $l \neq L$ **then**
    $x^l \leftarrow 0$ {Set initial guess to 0}
  end if
  $\vec{x}^l \leftarrow \mathrm{S}^l \vec{x}^l$ {Pre-smoothing}
  if $l \neq 0$ **then**
    $\vec{b}^l \leftarrow \vec{y}^l - \tilde{L}^l \vec{x}^l$ {Calculate the residual}
    $\vec{y}^{l-1} \leftarrow I_l^{l-1} \vec{b}^l$ {Transfer the residual to the coarser mesh}
  end if
end for
for l from 0 to L **do**
  if $l \neq 0$ **then**
    $\vec{x}^l \leftarrow \vec{x}^l + I_{l-1}^l \vec{x}^{l-1}$ {Transfer the correction to the finer mesh}
  end if
  $\vec{x}^l \leftarrow \mathrm{S}^l \vec{x}^l$ {Post-smoothing}
end for
$\bar{\vec{x}} \leftarrow \vec{x}^L$

Figure 3.5: Algorithm of a MG v-cycle.

to the coarser mesh. This iteration is repeated until the coarsest level is reached. Then we start to move towards finer meshes. In each step the approximate solution of each level is interpolated into the finer mesh and added, as a correction term, to the solution approximation of that level, after which a few steps of smoothing are performed. Finally, when the finest level is reached, a correction term that has contributions from the whole mesh hierarchy is added to the initial approximation to the solution.

The scheme presented in Figure 3.5 is known as a v-cycle, for the order in which levels are visited, some more sophisticated strategies exist, where the coarsest level is visited twice (a w-cycle) or more times before coming back to the finest level [96]. Usually a v-cycle reduces the error, measured as the norm of the residual, by around one order of magnitude, so typically several v-cycles are required to find a converged solution.

When a good initial approximation is not known, an approach known as Full Multigrid (FMG) can be used. In FMG the original problem is solved first in the coarsest mesh, then the solution is interpolated to the next mesh in the hierarchy, where it is used as initial guess. The process is repeated until the finest mesh is reached. It has been shown that the cost of solving the Poisson equation by FMG depends linearly with the number of mesh points [96].

Chapter 4

Software Package

Contents

In order to be able to efficiently carry out computer simulations, source code have to be carefully written and optimised. This project focuses in the usage and optimisation of the TDDFT software-package Octopus, and this chapter provides a description of it.

The simulation at a quantum level of systems consisting of thousands of atoms makes it possible to understand a wide variety of physical, chemical and biological phenomena. Despite the remarkable recent improvements in scientific codes and HPC infrastructures, the size of the systems that can be simulated using TDDFT is still very limited, and performing calculations with thousands of atoms is still a significant challenge.

It has been proven that the capacity of such theories to provide accurate results on the description of a big variety of phenomena at a relatively cheap computational cost; specially, TDDFT is being used to accurately predict, *ab-initio*, the optical absorption spectra of biological systems. When properly validated, TDDFT calculations can be quite reliable, and they are increasingly used by non-experts to support and interpret experimental results.

4.1 OCTOPUS code: main overview

OCTOPUS [2, 3, 4] is a very efficient scientific software package used to study by first principles the properties of the excited states of large biological molecules, complex nanostructures, and solids. The code is mostly developed for Density Functional Theory (DFT) and Time-Dependent Density Functional Theory (TDDFT) calculations, which are convenient quantum-mechanic approaches to study the electronic structure of molecular systems and its time evolution behaviour, as explained in the previous Chapter 3. It is released under the GPL license, so it is freely available to the whole scientific community for use, study and modification. The code has been created around 2000 in the group of professor Angel Rubio who, at that moment, was as the University of Valladolid and used and developed extensively (included also the University of the Basque Country UPV/EHU) in the last years to study systems up to hundreds of atoms.

Over the past years, OCTOPUS has evolved into a fairly complex and complete tool, and is now used by dozens of research groups around the world. Due to the open nature of the project, it is hard to measure the total number of users. However, an estimate can be made from the number of downloads (an average of over 200 downloads per month), and from the number of participants in the users mailing list (300 participants). Even in this short time-scale, there were a considerable amount of papers published or submitted by independent groups presenting calculations performed with OCTOPUS.

For several tasks the code relies on external libraries. For example, linear algebra operations are handled using the BLAS and LAPACK libraries, and the Poisson equation is solved using very efficient massively parallel libraries (either Interpolating Scaling Function (ISF) [14] or Parallel Fast Fourier Transforms (PFFT) [15]). Also the Laplacian of the states has to be evaluated for every mesh point. OCTOPUS calculates it by finite differences, usually using a *star-stencil* with 24 neighbours. Support for BLACS and SCALAPACK is also available, but it has not been optimised yet.

The code is written mainly in Fortran 95 (trying to be as close as possible to the object oriented paradigm), with some parts in C, and using API of OpenCL [104], OpenMP and MPI. Currently it consists of 200,000 lines of code (excluding external libraries).

Since the code is publicly available, it is essential to provide documentation so

users can learn how to use it. The Octopus website[1] contains a user manual and several tutorials that teach users how to perform different types of calculations, including some basic examples. Additionally, all input variables have their own documentation that can be accessed through the website or from a command line utility. A mailing list is also available, where users can get help with specific questions about Octopus from the developers or other users.

One of the most important points in developing a scientific code is to ensure the correctness of the results. When a new feature is implemented, the developers validate the results by comparing them with known results from other methods and other codes. To ensure that future changes do not modify the results, we use an automated system (Buildbot) that runs the code periodically and compares the results with reference data. A short set of tests is executed each time a change is made in Octopus while a long one is executed every day. The tests are run on different platforms and with different compilation options, to ensure that the results are consistent for different platforms. Users should also run the testsuite to validate the build on their machine before running real calculations.

To avoid users inadvertently using parts of the code that are being developed or have not being properly validated, they are marked as "Experimental." These experimental features can only be used by explicitly setting a variable in the input file. In any case, users are expected to validate their results in known situations before making scientific predictions based on Octopus results.

4.2 Physics of Octopus

Octopus is a code that simulates the dynamics of electrons and nuclei under the influence of Time-Dependent fields. The electronic degrees of freedom are treated quantum-mechanically within TDDFT, while the nuclei are considered to behave as classical point particles. In this code, all quantities are discretized in real-space using a uniform mesh, and the simulations are performed in real time.

The main equation to solve is the Time-Dependent Kohn-Sham equation; the initial condition is typically obtained solving a ground state density functional theory problem, also using Octopus. The main quantities to represent are three dimensional functions: the electronic density and the single particle orbitals (Kohn-Sham states, or just, states). For big systems these are the most memory demanding variables.

In the code the functions are represented in a real-space mesh, and differential operators are approximated by high-order finite differences. The propagation of the time-dependent Kohn-Sham equation is done by approximating the exponential of the Hamiltonian operator by a Taylor expansion.

Octopus is used to calculate the excited states of an atomic system. This excited state is calculated after solving the Ground State (GS). Also, this GS can be set up arbitrarily, and can be used, for instance to analyse the reactivity of the system, or the absorption... these latter simulations are called nonequilibrium physics. After solving the GS, different kind of perturbations may be applied (mechanical, electronic field, light, protons...) to the same initial state. In fact, real-time TDDFT is a versatile method to model the response of an electronic system (molecular or crys-

[1] http://tddft.org/programs/octopus/

talline [105]) to different kinds of perturbations. It is useful to calculate properties like optical absorption spectra [106, 107], non-linear optical response [108, 109], circular dichroism [110, 111], van der Waals coefficients [112], Raman intensities [113], etc. The numerical advantage of real-time TDDFT is that the propagator preserves the orthogonality of the states [114]. In practice, this allows us to propagate each one of the states in an independent manner, which is ideal for parallelisation. Since the method does not require expensive orthogonalisation steps, the numerical cost scales with the square of the size of the system and not cubically as many other methods [115].

4.2.1 Hamiltonian

In OCTOPUS never is build the Hamiltonian explicitly, since it is not very efficient to store the finite difference operator and the pseudopotential projectors as a matrix (even a sparse matrix). It is more efficient to apply them using a routine that knows about the details of the operator. For example, the coefficients for the finite difference operator are the same for all points, so for a typical problem we only need to store 25 values instead of the 25×npoints that a sparse-matrix representation will use.

OCTOPUS uses Kohn-Sham (KS) formulation of DFT, which leads to a coupled set of single-particle equations whose solution yields the many-body electronic density $n(\vec{r},t)$. For example, for the TD case these equations read (atomic units are used throughout this document)

$$i\frac{\partial}{\partial t}\varphi_i(\vec{r},t) = \left[-\frac{1}{2}\nabla^2 + v_{\text{ext}}(\vec{r},t) + v_{\text{Hartree}}[n](\vec{r},t) + v_{\text{xc}}[n](\vec{r},t)\right]\varphi_i(\vec{r},t) \tag{4.1}$$

$$n(\vec{r},t) = \sum_i^{\text{occ}} |\varphi_i(\vec{r},t)|^2 \tag{4.2}$$

where $\varphi_i(\vec{r},t)$ are the single-particle KS states (also called KS orbitals), $v_{\text{ext}}(\vec{r},t)$ is the time-dependent external potential that can be the potential generated by the nuclei, a laser field, etc.; $v_{\text{Hartree}}[n](\vec{r},t)$ is the Hartree potential that describes the classical mean-field interaction of the electron distribution; and $v_{\text{xc}}[n](\vec{r},t)$ is the Exchange-Correlation (XC) potential that includes all non-trivial many-body contributions.

It is true that the KS equations are an exact reformulation of quantum mechanics, both for Time-Independent and for Time-Dependent. However, the exact form of the XC functional is unknown and, therefore, has to be approximated in any practical application of the theory. In OCTOPUS different approximations for this term are implemented, from local and semi-local functionals to more sophisticated orbital dependent functionals, including hybrids [116] and the optimised effective potential approach [117]. Asymptotic correction methods are also implemented [118]. The used local and local and semi-local XC functionals are implemented in a separate library, Libxc [119]. Currently it contains around 180 functionals for the exchange, correlation, and kinetic energies belonging to the local-density approximation (LDA), the generalised-gradient approximation (GGA), the meta-GGA, and hybrid functional families. Functionals for systems of reduced dimensionality (1D and 2D) are also included.

4.3 Meshes

Octopus uses a real-space mesh discretisation to represent fields such as the Kohn-Sham states and the electronic density. Each function is represented by its value over an array of points distributed in real-space and, internally, it is saved in a 1D vector. Differential operators are approximated by high-order finite-difference methods [88] while integration is performed by a simple sum over the mesh point coefficients.

The real-space mesh approach does not impose a particular form for the boundary conditions, so it is possible to model both finite and periodic systems directly. Moreover, in Octopus the mesh boundaries can have an arbitrary shape, avoiding unnecessary mesh points. For example, for molecular calculations the default box shape corresponds to the union of spheres centred around the atoms (see Figure 4.1 for an example).

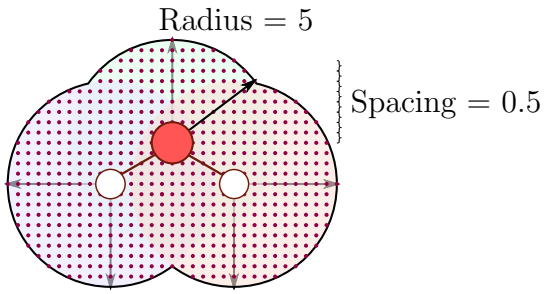


Figure 4.1: Simple 2D Octopus adaptive mesh with a radius of 5 and a spacing of 0.5.

One of the main advantages of the real-space mesh approach is that it is possible to systematically control the quality of the discretisation. By reducing the spacing and increasing the size of the box, the error is systematically decreased, and can be made as small as desired, at the cost of an increased computational cost. This is of particular significance for response properties [120, 121].

While the real-space scheme results in a large number of discretisation coefficients when compared with localised basis set representations, the discretised Hamiltonian is very sparse. In fact, the number of non-zero components depends linearly on the number of coefficients. Moreover, the Hamiltonian only requires information from near neighbours, which is advantageous for parallelisation and optimisation.

Finally, since the description of the core regions is expensive with a uniform-resolution discretisation, in Octopus the ion-electron interaction is usually modelled using norm-conserving pseudopotentials [122]. At the moment, the code can read pseudopotentials in several formats: the Siesta format [123], the Hartwigsen-Goedecker-Hutter format [124], the Fritz-Haber format [125] and its Abinit version [126], and the Quantum Espresso universal pseudopotential format [127]. Relativistic corrections, like spin-orbit coupling, can also be included by using relativistic pseudopotentials[128, 129]

4.4 Parallelisation of Octopus

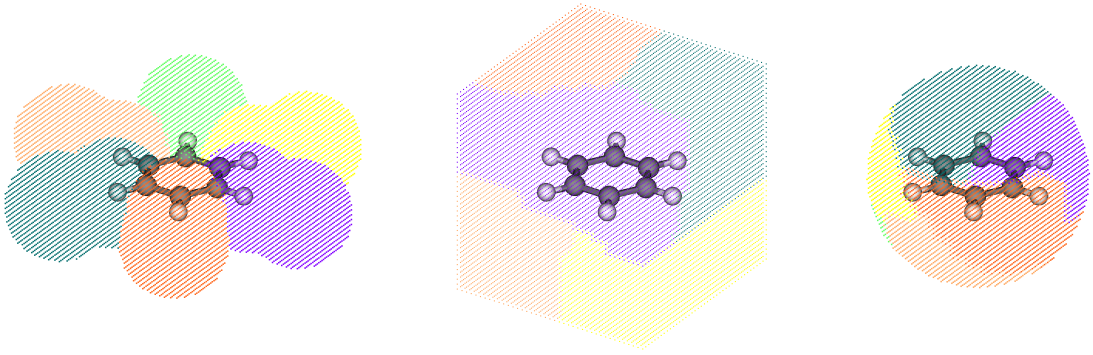
Octopus simulations involve a huge amount of computation and memory when obtaining electronic properties of molecules with a big number of atoms. Thus, a

parallel version of Octopus must be used to obtain precise results with realistic size systems in a reasonably bounded time. Octopus applies multi-level parallelisation, using MPI for message-passing among nodes, and OpenMP for intranode (core) shared memory parallel computation. It can also take advantage of GPU architectures. There are three levels of parallelisation that are relevant for the kind of systems we want to study:

- First, the real-space is divided in domains assigned to different processors. The application of differential operators requires the boundary regions to be communicated. This is done asynchronously, overlapping computation and communication. The scaling of this strategy is limited by the number of points in the mesh which increases linearly with the system size.

  The mesh is the data-structure used to represent the space. The real-space has to be bounded and discretised, and it can be done in one, two or three dimensions. On top of this mesh are represented such things as the potential energy (ν), the electronic density of the system (ρ), the system wavefunction, etc. Technically, the observables are implemented in a 1D vector, from 1 to mesh%np in every running process. This vector also runs globally with indexes from 1 to mesh%np_global (but this is hardly build during the execution).

  The mapping between those two representations (3D global to 1D internal) is done with the `lxyz` vector in one direction and with `lxyz_inv` in the other. The `lxyz` vector is a 2D vector, where the first index the global internal point ($\mathbb{N}$ <= mesh%np_global) and the second index is the chosen Cartesian index ($x = 1, y = 2, z = 3$); the resulting value will be the Cartesian point. To do the opposite mapping there is the `lxyz_inv` 3D vector, each dimension Cartesian index. From mesh 3D indexes (x, y, z), the corresponding internal point is obtained.



a) Adaptive b) Parallelepiped c) Spherical

Figure 4.2: Different mesh partitions for the benzene molecule. The mesh is partitioned in 6 domains using adaptive, parallelepiped and spherical shapes.

The shape of the mesh data-structure can be *adaptive*, parallelepiped or spherical (Figure 4.2). In the former case (which is the default) the mesh is made by an union of spheres centred in the atoms of the simulated system; in the other two cases the shape is regular: a parallelepiped and a sphere, respectively. Two parameters define the mesh: the radius (from each atom in the

adaptive case; the length of each edge in the parallelepiped case; and from the centre in the case of the sphere, an example can be found in Figure 4.1) and the spacing, which is the distance between two consecutive mesh points. All the mesh points are usually taken to be equally spaced.

- Then, the processors are divided in groups, each of one gets assigned a number of states (orbital). This is a very efficient scheme, since the propagation of each orbital is almost independent. The limitation to scalability is given by the number of available states, but this number increases, as in the previous case, linearly with the system size. An example of those independent orbitals can be found in Figure 4.3

  A state is the representation of an electronic orbital. Each one of the electronic states under study is defined by a concrete data-structure, which is represented over the mesh.

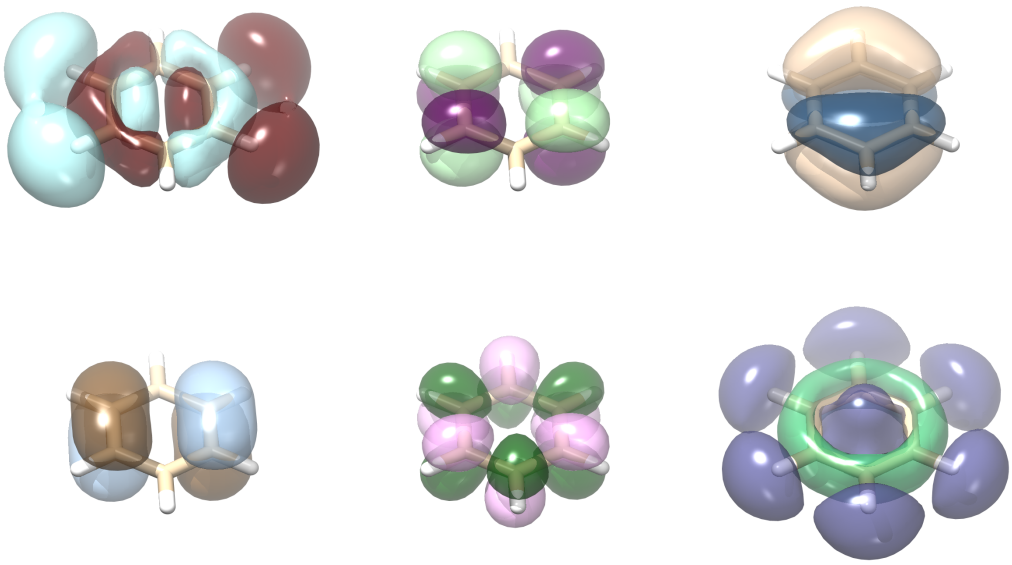


Figure 4.3: Example of different atomic orbitals (Kohn-Sham state) for a benzene molecule, which can be calculated mainly independently.

- Also, an auxiliary data-structure, called cube, is used to to solve Fast Fourier Transforms (FFT), which are needed to calculate the electrostatic Hartree potential. The evaluation of FFTs in 3D demands to represent the data in parallelepiped meshes. In addition, the cutoff technique [71] used in Octopus to reduce the effect of the undesired periodic images of the inverse FFT also requires cubic meshes.

  The edge of the cube is by default twice the length of the largest axis of the mesh; thus, it is at least 8 times bigger than the corresponding mesh. The cube is filled with the mesh points (explained in the next section), and all the extra points are padded with zeros.

- Finally, each process can run several OpenMP threads. The scalability is limited by regions of the code that do not scale due to limited memory bandwidth.

- An additional parallelisation level also exists. For independent particles, the Schrödinger Hamiltonian can be exactly partitioned according to the k-point and spin labels. This means that each one of the subproblems, i.e. for each k-point and spin label, can be solved independently of the others, reducing thereby the dimension of the Hamiltonian and the computational complexity. Mathematically, in (Time-Dependent) DFT this is no longer true as the subproblems are mixed by the density. However, a large part of the numerical solution can still be partitioned effectively, making the parallelisation in k-points and spin very efficient as little communication is required. Such a scheme does not always help for scaling, unfortunately. For example, for finite systems only a single k-point is used and in many cases we are interested in spin-unpolarised calculations. In this extreme case, there is absolutely no advantage in this parallelisation level.

As the size of the system grows, two factors affect the computing time: first, the space region to simulate is larger, and second, there are more electrons to simulate (which is directly related to the number of electronic states). By dividing each of these degrees of freedom among processors, multi-level parallelization ensures that the total parallel efficiency remains constant as we increase the system size and the number of processors.

A lower bound of the memory requirements of the simulation can be estimated from the number of simulated states $\times$ number of mesh points.

Thanks to its current parallel capabilities, Octopus was chosen as a benchmark code for the Partnership for Advanced Computing in Europe (PRACE) initiative. This means, as shown, that for the execution of this project the researcher has access to several supercomputing systems, including prototype machines, and has collaborated with experts of supercomputing centres.

4.5 Execution modes

Octopus simulates physical systems basically in two *phases*: firstly, the electronic Ground State is calculated, and secondly, the converged Kohn-Sham wave-functions are propagated in time under the effect of an external perturbation. The Ground State (GS) is obtained applying Density Functional Theory (DFT), whereas the TDDFT theory is used to obtain the Time-Dependent (TD) solution.

Memory needs increase (roughly) quadratically with the system size under simulation. Generally, the GS requires the use of real numbers for quantum states, while the TD runs require the use of complex numbers. Thus, the amount of memory in a time-dependent run usually doubles that of the corresponding Ground State. On the other hand, total computation amount (flop/s) does not increase in the same way for the two modes: the Ground State increases roughly as $S^{5/2}$, while the Time-Dependent increases as $S^{3/2}$ (S is the system size, i.e., the number of atoms).

These two run modes scale differently with respect to the number of processes, and pose different problems. More precisely, the Time-Dependent run mode scales better than the Ground State calculation, as we will see in the next chapters. The different scalabilities are due to the specific parallelisation schemes available in each run mode.

4.6 Profiling in Octopus

Profiling techniques and infrastructure are already embedded inside the Octopus code. The main functions of Octopus could be profiled and the user has only to change a variable in the input file, in order to have profiling results.

To profile itself, there are some auxiliary functions that are called twice in every important function: one at the beginning and other at the end. When the program is close to finish, some statistics are calculated, including the time between those two calls. Profiling output is shown in text files and one file is generated for every MPI process. Exhaust statistics could be taken from this huge amount of information.

We can demonstrate that the ad-hoc profiler is good enough at estimating the memory usage. Assuming that the result of the Valgrind Massif profiler [130] is acceptable, Octopus shows only a small discrepancy (which can also be justified). Internal profiler makes a tiny underestimation because it can not take into consideration memory allocated by the external libraries.

Chapter 5

First experiments and obtained results

Contents

In this chapter we will explain the initial tests we have made to analyse the behaviour of the parallel execution of Octopus, and to get the speed-ups for different problem sizes. As the final objective is to be able to efficiently execute Octopus using tens or hundreds of thousands of processors, we need to do previously a careful analysis of the executions, varying the number of processors and the size of the simulated physical systems.

As mentioned in Chapter 4, OCTOPUS simulations are divided in two parts: first, a Ground State (GS) must be obtained, and, then, this initial state is used to perform a Time-Dependent (TD) run. For the GS calculation, two types of executions have been developed: a GS test, to explore the more adequate number of computing nodes for the atom system under examination, that involves a few iterations; and the full GS calculation, an iterative process that obtains the GS, the starting point of any Time-Dependent simulation. Unfortunately, this iterative process has resulted a very time-consuming process, specially for the bigger systems. But the most relevant executions for this type of research are the TD ones, and, as we will see, the obtained speed-ups are very good, and we will try to go further, identifying problems and finding solutions.

The simulated systems are chunks of the LHC–II molecule. Different computers and number of computing nodes have been used to obtain the parallel speed-ups.

5.1 Previous tests

Before this project had started, other tests were made to analyse the behaviour of the parallel OCTOPUS code. The starting point was encouraging enough to go ahead, like Figure 5.1 shows.

For that previous tests to analyse the parallel scalability of the method in High Performance Computing architectures, a smaller part of the LHC–II molecule was simulated, composed of 650 atoms. The calculation of the absorption spectra required approximately 12 hours in 512 cores in Vargas. The scaling for this calculation in two different supercomputers —MareNostrum II and Vargas— can be seen in Figure 5.1.

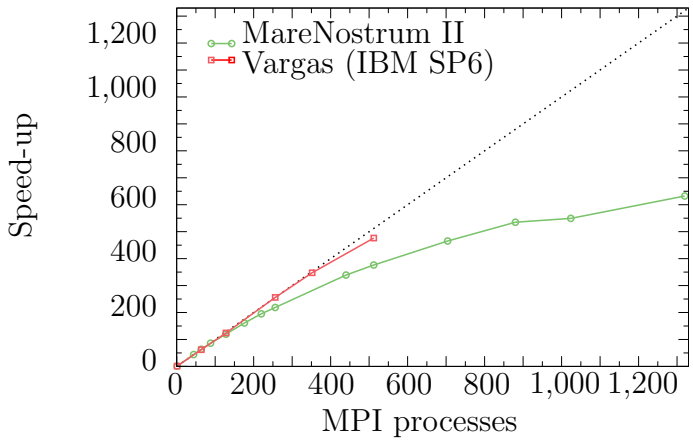


Figure 5.1: OCTOPUS *speed-up on previous tests (done by X. Andrade, 2009).*

The growth of the computational cost is quadratic with the number of atoms, since both the number of states and the size of the domains increase linearly. So it was expected the complete system (around 10 times bigger) to require around 100 times more CPU time, and, since the parallelisation is over the two degrees of complexity (domains and states), they expected to maintain the level of parallel efficiency over an equivalent increment of two orders of magnitude in the number of processes.

Considering that the complete system has 10,000 Kohn-Sham states, if we allocate 10 states per process and divide the real space in 128 or 256 domains, we

could expect the implementation in OCTOPUS to scale reasonably beyond 100,000 processes. This makes it an ideal case for a large massively parallel supercomputer.

5.2 Computers and atomic system sizes

The machines used for this study are Vargas, MareNostrum II, Jugene and Ganbo (presented in Chapter 2, Section 2.2.3).

The computer architecture and its configuration determine the sizes of the atomic systems we can simulate. These atomic systems are chunks of the LHC–II molecule, and they are composed of 180, 441, 650, 1365, 2676 and 5879 atoms. To simulate them, the main constraint is the memory per node that is available, that limits the maximum size of the system. So, depending on the computer we use, we have tested different sizes of atomic systems. A resume of the executed systems can be found in Table 5.1. At this beginning point, we had problems converging systems bigger than 1365 atoms, so we could only do TD and GS calculations and speed-up measurements for equal or smaller systems.

		No. of atoms					
		180	441	650	1365	2676	5879
Machine	Ganbo	yes	yes	-	-	-	-
	Vargas	yes	yes	yes	-	-	-
	MareNostrum II	yes	yes	yes	-	-	-
	Jugene	yes	yes	yes	yes	-	-

Table 5.1: Computers and atomic system sizes.

We have, also, another limit: the size of jobs we could reserve in the machines. For example, the minimum size of the partition to use in Jugene is 32 nodes, since it is a HPC machine for parallel jobs. Each node has 4 cores, so the minimum number of cores can be reserved is 128. This force us to normalise speed-ups to this number.

5.3 Results of the experiments

A great amount of tests have been done in within the initial stage of this project to explore the use of new computers and higher atomic systems. We will summarise all the tests here; all the remaining explanations can be found in appendix C.II.

The tests are divided into two main groups: the Ground State calculation (GS), and the Time-Dependent (TD) simulations. The results of a Ground State must be calculated before starting different TD simulations. In fact, TD simulations are the physically interesting objectives of OCTOPUS.

5.3.1 The Ground State calculations

First of all, we have to measure how long take the GS executions and how they scale with the number of processes. Once the GS tests are done, real calculation of the GS, until convergence, have to be done.

The GS calculation is an iterative process that obtains the basic state of an atomic system. But before starting those iterations, some initialisations are required, to create the mesh, to reserve memory, to prepare the MPI environment... So, the overall execution time is the sum of both parts (initialisations + iterations). In fact,

this overall time is not very meaningful, because: (1) it changes only a bit with different number of processes, and (2) the initialisation has a big influence, as we have limited the GS test to only a few iterations (Figure 5.2). So, we have focused in the iteration times.

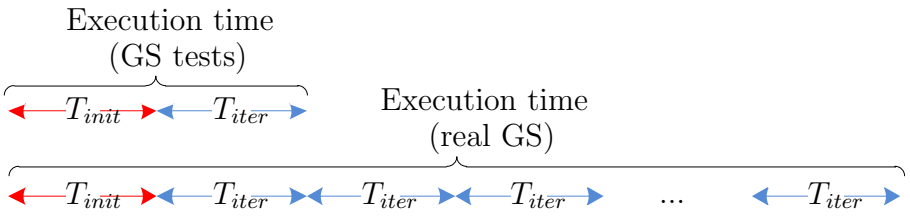


Figure 5.2: The total execution time of GS tests and real GS (initializations + iterations). For the tests the initialization part has a big influence, while in a real GS it is negligible.

The GS calculations can only be parallelised in domain; it is not possible to parallelise them in states. Domain parallelisation, as mentioned in the introduction, is the division of the real-space into domains, that are assigned to different processes. This type of parallelisation is limited by the number of points in the mesh, so, in principle, the scaling could not be very important, because, beyond certain point, we do not get more precision with a finer mesh.

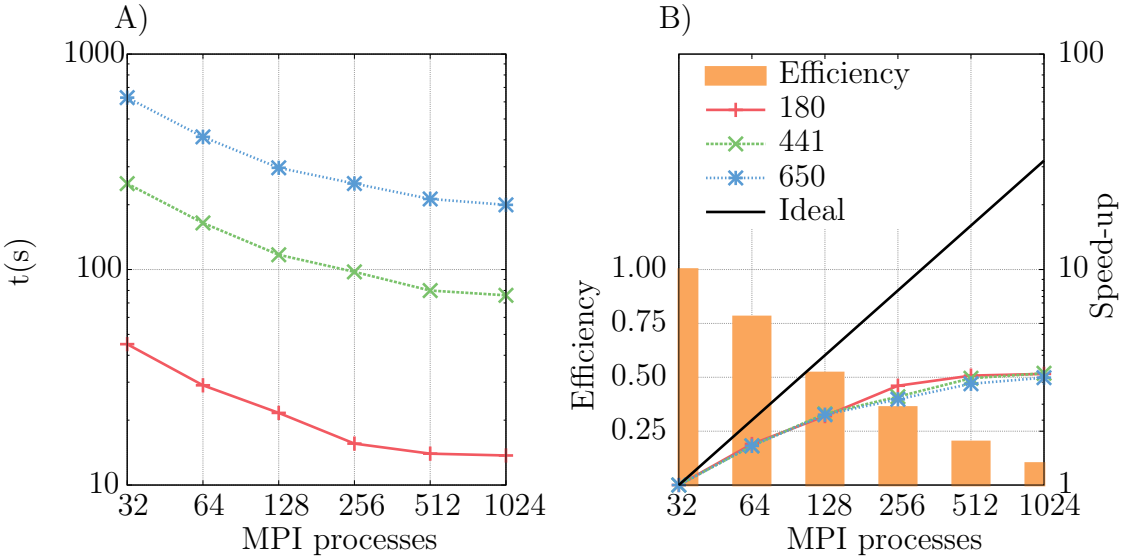


Figure 5.3: GS iteration time (A), and efficiency and speed-up (B), in the Jugene machine. X axis is the number of MPI processes. Every line represents an experiment with a different number of atoms. The speed-ups are normalised to 32 processes. For the three system sizes, only a moderate speed-up is shown until 256 processes; with 512, the efficiency is only 20%; and with 1024, 10%.

The first set of test was devoted to analyse the speed-up with a small atomic system and a reduce number of processes, because the HPC resources are quite expensive and limited. The significant part of the code of the GS is the Self Consistent Field (SCF), where iterative calculations are done; Figure 5.3 shows the obtained results. We obtain a moderate increment in the speed-up until 256 processes. The speed-up seems to reach a plateau near 512-1024 processes, with an efficiency of only

0.1 with 1024. Therefore, we have found the limit of the parallelisation of the GS calculation between 512 and 1024 processes, independently of the number of atoms used.

In summary, Ground State tests were need to obtain the more adequate number of processors for an efficient GS parallel execution. We have estimated that number in between 256-512 processes (efficiency of 0.36 and 0.2 respectively), because we obtain a fast execution without wasting too much resources. As mentioned, that final GS execution must be done only once, and it is the starting point for different and longer TD calculations.

5.3.2 Time-Dependent (TD) calculations

In a second step, we have measured the execution time of the Time-Dependent simulations: the elapsed[1] time of all the programs, and also, the time of one iteration (to be more precise, we did 10 iterations and calculate the average). The OCTOPUS code has to do initializations (create the mesh, reserve memory, prepare the MPI environment...) before can start doing actual simulations.

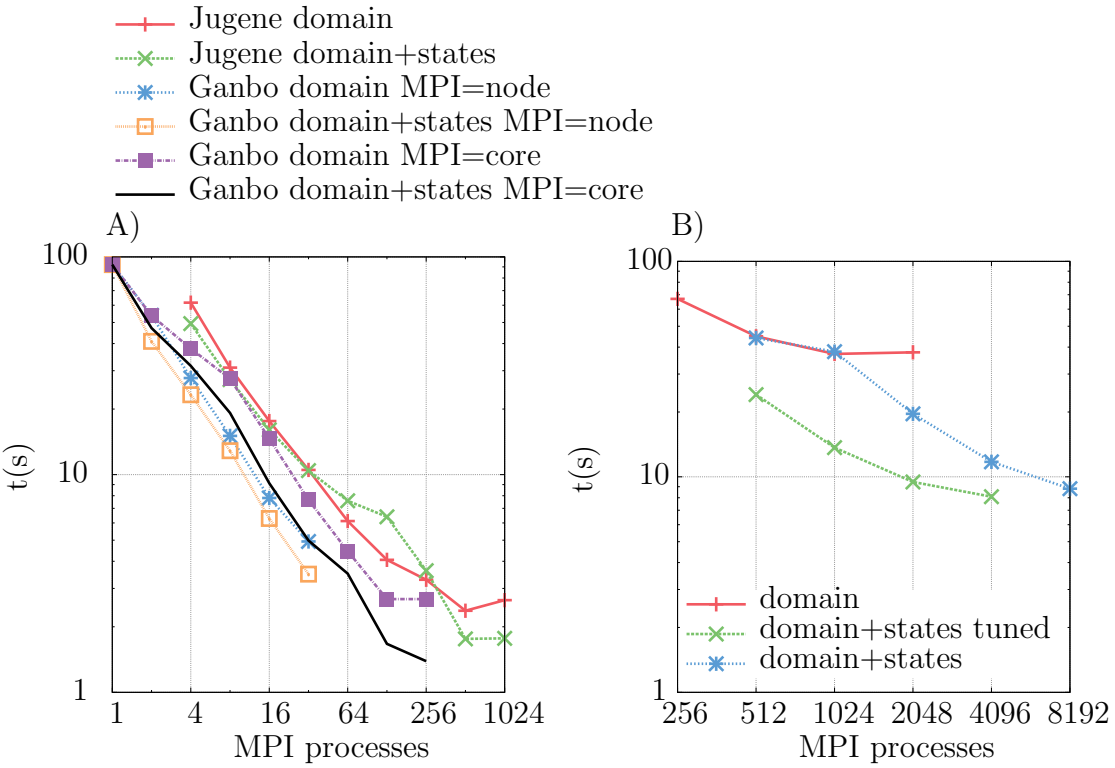


Figure 5.4: Execution time of a TD iteration. Two systems are shown: 180 atoms in (A) and 1365 atoms in (B) and two machines (Ganbo and Jugene). In Ganbo, as it has 8 cores per node, we can either use 1 processor (8 cores) for each MPI process (and use the 8 cores for OpenMP parallelisation), or we can use a core for each MPI process. "Tuned" label indicates that each processor is using 22 states. A deep argumentation of this option is shown below. "Domain" label means that domain parallelisation is used, and "domain+states" indicates that both parallelisation schemes are used.

[1] *Elapsed time:* is the global execution time of a function, including all the function calls inside it.

The overall execution time is not very meaningful, there is no great improvement in that overall time and sometimes is even worse with more processors. So we focused our effort to measure the execution time of an iteration, as OCTOPUS does an iterative process to find a solution. Like in the GS calculations, we limited our tests to a fixed number of iterations (Figure 5.2 is also valid for TD calculations; the initialization part is important in these tests, while it is negligible in real calculations).

Figure 5.4 shows the execution time for one iteration of the TD calculation, using two different machines (Jugene and Ganbo) and a small and a big atomic system, of 180 and 1365 atoms respectively. Figure 5.5 shows the achieved speed-ups.

