Hardware, Software co-synthesis med genetisk multiobjektive algorithmer

Peter Kjærulff, s991267

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Resume

Denne rapport omhandler arbejdet med at udvikle en genetisk algorithme til undersøgelse af forskellige indlejrede systemdesigns i den indledende fase i udviklingen af et indlejret system. Algorithmens løsninger tager højde for brugernes interesse i både at optimere på pris og energiforbrug og er således multiobjektive. Algorithmen er afprøvet i et casestudie hvor designmuligheder for en smartphone undersøges og analyseres.

Abstract

This report concern the work developing a genetic algorithm for exploration of different embedded system designs i the early stage of the design process. The algorithms solutions takes in to account the users interests in both optimization of price and energy consumption and is in this way multiobjektive. The algorithem is used in a case studie where desigs for a smartphone i explored and anlyzed.

Forord

Jeg vil gerne takke begge mine vejledere Thomas K. Stidsen og Jan Madsen for god vejledning og den megen interesse der er blevet vist for projektet. Derudover skal Shankar Mahadevan have en stor tak for at den tid han har givet til at svare på sprøgsmål vurdere resultater.

Indhold

Kapitel 1

Indføring i problemet

1.1 Introduktion

Indlejrede systemer er computersystemer indbygget i apparater, hvor de har en specifik funktion, der bidrager til apparatets samlede funktionalitet. De indlejrede systemer har ofte få krævende opgaver de skal løse i modsætning til almindelige PC'ere der skal kunne løse mange forskellige opgaver. De indlejrede systemer vinder mere og mere indpas i vores hverdag hvor de efterhånden optræder i mange produkter, så som mikrobølgeovne, høreapperater, biler og mobiltelefoner. Udover at der ofte stilles store krav til systemerne på helt specifikke områder som hastighed, energiforbrug og størrelse vil der også i fremtiden blive stillet høje krav til udviklingstiden af systemerne. Dette sammen med den hastige teknologiske udvikling inden for området bidrager til øgede krav til de ingeniører, der skal designe systemerne.
Et område der for tiden er i hastig udvikling er markedet for mobil forbrugerelektronik så som mobil telefoner, mp3-afspillere og PDA'ere. Med udviklingen inden for teknologien som den forløber nu har disse produkter en meget kort levetid før de er forældede på markedet. Det betyder at det er af afgørende betydning at minimere udviklingstiden så produktet når tidligere ud i butikkerne. En ofte anvendt fremgangsmåde i udvikling af indlejrede systemer er at tage udgangspunkt i et allerede kendt og anvendt design og designe det nye system ud fra dette manuelt. Dette er både meget tidskrævende og giver et meget konservativt valg af teknologi. Indføringen af automatiserede metoder til undersøgelsen af designs vil udover at råde bod på disse to problemer kunne give ingeniørerne et større overblik over de designmuligheder de har, samt en fornemmelse af de tradeoffs der kunne ligge imellem forskellige designkriterier. Det følgende afsnit vil dels beskrive skridtene

i cosynthesis problemet nærmere, dels give den første introduktion til de hardware komponenter der indgår i et indlejret system.

1.2 Co-synthesis

Opgaven at finde en hardware arkitektur for et indlejret system kaldes co-synthesis. Udover valget af hardware komponenter indebærer co-synthesis også en fordeling af beregningsopgaver på processorerne i systemet samt en evaluering af systemets kørselstid. Det sidste kræver løsningen af et skemalægningsproblem. Co-synthesis betragtes ofte som sammensat af fire delproblemer.

Delproblemerne i co-synthesis:

1) Allokeringen : Bestemmelse af antal og type af hardware komponenter der skal bruges.

2) 'Mapping' : Bestemmelse af hvilke dele af softwaren der skal afvikles på hvilken hardware.

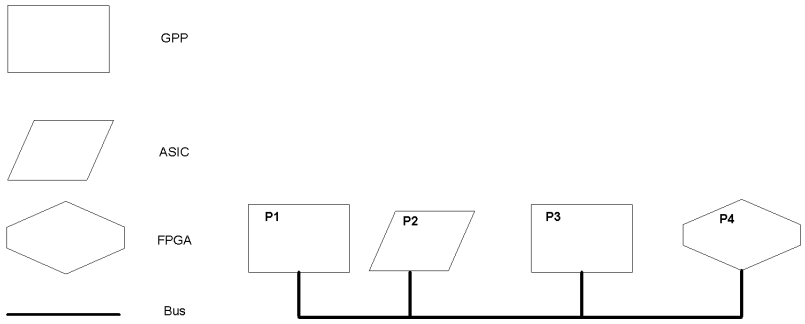
3) Skemalægning : Bestemmelse af til hvilken tid hver opgave skal afvikles.

4) Evaluering af arkitekturen : Her evalueres arkitekturen med hensyn til et eller flere designkriterier.

Delproblemerne i co-synthesis er alle indbyrdes afhængige. For at løse mappingproblemet kræves således en løsning af allokeringsproblemet, skemalægningen kræver en løsning til mappingproblemet, og evalueringen af arkitekturen kræver løsningen af et skemalægningsproblem.

1.3 Hardware komponenterne

Hardware komponenter udgøres af processorerne, bufferne, busserne samt broerne. Processorerne laver de beregninger der ligger i applikationerne, mens busserne foretager dataoverførsel imellem processorerne. Af processorer haves tre forskellige typer; GPP'er (General Purpose Processors), ASIC'er (Application Specific Integrated Circuits) og FPGA'er (Field Programmable Gate Arrays). ASIC'erne og FPGA'erne har lavere energiforbrug end GPP'erne til gengæld er de dyrere, og ikke alle opgaver kan udføres på dem. Herunder er vist et eksempel på en arkitektur.



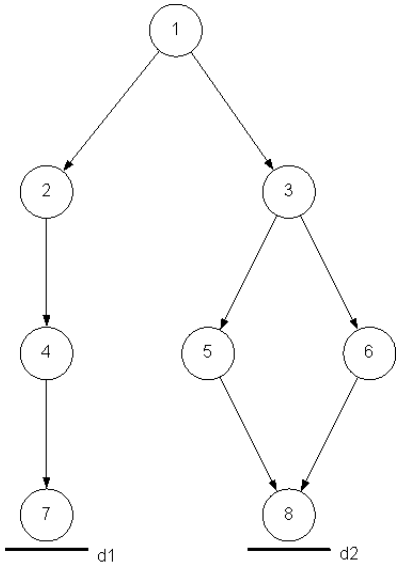
(a) Signaturforklaring (b) Arkitektureksempler

Figur 1.1: Eksempler på embedded system arkitektur. (a) Signaturforklaring (b) Arkitektureksempel

Første skridt i co-synthesis problemet var som tidligere nævnt at finde en arkitektur for det indlejrede system. Lad os antage at den ovenstående arkitektur ønskes undersøgt nærmere. Næste skridt er at lægge softwaren ned på processorerne også kendt som mapping. For at evaluere mappingen er det nødvendigt enten at lægge softwaren ned på hardwaren eller at have en model af hvordan softwaren opfører sig på hardwaren. Det første er meget tidskrævende og gøres først til sidst i designprocessen. Det er derfor nødvendigt med en model for softwaren i det indlejrede system. Denne model er opgavegrafen.

1.3.1 Opgavegraf

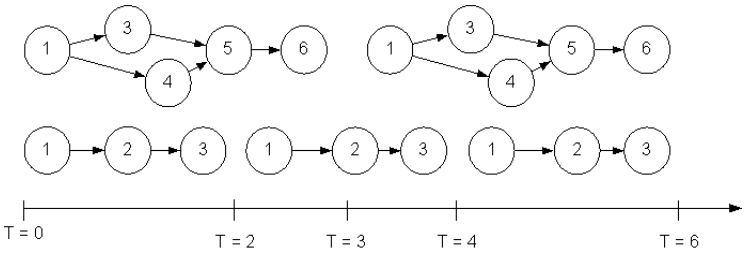
Opgavegrafen er en orienteret ikke cyklisk graf hvor knuderne angiver beregningsopgaver i softwaren, mens kanterne angiver dataafhængigheder imellen beregningsopgaverne. En knudes indgående kanter beskriver de data, der kræves før beregningen af en opgave kan påbegyndes, mens de udgående kanter angiver data, der sendes videre til andre beregninger. Hvis der er givet en begrænsning på hvor længe applikationen må være om at eksekvere, vil dette være givet som deadlines på slutknuderne i grafen.



Figur 1.2: Eksempel på en opgavegraf, med deadlines d1 og d2.

Dette projekt er begrænset ved kun at arbejde med opgavegrafer hvor der haves en fælles deadline på slutknuder i opgavegraferne og kun på disse. Der er dog ikke noget fundamentalt til hindere for at der kan indføres forskellige deadlines samt deadlines på andre knuder end slutknuderne i modellen.

Hvis der indgår flere applikationer i det indlejrede system vil der ligeledes være flere opgavegrafer, i det tilfælde taler man om 'multirate embedded systems'. Haves et sådan system er det nødvendigt at undersøge om alle opgavegrafer kan nå deres deadlines, når de skal dele ressourcer. Til dette formål genererer man hyperperioden, der er en worst case for antallet af opgavegrafer der skal afvikles samtidigt. I hyperperioden skal alle opgavegrafer afvikles samtidigt og opgavegrafer med korte deadlines gentages indtil hyperperioden er færdig. Længden af hyperperioden er den mindste fælles divisor af opgavegrafernes deadlines.

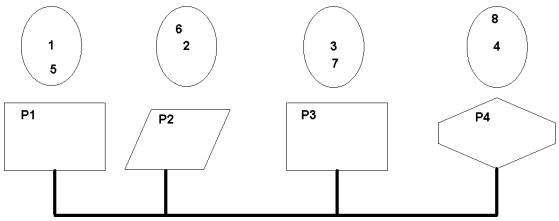


Figur 1.3: Hyperperioden for et indlejret system med to applikationer. Perioden for opgavegrafernes gentagelse er lig deres deadline.

Figuren ovenfor viser en hyperperiode med to opgavegrafer. Hyperperioden er mindste fælles divisor af de to opgavegrafers deadlines hhv 2 og 3, altså en hyperperioden på 6.

1.3.2 Mapping

I tildelingen af opgaver til processorer er det vigtigt at få en god fordeling af opgaverne imellem processorerne så enkelte processorer ikke bliver overbebyrdede og dermed skaber en flaskehals i systemet. Samtidigt er det fordelagtigt at samle opgaver med indbyrdes datakommunikation på samme processor for at mindske kommunikationen over busserne, da dette tager længere tid og kræver mere energi end intern dataoverførsel. Nedenfor er vist et eksempel på mapping af opgaver til processorer. Det skal bemærkes at i 'mappingen' er rækkefølgen af opgavernes udførsel endnu ikke fastlagt.

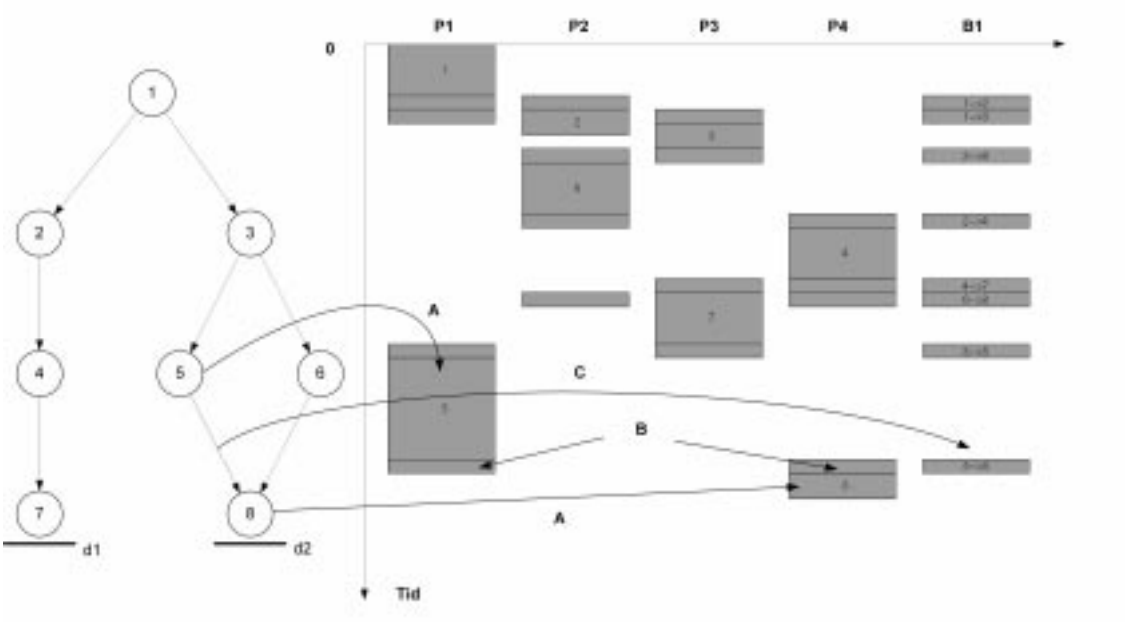


Figur 1.4: Mapping af opgavegrafen fra 1.2 til arkitekturerne fra 1.1. Over hver processor haves en uordnet mængde af opgaver tildelt denne.

1.3.3 Skemalægning

Det næste skridt i co-synthesis efter mappingen er skemalægningen. Skemalægningen er nødvendigt for at undersøge om applikationerne bliver færdige inden for de givne deadlines. Nedenfor er vist

et skema for skemalægningen af opgavegrafen fra 1.2 til arkitekturen fra 1.1.



Figur 1.5: Eksempel på skemalægningen af en opgavegraf. **A**: Beregningsopgave 5 skemalagt på **P2** og beregningsopgave 8 skemalagt på **P4**. **B**: Processorerne **P2** og **P4** optaget med hhv at sende og modtage kommunikations opgave 5→8. **C**: Kommunikationen 5→8 skemalagt på bus.

1.4 Udvidelse af Co-synthesis problemet - indføring af buffere og broer

Det ovenstående har beskrevet co-synthesis problemet som det er løst hos [1] uden buffere og broer. I det næste vil disse elementer blive beskrevet og eksemplificeret som en del af co-synthesis problemet. Først vil modellen for elementerne blive beskrevet, dernæst vil deres indførsel i skemalægningsalgorithmen blive beskrevet.

1.4.1 Buffere og broer

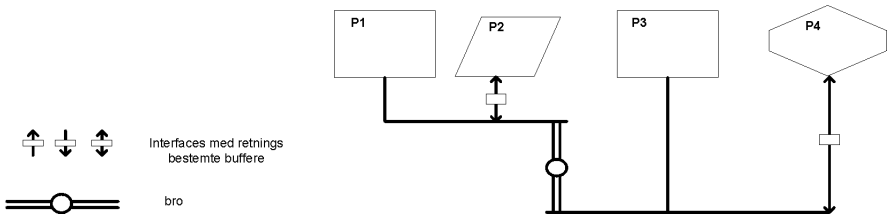
Tilslutningen imellem processorerne og busserne laves med et interface. Til hver processors interface kan der være tilknyttet en buffer. Bufferen gør det muligt for processoren at arbejde videre, mens bufferen kommunikerer med bussen. Bufferen kan desuden bruges som et lager, hvor kommunikationsopgaven ligger, indtil kommunikationen sættes i gang. Der findes 3 typer af buffere;

outputbufferen: Der behandler kommunikationsopgaver der sendes til bussen.

inputbufferen : Der behandler kommunikationsopgaver der modtages fra bussen.

input/outputbufferen: Der kan behandle kommunikation både til og fra processoren.

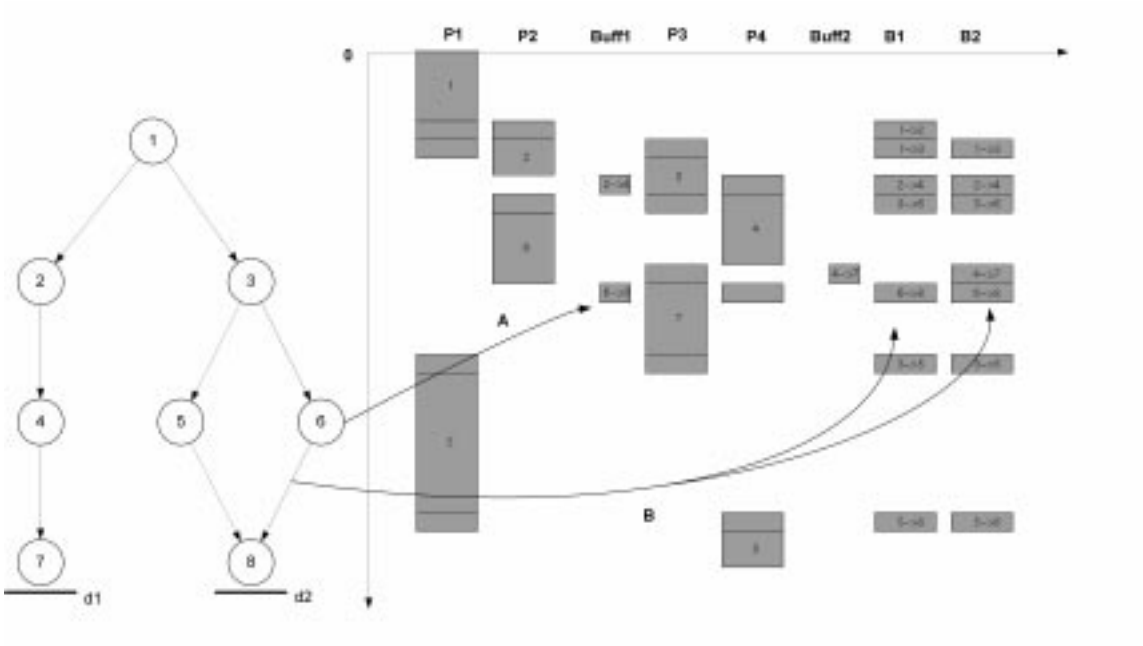
Hver af bufferne har en størrelse, der bestemmer hvor meget data der kan gemmes på den. Hvis der ikke er tilstrækkelig med hukommelse til en kommunikationsopgave i bufferen vil processoren selv skulle behandle dataen og vil derfor ikke være til rådighed til nye beregninger, før kommunikationen er overstået. I modellen antages det at der ingen kommunikationstid er imellem processor og buffer samt bus og buffer. Det antages desuden at bufferne kan levere data til busserne med deres frekvens. Herunder er vist en arkitektur der benytter de nye hardware komponenterne.



(a) Signaturforklaring (b) Arkitektureksempler

Figur 1.6: (a) Signaturforklaring, (b) Arkitektureksempel

I arkitekturen ovenfor er der også benytte droer. Broerne benyttes til at forbinde busser i arkitekturen. De er derfor nødvendige hvis der er mere end en bus i arkitekturen. I arkitekturen ovenfor har indførslen af en bro muliggjort at **P1** og **P2** kan kommunikere samtidigt med **P3** og **P4** . Herunder er vist et skema hvori de nye hardware komponenter indgår.



Figur 1.7: Eksempel på skemalægning af en opgavegraf på arkitektur med buffere og broer. **A**: Buffere kan benyttes til kommunikation imellem busser og processorer i stedet for at optage **P2**. **B**: Skemalægning af kommunikation på mere end en bus. Da opgaverne 6 og 8 er tildelt processorerne **P2** og **P4** på forskellige busser, **B1** og **B2** vil kommunikationen løbe over to busser.

Skemalægning der i sin grundform er kendt for at være et NP-complete problem [2], optræder i forskellige varianter indenfor co-synthesis. I mange indlejrede systemer i dag sker skemalægningen af opgaver dynamisk. En mobiltelefon har f.eks et operativsystem der allokerer hukommelse til de applikationer brugeren sætter i gang. Den skemalægning der fortages af operativsystemet er et eksempel på dynamisk skemalægning. Man kan også vælge at løse skemalægningsproblemet som et statisk problem i stedet, hvilket også har en vis interesse. Hyperperioden udgør som tidligere nævnt en worst case for antallet af opgaver der kan indtræffe at køre samtidigt. Hvis der derfor kan findes et statisk skema for dette tilfælde, vil det med sikkerhed kunne siges at systemet kan klare de belastninger der måtte komme. I co-synthesis problemet vil man typisk begrænse sig til at betragte det statiske problem, hvilket også er tilfældet i dette projekt.
Skemalægningsalgorithmerne kan udover at være statiske eller dynamiske være 'preemptive' eller

'non-preemptive'. I en 'preemptive' algorithme er det muligt at afbryde en opgave og starte den op igen senere, hvilket det ikke er i en 'non-preemptive'. Det ville have været intressant at drage nytte af 'preemption' i skemalægningsalgorithmen, men det har desværre ikke været tid til dette i projektet.

1.5 Tidligere arbejde

Der er efterhånden foretaget en del forskning inden for algorithmer til co-synthesis, der er dog meget forskel på detaljeringsgraden af de modeller der bruges, generaliteten af de problemer der løses såvel som hvilke kriterier der optimeres på. Den mest succesfulde og mest brugte algorithme type er genetiske algorithmer. Sammenlignet med de MIP problemer der er lavet for problemet har de genetiske algorithmer for det første den fordel at objektkriterierne kan holdes adskilt og optimeres hver for sig i modsætning til MIP problemerne hvor kriterierne lægges sammen i en vægtet sum, og for det andet kan de genetiske algorithmer håndtere meget større opgavegrafer.
Af tidligere arbejde inden for co-synthesis kan nævnes [1] og [4]. I begge benyttes genetiske algorithmer til løsning af 'multirate embedded systems'. I [1] benyttes en genetiske algorithme til løsning af allokering og 'mapping' problemet, mens en 'statisk list scheduling' algorithme benyttes til skemalægningen. Arkitekturerne der undersøges kan indeholde GPP'er, ASIC'er og FPGA'er.
I [4] fokuseres der på optimering af energiforbruget ved ændring af processorernes frekvens. Det er en kendt teknik, men dette er det første sted hvor muligheden er set undersøgt indgående som en del af co-synthesis problemet. Den udviklede algorithme er dog begrænset til at undersøge mappingen for én arkitektur af gangen og der optimeres udelukkende på energiforbruget.
I sammenligning med de ovenstående tilgangsvinkler til problemet lægger dette projekt sig tættest på [1]. Der vil blive brugt en statisk list schedulingsalgorithme til skemalægningen og de samme objektkriterier, nemlig pris, energiforbrug og deadline violation, optimeres. Modellen for de indlejrede systemer i dette projekt vil dog være en udvidelse af modellen hos [1], idet der medtages buffere og broer i arkitekturen. Desuden foretager skemalægningsalgorithmen 'mappingen' af kommunikationsopgaver til busser, hvilket gør at kommunikationen kan mappes med henblik på at opnå en tidlig afslutningstid af opgavegrafen.

Kapitel 2

En Model for Embedded Systems Komponenter

2.1 En model for indlejrede systemer

I dette kapitel vil datasættet der definerer et co-synthesis problem blive beskrevet. Datasættet omfatter opgavegraferne samt tabeller over hardware komponenternes specifikationer. Udfra disse data er det muligt at finde kørselstid og energiforbrug for en enkelte opgaver, hvilket vil blive vist. Det samlede energiforbrug for et indlejret system findes ved at samle energiforbruget for alle opgaver i hyperperioden, hvilket beskrives i slutningen af dette kapitlet.

2.1.1 Opgavegraf

De to tabeller nedenfor viser datasættet for en opgavegraf. Tabel 2.1 angiver for hver opgave dens type - 'ttype'- samt deadline. Typen benyttes til at slå beregningstid og energiforbrug op i en tabel der angiver dette for hver processor i datasættet. Deadlinen er den senest tilladelige tid den pågældende opgave må være færdig.

Opgave	Type	Deadline [sek]
0	ttype: 1	inf
1	ttype: 2	inf
2	ttype: 2	inf
3	ttype: 1	inf
4	ttype: 4	inf
5	ttype: 1	inf
6	ttype: 1	$3.5 * 10^{-5}$
7	ttype: 2	$3.5 * 10^{-5}$

Tabel 2.1: Tabel over beregningsopgaver i opgavegrafen.

Afhængighederne imellem beregningsopgaverne er beskrevet med kanterne i opgavegrafen. Til hver kant er der som ved beregningsopgaverne tilknyttet en type 'ktype'. Denne benyttes til opslag af datamængden der skal overføres imellem to beregningsopgaver. I det efterfølgende benyttes notationen $ktype(i, j)$ for opslaget af ktypen af kommunikation imellem opgave i og j.

Kommunikation k fra opgave i til j	ktype
$i \rightarrow j$	
$0 \rightarrow 1$	ktype: 6
$0 \rightarrow 2$	ktype: 2
$1 \rightarrow 3$	ktype: 3
$2 \rightarrow 4$	ktype: 3
$2 \rightarrow 5$	ktype: 2
$3 \rightarrow 6$	ktype: 2
$4 \rightarrow 7$	ktype: 1

Tabel 2.2: Tabel over kommunikationsopgaver i opgavegrafen.

2.2 Processorerne

Der haves som tidligere nævnt tre processortyper; GPP'er, ASIC'er og FPGA'er. Til hver processor i datasættet er der tilknyttet en tabel dels med data for processorens fysiske spcefikationer, tabel 2.3, dels med data for alle opgavetypers afvikling på processoren, tabel 2.4. Nedenfor ses et udsnit af disse.

pris	Statisk energi forbrug [milli Joules]	frekvens [kHz]	Energiforbrug ved kommunikation [milli Joules]	Hukommelse i buffer [bytes]
	$StPwr\_PE\_p$	$freq_p$	$KomPwr\_PE_p$.	
100	800	25000	2000	13

Tabel 2.3: Tabel over processorernes generelle specifikationer.

Type	Antal cykler	Dynamisk energiforbrug [mili Jules]	Kan udføres på aktuelle processor
t	$ExeCyc_{p,t}$	$DynPwr_{p,t}$	
0	2882	25000	1
1	200	25000	1
2	100	25000	1
3	981	25000	1
4	395	25000	1
5	2952	25000	1
6	2000	25000	1
7	160	25000	1
8	115	25000	1
9	552	25000	1
10	15200	25000	1
11	15596	25000	1
12	2882	25000	1
13	3617	25000	1
14	23628	25000	1
15	85864	25000	1
16	3460	25000	1

Tabel 2.4: Tabel over de enkelte opgavetypernes resourcer forbrug på en processor.

Den øverste linje i tabel 2.3 angiver de data der beskriver processorens fysiske specifikationer og som derfor er uafhængige af hvilken opgave der kører på processoren. Priserne er angivet på en

normeret skala i forhold til den første processor i data sættet hvis pris er sat til 100. Det 'Dynamisk energiforbrug' i tabel 2.4 er energiforbruget for udførslen en opgaven af den pågældende type. Den sidste kolonne i tabellen angiver om opgaven kan eksekveres på den pågældende processor. Her nedenfor er vist hvordan de enkelte opgavers energiforbrug og kørsels tid findes.

Energiforbrug: Energiforbruget for en opgave tildelt til en processor er givet direkte ved opslag i tabellen under det dynamiske energiforbrug for opgaven. Det dynamiske energiforbrug angives ved $DynPwr_{p,t(o)}$ hvor p angiver processoren og $t(o)$ typen af en opgave o.
Under kommunikation har processorerne et lavere energiforbrug end under beregninger, dette energiforbrug er det samme uanset længden eller størrelsen af kommunikationen og står derfor under processorens generelle specifikationer. Energiforbruget under kommunikation angives ved $KomPwr\_PE_p$.

Det samlede energiforbrug for en beregningsopgave o mappet til p, $Pwr\_Opg_{p,o}$ bliver således:

$$\begin{aligned} Pwr\_Opg_{p,o} &= DynPwr_{p,t(o)} + KomPwr\_PE_p * num\_Comm \\ \text{hvor: } num\_Comm &= \text{antallet af kommuniktions opgaver,} \\ &\quad \text{hvor } p \text{ sender eller modtager data til opgaven } o \end{aligned}$$

Kørselstid Tabellen herover angiver som sagt data for kørslen af for skellige opgavetyper på processoren. Anden kolonne 'Antal cykler' angiver antal af clock cycler, en opgave tager på den pågældende processor. Eksekveringstiden for en opgave o på processoren p findes ved at dividere dets antallet af clock cykler med processorens frekvens.

$$Ekse(p,o) = \frac{ExeCyc_{t(o)}}{freq_p}$$

2.3 Busser

Busserne har som processorerne en række data tilknyttet der beskriver dem. Prisen, statisk energiforbrug samt frekvens er variable der dækker det samme som for processorerne.

pris	Statisk energi forbrug	frekvens	max antal brugere	Pakke størrelse	Energiforbrug ved kommunikation
	[milli Joules]	[kHz]		[bytes]	[milli Joules]
	$StPwr\_Bus_b$	$freq_b$		$PckSize_b$	$DynPwr\_Bus_b$
65	775	66000	7	8	4881.648

Tabel 2.5: Tabel med specifikationer for en bus

Datamængden der skal overføres i en kommunikationsopgave findes i en tabel hvor 'ktype' benyttes til opslag.

ktype	Datamængde
0	16
1	330
2	320
3	36
4	⋮

Tabel 2.6: Tabel over datamængderne i kommunikationsopgaverne.

For alle busser er angivet et maximalt antal processorer og broer der kan være tilknyttet bussen. Dette er angivet i kolonnen 'max antal bruger'. For busserne gælder det som for processorerne at de har et bidrag til det samlede systems energiforbrug.

Kommunikationstid Hver bus har en fast pakkestørrelse der angiver hvor meget data den kan overføre per clock cykel. Kommunikationstiden $Komm\_tid(ktype(i,j))$ for kommunikationsopgaven $i \rightarrow j$ med kommunikation på bussen b bliver således:

$$\begin{aligned} Komm\_Cycler_{b,i\rightarrow j} &= \frac{Datamængde_{ktype(i,j)}}{PckSize_b} \\ Komm\_tid(i) &= \frac{Komm\_Cycler_{b,i\rightarrow j}}{freq_b} \end{aligned}$$

Energiforbrug Det dynamiske energiforbrug er givet pr. pakke overførsel, energiforbruge for dataoverførslen under en kommunikationsopgave $i \rightarrow j$ på bussen b bliver da:

$$Pwr\_Bus_{p,i\rightarrow j} = DynPwr\_Bus_b * Komm\_Cycler_{b,i\rightarrow j}$$

2.4 De nye hardware komponenter

De nye hardware komponenter er af gode grunde ikke med i datasættet. I det følgende vil det blive beskrevet hvordan de er indført i modellen.

2.4.1 Broer

Broerne antages at optage en pin på hver bus de er forbundet med. Broerne antages at kunne overføre data med samme hastighed som processorerne. De har i den nuværende model ingen pris.

2.4.2 Bufferne

Bufferne er allerede beskrevet i datasættet for processorerne, der er dog ingen information om hvorvidt det er en input/output eller delt buffer. Det antages derfor indtil videre at alle bufferne er delte buffere. Det antages endvidere at bufferne i modellen kan overføre data med samme hastighed som den bus den er forbundet med, samt at dataoverførslen fra processoren til bufferen kan ske løbende så dataen er til rådighed på bufferen i det øjeblik processoren er færdig med beregningerne.

2.5 Beregning af energiforbruget

Energiforbruget for en arkitektur findes ud fra et mappingen af opgaver samt hardware komponenternes specifikationer. Energiforbruget har fem kilder; processorernes statiske energiforbrug, processorernes beregninger, processorernes kommunikation, bussernes statiske energiforbrug samt bussernes energiforbrug under kommunikation. Det samlede energiforbrug for en arkitektur, ark, er vist her udenr.

Statisk energiforbrug
$E\_PE_{st} = \sum_{p \in ark} StPwr\_PE_p$

Energiforbrug ved beregnings opgaver
$E\_PE_{dyn} = \sum_{p \in ark} Pwr\_Opg_{p,o}$, hvor opgave o er mappet til p

Statisk energiforbrug for busser
$E\_Bus_{st} = \sum_{b \in ark} StPwr\_Bus_b$

Energiforbrug for kommunikation på bussen
$E\_Bus_{dyn} = \sum_{b \in ark, i, j} Pwr\_Bus_{p, i \rightarrow j}$, hvor $i \rightarrow j$ er mappet til b

Det totale energiforbrug bliver således:

$$E\_tot = E\_PE_{st} + E\_PE_{dyn} + E\_Bus_{st} + E\_Bus_{dyn}$$

2.6 Casestudiet

Projektet indeholder et casestudie omhandlende det indlejrede system i en smartphone. Telefonen indeholdender fem applikationer; gsm-encoder, gsm-decoder, jpeg-encoder, jpeg-decoder samt mp3-decoder og har således fem opgavegrafter i sin hyperperiode. Datasættet er indsamlet i forbindelse med et phd projekt af M. T. Schmitz [3] hvor der blev fokuseret på at mindske energiforbruget. Det

har ikke været muligt at finde andre data sæt fra virkelige datasæt. Der findes dog programmet TGFF - 'Task Graphs For Free' - [5] til at generere kunstige data.

Kapitel 3

Den genetiske algorithme

I dette afsnit vil den genetiske algorithme blive beskrevet. Først vil der været en kort formalisering af betegnelser der vil blive brugt i resten af rapporten, dernæst vil der være en introduktion til PISA-frameworket [8] - 'A Platform and Programming Language Independent Interface for Search Algorithms' - der er benyttet.

3.1 Princippet for genetiske algorithmer

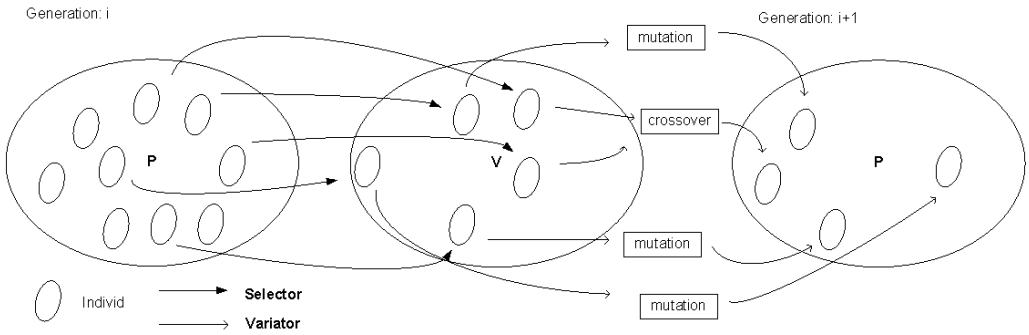
Genetiske algorithmer er en optimeringsheuristik inspireret af den naturlige selektion i naturen. Ved at repræsentere forskellige løsninger til sit problem som individer forsøger man at finde bedre løsninger/individer ved over flere generationer at lade de bedste udvikle sig. Udvælgelsen af individer til reproduktion - selektion - sker på baggrund af en af en 'fitness' værdi, der typisk er den eller de objektkriterier der optimeres efter. Udover valget af gode individer til videre optimering kan selektionen også indeholde valg af dårlige løsninger der fjernes fra populationen.
Optimeringsproblemet er, for at blive i analogien, repræsenteret som en genetiske streng i individet. Individer med forskellige genetiske strenge repræsenterer således forskellige løsninger til problemet. For at udforske løsningsrummet er det nødvendigt at skabe variation i det genetiske materiale. Til at gøre dette benyttes mutation og crossover til at skabe ny individer og dermed nye løsninger. I crossoveren parres to individer med en høj 'fitness' værdi. Det nye individ, der kommer ud af parringen, indeholder stumper af begge sine to forældres genetiske strenge. Løsningen som det nye

individ repræsenterer er således et forsøg på ud fra to gode løsninger at at lave en ny og bedre. For at algorithmen ikke skal blive fanget i lokale minima benyttes mutationsoperatoren. Mutationsoperatoren laver tilfældige ændringer i den genetiske streng, hvilket skaber løsninger med nye karakteristika. Der opereres typisk med to mængder af individer i genetiske algorithmer:

P**:** Population af individer. Dette er mængden af de mest lovende individer der arbejdes videre på.

V**:** Mængde af individer der i en given iteration er er udtaget til mutation og reproduktion på baggrund af en god 'fitness'- værdi.

Den gentagne mutation og crossover imellem generationerne er illustreret i fig 3.1.



Figur 3.1: Mutation og crossover imellem generationerne.

Det skal bemærkes at mutationer kan udføres uden forudgående crossover i den genetiske algorithme i dette projekt. Hvis crossoveren vælges helt fra ved kørsel, falder algorithmen ind under kategorien randomiserede søgealgorithmer. Herunder er princippet i de genetiske algorithmer illustreret i pseudokode.

Algorithm 1: Princippet i en genetisk algorithme

1: Initialiser P ved
2: **repeat**
3: udvælge individer til crossover V fra P
4: lav crossover på individerne V udvalgt til crossover
5: lav mutation på de individer i V udvalgt til mutation
6: beregn fitness værdien af individerne
7: tilføj de nye individer til P
8: **until** stop kriterie mødt

3.2 PISA-frameworket

PISA er et framework der adskiller et multiobjektivt optimeringsproblem i to dele; en problemspecifik del der indeholder mutationsoperatorer og beregning af objektværdier, samt en problemuafhænging del der foretager udvælgelsen af nye individer på baggrund at objektværdierne, samt eventuel oprydning i populationen. Den problemspecifikke del kaldes for en 'variator' - eftersom det er denne der skaber variation i populationen, mens en udvælgelsesstrategi kaldes en 'selector' - den vælger individerne.

Der er flere gode grundet til at bruge PISA-frameworket. For det første giver PISA-frameworket direkte adgange til en række kendte udvælgelsesstrategier. For det andet giver opsplitningen af den genetiske algorithme i 'selector/variator'-delene en tidsbesparelse i udviklingstiden. Til sidst er der med frameworket værktøjer til test og sammenligning af forskellige multiobjektive optimeringskørsler.

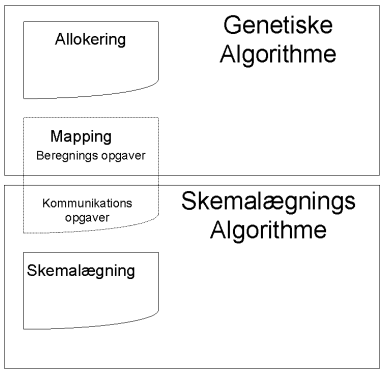
3.3 Co-synthesis problemet overført til den genetiske algorithme

Som tidligere nævnt består co-systhesis af de tre delproblemer; allokering, 'mapping' og skemalægning. Det lå i projektets problemformulering at en genetiske algorithme skulle afprøves. Spørgsmålet var om den genetiske algorithme skulle løse alle tre problemer eller om der skulle laves en adskillelse imellem de tre delproblemer.

Da skemalægningsproblemet er et NP-Complete problem synes det at være bedst at adskille dette problem fra de to øvrige og benytte en løsningsstrategi der er kendt for at virke. 'List scheduling' algorithmerne er idelle til at løse skemalægningsproblemet som det optræder i denne sammenhæng af to årsager. For det første medfører hver ændring der laves i arkitekturen eller mappingen at der skal løses et nyt skemalægningsproblem, altså skal der løses rigtigt mange skemalægningsproblemer. For det andet er det ikke interessant at optimere skemalægningsproblemet, men blot at finde en løsning der overholder tidsbegrænsningerne. Derfor er det ikke nødvendigt med beregningstunge metoder der leder efter gode løsninger. Der ønskes blot en mulig løsning. En 'list scheduling' algoritme er derfor et godt valg på grund af sin hurtige kørselstid.

