



Univerza v Ljubljani  
Fakulteta *za računalništvo*  
*in informatiko*

# **Benchmarking (SPARO)**

## Sistemska programska oprema

Martin Resman, 3.letnik UNI

27.12.2019, Ljubljana

Profesor Tomaž Dobravec

## Kazalo

Benchmarking .....	1
1. Uvod .....	3
2. Kaj je benchmarking.....	4
2.1 Splošno .....	4
2.2 Ogrodje.....	5
2.3 Izzivi in problemi .....	8
3. Standardizacija in sprejetje .....	9
4. Tipi in funkcionalnost benchmarkov .....	11
5. V ozadju znanih benchmarkov .....	13
6. DYI benchmark .....	15
7. Zaključek .....	17
8. Viri in literatura .....	18

## 1. Uvod

Naslov seminarske naloge sem si izbral zato, ker sem sicer seznanjen z benchmarkingom in benchmark aplikacijami, vendar nisem bil seznanjen s tem, kako sistem deluje v ozadju. Kaj se zgodi, ko kliknemo »start« v aplikaciji? Kaj se na sistemu testira in pod kakšnimi pogoji? Zanimalo me je tudi, kako se zagotovi standardizacija in pravičnost med različnimi sistemi. Na vsa ta vprašanja sem si v sklopu te seminarske naloge poskušal čim bolje odgovoriti oziroma pojasniti.

Beseda »benchmark« je prevzeta tujka, vendar sem si za svojo seminarsko nalogo skonstruiral slovenski izraz »standardizirana primerjalna analiza računalniške opreme« ali SPARO na kratko, saj menim, da nam, računalničarjem, ta izraz pove veliko več.

## 2. Kaj je benchmarking (v nadaljevanju SPARO)

### 2.1 Splošno

SPARO je dejanje izvršitve računalniškega programa, ali drugih operacij pod določenimi pogoji, z namenom da bi se ocenilo relativno zmogljivost objekta. Običajno se izvrši več standardnih testov in preizkušenj.

Zaradi napredka in razvoja v računalništvu na področju računalniške arhitekture in zaradi vedno bolj kompleksnih sistemov se je primerjava sistemov otežila. Težko je presoditi kateri sistem je boljši samo s pogledom specifikacij sistema. Zato so se razvili testi, ki nam omogočajo, da primerjamo različne sisteme z različnimi arhitekturami. S temi testi nas konec koncev niti ne zanima npr. ura CPU, saj nam testi nudijo način merjenja zmogljivosti sistema v pravem svetu ne da bi se ukvarjali s primerjavo specifikacij.

Različni benchmark programi uporabljajo različne teste za različna področja sistema (CPU, GPU, RAM, itd), zato z uporabo večih takih programov lahko pridobimo »večjo sliko« zmogljivosti našega sistema.

Benchmarki so narejeni, da oponašajo/posnemajo podrobno obremenitev na komponenti ali sistemu. Sintetični benchmarki to naredijo s posebej narejenimi programi, ki obremenijo komponento. Aplikacijski benchmarki pa izvršijo/zaženejo realistične programe na sistemu. Aplikacijski običajno podajo boljše meritve resničnih zmogljivosti sistema, vendar so pa sintetični benchmarki zelo uporabni za testiranje individualnih komponent.

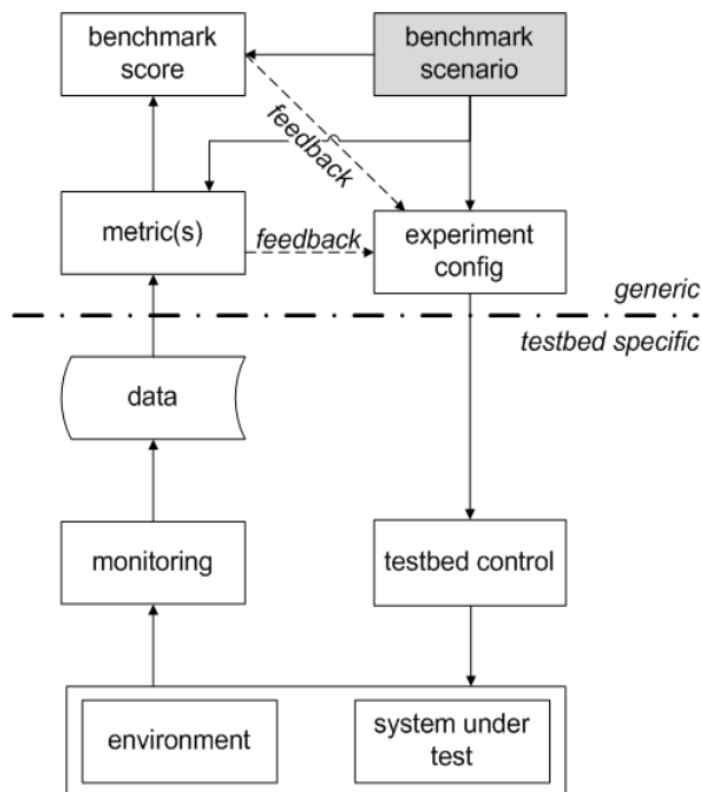
Rezultat benchmarka je običajno podan v obliki nekih točk (ni pa nujno, na primer benchmark za GPU je lahko povprečni FPS, ki ga sistem doseže pri uprizarjanju/risanju raznih objektov različnih kompleksnosti in resolucije na ekranu). Same točke povejo zelo malo, ko pa enkrat dva sistema stestiramo z istim testom, lahko primerjamo dosežene točke in začnemo soditi kateri sistem nudi boljšo zmogljivost.