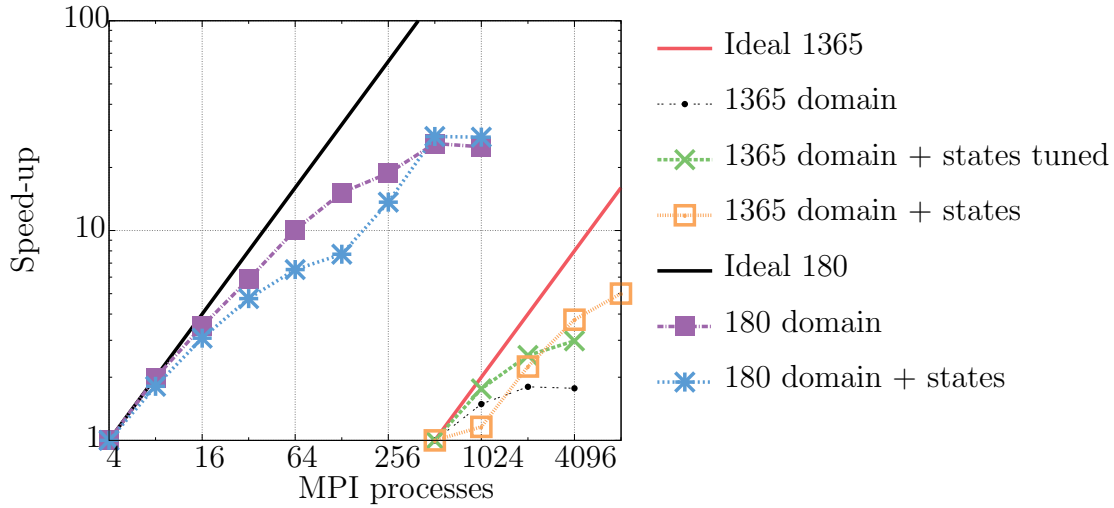


Figure 5.5: TD iteration speed-up in Jugene. Two systems are shown (180 and 1365 atoms). The speed-ups are normalised to 4 in the 180 atoms system and to 512 in the 1365 atoms system.

The obtained results show that speed-ups are not growing linearly, and so, the efficiency of the execution is very low when the number of processors is high. For the system with 1365 atoms, it is not worth to use more than 2-4K processors, and for the system with 180 atoms that maximum value is 512. For these maximum values the last efficiencies are below the 40% and suggest us a limit. If a deeper parallelisation must be achieved, we need to study why the system do not scale linearly or, at least, more efficiently. We have to find issues and get more detailed information.

Parallelisation strategies

But, before going forward, we have to find the best parallelisation strategy. So, we have made tests tuning the available parallelisation levels; domain and states parallelisations are the alternatives that we can work with. The options that we have tried are:

- Parallelisation over domains. The mesh is split over all the processes.
- Automatic parallelisation over domain + over states. We let the code decide the number of nodes used for each kind of split.

- Manually optimised parallelisation over domain + over states. We have done some tests to decide the best number of nodes for each parallelisation.

We have obtained the best results (less time for the execution of an iteration) manually tuning the nodes for parallelisation and doing domain and states parallelisation (Figure 5.6).

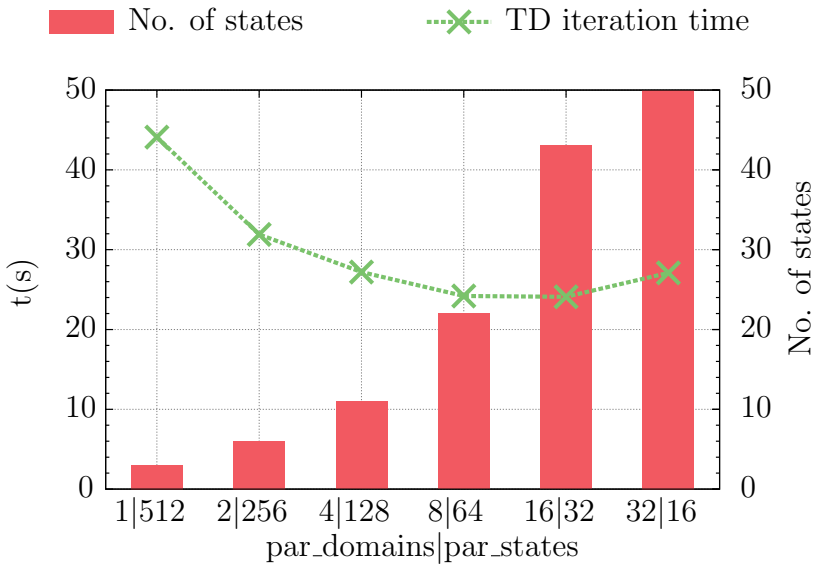


Figure 5.6: An execution of the 1365 atoms system in Jugene, using 512 processes, varying the number of nodes dedicated to split the domain or to split states. The representation we used in X axis label is par_domains|par_states. The first part of the label is the number of nodes that are dedicated to split the domain. The second part of the label is the number of nodes that split the states. The columns in the graphic show the number of states that has each process in average (with this system we have approximately 1800 states, that have to be divided with the second number of the label).

We get the best result when we split the mesh in such a way that we let each processor run 22 states. Figure 5.6 shows that the best way to do the split can be either 8|64 or 16|32 (processes for domains|processes for states). But, as we know from previous experience, the ideal split used to be around 10 states per node and the most efficient nearest value in this case is 22 states per node; so, we have decided to use the 8|64 partition. When we increase the number of nodes we will keep the number of states per node fixed (in this case, 22).

Poisson solver

Using the profiling infrastructure of the Octopus code we have done a more detailed analysis. In this manner, we have been able to obtain the execution times of each function. As we are running in a big amount of processes, an equivalent number of files are generated. These files have been processed to obtain the adequate information. We have selected two kind of functions to generate Figure 5.7: those that need a big amount of time, and those that have a big variance.

From Figure 5.7 we can argue that almost all the functions scale efficiently, i.e. they need less execution time with more processes. However, one of them, the Poisson solver function, shows an "anomalous" behaviour; instead of decreasing its execution time with more processes, it increases. Moreover, this analysis shows that

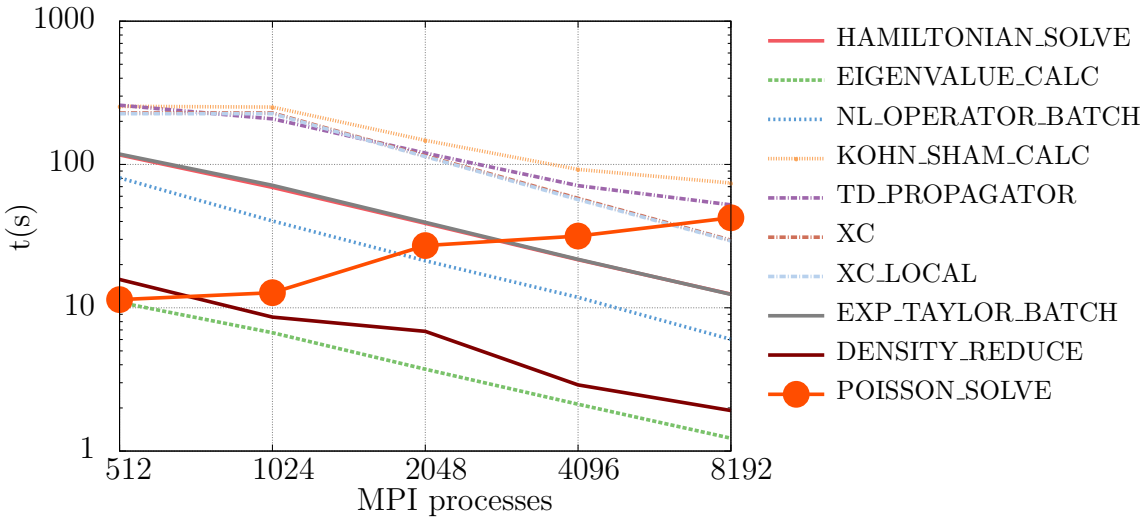


Figure 5.7: Cumulative execution time for all the most important Octopus *functions for the 1365 atoms system in Jugene. All the functions but the Poisson solver function are scaling quite well.*

Processes	512	1024	2048	4096	8192
Iteration execution time (s)	44,89	34,03	19,61	11,74	8,81
Poisson solver execution time (s)	1,14	1,27	2,72	3,21	4,25
Poisson solver percentage (%)	3	4	14	27	48

Table 5.2: Poisson solver execution time as a percentage of the TD iteration execution time, for a system of 1365 atoms in Blue Gene/P.

the Poisson function accounts only for a little percentage of the execution time of an iteration when few processors are used. But, when a huge number of processors is used, it comes up until the 50% of the iteration time (Table 5.2 shows this evolution). Whereas we have previously only mentioned the limited speed-up of the TD run, now we can expose that the Poisson solver is the main source of this weakness.

This function solves a Poisson equation that needs to work with "global" data. Previous to the calculus, the data distributed among the processes must be collected in one process (executing a MPI function: `MPI_ALLGATHER`, all the other calls are shown in Figure 5.8). In this function the Poisson equation is solved using an old version of the Interpolating Scaling Function (ISF) method. Since this method uses Fast Fourier Transforms (FFT), two `MPI_ALLTOALL` have to be done [57]. After collecting all data (`MPI_ALLGATHER`), at the end of the function, the results have to be distributed again to all the nodes (a `MPI_SCATTER` is needed for that).

Figure 5.8: MPI calls inside the Poisson subroutine.

All those MPI calls involve all the processes solving the equation, so we could suggest that, in principle, the scaling could not be good. Because the number

of processes is increasing, although each process has less computation to do, the communication time will maintain or increase. A reduction of the computation has to be followed by a reduction of the communication cost, which is really difficult using MPI global communication functions, such as gather and scatter.

5.4 Memory and execution time constraints

Octopus is a complex piece of software, with a lot of interrelated functions that need to be carefully tuned if good parallel efficiency must be reached. Of course, the behaviour of the parallel version of these functions can change with the number of processes. Besides the mentioned problems in the tests, it is important to mention the limitations we have had with the memory and execution times.

5.4.1 Memory limitations

RAM memory of current processors ranges from 1 GB up to 32 GB; therefore, a parallel supercomputer offers us a great amount of total memory, usually lots of TB. Despite this huge amount of RAM memory, some applications can be limited by the memory of an individual computing node. This is the case of Octopus, where the data structures representing an atomic system must be loaded in every process. In fact, as memory per computing node in Jugene is not very high (2 GB), we have been not able to simulate the biggest atomic system that we had.

Although the atomic system simulation could fit in memory, we found problems to converge the system, i.e. to get the real GS. After we have done the GS tests we have to converge the atomic system to begin with the TD tests. These simulations until convergence were not as we expected, and we had to customise input variables to get a real GS.

5.4.2 Execution times

The time spent executing the different tests has been huge (much more than 1,500,000 core hours), and we have already mentioned the two main reasons: problems to fit systems in memory (we have had execution allocation errors) and the difficulties to converge the GS in the firsts attempts. For those reasons, we have used a lot of computational time and we had reached the limit in assigned hours in Jugene supercomputer. Appendix C.I shows an analysis of the consumed execution times. Therefore, to be able to finish these preliminary tests, we asked the manager of Jugene for more computing time.

5.5 First alternatives to improve the Poisson solver

The results of the tests we have executed in different computers and with different system sizes show that if high speed-ups —in the range 10,000-100,000— are to be obtained, we must improve the efficiency of the Poisson solver. We can try several ways to obtain this objective. The very first approach, but not the better one, is to limit the number of nodes we use to solve the Poisson equation, so bounding the execution time of this part of the code. This strategy can offer a solution for moderate size computers, but will fail with the highest ones (Amdahl's law).

To obtain a better performance, we have to change the algorithm that solves the Poisson equation. There are several approaches. First, we could use the Fast Multipole Method (FMM) [80]. This method consists in splitting the mesh into different

sub-grids. In this manner, the closest point calculations are done exhaustively, and approximations are used for further points. Applying this method we can save a lot of computation.

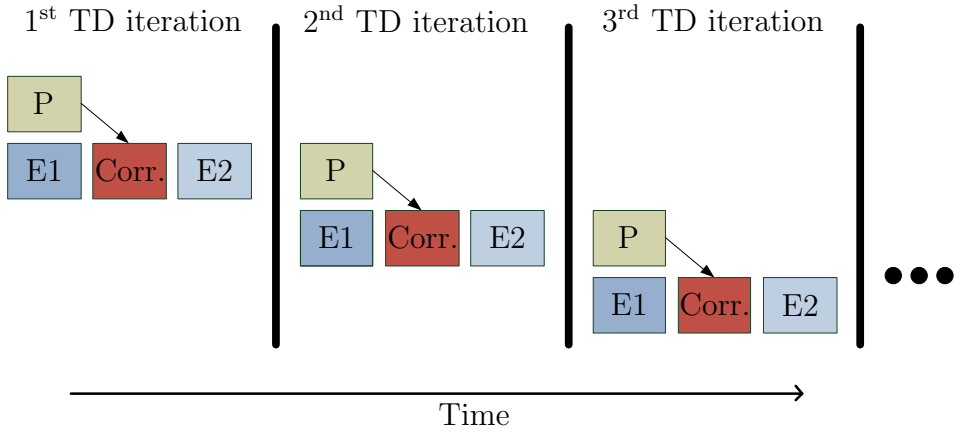


Figure 5.9: Approximated propagator for the Poisson solver. Each TD iteration has one Poisson solver (P box in the Figure) and two exponential (E1 and E2). As the first exponential depends on the Poisson solver and is calculated at the same time as the Poisson solver, a correction has to be applied later (Corr.). In this way, we overlap the executions of P and E1.

Another possible solution could be to try to pipeline different functions. It is likely that the execution can continue with some calculations without attending the final results of the Poisson solver, making an approximation at the beginning and applying a correction later.

In fact, a TD iteration consists in applying (1) the Poisson solver, plus two exponential functions: (2) $exp\big(-i\frac{\Delta t}{2}H(t+\Delta t)\big)$ and (3) $exp\big(-i\frac{\Delta t}{2}H(t)\big)$. In principle, these functions have to be done sequentially. But, there is a way that we can shift to the beginning the (2) exponential and calculate it together with the (1) Poisson solver, using an approximation. After (1) and (2) are calculated, a correction has to be applied to the result of the (2) exponential, before calculating the (3) exponential (Figure 5.9). This correction is not very heavy computationally, so overlapping the two executions, (1) and (2), we could perform faster. Even if implemented, this solution was later dismissed, in favour of more efficient solvers. Nevertheless the scalability of this strategy will be limited by the Amdahl's law.

5.6 Conclusions

The work presented in this chapter was mainly done in my master thesis. The aim was to do an initial efficiency analysis of the parallel version of Octopus code. The test we have done helped us to identify the bottlenecks that prevents the code scaling efficiently with the number of processes beyond 512-1024. We have checked the scaling of the program varying the number of processes in four supercomputers (Jugene, MareNostrum II, Vargas and Ganbo), with four different system sizes.

All the Octopus executions have an initialisation part and iterative part (Figure 5.2); we have focused our analysis in the iterative part of the executions. Three types of experiments have been done:

1. GS tests: tests to measure the times needed to obtain the ground state of an atomic system, in order to tune the most adequate number of processes to do the real GS calculation. The scalability of these tests was poor. In any case, we have obtained moderate speed-ups, so we could use around 256-512 processes efficiently to do such calculations.

2. Real GS: is an execution until convergence of the atomic system to start with TD simulations. It has to be done only once and with the best number of processes, as mentioned above. We have used more computer time than the ideal, because of problems with the lack of memory and with the convergence of the system.

3. TD tests: tests to measure the execution times of the Time-Dependent iterations (average of 10 iterations), the most important one for the project and from the physical point of view.

In the TD tests the scalability was high until 2-4K processes (Figure 5.5), but with more processes the efficiency was poor (less than 40%). Using profiling techniques, we have analysed the code and identified the problem. We found some bottlenecks that prevent us going further and we conclude that we have to improve the Poisson solver function, if massively parallel execution want to be done. That solver accounts for a little percentage of the total execution time when using few processes, but for instance, in 8K processes it is the half of the iteration execution time (Table 5.2).

In the next chapter we will see a complete survey of different Poisson solvers and how we have overcame this problem.

Chapter 6

Poisson solver

Contents

In the previous chapter we conclude that the Poisson solver is the source of the most important bottleneck. Therefore, we are going to analyse different available alternatives with more detail, in order to be able to find the best option. At first we give the specifications of the implementation, and, after, we compare their results.

The Poisson solver or the calculation of the potential associated to a charge distribution is an important step in many numerical implementations of DFT/TDDFT. In addition, the calculation of the electrostatic potential associated to a charge density can appear in other contexts in electronic structure theory, like the approximation of the exchange term [131, 132, 118], or the calculation of integrals that appear in Hartree–Fock [133, 134] or Casida [135] theories. It is understandable that the calculation of the electrostatic potential has received much interest in the recent past within the community of electronic structure researchers [136, 137, 71, 138, 14, 139, 140].

Since at present the complexity of the problems requires High Performance Computing platforms [23], every algorithm for a time-consuming task must not only be efficient in serial, but also needs to keep its efficiency when run in a large number of processors (e.g. more than 10,000 CPUs). The electrostatic interaction is non-local, and thus the information corresponding to different points interacting with each other can be stored in different computing units, with a non-negligible time for data-communication among them. This makes the choice of the algorithm for the calculation of the Hartree potential critical, as different algorithms also have different efficiencies. Thanks to the efficiency offered by the current generation of solvers, the calculation of the Hartree potential usually contributes a minor fraction of the computational time of a typical DFT calculation. However, there are cases where the Poisson solver hinders the numerical performance of electronic structure calculations. For example, in parallel implementations of DFT it is common to distribute the Kohn-Sham orbitals between processors [141]; for real-time TDDFT and molecular dynamics in particular, this is a very efficient strategy [142, 143, 2]. Nonetheless, since a single Poisson equation needs to be solved independently on the number of orbitals, the calculation of the Hartree potential becomes an important bottleneck for an efficient parallelisation [13, 2] as predicted by Amdahl's law, if it is not optimally parallelised. Such a bottleneck also appears in other contexts of computational chemistry and physics, like Molecular Dynamics [144].

In this chapter we will analyse the relative efficiencies and accuracies of some of the most popular methods to calculate the Hartree potential created by charge distributions. Our purpose is to provide estimates on the features of this solvers that make it possible to choose which of them is the most appropriate for the electronic structure calculations. First, we discuss the details of our implementation and the parallel computers we use. Following, we present the results of our numerical experiments. We finish by stating our conclusions. Remember that the theoretical introduction to the Poisson equation is given in the Chapter 3 (Section 3.5). More specific derivations and analysis are provided in the Appendix D.

6.1 Poisson implementations

For our tests on the features of Poisson solvers in the context of quantum mechanics, we chose the Octopus code [3, 4, 2] —presented in Chapter 4—, since it is representative of the trends for quantum *ab-initio* simulation for the Poisson problem.

Because the block of calculating the Hartree potential is made in a real-space representation in most simulation packages, the results of our comparison are not particular to Octopus and are rather general and package independent. However, other implementations could show slight differences in their performance features.

To ensure a fair comparison between the methods, we tried to use implementations as efficient as possible. In particular, state-of-the-art massively parallel implementations were used in the case of the FFT, ISF, and FMM methods. The corresponding packages are publicly available, and they can be integrated into other codes. The Conjugate Gradients and Multigrid solvers are less portable, since they are *ad hoc* implementations for our code (they do not appear in isolated libraries). Although there exist more competitive implementations of Conjugate Gradients and Multigrid, our tests of them are expected to provide conclusions that can be extrapolated.

Further details about each implementation are given next.

6.1.1 Parallel Fast Fourier Transforms

In three dimensions, a discrete FT of size $n_1 \times n_2 \times n_3$ can be evaluated using 1D FFTs along each direction, yielding a fast algorithm with arithmetic complexity $\mathcal{O}(n_1 n_2 n_3 \log_2(n_1 n_2 n_3))$. In addition, the one-dimensional decomposition of the 3D-FFT provides a straightforward parallelisation strategy based on a domain decomposition strategy. We now present the two-dimensional data decomposition that was first proposed by H. Q. Ding *et al.* [145] and later implemented by M. Eleftheriou *et al.* [146, 147, 148].

The starting point is to decompose the input data set along the first two dimensions into equal blocks of size $\frac{n_1}{P_1} \times \frac{n_2}{P_2} \times n_3$ and distribute these blocks on a two-dimensional mesh of $P_1 \times P_2$ processes. Therefore, each process can perform $\frac{n_1}{P_1} \times \frac{n_2}{P_2}$ one-dimensional FFTs of size n_3 locally. Afterwards, a communication step is performed that redistributes the data along directions 1 and 3 in blocks of size $\frac{n_1}{P_1} \times n_2 \times \frac{n_3}{P_2}$, such that the 1D-FFT along direction 2 can be performed locally on each process. Then, a second communication step is performed, that redistributes the data along direction 2 and 3 in blocks of size $n_1 \times \frac{n_2}{P_1} \times \frac{n_3}{P_2}$. Now, the 3D-FFT is completed by performing the 1D-FFTs along direction 1. This algorithm is illustrated in Figure 6.1. For $n_1 \geq n_2 \geq n_3$ the two-dimensional data decomposition allows the usage of at most $n_2 \times n_3$ processes.

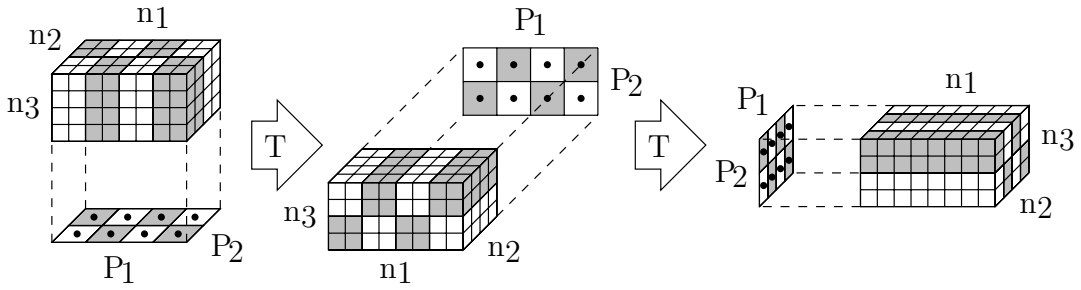


Figure 6.1: Distribution of a three-dimensional dataset of size $n_1 \times n_2 \times n_3 = 8 \times 4 \times 4$ on a two-dimensional process mesh of size $P_1 \times P_2 = 4 \times 2$.

Since it is not trivial to implement an efficient FFT routine in parallel, we rely on an optimised implementation. Fortunately, there are several publicly available FFT software libraries based on the two-dimensional data decomposition. Among them are the FFT package from Sandia National Laboratories [149, 150], the P3DFFT software library [151, 152], the 2DECOMP&FFT package [153, 154], and the PFFT software library [15]. Other efficient implementations exist, but unfortunately they are not distributed as stand-alone packages [155]. Our test runs are implemented

with the help of the PFFT software library [15], which utilises the FFTW [156, 157] software package for the one-dimensional FFTs and the global communication steps. PFFT has a similar performance to the well-known P3DFFT, as well as some user-interface advantages [15], so we chose it for our survey.

6.1.2 Interpolating Scaling Function

For the ISF solver we used the latest version of the original package provided by its authors [14]. This version is distributed as part of the BigDFT code [74], but can be compiled as a standalone library, and we will therefore refer to it as libISF. This library includes its own parallel FFT routine [76], which divides the mesh into smaller parallelepiped plane-like domains disjoints in its x (or y or z) coordinate, so we expect this part of the solver to have different scaling properties than the solver based on the PFFT library.

The accuracy of the ISF solver is defined by the choice of expanding the $1/r$ kernel into 81 Gaussians. Choosing a larger number of Gaussians would result in more precision, without any increase in the computations (only the initialisation will be slightly slower), yet the default value of 81 is appropriate for our purposes.

6.1.3 Fast Multipole Method

In this case we have used a massively parallel version of the Fast Multipole Method [81], which we will denote as libFMM. The concrete implementation used in this work is explained in [16, 81] and it is included in the Scafacos package ("scalable fast Coulomb solver")[17, 18], which is a general parallel library for solving the Coulomb problem. Additionally to the libFMM library, it also provides other state-of-the-art methods with a common interface.

Effective parallelisation of this method is attained by a domain decomposition. Moreover, libFMM allows the user to tune the relative error of the calculations. Its expression is the quotient $(E_{ref} - E_n)/E_n$, where E_n is the Hartree energy calculated with the FMM method and E_{ref} is an estimation of what its actual value is. We chose for our calculations a relative error of 10^{-4}. Note that this error corresponds only to the pairwise term of the Hartree potential, before the correction for charge distribution is applied (see Section 3.5.3 and Appendix D.IV).

6.1.4 Conjugate Gradients and Multigrid

Different versions of the Conjugate Gradients and Multigrid algorithms can have a big efficiency difference for some problems [158]. In this work, we chose the seminal version of Conjugate Gradients [90] and the standard version of Multigrid using Gauss-Seidel smoothing (with 1 cycle for presmoothing and 4 cycles for postsmoothing). We choose them because they are at present widely used. The Hestenes version is among the most popular versions of Conjugate Gradients [159, 160, 161] and it is commonly used in programs and libraries seeking efficiency [162, 159]. The Multigrid algorithm with Gauss-Seidel smoothing is also quite popular at present [163, 164, 158]. The fact that our implementations of Multigrid and Conjugate Gradients might not be the fastest available is because of their accuracy issues: on the one hand, both methods fail when the shape of the simulation mesh is not compact and the nuclei are not far from its border (which precludes most of the practical cases); on the other hand, even for such boxes, the accuracy of the calculations

for both methods is much lower than that of the most competitive solvers (i.e. ISF, FMM, and PFFT), as will be shown in Section 6.2.1. The limited accuracy of Multigrid and Conjugate Gradients for our problem did not make it advisable to focus on thorough implementations for them. Therefore, the efficiency properties of Multigrid and Conjugate Gradients presented in Section 6.2.2 must be understood as an estimate of the properties of the corresponding solver families, in contrast to the efficiency properties of the most accurate methods (ISF, FMM, PFFT), whose implementations correspond to libraries thoroughly selected among the most efficient existing ones.

It is to be stressed that the Multigrid and Conjugate Gradients solvers do not produce appropriate results when the set of mesh points corresponds to adaptive shape (the default case in OCTOPUS), and must be used with compact shapes, such as spherical or parallelepiped. This is because in adaptive mesh we impose the value of the density to be strictly 0 in some points, and representing it with a smooth series as a multipole expansion can lead to very steep changes in the density, and therefore to errors.

With respect to the parallelisation of our CG implementation, it is based on the domain decomposition approach, where the mesh is divided into subregions that are assigned to each process. Since the application of the finite-difference Laplacian only requires near-neighbour values, only the boundary values need to be shared between processors (see refs. [3, 2] for details). Since our implementation can work with adaptive shape meshes, dividing the meshes into subdomains of similar volume while minimising the area of the boundaries is not a trivial problem, so the Metis library [165] is used for this task.

Just as in the case of Conjugate Gradients, the parallelisation of Multigrid is based on the domain decomposition approach. However in the case of Multigrid some additional complications appear. First of all, as coarser meshes are used the domain decomposition approach becomes less efficient as the number of points per domain is reduced. Secondly, the Gauss-Seidel procedure used for smoothing should be applied to each point sequentially [96] so it is not suitable for domain decomposition. In our implementation we take the simple approach of applying Gauss-Seidel in parallel over each domain. For a large number of domains, this scheme would in fact converge to the less-efficient Jacobi approach.

Concerning the chosen input parameters for our test calculations, we fix the multipole expansion (the order of the multipole expansion of the charges whose potential is analytically calculated [66]) to 7 for Multigrid and Conjugate Gradients. In addition, for Multigrid we have used all available Multigrid levels (number of stages in the mesh hierarchy of the Multigrid solver). Further information about the concrete parameters can be found in the Section D.II of the Appendix D.

6.2 Results

In this Section we present the results of our tests to measure the execution time and accuracy of the methods discussed in Chapter 3 (Section 3.5). Although in principle the accuracy should not depend significantly on the particular implementation of the method used, the same is obviously not true for the execution time. Therefore, in order to avoid any ambiguity, whenever the term Poisson solver is used in the following, it always refers to a particular implementation of a given method. The

chosen parameter for the tests presented here are show in Chapter D (Section D.II).

Our tests consist of the calculation of the Hartree potential v for a given charge density ρ, but the efficiency conclusions would be also valid for cases where a classical generalised potential ($(v, \mathbf{A})$, analogous to the Hartree potential) is calculated as a function of the charge density and the magnetisation ($(\rho, \mathbf{m})$, or equivalently $(\rho_\uparrow, \rho_\downarrow)$). This is because such a "classical magnetic term" would be also pairwise.

6.2.1 Accuracy

All the quantities involved in *ab-initio* calculations have to be calculated with as lowest possible errors to ensure the desired accuracy of the final result, and the Hartree potential is not an exception. In order to gauge the accuracy of the analysed solvers, we calculate the Hartree potential created by Gaussian charge distributions. Such potentials can be analytically calculated, and hence the error made by any method can be measured by comparing the two results: numerical and analytical. The chosen input charge distributions correspond to Gaussian functions $\rho_a(\vec{r}) = \exp(-|\vec{r}|^2/\alpha^2)/\alpha^3\pi^{3/2}$, with α chosen to be 16 times the spacing, i.e. 3.2 Å (this guarantees that the smoothness of the Gaussian test function is similar to the smoothness of densities of general physical problems). Such charge distributions are represented in cubic meshes with variable size ($2L_e$) and constant spacing between consecutive points (spacing = 0.2 Å) in all three directions. The use of other values for the spacing does not alter the accuracy in a wide value range, and increases the numerical complexity proportionally with the corresponding increase of the mesh points (see Figure D.1 (page 137) of the Appendix D.II). We use two different quantities to measure the accuracy of a given method, the error in the potential, Ξ_v, and the error in the electrostatic energy, Ξ_E. We define them as

$$\Xi_v := \frac{\sum |v_a(r) - v_n(r)|}{\sum |v_a(r)|} \ , \tag{6.1a}$$

$$E_a = \frac{1}{2}\int \mathrm{d}r \ \rho_a(r) v_a(r) = -\frac{2}{\alpha^3\pi^{1/2}}\int_0^\infty \mathrm{d}r \, r \, \exp(-r^2/\alpha^2)\,\mathrm{erf}(r/\alpha) \ , \tag{6.1b}$$

$$E_n = \frac{1}{2}\sum \rho(r) v_n(r) \ , \tag{6.1c}$$

$$\Xi_E := \frac{E_a - E_n}{E_a} \ , \tag{6.1d}$$

where $\vec{r}_{ijk}$ are all the points of the analysed mesh, $r = |\vec{r}|$, and erf stands for the error function. v_a is the analytically-calculated Hartree potential, v_n is the potential calculated numerically, and E_a and E_n, respectively, are their associated electrostatic energies. These quantities provide an estimate of the deviations of the calculated potential and energy from their exact values. Ξ_v gives a comprehensive estimate of the error in the calculations of the Hartree potential, for it takes into account the calculated potential in all points with equal weight, while Ξ_E provides an estimate of the error in the potential in high-density regions.

In Table 6.1 we display the errors in the potential and the energy for the tested methods. The FFT and ISF methods provide in general best accuracies. Our implementation of the Multigrid solver does not work for big mesh sizes due to memory limitations. The errors in energy Ξ_E are less sensitive to local deviations than the errors in potential Ξ_v, since the former are weighted with the density. Ξ_E varies

just lightly with the different sizes for the most reliable methods (ISF, FFT, and FMM, though FMM is less accurate than the other two). The values of Ξ_v are also essentially constant for ISF and FMM. However, for the FFT method an increase of the accuracy with the system size is clear. This is because the solution of the FFT corresponds to the superposition of infinite periodic images; for smaller sizes, the neighbour images are closer from the centre of the mesh where we measure the potential, and therefore the images contribute more significantly to distort the potential (the closer the centres of two Gaussian functions, the bigger their superposition). The errors in both potential and energy of the Multigrid method oscillate a bit, as a consequence of the different number of used multigrid levels. The errors for the Conjugate Gradients method increase with the size of the system because the bigger the number of points, the harder it is for the multipolar expansion (see Chapter 3, Section 3.5.4) to adapt to all the mesh points. As stated in Section 6.1, the Multigrid and Conjugate Gradients methods do not provide acceptable accuracies whenever the simulation mesh is not compact. The accuracy of the FMM method is mainly limited by the approximation of the charge densities as sets of discrete charges (the results obtained with the FMM are almost identical to those of direct pairwise summation).

Potential error, Ξ_v:

L_e (Å)	FFT	ISF	FMM	CG	Multigrid
7.0	$3{\cdot}10^{-4}$	$6{\cdot}10^{-9}$	$9{\cdot}10^{-5}$	$3{\cdot}10^{-5}$	$1{\cdot}10^{-6}$
10.0	$2{\cdot}10^{-8}$	$1{\cdot}10^{-9}$	$2{\cdot}10^{-4}$	$3{\cdot}10^{-5}$	$3{\cdot}10^{-7}$
15.8	$1{\cdot}10^{-8}$	$1{\cdot}10^{-9}$	$2{\cdot}10^{-4}$	$5{\cdot}10^{-5}$	$4{\cdot}10^{-6}$
22.0	$2{\cdot}10^{-10}$	$2{\cdot}10^{-9}$	$4{\cdot}10^{-4}$	$5{\cdot}10^{-4}$	$8{\cdot}10^{-7}$
25.8	$< 9 \cdot 10^{-13}$	$3{\cdot}10^{-9}$	$4{\cdot}10^{-4}$	$6{\cdot}10^{-3}$	—
31.6	$< 9 \cdot 10^{-13}$	$3{\cdot}10^{-9}$	$4{\cdot}10^{-4}$	$1{\cdot}10^{-2}$	—

Energy error, Ξ_E (eV):

L_e (Å)	FFT	ISF	FMM	CG	Multigrid
7.0	$2{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$5{\cdot}10^{-6}$	$2{\cdot}10^{-5}$	$1{\cdot}10^{-6}$
10.0	$1{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$6{\cdot}10^{-6}$	$5{\cdot}10^{-6}$	$4{\cdot}10^{-7}$
15.8	$1{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$6{\cdot}10^{-6}$	$5{\cdot}10^{-5}$	$2{\cdot}10^{-6}$
22.0	$1{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$5{\cdot}10^{-6}$	$5{\cdot}10^{-4}$	$6{\cdot}10^{-7}$
25.8	$1{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$7{\cdot}10^{-6}$	$3{\cdot}10^{-3}$	—
31.6	$1{\cdot}10^{-8}$	$2{\cdot}10^{-8}$	$6{\cdot}10^{-6}$	$6{\cdot}10^{-3}$	—

Table 6.1: Potential, Ξ_v, and energy, Ξ_v, errors of different methods in the calculation of the Hartree potential created by a Gaussian charge distribution represented on cubic meshes of variable edge $2L_e$ Å and spacing 0.2 Å.

In order to assess how the accuracy of the Poisson solver affects an actual DFT calculation, we calculated the GS of a system of one chlorophyll containing 180 atoms (smallest system of Figure 1.2 of the Section 1.3). To this end, we used pseudopotentials (Troullier and Martins type), so 460 electrons are treated in the calculation. The mesh shape was a set of spheres of radius 4.0 Å centred at the nuclei, and the mesh spacing was 0.23 Å. The Exchange-Correlation (XC) functional used

was the VWN-LDA functional [166]. In Table 6.2 we display the value of the Hartree energy, the highest eigenvalue, and the HOMO-LUMO gap[1]. The FFT method is expected to provide the most accurate results (by considering the results of Table 6.1). The maximum difference in the Hartree energy divided by the number of electrons is less than 0.015 eV. The maximum difference in the HOMO-LUMO gap is less than 0.0092 eV. In this test case, the differences among all five methods can be considered negligible (much lower than the errors introduced by the XC functional, the pseudopotentials and the discretisation). However, note that the deviation of the Hartree energy for the Multigrid and Conjugate Gradients methods with respect to the most accurate method (FFT) is rather larger than that of the ISF method, and so are the HOMO and the HOMO-LUMO gap.

	Hartree energy (eV)	HOMO (eV)	HOMO-LUMO gap (eV)
FFT	240,820.70	-4.8922	1.4471
ISF	240,821.48	-4.8935	1.4489
FMM	240,817.29	-4.8906	1.4429
CG	240,815.01	-4.9009	1.4521
Multigrid	240,814.69	-4.8979	1.4506

Table 6.2: Influence of the different methods on the Hartree energy, HOMO energy level and HOMO-LUMO gap corresponding to the ground state (calculated through DFT with pseudopotentials) of a chlorophyll stretch with 180 atoms.

6.2.2 Execution time

In order to gauge the performance of the Poisson solvers, we have measured the solution time for each solver as a function of the number of processes. This measured time only includes the operations directly related to the solution of the Poisson equation, and excludes the initialisation time of the Poisson solver. The selected machines are two Blue Gene/P (Jugene and Genius), Curie and Ganbo (presented in Chapter 2, Section 2.2.3). We ran one MPI process per node on Blue Gene/P's due to limited amount of memory per node, and one single MPI process per CPU core on Curie and Ganbo. In the latter two, further tests of the efficiency of OpenMP were also done[2]. Runs up to 4096 MPI processes were made in Genius, Jugene and Curie machines. Whereas, runs up to 512 were made in Ganbo.