Mappingproblemet er ikke lige så vel udforsket som skemalægningsproblemet og der er ikke nogen "state of the art"metode der er oplagt til løsningen af dette. Det er derfor valgt ikke at lave en selvstændig metode til mappingproblemet. Mappingproblemet er i stedet opsplittet imellem skemalægningsalgorithmen og den genetiske algorithme, sådan at tildeling af opgaver til processorer foretages af den genetiske algorithme, og tildeling af kommunikationsopgaver til busser og buffere

foretages af skemalægningsalgorithmen. Denne opdeling er lavet for at bevare en vis fleksibilitet i skemalægningsalgorithmen samtidig med at den ikke får hele mappingproblemet at løse også. Ved at overlade kommunikationsopgavernes mapping til skemalægningsproblemet er det desuden muligt at vælge kommunikationsvejene mere intelligent. Dette vil der blive redegjort nærmere for i beskrivelsen af skemalægningsalgorithmen.



Figur 3.2: Opdeling af co-synthesis del problemerne

3.3.1 Co-synthesis problemet løst med PISA-framworket

Som nævnt ovenfor er der holdt en adskillelse imellem udvægelsesstrategierne og modelleringen af problemstillingen i PISA-frameworket. Dette har gjort det muligt at fokusere på den genetiske algorithme og skemalægningsalgorithmen uden i første omgang at bekymre sig om udvælgelsesstrategierne.
I nedenstående pseudokode er main loopet for den samlede algorithme for hele co-synthesis problemet vist. Koden viser den overordnede opdeling i mellem 'variator', skemalægningsalgorithme og 'selector'. Variablene *ind* og *ind*′ er individer hhv før og efter de er blevet ændret ved mutation eller crossover. Individet vil blive beskrevet nærmere i næste kapitel.

Algorithm 2: Genetiske Algorithme

```
1: indlæs_data()  /* Indlæs data for opgavegrafer,processor og busser */
2: opbyg_hyper_periode()  /* Opbyg hyperperioden */
3: P = rand_ini()  /* Konstruer randomiset initial population */
4: V = selector(P)  /* Selector laver første udvælgelse V */
5: repeat
6:    for alle ind i V do
7:       ind′ = variate(ind)  /* muter ind eller lav crossover med andet individ */
8:       P := {P, ind′}  /* tilføj individet til populationen */
9:       skemalæg(ind′)  /* løs skemalægnings problemet for det nu ændrede individ ind′ */
10:      calc_obj(ind′)  /* beregn de nye objectkriterier */
11:   end for
12:   V = selector(P)  /* Selector laver udvælgelse */
13: until stopkriterie mødt
```

Det skal her bemærkes at er der løses et nyt skemalægningsproblem for hvert kald af en crossover eller mutationsoperator. For en samlet kørsel på vil der således blive løst $|V| * |$antal generationer$|$ skemalægningsproblemer. Løsningen af skemalægningsproblemet er derfor den helt dominerende faktor for kørselstiden af den samlede algorithme.

Kapitel 4

Individets repræsentation af indlejrede systemarkitekturer

Som nævnt i det foregående kapitel repræsenterer individet en løsning på optimeringsproblemet, i dette tilfælde en arkitektur med mapping af opgaver til hardware komponenter. Udover at repræsentere arkitektur og mapping er der i individet en liste af prioritetsmål, der benyttes i skemalægningsalgorithmen, samt objektkriterierne.

4.1 Objektivværdier

Udover de to objektværdier pris og energiforbrug der ønskes optimeret, er deadlineoverskridelser medtaget som et objektkriterie i algorithmen. Ved at tillade løsninger med deadlineoverskridelser i populationen kan man bevare en større variation i populationen. Desuden gør dette det muligt at bruge algorithmen til at lede efter lovlige løsninger ud fra en startløsning med deadlineoverskridelser.
Individet indeholder udover objektkriterier for den samlede løsning to vektorer der indeholder energiforbrug og deadlineoverskridelser for de enkelte processorer. Disse benyttes i et par af mutationsoperatorerne. Mere om det senere.

double **Pris :** Den samlede pris for alle hardwarekomponenter i arkitekturen.

double **Energi_forbrug :** Det samlede energiforbrug.

double **Summerede_Deadline_Overskridelser :** Summen af alle deadlineoverskridelser i hyperperioden.

vector < *double* > **Deadline_Overskridelser_PE [p]:** Summen af alle deadlineoverskridelser for de enkelte processorer.

vector < *double* > **Energi_forbrug_PE [p]:** Energiforbrug på de enkelte processor.

4.2 Repræsentation af arkitekturen

Arkitekturen er beskrevet med en liste indeholdende processorerne i arkitekturen, samt en liste med busserne.

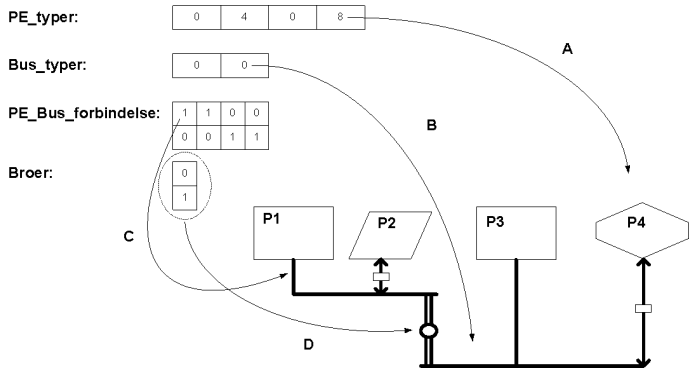
Hvilken processor der er forbundet med hvilken bus gemmes i en matrix hvor en række angiver en bus og en søjle en processor. Broernes forbindelse med busserne er ligeledes beskrevet med en matrix.

vector < *double* >**PE_typer [p]:** Typen af processorerne i arkitekturen.

vector < *double* >**Bus_typer [b]:** Typen af busserne i arkitekturen.

vector < *vector* < *double* >>**PE_Bus_Forbindelse [p][b]** En todimensionel matrix beskrivende forbindelserne imellen processorer og busser; hvis **PE_Bus_Forbindelse**$(i, j) = 1$ angiver det at processor i er tilsluttet bus j. Er **PE_Bus_Forbindelse**$(i, j) = 0$ er det ikke tilfældet.

vector < *vector* < *double* >>**broer[b][1,2]** Matrix af størrelsen $2 \times (AntalBroer)$. En given bro j er beskrevet med en række i matrisen, hvor værdierne **broer**$(j, 1) = bus_1$ og **broer**$(j, 2) = bus_2$ indikerer at broen forbinder bus_1 og bus_2.

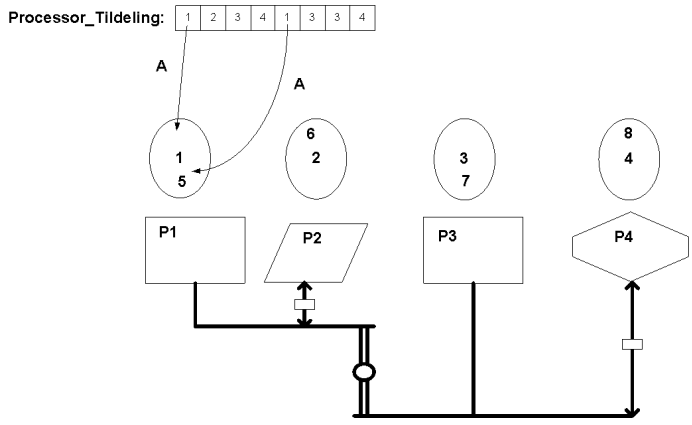


Figur 4.1: Arkitekturens repræsentation i individet. **A**: processor **P4** er af typen 6 hvilket er en ASIC. **B**: Bus **B2** er af type 0. **C**: **PE_Bus_Forbindelse**$(1, 2) = 1$ svarende til at processor **P1** er forbundet til bus **B2**. **D**: En enkelt bro der forbinder bus **B1** og **B2**.

4.3 Repræsentation af mapping

Alle opgavers tildeling til processorer er beskrevet i hyperperiodens mapping, der ligger i vektoren **Processor_Tildeling**.

vector $<$ *int* $>$ **Processor_Tildeling [o]:** Vektor af integers hvori indekset angiver nummeret på en opgave og værdien processoren der udfører opgaven.



Figur 4.2: Mapping repræsenteret i individer. **A**: Opgave 1 og 5 tildelt processor **P1**.

4.4 Prioritetsmål for opgaverne

Den valgte type af skemalægningsalgorithme - 'list scheduling algorithme' - opererer med en liste af prioriterede opgaver. Listen af prioriterede opgaver opbygges efter opgavernes prioritetsmål, sorteret så opgaver med høj prioritetsmål står forrest i listen. Prioritetsmålene haves i listen **prioritets_mål**.

vektor < *double* > **prioritets_mål [o]:** Prioritetsmål bestemmende opgavernes prioritet i skemalægningsalgorithmen.

Kapitel 5

Crossover og mutation

De genetiske operatorer mutation og crossover vil blive beskrevet i dette kapitel. Kaldet af operatorerne styres af variatoren der ud over at indføre variation i populationen sørger for en vis tilfældighed i hvilken af operatorerne der kaldes. Der haves i alt syv mutationsoperatorer og en crossover. Af mutationsoperatorerne ændrer de fire på hardwarekomponenterne, to ændrer i mappingen og en enkelt laver mutation på prioritetsmålene. Crossoveren flytter en processor samt de opgaver der er tildelt denne fra et individ over i et andet.

5.1 Mutation på arkitekturen

De fire operatorer, der opererer på arkitekturen udfører følgende operationer; tilføjelse af bus, ændring af en processors type, fjernelse af en processor samt tilføjelse af en processor.

5.1.1 Mutation: Tilføje bus

Denne operator tilføjer en enkelt bus til arkitekturen med en enkelt bro og flytter to processorer over på den nye bus. Operatoren vil kun blive kaldt hvis der er mindst fire processorer, da der ellers ikke vil være nogen gevinst ved at tilføje en bus.

5.1.2 Mutation: Ændre processortype

Denne operator vælger tilfældigt en processor i arkitetkturen der skal have sin type ændret, dernæst vælges tilfældigt blandt samtlige processortyper, den type som processoren skal skiftes til. Før processortypen skiftes endeligt, skikres det at samtlige opgaver kan udføres på den nye processor. Er der blot én opgave der ikke kan udføres på den nye processortype, laves skiftet ikke.

5.1.3 Mutation: Fjerne processor

Denne operator vælger tilfældigt en processor der skal fjernes fra arkitekturen. Før processoren fjernes, kontrolleres det at samtlige opgaver stadig kan udføres på en processor i arkitekturen. Ved fjernelse af en processor skal de opgaver der tidligere blev udført på denne tildeles andre processorer i arkitekturen. For at sikre en balanceret vægtning af opgaver på processorerne fordeles opgaverne ud på de tilbageværende processorer.

5.1.3.1 Fjerne bus

Dette er ikke en mutationsoperator på linje med de andre, men en metode til at rydde op i busser der er blevet overflødige efter at den sidste processor er blevet fjernet fra arkitekturen. Kun busser uden processorer tilknyttet og kun én bro tilsluttet kan ved sikkerhed siges at være overflødig i designet da den ellers kunne have en funktion under kommunikation. Disse fjernes således, hvis de er opstået efter fjernelsen af en processor.

5.1.4 Mutation: Tilføje processor

Denne operator tilføjer en processor af en tilfældig type til en tilfældig bus i systemet. Processortildelingen ændres for $\frac{|opgaver|}{|processoer|}$ opgaver til den nye processor, hvor $|opgaver|$ er antallet af opgaver i hyperperioden og $|processoer|$ er antallet af processorer efter tilføjelsen. Mappingen ændres for at holde en fornuftig vægtning af opgaver på processorerne.

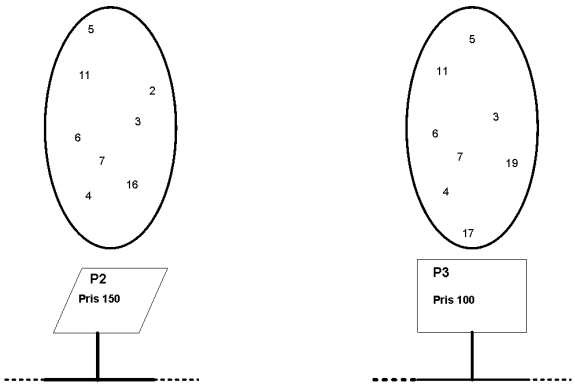
5.2 Crossover

Crossover operatoren er designet med henblik på at skabe en forbedring i det individ der kommer ud af operationen samtidig med at så stor en del som muligt af arvemassen fra de to individer der indgår bevares. For at foretage crossover operationer, der bevarer mest muligt af arvemassen,

foretages operationen på individer der har dele af arvemassen, der ligner hinanden mest muligt, samtidig med at der er en forbedring inden for et af objektiverne; pris, total deadlineoverskridelse, eller energiforbrug. Til at vurdere graden af individernes lighed i arvemasse indføres et 'lighedsmål' $L(p_1, ind', p_2, ind^*)$ imellem to processorer p_1 og p_2 fra to forskellige individer ind' og ind^*. Lighedsmålet er da antallet af opgaver der i ind' er tildelt p_1 og i ind^* er tildelt p_2.

$$L(p_1, ind', p_2, ind^*) = \sum_{o=0}^{o=|opgaver|} PE_{o,ind'} = p_1 \vee PE_{o,ind^*} = p_2 \tag{5.1}$$

Hvor $PE_{o,ind'}$ er processoren der udfører opgave o i ind'.



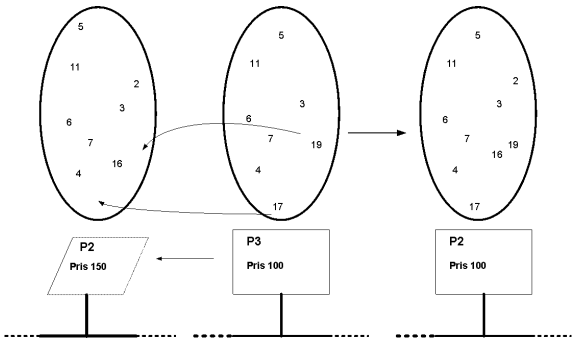
Figur 5.1: Lighedsmålet findes for to processorer. Mængden af opgave der går igen i tildelingen til de to processorer er {3,4,5,6,7,11}. Sættets størrelse udgør lighedsmålet imellem de to processorer. Lighedsmålet er således på 6.

For at bevare det størst mulig lighedsmål søges hele mutationspuljen V igennem for at finde det individ der giver det største lighedsmål.

Hvis der bliver fundet et individ til crossover, foretages crossoveroperationen på individet ind' ved at ændre processortypen af p_1 til processor typen af p_2 samt kopiere den del af opgavetildelingen af p_2, der ikke allerede findes i p_1, til denne. Crossoveroperatoren er således en envejsoperator hvor en bedre processor p_2 fra individet ind^* bliver kopieret over i individet ind' med mapping og prioritering af opgaver.

Algorithm 3: Bestemmelse af individet ind^* til crossover med ind'

```
1: største_ligheds_mål = 0
2: individ_2 = Ø
3: for alle individer i ∈ Q do
4:    for alle processorer p2 i i do
5:       for alle processorer p1 i ind' do
6:          if L(p1, ind', p2, ind*) ≥største_ligheds_mål og forbedring i det valgte objektiv then
7:             største_ligheds_mål = L(p1, ind', p2, i)
8:             individ_2 = i
9:          end if
10:      end for
11:   end for
12: end for
```



Figur 5.2: Crossover imellem **P2** og **P3**. **P2** ændres til typen af **P3** og tildelingen af opgaver på **P2** ændres så denne også har opgaverne fra **P3**'s tildeling.

5.3 Mutation: Ændre mapping

Der haves to mutationsoperatorer på mappingen; en randomiseret og en intelligent. Den randomiserede metode benyttes til at lave variation i startpopulationen og som backup, hvis den intelligente fejler. Den intelligente mutationsoperator søger at lave en vægtning af opgaverne ved at betragte på hvilke processorer der er deadlineoverskridelser.

5.3.1 Intelligent mutationsoperator på mapping

Den intelligente mutationsoperator tager de opgaver der har den største deadlineoverskridelse og flytter dem over på processorer uden deadlineoverskridelse. Hvis dette ikke er muligt, kaldes en randomiseret mutation på individets mapping.

Algorithm 4: intelligent_mapping_mutation(ind')

1) finder opgavegrafer med deadline overskridelser fra individet ind og sorterer dem efter størrelsen af overskridelsen med de største først
2) sætter antallet af opgave at mutere $|Num_{mutate}|$ til maximun af fire og antallet af opgavegrafer med deadlineoverskridelse
3) finder de processorer hvorpå der ikke optræder deadlineoverskridelser
4) laver ny processor-tildeling på $|Num_{mutate}|$ tilfældige valgte opgaver fra opgavegraferne med deadlineoverskridelser.
   Hvor tildelingen ændres til en af processorerne uden deadlineoverskridelser.

Som det ses af ovenstående pseudokode er der en øvre grænse for antallet af opgaver, der vil blive muteret på fire. Dette er valgt for ikke lave store ændringer i mappingen, når der kun er få deadlineoverskridelser tilbage. Ved at mutere mange opgaver risikeres blot at flytte deadlineoverskridelserne over på en anden processor.

5.3.2 Randomiseret mutationsoperator på mapping

Den randomiserede mutationsoperator forsøger at flytte alle opgaver i hyperperioden til en ny processor i arkitekturen. Det er en meget voldsom operation der har til hensigt at slippe ud af et lokalt minimun. Operatoren kaldes kun i det tilfælde at alle deadlines er overholdt, i hvilket tilfælde der altså er fundet et lovligt design. Ved at lave en stor mutation vil der fremkomme en helt ny løsning, der åbner for nye områder i løsningsrummet.

5.4 Mutation: Ændring af prioriteringsmål

Der blev experimenteret med forskellige opbygninger af den prioriterede liste til skemalægnings algorithmen. Af de forskellige metoder der blev afprøvet var metoden der beskrives i dette afsnit den der gav de bedste resultater.
Som tidligere nævnt benytter 'list scheduling' algorithmerne en prioriteret liste af opgaverne til at bestemme rækkefølgen af opgavernes skemalægning. Prioriteringen bestemmes af et prioritetsmål der ligger imellem 0 og 100, hvor en opgave med et højt prioritetsmål vil blive skemalagt før en

opgave med lavere. Metoden **muter_prioritets_mål(**ind'**)** ændrer et tilfældigt antal opgaveprioritets mål. Det nye prioritetsmål sættet til et tilfældigt tal imellem 0 og 100.

Algorithm 5: muter_prioritets_mål(ind')

```
1: Antal opgaver at mutere = rand(0,antal opgaver)
2: for i = 0, i < antal opgaver at mutere, i++ do
3:    vælg tilfældigt en opgave o_m at mutere
4:    muteret_prioritets_vægt[o_m] = rand(0,100)
5: end for
```

5.5 Variationsoperatoren

De ovenstående crossover- og mutationsoperatorer kombineres i det endelige program i en fælles variationsoperator. Denne operator sørger dels for, at der er tilfældighed i hvilke mutationsoperatorer der kaldes, dels at de enkelte operatorer kaldes efter brugerens parametervalg. Nedenstående tabel viser de parameterindstillinger en bruger må tage stilling til for at køre algorithmen. Parametrene er sandsynlighederne for at de enkelte mutationsoperatorer kaldes.

Sandsynlighed for kald af mutationsoperatoren	mutationsoperatoren
p_{muter_bus}	Tilføjer bus
$p_{ændre\_PE}$	Ændrer en processors type
$p_{tilføje\_PE}$	Tilføjer en processor
p_{fjern_PE}	Fjerner en processor
$p_{crossover}$	Laver crossover imellem to individer
p_{muter_map}	Kalder den intelligente mapping mutationsoperator, hvis den fejler kaldes den randomiserede
$p_{muteret\_vægtliste}$	Udfører mutation på prioritetsvægtene.

Tabel 5.1: Sandsynlighederne for kald af mutationsoperatorerne. $p_i \in [0,1] \wedge \sum p_i = 1$

Det skal bemærkes at mutationsoperatorerne; fjerne processor, ændre processor og ændre bus kan fejle. Dertil kommer at det ikke er interessant at mutere den prioriterede vægtliste hvis alle deadlines er overholdt. Hvis det sker at den udtrukne mutationsoperator derfor ikke udføres, vil først den intelligente, dernæst den randomiserede mutationsoperator på mappingen blive forsøgt. Den randomiserede mutationsoperator vil kun kunne fejle hvis der der kun haves én processor i arkitekturen, eller hvis alle opgaver er fordelt på ASIC'er eller FPGA'er på en sådan måde, at

ingen opgaver kan udføres af en anden processor i arkitekturen. Hvis dette skulle ske, vil individet slippe igennem uden at blive ændret. Der er dog ikke noget der tyder på, at det udgør et større problem.

Kapitel 6

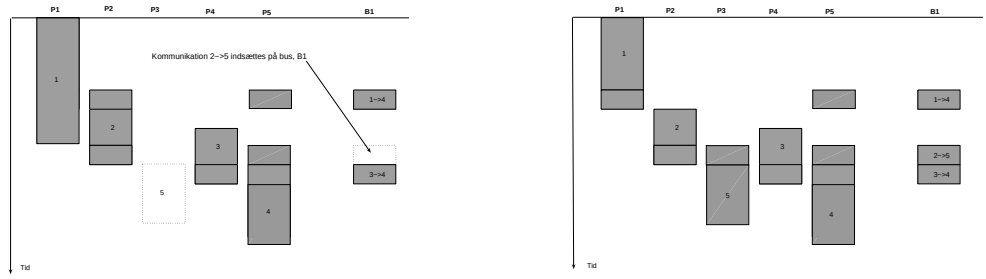
Static List Scheduling Algorithme

Skemalægning er som sagt det sidste skridt i Co-synthesis der er nødvendigt for at evaluere kørselstiden. Der er i valget af en heuristik lagt vægt på hurtige kørselstider for at give mulighed for flere evalueringer i den genetiske algorithme. Den 'list scheduling' algorithme der er brugt i dette projekt tillader desuden en stor grad af fleksibilitet for kommunikationsopgaverne, eftersom skemalægningsalgorithmen også foretager mapping af busser til kommunikationsopgaverne.

6.1 Generelt om skemalægningsalgorithmen

Centralt i skemalægningen er bestemmelsen af starttidspunktet for kommunikations- og beregningsopgaverne. Der er to forhold der bestemmer opgavens starttidspunkt; 1) sluttidspunktet for de umiddelbart foregående opgaver i opgavegrafen , dvs dataafhængighederne, og 2) de ledige resourcer til afviklingen af opgaven på processorer og busser.
Sluttidspunktet for de foregående opgave i opgavegrafen er en simpel opgave at bestemme. I list scheduling algorithmen skemalægges opgaverne i en rækkefølge der følger dataafhængighederne i opgavegrafen. Det vil sig opgaverne skemalægges med kanternes retning fra startknuden og nedefter. Dette betyder at sluttidspunktet for de foregående opgaver vil være til rådighed ved opslag i den del af skemaet der er fastlagt.
De ledige resourcer på hardware komponenterne er det andet forhold der bestemmer en opgaves

starttidspunkt. Man kan her vælge to strategier til at finde de ledige resourcer. Man kan enten søge skemaet igennem for at lede efter ledige huller i skemaet hvor opgaverne kan indsættes, eller man kan skemalægge opgaven til sidst i skemaet hvor man ved at der er resourcer til rådighed. Det er valgt at lave indsættelse for kommunikationsopgaverne i algorithmen, men ikke for beregningsopgaverne. Kommunikationsopgaverne er generelt meget kortere end beregningsopgaverne. Derfor vil busserne være optaget mindre tid af gangen, hvilket vil give flere huller til indsættelse. Indsættelse af beregningsopgaver synes umiddelbart ikke at være lige så fordelagtig som kommunikationsopgaverne. Det var derfor ikke en første prioritet at indføre mulighed for indsættelse af beregningsopgaver i algorithmen. Det ville dog være en interessant ting at undersøge specielt i kombination med 'pre-emption'.



(a) (b)

Figur 6.1: Indsættelse af kommunikationsopgave i skemaet. (a) Ledig plads til indsættelse af komunikation $3 \rightarrow 5$. (b) Indsættelse af kommunikatione udført og opgave 5 fastlægges.

6.1.1 Hovedløkken i skemalægningsalgorithmen

Det grundlæggende princip i 'static list scheduling' algorithmen er at opstille en prioriteret liste for opgaverne som afgør rækkefølgen de skemalægges efter. Herefter skemalægges opgaverne i rækkefølge efter den prioriterede liste. Før en opgave kan skemalægges, er det dog nødvendigt at sikre, at alle foregående opgaver allerede er skemalagt, hvilket præcist svarer til at sikre dataafhængighederne. Til at løse dette problem indføres en klarliste der løbende opdateres. Hver opgave har i denne liste en **sand/falsk** værdi der netop angiver om en opgave har samtlige umiddelbart foregående opgaver skemalagt og dermed er klar til skemalægning.

Algorithm 6: Opdater_Klar_Liste(x)

```
1: for alle opgaver o, der modtager data fra x, o → x do
2:    Klar_Liste[o] = sand
3:    for alle opgaver z med kommunikation til o, z → o do
4:       if z ikke skemalagt then
5:          Klar_Liste[x] = falsk
6:       end if
7:    end for
8: end for
```

Hovedløkken i skemalægning er vist herunder. For hver iteration skemalægges den opgave med højest prioritet der også der har en **sand**-værdi i **Klar_Liste[..]**. Algorithmen gennemgår i hver iteration fire trin:

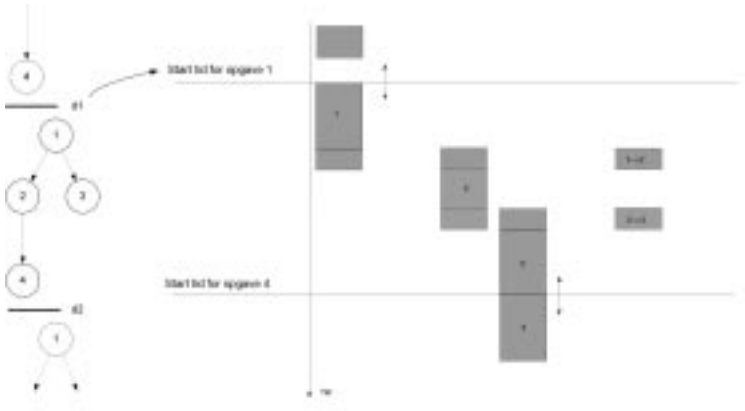
Algorithm 7: Princippet for skemalægningsalgorithmen

```
1: repeat
2:    1) Sæt x lig første klare opgave i den prioriterede liste til skemalægning
3:    2) Find mapping og tidspunktet for eventuelle kommunikationsopgaver i → x
4:    3) Find tidligste starttid for opgave x
5:    4) Skemalæg x og opdater klar listen.
6:       gå til 1)
7: until Alle opgaver skemalagt
```

Algorithmens kompleksitet er $O(n^3)$ hvor n er antallet af opgaver i hyperperioden. Hovedløkken indeholder det første gennemløb af alle opgaver, mens opdateringen af klarlisten i sig selv har kompleksiteten $O(n^2)$.

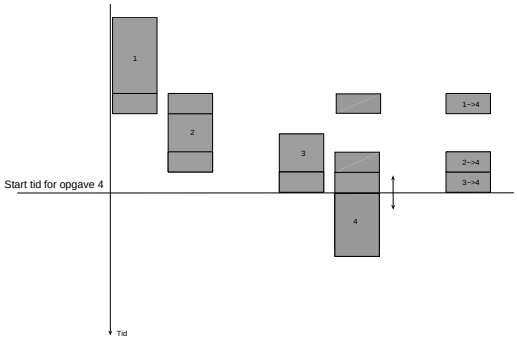
6.2 Skemalægningen af beregningsopgaver

Beregningsopgavernes starttid og dermed deres position i skemaet findes som sagt med en metode uden indsættelse. Dette betyder at en beregningsopgave tidligst kan starte, når det sidste opgave på processoren er færdig. Oven i dette har hver enkelt opgavegraf en tidligste starttid, der sikrer at opgaver fra grafen ikke startes før startidspunktet for opgavegrafens periode. I figuren neden for er vist er udsnit af skemaet, hvor en opgavegraf på fire opgaver skemalægges. Starttiden af opgave 1 er bestemt af periodens starttid og starttiden for opgave 4 er bestemt af den foregående opgave på processorens sluttid.



Figur 6.2: Udsnit af et skema til illustrering af opgavernes starttid. Opgave 1's starttidspunkt er bestemt af starttidspunktet for opgavegrafens periode, **d1**. Opgave 4's starttidspunkt er bestemt af den foregående opgaves stuttidspunkt.

Hvis der er indgående kommunikation til en beregningsopgave skal dette også tages i betragtning. Efter at den tidligest starttid er fundet mht. til opgavegrafens periodestart og sidste opgave på processoren, løbes samtlige kommunikationsopgaver til opgaven igennem. Hvis den sidste indgående kommunikation sker senere vil starttiden blive opdateret.



Figur 6.3: Starttiden for beregningsopgaver 4 bestemmes af afslutningen af sidste indgående kommunikation. I dette tilfælde kommunikation $3 \rightarrow 4$.

Neden for er vist en pseudukode for skemalægningen af opgave x. Den til x tildelte processor p_x findes ved opslag i individets mapping af opgaver.

Algorithm 8: Bestemmelse af starttiden for beregningsopgave x

```
1: Start_Tid[x] = tidligest starttid for x's tilhørende opgavegraf
2: if Start_Tid[x] < sluttiden for sidste opgave på p_x then
3:    Start_Tid[x] = sluttiden for sidste opgave på p_x
4: end if
5: for alle opgaver o, der har kommunikation til x, o → x do
6:    if Start_Tid[x] < sluttid for kommunikation o → x then
7:       Start_Tid[x] = sluttid for kommunikation o → x
8:    end if
9: end for
10: Slut_Tid[x] = Start_Tid[x]+Ekse(x)
```

$Ekse(o)$ er kørselstiden for en beregningsopgave der blev indført kapitel 2.2.

6.3 Mapping og skemalægning af kommunikationsopgaver

Som tidligere nævnt er det valgt at lade skemalægningsalgorithmen foretage mappingen af kommunikationen da det kan gøres mere intelligent her. Hvis en kommunikationsopgave kan overføres over mere end én vej fra afsenderprocessoren til modtagerprocessoren, er det muligt at vælge den vej, der tillader den tidligste start af kommunikationsopgaven. Fastlæggelsen af kommunikationen indebærer de tre følgende skridt:

1) Bestemmelse af de mulige kommunikationsveje.

2) Bestemmelse af kommunikationsvejen med tidligste starttid.

3) Mapping og skemalægning af kommunikationen på busser, buffere og processorer.

I det resterende af dette kapitel vil disse tre skridt af skemalægningsalgorithmen blive beskrevet. Først beskrives metoden til bestemmelsen af kommunikationsvejene.

6.3.1 Bestemmelse af kommunuikationsvejene i arkitekturen

Metoden til bestemmelse af kommunikationsvejene i systemet finder samtlige kommunikationsveje fra interfacet $inter_{(p_1,b_1)}$ til p_2. En kommunikationsvej v er givet ved en liste af busser $v = b_1, b_2, ..., b_i, b_v^*$, hvor b_1 er tilsluttet p_1 med interfacet $inter_{(p_1,b_1)}$. Den sidste bus på vejen benævnes b_v^*. Er b_v^* tilsluttet modtager processoren p_2, er en gyldig vej fundet.
Metoden **find_komm_veje(**$inter_{(p_1,b_1)}$**,**p_2**)** er en bredde først algorithme, der afsøger alle kommunikationsveje ved gradvis at opbygge vejene ud fra interfacet $inter_{(p_1,b)}$. For alle delvist opbyggede veje v søger metoden at finde en ny bus der kan tilføjes efter den sidste bus på vejen b_v^*. Denne fremgangsmåde følges for alle veje indtil modtager bufferen findes eller busserne er søgt igennem det antal gange der svarer til den længst mulige vej i systemet.

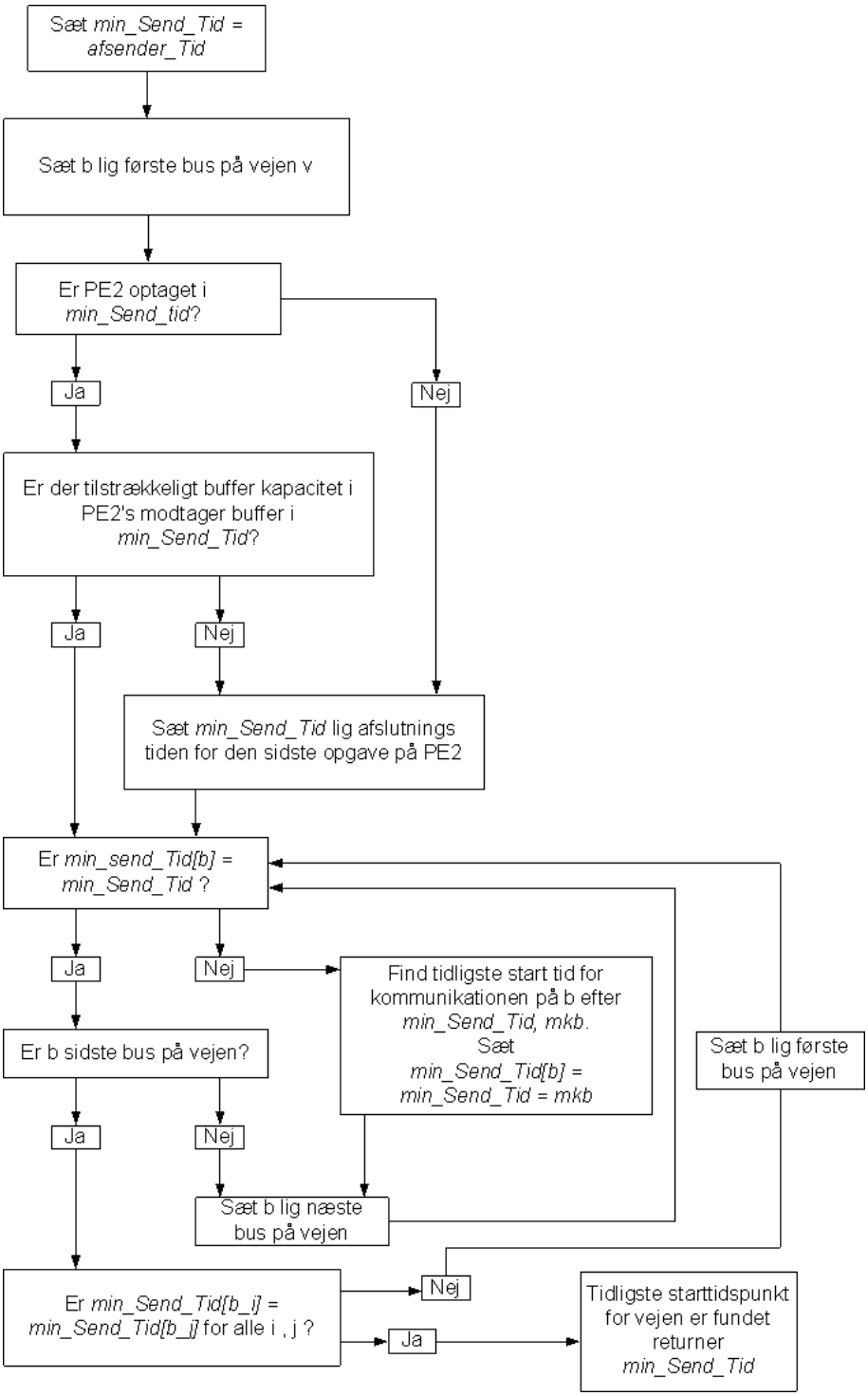
Algorithm 9: **find_komm_veje**($inter_{(p_1,b_1)}$,p_2)

```
1: initialiser en enkelt vej der består af b_1
2: for t = 1 til antal busser do
3:   for alle vejene v hvor b_v^* ikke er tilsluttet p_2 do
4:     bus_tilføjet_på_vej = false
5:     for alle busser b do
6:       if b tilsluttet b_v^* og b ikke tilsluttet p_1 then
7:         if bus_tilføjet_på_vej = false then
8:           tilslut b til aktuelle vej
9:           if hastigheden for b ≤ nuværende hastighed på vejen then
10:            Sæt hastighed på vejen lig hastighed for b
11:          end if
12:        else
13:          gem v som en ny vej og tilføj b
14:          if hastigheden for b ≤ nuværende hastighed på vejen then
15:            Sæt hastighed på vejen lig hastighed for b
16:          end if
17:        end if
18:      end if
19:    end for
20:  end for
21: end for
22: fjern alle veje v der ikke har b_v^* tilsluttet p_2.
```

Metoden finder udover vejene den højeste hastighed kommunikationen kan sendes med over en vej, hvilken er lig hastigheden af den langsomste bus. De veje der ved metodens slutning ikke er forbundet med p_2 fjernes.

6.3.2 Bestemmelse af mindste kommunikationstid imellem processorer

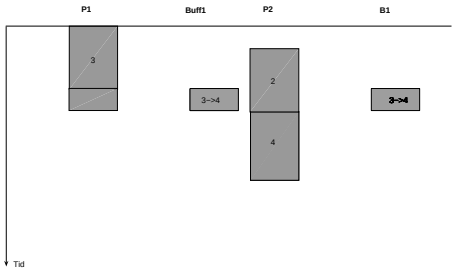
Givet de mulige kommunikationsveje i arkitekturen fundet med metoden **find_komm_veje**(p_1, p_2) er det næste skridt at finde den faktiske kommunikationsvej der tillader den tidligste starttid for kommunikationen.
Metoden der gør dette **min_Send_Tid_over_vej**$(v, o_1, o_2, afsender\_Tid)$ tager som input sluttidspunktet af afsenderopgaven o_1 i kommunikationen, $k : o_1 \rightarrow o_2$. Dette sluttidspunkt udgør det tidligste starttidspunkt for kommunikation og benævnes $afsender\_Tid$. Udover $afsender\_Tid$ tager metoden en mulig vej v som input. Kommunikationsvejen er som tidligere nævnt en liste af de busser kommunikationen løber over.
Under hele kommunikationen skal afsenderprocessor, modtagerprocessor og samtlige busser på vejen være til rådighed for at kommunikationen kan gennemføres. Derfor vil det tidligste tidspunkt kommunikationen kan starte være det tidspunkt efter $afsender\_Tid$, hvor samtlige disse resourcer er klar til kommunikationen.
På næste side er flow chartet for **min_Send_Tid_over_vej**$(v, o_1, o_2, afsender\_Tid)$ vist. Tidligste starttidspunkt for kommunikationen, min_Send_Tid, sættes ved metodens start lig $afsender\_Tid$. Tidligste starttidspunkt for kommunikationen på busserne gemmes i $min\_Send\_Tid[b]$. Funktionen har således fundet det tidligste starttidspunkt for kommunikationen når $min\_Send\_Tid = min\_Send\_Tid[b]$ for alle busser b.