## 2.2 Ogradje

Benchmark vsebuje seznam specifikacij, ki so potrebni za merjenje zmogljivosti testiranega sistema. Te specifikacije so :

- scenarij – podroben opis priprave eksperimenta, ki je potrebna za izpeljavo benchmarka (lahko npr. vsebuje parametre za konfiguracijo protokola, omrežno topologijo),
- kriterij evalvacije – opisuje, kaj je končni rezultat benchmarka (npr. energijska učinkovitost, robustnost)
- metrika – kvantitativna meritev specifične kvalitete sistema
- točke

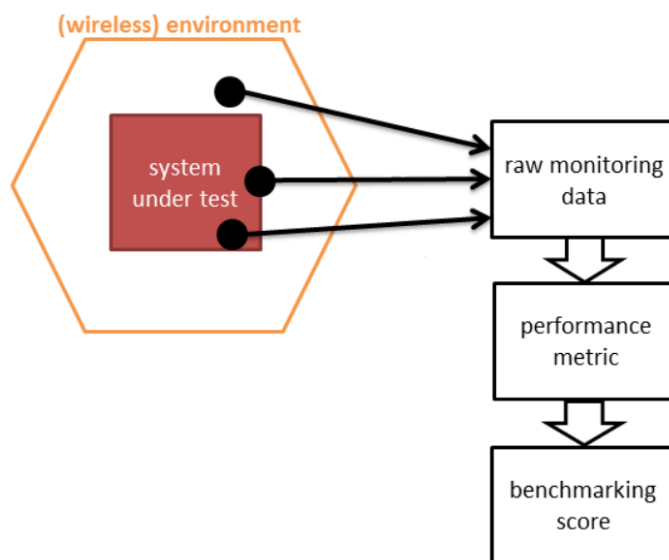
Poleg same definicije benchmarka, je zelo pomembna tudi dejanska izvršitev benchmarka v eksperimentno-prijaznem okolju (spreminjanje parametrov), delitev benchmarkov z drugimi in narediti obstojno oziroma trajno ogradje.



Komponente pod črtkano črto predstavljajo komponente odvisne od izvajalca benchmarka, torej testiranca, med tem so pa komponente nad črtkano črto poljubne (odvisne od samega testa). Vsak tester bo implementiral SPARO scenarij na svoj način s pomočjo testnega kontrolerja (testbed control). Ta konfigurira testni sistem ter tudi okolje (večinoma v brezžičnih omrežjih). Primer: tester/eksperimentator želi testirati zanesljivost brezžičnega

sistema za konec-do-konec izgubo paketov. Za to bi rad uporabil nekaj brezžičnih vozlišč, ki bi generirali in pošiljali interferenco. Ta vozlišča so del okolja, namesto del sistema, saj se njih ne bo evaluiralo.

Komponenta za nadzorovanje med eksperimentom zbira podatke (od sistema in tudi okolja). Različni testerji (testbed) zbirajo podatke na različne načine in v različnih formatih (podatkovne baze, CSV datoteke, ..).



V naslednjem koraku, se podatki s pomočjo specifikacij metrik (ki jih definira SPARO) pretvorijo. Tako se iz podatkov, ki nima veze njihov izvor ali tip, kako so shranjeni v testbedu, dobi nedvoumne metrike. Nedvoumne metrike je možno dobiti/podati le z točno določenimi in sprejetimi definicijami, kako se metrike določijo iz podatkov, in tako da je zagotovljeno, da so bili podatki izmerjeni v skladu standardiziranih metodologij.

