

CPHSTL's benchmark værktøj

Christian Ulrik Sættrup & Jakob Gregor Pedersen

CPH STL Report 2003-1, April 2003.

Indhold

1	Indledning	2
2	Installationsvejledning	3
3	Benchmark værktøjet	4
3.1	Programbeskrivelse	4
3.1.1	Designvalg	6
3.2	Brugervejledning	7
3.2.1	Programmet	7
3.2.2	Skemaer	7
3.3	Case Study	10
3.3.1	Opgaven	10
3.3.2	Løsningen	11
4	Benchmark-skema generatorsiden	16
4.1	Program beskrivelse	16
4.2	Bruger vejledning	16
4.2.1	General input	17
4.2.2	Remote execution	18
4.2.3	Compiler settings	18
4.2.4	Gnuplot settings	18
4.2.5	Time settings	18
4.2.6	Memory measurement	19
4.2.7	Function settings	19
4.2.8	Generate form	20
A	PAPI	22
B	SSh keys	23
C	Parametre	24

Kapitel 1

Indledning

Dette dokument omhandler et benchmark værktøj tilknyttet CPHSTL. Værktøjets formål er først og fremmest, at lette måling af diverse ydelsesindikatorer i forbindelse med udvikling af programmer til CPHSTL, der er dog intet i vejen for at benytte det på andet end CPHSTL's programmer. Udover at lette måling af ydelse er værktøjet også udviklet på baggrund af et ønske om at standardisere disse målinger, for at lette overskueligheden og sammenligneligheden af CPHSTL's programmer. Vi forventer, at læseren er bekendt med Python. Er dette ikke tilfældet kan kapitel 4 om generatorsiden læses, der kan generere benchmark-skemaer. Så man stadig kan benytte værktøjet selvom man ikke er bekendt med Python.

Værktøjet er initialt skrevet af Jyrki Katajainen og senere udvidet af dette dokumentets forfattere.

Da værktøjets input også er skrevet i et programmeringssprog tilbydes uanede tilpasningsmuligheder. Ønskes andre ydelsesindikatorer end dem, der er til rådighed i værktøjet, er det muligt for en bruger at skrive sin egen driver og bruge denne istedet for en af de indbyggede. Ønskes resultatet af en benchmark kørsel præsenteret på en anden måde end de umiddelbart understøttede, kan en bruger udvikle sin egen runner og benytte denne i stedet. Det er altså kun nødvendigt at udskifte en lille komponent, og således kan værktøjet hjælpe langt hen af vejen, selvom den ønskede funktionalitet ikke er indbygget.

De mange muligheder komplicerer desværre benchmarkproceduren en del og harmonerer derfor ikke så godt med formålet om at lette disse målinger. Netop derfor er der i forbindelse med værktøjet udviklet en hjemmeside til at lette brugen af værktøjets indbyggede funktionalitet. I de følgende kapitler gennemgås brugen af værktøjet og det underliggende programmer beskrives, således at eventuelt kommende ændringer eller udvidelser kan udføres forholdsvis smertefrit af udefra kommende.

Kapitel 2

Installationsvejledning

Hvis det ikke allerede er gjort, skal værktøjet hentes fra `cphstl.dk` enten via CVS (se [5]) eller via `cphstl`'s hjemmeside (<http://www.cphstl.dk>). Vælges hjemmesiden er det nemmeste at klikke på *Downloads* i venstre side af hjemmesiden for derfor at vælge `CPHSTL benchmark tool`. Herefter skal værktøjet pakkes op, dette gøres med:

```
#prompt#:tar xzvf benchmark.tar.gz
```

Værktøjet pakkes nu ud i det nuværende bibliotek. For at værktøjet kan finde nødvendige filer, skal der sættes en shell-variabel, som indeholder stien, hvor værktøjet er installeret (nærmere bestemt stien hvor `benchmark.py` ligger). Dette kan muligvis gøres med følgende kommando afhængig af shell'en der bruges:

```
#prompt#:setenv PYTHONPATH stien_til_din_installation:$PYTHONPATH.
```

Hvis det ikke virker så prøv:

```
#prompt#:export PYTHONPATH=stien_til_din_installation:$PYTHONPATH.
```

Du kan checke om systemet har sat variabelen med:

```
#prompt#:echo $PYTHONPATH.
```

Desværre "glemmer" shell'en ovenstående sti mellem logins, så medmindre man har en sygelig trang til at gentage det, vil det være en god idé at indsætte ovenstående i ens startup fil. Bruges `bash` som shell skal linien indsættes i filen `.bashrc`, bruges istedet `tcsh` skal linien indsættes i `.tcshrc` filen osv. Hvis man vil benytte værktøjet til at måle depotkiks eller sidefejl, er det nødvendigt at have en maskine med PAPI (se bilag A) tilgængelig. På denne maskine skal man tilføje stien til PAPI biblioteket til lænkerens søgesti, f.eks.:

```
#prompt#:export LD_LIBRARY_PATH=stien_til_PAPI_lib/:$LD_LIBRARY_PATH
```

Værktøjet er nu installeret og klar til brug, det anbefales dog kraftigt at brugervejledningen læses inden brug.

Kapitel 3

Benchmark værktøjet

I dette kapitel gennemgås benchmarkværktøjets klasser i programbeskrivelsen. I brugervejledningen gennemgås det, hvordan man benytter værktøjet til at lave sine egne benchmarks og i case study gennemgås et eksempel. Web-baseret dokumentation af værktøjet kan ses på www.cphst1.dk ved at vælge *benchmark* på startsiden. En liste af klassernes attributter og deres forvalg forefindes i bilag C.

