NTNU
Det skapende universitet

BSPlab - experiment manager (BEM)

Erlend Søreide Klepaker

Master i informatikk
Oppgaven levert: Juni 2006
Hovedveileder: Lasse Natvig, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Sammendrag

Dette dokumentet beskriver utviklingen av en grafisk eksperimentomgivelse for BSPlab. BSPlab er en parallell datamaskinsimulator, som gjør det mulig å simulere kjøringer av programmer skrevet for BSP-modellen (*Bulk Synchronous Paralell)* på forskjellige datamaskinarkitekturer. Målet med oppgaven er å utvikle grafiske omgivelser for denne simulatoren, som lar brukeren sette opp simuleringer ved hjelp av en rekke parametere, lar brukeren kjøre simuleringen og motta informasjon fra BSP programmet under kjøring og har verktøy for å la brukeren behandle resultatdata fra simulering visuelt etter kjøring. Utviklingen av denne grafiske eksperimentomgivelsen er i all hovedsak gjort i programmeringsspråket Python.

Abstract

This document describes the development of a graphic experiment environment for BSPlab. BSPlab is s parallel computer simulator, which makes it possible to simulate executions of programs written for the BSP-model *(Bulk Synchronous Parallel)* on different computer architectures. The goal with this assignment is to develop graphical environment for this simulator, which makes the user able to set up experiment, run these and make visual presentation of the results

Forord

Dette dokumentet er den endelige rapporten for min masteroppgave ved IDI, NTNU. Jeg vil her benytte sjansen til å rette en stor takk til min veileder på denne oppgaven Lasse Natvig, for god støtte og mange gode innspill underveis i prosessen.

Trondheim
1.juni 2006

Erlend Søreide Klepaker

Innholdsliste

Innledning

1.1 Orginal Problembeskrivelse (Oppgavetekst)

BSPlab - experiment manager (BEM)

Oppgaven går ut på å utvikle et grafisk grensesnitt for styring av BSPlab eksperimenter. Verktøyet har arbeidstittelen **BEM**. **BEM** skal la brukeren velge arkitektur, sette parametere i valgte arkitekturer, samt sette opp ett eller flere eksperimenter. Deretter startes eksperimentet, og valgte ytelsesparametere vises dynamisk på skjermen som grafiske plot etter hvert som eksperimentet kjører. Utprøving av ulike former for visualisering kan være aktuelt. Programmeringsspråket Python skal i hovedsak benyttes, videre er matplotlib meget aktuelt.

1.2 Tolking og endringer gjort i forhold til problembeskrivelse

Min oppgave går ut på å utvikle et grafisk brukergrensesnitt til BSPlab. Dette skal gjøres i Python. I hovedsak skal brukegrensesnittet tilpasses BSPlab slik den fremstår i dag, og minst mulig endringer skal gjøres på BSPlab. Underveis i prosessen valgte jeg i samråd med min veileder å flytt fokus fra dynamisk plotting under kjøring av et eksperiment og over på det plotteverktøyet som jeg har utviklet.

1.3 Dokumentet

Kapittel 1 Innledning

Dette er innledningskapittelet i dokumentet. Det inneholder problembeskrivelse, tolkning og endringer gjort i forhold til denne og denne oversikten over dokumentet.

Kapittel 2 Bakgrunn

Dette kapittelet inneholder aktuell bakgrunnsinformasjon for denne oppgaven.

Kapittel 3 Utvikling av et grafisk eksperiment omgivelse for BSPlab

I dette kapittelet blir selve utviklingen av systemet beskrevet nærmere.

Kapittel 4 Testing

Her blir prosessen med testing av det utviklede systemet beskrevet. Kapittelet har også en egen del som går nærmere inne på testing gjort for å finne ut hvorvidt systemet påvirker simuleringsresultatene i BSPlab

Kapittel 5 BSPlab tutorial

Dette er brukerguiden til systemet. Her blir systemets funksjonalitet og hvordan denne brukes beskrevet. Dette kapittelet er skrevet på engelsk.

Kapittel 6 Konklusjon

Avsluttingskapittelet som beskriver resultatet og erfaringer tilegnet gjennom prosessen med denne oppgaven.

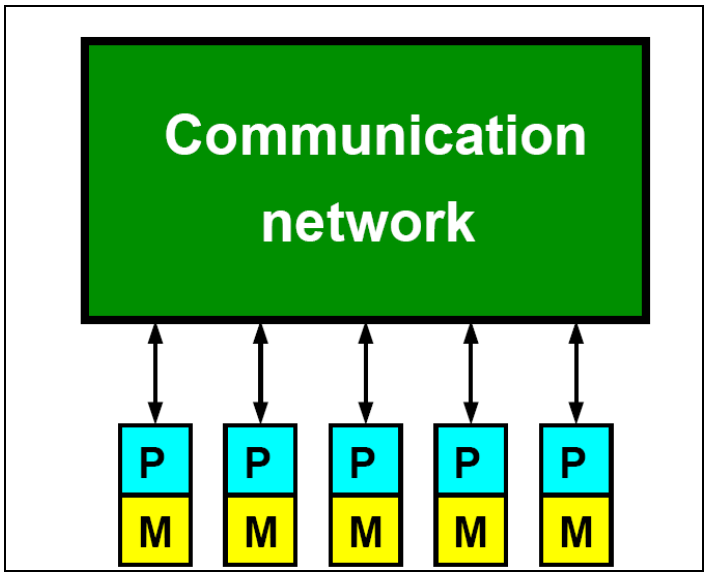
2 Bakgrunn

2.1 BSP modellen

BSP (*"Bulk Synchronous Parallel")* modellen ble foreslått av Leslie Valiant i 1990 og er beskrevet i artikkelen *" A Bridging Model for Parallel Computation"* [Valiant, 1990]. Tanken til Valiant var at BSP modellen skulle være et grensesnitt mellom programmerere og datamaskinarkitekter på parallelle datamaskiner, på den samme måten som Von Neuman modellen [Wikipedia, Von Neuman] er det for sekvensielle datamaskiner.

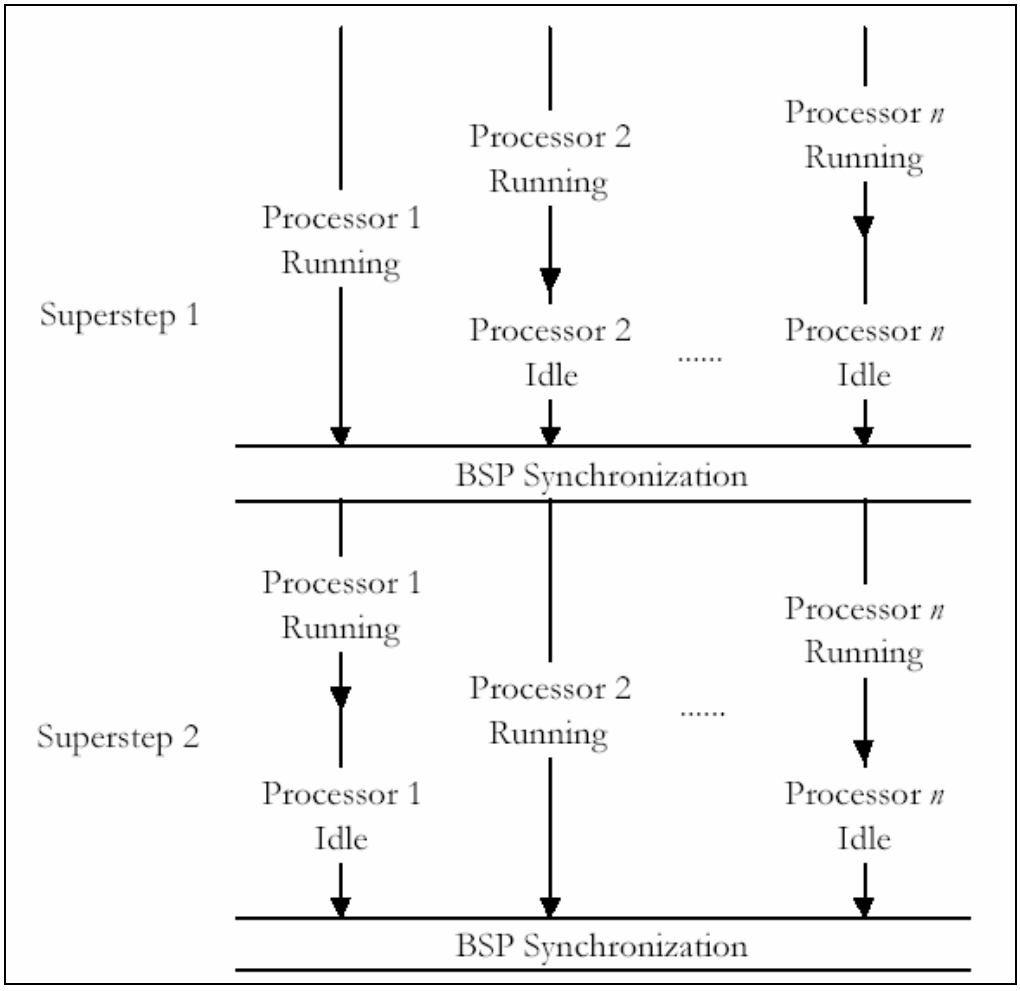
En parallell datamaskin beskrevet i BSP modellen består av følgende 3 elementer:

1. **Prosessorer** med tilknyttet minne.
2. Et **kommunikasjonsnettverk**, som sørger for kommunikasjon mellom prosessorene.
3. Mekanismer for å gjennomføre **barriere-synkronisering**.



Figur 2-1 Utformingen av en BSP datamaskin [Bisseling, 2005]

2.1.1 Barrieresynkronisering og superstep



Figur 2-2 Superstep og barrieriesynkronisering[Dybdahl,Uthus , 1997]

Kjøringen av et BSP program er delt opp i bolker, kalt superstep. Innenfor hvert superstep skjer databehandlingen på prosessorene asynkront. Når en prosessor er ferdig med sin databehandling innenfor en slik bolk, gjør denne et synkroniseringskall. Prosessoren som har gjort dette kallet vil ikke fortsette videre med databehandlingen, men vente på at alle de andre prosessorene i maskinen har er ferdige med sin databehandling. Når alle prosessorer har gjennomført dette kallet, utføres en barrieresynkronisering. På denne måten kan man forsikre seg at alle prosessorer har kommet like langt i kjøringen ved enden av hvert superstep.

2.1.2 Parametere som beskriver en BSP maskin

William McColl [McColl, 1990] har beskrevet 4 parametere som beskriver ytelsen til en BSP-maskin:

- ***s***- prosessorhastighet.
- ***p***- antall prosessorer.
- **l**- tiden en barriere synkronisering tar.
- ***g***- antall operasjoner utført av alle prosessorer pr. sekund delt på antall ord kommunikasjonsnettverket leverer på et sekund.

Ved hjelp av disse parametrene kan man beregne ytelsen til parallelle programmer på enhver BSP maskin.

2.2 BSP standard worldwide library

BSP standard worldwide library[] er et forsøk på å lage et standardisert bibliotek for BSP. Dette biblioteket inneholder 21 BSP funksjoner, som er et relativt lavt tall i forhold til andre parallelle biblioteker. Dette gjør at BSP er en oversiktlig og ikke for omfattende parallell modell å sette seg inn i.

BSPlab har ikke implementert siste versjon BSP standarden BSPlib v1.4, men versjon 0.72a. Man har derfor ikke tilgang til alle funksjonene i BSP standarden i BSPlab.

Siste versjon av BSPlib kom i september 1998, og siden den gang har det ikke skjedd noen videre utvikling av BSPlib.

.

2.3 BSPlab

BSPlab ble utviklet i diplomoppgaven til Haakon Dybdahl og Ivan Uthus i 1996 [Dybdal, Uthus 1996]. Jeg vil her gi en beskrivelse av de aspektene i BSPlab som er mest aktuelle for min oppgave. For en mer grundig innføring av BSPlab anbefaler jeg lesing av Lasse Natvigs artikkel ”*Simulating Parallel Architectures with BSPlab”* [Natvig, 2001].

Våren 2005, ble det gjennomført en oppdatering på BSPlab. Dette ble gjennomført av Erik Østby og Torje Lundereng, i deres avsluttende diplomoppgave ved NTNU. [Østby,Lundereng, 2005] Målet med denne oppdateringen var å gjøre det mulig å kjøre BSPlab på andre plattformer enn Microsoft Visual Studio. Det ble laget en ny versjon som kan kjøres på en kostnadsfri plattform kalt MinGW [MinGW,2006]. Det ble også laget en versjon av BSPlab som kan kjøres på Linux.

2.3.1 BSPlab Arkitekturer

Det er mulig å simulere BSP-programmer på en rekke forskjellige datamaskinarkitekturer i BSPlab. I denne delen vil jeg beskrive disse maskinene.

2.3.1.1 **Null**

I en nullmaskin er både kommunikasjon og barrieresynkronisering kostnader abstrahert vekk. Det eneste som kan ta tid, er kjøring av kode på de forskjellige prosessorene.

2.3.1.2 **Simple**

I motsetning til nullmaskinen der man abstraherer vekk kommunikasjonstid og barrieresynkroniseringstid, kan man her sette en fast verdi på disse kostnadene. Verdiene man kan sette er:

- tid for å sende en byte
- tid for å synkronisere en prosessor

2.3.1.3 **Distributed shared memory**

Her er prosessorene i BSP-maskinen koblet sammen på en buss. I BSPlab blir denne maskinarkitekturen derfor kalt en buss maskin.

2.3.1.4 Network (Tightly coupled multiprocessors)

Her er prosessorene koblet sammen ved hjelp av et nettverk. Det finnes en rekke forskjellige nettverkstopologier å velge mellom.

2.3.1.5 NOW (Network Of Worktstations)

Her har man en rekke arbeidsstasjoner som kommuniserer gjennom et Ethernet. Man kan også simulere støy på nettverket, som er annen trafikk som påvirker kommunikasjonen til BSP programmet.

2.3.2 BSPlab parametere

For å sette opp en simulering i BSPlab trenger man en parameterfil. Denne inneholder all informasjonen som BSPlab trenger for å kjøre et BSPlab-program. Se kapitel 5.4.2 i brukerveiledningen for nærmere informasjon om parameterne som brukes i BSPlab.

2.3.3 Måling av tidsbruk i BSPlab

Det finnes to forskjellige måter å måle tidsbruken til et BSPlab program i BSPlab.

- Automatisk
- Manuelt

Ved automatisk tidsmålingen måler BSPlab den faktiske tidsbruken på maskinen simuleringen kjøres på, og bruker denne for å beregne tidskostnadene ved en simulering.

Ved manuell tidsmåling, er det brukeren selv som forteller simulatoren hvor lang tid kjøring av simuleringen tar. Ved å bruke funksjonen bsp hold(sekunder) i koden til BSPlab-programmet kan brukeren fortelle simulatoren hvor lang tid kjøring av en gitt kodedel tar.

2.3.4 BSPlab resultatfil

Under kjøring av en BSPlab simulering produserer BSPlab en resultatfil, som innholder de resultatene brukeren har valgt å måle under simuleringen. For nærmere informasjon om hvilke verdier det er mulig å måle under simulering se kapittel 5.4.3.

Under vises en eksempelutskrift fra denne resultatfilen.

```
SuperstepTimeSpent;6;0;;;Start:;1.59852748e-005;Stop:;1.70509598e-005;Used:;1.06568498e-006
```

2.4 Andre grafiske omgivelser

Under arbeidet med denne oppgaven har jeg satt meg inn i en del andre grafiske omgivelser for parallelle systemer. Jeg har prøvd å finne andre omgivelser med noe av den samme funksjonaliteten som jeg skal lage i min oppgave. I dette kapittelet vil jeg beskrive disse omgivelsene. Disse grafiske verktøyene skiller seg fra det systemet jeg skal lage, ved at de brukes på faktiske datamaskiner, der jeg skal utvikle et system som skal ligge på toppen av en simulator. Dette gjør at de dataene jeg har tilgang til fra simulatoren, ofte er ganske annerledes og mer begrenset en hva disse systemene baserer seg på. Spesielt gjelder dette under kjøring av BSP-programmene.

Jeg har først og fremst prøvd å finne grafiske BSP omgivelser, men det er ganske begrenset hva som er laget fra før. Jeg har derfor også sett nærmere på noen grafiske omgivelser laget for andre parallelle datamodeller.

2.4.1 The Oxford BSP toolset profiling tool

The Oxford BSP toolset [The Oxford BSP toolset,1998] er en implementering av BSPlib, som kan kjøre på en rekke plattformer. Med denne følger det med noen forskjellige grafiske verktøy.

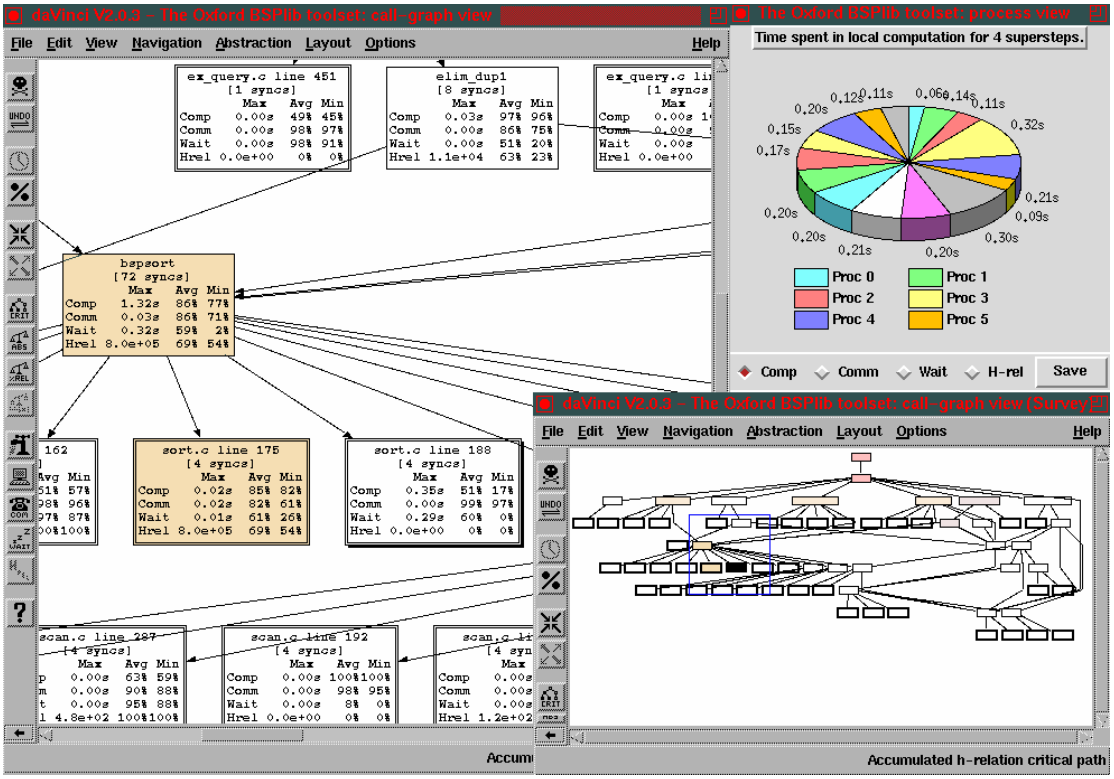
Disse er:

- call-graph tool

- performance profiler and prediction tool

BSP Toolset call-graph tool

Dette er et verktøy som analyserer informasjon produsert under kjøring av BSPlib-programmer. Målet med verktøyet er å finne ubalanse mellom forskjellige prosessorer [Hill, Jarvis Siniolkism, Vasilev, 1998]. Ideelt sett bør databeregning, og kommunikasjon fordeles jevnt over alle prosessorene i BSP-maskinen. Hvis det ikke er en slik balanse kan man finne ut det ved hjelp av dette verktøyet. Verktøyet påpeker også hvilke deler av koden i BSPlib-programmet som er kjørt som bidrar til denne ubalansen. Ved hjelp av dette verktøyet kan man altså optimalisere koden, slik at man får en best mulig balanse av databehandling og kommunikasjon mellom de ulike prosessorene i maskinen.

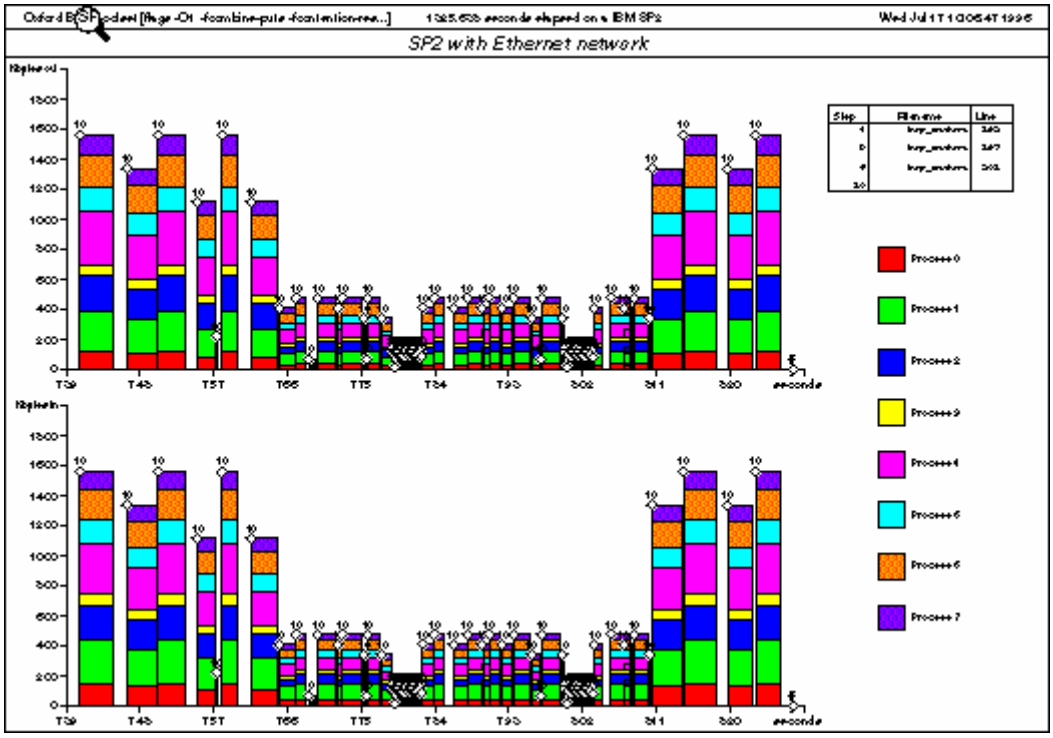


Figur 2-3 The Oxford BSP toolset call-graph tool, [BSP worldwide]

Figuren over viser tiden brukt på databehandling på forskjellige prosessorer i en BSPlib-maskin. Man har også merket ut deler av koden som fører til ubalanse mellom prosessorene.