Dobljene metrike lahko eksperimentator kar direktno uporabi, da oceni zmogljivost sistema. Vendar je pa običajno še ena stopnja abstrakcije. Ena ali več metrik se uporabi in kombinira, da se dobi končne izide/score. Vsak ta izid nam potem lahko pomaga ugotoviti zmogljivost sistema. Način določitve kombiniranja metrik je seveda zapleten, ampak se to ponavadi splača, saj nam to omogoča, da bolje razumemo zmogljivost sistema in jo analiziramo.

Če se lotimo izvesti eksperiment samo enkrat (torej s fiksnimi konfiguracijskimi parametri za testni sistem), nam benchmark izidi ne povejo nič novega, kar nam ne bi že metrike povedale. Pravzaprav nam lahko kakšne stvari skrijejo. Vendar, za temeljito analizo zmogljivosti, je potrebno izvesti mnogo (sto, tisoč, ..) testov. V tem primeru bi bilo pa zelo zamudno gledati in analizirati vsako metriko posebi. Če je benchmark izid definiran tako, da kombinira vse pomembne in relevantne metrike z uporabo kriterja evalvacije, nam končni izid zelo hitro pove kateri eksperimenti so odvisni eden od drugega. Te izidi se potem lahko uporabijo, da se hitro in pravično primerja zmogljivost sistema pod različnimi pogoji in v različnih okoljih, kasneje pa tudi med drugimi sistemi.

Z dobro definiranimi končnimi izidi benchmarkov (torej kaj merimo), je uporaba teh benchmarkov in primerjava rezultatov možna za ne-strokovnjake, saj ni potrebno nobeno poglobljeno znanje delovanja metrik. Te izide se potem lahko tudi uporabi, da se rekonfigurira same eksperimente in s tem pohitri samo testiranje sistema (se lahko avtomatizira). Programi lahko na podlagi izidov ugotovijo vpliv določene konfiguracije eksperimenta na zmogljivost in na primer predlagajo novo konfiguracijo (ponovni test, le z drugačnim parametrom), še enkrat ocenijo izid in na koncu sami optimizirajo določen sistem.

Uporabnik bi eventuelno lahko definirali razne parametre v sistemu, ki naj se spreminjajo in potem pustil ogrodju benchmarka in testbed, da avtonomno izvedeta teste in se odzoveta na njih in na koncu uporabniku prikazala lahko razumljiv pregled zmogljivosti njegovega sistema.

## 2.3 Izzivi in problemi

Da se dobi uporabne podatke je običajno potrebnih več iteracij med testiranjem. Interpretacija dobljenih podatkov je tudi zelo težka. Drugi izzivi, ki se pri benchmarkih pojavijo so:

- nekateri proizvajalci/prodajalci strojne opreme nastavijo/pripravijo svojo opremo, da bo bolje delovala na standardnih benchmarkih
- benchmarkov je toliko, da proizvajalcu ni težko najti takega, ki bo njegov izdelek pokazala v najboljši luči (bench-marketing)
- pojavili so se proizvajalci, ki goljufajo na SPARO, sisteme pripravijo tako, da se na testih bolje odrežejo, kakor v realnih situacijah. Primer tega je bilo v 1980tih, kjer so nekateri prevajalniki lahko odkrili posebno matematično operacijo, ki se je uporabila v splošno znanih SPARO s plavajočo vejico in jo zamenjala s hitrejšo a matematično enakovredno operacijo.
- veliko benchmarkov se osredotoči le na hitrost sistema in zanemari druge pomembne lastnosti sistema (kvaliteta storitve/sistema, »Total cost of ownership«, breme okolja – torej prostor, elektrika, hlajenje)
- pogosto se ne upošteva osnovnih principov znanstvenih metod (na primer: majhen vzorec, fiksne spremenljivke in ne možnost ponovitve rezultatov)
- zmogljivost veliko sistemov (predvsem strežnikov) se precej zmanjša nad 80% uporabe sistema. Benchmarki bi morali to upoštevati, vendar dostikrat proizvajalci publicirajo/objavijo bencharke za svoj sistem pri stalni 80% uporabi, kar pa ni realistično, saj velikokrat pride do skokov čez to mejo uporabe.
- zmogljivost za uporabnika ima lahko drugačen pomen kot ga SPARO pokaže. Uporabnike običajno zanima predvidljivost oziroma zanesljivost sistema. Testi pa pokažejo povprečne izide, ki so morda bolj za IT oddelek uporabne, namesto na primer največji odzivni čas storitve na sistemu.