In our efficiency tests we calculated the potential created by Gaussian charge distributions as those explained in Section 6.2.1. These charge distributions are represented in parallelepiped meshes with edge length $2L_e$, for L_e equal to 7.0, 10.0, 15.8, 22.0, 25.8 and 31.6 Å respectively. The mesh points were equispaced with a spacing of 0.2 Å (additional tests with variable spacing are presented in Appendix D.II). The smallest simulated system, with $L_e = 7.0$ Å, contained 357,911 mesh points, while the largest one with $L_e = 31.6$ Å contained 31,855,013 mesh points.

[1]The HOMO-LUMO gap is often used to test the accuracy of a calculation method [167] because it provides an estimate for the first electronic excitation energy [168]. The HOMO itself provides an estimate for the ionization energy [169].

[2]Multithreading is implemented in PFFT and libISF, our Conjugate Gradients solver shows only a small improvement, and the libFMM and Multigrid solvers do not take advantage of this feature.

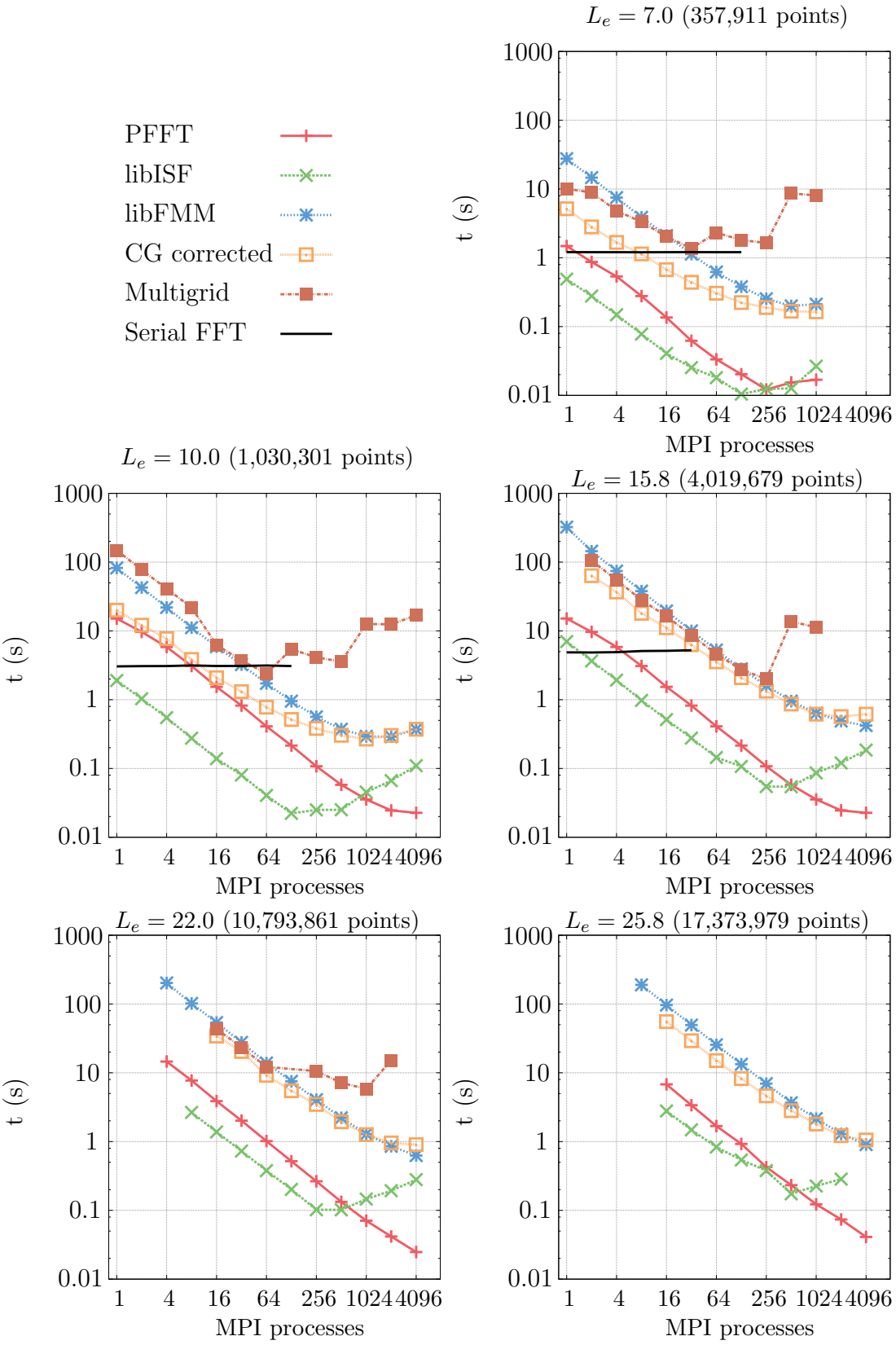


Figure 6.2: Execution times for the calculation of the Hartree potential created by a Gaussian charge distribution on a Blue Gene/P machine as a function of the number of MPI processes for six different Poisson solvers and for different simulated system sizes (number of mesh points).

Due to memory constraints, the serial FFT was limited to $L_e = 15.8$ Å (4,019,679 mesh points), Multigrid to $L_e = 22.0$ Å (10,793,861 mesh points) and the parallel runs for all solvers in the Blue Gene/P were limited up to $L_e = 25.8$ Å (17,373,979 points). As explained in Section 3.5.1, the FFT method requires an enlarged cell when applied to finite systems. The size of this mesh ranges from 143^3 to 637^3 points (up to 525^3 on Blue Gene/P).

As stated in Section 6.1, the efficiency properties of our implementation of the Multigrid and Conjugate Gradients solvers presented in this section must be understood as an estimate of the properties of the corresponding methods, in contrast to the efficiency properties of the most accurate methods (ISF, FMM, and FFT), which correspond to highly optimised (state-of-the-art) algorithms and implementations (libISF, libFMM, and PFFT).

Figure 6.2 presents the execution times obtained in tests on a Blue Gene/P machine, each graph representing a different problem size. As can be observed there, the tests show a similar trend with regard to the system size and the number of MPI processes: execution times decrease with the number of processes until saturation, and larger systems allow the efficient use of a higher number of parallel processes, i.e., the solvers "saturates" at a higher number of processes. This is especially true for the PFFT and libFMM solvers. Thus, this behaviour leaves the way open to simulate physical systems of more realistic size if tens of thousands of cores are available.

For a given system size (N points), libISF is the fastest solver if the number of processes used in the solution is low-medium. One of the reasons for this behaviour is that, in contrast to the FFT method, libISF does not require the box size to be increased for accuracy reasons. This advantage should nevertheless disappear when dealing with periodic systems. The execution time of libISF decreases with the number of MPI processes until a saturation point (p_m processes) is reached. p_m, the congestion point, is proportional to N_{cx} (or N_{cy} or N_{cz}) being $N_{cx} \times N_{cy} \times N_{cz}$ the number of points of the minimal parallelepiped mesh containing the original mesh (such an auxiliary mesh is necessary for the discrete Fourier transform which is carried out by libISF). This is because libISF divides such a parallelepiped into smaller parallelepiped plane-like domains disjoints in its x (or y or z) coordinate. Since the minimal width of one of these domains is 1 (one point in direction x), the maximum number of MPI processes to use in a parallel run is roughly N_{cx} (actually the minimal execution times commonly happens for a number of MPI processes which is slightly bigger –e.g. 10% – than N_{cx}; this is because the Goedecker's FFT used by ISF works only for particular values of mesh sizes, so the mesh must often be slightly enlarged to match a doable size). Using more MPI processes leads to increased execution times, because all the processes take part in the internal `MPI_Alltoallv` communications, even if they have not done any useful work. This behaviour can be delayed using OpenMP threads inside MPI processes. As we will see later in Figure 6.10, the best efficiency is achieved with a combination of MPI processes (that are limited by the mesh size) and OpenMP threads, thus using a higher number of cores in parallel.

The Conjugate Gradients and libFMM solvers seem to be very efficient approaches in terms of scalability. Particularly attractive is the good performance obtained with the PFFT solver. If the number of processes is high enough, the

execution times are always lower than the corresponding to other solvers. A better approach to the data distribution and highly effective communications are the reasons behind. Furthermore, the prefactor of the FFT method is small by construction, and this largely compensates the fact that FFT does scale with $N \log_2 N$ (so, higher than N). Nevertheless, when the number of processes is low enough, PFFT is not as fast as libISF, since the FFT method requires the box size to be increased, as explained above.

The execution times obtained using the Multigrid solver depend on the chosen number of multigrid levels, being faster when the number of levels is high. The number of levels in the Multigrid solver is equal to $\lfloor \log_8 N \rfloor - 3$, where N is the total amount of mesh points. However, for a fixed problem size, the number of levels in practice must be reduced as the number of processes increases, due to the need to assure a minimum workload to each processor. Results (see Figures 6.2 and 6.3) show that the Multigrid solver offers the poorest performance even when using a high number of levels. Every processor needs a minimum number of points to work with; this implies a maximum number of usable multigrid levels for a given number of processors (and this number may be lower than $\lfloor \log_8 N \rfloor - 3$). When this *optimal* number of multigrid levels is not reachable (e.g., for a high number of processors), then the convergence of the Multigrid solver is slower, and the execution time increases (see Figure 6.2). This saturation point appears with a relatively low number of parallel processes.

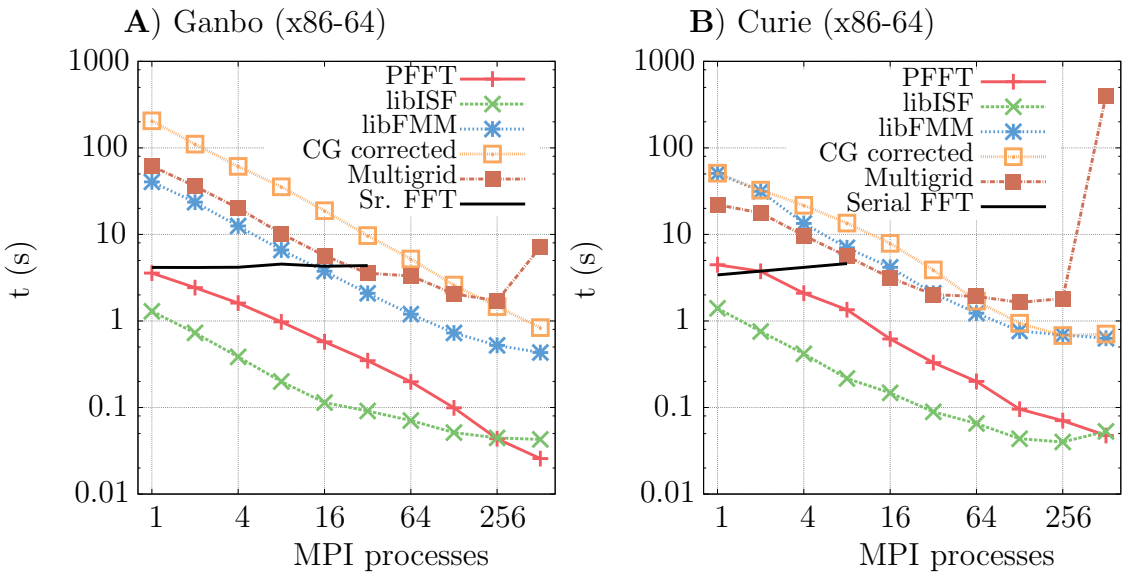


Figure 6.3: Execution times of the Poisson solvers ($L_e = 15.8$ Å - 4,019,679 mesh points) in Ganbo (A) and Curie (B) supercomputers as a function of the number of MPI processes (each one executed in a CPU processor core). The execution times for the libISF solver correspond to the fastest MPI+OpenMP combination (see Figure 6.10 for more details).

The Multigrid and Conjugate Gradients methods require the calculation of a correction due to the multipolar expansion used. The calculation of this correction implies a similar time for both our solvers, and represents between 0.2% and 8% of the Conjugate Gradients solver total execution time, and between 1.7% and 3.2% in the case of the Multigrid solver. In essence, it does not add a significant extra time. Further details can be found in Appendix D.III.

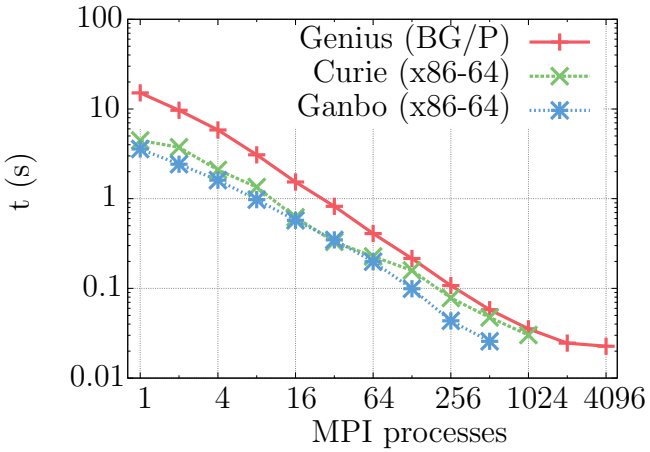


Figure 6.4: Execution times of the PFFT solver in Genius (Blue Gene/P architecture), Curie and Ganbo (x86-64) for a system size of $L_e = 15.8$ Å (4,019,679 mesh points) as a function of the number of MPI processes.

A similar overall behaviour of the different solvers, with minor differences, is also shown in the other two machines, Curie and Ganbo. Nevertheless, one appreciable difference is that the networks are not as efficient as in Blue Gene/P, and when a solver saturates, the execution time increases substantially. Figure 6.3 presents the obtained results for a particular case: $L_e = 15.8$ Å (4,019,679 mesh points). As it can be observed, the best results are obtained again with the PFFT solver for a high number of parallel processes and with libISF for a lower-medium number of processes (the libISF solver needs to combine OpenMP and MPI approaches, otherwise the time increases heavily when using a high number of processes). Relative performance between the solvers varies slightly from the above analysed cases, but the conclusions are very similar.

For the serial case, executions are between 1.4 and 5 times faster in Curie than in Ganbo for the different solvers and system sizes, but this difference tends to disappear in the parallel versions, where communication times must be added to computing time. These differences can be attributed to the different cache sizes and abilities for vector processing of the processors of Curie (X7560-Nehalem) and Ganbo (E5645-Westmere).

In order to compare more clearly the results obtained with different computers, Figure 6.4 shows the execution times of a concrete solver (PFFT) for the case of $L_e = 15.8$ Å (4,019,679 mesh points) in Genius (Blue Gene/P architecture), Curie (x86-64) and Ganbo (x86-64). Execution times are higher in the Blue Gene/P computer when the number of processes is relatively small, but it allows an efficient use of a high number of processes before saturation. The processors of the Blue Gene/P machine are slower than those of Curie and Ganbo, but communication infrastructures are more effective. This leads us to conclude that the execution time of the Poisson solver parallel code depends not only on the individual processor performance, but also on interprocessor communication network of the parallel computer, specially when using a high number of processors.

Our tests have shown that the implementations of the novel Poisson solvers, PFFT and libFMM, do offer good scalability and accuracy, and could be used efficiently when hundreds or thousands of parallel processes are needed for the analysis

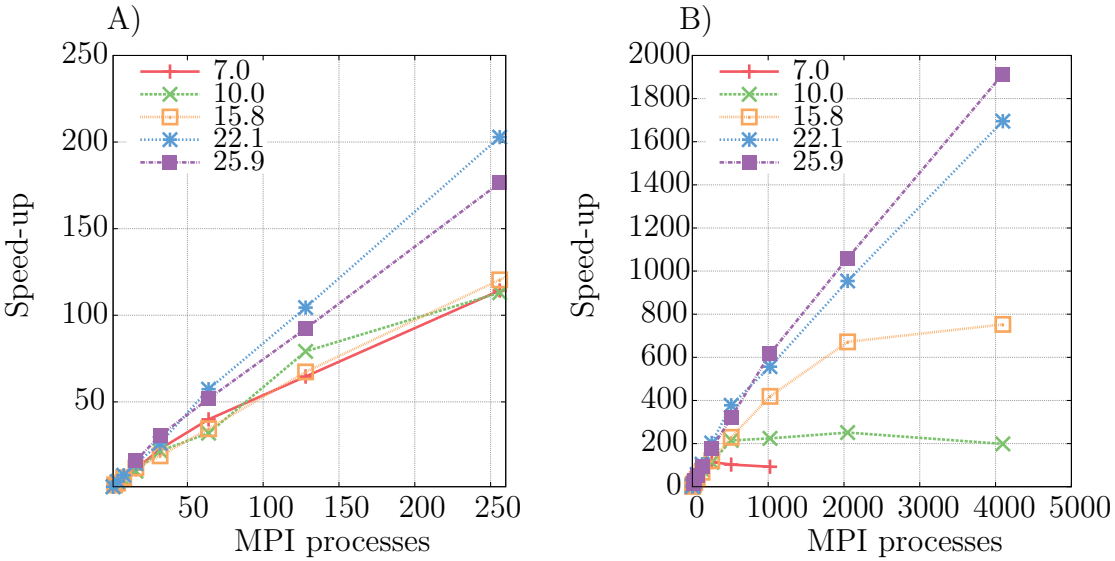


Figure 6.5: Speed-up of the PFFT Poisson solver in a Blue Gene/P computer for different system sizes (given by L_e, semi-legth of the parallelepiped edge). Largest systems saturate with more processes than the smallers ones. A) linear speed-up is shown up to 256 processes, independently of the simulated system. B) saturation point is higher when the simulated system is larger.

of representative systems. libISF should be the chosen solver when low-medium numbers of processes are used, and PFFT should be chosen instead for high numbers of processes (the concrete number of processes that makes PFFT more competitive depends on the system size). Figure 6.5 shows the speed-ups obtained using PFFT for the different system sizes in a BlueGene/P supercomputer. Almost linear performances can be observed until saturation for all solvers. Before saturation, the obtained efficiency factors are always above 50%. As expected, large systems, which have higher computation needs, can make better use of a high number of processes. Tests run in Ganbo and Curie machines show similar trends (although with efficiency problems of PFFT for some values of MPI processes).

6.2.3 Weak-scaling

Apart from the execution time of a concrete parallel run, a very useful quantity to measure in order to estimate its performance is the weak-scaling. In order to maintain the same number of mesh points per processor in all cases, we executed different system sizes and, for these tests, adapted them to L_e equal to 7.8 Å (493,039 mesh points), 10.0 Å (1,030,301 points), 15.8 Å (4,019,679 points), 20.0 Å (8,120,601 points), 25.0 Å (15,813,251 points) and 31.6 Å (31,855,013 points, only in Ganbo) for the Poisson solver on Blue Gene/P and Ganbo. We did parallel runs using 4, 8, 32, 64 and 128 MPI processes (also 216 and 256 processes in Ganbo). So, the number of mesh points processed in each parallel MPI process is roughly 125,000 for all the cases. Figure 6.6 shows the obtained results, with data taken from the profiling output. In it, the normalised weak-scaling for p processes is measured as the quotient: [time to calculate the Hartree potential of a system of about $125{,}000 \times P$ points using p processors] divided by [time to calculate the Hartree potential of a system of 493,039 points with 4 processors].

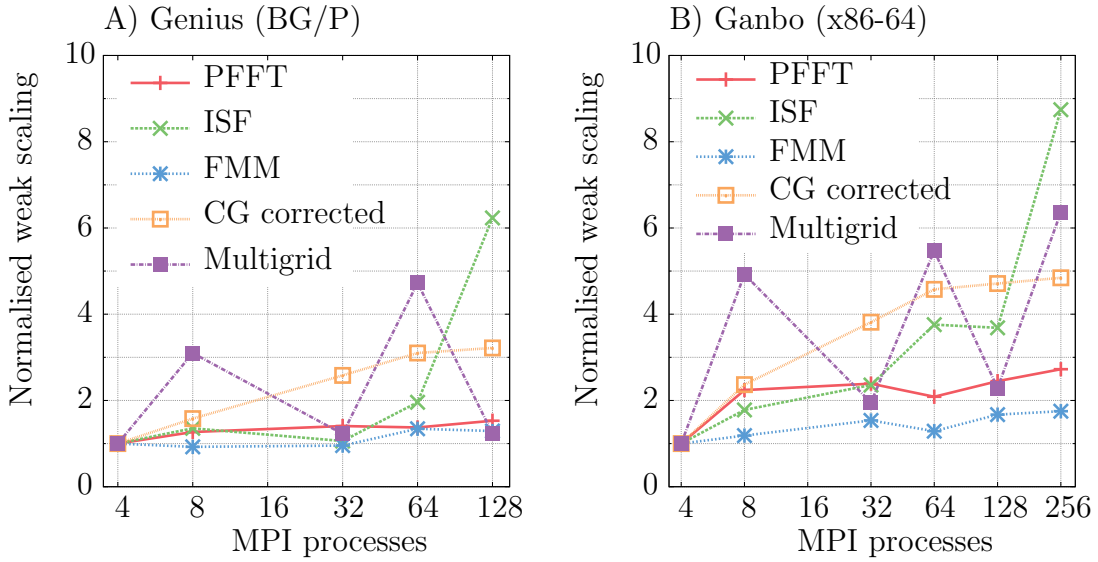


Figure 6.6: Normalised weak-scaling (i.e. execution time for the calculation of the Hartree potential in a system of about 125,000×p points using p processors divided by the execution time for a system of 493,039 points with 4 processors) of the Poisson solvers in Blue Gene/P (A) and Ganbo (x86-64) (B). Each MPI process has roughly 125,000 points and selected system sizes are given by L_e equal to 7.8 Å (493,039 mesh points), 10.0 Å (1,030,301 points), 15.8 Å (4,019,679 points), 20.0 Å (8,120,601 points), 25.0 Å (15,813,251 points) and 31.6 Å (31,855,013 points, only in Ganbo) and a spacing of 0.2 Å.

The weak-scaling factor of the Conjugate Gradients and Multigrid solvers shows an important increase with the number of processes, and this can be used to predict that parallel execution will saturate at a moderate number of processes (as observed in Figure 6.2). In contrast, libFMM shows the best results in the analysed range: weak-scaling increases very moderately, so we can conclude that communication times are acceptable and that the solver will offer good speed-up results when using thousands of processes. PFFT also gives very good results in the Blue Gene/P system; they are similar to those of libFMM, so similar conclusions can be stated. Nonetheless, the results obtained in Ganbo up to 256 processes seem to indicate a trend to increase, which implies that communications will play a more important role if a high number of processes is to be used. Similar behaviour is shown with the libISF solver, but with a sharper increase with large number of processes. Thus, Figure 6.6 highlights the efficiency of the communication network and protocols of the parallel computer: the weak-scaling factor is markedly better and bounded in Blue Gene/P, a system with a very high performance communication network. From these tests, we can conclude that the communication needs of the Poisson solvers fit better in the network of a Blue Gene/P machine than in that of a machine with x86-64 processors and Infiniband network (e.g. Ganbo). This stresses the importance of efficient communication architectures when dealing with the calculation of pairwise potentials.

6.2.4 Non-cubic boxes

A final test was done using a non-cubic box: the Poisson solver timings of a parallelepiped mesh of size 15.8 Å × 15.8 Å × 32.0 Å (solid lines in Figure 6.7) were compared with those of a cubic mesh with L_e = 20.0 Å (dashed lines in Figure 6.7). Both meshes had a spacing of 0.2 Å and the number of mesh points were as similar as possible (8,115,201 and 8,120,601 mesh points). As it can be viewed in Figure 6.7, the execution times of the libISF solver are quite similar for both meshes. The Conjugate Gradients and Multigrid solvers show only small differences between both cases, although the execution times are slightly lower in the non-cubic mesh. Conversely, libFMM times are around 50% higher in the non-cubic mesh. Above all, the PFFT solver shows the biggest penalty here, with execution times being 75% higher in the non-cubic mesh. The reason behind this is that, for finite systems, the FFT method that we use requires the use of an enlarged mesh of cubic shape (see Chapter 3, Section 3.5.1). The penalty should disappear when solving the Poisson equation for periodic systems, as the enlargement of the mesh is not required in those cases. This penalty is expected to make the crossover between libISF and PFFT to occur at higher values of the number of cores for systems contained in non-cubic meshes[3] (if compared with cubig mesh with the same number of points). For a given number of mesh points $N = N_1 \times N_2 \times N_3$ (assuming $N_1 \geq N_2, N_3$), the bigger N_1 in comparison with N_2, N_3, the bigger the expected number of cores for the crossover.

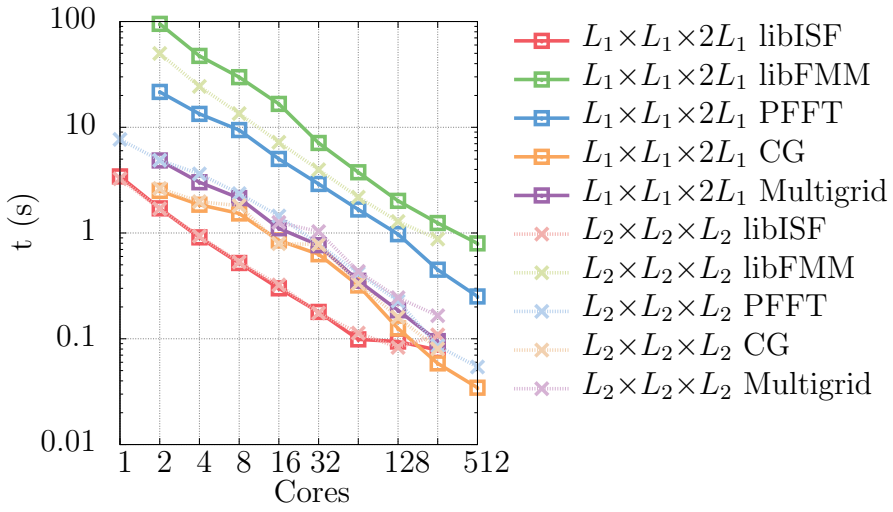


Figure 6.7: Execution times for the Hartree potential calculation as a function of the solver for a mesh of size 15.8 Å × 15.8 Å × 32.0 Å (solid line) and for a mesh of size 20.0 Å × 20.0 Å × 20.0 Å (dashed line), in Ganbo (x86-64) with a unique OpenMP thread per MPI process. Spacing equal 0.2 Å; both meshes contain approximately the same number of points.

For very time-consuming simulations, some tests should be performed in order to choose the optimal Poisson solver. In addition, for every given simulation code,

[3]Please note that despite the fact that meshes for DFT and TDDFT calculations are usually irregular, both libISF and PFFT require the application of discrete Fourier transforms, and therefore a cubic mesh containing the original mesh is used for the calculation of the Hartree potential when using libISF or PFFT.

it would be advisable to implement one subroutine which automatically chooses the solver which is expected to be optimal for the tackled problem.

6.2.5 Execution complexity

We have also analysed the execution complexity by measuring the execution time on Blue Gene/P as a function of the volume of the system for PFFT and FMM in three cases: 16, 32 and 64 parallel processes. The mesh points ratio among the smallest ($L_e = 7.0$ Å) and the largest ($L_e = 25.8$ Å) studied systems is about 49. Our results agree with the $\mathcal{O}(N)$ theoretical complexity of the FMM method and the $\mathcal{O}(N \log N)$ theoretical complexity of the PFFT method (see Figure 6.8). Note that computation time is appreciably higher when using the FMM solver due to its higher prefactor (i.e. the quantity that multiplies N or $N \log N$ in the expression of the numerical complexity). The quotient between prefactors is machine dependent; its value is about 5.5 for Ganbo, about 9 for Curie and about 12 for Blue Gene/P. One may think that for very big values of N at a constant number of processes p, the growth of the term $\log N$ would eventually make FMM faster than PFFT. This is strictly true, but the extremely huge N of this eventual crossover precludes it in practice.

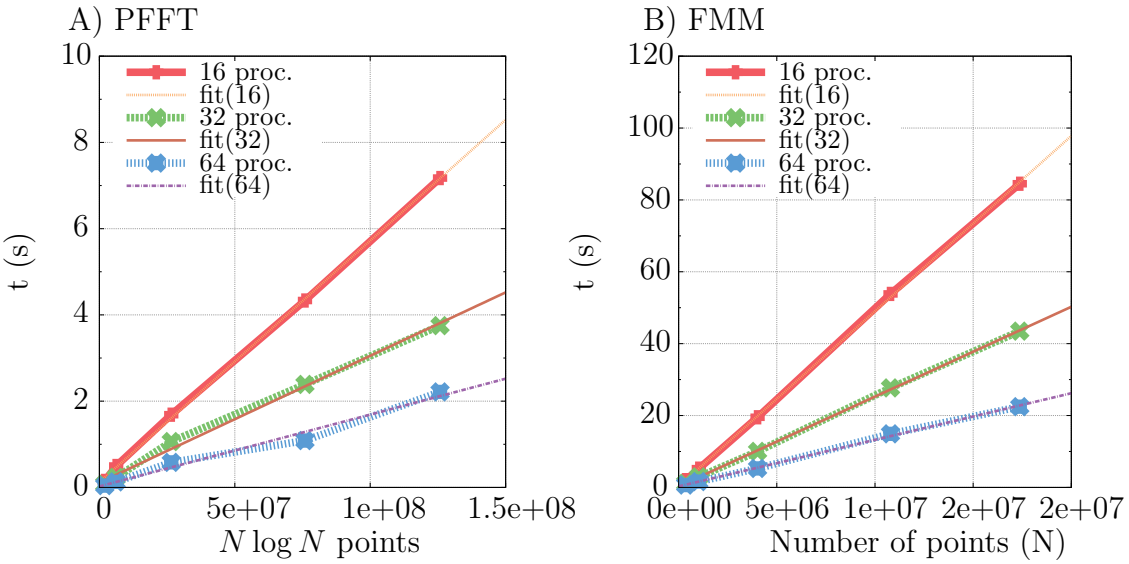


Figure 6.8: Computational complexities of the PFFT (left) and FMM (right) Poisson solver in the Blue Gene/P supercomputer for a given system. PFFT's fits $\mathcal{O}(N \log N)$, while FMM's its $\mathcal{O}(N)$. PFFT: $fit(16) = 5.6041 \cdot 10^{-8} + 0.1182$, $fit(32) = 2.9407 \cdot 10^{-8} + 0.1149$, and $fit(64) = 1.6656 \cdot 10^{-8} + 0.0228$. FMM: $fit(16) = 4.8760 \cdot 10^{-6} + 0.27730$, $fit(32) = 2.4991 \cdot 10^{-6} + 0.26903$, and $fit(64) = 1.2960 \cdot 10^{-6} + 0.28681$

6.2.6 Multithreading

Multithreading tests with OpenMP were also done in order to gauge the maximum performance for all processor cores. The libFMM, Conjugate Gradients and Multigrid solvers do not reduce the execution times when increasing the number of OpenMP threads, while the PFFT and libISF solvers do. Runs using a unique MPI process with many different number of OpenMP threads were done for all

the analysed solvers. The tests summarised in Figure 6.9 correspond to runs of the cubic system of edge $L_e = 15.8$ Å and spacing 0.2 Å, in a unique node of the Ganbo (x86-64). One can see no improvement for Multigrid and FMM solvers using OpenMP, while for the Conjugate Gradients a very slight improvement appears with 2 OpenMP threads. In contrast, ISF and PFFT are faster whilst more threads are used. We can say that both show a very similar trend, although the increase of the performance of the ISF library is slightly higher than that of PFFT.

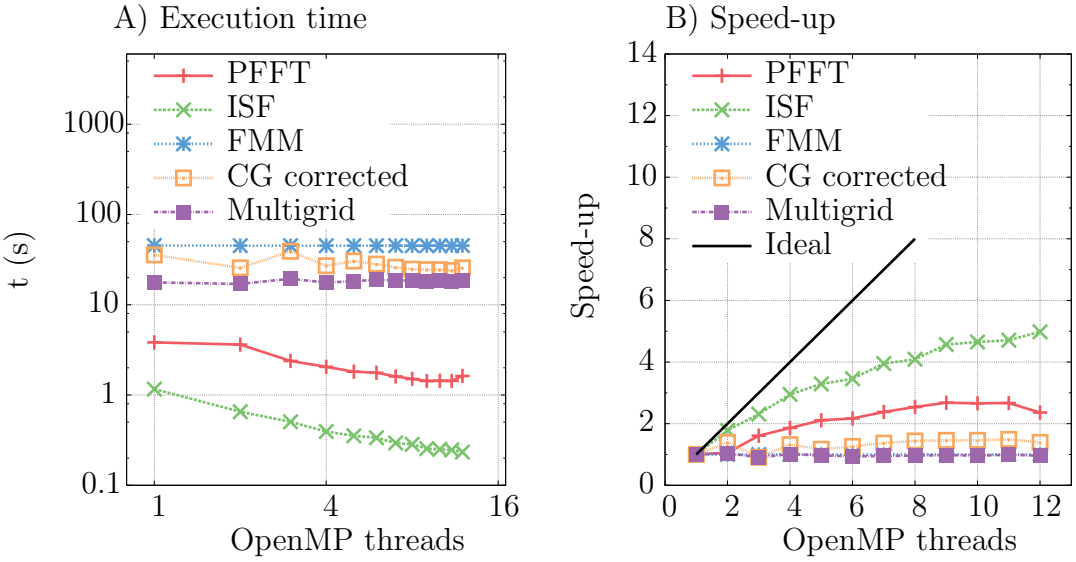


Figure 6.9: Execution time and speed-up for the calculation of the Hartree potential created for a Gaussian charge distribution represented in a cubic mesh of edge $L_e = 15.8$ Å and spacing 0.2 Å in a x86-64 architecture (Ganbo computer), with one MPI process and from 1 to 12 OpenMP threads.

Figure 6.10 shows the execution times in Ganbo (x86-64) for the system of $L_e = 15.8$ Å and spacing 0.2 Å as a function of the number of cores and the number of OpenMP threads for the libISF (A) and the PFFT (B) solvers. For each number of cores, we have measured the execution times varying the number of OpenMP threads per MPI process (maintaining always the number of parallel tasks equal to the total number of cores). For the case of the PFFT solver, the best results are always obtained when all the available cores are used for MPI processes (without OpenMP threads). Conversely, for the libISF solver the combination between OpenMP and MPI should be chosen with more care; for a low number of cores (less than the number of mesh planes), it seems to be better to use a MPI process per core (with very small differences compared with other combinations). Instead, if a higher number of cores is available, then OpenMP threads must be used to overcome the performance limits imposed by the number of mesh planes, so keeping the number of MPI process below the number of planes. In any case, the maximum number of OpenMP threads that can be used is limited by the actual machine, in this case between 4 (Blue Gene/P machines) and 32 (Curie x86-64).

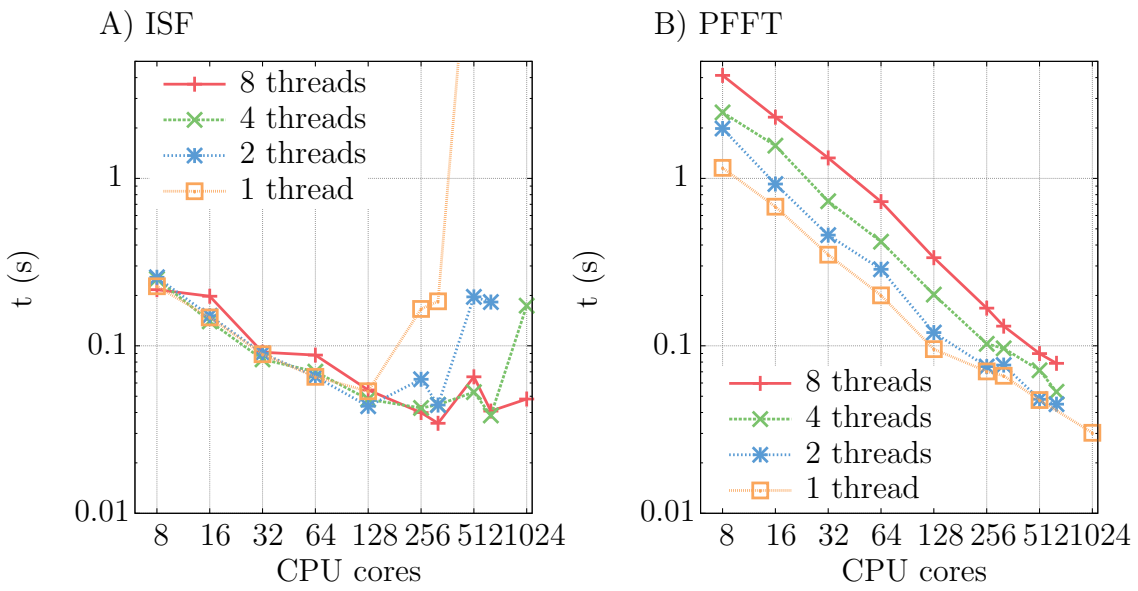


Figure 6.10: Execution times for the system of L_e = 15.8 Å (4,019,679 mesh points) corresponding to the A) libISF solver and to the B) PFFT solver, using from 8 to 1024 cores, and varying the number of OpenMP threads from 1 to 8, in Curie (x86-64). The product of [number of MPI processes] × [number of OpenMP threads] is always equal to the number of used cores. The same trend is also shown in the Ganbo (x86-64) machine.