Figur 6.4: Flow chart for **min_Send_Tid_over_vej**($v, o_1, o_2, Send\_Tid$)

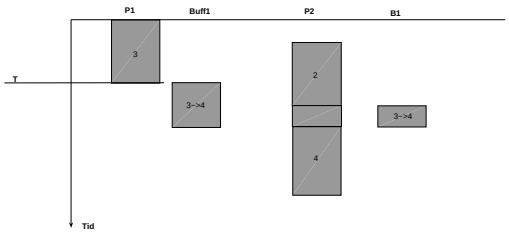
6.3.3 Endelig skemalægning af kommunikationen

Efter at kommunikationsvejene er fundet, samt kommunikationens starttidspunkt og varighed, er der kun tilbage at skemalægge kommunikationen på bufferne, busserne og eventuelt processorerne. I bestemmelsen af *min_Send_tid* på kommunikationsvejene blev der draget nytte af eventuel bufferkapacitet på modtagerprocessorens buffer i det tilfælde at processoren var optaget. En processors buffer benyttes kun til indgående kommunikation hvis processoren er optaget når kommunikationen starter. Da skemalægningsalgorithmen ikke benytter indsættelse vil det ikke give nogen fordel at benytte bufferen før den beregningsproces der er ved at blive skemalagt.



Figur 6.5: Modtagelse af kommunikationen 3→4 foretages af **P2**'s buffer, **Buff1**, mens opgave 2 køer. Dette muliggør tidligere start af opgave 4.

For den udgående kommunikation benyttes processorernes buffere derimod hver gang de har tilstrækkelig plads til dataoverførslen. Ved at lade bufferen foretage den udgående kommunikation er der mulighed for at processoren kan påbegynde den næste opgave tidligere.



Figur 6.6: Processor **P1**'s buffer, **Buff1** foretager kommunikation til bussen. Dette muliggør start af en ny opgave til tiden T på **P1**.

6.3.3.1 Datatyper hørende til kommunikationsopgaverne

Skemaet samt mapping for kommunikationsopgaverne gemmes i to vektorer tilknyttet hver enkelt bus. Den ene vektor indeholder afsenderopgaven for kommunikationen på index i og starttiden for kommunikationen til i+1. Den anden indeholder modtageropgaven for kommunikationen på index i og sluttiden for kommunikationen på index i+1.
Der er i algorithmen behov for at søge i de tidligere fastlagte kommunikationsopgaver for at finde hvornår kommunikationen er fastlagt. Det er derfor fordelagtigt at have skemaet for kommunikationesopgaverne sorteret over tid. Det er dog dyrt at indsætte elementer i C++ vectorer og derfor holdes busskemaet usorteret. I stedet benyttes en linket liste til at gemme rækkefølgen af opgaverne i skemaet; dette gør at indsættelsen sker i en linket liste, mens nye elementer blot tilføjes til slutningen af vektorerne.

Kapitel 7

Evaluering af approximationssæt

I dette afsnit vil der blive givet en kort indføring i de forhold, der adskiller evalueringen af multiobjektive optimeringsalgorithmer fra algorithmer med ét objektiv. Dette følges op med indføring af tre kvalitetsindikatorer der vil bilve benyttet til evalueringen af eksperimenterne i næste kapitel.

7.1 Approximationssæt

I multiobjektiv optimering er der sjældent en optimal løsning på problemet. Problemet vil ofte have en karakter hvor der kan findes en kurve der beskriver tradeoffet mellem objektiverne, denne kurve kaldes pareto fronten. Hver enkelt punkt på pareto fronten udgør en løsning med en kombination af objektiverne, som ingen af de andre løsninger har bedre. Pareto fronten er mere formelt defineret som mængden af løsninger til optimeringsproblemet der ikke er domineret. En løsning x siges at være domineret af løsningen y, hvis y er mindst lige så god som x på alle objektiver og bedre på mindst et. Herunder er relationen defineret for er minimeringsproblem.

$$\begin{aligned} y \text{ dominerer } x \text{ hvis:} \quad & f_i(y) \leq f_i(x) \text{ for alle objektiver og} \\ & f_j(y) < f_j(x) \text{ for mindst et.} \end{aligned}$$

I det følgende vil relationen svag dominans også blive benyttet.

$$y \text{ dominerer svagt } x \text{ hvis:} \qquad f_i(y) \leq f_i(x) \text{ for mindst objektiver}$$

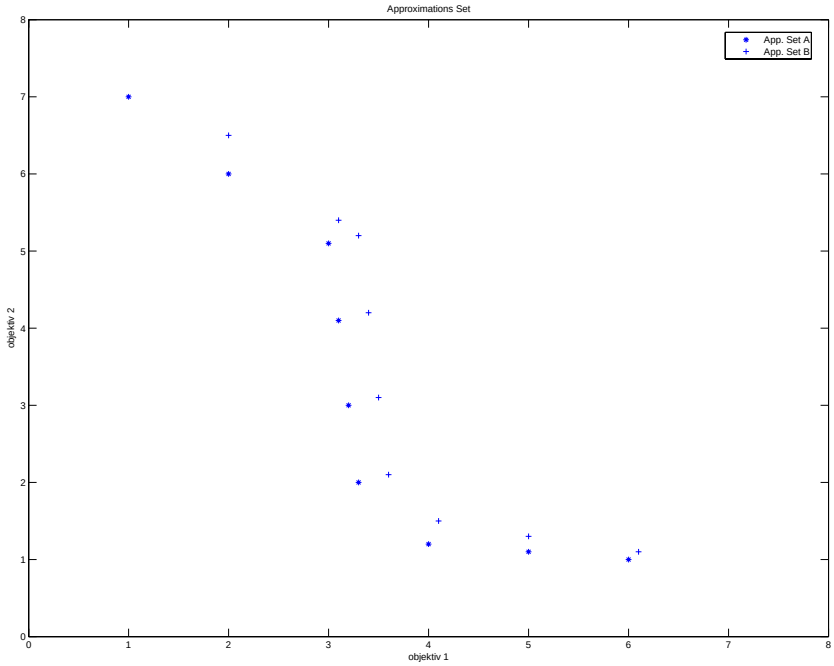
Er det ikke muligt at find pareto fronten for det give problem, må man ty til en tilnærmelse til denne. Dette kaldes approximationssættet. Approximationssættet er en mængde af løsninger der tilnærmer pareto fronten. Approximationssættet findes ofte med en form for heuristik. Sammenligningen af to optimeringskørsler for et multiobjektivt optimeringsproblem indeholder således sammenligningen af approximationssæt i stedet for skalarer. Dette giver nogle problemer som der ikke haves med et objektiv, som det vil ses i det efterfølgende.

7.1.1 Sammenligning af approximationssæt

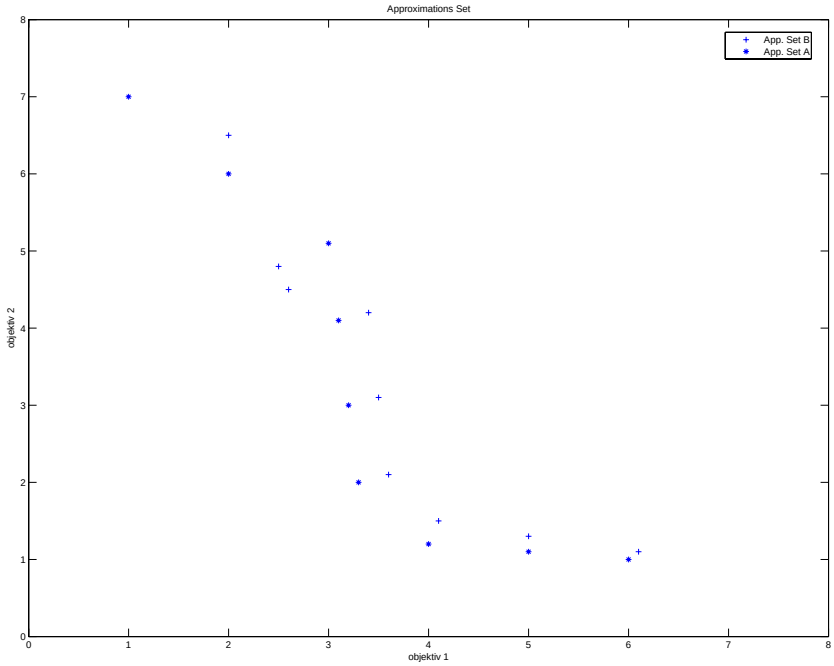
Ved sammeligningn af to approximationssæt A og B defineres der fire relationer imellem sættene.

1) A dominerer B :	hvis der for hver løsning $y \in B$ findes en løsnnig i A der dominerer denne, se fig 7.2.
2) B dominerer A :	hvis der for hver løsning $y \in A$ findes en løsnnig i B der dominerer denne
3) A og B er ikke sammenlignelige :	der findes mindst en løsning i A der svagt dominerer en eller flere løsninger i B og ligeledes den anden vej, se 7.2
4) A og B er indifferent :	A dominerer svagt B og B dominerer svagt A

Disse relationer viser meget godt problemet med sammemligningen af approximationssæt. Det første problem er at relation 3) ikke fortæller noget om hvilket sæt der er det bedste. Populært sagt vil sættene være bedre på forskellige områder, hvilket gør at et sæt ikke kan siges at være bedst. For det andet siger relationerne 1) og 2) ikke noget om i hvor høj grad et sæt er bedre end et andet. Dette er begge problemer som kvalitetskriterierne søger at løse. Desuden giver kvalitetskriterierne mulighed for at vurdere afstanden til pareto fronten hvis denne er kendt.



Figur 7.1: A dominer B



Figur 7.2: A og B ikke sammelignelige

7.1.2 Indføring af kvalitetsindikator

En kvalitetsindikator er en funktion der transformerer et approximationssæt til en skalar - kvalitetsmålet - der angiver kvaliteten af sættet. Kvalitetsmålene er praktiske af to årsager. For det første giver de et mål for afstanden imellem approximationssæt, hvilket de førnævnte relationer ikke kan og for det andet gør de det muligt at benytte statistiske tests. Hvilket for randomiserede søgemetoder er helt nødvendigt for at kunnen tage højde for usikkerheden på løsnings resultaterne. Problemet med kvalietstindikatorerne er til gengæld at kvalitetsmålene ikke er fuldstændigt objektivte. Hver kvalitetsindikator udgør en bestemt vurdering af hvad en god løsning er. Derfor kan to kvalitetsindikatorer godt give modstridende resultater. Nogle af kvlitetsmålene behøver desuden et referancepunkt eller referencesæt, der også kan farve resultatet af kvalitetsmålet. De kvalitetsindikatorer der vil blive benyttet i dette projekt har også dette problem.

7.1.3 Kvalitetsmål

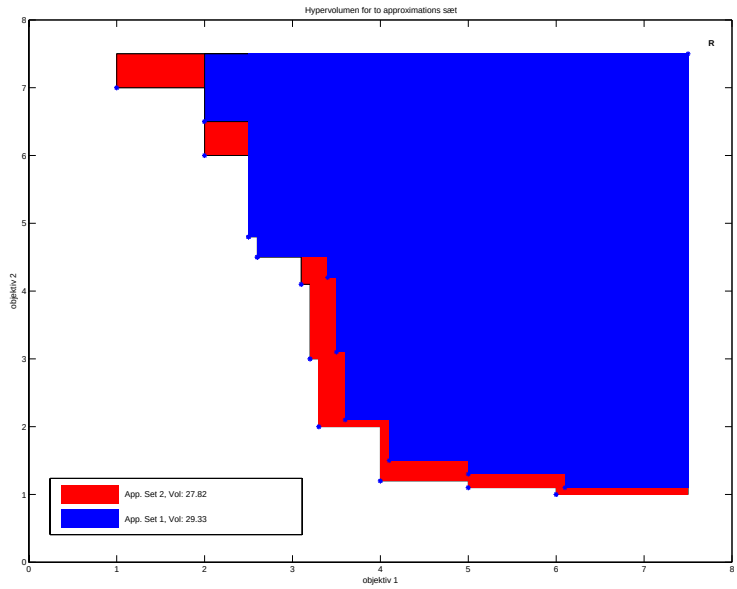
Der vil i dette projekt blive benyttet tre kvalitetsmål: 'dominance ranking', 'hyper volume' og 'unary epsilon indicator'. Kvalitetskriterierne er funktioner der givet et approximationssæt returnerer en skalar der angiver kvaliteten af approximationssættet.

- 'Dominance ranking'
  'Dominance ranking' angiver i hvor høj grad et approximationssæt A er domineret af en anden udvalgt mængde af sæt $\mathbf{B}$. Rangen af et approximationssæt A er givet ved antallet af sæt der dominerer det, ud af mængde af referancesæt $\mathbf{B}$. Jo lavere rang jo bedre er sættet derfor.

  $$DomRank\_Ind(A, \mathbf{B}) = 1 + |\{B_j \in \mathbf{B} \ : \ B_j \text{ dominerer } A\}| \tag{7.1}$$

  'Dominance ranking' vurderer således et approximationssæt op imod en mængde af approximationssæt. Indikatoren er derfor mest velegnet til vurdering af to sæt af kørsels resultater opimod hinanden. Til statistiske tests med indikatoren anbefales det i [11] at benytte et Mann-Withneq test.

- 'Hyper volume'
  Kvalitetsmålet 'hyper volume' angiver volumen af det rum som et givet approximationssæt svagt dominerer. Det er nødvendigt med et grænsepunkt G, der afgrænser området volumen skal beregnes for. Det kan være en startløsning eller punktet med de koordinater der udgør de dårligste objektivværdier fundet for alle approximationssæt i A. På figuren nedenfor er 'hyper volume' vist for to ikke sammenlignelige approximationssæt i forhold til grænsepunktet G.



Figur 7.3: Hypervolumen for to ikke sammenlignelige approximationssæt

'Hyper volume' er i PISA-frameworket transformeret så lave værdier af 'hyper volumen' indikerer bedre approximationssæt sammenlignet med større 'hyper volumen'. Til transformationen findes volumen af et referencepunkt R der kunne være; den optimale løsning, (0,0) eller ethvert punkt der altid vil give en større volumen end de approximationssæt der findes. 'Hyper volumen' defineres nu som volumen af det rum referancepunktet dækker, minus volumen af det rum approximationssættet dækker.

$$HypVol\_ind(A, R) = Vol(R) - Vol(A) \tag{7.2}$$

'Hyper volumen' har en fordel frem for dominance ranking, idet den ikke blot påviser om et approximationssæt dominerer det sæt det sammenlignes med, men også vil give forskellige kvalitetsmål for usammenlignelige approximationssæt. Kendes paretofronten, kan denne benyttes i stedet for referencepunktet. Indikatoren kan således benyttes til at vurdere afstanden til pareto fronten.

- 'Unary additiv epsilon indicator' 'Unary additiv epsilon indicator', fra nu af epsilon-indikator, behøver som 'hyper volumen' et referencepunkt for at kunne beregnes. Målet angiver det mindste tal, der lagt til referancepunktet R vil gøre dette R svagt domineret af A. Jo mindre

kvalitetsmålet er, jo tættere er approximationssættet A på det ønskede referencepunkt R. Altså ønskes dette kvalitetsmål også minimeret.

$$Eps\_Ind(A, R) = min\ \epsilon\ \{\text{ for hvilket } \exists x \in A : x \preceq_\epsilon R\} \tag{7.3}$$

I udtrykker ovenfor er relatione epsilon-dominans,$\preceq_\epsilon$ benyttet. Den er defineret som følger.

$$x \preceq_\epsilon y \Leftrightarrow \forall i \in 1..dim : x_i \leq \epsilon + y_i$$

Som for 'hyper volume' kan pareto fronten også benyttes hvis denne kendes. I det tilfælde findes indikatorværdien som den mindste afstandes der lagt til samtlige punkter i paretofronten vil gøre denne svagt domineret af A.

Ved benyttelse af kvalitetsmålene 'hyper volume' og 'Unary additiv epsilon indicator' skal man tage højde for størrelsesordenen af objektivværdierne. For at objektiverne skal vægte det samme i kvalitetsmålet, normeres objektiverne derfor til en værdi imellem 1 og 2, med følgende transformation.

$$x_i^{'} = \frac{x_i - x_{min}}{x_{max} - x_{min}} + 1 \tag{7.4}$$

hvor x_{max} og x_{min} kan være kendte eller estimerede maximun og minimum værdier.

7.2 SPEA2 - 'Strength Pareto Evolutionary Algorithm'

Før resultatet af parametertuningen bliver gennemgået og diskuteret i næste afsnit vil en af de selektionsstrategier der er benyttet i projektet blive præsenteret. SPEA2 [14] er valgt fordi det er en kendt metode med gode test resultater på en række problemer.

7.2.1 Elitisk udvælgelsesstrategi

En metode der har vist sig at give gode resultater i søgestrategierne er at opretholde et arkiv over de bedste løsningern i alle foregående iterationer. Hvor der med de bedste, menes de ikke dominerede. En metode der benytter denne teknik kaldes også elitisk. Der er forskelligt hvorvidt arkivet også tages med, når der skal udvælgels individer til reproduktion og om arkivet har en fast eller variabel størrelse. I SPEA2 er arkivstørrelsen fast og individerne i arkivet tages med i betragtning til reproduktion. Herunder er pseudokode for SPEA2 vist. I det følgende vil de tre skridt blive beskrevet.

Algorithm 10: SPEA2

1) **'Fitness'-beregning**
   Beregn 'fitness' værdi for alle individer i populationen P
2) **Miljøudvælgelse**
   Kopier alle ikke dominerede individer i P til arkivet A.
   Hvis antalet af ikke dominerede indivier overstiger $|A|$
   sorteres der i individerne på baggrund af deres indbyrdes afstand
   Hvis antallet er mindre end $|A|$ så fyld
   A med dominerede individer.
3) **Turneringsselektion**
   Foretag turneringsselektion for at finde individerne til
   mutations puljen V

7.2.2 'Fitness' værdi

'Fitness' værdien for et individ er i SPEA2 baseret på et styrkemål kombineret med et tætheds mål for individet afstand til de andre individer i populationen. Styrkemålet $S(i)$ for et individ i er et mål for antallet af løsninger et individ dominerer i population og arkivet.

$$S(i) = |\{j| \in \{P, A\} \text{ hvor } j \text{ dominerer } i\}| \tag{7.5}$$

Tæthedsmålet er beregnet ud fra k'te nærmeste nabo afstandsmålet. Det k'te nærmeste nabo afstandsmål findes for et individ ved at beregne afstanden til samtlige individer i objektkriterie-rummet. Herefter sorteres afstandene i aftagende orden. Den k'te afstand udgør nu afstandsmålet. Som k-værdi i afstandsmålet benyttes kvadratroden af det samlede antal individer i population og arkiv $k = \sqrt{|P| + |A|}$. Tæthedsmålet kan nu defineres som; $D(i) = \frac{1}{\rho_i^k + 1}$ hvor ρ er det k'te afstand mål for individet i. Den endelige 'Fitness'-værdi, $F(i)$ er summen af styrkemålet og tæthedsmålet.

$$F(i) = S(i) + D(i) \tag{7.6}$$

7.2.3 Miljøudvælgelse

Miljøudvælgelse dækker over den metode der benyttes til at fylde arkivet med individer. Arkivet består som tidligere nævnt af de ikke dominerede individer fra samtlige populationer indtil den pågældende iteration. Ved starten af miljøudvælgelsen haves en ny population af individer. Ved at finde de ikke dominerede individer i foreningsmængden af populationen og arkivet haves nu et midlertidigt arkiv A'. Det midlertidige arkiv A' kan have en størrelse der er mindre, lig med eller

større end arkivet. Hvis $|A'| < |A|$ fyldes de sidste pladser i A op med de individer fra P og A der har de største 'fitness'-værdier. Hvis $|A'| > |A|$ fjernes de individer der har den mindste afstand ρ^k til et andet individ fra A' indtil $|A'| = |A|$. Er $|A'| = |A|$, i forvejen gøres intet.

7.2.4 Turneringsselektion

Det sidste af de tre skridt er udvælgelsen af individer til mutationspuljen. Dette gøres ved at udvælge t-tilfældige individer, sammenligne dem og lade individet med den højeste 'fitness'-værdi gå til mutationspuljen.

Kapitel 8

Parametertuning

Dette kapitel vil omhandle parameterindstillingen af algorithmen, samt test og udvælgelse af selector. Kapitlet vil indeholde en beskrivelse af de opstillede eksperimenter, samt evalueringen af disses resultater vha de i foregående kapitel beskrevne værktøjer.

8.1 Test af parameterindstillinger

Parameterindstillingen af algorithmen omfatter indstillingen af syv sandsynligheder, som hver bestemmer sandsynligheden for kaldet af en mutationsoperator. Med syv kontinuerte parametre er det ikke muligt at teste alle parameterindstillingerne til bunds, det er derfor valgt i stedet at finde den rette vægtning der skal lægges imellem løsningen af allokeringsproblemet, mapping og skemalægningsproblemet, dvs hvor store sandsynlighederne er for kald af mutationsoperatorer der opererer på hvert af de tre delproblemer. For de fem operatorer, der opererer på arkitekturen, var det erfaringen at den samme sandsynlighed for de fem gav et godt resultat.

	p1	p2	p3	p4	p5	p6	p7
$p_{muterarkitektur}$	0.330	0.2	0.2	0.6	0.2	0.4	0.4
$p_{mutermapping}$	0.335	0.2	0.6	0.2	0.4	0.4	0.2
$p_{muteretvægtliste}$	0.335	0.6	0.2	0.2	0.4	0.2	0.4

Tabel 8.1: Parameterindstillinger testet.

Parameterindstilling af mutationsoperatorer på arkitekturen.

Sandsynlighed for kald af mutationsoperatoren	
p_{muter_bus}	$\frac{p_{muterarkitektur}}{5}$
$p_{ændre\_PE}$	$\frac{p_{muterarkitektur}}{5}$
$p_{tilføje\_PE}$	$\frac{p_{muterarkitektur}}{5}$
p_{fjern_PE}	$\frac{p_{muterarkitektur}}{5}$
$p_{crossover}$	$\frac{p_{muterarkitektur}}{5}$

Tabel 8.2: Sandsynlighederne for kald af mutationsoperatorerne på arkitekturen under parametertuningen.

Parametertuningen er lavet med SPEA2 da denne søgestrategi har givet gode resultater på en række problemer hidtil. Antallet af individer i tourneringen var sat til ti, hvilket erfaringsmæssigt gav gode resultater. Parametertuningen er lavet under antagelse af, at variatorens optimale parametertuning ikke afhænger af søgestrategien. Det har ikke været muligt at afprøve denne antagelse grundet den begrænsede tid til parametertuningen.

8.1.1 Test datasættet

Parametertuningen er lavet på en hyperperiode, hvor de enkelte opgavegrafer er genereret med TGFF [5], der er et værktøj til at generere tilfældige grafer. Hyperperioden er desuden lavet i en størrelse der er sammenlignelig med den af Case studiet. Da der ikke haves flere opgavegrafer fra virkelige applikationer, må det indtil videre antages at størrelsen af case studiet repræsenterer størrelsen af de indlejrede systemer generelt.

Data sæt			
Opgavegraf	antal opgaver	deadlines [sek]	antal kopier i hyperperioden
tg1	18	0.05	1
tg2	70	0.025	2
tg3	73	0.05	25
tg4	19	0.025	25
tg5	67	0.1	20
Samlet hyperperiode længde [sek]			0.1
Opgaver i hyperperioden			573

Tabel 8.3: Test data sættet

Som start løsning til parametertuningseksperimenterne er brugt en lille arkitektur med blot to GPP'er forbundet over en bus. Ideelt set skulle startarkitekturens betydning for resultaterne også testes mere indgående, dette har der desværre heller ikke været tid til.

8.1.2 Eksperimenternes størrelse

For algorithmen er der kørt 20 eksperimenter for hver parameterinstilling. Startpopulationen er på $|P|$=200 individer, og mængden af individer der muteres pr. generation er på $|V| = 100$. Hvert eksperiment løber over 100 generationer, hvilket svarer til knap 60 minutters kørsel. Eksperimenterne blev udført på IMM's sunfire server der har følgende specifikationer.

	antal processorer	RAM	clock frekvens	operativ system
sunfire 3800	8	16 GB	1200 Mhz	Solaris 9

8.1.3 Resultater

Resultatet af parametertuningseksperimentet er vist herunder. Tabellen viser middelværdi og varians for hyper volunen og epsilon indikatoren.

Indikator		**p1**	**p2**	**p3**	**p4**	**p5**	**p6**	**p7**
Hyper volune	Middelværdi	0.5068	0.3359	0.4672	0.5243	0.5162	0.6096	0.5267
	Varians	0.0591	0.0204	0.0595	0.0358	0.0595	0.0314	0.1142
Epsilon	Middelværdi	0.3412	0.1611	0.2489	0.3486	0.2705	0.4180	0.3642
	Varians	0.0348	0.0062	0.0290	0.0198	0.0231	0.0231	0.0848
Antal løsninger †		17	15	9	19	10	18	16

Tabel 8.4: Middelværdi og varians for hyper volune og epsilon indikator taget over 20 eksperimenter for hver parameter indstilling. † Antallet ud af 20 eksperimenter hvor der blev fundet løsninger der overholder deadlines. Da der ikke var kendt noget referencepunkt for test datasættet blev det fundet ud fra resultaterne, ved at tage den mindst opnåede værdi for hvert objekt. Det benyttede referencepunkt er $(865, 2.673661 * 10^7)$. Før indikatorværdierne er beregnet er løsninger der overtræder deadlines taget ud af sættene. Derefter er objektiverne normeret til værdier imellem 1 og 2 som vist i ligning 7.4.

Det første der skal bemærkes, er at algorithmen ikke giver løsninger der overholder deadlines i alle 20 kørsler. Parameterindstilling **p4** synes at være den bedste, hvis man vil have den største sandsynlighed for at finde brugbare løsninger. Dette er dog ikke argument nok for at vælge denne parameterindstilling. I indstilling **p4** vil algorithmen allokere flest processorer til det indlejrede system fordi der her haves en meget stor sandsynlighed for mutation på arkitekturen i denne. Det er lettere at overholde deadlines med en stor arkitektur, men det er også dyrere, hvilket parameterindstillingens dårlige hyper volume indikatorværdi viser.
Hvis man ser på indikatorværdierne er det i stedet indstilling **p2** der give det bedste resultat for både hyper volumen og epsilon indikatorerne. For at få en bedre vurdering af forskellen blev der lavet en Krusk-Wallis rank test på forskellen imellem parameterindstillingerne. I denne test viste **p2** sig bedre **p4** med et signifikansniveau på 0.010 for hyper volumen, som var den næstbedste i testen, og for epsilon indikatoren var signifikansniveauet ned til den næstbedste i rækken, nemlig **p3**, på 0.092.
'Dominance ranking' indikatoren er også benyttet. Rangen for hvert approximationssæt hørende til en parameterindstilling er beregnet i forholdt til approximationssættene for en anden parameterindstilling. I denne test viste **p2** sig også bedst her med et signifikansniveau over for **p1** på 0.0739.

8.1.3.1 Opsummering

Alt i alt viser resultaterne ikke nogen ubestridt bedst indstilling. Indstilling **p4** klarer sig bedst på de tre kvalitetsindikatorer, dog ikke med nogen stor signifikans. Eksperimentet viser også hvordan meget mutation på arkitekturen kan være en vej til at finde løsninger der overholder deadlines, hvilket dog er med dyrere arkitekturer til følge.

8.2 Test af udvælgelsesstrategier

I tilknytning til PISA-frameworket er de seks udvælgelsesstrategier tilrådighed. For hver af udvælgelsesstrategierne er der foretaget 20 eksperimenter. Hver enkelt af testene har den samme størrelse som for parametertuningen: 100 iterationer med en startpopulation på $|P|$=200 og $|V| = 100$.

Indikator		SEMO2	FEMO	SPEA2	NSGA2	ECEA	IBEA
HyperVolune	Middelværdi	0.3595	0.2337	0.3941	0.4805	0.6304	0.4936
	Varians	0.0377	0.0373	0.0153	0.0502	0.0245	9.3507e-04
Epsilon Indikator	Middelværdi	0.2508	0.1112	0.2573	0.2598	0.4461	0.2537
	Varians	0.0234	0.0085	0.0083	0.0280	0.0147	3.2640e-04
Antal løsninger		13	16	14	17	20	2

Tabel 8.5: Referencepunktet for testen af selektorerne blev fundet så samme måde som for parametertesten. Referencepunktet blev: $(865, 2.630268 * 10^7)$.

Middelværdi og varians er endnu engang udregnet for indikatorerne. Ligeledes udføres Krusk-Wallis test for hyper volumen og epsilon indikatorerne og en Mann-Withneq på 'dominace ranking' indikatoren. I disse tests viste FEMO sig bedst. På hyper volumen var testen for at FEMO var bedre end SEMO2 på signifikansniveau 0.075, og for epsilon indikatoren var signifikansen på 0.023 i en test op imod IBEA.
Til sidst blev FEMO testet parvis over for hver af de andre udvælgelsesstrategier på 'Dominance ranking' indikatoren. I denne test var den næstbedste udvælgelsesstrategier SEMO. Signifikansniveauet for at FEMO var bedre end SEMO var på 0.483. 'Dominance ranking' viser altså ingen signifikant forskel på de to. Givet signifikansnivauerne på de andre tests fremstår FEMO dog som den selektionsmetode der giver de bedste resultater.

8.2.0.2 Diskussion af selektorer

Det er værd at bemærke at søgestrategierne SPEA2 og NSGA2, der begge benytter 'dominance ranking' i beregning af 'fitness'-værdien, ikke klarer sig specielt godt sammenlignet med de andre.
I FEMO foretages i stedet udvælgelse på baggrund af en 'variation counter'. For hver mutation eller crossover et individ har deltaget i tælles denne op. Ved at vælge individer med den laveste værdi på sin 'variation counter' sørges der for at alle individer i populationen får samme mulighed for at generere nye individer.
ECEA, den udvælgelsesstrategi der giver flest løsninger, sørger også i høj grad for at brede sin søgning ud. Her lægges der et gitter ud over objektværdierne for populationen, hvorefter søgningen fordeles ligeligt i de forskellige rum i gitteret.

Grunden til at de 'brede' søgestrategier klarer sig bedre er givetvis at en del af de løsninger, der ligger på paretofronten og derfor har en høj 'fitness'-værdi er løsninger med deadline violations. Løsninger med få processorer, men store deadline violations ligger på grund af deres lave pris i det ikke domineredesæt af populationen og vil derfor være kandidat til nye løsninger i udvælgelsen. Problemet kan dog have en karakter der gør at disse løsninger aldrig kan overholde deadlines fordi de simpelthen er for små.
Dette kunne løses på to måder. Den første måde ville kræve at man havde et vist kendskab til problemet. I det tilfælde kunne man vurdere hvad det mindste antal processorer i arkitekturen var og undlade mutationer der ville bringe antallet af processor end under dette. Den anden metode ville indebære indførslen af en grænse på den maximale deadline violation som udvælgelsesstrategien ville tillade. Ved gradvist at sænke denne grænse under iterationsforløbet vil løsninger der klarer sig godt på objektiverne pris og energiforbrug på bekostning af deadline violations med tiden blive sorteret fra. Denne metode ville også fungere uden forudgående kendskab til problemet.

Kapitel 9

Casestudiet

I casestudiet vil de mulige indlejrede systemer til en smartphone blive undersøget. Casestudiet bygger på data fra en telefon der ér udviklet og sat i produktion. Som startløsning bruges i undersøgelsen en arkitektur, der ligeledes er implementeret og undersøgt i en anden sammenhæng.

9.1 Reduktion af hyperperioden

I casestudiet haves de fem opgavegrafer herunder. Det er valgt at forkort deadlinen på hhv jpeg encoderen og jpeg decoderen for at reducere længden af hyperperioden og dermed også antallet af opgaver.

Opgavegraf	antal opgaver	oprindelige deadlines	antal kopier i hyperperioden
jpeg dec	6	0.5	1
jpeg end	5	0.25	2
mp3 dec	16	0.025	20
gsm dec	34	0.02	25
gsm end	53	0.02	25
Samlet hyperperiode længde [sek]			0.5
Opgaver i hyperperioden			2511
Løsning tid pr skemalægningsproblem			2.5 sek

Tabel 9.1: Datasæt med oprindelige deadlines.

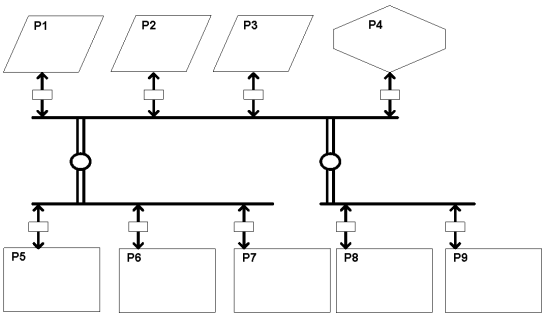
Som det ses af tabellen herunder, giver det en betragtelig reduktion i løsningstiden for skemalægningsproblemet, hvilket igen giver muligheder for evaluering af langt flere løsninger. Reduktionen af hyperperioden betyder at der skal udføres flere opgaver pr. sekund i hyperperioden, men løsningsresultaterne viste, at det forøgede antal løsningsevalueringer sagtens kunne opveje dette.

Opgavegraf	antal opgaver	reducerede deadlines	antal kopier i reducerede hyperperiode
jpeg dec	6	0.05	2
jpeg end	5	0.025	4
mp3 dec	16	0.025	4
gsm dec	34	0.02	5
gsm end	53	0.02	5
Samlet hyperperiode længde [sek]			0.1
Opgaver i hyperperioden			531
Løsning tid pr skemalægningsproblem			0.1 sek

Tabel 9.2: Datasæt med oprindelige deadlines.

9.2 WAND-arkitekturen

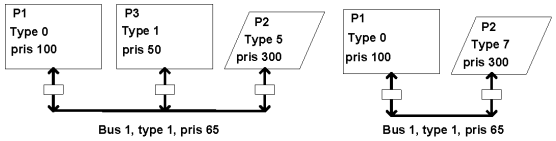
Som inputarkitektur til løsningen af casestudiet benyttes arkitekturen WAND [15]. Denne er benyttet i en trådløs multimedia terminal. Den vides derfor at være i stand til at køre multimedie applikationerne som i case studiet.



Figur 9.1: WAND arkitekturen fra [15]

9.3 Resultatet af optimeringsalgorithmen

Løsning af casestudiet gav to ikke dominerede løsninger der overholdt deadlines. Fra WAND arkitekturen med 9 processorer var der to løsninger på to og tre processorer. De ses herunder.

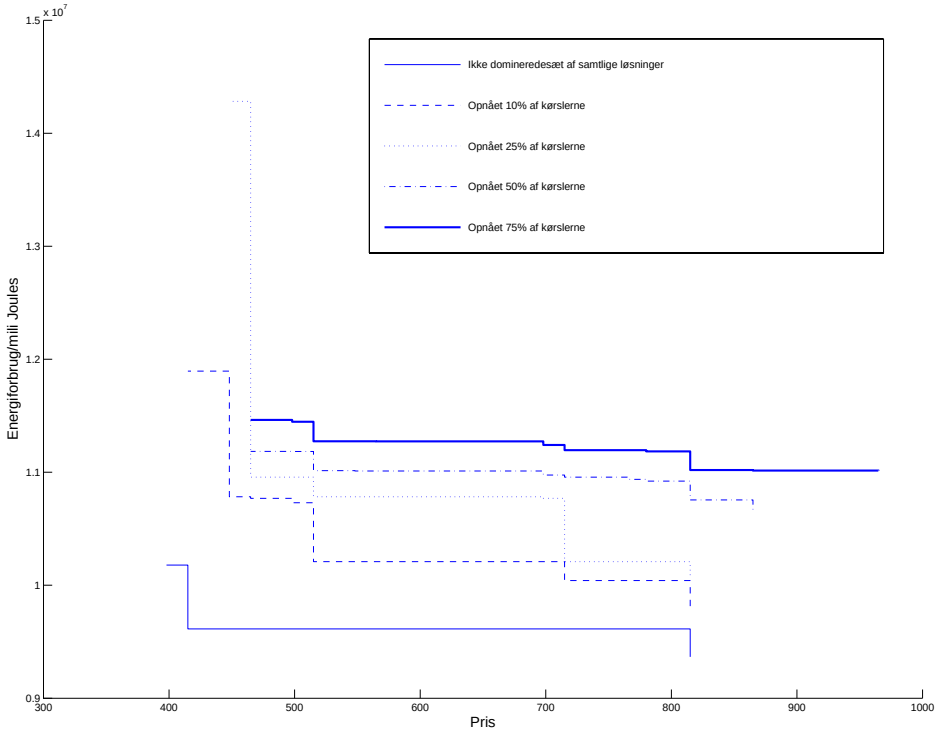


(a) (b)

Figur 9.2: To løsninger til case studiet. (a) Samlet pris: 515, samlet energiforbrug: $1.03708 * 10^7$ mili Jules. (b) Samlet pris: 465, samlet energiforbrug: $1.40955 * 10^7$ mili Jules

Imellem to arkitekturer ses der et tradeoff imellem pris og energiforbrug ved valget af to processorer frem for en ASIC.

For dels at vurdere spredningen i algorithmens løsningsresultater dels for at finde en approximations til pareto fronten blev algorithmen kørt 100 gange. Ud fra disse resultater kan blev de n'te opnåede kurver i objektivrummet bestemt. Den n'te opnåede kurve er således kurven der viser, hvilket område af objektivrummet er bliver dækket i n af de 100 eksperimenter.



Figur 9.3: Kurverne visende rummet der dækkes i hhv 10%, 25% og 75% af kørslerne samt det ikke dominerede sæt af samtlige løsninger

På kurverne ses det tydeliget at der ér at tredeoff imellem pris og energiforbrug. Af kurverne ses det desuden at de opnåede resultater i hhv 10% 25% og 50% af kørslerne ligger forholdsvist tæt. Altså er der ikke den store spredning på løsningsresultaterne fra algorithmen. Løsningerne i det ikke domineredesæt af samtlige løsninger udgør den bedste tilnærmelse af pretofronten ud fra alle løsninger. Dette sæt ligger stadigt markant bedre end 10%'s kurven, der er således stadigvæk mulighed for forbedringer i tilnærmelsen af paretofronten.

9.4 Andre anvendelser af algorithmen

Udover at algorithmen kan anvendes til co-synthesis, kan algorithmen i kraft af sin fleksible parameterindstilling også benyttest til at undersøge mere specifikke ting. Arkitekturen kan låses fast, og mappingen kan undersøges mere udførligt for en arkitektur. Antallet af processorer kan også låses fast og man kan undersøge arkitekturer med et bestemt antal processorer. Hvis man ønsker at undersøge arkitekturmulighederne mere grundigt for et indlejret system, er det oplagt at benytte algorithmen flere gange, til for eksempel at undersøge arkitekturer med et begrænset antal processorer.
For at gør det muligt at lave en mere omfattende undersøgelse af buffernes betydning i arkitekturen er der i algorithmen indført muligheden for at specificere nye buffertyper for alle processorer i arkitekturen ved parameterindstilling af algorithmen. Man kan således undersøge hvad betydningen er, hvis alle processorer har en outputbuffer på f.eks. 50 bytes, eller hvis processorne ingen buffere har. Det skal dog endnu engang understreges at parameterindstillingen sætter antallet af buffere for alle processorer i arkitekturen. Hvis man ønsker forskellige bufferstørrelser, i arkitekturen, må man ændre bufferstørrelsen i datasættet.