3.1 Programbeskrivelse

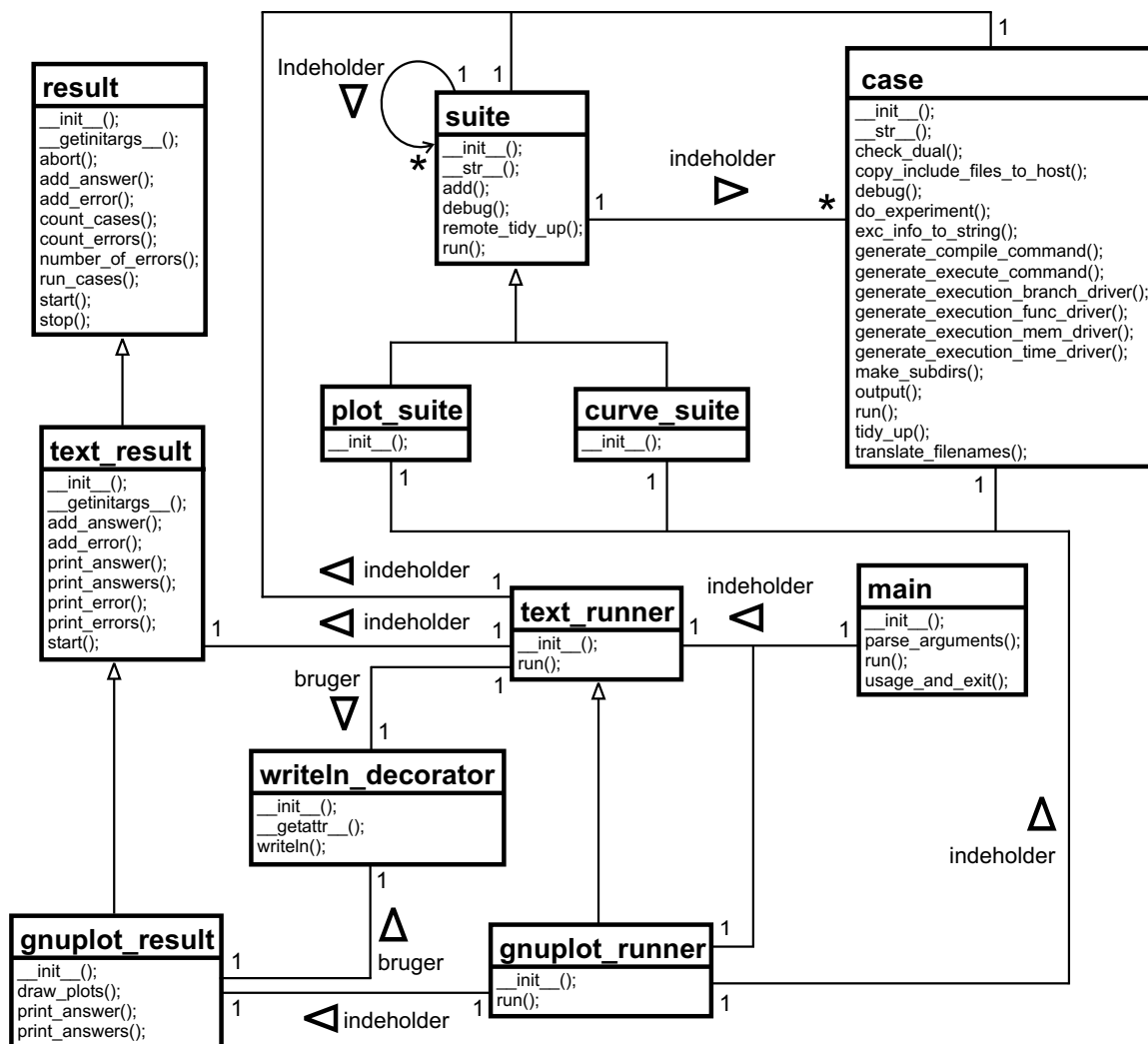
Værktøjet er skrevet i Python [1] og er modulært opbygget vha. klasser. Figur 3.1 viser disse klasser og deres indbyrdes forhold. Der knytter sig nogle kommentarer til figuren, som er udarbejdet i henhold til UML specifikationen (se [3]). Forbindelser, som deler sig, f.eks. mellem *main*-klassen, *text_runner*-klassen og *gnuplot_runner*-klassen, angiver at enten den ene eller den anden forbindelse gælder, men ikke dem begge. I tilfældet med *main*-klassen, *text_runner*-klassen og *gnuplot_runner*-klassen betyder det altså, at *main*-klassen enten indeholder en instans af *text_runner*-klassen eller en instans af *gnuplot_runner*-klassen. Løkken ved *suite*-klassen angiver, at denne klasse indeholder 0 eller flere instanser af sin egen klasse. Nedenfor gennemgås disse klasser, hvorefter et afsnit om designvalg følger.

main

Denne klasses primære opgave er at indlæse eventuelle argumenter fra kommandolinjen, parse disse og derefter kalde den valgte runner. Desuden kan klassen udskrive kommandolinjemulighederne. Det er fra denne klasse, at udførelsen af en benchmark kørsel starter.

text_runner & gnuplot_runner

text_runner klassen sørger for afvikling af benchmark-suites eller enkelte benchmarktilfælde. Resultatet af denne afvikling udskrives af denne klasse med hjælp fra *_writeln_decorator*-klassen til brugerens skærm, sammen med eventuelle fejl opstået under afviklingen. *Gnuplot_runner* klassen arver fra *text_runner*-klassen og udvider denne med *gnuplot* funktion-



Figur 3.1: Klassediagram for benchmark værktøjet

alitet. Denne klasse afvikler ligesom text_runner-klassen benchmark-suiter eller enkelte benchmarktilfælde, men returnerer istedet resultatet af afviklingen i et gnuplot script og en postscript-fil. Eventuelle fejl vises stadig på skærmen.

result, text_result & gnuplot_result

result-klassen fungerer som et bogholderi med hensyn til resultater og fejl fra benchmarktilfælde. text_result-klassen arver fra result-klassen og udvider denne med funktionalitet til at udskrive resultater og eventuelle fejl fra result-klassen. Denne klasse bruges af text_runner-klassen til at udføre det grove arbejde.

Ligesom text_runner-klassen bruger text_result-klassen bruger gnuplot_runner-klassen gnuplot_result-klassen til at udføre selve udskrivningen. Denne klasse udvider text_result-klassen med funktionalitet til at skrive gnuplots.

suite, curve_suite & plot_suite

En instans af suiteklassen afvikler enten instanser af suite-klassen eller en suite af benchmarktilfælde. I tilfælde af fjernudførelse sørger suite-klassen desuden for oprydning på fjernmaskinen og en eventuel gatewaymaskine. `curve_suite`-klassen, som arver fra suite-klassen, angiver hvorledes en enkelt kurve i et gnuplot skal tegnes, og benyttes således af `gnuplot_runner`-klassen. `plot_suite`-klassen benyttes ligeledes af `gnuplot_runner`-klassen. Denne bruges til at angive overordnede specifikationer for et plot.

case

Denne klasse er uden tvivl den største klasse i værktøjet. Denne klasse repræsenterer et enkelt benchmark tilfælde. Heri bliver driveren genereret, hvis en af de indbyggede benyttes, nødvendige filer kopieret til fjernmaskinen i tilfælde af fjernudførelse, driveren oversat og til sidst kørt. Klassen tager sig desuden af oprydning på lokalmaskinen samt på gatewaymaskinen og fjernmaskinen, hvis tilfældet ikke er en del af en suite.