6.3 Conclusions

In this chapter about Poisson solvers we analysed the relative performance of several implementations of celebrated methods (Fast Fourier Transforms, Interpolating Scaling Function, Fast Multipole Method, Conjugate Gradients and Multigrid) for the calculation of the classical Hartree potential created by charge distributions represented in real-space. The fundamentals of these methods where previously presented in Chapter 3 (Section 3.5), and, in the first part of this chapter (Section 6.1), their corresponding parallel implementations. In the second part, we summarise the computational aspects of the tests we carried out to measure the implementations' relative performances. These tests were run on four supercomputers, two Blue Gene/P (Jugene and Genius) and two x86-64 (Curie and Ganbo). In our tests, we focused on measuring accuracies, execution times, speed-ups, and weak-scaling parameters. Test runs involved up to 4,096 parallel processes, and solved system sizes from about 350,000 mesh points to about 32,000,000 mesh points.

Our results show that the PFFT solver is the most efficient option for a high number of cores, so PFFT should be the default option to calculate the Hartree potential when using a number of cores which is beyond a given (problem dependent) threshold. For lower number of cores, the libISF solver should be preferred. We traced back this different behaviour to the parallelisation strategy of the FFT used in both cases. Indeed, the two-dimensional data decomposition of PFFT goes beyond the one-dimensional data decomposition used in the libISF package. One can point that, for the sake of enhanced efficiency of the Poisson solver, the future implementation of an ISF method which makes use of the PFFT library would be of great interest. The specific number of cores which make PFFT faster than ISF depends on the given problem: the mesh shape, the number of mesh points, and

also the computing facility. In addition, it will also depend on the MPI/OpenMP balance chosen for the ISF solver. For a given number of mesh points, the bigger the deviation of the mesh shape from a cube, the bigger the number of cores for the ISF-PFFT efficiency crossover.

In some special cases, the charges might not lie in equispaced points (e.g. when curvilinear coordinates are used), and so the PFFT and libISF methods cannot be used, as they require the calculation of FFTs. In these cases, the FMM should be chosen instead, since it works, it is linearly-scaling and it is accurate regardless of the charge density's spatial location. The libFMM solver also shows good performance and scaling, yet its accuracy is lower and its execution times are less competitive than those of the PFFT and libISF solvers on the analysed machines. In contrast to libISF, libFMM scales almost linearly up to high values of the number of processes, but since its numerical complexity has a larger prefactor, it would only be competitive with libISF when the number of parallel processes increases significantly. The performance of the Conjugate Gradients solver has a trend similar to that of libFMM, as does the Multigrid solver for low values of the number of processes. Weak-scaling tests show that communication costs are the smallest for the libFMM solver, while they are acceptable for all the others.

ISF and FFT methods are more accurate than FMM, CG, and MG methods. Hence, they should be chosen if accurate calculations of electrostatics are required. However, according to our tests, the accuracy of all the analysed methods is expected to be appropriate for Density Functional Theory and Time-Dependent Density Functional Theory calculations (where the calculation of the Hartree potential is an essential step) because these have other sources of error that will typically have a much stronger impact (see Section 6.2.1). Nevertheless, neither the Multigrid nor the Conjugate Gradients methods can reach acceptable accuracy if the data set is not represented on a compact (spherical or parallelepiped) mesh.

Chapter 7

Memory and performance improvements

Contents

In this chapter we will present recent memory improvements in the Octopus code and we will show its extreme performance. Altough the results to be presented are limited to Octopus, lessons learned in this work can be readily applied to any other real-space mesh based code.

Previously in this work, Octopus was prepared to run with relatively small atomic systems (hundreds of atoms at most), in machines with hundreds of processor cores. In the aim to reach bigger systems with thousands of atoms, it is necessary to use also thousands of processors, and this is the scenario where new problems appear. It is to be stressed that our data transfer improvements correspond to the calculation of the Hartree potential, which is customarily calculated using real-space meshes also for codes using basis sets.

This chapter is structured as follows. In Section 7.1, we explain the strategies we followed to increase the efficiency and optimise the memory management. Some measurements on the current performance of the code are presented in Section 7.2. Finally, in Section 7.3 we outline the main conclusions of this part.

7.1 Improvements in the memory usage

Octopus has been used successfully during the last years to analyse complex nanostructures, but it had some limitations when a high number of processes must be used to simulate systems with thousand of atoms. Most of the limitations are related to the memory requirements, which is a limited resource in any computer. So, we have analysed how Octopus uses the computer memory, to optimise its usage.

7.1.1 Mesh partitioning

The largest data-structures, namely the functions represented on the mesh, have to be distributed between the available computer nodes. An important issue in this domain parallelisation is selecting which points of the mesh are assigned to each processor. This task, known as mesh partitioning, is not trivial for meshes of adaptive shape. Not only the number of points must be balanced between processors but also the number of points in the boundary regions must be minimised. This issue is crucial, because communication costs are directly related to the number of boundary points. An example of a mesh partitioning is previously shown in Figure 4.2 (page 46, each colour represents a domain). Octopus relies on external libraries for this task: Metis [170] and ParMetis [20]. These libraries implement several algorithms and the quality of the partition will depend on the selected one.

Serial partitioning

Previous to this work, Metis was the default library in Octopus to partition the mesh. It is a serial library, so the process that calls it does need all the mesh data, which have to be gathered from all the processes. Moreover, to reduce communication needs, all the processes called the library and, consequently, all of them needed to store the whole data-structure. This is a valid approach if small-medium size meshes are used, but it is unfeasible with million of mesh points, because the amount of memory per process is limited.

Parallel partitioning

With the aim of avoiding the mentioned memory problems with the mesh partitioning, we have adapted Octopus to use a new a highly parallel library called ParMetis. ParMetis is a parallel library implemented in MPI and built on top of the Metis library (version 5.1). As Metis, this library makes partitions of graphs, so a transformation of the Octopus mesh structure into a graph is required previously. We use a graph data-structure to store the neighbour information of all points.

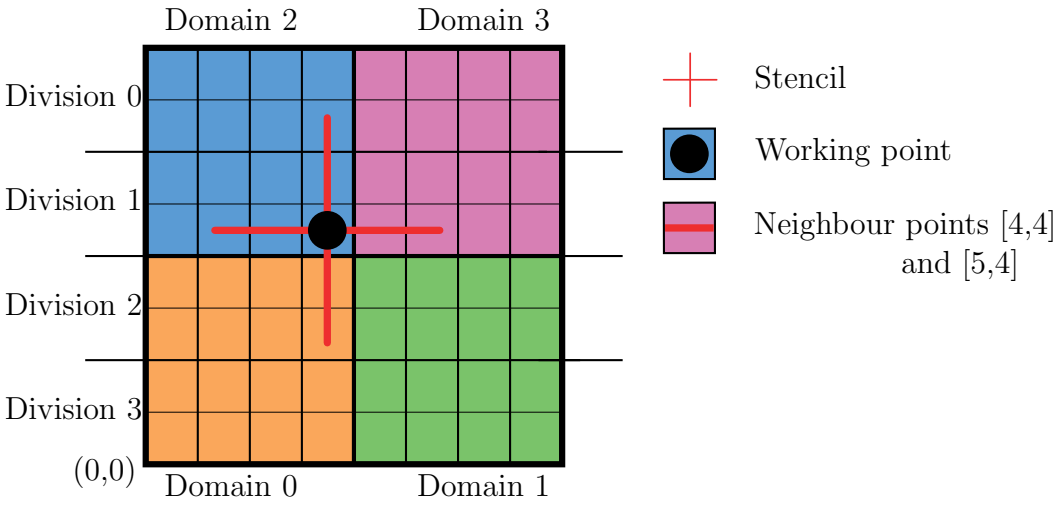


Figure 7.1: A simplified example of the Octopus *mesh partition. The initial (non-optimised) division schedules chunks of* 2×8 *points (division* i *is assigned to process* i*). Afterwards, each process uses ParMetis locally to create the actual domain partitions (4x4 points), aiming to minimise the boundary points.*

Therefore, each mesh point is represented by a vertex in the graph, and neighbours points are connected through an edge. The information about the neighbours of a point is necessary e.g. to calculate the discretised Laplacian operator, which is done using a *stencil* (a given function of the values of a function in a given point and in its neighbours). ParMetis works in parallel and, so, all data-structure are distributed. Each process works with a contiguous chunk of the graph, with local matrices of vertexes and edges, and all the processes know how the graph is initially distributed.

In order to use the new library, we have developed a new distributed version of the mesh partitioning for Octopus. At the beginning, each process obtains an arbitrary (not optimised) portion of the mesh (called "division"), whose size is roughly N/p (being N the number of mesh points and p the number of MPI processes involved in the partitioning). Then, using this initial division, each process calls ParMetis to obtain the actual domain partition of the graph. Finally, local results are informed to the corresponding processes (the real owners of the mesh points).

So, the final domain partition of the mesh is saved in a distributed way. Consequently, when a process works with a neighbour point that does not belong to its partition, it must identify the owner of the point. This information can be obtained from the initial owner of the points, that processed it using ParMetis.

Figure 7.1 shows an example of this procedure. Let us assume that we are working with the blue mesh point with the circle inside, point [3,4], belonging to domain 2 (process 2) and labelled as *working point*. Using a stencil of length 2, it wants to obtain the two neighbours on the right (points [4,4] and [5,4]). According to the initial division of the mesh, it knows that process 1 (division 1) calculated the final owner of those points. So, it will ask process 1 for this information (both points are assigned to process 3) before doing the actual communication.

This new distribution strategy, based in ParMetis, allows us to use only local data to do the final partition/decomposition, reducing greatly the use of memory in each process.

7.1.2 Data transfers between different partition types

In the context of scientific simulation it is common to use external libraries to do standard operations in an highly optimised way. These libraries could use different data distributions. One example of this situation are the different mesh partitions used in Octopus and in its Poisson solvers [19] (the Interpolating Scaling Function (ISF) [14] and the Parallel Fast Fourier Transforms (PFFT) [15] libraries).

Both Poisson solvers are based on FFT methods, where the spatial functions are represented in a cubic mesh. The ISF library splits the mesh into domains which are parallel plane-like parallelepipeds (Figure 7.2C), while PFFT makes a two-dimensional split of the mesh into column-like domains (Figure 7.2B). However, Octopus divides the mesh in a three-dimensional manner into more compact domains (Figure 7.2A). Therefore, the data that a process deals with in the Octopus main program are not the same as the PFFT and ISF libraries use inside, and hence a data transfer is necessary, independently of the shape of the mesh.

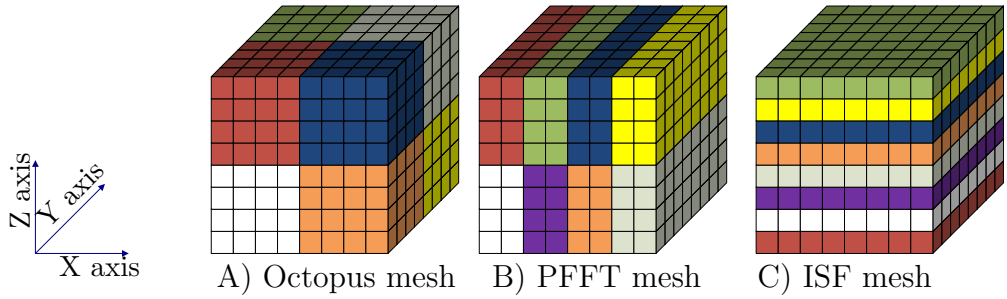


Figure 7.2: Simplified domain decomposition of the simulation mesh. Each little cube represents a mesh point (8^3 points in total) and each colour represents a partition (8 domains). A) Octopus mesh with a 3D domain decomposition; B) PFFT mesh with a 2D decomposition; C) ISF mesh with a 1D decomposition.

The simple way to carry this data transfer out is to gather all the data (density ρ or potential v) in all the processes before distributing them accordingly to the new mesh partition. In fact, this was the option used in the previous version of Octopus. This solution works if the number of processes is not very high, but efficiency decays when the number of processes increases, because of the global MPI communication. Therefore, a new data transfer strategy had to be implemented to be used in massively parallel machines.

We have overcome this problem in a highly efficient way. At the initialisation stage a mapping between the Octopus mesh partition and the FFT mesh partition is established and saved. This mapping is used when running the actual solver to efficiently communicate only the strictly necessary data between processes. Each of the different domain decompositions (for instance, Octopus and FFT partitions) is represented by a MPI group/communicator, that might differ in the number of processes (Octopus might use processes for different parallelisation levels), but includes all the mesh points. At a given time, data points have to be sent from one group to the other. Unfortunately, MPI does not allow to send information between different groups unless they are disjoint, which is not, by definition, the current case. This means that communication will have to be done inside one of the groups. This is not a problem, because we can determine the rank of the receiver

processes through the `MPI_COMM_WORLD` global communicator. Therefore, all the data transfers between processes can be done in a unique `MPI_Alltoallv` function. It is important to note that, using this improvement, each process only transfers each point once. Therefore, the total amount of information that must be sent between all the processes is equal to the number of points in the mesh, and it is independent of the number of processes. The data transfer process has been encapsulated in an specific Fortran module in OCTOPUS,[1] achieving almost perfectly linear parallel scaling.

7.1.3 Linear Combination of Atomic Orbitals

One of the first steps in the calculation of the Ground State for a given system is the construction of an initial guess for the wavefunction and to build the Hamiltonian matrix previous to the Self Consistent Field (SCF) iterations. This process has been implemented in the OCTOPUS code by performing a Linear Combination of Atomic Orbitals (LCAO) obtained from the pseudopotentials. It was proved that this stage had not been efficiently implemented.

We did an optimisation of the LCAO implementation to reduce memory cost and execution time. A memory allocation problem has also been observed during the initialisation, previous to the GS SCF cycle. Mathematically, LCAO consists in a diagonalisation of a matrix of size $N \times N$ (N is equal to the number of orbitals of the atomic system). This is a small matrix for a small system, but with a quadratic scalability. Therefore, it becomes a huge matrix for systems with many states.

The memory allocation obstacle, in a first step, has been overcome using an alternative serial implementation, which reduced the memory requirements to a unique process. Besides, the alternative implementation has decreased the execution time of the LCAO step. Later on, the memory problem can be definitely overcome using the parallel alternative implementation of LCAO, which uses ScaLAPACK external library, instead of the serial LAPACK library. With this parallel library, large matrices are distributed among all the MPI processes. As it has been demonstrated through the benchmark tests presented in reference [171], developments in the LCAO allows to run bigger simulations and, therefore, more interesting systems.

7.1.4 Other improvements

Apart from the memory optimisations explained above, the older version of the code used three matrices of size $p \times p$, as temporary arrays, being p the number of running processes. Those matrices are small with few processes, but they become problematic when we need to use a high number of processes, for intance > 2048. Assuming 8 byte integers, with 2048 processes the size of each matrix is of 32 MiB, yet not very large, but with 8192 processes the size increases to 0.5 GiB, and with 64K processors the size is already prohibitive (32 GiB) for current machines. After implementing a new logic, we got rid of them, because they were only used in the initialisation process.

Other inefficiencies related to initialisation processes (serial evaluation of the local and boundary points, for example) also had impact in the execution time

[1] See the source code of our implementation in the file `src/grid/mesh_cube_parallel_map.F90` at public Subversion repository: http://www.tddft.org/svn/octopus and the documentation in the wiki: http://www.tddft.org/programs/octopus).

when simulating big systems, and have been removed.

7.2 Results

Together with the optimisation of the data flow, the goal of this Section was to address the huge memory requirements of Octopus when running in a large HPC environment. The changes made in the code allow us to run it in a much more efficient manner, opening the door for the simulation of the interaction of light with big biomolecules. In this Section we will first present the improvements in the memory usage, and next will proceed to demonstrate the extreme scalability of the code in its current form. No scalability tests for older versions of the code will be shown, as they were not able to handle the largest systems discussed below.

Tests have been done for different chunks of the Light Harvesting Complex II molecule, of 180, 650, 1365 and 2676 atoms (Figure 1.2, page 7). Those systems have from 452,878 mesh points and 250 electronic states to 4,106,680 points and 3,656 states.

Tests were run using three different computers: a Blue Gene/P, a Blue Gene/Q and the 12 cores island of the Ganbo cluster (x86-64), explained in detail in Chapter 2 (Section 2.2.3). We want to remark that the amount of memory per processor core is relatively low today, and it is not increasing significantly. For instance, in the computers that we have used, the quantity of memory per core goes from 0.5 GiB in the BG/P to 4 GiB in Ganbo, when all available the cores are used to calculate.

7.2.1 Memory measurement

Octopus provides a tool to measure the memory usage and the timing of the most important functions, presented in Chapter 4 (Section 4.6). A deep profiling of the memory usage has been done mainly using this internal profiler and validated with Valgrind's Massif tool. For example, when running the system of 180 atoms in 32 MPI processes, Massif estimates memory usage up to 96 MiB, whereas Octopus estimates 88.5 MiB. This difference is owing to the fact that Octopus does not take into account the memory of linked libraries such as MPI, ISF, BLAS, etc. Despite this small difference, we have used the internal profiler because its execution overhead is much lower than that of Massif tool.

Figure 7.3 shows the memory usage per process of the system of 180 atoms, before and after the optimisations we have done. Memory usage per process is decreasing as more processes are used, but unfortunately the oldest version (4.0) of Octopus tends to increase the use of memory after a relatively small number of processes (256) and, so, makes impossible the use of thousands of processes. On the contrary, the last version has eliminated this increasing tendency. In fact, the memory need has been decreased by a factor of four in the studied range, with a clear tendency to be more efficient when using more processes. To realise the importance of this restriction it has to be considered that the amount of memory per CPU core hardly goes beyond 4 GiB.

7.2.2 Data transfer time measurement

The new implementation of the mesh data transfer between different partitions (explained in Section 7.1.2) shows really good performance as can be observed in Figure 7.4. The time spent transferring data is really low, compared with the execution

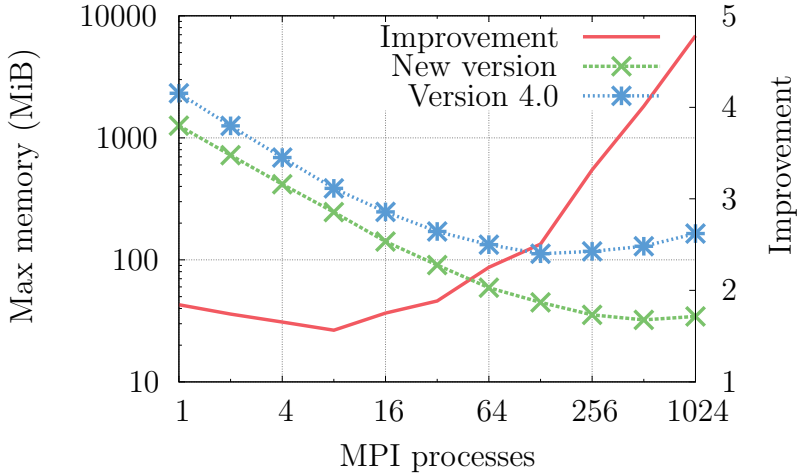


Figure 7.3: Ground State maximum memory usage (from Octopus *profiler) per MPI process for the system of 180 atoms.*

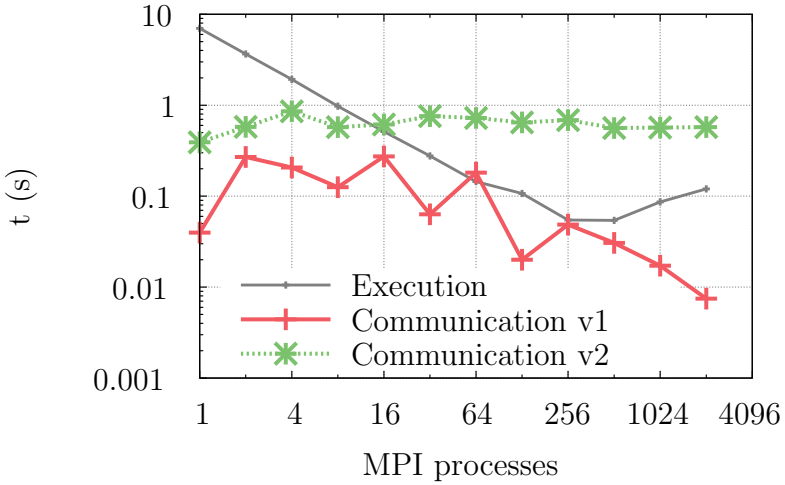


*Figure 7.4: Time for the new parallel communication pattern (*Communication v1*), compared with the old serial communication (*Communication v2*), for a system of 4,019,679 mesh points in the Blue Gene/P machine. While the old version does not scale and becomes a bottleneck, the improvement of the new approach is huge and it has a much better scalability. Total execution time of the Poisson solver is the sum of the* Execution *and* Comm. *times.*

time of the library (PFFT in this case). It is shown a particular case of a cubic mesh of edge 31.6 and spacing of 0.2, which makes a total amount of 4,019,679 mesh points and needs $325^3 = 34,328,125$ cube points for the PFFT. The data transfer accounts from 0.5% to 55% of the total Poisson execution time.

Input/output optimisations

Input/output operations play an important role twice during Octopus runs, and it is necessary to minimise the time involved in these operations, essentially because CPU cores remain idle in the meanwhile, and, thus, they can heavily increase the

total execution time. On the one hand, we need to read and write restart data. As the time slice assigned to an specific run in a HPC centre is limited (and shorter than the real needs), executions must be often stopped and resumed, and a huge amount of data must be saved and reloaded each time. And, on the other hand, calculated density and other observables are usually saved each iteration; in particular, we have to output density data every iteration, in order to be able to plot and analyse execution data.

Although writing restart data to the local scratch drive should be much faster than writing to the global file system (either NFS, Lustre or GPFS), local discs can not be used for those purposes, because surely different processors will be assigned to the application (in fact, some machines do not allow using local discs, to avoid RAM-HD data traffic and latency, or directly, they do not have any hard drive).

Thus, to minimise core hours consumption due to I/O operations, we have developed a new strategy to write restart files more efficiently when only domain parallelisation is available (i.e., in a GS run). In the previous version, data were collected (gather) in a process and then written from this process to the file system. As a first option, we tried to use MPI I/O functions to allow all processes to access the same file, but results shown no improvements. So we have change the write strategy; now, all the processes in the domain write down to different files. First, a gather of different data to each process is done, and then, all together write the files (a file per process). The performance improvement can be seen, for example, in Hydra: with the previous strategy we needed 1 h 26 min 45 s to write the restart data for the simplified trimer[2], whereas the improved writing only takes 7 min 35 s. Time is reduced by more than an order of magnitude, and we obtain a bandwidth of 2.15 GB/s, compared with the previous one of only 0.19 GB/s.

Besides, we only write data in a binary format, and we have developed an external tool to process offline these binary files, to obtain data in different formats; for instance, to visualise them.

7.2.3 Execution times

Improvements made in the Octopus code allowed us to simulate bigger systems than ever. Next paragraphs show the results obtained with both executions modes of Octopus: Ground State and Time-Dependent.

Ground State

Figure 7.5 shows the execution time of the Ground State calculation of systems of 180, 441 and 650 atoms in Blue Gene/P and Ganbo.

The trend for all the atomic systems and for both machines is equivalent, being Ganbo 3 times faster. The scalability is more than acceptable; for instance, the 180 atoms system scales well up to 256 processes, while with the 650 atoms system the scalability is almost perfect until 256 processes, showing improvements up to 2048 MPI processes.

Time-Dependent

The execution times and relative speed-ups of the Time-Dependent simulations in the BG/Q machine are show in Figure 7.6, with systems of 180, 650, 1365 and 2676

[2] 15,644,983 mesh points, 8400 states, 120 MB per restart file and a total of 980 GB

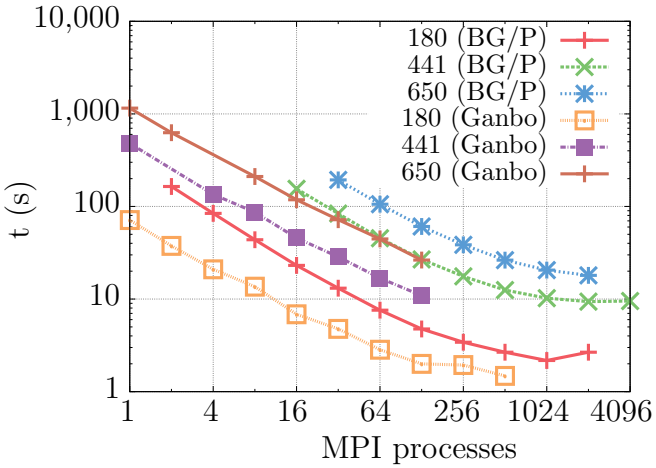


Figure 7.5: Ground State iteration execution time in Blue Gene/P and Ganbo (x86-64)

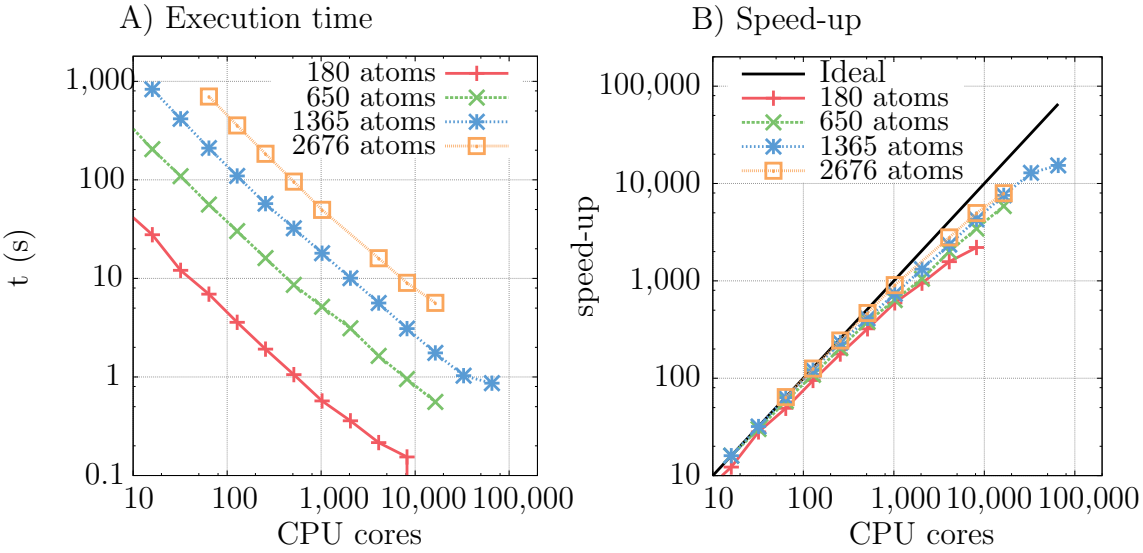


Figure 7.6: 180, 650, 1365 and 2676 atoms system running in the Blue Gene/Q up to 65,536 MPI processes (1 MPI process per core).

atoms. Runs are made using one MPI process per core, thus 16 processes per node. Extremely good scalability is shown for the smallest system, that, because of the reduction of the memory needs, can be now run in only 4 MPI processes, and it is still efficient with 8K processes. For the system of 650 atoms, there is no sign of saturation up to 16K processes. The system of 1365 atoms is highly parallel up to 32K processors and we have been able to run it in 64K processors, improving by far previous results, for example those presented in Chapter 5 (Section 5.3.2). Finally, almost perfect scalability has been reached with the system of 2676 atoms up to 16K processors (we were limited to the tests shown here, because we had a limited CPU quota for our project), which was not possible to run it before in Chapter 5.

Scaling tests also were done in the BG/P machine, showing a very similar behaviour. In this case, runs were made using one MPI process per node and four OpenMP threads (one per node core). Specially remarkable is the performance we have obtained with the 650 atoms system in this machine, which does not saturate until the maximum available CPU cores (128K). Deserves to mention as well the system of 180 atoms, which is really efficient from 4 up to 64K MPI processes. The

new efficient memory usage allowed us to run in this machine a bigger system of 5879 atoms in 32K processors.

7.3 Conclusions

Octopus is a scientific software package based on TDDFT theory, which is being used successfully by dozens of research groups around the world. To date, it was mostly used to analyse medium-sized complex nanostructures. Modifications were necessary to enable the code to work with bigger systems (thousands of atoms and beyond).

In this chapter we have analysed and solved some of the limitations that Octopus had in this new scenario, mainly the amount of memory and the transfers of the main data-structure in some phases of the computation. Both problems have been solved in a very efficient manner. On the one hand, we have used ParMetis to do the partitioning of the mesh using only local data, avoiding the use of the whole data mesh in each process and, consequently, reducing greatly memory needs. On the other, we have optimised data transfer between processors when different domain decompositions must be used. Additionally, problems found at the initialisation with the LCAO technique had been overcome, using less memory and parallelising it. Finally, a new strategy for the input/output has been implemented, which is now much faster.

The benefits of these improvements are significant. Octopus uses now a much more limited amount of memory per processor, and it is ready to simulate larger systems. Indeed, we have shown excellent scalability up to 64k and 128k cores, almost independent of the system size (number of atoms) beyond a given size. Although we have shown the executions of different chunks of the Light Harvesting Complex II, the real aim is to be able to run the entire Light Harvesting Complex II with more than 17,000 atoms. The recent improvements make possible to obtain scientifically relevant results for these large systems at a reasonable time.

Chapter 8

Light Harvesting Complex–II

Contents

After solving the computational problems in the previous chapters, we will show a real use case in the present one. Following, we will see actual TDDFT simulations of the Light Harvesting Complex II (LHC–II) molecule (with its physical analysis). The final aim of this chapter (and future works too) is to have a better understanding of the photosynthesis process. By the way, we show the efficiency of such big simulations.

LHC–II has for decades been the focus of experimental research aimed at understanding the biological aspects of what is truly the power-house of life on Earth and arguably one of the most important proteinaceous structures in Nature. More recently, the extremely high light–capturing quantum efficiency of LHC–II has attracted a great deal of interest from a completely different group of scientists, namely those working toward developing solar energy capture devices.

LHC–II is a trimeric protein assembly with three–fold symmetry (Figure 8.1). Each monomer unit contains, complexed within the protein framework, 14 chlorophyll (Chl) molecules (both type *a* and *b*) that are the key functional units for light–harvesting [31, 172]. Additionally, each monomer contains four secondary carotenoid chromophores (lutein (×2), neoxanthin and violoxanthin) that play important roles in photoprotection under conditions of overly strong illumination [173, 174, 175]. In all, LHC–II contains approximately 17,000 atoms, 7,000 of which belong to the chromophores. Usually bound within cell membranes in vivo, LHC–II possesses a robust structure and maintains structural integrity during purification and crystallisation meaning that the detailed structure is accurately known and also that it has become possible to perform experimental studies of isolated LHC–II [176, 177, 178].

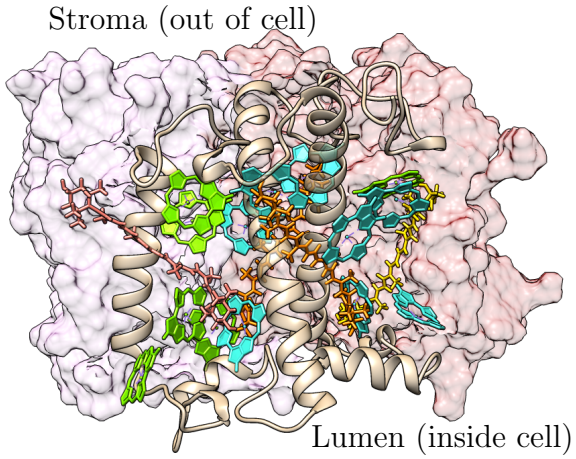


Figure 8.1: LHC–II trimer, showing locations of protein, carotenoid and Chl molecules (Chl a *in blue and Chl* b *in green) of one monomeric unit. For a better visualisation, hydrogen atoms, Chl phytyl chains and primary structure of the protein are not shown.*

Models based on excitonic coupling between the chlorophyll units and/or modulation of the chromophores' energy levels by the protein environment have been invoked in an attempt to explain the high efficiency of this process [179, 180, 181]. Whereas in bacterial light harvesting antennae the former, excitonic, mechanism appears to be dominant, there is evidence pointing towards a more micro–environment driven mechanism being most important in plant LHC–II [36, 177, 178]. Although a balance between both effects will contribute to the functioning of LHC–II, it is extremely difficult to experimentally discern the relative importance of the two.

8.1 Preparation of LHC–II geometry

The initial geometry of LHC–II was extracted from the crystal structure of the spinach major Light Harvesting Complex II (PDB accession code 1RWT) [182] obtained from the Protein Data Bank and optimised using the PM7 semi-empirical

electronic structure method with the program MOPAC [183, 184]. Several lipid molecules originating in the experimental crystallisation mixture that remained in the crystal structure, β-nonylglucoside and digalactosyl diacyl glycerol, were removed as these were not part of the biological unit. Furthermore, these were expected to have little or no impact on the excited state electron dynamics of the LHC–II chromophores.

Hydrogens were added using the UCSF Chimera molecular modelling package [185]. At this stage it was necessary to select the protonation state of acidic residues located at the stromal and lumenal membrane interfaces of LHC–II. It was decided to protonate all acidic side chains at these interfaces since it was considered that in the cellular environment the charges associated with these side chains would be effectively shielded by counter-ions in solution or by association with other charged bio-molecular species. Since accurate modelling of this shielding was beyond the scope of the present study, the simplified approach of capping these side chains with protons was considered most likely to provide a reasonable model of the charge environment in these regions of the LHC–II structure. The initial structure prepared in this way possessed a net charge of +30. The total number of atoms in this structure was 17,280.

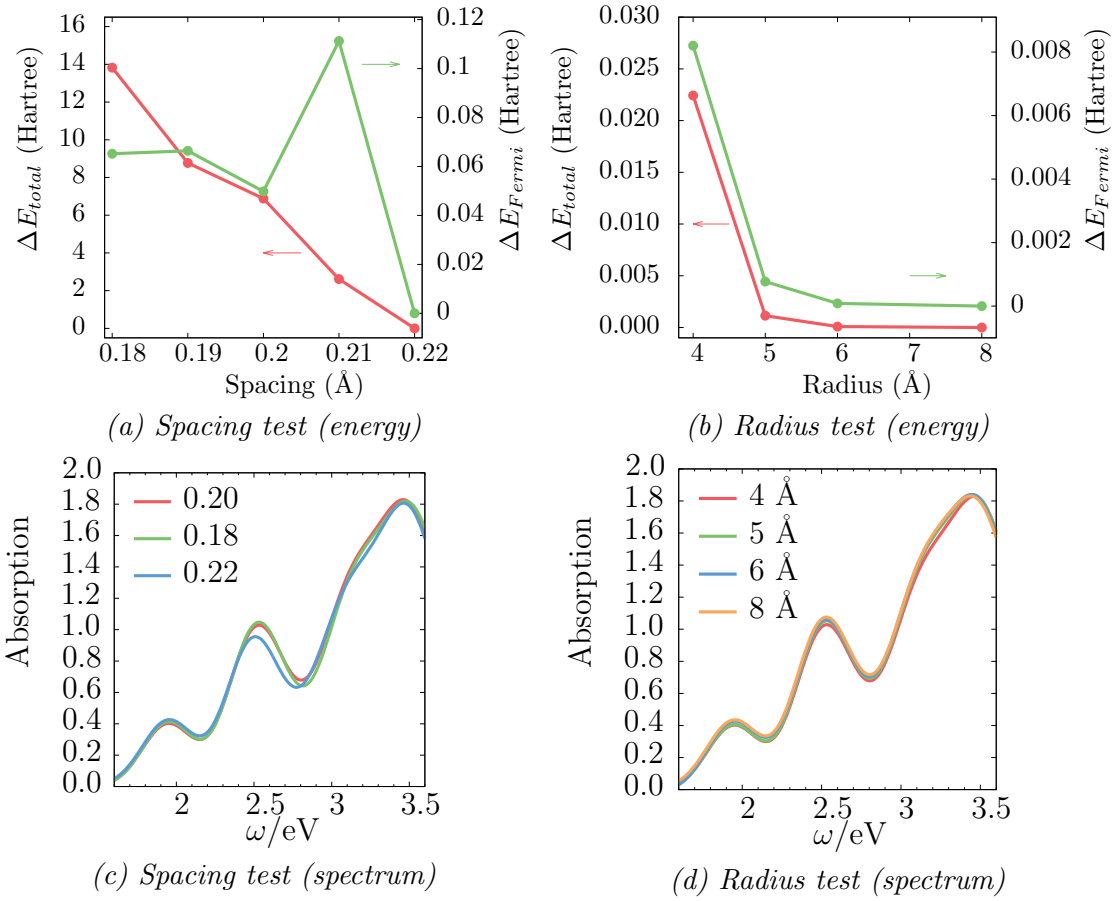
Following the initial placement of the hydrogens on the LHC–II structure it was necessary to refine their positions and, since there was a strong possibility that protonation states could have been misassigned, it was decided to use a method that would permit the migration of protons away from unfavourable sites or towards more favourable ones. This requirement means that approaches that used classical force fields, and therefore had fixed bonding connectivities, could not be employed. A solution to this problem lies in quantum mechanical electronic structure methods, where no bonds are enforced and atomic nuclei are free to move in the potential created by the electronic structure of the system being studied.