BSP toolset performance profiler and prediction tool

Dette er et verktøy for å fremstille ytelsestall i grafisk form for et BSPlib-program. Det er også mulig å predikere ytelse på en tenkt BSP-maskin ved hjelp av dette verktøyet. Slik kan man forutsi teoretisk ytelse på en gitt BSP-maskin ved hjelp av ytelsesmodeller. Man kan dermed også sammenligne teoretisk ytelse med faktisk ytelse ved hjelp av dette verktøyet



Figur 2-4 The Oxford BSP toolset profiling tool, performance [BSP worldwide]

Eksempelfiguren over viser kommunikasjonsmengde og kommunikasjonstid over Ethernet for et BSPlib-program [Hill, Crumpton, Burgess, 1996].

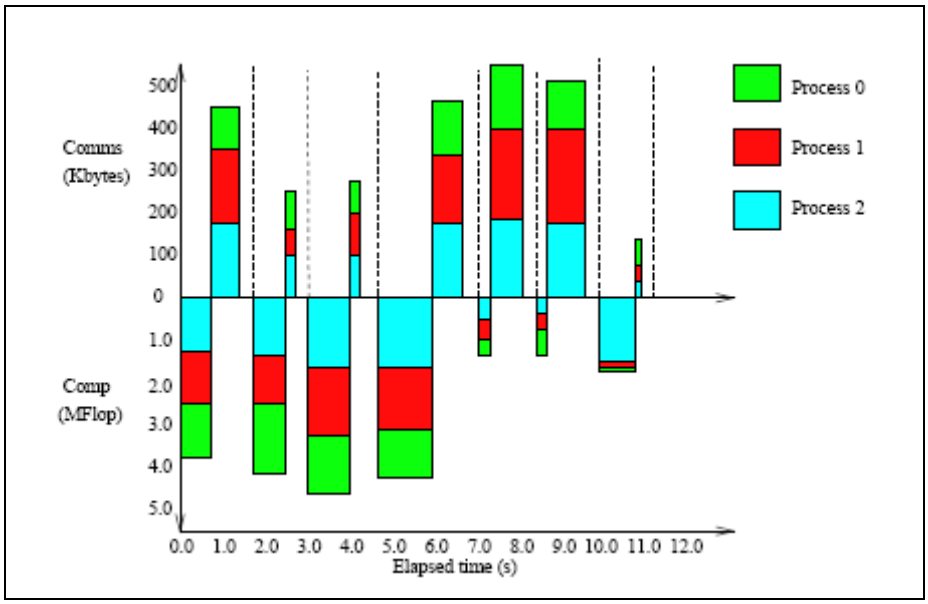
Det har ikke vært noe aktivitet på Oxford BSP Toolset prosjektet på mange år, (siste versjon kom i 1998), så det virker som at dette prosjektet har dødd ut.

2.4.2 Visual BSP Programing Enviroment for Distributed Computing

Dette er et system utviklet av Alex Wilson i hans masteroppgave ved Oxford University i 1999 [Wilson, Martin,2000]. Dette er først og fremst et grafisk system for å programmere og debugge BSPlib programmer utviklet i Java.

Selv om dette systemet ligger utenfor det systemet jeg skal utvikle når det gjelder funksjonalitet, har jeg valgt og nevne det her. Dette fordi Wilson beskriver en

del løsninger, under fremtidig arbeid, som faller mer inn under den funksjonaliteten i det systemet jeg skal utvikle. Han beskriver blant annet viktigheten av løsninger for å visualisere kostnader ved databeregning, kommunikasjon og synkronisering.



Figur 2-5 Kommunikasjon og databeregningskostnader [Wilson, Martin,2000]

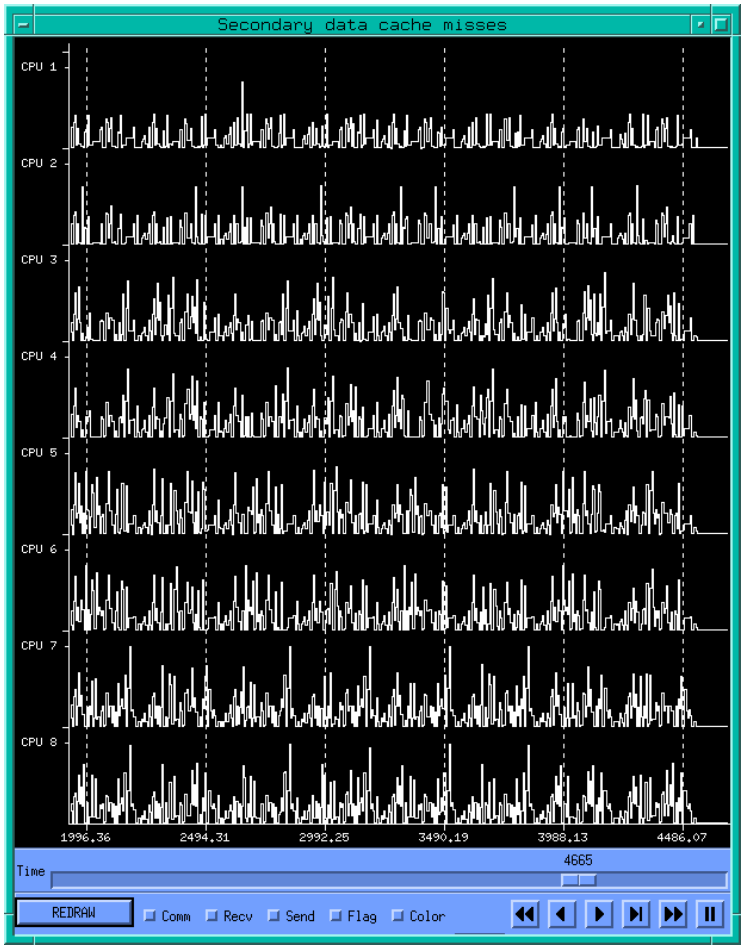
Figuren viser hvordan Wilson ser for seg at man kan presentere kostnader ved kjøring av BSP– programmer.

Det har ikke vært noe videre utvikling på prosjektet siden 1999.

2.4.3 PARAVER

Paraver er et verktøy utviklet ved European Center for Paralleism of Barcenlona ved universitetet i catalonie[CEBPA, 2006]. Dette er et verktøy som inneholder rikelig med funskjonalitet for å grafiske ytelsesmålinger. Verktøyet kan brukes på en rekke parallelle plattformer deriblandt MPI og OPENMP.

[CEBPA, 2006]. http://www.cepba.upc.es/paraver/index.html



Figur 2-6 Eksempelskjermbilde fra PARAVER

3 Utvikling av grafisk eksperimentomgivelse for BSPlab

I denne delen av rapporten vil jeg beskrive utviklingen av systemet jeg har laget. Jeg beskriver hvilke verktøy som er brukt, hvorfor, og de viktigste valgene og løsningene som er valgt under utviklingen. Man kan fort grave seg veldig ned i detaljer når man skal beskrive en slik utviklingsprosess. Jeg har derfor i størst mulig grad prøvd å beskrive hva som er laget. For detaljer rundt tekniske løsninger viser jeg til kildekoden som er vedlagt denne rapporten.

3.1 Funksjonsbeskrivelse

Før kjøring:

- Sette parametere i parameterfil
    - Velge arkitektur og vise aktuelle parametere for denne arkitekturen
        - Sette et utvalg av de mest brukte parametrene
        - Sette alle parametrene
- Sette opp et prosjekt
    - Velge et eller flere BSPLab program
    - Sette parameter å kjøre disse programmene på
        - Kunne sette opp flere kjøringer av det samme programmet, med forskjellige parameterverdier (eks. kjøre et program på N forskjellige arkitekturer og se hvordan resultatet variere fra en arkitektur til en annen)
        - Kunne kjøre et program på en parameterfil (en enkelt gjennomkjøring)
    - Lagre prosjektet i en egen prosjektfil (mittprosjekt.bpr)
- Starte kjøring av et prosjekt

Under kjøring:

- Generell output til et konsollvindu (feilmeldinger, kjøremeldinger med mer)
- Superstepteller som til en hver tid forteller brukeren hvilken superstep man er i
- Vise output fra prosessorer i egne prosessorvinduer (egen metode i BSPLab)
    - Brukeren har muligheten til å velge hvilke vinduer som skal vise
    - Tar vare på X siste utskrifter for hver prosessor, slik at man har muligheten til å gå tilbake å se utskrifter

Etter kjøring:

- Utskrifter samlet under kjøring som beskrevet over vil fortsatt være tilgjengelig, slik at man kan studere disse nærmere etter kjøring.
- En liste(eller annet passende element) som viser de ulike logg- grupperingene som brukeren valgte å logge før kjøring. Ved hjelp av denne kan brukeren:
    - Vise loggdata i graf form
        - Eks. vise graf over tidsforbruk pr. prosessor pr superstep

3.2 Fremgangsmåte

Jeg hadde lite kjennskap til Python og BSPlab når jeg startet på denne oppgaven. Så hele utviklingsprosessen ble også en læringsprosess der det meste måtte læres underveis. Dette påvirker selvfølgelig måten man kan gå frem på, for å utvikle et slikt system.

Når jeg skal beskrive utviklingsprosessen har jeg delt den inn i 4 deler:

- Setup
- Kjøreomgivelsen
- Graf-verktøy
- Rapportverktøy

Disse delene tilsvarer de ulike skjermbildene i systemet, og ble utviklet hver for seg. Utviklingen av de enkelte delene ble gjort stegvis, der jeg først laget en enkel prototype og deretter la til mer og mer funksjonalitet. Til slutt ble de ulike delene satt sammen til et system.

3.3 Valg av verktøy

Jeg har brukt en rekke forskjellig verktøy i denne oppgave. Hovedtyngden har naturlig nok ligget på Python, men også en del andre verktøy har vært nødvendige for å kunne løse oppgaven.

3.3.1 Python

Det var gitt i oppgaven at Python var det programmeringsspråket som i hovedsak skulle brukes i utviklingen av systemet. Python egner seg godt til en slik oppgave fordi det i utgangspunktet er en ganske lett programpakke, men at man kan legge til ekstra funksjonalitet ved hjelp av utvidelsespakker. Dermed egner Python applikasjoner seg godt til å ligge på toppen av andre applikasjoner.

3.3.2 Utvidelsespakker i Python

Det finnes et rikt utvalg av utvidelsespakker man kan legge til Python, for å få ekstra funksjonalitet. Siden jeg ikke hadde noe kjennskap til Python før jeg startet på denne oppgaven, måtte jeg også sette meg inn i hvilke utvidelsespakker som burde brukes. Det å velge mellom forskjellige pakker er ofte ikke noen nøyaktig vitenskap, og i flere av tilfellene der jeg har sammenlignet forskjellige pakker, har disse kommet like godt ut når det gjelder funksjonalitet. Jeg har derfor i tillegg til å sette meg inn i de forskjellige pakkene på deres hjemmesider, brukt diverse internettfora for å finne ut hva andre brukere mente om de forskjellige pakkene.

3.3.2.1 GUI pakke

Det finnes en rekke alternative grafikkpakker som kan brukes for å lage grafiske brukergrensesnitt i Python. Det var viktig for meg å velge en pakke som hadde god dokumentasjon. I tillegg ville jeg velge en pakke som kunne fungere på flere plattformer enn Windows. Selv om jeg i min oppgave skulle lage en Windows applikasjon, ville det vanskeliggjøre eventuelle senere versjoner på andre plattformer, hvis jeg hadde valgt en GUI pakke som bare fungerte på Windows.

Etter å ha undersøkt en del rundt GUI verktøy i Python, var det to GUI-pakker som pekte seg ut som spesielt aktuelle. Disse var Tkinter [Tkinter, 2006] og wxPython [wxPython, 2006]. Begge disse grafikkpakkene fungerer på de fleste plattformer, og inneholdt den funksjonaliteten jeg vil trenge. I tillegg var det mulig å integrere figurer laget både i Matplotlib og scipy (se: kapittel 3.3.2.2 for mer informasjon) ved hjelp av disse grafikkpakkene.

Tkinter har den fordelen at den følger med standard distribusjonen av Python. Så dette er egentlig ikke en utvidelsespakke slik som wxPython. En annen fordel er at Tkinter har vært en del av Python lenge, og dermed er godt utprøvd og uttestet. En fordel som også ble nevnt i diverse forum på internett var at Tkinter er lett å sette seg inn i.

wkPython er basert på C++ verktøyet wxWidgets og er en utvidelse til Python. wxPython er bra dokumentert, og man kan også bruke dokumentasjonen til wxWidgets. Selv om denne dokumentasjonen er skrevet for C++ er det ikke noe problem å bruke den for Python versjonen. De forskjellene som er mellom C++ versjonen og Python versjonen går stort sett igjen gjennom dokumentasjonen. (Eks: wxWindow som er et vindu-objekt i wxWidgets, skrives wx.Window i wxPython). I tillegg finnes det en god samling av eksempelprogrammer skrevet i wxPython. Ved hjelp av denne kan man sette seg inn i hvordan man programmerer de mest brukte elementene i wxPython.

Tkinter er ”*lett å lære, men vanskelig å bruke”*. Med dette menes at selv om det er lett å sette seg inn i Tkinker og å lage enkle brukergrensesnitt med det, blir det fort komplisert når man skal skalere opp og lage mer komplekse brukergrensesnitt. Mens wxPython er vanskeligere å sette seg inn i, men når du først har forstått hvordan wxPython brukes, er det et kraftfullt verktøy for å lage mer kompliserte brukergrensesnitt.

Valget falt til slutt på wxPython. Begge GUI verktøyene ville kunne fungert bra for min oppgave. Grunnen til at jeg til slutt valgte wxPython over Tkinter var at jeg likte wxPython dokumentasjonen bedre en dokumentasjonen til Tkinter. Jeg fikk også et generelt inntrykk av at wxPython var et kraftigere verktøy med flere muligheter en Tkinter. Og at Tkinter ble sett på som gårsdagens teknologi, mens wxPython er fremtiden.

Jeg avslutter denne delen med et sitat fra mannen som har laget Python:

> ***“wxPython is the best and most mature cross-platform GUI toolkit, given a number of constraints. The only reason wxPython isn't the standard Python GUI toolkit is that Tkinter was there first”.***
>
> **–** Guido van Rossum

3.3.2.2 **Plottepakke**

En viktig del av denne oppgaven var å kunne la systemet fremstille simulerings-resultatene visuelt. For å kunne gjøre dette trengte jeg en utvidelses pakke som gjorde dette mulig i Python. Matplotlib [MatPlotlib, 2006] er allerede nevnt i oppgaveteksten, mens en annen pakke kalt scientific Python (SciPy)[SciPy, 2006] også var et alternativ.

Etter å ha gått igjennom de dataene som BSPlab produserer, og dermed de dataene som var aktuelle for fremstilling, ble det klart at begge pakkene hadde rikelig

med funksjonalitet for min oppgave. Så igjen ble det snakk om smak og behag. Valget falt til slutt på matplotlib. Grunnen til at det var denne plottepakken som til slutt ble valgt var at den var godt dokumentert, hadde en bra samling med eksempelprogrammer, og at det virket som at miljøet rundt programvaren var veldig aktivt. Når det gjaldt SciPy hadde det ikke kommet noen versjon til den siste utgaven av Python da jeg startet på oppgaven min. Dette ville si at hvis jeg skulle bruke denne pakken måtte jeg også ha brukt en eldre utgave av Python. (I etterkant er det kommet en ny versjon av scipy til siste utgave av Python).

3.3.2.3 **Andre Pythonpakker brukt**

Jeg har også tatt i bruk noen andre Pythonpakker. En av de viktigste er ReportLab [ReportLab, 2006]. Dette er et verktøy for å kunne generere PDF dokumenter i Python. Denne Pythonpakken er brukt i rapportverktøyet i systemet.

Numeric Python [NumericPython, 2006] er en pakke som kreves av matplotlib, og er ikke brukt av meg direkte.

Python Imaging Library (PIL) [PIL, 2006] er en pakke som ReportLab trenger for å kunne behandle bilder i rapportene som genereres.

3.3.3 Verktøy for distribusjon

3.3.3.1 **Py2Exe**

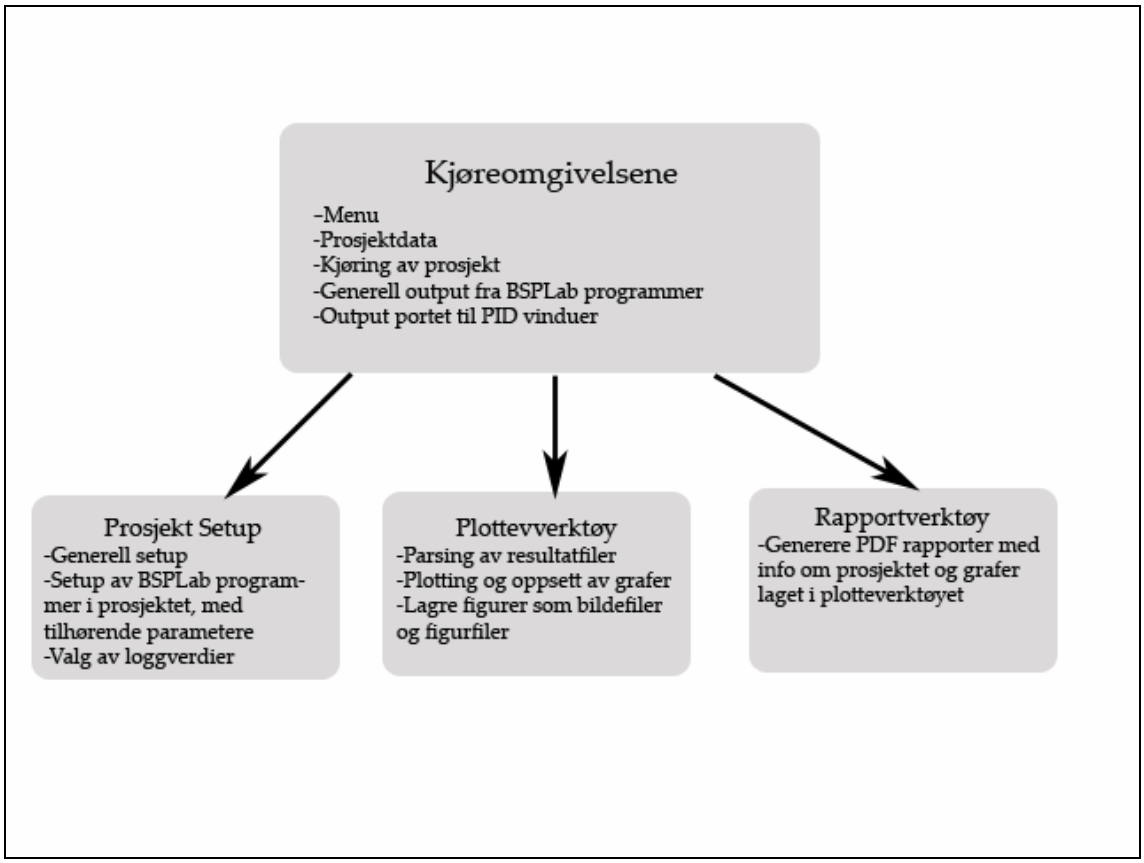
Py2Exe [Py2exe] er en utvidelse av Python Distribution Utilities (fork:Distutils) [Distutilis] som gjør det mulig å generere kjørbare Windows programmer av et Python script. Fordelen med dette er at brukeren ikke trenger å ha Python, eller noen av utvidelsespakkene som er brukt, installert for å kunne kjøre programmet. Dette forenkler distribusjon veldig, da man kan forholde seg til systemet som en enhet, og ikke lenger er avhengig av andre programmer for å kjøre systemet.

3.3.3.2 **Inno setup**

Inno Setup [Inno Setup, 2006] er et verktøy for å lage installasjonsveivisere til Windows. Ved å bruker dette verktøyet kan installasjonen gjennomføres på en enkel og lettfattelig måte, som er godtkjent for de fleste Windows-brukere.

3.4 Oppbygningen av systemet

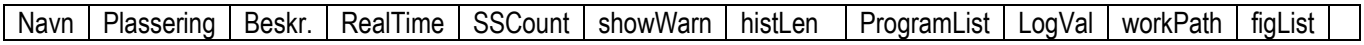
Som nevnt i innledningen av dette kapitelet, er systemet jeg har utviklet delt i 4 deler. Hver enkelt av disse delene er et eget skjermbilde med tilhørende funksjonalitet. Se Figur 3-1. Videre i dette kapittelet skal jeg beskrive nærmere funksjonaliteten i de forskjellige delene av systemet, hvordan skjermbildene er utformet og hvilke løsninger som er valgt når disse skulle implementeres.



Figur 3-1 Oppdelingen av systemet

3.4.1 Overliggende oppbygning

Systemet har en datastruktur som inneholder all informasjonen systemet trenger under kjøring. Ved hjelp av denne kommuniserer de forskjellige delene av systemet med hverandre. Det er også denne datastrukturen som ligger til grunn når man skal lagre og åpne et BSPlab prosjekt.