3.1.1 Designvalg

Dette afsnit omhandler nogle af de valg, der er truffet i forbindelse med udviklingen af værktøjet. Da vi kun har udvidet benchmark værktøjet, kan vi kun udtale os om designvalg i forbindelse hermed, ligesom det vil være umuligt at behandle alle valg. Især i forbindelse med fjernudførelse har vi måtte træffe en del valg.

Vi har valgt at oversætte drivere på fjernmaskinen, da visse drivere kræver et specielt miljø for at kunne oversættes. Dette gælder bla. for memory driveren, som kræver at PAPI (se bilag A) er installeret for at kunne oversættes. Dette valg indebærer desværre en masse kopiering af filer til fjernmaskinen med resulterende høj køretid af værktøjet til følge.

I forbindelse med en gatewaymaskine har vi valgt først at kopiere filerne til gatewaymaskinen og derefter til fjernmaskinen. Alternativet til denne metode ville være at forbinde til gatewaymaskinen og udføre en kopi-kommando på denne, som kopierer fra lokalmaskinen til fjernmaskinen. Da det er muligt at nægte forbindelser den ene vej samtidig med forbindelser tillades den anden vej i et netværk, er det ikke altid muligt at forbinde fra gatewaymaskinen til lokalmaskinen. Da dette netop er tilfældet på DIKU har vi valgt den første metode, selvom dette også bidrager gevaldigt til køretiden af værktøjet.

Når nødvendige filer kopieres til fjernmaskinen bliver de samlet i et unikt bibliotek på fjernmaskinen for at lette oprydning og hjælpe på udførelstiden af værktøjet. I forbindelse med fjernudførelse af en suite af benchmarks gøres en del antagelser for at mindske køretiden. Under kopiering opretholdes en liste af allerede kopierede filer, således at hvis en senere benchmark har brug for de samme filer, bliver de ikke kopieret igen. For at holde denne liste simpel antages det, at alle benchmarks i samme suite køres på samme maskine. Denne antagelse går igen under oprydningen, hvor kun biblioteket på maskinen, hvor den første benchmark blev udført, bliver slettet.

Vi har valgt at bruge den samme protokol til at skabe forbindelse til både gateway og

fjernmaskinen for at forenkle værktøjet. Det samme gør sig gældende for protokollen, som bruges til at overføre filer. Der er heller ikke eksplicit mulighed for at angive forskellige brugernavne på maskinerne. Som standard benytter protokollerne samme brugernavn på fjernmaskinen, som brugeren der kører kommandoen. Det problem kan man dog komme ud over ved at angive hostnavnet som `andetbrugernavn@fjernmaskinen.dk`. Funktionerne, der genererer driver filer, sætter også oversætter switches, sådan at driveren kan kompiles. Disse switches er dog specielt til gcc version 3.2.2. som er den, der er installeret på DIKU. Disse er ikke nødvendigvis de samme i ældre versioner eller for andre oversættere. Hvis ens oversætter ikke virker med de forvalgte switches, skal man blot overskrive variabelen `case.compiler_options` efter driveren er genereret.

3.2 Brugervejledning

For at bruge benchmarkværktøjet skal man enten skrive sit eget lille Python program, der sætter de nødvendige variable, eller generere dette program via benchmark-skema generatorsiden (se kapitel 4). Der er også mulighed for at køre benchmarks direkte fra kommando linien. For at se hvordan køres `./benchmark.py --help`.

3.2.1 Programmet

For at køre benchmarkværktøjet kræves det, at der findes en testklasse (navnet kan vælges frit) på formen:

```
class test {
    test(int n);
    int primal();
}
```

Der kræves altså en klasse, der indeholder en konstruktor, som danner test tilfældet og en funktion `primal` på klassen, der udfører den kode, som ønskes benchmarket. Hvis klassen også indeholder en dualfunktion, vil tiden, der bliver brugt i denne, trækkes fra den der bruges i `primal` funktionen.

3.2.2 Skemaer

Ved at skrive sit eget skema opnås større alsidighed end ved brug af skema generatorsiden. Ideen bag disse skemaer er, at tilpasse værktøjets indbyggede klasser til et specifikt formål. Dette gøres ved at nedarve fra værktøjets klasser og derefter udvide med ny funktionalitet og/eller overdefinere funktioner for at opnå ens formål. Nedenfor gennemgås hvorledes dette gøres. En mere detaljeret gennemgang kan ses i case study afsnittet. Den bedste måde at forstå benchmarkskeemaer på, er at se en masse eksempler. Sådanne findes i kataloget `Examplebase/` i værktøjets installationskatalog.

case

Den mest basale klasse er `case`. Denne klasse er man nødt til at nedarve for at lave et benchmark tilfælde, der kan køre ens kode. Der skal angives en driverfil (linie 8 nedenfor), som er den fil, der oversættes, køres og måles på. Bruges en af de indbyggede driver-generatorer, skal konstruktor kaldet angives (linie 7 nedenfor). Det er muligt at ændre værdierne af klassens attributter ved blot at sætte dem til noget nyt (se bilag C med attributternes forvalg). For at få den nedarvede klasse initialiseret med standardværdierne, er det nødvendigt at kalde den nedarvede konstruktor (linie 3 nedenfor). Er standard funktionerne ikke tilfredstillende kan disse overdefineres. For eksempel kan `tidy_up` overdefineres til bare en `pass`. På denne måde bliver alle filer efterladt, og det er muligt at se hvilke filer, der er blevet genereret osv. Herunder er et meget simpelt eksempel på en klasse med en konstruktor, der afhænger af to værdier, og en driverfil genereret af den indbyggede tidsmålingsfunktion. Programmet, som ønskes benchmarket, er defineret i filen `minklasse.cpp`. Funktionen `output` overdefineres til returnere en tuple, der indeholder resultatet og den anden af de værdier, som konstruktoren er blevet kaldt med.

```
1 class min_case(benchmark.case):
2     def __init__(self, n, k):
3         benchmark.case.__init__(self)
4         self.n = n
5         self.k = k
6         self.include_files.extend(['minklasse.cpp'])
7         self.constructor_call='minklasse(' + str(n) + ', ' + str(k) + ')'
8         self.driver_file = self.generate_cpu_time_driver()
9
10    def output(self):
11        return (self.k, self.driver_output)
```

Der kan vælges følgende indbyggede driver-generatorer:

- `generate_execution_time_driver` - tidsmåling.
- `generate_execution_mem_driver` - sidefejl og cache målinger
- `generate_execution_branch_driver` - målinger af betinget hop
- `generate_execution_function_driver` - funktions statistik

Vælges tidsmåling er det en god idé at vælge måleenheden ved at sætte variabelen `time_unit` og angive, hvorvidt dualfunktionen eksisterer ved at sætte `dual_exists` variabelen. Vælges sidefejl og cache målinger er det en god idé at vælge, hvorledes målingerne skal tages ved at sætte `papitype` variabelen. Desuden skal `output` funktionen overdefineres til at returnere den ønskede måling, hvis `gnuplot_runner` benyttes. Driveren returnerer en streng, der indeholder major page faults, minor page faults, L1 datacachemiss og L1 datacacheaccesses adskilt med semikolon. For at vælge den ønskede måling, skal man parse outputtet, jævnfør nedenstående eksempel.