Due to the size of the LHC–II system, the only feasible choice in this case was to use a semi-empirical electronic structure method. The recently developed PM7 Hamiltonian, as implemented in the MOPAC semi-empirical electronic structure package, was selected, as this has been shown to reproduce very well molecular geometries calculated with high level quantum chemical methods, and also provides a good model of weaker interactions such as hydrogen-bonding and dispersion forces [183]. After several rounds of optimisation, followed by checking for and dealing with protons that had migrated from their original sites, an overall charge neutral LHC–II structure was obtained. This final structure contained 17,250 atoms.

8.2 Preparation of input parameters

8.2.1 Testing convergence of calculation parameters

All calculations relating to the absorption spectrum of the LHC–II complex were performed with the real-space/real-time code Octopus. The PBE exchange-correlation functional was used in all calculations [186, 187]. Convergence of the Ground State energy and the resultant absorption spectra with respect to integration mesh spacing and the radius of atom-centred integration spheres was tested in the ranges 0.18 Å to 0.22 Å (mesh spacing) and 4 Å to 8 Å (sphere radius). Chlorophyll *a*602 was removed from the optimised LHC–II structure and used for these tests. Testing the



(a) Spacing test (energy)

(b) Radius test (energy)

(c) Spacing test (spectrum)

(d) Radius test (spectrum)

Figure 8.2: Plots 8.2a and 8.2b show the total (red) and Fermi (green) energy differences taking as a reference the energetically most stable Ground State solution for each case. The resulting spectra computed using the different parameters are shown in plots 8.2c and 8.2d

mesh spacing, the sphere radius was held fixed at 4 Å. Similarly, testing the sphere radius, the mesh spacing was held fixed at 0.20 Å.

Time-propagation of the Ground State charge densities obtained with these parameter combinations was performed with an initial perturbation of 0.05 eV and time-step of 0.003 $\hbar$/eV (∼0.002 fs). The propagation was carried out for a total of 10,000 iterations and the resulting time-dependent dipole polarisabilities were Fourier transformed in order to obtain absorption spectra. The results of these tests are shown in Figure 8.2.

Analysis of the effect of varying the spacing parameter indicated that, although convergence is not reached at spacing 0.20 Å for the Ground State calculation, the absorption spectrum obtained appears to be converged at this value (Figure 8.2a). On the other hand, the radius parameter seems to be converged at 4 Å since the energy difference between 4 and 5 Å is smaller than 0.025 eV (Figure 8.2b). The spectra obtained for all values of the radius parameter are essentially identical and showed only a small dependence on the spacing parameter within the range tested (Figures 8.2c and 8.2d).

Eigensolver	Avg. iteration time (s)	Number of iterations
RMMDIIS	1407.1	121
Lanczos	5850.1	33
CG_new	36968.3	25

Table 8.1: Eigensolver tests done in Ganbo with a spacing of 0.5 Å for the complete monomer. RMMDIIS is the only feasible solver, if we want to have a converged GS in a bounded time.

8.2.2 Special case: Complete monomer

Special tuning of the input parameters was needed for the complete monomer because the previously calculated ones, whilst optimal for the smaller systems studied, proved impossible to converge when applied to the problem of calculating the Ground State density of the complete monomer system. Such a big system was never attempted to run with a DFT code to our knowledge and it has many states close each other (essentially degenerate), greatly complicating its convergence.

Different eigensolvers were examined: Conjugate Gradients [188], Preconditioned Lanczos scheme [189] and Residual Minimization Scheme, Direct Inversion in the Iterative Subspace (RMMDIIS) [125]. Those solvers were not only used independently, but also combining them. Unfortunately, Conjugate Gradients method has a low performance in terms of execution time and makes it usage prohibitive. In average, it is more than 25 times slower than the RMMDIIS solver. Thus, we tried to reduce the eigensolver iterations to only 5 (from the default of 25), aiming to reduce the fluctuations. No advance in the relative density is shown during the SCF iterations. Similar problem appears with the Lanczos eigensolver. Changing the number of eigensolver iterations to 5, 25, 50 and 100 does not change this situation. Eingesolver tests are done in 128 cores (16 cores island) in the cluster Ganbo (Chapter 2, Section 2.2.3), using a spacing of 0.5 Å (to be able to fit memory) and 100 extra states. The execution times are summarised in Table 8.1, showing that the only affordable execution times are obtained using RMMDIIS solver; Lanczos and CG_new solvers are too slow (specially the last one).

A range of extra states was analysed for RMMDIIS, from 0 to 1270. For this solver the theory says that around 10-20% of extra states are needed to correctly converge. However, our tests have shown that the usage of 100 extra states (which is the 1.28%) is the best option with this particular huge system. Smearing quantities were modified in order to improve the convergence. The tried values were 0.2, 0.1 and 0.01.

Two mixing strategies were used: linear and Broyden [190, 191]. This mixing density oscillates from 5% to 15% (percentage of the new density). Using a low amount of the current iteration density and bigger amount of the previous iteration density, allows to reduce the fluctuations. Besides mixing density, mixing potential also was tried. At the end, none of the alternatives worked for the complete monomer, and, we could not converge it.

8.3 Results

The electronic structure of the Chl antenna molecules of LHC–II is well understood as are the nature of the excitations involved in photo–absorption by Chl in the relevant regions of the solar spectrum. Due to obvious difficulties in applying these methods to systems of this size, many of these studies have focused on the individual chromophores in isolation, with some simple model of the complex micro–environment, or by a small explicit portion of the complex.

Here, we wish to report a first–principles study of the electronic absorption spectrum of the full LHC–II Chl chromophore network in which all atoms were treated at the same theoretical level using TDDFT. All TDDFT calculations used the PBE Exchange-Correlation functional [187, 186] and were performed with the real-space TDDFT codeOctopus.

8.3.1 Isolated chromophores

In order to initially establish the linear optical response characteristics for comparison with the LHC–II-bound chromophores the absorption spectra of each was calculated in isolation to remove all influence from the protein surroundings. The geometries used were those obtained from the PM7 optimisation of the entire LHC–II system, thus the chromophores were in their relaxed protein-bound conformations so that any difference in their response would be purely due to the lack of microenvironment. Within the protein complex we can distinguish two general groups of chromophores, as show in Figure 8.3: 14 chlorophyll molecules (8 chlorophyll *a* and 6 chlorophyll *b*), and 4 carotenoid molecules (2 lutein, 1 xanthophyll and 1 neoxanthin).

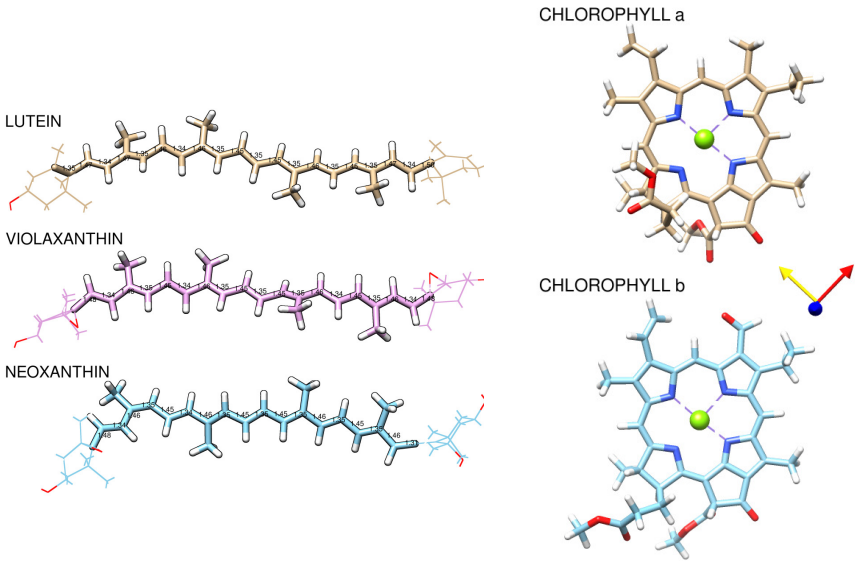


Figure 8.3: Structure of the chromophores present in the LHC–II complex. The phytyl chain of the chlorophyll molecules has been omitted to aid visualisation of the central macrocycle. The terminal rings of the carotenoid molecules are de-emphasised using wire representation in order that the conjugated π-system be more easily visualised. Carotenoid backbone bond length alternation (BLA) is shown. Oxygen atoms are coloured in red, nitrogen atoms in blue and hydrogen atoms in white. Remaining atoms are carbon atoms.

Initial calculations focused on the spectra of the isolated Chl chromophores (Figure A.1 of Appendix A) with the geometries obtained from the structure optimisation of the full complex (Figure A.2 of Appendix A). The peaks corresponding to the Q– and Soret–bands were calculated to occur between 1.9–2.0 eV and 2.5–2.8 eV, respectively (Figure 8.4). The experimental Q-band energy obtained from the LHC–II spectrum is 1.86 eV, whilst the Soret–band contains two peaks at 2.62 and 2.85 eV (assigned to Chl *b* and *a*, respectively) [22]. Recent *in vacuo* experimental studies of the Chl absorption spectra put the lowest energy excitation (Q–band) at 1.93 eV (Chl *a*) and 1.98 eV (Chl *b*), whilst the Soret–band was found to have its maximum at 3.06 eV (Chl *a*) and 3.00 eV (Chl *b*) [192, 193]. Good performance of the TD-PBE method for the Q–band excitation and slightly worse performance for the Soret–band might reasonably be expected since the former is a predominantly single electron excitation, which is well described by TDDFT, whilst the latter is known to possess significant multiple–electron character that is not properly described in the TDDFT framework [194]. However, tests of the current methodology with the optimised geometries that were used in the analysis of the experimental *in vacuo* absorption spectra [192, 193] indicate that the real-space TD-PBE approach in fact performs very well when compared with the experimental Q– and Soret–band maxima (see Figure A.4 of Appendix A).

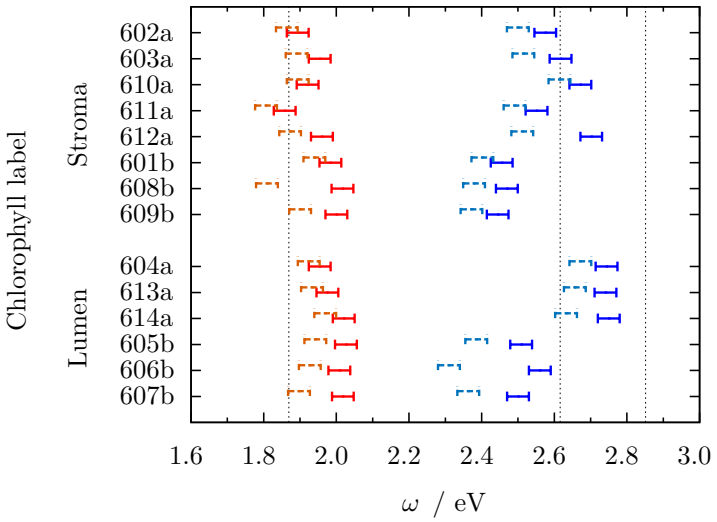


Figure 8.4: Distribution of peak energies in Chl TDDFT spectra. Q–band occur between 1.9-2.0 eV and the Soret–band between 2.5-2.9 eV. The numbering from PDB file 1RWT. Solid lines correspond to calculations on isolated chlorophylls and dashed lines to spectra of the individual chlorophyll within the LHC–II monomer chlorophyll network. Vertical lines indicate the position of the maxima of the Q– (1.86 eV) and Soret–band (2.62 and 2.85 eV) found experimentally [22].

In addition, we calculated the absorption spectra of the carotenoid molecules in isolation after extraction from the optimised LHC–II structure. Although the applicability of TDDFT methods to the calculation of the excited states of polyene molecules such as the carotenoids is known to be problematic [194], the agreement between computed and experimental results are surprisingly good (see Figure A.2 of Appendix A). This can most likely be attributed to the nature of the geometry optimisation which incorporated the effects of the LHC–II complex environment

[195]. However, due to the lack of the polyene chain motions in the simulations, whilst the energy of the peak maximum was well reproduced the peaks themselves were overly intense and interfered with the interpretation of the chlorophyll network spectrum.

8.3.2 Effect of protein environment on chlorophyll spectra

A test evaluation of the effect of the protein environment on the spectrum of chlorophyll was performed for Chl 608*b* (numbering from PDB file 1RWT). This Chl from the stromal–side of LHC–II was selected because of its more complete protein surroundings as compared with other Chl which overlapped significantly with neighbouring chromophores (Figure 8.5). The plots in Figure 8.6 compare the calculated spectra with an experimental LHC–II absorption spectrum [22] and show that the overall effect of the protein environment is a red–shifting of the Chl spectrum with the low energy Q–band being shifted by -0.1 eV but the Soret–band undergoing a larger shift of -0.3 eV. The red–shift of 0.1 eV for the low energy excitation is the same as that seen in the experimental *in vacuo* spectra of both the *a* and *b* forms of Chl [192]. A second effect observed was that the ratio of peak intensities Q:Soret (1:10 *in vacuo*) also changed on inclusion of the protein environment to 1:2, the same as seen for the experimental spectrum in Figure 8.6. The appearance of the small peak at ∼1.65 eV is most likely due to transfer of oscillator strength from the red–shifted Soret–band peaks. Similar promotion of dark states was also observed in the local dipole analysis of the individual Chls (see Figure A.2).

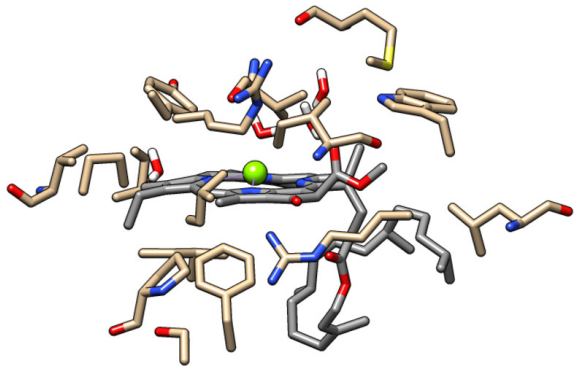


*Figure 8.5: Model system used in evaluation of effect of protein environment on chlorophyll absorption spectrum. Chlorophyll 608*b *carbons shown in grey, amino acid side chain carbons in brown. Except for water molecules, all hydrogen atoms are omitted to aid visualisation. This Chl was selected because of its more complete protein surroundings as compared with other Chl which overlapped significantly with neighboring chromophores (Figure A.8 of Appendix A).*

8.3.3 Simplified monomer, dimer and trimer

Based on the data obtained from the Chl 608*b*/protein study it was decided to remove the majority of the protein from the optimised monomer structure, not only to reduce the dimension of the electronic structure calculation but also to facilitate convergence of the Ground State DFT calculation, which had been found to be problematic when the full environment was included. Amino acid side chains directly complexed to Chl Mg^{2+} ions and charged species lying close to the chlorin

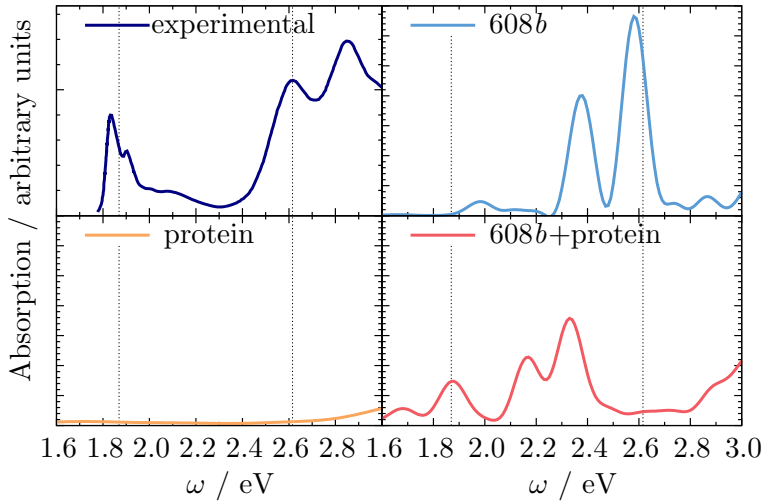


*Figure 8.6: Influence of protein environment on the spectrum of Chl 608*b. *Vertical lines indicate the experimental position of the Q–band (1.86 eV) and Chl* b *component of the Soret–band (2.62 eV) [22].*

ring were included since these were expected to play a major role in modulating the details of the monomer spectrum (Figure A.1 of Appendix A). Carotenoid molecules were also removed from the chromophore network because they had been found to produce an anomalously large peak close to the Q–band of chlorophyll molecules as mentioned above. The degree to which inclusion of the carotenoid molecules alters the absorption spectrum of the chromophore network can be seen in Figure A.5. The simplified system contained 2025 atoms (corresponding to 5200 electronic states) per monomer and the resulting calculations were found to converge rapidly.

The spectrum of the LHC–II monomer, dimer and trimer chlorophyll networks are shown in Figure 8.7 and compared with that obtained from experiment. The experimental absorption spectrum (filled grey curve) displays the Q– and Soret–band features corresponding to the Chl excitations at approximately 1.85 and 2.6/2.8 eV for the b/a type, respectively [22]. The spectra obtained from our real-time TDDFT calculations show good agreement with experiment. Very little deviation was observed for the Q–band, while the Soret–band was found to display a slightly larger deviation of $\sim$ 0.35 eV. The monomer and dimer spectra are in excellent agreement in terms of absorption energies, with the intensity of the dimer spectrum being approximately doubled (although scaled by 0.5 in Figure 8.7 for ease of comparison with the monomer spectrum). From this, and a localised charge-density analysis (Figures A.6 and A.7 of Appendix A), it can be seen that the contribution of each monomer to the total spectrum is essentially independent and only minor inter-monomer perturbations exist.

8.3.4 Contribution of each chlorophyll

In order to study the contribution of the constituent Chl chromophores, the absorption spectrum for each were computed *in situ* by selecting the corresponding charge density for each one based on a Bader charge–topological analysis (Figure A.7) and studying its time-dependent induced dipole moment obtained from the TDDFT charge density propagation. Analysed in this way, these spectra differ from the separate Chl spectra in Figure A.2 due to the fact that they include the electro-

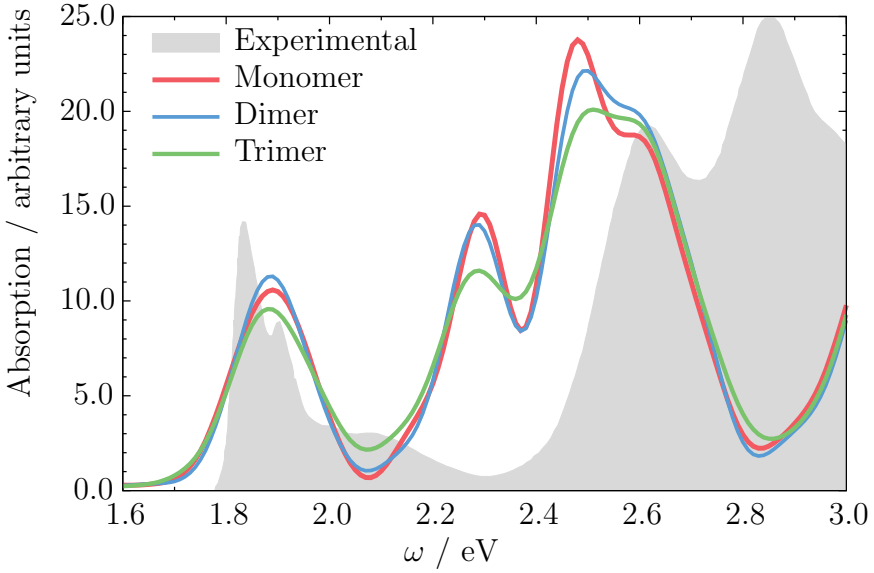


Figure 8.7: LHC–II monomer (red), dimer (blue) and trimer (green) TDDFT spectrum. The absorption intensity of the dimer has been scaled by half and the trimer by one third to aid in comparison of the spectra. The filled gray curve is the experimental absorption spectrum of LHC–II [22]. Here the lower absolute accuracy of TDDFT for the predominantly multi-electron Soret–band excitation energies can be seen.

static influence of the relaxed DFT charge density of the remainder of the system, thus providing an accurate representation of the absorption spectrum of a given chromophore within the LHC–II minus the shift due to the protein environment which, as discussed above, is essentially constant.

Figure 8.4 shows the variation in the location of the peaks for each individual chlorophyll molecule within the chromophore network. In general, a red–shift of the individual spectra is seen as a result of the electrostatic interactions with the other chromophores. The influence of the chromophore network on the individual chlorophylls appears to be in agreement with what was found previously for the protein environment effect i.e. that an electrostatic interaction is sufficient to explain the observed spectral shifts. The largest observed effect was a strong red–shift ($\sim$ 0.2 eV) on the Soret–band of the Chl *b* molecules chlorophylls in the lumenal side of the LHC–II complex. Contrary to the non-specific electrostatic red–shift, this larger alteration in absorption energy can be attributed to an electronic interaction due to the a close overlap between 605*b*, 606*b* and 607*b*. This is highlighted in Figure A.8 which shows schematically the spatial distribution of the chlorophyll molecules within a monomer.

Figure 8.8 shows the contributions of the Q– and Soret– band of the different chlorophylls obtained from the local Bader analysis of the total monomer charge density. This approach facilitates the separation of the chromophore contributions to the photo-absorption profile of the LHC–II system. These data indicate that the LHC–II Q–band can be attributed in most part to Chl *a*. Furthermore, in agreement with the data obtained from the local dipole analysis, this suggests that the small Q–band shoulder observed experimentally is due to differences in absorption energy between lumenal and stromal chromophores. Given that energy transfer pathways are determined in this system by exciton frequencies, our results suggest that the

most probable energy transfer after an excitation in the Q–band region will take place following an energy descent path from lumen to stroma, in agreement with data obtained using femtosecond transient absorption spectroscopy [196].

Two key observations were made possible regarding the Soret–band in the present work. Our simulations indicate that the Chl *a* molecules located on the lumenal side are subject to a blue–shift relative to the Chl *a* on the stromal side of LHC–II, leading to a broadening of the Soret–band of the LHC–II complex. This effect enables LHC–II to absorb light across a broader range of frequencies in the solar spectrum. In addition, our calculations confirm that the peak at approximately 2.3 eV (2.6 eV experimentally) is due to the Chl *b* molecules. As a result of our local dipole analysis it was found that lumenal chlorophylls (605*b*–606*b*–607*b*) produce a broadening of the Soret–band to the red. These results, which display a significant difference in Soret–band energy in the chlorophylls in different locations within the complex, suggest a route for the LHC–II system to absorb blue light on the stromal side and then transfer the excitation energy down the energy gradient to the center of the complex where 607*b* is located.

The spectral alterations outlined above are most pronounced for a small subset of the chromophores studied; the majority of the chlorophyll molecules' absorption spectra do not undergo large alterations when the influence of the chromophore network is taken into account (Figure A.2). This suggests that the overall electronic structure of the chlorophyll molecules is not altered significantly when the entire network is included, and the observed slight red–shift can be attributed to a Coulombic effect. This suggests that most of the Exciton Energy Transfer (EET) mechanism can be well described by the electric dipole moment coupling scheme proposed by T. Förster [197]. However, due to the specific relative orientation of lumenal Chl *b* molecules, a strong red–shift and change of the shape of the Soret–band are observed, which suggest that non-electrostatic effects are relevant and Förster's theory will not be adequate to study the EET mechanism close to center of the LHC–II trimer.

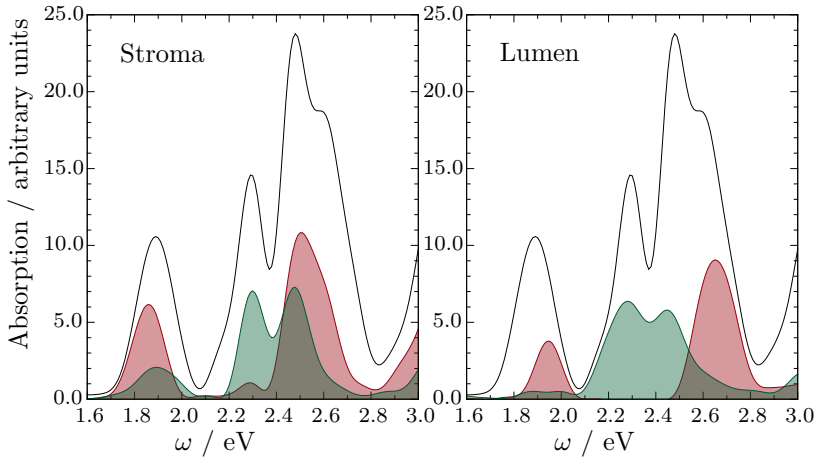


Figure 8.8: LHC–II monomer TDDFT spectrum. Contribution by chlorophyll type at stromal side (left) and lumenal side (right) of the membrane. Chl a *are in red and chl* b *in green, the gray line shows the calculated monomer spectrum. Individual chlorophyll contributions are summed to give the curves shown here.*

8.4 Conclusions

In summary, the absorption spectrum of the full chlorophyll network of the major light harvesting complex, LHC–II, has been simulated using first–principles electronic structure methodology. We note that this is probably the largest system treated by PM7 and TDDFT up to date. We have demonstrated that the electrostatic and non-electrostatic effect of the environment as well as specific chlorophyll-chlorophyll interactions are essential for theoretical investigations of systems of this type and that these effects combine to produce significant modulation of the LHC–II absorption spectrum. Local analysis of the contribution of each chlorophyll has been introduced and shown to be a powerful tool for use in studying possible interaction mechanisms between chromophores.

This work confirms that after light absorption in the Q–band region of the LHC–II spectrum the most probable excitation energy propagation will take place from lumen to stroma (Figures 8.8 and 8.9). In addition, the most significant/strongest optical absorption is to be expected in the Soret–band region and can be attributed to the chlorophylls situated in the stromal side of the thylakoid membrane. This suggests that light energy of around 2.5 eV absorbed in the stromal half of LHC–II should be transferred to the low–energy lumenal centre of the trimer complex formed by the grouping of Chl *b* molecules. On–going studies using real-time propagation TDDFT on the chlorophyll network will quantify the potential coupling between chromophores and distinguish its different components in order to understand the EET of the LHC–II.

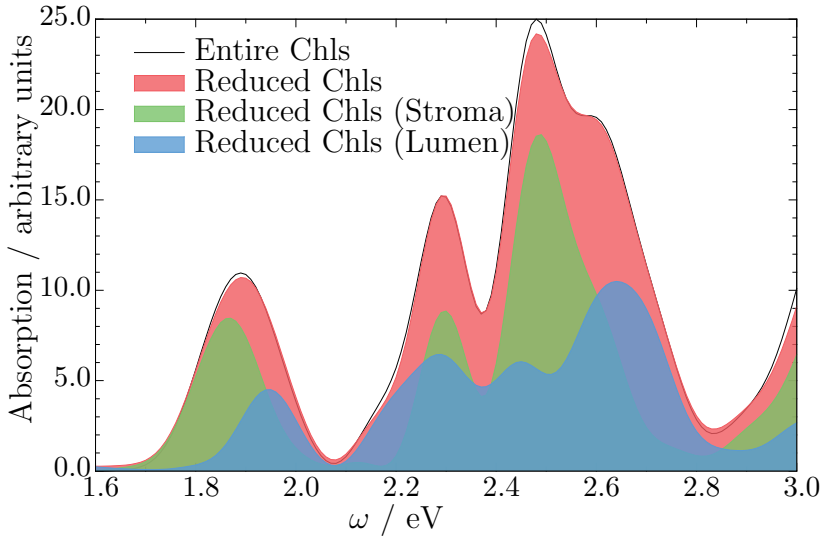


Figure 8.9: Contribution on the absorption spectra of chlorophyll networks (solid line) from stromal (green shade) and lumenal (blue shade) half-sides of LHC–II. This representation evidences that the chlorophylls located on the stroma have higher probablity of light absortion than the chlorophylls located on the lumenal part. Orange shade are obtained as a superposition of all chlorophylls contribution, and confirms the validity of the model used to select the Bader volumes.

Chapter 9

Conclusions

Contents

To finish with this PhD thesis we present the conclusions in this final chapter.

9.1 Main conclusions

Computer simulation is one of the most powerful techniques scientists use to understand all kind of physical phenomena. Supercomputer facilities offer today the possibility of using hundred of thousands of computing cores to accelerate the execution of those simulations, giving us the opportunity to investigate the behaviour of very complex systems. For instance, HPC enables the simulation of atomic and molecular systems according to the fundamental equations of quantum mechanics. In particular, this thesis is concerned with the use of massively parallel computers to study the light harvesting complexes in green plants and, so, the first steps of the photosynthesis.

The study is centred on a concrete scientific software package, OCTOPUS, although the obtained results are not limited to this application. OCTOPUS is mostly developed for Density Functional Theory (DFT) and Time-Dependent Density Functional Theory (TDDFT) calculations, and it is used to study by first principles the properties of the excited states of large biological molecules and complex nanostructures.

Partnership for Advanced Computing in Europe (PRACE) has offered us the possibility of using the fastest supercomputers in Europe: Jugene and Juqueen (JSC, Germany), Curie (CEA, France), Fermi (Cineca, Italy) and MareNostrum II and III (BSC, Spain). Also, we have used Genius and Hydra (RZG, Germany). Other clusters have been used for software development and tests.

Obtaining a linear speed-up from these systems is far from obvious, and great efforts must be devoted to understand the nature of the scientific code, the parallel executions, the need of data communication among lot of processes, the efficient use of an always limited amount of memory per core, etc. This thesis has dealt with all those items, with a clear objective in mind: the opportunity of using thousands of processors to simulate real size molecular systems, with the aim of understanding their physical behaviour.

At the starting of the project, the code was available to run in parallel, yet it was hardly tested; i.e. the actual high performance was unknown. Although they were promising, tests up to only hundreds of processes were done. Hence, a more extensive study was prepared. In this first study, Ground State and Time-Dependent calculations modes were tested, and runs up to 8,192 processors were performed in Jugene, MareNostrum II, Vargas and Ganbo machines. In the TD tests the scalability was high until 2-4K processes, but with more processes the efficiency was poor (less than 40%). Different parallelisation strategies were tried and, in general, a bottleneck was found in the Poisson solver. That solver accounted for a little percentage of the total execution time when using few processes, but, for instance, with 8K processes it took the half of the iteration execution time. So, after this first bunch of tests, we concluded that there was a several performance limitation because of a non optimal Poisson solver implementation.

Consequently, an extensive research of the Poisson solvers alternatives has been done. We have analysed the relative performance of several implementations of celebrated methods for solving the Poisson equation: Parallel Fast Fourier Transforms, Interpolating Scaling Function, Fast Multipole Method, Conjugate Gradients and Multigrid. Namely, with Interpolating Scaling Function and Parallel Fast Fourier

Transforms Poisson solvers the code runs efficiently up to thousands of processes. Tests were run on four supercomputers, two Blue Gene/P (Jugene and Genius) and two x86-64 (Curie and Ganbo), and we have measured accuracies, execution times, speed-ups, and weak-scaling parameters. Test runs involved up to 4,096 parallel processes, and solved system sizes from about 350,000 mesh points to about 32,000,000 mesh points. Our results show that the PFFT solver is the most efficient option for a high number of cores. Nevertheless, for lower number of cores, the ISF solver should be preferred.

Not only is an improved Poisson solver required for the efficient use of a high number of processes. Additional modifications for OCTOPUS have been necessary to enable the code to work with bigger systems (thousands of atoms and beyond). Thus, we have analysed and solved some of the limitations that OCTOPUS had in this new scenario, mainly the amount of memory per core and the transfers of the main data-structure in some phases of the computation. Both problems have been solved in a very efficient manner. On the one hand, we have used ParMetis to do the partitioning of the mesh using only local data, avoiding the use of the whole data mesh in each process and, consequently, reducing greatly memory needs. On the other, we have optimised data transfer between processors when different domain decompositions must be used.

The benefits of these improvements are significant. OCTOPUS uses now a much more limited amount of memory per processor, and it is ready to simulate larger systems. Indeed, we have shown excellent scalability up to 64k and 128k cores, almost independent of the system size. With this new version of OCTOPUS, including all the improvements, we have been able to do a TDDFT simulation of the LHC–II complex. Specifically, the absorption spectrum of the full trimeric chlorophyll network (as well as of the smaller chunks of the LHC–II) has been simulated using first-principles electronic structure methodology, The full system has 8400 atomic states and needs 15,644,983 domain points to be simulated, totalling 1.8 TB for the restart.

We have demonstrated that the electrostatic and non-electrostatic effect of the environment as well as specific chlorophyll-chlorophyll interactions are essential for theoretical investigations of systems of this type, and that these effects combine to produce significant modulation of the LHC–II absorption spectrum. Local analysis of the contribution of each chlorophyll has been introduced and shows to be a powerful tool for use in studying possible interaction mechanisms between chromophores.

It is really amazing to see how, from a first principles calculation (only having the molecule geometry and the implementation of the quantum theories), we can obtain results which agree with the experiments. Thus, we can say that the main objectives of this doctoral thesis have been achieved: to be able to efficiently run a complex TDDFT simulation in a very large number or processors, and, therefore, to be able to study large molecular systems using HPC facilities.

During the development of this PhD thesis I have participated in several HPC projects. As a consequence of the collaborative work I have done with the OCTOPUS development team, I have acquired a deep experience and understanding of the functioning of a HPC environment, both from the user point of view and also as an administrator of a local cluster. Software development includes the use of parallel MPI libraries and scripting, profilers, Fortran and so on.

9.2 Future works

First principle TDDFT code opens a new world of opportunities to simulate problems that were not possible before. TDDFT simulations are now open to new size of problems, and interesting phenomena can be simulated. Even if we have demonstrated that it is possible to run the complete monomer of LHC–II complex, we have not been able to converge to the Ground State. Thus, this is a work that must be done in a near future.

Regarding to the Poisson solvers, there is still room for the improvements. The ideal solver for our case will be a future implementation of an ISF method which makes use of the PFFT library, instead of the internal FFT library. Indeed, the two-dimensional data decomposition of PFFT goes beyond the one-dimensional data decomposition used in the ISF package. Therefore, the scalability will be improved.

Apart from a higher number of processing cores, the huge performance of new supercomputers is based also on the use of accelerators, mainly vector coprocessors (Intel PHI) and GPUs. In fact, Octopus offers already the possibility to compile for OpenCL. So, it should be analysed how to use these possibilities to reach better speed-ups.

9.3 Publications and Courses

The output of this thesis can be summarised in 3 peer-reviewed papers, 1 more submitted, 1 peer-reviewed congress article, another congress, a book, 3 posters and several assistance to workshops (some with oral presentations). A complete list is following:

Peer-reviewed papers

- **Insights into the modulation of light absorption by chlorophyll in green plants**
  J. Jornet-Somoza, J. Alberdi-Rodriguez, B.F. Milne, X. Andrade, M.A.L. Marques, F. Nogueira, M.J.T. Oliveira, and A. Rubio
  Physical Chemistry Chemical Physics **submitted**

- **Real-space grids and the Octopus code as tools for the development of new simulation approaches for electronic systems**
  X. Andrade, D. Strubbe, U. Giovannini, A.H. Larsen, M.J.T. Oliveira, J. Alberdi-Rodriguez, A. Varas, I. Theophilou, N. Helbig, M. Verstraete, L. Stella, F. Nogueira, A. Aspuru-Guzik, A. Castro, M.A.L. Marques and A. Rubio
  Physical Chemistry Chemical Physics **February 2015**.

- **A survey of the parallel performance and accuracy of Poisson solvers for the electronic structure calculations**
  P. García-Risueño, J. Alberdi-Rodriguez, M.J.T. Oliveira, X. Andrade, M. Pippig, J. Muguerza, A. Arruabarrena, A. Rubio
  Journal of Computational Chemistry Volume 35, Issue 6, pages 427-444 **5 March 2014**.

- **Time-dependent density-functional theory in massively parallel computer architectures: the OCTOPUS project**
  X. Andrade, J. Alberdi-Rodriguez, D.A. Strubbe, M.J.T. Oliveira, F. Nogueira, A. Castro, J. Muguerza, A. Arruabarrena, S.G. Louie, A. Aspuru-Guzik, A. Rubio, M.A.L. Marques
  Journal of Physics: Condensed Matter Vol. 24 Issue 23 **May 2012**.

Peer-reviewed conference paper

- **Recent Memory and Performance Improvements in OCTOPUS Code**
  J. Alberdi-Rodriguez, M.J.T. Oliveira, P. García-Risueño, F. Nogueira, J. Muguerza, A. Arruabarrena, A. Rubio
  International Conference in Computational Science and Its Applications – ICCSA 2014 Lecture Notes in Computer Science Volume 8582, 2014, pages 607-622 **2 July 2014**.

