

Vizualizace profilovacích informací překladače GCC v prostředí IDE CodeLite

Visualization of the Profiling Information of the GCC Compiler in
CodeLite IDE

Bc. Václav Špruček



ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Václav ŠPRUČEK**
Osobní číslo: **A09767**
Studijní program: **N 3902 Inženýrská informatika**
Studijní obor: **Informační technologie**

Téma práce: **Vizualizace profilovacích informací překladače GCC
v prostředí IDE CodeLite**

Zásady pro vypracování:

1. Vytvořte rešerši na téma profilování programového kódu a měření výkonu aplikací.
2. Vytvořte rešerši na téma profilování aplikací pomocí nástroje gprof.
3. Pomocí programovacího jazyka C/CPP, knihovny wxWidgets a nástrojů gprof a Graphviz vytvořte zásuvný modul pro IDE CodeLite schopný grafickou a tabulkovou formou vizualizovat informace z profileru překladače.
4. Vizualizujte časové statistiky běhu programu a využití volaných funkcí.
5. Modul publikujte pod licencí GPL v2.
6. Vytvořte programovou a uživatelskou dokumentaci.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **BLIŽŇÁK, Michal. Systémové programování. 1. vyd. Zlín: UTB ve Zlíně, 2005. 202 s. ISBN 80-7318-364-1**
2. **MASTERS, Jon. Linux profesionálně: programování aplikací. 1.vyd. Brno: Zoner Press, 2008. 539 s. ISBN 978-80-86815-71-8**
3. **SMART, Julian, HOCK, Kevin. Cross-Platform GUI Programming with wxWidgets, Prentice Hall, 2006, ISBN 0-13-147381-6**
4. **PRATA, S. Mistrovství v C++: Computer Press, 2007, 3. vydání, 1120s, ISBN 978-80-251-1749-1**
5. **CORMEN, T. H., LEISERSON, Ch. E., RIVEST, R. L., STEIN, C: Introduction to Algorithms, The MIT Press, Cambridge, 2009, 3rd edition, ISBN 978-0-262-03384-8**
6. **STEELE, J, ILIINSKI, N: Beautiful Visualization, OReilly Media, Inc., Sebastopol, 2010, ISBN 978-1-449-37986-5**
7. **GANSNER, E.R., NORTH, S.C. An open graph visualization system and its applications to software engineering. Softw. Pract. Exper. 30, 11 (Sep. 2000), pp. 1203-1233, ISSN 0038-0644**

Vedoucí diplomové práce:

Ing. Michal Bližňák, Ph.D.

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

24. února 2012

Termín odevzdání diplomové práce:

21. května 2012

Ve Zlíně dne 24. února 2012



prof. Ing. Vladimír Vašek, CSc.
děkan



doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

ABSTRAKT

Cílem práce bylo vytvořit zásuvný modul pro vývojové prostředí CodeLite. Modul představuje dynamická knihovna umožňující vizualizovat profilovací informace překladače GCC. Vizualizace je řešena zobrazením grafu, ze kterého je zřejmé propojení jednotlivých uzlů s pomocí hran na základě vzájemného volání a vytížení funkcí v běžícím programu. Nastavení zásuvného modulu umožňuje filtrovat jednotlivé uzly a hrany dle jejich vytížení a snadněji tak porozumět chování běžícího programu. Graf volání funkcí lze využít k lepší identifikaci a optimalizaci nejčastěji prováděného kódu programu.

Klíčová slova: zásuvný modul, vizualizace, graf volání funkcí, optimalizace

ABSTRACT

The aim of this work was to create a plug-in for development environment CodeLite. The module is a dynamic library, allowing visualizing profiling information of GCC interpreter. Visualization is solved by showing a graph from which the evident connections between the nodes with edges by mutual call and load level in a running program. Setting of plug-in allows filtering the nodes and edges according to their usage and allow easier understand the behavior of a running program. Call-Graph function can be used for better identification and optimizing of most frequently performed program code.