Idealno naj bi bili SPARO le nadomestek za realne aplikacije, če aplikacija ni dosegljiva, je predraga ali pa pre težka namestiti na drug sistem.



### 3. Standardizacija in sprejetje

Dobri benchmarki delujejo na teh osnovnih principih:

- bistvenost/relevantnost – merijo naj bistvene oziroma pomembne sposobnosti sistema. Tudi, če je bil test perfekten in imamo veliko rezultatov, če nam to na koncu nič ne pomeni, je bilo vse za manj. Relevantnost je odvisna od same uporabe rezultatov: za nekatere scenarije so lahko zelo pomembni za nekatere druge pa ne. Razvijalec SPARO mora pri razvoju upoštevati na kakšen način in na katerih področjih se bo uporabljal in planirati SPARO tako, da bo tam relevanten. SPARO, ki so namenjeni za uporabo na specifičnem področju imajo omejeno uporabo (a nam povejo veliko), medtem ko so SPARO, ki so narejeni za uporabo na večjih področjih, manj pomembni.

- reprezentativnost – metrike naj akademski in industrijski svet sprejme in se z njimi strinja. SPARO običajno razvijajo več strokovnjakov oziroma podjetij in tako pride tudi do več kompromisov pri razvojnih ciljih. Čeprav te kompromisi utežijo postopek razvoja aplikacije, to povzroči da se kompromisi ustvarijo na tak način, da se več zainteresiranih strank/partij strinja in sprejme končni SPARO,

- pravičnost – vsak sistem se pravično primerja z drugim sistemom. SPARO potrebujejo različne SW in HW komponente, da nudijo primerno okolje za izvršitev. Pogosto je potrebno omejiti katere komponente se lahko uporabi. Nekatere omejitve so tehnične (program se lahko izvaja le na tem OS, sistem mora imeti dovolj diskov itd.). Nekatere omejitve so potrebne za zagotavljanje pravičnosti (mora se specificirati katere verzije katerega SW se uporabi za izvršitev benchmarka, saj nekatere verzije morda nimajo kakšne funkcionalnosti, ki je za pravo uporabo potrebna, na primer varnost, vendar se zaradi tega bolje in hitreje izvaja). Tukaj je potrebno uravnotežiti število omejitev, saj če jih je preveč, bodo morda bili pomembni rezultati razveljavljeni, če pa jih je premalo, se pa lahko »onesnaži« nabor objavljenih rezultatov in s tem onemogoči ali oteži primerjavo sistemov s primernimi rezultati,

- ponovljivost – rezultate testov se lahko ponovi s ponovnim testiranjem v enakem okolju. To vključuje ponovljivost med testi in ponovljivost dveh različnih testerjev (eksperimentator) da samostojno prideta do istih rezultatov. V idealnem svetu, bi bili izidi SPARO funkcija konfiguracije SW in HW, ker pa je to realni svet, je računalniški sistem preveč kompleksen in to povzroča variacije zmogljivosti aplikacije. Te variacije povzročajo na primer razvrščanje niti, dinamično prevajanje, fizična postavitve na disku, probleme z omrežjem ter sama interakcija uporabnika s sistemom. Ta problem se rešuje tako da se SPARO izvede za več časa in s tem dobi večji merilni vzorec,

- cenovno optimalno/učinkovito – SPARO naj bodo ekonomični,

- skalabilnost – teste naj se lahko izvede na enem ali večih procesorjih ali pa nitih. Ta princip je lahko kar obetaven, saj se od SPARO pričakuje, da dela na razno raznih sistemih, ki se kar precej razlikujejo po arhitekturi in razpoložljivih resorsih. Razvijalci SPARO morajo najti