Kapitel 10

Konklusion

Projektet har vist hvordan multiobjektive genetiske algorithmer kan benyttes til at undersøge de mulige arkitekturer for et indlejret system. I arkitekturerne er der medtaget buffere og broer, hvilket så vidt vides, ikke er gjort af andre før.
Skemalægningen er foretaget med en list scheluling algorithme, der foretager mapping af kommunikationsopgaver. I skemalægningen skemalægges kommunikationen således, at bufferne benyttes hvis det giver mulighed for tidligere start eller afslutning af kommunikationsopgaver.
Den genetiske algorithme er udviklet i PISA-frameworket, hvorigennem en række kendte udvælgelsesstrategier er til rådighed. Af disse viste FEMO - 'Fair Evolutionary Multiobjective Optimizer' - sig at producere de bedste resultater.
Algorithmen er testet i et casestudie med et indlejret system for en smartphone. Løsningerne der er fundet er markant billigere arkitekturer og udgør et sæt, der viser tradeoff imellem pris og energiforbrug.

Litteratur

[1] Robert P. Dick & Niraj K. Jha, *MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-Synthesis of Distributed Embedded Systems.* Oct 1998.

[2] Michael R. Garey & David S. Johnson.*Computers and Intractability: A Guide to the Theory of NP-Completeness.* W HF reeman & Co, 1979.

[3] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles.*System-Level Design Techniques for Energy-Efficient Embedded Systems.* Kluwer Academic Publishers, 2004.

[4] M. T. Schmitz, B. M. Al-Hashimi and P. Eles *Cosynthesis of Energy-Efficient Multimode Embedded Systems With Considerationn of Mode-Execution Probabilities.* IEEE transactions on computer-aided design of intregrated circuits and systems, Vol. 24, No 2 Feb 2005

[5] Robert P. Dick, Davif L. Rhodes & Wayne Wolf *TGFF: Task Graphs for Free.* Proceedings of the Sixth International Workshop on Hardware/Software Codesign (CODES/CASHE'98) p.97-101

[6] Kianzad, V., Bhattacharyya, S.S, *CHARMED: A Multi-objective Co-synthesis Framework for Multi-mode Embedded Systems.* Application-Specific Systems, Architectures and Processors, p 28-40, 2004.

[7] Bharat P. Dave & Niraj K. Jha, *COFTA: Hardware-Software Co-Synthesis of Heterogeneous Distributed Embedded Systems for Low Overhead Fault Tolerance.* TRANSACTIONS ON COMPUTERS, VOL. 48, NO. 4, Arp 1999.

[8] Stefan Nleuler, Marco Laumanns, Lothar Thiele and Eckart Zitzler, *PISA - A Platform and Programming Language Independent Interface for Search Algorithms.* Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003. Proceedings (Lecture Notes in Computer Science Vol.2632) p 494-508, 2003.

[9] YU-KWONG KWOK & ISHFAQ AHMAD, *Static scheduling algorithms for allocating directed task graphs to multiprocessors.* ACM Computing Surveys, Vol. 31, No. 4, Dec 1999.

[10] Levman. J, Khan. G & Alirezaie. J,*Hardware-software co-synthesis of bus architecture embedded devices.* Canadian Conference on Electrical and Computer Engineering, 69-72 Vol.1, May 2004.

[11] Joshua D. Knowles, Lothar Thiele and Eckart Zitzler, *A tutorial on the Performance Assessment of StochasticMultiobjective Optimizers.* TIK-Report No. 214 July 2005.

[12] Tobias Blickle, Jürgen Teich & Lothar Thiele, *System-Level Synthesis Using Evolutionary Algorithms.* Design Automation for Embedded Systems, 3, 23-58, 1998.

[13] Robert P. Dick, *Multiobjective Syntesis of Low-Power Real-Time Distributed Embedded Systems.* A Dissertation presented to the Faculty Of Princeton University, Nov 2002.

[14] Eckart Zitzler, Marco Laumanns & Lothar Thiele *SPEA2: Improving the Strength Pareto Evolutionary Algorithm.* TIK-Report 103, May 2001

[15] P. Lettieri and M. B. Srivastava.*Advances in wireless terminal.* In IEEE Personal Communications, pages 6-19. IEEE, 1999.

Bilag A

Kildekode

I appendixet haves kildekoden for programmet. De dele af koden der foretager komminiukation med selektoren og operationer på polulationerne foretager af metoder fra PISA-fremeworket. Beskrivelsen af disse samt kilde koden kan findes på projektets hjemmeside;

`http://www.tik.ee.ethz.ch/pisa/`

A.0.1 data.hpp

```
#ifndef DATA_H
#define DATA_H

using namespace std;

#include <string>
#include <vector>
#include <iostream>
#include <fstream>
#include <math.h>

class TaskGraph{

private:
static const int MAX_taskes = 4000;
string name;
double HP;
double TOLERABLE_TIMING_PENALTY;

int num_taskes;
int num_edges;
int num_org_taskes;
vector<int> source_nodes;
int* task_type;
int* edge_type;
vector<double> deadlines;
vector<int> original_task_number;
vector<int> original_task_graph;
/** dependencies between task graphs **/
/** a dependenci indicates that the predecessor has to be
    schedueld before the successor - it could be see as a edge with
    on data  **/

vector<int> predecessor;
vector<int> successor;

vector<int> task_index_Starting_Times;
vector<double> Starting_Times;

public:

~TaskGraph(){ delete [] task_type; delete [] edge_type;};

void initilize_edges();
void initilize_taskes();
```

```
void initilize_taskes(int);
void read_task(string);
void read_task_tgff(string);
void build_hyper_period();
int get_num_taskes(){return num_taskes;}
int get_num_org_taskes(){return num_org_taskes;}
  int get_num_edges(){return num_edges;}
int get_num_sources(){return source_nodes.size();}
int get_source_node(int i){return source_nodes.at(i);}
string get_name(){return name;}
void set_task_type(int,int);
void set_edge_type(int,int,int);
int get_task_type(int);
int get_edge_type(int index1,int index2){
assert( index1 >= 0);assert( index2 >= 0);
assert( index1 < num_taskes);assert( index2 < num_taskes);
return edge_type[index1+(index2)*num_taskes];}
double get_HP(){return HP;}

void set_min_start_time(int task,double start_time){
 assert( task >= (int) Starting_Times.size());
 Starting_Times.push_back(start_time);}

  double get_min_start_time(int task){
 assert( task < (int) Starting_Times.size());
 return Starting_Times.at(task);}

/** FUNCTIONS FOR HYPER PERIOD **/
int get_original_task_graph(int index){
assert(index < (int)original_task_graph.size() );
return original_task_graph.at(index);}

void set_original_task_graph(int index,int TG){
        if(index < (int)original_task_graph.size()){
original_task_graph.at(index) = TG;}
else{ assert( index == (int)original_task_graph.size());
original_task_graph.push_back(TG);}}

int get_original_task_number(int task){return original_task_number.at(task);}
void set_original_task_number(int task,int old_task_number){
        if( task < (int)original_task_number.size()){
original_task_number.at(task) = old_task_number;}
else{ assert( task == (int)original_task_number.size());
original_task_number.push_back(old_task_number);}}


vector<int> get_dependencies_to_predecessor_list(){return successor;}
```

```
void set_dependenci(int pre,int succ){
   successor.push_back(succ);
   predecessor.push_back(pre);}

vector<int> get_predecessors(int task){
 vector<int> preds;
 for( unsigned int i = 0; i < successor.size();i++){
  if( successor.at(i) == task){
   preds.push_back(predecessor.at(i));}
  }
  return preds;}

int get_num_dependenci(){ return predecessor.size();}
int get_predecessor(int indx){return predecessor.at(indx);}
int get_successor(int indx){return successor.at(indx);}



double get_deadline(int task){assert(task<num_taskes);return deadlines.at(task);}
void sort_taskes();
};

void read_tech_file(string); // reads the technology file
void read_taskes();
void read_GPP(); // reads the technology file
void read_ASIC();
void read_FPGA();
void read_LINK();
void read_COMM();
void write_som(ifstream&);
void clear_mem_for_data();

class Tech{

private:
double cost;

public:
virtual ~Tech() = 0;


string name;
string type_;
string type(){return type_;}
string get_name(){return name;}
int num_taskes;
```

```
void set_num_taskes(string); // findes number of taskes

virtual double get_cost() = 0;
virtual double can_task_execute(int) = 0;
virtual double pins_available() = 0;
virtual double get_execution_time(int) = 0;
virtual double get_dyn_power(int) = 0;
virtual double get_StPwr() = 0;
virtual double get_commPower() = 0;
virtual double get_areal() = 0;
virtual double get_CommMem()= 0;


};

class GPP : public Tech{

private:
double spec[14];

int* taskSpec;

public:
GPP(int);
~GPP();
void read_GPP(ifstream&);
void set_spec(int[14]);

double pins_available(){return spec[3];}

virtual double get_cost(){return spec[0];}

virtual double get_execution_time(int type){
assert(get_taskSpec(type,7) > 0);
return (taskSpec[type*8+2]/(spec[2]*1000));}

virtual double get_dyn_power(int type){ return get_taskSpec(type,3);}

virtual double get_StPwr(){return spec[1];}
virtual double get_commPower(){return spec[12];}
int get_taskSpec(int type,int spec_){ return taskSpec[type*8+spec_];}
double get_spec(int spec_){return spec[spec_];}
virtual double can_task_execute(int task_type){ return get_taskSpec(task_type,7);};
virtual double get_areal(){ return 0;}
virtual double get_CommMem(){ return spec[13];}
};
```

```
class ASIC : public Tech{

private:
double spec[14];

int* taskSpec;

public:
ASIC(int);
~ASIC();
void read_ASIC(ifstream&);
double pins_available(){return spec[3];}

virtual double get_cost(){return spec[0];}

virtual double get_execution_time(int type){return (get_taskSpec(type,1)/(spec[2]*1000));}
virtual double get_dyn_power(int type){ return get_taskSpec(type,2);}

virtual double get_StPwr(){return spec[1];}
virtual double get_commPower(){return spec[12];}
virtual double get_areal(){return spec[6];}

double get_CommArea(){return spec[14];}
virtual double get_CommMem(){ return 13;}

int get_taskSpec(int type,int spec_){ return taskSpec[type*5+spec_];}
virtual double can_task_execute(int task_type){ return get_taskSpec(task_type,4);};

};

class FPGA : public Tech{

private:
double spec[16];

int* taskSpec;

public:
FPGA(int);
~FPGA();
void read_FPGA(ifstream&);
double pins_available(){return spec[3];}

virtual double get_cost(){return spec[0];}

virtual double get_execution_time(int type){return (get_taskSpec(type,1)/(spec[2]*1000));}
```

```
virtual double get_dyn_power(int type){ return get_taskSpec(type,2);}

virtual double get_StPwr(){return spec[1];}
virtual double get_commPower(){return spec[12];}
virtual double get_areal(){ return spec[6];}

double get_CommCLB(){ return spec[14];}
virtual double get_CommMem(){ return 13;}

int get_taskSpec(int type,int spec_){ return taskSpec[type*5+spec_];}
        virtual double can_task_execute(int task_type){ return get_taskSpec(task_type,4);};

};

class COMM{

private:

vector<int> edge_type;
vector<int> comm_amount;

public:

void read_COMM(ifstream&);

int get_comm_amount(unsigned int edge_type_){
assert( edge_type_ < edge_type.size());
return comm_amount.at(edge_type_);}

};

class LINK{

private:

double spec[10];
string name;

public:
string get_name(){return name;}
double get_cost(){return spec[0];}
double get_StPwr(){return spec[1];}
double get_aPower(){return spec[9];}
double get_spec(int index){assert(index < 10);return spec[index];}
double get_link_com_time(int edge,COMM* co){
int cycels = (int) ceil( (co->get_comm_amount(edge) / spec[6]) );
assert(cycels > 0);
```

```
return cycels/(spec[2]*1000);}

double get_comm_PowerCon(int edge,COMM* co){
int cycels = (int) ceil( (co->get_comm_amount(edge) / spec[6]) );
return (spec[9]*cycels);}

double get_max_users(){return spec[5];}
void set_spec();
void read_LINK(ifstream&);

double pin_recruitment(){return spec[3];};
};
 #endif /* DATA.H */
```

A.0.2 read_data.hpp

```
#include "variator.h"
#include "variator_user.h"
#include "variator_internal.h"
#include <math.h>

/*------------------- Functions for TaskGraph --------------*/

void TaskGraph::set_task_type(int index,int value){task_type[index] = value;}

void TaskGraph::set_edge_type(int index1,int index2,int value)
{ assert( index1 >= 0);
assert( index2 >= 0);
edge_type[index1+(index2)*num_taskes] = value;}

void TaskGraph::initilize_taskes(){
task_type = new int[MAX_taskes];
for( int i = 0; i < MAX_taskes; i++){
  task_type[i] = 0;}
}

void TaskGraph::initilize_taskes(const int size){
task_type = new int[size];
for( int i = 0; i < size; i++){
  task_type[i] = 0;}
}
void TaskGraph::initilize_edges(){
edge_type = new int[num_taskes*num_taskes];
for( int i = 0; i < num_taskes*num_taskes; i++){
  edge_type[i] = -1;}
```

```
}

int TaskGraph::get_task_type(int index){
assert(0 <= index);
assert(index < get_num_taskes());
return task_type[index];}

void TaskGraph::read_task(string name_){
name = name_;

        string tmp1;
        string input = name_;

 ifstream taskfile_in(input.c_str());
 taskfile_in.close();
 taskfile_in.open(input.c_str(),ios::in);

 string tmp;
 taskfile_in >> tmp;

 while(taskfile_in >> tmp){
  if( string::npos !=tmp.find("HYPERPERIOD",0) ){  taskfile_in >> HP; }
    if( string::npos !=tmp.find("TOLERABLE_TIMING_PENALTY",0) ){
   taskfile_in >> TOLERABLE_TIMING_PENALTY; break;}
  }
taskfile_in >> tmp;
taskfile_in >> tmp;
num_taskes = 0;
initilize_taskes();

while( taskfile_in >> tmp && (tmp == "Task:")){

  taskfile_in >> tmp;
taskfile_in >> tmp;
taskfile_in >> tmp;

taskfile_in >> tmp;
taskfile_in >> tmp;
   taskfile_in >> tmp;
set_task_type(num_taskes, atoi(tmp.c_str()));

taskfile_in >> tmp;
taskfile_in >> tmp;
taskfile_in >> tmp;
taskfile_in >> tmp;
taskfile_in >> tmp;
```

```
taskfile_in >> tmp;
double deadline_tmp = atof(tmp.c_str());

if( deadline_tmp == 0 ){ deadlines.push_back(0);}
else{ deadlines.push_back(deadline_tmp); }
num_taskes++;}

initilize_edges();
while( (tmp == "Edge:")  ){
  int source;
int dest;
int value;
  taskfile_in >> tmp;
taskfile_in >> tmp;
taskfile_in >> source;
taskfile_in >> tmp;



taskfile_in >> tmp;
taskfile_in >> tmp;
taskfile_in >> tmp;
taskfile_in >> dest;
taskfile_in >> tmp;
taskfile_in >> tmp;
taskfile_in >> value;
set_edge_type(source,dest,value);
taskfile_in >> tmp;}

}
void TaskGraph::read_task_tgff(string name_){
name = name_;
        string tmp1;
int max_type = 0;

 //string input = "../data/TaskGrphs/";
 //input = input.append(name_);
 string input = name_;
 ifstream taskfile_in(input.c_str());
 taskfile_in.close();
 taskfile_in.open(input.c_str(),ios::in);

 string tmp;
 taskfile_in >> tmp;
 taskfile_in >> HP;

   taskfile_in >> tmp;
```

```
   taskfile_in >> tmp;
   taskfile_in >> tmp;
 TOLERABLE_TIMING_PENALTY = 100000;
 taskfile_in >> tmp;
   taskfile_in >> tmp;

num_taskes = 0;
initilize_taskes();

 while( taskfile_in >> tmp && (tmp == "TASK")){
taskfile_in >> tmp;
/*int pos = tmp.find_first_of("_",0);
string tmp_task = substr(pos);
set_task_type(num_taskes, atoi(tmp_task.c_str()));
*/
taskfile_in >> tmp;
   taskfile_in >> tmp;
set_task_type(num_taskes, atoi(tmp.c_str()));

deadlines.push_back(0);
;
num_taskes++;
}

 initilize_edges();

while( (tmp == "ARC")  ){
  int source;
int dest;
int value;

  taskfile_in >> tmp;
taskfile_in >> tmp;
taskfile_in >> tmp;

int pos = tmp.find_first_of("_",0);
string tmp_task = tmp.substr(pos+1);
source = atoi(tmp_task.c_str());

taskfile_in >> tmp;
taskfile_in >> tmp;

pos = tmp.find_first_of("_",0);
tmp_task = tmp.substr(pos+1);
dest = atoi(tmp_task.c_str());

taskfile_in >> tmp;
```

```
taskfile_in >> tmp;
pos = tmp.find_first_of("_",0);
tmp_task = tmp.substr(pos+1);
value = atoi(tmp_task.c_str());
set_edge_type(source,dest,value);
if( value > max_type ){max_type = value;}

taskfile_in >> tmp; // edge
}


while( (tmp == "SOFT_DEADLINE")  ){
taskfile_in >> tmp; // from
taskfile_in >> tmp;
taskfile_in >> tmp; // to

int k = tmp.find( "_", 0 );
string tmp2 = tmp.substr(k+1,tmp.size());
int ttask = atof(tmp2.c_str());

taskfile_in >> tmp;
taskfile_in >> tmp;// soft deadline
deadlines.at(ttask) = atof(tmp.c_str());
taskfile_in >> tmp;}
taskfile_in.close();
}

void read_taskes(){
   for( int t = 0; t < num_TG;t++){
   cout << " Reading task graph # " << t+1 << endl;
   TaskGraphs[t] = new TaskGraph;
if( string::npos !=TaskGraph_names[t].find(".tg",0) ){
       TaskGraphs[t]->read_task(TaskGraph_names[t]);}
if( string::npos !=TaskGraph_names[t].find(".tgff",0) ){
       TaskGraphs[t]->read_task_tgff(TaskGraph_names[t]);}
   }
  cout << "done reading task graphs " << endl;



if( build_hyperperiod ){

HyperGraph = new TaskGraph;
cout << "building hyperperiod " << endl;
HyperGraph->build_hyper_period();
    cout << " hyperperiod build " << endl;
}
```

```
cout << " get_num_taskes() " <<  HyperGraph->get_num_taskes() << endl;
}

/*------------------- clear memory --------------*/

void clear_mem_for_data(){

delete HyperGraph;

for( int t = 0; t < num_TG;t++){
cout << " deletting tg #" << t << endl;
 delete TaskGraphs[t];}

for(unsigned int i = 0; i < Techs.size(); i++){
 delete Techs.at(i);}

for(unsigned int i = 0; i < Links.size(); i++){
 delete Links.at(i);}

for(unsigned int i = 0; i <Comm.size();i++){
 delete Comm.at(i);}
 cout << " clear mem finished " << endl;
}

/*------------------- Functions for Technology --------------*/

void read_tech_file(string filename){

        string tmp1, tmp2;

// string input = "../data/";
// input = input.append(filename);
string input = filename;
cout <<  input  << endl;

        int num_taskes;
{
ifstream taskfile_in(input.c_str());
int type = 0;

        while(taskfile_in >> tmp1){


 if( tmp1 == "type"){
 taskfile_in >> tmp1;
 taskfile_in>> tmp1;
 taskfile_in>> tmp1;
```

```
 taskfile_in>> tmp1;
 taskfile_in >> tmp1;
 taskfile_in>> tmp1;
 taskfile_in >> tmp1;
  while(taskfile_in >> num_taskes){
   if(type == num_taskes){
   type++;num_taskes++;
   }
 }
}

       }
       num_taskes = type;
       taskfile_in.close();

       }


       /* reads data  */
       ifstream taskfile_in(input.c_str());

       int i = 0;
       while(taskfile_in >> tmp1 && i < 800 ){

if( tmp1 == "@GPP"){
 GPP* PE_tmp = new GPP(num_taskes);
   PE_tmp->read_GPP(taskfile_in);
 Techs.push_back(PE_tmp);
  i++;
  }

if( tmp1 == "@ASIC"){
 ASIC* ASIC_tmp = new ASIC(num_taskes);
 ASIC_tmp->read_ASIC(taskfile_in);
 Techs.push_back(ASIC_tmp);
  i++;
  }
   if( tmp1 == "@FPGA"){
  FPGA* FPGA_tmp = new FPGA(num_taskes);
  FPGA_tmp->read_FPGA(taskfile_in);
    Techs.push_back(FPGA_tmp);
  i++;
  }

if( tmp1 == "@LINK"){
cout << " NEW LINK " << num_taskes << endl;
  LINK* LINK_tmp = new LINK;
```

```
  LINK_tmp->read_LINK(taskfile_in);
  Links.push_back(LINK_tmp);
  i++;
  }

if( tmp1 == "@COMM_AMOUNT"){
cout << " @COMM_AMOUNT " << endl;
  COMM* comm_tmp = new COMM;
  comm_tmp->read_COMM(taskfile_in);
  Comm.push_back(comm_tmp);
  i++;
  }

  assert( i < 800);



       }
}; // end read_tech

Tech::~Tech(){ };

/* GPP */

GPP::GPP(int num_taskes_){ taskSpec = new int[8*num_taskes_];
      num_taskes = num_taskes_;
   type_ = "GPP";
};
GPP::~GPP(){ delete taskSpec;}

void GPP::read_GPP(ifstream& taskfile_in){

string tmp1;
taskfile_in >> tmp1;
name = tmp1;
taskfile_in >> tmp1;


char tmp_c[400];
taskfile_in.getline(tmp_c,400);taskfile_in.getline(tmp_c,400);



    taskfile_in >>spec[0]; // price
    taskfile_in >> spec[1]; // StPwr
    taskfile_in >> spec[2]; // freq
    taskfile_in >> spec[3]; // pins
```

```
    taskfile_in >> spec[4]; // BEvSleep
    taskfile_in >> spec[5]; // BEvIDLE
    taskfile_in >> spec[6]; // AddsMem
    taskfile_in >> spec[7]; // DVS
    taskfile_in >> spec[8]; // Vmax
    taskfile_in >> spec[9]; // vt
    taskfile_in >> spec[10]; // CommBuffer
    taskfile_in >> spec[11]; // CommTime
    taskfile_in >> spec[12]; // CommPower
    taskfile_in >> spec[13]; // CommMem

 bool done_reading = false;
 while(taskfile_in >> tmp1 && done_reading == false){

   if( tmp1 == "type" && done_reading == false){

 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;

 int count = 0;

 while(  count <= num_taskes-1 && done_reading == false){

           taskfile_in >> taskSpec[count*8+0];
   taskfile_in >> taskSpec[count*8+1]; // version;
   taskfile_in >> taskSpec[count*8+2]; //ExeCyc;
   taskfile_in >> taskSpec[count*8+3]; //DynPwr;
   taskfile_in >> taskSpec[count*8+4]; //StMem;
   taskfile_in >> taskSpec[count*8+5]; //DynMem;
   taskfile_in >> taskSpec[count*8+6];//Preem;
   taskfile_in >> taskSpec[count*8+7];//Exable;

      if(count >= num_taskes-1){ done_reading = true;}

 count++;
 } // end while num taskes

   char buff[800];
   taskfile_in.getline(buff,809);
   done_reading = true; break;
} // ned if == type
} // end while
```

```
}; // end read_GPP


/* ASIC */

ASIC::ASIC(int num_taskes_){  taskSpec = new int[5*num_taskes_];
num_taskes = num_taskes_;};
ASIC::~ASIC(){ delete taskSpec;}


void ASIC::read_ASIC(ifstream& taskfile_in){

  string tmp1;
  taskfile_in >> tmp1;
  name = tmp1;
  taskfile_in >> tmp1;
  char tmp_c[400];
  taskfile_in.getline(tmp_c,400);taskfile_in.getline(tmp_c,400);

    taskfile_in >>spec[0]; // price
    taskfile_in >> spec[1]; // StPwr
    taskfile_in >> spec[2]; // freq
    taskfile_in >> spec[3]; // pins
    taskfile_in >> spec[4]; // BEvSleep
    taskfile_in >> spec[5]; // BEvIDLE
    taskfile_in >> spec[6]; //  Area
    taskfile_in >> spec[7]; // DVS
    taskfile_in >> spec[8]; // Vmax
    taskfile_in >> spec[9]; // vt
    taskfile_in >> spec[10]; // CommBuffer
    taskfile_in >> spec[11]; // CommTime
    taskfile_in >> spec[12]; // CommPower
    taskfile_in >> spec[13]; //CommArea


 bool done_reading = false;

 while(taskfile_in >> tmp1){
 if( tmp1 == "type"){
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 int count = 0;
 while(  count <= num_taskes-1 && done_reading == false){
   taskfile_in >> taskSpec[count*5];//task;
```

```
   taskfile_in >> taskSpec[count*5+1];//ExeCyc;
   taskfile_in >> taskSpec[count*5+2];//DynPwr;
   taskfile_in >> taskSpec[count*5+3];//Area;
   taskfile_in >> taskSpec[count*5+4];//Exable;

      if(count >= num_taskes-1){ done_reading = true;}

 count++;
 } // end while num taskes
   // taskfile_in >> task && c

  char buff[800];
  taskfile_in.getline(buff,800);
  taskfile_in.getline(buff,800);
          taskfile_in.getline(buff,800);
    done_reading = true; break;
} // ned if == type
 };// end while

}

FPGA::FPGA(int num_taskes_){  taskSpec = new int[5*num_taskes_];
    num_taskes = num_taskes_;};
FPGA::~FPGA(){ delete taskSpec;}

void FPGA::read_FPGA(ifstream& taskfile_in){

 string tmp1;
 taskfile_in >> tmp1;
 name = tmp1;
 taskfile_in >> tmp1;
 char tmp_c[400];
 taskfile_in.getline(tmp_c,400);taskfile_in.getline(tmp_c,400);

 taskfile_in >> spec[0]; // price
 taskfile_in >> spec[1]; // StPwr
 taskfile_in >> spec[2]; // freq
 taskfile_in >> spec[3]; // pins
 taskfile_in >> spec[4]; // BEvSleep
 taskfile_in >> spec[5]; // BEvIDLE
 taskfile_in >> spec[6]; // CLBs
 taskfile_in >> spec[7]; // DVS
 taskfile_in >> spec[8]; // Vmax
 taskfile_in >> spec[9]; // vt
 taskfile_in >> spec[10]; //CommBuffer
 taskfile_in >> spec[11]; //CommTime
 taskfile_in >> spec[12]; //CommPower
```

```
 taskfile_in >> spec[13]; //CommCLB
 taskfile_in >> spec[14]; //ReconT
 taskfile_in >> spec[15]; //ReconPwr

 bool done_reading = false;


 while(taskfile_in >> tmp1){
 if( tmp1 == "type"){
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;

 int count = 0;
 while(  count <= num_taskes-1 && done_reading == false){
   taskfile_in >> taskSpec[count*5];//task;
   taskfile_in >> taskSpec[count*5+1];//ExeCyc;
   taskfile_in >> taskSpec[count*5+2];//DynPwr;
   taskfile_in >> taskSpec[count*5+3];//CLB;
   taskfile_in >> taskSpec[count*5+4];//Exable;

      if(count >= num_taskes-1){ done_reading = true;}

 count++;
 } // end while num taskes
char buff[800];
taskfile_in.getline(buff,809);
done_reading = true; break;

} // ned if == type
 };// end while
};


void COMM::read_COMM(ifstream& taskfile_in){
 string tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;


 bool read = true;

 do{
```

```
 taskfile_in >> tmp1;
 if(tmp1 == "}"){read = false;break;}
 edge_type.push_back(atoi(tmp1.c_str()));
 if(tmp1 == "}"){read = false;}
 taskfile_in >> tmp1;
 comm_amount.push_back(atoi(tmp1.c_str()));

 }while(read);
}

void LINK::read_LINK(ifstream& taskfile_in){
 string tmp1;
 name = tmp1;
 taskfile_in >> tmp1;
 taskfile_in >> tmp1;
 char tmp_c[400];
 taskfile_in.getline(tmp_c,400);taskfile_in.getline(tmp_c,400);
 taskfile_in >> spec[0]; // price
 taskfile_in >> spec[1]; // StPwr
 taskfile_in >> spec[2]; // freq
 taskfile_in >> spec[3]; // pins
 taskfile_in >> spec[4]; // BEvSleep
 taskfile_in >> spec[5]; // maxUser
 taskfile_in >> spec[6]; // PckSize
 taskfile_in >> spec[7]; // PckOver
 taskfile_in >> spec[8]; // wTimeRate
 taskfile_in >> spec[9]; // aPower
}

/*----------------------------------------
build_hyper_period() |-------------------------------------------------*/

void  TaskGraph::build_hyper_period(){

 int num_taskes_in_hyperperiod = 0;
 num_org_taskes = 0;
 double hyperperiod_length = 0;
 name = "Hypergraph";
 for( int t = 0; t < num_TG;t++){
  if( TaskGraphs[t]->get_HP() > hyperperiod_length ){
  hyperperiod_length =  TaskGraphs[t]->get_HP();
  HP = hyperperiod_length;}}


  cout << "hyperperiod_length " << hyperperiod_length << endl;
  double remainder = -1;
   for( int t = 0; t < num_TG;t++){
```

```
   double td1 = hyperperiod_length;
   double td2 = TaskGraphs[t]->get_HP();
      remainder = fmod( hyperperiod_length, TaskGraphs[t]->get_HP());
      remainder = fmod(td1,td2);

 cout << remainder << " fmod(" <<  hyperperiod_length<<","<< TaskGraphs[t]->get_HP() <<")"<< endl;

cout << (remainder != 0.0)<< "&&"<<(remainder -TaskGraphs[t]->get_HP() != 0.0)<< endl;
double d1 = (remainder - TaskGraphs[t]->get_HP());

     if( (remainder != 0.0) ){
      if( d1 > 1*10^(-15) || d1 < -1*10^(-15)  ){
      cout <<  remainder <<"!= 0.0  && "<< remainder << "!=" << TaskGraphs[t]->get_HP()<< endl;
      cout <<" type hyperperiod, LCM - tg periods " << endl;
      hyperperiod_length = 0.25;

/*    Giv input
      cin >> hyperperiod_length;
   For Case study*/
 hyperperiod_length = 0.1;
      HP = hyperperiod_length;
      cout << " HYPER PERIOD " << HP << endl;
      break;}
      }
   }

 for( int t = 0; t < num_TG;t++){
 int num_copies = (int) (hyperperiod_length/(TaskGraphs[t]->get_HP()));
 copies.push_back(num_copies);

  for( int c = 0; c < num_copies; c++){
  num_taskes_in_hyperperiod+= TaskGraphs[t]->get_num_taskes();}
}

 num_taskes = num_taskes_in_hyperperiod;

 string ter;
 initilize_taskes();
 initilize_edges();
 int start_index = 0;// START TASK FOR NEW TG IN THE HYPE GRAPH
 int task_index = 0;
 int TG_copies = 1;
 for( int t = 0; t < num_TG;t++){

 double sum_of_HP_for_TG = 0;
 TaskGraph* TG_tmp = TaskGraphs[t];
```

```
 num_org_taskes += TG_tmp->get_num_taskes();
 while( (sum_of_HP_for_TG+TG_tmp->get_HP())  <= (hyperperiod_length+0.0000001) ){

  TG_copies++;
  vector<int> source_nodes_with_dependenci;
  vector<int> exit_nodes;
  for( int task = 0; task < TG_tmp->get_num_taskes();task++){
   set_original_task_graph(task_index,t);
   set_original_task_number(task_index,task);
set_task_type(task_index,TG_tmp->get_task_type(task));

set_min_start_time(task_index,sum_of_HP_for_TG);

     bool source = true;
     bool exit_node = true;
     for( int t_ = 0; t_ < TG_tmp->get_num_taskes(); t_++){
      if( TG_tmp->get_edge_type(t_,task) >= 0  ){ source = false;}
      if( TG_tmp->get_edge_type(task,t_) >= 0  ){ exit_node = false;} }
      if( source == true ){
 if( sum_of_HP_for_TG == 0){
  /* source node */
  source_nodes.push_back(task_index);} /** source node for hypergraph **/
 else{
  /* source node with dep */
  source_nodes_with_dependenci.push_back(task_index);
    }
 }
if( exit_node == true ){ exit_nodes.push_back(task_index);}

      if( TG_tmp->get_deadline(task) > 0 ){
deadlines.push_back(TG_tmp->get_deadline(task)+sum_of_HP_for_TG);
      }else{ deadlines.push_back(0);}
     assert( start_index+task == task_index);
        task_index++;
  }// loop taskes


  for( int from = 0; from < TG_tmp->get_num_taskes(); from++){
   for( int to = 0; to <  TG_tmp->get_num_taskes(); to++){
    if( TG_tmp->get_edge_type(from,to) >= 0){
  set_edge_type(from+start_index,to+start_index,TG_tmp->get_edge_type(from,to));
  assert( (from+start_index) < (to+start_index) );
  }
    }
   }

/* Setting dependencies */
```

```
start_index += TG_tmp->get_num_taskes();
if( sum_of_HP_for_TG+TG_tmp->get_HP() < hyperperiod_length+0.0000001 ){
 for(unsigned int i = 0; i < source_nodes_with_dependenci.size(); i++){
  for( unsigned int j = 0; j < exit_nodes.size(); j++){
   //if( get_original_task_graph(exit_nodes.at(j)) == get_original_task_graph(
/** NB NB Sættet kun til den sidste - det skal være til alle exit knuder --- **/
set_dependenci( exit_nodes.at(j)-TG_tmp->get_num_taskes(), source_nodes_with_dependenci.at(i) );}
}
sum_of_HP_for_TG += TG_tmp->get_HP();
}

}/* end while-loop over TG copies */
}/* end loop over TaskGraphs */
}
```

A.0.3 read_parameters.cpp

```
 ifstream parameters(paramfile);
 string tmp1;

 vector<int> PEs;
 int NumberOfStartPEs = 0;
 int seed;
 {
 string tmp1;
 parameters >> tmp1;
  if(tmp1 == "seed" ){
  parameters >> tmp1;
  seed = atoi(tmp1.c_str());
  cout << " seed " << seed << endl;
  }
 }


 {
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
  num_TG =atoi(tmp1.substr(k+1,l-k-1).c_str());
  }
 }
 }
```

```
/* TaskGraph - dir */
string dir;
{parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
          dir = tmp1.substr(k+1,l-k-1);
  k = l;
  }
 }
 }


/* TaskGraph - files */
TaskGraph_names.clear();
{parameters >> tmp1;
   string tmp0;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  tmp0 = dir;
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
  tmp0.append(tmp1.substr(k+1,l-k-1));
  cout << tmp0 << endl;
  TaskGraph_names.push_back(tmp0);
  k = l;
  }
 }


 assert(num_TG <= (int) TaskGraph_names.size());
//num_TG = TaskGraph_names.size();
}

/* Technologi fil*/
string Tech_file;
{
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 string tmp0 = dir;
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
```

```
  l = i;
          Tech_file = tmp0.append(tmp1.substr(k+1,l-k-1));
  cout << Tech_file << endl;
  }
 }

/* MaxIterations */
{string MaxIteration_string;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
          MaxIteration_string = tmp1.substr(k+1,l-k);
  }
 }
 MaxIterations = atoi(MaxIteration_string.c_str());
}

/* Execution_Time */
 {string Exe_Time;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
         Exe_Time = tmp1.substr(k+1,l-k-1);
cout << Exe_Time << "<-" << endl;

  }
  Execution_Time = atof(Exe_Time.c_str());
 }
}

/* objectives */
objectives.clear();
 {string obj;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
         obj = tmp1.substr(k+1,l-k-1);
if( obj == "Time"){objectives.push_back(0);}
```

```
if( obj == "Energy" ){objectives.push_back(1);}
if( obj == "Price" ){objectives.push_back(2);}
if( obj == "HDV" ){objectives.push_back(3);}
k = l;
  }
 }
}



  /* prob_mutate_bus_layout */
{
 string prob;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
      string numPE;
      prob = tmp1.substr(k+1,l-k-1);
      prob_mutate_bus_layout = atof(prob.c_str());
      k = l;
  }
 }

}


/* Mutation Prob */
MP.clear();
{
 string prob;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
      string numPE;
         prob = tmp1.substr(k+1,l-k-1);
cout << " prob " << prob << endl;
// cout << atof(prob.c_str()) << endl;
MP.push_back(atof(prob.c_str()));
k = l;
  }
 }
```

```
cout<<(double)MP.at(0)+MP.at(1)+MP.at(2)+MP.at(3)+MP.at(4)+MP.at(5)+MP.at(6) << endl;
double sum =(MP.at(0)+MP.at(1)+MP.at(2)+MP.at(3)+MP.at(4)+MP.at(5)+MP.at(6)-1.0);

/*
bool b1 = (bool) (sum > 1*10^(-15));
bool b2 = (bool) (sum < -1*10^(-15));*/
cout << sum << endl;
cout << sum*100 << endl;
bool b1 = (bool) (sum > 0.0001);
bool b2 = (bool) (sum < -0.0001);
cout <<b1 << " " << b2 << endl;


if( b1 || b2 ){cout << " error in mutation probabilities " << endl;exit(-1);}

}
 /*Mamapping  */
 {string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
          tmp= tmp1.substr(k+1,l-k-1);
  }
 }
  MP_mapping = atof(tmp.c_str());
}


 /*CrossOverPropability  */
 {string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
          tmp= tmp1.substr(k+1,l-k-1);
  }
 }
  CrossOverPropability = atof(tmp.c_str());
}


/*obj_crossover*/
```

```
{string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
          tmp= tmp1.substr(k+1,l-k-1);
  }
 }
  if( tmp == "Time"){obj_crossover = 0;}
  if( tmp == "Energy" ){obj_crossover = 1;}
  if( tmp == "Price" ){obj_crossover = 2;}
  if( tmp == "Areal" ){obj_crossover = 3;}
}

/* PeriodicMapping */
{string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
          tmp= tmp1.substr(k+1,l-k-1);
  }
 }
 PeriodicMapping = atoi(tmp.c_str());
}

/* MotationOnPreList:0; */
{string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
          tmp= tmp1.substr(k+1,l-k-1);
  }
 }
 MotationOnPreList = atof(tmp.c_str());
}

/* waight_pre_meshs */
{string tmp;
 parameters >> tmp1;
```

```
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
          tmp= tmp1.substr(k+1,l-k-1);
  }
 }
 waight_pre_meshs = atof(tmp.c_str());
}

/*NumberOfStartPEs*/
 {string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
         tmp= tmp1.substr(k+1,l-k-1);
  }
 }
  NumberOfStartPEs= atoi(tmp.c_str());
}

/*MaxPEs*/
 {string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
         tmp= tmp1.substr(k+1,l-k-1);
  }
 }
  MaxPEs= atoi(tmp.c_str());
}

/*use_tech_buffer_spec*/
 {string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
```

```
        tmp= tmp1.substr(k+1,l-k-1);
  }
 }
 if( atoi(tmp.c_str()) == 1 ){
 use_tech_buffer_spec = true;}
 else{use_tech_buffer_spec = false;}

}


/*Buffer Sizes*/
BufferSpec.clear();
 {
for( int i = 0; i < 3; i++){
 string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
        tmp= tmp1.substr(k+1,l-k-1);
  }
 }
  BufferSpec.push_back(atoi(tmp.c_str()));
 } // end for i
}

/*Var Buffer size*/
 {
 for( int i = 0; i < 3; i++){
 string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
        tmp= tmp1.substr(k+1,l-k-1);
  }
 }
  BufferSpec.push_back(atoi(tmp.c_str()));
  } // end for i
}

/* test_scheduel yes/no*/
{
```

```
 string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
         tmp= tmp1.substr(k+1,l-k-1);
  }
 }
 if( 1 == atoi(tmp.c_str())){
  do_test = true;}
 else{
    do_test = false;}
}

{  string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
         tmp= tmp1.substr(k+1,l-k-1);
  }
 }
 if( 1 == atoi(tmp.c_str())){
  input_assignment = true;}
 else{
    input_assignment = false;}
}
{  string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
         tmp= tmp1.substr(k+1,l-k-1);
  }
 }

  architectures =  atoi(tmp.c_str());

}
{  string tmp;
 parameters >> tmp1;
```

```
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
         tmp= tmp1.substr(k+1,l-k-1);
  }
 }
 if( 1 == atoi(tmp.c_str())){
   build_hyperperiod = true;}
 else{
     build_hyperperiod = false;}
}

{  string tmp;
 parameters >> tmp1;
 int k,l;
 k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i;
         tmp= tmp1.substr(k+1,l-k-1);
  }
 }
 if( 1 == atoi(tmp.c_str())){
   print_scheduel_info = true;}
 else{
   print_scheduel_info = false;}
}

parameters.close();
}
```

