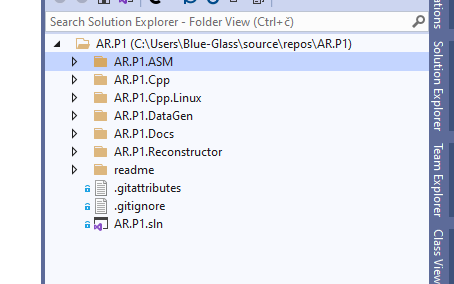
Dokumentacija

# Uvod

Rešenje ima za cilj implementaciju Fast Fourier Transform (FFT) algoritma. Konkretno, Cooley Tukey[[1]](#endnote-1) varijantu FFT.

Rešenje se sastoji od sledećih projekata, gde ime svakog sadrži prefiks „AR.P1“

1. ASM – NASM AMD x64 Linux implementacija  
   Kompajluje se pomoću GCC-a, koristi neke minimalne funkcije iz GNU C biblioteke i koristi Linux sistemske pozive.
2. Cpp – C++ implementacija, AMD x64 Windows
3. Cpp.Linux – C++ implementacija, AMD x64 Linux
4. DataGen - .NET CLI alat  
   Generator ulaznih .WAV fajlova za rad prethodnih implementacija
5. Docs – Dokumentacija rešenja
6. Reconstructor - .NET konzolna aplikacija  
   Za konverziju binarnog izlaza implementacija u čitljivi CSV fajl

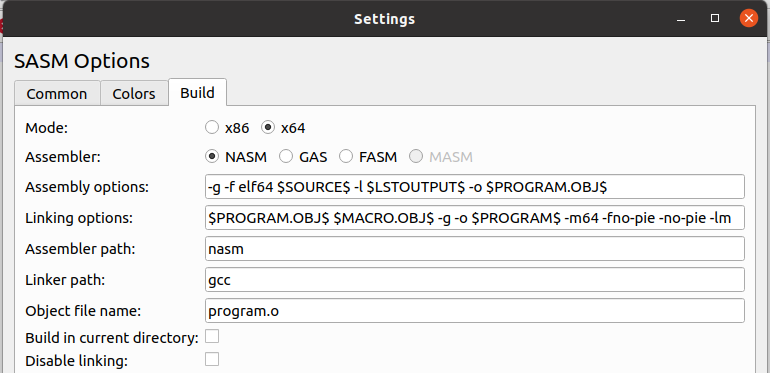


Prikaz 1

# Kompajlovanje

Svi projekti, sem ASM projekta, se mogu kompajlovati direktno iz MS Visual Studio alata.

Za kompajlovanje ASM projekta, bez ikakvih modifikacija, sugerisano je da se koristi SASM[[2]](#endnote-2) grafičko okruženje za kompajlovanje pomoću GCC kompajlera. Ispod je priloženo podešavanje koje je korišteno za kompajlovanje pri razvoju projekta.

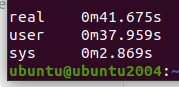


Prikaz 2

# Poređenje po brzini

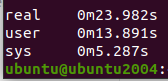
Obzirom na specifičnosti implementacije, poređenje C++ i assembly implementacija se vrši između ASM projekta i Cpp.Linux projekta (zbog istog kompajlera, OS podrške, ...).

Cpp.Linux  
Bez kompajlerskih optimizacija



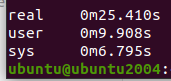
Prikaz 3

-Os optimizacija veličine koda



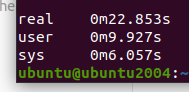
Prikaz 4

-O2 optimizacije



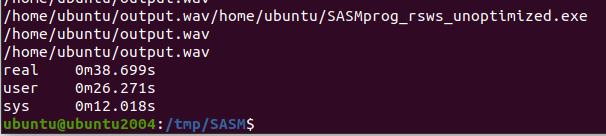
Prikaz 5

-O3 maksimalne optimizacije



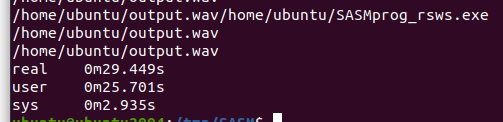
Prikaz 6

ASM  
Osnovna implementacija



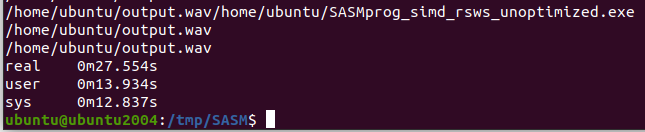
Prikaz 7

RSWS - Optimizacija konverzije ulaznog fajla



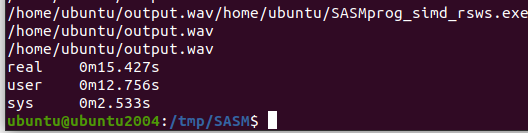
Prikaz 8

SIMD - SIMD optimizacije algoritma, bez optimizacija konverzije



Prikaz 9

SIMD RSWS - optimizacije algoritma, sa optimizacijama konverzije



Prikaz 10

Za dobavljanje podataka, korišten je *bash shell* i *time* CLI alat, upotrebom sledećih komandi:

time ( for i in {1..1000}; do ~/AR.P1.Cpp.Linux.out ~/output.wav; done )

time ( for i in {1..1000}; do ~/SASMprog\_simd\_rsws.exe ~/output.wav; done)

Sekvencijalno, programi obe implementacije su pozvani 1000 puta kako bi se efekti standardne devijacije i uticaja faktora vrituelne mašine (na kojoj su programi izvršavani) na vremena izvršavanja smanjili.

Ispod je dat tabelarni prikaz vremena izvršavanja, u sekundama, sa podelom ukupnog vremena izvršavanja (real) na vreme koje CPU provede u user mode-u (user) i vremenom koje provede u kernel modu (sys).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [s] | Cpp.Linux | | | | ASM | | | |
| No opts | -Os | -O2 | -O3 | No opts | RSWS | SIMD | SIMD, RSWS |
| real | 41.675 | 23.982 | 25.410 | 22.853 | 38.699 | 29.449 | 27.554 | 15.427 |
| user | 37.959 | 13.891 | 9.908 | 9.927 | 26.271 | 25.701 | 13.934 | 12.756 |
| sys | 2.869 | 5.287 | 6.795 | 6.057 | 12.018 | 2.935 | 12.837 | 2.533 |

Prikaz 11

Ispod je dat prikaz ubzranja rada pri različitim optimizacijama u okviru Cpp.Linux projekta.

|  |  |  |  |
| --- | --- | --- | --- |
| % = [vreme nivoa optimizacije] / [No opts] | -Os | -O2 | -O3 |
| real | 57.55 | 60.97 | 54.84 |
| user | 36.59 | 26.10 | 26.15 |
| sys | 184.28 | 236.84 | 211.12 |

Prikaz 12

Ispod je dat prikaz ubrzanja rada pri različitim optimizacijama u okviru ASM projekta.

|  |  |  |  |
| --- | --- | --- | --- |
| % = [vreme nivoa optimizacije] / [No opts] | RSWS | SIMD | SIMD, RSWS |
| real | 76.10 | 71.20 | 39.86 |
| user | 97.83 | 53.04 | 48.56 |
| sys | 24.42 | 106.81 | 21.08 |

Prikaz 13

Ispod je dat prikaz međusobnih poređenja, između Cpp.Linux i ASM projekata.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| % = [% ASM vreme] / [% Cpp.Linux vreme] | No opts vs No opts | RSWS vs -Os | SIMD vs -O2 | SIMD, RSWS vs -O3 |
| real | 92.86 | 122.80 | 108.44 | 67.51 |
| user | 69.21 | 185.02 | 140.63 | 128.50 |
| sys | 418.89 | 55.51 | 188.92 | 41.82 |

Prikaz 14

Pri čemu, manje vrednosti procenata indikuju brže izvršavanje prvog od dva slučaja koja se porede.  
Npr. poređenjem SIMD, RSWS i -O3 vidimo da SIMD, RSWS varijanti treba 67% vremena -O3 varijante za izvršavanje, odnosno brža je za .