Eksempel hvor L1 datacache miss hentes ud:

Output

```
def output(self):
    cachemiss = string.split{self.driver_output,",";"}[2]
    return (self.k, cachemiss)
```

Desuden skal `papi_path` sættes til stien hvor PAPI er installeret, hvis sidefejl og cache målinger vælges. Vælges målinger af betinget hop skal `papi_path` også sættes samtidig med at `output` funktionen skal overdefineres til at returnere den ønskede måling. Driveren returnerer en streng, der indeholder forfejlet hop forudsigelse og antal af betingede hop adskilt af semikolon. For at vælge den ønskede måling skal man parse outputet, ligesom i ovenstående eksempel.

Hvilken funktion, der skal profileres, skal vælges ved at sætte variabelen `counted_function`, hvis funktionsstatistik vælges. Desuden skal `output` funktionen overdefineres, hvis `gnuplot_runner` bruges. Outputet af denne driver er den linie fra programmet `gprof`, som svarer til den valgte funktion. Output skal som før parses. En beskrivelse af output linien kan findes i `man gprof`.

Her er et eksempel, der henter antallet af kald ud:

Output

```
def output(self):
    return (self.n, string.split(self.driver_output)[3])
```

suite

`suite` klassen bruges til at udføre en række af benchmarks. Ligesom med `case` er det muligt at overdefinere funktioner og variable. Det vigtigste at huske er at tilføje benchmark tilfælde ved at køre `suite.add(tilfælde)`. `suite` har to nedarvninger `plot_suite` og `curve_suite`. Det eneste de to nedarvninger gør er at tilføje forvalg til `gnuplot` for henholdsvis hele plots og enkelte kurver. Disse nedarvninger skal kun benyttes, når resultatet af en benchmark kørsel ønskes udskrevet i et `gnuplot`. Her er et eksempel:

```
class min_curve(benchmark.curve_suite):
    def __init__(self):
        benchmark.curve_suite.__init__(self)
        self.title = 'simpel_curve_k=5'
        self.k=5;
        for n in range(10,21):
            self.add(min_case(n,k))

class min_plot(benchmark.plot_suite):
```

```
def __init__(self):
    benchmark.plot_suite.__init__(init)
    self.add(min_curve())
```

result og runner

`result` og dens to nedarvninger `text_result` og `gnuplot_result` samt `runner` og dens to nedarvninger `text_runner` og `gnuplot_runner` har man stort set aldrig brug for at overdefinere. Der kan dog være special tilfælde, hvor man har brug for det. Der er det vigtigt at holde sig for øje, at hvis man laver en ny result-klasse, der nedarver `result`, skal man også lave en ny runner-klasse, der benytter den nye result-klasse.

main

Ligesom for `runner` og `result` er der stort set aldrig brug for at overdefinere denne klasse. For at få kørt sine benchmarks, kan man benytte den indbyggede `main`-klasse. Det gøres således:

```
if __name__ == '__main__':
    benchmark.main(task = min_plot(), runner = benchmark.text_runner)
```

Det ses at `main` skal kaldes med en opgave, som enten er en enkelt benchmark eller en suite af benchmarks, som i ovenstående tilfælde. Desuden skal `main` kaldes med en runner. Vælges en af de indbyggede er mulighederne:

- `text_runner` - udskriver resultaterne til skærmen
- `gnuplot_runner` - udskriver resultaterne til et gnuplot

3.3 Case Study

Vi gennemgår her, hvorledes benchmarkværktøjet benyttes til at løse en given opgave.

3.3.1 Opgaven

Der er til CPHSTL udviklet en asymptotisk hurtigere udgave af `inplace_merge`. Det ønskes at vide, hvor hurtig den er i forhold til den, der er indbygget i standard STL. Samtidig ønskes en oversigt over hvor mange gange sammenligningsfunktionen bliver kaldt, således at det kan sammenlignes med teorien. Til sidst er det også interessant at vide, om algoritmen udnytter cachen fornuftigt, dvs. at også cachemissforholdet ønskes målt. Vi vælger, at den skal måles fra 10.000-50.000 i skridt af 10.000.

3.3.2 Løsningen

I konstruktoren vil vi danne en tilfældig liste af tal, hvor længden er fastsat af værdien, konstruktoren bliver kaldt med. I primal kalder vi så `cphstl::inplace_merge_ukkonen(...)` og i dual `std::inplace_merge(...)`. Så måler benchmark værktøjet forskellen i tid. Testprogrammet kommer så til at se ud som nedenunder:

```
#include <algorithm>

int compare(int a, int b){
    return a-b;
}

class inpluko{
    int * array;
    int * ukkoarray;
    int * start ,*midt ,* slut ;
    int * ukkostart ,*ukkomidt ,*ukkoslut ;
    int lengthA ,lengthB ,length ;

public:
    inpluko(int n){
        length=n;

        ukkoarray = new int [length];
        lengthA = length / 2;

        ukkostart=ukkoarray;
        ukkomidt=ukkoarray+lengthA;
        ukkoslut=ukkoarray+length;

        int * temp;
        for (temp=ukkoarray ; temp<ukkoslut ; temp++){
            *temp = rand() % length;
        }
        std::sort (ukkostart , ukkomidt);
        std::sort (ukkomidt , ukkoslut);

        array = new int [length];
        memcpy (array , ukkoarray , length*sizeof(int));
        start=array;
        midt=array+lengthA;
        slut=array+length;
    }

    ~inpluko(){
        delete array;
        delete ukkoarray;
    }

    int primal(){
```

```

cphstl::inplace_merge_ukkonen((int *) ukkostart ,
    (int *) ukkomidt , (int *) ukkoslut , compare);
return 0;
}

int dual(){
    std::inplace_merge((int*) start , (int *) midt ,
        (int *) slut , compare);
    return 0;
}
};

```

inpluko_benchmark.c++

I konstruktoren dannes to sorterede følger i et kontinuert stykke hukommelse. Primal kører den nye version af `inplace_merge` på en kopi af denne, og `dual` kører standard versionen på en anden kopi.

case klassen

Når alle fejl er fjernet fra koden, benytter man benchmark-skema generatorsiden på www.cphstl.dk, og retter eventuelt resultatet til, da det er meget nemmere end at skrive skemaerne fra bunden. Her vil vi dog gennemgå, hvordan man skriver et fra bunden. Vi har valgt at beskrive `cacherratio` eksemplet. Først har vi brug for en ny case klasse, der kan beskrive driverfilen.