A.0.4 variator_user.h

```
/*============================================================================
  PISA  (www.tik.ee.ethz.ch/pisa/)

  ============================================================================
  Computer Engineering (TIK)
  ETH Zurich

  ============================================================================
  PISALIB
```

```
  Pisa basic functionality that have to be implemented by the user

  Header file.

  file: variator_user.h
  author: Fabian Landis, flandis@ee.ethz.ch

  revision by: Stefan Bleuler, bleuler@tik.ee.ethz.ch
  last change: $date$

  ============================================================================

  variator for Master Thesises in co-synthesis
  Peter Kjærulff 28/4-2005
*/

#include "data.hpp"

#ifndef VARIATOR_USER_H
#define VARIATOR_USER_H

#define PISA_UNIX /**** replace with PISA_WIN if compiling for Windows */

#include <pthread.h> /** for multipel threads in the program (linux) **/

/* maximal length of filenames */
#define FILE_NAME_LENGTH 128  /**** change the value if you like */

/* maximal length of entries in local cfg file */
#define CFG_NAME_LENGTH 128   /**** change the value if you like */

/*----------------| declaration of global variables |------------------*/

/* defined in variator_user.c */

extern char paramfile[]; /* file with local parameters */

extern char *log_file; /* file to log to, defined in variator_user.c */

extern TaskGraph* TaskGraphs[100];

extern TaskGraph* HyperGraph;

extern vector<Tech*> Techs;

extern vector<LINK*> Links;
```

```
extern vector<COMM*> Comm;

extern vector<int> objectives;

extern int obj_crossover;

extern double prob_mutate_bus_layout;

extern vector<double> MP;

extern double MP_mapping;

extern bool PeriodicMapping;

extern double CrossOverPropability;

extern bool StalicPreList;

extern double MotationOnPreList;

extern double waight_pre_meshs;

extern bool use_tech_buffer_spec;

extern vector<int> BufferSpec;

extern vector<int> copies;

extern bool do_test;

extern bool input_assignment;

extern int architectures;

extern bool build_hyperperiod;

extern int MaxPEs;

extern bool print_scheduel_info;
/*----------------------| individual struct |--------------------------*/

#include "TaskAssignment.hpp"

class individual_t
{
     private:
```

```
    double cost;
    double energy_con;
    double time;
    double HDV;
    double smalest_slack;


    int num_PEs;
    int num_Links;
    int num_TGs;
    vector<int> Allocation; /* vector with amount of the different PE types */
    vector<int> PE_type; /* vector with PE type for each PE */
    vector<TaskAssignment> TaskAssignments;
    vector<int> Links_indiv;

    vector<int> PE_Link_Connection;
    vector<double> buffer_use;
    vector<double> mutated_priority_weight;
    vector<int> TaskesWithDeadlineViolation;
    vector<double> DeadlineViolations;
    vector<double> Energy_con_on_PEs;
    vector<double> HDV_PE;

    /* in variator.h a typedef is used to the define the type
    individual based on the individual_t struct defined here. */
    void make_com_connection_2busses(int);
    void make_com_connection();

    /*------------------| functions for individual class |----------------*/


    public:
    vector<int> bus1;
    vector<int> bus2;

    individual_t(int);
    individual_t(individual* to_copy);
    /** destrouctor --> default destructor **/

   void set_buffer_use(int PE,int bus,double size1,double size2,double size3){
   buffer_use.push_back(PE),buffer_use.push_back(bus),
     buffer_use.push_back(size1),buffer_use.push_back(size2),buffer_use.push_back(size3);}

    vector<double> get_buffer_use(){return buffer_use;}
    void reset_TaskWDV(){
TaskesWithDeadlineViolation.erase(TaskesWithDeadlineViolation.begin(),TaskesWithDeadlineViolation.end());
     DeadlineViolations.erase(DeadlineViolations.begin(),DeadlineViolations.end());}
```

```
     void set_TaskWDV(int task,double DV){
     vector<int>::iterator it1_t = TaskesWithDeadlineViolation.begin();
     vector<int>::iterator it2_t = TaskesWithDeadlineViolation.end();
     vector<double>::iterator it1_DV =  DeadlineViolations.begin();
     vector<double>::iterator it2_DV =  DeadlineViolations.end();
      for( ; it1_t != it2_t; it1_t++,it1_DV++ ){
        if( DV > *it1_DV ){
    TaskesWithDeadlineViolation.insert( it1_t, 1, task );
    DeadlineViolations.insert(it1_DV, 1, DV);
    break;}
     }
     if( it1_t == it2_t){
      TaskesWithDeadlineViolation.push_back(task);
DeadlineViolations.push_back(DV);}
     }

     vector<int> get_TasksWDV(){return  TaskesWithDeadlineViolation;}
     vector<double> get_DVs(){return  DeadlineViolations;}

     void set_num_PEs(int PE){ num_PEs = PE;}
     int get_PE_type(int i){return PE_type.at(i);}
     int get_num_PEs(){return num_PEs;}
     int get_num_Links(){return num_Links;}
     int get_Link_Type(int i){return Links_indiv.at(i);}
     int get_num_TGs(){return num_TGs;}


     void get_architecture(int);
     int get_Allocation_size(){return Allocation.size();}
     int get_Allocation(int index){ return Allocation[index];}
     vector<int> get_Allocation(){return Allocation;}
     int get_PE_executing_task(int task){
        /** This one is for the HyperGraph **/
     return TaskAssignments.at(0).get_PE_executing_task(task);}

     int get_PE_executing_task(int task,int tg){
   assert( tg < num_TGs);
     return TaskAssignments.at(tg).get_PE_executing_task(task);}

     void add_PE(int new_PE_type){
   /* number of PEs is increased */
  assert( (unsigned int) new_PE_type < Allocation.size() );
  PE_type.push_back(new_PE_type);
  Allocation.at(new_PE_type)++;
    num_PEs++;
  for( int index = 0; index < num_Links; index++){
PE_Link_Connection.push_back(0);}
```

```
 }

TaskAssignment get_TaskAssignment(int tg){
 assert( tg < num_TGs);
 return TaskAssignments.at(tg);}

 /* SET UP Links */
void initilaze_link_connection();
void set_PE_connection(int, int);
void set_bridge_connection(int, int);
int get_link_connection(int PE, int Link){ return(PE_Link_Connection.at(PE*num_Links+Link));   }
int connected_to_same_bus(int,int);
int bus_connected_to_assigned_PE(int,int,TaskGraph*);

/* functions related to the objectives */
 double get_cost();
 double get_cost_PE(int);
 void set_cost(double cost_){cost = cost_;}

 double find_energy_con(vector<double>&,vector<double>&,vector<double>&,
     vector<double>&,vector<int>&,vector<vector<double> >&,vector<vector<double> >&,int);
 void set_energy_con(double EC){energy_con = EC;}
 double get_energy_con(){return energy_con;}
 double get_energy_con_PE(int);

 double find_max_deadline_violation(vector<double>&,vector<double>&);
 double find_HDV(vector<double>&,vector<double>&);
 void find_smalest_slack(vector<double>&,vector<double>&);

 double get_HDV(){return HDV;}
 double get_HDV_PE(int);
 double get_smalest_slack(){return smalest_slack;}
 double get_time(){return time;}
 double get_time_PE(int);
 void set_time(double time_){time = time_;}
 void set_HDV(double HDV_){HDV =HDV_;}

 double get_objective_value_PE(int,int);

 int mutate_bus_layout();
 int mutate_PE_bus_connections();
 int mutate_PE(double);
 void remove_unused_busses();
 void mutate_mapping_ini();
 void mutate_mapping();
 int mutate_priority_weight(int,double);
 vector<double> get_mutated_priority_weight(){return mutated_priority_weight;}
```

```
     vector<double> similarity_mesurement(individual_t*,int obj);
     double improvement_in_obj(individual_t*,int,int,int);
     void crossover(individual_t*,int,int,int);
     bool check_tasks_goes_on_PEs(vector<int>);

     vector<double> find_pre_mesurement(int);



     int scheduling_GNI(int,int,int);
     int scheduling_GNI(int,int);
     void scheduel_task(int,vector<double>&,vector<double>&,vector<double>&,TaskGraph* TG);

     double get_execution_time(int task,TaskGraph* TG,int tg){
     return ( Techs.at(PE_type.at(get_PE_executing_task(task,tg) ))->
      get_execution_time(TG->get_task_type(task)) ); }
     double get_comm_time(int task1,int task2,int bus,TaskGraph* TG){
     return Links.at(Links_indiv.at(bus))->get_link_com_time(TG->get_edge_type(task1,task2),Comm.at(0));}

     int scheduling_ISH(int);

     /* Helper functions for scheduling */
     vector<double> find_earlist_execution_time(int,vector<vector<double> >&,vector<double>&, TaskGraph* );
     double insert_task(int,int,int,int,vector<vector<int> >&,vector<vector<double> >&, TaskGraph* );
     double add_task(int,int,vector<vector<int> >&,vector<vector<double> >&,vector<double>&,TaskGraph*);
     void write_scheduel(vector<double>&,vector<double>&,vector<double>&
       ,vector<int>,vector<vector<double> >&,vector<vector<double> >&,int);
     void print_deadline_violations(vector<double>& Finish_times,vector<double>& Waiting_time,int id);


     void insert_comm(vector<vector<double> >&,vector<vector<double> >&,int,int,int,double,TaskGraph*);


     void print_info();
};/* END CLASS 'individual' */


bool check_allocation(vector<int>,int);
double drand(double);

void free_individual(individual *ind);
    void write_output_file(int);
void write_Assignments(int);
string add_id(string,int );
string add_nrun(string,int );
```

```
    double get_objective_value(int identity, int i);

    /* Helper functions for scheduling (not neading an individual */
    vector<int> find_b_levels(int);

    int update_ready_list(int,vector<int>&,vector<bool>&,TaskGraph*);
    void wirte_schedul(vector<vector<int> >,vector<vector<double> >);
    void wirte_schedul(string,vector<vector<int> >,vector<vector<double> >);
void save_archive_info(int);
void save_population_info(int,int*,int);

/*-------------------------| statemachine |-----------------------------*/


int state0();
/* Do what needs to be done in state 0.

   pre: The global variable 'paramfile' contains the name of the
        parameter file specified on the commandline.
        The global variable 'alpha' contains the number of indiviuals
        you need to generate for the initial population.

   post: Optionally read parameter specific for the module.
         Optionally do some initialization.
         Initial population created.
         Information about initial population written to the ini file
         using write_ini().
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/


int state2();
/* Do what needs to be done in state 2.

   pre: The global variable 'mu' contains the number of indiviuals
        you need to read using 'read_sel()'.
        The global variable 'lambda' contains the number of individuals
        you need to create by variation of the individuals specified the
        'sel' file.

   post: Optionally call read_arc() in order to delete old uncessary
         individuals from the global population.
         read_sel() called
         'lambda' children generated from the 'mu' parents
```

```
         Children added to the global population using add_individual().
         Information about children written to the 'var' file using
         write_var().
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/


int state4();
/* Do what needs to be done in state 4.

   pre: State 4 means the variator has to terminate.

   post: Free all memory.
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/


int state7();
/* Do what needs to be done in state 7.

   pre: State 7 means that the selector has just terminated.

   post: You probably don't need to do anything, just return 0.
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/


int state8();
/* Do what needs to be done in state 8.

   pre: State 8 means that the variator needs to reset and get ready to
        start again in state 0.

   post: Get ready to start again in state 0, this includes:
         Free all memory.
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/
```

```
int state11();
/* Do what needs to be done in state 11.

   pre: State 11 means that the selector has just reset and is ready
        to start again in state 1.

   post: You probably don't need to do anything, just return 0.
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/

int is_finished();
/* Tests if ending criterion of your algorithm applies.

   post: return value == 1 if optimization should stop
         return value == 0 if optimization should continue
*/

#endif /* VARIATOR_USER.H */
```

A.0.5 constructor.cpp

```
/*-------------------------| individual_t constructor |------------------------------*/


individual_t::individual_t(int arch){

   cost = 0;
   energy_con = 0;
   time = 0;
   HDV = 0;
   smalest_slack = 0;


  /** Allocation Part **/
  num_PEs = 0;
  for(unsigned int i = 0; i < Techs.size();i++){Allocation.push_back(0); }


  /** Setting up Link and bridges **/

  if( arch > 0){
   get_architecture(arch);
```

```
 for(int i = 0; i < num_PEs; i++){
 Allocation.at(PE_type.at(i))++;}

         if( !check_allocation(Allocation,num_TG) ){cout << " Error Bad allocation " << endl;exit(-1);}
  }else{
   cout << " error - no input architecture given " << endl; exit(-1);

    for(unsigned int i = 0; i < Allocation.size(); i++){
   for( int j = 0; j < Allocation.at(i); j++){
PE_type.push_back(i);}
  }
initilaze_link_connection();
make_com_connection_2busses(1);
  }


  /** Assignment Part **/

  TaskAssignment TA_tmp;
  num_TGs = 1; //num_TG;
  if( input_assignment){  TA_tmp.read_ARTS_assignment();
  }else{
  TA_tmp.build_hyper_graph_assignment(PE_type);}

  TaskAssignments.push_back(TA_tmp);


 mutated_priority_weight.assign( HyperGraph->get_num_taskes(),0 );
 Energy_con_on_PEs.assign(get_num_PEs(),0 );
 HDV_PE.assign(get_num_PEs(),0 );}

/*-------------------------| COPY CONSTAUCTOR |-------------------------------*/

individual_t::individual_t(individual* to_copy){

cost = to_copy->cost;
energy_con = to_copy->energy_con;
time = to_copy->time;
HDV = to_copy->HDV;
smalest_slack = to_copy->smalest_slack;

num_PEs = to_copy->num_PEs;
num_Links = to_copy->num_Links;
num_TGs = to_copy->num_TGs;


for(unsigned int i = 0; i < to_copy->Allocation.size(); i++){
```

```
   Allocation.push_back(to_copy->Allocation.at(i));}

for( int tg = 0; tg < num_TGs; tg++){
TaskAssignments.push_back(to_copy->TaskAssignments.at(tg)); }

for( int i = 0; i < num_PEs;i++){
 PE_type.push_back(to_copy->PE_type.at(i));}

for( int p = 0; p < get_num_PEs();p++){
 for( int b = 0; b < get_num_Links();b++){
  PE_Link_Connection.push_back(to_copy->PE_Link_Connection.at(p*get_num_Links()+b));
  }
}

for(int l = 0; l < num_Links; l++){
  Links_indiv.push_back(to_copy->Links_indiv.at(l));}
  assert( num_Links == (int)Links_indiv.size() );

     if( num_Links <= 0){ cout << " Errro num_Links " << num_Links << endl;exit(-1);}


for(unsigned int i = 0; i < to_copy->bus1.size();i++){
bus1.push_back(to_copy->bus1.at(i));
bus2.push_back(to_copy->bus2.at(i));}


TaskesWithDeadlineViolation = to_copy->TaskesWithDeadlineViolation;
DeadlineViolations  = to_copy->DeadlineViolations;
HDV_PE =  to_copy->HDV_PE;
mutated_priority_weight = to_copy->mutated_priority_weight;
Energy_con_on_PEs =  to_copy->Energy_con_on_PEs;}
```

A.0.6 TaskAssignment.hpp

```
#include "data.hpp"

#ifndef TaskAssignment_H
#define TaskAssignment_H

class TaskAssignment
{
    private:

      vector<int> PE_executing_task;
 //vector<int> TaskGraph_for_task;
```

```
   //vector<int> Original_task_number_for_task;


 public:
/*
 TaskAssignment(int method)
 en constructor der tager en int der angiver
 metoden TaskAssignment skal obgygges på */
   void set_PE_executing_task(int task,int PE){ PE_executing_task.at(task) = PE;}
 int get_num_Taskes(){return PE_executing_task.size();}
 int get_PE_executing_task(int task){ return PE_executing_task.at(task);}


   void initialize_PE_executing_task(int num_taskes){
   for( int task = 0; task < num_taskes; task++){
  PE_executing_task.push_back(0);} }

   void set_Assignment(int PE, int task){ PE_executing_task.at(task) = PE;    }



/*-----------------------------------------------------------------------------------------------*/

void build_hyper_graph_assignment(vector<int> PE_type){
/*******Runs through all taskgraphs:
tjecks for given assignments and spines together the TaskGraphs to the hypergraph
************************************/
unsigned int PE = 0;
initialize_PE_executing_task(HyperGraph->get_num_taskes());

 for( int task = 0; task < HyperGraph->get_num_taskes(); task++){
 int current_task = task;
  if( current_task < 0){break;}
while( Techs.at(PE_type.at(PE))->can_task_execute(HyperGraph->get_task_type(current_task)) <= 0){
PE++;
if(PE >= PE_type.size()){PE = 0;}
}
   set_PE_executing_task(current_task,PE);
    PE++;
    if(PE >= PE_type.size()){PE = 0;}
  } // END NUM TASKES
}


void read_ARTS_assignment(){

initialize_PE_executing_task(HyperGraph->get_num_taskes());
```

```
ifstream in("Assignments.in");
string str,str1,str2,tg_name;
in >> str;
in >> str;
/*in >> tg_name;
in >> str;
in >> str;*/
int pos;
int tg = -1;
int num_tgs_read = 0;
       while( in >> str && (num_tgs_read < num_TG) ){

       bool tg_found = false;
for( int tg_ = 0; tg_ < num_TG; tg_++){
 if( str.substr(1,str.size()-2) == TaskGraphs[tg_]->get_name() ){
  tg = tg_;
  num_tgs_read++;
  tg_found = true;
  in >> str;
 }
}

vector<int> tmp_assigned_PE;
while(str == "task:" && tg_found){
 in >> str;
 pos = str.find(",",0);
 str1 = str.substr(0,pos);
 str2 = str.substr(pos+1,str.size());

 in >> str;
 //set_PE_executing_task( atoi(str1.c_str())-1,atoi(str2.c_str()) );
 tmp_assigned_PE.push_back(atoi(str2.c_str()));
 }


 /* setting assignment */
 int org_task_num = -1;
 if( tmp_assigned_PE.size() ){
  for( int t = 0; t < HyperGraph->get_num_taskes(); t++){
   if( HyperGraph->get_original_task_graph(t) == tg ){
   org_task_num = HyperGraph->get_original_task_number(t);
   set_PE_executing_task( t,tmp_assigned_PE.at(org_task_num)  );}
  }
 }

}// end loop over tg's
```

```
in.close();
}

void print_info();
void print_info() const;
    void print_info(const char* );
};
#endif /* TaskAssignment.H */
```

A.0.7 variator_user.c

```
/*========================================================================

  variator for Master Thesises in co-synthesis
  Peter Kjærulff 28/4-2005

  extended from the varaitor framework from the PISA - project:

  PISA  (www.tik.ee.ethz.ch/pisa/)
  =======================================================================
  Computer Engineering (TIK)
  ETH Zurich

  file: variator_user.c
  author: Fabian Landis, flandis@ee.ethz.ch

  =======================================================================


  ========================================================================*/

using namespace std;

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>


/* tilføjede libs */
#include <iostream>
#include <string>
#include <fstream>
#include "timing.hpp"
#include "variator.h"
```

```
#include "variator_user.h"
#include "variator_internal.h"
#include "buffer.hpp"
#include "scheduling.hpp"

/*----------------------| global variables |---------------------------*/

/* declared extern in variator_user.h and used in other files as well */

char paramfile[FILE_NAME_LENGTH]; /* file with local parameters */

char *log_file = "variator_error.log";

TaskGraph* TaskGraphs[100];

TaskGraph* HyperGraph;

vector<Tech*> Techs;

vector<LINK*> Links;

vector<COMM*> Comm;

vector<int> objectives;

int obj_crossover;

double prob_mutate_bus_layout;

vector<double> MP;

double MP_mapping;

double CrossOverPropability;

bool PeriodicMapping;

bool StalicPreList;

double MotationOnPreList;

double waight_pre_meshs;

bool use_tech_buffer_spec;

vector<int> BufferSpec;
```

```
int total_num_PEs;

vector<int> copies;

bool do_test;

bool input_assignment;

int architectures;

bool build_hyperperiod;

int MaxPEs;

bool print_scheduel_info;

/*****************************************************/

/* constructor and copy_constructor in seperate file */
#include "constructor.cpp"

void individual_t::initilaze_link_connection(){
assert( PE_Link_Connection.size() == 0);
for( int index = 0; index < num_Links*num_PEs; index++){
PE_Link_Connection.push_back(0);}

unsigned int i = 0;
assert(Links_indiv.size() == 0);
for(int l = 0; l < num_Links;l++){
Links_indiv.push_back(i);i++;
if(i== Links.size()-1 ){i=0;}
}
}

void individual_t::get_architecture(int arch){

string architecure_file_in = "architecture_";
char tmp_arr[3];
sprintf(tmp_arr, "%i", arch);
architecure_file_in.append(tmp_arr);
architecure_file_in.append(".in");

ifstream in(architecure_file_in.c_str());

int numPEs_tmp = 0;
int numLinks_tmp = 0;
int numBridges_tmp = 0;
```

```
int PE_T;
string tmp,tmp1;
in >> tmp1;int k,l; k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i; tmp= tmp1.substr(k+1,l-k-1); }
 }
 numPEs_tmp = atoi(tmp.c_str());
 num_PEs = numPEs_tmp;

 for( int i = 0; i < numPEs_tmp; i++){
  in >> PE_T;
  PE_type.push_back(PE_T);}


 in >> tmp1; k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i; tmp= tmp1.substr(k+1,l-k-1); }
 }
 numLinks_tmp = atoi(tmp.c_str());
 num_Links = numLinks_tmp;

    initilaze_link_connection();

 for( int i = 0; i < numLinks_tmp; i++){
  in >>tmp1;
  in >>tmp1;
  Links_indiv.at(i) = ( atoi(tmp1.c_str()) );
  for( int p = 0; p < numPEs_tmp; p++){
  in >> tmp1;
  if( atoi(tmp1.c_str()) == 1){
  set_PE_connection(p,i);}}
  }


 in >> tmp1; k = tmp1.find( ":", 0 );
 for(unsigned int i = 0; i < tmp1.length(); i++){
  if(tmp1[i] == ',' || tmp1[i]  == ';'){
  l = i; tmp= tmp1.substr(k+1,l-k-1); }
 }
 numBridges_tmp = atoi(tmp.c_str());

 for( int b = 0; b < numBridges_tmp; b++){
   in >> tmp1;
  bus1.push_back(atoi(tmp1.c_str()));
  in >> tmp1;
```

```
  bus2.push_back(atoi(tmp1.c_str()));
  }

in.close();
}


void individual_t::make_com_connection_2busses(int shift){

vector<int> PEs_connected_to_Link;
int first_PE = 0;
for(int l = 0; l < num_Links;l++){
PEs_connected_to_Link.push_back(0);}

   for(int link = 0; link < num_Links; link++){

if( link == 2 || link == 3 || link == 4 ){first_PE = 1;}
for(int current_PE = first_PE; current_PE < num_PEs; current_PE++){
Tech* current_teck = Techs.at(PE_type[current_PE]);

if( current_teck->pins_available() > Links.at(Links_indiv.at(link))->pin_recruitment() ){
if(PEs_connected_to_Link.at(link)+1 > Links.at(Links_indiv.at(link))->get_max_users()){
break;}
else{
PEs_connected_to_Link.at(link)++;
set_PE_connection(current_PE,link);}
}
 }
first_PE = first_PE+shift;
if( first_PE >= num_PEs ){ first_PE = 0;}
}
}


/************************
bool check_allocation(vector<int> ini_PEs)
*****************************************/

bool check_allocation(vector<int> ini_PEs,int num_TGs){
/* Checks that all taskes can be executed on a PE */

bool bad_allocation = false;
for( int i = 0; i < num_TGs; i++){
for( int task = 0; task < TaskGraphs[i]->get_num_taskes(); task++){
bool can_execute = false;
for(unsigned int PE = 0; PE < ini_PEs.size(); PE++){
         if( ini_PEs.at(PE) >= 0 ){
```

```
      if( Techs.at(PE)->can_task_execute(TaskGraphs[i]->get_task_type(task)) == 1){
   can_execute = true;break;}
 }
}
if(can_execute == false){
cout << " task " << task << " " << "TG: " << i << endl;
bad_allocation = true;}
}
}
if( bad_allocation == true ){cout << " bad allocation " << endl; return false;}
return true;}

/************************
void set_PE_connection(int PE, int Link)
*****************************************/

void individual_t::set_PE_connection(int PE, int Link){
     int Busses_connected_to_PE = 0;
     for( int i = 0; i < num_Links;i++){
if(get_link_connection(PE,i) == 1){Busses_connected_to_PE++;}}
if( Busses_connected_to_PE > (int) Links.at(Links_indiv.at(Link))->get_max_users() ){
 cout << " Error limit on connections reached " << endl;
}else{
  PE_Link_Connection.at(PE*num_Links+Link) = 1;
}
}

/************************
void set_bridge_connection(int PE, int Link)
*****************************************/

void individual_t::set_bridge_connection(int PE, int Link){
     int Busses_connected_to_PE = 0;
     for( int i = 0; i < num_Links;i++){
if(get_link_connection(PE,i) == 1){Busses_connected_to_PE++;}}
if( Busses_connected_to_PE > (int) Links.at(Links_indiv.at(Link))->get_max_users() ){
 cout << " Error limit on connections reached " << endl;
}else{
  PE_Link_Connection.at(PE*num_Links+Link) = 1;
}
}
/*********************
void free_individual(individual *ind)
*********************/

void free_individual(individual *ind){delete ind;}
```

```
/************************************************/

int individual_t::connected_to_same_bus(int task1,int task2){
int link = -1;
int tg = 0;
for( int l = 0; l < num_Links; l++){
if( get_link_connection(get_PE_executing_task(task1,tg),l)
    == get_link_connection(get_PE_executing_task(task2,tg),l) ){
 link = l;}
}
return link;}

/************************************************/

int individual_t::bus_connected_to_assigned_PE(int bus,int task2,TaskGraph* TG){

   int tg = 0;
if( (get_link_connection(get_PE_executing_task(task2,tg),bus) == 1) ){
return 1;
}else{return 0;}
}

#include "functions_EA_plan.cpp"


double get_objective_value(int identity, int i)
/* Gets objective value of an individual.

   pre: 0 <= i <= dimension - 1 (dimension is the number of objectives)
7
   post: Return value == the objective value number 'i' in individual '*ind'.
         If no individual with ID 'identity' return value == -1.
*/
{
 assert(0 <= i && i < dimension); /* asserting the pre-condition */
 int j = objectives.at(i);

    double objective_value = -1.0;
    individual* tmp_individual = get_individual(identity);

        if( j == 0){
    objective_value =  tmp_individual->get_time();
}else if( j == 1){
       objective_value = tmp_individual->get_energy_con();
}else if( j == 2){
      objective_value = tmp_individual->get_cost();
}else if( j == 3){
```

```
      objective_value = tmp_individual->get_HDV();
}else{
  cout << " in deafult . error in get_obj" << endl;
  exit(-1);
}

     return (objective_value);
}

int variate(int *selected, int *result_ids)
{

     /**  vector<double> MP containes probabilities for the different mutation operators **/
     vector<double> rand_doubles;
     for(int i = i = 0; i < mu; i++){
      if( get_individual(selected[i])->get_time() > 0 || Iteration == 1 || MP.at(0) == 1){
      rand_doubles.push_back(drand(1));
      }else{
      rand_doubles.push_back(drand(1-MP.at(0))+MP.at(0));
      }
     }

/* for( unsigned int t = 0; t < rand_doubles.size(); t++){
  cout << rand_doubles.at(t) << " ";}
  cout << endl;
string ter; cin >> ter;*/

     /* Select x individuals from mu for crossover */
     vector<int> picked_for_crossover;
     picked_for_crossover.assign(mu,0);

     int x = 0;
     for( int i = 0; i < mu; i++){
     if((MP.at(0)+MP.at(1)+MP.at(2)+MP.at(3)+MP.at(4))
       <=rand_doubles.at(i)&& rand_doubles.at(i) <(MP.at(0)+MP.at(1)+MP.at(2)+MP.at(3)+MP.at(4)+MP.at(5))){
     /*  prob PM.at(5) */

      picked_for_crossover.at(i) = 1;}
     }
     /* Find y individuals from mu for donation |y| = |x| */
     vector<individual_t> donators;

     for( int i = 0; i < mu;i++){
      if( picked_for_crossover.at(i) == 1 ){
       /* find indiv with greatest improvement and positive simm_mesurement amung all mu individuals*/

       int best_match_index = -1;
```

```
       double largest_simm_mesurement = 0;
       vector<double> simm_mess_PE_pair;
       for( int j = 0; j < mu; j++){
              simm_mess_PE_pair =
       get_individual(selected[i])->similarity_mesurement(get_individual(selected[j]),obj_crossover);
              if( simm_mess_PE_pair.at(2) > largest_simm_mesurement){
        largest_simm_mesurement = simm_mess_PE_pair.at(2);
best_match_index = j;
       }
       }

       if( best_match_index < 0 ){
        picked_for_crossover.at(i) = 0;}
       else{
       individual_t best_match = individual_t(get_individual(selected[best_match_index]));
       donators.push_back(best_match);}
}
      }



     /* simm_mess_PE_pair - vector that stores the for the PE paris */
    vector<double> simm_mess_PE_pair;
    int donater_index = 0;

     /* copying all individuals from selected */
     for(int i = 0; i < mu; i++){

     cout << " variating indiv : " << i << " of " << mu << endl;

     individual_t* tmp_ind = get_individual(selected[i]);
     individual_t* tmp_indiv = new individual_t(tmp_ind);



     if( picked_for_crossover.at(i) == 0 ){


        int mutation_succeded = tmp_indiv->mutate_PE(rand_doubles.at(i));

tmp_indiv->remove_unused_busses();


mutation_succeded = tmp_indiv->mutate_priority_weight(100,rand_doubles.at(i));

if( (mutation_succeded == 0 && MP.at(0) != 1.0)||(MP.at(0)+MP.at(1)+MP.at(2)+MP.at(3)+MP.at(4)+MP.at(5))
  <=rand_doubles.at(i)&&rand_doubles.at(i)<(MP.at(0)+MP.at(1)+MP.at(2)+MP.at(3)+MP.at(4)+MP.at(5)+MP.at(6) ) ){
```

```
 tmp_indiv->mutate_mapping();}



    }else{
/** enten vælges et andet individ til cross over tilfældigt
eller også vælges det der har PE med størst simm_mesurement,
og forbedring mht objectiv **/


cout << " cross over " << endl;
simm_mess_PE_pair = tmp_indiv->similarity_mesurement(&donators.at(donater_index),obj_crossover);



if( simm_mess_PE_pair.at(2) > 0){

assert(tmp_indiv->improvement_in_obj(&donators.at(donater_index),obj_crossover,
 (int)simm_mess_PE_pair.at(1),(int)simm_mess_PE_pair.at(2))>0);

tmp_indiv->crossover(&donators.at(donater_index),
  obj_crossover,(int)simm_mess_PE_pair.at(1),(int)simm_mess_PE_pair.at(2));

}else{ cout << " error in donator list - crossover" <<endl;exit(-1);}

donater_index++;
  }/** end cross over **/


/* chekt that all taskes is mapped correctly if debug mode */
#ifndef NDEBUG
for( int t = 0; t < HyperGraph->get_num_taskes();t++){
assert(Techs.at(tmp_indiv->get_PE_type(
 tmp_indiv->get_PE_executing_task(t)))->can_task_execute(HyperGraph->get_task_type(t))==1);}
#endif

result_ids[i] = add_individual(tmp_indiv);
          if(result_ids[i] == -1){
               log_to_file(log_file, __FILE__, __LINE__,
                           "copying + adding failed");
               return (1);
          }
     }
cout << " done variating " <<  mu << endl;
return 0;
}
```

```
/*-------------------------| statemachine functions |-------------------*/

int state0()
/* Do what needs to be done in state 0.

   pre: The global variable 'paramfile' contains the name of the
        parameter file specified on the commandline.
        The global variable 'alpha' contains the number of indiviuals
        you need to generate for the initial population.

   post: Optionally read parameter specific for the module.
         Optionally do some initialization.
         Initial population created.
         Information about initial population written to the ini file
         using write_ini().
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/
{
     clear_iteration_info();
     int result; /* stores return values of called functions */
     int *initial_population; /* storing the IDs of the individuals */
     initial_population = (int *) malloc(alpha * sizeof(int));
     if (initial_population == NULL)
     {
          log_to_file(log_file, __FILE__, __LINE__, "variator out of memory");
          return (1);
     }

     #include "read_parameters.cpp"
     srand(seed);
     /* Read TaskGraph and Technology files*/
     Iteration = 1;
     Start_Time_Stamp = gettimestamp();


     /* create first alpha individuals */
     int arch = 0;
     for(int i = 0; i < alpha; i++){
     cout << " alpha:" << i << endl;
     if( architectures > 0 ){
      arch++;
      if(arch > architectures ){arch = 1;}}

      individual_t* tt;
      tt = new individual_t(arch);
```

```
      tt->mutate_priority_weight(100,0); /* (100,0) -> mutate all priorities */

if( i == 0 ){

}else{

 int rand_num_mutations = rand() % 5;
 while( rand_num_mutations > 0){
  tt->mutate_PE(drand(1-MotationOnPreList)+MotationOnPreList);
  tt->mutate_priority_weight(100,0); /* (100,0) -> mutate all priorities */
     tt->mutate_mapping_ini();

  rand_num_mutations--;}
}

tt->set_cost(100000);
tt->set_energy_con(1000000000);
tt->set_time(100000);


#ifndef NDEBUG
for( int t = 0; t < HyperGraph->get_num_taskes();t++){
if( Techs.at(tt->get_PE_type(
 tt->get_PE_executing_task(t)))->can_task_execute(HyperGraph->get_task_type(t))==0){
cout << " task " << t << " org t" << HyperGraph->get_original_task_number(t) <<  endl;}
assert(Techs.at(tt->get_PE_type(
 tt->get_PE_executing_task(t)))->can_task_execute(HyperGraph->get_task_type(t))==1);
}
#endif

scheduling* sch = new scheduling(tt);
  sch->scheduling_GNI(0,0);
delete sch;
initial_population[i] = add_individual(tt);

        if(initial_population[i] == -1){
              return(1);}

     }
  result = write_ini(initial_population);
  cout << " ini written " << endl;

     if (result != 0)
     {
          log_to_file(log_file, __FILE__, __LINE__,
                      "couldn't write ini");
          free(initial_population);
```

```
          return (1);
     }

     free(initial_population);
     return (0);

}


int state2()
/* Do what needs to be done in state 2.

   pre: The global variable 'mu' contains the number of indiviuals
        you need to read using 'read_sel()'.
        The global variable 'lambda' contains the number of individuals
        you need to create by variation of the individuals specified the
        'sel' file.

   post: Optionally call read_arc() in order to delete old uncessary
         individuals from the global population.
         read_sel() called
         'lambda' children generated from the 'mu' parents
         Children added to the global population using add_individual().
         Information about children written to the 'var' file using
         write_var().
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/
{
     int *parent_identities, *offspring_identities; /* array of identities */
     int result; /* stores return values of called functions */

     parent_identities = (int *) malloc(mu * sizeof(int));
     if (parent_identities == NULL)
     {
          log_to_file(log_file, __FILE__, __LINE__, "variator out of memory");
          return (1);
     }

     offspring_identities = (int *) malloc(lambda * sizeof(int));
     if (offspring_identities == NULL)
     {
          log_to_file(log_file, __FILE__, __LINE__, "variator out of memory");
          return (1);
     }
```

```
     result = read_sel(parent_identities);

     if (result != 0) /* if some file reading error occurs, return 2 */
          return (2);

     result = read_arc();
     if (result != 0) /* if some file reading error occurs, return 2 */
          return (2);

     save_archive_info(Iteration);
     write_iteration_info();
  result = variate(parent_identities, offspring_identities);
write_iteration_info("  done variating ");


for( int i = 0; i < lambda; i++){

string str_to_ite_info = "   -scheduling indiv #";
char tmp_arr[3];
sprintf(tmp_arr, "%i", i );
str_to_ite_info.append(tmp_arr);

write_iteration_info(str_to_ite_info);


        /*get_individual(offspring_identities[i])->print_info();*/
scheduling sch(get_individual(offspring_identities[i]));
sch.scheduling_GNI(0,0);


}

    save_population_info(Iteration,offspring_identities,lambda);
write_iteration_info("  done scheduling ");
Iteration++;

     if(result != 0)
          return (1);


    /******** END VARIATE *******/

     result = write_var(offspring_identities);
```

```
     if (result != 0)
     {
          log_to_file(log_file, __FILE__, __LINE__,
                      "couldn't write var");
          free(offspring_identities);
          free(parent_identities);
          return (1);
     }

     free(offspring_identities);

     free(parent_identities);



  write_iteration_info("  Out Of State 2 ");
  return (0);
}


int state4()
/* Do what needs to be done in state 4.

   pre: State 4 means the variator has to terminate.

   post: Free all memory.
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/
{     /**** Here comes your code if you need to do anything before
            terminating. E.g. call some kind of output function, which
            informs about the results of the optimization run. */
     int result;
     result = read_arc();

     if (0 == result) /* arc file correctly read
                         this means it was not read before,
                         e.g., in a reset. */
     {

/* Writing output informatiuon */
  ofstream out("TimeAndIters.out",ios::trunc);
  out << "Iteration > MaxIterations" << Iteration <<">"<< MaxIterations << endl;
  double Current_Time_Stamp = gettimestamp();
      out << "Runing_time >  Execution_Time ";
  out <<  Current_Time_Stamp-Start_Time_Stamp  << " > "  << Execution_Time << endl;
```

```
  time_t time2;
          time(&time2);
  out << " elapse time sec, min " << difftime(time2,time1);
  out <<" , " << difftime(time2,time1)/60 << endl;
  out.close();
  write_output_file(run_num);
  write_Assignments(run_num);

/* this par is only wiritten if print_scheduel_info is set =1 in parameter file */
  int current_id;
  current_id = get_first();

while(current_id >= 0 && print_scheduel_info){
  individual *temp2 = get_individual(current_id);
  scheduling* sch2 = new scheduling(temp2);
  sch2->scheduling_GNI(0,0,current_id);
  delete sch2;
  current_id = get_next(current_id);
  }
   }/* end if arc correctly read */


   /* clear parameters for new optimization run...*/


return (0);}


int state7()
/* Do what needs to be done in state 7.

   pre: State 7 means that the selector has just terminated.

   post: You probably don't need to do anything, just return 0.
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/
{
     return(0);
}


int state8()
/* Do what needs to be done in state 8.

   pre: State 8 means that the variator needs to reset and get ready to
```

```
        start again in state 0.

   post: Get ready to start again in state 0.
         Return value == 0 if successful,
                      == 1 if unspecified errors happened,
                      == 2 if file reading failed.
*/
{
     /**** Here comes your code if you need to do anything before
           resetting. */
   int result;
   Iteration = 1;

   result = read_arc();

     if(0 == result) /* arc file correctly read
                        this means it was not read before,
                        e.g., in a reset. */
     {

/* Writing output informatiuon */
  ofstream out("TimeAndIters.out",ios::trunc);
  out << "Iteration > MaxIterations" << Iteration <<">"<< MaxIterations << endl;
  double Current_Time_Stamp = gettimestamp();
      out << "Runing_time >  Execution_Time " <<
  Current_Time_Stamp-Start_Time_Stamp;
  out  << " > "  << Execution_Time << endl;
  time_t time2;
          time(&time2);
  out << " elapse time sec, min " << difftime(time2,time1) <<" , " << difftime(time2,time1)/60 << endl;
  out.close();
  write_output_file(run_num);
  write_Assignments(run_num);
  run_num++;
  /* this part is only wiritten if print_scheduel_info is set =1 in parameter file */
  int current_id;
  current_id = get_first();

while(current_id >= 0 && print_scheduel_info){
  individual *temp2 = get_individual(current_id);
  scheduling* sch2 = new scheduling(temp2);
  sch2->scheduling_GNI(0,0,current_id);
  delete sch2;
  current_id = get_next(current_id);
  }
   }/* end if arc correctly read */
```

```
     return (0);
}


int state11(){return (0);}


int is_finished()
/* Tests if ending criterion of your algorithm applies.

   post: return value == 1 if optimization should stop
         return value == 0 if optimization should continue
*/
{    cout << " is_finished() " << endl;
     int return_value = 0;
     /**** Here comes your code for checking the termination criteria. */

    double Current_Time_Stamp = gettimestamp();


     if( Iteration >= MaxIterations || (Current_Time_Stamp-Start_Time_Stamp >  Execution_Time) ){

     return_value = 1;

  }


     return return_value;
}

void write_output_file(int num_run)
{

     string fname = "FinalPopulation.out";
     fname = add_nrun(fname,num_run);
     ofstream out(fname.c_str(),ios::trunc);
     int j, current_id;
     individual *temp;

  for (j = 0; j < dimension; j++)
  {  out << objectives.at(j) << " ";}
  out << endl;

     current_id = get_first();
     while (current_id != -1){

  temp = get_individual(current_id);
```

```
  for (j = 0; j < dimension; j++){
      out << get_objective_value(current_id, j) << " ";
  }

     out << endl;
  current_id = get_next(current_id);
     }

     out.close();
}



void write_Assignments(int num_run)
{
  ofstream out("FinalAssignment.out",ios::trunc);
  out.close();
     unsigned int j;
     int current_id;
     individual *temp;

     current_id = get_first();

     while (current_id != -1)
     {
  temp = get_individual(current_id);
  ofstream out("FinalAssignment.out",ios::app);
  out << " id : " << current_id << endl;

  out << " Cost: " << temp->get_cost() << endl;
          out << " Energy : " << temp->get_energy_con() << endl;
        out << " Time : " << temp->get_time() << endl;
  out << " HDV : " << temp->get_HDV() << endl;
  out << " smallest slack " << temp->get_smalest_slack() << endl;

 for( int PE = 0; PE < temp->get_num_PEs(); PE++){
 out << PE <<": PE type:" << temp->get_PE_type(PE) << endl;}
   out << endl;

  out << " Allocation : " ;
  for( j = 0; j < temp->get_Allocation().size(); j++){
             out << temp->get_Allocation(j) << " ";}
     out << endl;

  out << " PE_type : ";
  for (j = 0; j < (unsigned int)temp->get_num_PEs(); j++){
             out << temp->get_PE_type(j) << " ";}
```

```
     out << endl;

out << " buffer use: " << endl;
out << " 'PE bus : buff_Size_in,buff_Size_out,buff_Size_Shared' " << endl;
vector<double> buff_use = temp->get_buffer_use();
int t = 0;
for(unsigned int i = 0; i < buff_use.size(); i++){
 out << buff_use.at(i) << " ";
 if( t == 4){out << endl; t = 0;}else{t++;}
}
out << endl;
out.close();

  for( int tg = 0; tg < temp->get_num_TGs(); tg++){
   temp->get_TaskAssignment(tg).print_info("FinalAssignment.out");}


       {ofstream out("FinalAssignment.out",ios::app);
        out << " Link PE connection: " << endl;
  out << " 'id , type< connections.... '" << endl;
  for(int i = 0; i < temp->get_num_Links(); i++){
   out << "L" << i << " " << temp->get_Link_Type(i) << " ";

   for( int p = 0; p < temp->get_num_PEs(); p++){
    out << temp->get_link_connection(p,i) << " ";}
     out << endl;
  }

out << " Bridges: " << endl;
  out << " BUSX <-connects-> BUSY " << endl;
 for(unsigned int b = 0; b < temp->bus1.size();b++){
   out << temp->bus1.at(b) << "<-->" << temp->bus2.at(b) << endl;}
  out << endl;
out.close();}

current_id = get_next(current_id);

     }

     out.close();
}

string add_id(string filename_root,int current_id){

string tmp;
tmp.append("_id_");
char tmp_arr[3];
```

```
sprintf(tmp_arr, "%i", current_id);
tmp.append(tmp_arr);
unsigned int loc = filename_root.find( ".", 0 );
filename_root.insert(loc,tmp);

return filename_root;
 }

string add_nrun(string filename_root,int num_run){

string tmp;
tmp.append("_run_");
char tmp_arr[3];
sprintf(tmp_arr, "%i",num_run);
tmp.append(tmp_arr);
unsigned int loc = filename_root.find( ".", 0 );
filename_root.insert(loc,tmp);

return filename_root;
 }

string add_tg(string filename_root,int tg){

string tmp;
tmp.append("_tg_");
char tmp_arr[3];
sprintf(tmp_arr, "%i", tg);
tmp.append(tmp_arr);
unsigned int loc = filename_root.find( ".", 0 );
filename_root.insert(loc,tmp);

return filename_root;
 }
#include "scheduling.cpp"
```