Pri uzimanju u obzir ovih cifara, treba napomenuti da Cpp.Linux projekat ne koristi AVX instrukcije za optimizaciju glavnog dela FFT algoritma, ali koristi AVX instrukcije za učitavanje i obradu .WAV fajla.  
Stoga, ako bi se iste AVX instrukcije upotrebile u Cpp.Linux projektu, moguće je videti drugačije rezultate.

# Poređenje po broju instrukcija i samim instrukcijama

Upotrebom callgrind alata iz valgrind skupa alata, generisani su dump-ovi koji su upotrebljeni u svrhe poređenja broja instrukcija i koda izvršnog koda svih varijanti. Dump-ovi su generisani komandom:

valgrind --tool=callgrind ./SASMprog\_\*varijanta\*.exe ./output.wav

Ispod je priložena tabela sa procenjenim brojem CPU ciklusa pri izvršavanju različitih varijanti izvršnog koda.

Cpp.Linux

|  |  |
| --- | --- |
|  | Procenjeni broj CPU ciklusa |
| No opts | 161 995 441 |
| -Os | 64 393 178 |
| -O2 | 56 950 372 |
| -O3 | 54 740 209 |

ASM

|  |  |
| --- | --- |
|  | Procenjeni broj CPU ciklusa |
| No opts | 93 716 528 |
| RSWS | 92 859 398 |
| SIMD | 85 811 196 |
| SIMD, RSWS | 84 954 059 |

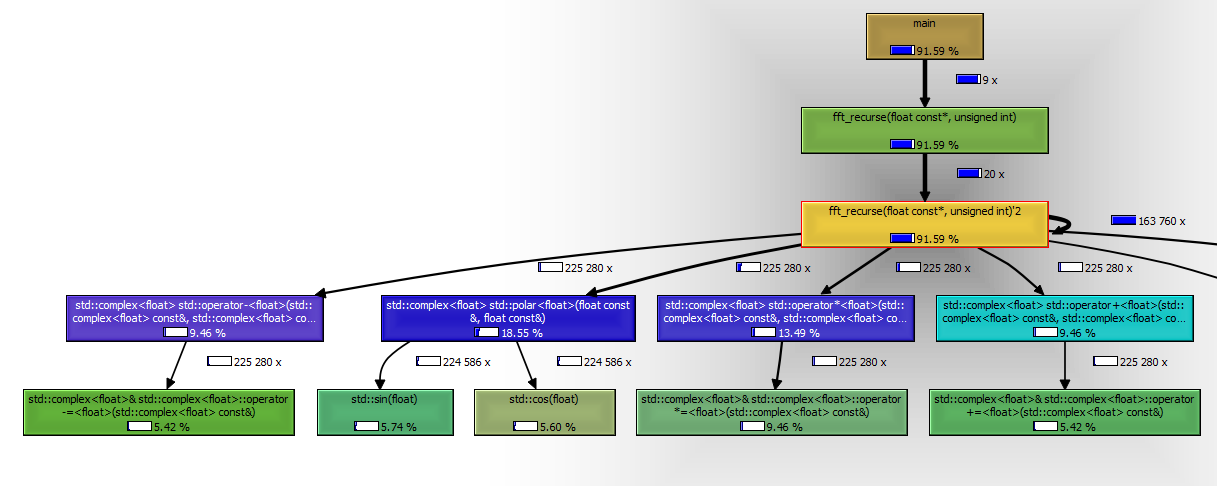
Interesantno je da -O2 nivo optimizacije rezultuje u relativno velikom procenjenom broju CPU ciklusa, iako je izvršavanje brže u odnosu na neke varijante.

Takođe interesantno zapažanje je da RSWS varijanta ima najveći broj procenjenih instrukcija u odnosu na ostale varijante, ali je ipak bila brža u odnosu na neke druge varijante. Ovo može biti rezultat dubokog pipeline-ovanja koje je moguće sa AVX instrukcijama, koje su, između ostalog, korištene u RSWS varijanti.

## Analiza instrukcija

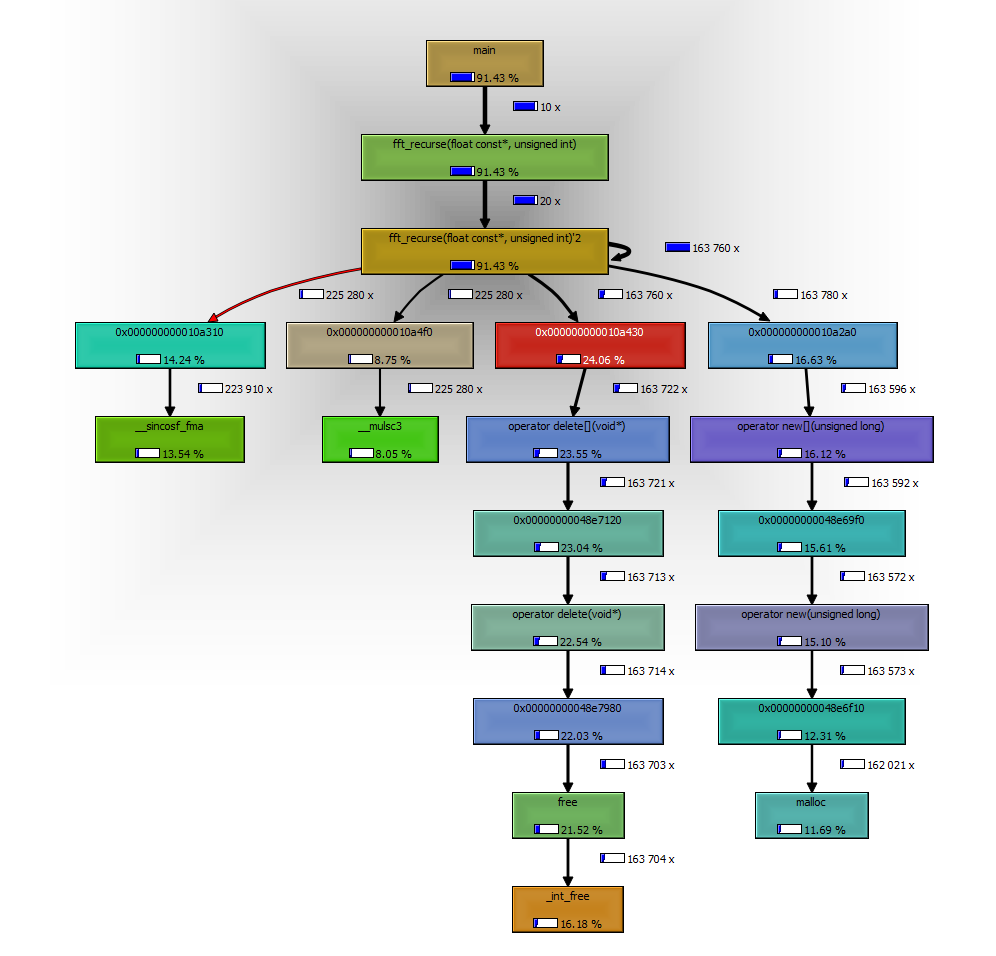
Ispod su priloženi grafovi poziva rutina za Cpp.Linux i ASM varijante.

Vidimo da Cpp.Linux No opts varijanta značajan broj ciklusa provodi izvršavanjem funkcija za kreiranje komplesnog broja iz polarne forme kompleksnog broja. Svekupuno, 50.56%. Pri tome se koriste sinusna i kosinusna funkcija, koje zajedno uzimaju desetak posto ukupnog izvršavanja FFT algoritma, što nije uopšte zanemarljivo obzirom da se obe funkcije moraju evaluirati minimalno za svaki ulazni uzorak u algoritmu.