Inpluko case

```

1  class inpluko_case(benchmark.case):
2      def __init__(self, n):
3          benchmark.case.__init__(self)
4          self.n = n
5          self.computer = 'cphstl.projektlab.diku.dk'
6          self.papi_path = '/usr/local/papi'
7          self.papitype = 'single'
8          self.include_files.extend(['inpluko.h', 'inpluko_benchmark.c++'])
9          self.constructor_call = 'inpluko(' + str(n) + ')'
10         self.connection_protocol = 'ssh'
11         self.transfer_protocol = 'scp'
12         self.dual_exists = 0
13         self.driver_file = self.generate_execution_mem_driver()
14
15     def output(self):
16         cache_misses = string.split(self.driver_output, ";")[2]
17         cache_accesses = string.split(self.driver_output, ";")[3]
18         result = float(cache_misses)/float(cache_accesses)
19         return (self.n, result)

```

Det første man gør er at køre konstruktoren på den nedarvede klasse, sådan at man ikke skal foretage al initialiseringen selv. For det første kræver cachemålingerne, at benchmarkene udføres på en maskine med PAPI understøttelse, derfor sætter vi det til at køre på `cphst1.projektlab.diku.dk` (se linie 5). I linie 10 og 11 angiver vi at, hvis den skal forbinde til en fjernmaskine, skal det foregå med `ssh` og `scp`. Det kræves, at man kan forbinde uden at skulle angive kodeord (se hvordan dette gøres i bilag B). Hvis skriptet bliver udført på en anden maskine vil den automatisk prøve at forbinde dertil. Når man skal bruge PAPI skal man huske at angive stien, det er gjort i linie 6. I linie 7 angiver vi, at vi ikke vil bruge opvarmning af cachen. For at være sikker på, at koden er tilgængelig for den genererede driver tilføjer vi de filer, der indeholder koden, til linie 8, så bliver de også automatisk kopieret til fjernmaskinen. Derefter angiver vi konstruktorkaldet i linie 9, det består af konstruktorkaldet til vores `inpluko` klasse med det `n` som `inpluko_case` blev kaldt med, sådan at det er nemt at lave en række af cases med varierende længde af følger. Vi benytter ikke `dual` funktionen i dette tilfælde og det er derfor slået fra. Tilslidst bruger vi den indbyggede funktion til at generere en driverfil med hukommelsesunderstøttelse. For at få det rigtige output bliver vi nødt til at overdefinere `output` funktionen. I linie 16 og 17 henter vi `miss` og `access` statistikken ud fra driver outputet. Derefter udregner vi ratioen og returnerer en tuple af `n` værdien og `cachemissratioen`, så vi kan plote ratioen mod længden af følgerne.

curve klassen

Nu har vi en klasse, der beskriver et benchmark tilfælde, men vi har brug for at lave en kurve i et `gnuplot` skript med de specificerede værdier. Dette gøres nedenfor.

Inpluko case

```
21 class inpluko_curve(benchmark.curve_suite):
22     def __init__(self):
23         benchmark.curve_suite.__init__(self)
24         self.title = 'ratio'
25         self.add(inpluko_case(10000))
26         self.add(inpluko_case(20000))
27         self.add(inpluko_case(30000))
28         self.add(inpluko_case(40000))
29         self.add(inpluko_case(50000))
```

Vi vil have kurven til at have navnet `ratio`. Derefter tilføjer vi de benchmarks, vi vil måle på.

plot klassen

Så mangler vi blot en plot klasse, som kan indeholde kurven.

```

30 class inpluko_plot(benchmark.plot_suite):
31     def __init__(self):
32         benchmark.plot_suite.__init__(self)
33         self.title = 'Inplace_merge_by_kukkonen'
34         self.xlabel = 'n'
35         self.ylabel = 'Cache_miss_ratio'
36         self.add(inpluko_curve())
37         self.gnuplot_commands += """
38 _set_title_%(title)s'
39 _set_xlabel_%(xlabel)s'
40 _set_ylabel_%(ylabel)s'
41 """ % self.__dict__

```

Størstedelen af denne klasse er blot til for, at der skal stå de rigtige ting på det færdige plot. Den interessante del er linie 36, hvor vi tilføjer vores kurve til de suiteer, der skal udføres af plot klassen.

main

Til sidst skal vi sørge for at vi kan udføre programmet.

Inpluko case

```

42 if __name__ == '__main__':
43     benchmark.main(task = inpluko_plot(), runner = benchmark.gnuplot_runner)

```

Her sørger vi for, at hvis skriptet udføres, så kaldes main klassen i benchmark med `inpluko_plot()` og `gnuplot_runner`.

udførelse

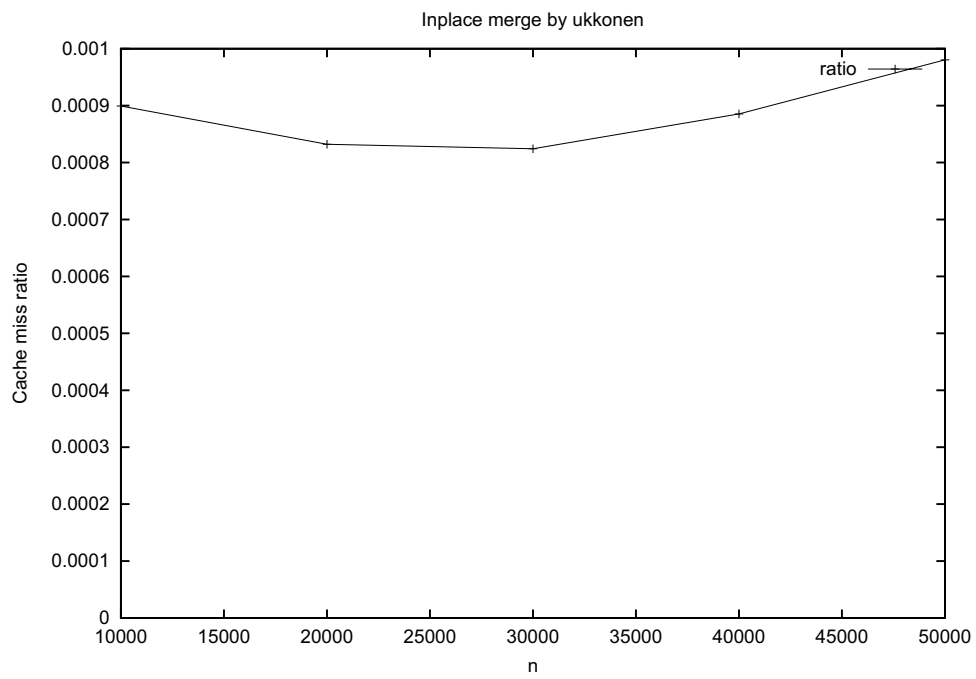
De to andre benchmarkfiler (`inpluko_time.py` og `inpluko_func.py`) kan ses på www.cphstl.dk. `inpluko_cache.py` køres og man får følgende output.

```

skade test > ./inpluko_cache.py
.....
=====
Wrote a gnuplot script in file: /net/urd/home/disk17/chrulle/jyrki/Tool/Benchmark/test/plot.212850072442.gp
Wrote a plot in file: /net/urd/home/disk17/chrulle/jyrki/Tool/Benchmark/test/plot.212850072442.ps
=====
Ran 5 benchmark cases in 82.428 s
OK
skade test >