Keywords: plug-in, visualization, Call-graph function, optimizing

Velké poděkování patří vedoucímu diplomové práce panu Ing. Michalu Bližňákovi, Ph.D. za poskytnuté rady, velkou ochotu a odborné vedení během tvorby této práce.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD.....	9
I TEORETICKÁ ČÁST.....	10
1 VÝVOJOVÉ NÁSTROJE	11
1.1 MS VISUAL STUDIO	11
1.2 NETBEANS	11
1.3 CODELITE	12
1.3.1 Zásuvné moduly	12
2 MULTIPLATFORMNÍ PROGRAMOVÁNÍ.....	13
2.1 PŘEKLADAČ GCC	13
2.2 KNIHOVNA WXWIDGETS	13
3 PROFILOVÁNÍ PROGRAMOVÉHO KÓDU.....	14
3.1 ZPŮSOBY PROFILOVÁNÍ	14
3.2 PROFILOVACÍ NÁSTROJE	14
3.2.1 Intel® VTune™ Amplifier XE	14
3.2.2 Very Sleepy	15
3.2.3 Valgrind.....	15
4 PROFILOVÁNÍ POMOCÍ NÁSTROJE GPROF	16
4.1 POUŽITÍ NÁSTROJE GPROF	16
4.2 POPIS VÝSTUPU PROFILERU	16
4.2.1 Plošný profil	16
4.2.2 Graf volání	17
5 MĚŘENÍ VÝKONU APLIKACÍ.....	20
5.1 ZPŮSOBY MĚŘENÍ	20
5.2 CÍL MĚŘENÍ	21
6 VIZUALIZAČNÍ NÁSTROJE.....	23
6.1 GRAPHVIZ.....	23
6.1.1 Program DOT.....	23
6.1.2 Jazyk DOT	23
II PRAKTICKÁ ČÁST	28
7 UŽIVATELSKÁ DOKUMENTACE	29
7.1 INTEGRACE DO CODELITE	29
7.2 PŘEDSTAVENÍ PLUGINU	31
7.3 UŽIVATELSKÉ ROZHRANÍ	31
7.3.1 Popis nastavení dialogu	33
7.4 PRÁCE S PLUGINEM.....	34
7.5 UKÁZKY VÝSTUPU.....	38
8 PROGRAMOVÁ DOKUMENTACE.....	42
8.1 STRUKTURA PROJEKTU	42
8.1.1 Popis tříd	42
8.1.2 Diagram tříd	43
8.1.3 Příklad užití	44

8.1.4	Důležitý programový kód	45
8.2	VÝSTUPNÍ DATA	47
8.2.1	Popis souborů	47
ZÁVĚR		48
ZÁVĚR V ANGLIČTINĚ.....		49
SEZNAM POUŽITÉ LITERATURY.....		50
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....		52
SEZNAM OBRÁZKŮ		53

ÚVOD

Oblast vývoje software (aplikací) se v dnešní době neobejde bez kvalitních vývojových nástrojů. Ty je možné rozdělit na placené (například Visual Studio od firmy Microsoft) a neplacené, tedy volně dostupné, mezi něž spadá i vývojové prostředí CodeLite, které bylo použito pro napsání zdrojového kódu této diplomové práce. Dalším kritériem pro rozdělení je míra vhodnosti pro daný programovací jazyk, v případě této diplomové práce se jedná o jazyky C a C++. Výhodou volně dostupných nástrojů je jejich dosažitelnost pro každého, finanční nenáročnost, nezávislost a „přirozenost“ vývoje postavená na principech open-source. Vývojový nástroj CodeLite lze spustit na více platformách, což umožní nainstalovat naprogramovaný zásuvný modul (plugin) na třech operačních systémech: Windows, Linux a Mac OS X. Zdrojový kód programu využívá další multiplatformní knihovnu wxWidgets, jež umožnila vytvoření rozhraní pro nastavení samotného zásuvného modulu. Dynamická knihovna byla kompilovaná společně s aktuálním zdrojovým kódem CodeLite a přispěla k rozšíření jeho funkcionality.