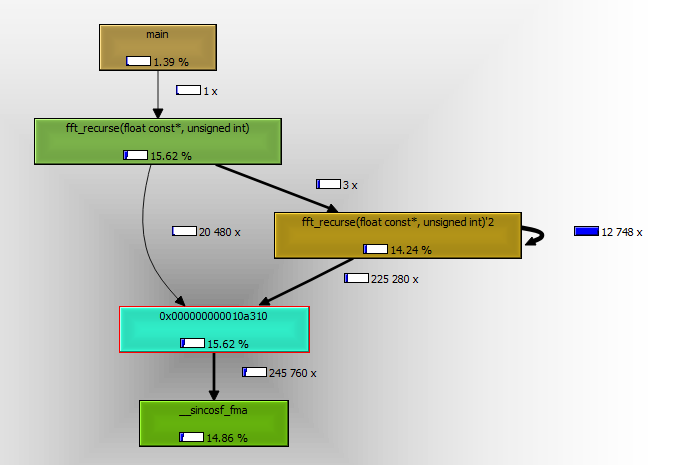
Prikaz 15 Cpp.Linux No opts

Primećujemo da -Os nivo optimizacija sažima konverzije iz polarnog oblika kompleksnih brojea, potencijalno inline-ovanjem, kako bi se redukovao ukupan broj instrukcija i njihova dužina. Primećujemo takođe da se sada ne pozivaju funkcije STL-a za konverziju.



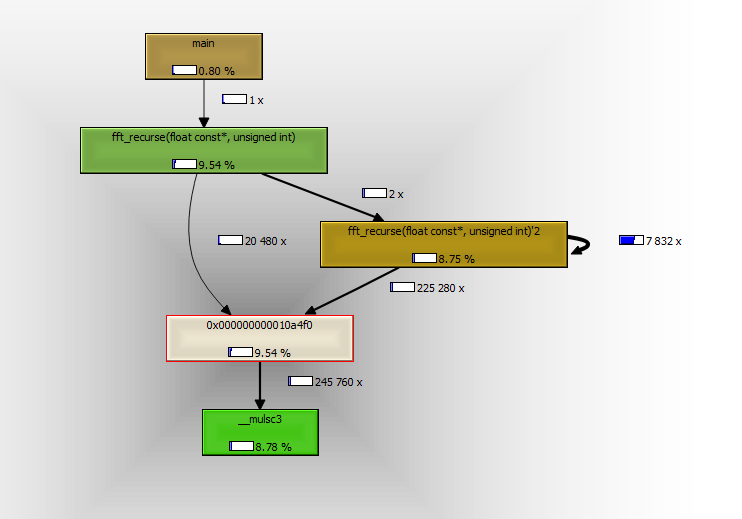
Prikaz 16 Cpp.Linux -Os

Umesto toga, koristi se rutina \_sincosf\_fma[[3]](#endnote-3) koja istovremeno evaluira sinusnu i kosinusnu funkciju, s tim da više ciklusa ide ka tome.



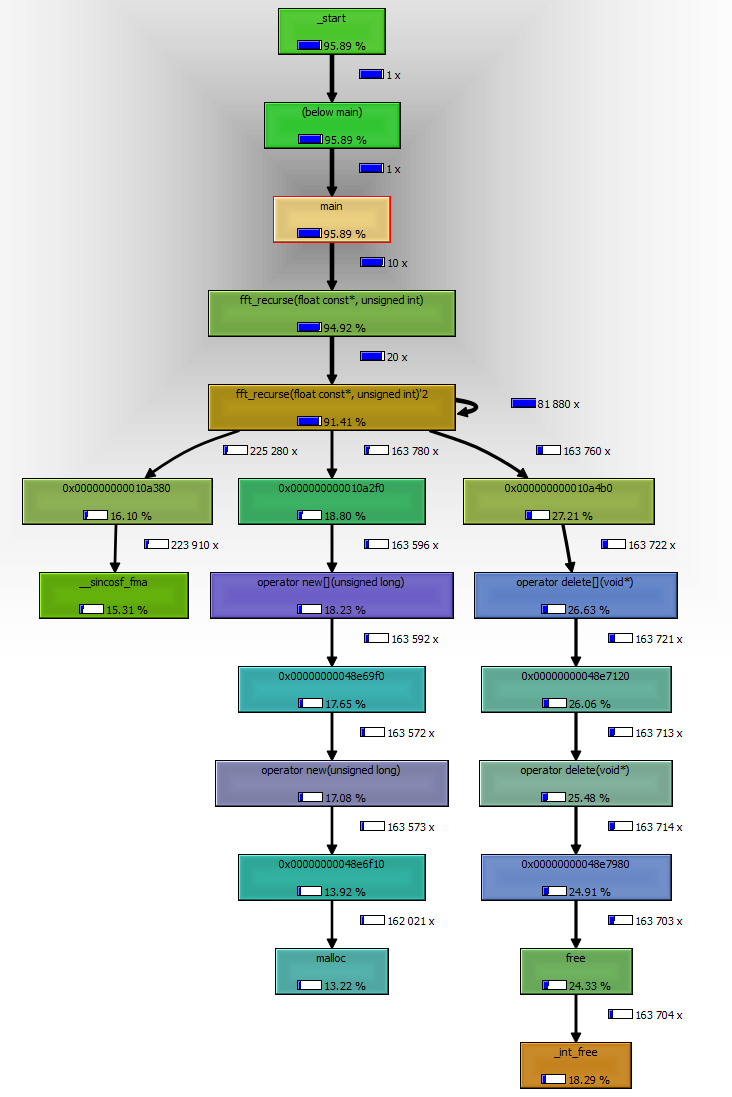
Prikaz 17Cpp.Linux -Os \_sincosf\_fma

Takođe koristi \_mulsc3[[4]](#endnote-4) rutinu, koja se koristi za evaluiranje kompleksnih brojeva, pri čemu se broj a diže na stepen b.



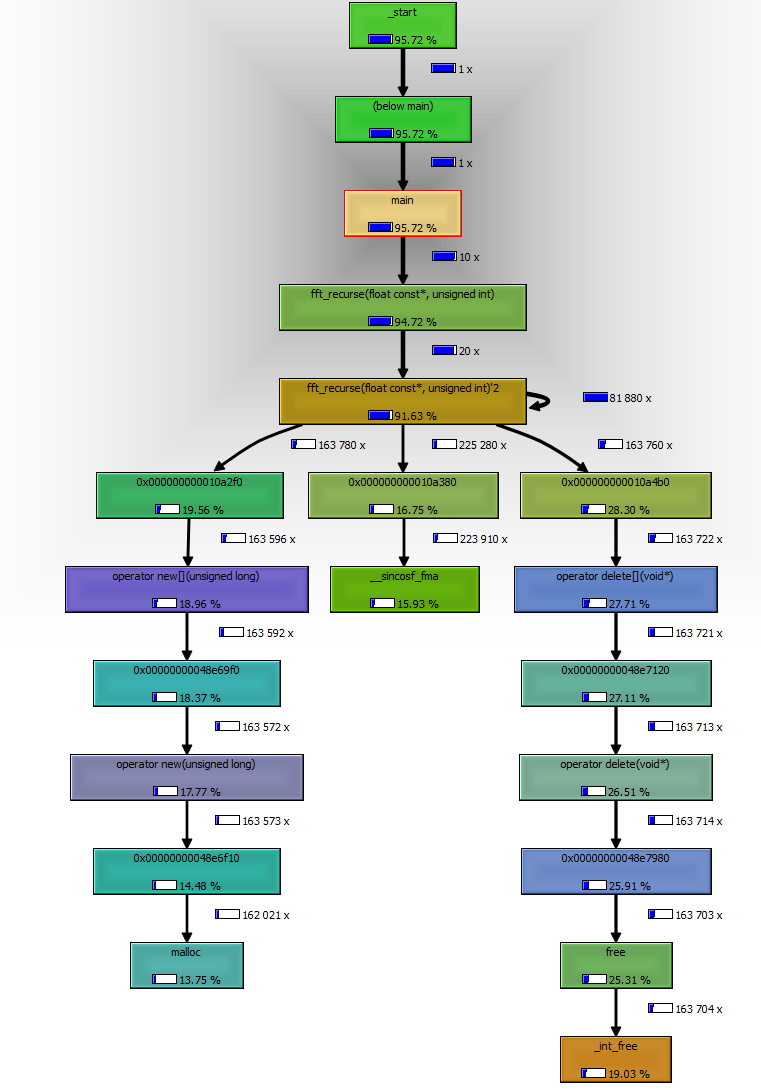
Prikaz 18 Cpp.Linux -Os \_mulsc3

Interesantno je primetiti da -O2 nivo optimizacija koristi \_sincosf\_fma rutinu. Pritom, vidimo značajnu razliku u broju instrukcija za alokaciji i dealokaciji dinamičke memorije.