```

Det færdige plot ligger nu klart i ps format:



Kapitel 4

Benchmark-skema generatorsiden

I mange tilfælde vil den ekstra funktionalitet, som et fuldt programmeringssprog tilbyder, slet ikke være nødvendig. I så fald kan det betyde, at folk ikke gider bruge værktøjet til at lave benchmarks, og ideen med at standardisere benchmarkene går tabt. Vi har derfor valgt at lave en hjemmeside, hvor man udfylder et simpelt skema, som så genererer det tilsvarende skema i Python. Det er vores hensigt, at denne side så kan bruges, når man blot skal lave en simpel benchmark, hvis man ikke har forstand på Python eller blot vil have et udgangspunkt for en mere kompliceret benchmark.

4.1 Program beskrivelse

Generatorsiden er skrevet i PHP (se [2]) og er egentlig ganske enkel. En skjult variabel bruges til at afgøre, om input siden eller siden med det genererede skema skal vises. Er den skjulte variabel ikke sat, vises input siden, hvorefter brugeren udfylder skemaet og indsender det ved at trykke på `generate!`. Dette sætter bla. den skjulte variabel, som fortæller scriptet, at siden med det genererede skema skal vises. Det genererede skema gemmes desuden med små modifikationer i filen `benchmark_form.py`, som via et link kan downloades af brugeren. Da samtlige genererede skemaer gemmes i samme fil, som overskrives hver gang, er det muligt for en bruger at få en anden brugers skema. Dette er dog kun muligt, hvis en bruger genererer et nyt skema inden, en anden bruger har fået gemt sit. Da vi ikke forudser, at siden bliver besøgt særligt meget, har vi ikke forsøgt at undgå dette problem. Man kunne dog gemme skemaerne i unikke filer og notere i en bogholderfil hvilke filer, der er blevet oprettet, og hvornår dette er sket. Så kunne man lave garbage-collection, hver gang scriptet kører ved at slette oprettede filer, der er mere end f.eks. 1 time gamle.

4.2 Bruger vejledning

Generatorsiden kan tilgås direkte på følgende URL:
<http://www.cphstl.dk/WWW/generateform.php>

eller ved navigation fra CPHSTL's hjemmeside <http://www.cphstl.dk> (klik på benchmark i venstre side osv.). Når først generatorsiden er tilgængelig mødes brugeren med en input del, som er opdelt i følgende 8 sektioner:

1. General input.
2. Remote execution.
3. Compiler settings.
4. Gnuplot settings.
5. Time settings.
6. Memory measurement.
7. Function settings.
8. Generate form.

4.2.1 General input

Den første sektion skal altid fyldes ud uanset hvilken type benchmark, der ønskes genereret. Sker dette ikke, får brugeren besked herom, når skemaet i Python skal genereres. I denne sektion angives en benchmark titel, som generatoren bruger til at navngive de generede klasser og metoder i Python. Denne variabel har altså ikke noget med selve benchmarken at gøre, men er med udelukkende af praktiske årsager. Dernæst angives navnet på den konstruktor, som benchmark programmet skal kalde. Bemærk at kun navnet skal angives og ikke hele kaldet til konstruktoren. Generatorsiden understøtter kun konstruktorer med en variabel og fastsatte templatevariable, f.eks. `spin<int>(n)`, og altså ikke `spin<std::less>(n,x)`. Skal en sådan funktion benchmarkes, må man selv tilrette skemaet til sin endelige form. Generatoren vil herefter kalde konstruktoren med hver af værdierne i måleintervallet en efter en. Dette måleinterval angives også i denne sektion. Det skal dog bemærkes, at måleintervallet er lukket i begge ender, så ønskes f.eks. kun en enkelt måling, skal det samme tal indtastes som start- og slutværdi. I driver feltet angives hvilken type målinger, der skal foretages. Der er følgende valgmuligheder:

1. Time - vælges hvis udførselstiden af ens program ønskes.(5)
2. Memory - vælges hvis målinger vedrørende hukommelsen ønskes.(6)
3. Branch - vælges hvis branch misprediction forholdet ønskes. (6)
4. Function - vælges hvis funktions statistik ønskes.(7)

Alt efter hvilken driver, der vælges, skal enten sektion 5, 6 eller 7 udfyldes. Tallet i parentes i ovenstående liste angiver hvilken sektion, der skal udfyldes. Sidst men ikke mindst skal der i denne sektion angives, om resultatet af benchmarken skal udskrives til skærmen, eller om værktøjet skal generere et gnuplot. Dette gøres i runner feltet.

4.2.2 Remote execution

Denne sektion skal kun udfyldes, hvis udførslen af benchmarken skal ske på en fjernmaskine. Først skal navnet på fjernmaskinen angives. Er det ikke muligt at forbinde direkte til fjernmaskinen, er der mulighed for at angive en gatewaymaskine, som der forbindes igennem. Herefter skal det angives, hvilken protokol benchmark værktøjet skal anvende til at forbinde til fjernmaskinen og en eventuel gatewaymaskine. Bemærk at det ikke er muligt at bruge forskellige protokoller til at forbinde til henholdsvis fjernmaskinen og gatewaymaskinen. Til sidst skal det angives, hvilken protokol værktøjet skal anvende, for at kopiere nødvendige filer til fjernmaskinen. Bemærk denne kopiering foregår gennem en eventuel gatewaymaskine, hvorfor valget af protokol her også påvirker hvilken protokol, der anvendes til at kopiere filer til gatewaymaskinen. Vær opmærksom på, at det kræves, at der ikke skal angives kodeord ved indlogging (se evt. hvordan i bilag B).