Figur 3-2 Oppbygning av prosjektdatastruktur

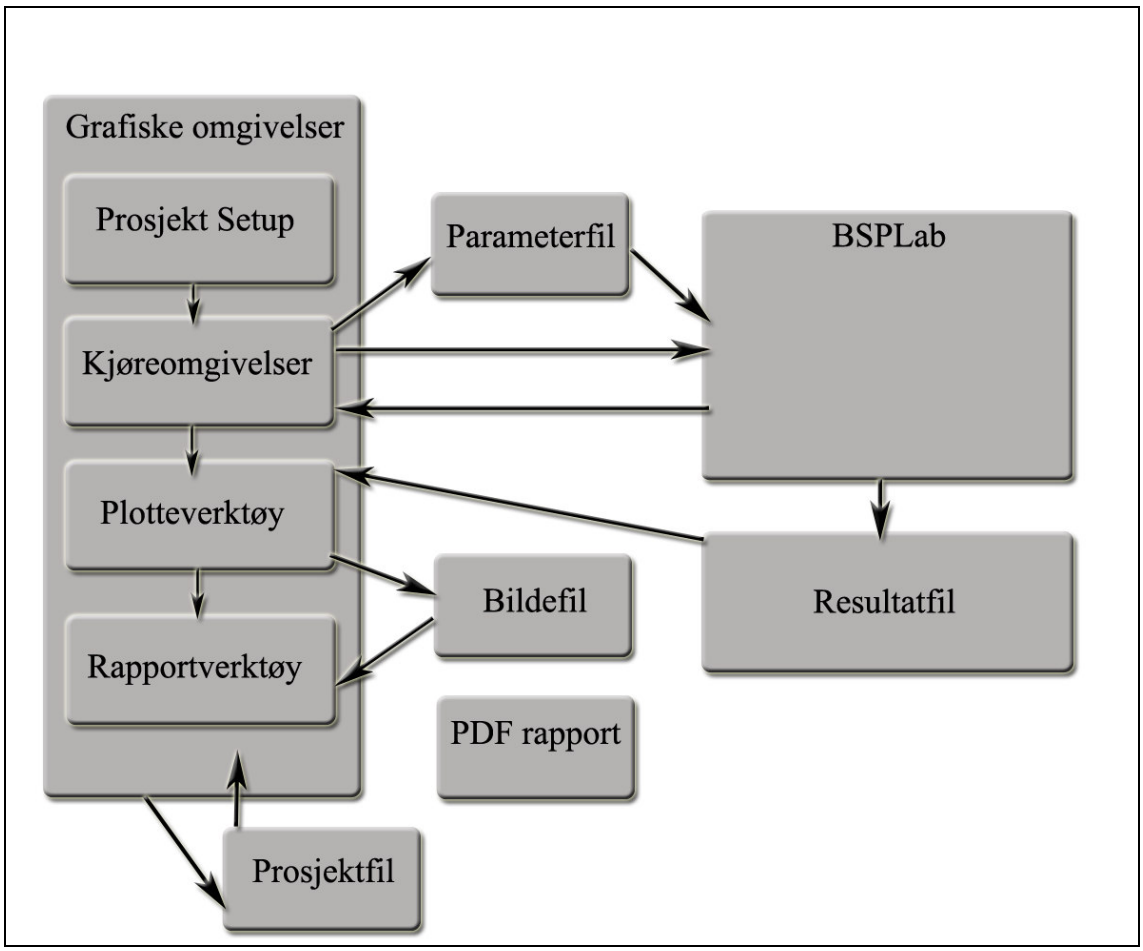
Forklaring til figur Figur 3-2:
Figuren viser hoved-datastrukturen som innholder all informasjonen som systemet trenger under kjøring.

Det første feltet er navnet på prosjektet, det andre er hvor prosjektet er plassert. Feltene Beskr., RealTime, showWarn er nærmere beskrevet i kapittel 3.4.3.2 om *generell setup.*

ProgramList er en datastruktur som inneholder all informasjonen om BSPlab programmene i prosjektet og er beskrevet i kapittel 3.4.3.3 *Setup av BSPlab-programmer i prosjektet.*

WorkPath er plasseringen av systemet. Denne er nødvendig for å vite hvor eksterne filer som blir brukt under kjøring er plassert.

figList er en liste over alle figurene som er laget i systemet og hvor de er plassert. Beskrevet nærmere i kapittel 3.4.4 *Plotteverktøy.*



Figur 3-3 Kobling mellom systemet, BSPlab og eksterne filer

Figur 3-3 viser koblingen mellom den grafiske eksperimentomgivelsen jeg har laget, og BSPlab.

Før kjøringen av et BSPlab-program kan starte, produserer systemet en parameterfil som inneholder alle parametrene BSPlab trenger. Denne filen leses av BSPlab før kjøringen av programmet starter.

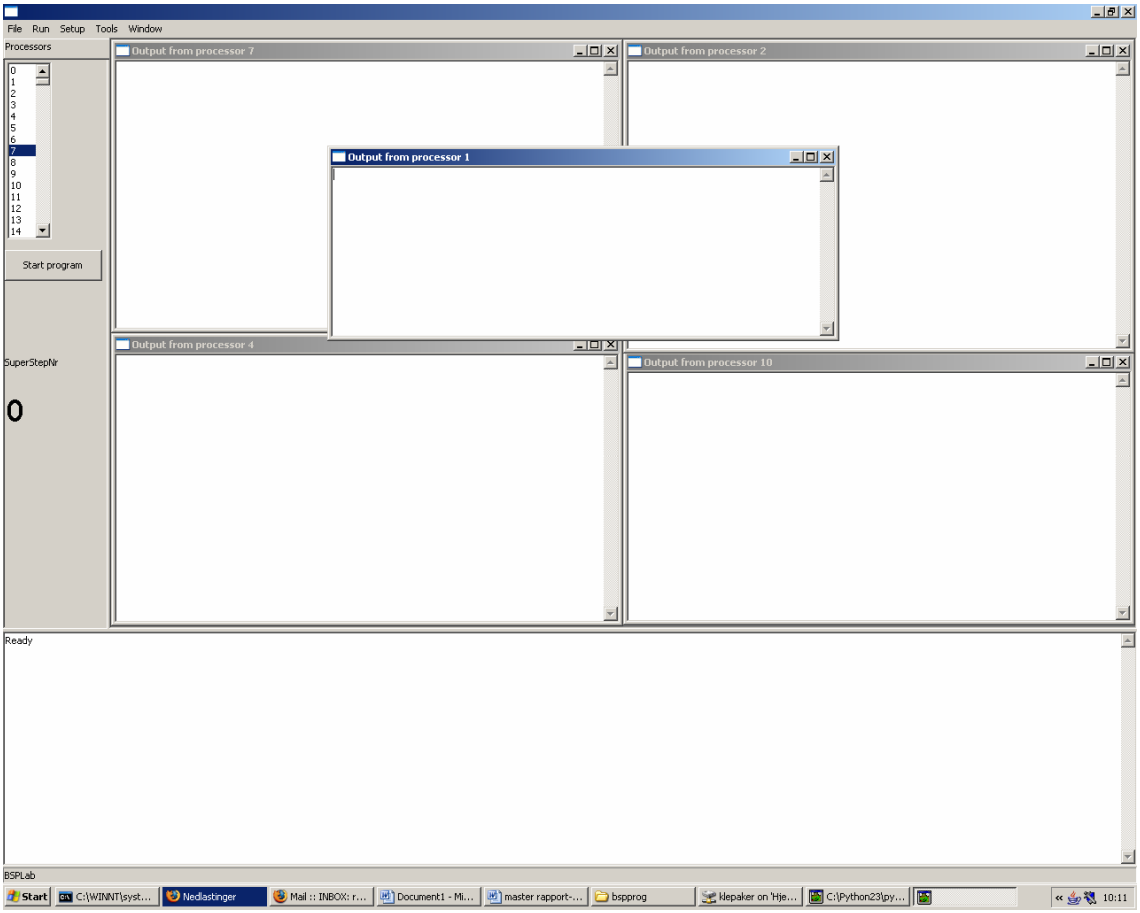
Fra kjøreomgivelsen setter systemet i gang kjøringen av BSPlab, og under kjøringen mottar kjøreomgivelsene utskrifter fra BSPlab. BSPlab produserer en resultatfil som inneholder valgte loggverdier. Denne prosessen gjentas for hvert program i prosjektet.

Når brukeren åpner plotteverktøyet leses alle resultatfilene av plottedelen i systemet. I plottedelen produseres bildefiler, som senere kan brukes i rapportverktøyet. All informasjonen i de grafiske omgivelsene, kan lagres i en prosjektfil. Denne kan senere åpnes av systemet. Kommunikasjonen internt i systemet skjer ved hjelp av prosjektdatastrukturen beskrevet tidligere i dette kapittelet.

3.4.2 Kjøreomgivelsen

Denne delen av systemet var den første som ble utviklet. Det er herfra man starter, kjører og mottar output fra BSPlab programmer. Skjermbildet til denne delen av systemet er toppvinduet i systemet. Det er her menylinjen og prosjektdatastrukturen er plassert. Dette vinduet er også foreldrevindu for setupvinduet, rapportvinduet og plottevinduet.

3.4.2.1 Utforming av skjermbildet



Figur 3-4 Skjermbildet til kjøreomgivelsen

Siden dette er toppnivåvinduet i systemet er menyen plassert i dette vinduet. Dvs. at de andre vinduene i systemet blir plassert inne i dette vinduet. Man har altså alltid tilgang til menylinjen som ligger i toppen på dette vinduet.

Vinduet er av typen *MDI parent frame*. MDI står for *multiple document interface* [MDI Wikipedia, 2006], og er en løsning som gjør at dette vinduet kan romme andre vinduer i sitt klientområde. Se eksempel Figur 3-4. Grunnen til at denne løsningen ble valgt, var at jeg trengte en fleksibel måte å skille utskrift fra forskjellige prosessorer i BSPlab-programmene som blir kjørt. Med denne løsningen kan brukeren selv velge hvilke prosessorvinduer han vil vise, og hvordan han vil plassere disse innenfor skjermbildet.

Brukeren velger hvilke prosessorvinduer han vil åpne ved å dobbeltklikke på prosessornummeret i listen til venstre i skjermbildet. All utskrift som ikke skal rute til et prosessorvindu blir skrevet ut i hovedutskriftsrammen i bunnen av skjermbildet. For å starte kjøringen av BSPlab-programmene trykker brukeren på knappen *"Start programs"*. Skjermbildet har også en superstepteller, som viser hvilket superstep man er i under kjøring av BSPlab-programmene.

3.4.2.2 Teknologien bak skjermbildet

3.4.2.2.1 Generering av parameterfiler

Før man kan starte kjøringen av et BSPlab program trenger man en parameterfil. (Nærmere info se kapittel 3.4.3). Denne inneholder all informasjonen BSPlab-programmet trenger. Derfor må systemet produsere en slik parameterfil før kjøringen av et BSPlab-program starter. Måten dette gjøres på i systemet er at man leser parametrene fra programdatastrukturen, og skriver disse linje for linje i en ny fil kalt parameters.dat. Denne filen blir plassert i mappen til BSPlab-programmet.
I de tilfellene der brukeren har valgt å kjøre det samme BSPlab-programmet flere ganger, men med endringer av parametrene, blir det produsert en ny parameterfil (den forrige blir overskrevet) mellom hver kjøring av programmet.

3.4.2.2.2 Kjøring av BSPlab programmer ved hjelp av en pipeline

Får å kjøre hvert enkelt BSPlab-program settes det opp en pipeline mellom systemet og BSPlab-programmet. I Python gjøres dette ved hjelp av funksjonen: *os.popen().* Når man bruker denne funksjonen blir BSPlab-programmet startet, og man får tilgang til all utskrift til standard output og standard error(stdout, stderr) som er utskriftstrømmer fra BSPlab-programmene. Disse utskriftstrømmene leses så linje for linje. Hvis linjen har en merkelapp(se kapittel 3.4.2.2.4) i starten parses linjen slik at man finner ut hvilket prosessorvindu den skal sendes til, hvis ikke sendes hele linjen til hovedutskriftsrammen. Slik behandles all utskrift til BSPlab programmet er terminert.

Pipelinen som åpner BSPlab-programmene befinner seg i en egen tråd. Dette er nødvendig for å kunne mate utskriftene dynamisk til brukergrensesnittet under kjøring. Hvis man hadde åpnet denne pipelinen fra hovedtråden til programmet, ville brukergrensesnittet "frosset" under kjøring. Ingen interaksjon fra brukeren ville vært mulig.

3.4.2.2.3 Bufring av utskrift fra BSPlab

Siden utskriftene skal vises dynamisk under kjøring er det viktig at disse ikke blir bufret før de ankommer systemet. Dette var et problem i starten av utviklingen. Utskriftene kom i grupper og ikke i en jevn strøm slik som det var ønskelig. Lenge trodde jeg at bufringen skjedde i Python. Det er mulig å sette opp en pipeline uten bufring, men selv om dette ble gjort ble utskriftene bufret ett eller annet sted. Det viste seg etter hvert at det var utstrømmene fra BSPlab som ble bufret. Etter mye om og men fant jeg ut hvordan man kunne skru av bufringen i BSPlab. Ved å legge til linjen:

```
setvbuf ( stdout , NULL , _IOFBF , 1024 )
```

Dette sørget for at utskriftene kom i en jevn strøm, slik som det var ønskelig.

3.4.2.2.4 Ruting av utskrifter til BSPlab

Det var ønskelig med en funksjon som gjorde det mulig å separere utskrifter ut fra hvilken prosessor i simulatoren som produserte den. For å kunne virkeliggjøre dette måtte jeg legge til en ekstra utskriftsfunksjon i BSPlab.

```
pid_print(message, args)
```

Denne funksjonen fungerer på den samme måten som C++ funksjonen *printf* [cplusplus.com, 2006]

Det funksjonen gjør er at den legger til en merkelapp(*tag*) først i utskriften. Denne merkelappen forteller at dette er en utskrift som skal rutes til et prosessorvindu, og hvilket prosessorvindu den skal rutes til.

Under kjøring er det kun mulig å porte prosessorutskrifter til vinduer som er åpne. Jeg måtte derfor lage en datastruktur som tok vare på utskriftene til de prosessorvinduene som ikke var åpne. Denne ble implementert som en dobbeltlinket liste for hvert prosessornummer. Brukeren kan selv velge hvor mange utskrifter man maksimalt skal ha i historikken til hver prosessor. Dette fordi man i en situasjon med mange prosessorer og mange utskrifter, ville kunne oppleve at det ble meget ressurskrevende å skulle ta vare på alle utskriftene.

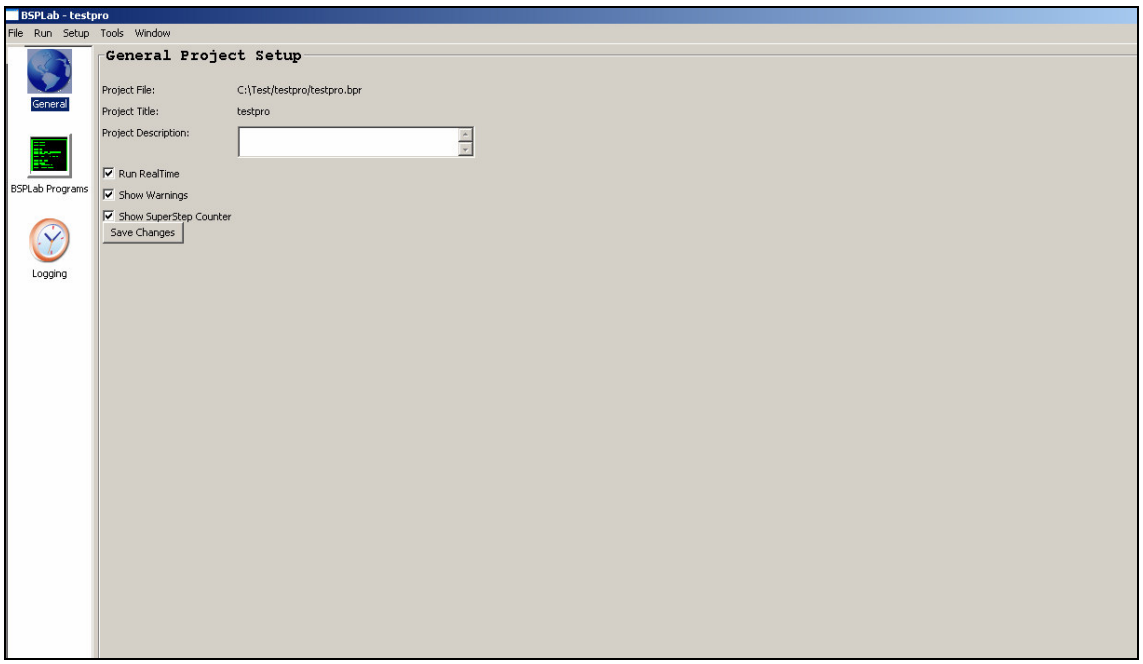
Eks: Hvis brukeren velger å lagre 10 utskrifter i historikken til hver prosessor, vil de 10 siste utskriftene automatisk komme frem ved åpning av et prosessorvindu. Dette kan være under, eller etter kjøring av et BSPlab program.

I de prosessorvinduene som er åpne under kjøring, vil ikke denne begrensningen av antall utskrifter gjelde. Her vil man ha tilgang til alle utskriftene så lenge prosessorvinduet er åpent.

3.4.3 Prosjektoppsett

Setupdelen er en viktig del av systemet. Det er her man velger hvilke BSPlab-programmer som skal være med i prosjektet, setter parametrene for disse programmene og bestemmer hvilke verdier som skal logges under kjøring. Setupdelen av systemet er tett knyttet opp til parameterfilen i BSPlab. Det er her brukeren bestemmer hvilke verdier denne filen skal inneholde.

3.4.3.1 Utforming av skjermbildet



Figur 3-5 Setup skjermbildet

Setupdelen av systemet er tredelt. I den første delen legger man inn en del generelle opplysninger som gjelder for hele prosjektet. Den andre delen er der man legger til BSPlab-programmer til prosjektet og setter parametrene for disse. I den tredje delen velger man hvilke verdier som skal logges under kjøring.

For å navigere mellom disse tre delene trykker man bare på det riktige ikonet i menyen på venstresiden av skjermbildet. I hver enkelt del er det en lagreknapp som brukeren må trykke på for å lagre endringer.

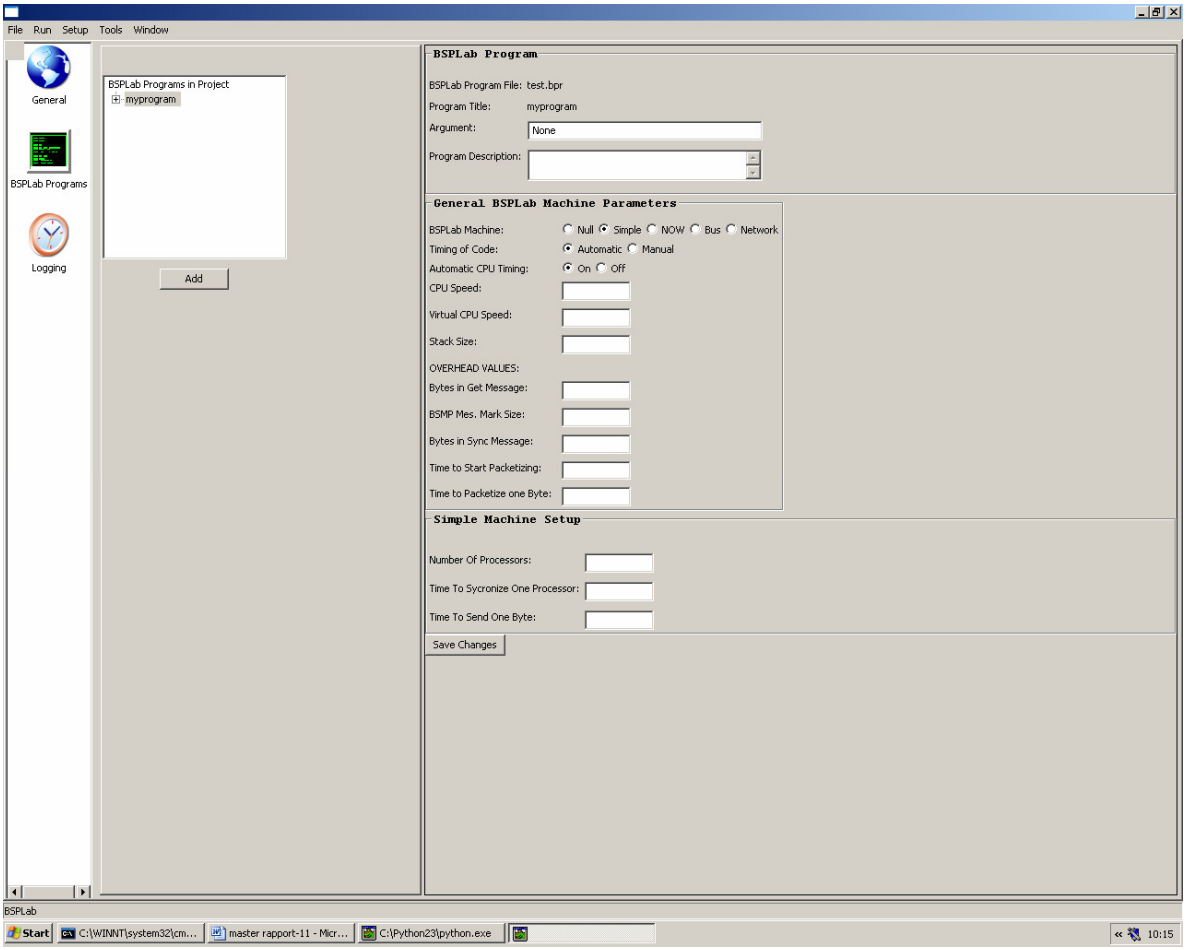
3.4.3.2 Generell Setup

Her vises navnet på prosjektet og hvor det er plassert. Man kan legge til en beskrivelse av prosjektet som kan brukes i rapportverktøyet. I tillegg bestemmer man også her hvorvidt man skal skrive ut advarsler under kjøring av BSPlab-programmene og om man skal telle opp superstep i supersteptelleren. Man kan også velge å gi BSPlab programmene *realtime priority*. Dette gir BSPlab-programmene prioritet over andre prosesser på maskinen simuleringen foregår på. Man kan også legge til hvor mange

utskrifter man skal lagre i historikken til prosessorutskrift vinduene. (Se kapittel 3.4.2.2.4)

3.4.3.3 Setup av BSPlab-programmer i prosjektet

Her legger man til nye BSPlab-programmer til prosjektet og setter parametere for disse.



Figur 3-6 Skjermbildet for programsetup

Det er to forskjellige måter man kan legge til BSPlab-programmer til prosjektet på i systemet:

- Legge til kopi av ferdig kompilert BSPlab-program
- Legge til et DevC++ prosjekt

Når man legger til en kopi av et ferdig kompilert BSPlab-program opprettes en ny mappe i prosjektmappen. BSPlab-programmet kopieres så til denne mappen. Grunnen til at jeg har valgt å kopiere programmet til prosjektmappen, istedenfor å peke til programmet der det befinner seg, er at hvis man senere gjør endringer i dette programmet vil ikke disse endringene påvirke prosjektet. Dvs. at programmet må være ferdigutviklet før man legger det til prosjektet.

Ved å legge til et DevC++ prosjekt istedenfor en kopi av et BSPlab- program har brukeren mulighet til å gjøre endringer i programmet etter at det er lagt til prosjektet. Dette DevC++ prosjektet blir lagret i en ny mappe i prosjektmappen, og DevC++ blir åpnet av systemet. Brukeren kan da skrive et nytt BSPab-program, eller legge til et eksisterende, og kompilere dette.
Med en gang programmet er kompilert vil dette blir kjørt på den samme måten som om man hadde lagt til en kopi av et program.