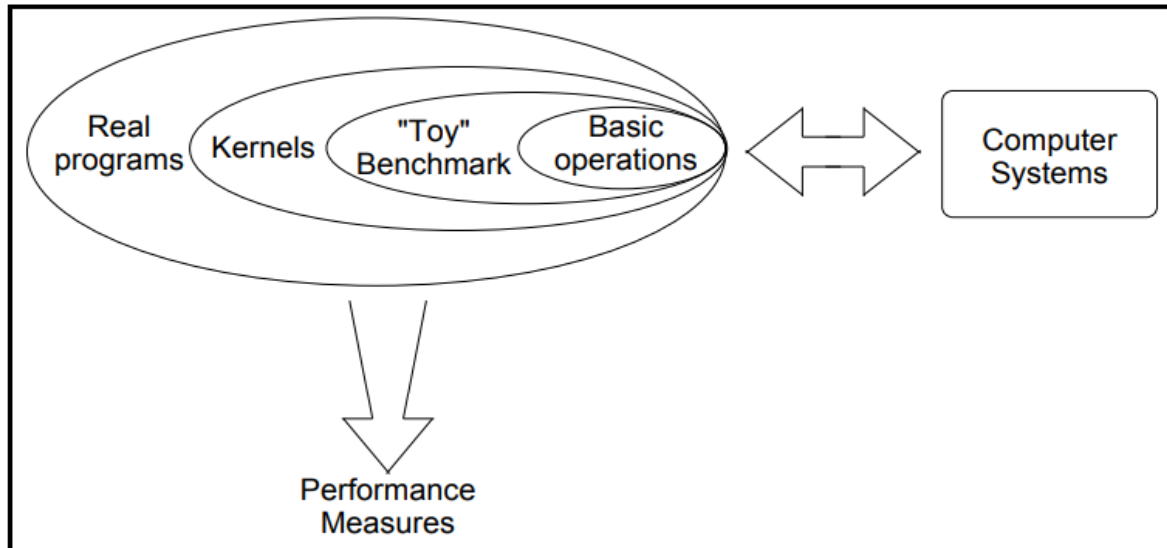
ravnotežje med tem, da bi SPARO naredili preveč skalabilen in tem, da se še obnaša kot prava aplikacija (ki pa pogosto imajo težave s skalabilnostjo),

- transparentnost – testne metrike naj bodo lahke za razumet,

- uporabnost – večina uporabnikov SPARO so tehnično sofisticirani zato ni velikega fokusa, da se naredi benchmark lahek za uporabo, kot je z navadno potrošniško aplikacijo. Vendar je nezahtevnost uporabe vseeno pomembna, saj s tem omogoča uporabniku, da se sam preverja. Ko je uporabnik sposoben sam preveriti svoje teste in obremenjenost, ima veliko večje zaupanje v to, da se SPARO s to obremenjenostjo pravilno izvršuje.

#### 4. Tipi in funkcionalnost benchmarkov

SPARO se v osnovi delijo na sintetične in aplikacijske, vendar to ni njihova edina delitev. Bolj poglobljena delitev je na primer sledeča:



V notranjem levelu so sintetični SPARO (»basic operations« na sliki), ki izvajajo le osnovne operacije, kot so na primer množenje in seštevanje. Primer teh je program Dhrystone in je namenjen merjenju hitrosti sistema pri izvajanju računanja s celimi števili (podobno Whetstone, vendar da vsebuje še realna števila).

Na drugi stopnji, so tako imenovani »igrače« SPARO, ki so majhni programi in izvajajo klasične uganke/izzive (Stolphi Hanoi, Sito Eratostela). Taki SPARO so manj uporabni in nam ne povejo dosti o zmogljivosti sistema v pravem svetu.

Kerneli so deli kode, izluščene iz pravih programov. Ta koda je bistvo tega SPARO, saj se večino računanja dogaja tukaj. Primera tega sta: Livermore Loops, ki je namenjen za paralelne računalnike, in Linpack, ki meri hitrost sistema pri računanu z realnimi števili. Ta tip SPARO se ukvarja s sposobnostjo, kako sistem izvršuje neke račune in običajno meri le zmogljivost procesorja.

Na zunanjem levelu so pa »realni« programi, oziroma SPARO, ki uporabljajo obremenitve vzete iz realnih programov (aplikacijski SPARO). Organizaciji SPEC in TPC razvijata znane SPARO takega tipa. Nekateri benchmarki v tej kategoriji uporabljajo komponente prevajalnika za jezik C, procesiranje besed, transakcije kreditnih kartic.

SPARO se lahko deli tudi po tem kateri del sistema ovrednotijo:

- Procesor – matematični testi s plavajočo vejico, ki testira procesorjevo sposobnost opraviti več osnovnih matematičnih operacij. Rezultati so običajno podani v časovni enoti (npr. milisekunde). Manjši je torej boljši. Kompresijski testi, ki testirajo kako hitro lahko procesor

stisne večje bloke podatkov brez izgub. Rezultati so lahko podani kot hitrost (KB/s ..) tako da večja številka je boljše.