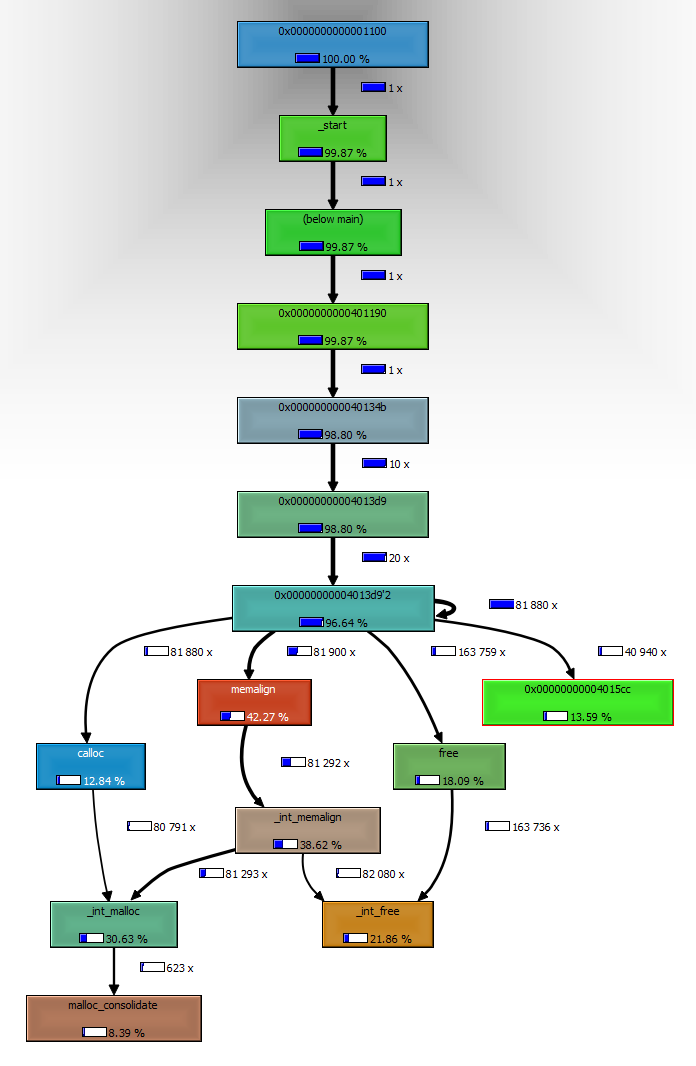
Prikaz 19 Cpp.Linux -O2

Sa nivoom -O3, imamo ponovo upotrebu \_sincosf\_fma, pri čemu su najverovatnije inline-ovane funkcije iz STL-a. Velika razlika je ponovo u načinu pozivanja i broju izvršenih instrukcija za alokaciju i dealokaciju dinamičke memorije.



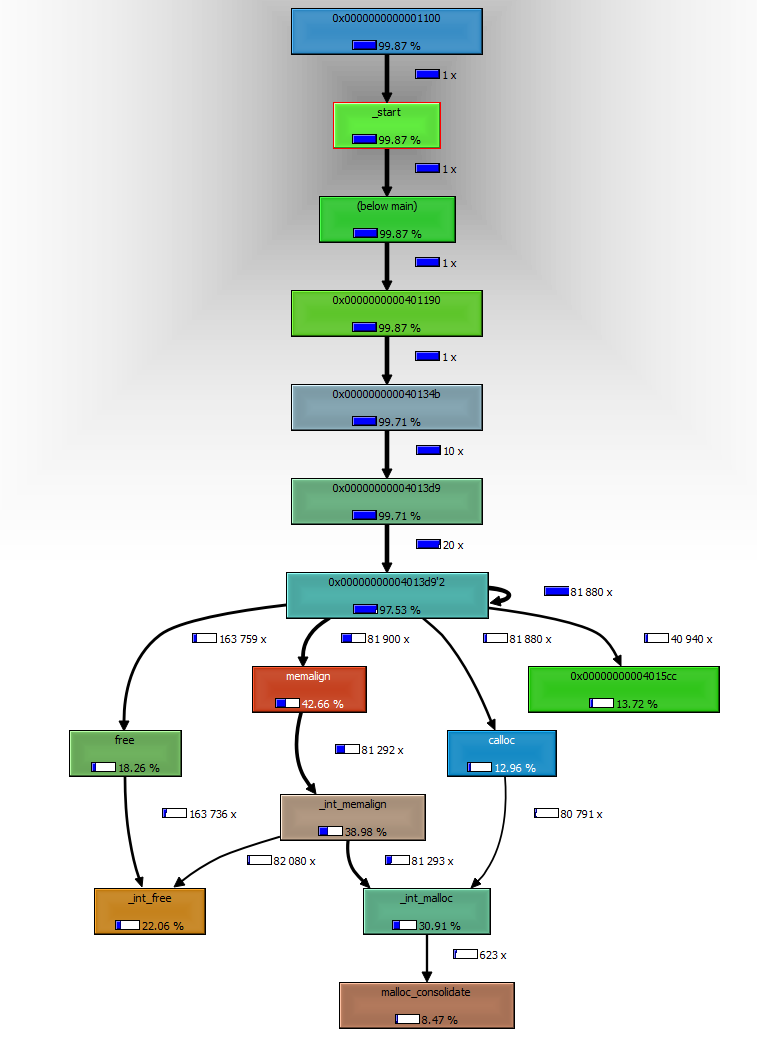
Prikaz 20 Cpp.Linux -O3

Interesantno je primetiti da samo 13.59% instrukcija odlazi ka samom algoritmu, dok preko 80% ciklusa u ASM No opts varijanti odlazi ka dinamičkom alociranju memorije pomoću calloc, memalign i free rutina. Ovo nam indikuje da, pri optimalnijem alociranu memorije, možemo videti značajna poboljšanja u radu ASM implementacija. Takođe bi bilo interesantno odrediti granicu kada se duže izvršavanje memalign rutine u kombinaciji sa aligned AVX instrukcijama ne isplati, u odnosu na situaciju gde se koristi calloc/malloc bez align-ovanja adresa, sa unaligned AVX instrukcijama.



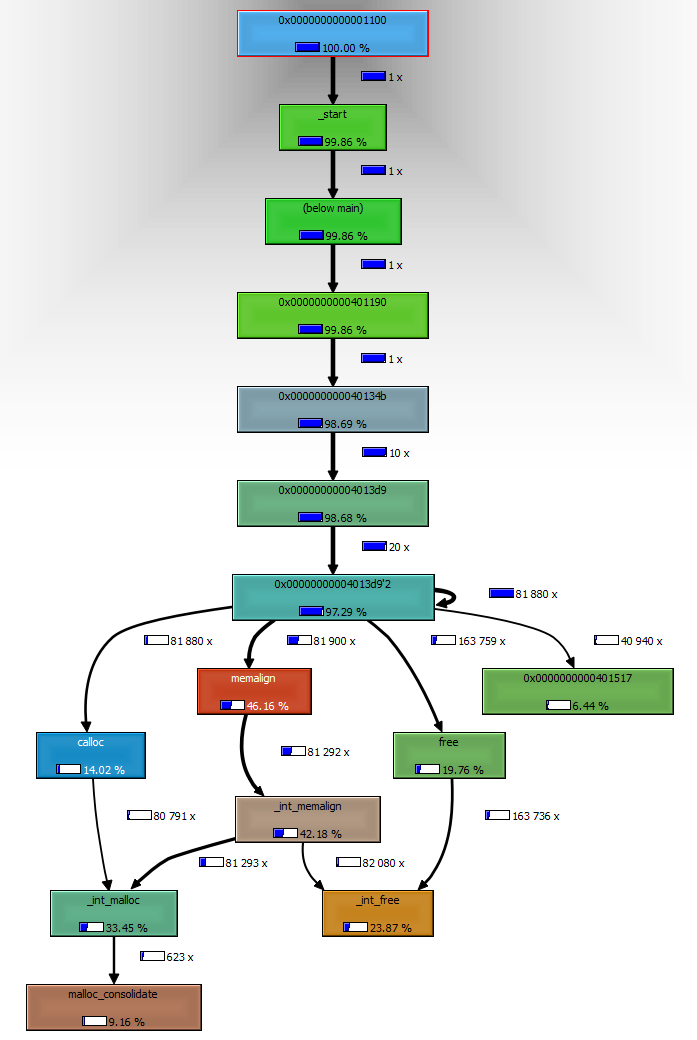
Prikaz 21 ASM No opts

U strukturi poziva RSWS varijante, nema značajne razlike u odnosu na No opts varijantu.