Programmene som blir lagt til i prosjektet vises i et programtre.(Se Figur 3-8)

Når et program er lagt til prosjektet blir informasjonen om dette lagt til i en programliste i prosjektdatastrukturen (se Figur 3-2 Oppbygning av prosjektdatastruktur).

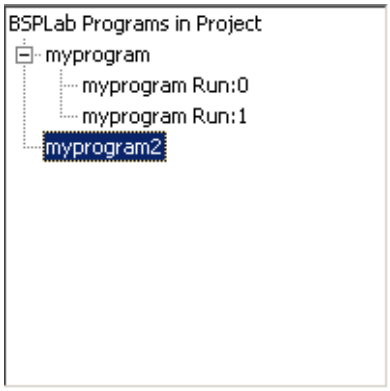
[[tittel] [beskrivelse] [argument] [[gen. param]] [[maskinparam]]]

Figur 3-7 Programdatastruktur

Denne datastrukturen inneholder tittelen på programmet. Brukeren kan også legge til en beskrivelse av programmet og et argument til programmet som skal kjøres. Når brukeren har satt parametrene for programmet vil disse også bli lagt inn i programdatastrukturen. Parametrene blir lagt inn i to forskjellige lister. Generelle parametere som gjelder for alle BSPlab-programmer og de parametrene som er avhengig av hvilken type BSPlab maskin brukeren har valgt.

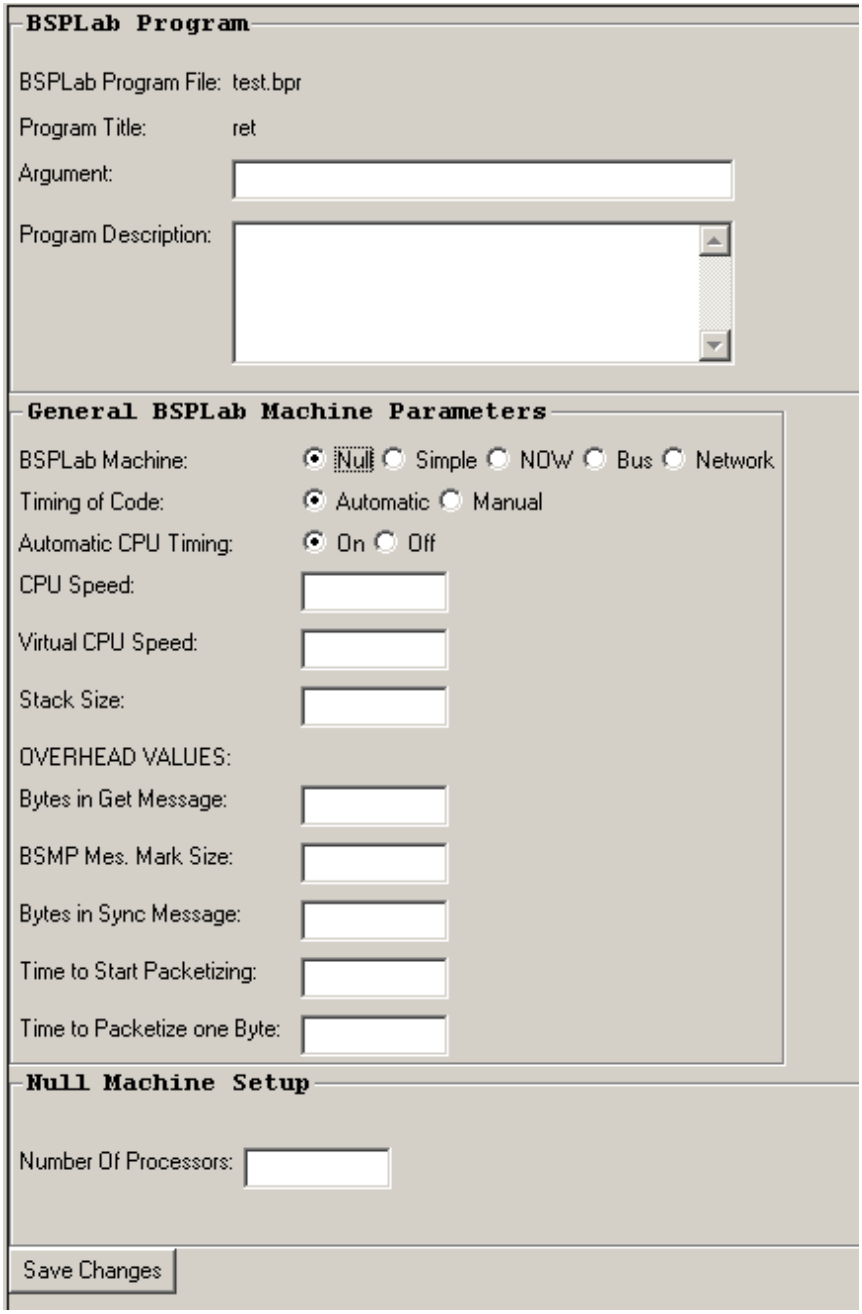
I tillegg til at man kan legge til flere ulike BSPlab-programmer i et prosjekt, kan man legge til flere kjøringer av det samme programmet. Dette gjøres ved å høyreklikke på programmet man vil legge til en kjøring på i program-treet.(se Figur 3-8) Det vil da bli lagt til et nivå til i treet der antall kjøringer vises (se Figur 3-9).

Figur 3-8 Trestruktur som viser programmene i prosjektet



Figur 3-9 Trestruktur etter at ekstra kjøringer er lagt til

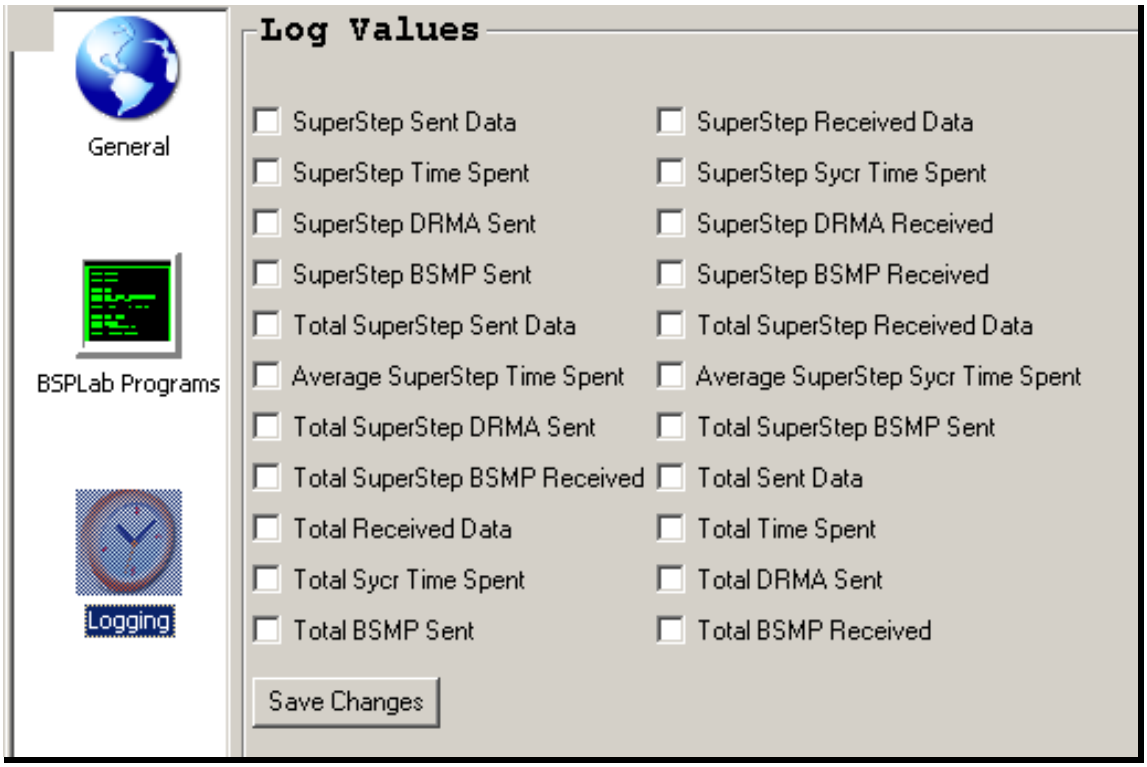
Når brukeren har lagt til ett eller flere programmer, eller flere kjøringer av det samme programmet må brukeren sette parametrene for disse. Dette gjøres ved at brukeren velger programmet eller kjøringen i trestrukturen som inneholder alle programmene, med ekstra kjøringer. Brukerne fyller så inn verdiene til de ulike parametrene. Den første delen av parametrene er generelle parametere som gjelder for alle BSPlab programmer, den andre delen er knyttet opp til hvilken type BSPlab maskin brukeren har valgt. Skjermbildet forandrer seg etter hvilken maskintype brukeren har valgt. Så kun de parametrene som er knyttet til en valgt maskintype vises. Når brukeren har satt alle parametrene trykker han på en lagreknapp og verdiene blir lagt til programdatastrukturen. Dette må gjøres for alle programmer/kjøringer i prosjektet.



Figur 3-10 Del av skjermbilde der brukeren setter parametere for et BSPlab-program

3.4.3.4 Oppsett av Loggverdier

I denne delen av systemet velger brukeren hvilke verdier som skal logges underkjøring. Dette gjøres enkelt ved å krysse ut checkboksene til de ønskede loggverdiene.
Når de ønskede loggverdiene er valgt trykker brukeren på lagreknappen og en liste med alle valgte loggverdier blir lagt til prosjektdatastrukturen.

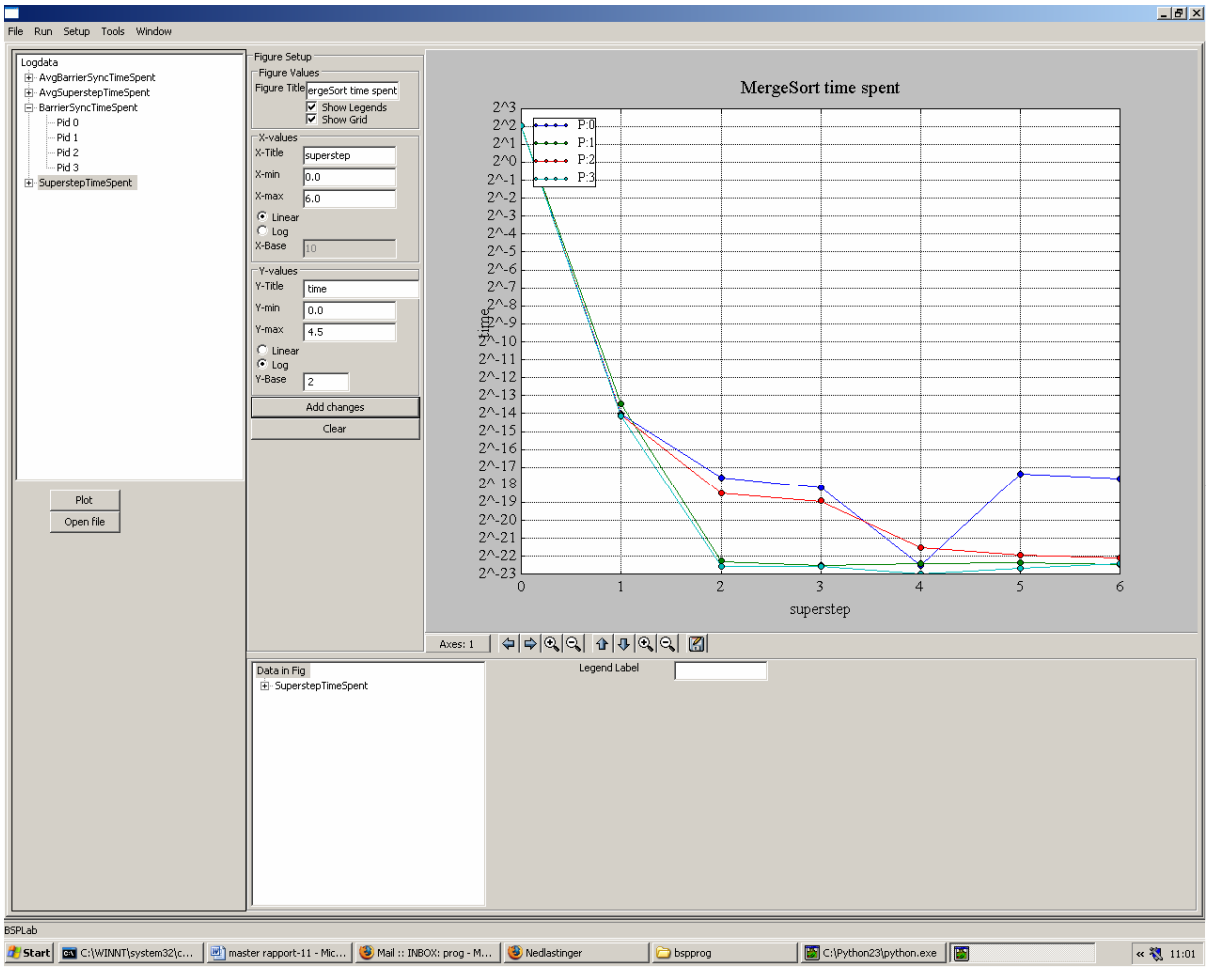


Figur 3-11 Skjermbilde for å velge loggverdier

For nærmere informasjon om loggverdiene se kapittel 5.

3.4.4 Plotteverktøy

I denne delen av systemet kan man plotte resultatene fra simuleringene i grafer. Her har jeg integrert et matplotlib *figurecanvas* i skjermbildet. Dette er et GUI element som gjør det mulig å integrere graf-figurer i wx.Python. (se figuren under)



Figur 3-12 Skjermbildet fra plotteverktøyet

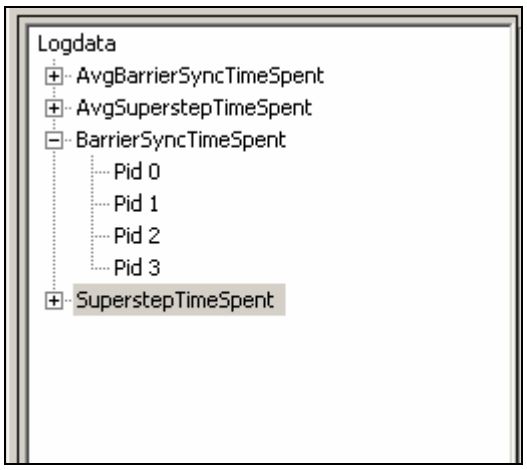
3.4.4.1 Lesing og parsing av resultatfiler

Det første som måtte gjøres, for å kunne plotte simuleringsresultatene i grafer, var å lage mekanismer for å lese og parse resultatfilene som BSPlab produserer. BSPlab produserer en slik resultatfil for hvert program som blir kjørt. Her måtte jeg gå inn å gjøre en liten endring i BSPlab. Hver enkelt resultatfil inneholder en header med generell informasjon om simuleringen. Denne kan variere i innhold og størrelse. Jeg måtte derfor gå inn i BSPlab å gjøre en endring i koden slik at BSPlab lager en merkelapp i den siste linjen i denne headeren. Dette for å vite når headeren sluttet og når resultatutskriftene begynner.

Resultatene i resultatfilen kan være av to forskjellige typer med to forskjellige formater (se kapittel 2.3.4 BSPlab resultatfil). Jeg måtte derfor legge inn kode som kunne tolke hvilken type resultatutskriften var av.
Resultatfilen leses linje for linje. Verdiene blir plassert i en datastruktur sortert på tittelen til loggverdien (eks. SuperstepTimeSpent).

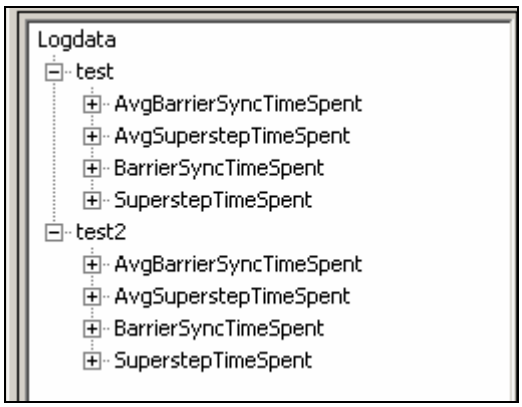
3.4.4.2 **Plotting av loggverdier**

Etter at alle resultatfilene er lest, og resultatene lagt til i datastrukturen, leses denne og innholdet blir lagt til i en trestruktur. (Se Figur 3-13 Trestruktur med loggdata)



Figur 3-13 Trestruktur med loggdata

I de tilfellene der et prosjekt inneholder mer en et program, eller flere kjøringer av det samme programmet, vil verdiene for alle kjøringene bli lagt til i treet. Se figuren under.

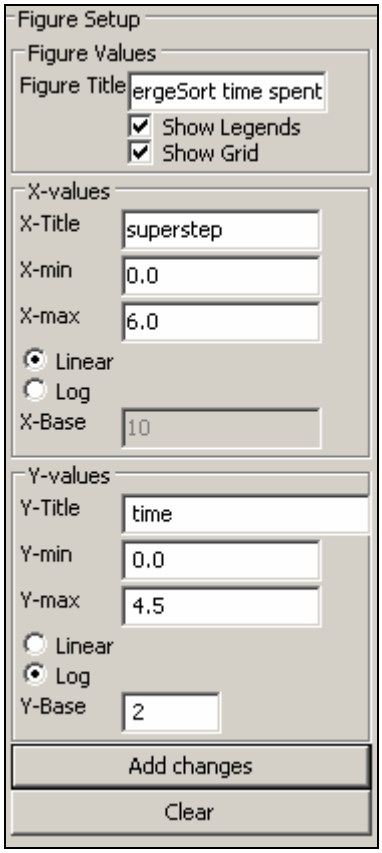


Figur 3-14 Loggtre som inneholder loggverdier for programmene *test* og *test2*

Ved å velge ønsket loggverdi i denne trestrukturen og trykke på plottknappen, blir denne plottet i grafen. I treet har man to nivåer for hver enkelt loggverdi. (Se Figur 3-13 Trestruktur med loggdata) Ved å velge det øverste nivået, vil valgte loggdata for alle prosessorene bli lagt til i grafen. Hvis man velger nivået under vil kun dataene for den prosessoren som er valgt bli lagt til i grafen. Man kan plotte data i mange omganger man vil, men kun en verdi av gangen. Med denne løsningen er det mulig for brukeren å enkelt plotte verdiene for alle prosessorene, men også mulig å kun plotte verdier for utvalgte prosessorer.

3.4.4.3 **Oppsett og redigering av graf- figurer**

Når brukeren har valgt hvilke data som skal være med i figuren, kan han redigere figuren. På venstresiden av figuren finnes det en oppsettsmeny der brukeren kan legge inn en rekke forskjellige verdier for figuren.



Figur 3-15 Figur- oppsettsmeny

Her kan man sette tittel på figuren og benevninger på aksene. Man kan velge om figuren skal ha et rutenett og om benevningen (legends) på de dataene som er plottet i figuren skal vises. I tillegg kan man sette max/min verdier for aksene. Man kan velge om aksene skal ha en lineær eller logaritmisk skala. Og hvis man setter logaritmisk skala setter man et basetall for denne. For å utføre valgte endringer på figuren, trykker man på knappen *Add changes.* For å fjerne alle data som er plottet i figuren trykker man på knappen *Clear.*

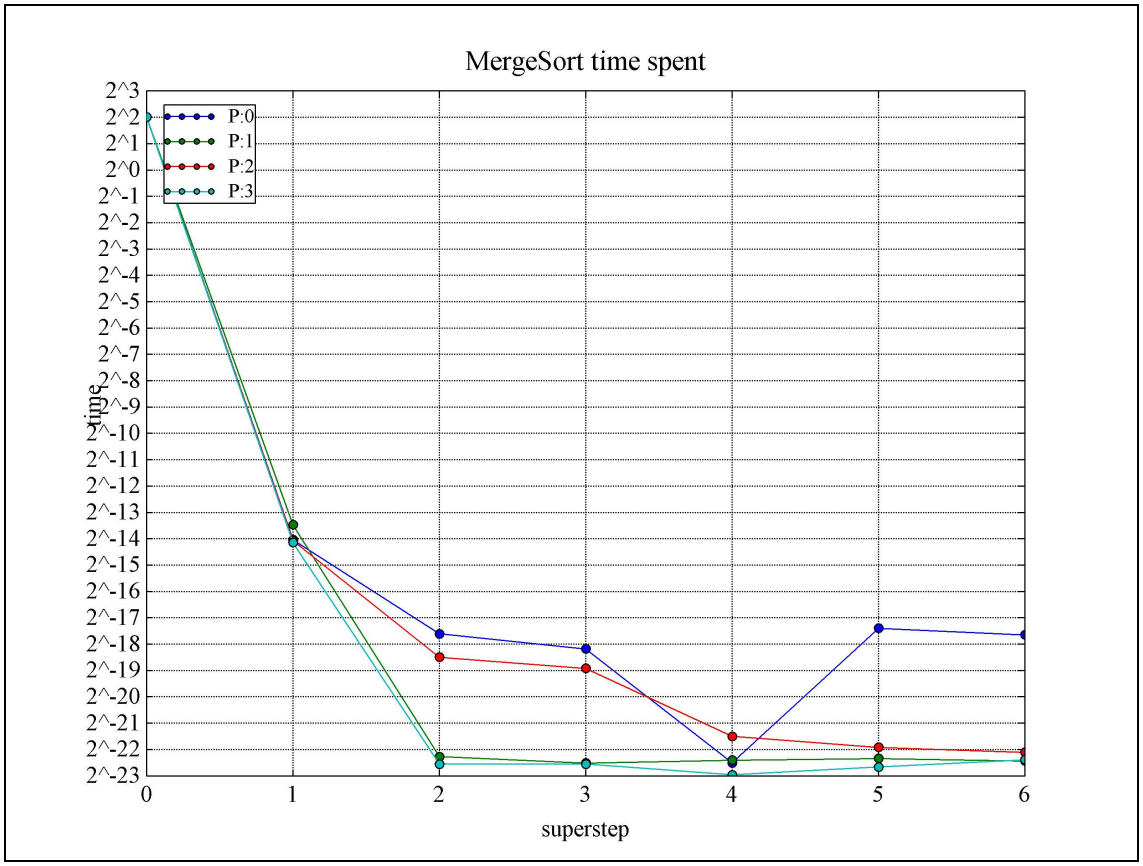
Under selve grafen finnes det en menylinje, som brukeren kan bruke for å flytte seg langs aksene på grafen, zoome inn og ut og i tillegg lagre grafen.