A.0.8 functions_EA_plan.cpp

```
/*****************
mutate_PE
: schwiches a rand chousen PE to another PE type
NOTE PIN recuriment is currently not conserned
*******************/
/** Functions in this file:
void individual_t::mutate_PE(vector<double> MP)
void individual_t::mutate_mapping_ini()
```

```
void individual_t::mutate_mapping()
void individual_t::mutate_priority_weight()
bool individual_t::check_tasks_goes_on_PEs(vector<int> PE_types)
double individual_t::similarity_mesurement(individual_t* )
double individual_t::improvement_in_obj(individual_t*,int obj )
double individual_t::get_cost()
double individual_t::get_cost_PE(int)
double individual_t::find_energy_con(vector<double>& Start_times,vector
  <double>& Finish_times, vector<double>& Waiting_time,vector<double>& Waiting_time_in,
   vector<int>& SchedulingSeq,vector<vector<double> >& Sending_times,vector<vector<double> >& Receiving_times,
int current_id)
double individual_t::get_energy_con_PE(int PE)
double individual_t::find_max_deadline_violation(vector<double>& Finish_times,vector<double>& Waiting_time){
double individual_t::get_time_PE(int PE)
void individual_t::print_deadline_violations(vector<double>& Finish_times,vector<double>& Waiting_time,int id)
double individual_t::get_objective_value_PE(int i,int PE)
void individual_t::print_info()
vector<int> individual_t::sort_taskes(int HyperGraph__)

**/
int individual_t::mutate_bus_layout(){
/** adds one more bus to the system - and moves a random number of PE
to the new bus **/
 double randd= drand(1);
 int mutation_succeeded = 0;

if( randd <= 1 && get_num_PEs() >= 4 ){

  int random_int = rand();
int link_to_connect_new_bus_to = random_int % get_num_Links();
random_int = rand();
int new_type = random_int % Links.size();


/** copies old PE connection **/
    vector<vector<int> > old_connection;
        for( int PE = 0; PE < get_num_PEs(); PE++){
  vector<int> tt;old_connection.push_back(tt);
   for( int b = 0; b < get_num_Links(); b++){
     old_connection.at(PE).push_back(get_link_connection(PE,b));
        }
         }

       vector<int> move_PE_to_new_bus;
       move_PE_to_new_bus.assign(get_num_PEs(),0);

       int _1th_PE_to_move = rand()%get_num_PEs();
```

```
       move_PE_to_new_bus.at(_1th_PE_to_move) = 1;
       int _2nd_PE_to_move = _1th_PE_to_move;
       int i = 0;
       while( _2nd_PE_to_move == _1th_PE_to_move && i < 10000){
              _2nd_PE_to_move = rand()%get_num_PEs();i++;

/** Der skal også tjekkes på antallet af usere på bussen **/

       if( i >= 10000 ){ cout << " error in mutate bus_layout " << endl; break;}}

       move_PE_to_new_bus.at(_2nd_PE_to_move) = 1;
       PE_Link_Connection.resize(PE_Link_Connection.size()+get_num_PEs(), 0);


for( int index = 0; index < (num_Links+1)*(get_num_PEs()); index++){
PE_Link_Connection.at(index) = 0;}

 assert( (int) PE_Link_Connection.size() == (num_Links+1)*(get_num_PEs()));

/** Adding a bus more **/
num_Links++;

Links_indiv.push_back(new_type);
bus1.push_back(link_to_connect_new_bus_to);
   bus2.push_back(num_Links-1);
assert(num_Links-1 != link_to_connect_new_bus_to);


/** Setting Link Connections **/
for(unsigned int PE = 0; PE < PE_type.size(); PE++){
  if( move_PE_to_new_bus.at(PE) ){
 set_PE_connection(PE,num_Links-1);

 }else{
  for(unsigned int b = 0; b < old_connection.at(PE).size(); b++){
     if( old_connection.at(PE).at(b) > 0 ){
   set_PE_connection(PE,b);
 assert( get_link_connection(PE,b)) ; }
  }
         }
}


      /* control on all PE is connected */
      for( int p = 0; p < get_num_PEs();p++){ int t = 0;
       for( int i = 0; i < get_num_Links();i++){
        if( get_link_connection(p,i) ){ t++;}}
```

```
assert(t > 0);}
}
return mutation_succeeded;
}

/***************************************
int individual_t::mutate_PE_bus_connections(){
***************************************/

void individual_t::remove_unused_busses(){
 /** is a bus has only one bridge connected and no PE's it must be removed**/
/** this method only removed 1 buss and 1 bridge - but poccibly more times **/
 vector<int> unused_busses;

 for(int b = 0; b < get_num_Links();b++){
  int num_PEs_connected = 0;
  int num_bridges_connected = 0;
   for( int p = 0; p < get_num_PEs(); p++){
    if( get_link_connection(p,b) ){
    num_PEs_connected++;} }

    for( int bridge = 0; bridge < bus1.size(); bridge++){
     if( b == bus1.at(bridge) || b == bus2.at(bridge) ){
      num_bridges_connected++;}}

    if( num_PEs_connected == 0 && num_bridges_connected == 1){
     unused_busses.push_back(b);}  }

  for( int b = 0; b < unused_busses.size(); b++){
   int bus_to_remove = unused_busses.at(b)-b;

   /** copies old PE connection **/
    vector<vector<int> > old_connection;
        for( int PE = 0; PE < get_num_PEs(); PE++){
  vector<int> tt;old_connection.push_back(tt);
   for( int b = 0; b < get_num_Links(); b++){
     old_connection.at(PE).push_back(get_link_connection(PE,b));
        }
         }
       PE_Link_Connection.resize(PE_Link_Connection.size()-get_num_PEs(), 0);


for( int index = 0; index < (num_Links-1)*(get_num_PEs()); index++){
PE_Link_Connection.at(index) = 0;}

 assert( (int) PE_Link_Connection.size() == (num_Links-1)*(get_num_PEs()));
```

```
/** Removing bus **/
num_Links--;
vector<int>:: iterator iter = Links_indiv.begin();
Links_indiv.erase(iter+bus_to_remove);

for( int bridge = 0; bridge < bus1.size(); bridge++){
 if( bus_to_remove != bus1.at(bridge) && bus_to_remove != bus2.at(bridge) ){
  if( bus1.at(bridge) > bus_to_remove){ bus1.at(bridge)--;}
     if( bus2.at(bridge) > bus_to_remove){ bus2.at(bridge)--;} }
 }

vector<int>:: iterator iter1 = bus1.begin();
vector<int>:: iterator iter2 = bus2.begin();

for( int bridge = 0; bridge < bus1.size(); bridge++){
  if( bus_to_remove == bus1.at(bridge) || bus_to_remove == bus2.at(bridge) ){
           bus1.erase(iter1);
    bus2.erase(iter2);
    break;}
     iter1++;
     iter2++;
    }


/** Setting Link<->PE Connections **/
for(unsigned int PE = 0; PE < PE_type.size(); PE++){
  for(unsigned int bb = 0; bb < old_connection.at(PE).size();bb++){
    if( old_connection.at(PE).at(bb) == 1){
     if( bb > bus_to_remove ){
   set_PE_connection(PE,bb-1);}
     else{set_PE_connection(PE,bb);} /* her nå den være mindre */
   }
  }
        }

        // tjkker alle PE'er er connectede
   for( int p = 0; p < get_num_PEs();p++){ int t = 0;
          for( int i = 0; i < get_num_Links();i++){
           if( get_link_connection(p,i) ){ t++;}}
     assert(t > 0);
     }
} /* end for num unused busses */
}

/****************************************
int individual_t::mutate_arch(double)
```

```
****************************************/


int individual_t::mutate_PE(double rand_num){

 int mutation_succeeded = 0;
 if( rand_num <= MP.at(0) ){

 /* mutate mutated waight list */

 }else if( MP.at(0) <= rand_num && rand_num < (MP.at(0)+MP.at(1)) ){

  /* change PE */
mutation_succeeded = 1;
   int random_int = rand();
int index_on_PE = -1;
index_on_PE  = random_int % get_num_PEs();


int new_PE_type =  rand() % (int)Techs.size() ;

   Allocation.at(PE_type.at(index_on_PE))--;
int old_PE_type = PE_type.at(index_on_PE);

PE_type.at(index_on_PE) = new_PE_type;
    Allocation.at(PE_type.at(index_on_PE))++;

for(unsigned int tg = 0; tg < TaskAssignments.size(); tg++){
 for( int task = 0; task < HyperGraph->get_num_taskes() && mutation_succeeded == 1;task++){
  if(Techs.at(new_PE_type)->can_task_execute(HyperGraph->get_task_type(task))== 1){
  }else{
   /* Schwiching back to old PE type */
   /* tag evt en tilfeldig GPP */
         Allocation.at(old_PE_type)++;
      PE_type.at(index_on_PE) = old_PE_type;
     Allocation.at(new_PE_type)--;
 mutation_succeeded = 0;
         break;}
}
  } /* end for TaskAssignments.size() */


  for( int t = 0; t < HyperGraph->get_num_taskes();t++){
  assert(Techs.at(get_PE_type(get_PE_executing_task(t)))->can_task_execute(HyperGraph->get_task_type(t))==1);}
```

```
 }else if( (MP.at(0)+MP.at(1)) <= rand_num  && rand_num< (MP.at(0)+MP.at(1)+MP.at(2))  ){
 /** Remove a PE **/
mutation_succeeded = 1;

if( get_num_PEs() == 1 ){mutation_succeeded = 0;}
else{ int PE_to_remove = rand() % get_num_PEs();
vector<int> PE_type_copy = PE_type;
vector<int>::iterator theIterator = PE_type_copy.begin();
theIterator = theIterator+PE_to_remove;
  PE_type_copy.erase(theIterator);

if(check_tasks_goes_on_PEs(PE_type_copy)&& Allocation.at(PE_type.at(PE_to_remove)) > 0 ){

   Allocation.at(PE_type.at(PE_to_remove))--;
       vector<vector<int> > old_connection;

   for( int PE = 0; PE < get_num_PEs(); PE++){
   vector<int> tt;old_connection.push_back(tt);
    for( int b = 0; b < get_num_Links(); b++){
      old_connection.at(PE).push_back(get_link_connection(PE,b));
         }
           }

    PE_Link_Connection.erase(PE_Link_Connection.end()-num_Links,PE_Link_Connection.end() );
   for( int index = 0; index < num_Links*(get_num_PEs()-1); index++){
PE_Link_Connection.at(index) = 0;}
 assert( (int) PE_Link_Connection.size() == num_Links*(get_num_PEs()-1));

           vector<int> old_PE_executing_task; /* ændre navnet til new PE_index */


           for(unsigned int tg = 0; tg < TaskAssignments.size(); tg++){
         int task = 0;
    int PE_index = 0;
    int PE_type_at_index = PE_type_copy.at(PE_index);
    vector<bool> allready_moved;
    allready_moved.assign(HyperGraph->get_num_taskes()+1, false );


      while( task < HyperGraph->get_num_taskes()){
   assert( PE_type_copy.size() == (unsigned int)get_num_PEs()-1);

old_PE_executing_task.push_back(get_PE_executing_task(task,tg));

if( get_PE_executing_task(task,tg) >= PE_to_remove ){
 if( get_PE_executing_task(task,tg) == PE_to_remove ){
```

```
    while( ( Techs.at(PE_type_at_index)->can_task_execute(HyperGraph->get_task_type(task)) <= 0 )  ){
  if( PE_index < (int)(PE_type_copy.size()-1) ){PE_index++;}
  else{ PE_index = 0;}
     PE_type_at_index = PE_type_copy.at(PE_index);}

  assert(PE_index >= 0);
     assert( Techs.at(PE_type_at_index)->can_task_execute(HyperGraph->get_task_type(task)) > 0);
  if( PeriodicMapping == 0){
   TaskAssignments.at(tg).set_PE_executing_task(task,PE_index);}
  else{
   int tg_of_task_to_mutate = HyperGraph->get_original_task_graph(task);
     int task_to_move = HyperGraph->get_original_task_number(task);

 if( !allready_moved.at(task) ){

  for( int ii = 0; ii <  TaskAssignments.at(0).get_num_Taskes(); ii++){

   if( HyperGraph->get_original_task_graph(ii) == tg_of_task_to_mutate
     && HyperGraph->get_original_task_number(ii) == task_to_move ){
     allready_moved.at(ii) = true;

    PE_type_at_index = PE_type_copy.at(PE_index);
    assert( Techs.at(PE_type_at_index)->can_task_execute(HyperGraph->get_task_type(task)) );
    TaskAssignments.at(0).set_PE_executing_task(ii,PE_index);}
       }
 }
  }/* end else */
  PE_index++;
  if( PE_index >= get_num_PEs()-1){PE_index = 0;}
   /* distribute taskes to other PE's */
  PE_type_at_index = PE_type_copy.at(PE_index);
  task++;

 }else{
  /** PE_types 0 2 2 3 X 1 3 5
                       | <-- removed
        'task' assigned to PE after X */

 if( PeriodicMapping == 0){ TaskAssignments.at(tg).set_PE_executing_task(task,old_PE_executing_task.at(task)-1)
 else{
if( !allready_moved.at(task) ){
          allready_moved.at(task) = true;
      TaskAssignments.at(tg).set_PE_executing_task(task,old_PE_executing_task.at(task)-1);
       }
  }
```

```
  task++;}
}else{ task++;} // allready_moved.at(task) = true;

     } /* end while( task < NumTaskes */
}

/*
for( int ii = 0; ii <  TaskAssignments.at(0).get_num_Taskes(); ii++){
if(  TaskAssignments.at(0).get_PE_executing_task(ii) > get_num_PEs()-1){
  string ter; cin >> ter;}}
*/
 for(unsigned int PE = 0; PE < PE_type.size(); PE++){
 int new_PE_index = PE;
  if( PE != (unsigned int) PE_to_remove ){
   if( PE > (unsigned int) PE_to_remove ){ new_PE_index = PE-1;}

   for(unsigned int b = 0; b < old_connection.at(PE).size(); b++){
     if( old_connection.at(PE).at(b) > 0 ){
   set_PE_connection(new_PE_index,b);
 assert( get_link_connection(new_PE_index,b)) ; }
  }
 }
        }

       set_num_PEs(get_num_PEs()-1);
       PE_type = PE_type_copy;

      /* tjkker alle PE'er er connectede */
      for( int p = 0; p < get_num_PEs();p++){ int t = 0;
       for( int i = 0; i < get_num_Links();i++){
        if( get_link_connection(p,i) ){ t++;}}
      assert(t>0);}
  }/* end if numPEs > 1 */

 }
 }else if(  (MP.at(0)+MP.at(1)+MP.at(2)) <= rand_num && rand_num< (MP.at(0)+MP.at(1)+MP.at(2)+MP.at(3))  ){
  /** Add a PE **/
 if( get_num_PEs() < MaxPEs ){

/** Finde new PE type and index **/
mutation_succeeded = 1;
int random_int = rand();
int new_PE_type  = random_int % Techs.size();
int new_PE_index = get_num_PEs();
```

```
/** Make a singel connection to a random chosen bus **/
random_int = rand();
random_int = random_int % get_num_Links();
int connections = 0;

for( int PE = 0; PE < get_num_PEs();PE++){
 if( get_link_connection(PE,random_int) > 0 ){
  connections++;}}

if( connections > Links.at(Links_indiv.at(random_int))->get_max_users() ){
 mutation_succeeded = 0;
}else{

add_PE(new_PE_type);
set_PE_connection(new_PE_index,random_int);

for(unsigned int tg = 0; tg < TaskAssignments.size(); tg++){
 for( int i = 0; i < TaskAssignments.at(tg).get_num_Taskes();i = i+get_num_PEs()){
   /* perhaps choose taskes to shwitch randomly */


  assert(i < TaskAssignments.at(tg).get_num_Taskes()  );
int task = i;assert(task >= 0);



if(Techs.at(new_PE_type)->can_task_execute(HyperGraph->get_task_type(task))==1.0 ){

 if( PeriodicMapping == 0){TaskAssignments.at(tg).set_PE_executing_task(task,new_PE_index);
 cout << " error only peridic mapping " << endl;exit(-1);
 }else{
  int tg_of_task_to_mutate = HyperGraph->get_original_task_graph(task);
  int task_to_move = HyperGraph->get_original_task_number(task);
  for( int ii = 0; ii <  TaskAssignments.at(0).get_num_Taskes(); ii++){
   if( HyperGraph->get_original_task_graph(ii) == tg_of_task_to_mutate
    && HyperGraph->get_original_task_number(ii) == task_to_move ){
assert(Techs.at(new_PE_type)->can_task_execute(HyperGraph->get_task_type(task))== 1);
assert(Techs.at(new_PE_type)->can_task_execute(HyperGraph->get_task_type(ii))==1);
TaskAssignments.at(0).set_PE_executing_task(ii,new_PE_index);}
    }
  }
     }
}
       }
  } /* end else - if sufficient with pins */

 // set_num_PEs(get_num_PEs()+1);
```

```
 }

 }else if( (MP.at(0)+MP.at(1)+MP.at(2)+MP.at(3)) <=
  rand_num && rand_num<(MP.at(0)+MP.at(1)+MP.at(2)+MP.at(3)+MP.at(4))  ){

 mutation_succeeded = mutate_bus_layout();


 }/* end elseif */

if((MP.at(0)+MP.at(1)+MP.at(2)+MP.at(3)+MP.at(4))<=rand_num &&rand_num<=
(MP.at(0)+MP.at(1)+MP.at(2)+MP.at(3)+MP.at(4)+MP.at(5)) ){
/* crossover */
}



if( mutation_succeeded ){
 HDV_PE.assign(get_num_PEs(),0 );
 Energy_con_on_PEs.assign(get_num_PEs(),0 );}
 return mutation_succeeded;
 } /* end Mutate_PEs() */

 /************************
ini mutation
************************/
void individual_t::mutate_mapping_ini(){

int task_to_mutate = rand() % HyperGraph->get_num_taskes();
int tg_of_task_to_mutate = HyperGraph->get_original_task_graph(task_to_mutate);
int number_of_taskes_to_mutate = rand() % HyperGraph->get_num_org_taskes();
int new_PE_exe_task = rand() % get_num_PEs();

vector<bool> all_ready_mutated;
all_ready_mutated.assign( HyperGraph->get_num_taskes(),false);


 int count = 0;
 while( count <= number_of_taskes_to_mutate && get_num_PEs() > 1 ){

  if( (int) (Techs.at(PE_type.at(new_PE_exe_task))->
can_task_execute(HyperGraph->get_task_type(task_to_mutate))) == 1 ){

 for( int i = 0; i <  TaskAssignments.at(0).get_num_Taskes(); i++){
 if( !all_ready_mutated.at(i) ){
  if( HyperGraph->get_original_task_graph(i) == tg_of_task_to_mutate &&
  HyperGraph->get_original_task_number(i) == HyperGraph->get_original_task_number(task_to_mutate)){
```

```
  TaskAssignments.at(0).set_PE_executing_task(i,new_PE_exe_task);
  all_ready_mutated.at(i) = true;
  count++;
  }
 }
}
       }
   task_to_mutate = rand() % HyperGraph->get_num_taskes();

   new_PE_exe_task = rand() % get_num_PEs();

   if( get_PE_executing_task(task_to_mutate) == new_PE_exe_task){ /* assuring we mapp to a new PE */
    if( get_PE_executing_task(task_to_mutate) >= 1 ){
      new_PE_exe_task--;
    }else{
    if( get_PE_executing_task(task_to_mutate) < get_num_PEs() ){
      new_PE_exe_task++;}}
      count++;
   }
   tg_of_task_to_mutate = HyperGraph->get_original_task_graph(task_to_mutate);

   }// end while


 }


/*************************************************************
int individual_t::mutate_mapping()
*************************************************************/

void individual_t::mutate_mapping(){

double rand_num = drand(1);


if( rand_num <= 1 ){  //MP_mapping
 if( PeriodicMapping == 0){


  int number_of_taskes_to_mutate = rand()% HyperGraph->get_num_taskes();
  int count = 0;
  int task_to_mutate = -1;
  int new_PE_exe_task = -1;
   while( count < number_of_taskes_to_mutate && get_num_PEs() > 1 ){
```

```
// mutation_succeeded = 1;
task_to_mutate  = rand() % HyperGraph->get_num_taskes();
  new_PE_exe_task = rand() % get_num_PEs();

if( (int) (Techs.at(PE_type.at(new_PE_exe_task))->
can_task_execute(HyperGraph->get_task_type(task_to_mutate))) == 1 ){
  TaskAssignments.at(0).set_PE_executing_task(task_to_mutate,new_PE_exe_task);}
  count++;
   }/* end while */


 }else{

  vector<int> TaskesWDV = get_TasksWDV();

  /* find task graphs with DV */
  vector<int> ViolationsOnTg;
  for( unsigned int tg = 0; tg < copies.size();tg++){
   ViolationsOnTg.push_back(0);}

   /* find PE's with DV */
  vector<int> ViolationsOnPE;
  for(  int p = 0; p < get_num_PEs();p++){
   ViolationsOnPE.push_back(0);}

  for( unsigned int i = 0; i < TaskesWDV.size(); i++){
  ViolationsOnTg.at(HyperGraph->get_original_task_graph(TaskesWDV.at(i)))++;
  ViolationsOnPE.at(get_PE_executing_task(TaskesWDV.at(i)))++;}

  vector<int> PEs_with_no_DV;
  int min_num_DV = 100;
  int PE_with_min_num_DV = 0;
  for( int p = 0; p < get_num_PEs();p++){
   if(  ViolationsOnPE.at(p) == 0){
    PEs_with_no_DV.push_back(p);}
   if( ViolationsOnPE.at(p) < min_num_DV){
min_num_DV = ViolationsOnPE.at(p);
PE_with_min_num_DV = p;}
   }

  int number_of_taskes_to_mutate = 200000; //rand()% 4+1 ; /* 0 1 2 3*/
  if( number_of_taskes_to_mutate > (int) TaskesWDV.size()-1 ){
   number_of_taskes_to_mutate = TaskesWDV.size()-1;}

  int count = 0;
  int task_to_mutate = -1;
  int tg_of_task_to_mutate = -1;
```

```
  int new_PE_exe_task = -1;
  if( PEs_with_no_DV.size() > 0 && TaskesWDV.size() > 0  && get_num_PEs() > 1){
   while( count < number_of_taskes_to_mutate ){

if( PEs_with_no_DV.size() > 0){
   new_PE_exe_task = rand() % PEs_with_no_DV.size();
 new_PE_exe_task = PEs_with_no_DV.at(new_PE_exe_task);
}else{
 new_PE_exe_task = PE_with_min_num_DV;}

tg_of_task_to_mutate = HyperGraph->get_original_task_graph(TaskesWDV.at(count));



int diff = TaskGraphs[tg_of_task_to_mutate]->
get_num_taskes()-HyperGraph->get_original_task_number(TaskesWDV.at(count));
int rand_int = rand() % (TaskGraphs[tg_of_task_to_mutate]->get_num_taskes()-diff);

task_to_mutate = TaskesWDV.at(count) - rand_int;

int count2 = 0;


if( (int) (Techs.at(PE_type.at(new_PE_exe_task))->
can_task_execute(HyperGraph->get_task_type(task_to_mutate))) == 1 ){
         assert( (Techs.at(PE_type.at(new_PE_exe_task))->
  can_task_execute(HyperGraph->get_task_type(task_to_mutate))) == 1 );
 for( int i = 0; i <  TaskAssignments.at(0).get_num_Taskes(); i++){
  if( HyperGraph->get_original_task_graph(i) == tg_of_task_to_mutate &&
   HyperGraph->get_original_task_number(i) == HyperGraph->get_original_task_number(task_to_mutate)){
     TaskAssignments.at(0).set_PE_executing_task(i,new_PE_exe_task);}
 }

}

  count++;
   }/* end while */

  }else{  /* use randomisid change of mapping on alle taskes */
   mutate_mapping_ini(); }

 }/* end else ( not periodic )*/
}/* end if picked for mutation */


}
```

```
/*****************
void individual_t::mutate_priority_weight()
*********************************************/
int individual_t::mutate_priority_weight(int num,double randd){

 if( randd <= MP.at(0) ){ //MotationOnPreList



/* vector<int> TaskesWDV = get_TasksWDV();
 int number_of_taskes_to_mutate = rand()% 4+1 ; //  1 2 3 4
 if( number_of_taskes_to_mutate > (int) TaskesWDV.size()-1 ){
 number_of_taskes_to_mutate = TaskesWDV.size()-1;}*/



/* int number_of_taskes_to_mutate = (HyperGraph->get_num_taskes());
*/

 double t = get_time();
 double HP = HyperGraph->get_HP();

 cout << t << " " << HP << endl;
 cout << t/HP << endl;
 double fraq = 0;
 if( t/(2*HP) >= 1){ fraq = 1;}
 else{fraq = t/HP;}
 if( num == 100 ){fraq = 1;} /* user demands full mutation (initial  pop )*/


//(int)HyperGraph->get_num_taskes()*fraq
 int number_of_taskes_to_mutate = rand() % (int)HyperGraph->get_num_taskes()*(fraq);
 int tg_of_task_to_mutate = -1;
 int rand_int = -1;
 int task_to_mutate = -1;

 vector<bool> all_ready_mutated;
 all_ready_mutated.assign( HyperGraph->get_num_taskes(),false);
 int count = 0;
 while( count < number_of_taskes_to_mutate ){


     task_to_mutate = rand() % (HyperGraph->get_num_taskes());


 /* movement_in_pre_list */
```

```
 mutated_priority_weight.at(task_to_mutate) = rand() % 101;
 all_ready_mutated.at(task_to_mutate) = true;
 count++;

 }// end while


}
return 1;}/* end mutate priority_weight */

/************************
check_taskes_goes_on_PEs
************************/

bool individual_t::check_tasks_goes_on_PEs(vector<int> PE_types){

  bool can_task_execute = false;
  for(unsigned int tg = 0; tg < TaskAssignments.size(); tg++){
for( int task = 0; task < HyperGraph->get_num_taskes();task++){

can_task_execute = false;
for(unsigned int i = 0; i < PE_types.size();i++){
 if( Techs.at(PE_types.at(i))->can_task_execute(HyperGraph->get_task_type(task)) ){
  can_task_execute = true;}
}
if(can_task_execute == false){break;}
}
  }
return can_task_execute;}

/*****************
similarity_mesurement(individual_t*,int obj )
*******************/
vector<double> individual_t::similarity_mesurement(individual_t* ind2,int obj){
/** findes the simm_mesurement between this and ind2, only if the
donator ind2 improves the objective value is the sommmesurement stored**/

vector<int> PE_ind1;
vector<int> PE_ind2;
vector<int> sim_mess;

vector<double> result;

int num_shared_taskes = 0;
int max_simm_mess = 0;
bool task_does_not_go_on_new_PE = false;
```

```
for( int p1 = 0; p1 < get_num_PEs(); p1++){
 for(int p2 = 0; p2 < ind2->get_num_PEs();p2++){

 task_does_not_go_on_new_PE = false;
  num_shared_taskes = 0;
  for( int task = 0; task < HyperGraph->get_num_taskes(); task++ ){
   if( get_PE_executing_task(task) == p1){
    if(Techs.at(ind2->get_PE_type(p2))->can_task_execute(HyperGraph->get_task_type(task))==1){
    // ind2->get_PE_executing_task(task)
     if(  ind2->get_PE_executing_task(task) == p2 ){
     num_shared_taskes++;}
     }else{task_does_not_go_on_new_PE = true;}
   }
  }

   if( improvement_in_obj(ind2,obj,p1,p2) > 0 && task_does_not_go_on_new_PE == false){
 vector<int> PE_types_after_crossover = PE_type;
 PE_types_after_crossover.at(p1) = ind2->get_PE_type(p2);
 if( check_tasks_goes_on_PEs(PE_types_after_crossover) ){
   PE_ind1.push_back(p1);
       PE_ind2.push_back(p2);
   sim_mess.push_back(num_shared_taskes);}
}
   if( num_shared_taskes > max_simm_mess ){max_simm_mess = num_shared_taskes;}
  }
 }


  /* findes largest simm with improvement **/

for(unsigned int i = 0;i < PE_ind1.size();i++){
improvement_in_obj(ind2,obj,PE_ind1.at(i),PE_ind2.at(i));
if( sim_mess.at(i) == max_simm_mess ){
 result.push_back(sim_mess.at(i));
 result.push_back(PE_ind1.at(i));
 result.push_back(PE_ind2.at(i));
 //break;
 }
}
if(result.size() == 0){
 result.assign(3,0);}

return result;}

/****************
double improvement_in_obj(individual_t*,int ,int,int)
  ************************/
```

```
double individual_t::improvement_in_obj( individual_t* ind2,int obj,int PE1,int PE2 ){
 double obj1 = get_objective_value_PE(obj,PE1);
 double obj2 = ind2->get_objective_value_PE(obj,PE2);
return (obj1-obj2);}

/*****************
crossover(int PE)
*******************/
void individual_t::crossover(individual_t* ind2,int obj,int PE1,int PE2 ){
/** this* individual must be improved, ind2 is donater
    PE1 is por preforming PE inf this individual nad PE2 is well
    performing PE in ind **/

assert(improvement_in_obj(ind2,obj,PE1,PE2)> 0);

/** ændre PE type til donatorens PE type **/
int old_PE_type = ind2->get_PE_type(PE2);
Allocation.at(PE_type.at(PE1))--;
PE_type.at(PE1) = ind2->get_PE_type(PE2);
Allocation.at(PE_type.at(PE1))++;

vector<double> mpw_tmp = ind2->get_mutated_priority_weight();

int num_taskes_not_good_for_mutation = 0;

if( check_tasks_goes_on_PEs(PE_type) ){
 /** løbe taskes igennem og flytte de taskes der ikke kan udføres på den nye type **/
 for(int task = 0; task < HyperGraph->get_num_taskes(); task++){

if( ind2->get_PE_executing_task(task) == PE2 || get_PE_executing_task(task) == PE1){

if(Techs.at(PE_type.at(PE1))->can_task_execute(HyperGraph->get_task_type(task))== 1){
       TaskAssignments.at(0).set_PE_executing_task(task,PE1);
mutated_priority_weight.at(task) = mpw_tmp.at(task);
   }else{
    num_taskes_not_good_for_mutation++;
    for(int p = 0; p < get_num_PEs(); p++){
  if(Techs.at(PE_type.at(PE1))->can_task_execute(HyperGraph->get_task_type(task)==1)){
 get_TaskAssignment(0).set_PE_executing_task(task,p);}
     }
      }
  }
 } /** for all taskes **/
 }else{
 cout << " Error crossover could not succed " << endl; exit(-1);
 PE_type.at(PE1) = old_PE_type;}
```

```
}



/*****************
get_cost
*******************/

double individual_t::get_cost(){
 /** PEs **/
 cost = 0;
/* for(unsigned int j = 0; j < Allocation.size();j++){
  if( Allocation.at(j) > 0){
  for( int t = 0; t < Allocation.at(j);t++){ */
  for(int p = 0; p < get_num_PEs();p++){
    cost += Techs.at(PE_type.at(p))->get_cost(); }

/**Links**/
for( int i = 0; i < get_num_Links(); i++){
cost += Links.at(Links_indiv.at(i))->get_cost();}
     return cost;
}

/*****************
get_cost_PE(int)
*******************/

double individual_t::get_cost_PE(int PE){
return (Techs.at(PE_type.at(PE)))->get_cost(); }

/*****************
get_cost_energi_con
*******************/
double individual_t::find_energy_con(vector<double>& Start_times,vector<double>& Finish_times,
vector<double>& Waiting_time,vector<double>& Waiting_time_in,vector<int>& SchedulingSeq,
  vector<vector<double> >& Sending_times,vector<vector<double> >& Receiving_times,int current_id){

  int PE1,PE2;
  double comm_interval = -1;
  energy_con = 0;

  Energy_con_on_PEs.assign( get_num_PEs(),0 );

  for(unsigned int PE_index = 0; PE_index < PE_type.size();PE_index++){  // loop PEs /drop denne !!!!
 for(int task = 0; task < HyperGraph->get_num_taskes();task++){
  if( (unsigned int) get_PE_executing_task(task) == PE_index ){
            int task_type = HyperGraph->get_task_type(HyperGraph->get_original_task_number(task));
```

```
          energy_con += Techs.at(PE_type.at(get_PE_executing_task(task)))->get_dyn_power(task_type);
    Energy_con_on_PEs.at(PE_index) += Techs.at(PE_type.at(get_PE_executing_task(task)))->get_dyn_power(task_typ

    for( int task2 = task; task2 < HyperGraph->get_num_taskes();task2++){

    /* send and receive PE's energyconsumption under communikation */
     if( HyperGraph->get_edge_type(task,task2) >= 0 ){
         energy_con += Techs.at(PE_type.at(get_PE_executing_task(task)))->get_commPower();
              energy_con += Techs.at(PE_type.at(get_PE_executing_task(task2)))->get_commPower();}

  }
 }
 }
/* Static Power con */
energy_con += Techs.at(PE_type.at(PE_index))->get_StPwr();
}
// Communication task Consumption on 1) bus 2) on Sending/Recieving PE //

/* 1) Bus */
/* static*/
if( get_num_PEs() > 1){
for( int l = 0; l < get_num_Links(); l++){
 if( Receiving_times.at(l).size() > 0){
energy_con = energy_con+(Links.at(Links_indiv.at(l))->get_StPwr());}
  }
}

/* 2) Sending/Recieving PE + dyn on bus*/
/* energi forbrugt for konmmunikations af buffere og PE's antags at
  være det samme og at være extra oven i det konstante forbrug */

  for(unsigned int i = 0; i < Sending_times.size();i++){

   for(unsigned int j = 0; j < Sending_times.at(i).size();j=j+2){


double comm_pwr = Links.at(Links_indiv.at(i))->
get_comm_PowerCon(HyperGraph->get_edge_type((int)Sending_times.at(i).at(j+1),
(int)Receiving_times.at(i).at(j+1)),Comm.at(0));
energy_con += comm_pwr;

}
  }

return energy_con;}

/*****************
```

```
get_max_deadline_violation
*******************/

double individual_t::find_max_deadline_violation(vector<double>& Finish_times,vector<double>& Waiting_time){

reset_TaskWDV();

double largest_deadline_violation = 0;
double sum_of_deadline_violations = 0;
for( int t = 0; t < HyperGraph->get_num_taskes(); t++){
       if( HyperGraph->get_deadline(t) > 0 ){
if( HyperGraph->get_deadline(t) < (Finish_times.at(t)-Waiting_time.at(t))){
 sum_of_deadline_violations += (Finish_times.at(t)-Waiting_time.at(t)) - HyperGraph->get_deadline(t);
 set_TaskWDV(t,(Finish_times.at(t)-Waiting_time.at(t)) - HyperGraph->get_deadline(t));
if( Finish_times.at(t) - HyperGraph->get_deadline(t) > largest_deadline_violation){
largest_deadline_violation = Finish_times.at(t) - HyperGraph->get_deadline(t);
}
}
       }
}
//time = largest_deadline_violation;
time = sum_of_deadline_violations;
return time;
}

/*****************
get_time_PE(int PE)
*******************/

double individual_t::get_time_PE(int PE){
double sum_time = 0;
vector<int> taskes_WDV = get_TasksWDV();
vector<double> DW = get_DVs();
for(unsigned int task = 0; task < taskes_WDV.size() ;task++){
 if( get_PE_executing_task(taskes_WDV.at(task)) == PE ){
  sum_time += DW.at(task);}
  }
return sum_time;}


/*****************
get_cost_energi_con_PE(int)
*******************/
double individual_t::get_energy_con_PE(int PE){return Energy_con_on_PEs.at(PE);}

/*****************
get_HDV()
```

```
********************/

double individual::find_HDV(vector<double>& Finish_times,vector<double>& Waiting_time){

double HDV = 0;
vector<int> taskes_WDV = get_TasksWDV();
HDV_PE.assign(get_num_PEs(),0);
for(unsigned int i = 0; i < taskes_WDV.size() ;i++){
   int task = taskes_WDV.at(i);
   if( (Finish_times.at(task)-Waiting_time.at(task))  > HyperGraph->get_HP()){
   HDV +=(Finish_times.at(task)-Waiting_time.at(task))-HyperGraph->get_HP() ;

   if( (Finish_times.at(task)-Waiting_time.at(task))-HyperGraph->get_HP()
    > HDV_PE.at(get_PE_executing_task(task)) ){
   HDV_PE.at(get_PE_executing_task(task)) =
    Finish_times.at(task)-Waiting_time.at(task)-HyperGraph->get_HP();}


   }
  }
return HDV;}


/*****************
find_smalest_slack(vector<double>& ,vector<double>&)
********************/

void individual::find_smalest_slack(vector<double>& Finish_times,vector<double>& Waiting_time){

smalest_slack = 10000000;
for(unsigned int task = 0;task <HyperGraph->get_num_taskes() ;task++){
      if(  HyperGraph->get_HP() - (Finish_times.at(task)-Waiting_time.at(task)) < smalest_slack){
    smalest_slack = HyperGraph->get_HP() - (Finish_times.at(task)-Waiting_time.at(task));}
  }
}


/*****************
get_HDV_PE(int)
********************/

double individual::get_HDV_PE(int PE){
 return HDV_PE.at(PE);}



void individual_t::print_deadline_violations(
```

```
vector<double>& Finish_times,vector<double>& Waiting_time,int id){


string out_str = "deadline_violations.out";
out_str = add_id(out_str,id);
ofstream out(out_str.c_str());

double largest_deadline_violation = 0;
double sum_of_deadline_violations = 0;
for( int t = 0; t < HyperGraph->get_num_taskes(); t++){
       if( HyperGraph->get_deadline(t) > 0 ){
         out << t <<" " <<(Finish_times.at(t)-Waiting_time.at(t)) <<"-"<< HyperGraph->get_deadline(t)<<"=";
 out << (Finish_times.at(t)-Waiting_time.at(t)) - HyperGraph->get_deadline(t) << endl;
       }
}
out.close();}


/*****************
get_objective_value_PE(int obj,int PE)
*******************/

double individual_t::get_objective_value_PE(int i,int PE)
/* Gets a PEs contribution to the obejvive value */
{

    int j = objectives.at(i);
    double objective_value = -1.0;

     if( j == 0){
    objective_value = get_time_PE(PE);
}else if( j == 1){
       objective_value = get_energy_con_PE(PE);
}else if( j == 2){
    objective_value = get_cost_PE(PE);
}else if( j == 3){
    objective_value = get_HDV_PE(PE);
}else{
  cout << " in deafult . error in get_obj_PE" << endl;
  exit(-1);
}

return (objective_value);}


/**********
save_archive_info(Iteration,offspring_identities,lambda);
```

```
*****************************************************************/
void save_archive_info(int Iteration){

  ofstream out("Population_Info.out",ios::app);
  FILE *fp; fp = fopen(arc_file, "r");
  assert(fp != NULL);

  int size,current;
  fscanf(fp, "%d", &size);

  out << Iteration << " " << size << " " << 0 << " " << 0 << endl;

  for(int dim = 0; dim < dimension; dim++){
   out << objectives.at(dim) << " ";
   double min = 10000000;
double max = 0;
double sum = 0;
double val = 0;
current = get_first();
   for( int i = 0; i < size; i++){
    val = get_objective_value(current,dim);
assert(val >= 0);
sum += val;
if( val > max ){max = val;}
if( val < min ){min = val;}
current = get_next(current);
}
out << min << " ";
double avage = sum/size;
out << avage << " " << max << endl;

  }
fclose(fp);
out.close();
 }


/**********
save_population_info(Iteration,offspring_identities,lambda);

*****************************************************************/
void save_population_info(int Iteration,int* identities,int size){

  ofstream out("Population_Info.out",ios::app);

  out << Iteration << " " << 0 << " " << 0 << " " << 0 << endl;
```

```
  for(int dim = 0; dim < dimension; dim++){
   out << objectives.at(dim) << " ";
   double min = 10000000;
double max = 0;
double sum = 0;
double val = 0;
   for( int i = 0; i < size; i++){
    val = get_objective_value(identities[i],dim);
sum += val;
assert(val >= 0);
if( val > max ){max = val;}
if( val < min ){min = val;}
}
out << min << " ";
double avage = sum/size;
out << avage << " " << max << endl;

  }
out.close();
 }

/************************
Print files individual and TaskAssignment
************************/

void individual_t::print_info(){


ofstream out_file1("individual_t.out",ios::trunc);

out_file1<< " Price " << get_cost() <<" " << cost << endl;
out_file1<< " Energy_con " << energy_con << endl;
out_file1<< " Time " << time << endl;
out_file1<< " Hard DV " << get_HDV() << endl;
out_file1<< " Smalest Slack " << get_smalest_slack() << endl << endl;

for(int i = 0; i < get_num_PEs();i++){
out_file1 << i <<": PE type:" << PE_type.at(i) <<" ";
 for( int j = 0; j < dimension; j++){
  out_file1 << " ob("<<j<<"): "<<get_objective_value_PE(j,i) << " ";}
out_file1 << endl;}
out_file1 <<" PEs written" << endl;
out_file1.close();

 for( int tg = 0; tg < 1; tg++){ TaskAssignments.at(tg).print_info("individual_t.out");}
```

```
ofstream out_file2("individual_t.out",ios::app);
out_file2 << "Bus/PE connections" << endl;
assert( ((int) Links_indiv.size()) == num_Links);
for( int i = 0; i < num_Links;i++){
out_file2 << "L" << i << " ";
 for( int PE = 0; PE < num_PEs; PE++){
  out_file2 << get_link_connection(PE,i) << " ";}
  out_file2 << Links.at(Links_indiv.at(i))->get_name() <<
    " " << Links.at(Links_indiv.at(i))->get_cost() << endl;}
  out_file2 << endl;


        out_file2 << "Bridges:" << endl;
 for(unsigned int b = 0; b < bus1.size();b++){
   out_file2 << "B" << b << " " << bus1.at(b) << " " <<  bus2.at(b) << endl;}
 out_file2 << endl;
out_file2.close();
}


/**********************
Printing a task Assignment To a file 'file_mane'
***************************************/

void TaskAssignment::print_info(const char* file_name){


ofstream out_file( file_name,ios::app);
int t = 0;
for(unsigned int i = 0; i < PE_executing_task.size();i++){
if( t < PE_executing_task.at(i) ){ t = PE_executing_task.at(i);}
}

for(unsigned int i = 0; i < PE_executing_task.size();i++){
out_file <<PE_executing_task.at(i) << " ";}
out_file << endl;

vector<vector<int> > tmp;
vector<int> tt;

for( int i = 0; i <= t; i++){
 tmp.push_back(tt);}

for(unsigned int i = 0; i < PE_executing_task.size();i++){
 tmp.at(PE_executing_task.at(i)).push_back(i);}

out_file << "Assignment printed without seq" << endl;
```

```
for(unsigned int i = 0; i <tmp.size();i++){

if( tmp.at(i).size() > 0 ){
  for(unsigned int j = 0; j < tmp.at(i).size();j++){
 out_file << tmp.at(i).at(j) << " "; }
}else{
   out_file << " -- empty -- "; }
 out_file << endl;}
 out_file << endl;

out_file.close();}

/*******************************
Functions Related to ItterationInfo.out
*******************************/

void clear_iteration_info(){
ofstream iteration_info("iteration_info.out",ios::trunc);
iteration_info.close();}

void write_iteration_info(){

ofstream iteration_info("iteration_info.out",ios::app);

iteration_info << " Iteration : " << Iteration << endl;
iteration_info.close();}

void write_iteration_info(string tmp){

ofstream iteration_info("iteration_info.out",ios::app);
iteration_info << tmp << endl;
iteration_info.close(); }

double drand(double range){
     double j;
     j=(range * (double) rand() / (RAND_MAX + 1.0));
     return (j);}




/***********************************************
SORT TASKES on b_levels - on a individual obj
***********************************************/

vector<double> individual_t::find_pre_mesurement(int HyperGraph__){
/** Longets path - to exit node **/
```

```
TaskAssignment TA = TaskAssignments.at(0);

vector<int> Sorted_Taskes1;
vector<double> b_levels;
vector<int> copies_ = copies;

vector<int> num_Taskes_in_TG;
int Max_Num_Copies = 0;
for( int i = 0; i < num_TG; i++){
 num_Taskes_in_TG.push_back(TaskGraphs[i]->get_num_taskes());
 if( Max_Num_Copies < copies.at(i) ){
  Max_Num_Copies = copies.at(i);}}

vector<double> ttt;
for( int i = 0; i < HyperGraph->get_num_taskes(); i++){
 ttt.push_back(0);}


int bus_used_for_comm_time = 0;

b_levels.insert(b_levels.begin(),HyperGraph->get_num_taskes()+1,0);

// calculate_b_level
{ // scope for b-level temps temps
bool sink_node = true;
for(int i = HyperGraph->get_num_taskes()-1; i >= 0; i--){
double max = 0;
sink_node = true;

for( int child = 0; child < HyperGraph->get_num_taskes();child++){
if( HyperGraph->get_edge_type(i,child) >= 0){
if( get_execution_time(child,HyperGraph,0)+
get_comm_time(i,child,bus_used_for_comm_time,HyperGraph) + b_levels.at(child) > max){

max = get_execution_time(child,HyperGraph,0)+
  get_comm_time(i,child,bus_used_for_comm_time,HyperGraph) + b_levels.at(child);
sink_node = false;}

}
}

if( sink_node == true ){ max = get_execution_time(i,HyperGraph,0);}

b_levels.at(i) = max;
```

```
ttt.at(i) = max;

double mod_b_level = b_levels.at(i);

    if( copies.at(HyperGraph->get_original_task_graph(i)) > 1 ){
     if( copies_.at(HyperGraph->get_original_task_graph(i)) > 0 ){
        mod_b_level += Max_Num_Copies-copies_.at(HyperGraph->get_oliste af taskes der harriginal_task_graph(i));
 ttt.at(i) += Max_Num_Copies-copies_.at(HyperGraph->get_original_task_graph(i));
         num_Taskes_in_TG.at( HyperGraph->get_original_task_graph(i))--;

 if( num_Taskes_in_TG.at( HyperGraph->get_original_task_graph(i)) == 0){
  copies_.at(HyperGraph->get_original_task_graph(i))--;
  num_Taskes_in_TG.at( HyperGraph->get_original_task_graph(i)) =
   TaskGraphs[HyperGraph->get_original_task_graph(i)]->get_num_taskes();}
      }
     }
 vector<int>::iterator iter_begin = Sorted_Taskes1.begin();
 vector<int>::iterator iter_end = Sorted_Taskes1.end();
 int ii =0; bool inserted = false;
       for( ; iter_begin != iter_end; iter_begin++){
   if( mod_b_level > ttt.at(Sorted_Taskes1.at(ii)) ){
    Sorted_Taskes1.insert( iter_begin, 1, i );
    inserted = true;break;}
    ii++;}
          if( inserted == false ){ Sorted_Taskes1.push_back(i);}
}
}
Sorted_Taskes1.push_back(-1);
ttt.push_back(-1);
return ttt;}
```