Other publications

- **Memory Optimization for the OCTOPUS Scientific Code**
  J. Alberdi-Rodriguez, A Rubio, M.J.T. Oliveira, A. Charalampidou, D. Folias
  PRACE white paper **21 January 2015**

- **Examination of chlorophyll-chlorophyll excitation energy transfer based on local induced dipoles analysis**
  J. Jornet-Somoza, J. Alberdi-Rodriguez, F. Noguiera, A. Rubio
  50th Symposium on Theoretical Chemistry **September 2014**

- **High-performance electronic structure calculations: optimization of the evaluation of the Hartree potential**
  P. García-Risueño, J. Alberdi-Rodriguez, M.J.T. Oliveira, X. Andrade, M. Pippig, J. Muguerza, A. Arruabarrena, A. Rubio
  White nights of material science: From physics and chemistry to data analysis, and back **16-20 June 2014**

- **Analysis of performance and scaling of the scientific code OCTOPUS**
  J. Alberdi-Rodriguez
  LAP Lambert Academic Publishing, ISBN: 978-3-8484-1835-0, **March 2012**.

- **A survey of the performance of classical potential solvers for charge distributions**
  J. Alberdi-Rodriguez, P. García-Risueño
  Fast Methods for Long-Range Interactions in Complex Systems workshop **September 2011**.

- **Improving OCTOPUS towards a new generation of HPC systems**
  X. Andrade, J. Alberdi-Rodriguez.
  Jülich Blue Gene/P extreme scaling workshop 2010 **April 2011**.

- **OCTOPUS: a versatile tool for realtime TDDFT simulation of thousands of atoms**
  X. Andrade, J. Alberdi-Rodriguez
  Psi-k 2010 conference **2010**.

This is a list of presentations.

- **Optimization of new algorithms in Octopus for methods to calculate the photosynthesis**
  J. Alberdi-Rodriguez
  7th RES Users' Conference BSC (Barcelona Supercomputing Center). Barcelona **13/09/2013**.

- **Presentación de proyecto con acceso a PRACE: Artificial photosynthesis**
  J. Alberdi-Rodriguez, X. Andrade, M.A.L. Marques, F. Nogueira, A. Castro A. Rubio
  Supercomputación Científica en Euskadi. BSC (Barcelona Supercomputing Center). Donostia **08/03/2013**.

- **First principles modeling with Octopus: massive parallelization towards petaflop computing and more**
  A. Castro, J. Alberdi-Rodriguez, A. Rubio
  RES Scientific Seminar of Parallel Simulations in the Network. Zaragoza **30/11/2010**.

- **Profiling and Optimizing**
  J. Alberdi-Rodriguez, X. Andrade
  Basic techniques and tools for development and maintenance of atomic-scale software CECAM (Centre Européen de Calcul Atomique et Moléculaire). Zaragoza **21-25/06/2010**.

This is a list of courses that I attended.

- **Introduction to SuperMUC – the new Petaflop Supercomputer at LRZ**
  Munich, *Leibniz-Rechenzentrum - der Bayerischen Akademie der Wissenschaften* **08-11/07/2013**.

- **European-U. S. Summer School on HPC Challenges in Computational Sciences**
  Dublin, *PRACE - XSEDE* **24-28/06/2012**.

- **Programming and Tuning Massively Parallel Systems**
  Barcelona, *Barcelona Supercomputing Center* **18-22/07/2011**.

- **PRACE Training Week**
  Barcelona, *Barcelona Supercomputing Center* **06-09/09/2010**.

- **Time-Dependent Density-Functional Theory: Prospects and Applications**
  Benasque, *Centro de ciencias Benasque Pedro Pascual* **02-10/01/2010**.

- **BSC — PRACE code porting and optimization workshop**
  Barcelona, *Barcelona Supercomputing Center* **21-23/10/2009**.

- **CUDA for scientific computing**
  Espoo (Finland), *CSC — IT Center for Science Ltd* **10-11/09/2009**.

- **Numerical schemes for disperse equations**
  Derio, *BCAM — Basque Center for Applied Mathematics* **08-12/02/2010**.

This is a list of meetings that I attended.

- Octopus **meeting**
  Jena, *The Martin Luther University of Halle-Wittenberg* **26-30/01/2015**.

- Octopus **meeting**
  Lyon, *Université Lyon* **22-23/10/2012**.

- **LibPSP coding party**
  Coimbra, *University of Coimbra* **29/08/2011-02/09/2011**.

This is a list of collaboration that I had.

- Coimbra, Universidade de Coimbra

- Berlin, Humboldt-Universität zu Berlin

Bibliography

[1] Andrade, X.; Strubbe, D.; De Giovannini, U.; Larsen, A. H.; Oliveira, M. J. T.; Alberdi-Rodriguez, J.; Varas, A.; Theophilou, I.; Helbig, N.; Verstraete, M. J.; Stella, L.; Nogueira, F.; Aspuru-Guzik, A.; Castro, A.; Marques, M. A. L.; Rubio, A. *Phys. Chem. Chem. Phys.*, **2015**, pp –. ii

[2] Andrade, X.; Alberdi-Rodriguez, J.; Strubbe, D. A.; Oliveira, M. J. T.; Nogueira, F.; Castro, A.; Muguerza, J.; Arruabarrena, A.; Louie, S. G.; Aspuru-Guzik, A.; Rubio, A.; Marques, M. A. L. *Journal of Physics: Condensed Matter*, **2012**, *24*(23), 233202. ii, 42, 64, 67

[3] Castro, A.; Appel, H.; Oliveira, M.; Rozzi, C. A.; Andrade, X.; Lorenzen, F.; Marques, M. A. L.; Gross, E. K. U.; Rubio, A. *Phys. Status Solidi B*, **2006**, *243*, 2465. ii, 42, 64, 67, 140

[4] Marques, M. A. L.; Castro, A.; Bertsch, G. F.; Rubio, A. *Comput. Phys. Commun.*, **2003**, *151*, 60. ii, 42, 64, 140

[5] Amdahl, G. M. Validity of the single processor approach to achieving large scale computing capabilities. in *AFIPS sping joint computer conference*, volume 30, pp 483–485. AFIPS Press, April 1967. iii

[6] Gustafson, J. L. *Communications of the ACM*, May 1988, *31*(5), 532. iii, 12

[7] Dongarra, J.; Luszczek, P. in *Encyclopedia of Parallel Computing*, Padua, D., Ed., pp 1033–1036. Springer US, 2011. iii, 14

[8] Hohenberg, P.; Kohn, W. *Phys. Rev.*, **1964**, *136*(3B), B864. iii, 27

[9] Kohn, W.; Sham, L. J. *Phys. Rev.*, **1965**, *140*(4A), A1133. iii, 27

[10] Runge, E.; Gross, E. K. U. *Phys. Rev. Lett.*, **1984**, *52*(12), 997. iii, 28

[11] Marques, M. A.; Gross, E. K. in *A Primer in Density Functional Theory*, Fiolhais, C.; Nogueira, F.; Marques, M., Eds., volume 620 of *Lecture Notes in Physics*, pp 144–184. Springer Berlin Heidelberg, 2003. iii, 26, 28

[12] Marques, M. A.; Maitra, N. T.; Nogueira, F. M. *Fundamentals of Time-Dependent Density Functional Theory*, volume 837. Springer Berlin Heidelberg, 2012. iii, 28

[13] Alberdi-Rodriguez, J. *Analysis of performance and scaling of the scientific code Octopus.* LAP LAMBERT Academic Publishing, 2010. iv, 64

[14] Genovese, L.; Deutsch, T.; Neelov, A.; Goedecker, S.; Beylkin, G. *J. Chem. Phys.*, **2006**, *125*(7), 074105. Genovese, L. and Deutsch, T. and Neelov, A. and Goedecker, S. and Beylkin, G. v, 33, 42, 64, 66, 86, 135

[15] Pippig, M. *SIAM J. Sci. Comput.*, **2013**, *35*, C213 . v, 42, 65, 66, 86, 135

[16] Kabadshow, I.; Dachsel, H. The Error-Controlled Fast Multipole Method for Open and Periodic Boundary Conditions. in *Fast Methods for Long-Range Interactions in Complex Systems*, IAS Series, Volume 6, Forschungszentrum Jülich, Germany, 2010. CECAM. v, 37, 66, 139

[17] Bolten, M.; Fahrenberger, F.; Halver, R.; Heber, F.; Hofmann, M.; Kabadshow, I.; Lenz, O.; Pippig, M.; Sutmann, G. ScaFaCoS, C subroutine library. http://scafacos.github.com/. v, 66

[18] Arnold, A.; Fahrenberger, F.; Holm, C.; Lenz, O.; Bolten, M.; Dachsel, H.; Halver, R.; Kabadshow, I.; Gähler, F.; Heber, F.; Iseringhausen, J.; Hofmann, M.; Pippig, M.; Potts, D.; Sutmann, G. *Phys. Rev. E*, Dec 2013, *88*, 063308. v, 66

[19] García-Risueño, P.; Alberdi-Rodríguez, J.; Oliveira, M. J. T.; Andrade, X.; Pippig, M.; Muguerza, J.; Arruabarrena, A.; Rubio, A. *Journal of Computational Chemistry*, **2014**, *35*(6), 427. vi, 86

[20] Karypis, G.; Kumar, V. Parallel multilevel k-way partitioning scheme for irregular graphs. in *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '96, Washington, DC, USA, 1996. IEEE Computer Society. vi, 84

[21] Alberdi-Rodriguez, J.; Oliveira, M. J.; García-Risueño, P.; Nogueira, F.; Muguerza, J.; Arruabarrena, A.; Rubio, A. in *Computational Science and Its Applications – ICCSA 2014*, Murgante, B.; Misra, S.; Rocha, A. M. A.; Torre, C.; Rocha, J. G.; Falcão, M. I.; Taniar, D.; Apduhan, B. O.; Gervasi, O., Eds., volume 8582 of *Lecture Notes in Computer Science*, pp 607–622. Springer International Publishing, 2014. vi

[22] Siefermann-Harms, D. *Biochim. Biophys. Acta - Reviews on Bioenergetics*, **1985**, *811*(4), 325. vii, 99, 100, 101, 102, 128

[23] García-Risueño, P.; Ibáñez, P. E. *International Journal of Modern Physics C*, **2012**, *23*, 1230001. 2, 64

[24] Hemminger, J.; Fleming, G. R.; Ratner, M. A. Directing matter and energy: five challenges for science and the imagination. a report from the basic energy science advisory committee. Technical report, U.S. Department of Energy, 2007. 5

[25] Fleming, G. R.; Ratner, M. A. *Physics Today*, **2008**, *61*, 28. 5

[26] Richter, H., Ed. *Materials for Key Enabling Technologies*. European Science Foundation, Brussels, second edition, 2011. 5

[27] *A European strategy for Key Enabling Technologies – a bridge to growth and jobs*. European Commission, Brussels, 2012. 5

[28] Long, S. P. *Plant Cell Environ.*, **2014**, *37*, 19. 5

[29] Ray, D. K.; Mueller, N. D.; West, P. C.; Foley, J. A. *PLOS One*, **2013**, *8*, e66428. 5

[30] Ruban, A. V.; Johnson, M. P.; Duffy, C. D. P. *Energy Environ. Sci.*, **2011**, *4*, 1643. 5

[31] Fleming, G. R.; Schlau-Cohen, G. S.; Amarnath, K.; Zaks, J. *Faraday Discuss.*, **2012**, *155*, 27. 5, 94

[32] Allakhverdiev, S. I.; Thavasi, V.; Kreslavski, V. D.; Zharmukhamedov, S. K.; Klimov, V. V.; Ramakrishna, S.; Los, D. A.; Mimuro, M.; Nishihara, H.; Carpentier, R. *J. Photochem. Photobiol. C: Photochem. Rev.*, **2010**, *11*, 101. 5

[33] Krassen, H.; Schwarze, A.; Friedrich, B.; Ataka, K.; Lenz, O.; Heberle, J. *ACS Nano*, **2009**, *3*, 4055. 5

[34] Berardi, S.; Drouet, S.; Francas, L.; Gimbert-Surinach, C.; Guttentag, M.; Richmond, C.; Stoll, T.; Llobet, A. *Chem. Soc. Rev.*, **2014**, *43*, 7501. 5

[35] Searle, S. Y.; Malins, C. J. *Biomass Bioenerg.*, **2014**, *65*, 3. 5

[36] Palacios, M. A.; de Weerd, F. L.; Ihalainen, J. A.; van Grondelle, R.; van Amerongen, H. *J. Phys. Chem. B*, **2002**, *106*, 5782. 5, 94

[37] Flynn, M. J. *IEEE Transactions on Computer*, September 1972, *C-21*(9), 948. 10

[38] Fatahalian, K.; Houston, M. *Communications of the ACM*, October 2008, *51*(10), 50. 10

[39] Klimovitski, A. *Intel Developer UPDATE Magazine*, March 2001. 10

[40] Advanced Micro Devices, Inc. *3DNow! technology manual*, March 2000. 10

[41] Amdahl, G. M. Validity of the single processor approach to achieving large scale computing capabilities. in *AFIPS spring joint computer conference*, volume 30 of *AFIPS '67 (Spring)*, pp 483–485, New York, NY, USA, April 1967. ACM. 12

[42] Moore, G. E. *Electronics*, **1965**, *38*(8). 14

[43] Sharma, S.; Hsu, C.-H.; Feng, W.-c. Making a case for a green500 list. in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pp 8–pp. IEEE, 2006. 15

[44] Yoo, A. B.; Jette, M. A.; Grondona, M. Slurm: Simple linux utility for resource management. in *Job Scheduling Strategies for Parallel Processing*, pp 44–60. Springer, 2003. 16

[45] Georgiou, Y. *Contributions for Resource and Job Management in High Performance Computing*. PhD thesis, PhD thesis, LIG, Grenoble-France, 2010. 16

[46] Gilge, M. et al. *IBM System Blue Gene Solution Blue Gene/Q Application Development*. IBM Redbooks, 2013. 16

[47] Sosa, C.; Knudson, B. *IBM system Blue Gene solution: Blue Gene/P application development*. IBM International Technical Support Organization, 2008. 17

[48] Sosa, C.; Knudson, B. *IBM System Blue Gene Solution: Blue Gene/P Application Development*. Redbooks. IBM, November 2009. 17, 18

[49] Stephan, M. *JUGENE – An overview*. Juelich forschungszentrum, February 2008. 18

[50] *Power Architecture Community Newsletter*, February 2005. 18

[51] Gibbs, B.; Arasaratnam, O.; Arenburg, R.; Elkin, B.; Grima, R.; Kelly, J. *The IBM eServer BladeCenter JS20*. Redbooks. IBM, March 2005. 18

[52] Dongarra, J.; Foster, I.; Fox, G.; Gropp, W.; Kennedy, K.; Torczon, L.; White, A. *Sourcebook for parallel computing*. Morgan Kaufmann, San Franciso, USA, 2003. 20

[53] Chapman, B.; Jost, G.; van der Pas, R. *Using OpenMP*. Scientific and Engineering Computation series. The MIT Press, Cambrige, Massachsetts, London England, 2007. 20

[54] Gropp, W.; Lusk, E.; Skjellum, A. *Using MPI*. Scientific and Engineering Computation series. The MIT Press, Cambrige, Massachusetts, London England, 1999. 20

[55] El-Ghazawi, T.; Carlson, W. *UPC: distributed shared memory programming*. John Wiley and Sons, 2005. 20

[56] Wadleigh, K. R.; Crawford, I. L. *Software Optimization for High Performance Computing*. Prentice Hall PTR, 2000. 21

[57] Goedecker, S.; Boulet, M.; Deutsch, T. *Computer Physics Communications*, **2003**, *154*(2), 105. 21, 58

[58] Mitchell, D. J.; Prince, T. **2013**. 22

[59] Harrison, W. A. *Electronic structure and the properties of solids: the physics of the chemical bond*. Courier Dover Publications, 2012. 24

[60] Burke, K. et al. *Department of Chemistry, University of California*, **2007**. 25, 28

[61] Cramer, C. J. *Essentials of computational chemistry: theories and models*. John Wiley & Sons, 2013. 25, 30

[62] Park, D. *Introduction to the quantum theory*. Courier Dover Publications, 2012. 25

[63] Perdew, J.; Kurth, S. in *A Primer in Density Functional Theory*, Fiolhais, C.; Nogueira, F.; Marques, M., Eds., volume 620 of *Lecture Notes in Physics*, pp 1–55. Springer Berlin Heidelberg, 2003. 26

[64] Leach, A. R. *Molecular modelling: Principles and applications*. Pearson Education - Prentice Hall, Harlow, 2nd edition, 2001. 30

[65] Nayfeh, M. H.; Brussel, M. K. *Electricity and magnetism*. John Wiley & sons, 1985. 30

[66] Castro, A.; Rubio, A.; Stott, M. J. *Canadian Journal of Physics*, **2003**, *81*, 1151. 30, 31, 32, 38, 67

[67] Olivares-Amaya, R.; Stopa, M.; Andrade, X.; Watson, M. A.; Aspuru-Guzik, A. *The Journal of Physical Chemistry Letters*, **2011**, *2*(7), 682. 30

[68] Watson, M. A.; Rappoport, D.; Lee, E. M. Y.; Olivares-Amaya, R.; Aspuru-Guzik, A. *The Journal of Chemical Physics*, **2012**, *136*(2), 024101. 30

[69] Cooley, J. W.; Tukey, J. W. *Math. Comput.*, **1965**, *19*, 297. 31, 32

[70] Martyna, G. J.; Tuckerman, M. E. *J. Chem. Phys.*, **1999**, *110*, 2810. 32, 38

[71] Rozzi, C. A.; Varsano, D.; Marini, A.; Gross, E. K. U.; Rubio, A. *Phys. Rev. B*, **2006**, *73*, 205119. 32, 47, 64

[72] Van Loan, C. *Computational frameworks for the fast Fourier transform*, volume 10. Society for Industrial Mathematics, 1992. 33

[73] Bluestein, L. I. *IEEE Trans AU*, **1970**, *18*(4), 451. 33

[74] Genovese, L.; Neelov, A.; Goedecker, S.; Deutsch, T.; Ghasemi, S. A.; Willand, A.; Caliste, D.; Zilberberg, O.; Rayson, M.; Bergman, A.; Schneider, R. *The Journal of Chemical Physics*, **2008**, *129*(1), 014109. 33, 66

[75] Daubechies, I. *Ten Lectures on Wavelets*. SIAM, Philadelphia, PA, 1998. 33

[76] Goedecker, S. *Wavelets and their application for the solution of partial differential equations in physics*. Presses Polytechniques et Universitaires Romandes, 1998. 33, 66

[77] Greengard, L. F.; Rokhlin, V. *J. Comp. Phys.*, **1987**, *73*, 325. 34, 139

[78] Greengard, L. F.; Rokhlin, V. *The Rapid Evaluation of Potential Fields in Three Dimensions*. Springer Press, Berlin, Heidelberg, 1988. 34, 139

[79] Cheng, H.; Greengard, L. F.; Rokhlin, V. *J. Comp. Phys*, **1999**, *155*, 468. 34, 139

[80] Greengard, L.; Rokhlin, V. *Acta Numerica*, **1997**, *6*(-1), 229. 34, 35, 37, 59, 139

[81] Dachsel, H. *J. Chem. Phys.*, **2010**, *132*, 119901. 34, 66, 135, 138, 139

[82] Cipra, B. A. *SIAM news*, **2000**, *33, 4*. 34

[83] White, C. A.; Johnson, B. G.; Gill, P. M. W.; Head-Gordon, M. *Chem. Phys. Lett.*, **1994**, *230*, 8. 34

[84] White, C. A.; Johnson, B. G.; Gill, P. M. W.; Head-Gordon, M. *Chem. Phys. Lett.*, **1996**, *253*, 268. 34

[85] Strain, M.; Scuseria, G.; Frisch, M. *Science*, **1996**, *271*, 51. 34, 139

[86] White, C. A.; Head-Gordon, M. *J. Chem. Phys.*, **1994**, *101*, 6593. 34

[87] Saad, Y. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003. 37, 38

[88] Chelikowsky, J. R.; Troullier, N.; Saad, Y. *Phys. Rev. Lett.*, **1994**, *72*, 1240. 37, 45

[89] García-Risueño, P.; Echenique, P. *J. of Phys. A: Math. and Theor.*, **2012**, *45*, 065204. 37

[90] Hestenes, M. R.; Stiefel, E. *J. of research of the national bureau of standards*, **1952**, *49*, 409. 37, 66

[91] Fernández-Serra, M. V.; Artacho, E.; Soler, J. M. *Phys. Rev. B*, **2003**, *67*, 100101. 38

[92] Wesseling, P. *An introduction to multigrid methods*. John Wiley & Sons, 1992. 38

[93] Zhang, J. *J. Comp. Phys.*, **1998**, *149*, 449. 38

[94] Briggs, W. L. *A multigrid tutorial*. Wiley, New York, 1987. 38

[95] Brandt, A. *Math. Comput*, **1977**, *31*, 333. 38

[96] Trottenberg, U.; Oosterlee, C.; Schüller, A. *Multigrid*. Academic Press, 2001. 38, 40, 67

[97] Rostgaard, C.; Jacobsen, K. W.; Thygesen, K. S. *Phys. Rev. B*, **2010**, *81*, 085103. 38

[98] Briggs, E. L.; Sullivan, D. J.; Bernholc, J. *Phys. Rev. B*, **1996**, *54*, 14362. 38

[99] Beck, T. T. *Rev. Mod. Phys.*, **2000**, *72*, 1041. 38

[100] Torsti, T.; Heiskanen, M.; Puska, M. J.; Nieminen, R. M. *Int. J. Quant. Chem.*, **2003**, *91*(2), 171. 38

[101] Mortensen, J. J.; Hansen, L. B.; Jacobsen, K. W. *Phys. Rev. B*, **2005**, *71*, 035109. 38

[102] Shapira, Y. *Matrix-Based Multigrid: Theory and Applications*. Numerical Methods and Algorithms. Kluwer Academic Publishers, 2003. 38

[103] Mandel, J.; McCormick, S. *J. Comput. Phys.*, **1989**, *80*(2), 442. 38

[104] Munshi, A., Ed. *The OpenCL Specification*. Khronos group, Philadelphia, 2009. 42

[105] Bertsch, G. F.; Iwata, J.-I.; Rubio, A.; Yabana, K. *Phys. Rev. B*, **2000**, *62*(12), 7998. 44

[106] Yabana, K.; Bertsch, G. F. *Phys. Rev. B*, **1996**, *54*(7), 4484. 44

[107] Castro, A.; Marques, M. A. L.; Alonso, J. A.; Rubio, A. *Journal of Computational and Theoretical Nanoscience*, **2004**, *1*(3), 231. 44

[108] Castro, A.; Marques, M. A. L.; Alonso, J. A.; Bertsch, G. F.; Rubio, A. *Eur. Phys. J. D*, **2004**, *28*, 211. 44

[109] Takimoto, Y.; Vila, F. D.; Rehr, J. J. *J. Chem. Phys.*, **2007**, *127*(15), 154114. 44

[110] Yabana, K.; Bertsch, G. F. *Phys. Rev. A*, **1999**, *60*, 1271. 44

[111] Varsano, D.; Espinosa Leal, L. A.; Andrade, X.; Marques, M. A. L.; di Felice, R.; Rubio, A. *Phys. Chem. Chem. Phys.*, **2009**, *11*, 4481. 44

[112] Marques, M. A. L.; Castro, A.; Malloci, G.; Mulas, G.; Botti, S. *J. Chem. Phys.*, **2007**, *127*(1), 014107. 44

[113] Aggarwal, R.; Farrar, L.; Saikin, S.; Andrade, X.; Aspuru-Guzik, A.; Polla, D. *Solid State Commun.*, **2012**, *152*(3), 204 . 44

[114] Castro, A.; Marques, M. A. L.; Rubio, A. *J. Chem. Phys.*, **2004**, *121*, 3425. 44

[115] Helgaker, T.; Jorgensen, P.; Olsen, J. *Molecular Electronic-structure Theory*. John Wiley & Sons Inc, 2012. 44

[116] Becke, A. D. *J. Chem. Phys.*, **1993**, *98*(2), 1372. 44

[117] Kümmel, S.; Perdew, J. P. *Phys. Rev. Lett.*, **2003**, *90*, 043004. 44

[118] Andrade, X.; Aspuru-Guzik, A. *Phys. Rev. Lett.*, **2011**, *107*, 183002. 44, 64

[119] Marques, M. A. L.; Oliveira, M. J. T.; Burnus, T. *Computer Physics Communications*, **2012**, *183*(10), 2272. 44

[120] Vila, F. D.; Strubbe, D. A.; Takimoto, Y.; Andrade, X.; Rubio, A.; Louie, S. G.; Rehr, J. J. *J. Chem. Phys.*, **2010**, *133*(3), 034111. 45

[121] Rappoport, D. *ChemPhysChem*, **2011**, *12*(17), 3404. 45

[122] Troullier, N.; Martins, J. L. *Phys. Rev. B*, **1991**, *43*(3), 1993. 45

[123] Soler, J. M.; Artacho, E.; Gale, J. D.; García, A.; Junquera, J.; Ordejón, P.; Sánchez-Portal, D. *J. Phys.: Condens. Matter*, **2002**, *14*(11), 2745. 45

[124] Hartwigsen, C.; Goedecker, S.; Hutter, J. *Phys. Rev. B*, **1998**, *58*, 3641. 45

[125] Fuchs, M.; Scheffler, M. *Comput. Phys. Commun.*, **1999**, *119*, 67. 45, 97

[126] Gonze, X.; Amadon, B.; Anglade, P.-M.; Beuken, J.-M.; Bottin, F.; Boulanger, P.; Bruneval, F.; Caliste, D.; Caracas, R.; Côté, M.; Deutsch, T.; Genovese, L.; Ghosez, P.; Giantomassi, M.; Goedecker, S.; Hamann, D.; Hermet, P.; Jollet, F.; Jomard, G.; Leroux, S.; Mancini, M.; Mazevet, S.; Oliveira, M.; Onida, G.; Pouillon, Y.; Rangel, T.; Rignanese, G.-M.; Sangalli, D.; Shaltaf, R.; Torrent, M.; Verstraete, M.; Zerah, G.; Zwanziger, J. *Comput. Phys. Commun.*, **2009**, *180*(12), 2582. 45

[127] Giannozzi, P.; Baroni, S.; Bonini, N.; Calandra, M.; Car, R.; Cavazzoni, C.; Ceresoli, D.; Chiarotti, G. L.; Cococcioni, M.; Dabo, I.; Corso, A. D.; de Gironcoli, S.; Fabris, S.; Fratesi, G.; Gebauer, R.; Gerstmann, U.; Gougoussis, C.; Kokalj, A.; Lazzeri, M.; Martin-Samos, L.; Marzari, N.; Mauri, F.; Mazzarello, R.; Paolini, S.; Pasquarello, A.; Paulatto, L.; Sbraccia, C.; Scandolo, S.; Sclauzero, G.; Seitsonen, A. P.; Smogunov, A.; Umari, P.; Wentzcovitch, R. M. *J. Phys.: Condens. Matter*, **2009**, *21*(39), 395502 (19pp). 45

[128] Castro, A.; Marques, M. A. L.; Romero, A. H.; Oliveira, M. J. T.; Rubio, A. *J. Chem. Phys.*, **2008**, *129*(14), 144110. 45

[129] Oliveira, M. J. T.; Nogueira, F.; Marques, M. A. L.; Rubio, A. *J. Chem. Phys.*, **2009**, *131*(21), 214302. 45

[130] Nethercote, N.; Walsh, R.; Fitzhardinge, J. Building workload characterization tools with valgrind. in *Workload Characterization, 2006 IEEE International Symposium on*, pp 2–2. IEEE, 2006. 49

[131] Perdew, J. P.; Zunger, A. *Phys. Rev. B*, **1981**, *23*(10), 5048. 64

[132] Umezawa, N. *Phys. Rev. A*, **2006**, *74*, 032505. 64

[133] Hartree, D. R. *The calculation of atomic structures.* J. Wiley, New York, 1957. 64

[134] Fock, V. A. *The Theory of space, time and gravitation.* Pergamon Press, 1964. 64

[135] Casida, M. E.; Jamorski, C.; Bohr, F.; Guan, J.; Salahub, D. R. in *Nonlinear Optical Materials.* chapter 9, pp 145–163. 1996. 64

[136] R., K.; E., A.; J., N. *Chemical Physics Letters*, **1995**, *238*(1), 173. 64

[137] Beck, T. L. *Int. J. Quant. Chem.*, **1997**, *65*(5), 477. 64

[138] Rudberg, E.; Salek, P. *J. Chem. Phys.*, **2006**, *125*(8), 084106. 64

[139] Genovese, L.; Deutsch, T.; Goedecker, S. *J. Chem. Phys.*, **2007**, *127*(5), 054704. 64

[140] Cerioni, A.; Genovese, L.; Mirone, A.; Sole, V. A. *J. Chem. Phys.*, **2012**, *137*(13), 134108. 64

[141] Gygi, F.; Draeger, E. W.; Schulz, M.; de Supinski, B. R.; Gunnels, J. A.; Austel, V.; Sexton, J. C.; Franchetti, F.; Kral, S.; Ueberhuber, C. W.; Lorenz, J. Large-scale electronic structure calculations of high-Z metals on the BlueGene/L platform. in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM. 64

[142] Alonso, J. L.; Andrade, X.; Echenique, P.; Falceto, F.; Prada-Gracia, D.; Rubio, A. *Phys. Rev. Lett.*, **2008**, *101*, 096403. 64

[143] Andrade, X.; Castro, A.; Zueco, D.; Alonso, J. L.; Echenique, P.; Falceto, F.; Rubio, A. *J. Chem. Theory Comput.*, **2009**, *5*, 728. 64

[144] Fang, B.; Deng, Y.; Martyna, G. *Computer Physics Communications*, **2007**, *176*(8), 531 . 64

[145] Ding, H. Q.; Gennery, D. B.; Ferraro, R. D. A Portable 3D FFT Package for Distributed-Memory Parallel Architectures. in *Proceedings of 7th SIAM Conference on Parallel Processing*, pp 70–71. SIAM Press, 1995. 65

[146] Eleftheriou, M.; Moreira, J. E.; Fitch, B. G.; Germain, R. S. A Volumetric FFT for BlueGene/L. in *HiPC*, Pinkston, T. M.; Prasanna, V. K., Eds., volume 2913 of *Lecture Notes in Computer Science*, pp 194–203. Springer, 2003. 65

[147] Eleftheriou, M.; Fitch, B. G.; Rayshubskiy, A.; Ward, T. J. C.; Germain, R. S. *IBM Journal of Research and Development*, **2005**, *49*, 457. 65

[148] Eleftheriou, M.; Fitch, B. G.; Rayshubskiy, A.; Ward, T. J. C.; Germain, R. S. Performance Measurements of the 3D FFT on the Blue Gene/L Supercomputer. in *Euro-Par 2005 Parallel Processing*, Cunha, J. C.; Medeiros, P. D., Eds., volume 3648 of *Lecture Notes in Computer Science*, pp 795–803. Springer, 2005. 65

[149] Plimpton, S. J.; Pollock, R.; Stevens, M. Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations. in *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, 1997)*, Philadelphia, 1997. SIAM. 65

[150] Plimpton, S. Parallel FFT subroutine library, 2011. http://www.sandia.gov/ sjplimp/docs/fft/README.html. 65

[151] Pekurovsky, D. *SIAM J. Sci. Comput.*, **2012**, *34*, C192 . 65

[152] Pekurovsky, D. P3DFFT, Parallel FFT subroutine library, 2011. http://www.sdsc.edu/us/resources/p3dfft. 65

[153] Li, N.; Laizet, S. 2DECOMP & FFT - A Highly Scalable 2D Decomposition Library and FFT Interface. in *Cray User Group 2010 conference*, pp 1 – 13, Edinburgh, Scotland, 2010. 65

[154] Li, N. 2DECOMP&FFT, Parallel FFT subroutine library, 2011. http://www.hector.ac.uk/cse/distributedcse/reports/incompact3d/incompact3d/index.html. 65

[155] Truong Duy, T. V.; Ozaki, T. *ArXiv e-prints*, **2012**. 65

[156] Frigo, M.; Johnson, S. G. *Proceedings of the* IEEE, **2005**, *93*, 216. 65

[157] Frigo, M.; Johnson, S. G. FFTW, C subroutine library. http://www.fftw.org, 2009. 65

[158] Wang, J.; Luo, R. *Journal of Computational Chemistry*, **2010**, *31*(8), 1689. 66

[159] Saad, Y.; Chelikowsky, J.; Shontz, S. *SIAM Review*, **2010**, *52*(1), 3. 66

[160] Mardal, K.-A.; Winther, R. *Numerical Linear Algebra with Applications*, **2011**, *18*(1), 1. 66

[161] Weijo, V.; Randrianarivony, M.; Harbrecht, H.; Frediani, L. *Journal of Computational Chemistry*, **2010**, *31*(7), 1469. 66

[162] Balay, S.; Brown, J.; Buschelman, K.; Eijkhout, V.; Gropp, W.; Kaushik, D.; Knepley, M.; Curfman McInnes, L.; Smith, B.; Zhang, H. *PETSc Users Manual, Revision 3.3.* Mathematics and Computer Science Division, Argonne National Laboratory, 2012. 66

[163] Takacs, S.; Zulehner, W. *Computing and Visualization in Science*, **2011**, *14*(3), 131. 66

[164] Adams, M. A distributed memory unstructured gauss-seidel algorithm for multigrid smoothers. in *Supercomputing, ACM/IEEE 2001 Conference*, pp 14–14, 2001. 66

[165] Karypis, G.; Kumar, V. *SIAM Journal on Scientific Computing*, **1998**, *20*, 359. 67

[166] Vosko, S. H.; Wilk, L.; Nusair, M. *Canadian Journal of Physics*, **1980**, *58*(8), 1200. 69

[167] Wanko, M.; García-Risueño, P.; Rubio, A. *physica status solidi (b)*, **2012**, *249*(2), 392. 70

[168] Zhan, C.-G.; Nichols, J. A.; Dixon, D. A. *The Journal of Physical Chemistry A*, **2003**, *107*(20), 4184. 70

[169] Janak, J. F. *Phys. Rev. B*, **1978**, *18*, 7165. 70

[170] Karypis, G.; Kumar, V. METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0. Technical report, 1995. 84

[171] Alberdi-Rodriguez, J.; Rubio, A.; Oliveira, M.; Charalampidou, A.; Folias, D. *PRACE white paper*, **2015**. 87

[172] Scholes, G. D.; Mirkovic, T.; Turner, D. B.; Fassioli, F.; Buchleitner, A. *Energy Environ. Sci.*, **2012**, *5*, 9374. 94

[173] Frank, H. A.; Cogdell, R. J. *Photochemistry and Photobiology*, **1996**, *63*(3), 257. 94

[174] Young, A.; Britton, G. *Carotenoids in photosynthesis.* Chapman and Hall Ltd, 1993. 94

[175] Polívka, T.; Sundström, V. *Chemical Reviews*, **2004**, *104*(4), 2021. 94

[176] Lee, H.; Cheng, Y.-C.; Fleming, G. R. *Science*, **2007**, *316*, 1462. 94

[177] Calhoun, T. R.; Ginsberg, N. S.; Schlau-Cohen, G. S.; Cheng, Y.-C.; Ballottari, M.; Bassi, R.; Fleming, G. R. *J. Phys. Chem. B*, **2009**, *113*, 16291. 94

[178] Collini, E.; Wong, C. Y.; Wilk, K. E.; Curmi, P. M. G.; Brumer, P.; Scholes, G. D. *Nature*, **2010**, *463*, 644. 94

[179] Broess, K.; Trinkunas, G.; van der Weij-de Wit, C. D.; Dekker, J. P.; van Hoek, A.; van Amerongen, H. *Biophys. J.*, **2006**, *91*, 3776. 94

[180] Ishizaki, A.; Calhoun, T. R.; Schlau-Cohen, G. S.; Fleming, G. R. *Phys. Chem. Chem. Phys.*, **2010**, *12*, 7319. 94

[181] van Grondelle, R.; Novoderezhkin, V. I. *Phys. Chem. Chem. Phys.*, **2006**, *8*, 793. 94

[182] Liu, Z.; Yan, H.; Wang, K.; Kuang, T.; Zhang, J.; Gui, L.; An, X.; Chang, W. *Nature*, **2004**, *428*, 287. 94

[183] Stewart, J. J. P. *J. Molec. Model.*, **2013**, *19*, 1. 95

[184] Stewart Computational Chemistry, Colorado Springs, CO, USA, HTTP://OpenMOPAC.net. *MOPAC2012, James J. P. Stewart, Stewart Computational Chemistry, Colorado Springs, CO, USA, HTTP://OpenMOPAC.net*, 2012. 95

[185] Pettersen, E. F.; Goddard, T. D.; Huang, C. C.; Couch, G. S.; Greenblatt, D. M.; Meng, E. C.; Ferrin, T. E. *J. Comput. Chem.*, **2004**, *25*, 1605. 95

[186] Perdew, J. P.; Burke, K.; Ernzerhof, M. *Phys. Rev. Lett.*, **1997**, *78*, 1396. 95, 98

[187] Perdew, J. P.; Burke, K.; Ernzerhof, M. *Phys. Rev. Lett.*, **1996**, *77*, 3865. 95, 98

[188] Jiang, H.; Baranger, H. U.; Yang, W. *Phys. Rev. B*, **2003**, *68*, 165337. 97

[189] Saad, Y.; Stathopoulos, A.; Chelikowsky, J.; Wu, K.; Öğüt, S. *BIT Numerical Mathematics*, **1996**, *36*, 563. 97

[190] Broyden, C. G. *Math. Comp.*, **1965**, *19*, 577. 97

[191] Johnson, D. D. *Phys. Rev. B*, **1988**, *38*, 12807. 97

[192] Milne, B. F.; Rubio, A.; Nielsen, S. B. *Angew. Chem. Int. Ed. Eng.*, **2015**, *54*, 2170. 99, 100

[193] Stockett, M. H.; Musbat, L.; Kjær, C.; Houmøller, J.; Toker, Y.; Rubio, A.; Milne, B. F.; Brønsted Nielsen, S. *Phys. Chem. Chem. Phys.*, **2015**, *DOI: 10.1039/C5CP01513H.* 99

[194] Dreuw, A.; Head-Gordon, M. *J. Am. Chem. Soc.*, **2004**, *126*, 4007. 99

[195] Dreuw, A. *J. Phys. Chem. A*, **2006**, *110*, 4592. 99

[196] Linnanto, J.; Martiskainen, J.; Lehtovuori, V.; Ihalainen, J.; Kananavicius, R.; Barbato, R.; Korppi-Tommola, J. *Photosynthesis Research*, **2006**, *87*(3), 267. 103

[197] Förster, T. *Annalen der Physik*, **1948**, *437*(1-2), 55. 103

[198] Lichtenthaler, H. K.; Buschmann, C. *Current protocols in food analytical chemistry*, **2001**. 127

[199] Kabadshow, I.; Dachsel, H.; Hammond, J. Poster: Passing the three trillion particle limit with an error-controlled fast multipole method. in *Proceedings of the 2011 companion on High Performance Computing Networking, Storage and Analysis Companion*, SC '11 Companion, pp 73–74, New York, NY, USA, 2011. ACM. 135

[200] Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Scuseria, G. E.; Robb, M. A.; Cheeseman, J. R.; Scalmani, G.; Barone, V.; Mennucci, B.; Petersson, G. A.; Nakatsuji, H.; Caricato, M.; Li, X.; Hratchian, H. P.; Izmaylov, A. F.; Bloino, J.; Zheng, G.; Sonnenberg, J. L.; Hada, M.; Ehara, M.; Toyota, K.; Fukuda, R.; Hasegawa, J.; Ishida, M.; Nakajima, T.; Honda, Y.; Kitao, O.; Nakai, H.; Vreven, T.; Montgomery, J. A. Jr.; Peralta, J. E.; Ogliaro, F.; Bearpark, M.; Heyd, J. J.; Brothers, E.; Kudin, K. N.; Staroverov, V. N.; Kobayashi, R.; Normand, J.; Raghavachari, K.; Rendell, A.; Burant, J. C.; Iyengar, S. S.; Tomasi, J.; Cossi, M.; Rega, N.; Millam, J. M.; Klene, M.; Knox, J. E.; Cross, J. B.; Bakken, V.; Adamo, C.; Jaramillo, J.; Gomperts, R.; Stratmann, R. E.; Yazyev, O.; Austin, A. J.; Cammi, R.; Pomelli, C.; Ochterski, J. W.; Martin, R. L.; Morokuma, K.; Zakrzewski, V. G.; Voth, G. A.; Salvador, P.; Dannenberg, J. J.; Dapprich, S.; Daniels, A. D.; Farkas, O.; Foresman, J. B.; Ortiz, J. V.; Cioslowski, J.; Fox, D. J. Gaussian 09 Revision A.1, 2009. Gaussian Inc. Wallingford CT 2009. 139

Acknowledgements

Quiero agradecer profundamente a Angel Rubio y a Xavier Andrade por haberme dado una oportunidad única de involucrarme e introducirme de lleno en el ámbito científico. Gracias por proponerme un problema muy interesante, introducirme en un entorno internacional y darme la posibilidad de conocer a tanta gente inteligente.