Figur 3-16 Menylinje under figur

I utgangspunktet er dette en standard menylinje som følger med matplotlib, men funksjonaliteten til denne standardlinjen fungerte dårlig for dette bruket. Ved bruk av pilknappene for flytting langs aksene opplevde jeg at hvis verdiene på aksene var veldig lave, flyttet man seg alt for langt langs aksene med standard menylinjen.
Jeg valgte derfor å gå inn å overskrive metodene som sørget for funksjonaliteten til denne menylinjen. Ved flytting langs aksene ved hjelp av pilknappene valgte jeg å skrive en metode som flyttet en prosentandel av aksen for hvert klikk. På denne måten flyttet man seg mer i forhold til verdiene på aksene.
Ved å overskrive metoden for lagring av figur, ble det mulig å lagre figuren i to forskjellige formater ved trykk på lagreknappen (se kapittel 3.4.4.4).

3.4.4.4 **Lagring av bildefil og figurfil**

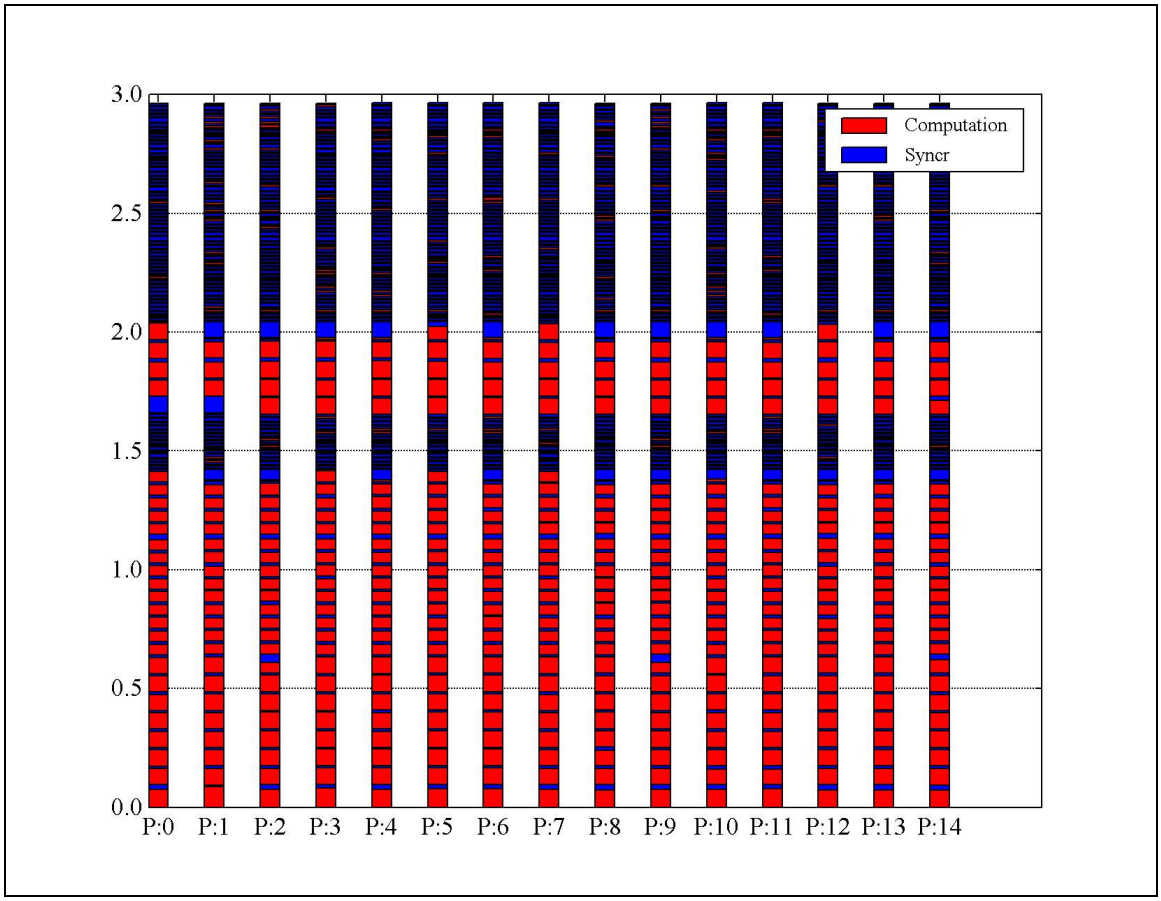
Når brukeren velger å lagre figuren blir det lagret to filer på lokasjonen brukeren velger. Den første filen er selve bildefilen, som blir lagret i *jpeg* format. Den andre filen er en *fig* fil som inneholder all informasjonen om figuren. Grunnen til at jeg valgte å lage denne ekstra filen, var at ved hjelp av denne blir det mulig for brukeren å hente frem igjen lagrede figurer i plotteverktøyet og gjøre endringer i etterkant. Dette hadde ikke vært mulig hvis man bare hadde lagret bildefilen. Denne ekstra filen blir lagret på den samme lokasjonen som bildefilen, og får det samme navnet, men med filendelsen *fig*.



Figur 3-17 Eksempelgraf laget i plotteverktøyet

Figur 3-17 viser en enkel graf laget i plotteverktøyet. Programmet som er kjørt er et sorteringsprogram basert på mergesort algoritmen. Her er tidsbruken til hver enkelt prosessor, på hvert enkelt superstep, plottet i grafen. BSPlab-maskinen programmet har kjørt på har brukt 4 prosessorer og kjøringen har gått over 7 superstep.

Jeg har også implementert en standardfigur i plottesystemet. Grunnen til dette er at jeg må inn å behandle dataene før de kan plottes i en figur. Dette er altså en type figur man ikke kan lage i det ordinære plotteverktøyet. Denne figuren viser balansen mellom tid bruk på databehandling og tid brukt på barrieresynkronisering fordelt på de forskjellige prosessorene i BSP-maskinen. Dette kan være interessant å fremskaffe denne informasjonen for å observere eventuelle skjevheter mellom de forskjellige prosessorene. Er det for eksempel noen prosessorer som må bruke mesteparten av sin tid på å vente på at de andre prosessorene skal blir ferdige?

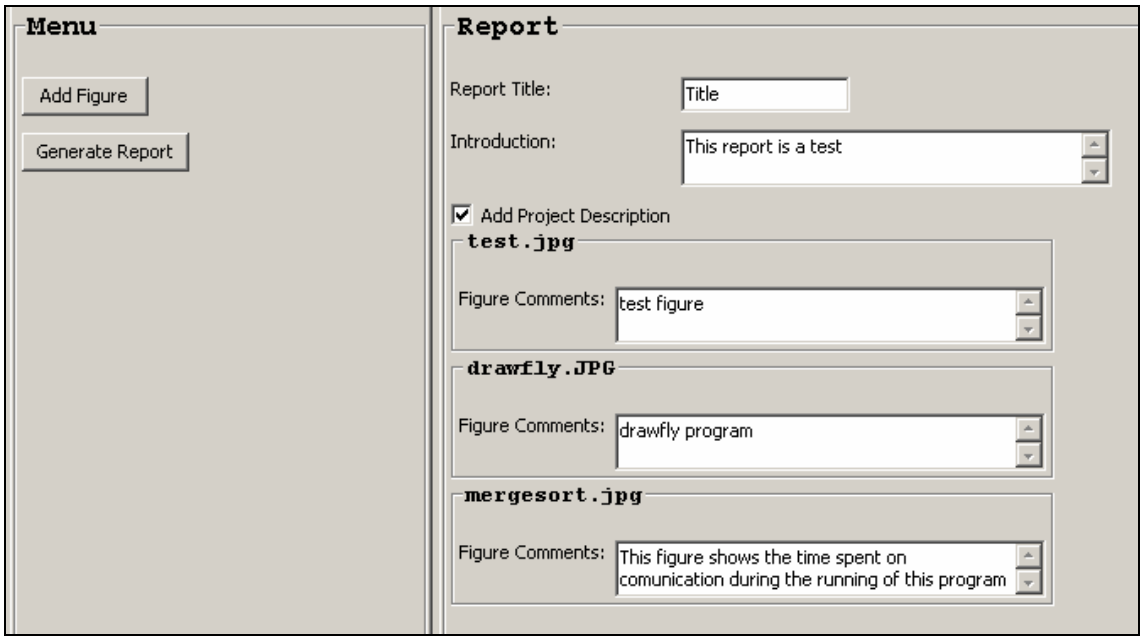


Figur 3-18 Figur som viser balanse mellom tid brukt på databehandling og tid brukt på synkronisering

På grunn av tidsmangel mot slutten av utviklingstiden er dessverre ikke denne standardfiguren, integrert i plotteverktøyet. Dvs. at pr. dags dato har ikke systemet denne funksjonaliteten.

3.4.5 Rapportverktøy

Denne delen av systemet lar brukeren generere PDF-rapporter som inneholder utvalgte graf-figurer laget i plottverktøyet.



Figur 3-19 Skjermbilde fra rapportverktøyet

Brukeren kan legge til en overskrift, en innledning og prosjektbeskrivelse. Så legges figurer, utviklet ved utviklet ved hjelp av plotteverkøyet i systemet, til rapporten.

Verktøyet som er brukt for å utvikle PDF delen av dette systemet heter reportlab.

Rapportverktøyet er dessverre ikke integrert i systemet, på grunn av tidsmangel mot slutten av prosjektperioden.

3.5 Klargjøring for distribusjon

Når jeg skulle klargjøre systemet for distribuering, gjorde jeg det i to steg. Det første jeg gjorde var å generere all Python koden om til et kjørbart Windowsprogram ved bruk av verktøyet Py2Exe. Steg to var å lage en installasjonsveiviser ved hjelp av Inno Setup. Etter at disse to stegene er gjennomført har brukeren kun en enkelt installasjonsfil å forholde seg til. Når installasjonen er gjennomført, er systemet er klart til kjøring.

3.5.1 Py2Exe

Før jeg kunne generere om Python kildefilene til en exe-fil med tilhørende biblioteker, måtte jeg skrive en setup-fil.

```
from distutils.core import setup
import py2exe
import glob

setup(
    data_files = [(r'matplotlibdata',
glob.glob(r'c:\python23\share\matplotlib\*')),
                  (r'matplotlibdata',
[r'c:\python23\share\matplotlib\matplotlibrc'])],

    version = "0.0.1",
    description = "Experiment Environment For BSPlab",
    name = "BSPlab experiment environment",
    options = {
    'py2exe': {
                'dll_excludes': ['libgdk-win32-2.0-0.dll'
                                 'libgobject-2.0-0.dll,
                                 'libgdk_pixbuf-2.0-0.dll'],
                "packages": ["pytz",'matplotlib'],
              }
       },
    windows = [
        {
            "script": "BSPlabGraphicalEnv.py",
            "icon_resources": [(1, "i.ico")],
            "data_files":[("l.gif",'b.gif','t.gif')]
        }
    ],

    )
```

Figur 3-20 Setupscript til py2exe

Py2exe er et åpen kildekode prosjekt, laget på frivillig basis. Dessverre er dokumentasjonen heller dårlig. Dokumentasjonen er til dels mangelfull, og mye av det som er dokumentert er utdatert.

I utgangspunktet skal Py2Exe selv finne ut hvilke moduler som skal være med i distribusjonen som blir generert. Men det oppsto problemer med å inkludere matplotlib. Jeg måtte derfor inkludere matplotlib i script filen. I tillegg fikk jeg en del feilmeldinger om feil i DLL-filer under genereringen. Løsningen på dette problemet var å ekskludere disse filene. Dette hadde ingen påvirkning på resultatet.

3.5.2 Inno setup

I dette verktøyet lager man en installasjonsveiviser til Windows. Dette gjøres ved hjelp av et relativt enkelt setupscript. I dette scriptet forteller man hvilke filer installasjonspakken skal inneholde og hvor disse befinner seg. I mitt tilfelle var det distribusjonen laget med py2exe, BSPlab, og DevC++ som skulle inkluderes i installasjonspakken. Etter å ha kjørt scriptet i Inno setup compiler sitter jeg igjen med en installasjonsfil som inneholder alle elementene. Ved kjøring av denne starter installasjonsveiviseren som fungerer på alle versjoner av Windows.

4 Testing

4.1 Brukbarhetstesting

Ved en prototype utviklingsprosess som jeg har utviklet dette systemet i, skjer mye av testingen underveis i prosessen. Etter hvert som en protype er utviklet testes denne, og feil rettes.

"I programvareutvikling har det utviklet seg en tendens at prototyping og testing av programvaren foregår parallelt. Når en prototype av en applikasjon utvikles har man samtidig en god mulighet til å teste ut brukergrensesnitt, for siden å re-designe på bakgrunn av testresultatene. Denne metoden kan gi produktet bedre kvalitet og mulighet for å bli en suksess" [Mayhew, 1992]

I og med at utviklingsprosessen min har vært 4-delt, der hver enkelt del av systemet har blitt utviklet hver for seg for så å bli satt sammen til et system, har det vært nødvendig for meg å benytte med av en god del testdata underveis i prosessen. For eksempel ville det ikke være mulig å teste kjøromgivelsen før denne var koblet til setupdelen av systemet hvis, jeg ikke hadde produsert testdata og ekstra testkode for uttestingen. Selv om det har krevd en del ekstra koding og tid å bruke en slik fremgangsmåte, har jeg på denne måten kunne fått testet de forskjellige delene av systemet etter hvert som det ble utviklet.

Etter at systemet ble satt sammen til en del, har jeg gjennomfør testing på systemet som helhet, og forsikret meg om at systemet og dets funksjonalitet fungerer som det skal.

4.2 Resultattesting

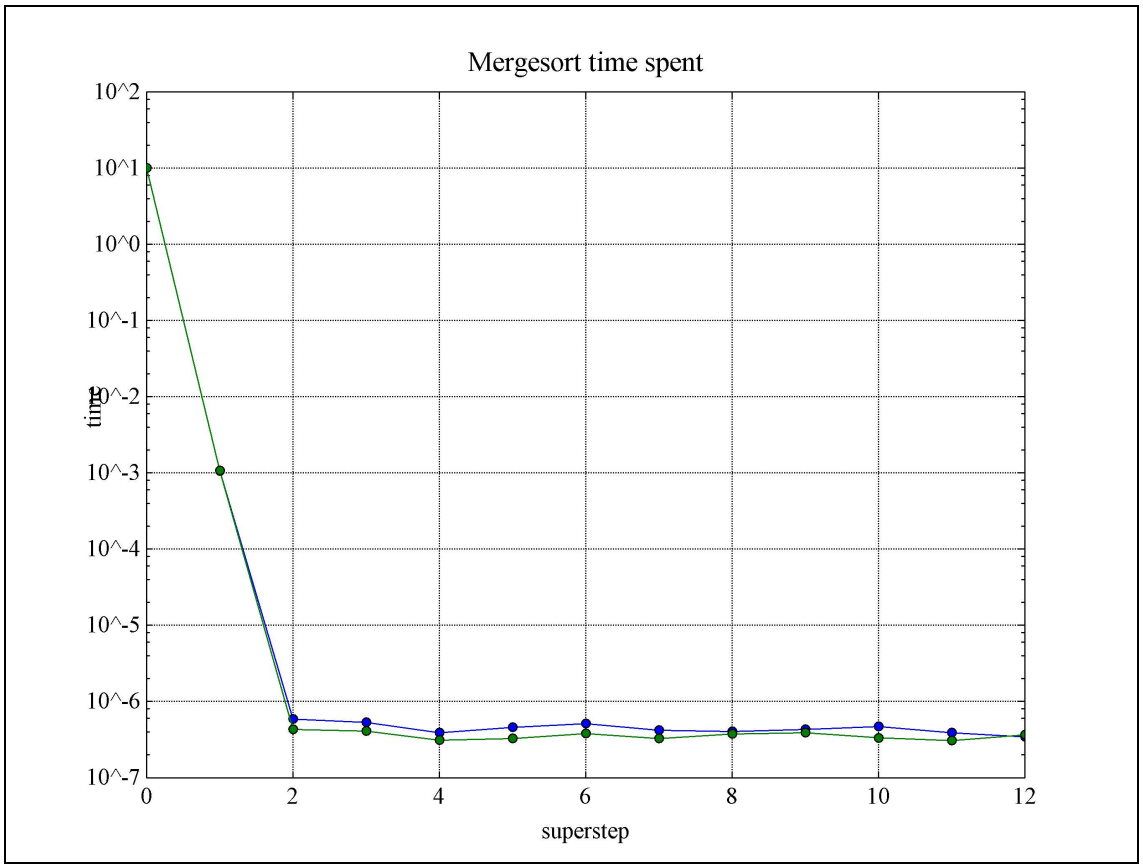
En faktor som var viktig når jeg skulle teste systemet, var hvorvidt systemet påvirket resultatene fra simuleringene. Dette kunne være en problemstilling i de tilfellene hvor BSPlab-programmer ble kjørt med automatisk timing.

I den første testen jeg gjorde brukte jeg et BSPlab-program, som sorterte en mengde data ved hjelp av mergesort algoritmen. Dette er et testprogram som følger med BSPlab, og ble blant annet brukt for å teste tidsforbruket i diplomoppgaven ”BSPlab til folket i 2005 [Østby, Lundereng, 2005]. Kildekoden til dette programmet er vedlagt i Appendix B.

Jeg kjørte testprogrammet på en *Simple* BSPlab-maskin (se kapitel 2.3.1) med 10 prosessorer. Siden jeg skulle gjennomføre tester på hvorvidt systemet hadde noen påvirkning på tidsmålingene i BSPlab, og ikke en generell resultattesting av BSPlab, valgte jeg å kjøre simuleringene på en såpass enkel arkitektur. Ved å lage simuleringen enklest mulig eliminerer jeg vekk flest mulig faktorer som kan påvirke resultatet, og som egentlig ikke er interessante i denne sammenhengen.

Måten testingen ble gjennomført på var at jeg kjørte simuleringen en rekke ganger innenfor systemet, og tilsvarende utenfor, og sammenlignet resultatene. Jeg sjekket også hvorvidt det påvirket resultatet om BSPlab-programmet ble kjørt med *realtime* prioritet. Dette er en funksjonalitet i BSPlab som gir BSPlab sanntidsprioritet, dvs. at operativsystemet gir programmet prioritet over andre prosesser på vertsmaskinen. (For nærmere forklaring se ”BSPlab til folket” [Østby, Lundereng, 2005].)

Mergesort testresultater



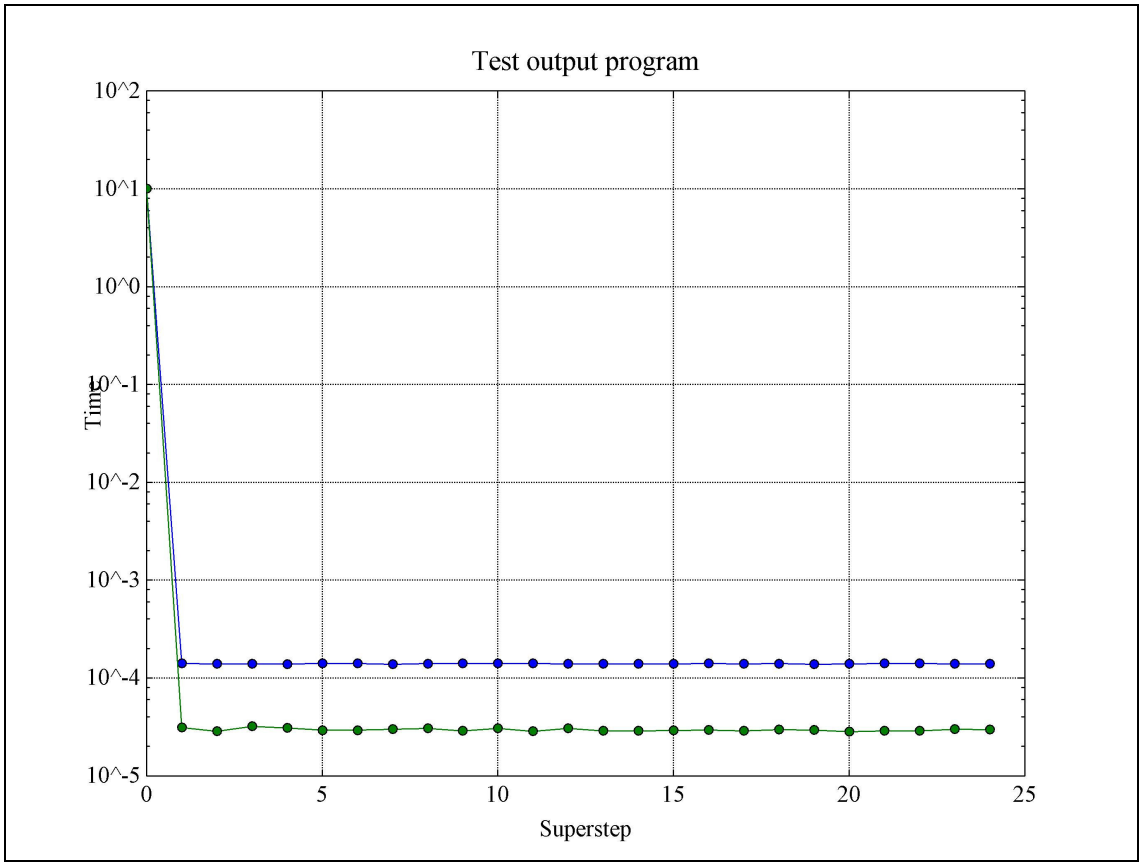
Figur 4-1 Testresultat mergsort program

Figur 4-1 viser resultatet fra testingen. Den blå linjen viser tidsbruken for programmet når det er kjørt utenfor systemet, og den grønne resultat ved kjøring fra systemet. Som vi ser i figuren er variasjonen i resultatene meget små. Det virker som systemet ikke har noen påvirkning på resultatene for dette testprogrammet. Jeg har også sammenlignet to identiske kjøringer utenfor systemet. Og også her oppstår det en liten variasjon mellom kjøringene. Man ser også at programmene har en relativt høy oppstartskostnad, men at denne gjør seg gjeldene både når man kjører programmet gjennom systemet og utenfor, og at systemet ikke påvirker oppstartkostnaden på noe vis. Kjøring med og uten *realtime* prioritet gjorde ingen utslag i resultatene. Jeg er usikker på hvorvidt denne funksjonaliteten fungerer riktig i BSPlab. Men siden dette ligger utenfor min oppgave har jeg ikke tatt meg tid til å gå nærmere inn å sjekke dette.