4.2.3 Compiler settings

I denne sektion udfyldes forskellige oversætter indstillinger. Den første af disse er hvilken oversætter, som værktøjet skal bruge. Hvis kaldet til oversætteren kræver en sti til oversætteren, skal denne medtages i dette felt. I tilfældet af fjernudførelse, skal det desuden bemærkes, at oversætteren skal være installeret på fjernmaskinen, ligesom en eventuel sti skal passe til fjernmaskinen. Udfyldes dette felt ikke, bruges `g++`. Dernæst er der mulighed for at give en semikolon separeret liste af options/switches med til oversætteren. Følgende options medtages automatisk: `-Wall -W -pedantic -ansi`. Visse optimeringer som f.eks. `-O3` giver problemer, når `function` driveren er valgt i sektion 1. Det sidste felt giver mulighed for at give en semikolon separeret liste af filer, som skal inkluderes i benchmarken for at sikre korrekt afvikling. Som et minimum skal filen med det C++ program, som skal benchmarkes, inkluderes. I tilfældet af fjernudførelse bliver disse filer automatisk kopieret vha. den i sektion 2 valgte protokol til fjernmaskinen.

4.2.4 Gnuplot settings

Denne sektion skal kun udfyldes, hvis `gnuplot` er valgt som runner i sektion 1. Det første felt i denne sektion giver mulighed for at angive en titel til det genererede gnuplot. Det andet og sidste felt giver mulighed for angive en titel til den tegnede kurve. Dette kan være en hjælp, hvis man senere vil tilføje flere kurver, da det således gør det muligt at identificere de enkelte kurver.

4.2.5 Time settings

Denne sektion skal kun udfyldes hvis `time` er valgt som driver i sektion 1. I første felt skal det angives, hvilken tidsenhed tidsmålingen skal angives i. Dernæst skal det angives, hvorvidt ens functor indeholder en dual funktion. I det tilfælde, at en dual funktion eksisterer, vil resultatet af tidsmålingen være udførselstiden for primal funktionen fratrukket

udførelstiden for dual funktionen.

4.2.6 Memory measurement

Denne sektion skal kun udfyldes, hvis **memory** eller **branch** er valgt som driver i sektion 1. Hvis **memory** er valgt som driver, skal der i det første felt vælges, hvilken type måling der ønskes. Der er følgende valgmuligheder:

- Minor page faults - måler antallet af sidefejl, som IKKE kræver I/O (disk adgang).
- Major page faults - måler antallet af sidefejl, som kræver I/O.
- Cache miss - måler antallet af depotkiks på level 1 datadepotet.
- Cache miss ratio - måler forholdet mellem depotkiks og depotadgange på level 1 datadepotet.
- Cache access - måler antallet af adgange til level 1 datadepotet.

I det andet felt skal det angives, hvorledes ovenstående måling skal foretages, hvis **memory** er valgt som driver. Der er følgende muligheder:

- single run - kører programmet en gang og returnerer resultatet af denne kørsel.
- quad run - kører programmet fire gange og returnerer gennemsnittet af disse fire kørsler.
- warmed up - kører programmet en gang uden at måle og derefter en gang, hvor der måles, hvorefter resultatet af den sidste kørsel returneres.

I det sidste felt skal stien til PAPI angives, hvis **memory** eller **branch** er valgt som driver. I tilfældet af fjernudførelse skal denne sti være gyldig for fjernmaskinen. I skrivende stund er denne sti `/usr/local/papi` for værten `cphst1.projektlab.diku.dk`, som er en maskine på DIKU, der tillader PAPI målinger.

4.2.7 Function settings

Denne sektion skal kun udfyldes, hvis **function** er valgt som driver i sektion 1. Navnet på den funktion, der ønskes målt, skal angives i det første felt i denne sektion. Der er desværre kun mulighed for at angive en funktion. Til sidst skal det angives, hvilken type måling der ønskes. Der er følgende muligheder:

- no. of calls - måler antallet af gange den pågældende funktion bliver kaldt.
- relative cputime - måler andelen af den samlede udførelstid, der foregår i den pågældende funktion.

4.2.8 Generate form

Nu er der kun tilbage at generere skemaet. Vælg **generate!** for at gøre dette eller vælg **reset!** for at nulstille skemaet og starte forfra. I tilfældet af mangelfuld udfyldning af skemaet bliver brugeren gjort opmærksom på dette og får mulighed for at gå tilbage og rette fejlen. Hvis skemaet er korrekt udfyldt, genereres det tilsvarende skema i Python og vises for brugeren. Der er desuden mulighed for at gemme skemaet som en fil ved at højre klikke på linket lige under det generede skema og vælge gem som. For at køre den gemte fil direkte fra en kommando prompt, er det nødvendigt at gøre filen eksekverbar. Dette gøres med følgende kommando:

```
#prompt#: chmod u+x filnavnet_på_dit_skema.
```

Litteratur

- [1] Python dokumentation på www.python.org.
- [2] PHP dokumentation på www.php.net.
- [3] UML Distilled, Martin Fowler et al., Addison-Wesley 1998.
- [4] Papi dokumentationen på <http://icl.cs.utk.edu/projects/papi/>.
- [5] The Cederqvist på www.cvshome.org

Bilag A

PAPI

PAPI står for Performance Application Programming Interface. Det er en grænseflade, der giver adgang til de ydelsestællere, der findes i moderne processorer. Denne grænseflade er ens, ligegyldigt hvilken processor type man bruger (x86, Sparc osv.). Grænsefladen gør det således muligt for værktøjet at måle f.eks. antallet af depotkiks uden at kende procesortypen.

Et krav for at PAPI virker er, at der er installeret en kerne patch. Denne patch tilføjer nødvendige rutiner, som sørger for adgang til de specialregistre i processoren, hvori ydelsestællerne befinder sig. Disse registre er det ikke muligt at få adgang til fra user space. Når denne patch er installeret, skal PAPI-tællekommandoerne indsættes i den kode, der ønskes målt, som herefter skal linkes med papi-biblioteket.

Yderligere information om PAPI findes i [4].