Agustin Arruabarrenari eta Javier Muguerzari nire eskerrik beroena; beti laguntzeko prest egon zaretelako eta zuen onena jaso dudalako. Izugarrizko ezagutza eta interesa jarri duzuelako, ahaleginik handiena egin duzuelako proiektu hau aurrera atera dadin. Badakizue lan honen zati handi bat zuena dela.

Quiero mencionar especialmente a Pablo García Risueño por su extremada paciencia e insistencia. Por sus enormes ganas de trabajar y hacerme trabajar.

Joaquim Jornet Somoza ha sido otra de las personas de vital importancia para que todo nuestro trabajo haya salido adelante. Es con el que más estrechamente he colaborado y una de las personas más inteligentes que he conocido.

Many thanks to the team that work in the Light Harvesting Complex II project: Joaquim, Bruce, Xavier, Miguel, Fernando, Micael and Angel. As well I want to thank the Octopus developers, specially: David Strubbe, Xavier, Micael and Alberto.

A la mia amica Elena, per avermi insegnato italiano e per molto di più. Obrigado Gustavo, por ser um verdadeiro amigo.

Obrigado Micael e Fernando: por dar a oportunidade de conhocer novos lugares. Por ficar perto e ajudar me sempre que eu preciso.

Anneri, bidelagunari, bide berriak irekitzeagatik. Emandako babes guztiarengatik eta eskaintzen didazun ilusio guztiarengatik.

Familiari; gurasoei (Lourdes eta Iñaki), naizen guztia izateko aukera eman didazuelako. Anaiari (Aitor), hainbeste erakusteagatik eta beti hor ondoan egoteagatik. Ilobei (Jon eta Miren) beti irribarre bat eskaintzeagatik, eta beraien amari (Idoia) ere bai. Amona, aitona zenari, amama, aitite eta gainontzeko izeba-osaba eta lehengusu-lehengusinei ere nire eskerrik sakonena. Familia handi honetako parte izateaz oso harro nago.

Benetako lagunei: Maialen, Nerea eta Endika. Oierri, kirolaz ere gozatzeko aukera emateagatik, eta ez horregatik bakarrik. Horrenbeste erakutsi didan Estibalitzi ere eskerrak.

Kuadrilari ...edo: Endika eta Nerea, Imanol eta Begoña, Dannel eta Maddalen, Oier eta Ane, Egoitz eta Aintzane, Egoitz, Asier eta Karmele, Alain eta Yoana, Aritz, Jon (Antzizar, Sanz eta Lizarraga), Iñaki, Joseba, Xabi, Markel, Juanjo, Xiker, Mikel, Julen, Dame eta Gorka.

I entirely acknowledge the Nano-Bio Spectroscopy group: Angel, Cecilia, Cristina, Larri, Xavier, Víctor, Bruno, Fulvio, Kaike, Peizhe, Feffe, Camila, Mehdi and

Nasim, Claudio, Seymur, Lorenzo, Yann, Marius, Amilcare, Ask, Nicole, Jessica, Johanna, Ali, Leonardo, Umberto, Ilya, Stefan, Matteo, Pierlugi, Martin, Duncan, Alejandro, Roberto, Robert, Martha, Matthieu, David, Irina, Paul, Philipp, Ravindra, Livia, Hannes, Lede, Sener, Andre and also those that are not present in this list. It was amazing to have all those discussions with you and give let me know your part and vision of the world. I specially enjoyed having lunch with you. Alejandro, gracias por ayudarme con el cluster Ganbo y los ordenadores.

Eskerrak, halaber, ALDAPA taldekoei, gutxitan gerturatu arren, hurbil egoteagatik: Javier, Agustin, Ibai, Natxo, Olatz, Txus, Iñigo, Aizea, Igor, Ainhoa, José Luis, Javier, Otzeta, Aritz eta Iñaki. Zientzia ere euskaraz egin daiteke!

Eskerrik asko! Gracias! Thank you! Obrigado! Grazie! Merci! Danke!

9.3.1 Financing

I was granted by a fellowship of the University of the Basque Country, UPV/EHU, for 4 years, from the March of 2010 to February 2014. By a grant of the University of Coimbra from May 2014 to August 2014. A PIC from September 2014 until the end. For the computing time, I have to acknowledge the Jülich Supercomputing Centre (JSC), Rechenzentrum Garching, Cineca, Barcelona Supercomputing Center (BSC) and CeSViMa. This project is part of PRACE and RES initiatives. I acknowledge financial support from the European Research Council Advanced Grant DYNamo (ERC-2010-AdG-267374), Spanish Grant (FIS2013-46159-C3-1-P), Grupos Consolidados UPV/EHU del Gobierno Vasco (IT578-13 and IT395-10) and European Community FP7 project CRONOS (Grant number 280879-2) and COST Actions CM1204 (XLIC) and MP1306 (EUSpec). ALDAPA research group belongs to the Basque Advanced Informatics Laboratory (BAILab) supported by the University of the Basque Country UPV/EHU (grant UFI11/45).

Appendix A

LHC appendix

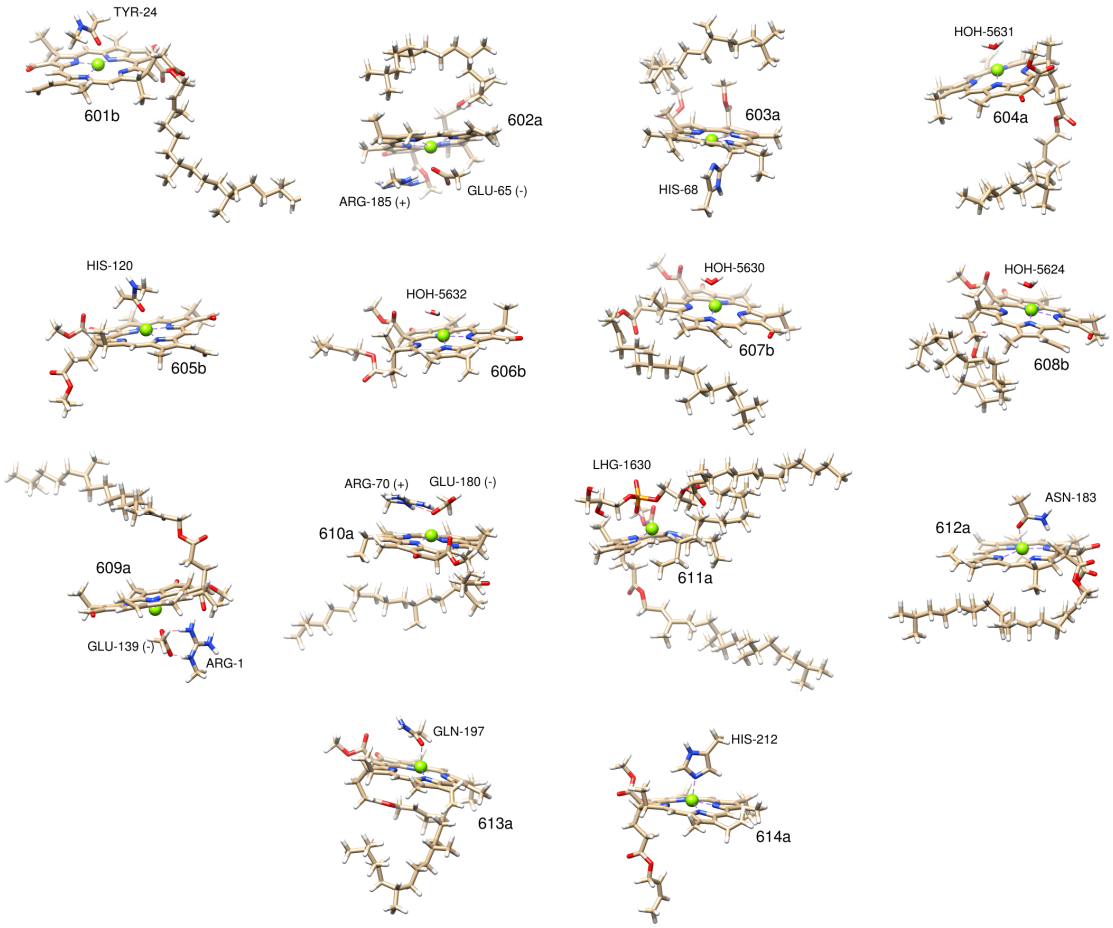


Figure A.1: Spacial geometry of each of the 14 molecules that form the chlorophyll network of the one monomer of the LHC–II. Each chlorophyll presents a different conformation due to its specific axial coordination residue. As mentioned in the main text, the all atoms structure of the LHC–II (17000 atoms) has been optimized using the PM7 semi-empirical Hamiltonian implemented in the Mopac semi-empirical electronic structure package. These individual structures have been used to compute the absorption spectra of each isolated chlorophyll.

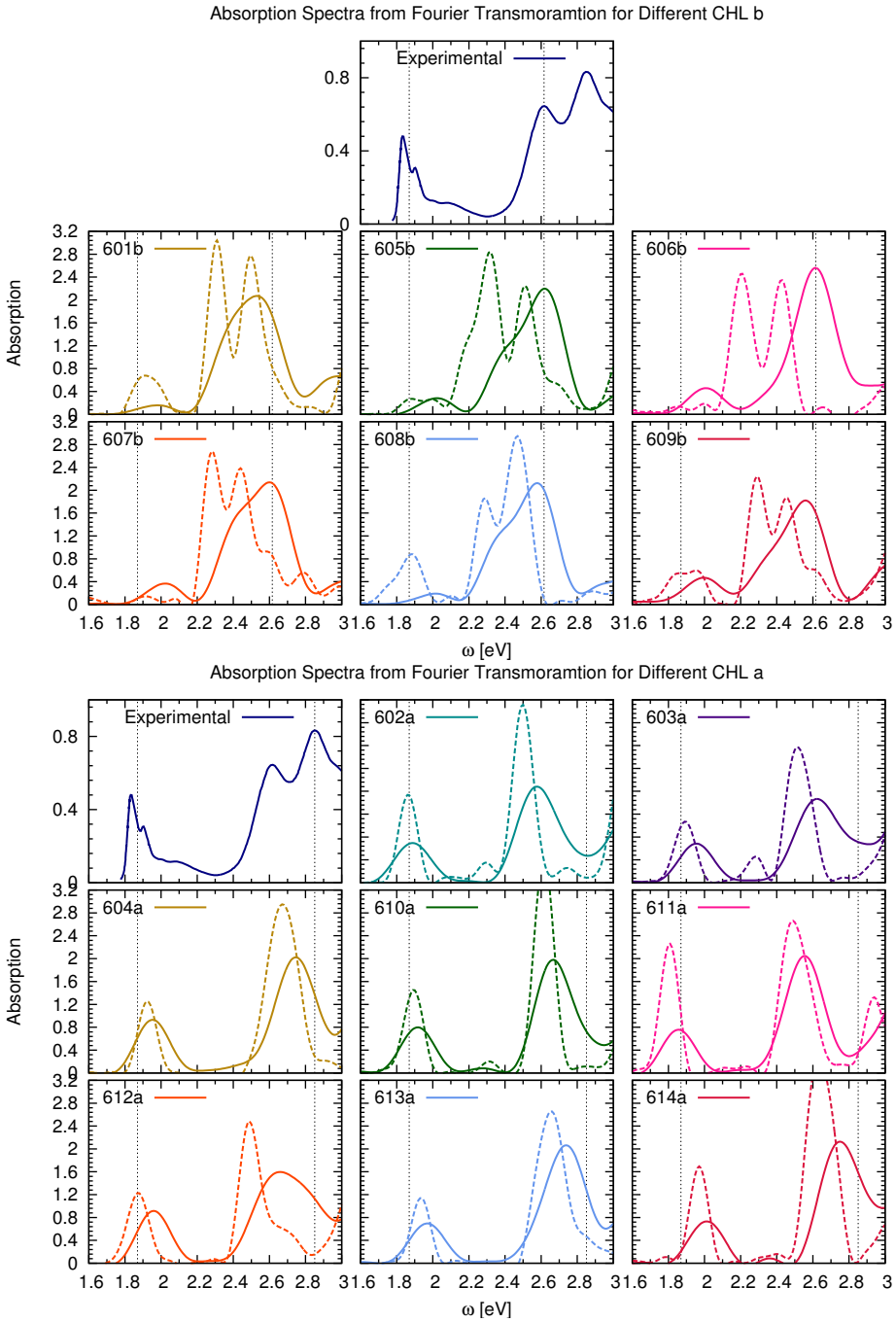


Figure A.2: Calculated absorption spectra for isolated LHC–II chromophores, from up to low chlorophyll a *and chlorophyll* b*. Spectra obtained for the isolated chlorophylls (solid lines) are compared with that obtained from local dipole analysis (dotted lines). Vertical dashed lines indicate positions of experimental absorption energies. Direct observation on the simulated spectra shows a strong effect on the electronic transition of the chlorophyl* b *due to the environment effect while a small oscillator strenght transfer effect is observed for cholorphylls* a

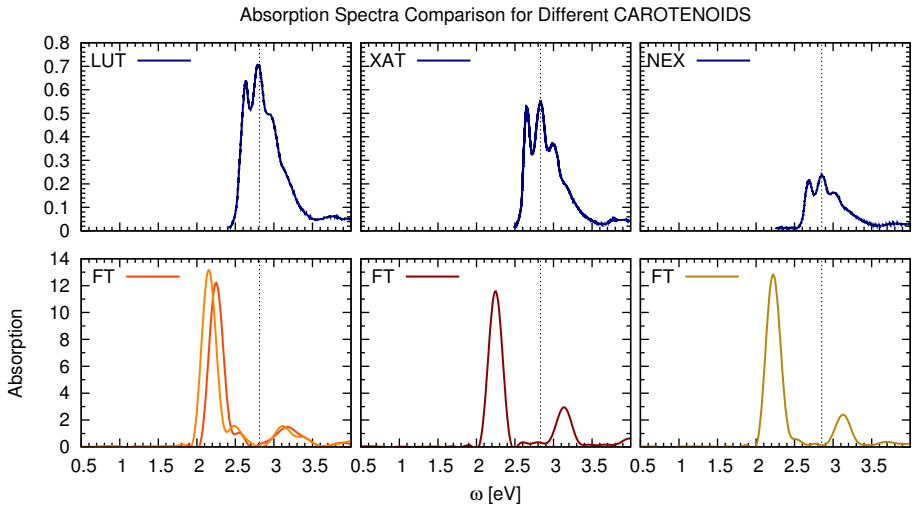


Figure A.3: Comparison between experimental [198] (up) and simulated (down) absorption spectra of the 3 types of carotenoids molecules: lutein (LUT), violoxanthin (XAT) and neoxanthin (NEX). A surprisingly good agreement between experimental and simulated (approx. $\Delta = 0.5$ eV) is attributed to the non-planar conformation of the polyene chains due to the structure optimization on the LHC–II complex.

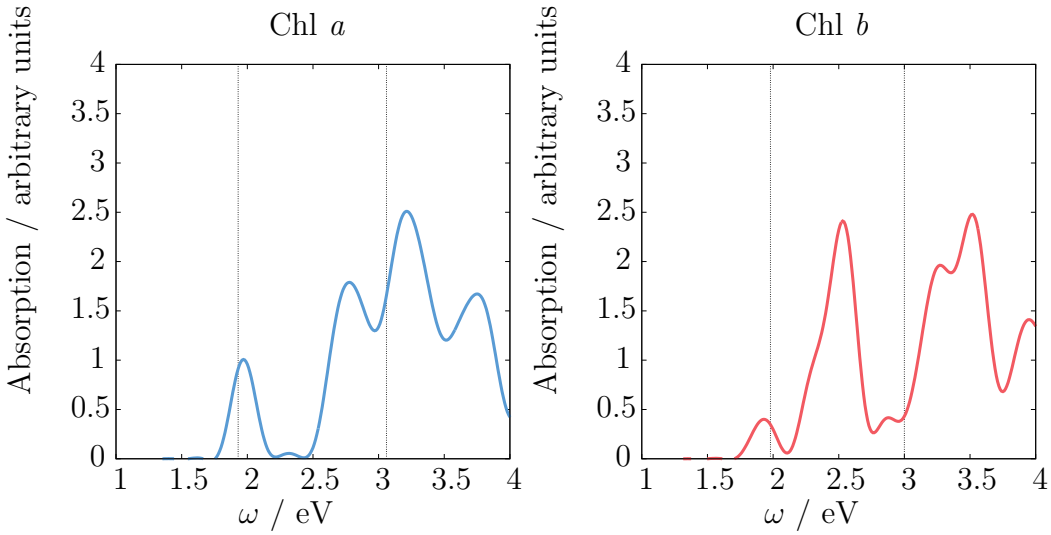


Figure A.4: Gas phase experiment compared with our TDDFT calculation.

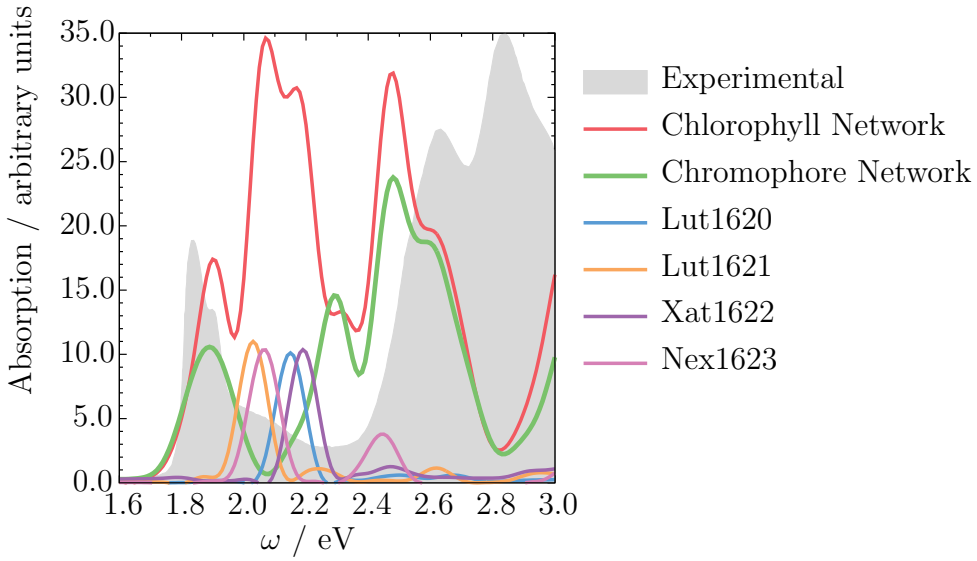


Figure A.5: Study of the effect of the inclusion of carotenoid molecules to the simulated absorption spectrum. The observed strong band around 2.1 eV on the spectrum of the chromophore network (upper red line) is assigned to the carotenoids effect by direct comparison with the spectra obtained for the chlorophyll network (blue line). The spectra decomposition for the carotenoids contributions confirm that the maxima absorption peaks do not strongly interact with the chlorophyll bands. Grey shade shows the experimental spectrum of the LHC–II [22].

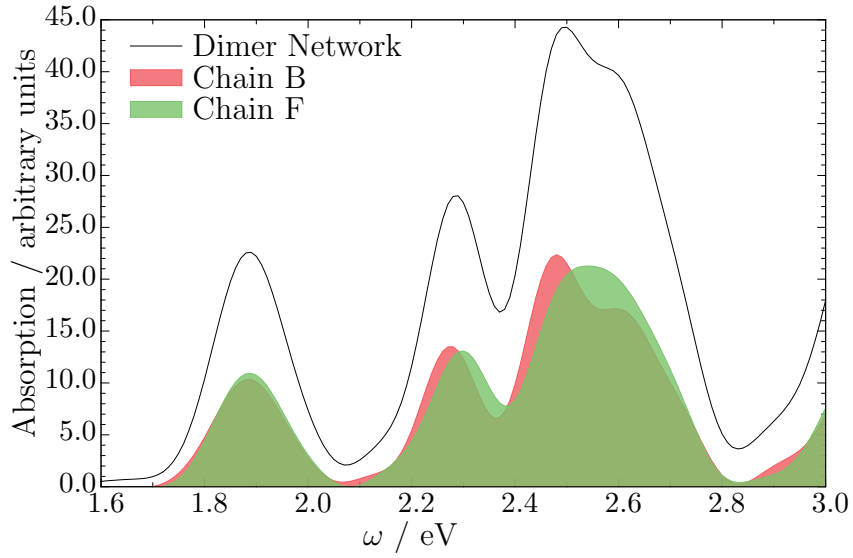


Figure A.6: Absorption spectra decomposition of the chlorophylls network of the LHC–II dimer on its former monomers: chain B (red shade) and F (green shade) according to the PDB file 1RWT nomenclature. The total absorption spectra for the chlorophyll network of the dimer is also represented (solid black line). The small difference on the shape of the Soret–band is due to the small structural differences between monomers, since no symmetry has been applied during the structural optimization.

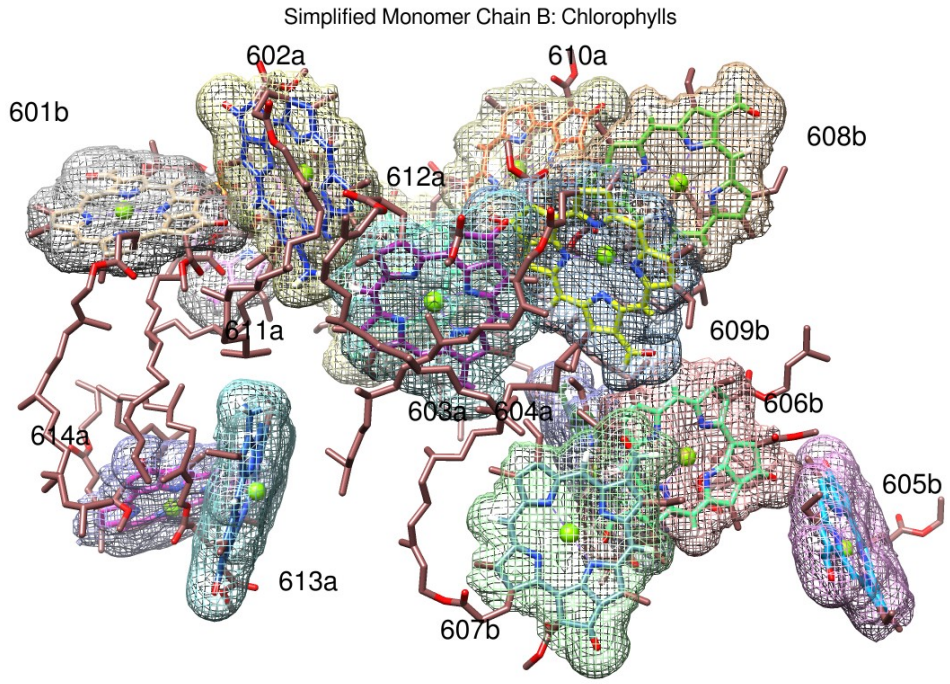


Figure A.7: Chlorophyll network of the LHC–II monomer (chain B) used on this work. The systems consists in 14 chlorophyll molecules for a total of 2025 atoms. Hydrogen atoms are not shown for a better visualization. Bader volumes for each chlorophyll are represented as mesh. The local absorption decomposition has been performed over these targeted densities

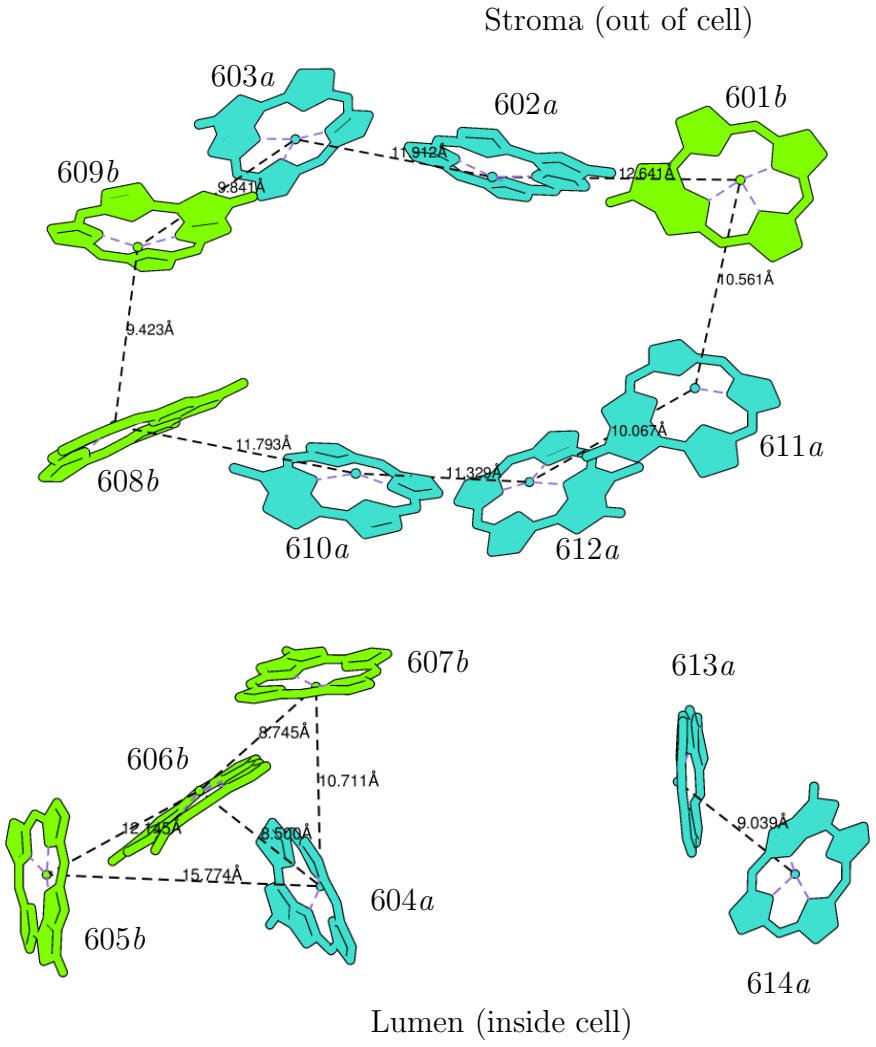


Figure A.8: Relative distribution of the different chlorophyll molecules of the stroma (up) and lumen (half-side). Mg–Mg distances between chlorophyll molecules are represented as a reference of the proximity between neighbor molecules. Chlorophyll molecules are displayed only as chlorin rings to improve clarity. Chlorophyll a *are represented in blue and chlorophyll* b *in green.*

Appendix B

HPC projects

B.I PRACE (Partnership for Advanced Computing in Europe) calls

- Project 2013081486
  MareNostrum III at BSC
  From 1 of September of 2013 to 31 of August of 2014
  PRACE Tier-0: 21,500,000 core-hours
  "LHC-ABS - The optical absorption spectra of a real Light Harvesting Complex from first-principles: the spinach case"

- Project 2013081562
  FERMI at CINECA (Blue Gene/Q)
  From 1 of September of 2013 to 31 of August of 2014
  PRACE Tier-0: 20,000,000 core-hours
  "LAIT - Light-harvesting in the time domain"

- Project 2010PA1404
  Juqueen (Blue Gene/Q)
  From 1 of May to 31 of July of 2013
  Type C (support from one PRACE expert)
  "Optimization of the code Octopus"

- Project 2010PA0660
  Curie Fat Nodes
  From 1 of January to 31 of March of 2012
  Type A (code scalability testing): 50,000 core-hours
  "Scalability of methods to calculate the electrostatic potentials created by charge distributions"

- Project 2010PA0763
  Jugene at the beginning, Juqueen (BG/Q) then
  From 25 of April to 30 of November 2012
  Type B (code development and optimization): 250,000 core-hours
  "Optimization of new algorithms in Octopus for methods to calculate the electrostatic potentials created by charge distributions"

- Project 2010PA0415
  Jugene (Blue Gene/P)
  From 2 of May to 31 of October 2011
  Type C (support from one PRACE expert): 250,000 core-hours
  "New algorithms in Octopus for the Pflops computing"

B.II RES (Red Española de Supercomputación) calls

- QCM-2015-1-0016
  LaPalma (PowerPC PPC970MP)
  From 1 of March to 30 of June 2015
  700,00 core-hours
  "First principle simulations of gas storage on nano-structured materials"

- QCM-2014-2-0036
  MareNostrum III
  From 1of July to 31 of October 2014
  84,000 core-hours
  "Improving the Performance of the IO and Density Fragment procedures on OCTOPUS code"

- QCM-2014-1-0041
  Magerit
  From 1 of March to 30 of June 2014
  100,000 core-hours
  "New local multipole implementation and visualization for Octopus code"

- FI-2013-3-0021
  Magerit
  From 1 of November 2013 to 30 of June 2014
  100,000 core-hours
  "Implementation of new exchange correlation functionals and molecular dynamics calculations in Octopus"

- FI-2013-1-0010
  MareNostrum III and Magerit
  From 1 of March to 30 of June 2013
  160,0000 core-hours (500,000 used)
  "Octopus GS and TD scaling testing in new architectures"

Appendix C

Executions

C.I Execution times

During the initial stage of the project, we had done lots of tests to measure different parameters. Those tests involved huge execution times. Also, we have found some problems that increased significantly the required total amount of computing hours. As the supercomputers are limited infrastructures, every user has a limited quota of usage, determined by the project that he or she has supplied. At the beginning, we were using a test account in Jugene supercomputer, and next table summarises the core hours we had used:

	GS test			**Real GS**			**TD test**		
No. of atoms	No. exec.	Hours	Core hour	No. exec.	Time	Hours	No. exec.	Hours	Core hour
180	13	0.87	1,122.42	1	0.15	38.47	26	2.53	24,768.82
441	10	1.52	1,831.08	-	-	-	-	-	-
650	6	1.87	2,753.24	-	-	-	-	-	-
1365	2	2.79	8,696.60	11	19.84	376,177.78	17	10.21	54,914.84
2676	-	-	-	2	11.43	374,538.24	-	-	-
5879	-	-	-	9	11.71	563,163.59	-	-	-
Sum	31	7.03	14,403.34	23	43.13	1,313,918.08	43	12.74	79,683.66
Total sum 1,408,005.08 core hours									

Table C.1: Summary of the execution times (GS tests, real GS calculations attempts, and TD tests) in Jugene.

In the Table C.1 we can see that almost all the time has been spent trying to obtain the real GS (executions until the converged ground state of the simulated system). We needed to tune some input parameters, and this required a considerable amount of computational time; finally, we could converge the system of 1365 atoms. But, unfortunately, we could not manage to run simulations of bigger systems, so we could not carry out the corresponding TD tests. In fact, to obtain the ground state solution currently we need to allocate more data than what can fit in the nodes memory, so all the executions for big systems broke. Several efforts have been done to overcome this problem by adjusting the input parameters and optimising memory usage in the code, that implied using large quantities of time for tests, but the problem is still not completely solved.

In total, we have used more than 1,408,000 core hours, or, what is the same, we have used the whole machine for almost 5 hours, or we have used one processor (4 cores) for 40 years.

C.II Number of executions

A great amount of experiment have been carried-out, as we had got many different variables to prove: we used three machines (Jugene, MareNostrum II and Ganbo), six atomic system sizes (180, 441, 650, 1365, 2676 and 5879) and three different calculation modes (GS tests, real GS and TD tests). We have also two parallelisation levels (domain and states) that have to be tuned properly, in addition to other physical input variables. So, at the end, this creates large number of possibilities that we had to try. The next Figure C.1 is an example of some of executions of each type. In Figure can be seen the order of execution times depending on the number of nodes and the atomic system size. Have to be taken in account that each of shown test has to be done more than once.

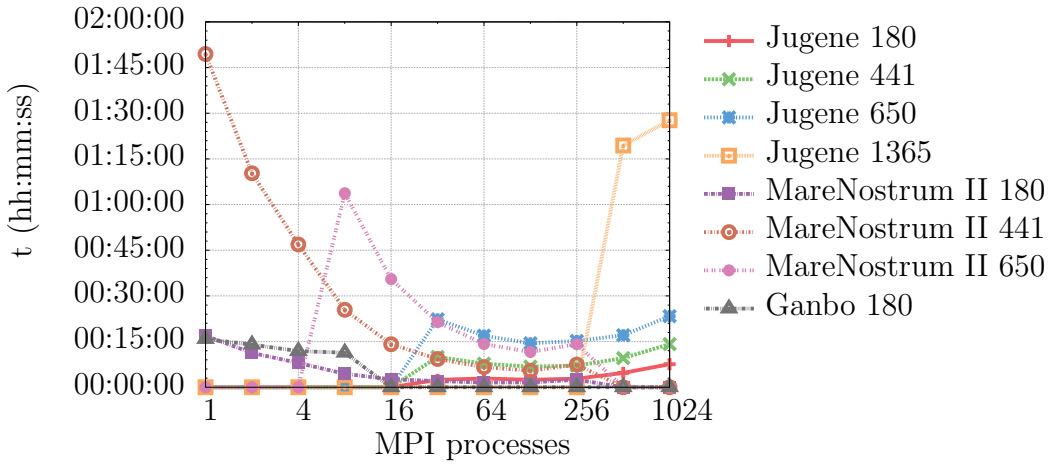


Figure C.1: Test GS overall execution time. Four systems are shown (180, 441, 650 and 1365 atoms) and three machines (Jugene, MareNostrum II and Ganbo).