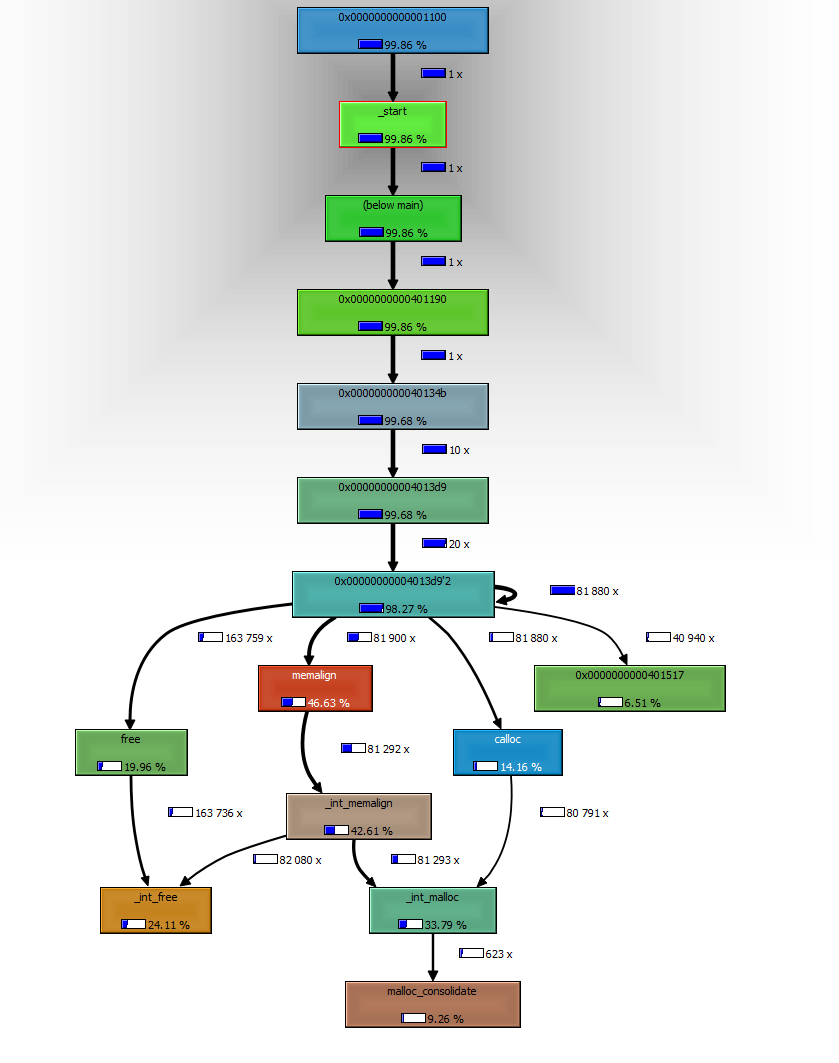
Prikaz 22 ASM RSWS

Primećujemo da u SIMD varijanti, još manji procenat ciklusa odlazi ka FFT rutini za izračunavanje, zbog upotrebe AVX instrukcija.



Prikaz 23 ASM SIMD

Kao i sa osnovnom RSWS varijantom, ne primećujemo značajnu razliku u odnosu na prethodnu varijantu.

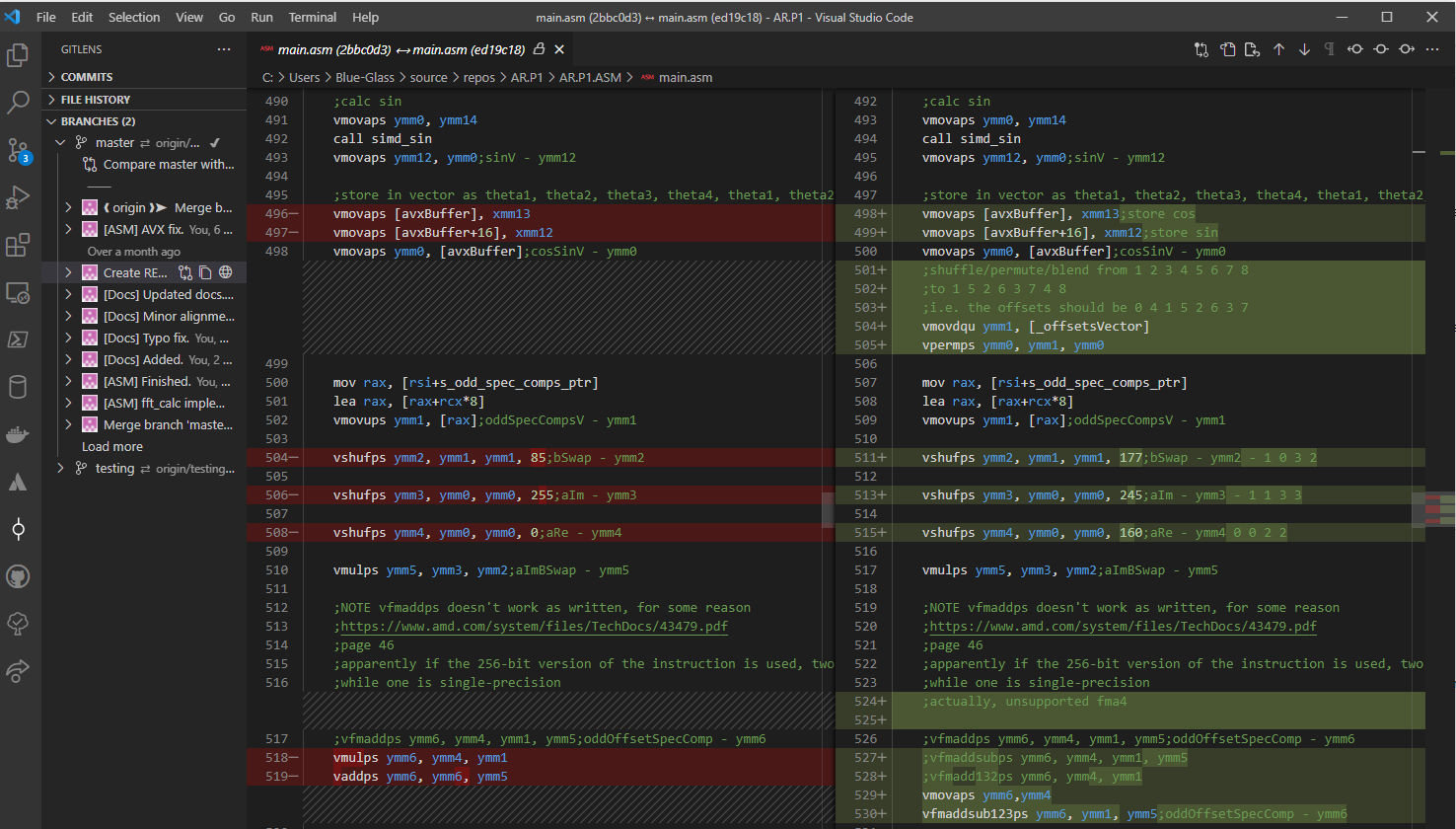


Prikaz 24 ASM SIMD, RSWS

# Napomena

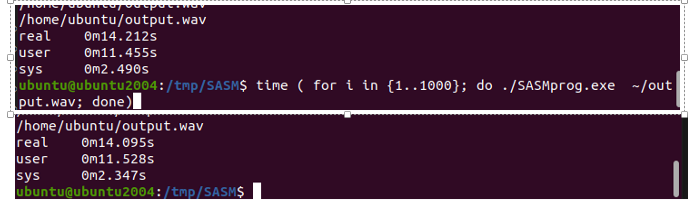
Nakon što su svi grafički i računski prikazi dobijeni, blago je modifikovana fft\_calc rutina. Modifikovane su samo AVX instrukcije na sledeće načine:

1. korigovane VSHUFPS bit maske
2. dodata jedna VPERMPS instrukcija
3. zamenjene instrukcije množenja, sabiranja i oduzimanja jednom FMA instrukcijom



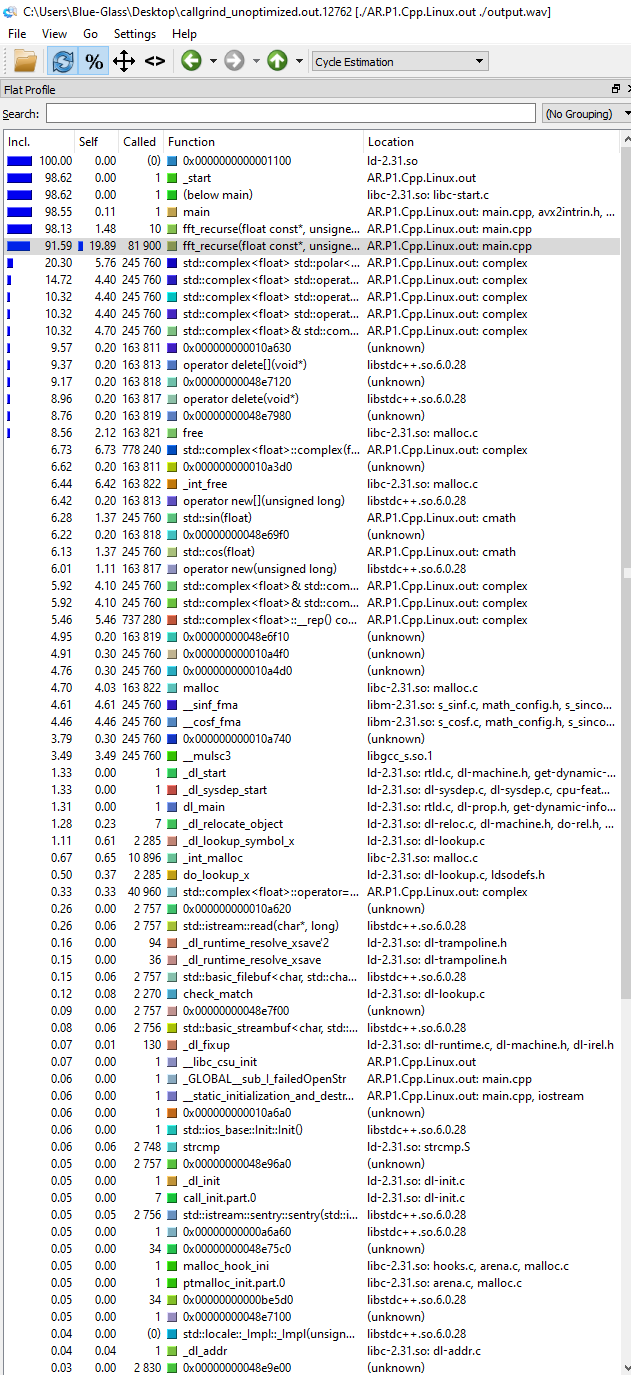
Prikaz Modifikacije fft\_calc

S tim u obziru. moguće je da dobijeni rezultati pri merenju budu drugačiji od tih iz ostatka ovog dokumenta. Inicijalna merenja daju sledeće rezultate za SIMD, RSWS varijantu.