V rámci diplomové práce byl použit externí nástroj Graphviz. Jedná se o vizualizační software, který dokáže ze zpracovaného výstupu překladače GCC vytvořit graf volání jednotlivých funkcí běžícího programu popsaného v jazyku DOT. Naprogramování dynamické knihovny, která zpracovává výstup překladače GCC pro nástroj Graphviz je jedním z cílů této diplomové práce. Výsledný obrázek grafu volání funkcí je zobrazen v záložce vývojového nástroje CodeLite a slouží k lepší identifikaci a optimalizaci prováděného kódu programu. Dále poskytuje lepší pochopení prováděného programu a snadněji ukazuje nejvytíženější funkce, tzv. úzké hrdlo programu.

V teoretické části jsou vysvětleny základní i specifické pojmy a popsány jednotlivé nástroje použité při vzniku této dynamické knihovny. V praktické části práce je popsáno uživatelské rozhraní, ukázky výstupu dané aplikace a způsob naprogramování nejdůležitějších částí zdrojových kódů pluginu.

I. TEORETICKÁ ČÁST

1 VÝVOJOVÉ NÁSTROJE

V úvodní kapitole budou představeny některé vývojové nástroje. Vývojovým nástrojem je program, který umožňuje snadné a přehledné napsání programového kódu, který je možné s pomocí překladače ve vývojovém prostředí přeložit a spustit výsledný program. Vývojové nástroje je možné rozdělit na platformě závislé a nezávislé, vhodné pro daný programovací jazyk nebo placené a neplacené. Dnešní vývojáři aplikací pro stolní počítače nebo webové aplikace mají k dispozici poměrně velké množství různých nástrojů. Níže budou představeny některé ze základních vývojových nástrojů a jejich možnosti profilování výkonu aplikace.

1.1 MS Visual Studio

Microsoft Visual Studio je vývojové prostředí od firmy Microsoft. Tento vývojový nástroj je vydáván v několika edicích každý rok. Poskytuje podporu pro několik programovacích jazyků jako například Visual Basic, Visual C#, Visual C++, and Visual Web Developer. Základní verze, která je dostupná zdarma má označení Express. Další verze jsou placené a mají označení Standard, Professional, Premium a Ultimate. [1]

Verze Ultimate nabízí několik ladících a diagnostických nástrojů. Důležitý a zajímavý nástroj, který je v této verzi integrován umožňuje profilování výkonu aplikace. Nástroj pomáhá odhalit úzká místa výkonu aplikace.

Profilování probíhá dvěma způsoby s pomocí metody „sampling“:

- režim s nižší režii pro relativní data o výkonu,
- režim přesného měření s vyšší režii pro přesnější výsledky.

1.2 NetBeans

NetBeans je volně dostupný vývojový nástroj s podporou programovacích jazyků Java, C++, PHP. Nástroj je multiplatformní s možností rozšíření pomocí pluginu na vlastní vývojové platformě. Prostředí je použitelné na operačních systémech Windows, Linux, Mac OS X a Solaris. Vývojové prostředí obsahuje nástroj NetBeans Profiler, který umožňuje profilování výkonu procesoru i paměťové náročnosti aplikací. [2]

1.3 CodeLite

Multiplatformní vývojový nástroj použitý k vytvoření pluginu této diplomové práce. Podporuje jazyk C a C++, využívá překladač MinGW a multiplatformní knihovnu wxWidgets. Tento nástroj je šířen pod licencí GPLv2. Na webových stránkách jsou ke stažení instalační balíčky pro platformy Linux, Windows, Mac OS X. Vývojáři mají k dispozici kompletní zdrojové kódy projektu „LiteEditor“. Díky dostupnosti zdrojových kódů je možné vytvářet vlastní pluginy, tak jako je použito v případě této práce. Stávající zásuvné moduly poskytují snadnější práci v tomto vývojovém prostředí a usnadňují vývoj a ladění aplikací. [3]