Appendix D

Supporting info of Poisson solvers

In this appendix we include some information to complement our work of Chapter 6. First, we provide some statements on the communication patterns of PFFT [15], ISF [14], and FMM [81] used by our implementations in Section D.I. Afterwards, we give more details on the tests we have done (Section D.II). We also compare the time of the multipolar expansion correction of the Multigrid and Conjugate Gradients methods with the corresponding total execution time in Section D.III. Finally, in Section D.IV, we discuss the correction term we devised to adapt the FMM method (originally devised to deal with point charges) to charge distributions.

D.I Communication patterns

During the Poisson solver Octopus migh need to use an external library. On those external codes (for example ISF [14] and PFFT [15]) the data distribution migh differ from that on Octopus. Fast Fourier Transforms require parallelepipedic meshes, but this is not necessarily the case of Octopus. In these cases, the original adaptive mesh has to be converted to a parallelepiped mesh by filling the new points with zeroes. At the initialization stage a mapping between the Octopus mesh decomposition and the FFT mesh decomposition is established and saved. This mapping is used when running the actual solver to efficiently communicate only the strictly necessary mesh data between processes, achieving almost perfectly linear parallel scaling.

The PFFT library requires two communication steps in addition to the box transformation. Required communication needs are two `MPI_Alltoall` calls for every calculated FFT. In total, six `MPI_Alltoall` calls are needed in every Poisson solver. However, ISF library is more efficient in this sence and it only needs two `MPI_Alltoall` to calculate the entire Poisson solver; so, in total only four `MPI_Alltoall` are needed.

Regarding to the FMM library, three MPI global communication functions have to be executed: `MPI_Allgather`, `MPI_Allreduce` and `MPI_Alltoall`. Additionally, synchronisation between different FMM levels has to be done using `MPI_Barrier` [199].

D.II Tuning the system parameters

In this Section we give additional information on several input parameters that we used in our tests of the Poisson solvers, so that they can be reproduced. These

parameters are the spatial form of the mesh in whose points the Hartree potential was calculated (`BoxShape`), the accuracy tolerance of the energy calculated with FMM (`DeltaEFMM`), the number of stages in the mesh hierarchy of the Multigrid solver (`MultigridLevels`) and the order of the multipole expansion of the charges whose potential is analytically calculated when using the Multigrid or the Conjugate Gradients solvers (`PoissonSolverMaxMultipole`).

Octopus is able to handle different mesh shapes, like spherical, parallelepiped or adaptive shapes. Apart from the mesh shape and size, the mesh is defined by its spacing parameter, i.e., the distance between consecutive mesh points. The *adaptive* mesh shape option in Octopus produces fair results with FMM. Serial FFT, PFFT, and ISF (which uses FFT) solvers create a parallelepiped mesh — which contains the original adaptive one— to calculate the Hartree potential. If Multigrid or Conjugate Gradients are used with a adaptive mesh, it is frequent to find a serious loss of accuracy, because adaptive boxes are usually irregular, and the multipole expansion (whose terms are based on spherical harmonics, which are rather smooth) cannot adapt well to arbitrary charge values in irregular meshes. These effects of box shape made us choose cubic meshes for our tests. In any case, one should take into account that if non-parallelepiped meshes are used, then the solvers based on reciprocal space (serial FFT, PFFT and ISF) spend additional time dealing with the additional points with null density, while FMM does not have this problem. Examples of this can be viewed in Table D.1, where it can be observed that the execution time is essentially the same regardless of the box shape for PFFT, while spherical and adaptive box shapes are much more efficient than parallelepiped shape for FMM (with a ratio of about 1.67).

Spherical meshes are still valid for Multigrid and Conjugate Gradients methods. We have compared the relative accuracy of cubic and spherical meshes for FMM, Multigrid, and CG methods containing the same number of points. When FMM is used, both errors decrease while the mesh size is increased, being relatively equal for both representations, until an accuracy plateau is reached for big meshes. The same behaviour is shown for the Multigrid solver and by the CG solver up to a given volume for a given system.

	PFFT			FMM		
$R\|\|L_e$(Å)	Adaptive	Sphere	Parallel.	Adaptive	Sphere	Parallel.
15.8	1.001	1.032	1.030	3.371	3.5413	5.945
22.0	2.535	2.664	2.667	10.500	10.447	16.922
25.8	4.321	4.265	4.393	16.321	16.212	27.861
31.6	8.239	8.034	8.271	27.821	28.707	48.206

Table D.1: Comparison of total times (s) required for the calculation of the Hartree potential as a function of the box shape and radius (R) or semi-edge length (L_e) for PFFT and FMM solvers.

Apart from their shape (whose effects are commented above) and their size (whose effects are commented in Chapter 6), the meshes are determined by the spacing between points. In Sections 6.2.1 and 6.2.2 we showed the accuracy and the execution time of the Poisson solvers as a function of the size (semi-edge) of

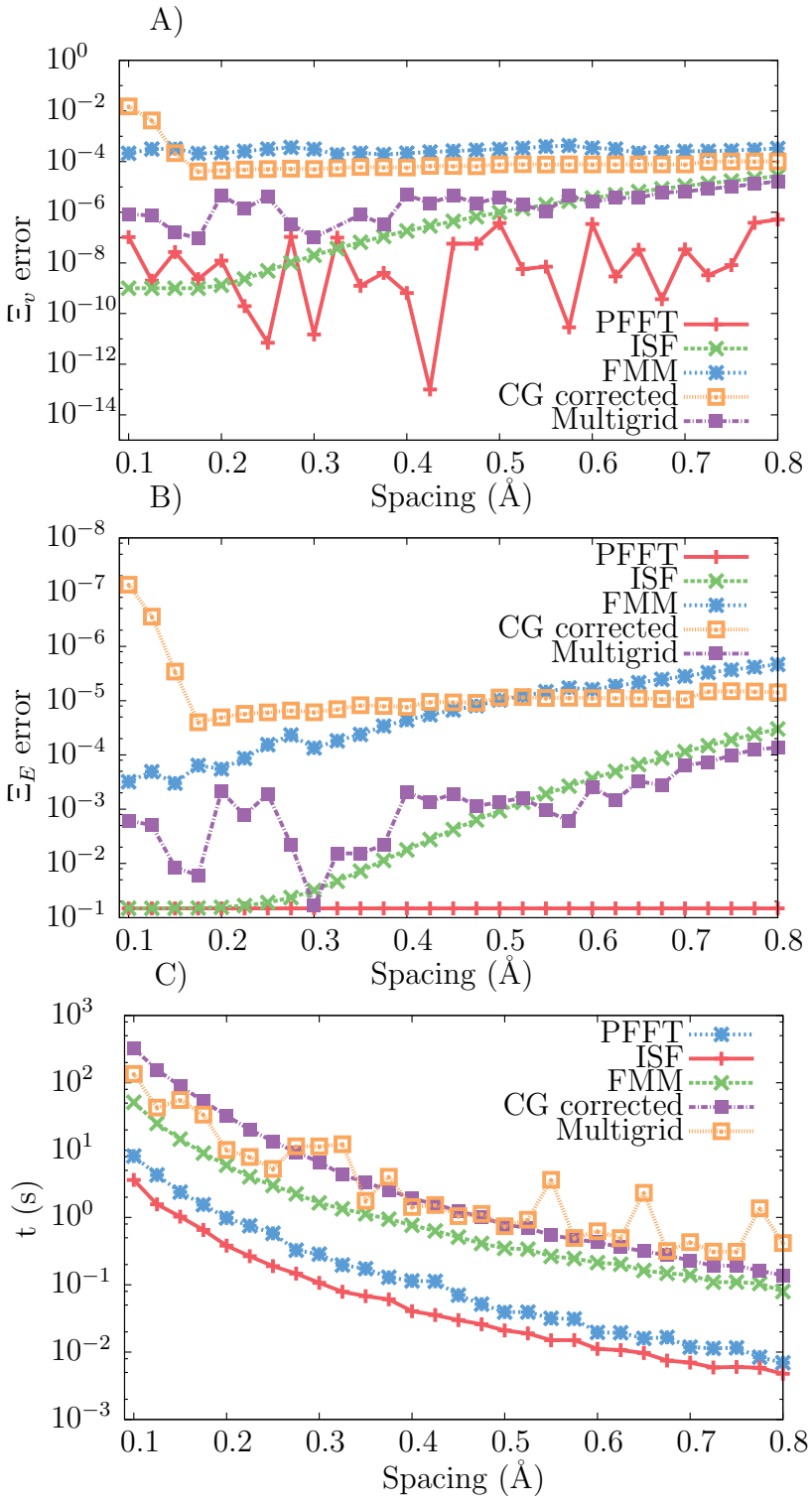


Figure D.1: A) Comparison of Ξ_v varying the spacing from 0.1 Å to 0.8 Å, of the system of $L_e = 15.8$ Å. B) Same for Ξ_E error. C) Comparison of time varying the spacing in the same range in Ganbo (x86-64). The times are given by running in 8 cores with 8 MPI processes.

cubic boxes, all using a spacing of 0.2 Å. This value was chosen because it is close to the values of the spacing typically used in the simulations of physical systems (where, unfortunately, the analytical calculation of the Hartree potential is not possible). The choice of other values of the spacing does not have a strong impact on the results of our tests, as can be viewed in Figure D.1. These tests used the same Gaussian charge distribution that was defined in the Section 6.2.1, with a cubic box of semi-edge $L_e = 15.8$ Å and variable spacing. The error Ξ_E as a function of the spacing (see Figure D.1A and Figure D.1B) is almost invariant within a wide radius around the used value of 0.2. The total number of mesh points increases with the cube of spacing^{-1}, and so does (essentially, regardless communication issues) the increasing of the computing time (see Figure D.1C). Too big values for the spacing lead to too coarse density samplings, and therefore to wrong Hartree potentials.

The implementations of Multigrid and Conjugate Gradients solvers we used in our tests are based on a multipole expansion. Each value of the input charge density represented on a mesh (ρ_{ijk}) is expressed with an analytical term via this multipole expansion, plus a numerical term. The Hartree potential created by the analytic part is calculated analytically, while the Hartree potential created by the numerical term is calculated numerically with either Multigrid or Conjugate Gradients method. So, since the use of the analytic term makes the numerical term smaller, numerical errors are expected to be reduced. Therefore, the smaller the difference between the charge density and its multipole expansion, the smaller the potential numerically calculated, and the higher the accuracy of the total Hartree potential calculation. An order for the multipole expansion (the input parameter `PoissonSolverMaxMultipole`, PSMM) must be chosen. It is to be stressed that arbitrarily higher values of PSMM do not lead to more accurate results; rather, there exists a PSMM that minimises the error for each problem. We ran some tests to obtain this optimal value. In them, we calculated the Ground State of a 180 atom chlorophyll system varying the value of PSMM. Table D.2 shows the obtained results; for PSMM= $8, 9$, and 10, the accumulation of errors was big enough for calculations not to converge (because beyond a given order, the spherical harmonics are too steep to describe the ρ, which is rather smooth). From the data of Table D.2, we chose PSMM= 7 for our simulations (with PSMM= 7 the HOMO-LUMO gap is equivalent to the reference value given by ISF, and the Hartree energy is the closest one).

	Hartree energy (eV)			HOMO-LUMO gap		
PSMM	ISF	CG	Multigrid	ISF	CG	Multigrid
4	240821.47	240813.51	240813.41	1.4489	1.2179	1.2178
6	240821.47	240813.93	240813.95	1.4489	1.4340	1.4353
7	240821.47	240815.01	240814.69	1.4489	1.4520	1.4506

Table D.2: Ground State values of the Hartree energy and the HOMO-LUMO gap as a function of the PSMM (input parameter of Multigrid and Conjugate Gradients solvers). Reference values are given by the ISF solver.

The implementation we used for FMM [81] allows one to tune the relative error of the calculations. Its expression is the quotient $(E_{ref} - E_n)/E_n$, i.e. the variable `DeltaEFMM`, where E_n is the Hartree energy calculated with the FMM method and

E_{ref} is an estimation of what its actual value is. We chose for our calculations a relative error of 10^{-4}. Note that this error corresponds only to the pairwise term of the Hartree potential, before the correction for charge distribution is applied (see Section 3.5.3 and D.IV).

D.III Correction time of MG and CG

As explained at the Chapter 6 (Section 3.5.4), the Multigrid and Conjugate Gradients solvers use a multipolar expansion for the charge, such that the potential created by this expansion can be analytically calculated, while the potential created by the remaining charge, i.e. original charge minus multipolar expansion, is calculated with either Multigrid or Conjugate Gradients. This analytical calculation requires similar times for both solvers. It requires between 0.2% and 8% of the Conjugate Gradients total execution time, and between 1.7% and 3.2% in the case of Multigrid for the standard tests performed. Therefore, in essence, it does not add a significant extra time. As an example, a system of $L_e = 15.8$ and spacing of 0.2 Å (4,019,679 mesh points) in Ganbo is shown in Table D.3. Very similar trend is shown in other system sizes and machines.

	Multigrid t(s)			CG corrected t(s)		
MPI proc.	Solver	Correction	Percentage	Solver	Correction	Percentage
1	89.02	7.16	8.04%	223.36	7.13	3.19%
2	57.46	3.53	6.14%	122.87	3.58	2.91%
4	29.61	1.87	6.32%	67.72	1.91	2.82%
8	15.60	1.00	6.41%	35.05	1.04	2.96%
16	8.62	0.53	6.20%	20.23	0.53	2.62%
32	4.57	0.28	6.16%	10.87	0.28	2.59%
64	2.84	0.14	4.92%	5.72	0.20	3.42%
128	2.18	0.07	3.17%	4.33	0.08	1.75%
256	5.19	0.08	1.49%	3.46	0.06	1.73%
512	10.04	0.02	0.23%	1.00	0.02	2.33%

Table D.3: Correction time (s) compared with the total solver time of Multigrid and Conjugate Gradients solvers in for a system of $L_e = 15.8$ Å (4,019,679 mesh points) in Ganbo machine from 1 to 512 MPI processes.

D.IV Correction terms for FMM

D.IV.1 General remarks

The Fast Multipole Method (FMM)[77, 78, 80, 79, 81] was devised to efficiently calculate pairwise potentials created by pointlike charges, like pairwise Coulomb potential. In the literature it is possible to find some modifications of the traditional FMM which deal with charges that are modelled as Gaussian functions [85]. Such modifications of FMM can be used into LCAO codes as Gaussian [200] or FHI-Aims. However, they are not useful when the charge distribution is represented through a set of discrete charge density values. The Fast Multipole Method presented in [16, 81] belongs to the family of FMM methods which calculate the electrostatic

potential created by a set of pointlike charges. This method is very accurate and efficient, but it needs some modifications to work in programs like OCTOPUS [3, 4], where the 3D mesh points actually represent charge densities. As stated in the Chapter 3 (Section 3.5), the electronic density is a $\mathbb{R}^3 \to \mathbb{R}$ field, where values of the $\mathbb{R}^3$ set correspond to an equispaced mesh (see Figure D.2C)). The variable $\rho_{j,k,l}$ is the charge density at the portion of volume (cell) centred in the point (j, k, l). Each cell is limited by the planes bisecting the lines that join two consecutive mesh points, and its volume is $\Omega = h^3$, being h the spacing between consecutive mesh points. The density $\rho_{j,k,l}$ is always negative and it is expected to vary slowly among nearby points.

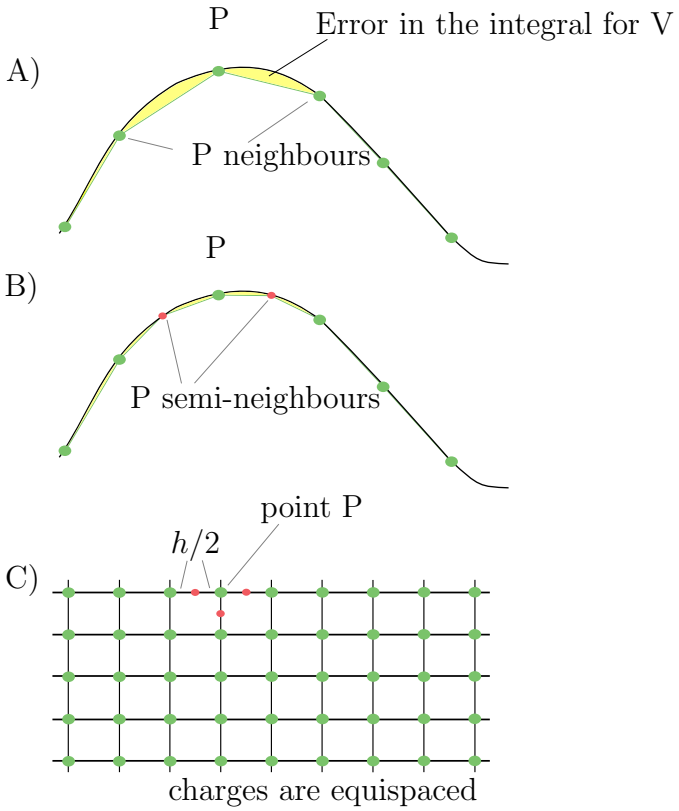


Figure D.2: Scheme of how the inclusion of semi-neighbours of point P (pink points) helps to improve the accuracy of the integration to calculate the Hartree potential. A) Scheme of the function whose integration will be approximated by a summation (surface under the green line), without considering semi-neighbours. B) id., considering semi-neighbours of point P; The error made by the approximation is proportional to the yellow surface in A) and B). C) 2D scheme of the mesh: green points are mesh points, while pink points are semi-neighbours of P.

The term v^{SI} in equation (3.28) can be calculated analytically as follows assuming

that the cell is a sphere of volume Ω:

$$\begin{aligned} v^{\mathrm{SI}}(\vec{r_0}) &= \int_\Omega d\vec{r}\frac{\rho(\vec{r})}{|\vec{r}-\vec{r_0}|} = \rho(\vec{r_0})\int_\Omega d\vec{r}\frac{1}{|\vec{r}-\vec{r_0}|} \\ &\simeq \rho(\vec{r_0})\int_0^R dr\int_0^{2\pi} d\phi\int_0^{\pi} d\theta\frac{r^2 sen(\theta)}{r} \\ &= \rho(\vec{r_0})2\pi h^2\left(\frac{3}{4\pi}\right)^{2/3} , \end{aligned} \tag{D.1}$$

where we have used the approximation of constant charge density within the cell. One may expect this approximate way to proceed to be less accurate than the numerical integration of $1/r$ in a cubic cell (what is also efficient, since the integration through an arbitrary size cube is proportional to the integral through a cube of unit volume). However, it happens the converse: the difference between both methods is small (about 1% of difference between integrals) but, due to error cancellations, the analytical method is slightly more accurate when calculating potentials.

The term $v^{\mathrm{corr.}}_{j,k,l}$ in (3.28) is included to calculate more accurately the potential created by the charge in cells nearby to (j,k,l). To devise a expression for it, we consider that the charge distribution is similar to sets of Gaussians centred in the atoms of the system. For Gaussian distributions, the greatest concavity near the centre of the Gaussian makes the influence of neighbouring points to be major for the potential. As we can see in the scheme of Figure D.2 A)-B), considering semi-neighbours of point P (in $\vec{r}_0 := (j,k,l)$), i.e., points whose distance to P is not h, but $h/2$, the integral of equation 16 of the paper can be calculated in a much more accurate way.

D.IV.2 Method 1: 6-neighbours correction

We build a corrective term by calculating the charge in the 6 semi-neighbours of every point of the mesh $\vec{r_0}$ (see Figure D.3 for a intuitive scheme). The total correction term is the potential created by the semi-neighbours ($v^{\mathrm{corr.}+}$) minus the potential created by the charge lying in the volume of the semi-neighbour cells that was already counted in v^{FMM} or in v^{SI} ($v^{\mathrm{corr.}-}$):

$$v^{\mathrm{corr.}}(\vec{r_0}) = v^{\mathrm{corr.}+}(\vec{r_0}) - v^{\mathrm{corr.}-}(\vec{r_0}) \ . \tag{D.2}$$

In order to calculate $v^{\mathrm{corr.}+}$, we use the formula por the 3rd degree interpolation polynomial:

$$f(0) = \frac{(-1)}{16}f\left(\frac{-3}{2}h\right) + \frac{9}{16}f\left(\frac{-h}{2}\right) + \frac{9}{16}f\left(\frac{h}{2}\right) - \frac{(-1)}{16}f\left(\frac{3}{2}h\right) \ , \tag{D.3a}$$

$$f\left(\frac{-h}{2}\right) = \frac{(-1)}{16}f\left(-2h\right) + \frac{9}{16}f\left(-h\right) + \frac{9}{16}f\left(0\right) - \frac{(-1)}{16}f\left(h\right) \ , \tag{D.3b}$$

$$f\left(\frac{h}{2}\right) = \frac{(-1)}{16}f\left(-h\right) + \frac{9}{16}f\left(0\right) + \frac{9}{16}f\left(h\right) - \frac{(-1)}{16}f\left(2h\right) \ . \tag{D.3c}$$

So, the semi-neighbours of $\vec{r_0} = (x_0, y_0, z_0)$ are

$$\begin{aligned}
\rho(x_0 - h/2, y_0, z_0) =& \frac{-1}{16}\rho\left(x_0 - 2h, y_0, z_0\right) + \frac{9}{16}\rho\left(x_0 - h, y_0, z_0\right) \\
&+ \frac{9}{16}\rho\left(x_0, y_0, z_0\right) - \frac{1}{16}\rho\left(x_0 + h, y_0, z_0\right) \ ; && \text{(D.4a)} \\
\rho(x_0 + h/2, y_0, z_0) =& \frac{-1}{16}\rho\left(x_0 - h, y_0, z_0\right) + \frac{9}{16}\rho\left(x_0, y_0, z_0\right) \\
&+ \frac{9}{16}\rho\left(x_0 + h, y_0, z_0\right) - \frac{1}{16}\rho\left(x_0 + 2h, y_0, z_0\right) \ ; && \text{(D.4b)} \\
\rho(x_0, y_0 - h/2, z_0) =& \frac{-1}{16}\rho\left(x_0, y_0 - 2h, z_0\right) + \frac{9}{16}\rho\left(x_0, y_0 - h, z_0\right) \\
&+ \frac{9}{16}\rho\left(x_0, y_0, z_0\right) - \frac{1}{16}\rho\left(x_0, y_0 + h, z_0\right) \ ; && \text{(D.4c)} \\
\rho(x_0, y_0 + h/2, z_0) =& \frac{-1}{16}\rho\left(x_0, y_0 - h, z_0\right) + \frac{9}{16}\rho\left(x_0, y_0, z_0\right) \\
&+ \frac{9}{16}\rho\left(x_0, y_0 + h, z_0\right) - \frac{1}{16}\rho\left(x_0, y_0 + 2h, z_0\right) \ ; && \text{(D.4d)} \\
\rho(x_0, y_0, z_0 - h/2) =& \frac{-1}{16}\rho\left(x_0, y_0, z_0 - 2h\right) + \frac{9}{16}\rho\left(x_0, y_0, z_0 - h\right) \\
&+ \frac{9}{16}\rho\left(x_0, y_0, z_0\right) - \frac{1}{16}\rho\left(x_0, y_0, z_0 + h\right) \ ; && \text{(D.4e)} \\
\rho(x_0, y_0, z_0 + h/2) =& \frac{-1}{16}\rho\left(x_0, y_0, z_0 - h\right) + \frac{9}{16}\rho\left(x_0, y_0, z_0\right) \\
&+ \frac{9}{16}\rho\left(x_0, y_0, z_0 + h\right) - \frac{1}{16}\rho\left(x_0, y_0, z_0 + 2h\right) \ . && \text{(D.4f)}
\end{aligned}$$

We consider all this six charges to be homogeneously distributed in cells whose volume is $\Omega/8$. The distance between the centre of these small cells and the centre of the cell whose v we are calculating (i.e., the cell centred in $\vec{r_0}$) is $h/2$. So the first part of $v^{\text{corr.}}(\vec{r_0})$ is

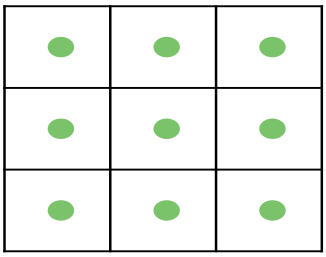
$$\begin{aligned}
v^{\text{corr.}+}(\vec{r_0}) &= \Big(\rho(x_0 - h/2, y_0, z_0) + \rho(x_0 + h/2, y_0, z_0) + \ldots + \\
&+ \rho(x_0, y_0, z_0 + h/2)\Big)\left(\frac{\Omega}{8}\right)\left(\frac{1}{h/2}\right) \ . && \text{(D.5)}
\end{aligned}$$

Since we have created these new 6 cells, we must subtract the potential created by their corresponding volume from that created by the cells whose volume is partly occupied by these new cells. This potential is:

$$\begin{aligned}
v^{\text{corr.}-}(\vec{r_0}) &= \Big(\rho(x_0 - h, y_0, z_0) + \rho(x_0 + h, y_0, z_0) + \rho(x_0, y_0 - h, z_0) + \rho(x_0, y_0 + h, z_0) \\
&+ \rho(x_0, y_0, z_0 - h) + \rho(x_0, y_0, z_0 + h)\Big)\left(\frac{\Omega}{16}\right)\left(\frac{1}{h}\right) + \alpha\ v^{\text{SI}}(\vec{r_0}) \ . && \text{(D.6)}
\end{aligned}$$

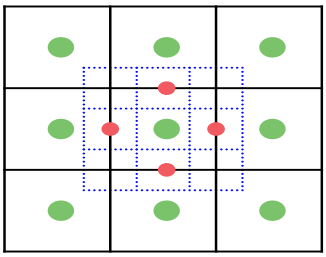
The aim of the term $\alpha\ v^{\text{SI}}$ (i.e., the variable `AlphaFMM` in Octopus) is to compensate the errors arising from the assumption that the charge is concentrated at the centre of the cells and reduced cells. The value of α is tuned to minimise the errors in the potentials.

A)



Original box: all cell's size is h^2 (2D)

B)



New: Semi-neighbours cell sizes are $h^2/4$

Side cells sizes are $7/8h^2$

Central cell size is $h^2/2$

(all in 2D)

Figure D.3: 2D example of the position of cells containing semi neighbours. Assume the centre of the plots is $\vec{r_0}$, the point where we want to calculate the correcting term for the potential. The volume of semi-neighbour cells is $h^2/4$ in 2D, and $\Omega/8$ in 3D. One half of the semi-neighbour cell occupies the volume of a neighbour cell (the cell whose centre is h away from $\vec{r_0}$). The other half of the semi-neighbour cell occupies the space of the $\vec{r_0}$-centred cell itself.

It is worth to re-express as follows the correction terms of eqs. (D.2), (D.5) and (D.6) avoiding to call to every variable more than once for the sake of getting higher computational efficiency:

$$
\begin{aligned}
v^{\mathrm{SI}}(\vec{r_0}) + v^{\mathrm{corr.}}(\vec{r_0}) = h^2 \Big[& \quad \rho(x_0,y_0,z_0)\big(27/32 + (1-\alpha)2\pi(3/4\pi)^{2/3}\big) \\
& +(1/16)\Big(\rho(x_0-h,y_0,z_0) + \rho(x_0+h,y_0,z_0) + \rho(x_0,y_0-h,z_0) \\
& \quad +\rho(x_0,y_0+h,z_0) + \rho(x_0,y_0,z_0-h) + \rho(x_0,y_0,z_0+h) \\
& \quad -(1/4)\big(\rho(x_0-2h,y_0,z_0) + \rho(x_0+2h,y_0,z_0) + \rho(x_0,y_0-2h,z_0) \\
& \quad +\rho(x_0,y_0+2h,z_0) + \rho(x_0,y_0,z_0-2h) + \rho(x_0,y_0,z_0+2h)\big)\Big)\Big] \,.
\end{aligned}
\tag{D.7}
$$

We ran tests using the error formula $E := \sqrt{\sum_i (v^{\mathrm{Exact}}(\vec{r_i}) - v^{\mathrm{FMM}}(\vec{r_i}))^2}$, with the index i running for all points of the system. The inclusion of the correcting term introduced in this Section typically reduced E in a factor about 50.

D.IV.3 Method 2: 124-neighbours correction

This method is similar to the one explained in the previous Section, but with two differences

- It uses 3D interpolation polynomials, instead of 1D polynomials. Then, it considers $5^3 - 1 = 124$ neighbours in a cube of edge $5h$ centred in $\vec{r_0}$ to calculate the corrective term for $V(\vec{r_0})$
- The interpolation polynomials representing $\rho(x,y,z)$ are numerically inte-

grated (after their division by r). This is, we calculate

$$v^{\text{corr.+}}(\vec{r_0}) \;=\; \int_{125\Omega} d\vec{r}\frac{\rho(\vec{r})}{|\vec{r}-\vec{r_0}|} \simeq \int_{125\Omega} d\vec{r}\frac{Pol(\vec{r})}{|\vec{r}-\vec{r_0}|}\ . \tag{D.8}$$

The integration is to be performed between -5/2h and 5/2h for x, y and z. The interpolation polynomial (with 125 support points) $Pol(\vec{r})$ is

$$Pol(\vec{r}) = \sum_{i=1}^{5}\sum_{j=1}^{5}\sum_{k=1}^{5}\rho(x_0+(i-3)h, y_0+(j-3)h, z_0+(k-3)h)\alpha_i(x)\alpha_j(y)\alpha_k(z)\ , \tag{D.9}$$

being

$$\alpha_1(\xi) := \frac{\xi^4}{24} - \frac{\xi^3}{12} - \frac{\xi^2}{24} + \frac{\xi}{12}\ , \tag{D.10a}$$

$$\alpha_2(\xi) := -\frac{\xi^4}{6} + \frac{\xi^3}{6} + \frac{2\xi^2}{3} - \frac{2\xi}{3}\ , \tag{D.10b}$$

$$\alpha_3(\xi) := \frac{\xi^4}{4} - \frac{5\xi^2}{4} + 1\ , \tag{D.10c}$$

$$\alpha_4(\xi) := -\frac{\xi^4}{6} - \frac{\xi^3}{6} + \frac{2\xi^2}{3} + \frac{2\xi}{3}\ , \tag{D.10d}$$

$$\alpha_5(\xi) := \frac{\xi^4}{24} + \frac{\xi^3}{12} - \frac{\xi^2}{24} - \frac{\xi}{12}\ . \tag{D.10e}$$

The quotient of the polynomials $\alpha_i(x)\alpha_j(y)\alpha_k(z)$ divided by $|\vec{r}-\vec{r_0}|$ can be numerically integrated through the cubic cell of edge $5h$ and centred in x_0. Such integrals can indeed be tabulated, because equation ((D.8)) implies $v^{\text{corr.+}}(\vec{r_0}) = v^{\text{corr.+}}(\vec{r_0})|_{h=1}\cdot h^2$. Terms of $\alpha_i(x)\alpha_j(y)\alpha_k(z)/|\vec{r}-\vec{r_0}|$ are often odd functions whose integral is null. The non-zero integrals taking part in ((D.1)) (with $h=1$) can be easily calculated numerically.

Therefore

$$\begin{aligned}
v^{\text{corr.+}}(\vec{r_0}) \;&=\; \sum_{i=1}^{5}\sum_{j=1}^{5}\sum_{k=1}^{5}\rho(x_0+(i-3)h, y_0+(j-3)h, z_0+(k-3)h)\cdot \\
&\quad \int_{125\Omega} d\vec{r}\frac{\alpha_i(x)\alpha_j(y)\alpha_k(z)}{|\vec{r}-\vec{r_0}|} \\
&=\; \sum_{i=1}^{5}\sum_{j=1}^{5}\sum_{k=1}^{5}\rho(x_0+(i-3)h, y_0+(j-3)h, z_0+(k-3)h)\cdot \\
&\quad \sum_{l=1}^{5}\sum_{m=1}^{5}\sum_{n=1}^{5}\alpha_{i,l}\alpha_{j,m}\alpha_{k,n}\int_{125\Omega} d\vec{r}\frac{x^{l-1}y^{m-1}z^{n-1}}{|\vec{r}-\vec{r_0}|} \\
&=\; \sum_{i=1}^{5}\sum_{j=1}^{5}\sum_{k=1}^{5}\rho(x_0+(i-3)h, y_0+(j-3)h, z_0+(k-3)h)\cdot \\
&\quad \sum_{l=1}^{5}\sum_{m=1}^{5}\sum_{n=1}^{5}\alpha_{i,l}\alpha_{j,m}\alpha_{k,n}\beta(l-1,m-1,n-1)h^2\ ,
\end{aligned} \tag{D.11}$$

where $\alpha_{i,l}$ is the coefficient of ξ^{l-1} if $\alpha_i(\xi)$ and

$$\beta(l,m,n) := \int_{-1/2}^{1/2} dx \int_{-1/2}^{1/2} dy \int_{-1/2}^{1/2} dz \frac{x^l y^m z^n}{\sqrt{x^2+y^2+z^2}} \ . \tag{D.12}$$

In this case, $v^{\text{corr.}-}$ is equal to all the contributions to $V(\vec{r_0})$ due to charges whose position (x,y,z) satisfies

$$|x-x_0| <= 2h;\ |y-y_0| <= 2h;\ |z-z_0| <= 2h \ , \tag{D.13}$$

including self-interaction integral.

This way to calculate $v^{\text{corr.}+}$ is not inefficient, because only 27 integrals are not null, and both α and β are known. In order to calculate $v^{\text{corr.}+}(\vec{r_0})$ we need 125 products and additions, what is essentially the same number of operations which is required in order to calculate the potential created in $\vec{r_0}$ by the neighbouring points (whose calculation can be removed and then saved). Nevertheless, results using this correction method were worse than that obtained using the first method, so only that one was implemented into the standard version of OCTOPUS.

Acronyms

AB *ab-initio*. iii, 4, 24, 27, 29, 30, 64, 68

API Application Program Interface. 42

CG Conjugate Gradients. iv, 31, 37, 38, 65–67, 69, 70, 72, 73, 76–81, 97, 106, 127, 128, 130, 131

DFT Density Functional Theory. i–iv, viii, V, VI, 23, 24, 27–30, 42, 44, 48, 64, 69, 77, 81, 100, 101, 106

DS data-structure. 22, 46, 47, 84, 85, 92

DSM Distributed Shared Memory. 10

FFT Fast Fourier Transforms. iv, v, 31–33, 47, 58, 65, 66, 68–70, 72, 73, 77, 80, 81, 86, 108, 127

flop/s floating point operations per second. 2, 11, 14–19, 48

FMM Fast Multipole Method. v, 31, 34, 37, 59, 65–69, 72, 78–81, 106, 127, 128, 130, 131

GS Ground State. 26–28, 43, 48, 52–56, 59, 61, 69, 87, 89–91, 95–97, 100, 106, 108, 125, 126, 130

HPC High Performance Computing. ii, iv, V, 2, 15, 42, 52, 54, 64, 90, 106, 107

ISF Interpolating Scaling Function. iv–vi, 31, 33, 42, 58, 65–70, 72, 79–81, 86, 106–108, 127, 128, 130

KS Kohn-Sham. 28, 30, 43–45, 48, 52, 64

LCAO Linear Combination of Atomic Orbitals. 23, 24, 28, 30, 87, 92, 131

LHC–II Light Harvesting Complex II. ii, vii, viii, V, VI, 5–7, 52, 53, 88, 92–95, 98–104, 107, 108, 115, 117–121

MG Multigrid. iv, 31, 37–40, 65–70, 72, 73, 76–81, 106, 127, 128, 130, 131

MPP Massively Parallel Processors. 10, 11

PFFT Parallel Fast Fourier Transforms. v, vi, 42, 65–67, 70, 72–81, 86, 106–108, 127, 128

PRACE Partnership for Advanced Computing in Europe. V, VI, 4, 48, 106, 116

RS real-space. 43, 45, 46, 54, 64, 80, 83, 84

SCF Self Consistent Field. 28, 29, 54, 87

SMP Simetric Multiprocessor. 10

TD Time-Dependent. 29, 43, 44, 48, 52, 53, 55, 56, 58–61, 90, 106, 125, 126

TDDFT Time-Dependent Density Functional Theory. i–iv, vii, viii, V, VI, 4, 23, 24, 28–30, 41–44, 48, 64, 77, 81, 92, 93, 98, 99, 101–104, 106–108

TI Time-Independent. 26, 44

XC Exchange-Correlation. 4, 28, 30, 44, 69, 70, 98

License

Izugarri merezi izan duen bide honek
asko erakutsi dit
gustatzen zaidan horretan jarduteko aukera eman dit
eta
nekiena baino askoz gehiago jakinda amaitzen dut
—
Hala ere
orain
nituena baino galdera gehiago ditut
—
Hasi dudan bide honek
amaierarik ez du

Maitiari