Siden systemet jeg har laget har en del funksjonalitet for å behandle utskrifter fra BSPlab programmene, ville jeg teste hvorvidt tidsbruken for et program med stor utskriftsmengde, ville bli påvirket av denne utskriftsbehandlingen i systemet.

Jeg skrev derfor et enkelt testprogram, som hadde som eneste oppgave å produsere utskrifter. Kildekoden til dette programmet finnes i Tillegg B.
For hvert superstep produserer programmet 30 utskrifter pr. prosessor. 15 av disse er vanlige printf utskrifter, mens 15 er utskrifter med pid_print funksjonen. (Ved kjøring utenfor systemet vil en slik pid_print utskrift tilsvare en vanlig printf utskrift) Jeg testet så på den samme måten som i testen over ved å kjøre programmet både fra systemet og utenfor systemet en rekke ganger, og sammenligne resultatene.

Testresultatet utskriftsprogram



Figur 4-2 Resultat fra utskrift testing (blå linje er programmet kjørt utenfor systemet, grønn kjørt fra systemet)

Resultatene fra disse testene kom ganske overraskende på meg. Tidsbruken var som Figur 4-2 viser faktisk mindre når programmet ble kjørt fra systemet, enn når det ble kjørt utenfor. Den eneste forklaringen jeg har på dette resultatet er at det må være mer ytelseskrevende for et BSPlab-program å skrive ut til command-vinduet i Windows enn å skrive ut gjennom systemet jeg har utviklet.

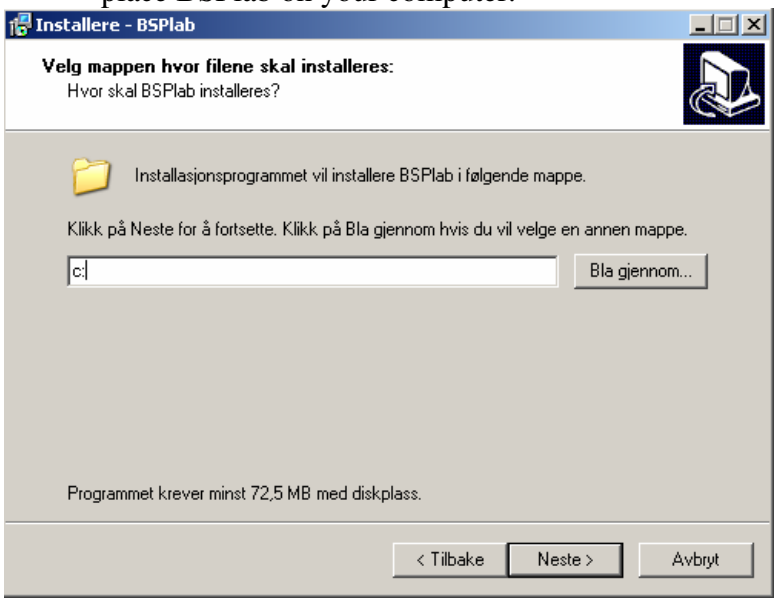
5 BSPlab Graphical Environment Tutorial

Here I will describe how to use the graphical user interface for BSPlab. This is not a tutorial on how to program BSPlab programs, and it is therefore required that the user has some knowledge about how to develop BSPlab programs.

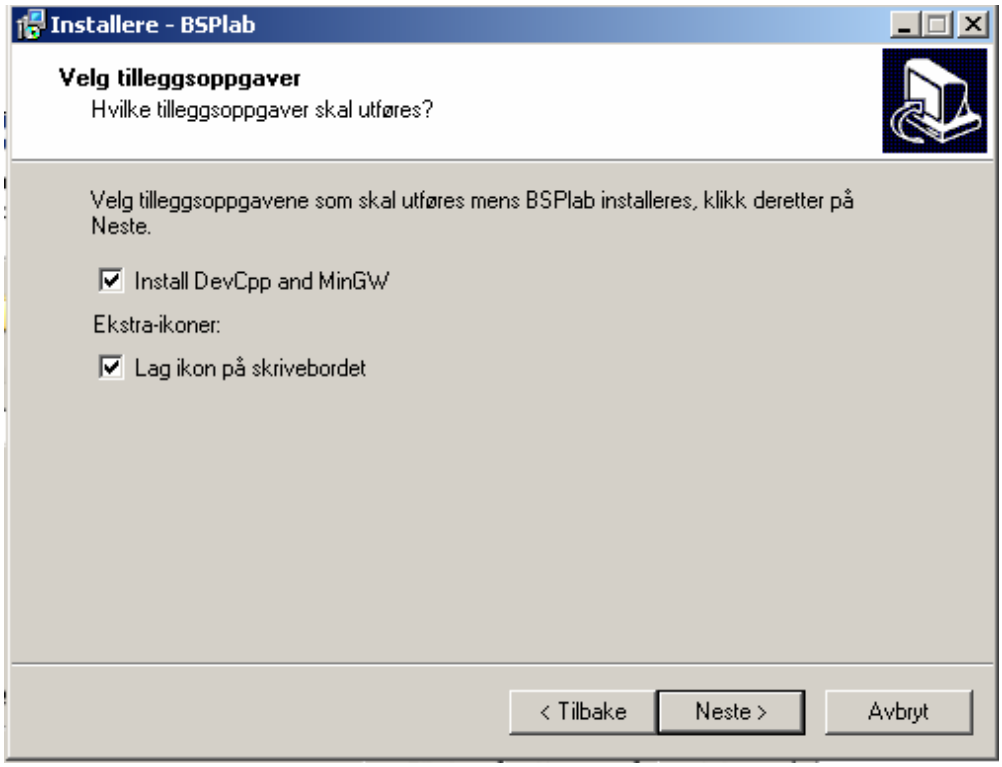
Some of the screenshots in this part has Norwegian text, this is because they are elements of the operating system. If the system operates on a Windows version with English language these part op the system will also be in English.

5.1 How to install BSPlab

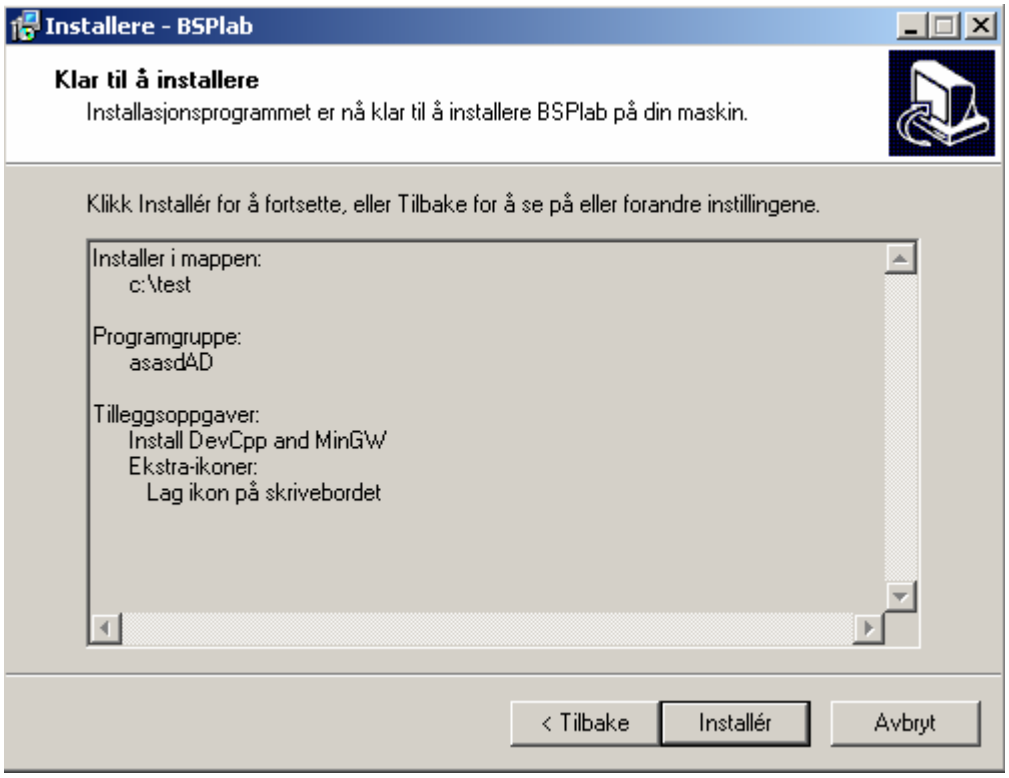
1. Launch the file named BSPlab_install.exe.
2. The installer will start.
3. First you have to choose language for the installing process.
4. The next thing you have to do (after a welcome screen) is to choose where to place BSPlab on your computer.



5. Then you can decide where to place the shortcut on the start menu



6. Decide whether you want to install DevCpp an MinGW (needed to make BSPlab programs)



7. Then press *Install* and BSPlab will be installed on your computer.

If you have chosen to install devCpp and MinGW the installer of these programs will start when finished installing BSPlab on your computer.

5.2 How to create a new BSPlab project

1. Open BSPlab.
2. Open the file menu in BSPlab and press *New*.

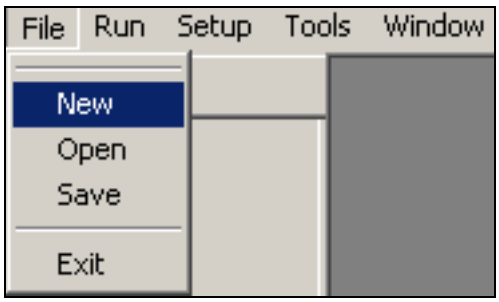


Figure 5-1 Filemenu

3. Then choose where to place the new BSPlab project.

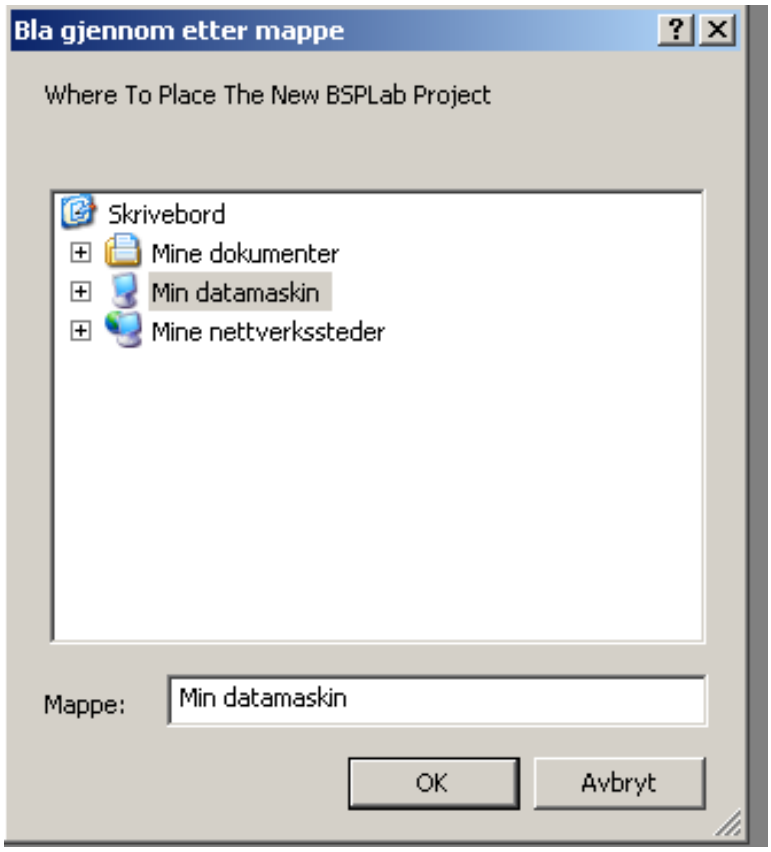


Figure 5-2 Where to place the new project

4. When you have decided where to place the project, you must give the new project a name.

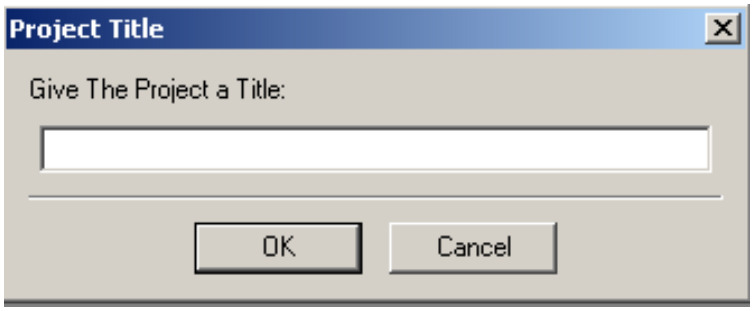


Figure5-3 Project title

5. A new folder with the project name will be placed on the decided location.

5.3 How to open and save the BSPlab project

To open or save a BSPlab project simply press open or save in the file menu.

Be aware that there is not implemented any functionality that checks whether you have saved the project before closing BSPlab, or opening a new project. You must therefore be sure that you have saved the project if you want to keep the changes you have made!

5.4 How to setup a BSPlab project

1. Create a new BSPlab project or open an existing BSPlab project. Described in sections 5.2 and 5.3.
2. Press *Project Setup* in the setup menu in BSPlab. The setup screen will open.

Figure 5-4 Project setup in the menu

The BSPlab project setup is divided into three parts (see Figure 5-5), the general setup, BSPlab programs and log setup. Use the navigation menu on the left side of the screen to navigate between these. How to setup a project in these three parts is described in the next sections.

5.4.1 General Setup

In this section you could give the project a description. This description may be added to a project report in the report-tool. See section **Error! Reference source not found.** for more information.

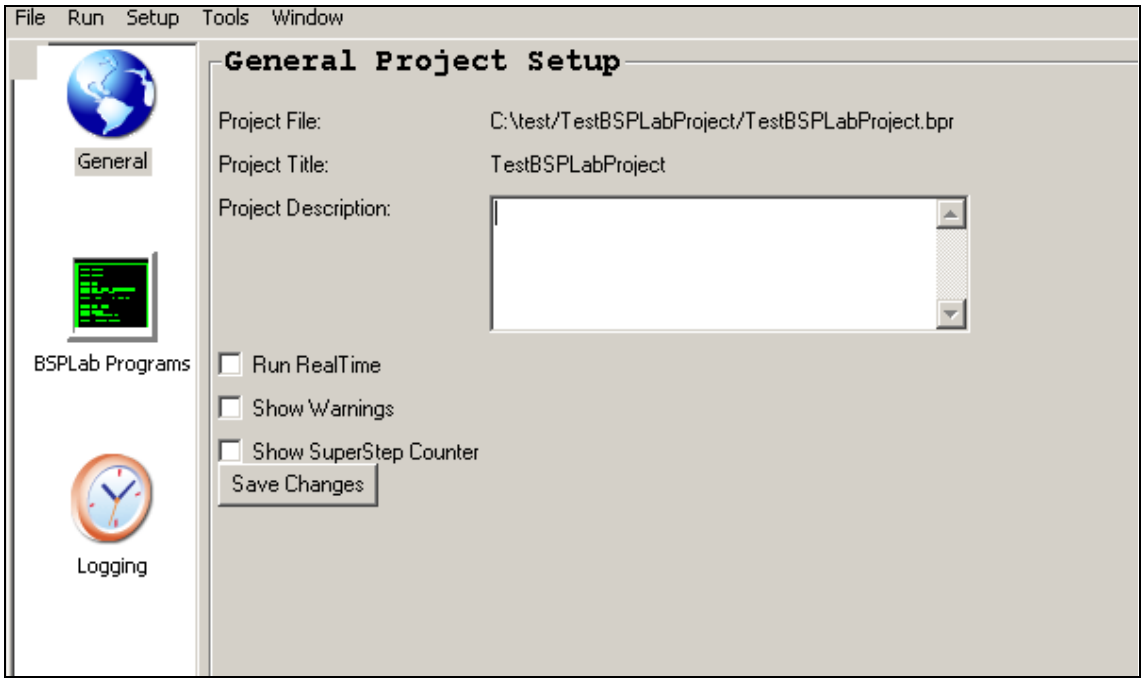


Figure 5-5 BSPlab gereral setup

As you see in Figure 5-5 there are three checkboxes in the general setup.

Run Realtime:

If this checkbox is checked the BSPlab programs in your project will get real-time priority when executed.

Be aware that this may cause your computer to freeze while the BSPlab programs are executed! This is normal and you will gain control over the computer when the programs are finished executing.

Show warnings:
Here you decide whether the BSPlab program should produce warning messages or not.
Show superstep counter:
If this checkbox is checked the superstepcounter in the run enviroment will count while executing the BSPlab programs. See section 5.5 for more information about the superstepcounter.
When you have made your choices in this section you must press the *Save changes button*.

5.4.2 BSPlab Programs

In this setup section you add BSPlab programs to your project and set the parameters for these programs.

5.4.2.1 How to add a new program to your project

There are two different ways to add a BSPlab program to your project:

1. You may add a compiled BSPlab program by pressing the button *Add.* This adds a copy of the program to the project folder. This means that any changes you make to the original program will not apply in the BSPlab project.

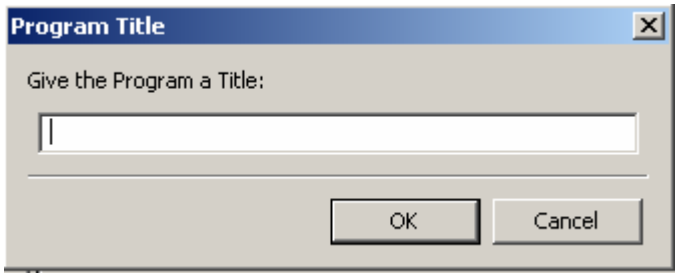
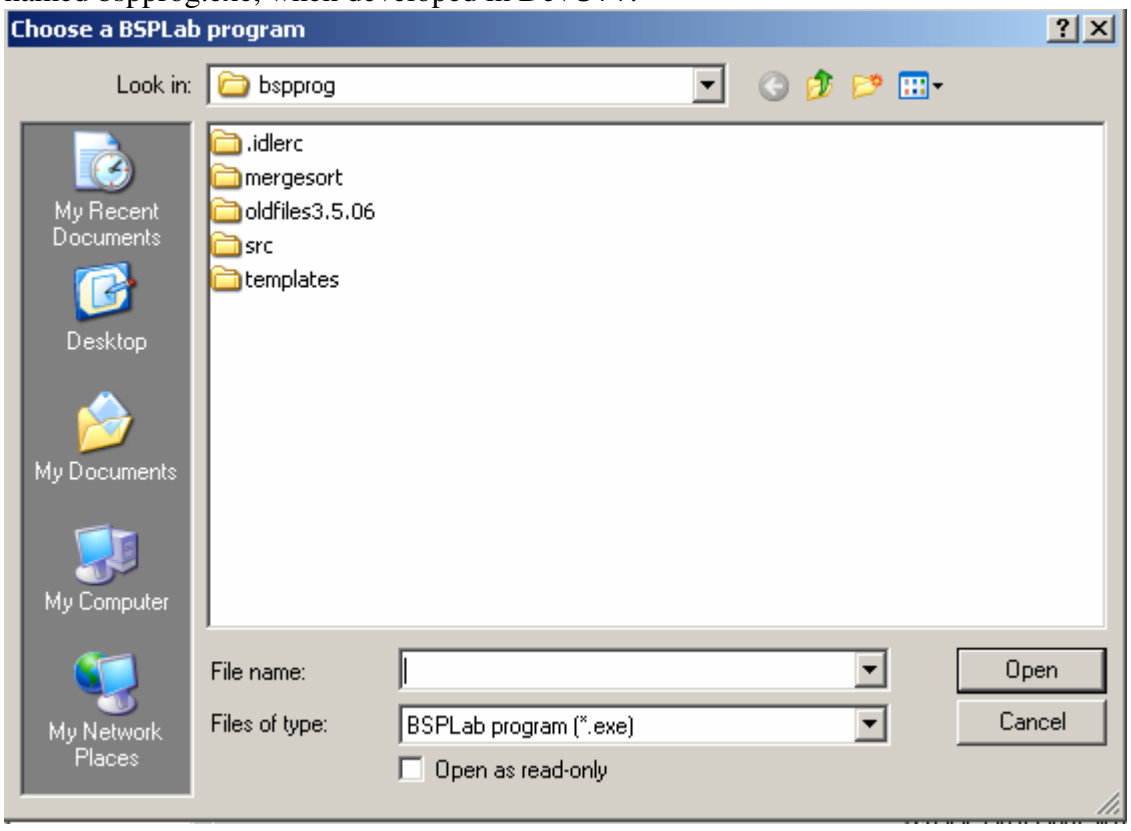


Figure 5-6 Program title

The first thing you must do when adding a BSPlab-program to your projeet is to give the project a title. This is necessary because most BSPlab programs are named bspprog.exe, when developed in DevC++.



After giving the new program a title, you must select where the BSPlab program is placed.

2. If you press the *New* button in the BSPlab program setup, you will add a new DevC++ project to your BSPlab project. This project will be placed in the project folder, and DevC++ will be opened. You can now write a new BSPlab program in this DevC++ project, or add existing code to this project. By doing this you can make changes to the BSPlab program and compile the DevC++ project

   First you must give the program a title. (See Figure 5-6)

   Then DevC++ will open with the empty DevC++ project.