1.3.1 Zásuvné moduly

Zásuvným modulem je soubor představující dynamicky linkovanou knihovnu, Soubor má příponu „dll“ na Windows a „so“ na Linuxech a po integraci s IDE CodeLite poskytuje vlastní funkčnost. V současné době CodeLite obsahuje přibližně 11 pluginů. V následujícím textu jsou uvedeny některé pluginy a externí nástroje:

- CppCheck – nástroj pro statickou analýzu programového kódu jazyka C a C++, který provádí kontrolu syntaxe zápisu jazyka a slouží k nalezení úniků paměti, alokace paměti, přetečení zásobníku, ukazatele;
- UnitTest++ – nástroj pro vytváření Unit Projektu pro testování a ověřování správné funkčnosti dílčích částí zdrojového kódu;
- External Tools – nástroj pro přidání externího nástroje. Pomocí tohoto nástroje se dají přidat programy a používat jejich vstupy a výstupy v CodeLite. V CodeLite je například možné spustit příkazový řádek s předem definovanými parametry. Nástroj je též možné použít k připojení externího profilovacího nástroje jako je například Valgrind (Callgrind).

2 MULTIPLATFORMNÍ PROGRAMOVÁNÍ

Vývoj aplikací pro různá zařízení a různé platformy si v dnešní době vyžaduje přístup psaní programového kódu pro více různých platforem. Rostoucí složitost hardwarového a softwarového prostředí si vynucuje existence nástrojů a technologií pro snadnější vývoj aplikací. Technologie multiplatformního programování poskytuje přenositelnost zdrojového kódu na různé cílové platformy. Platformu představuje různý operační systém jako například Windows, Linux, Mac OS X. Díky tomuto přístupu dochází k urychlení vývoje, minimalizaci chyb a unifikaci vzhledu a ovládání aplikací. Pro psaní multiplatformního programu je potřeba využít platformě nezávislý programovací jazyk a překladač.

2.1 Překladač GCC

GCC je kolekce aplikací určených pro překlad zdrojových kódů napsaných v různých programovacích jazycích. Nabízí podporu pro několik programovacích jazyků a je primárně určen pro překlad jazyků C a C++. Překladač je multiplatformní a podporuje například systémy Windows, Linux a Mac OS X. Pro jazyk C je potřeba volat překladač gcc a pro jazyk C++ překladač g++. [4]

2.2 Knihovna wxWidgets

Knihovna wxWidgets poskytuje podporu pro snadné, rychlé a profesionální vytvoření grafického uživatelského rozhraní programu nebo pluginu. Knihovna umožňuje psaní kódu programu pro několik různých softwarových platforem a překladačů s minimálními změnami kódu při přenosu mezi platformami.

Další vlastnosti knihovny poskytují zpracování zpráv, práci s kontexty zařízení, zpracování obrázků, podporu systémových funkcí cílového operačního systému a snadnou práci s datovými strukturami. Knihovna je implementována v jazyce C++ a její použití je možné v mnoha programovacích jazycích jako například C++, Python nebo Perl. [5]

3 PROFILOVÁNÍ PROGRAMOVÉHO KÓDU

Profilování programu je soubor statistik počtu alokace, dealokace paměti, počtu volání funkcí a celkové doby strávené v příslušných funkcích při běhu programu. Rozborem statistiky můžeme odhalit slabé místo programu. Existuje celá řada profilovacích nástrojů pro různé jazyky a operační systémy. Kapitola vysvětluje základní způsoby profilování a představí některé z profilovacích nástrojů.

3.1 Způsoby profilování

Profilování se obvykle skládá ze dvou kroků. První krok je možné nazvat měřením. Jde o získávání statistických údajů o běžícím programu. Tyto údaje se obvykle ukládají do nějakého pomocného logovacího souboru. Zaznamenávají se do něj zejména počty vyvolání jednotlivých funkcí, počty iterací v jednotlivých cyklech, počty přístupů do paměti, případně i počty přístupů na disk nebo k jiným zdrojům. Dále se zaznamenává i doba strávená při těchto zaznamenaných činnostech.