A.0.9 scheduling.hpp

```
#include "variator_user.h"

using namespace std;
#include <list>
#include <iterator>

class scheduling{

private:
individual_t* ind;

vector<int> Sorted_Taskes;
```

```
vector<int> successor; /* liste af taskes der har en afhæninghed til slut knuden i en foregående task graph */
vector<bool> is_there_dep;
vector<bool> Ready_List;

vector<bool> Task_Executed;
vector<int> ChildesBackForScheduling;
vector<double> Latest_Finish_time;

/* Start Finish times for taskes on PE ( including waiting time for for BUS comm )*/
vector<double> Start_times;
vector<double> Finish_times;

/* Waiting time before a PE sends to BUS */
vector<double> Waiting_time;
vector<double> Waiting_time_in;
vector<int> SchedulingSeq;

vector<list<int> > comm_task_pos;
vector< list<int>::iterator> poss_in_list;
vector<vector<double> > Send_Times;
vector<vector<double> > Receive_Times;
vector<double> last_comm_event;

double try_insertion(int bus,int p,double new_Send_Time,double
comm_time,vector<bool>& bus_ready,vector<double>& min_times);

void set_comm_task(int assigned_to_link, int from,int to,double min_Send_Time, double comm_time);

/* Nye hurtigere data-strukturer - muliggør opslag **/
vector<vector<int> > task_to_task_connection;
vector<vector<int> > last_bus_on_path;

/*--------------------------------*--------------------------------*/

void set_last_bus_on_path(int task1,int task2,int bus2){
 task_to_task_connection.at(task1).push_back(task2);
 last_bus_on_path.at(task1).push_back(bus2);}

int get_last_bus_on_path(int task1,int task2);

bool ReceiveBuff;
bool SendBuff;
bool SharedBuff;
vector<buffer> buffers;
vector<vector<int> >buffer_index;

int get_num_PEs(){return ind->get_num_PEs();}
```

```
int get_num_Links(){return ind->get_num_Links();}

int get_PE_executing_task(int i,int HyperGraph_){ '
return ind->get_PE_executing_task(i,HyperGraph_);}
int get_PE_executing_task(int i){ return ind->get_PE_executing_task(i);}

int bus_connected_to_assigned_PE(int bus,int t1 ,TaskGraph* TG){
return ind->bus_connected_to_assigned_PE(bus,t1,TG);}
int bus_connected_to_assigned_PE(int bus,int t1){
return ind->bus_connected_to_assigned_PE(bus,t1,HyperGraph);}

buffer* get_buffer(int task,int bus){ int PE = get_PE_executing_task(task);
 assert( ind->get_link_connection(PE,bus) );
         assert( buffer_index.at(PE).at(bus) > 0);return (&buffers.at(buffer_index.at(PE).at(bus)-1 ));}

double get_execution_time(int task,TaskGraph* HyperGraph,int HyperGraph_){
return ind->get_execution_time(task,HyperGraph,HyperGraph_);}
double get_execution_time(int task){ return ind->get_execution_time(task,HyperGraph,0);}

double get_comm_time(int i,int task,int bus,TaskGraph* HyperGraph){
assert(bus>= 0);return ind->get_comm_time(i,task,bus,HyperGraph);}
double get_comm_time(int i,int task,int bus){ assert(bus>= 0);
return ind->get_comm_time(i,task,bus,HyperGraph);}


vector<double> weight_priority_listes(double waight,vector<double> pre_mes1, vector<double> pre_mes2);
vector<int> make_pre_list(vector<double> pre_mesurements);
int update_ready_list(int current_task,TaskGraph* HyperGraph);
  int find_start_task(vector<int>);

bool is_there_dependencies_to_predecessor(int task){

return is_there_dep.at(task);/*
bool dep = false;
cout << " task " << task << " " << successor.size() << endl;
string ter; cin >> ter;
vector<int>::iterator i1 = successor.begin();
vector<int>::iterator i2 = successor.end();
for(int i = 0 ; i1 != i2 && i < task; i1++,i++){
if( *i1 == task ){dep =
true;*//*successor.erase(i1);*//*break;}}
return dep;*/}

bool is_predecessors_scheduled(int task);

int scheduel_task(int task);
vector<int> sort_taskes2();
```

```
void get_comm_time3(vector<double>& start_times,int i,int task);
double set_comm_time3(int i,int task,int HyperGraph);
vector<vector<double> > get_comm_time_lists(int,int,int);

public:
scheduling(individual_t* tt);

void* scheduling_GNI(void*);/* rutine used for parallazing*/
int scheduling_GNI(int task_graph,int HyperGraph);/* old rutine before paralazing */
int scheduling_GNI(int task_graph,int HyperGraph,int current_id);/* rutine that prints scheduel */
double scheduel_comm(double,double,int,int,int,vector<double> &Latest_Finish_time );
double scheduel_comm2(double min_Send_Time,double Send_Time,
double comm_time,int i,int task, vector<double>& Latest_Finish_time);

double min_Send_Time_considering_input_buffers(double,int,int,int,vector<double> &Latest_Finish_time );
double min_Send_Time_considering_output_buffers(double min_Send_Time,int task1,int bus);
void write_scheduel(int current_id);


/** Test funktioner **/
vector<double> finde_comm_start_slut(int,int);
double min_Send_Time_over_path(vector<double>,int,int,double);
double set_comm_over_path(vector<vector<double>>,int,int,vector<double>,double,vector<double>);

//void test_scheduel(int);
};
```

A.0.10 scheduling.cpp

```
/* Functions for scheduling of a taskes graph given an assignment/individual */

#include <stdio.h>
#include "variator_user.h"

/***********************
Constructor
***********************/

scheduling::scheduling(individual_t* tt){
ReceiveBuff = false;
SendBuff = true;
SharedBuff = false;
ind = tt;
```

```
successor = HyperGraph->get_dependencies_to_predecessor_list();

for( int t = 0; t < HyperGraph->get_num_taskes(); t++){
 is_there_dep.push_back(false);}
for(unsigned int t = 0; t < successor.size();t++){
  is_there_dep.at(successor.at(t)) = true;}



        int index = 1;
list<int>::iterator iter; //  = NULL;
iter = NULL;
for(int PE = 0; PE < get_num_PEs();PE++){
vector<int> tmp;
buffer_index.push_back(tmp);
 for(int b = 0; b < get_num_Links();b++){
    buffer_index.at(PE).push_back(0);
  poss_in_list.push_back(iter); /* inserting place for comm task */
  if( ind->get_link_connection(PE,b) > 0 ){
   buffer_index.at(PE).at(b) = index;
   index++;

   if( use_tech_buffer_spec ){ /* use buffer sizes specified in the tech file */
     int variable_size = 0;
     if( Techs.at(ind->get_PE_type(PE))->get_CommMem() > 0 ){variable_size = BufferSpec.at(5);}
    buffer tmp(PE,b,0,0,(int)Techs.at(ind->get_PE_type(PE))->get_CommMem(),0,0,variable_size);
/*PE_,bus_,size_Send_,size_Recive_,size_Shared_,
var_size_Send_,var_size_Recive_,var_size_Shared_) */
buffers.push_back(tmp);
}else{
    buffer tmp(PE,b,BufferSpec.at(0),BufferSpec.at(1),
BufferSpec.at(2),BufferSpec.at(3),BufferSpec.at(4),BufferSpec.at(5));
buffers.push_back(tmp);}


   }
  }
 }
}


/**********************
Greedy Non Inserting Heuristik for Scheduling
**********************/
/*
void* scheduling_GNI(void* sch){
```

```
scheduling* tmp = (scheduling*) sch;
tmp->scheduling_GNI(0,0,-1);
void* t = NULL;
return t;}
*/

int scheduling::scheduling_GNI(int t,int r){scheduling_GNI(0,t,-1);return 0;}



/***********************
Greedy Non Inserting Heuristik for Scheduling
***********************/

int scheduling::scheduling_GNI(int task_graph,int going_out,int current_id){

{
 vector<double> pre_mesurements = ind->find_pre_mesurement(task_graph);
 pre_mesurements =  weight_priority_listes(waight_pre_meshs,
  pre_mesurements,ind->get_mutated_priority_weight()  );
 Sorted_Taskes = make_pre_list(pre_mesurements);
}

for( int i = 0; i < HyperGraph->get_num_taskes(); i++){
Ready_List.push_back(false);
Task_Executed.push_back(false);
}

for(int i = 0; i < HyperGraph->get_num_sources(); i++){
 Ready_List.at(HyperGraph->get_source_node(i)) = true;}

vector<int> tmp2;
for( int i = 0; i < HyperGraph->get_num_taskes();i++){

last_bus_on_path.push_back(tmp2);
task_to_task_connection.push_back(tmp2);
Waiting_time.push_back(0);
Waiting_time_in.push_back(0);
}

vector<double> tmp;
list<int> tmp_list;
for(int l = 0; l < get_num_Links(); l++){
comm_task_pos.push_back(tmp_list);
Send_Times.push_back(tmp);
Receive_Times.push_back(tmp);
last_comm_event.push_back(0);}
```

```
for( int PE = 0; PE < get_num_PEs(); PE++){ Latest_Finish_time.push_back(0);}
for( int task = 0; task < HyperGraph->get_num_taskes(); task++){
Start_times.push_back(0);Finish_times.push_back(0);  }

int tasks_scheduled = 0;
int current_task = Sorted_Taskes.at(0);


current_task =  find_start_task(Sorted_Taskes);
int new_task = 0;
//out << "First Task  " << current_task << endl;
do{

new_task = scheduel_task(current_task);
SchedulingSeq.push_back(current_task);
current_task = new_task;
tasks_scheduled++;


}while(tasks_scheduled < HyperGraph->get_num_taskes() && current_task >= 0 );


ind->find_max_deadline_violation(Finish_times,Waiting_time);

if( current_id >= 0 ){ /* If the scheduel is to be printed */
ind->print_deadline_violations(Finish_times,Waiting_time,current_id);
write_scheduel(current_id);
}

for( unsigned int i = 0; i < buffers.size();i++){
ind->set_buffer_use(buffers.at(i).get_PE(),buffers.at(i).get_bus(),buffers.at(i).get_size_Send(),
buffers.at(i).get_size_Recive(),buffers.at(i).get_size_Shared());}


ind->get_cost();
ind->set_HDV(ind->find_HDV(Finish_times,Waiting_time));
ind->find_smalest_slack(Finish_times,Waiting_time);
ind->find_energy_con(Start_times,Finish_times,Waiting_time,'
Waiting_time_in,SchedulingSeq,Send_Times,Receive_Times,current_id);
return 0;};


/**********************
Find start task
**********************/
```

```
int scheduling::find_start_task(vector<int> SortedTaskes){

int start_task = -1;
for(unsigned int i = 0; i < SortedTaskes.size()-1;i++){
 for( int j = 0; j < HyperGraph->get_num_sources(); j++){
 if( SortedTaskes.at(i) == HyperGraph->get_source_node(j) ){
  start_task = SortedTaskes.at(i);
  break;}
  }
 }
return start_task;}

/***********************
update_ready_list
***********************/

int scheduling::update_ready_list(int current_task,TaskGraph* HyperGraph){

   vector<int> source_nodes_waiting;

  /* Deletes currently schedueld task */
  int new_current_task = -1;
  vector<int>::iterator t = Sorted_Taskes.begin();
  for(unsigned int i = 0; i < Sorted_Taskes.size();i++){
   if(current_task == Sorted_Taskes.at(i) ){
 Sorted_Taskes.erase(t);break; }
   else{t++;}
 }

 /* Adds new tg-copies depending on 'Latest_Finish_Time' on PEs
  and taskes yet to be schedueld between first node in new  tg_copy and and last task on PE */

 bool all_parents_of_i_executed = true;
 bool is_predecessors_schedueld_ = false;
 bool no_task_added_to_ready_list = true;


 int i = -1;

vector<int>::iterator it1 = Sorted_Taskes.begin();
vector<int>::iterator it2 = Sorted_Taskes.end();
it2--;


for( ; (it1 != it2) ;it1++){ /* for all tasks in ready_list */
   i = *it1;
```

```
  is_predecessors_schedueld_ = false;


  if( Ready_List.at(i) == false  ){

if(HyperGraph->get_edge_type(current_task,i) >= 0){ // if i is child of current task

all_parents_of_i_executed = true;
for( int j = 0; j < i;j++){
 if( HyperGraph->get_edge_type(j,i) >= 0 && !Task_Executed.at(j)){
    all_parents_of_i_executed = false;break; }
}

 if(all_parents_of_i_executed){
  Ready_List.at(i) = true; no_task_added_to_ready_list = false;}
}
  }

 } // end loop over taskes



 Ready_List.at(current_task) =  false;
 new_current_task = -1;
 for(unsigned int i = 0; i < Sorted_Taskes.size()-1;i++){
  if(Ready_List.at(Sorted_Taskes.at(i)))
  {new_current_task = Sorted_Taskes.at(i);break;}
 }

 if(  no_task_added_to_ready_list ){ // && new_current_task == -1 && Sorted_Taskes.size() > 1
 /* Skift evt til hvis ready_listen er tom */
//  cout << " no taskes added to ready list " << endl;
//  string ter; cin >> ter;

 it1 = Sorted_Taskes.begin();
 for( ; (it1 != it2) ;it1++){
   i = *it1;
  /** Finding new task graph copies **/

  if( !Task_Executed.at(i) && is_there_dependencies_to_predecessor(i) ){

   if( (HyperGraph->get_min_start_time(i) <=  Latest_Finish_time.at(get_PE_executing_task(i))) ){

   if( is_predecessors_scheduled(i) ){ Ready_List.at(i) = is_predecessors_scheduled(i); }

   }else{ if( is_predecessors_scheduled(i) ){
    source_nodes_waiting.push_back(i); no_task_added_to_ready_list = false;} }
```

```
  }

  }

  for(unsigned int i = 0; i < Sorted_Taskes.size()-1;i++){
   if(Ready_List.at(Sorted_Taskes.at(i)))
   {new_current_task = Sorted_Taskes.at(i);break;}
  }


 }

if( (new_current_task == -1) && Sorted_Taskes.size() > 1 ){
 /**  All tg-copies in ready list is schedueld within deadiline, Latest_Finish_Time
  for PEs must be increased **/
 double min_min_start_time = 10000000;
 vector<int> new_task_in_line;// = -1;
 for(unsigned int i = 0; i < source_nodes_waiting.size(); i++){
  if(HyperGraph->get_min_start_time(source_nodes_waiting.at(i))  < min_min_start_time ){
   min_min_start_time = HyperGraph->get_min_start_time(source_nodes_waiting.at(i));
  }
}

for(unsigned int i = 0; i < source_nodes_waiting.size(); i++){
 if(HyperGraph->get_min_start_time(source_nodes_waiting.at(i)) == min_min_start_time ){
    new_task_in_line.push_back(source_nodes_waiting.at(i));
}
}
if( new_task_in_line.size() > 1){

int new_t = rand() % new_task_in_line.size();
new_t = new_task_in_line.at(new_t);
Latest_Finish_time.at(get_PE_executing_task(new_t)) = HyperGraph->get_min_start_time(new_t);
new_current_task = new_t; }

new_current_task = new_task_in_line.at(0);
Latest_Finish_time.at(get_PE_executing_task(new_task_in_line.at(0))) =
HyperGraph->get_min_start_time(new_task_in_line.at(0));
}else{ if( (new_current_task == -1) ){ }}

return new_current_task;}


/**********************
scheduel_task -
**********************/
```

```
int scheduling::scheduel_task(int task){

int new_task;
/** Findes Earlist starting time for task 'task **/
int HyperGraph_  = 0;
double start_time = 0; //Latest_Finish_time.at( get_PE_executing_task(task,HyperGraph_));
double earliest_comm_in = 1000000;
vector<double> start_times;
start_times.push_back(0); // .at(0) start_time for task
start_times.push_back(0); // .at(1) start_time for task - when no buff

//cout << " ---------- scheduling task: " << task << "-------------------" << endl;

for(int i = 0; i < task; i++){ /*HyperGraph->get_num_taskes()*/
 if(HyperGraph->get_edge_type(i,task) >= 0 && (get_PE_executing_task(i) !=get_PE_executing_task(task)) ){
 //cout << " edge " << i << "->"<< task << endl;


get_comm_time3(start_times,i,task);

if( start_times.at(1) > start_time){
start_time = start_times.at(1);}

if( (start_times.at(0) < earliest_comm_in) && (start_times.at(0)
>= Latest_Finish_time.at( get_PE_executing_task(task,HyperGraph_))) ){
earliest_comm_in =  start_times.at(0);}
 }
}


if( earliest_comm_in == 1000000 ){earliest_comm_in =
Latest_Finish_time.at( get_PE_executing_task(task,HyperGraph_));}
if( start_time - earliest_comm_in > 0 ){  Waiting_time_in.at(task) = ( start_time - earliest_comm_in); }

  start_time =  earliest_comm_in;
  Start_times.at(task) = start_time;
  Finish_times.at(task) = start_time+get_execution_time(task)+Waiting_time_in.at(task);
  Task_Executed.at(task) = true;

if( Finish_times.at(task) > Latest_Finish_time.at(get_PE_executing_task(task,HyperGraph_))){
Latest_Finish_time.at(get_PE_executing_task(task,HyperGraph_)) = Finish_times.at(task);}

new_task = update_ready_list(task,HyperGraph);

if( new_task > 0){
 for(int t = 0; t < new_task; t++){ /*HyperGraph->get_num_taskes() */
  if( HyperGraph->get_edge_type(t,new_task) >= 0 ){
```

```
  /** Scheduels edge t->new_task and updates  Finish_times and Latest_Finish_time **/

  set_comm_time3(t,new_task,HyperGraph_);

 if( Finish_times.at(t) > Latest_Finish_time.at(get_PE_executing_task(t)) ){
  /** Current PE must wait to sendt to receiving PE **/
  Latest_Finish_time.at(get_PE_executing_task(t)) = Finish_times.at(t);}

}
  }
 }

return new_task;}

/***********************
write_scheduel
***********************/

void scheduling::write_scheduel(int current_id){

int PE;

string task_PE_assignment;
string StartFinishTimes;
string cord;
string Link_Comm;
string Waiting_Times;
string Waiting_Times_in;
string buffer_overview;
string scheduel_overview;
string tg_copy_org_tg_relation;


if( current_id >= 0){
task_PE_assignment = add_id("task_PE_assignment.out",current_id);
StartFinishTimes = add_id("StartFinishTimes.out",current_id);
cord = add_id("cord.out",current_id);
Link_Comm = add_id("Link_Comm.out",current_id);
Waiting_Times = add_id("Waiting_Times.out",current_id);
Waiting_Times_in = add_id("Waiting_Times_in.out",current_id);
buffer_overview = add_id("buffer_overview_id.out",current_id);
scheduel_overview = add_id("Scheduel_overview.out",current_id);
tg_copy_org_tg_relation = add_id("tg_copy_org_tg_relation.out",current_id);}
else{
task_PE_assignment = "task_PE_assignment.out";
StartFinishTimes = "StartFinishTimes.out";
cord = "cord.out";
```

```
Link_Comm = "Link_Comm.out";
Waiting_Times = "Waiting_Times.out";
Waiting_Times_in = "Waiting_Times_in.out";
buffer_overview = "buffer_overview_id.out";
scheduel_overview ="Scheduel_overview.out";
tg_copy_org_tg_relation = "tg_copy_org_tg_relation.out";}

vector<vector<double> > Scheduel;
vector<double> y_corrd;
vector<int> PE_exe;

ofstream out0((const char*) task_PE_assignment.c_str());

for( int index = 0; index < HyperGraph->get_num_taskes(); index++){
int task = SchedulingSeq.at(index);
PE = ind->get_TaskAssignment(0).get_PE_executing_task(task);
out0 << PE << " ";
y_corrd.push_back( (Finish_times.at(task)-Waiting_time.at(task)-
Waiting_time_in.at(task)-Start_times.at(task))/2+Start_times.at(task)+Waiting_time_in.at(task));
PE_exe.push_back(PE);
}
out0 << endl;


out0.close();

  ofstream out1((const char*)StartFinishTimes.c_str());
  out1 << Start_times.size() << " " << get_num_PEs() << " " << 0 << " " << 0 << endl;
  for(unsigned int t = 0; t < Start_times.size(); t++){
   out1 << t << " " <<  get_PE_executing_task(t,0) <<" ";
   out1 << Start_times.at(t) << " " << Finish_times.at(t) << endl;}
  out1.close();

  ofstream out2((const char*)cord.c_str());
  for(unsigned int i = 0; i < SchedulingSeq.size();i++){
  out2 << HyperGraph->get_original_task_number(SchedulingSeq.at(i)) << " ";}
  out2 << endl;
  for(unsigned int i = 0; i < y_corrd.size();i++){
  out2 << y_corrd.at(i) << " ";}
  out2 << endl;
  for(unsigned int i = 0; i < PE_exe.size();i++){
  out2 << PE_exe.at(i) << " ";}
  out2 << endl;
  out2.close();

  ofstream out3((const char*)Link_Comm.c_str());
```

```
  for(unsigned int i = 0; i < Send_Times.size();i++){
  out3 << i << " " << 0 << " " << 0 << " " << 0 << endl;
   for(unsigned int j = 0; j < Send_Times.at(i).size();j=j+2){
out3 << Send_Times.at(i).at(j) << " " << Receive_Times.at(i).at(j);
out3 << " " << HyperGraph->get_original_task_number((int)Send_Times.at(i).at(j+1))
<< " " << HyperGraph->get_original_task_number((int)Receive_Times.at(i).at(j+1)) << endl;}
   }
 out3.close();

ofstream out4((const char*)Waiting_Times.c_str());
for(unsigned int i = 0; i < Waiting_time.size();i++){
if(Waiting_time.at(i) > 0){
out4 << i << " " << Finish_times.at(i)-Waiting_time.at(i) << " " << Finish_times.at(i)<<endl;
}else{
out4 << i << " " << Finish_times.at(i) << " " << Finish_times.at(i) <<endl;}
}
out4.close();

ofstream out5((const char*)Waiting_Times_in.c_str());
for(unsigned int i = 0; i < Waiting_time_in.size();i++){
if(Waiting_time_in.at(i) > 0){
out5 << i << " " <<  Start_times.at(i) << " " << Start_times.at(i)+Waiting_time_in.at(i) <<endl;
}else{
out5 << i << " " << Start_times.at(i) << " " << Start_times.at(i) <<endl;}
}
out5.close();

for( unsigned int i = 0; i < buffers.size();i++){
buffers.at(i).write_buffer_schedule(current_id,i,330);}

ofstream out6((const char*)buffer_overview.c_str());
out6 << " Buffer , PE , Bus,  size_Send,  size_Receive,  size_Shared " << endl;
for( unsigned int i = 0; i < buffers.size();i++){
 out6 << i << " " << buffers.at(i).get_PE() << " " <<
 buffers.at(i).get_bus() << " " << buffers.at(i).get_size_Send() << " " <<
 buffers.at(i).get_size_Recive() << " " << buffers.at(i).get_size_Shared()  << endl;}
out6.close();

ofstream out7((const char*) tg_copy_org_tg_relation.c_str());
for( int i = 0; i < HyperGraph->get_num_taskes(); i++){
out7 <<i <<" " << HyperGraph->get_original_task_number(i) << " ";
<< HyperGraph->get_original_task_graph(i) << endl;}


{  ofstream out8((const char*)scheduel_overview.c_str());
for(int i = 0; i < HyperGraph->get_num_sources();i++){
out8 << HyperGraph->get_source_node(i) << " " <<
```

```
HyperGraph->get_original_task_graph(HyperGraph->get_source_node(i)) << " ";
out8 << get_PE_executing_task(HyperGraph->get_source_node(i))
<< " " << Start_times.at(HyperGraph->get_source_node(i)) << " " <<
HyperGraph->get_deadline(HyperGraph->get_source_node(i)) <<endl;}

for( int i = 0; i < HyperGraph->get_num_dependenci();i++){
out8 << HyperGraph->get_predecessor(i) << " " <<
HyperGraph->get_original_task_graph(HyperGraph->get_predecessor(i)) << " ";
out8 << get_PE_executing_task(HyperGraph->get_predecessor(i)) << " "
 << Finish_times.at(HyperGraph->get_predecessor(i)) << " " <<
 HyperGraph->get_deadline(HyperGraph->get_predecessor(i))<< endl;

out8 << HyperGraph->get_successor(i) << " " <<
HyperGraph->get_original_task_graph(HyperGraph->get_successor(i)) << " ";
out8 << get_PE_executing_task(HyperGraph->get_successor(i)) << " "
<< Start_times.at(HyperGraph->get_successor(i)) << " " <<
HyperGraph->get_deadline(HyperGraph->get_successor(i)) << endl;

int org_tg = HyperGraph->get_original_task_graph(HyperGraph->get_successor(i));


if( (HyperGraph->get_successor(i) + TaskGraphs[org_tg]->get_num_taskes()-1)
 < HyperGraph->get_num_taskes()){

int task = (HyperGraph->get_successor(i) + TaskGraphs[org_tg]->get_num_taskes()-1);
out8 << task << " " << HyperGraph->get_original_task_graph(task) << " ";
out8 << get_PE_executing_task(task) << " " << Finish_times.at(task)
<< " " << HyperGraph->get_deadline(task) << endl;}
}
out8.close();
}

}


/*---------------------------------------- Small functions ----------------------------------------*/

bool scheduling::is_predecessors_scheduled(int task){

 vector<int> preds = HyperGraph->get_predecessors(task);

 bool result = false;
 if( preds.size() > 0){
  for(unsigned int i = 0; i < preds.size();i++){
   if( !Task_Executed[preds[i]] ){
    result = false;break;
    }else{ result = true;}
```

```
   }
}else{result = true;}
return result;}

int scheduling::get_last_bus_on_path(int task1,int task2){
 int index = -1;
 for(unsigned int i = 0; i < task_to_task_connection.at(task1).size(); i++){
  if( task_to_task_connection.at(task1).at(i) == task2){
   index = last_bus_on_path.at(task1).at(i);break;
  }}
 return index; }


/*------------------------ try_insertion ------------------------*/

double scheduling::try_insertion(int bus,int p,double new_Send_Time,
double comm_time,vector<bool>& bus_ready,vector<double>& min_times){

double receive_time_index;
double send_time_index2;

list<int>::iterator index = (comm_task_pos.at(bus)).begin();
list<int>::iterator index2 = index;
index2++;

while( index!= (comm_task_pos.at(bus)).end() ){

  receive_time_index = Receive_Times[bus][*index];


      if( *index < (int) Receive_Times[bus].size() && index2 != comm_task_pos[bus].end() ){
        send_time_index2 = Send_Times[bus][*index2];

   if( receive_time_index  <= new_Send_Time && new_Send_Time+comm_time <= send_time_index2 ){
  /* insertion place found */
  bus_ready.at(p) = true;
  min_times.at(p) = new_Send_Time;
  poss_in_list.at(bus)= index;
         break;
 }else{
if( Receive_Times[bus][*index] > new_Send_Time ){
bus_ready.at(p) = false;
min_times.at(p) = Receive_Times[bus][*index];
new_Send_Time = Receive_Times[bus][*index];
}
     }
```

```
       }else{
       /* insertion failed */
       new_Send_Time = Receive_Times[bus].at(*index);
       min_times.at(p) = new_Send_Time;
       break;}
  index++;index2++;
 }

return new_Send_Time;}


void scheduling::set_comm_task(int assigned_to_link, int from,int to,double min_Send_Time, double comm_time){

 if( poss_in_list.at(assigned_to_link) != NULL){

 poss_in_list.at(assigned_to_link)++;
 comm_task_pos.at(assigned_to_link).insert(poss_in_list.at(assigned_to_link),
   Send_Times.at(assigned_to_link).size());
 }else{
  comm_task_pos.at(assigned_to_link).push_back(Send_Times.at(assigned_to_link).size());}

if(  min_Send_Time+comm_time >last_comm_event.at(assigned_to_link) ){
last_comm_event.at(assigned_to_link) = min_Send_Time+comm_time;}

Send_Times.at(assigned_to_link).push_back(min_Send_Time);
Send_Times.at(assigned_to_link).push_back(from);
Receive_Times.at(assigned_to_link).push_back(min_Send_Time+comm_time);
Receive_Times.at(assigned_to_link).push_back(to);
}

/*****************************
vector<int> scheduling::weight_priority_listes(
*****************************/

vector<double> scheduling::weight_priority_listes(double waight,
vector<double> pre_mes1, vector<double> pre_mes2){
 /** spines two list with preority mesortements together using a user defines waight **/

    vector<double> resulting_pre_mesurement;
    double min_v = *min_element(pre_mes1.begin(),pre_mes1.end()-1);
    double max_v = *max_element(pre_mes1.begin(),pre_mes1.end());

    if( min_v == max_v ){max_v = 2*min_v;}
    for( int i = 0; i < pre_mes1.size()-1; i++ ){

     pre_mes1[i] = (pre_mes1[i]-min_v )*(100/(max_v-min_v));
     assert( pre_mes1[i] >= 0);
```

```
     assert( pre_mes1[i] <= 100.00000001);
     assert( pre_mes2[i] >= 0);
     assert( pre_mes2[i] <= 100);
     resulting_pre_mesurement.push_back(pre_mes1[i]*(1-waight)+pre_mes2[i]*(waight));}
//     resulting_pre_mesurement.push_back(pre_mes1[i]*waight+pre_mes2[i]*(1-waight));}
     resulting_pre_mesurement.push_back(-1);
return resulting_pre_mesurement;}

/*****************************
vector<int> scheduling::make_pre_list(vector<double> pre_mesurements)
*****************************/

vector<int> scheduling::make_pre_list(vector<double> pre_mesurements){

    vector<double>::iterator largest;
    vector<int>::iterator index_begin;
    vector<int> sorted_taskes;
    vector<int> task_index;
    task_index.assign( pre_mesurements.size(),0);
    index_begin = task_index.begin();
    for( int i = 0; i < task_index.size(); i++ ) {task_index[i]=i;}
    while( pre_mesurements.size() > 0){
 largest = max_element(pre_mesurements.begin(),pre_mesurements.end());
 sorted_taskes.push_back(task_index.at(largest-pre_mesurements.begin()));
 task_index.erase(index_begin+(int)(largest-pre_mesurements.begin()));
 pre_mesurements.erase(largest);

 }
return sorted_taskes;}

#include "comm_paths.cpp"
#include "comm_conservative.cpp"
```