- GPU – 2D grafični testi se osredotočijo na risanju, premikanju in skaliranju črt, pisav in elementov na uporabniškemu vmesniku. Meri se v slikah na sekundo (FPS). 3D grafični testi vsebujejo risane nekaj ali mnogo 3D objektov, večih kompleksnosti, stopnje podrobnosti, senc, AA, na zaslon, ter testirajo lahko tudi razne APIje (kot na primer DirectX in OpenGL).

- I/O (vhodne in izhodne naprave, npr. disk) – SPARCO za trdi disk se dostikrat osredotočijo na sekvenčne in naključne beritvene hitrosti (sekvenčne – datoteka je shranjena v enem kosu na disku, naključne – ko je datoteka shranjena na večih mestih na disku in je sistem prisiljen dostopati do teh mest).

- telefone – tudi za telefone obstajajo benchmarki. Večinoma lahko iste stvari testirajo kot se testira na PC ali prenosnikih, vendar imamo tudi kakšne dodatne teste (npr. za branje in pisanje na SD kartico)

- podatkovna baza – merijo zmogljivost in odzivni čas DBMS (sistemov za upravljanje s podatkovno bazo)

## 5. V ozadju znanih benchmarkov

- Livermore loops – za paralelne računalnike, originalno napisano v Fortranu. Sestavljen iz 24 »do while loopov«, vsaka zanka pa izvrši enega od 24 kernelov, nekatere se lahko vektorizira, nekatere ne. Uporablja se za merjenje aritmetične zmogljivosti računalnikov in njihovih prevajalnikov. Meri se zmogljivost HW in prav tako sposobnost SW, da vektorizira učinkovito kodo. Rezultate se dobi v MFLOPsih.

- LINPACK – meri kako hitro bo računalnik rešil gosti  $N \times N$  sistem linearnih enačb  $Ax=b$ . Izmerjena zmogljivost nam pove koliko 64-bitnih operacij s plavajočo vejico, običajno seštevanje in množenje, lahko računalnik opravi na sekundo (FLOPS). Linkpack 100 – rezultat se dobi z Gavsovo eliminacijo in delnim pivotiranjem, z  $\frac{2}{3}n^2 + 2n^2$  operacij s plavajočo vejico, kjer je  $n$  100 (dimenzija matrike  $A$ ). Za paralelne računalnike lahko uporabimo HPLinkpac oziroma njegovo implementacijo HPL, ki je napisana v C. HPL generira linearni sistem enačb reda  $n$  in ga reši z LU dekompozicijo. Tukaj se rabi še MPI.

- SPEC CPU – meri zmogljivost procesorja tako da meri izvajalni čas programov, kot so prevajalnik GCC, kemijski program gamess in vremenski program WRF. Vsi programi so enakovredni v ocenjevanju in končni rezultat se dobi s povprečjem. Za merjenje računanja s plavajočo vejico se uporabi 14 aplikacij napisanih v Fortranu ali pa C, za cela števila pa 12 aplikacij napisanih v C ali C++. Zmogljivost se primerja z nekim referenčnim sistemom. Rezultat je lahko podan tudi kako hitro sistem opravi vseh 12 nalog.

- Winstone – benchmark za merjenje zmogljivosti pri izvršitvi Windows programov. Benchmark vsebuje procesiranje besed (Word, Excel), podatkovne baze (Access, Paradox), PowerPoint in nove različice Winstona tudi Photoshop, Cad, Visual C++. Winstone s skriptami izvrši ukaze znotraj teh aplikacij in si zapomni, koliko časa je računalnik porabil za njih. Ko je vse stestiral, uporabi izmerjeni čas in izračuna nek končni izid (glede na referenčni računalnik).

- Threadmark – 32-bit aplikacija, ki meri kako hitro se podatki z in na diski prenašajo ter zasedenost CPU. Naredi veliko merjenj enojnih in večnitnih zahtevkov čez večje število blokov. Merjenja nam dajo 128 individualnih podatkovnih točk, ki se otežijo in nato se vzame povprečje. Dobimo transfer-rate in zasedenost CPU. Zasedenost se dobi tako, da se prvič dobi neko referenco za CPU, ki se potem uporabi za izračuna zasedenosti CPU pri izvajanju IO operacij. CPU referenco izmeri tako, da CPU računa praštevila, ko se nič drugega na sistemu ne izvršuje. Število računov na neko časovno enoto nam pove kaj pomeni 100% zasedenost CPUja. Ko dobimo referenco, se začnejo izvajati IO operacije, vendar se računanje praštevil nadaljuje (ampak le v eni niti, ki nima prioritete – idle priority, torej se bo ta izvajala, ko se nič drugega ne izvaja in s tem ne bo vplivalo na IO operacije/merjenje). Po končanem merjenju se število končanih praštevil operacij skalira in primerja z referenco. Testne datoteke se na začetku ustvarijo, da se ne bi brez njih dostopalo kar do predpomnilnika. Vsaka datoteka je dvakrat večja od RAM, da se dodatno prepreči predpomnjenje. Do vsake datoteke dostopa samo ena nit. Merjenje se konča, ko katera koli nit pride do konca datoteke. Hitrost prenosa se izmeri na podlagi števila prenesenih bytov in času prenosa.