Bilag B

SSh keys

Når benchmarkværktøjet benyttes til at køre på fjernmaskiner, er det nødvendigt, at det er muligt at logge ind på fjernmaskinen uden at benytte kodeord. Er dette ikke muligt, vil værktøjet spørge om kodeordet, hver gang en SSh-kommando køres, og dette kan være mange gange i løbet af en kørsel. Desuden strandede kodeordsforespørgsler på gatewaymaskinen, når en sådan benyttes. Det er således umuligt at køre benchmarks over en gateway-maskine, hvis det ikke er muligt at logge ind uden brug af kodeord. Det er derfor vigtigt, at nedestående er udført på alle de maskiner, som det ønskes at logge ind på eller igennem. Hvis SSh er valgt som forbindelsesprotokol, kan private/public keys benyttes. Ideen er at generere et sæt af DSA nøgler med følgende kommando på maskinen, det ønskes at logge ind fra.

```
#prompt#: ssh-keygen -t dsa
```

Derefter tilføjes den offentlige nøgle `id_dsa.pub` til `.ssh/authorized_keys2` på maskinen, det ønskes at logge ind på. Forbindelser foregår herefter således: når en SSh-forbindelse oprettes, krypterer værtsmaskinen en besked med den offentlige nøgle, som kun kan afkrypteres med den tilsvarende private nøgle, og sender den til maskinen, der forbindes fra. Denne besked sendes tilbage i afkrypteret tilstand, hvorved ens identitet er blevet bevist og adgang opnås uden at skulle indtaste et kodeord.

Bilag C

Parametre

result:

variabel	beskrivelse	forvalg
answers	Liste af tupler: (case,resultatet af udførslen)	[]
errors	Liste af tupler: (case,fejl opstået under udførsel)	[]
should_abort	Boolsk: angiver om en kørsel skal afbrydes	False

_writeln_decorator:

variabel	beskrivelse	forvalg
stream	Strømmen af tegn som skal dekorerer.	Konstruktor variabelen

text_runner:

variabel	beskrivelse	forvalg
stream	output strømmen der udskrives til.	__writeln_decorator(std.err)
verbosity	hvor meget status skal der skrives ud.	1

case:

variabel	beskrivelse	forvalg
driver_file	Streng: Filnavnet på driverfilen	None
counted_function	Streng: Funktionen, der skal profileres.	None
driver_output	Streng: Resultatet af udførslen	' '
papitype	Streng: Hvilken type måling skal foretages. Mulighederne er: 'single', 'quad' og 'warmedup'	None
constructor_call	Streng: Konstruktør kaldet	None
id	Streng: unikt ID-nummer for benchmark tilfældet	str(random.random())[2:0]
title	Streng: benchmarktilfældets titel	' '
working_directory	Streng: Det nuværende arbejdsbibliotek	arbejdsbiblioteket
include_files	Liste af filer på lokalmaskinen, som skal inkluderes i driverfilen, samt kopieres til fjernmaskinen ved fjernudførelse	[]
copy_include_files	Intern liste: Holder styr på hvilke filer, der skal kopieres til fjernmaskinen. Ændring i indholdet af denne liste er på eget ansvar.	[]
include_paths	Liste af stier på lokalmaskinen, som oversætteren skal kigge i for at finde nødvendige filer.	arbejdsbibliotek og stier i shellvariablen PYTHONPATH
pythonpath	Liste af indholdet fra shellvariablen PYTHONPATH	
remote_include_paths	Liste af stier på fjernmaskinen, som oversætteren skal kigge i for at finde nødvendige filer	[]
papi_path	Streng indeholdende stien, hvor PAPI er installeret	None
include_statements	Liste af erklæringer, som skal inkluderes i driveren.	None
computer	Streng: navnet på maskinen hvor benchmarktilfældet skal udføres	uname()[1]
gateway	Streng: navnet på en gatewaymaskine	None
connection_protocol	Streng: Navnet på forbindelsesprotokollen	"rsh"
transfer_protocol	Streng: Navnet på overførselsprotokollen	"rcp"
compiler	Streng: kaldet til oversætteren.	"g++"
compiler_options	Liste af tilvalg til oversætteren.	["-Wall", "-W", "-pedantic", "-ansi"]
compile_command	Streng: Kaldet til oversætteren. Er denne None, genereres et kald automatisk.	None
execute_command	Streng: kaldet til at køre driveren. Er denne None, genereres et kald automatisk.	None
dual_exists	Boolsk: eksisterer dualfunktionen. Kun relevant i forbindelse med tidsmåling. Er denne None undersøger værktøjet selv om dualfunktionen eksisterer.	None
remote_benchmark_dir	Streng: navnet på biblioteket på fjernmaskinen, hvorfra benchmarken skal køres.	"./CPHSTL-Benchmarking-"+ id + "/"
remote_tidy_up	Boolsk: hvorvidt benchmarktilfældet skal rydde op på fjernmaskinen eller om suite-klassen tager sig af det.	1
filenames_translated	Intern, hvorvidt stier i include_paths listen er blevet oversat til fjernmaskinen. Ændring af denne variabel er på eget ansvar.	0
time_unit	Streng: tidsenheden, som tidsmålinger skal foretages i.	"s"
time_factors	Ordbog: der kan oversætte fra time.unit til en faktor i forhold til sekunder.	oversættelser for ns, micros, ms, s, min, h, day.

suite:

variabel	beskrivelse	forvalg
title	Streng: titel på suiten.	klassens navn
id	Suitens unikke id, der bruges til navngivning af filer.	random.random()[2]
benchmarks	Liste af benchmarks eller suites.	[]

curve_suite:

variabel	beskrivelse	forvalg
style	gnuplot	"linespoint"
linetype		None
linewidth	definitioner	None
pointtype	for	None
pointsize	kurven	None

plot_suite:

variabel	beskrivelse	forvalg
output	Streng: Det tegnede plots filnavn.	"plot." + id + ".ps"
terminal:	gnuplot	'postscript landscape noenhanced monochrome dashed defaultplex'
key:	definitioner	'right top Right noreverse samples 4 spacing 1.25 title '
style:	bruges i gnuplot_commands	'linespoints'
gnuplot_commands	Streng: Headeren til det gnuplotscript som bliver kørt i gnuplot result.	"set encoding iso_8859_1 set output "%(output)s" set terminal %(terminal)s set key %(key)s set title '%(title)s' set data style %(style)s "%self.__dict__"

text_result:

variabel	beskrivelse	forvalg
stream	Strøm som status beskeder skal udskrives til.	
verbosity	en variabel der beskriver hvor meget der skal skrives ud.	
show_all	skal alle status beskeder skrives ud.	(verbosity > 1)
dots	Boolsk: skal der udskrives prikker.	(verbosity == 1)
previous	bruges så der ikke udskrives fejl for den samme case flere gange.	None

main:

variabel	beskrivelse	forvalg
verbosity	hvor meget status skal der skrives ud.	1
program	det kørte programs navn.	os.path.basename(argv[0])
task	den opgave der skal køres.	None
runner	Den runner der skal benyttes.	text_runner