Druhý krok profilování lze nazvat profilovací analýza. V tomto kroku se získávají z naměřených údajů různé statistiky. Existují programy s grafickým rozhraním, které umožňují nejenom spočítat, která funkce byla volána nejčastěji, ale dokáží se na naměřená data podívat z různých úhlů a zobrazit výsledky přehledně pomocí tabulek nebo grafů volání funkcí. Pomocí profilovací analýzy se hledá místo (funkce) v programu, v nichž program tráví nejvíce času. Získáním přehledu o tom, jak se jednotlivé funkce podílejí na celkové době běhu programu nebo na přístupu k jiným zdrojům (paměť, síť, atd.), se dá lépe rozhodnout, která místa (funkce) a jakým způsobem lze optimalizovat. Takové místo v programu je možné nazvat „úzké hrdlo aplikace“. [6]

3.2 Profilovací nástroje

V textu níže budou představeny některé profilovací nástroje pro jazyk C a C++. Popsané nástroje poskytují různý výstup a data zpracovaná během profilování a prezentaci získaných dat.

3.2.1 Intel® VTune™ Amplifier XE

Intel® VTune™ Amplifier XE je nástroj k optimalizaci výkonu a vláken pro vývojáře C a C++, .Net a Fortran, kteří potřebují porozumět paralelnímu a sériovému chování aplikací ke zvýšení výkonu a škálovatelnosti.

Výkonný nástroj pro profilování výkonu a analýzu výkonu ve Windows a Linuxových aplikacích, poskytuje rychlý přístup pro škálovatelné informace pro rychlejší a lepší rozhodování vývoje programů.

Nástroj Intel VTune Amplifier XE slouží ke zjištění a nastavení optimálního výkonu, využití všech jader a instrukcí procesoru. [7]

3.2.2 Very Sleepy

Nástroj Sleepy je určený pro jazyky C a C++ k profilování výkonu procesoru pro operační systémy Windows. Upravená verze Very Sleepy má spoustu vylepšení jako je Call Graph profilování, vylepšené uživatelské rozhraní, ukládání a nahrávání, podpora GCC a 64-bit.

Podporuje nativní Windows aplikace, pokud mají standart PDB nebo debug informace DWARF2. Není nutná další kompilace aplikace – pouze se připojí k aplikaci v jejím průběhu. [8]

3.2.3 Valgrind

Valgrind distribuce v současnosti zahrnuje šest produkci zkvalitňujících nástrojů: detektor paměťových chyb, dva detektory chyb vláken, vyrovnávací paměť, analyzátor pro předpověď větve, vyrovnávací paměti pro generování grafu a analyzátor haldy. Obsahuje také tři experimentální nástroje: haldu (zásobník), globální pole s detektorem přetečení, druhý analyzátor haldy, která zkoumá, jak jsou používány části haldy a SimPoint generátor základní části vektoru. Běží na těchto platformách: X86/Linux, AMD64/Linux, ARM / Linux, PPC32/Linux, PPC64/Linux, S390X/Linux, ARM / Android (2.3.x), X86/Darwin a AMD64/Darwin (Mac OS X 10.6 a 10.7). Valgrind je Open Source a je šířen pod licencí GPL v2. [9]

Jednou ze součástí této distribuce je Callgrind. Je to profilovací nástroj obdobný nástroji gprof, ale je schopen získat více detailních informací o běhu programu.