- Heaven Benchmark – preveri stabilnost sistema (GPU, PSU, hlajenje), boljša verzija tudi stres testa GPU. Rendera okolje in sceno, ki je precej zahtevna in ves čas meri FPS ter druge stvari.
- 3DMark – meri zmogljivost računalnika pri risanju 3D grafike in sposobnost CPUja procesirati intenzivno obremenitev. Po merjenju vzame rezultata za obe stvari in vrne nek skupni rezultat, ki se uporabi za primerjanje.
- CineBench – benchmark, ki uporabi vse niti/jedra procesorja, da izriše neko sliko.
- FutureMark
- PiFast, Super PI – programa, ki izračunata PI na zelo veliko decimalnih mest.
- Whetstone, Dhrystone
- Fhourstones – celo številski benchmark, ki na učinkovit način izračuna in reši igro Connect-4. Napisan v Javi ali C.

## 6. DIY benchmark

Za veliko starejih benchmarkov se sploh ne potrebuje posebnih programih ampak se lahko sam koda skopira in se jo prevede ter požene. Edina stvar, ki se more potem še naresti je meriti čas.

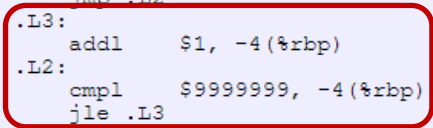
V C in assembly jeziki se da kar hitro narediti zelo preproste SPARO, eden takih je:

Noploop CPU benchmark – algoritm za merjenje hitrosti ure CPUja. Kot prvo v C napišemo program, ki šteje do 10 milijonov.

```
int
main() {
    int i;
    for (i = 0; i < 10000000; i++) { }
    return (0);
}
```

Potem program prevedemo v assembly (in uporabimo -O0, da je manj optimizacij). Najdemo našo zanko v kodi.

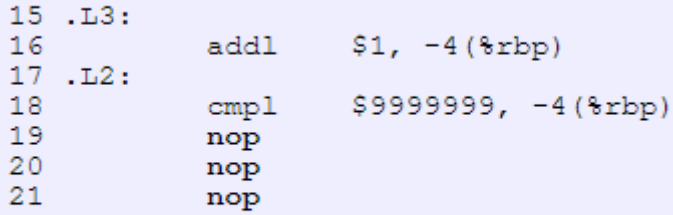
```
$ gcc -O0 -S noploop.c
$ cat -n noploop.s
[...]
```



```
12      .cfi_def_cfa_register 6
13      movl    $0, -4(%rbp)
14      jmp     .L2
15  .L3:
16      addl    $1, -4(%rbp)
17  .L2:
18      cmpl    $9999999, -4(%rbp)
19      jle     .L3
20      movl    $0, %eax
21      popq    %rbp
22      .cfi_def_cfa 7, 8
[...]
```

Na pomnilniški naslov, na katerega kaže rbp (minus -4) dodamo konstanto 1. To je i v naši zanki. Potem to vrednost primerjamo s konstanto 999 999 in če je manjše od tega se skoči nazaj na L3.

```
$ vi +'set nu' noploop.s
[...]
```



```
15  .L3:
16      addl    $1, -4(%rbp)
17  .L2:
18      cmpl    $9999999, -4(%rbp)
19      nop
20      nop
21      nop
[... etc, 2,000 nop's in total ...]
2018      nop
2019      jle     .L3
[...]
```

Pod ta compare dodamo veliko število NOP, mi bomo dodali 2000. NOP je velik 1 byte, tako da bo teh 2000 NOPov postalo kar 2000 bytov objektne kode. To bi računalnik moral brezproblema predpomniti in to bi moralo ostati na eni strani v pomnilniku.