A.0.11 comm_conservative.cpp

```
double scheduling::set_comm_time3(int i,int task,int TTTT){

/*********
This function findes does the following:
1) Findes the possible comm paths between PE_executing(i) and PE_executing(task)
2) Findes the paths that allown the earliest start time for the kommunikations event min_Send_Time
3) Schedules receive event on receiving PEs buffer ( if no is buffer avaible the waiting time on
the PE is added when task 'task' is secheduld
4) Schedules send event on Sending PEs buffer or PE it self
5) set the start and end time for the kommunikation event on alle busses on the path
```

```
************************/
double min_Send_Time = 100000000;

double Send_Time = Finish_times.at(i);

/* min_Send_Time_Considering_OutPut_Buffer*/
Send_Time = Latest_Finish_time.at(get_PE_executing_task(i));


vector<vector<double> > Paths;
vector<vector<double> > tmp_Paths_current_bus;
int path_with_min_Send_Time = -1;

if(get_PE_executing_task(i) == get_PE_executing_task(task) ){
  /** task 'i' and 'task' executes on same PE **/
}else{

  for(unsigned int b = 0; b < Send_Times.size() ;b++){


   if( bus_connected_to_assigned_PE(b,i)  ){
/** if the current bus is connected to i's PE **/

tmp_Paths_current_bus = get_comm_time_lists(i,task,b);

for(unsigned int p = 0; p < tmp_Paths_current_bus.size();p++){
    Paths.push_back(tmp_Paths_current_bus.at(p));}

  }// end if PE assigned to i, connected to bus 'b'
 }// end loop over busses
} // end if( get_comm_time(i,task,HyperGraph) != 0 )

/**  Inserting communication task in scheduel **/
if( (get_PE_executing_task(i) != get_PE_executing_task(task)) ) {
/* buss free for communication OR taskes assigned to same PE */

assert( Paths.size() > 0);
for(unsigned int p = 0; p < Paths.size(); p++){

   Send_Time = get_buffer(i,Paths.at(p).at(1))->min_Send_Time_COB(HyperGraph,i,task,
    Finish_times.at(i)-Waiting_time.at(i),Latest_Finish_time.at(get_PE_executing_task(i)));
  /* Send_Time = min_Send_Time_COB(Paths.at(p).at(1),i,task,Send_Time)    */


   double new_Send_Time = min_Send_Time_over_path(Paths.at(p),i,task,Send_Time);
double min_comm_time = 1000000;
unsigned int min_length = 100000;
```

```
if(new_Send_Time <= min_Send_Time){
 if( Paths.at(p).at(0) < min_comm_time){
min_Send_Time = new_Send_Time;
assert( min_Send_Time > 0 );
min_comm_time = Paths.at(p).at(0);
min_length = Paths.at(p).size();
path_with_min_Send_Time = p;}

if( Paths.at(p).at(0) == min_comm_time){
 if( Paths.at(p).size() < min_length ){
   min_Send_Time = new_Send_Time;
min_comm_time = Paths.at(p).at(0);
min_length = Paths.at(p).size();
path_with_min_Send_Time = p;}
}
    }
}

/*hvis input bufferen bruges skal dette også fast ligges evt her */
/* En kontrol er at der ikke kommer Vente tid på denne hvis det skal passe med det tidligere
tjek */

double Comm_Time = Paths.at(path_with_min_Send_Time).at(0);

int bus1 = (int) Paths.at(path_with_min_Send_Time).at(1);
int bus2 = -1;
if( Paths.at(path_with_min_Send_Time).size() == 2){
 bus2 = bus1;}else{
 bus2 = (int) Paths.at(path_with_min_Send_Time).at(Paths.at(path_with_min_Send_Time).size()-1);}


/*------------IN BUFFER PE,bus2---------------*/

double time_to_Receive_PE_ready  = 0;
double Waiting_Time = 0;
double time_for_ready_PE = Latest_Finish_time.at(get_PE_executing_task(task));


time_to_Receive_PE_ready = get_buffer(task,bus2)->store_buff_in(HyperGraph,i,
task,min_Send_Time,time_for_ready_PE);


assert( time_to_Receive_PE_ready < 1*10^(-15));
assert( time_to_Receive_PE_ready > -1*10^(-15));
```

```
/*------------OUT BUFFER PE,bus1---------------*/
double Finish_Time_For_i = (Finish_times.at(i)-Waiting_time.at(i));
{

Waiting_Time = get_buffer(i,bus1)->store_buff_out(HyperGraph,i,task,
Finish_Time_For_i,min_Send_Time+Paths.at(path_with_min_Send_Time).at(0) );

if( Waiting_Time > 0 ){
/** not enougt room in the buffer - Waiting time for PE must be found **/
 Waiting_Time =   min_Send_Time + time_to_Receive_PE_ready + Comm_Time -
  (Finish_times.at(i) - Waiting_time.at(i));
  }
 }

/**  A **/
/** Waiting_Time' er vente tiden på at modtager 'PE' og  bus(er)  er klar til at modtage kommunikationen.
    dvs den tid opgaven skal ligge på PE før den kan sendes. Er vente Tiden 0 ligger opgaven i bussen



/*if( Send_Time +Waiting_Time > Finish_times.at(i)+1*10^(-15) ){*/
 if( Latest_Finish_time.at(get_PE_executing_task(i))+
  Waiting_Time > Finish_times.at(i)+1*10^(-15) ){


 double old_finish_time = Finish_times.at(i)-Waiting_time.at(i);
 Waiting_time.at(i)  =  Waiting_Time;
 Finish_times.at(i) = old_finish_time+Waiting_Time;
   Latest_Finish_time.at(get_PE_executing_task(i)) =  Finish_times.at(i);
 }else{
 cout << Send_Time+Waiting_Time <<">"<< Finish_times.at(i)<<endl;
 exit(-1);}


}else{ /** Ingen vente tid dvs kanter ligger i bufferen **/


assert( time_to_Receive_PE_ready < (double) 1*10^(-15));
assert( time_to_Receive_PE_ready > -1*10^(-15));

}

set_last_bus_on_path(i,task,bus2);
for(unsigned int bus = 1; bus < Paths.at(path_with_min_Send_Time).size();bus++){
assert(min_Send_Time != 0);
set_comm_task((int)Paths.at(path_with_min_Send_Time).at(bus),i,task,
```

```
min_Send_Time,Paths.at(path_with_min_Send_Time).at(0));

   }

}

return (min_Send_Time-Send_Time);}

/*************************************************
void scheduling::get_comm_time3(vector<double>& start_times, int i,int task){
*************************************************/

void scheduling::get_comm_time3(vector<double>& start_times, int i,int task){
/* Does the following:
 1) findes the start and end time of the communikation task 'i->task'
 2) cheks if the receiving PE has a buffer YES/NO,
    if NO: return the start and end time where the PE must be ready receiving the kommuniukation
    if YES: return the time where the kommunikation is finished in the buffer
     and the PE can start using the data   */

double start_time = 0;
double start_time_no_buff = 0;
vector<int> path;
int bus = -1;
int start_pos = 1;

for( int link = 0; link < get_num_Links(); link++){

  if( Receive_Times.at(link).size() > 101 ){
   start_pos = Receive_Times.at(link).size()-101;}

for(unsigned int index = 1; index <Receive_Times.at(link).size();index = index+2){

if( Receive_Times.at(link).at(index) == task && Send_Times.at(link).at(index) == i ){

path.push_back(link);
if( Receive_Times.at(link).at(index-1) > start_time ){
start_time = Receive_Times.at(link).at(index-1);
}
if( Send_Times.at(link).at(index-1) > start_time_no_buff ){
start_time_no_buff = Send_Times.at(link).at(index-1);
}
break;
}
}
}
/* 2) */{
```

```
 bus = get_last_bus_on_path(i,task);

 if( (!get_buffer(task,bus)->in_buff(i,task) )&&
  (get_PE_executing_task(i) != get_PE_executing_task(task)) ){
  assert(start_time != 0);
  assert(start_time_no_buff != 0);
  start_times.at(0) = start_time_no_buff;
  start_times.at(1) = start_time;
 }else{ /* brug tiden hvor opgaven er færdig i input bufferen ( slå denne opå )*/
   start_time = (get_buffer(task,bus))->in_buff_task_to_PE(i,task);
  // cout << " start time - from buffer opslag " << start_time << endl;
   start_times.at(0) = start_time;
   start_times.at(1) = start_time;
   }
}/* end 2) */
/* start_times - now updated -- end get_comm_time3*/ }



/*********************************
double scheduling::min_Send_Time_considering_input_buffers(double min_Send_Time,int i,int task,
int bus, vector<double>& Latest_Finish_time)

returns 'min_Send_Time' considering the buffer : add to min_Send_Time if buffer full
*******************************************************************************/
double scheduling::min_Send_Time_considering_input_buffers(double min_Send_Time,
int i,int task,int bus, vector<double>& Latest_Finish_time){

double time_for_ready_PE = Latest_Finish_time.at(get_PE_executing_task(task));
double time_to_Receive_PE_ready = 0;
time_to_Receive_PE_ready = get_buffer(task,bus)->buffer_in_available(HyperGraph,i,
task,time_for_ready_PE,min_Send_Time);
   min_Send_Time += time_to_Receive_PE_ready;

return min_Send_Time;}
```

A.0.12 comm_paths.cpp

```
/**  Findes communications paths over bridges through architure,
Startting at bus;  'bus', edge task12->task2 **/

vector<vector<double> >scheduling::get_comm_time_lists(int task1,int task2,int bus){
/** returns list of paths through system for communication **/
/** <comm_time , b1 , b2 ......> **/
```

```
int max_runs_through_paths = ind->get_num_Links();
vector<vector<double> >result;
vector<double> tmp_path;
tmp_path.push_back(get_comm_time(task1,task2,bus,HyperGraph));
tmp_path.push_back(bus);
for( int i = 0; i < 2*max_runs_through_paths;i++){
result.push_back(tmp_path);}

vector<bool> allready_used_bridge;
for(unsigned int b = 0; b < ind->bus1.size(); b++){
allready_used_bridge.push_back(false);}
vector<vector<bool> > allready_used_bridge_at_path;
allready_used_bridge_at_path.push_back(allready_used_bridge);

int num_bridges_used = 0;

/** asserts that no infinite loops/cykels is made through system **/
int last_bus_in_path = -1;
int num_paths = 1; // 1 d7 sep
int runs_through_paths = 0;


vector<double> path_tmp;
vector<double> path_tmp_for_path;

if( bus_connected_to_assigned_PE(bus,task2,HyperGraph) == 0 ){

  /** findign paths through system **/
  while( (num_paths < (int)ind->bus1.size()+1 ) && (num_bridges_used < (num_paths*((int)ind->bus1.size()))
   ) && (runs_through_paths < max_runs_through_paths) ){

for( int paths = 0; paths < num_paths; paths++){
                last_bus_in_path = (int)result.at(paths).at( (int)result.at(paths).size()-1);


int new_paths = 0;
bool add_at_path = true;
path_tmp_for_path = result.at(paths);
vector<bool> used_paths_tmp = allready_used_bridge_at_path.at(paths);
for(unsigned int bridge = 0; bridge < ind->bus1.size(); bridge++){
path_tmp = path_tmp_for_path;

     if( add_at_path == true){
    last_bus_in_path = (int) result.at(paths).at(result.at(paths).size()-1);/*6 Oct */
if( bus_connected_to_assigned_PE(last_bus_in_path,task2,HyperGraph) == 0 ){
```

```
if( (ind->bus1.at(bridge) == last_bus_in_path || ind->bus2.at(bridge) ==
  last_bus_in_path) && (allready_used_bridge_at_path.at(paths).at(bridge)== false) ){
 // En ikke før under søgt bridge tilsluttet nuværende bus
 allready_used_bridge_at_path.at(paths).at(bridge) = true;
 num_bridges_used++;
 add_at_path = false;

 /** Tilføjer bussen på den anden side af bridgen **/
if( ind->bus1.at(bridge) == last_bus_in_path ){
result.at(paths).push_back(ind->bus2.at(bridge));
if(get_comm_time(task1,task2,ind->bus2.at(bridge),HyperGraph) >
result.at(paths).at(0) ){
result.at(paths).at(0) = get_comm_time(task1,
task2,ind->bus2.at(bridge),HyperGraph);}
}

if( ind->bus2.at(bridge) == last_bus_in_path ){
result.at(paths).push_back(ind->bus1.at(bridge));
if(get_comm_time(task1,task2,ind->bus1.at(bridge),HyperGraph) >
result.at(paths).at(0) ){
result.at(paths).at(0) =
  get_comm_time(task1,task2,ind->bus1.at(bridge),HyperGraph);}
}
 }

}
 }else{

      last_bus_in_path = (int) path_tmp.at(path_tmp.size()-1); /*6 Oct */
if( bus_connected_to_assigned_PE(last_bus_in_path,task2,HyperGraph) == 0 ){
 if( (ind->bus1.at(bridge) == last_bus_in_path || ind->bus2.at(bridge) ==
  last_bus_in_path) && (used_paths_tmp.at(bridge)== false) ){
 // En ikke før under søgt bridge tilsluttet nuværende bus
        used_paths_tmp.at(bridge) = true;
 num_bridges_used++;


   if( ind->bus1.at(bridge) == last_bus_in_path ){
path_tmp.push_back(ind->bus2.at(bridge));
if(get_comm_time(task1,task2,ind->bus2.at(bridge),HyperGraph) >  path_tmp.at(0) ){
  path_tmp.at(0) =  get_comm_time(task1,task2,ind->bus2.at(bridge),HyperGraph);}
new_paths++;
}
if( ind->bus2.at(bridge) == last_bus_in_path ){
path_tmp.push_back(ind->bus1.at(bridge));
if(get_comm_time(task1,task2,ind->bus1.at(bridge),HyperGraph) > path_tmp.at(0) ){
```

```
path_tmp.at(0) = get_comm_time(task1,task2,ind->bus1.at(bridge),HyperGraph);}
new_paths++;
}
assert(num_paths < result.size() );
result.at(num_paths) = path_tmp;
num_paths++;
allready_used_bridge_at_path.push_back(used_paths_tmp);
 }

 }
 }// end else

  }
}
  runs_through_paths++;
    }// end while

/** if last bus is not connectet to recoiving PE the path has a dead end and must be deleted **/
vector<int> Paths_To_Remove;
for(unsigned int  path = 0; path < result.size(); path++){
assert( (int) result.at(path).size() > 1);
assert( (unsigned int) path < result.size());
    if(  bus_connected_to_assigned_PE((int)result.at(path).at(result.at(path).size()-1),task2) == 0 ){
Paths_To_Remove.push_back(path); }
   }

  for( unsigned int i = 0; i < Paths_To_Remove.size();i++){
vector<vector<double> >::iterator Iter = result.begin();
Iter+=Paths_To_Remove.at(i);
   result.erase(Iter);
   for( int t = i; t < Paths_To_Remove.size();t++){
  Paths_To_Remove.at(t)--;}
  }
}
assert( result.size() > 0);
return result;}

/*******************************************************
double scheduling::min_Send_Time_over_path(...)

Findes min_Send_Time for at path througt an architure with allready
schedueld taskes
************************************************/

double scheduling::min_Send_Time_over_path(vector<double> path, int task1,int task2,double min_Send_Time){

vector<bool> bus_ready; // index bus # in path
```

```
vector<double> min_times;
for(unsigned int i = 1; i < path.size(); i++){ bus_ready.push_back(false);
       min_times.push_back(0);}
bool all_busses_ready = false;
double new_Send_Time = min_Send_Time;

int g = 0;

while( !all_busses_ready ){
for(unsigned int p = 0; p < (path.size()-1); p++ ){

 int bus = (int)path.at(p+1);
 poss_in_list.at(bus)= NULL;


if( (p) == (path.size()-2)){// in buff

 double min_Send_Time_CIB =
  min_Send_Time_considering_input_buffers(min_Send_Time,task1,task2,bus,Latest_Finish_time);

 if( (new_Send_Time <= min_Send_Time_CIB) && (min_Send_Time_CIB >= min_Send_Time) ){
 new_Send_Time = min_Send_Time_CIB;
 //bus_ready.at(p) = true;
 min_times.at(p) = new_Send_Time;}
}

int i1 = Receive_Times.at(bus).size()-1;

 if( (Receive_Times.at(bus).size() > 0) && (new_Send_Time <=
  Send_Times.at(bus).at(Send_Times.at(bus).size()-2))  ){
assert( path.at(0) != 0);
new_Send_Time = try_insertion(bus, p, new_Send_Time, path.at(0),bus_ready, min_times);
 }



  if(  Receive_Times.at(bus).size() == 0){
       bus_ready.at(p) = true;
       min_times.at(p) = new_Send_Time;
       poss_in_list.at(bus)= NULL;}

   if( (bus_ready.at(p) == false) && (Receive_Times.at(bus).size() >  0) &&
    (last_comm_event.at(bus) >= new_Send_Time)  ){
      /** Inserting At end **/
        new_Send_Time = last_comm_event.at(bus);
        min_times.at(p) = new_Send_Time;
bus_ready.at(p) = true;
```

```
poss_in_list.at(bus)= NULL;}

 /* updates min_Send_Time */
  if( p > 0){
if( new_Send_Time == min_times.at(p-1) ){ min_times.at(p) = new_Send_Time;}
   }else{
   if( new_Send_Time == min_times.at(min_times.size()-1) ){ min_times.at(p) = new_Send_Time;}}

}// end loop over p

all_busses_ready = true;
for(unsigned int b = 0; (b < bus_ready.size()-1) ;b++){
if( min_times.at(b) != min_times.at(b+1) ){all_busses_ready = false;}
}

if( g > 1000 ){cout << " error inf loop in min_Send_Time_over_path "<< endl;}
g++;}

return new_Send_Time;}
```

A.0.13 buffer.hpp

```
class buffer{


private:

int PE;
int bus;

double size_Send;
int var_size_Send;

double size_Recive;
int var_size_Recive;

double size_Shared;
int var_size_Shared;

vector<double> buff_in_Send;
vector<double> buff_out_Send;
vector<int> from_Send;
vector<int> to_Send;

vector<double> buff_in_Recive;
```

```
vector<double> buff_out_Recive;
vector<int> from_Recive;
vector<int> to_Recive;

vector<double> buff_in_Shared;
vector<double> buff_out_Shared;
vector<int> from_Shared;
vector<int> to_Shared;


public:

buffer(int PE_,int bus_,int size_Send_,int size_Recive_,int size_Shared_,
int var_size_Send_,int var_size_Recive_,int var_size_Shared_){

PE = PE_;
bus = bus_;
size_Send = size_Send_;
size_Recive = size_Recive_;
size_Shared = size_Shared_;
var_size_Send = var_size_Send_;
var_size_Recive = var_size_Recive_;
var_size_Shared = var_size_Shared_;
}

double get_size_Send(){return size_Send;}

double get_size_Recive(){return size_Recive;}

double get_size_Shared(){return size_Shared;}

/*-------------------------------------------------------------------------------*/
int get_PE(){return PE;}
int get_bus(){return bus;}
/*-------------------------------------------------------------------------------*/
double get_buffer_sepc(){ return size_Recive;}
/*-------------------------------------------------------------------------------*/

double used_mem(int buff_type,double task_finished,
double task_send,TaskGraph* TG){
double mem_used = 0;

  switch( buff_type ) {
     case 0:

   for(unsigned int i = 0; i < buff_in_Send.size(); i++){
if( (buff_in_Send.at(i) <= task_finished) &&
```

```
( task_finished < buff_out_Send.at(i) &&
buff_out_Send.at(i) <= task_send) ){
mem_used+= Comm.at(0)->get_comm_amount(TG->
get_edge_type(from_Send.at(i),to_Send.at(i)) );}
}
return mem_used;
      case 1:
  for(unsigned int i = 0; i < buff_in_Recive.size(); i++){
if( (buff_in_Recive.at(i) <= task_finished) && (
task_finished < buff_out_Recive.at(i) &&
buff_out_Recive.at(i) <= task_send) ){
mem_used+= Comm.at(0)->get_comm_amount(TG->
get_edge_type(from_Recive.at(i),to_Recive.at(i)) );}
}
  return mem_used;
     case 2:
  for(unsigned int i = 0; i < buff_in_Shared.size(); i++){
       if( ((buff_in_Shared.at(i) <= task_finished) &&
        (task_finished < buff_out_Shared.at(i) )) &&
buff_out_Shared.at(i) <= task_send){
       mem_used+= Comm.at(0)->get_comm_amount(TG->
        get_edge_type(from_Shared.at(i),to_Shared.at(i)) );}
}
  return mem_used;
  break;
     default:
return mem_used;
       break;}

};

/*-------------------------------------------------------------------------------*/
bool out_buff(){return (size_Send > 0 || var_size_Send > 0 ||
size_Shared > 0 || var_size_Shared > 0);}
/*-------------------------------------------------------------------------------*/

double min_Send_Time_COB(TaskGraph* TG,int from,int to,
double task_finished,double task_send){

assert( task_finished <= task_send +1*10^(-15));


 if( task_finished < task_send ){
 /** buffer being used for storing kommunication **/

if( (var_size_Shared == 1) || size_Shared > 0 ){
 return  task_finished;}
```

```
else{

/* finder den brugte buffer mængde */
double mem_used = used_mem(0,task_finished,task_send,TG);

if( var_size_Send == 1 ){
 if( size_Send <
   mem_used+Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to)) ){
 size_Send =
  mem_used+Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to));}
}

if( 0 <= size_Send-mem_used-
Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to)) ){
 return task_finished;
}else{
/* Der bliver lagt en vente tid ind på Processoren */
 return task_send;}
}
  }else{
   /** no use for storing in buffer **/
  return task_send;}
};


/*-------------------------------------------------------------------------*/

double store_buff_out(TaskGraph* TG,int from,int to,
double task_finished,double task_send){

assert( task_finished <= task_send+1*10^(-15) );

 if( task_finished < task_send ){
 /** buffer being used for storing kommunication **/

   if( (var_size_Shared == 1) || size_Shared > 0 ){
 return store_buff_Shared(TG,from,to,task_finished,task_send);
   }else{

double mem_used = used_mem(0,task_finished,task_send,TG);

if( var_size_Send == 1 ){
 if( size_Send < mem_used+Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to)) ){
 size_Send = mem_used+Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to));}
}
```

```
if( 0 <= size_Send-mem_used-Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to)) ){
 buff_in_Send.push_back(task_finished);
 buff_out_Send.push_back(task_send);
 from_Send.push_back(from);
   to_Send.push_back(to);


 return 0;
}else{
/* Der bliver lagt en vente tid ind på Processoren */
 return (task_send-task_finished);}
}
  }else{ /** no use for storing in buffer **/  return 0;}
};
/*-------------------------------------------------------------------------------*/
double buffer_in_available(TaskGraph* TG,int from,int to,
double time_for_ready_PE,double Receive_Time){
/** returns the waiting time **/

 if( time_for_ready_PE > Receive_Time){
    /** receiving PE busy **/
   if( (var_size_Shared == 1) || size_Shared > 0 ){
    if( var_size_Shared == 1){
     return 0;
    }else{
    double mem_used_shared = used_mem(2,time_for_ready_PE,Receive_Time,TG);
if( size_Shared <size_Shared-mem_used_shared-
Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to)) ){
 return 0;
}else{
 /** error buffer not large enough or capacity not large enough **/
 return (time_for_ready_PE-Receive_Time);}
   }

  }else{

if( var_size_Recive  == 1 ){ return 0;}

double mem_used = used_mem(1,time_for_ready_PE,Receive_Time,TG);
  if( 0 <= size_Recive-mem_used-
Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to)) ){
 return 0;
}else{
/* error buffer not large enough or capacity not large enough */;
    /* Der bliver lagt en vente tid ind på Processoren */
 return (time_for_ready_PE-Receive_Time);}
}
```

```
   }else{
  /** Receiving PE ready no waiting time **/
   return 0;}
  }; /* end storer buffer in */
/*--------------------------------------------------------------------------------*/

bool in_buff(){ return (size_Recive > 0 ||
var_size_Recive > 0 || size_Shared > 0 || var_size_Shared > 0);}


bool in_buff(int task1,int task2 ){
bool to_return = false;
double comm_amount =
Comm.at(0)->get_comm_amount(HyperGraph->get_edge_type(task1,task2));
if( (comm_amount > size_Recive) &&
(comm_amount > var_size_Recive) &&
(comm_amount > size_Shared) &&
(comm_amount > var_size_Shared) ){
 /** Buffer not big enough to holde data  **/
return false;
}else{

for( unsigned int i = 0; i < from_Recive.size(); i++){
 if( task1 == from_Recive.at(i) && task2 == to_Recive.at(i) ){
to_return = true;break;}
}

for( unsigned int i = 0; i < from_Shared.size(); i++){
 if( task1 == from_Shared.at(i) && task2 == to_Shared.at(i) ){
to_return = true;break;}
}
}

return to_return;
}

/*--------------------------------------------------------------------------------*/

double in_buff_task_to_PE(int task1,int task2){
double result = 0;
for( unsigned int i = 0; i < from_Recive.size(); i++){
 if( task1 == from_Recive.at(i) && task2 == to_Recive.at(i) ){
result= buff_out_Recive.at(i);break;}
}

for( unsigned int i = 0; i < from_Shared.size(); i++){
```

```
 if( task1 == from_Shared.at(i) && task2 == to_Shared.at(i) ){
result= buff_out_Shared.at(i);break;}
}
assert(result != 0);
return result;}

/*-------------------------------------------------------------------------*/

double store_buff_in(TaskGraph* TG,int from,int to,
double Receive_Time,double time_for_ready_PE){
/** Schedulerer edge 'from'->'to' til modtager bufferen i
tids intervallet [Receive_Time,time_for_ready_PE]
    hvis mulig, alternativt retuneres vente tiden som afsender
      processoren må vente på den modtagende PE er klar
    Receive_Time - er tiden hvor data kommunikationen startes på bussen      **/

 if( time_for_ready_PE > Receive_Time){
  /** receiving PE busy **/
   if( (var_size_Shared == 1) || size_Shared > 0 ){
 return store_buff_Shared(TG,from,to,Receive_Time,time_for_ready_PE);
  }else{

double mem_used = used_mem(1,time_for_ready_PE,Receive_Time,TG);


if( var_size_Recive  == 1 ){
 if( size_Recive <
  mem_used+Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to)) ){
   size_Recive = mem_used+Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to));}
 }

if( 0 <= size_Recive-mem_used-
Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to)) ){
/** Buffer big enougt  */
 buff_in_Recive.push_back(Receive_Time);
   buff_out_Recive.push_back(time_for_ready_PE);
 from_Recive.push_back(from);
   to_Recive.push_back(to);
 return 0;
}else{
/* error buffer not large enough or capacity not large enough */
    /* Der bliver lagt en vente tid ind på Processoren */
 return (time_for_ready_PE-Receive_Time);}
}

      }else{
  /** Receiving PE ready no waiting time **/
```

```
   return 0;}
  }; /* end storer buffer in */

/*-------------------------------------------------------------------------------*/


double store_buff_Shared(TaskGraph* TG,int from,int to,
double in_buff,double out_buff){


 if( in_buff < out_buff){

double mem_used = used_mem(2,in_buff,out_buff,TG);

if(  var_size_Shared == 1 ){
 if( size_Shared < mem_used+
  Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to)) ){
 size_Shared = mem_used+
  Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to)); }
}

if( 0 <= size_Shared-mem_used-
Comm.at(0)->get_comm_amount(TG->get_edge_type(from,to)) ){

 buff_in_Shared.push_back(in_buff);
   buff_out_Shared.push_back(out_buff);
 from_Shared.push_back(from);
   to_Shared.push_back(to);

 return 0;
}else{
   /* error buffer not large enough or capacity not  large enough */
   /* Der bliver lagt en vente tid ind på Processoren */
 return (out_buff-in_buff);}

  }else{
   /** Receiving PE ready no waiting time **/
  return 0;}
 };
/*-------------------------------------------------------------------------------*/
bool input_buff_free(TaskGraph* TG,int i,int task,
double new_Send_Time,double new_Send_Time_){
return true;};

/*-------------------------------------------------------------------------------*/

void write_buffer_schedule(int current_id,int buff,int max_length){
```

```
/** Der skriver en buffer scheduel ud hvis den er delt
 Der skrives to buffere ellers
  ( dvs i tilfælde af ingen buffer eller dedikere input/output-buffer **/

TaskGraph* TG = HyperGraph;
string bufferSchedule;
string bufferFromTo_Size;
if(current_id >= 0){
  bufferSchedule = add_id("buffer_schedule.out",current_id);
  bufferFromTo_Size = add_id("buffer_edge_spec.out",current_id);
 }else{
  bufferSchedule = "buffer_schedule.out";
  bufferFromTo_Size = "buffer_edge_spec.out";}


 if(  size_Shared > 0 ){
ofstream out((const char*) bufferSchedule.c_str(),ios::app);
out <<  buff << " " << 2 << " " << size_Shared << " ";
 for( int i = 0; i < (int) buff_in_Shared.size() || i < max_length ; i++){
      if( i < (int) buff_in_Shared.size() ){
  out << buff_in_Shared.at(i) << " " << buff_out_Shared.at(i) << " ";}
 else{
 out << 0 << " " << 0 << " ";}
 }
out << endl;
out.close();

/** Printing edge 'From'->'To' and size **/
ofstream out1((const char*) bufferFromTo_Size.c_str(),ios::app);
 for( int i = 0; i < (int) buff_in_Shared.size() || i < max_length ; i++){
     if( i < (int) buff_in_Shared.size() ){
  out1 << from_Shared.at(i) <<  " ";}else{out1 << 0 << " ";}}
  out1 << endl;
 for( int i = 0; i < (int) buff_in_Shared.size() || i < max_length ; i++){
      if( i < (int) buff_in_Shared.size() ){
  out1 << to_Shared.at(i) <<  " ";}else{out1 << 0 << " ";}}
  out1 << endl;
 for( int i = 0; i < (int) buff_in_Shared.size() || i < max_length ; i++){
     if( i < (int) buff_in_Shared.size() ){
  out1 << Comm.at(0)->get_comm_amount(TG->get_edge_type(from_Shared.at(i),
   to_Shared.at(i))) <<  " ";}
  else{out1 << 0 << " ";}}
     out1 << endl;

out1 << endl;
out1.close();
```

```
 }else{


ofstream out((const char*) bufferSchedule.c_str(),ios::app);
/** Send buffer **/
out << buff << " " << 0 << " " << size_Send << " ";
 for( int i = 0; i < (int) buff_in_Send.size() || i < max_length ; i++){
      if( i < (int) buff_in_Send.size() ){
  out << buff_in_Send.at(i) << " " << buff_out_Send.at(i) << " ";}
 else{
 out << 0 << " " << 0 << " ";}
 }
out << endl;
/** Recive **/
out << buff << " " << 1 << " " << size_Recive << " ";
 for( int i = 0; i < (int) buff_in_Recive.size() || i < max_length ; i++){
      if( i < (int) buff_in_Recive.size() ){
  out << buff_in_Recive.at(i) << " " << buff_out_Recive.at(i) << " ";}
 else{
 out << 0 << " " << 0 << " ";}
 }
out << endl;
out.close();

/* Out_put buffer */
/** Printing edge 'From'->'To' and size **/
ofstream out1((const char*) bufferFromTo_Size.c_str(),ios::app);
 for( int i = 0; i < (int) buff_in_Send.size() || i < max_length ; i++){
      if( i < (int) buff_in_Send.size() ){
  out1 << from_Send.at(i) <<  " ";}else{out1 << 0 << " ";}}
  out1 << endl;
    for( int i = 0; i < (int) buff_in_Send.size() || i < max_length ; i++){
       if( i < (int) buff_in_Send.size() ){
  out1 << to_Send.at(i) <<  " ";}else{out1 << 0 << " ";}}
     out1 << endl;
    for( int i = 0; i < (int) buff_in_Send.size() || i < max_length ; i++){
       if( i < (int) buff_in_Send.size() ){
  out1 << Comm.at(0)->get_comm_amount(TG->get_edge_type(from_Send.at(i),
   to_Send.at(i))) <<  " ";}
  else{out1 << 0 << " ";}}
     out1 << endl;

/* In_put buffer */
 for( int i = 0; i < (int) buff_in_Recive.size() || i < max_length ; i++){
      if( i < (int) buff_in_Recive.size() ){
  out1 << from_Recive.at(i) <<  " ";}else{out1 << 0 << " ";}}
  out1 << endl;
```

```
    for( int i = 0; i < (int) buff_in_Recive.size() || i < max_length ; i++){
       if( i < (int) buff_in_Recive.size() ){
  out1 << to_Recive.at(i) <<  " ";}else{out1 << 0 << " ";}}
     out1 << endl;
    for( int i = 0; i < (int) buff_in_Recive.size() || i < max_length ; i++){
       if( i < (int) buff_in_Recive.size() ){
  out1 << Comm.at(0)->get_comm_amount(
   TG->get_edge_type(from_Recive.at(i),to_Recive.at(i))) <<  " ";}
  else{out1 << 0 << " ";}}
     out1 << endl;

out1 << endl;
out1.close();

}
 }; // end Write buffer
 }; // end buffer class
```

A.0.14 timing.hpp

```
#ifndef _timing_hpp
#define _timing_hpp

#include <sys/resource.h>

double gettimestamp();

#endif
```

A.0.15 timing.cpp

```
#include "timing.hpp"

double gettimestamp(){

/*struct timeval ru_utime   user time used
  struct timeval ru_stime   system time used */
  double time_1;
  struct rusage usage_1;
  getrusage(RUSAGE_SELF,&usage_1);
  time_1 = usage_1.ru_utime.tv_sec;
  time_1 += 1.0e-6*((double) usage_1.ru_utime.tv_usec);
  return time_1;
```

```
}
```