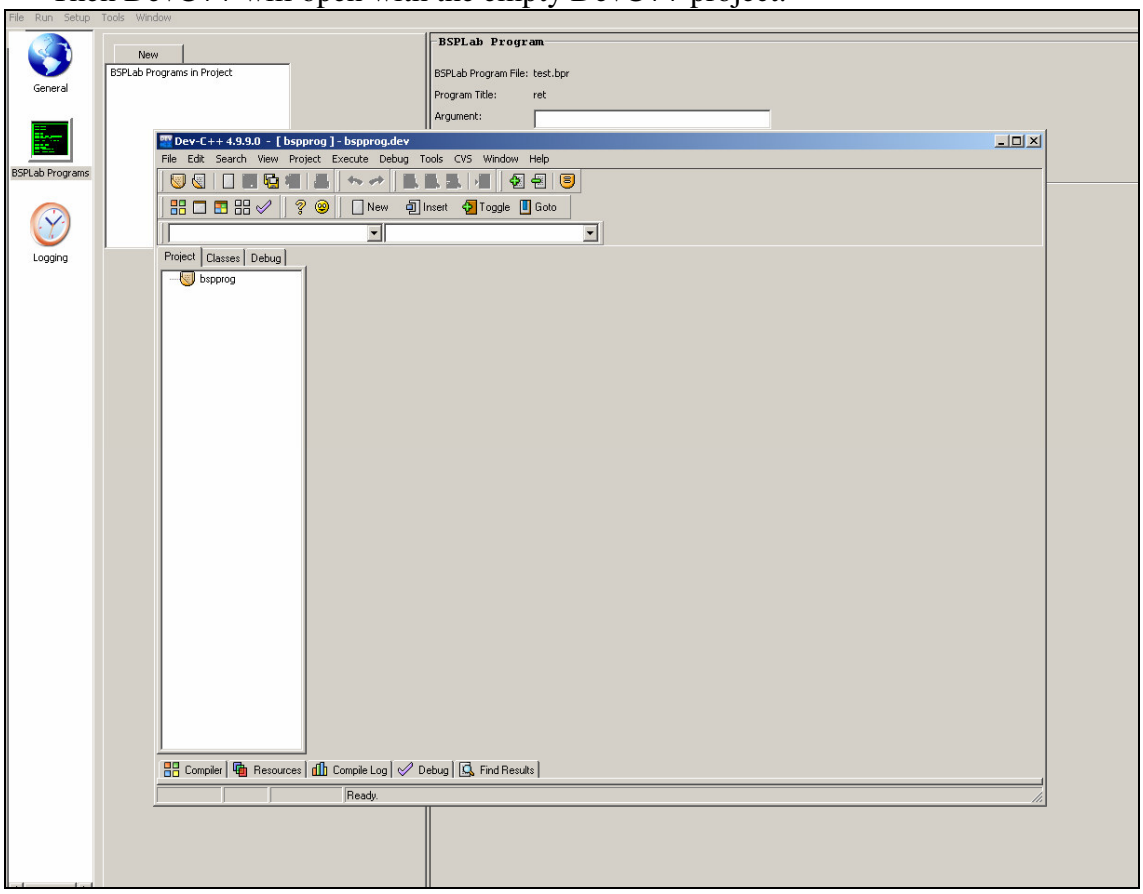


Figure 5-7 The system after opening DevC++

You must then program or add an existing program to this DevC++ project, and successfully compile it.

When you have added the program to the project, the title you gave the program will show up in the program list.

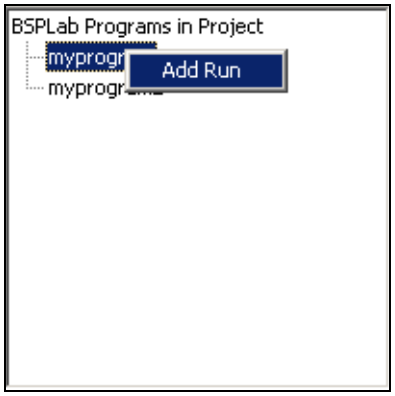


Figure 5-8 Programlist

If you want to run the same program more than once, you may choose the program in the program list and press the right key on the mouse. Then choose *Add Run*.

Figure 5-9 Programlist after adding an extra run to *myprogram*

Then you have to set the parameter-values for the programs and extra runs. To do this you simply choose the program, or run if you have added any extra runs, in the programlist. Now you can start setting the values in the panel right of the programlist. The form is shown on the next page.

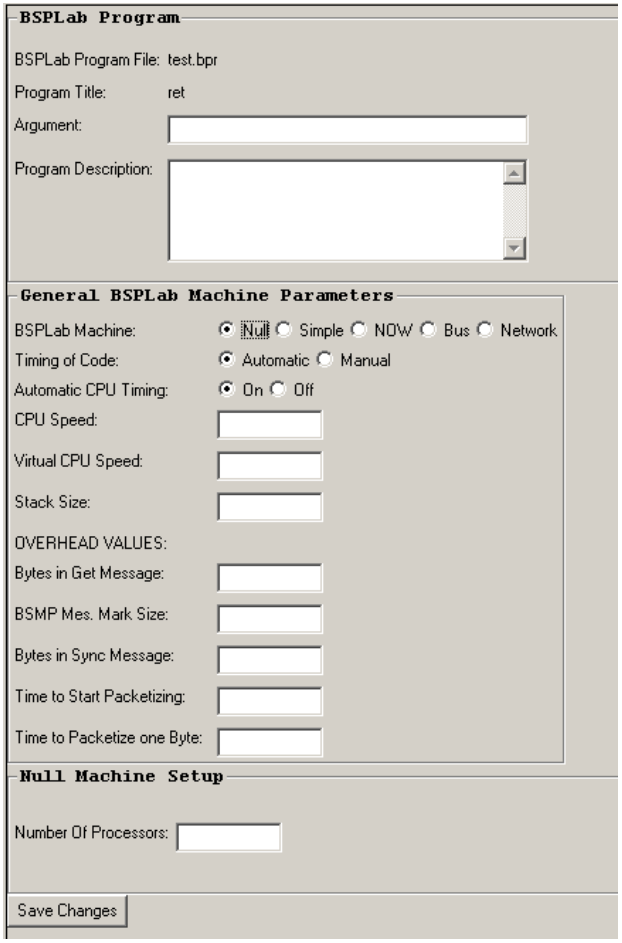


Figure 5-10 BSP program setup

The figure above shows the setup-form for the BSPlab-programs. To setup a BSPlab-program simply fill in the values in this form, and press *Save Changes.* This must be done for every program, or extra run of a program, in the project.

The argument parameter is an argument that is passed with the BSPlab program at execution. (myprogram.exe **argument**)

In program description you may describe the program, or add any notes you want to save about the program.

Then you must set the BSPlab parameter for this program. The different parameters in this part of the system are described on the next pages.

General BSPlab parameters

Parameter	
BSPLab machine	There are 5 different types of BSPLab machines to choose among. These are: 1. Null 2. Simple 3. NOW(Network of workstations) 4. Bus 5. Network
Timing of code	Automatic: The time it takes to perform the code is measured automatically Manual: Here the user decides what parts of the code that takes time. By inserting ***bspsim.hold(t)*** in the code the user says that this part of the code uses ***t*** seconds.
Automatic CPU timing	If checked: The simulator will benchmark the processor on the computer running the BSPLab program. If unchecked: The simulator will use the value from *CPU speed*.
CPU Speed	Used if *Automatic CPU timing is unchecked.* The value in this textbox tell the simulator how fast the computer running the simulation is compared to a 486, 100MHz computer. The value 1 in this textbox means that the running computer equals a 486. The number 10 says that the computer is 10 times as fast.
Virtual CPU Speed	The value in this textbox specifies how fast the BSPLab computer should be compared to a 486, 100MHz
Stack Size	This value sets the stack size for each processor. If set to 0 the default size of the operating system is used.

Bytes in Get Message	The size in bytes in messages that one processor send to request data from a remote node.
BSMP Message Mark Size	The count of bytes used to distinguish a BSMP message from a DRMA message
Bytes in Sync Message	The number of bytes in message used to perform a barrier synchronizing
Time to Start Packetizing	Time to transfer send data to communication controller
Time to Packetize one byte	The time the communication controller uses to packetize one byte

Parameter for Null, Simple, BUS, NOW

Number of Processors	Number of processors in the BSPLab machine

Parameters for the Simple machine

Time To Synchronize One Processor	The time to synchronize one processor
Time To Send One Byte	The time to send one byte

Parameters used in NOW and BUS machines

Barrier Tree Fan-In	The fan-in used in i barrier synchronizing reduction tree
Barrier Tree Fan-Out	The fan-out used in i barrier broadcast tree

Parameters for the NOW machine

Activate Ethernet Noise	Activates Ethernet noise(other traffic) on the Ethernet cable
Time Between Noise	Says how often the noise should appear on the Ethernet cable
Packet Overhead	Size of the overhead of the Ethernet packages
Min Data Size	The minimal size of packages sent over the Ethernet
Max Data Size	The maximal size of packages sent over the Ethernet
Max Retries	The maximal retries to resend a failed message over the Ethernet

Max Backoff	Used in the Ethernet backoff algorithm.

This table of parameter is ment as an introduction to the BSPlab-parameters. Please see the original BSPlab documentation [Dybdahl, Uthus,1997] for a full overview.

5.4.3 Logging

In this section of the BSPlab setup, you decide which values that should be log during the execution of a simulation
The data you select to log will be available in the graph plot-tool after running the BSPlab-programs in your project. You may also get the proberesult.txt file, containing all logged data from the program, in the project folder.

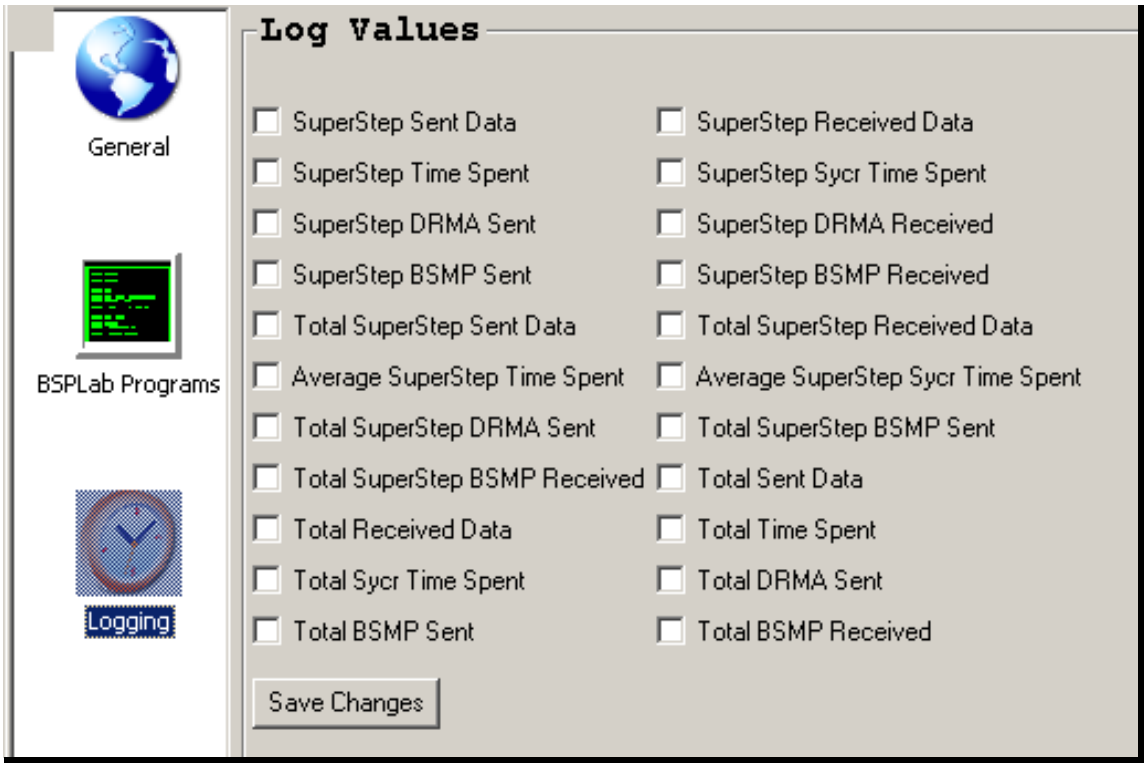


Figure 5-11 Log values

When you have decided which values to put in the log file, you must press the *save changes* button to apply these to your project.

In the next part of this tutorial I will explain the meaning of the different log values.

Log value	What is logged?
SuperStep Sent data	The total(both DRMA and BSMP) amount of data (bytes) sent by each processor at each superstep.
SuperStep Received data	The total amount of data(bytes) received by each processor at each superstep.
SuperStep Time Spent	How much computation time each processor use pr superstep
SuperStep Syncr Time Spent	How much time each processor use pr superstep on barrier synchronisation
SuperStep DRMA Sent	Amount of data each processor send using DRMA pr. SuperStep
SuperStep DRMA Received	Amout of data each processor receive using DRMA pr. SuperStep
SuperStep BSMP sent	Amount of data each processor send using BSMP pr. SuperStep
SuperStep BSMP Received	Amount of data each processor receive using BSMP pr. SuperStep
Total SuperStep Data Sent	The total amount of data sent by all processors pr. SuperStep
Total SuperStep Received Data	The total amount of data received by all processors pr. SuperStep
Average SuperStep Time Spent	Average computation time of all processors pr superstep
Average SuperStep Syncr Time Spent	Average barrier synchronisation time of all processors pr. superstep
Total SuperStep DRMA Sent	The total amount of data, for all processors, sent using DRMA pr. SuperStep
Total SuperStep BSMP Sent	The total amount of data, for all processors, sent using DRMA pr. SuperStep
Total SuperStep BSMP Received	The total amount of data, for all processors, received using DRMA pr. SuperStep
Total Sent Data	The total amount of data sent during the computation of the BSPlab program

Total Recived Data	The total amount of data received during the computation of the BSPlab program
Total Time Spent	The total computation time used during the computation of the program
Total Syncr Time Spent	The total time used on barrier synchronisation used during the computation of the program
Total DRMA Sent	The total number of bytes sent using DRMA during the computation of the program
Total BSMP Sent	The total number of bytes sent using BSMP during the computation of the program
Total BSMP Received	The total number of bytes received using BSMP during the computation of the program

Figure 5-12 Logvalues

5.5 How to run a BSPlab project

When the project setup is done, it is time to run the project. This is done in the run part of the system. This is the first screen you come to when opening BSPlab. To get here just navigate via the run menu in the menu line. Please make sure that you have completed the setup before starting the running of the programs. All three "*Save changes*" buttons in the setup part has to be pressed before running the prosject.

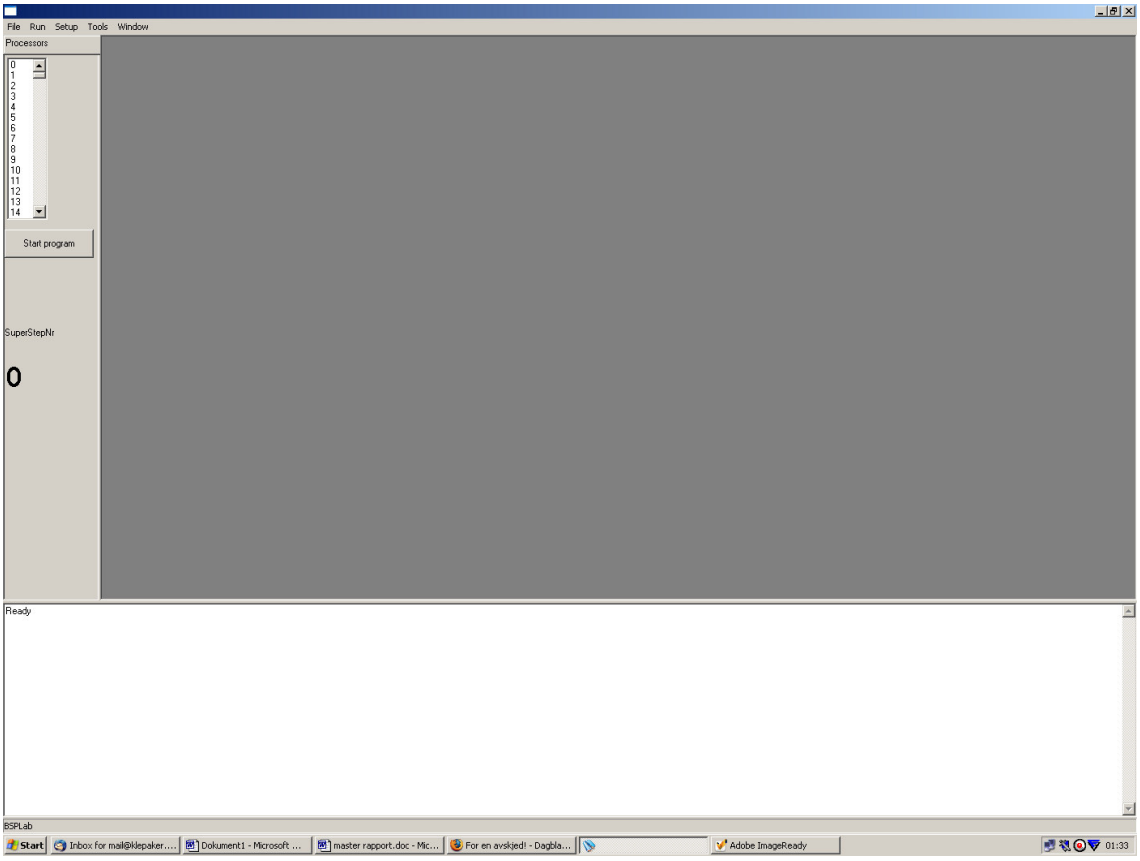


Figure 5-13 The project run screen

If you have programmed your BSPlab program using the pid_print function (see chapter 5.5.1 How to use the pid_print function) you may want to open some of the processor windows before starting the program. This is done by choosing the processor number in the processor list and double-click on the processor number with the mouse. You may open as many processor window as you like. It is also possible to open and close these windows during execution of a program. Be aware that if you have chosen to run the programs in the project with *realtime* priority, you will not be able to open these windows during execution.

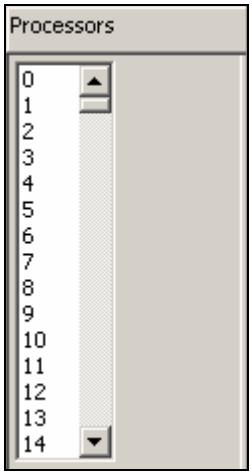


Figure 5-14 Processorlist

To start the running of the project, press the *Start Programs* button. The execution of the first program in the program will start.

Under execution all output that is routed to a processor window will be printed in the right window if this window is open. If not, a number (which you choose in the general setup part) of outputs will be saved. When opening a processor window, these output messages will be shown in this window, and new messages will be printed to the window as well.

All other output from the BSPlab-program will be printed in the general output frame.

Figure 5-15 The superstep counter

If you have chosen to use the superstep-counter, this counter will show the superstep-number of the BSPlab-program during execution. If you have more than one BSPlab-program this superstep-counter will be reset between each run.

When the first program is terminated the next program will start, and so on.

5.5.1 How to use the pid_print function

This function lets you print a message to one of the processor windows depending on which processor that produced the message.

The first thing you have to do is to include the pid_print header file to your BSPlab-program. This is done by adding the following code in the start of this program:

```
#include <pid_print.h>
```

When the header file is included you may use the pid_print function in your code like this:

```
pid_print(String “message”, args)
```

This function has the same functionality as the C++ function *printf.* (See http://www.cplusplus.com/ref/cstdio/printf.html, if you don’t know how to use this function)

Example:

```
pid_print("As you see I'm processor number %d of %d ", bsp_pid(),bsp_nprocs());
```

If this function is called by processor 0, 3 and 6 and there are totally 10 processors, the program runs over 3 supersteps so the function is called 3 times by each processor.

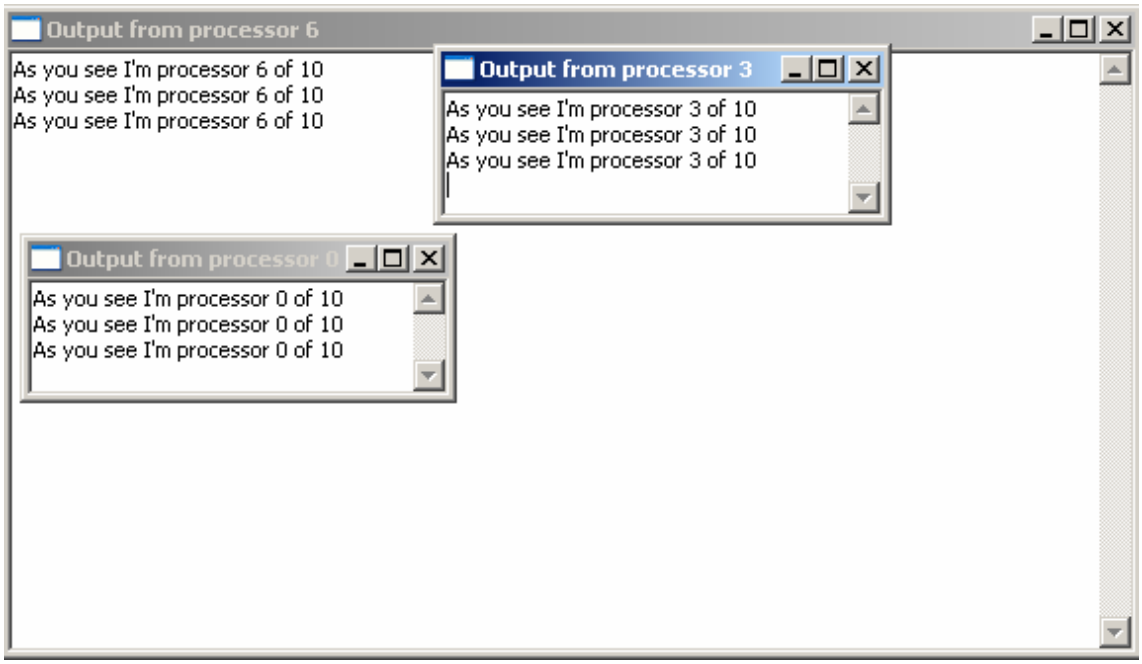


Figure 5-16 Example pid_print function

5.6 How to use the BSPlab graph-plot tool

When all the programs in your project has terminated, it is time to get the results. By using the graph-plot tool you will be able to visually present these results in graphs.

First open the graph plot tool, by choosing plot in the tool menu in the menu line. Now the graph-plot tool will open.