Zdaj pa stvar še prevedemo v binary in jo izvedemo. Z ukazom time dobimo koliko časa se je naš program izvajal. Vzemimo »user« čas in recimo, da je ta bil pri nas okoli 1,628 sekunde. Program je izvedel 20 milijard NOP ukazov, vsakega v eni urini periodi. Torej lahko dobimo  $20 \text{ milijard urinih period} / 1,628 \text{ sekunde} = 12,285 \text{ milijard urinih period na sekundo}$ . To bi pomenilo, da bi naš procesor imel frekvenco 12,285 GHz. To je seveda zelo veliko in moramo še upoštevati, da moderni računalniki lahko izvedejo več ukazov na urino period (superskalarna arhitektura). Koliko jih lahko je odvisno od ukaza in samega CPU. Verjetno je za NOP število 3 ali 4. To bi pomenilo, da bi bila frekvenca našega procesorja enaka  $12,285/4 = 3,071 \text{ GHz}$ .

Tak SPARO je seveda trivialen v današnjih dneh, vendar je vseeno zanimivo kako hitro se lahko naredi program, ki »testira« procesor.



## 7. Zaključek

Ob izdelavi moje seminarske naloge sem ugotovil, da so "Benchmark testi" zelo uporabna zadeva, saj nam olajšajo primerjavo sistemov in pomagajo pri premišljenih odločitvah. Izkazalo se je, da je sama struktura okoli benchmarkov zelo kompleksna, saj program (benchmark) poleg samega testa vsebuje tudi funkcije, ki rezultate ovrednotijo. Sam razvoj programa je bil očitno zelo premišljen, saj se ni upoštevala oz. ni bila razvijana samo strojna in programska oprema, pač pa se je veliko vlagalo tudi v bodočo uporabniško izkušnjo. To so naredili z uporabo osnovnih principov pravičnosti, relevantnosti in transparentnosti. Takoj pa, ko se pojavi nek izid (score), se pojavi tudi tekmovalnost in s tem tudi osebe, ki želijo zadevo izrabiti v svoje namene na nezakonit način, kar je seveda potrebno prepoznati in preprečevati (z otežitvijo testov in bolj striktnimi pravili). Obstaja več vrst benchmarkov (sintetični in aplikacijski), vsaka pa meri in ovrednoti neki del našega sistema - po delih, z uporabo sintetičnih benchmarkov ali pa kar v celoti, z uporabo aplikacijskih benchmarkov. Za veliko starejših benchmarkov se lahko na internetu najde kodo in se jo sam prekopira, prevede in zažene, ni sploh potrebno prenesti aplikacije. To je zelo prikupno, saj lahko dobesedno pogledaš kaj se bo dogajalo v testu, vse ti je na pregled in morda lahko tudi kaj spremeniš.

## 8. Viri in literatura

- Benchmark (computing). 21. 11.2019. Dosegljivo na URL: [https://en.wikipedia.org/wiki/Benchmark\\_\(computing\)#Types\\_of\\_benchmark](https://en.wikipedia.org/wiki/Benchmark_(computing)#Types_of_benchmark),
- How Benchmarks Work and When You Should Pay Attention to Them. Eric Ravenscraft, 21.2.2017. Dosegljivo na URL: <https://lifehacker.com/how-benchmarks-work-and-when-you-should-pay-attention-t-1792579167>,
- PC Benchmark Tests: What Are They, And Do They Actually Matter? Andy Betts, 30.3.2015. Dosegljivo na URL: <https://www.makeuseof.com/tag/pc-benchmark-tests-what-are-they-and-do-they-actually-matter/>,
- Benchmarking computers and computer networks. Dosegljivo na URL: <http://www-sop.inria.fr/members/Thierry.Turletti/WP11.pdf>,
- Computer System Performance Analysis and Benchmarking. Marko Aho, Christopher Vinckier, 1998. Dosegljivo na URL: [http://www.cs.inf.ethz.ch/37-235/studentprojects/vinckier\\_aho.pdf](http://www.cs.inf.ethz.ch/37-235/studentprojects/vinckier_aho.pdf),
- Benchmarks and Performance Tests. Dosegljivo na URL: <http://catalogue.pearsoned.co.uk/samplechapter/0130659037.pdf>,
- How to Buld a Benchmark. Jokim von Kristowski, 2015. Dosegljivo na URL: [https://www.researchgate.net/publication/273133047\\_How\\_to\\_Build\\_a\\_Benchmark](https://www.researchgate.net/publication/273133047_How_to_Build_a_Benchmark).