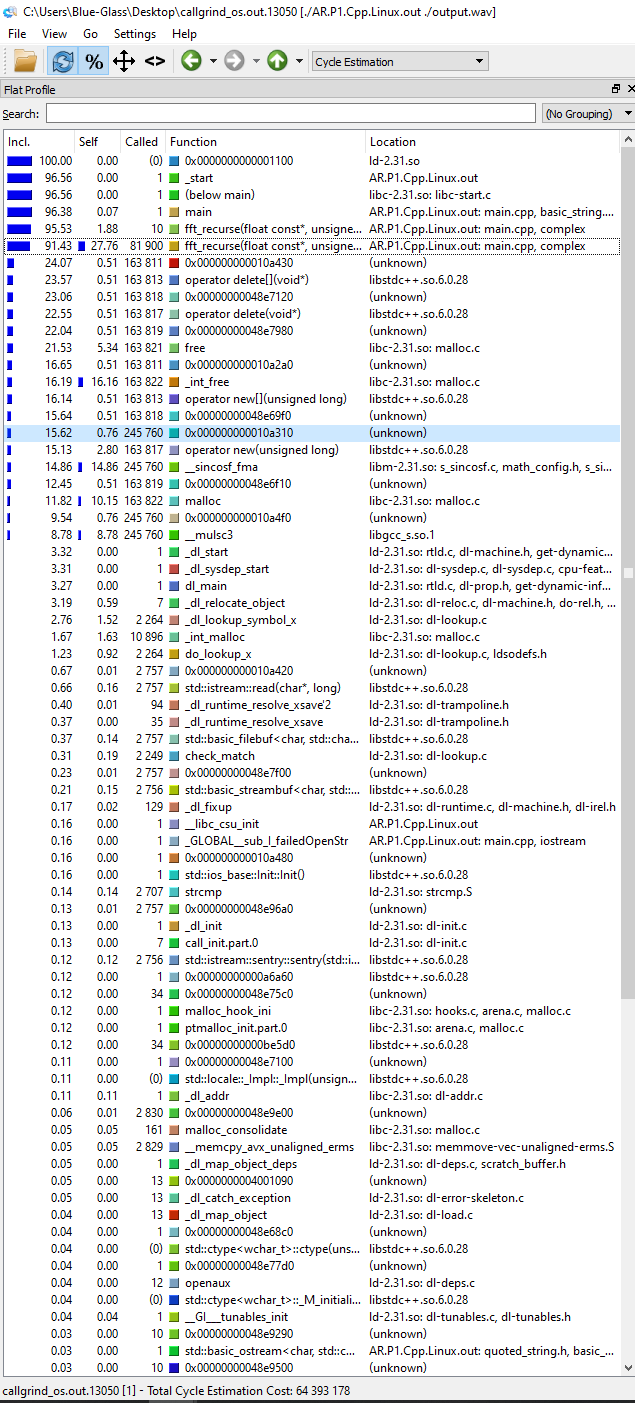


Prikaz Merenja SIMD,RSWS pri modifikaciji fft\_calc

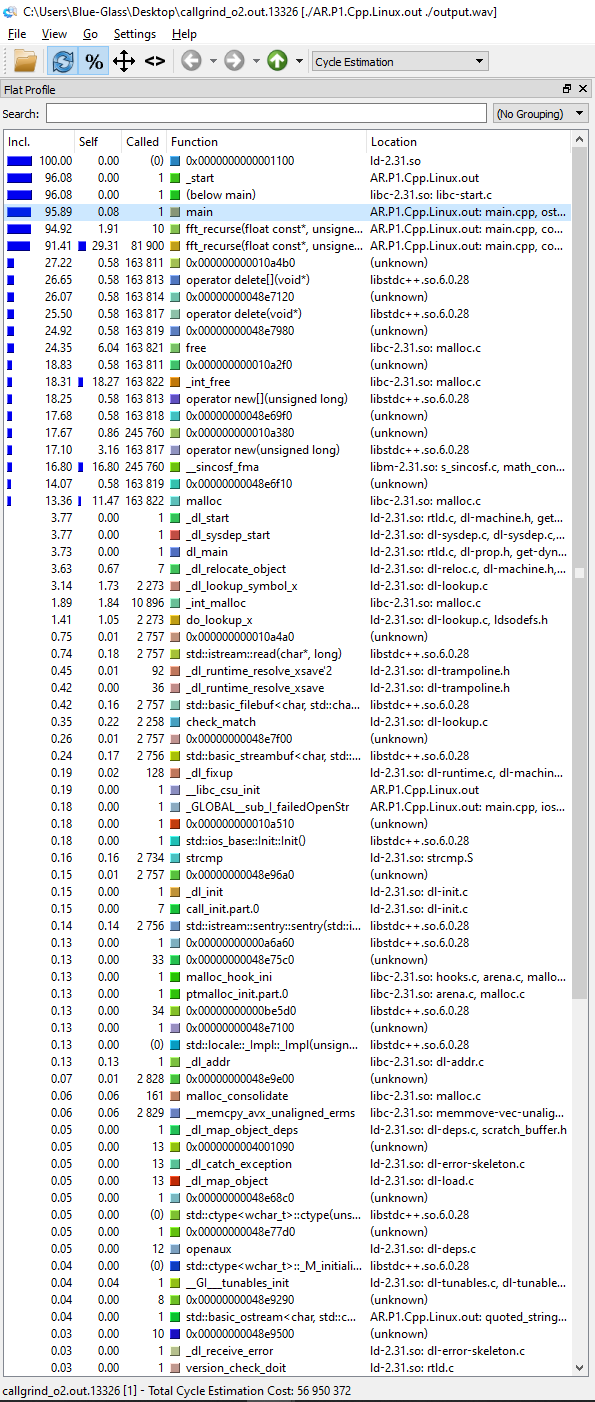
# Dodatak



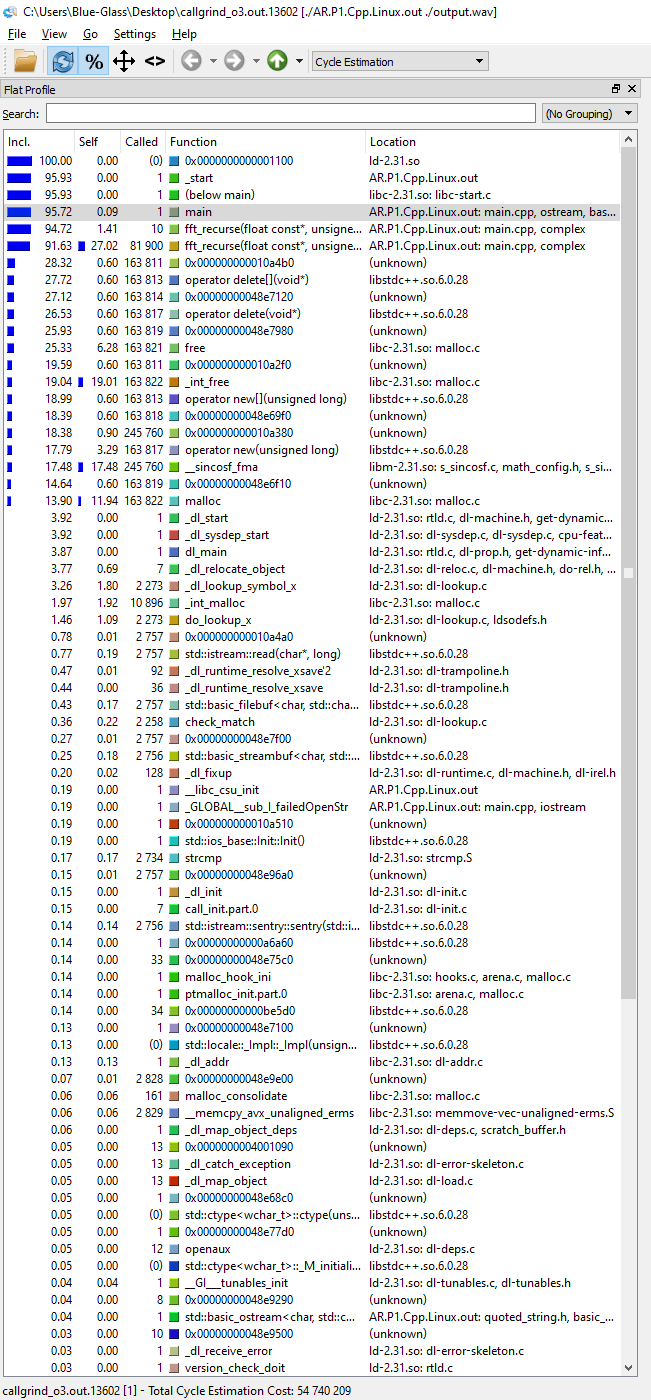
Prikaz 27 Cpp.Linux No opts



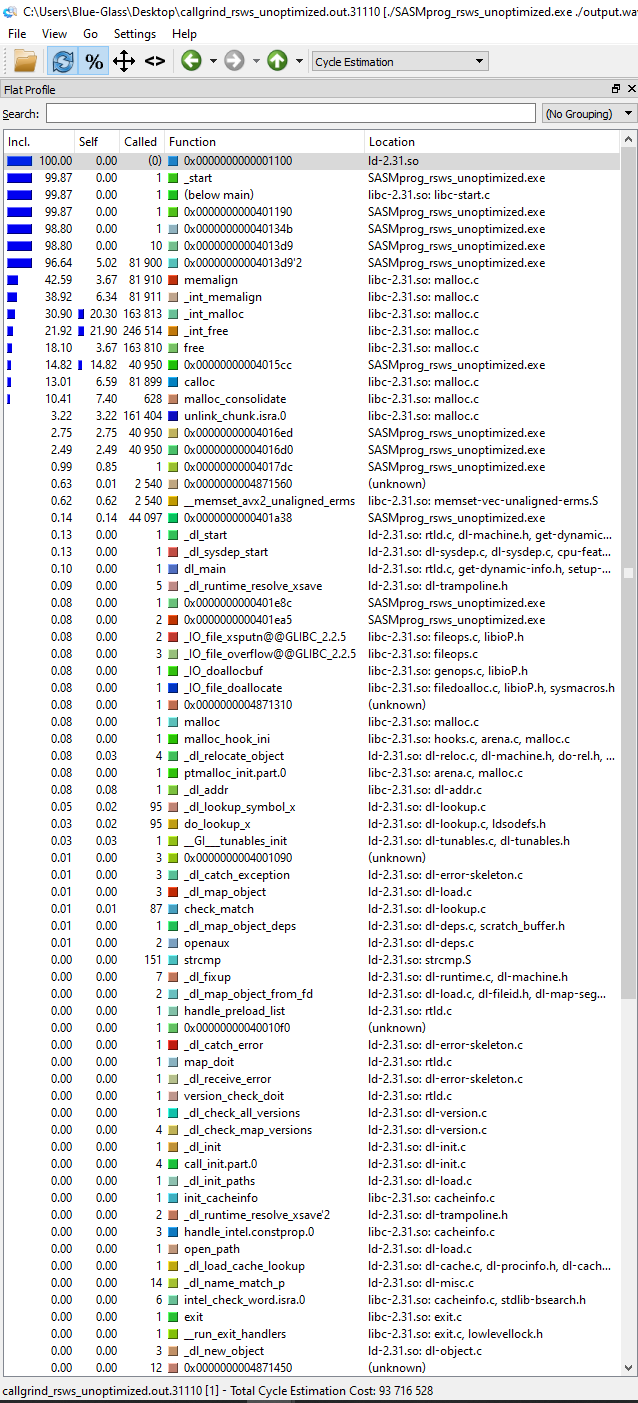
Prikaz 28 Cpp.Linux -Os



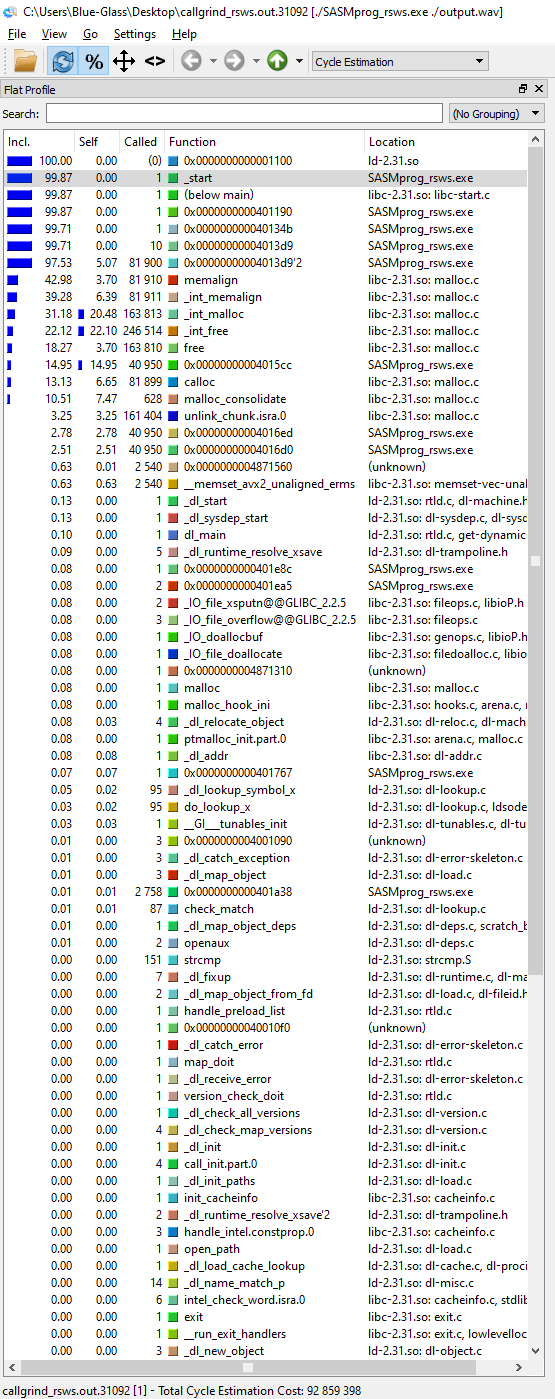
Prikaz 29 Cpp.Linux -O2



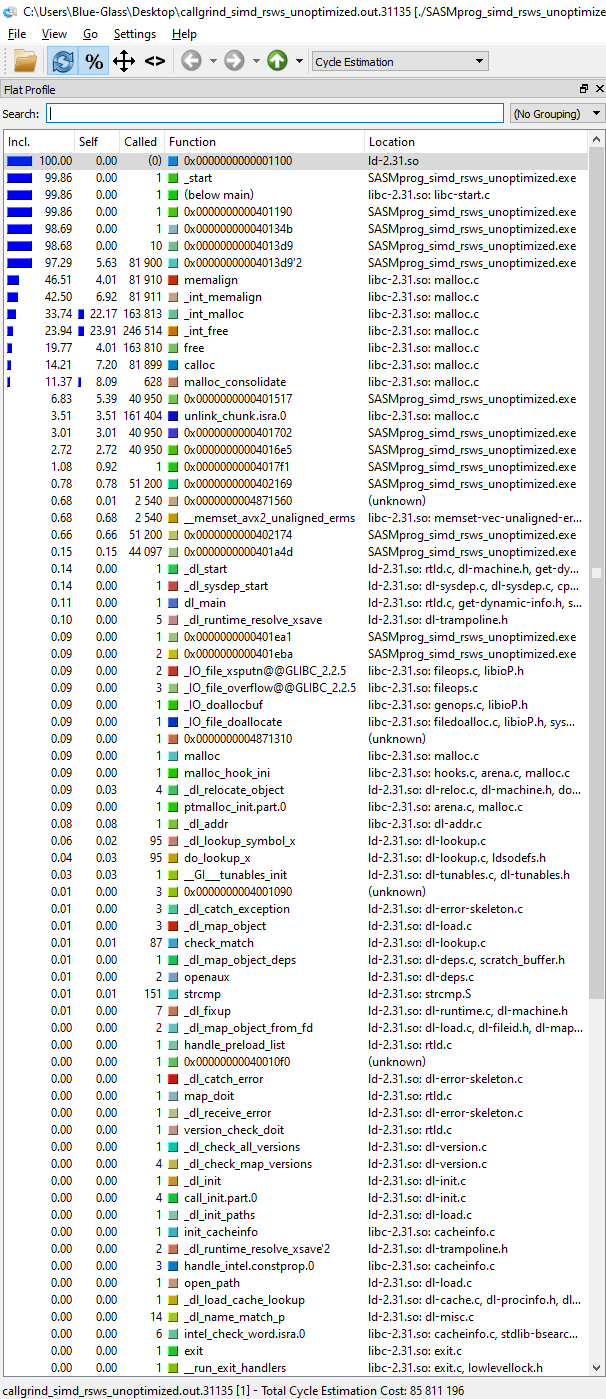
Prikaz 30 Cpp.Linux -O3



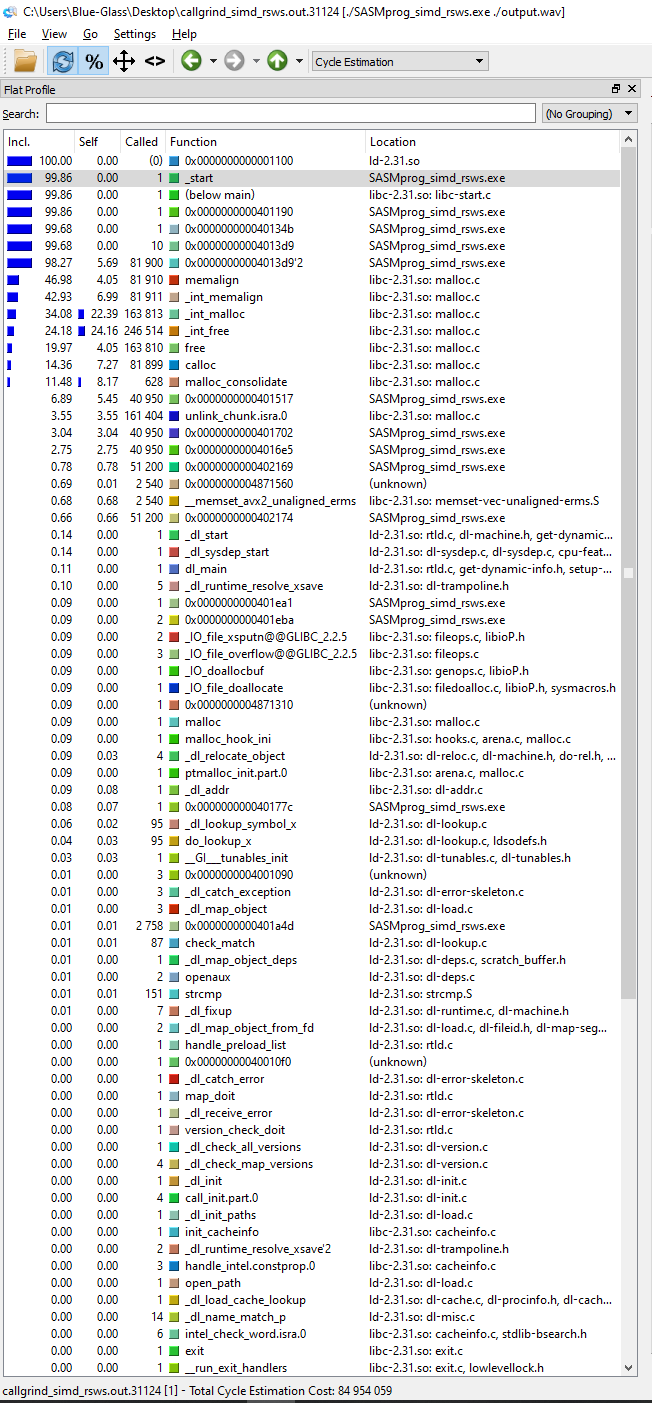
Prikaz 31 ASM No opts



Prikaz 32 ASM RSWS



Prikaz 33 ASM SIMD



Prikaz 34 ASM SIMD RSWS

1. Cooley, J. and Tukey, J. (1965) "An algorithm for the machine calculation of complex Fourier series", Mathematics of Computation, 19(90), pp. 297-301. doi: 10.1090/s0025-5718-1965-0178586-1. [↑](#endnote-ref-1)
2. SASM - Simple crossplatform IDE for NASM, MASM, GAS, FASM assembly languages (2021). Available at: <https://dman95.github.io/SASM/english.html> (Accessed: 28 December 2021). [↑](#endnote-ref-2)
3. [03/10] Improve performance of sinf and cosf - Patchwork (2021). Available at: <https://patchwork.ozlabs.org/project/glibc/patch/480576b5-fbb7-39b2-d80c-80c8931fde21@arm.com/#1973677> (Accessed: 28 December 2021). [↑](#endnote-ref-3)
4. Soft float library routines (GNU Compiler Collection (GCC) Internals) (2021). Available at: <https://gcc.gnu.org/onlinedocs/gccint/Soft-float-library-routines.html> (Accessed: 28 December 2021). [↑](#endnote-ref-4)