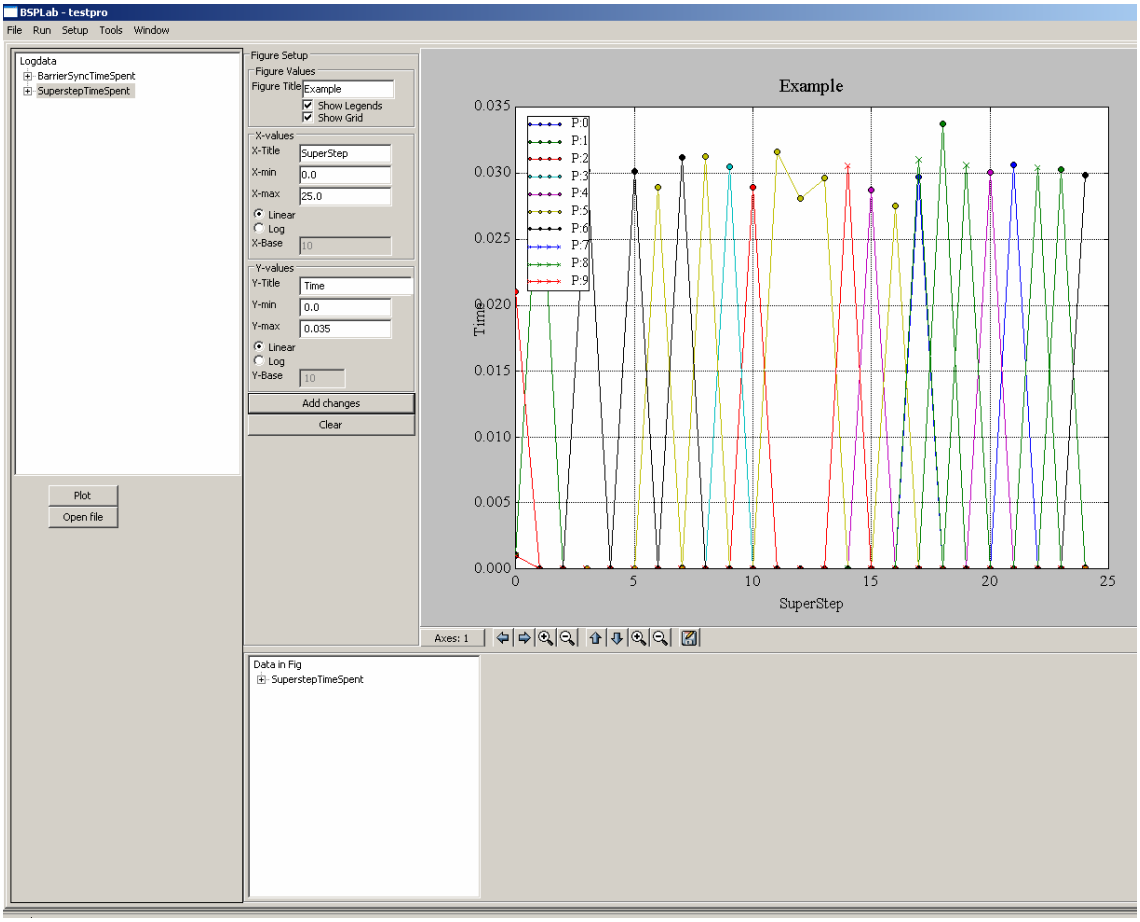


Figure 5-17 The graph-plot tool

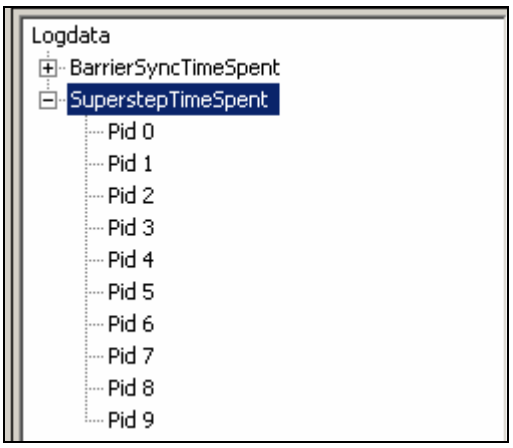


Figure 5-18 List of available plot results

All available results from the simulation are listed as shown in Figure 5-18 List of available plot results. To plot the results listed, simply select the result you want in your figure, and press the *Plot* button.

As you see in the figure there are two levels in the list. By choosing the top level you will add the results for all the processors to the figure. But you may also expand the top level element and choose to add the data of only one processor to your graph. You may of course add results for more than one processor in the same figure, but only one at the time.

If you have more than one BSPlab-program in your project, or more than one run of the same program, the results for all programs are listed in the result list.

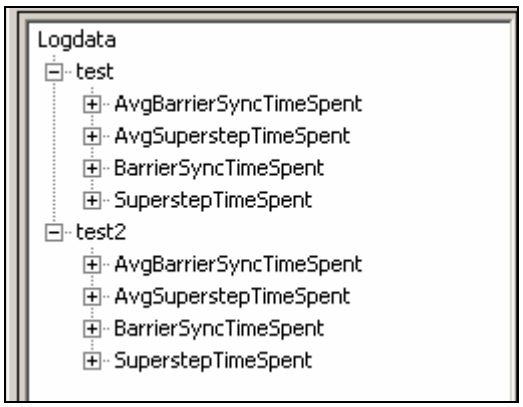


Figure 5-19 List when there is more than one program in the project

Adjusting the figure

When you have plotted all the results that you want in your figure, it is time for adjustments.

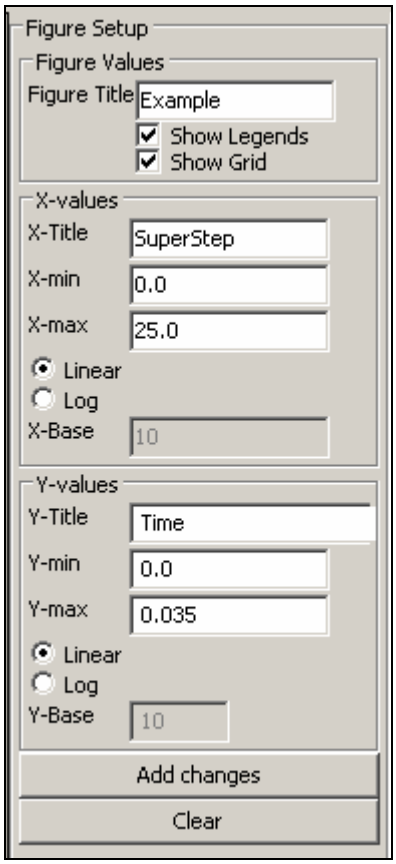


Figure 5-20 figure Setup

Figure 5-20 shows the figure setup menu. Here you can make adjustments to your figure.

Figure Title:
Is the title at the top of the figure
Show legends:
If checked the legends will be added to your figure.

Show grid:
If checked a grid will be added to the figure
X/Y-Title:
Add a label on the axes
X/Y-Min:
The start value on the axes
X/Y-Max:
The largest value to show on the axes
Linear/Log:
Choose whether you want the scale on the axis to be linear or logarithmic
Base:
If logarithmic axis scale is choosen, this number will be the base of the logarithmic scale.

To apply the changes you want to make on the figure, press the *add changes* button.

To clear the figure completely press the *clear* button.

Be aware that this will remove all data from the figure, and leave you with an empty canvas. So be sure that you have saved the figure if you want to keep it!

There is also a menu line below the figure that you can use to make simple adjustments. Here you may move along or zoom in/out on the axes.

Figure 5-21 The menu line below the figure

Saving or opening a figure

To save the figure press the save button in the menu line below the figure.

Then choose were to place the figure. When you save the figure an additional file is saved on the same location. This file has the same name as the picture (jpg) file, but has the ending *fig*. This is done so you may open and make changes to the figure. To do this simply open the graph-plot tool and press the *open* button below the *plot* button. Now you can choose the figure file from the location you saved it.

6 Konklusjon

Etter at arbeidet med denne oppgaven nå er ferdiggjort vil jeg oppsummere hvordan jeg føler arbeidet har gått. Jeg vil fortelle litt om de erfaringene og lærdommene jeg har tilegnet med under prosessen, og foreslå videre arbeid som kan gjøre de grafiske eksperimentomgivelsene enda bedre.

6.1 Vurdering av systemet

Alt i alt er jeg veldig godt fornøyd med det systemet jeg har utviklet. Det fungerer bra, og jeg føler jeg har fått lagt inn god funskjonallitet, og at den funksjonallitetne som er lagt inn i systemet er nyttig. Håpet er at det systemet jeg har utviklet kan være med på å senke terskelen for å ta i bruk BSPlab, ved at ting nå er mer lettforståeling og og mindre krevende å sette seg inn i.
De fleste databrukere er i dag mest vant til å jobbe opp mot grafiske brukergrensesnitt, og vil kanskje føle seg mer ”hjemme” i BSPlab med et grafisk brukergrensesnitt med mange vellkjente elementer.

6.2 Videre arbeid

Underveis i prosjektet har jeg hatt en del ideer og tanker om hvordan systemet jeg har utviklet burde fungere og hvilken funksjonalitet det kunne inneholde. En ideell verden der man ikke trengte å tenke på tidsbrukt og innleveringsfrister hadde det ikke vært noe problem å gjennomføre alle disse ønskene og ideene. Dessverre blir man nødt i prioritere vekk en del ting for å kunne bli ferdig i tide. Her vil jeg beskrive de ideene og tankene, om systemet som dessverre måtte prioriteres vekk.

Mulighet for kjøring av serier av simuleringer:

Dette er tenkt som en funksjonalitet der man kan sette opp serier av simuleringer på et BSPlab-program, og variere ett eller flere parametere for hver gjennomkjøring av programmet. Implementeringen av denne funksjonaliteten var tenkt som en løkke struktur. Brukeren kan velge hvilke parametere han vil forandre, sette en startverdi og

stoppverdi, og hvor mye det valgte parametere skal inkrementeres for hver gjennomkjøring.

Eksempel:
Brukeren velger å sette opp en seriekjøring av et BSPlab-program. Han velger at serien skal basere seg på antall prosessorer i BSPlab-maskinen. Startverdien skal være 2 og stoppverdien 128. For hver gjennomkjøring skal prosessorantallet økes med 4. Programmet vil først bli kjørt med 2 prosessorer så 6 så 10 osv.

Ved valg av flere parameterverdier som skal varieres i en slik seriekjøring, kan dette implementeres som ved hjelp av nøstede løkker.

Videreutvikling av plotteverktøy:

Jeg har også gjort meg en del tanker om hvordan plotteverktøyet i systemet kan utvides og forbedres. I dag er det brukeren som selv velger hvilke data som skal plottes i grafene. Dette er en fleksibel måte som gir brukeren stor valgfrihet og gode muligheter til å sette opp gode grafer.

Men det er noen begrensninger i hva brukeren kan fremstille ved hjelp av de dataene som finnes. Det kunne være ønskelig med flere muligheter å fremstille data på en det som finnes i verktøyet i dag. Dette vil kreve at systemet behandler resultatdataene på en annen måte i dag. En mulig måte å gjøre dette på er å lage et sett med standard figurer, som gjør det mulig å gå utenfor den funksjonaliteten som finnes i systemet i dag.

Integrering av rapportverktøyet:

Rapportverktøyet er så godt som ferdigutviklet, men på grunn av mangel på tid mot slutten av prosjektet måtte prioritere vekk dette, og det er derfor ikke integrert med resten av systemet. Det vil ikke være så mye arbeid å legge dette verktøyet til systemet.

Linux versjon:

Siden det også finnes en versjon av BSPlab til linux, kunne det være naturlig å lage en utgave av de grafiske omgivelsene også for Linux

6.3 Erfaringer gjennom prosjektet

Hvis jeg skulle beskrive prosessen med å utføre denne oppgaven ved hjelp av adjektiver ville jeg valgt disse:
Lærerikt, inspirerende, frustrerende, skuffende, tilfredsstillende, stressende og morsomt

Det har vært mange opp og nedturen. Tidsplaner som har sprukket, problemer som har dukket opp, problemer som har blitt løst og ikke minst mange nye ting som har blitt lært.

Hele prosessen har vært en eneste lang læringsprosess. Nye verktøy jeg ikke har brukt før, problemer jeg ikke har vært borti før osv. Når jeg startet på denne oppgaven hadde jeg aldri programmert i Python før og jeg hadde ingen kjennskap til BSPlab. Det har vært mye som måtte læres underveis, og til tider har frustrasjonsnivået vært faretruende høyt. Men den følelsen man da får etter å ha stått bom fast i ett eller annen problem som fremstår som uløselig, og man har greid å løse det kan vanskelig beskrives med ord.

I tillegg til den faglige lærdommen jeg har ervervet meg gjennom gjennomføringen av denne masteroppgaven, har jeg også lært meg selv bedre å kjenne. Jeg har fått en bedre forståelse av hvordan jeg takler både med og motganger, og ikke minst hvordan jeg takler stress. Dette er lærdom jeg tror jeg vil ha stor nytte av når jeg nå skal ut i arbeidslivet.

7 Bibliografi

[Bisseling, 2005] Rob H. Bisseling 2005, Lecture presentation , http://www.math.uu.nl/people/bisselin/PSC/psc1_2.pdf

[bsp-worldwide,2006] BSP Wordwide ,2006, http://www.bsp-worldwide.org/
[Py2exe] py2exe, www.py2exe.org

[CEBPA, 2006]. PARAVER, 2006, http://www.cepba.upc.es/paraver/index.html

[cplusplus.com, 2006] printf function, http://www.cplusplus.com/ref/cstdio/printf.html

[Distutilis] Python Distribution Utilities, http://www.python.org/doc/current/dist/

[Dybdahl, Uthus,1997] Håkon Dybdahl, Ivan Uthus, *Simulation of the BSP model on diferent computer architectures*. Diplomoppgave IDI, NTNU, 1997

[Hill,Jarvis Siniolkism,Vasilev, 1998] Jonathan M.D. Hill, Stephen Jarvis, Constantinos Siniolakis, Vasil P. Vasilev. ``*Portable and architecture independent parallel performance tuning using a call-graph profiling tool*'' 6th EuroMicro Workshop on Parallel and Distributed Processing (PDP'98).

[Hill, Crumpton, Burgess, 1996] Jonathan M. D. Hill, Paul I Crumpton, David A. Burgess, 1996 ``*The theory, practice, and a tool for BSP performance prediction*'' EuroPar'96, number 1124 in LNCS, pages 697-705

[Inno Setup, 2006] Inno Setup http://www.jrsoftware.org/isinfo.php

[Oxford BSP toolset, 1998] Oxford BSP toolset, 1998 , http://www.bsp-worldwide.org/implmnts/oxtool/

[PIL, 2006] Python Imaging Library, http://www.pythonware.com/products/pil/

[Python] Python, http://www.python.org

[Mayhew,1992] Mayhew, D.J. ,1992
Principles and guidelines in software user interface design, Prentice Hall, Englewood Cliffs, New Jersey

[MDI , 2006] MDI, http://en.wikipedia.org/wiki/Multiple_document_interface

[MinGW, 2006] MinGW, 2006, http://www.mingw.org/

[Natvig, 2001] Lasse Natvig, 2001, Paper fra konferasansen: High-Performance Computing and Networking, HPCN

[NumericPython, 2006] Numeric Python http://sourceforge.net/projects/numpy/

[Tkinter, 2006] Tkinter , http://wiki.python.org/moin/TkInter

[Valiant, 1990] Valiant, L. G. (1990). A Bridging Model for Parallel Computation, Communications of the ACM, Vol. 33, No. 8. side 103-111

[Wikipedia, Von Neuman] Von Neuman arkiktekturen, http://en.wikipedia.org/wiki/Von_Neumann_architecture
``*Portable and architecture independent parallel performance tuning using a call-graph profiling tool''* Jonathan M.D. Hill, Stephen Jarvis, Constantinos Siniolakis, and Vasil P. Vasilev. In 6th EuroMicro Workshop on Parallel and Distributed Processing (PDP'98). IEEE Computer Society Press, January 1998

[wxPython] wxPython, http://www.wxpython.org

[Østby,Lundereng, 2005] Erik Østby og Torje Lundereng, *BSPlab til folket*. Diplomoppgave, IDI NTNU, 2005
Ln artikkel

Appendix A Filvedlegg

Det er lagt til en del filer sammen med denne rapporten.
Først og fremst ligger installasjonsfilen som inneholder selve systemet og BSPlab vedlagt. I tillegg ligger all kildekode og andre filer som er utviklet vedlagt i mappen kildefiler.

Appendix B Testkode

```
/*============================================================================
PROGRAM      MERGE SORT
Written by   D. Kim   (dkim@home.korea.ac.kr)
Date         July 30, 1996
Description:

This is a merge sort program using BSPlib of Oxford Parallel.
The number of data to be sorted is given by SIZE, and the number of processors
used is decalared by NPROC (currently #PROC=4, #data=32).
The data array is splited into the number of processors with equal
number of data.
The partitioned array each is sorted sequentially by each assigned
processor
and then merged in log (#PROCS) steps.
Superstep synchronization is made at the merging process.

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

#ifndef BSP_H_
#include <bsp.h>
#endif

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define SIZE 81920
#define NPROC 10


void bsp_msort(int *my_array, int *your_array, int n, int *out_array)
{
   int i;
   int my_ptr, your_ptr;

   for (i=0, my_ptr=0, your_ptr = 0; i<2*n ; i++)
   {
      if ((my_ptr <n)&&(your_ptr<n))
         if (my_array[my_ptr] < your_array[your_ptr])
            out_array[i] = my_array[my_ptr++];
      else
         out_array[i] = your_array[your_ptr++];
      else if (my_ptr ==n)
         out_array[i] = your_array[your_ptr++];
      else if (your_ptr ==n)
         out_array[i] = my_array[my_ptr++];
      else printf("Something wrong in comparison\n ");
   }
```

```
}


void bsp_exchg_sort(int *array, int curr_size)
{
   int i, curr_bound, j, jj;
   int *result= (int*) calloc(curr_size,sizeof(int));
   int *your_data = (int*) calloc(curr_size,sizeof(int));
   int *mine = (int*) calloc(curr_size,sizeof(int));
   bsp_pushregister(your_data,curr_size*sizeof(int));
   bsp_sync();

   curr_bound = curr_size/NPROC;

   for (j=0; j< curr_size; j++)
      if (bsp_pid() == j /(curr_size/NPROC))
         mine[j%(curr_size/NPROC)] = array[j];

   for(i=1;i<NPROC ;i*=2)
   {

      for (j=0;j < NPROC; j+=2*i)
      {
         if ((bsp_pid()- i ) % (2*i) == 0)
         {

            bsp_put(bsp_pid() -i, mine, your_data,0,curr_bound*sizeof(int));
         }
         bsp_sync();

         for (j=0;j < NPROC; j+=2*i)
            if (bsp_pid() == j) bsp_msort(mine, your_data, curr_bound,result);
         bsp_sync();


      }
      for (j=0;j < NPROC; j+=2*i)
         if (bsp_pid() == j)
            for (jj=0; jj < 2* curr_bound; jj++)
               mine[jj] = result[jj];
      curr_bound *= 2;
      bsp_sync();

   }
   bsp_popregister(your_data);
   /* NOW COPY THE SORTED ARRAY TO MAIN ROUTINE */
   if (bsp_pid() == 0)
      for (jj=0; jj < curr_size; jj++)
         array[jj] = result[jj];
}


void print_array(int *array, int n)
{
```

```
   int i,j;
   for (i=0; i < NPROC; i++){
   if ( i== bsp_pid()){
   printf("\n INIT. DATA AT PID_%d = \n", bsp_pid());
   for(j=0;j<n;j++) printf("%d ",array[j]);
   printf("\n");
   fflush(stdout);
   }
   }
}


void init_sort(int *dat,int m, int pid)
{
   int i,j,amin, index,tmp;

   for (i=0; i< m; i++)
   {
      amin = dat[i];
      tmp =dat[i];
      index =i;
      for (j=i; j< m; j++)
         if (dat[j] < amin)
      {
         index=j;
         amin = dat[j];
      }
      dat[i] = amin;
      dat[index] = tmp;
   }
}


void copy_array(int *array, int *result, int n, int pid)
{
   int i;
   int  start = pid * (SIZE/NPROC);
   for  (i=0; i< n/NPROC; i++)
      array[start+i] = result[i];
}


void init(int *array, int n)
{
   int start;
   int i,j;
   int *result= (int*) calloc(n, sizeof(int));
   int *my_array= (int*) calloc(SIZE/NPROC, sizeof(int));
   for (i=0; i< NPROC; i++)
      if (bsp_pid()==i)
   {
      start = i * (SIZE/NPROC);
      for (j=0; j<SIZE/NPROC ; j++)
         my_array[j] = array[start +j];
```

```
      init_sort(my_array, SIZE/NPROC,i);
      copy_array(array, my_array,n,i);
   }
}


void bsp_main(int argc, char **argv)
{
   int i,j,n,*xs;


   //Modified by Haakon, kind of stupied to allocate
   //all processors when only using NPROC of them..
   //bsp_begin(bsp_nprocs());
   bsp_begin(NPROC);
   n=SIZE;
   xs = (int*) calloc(n,sizeof(int));

   for (i=0;i<n;i++)
      if ( i%2 ) xs[i]= i % n;
   else xs[i] = (n-i) % n;
   init(xs,n);                          /* Sequentially sort and placement to
each proc before merging */
   if (bsp_pid()==0)  print_array(xs,n);
   bsp_exchg_sort(xs,n);

   if (bsp_pid()==0) {
   printf("  FINAL SORTED ARRAY at PROC(%d)\n ",bsp_pid());

   for(j=0;j<n;j++) printf("%d ",xs[j]);
   printf("\n");
   fflush(stdout);
   }

   bsp_end();
}
```

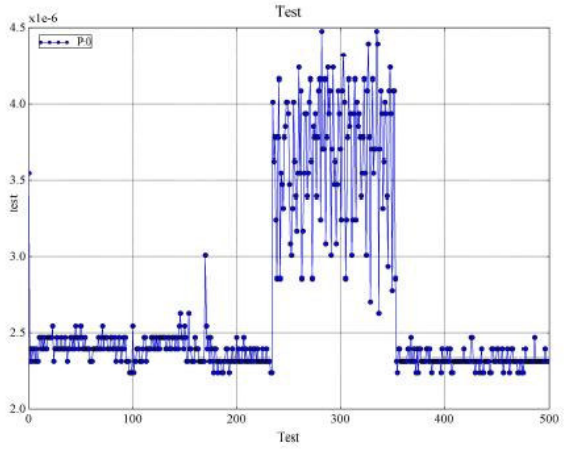
APPENDIX B TESTKODE

```
/*=========================================================================
This is a simple outputprogram in “hello world” style
It prints 15 general messages and 15 pid messages for each processor pr.
superstep. Runs over 25 supersteps
Erlend S. Klepaker
=========================================================================*/

#include <bsp.h>
#include <stdio.h>
#include <t.cpp>
#include <String>
void bsp_main(int argc, char **argv)
{
   int i;

   bsp_begin(bsp_nprocs());

   for(i=0;i<25;i++)
   {
     if (0==0  )
     {
         for(int y=0;y<15;y++)
         {
                  pid_print("As you see I'm processor %d of
%d",bsp_pid(),bsp_nprocs());
                  printf("HELLO world from process %d of
%d\n",bsp_pid(),bsp_nprocs());
         }

      bsp_sync();
  }

   bsp_end();
}
```

Appendix C Eksempelrapport

BSPLab oving 12

Dette er oving 12 i Datamaskin Arkitektur.
Gruppe medlemmer gruppe 2:
Anders
Ole
Jens



Her ser vi en graf vi har laget. Den viser hvor lang tid prosessor 0 har brukt fordelt paa SuperStep