Callgrind generuje graf volání funkcí za běhu programu. Shromážděná profilovací data mohou být uložena do výstupního souboru vícekrát v běhu programu nebo samostatně pro každé vlákno v případě více vláknového kódu. Pro interaktivní kontrolu a řízení získaných údajů mohou být zobrazeny grafickou vizualizací v nástroji KCachegrind. [10]

4 PROFILOVÁNÍ POMOCÍ NÁSTROJE GPROF

Principiálně podobný způsob sběru dat jako ruční doplňování kódu o pomocné výpisy nabízejí samotné překladače. Budeme se zabývat jazykem C a C++ a bude použit překladač GCC a nástroj *gprof*. [11]

4.1 Použití nástroje gprof

Při použití překladače GCC je nutné přeložit projekt s přepínačem *-p*, resp. *-pg*. Překladač pak do výsledné aplikace doplní kód, který po spuštění programu generuje plošný profil (Flat profile) a graf volání (Call Graph) a uloží jej do souboru *gmon.out*. Pomocí programu *gprof* se pak zobrazí výsledky analýzy v čitelné formě v textovém editoru. [11] Popis výstupu z programu *gprof* bude představen v následující kapitole.

4.2 Popis výstupu profileru

Získaná data se dají otevřít v textovém editoru s pomocí spuštění následujícího příkazu v příkazové řádce s parametry aplikace *test.exe* a souborem *gmon.out*. Příkaz pomocí programu *gprof* zapíše proud dat do souboru *out.txt*.

Příklad příkazu spuštění programu *gprof* v příkazové řádce lze napsat takto:

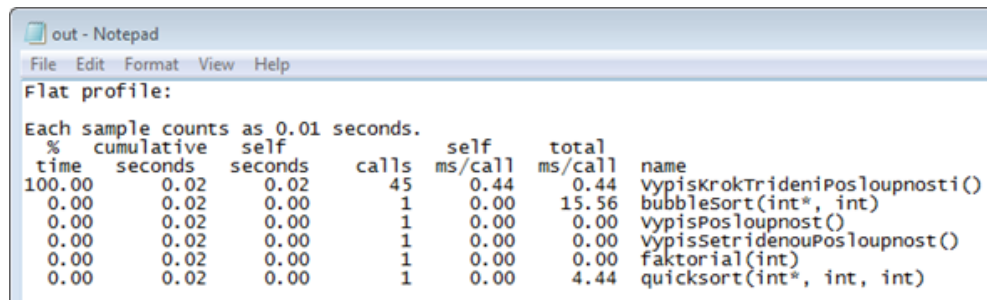
```
gprof.exe test.exe gmon.out>out.txt.
```

Obsah souboru *out.txt* lze rozdělit na dvě části:

- plošný profil,
- graf volání.

4.2.1 Plošný profil

Plošný profil ukazuje celkový čas strávený ve funkci. Na Obrázek 1 je ukázka výstupu plošného profilu ze souboru *out.txt*. Funkce jsou seřazeny sestupně dle procentuálního času stráveného ve funkcích při běhu programu.



```

out - Notepad
File Edit Format View Help
Flat profile:
Each sample counts as 0.01 seconds.
%time cumulative self      calls self    total   name
   time seconds seconds      ms/call ms/call ms/call
100.00    0.02    0.02        45      0.44    0.44  vypiskKrokTrideniPosloupnosti()
 0.00     0.02    0.00         1      0.00   15.56  bubbleSort(int*, int)
 0.00     0.02    0.00         1      0.00    0.00  vypisPosloupnost()
 0.00     0.02    0.00         1      0.00    0.00  vypisSetridenouPosloupnost()
 0.00     0.02    0.00         1      0.00    0.00  faktorial(int)
 0.00     0.02    0.00         1      0.00    4.44  quicksort(int*, int, int)

```

Obrázek 1: Plošný profil.

Základní popis parametrů je součástí výstupu souboru *out.txt*. Pro upřesnění významu jsou tyto údaje vysvětleny následovně:

- **%time** – procentuálně vyjádřený celkový čas, který program strávil ve funkci,
- **cumulative seconds** – celkový čas v sekundách, které program strávil v této funkci včetně součtu časů strávených ve funkcích volaných z této funkce,
- **self seconds** – čas v sekundách, které program strávil ve funkci. V tomto údaji nejsou započítány funkce, které jsou z této funkce dále volány,
- **calls** – počet volání funkce,
- **self ms/calls** – průměrný počet milisekund strávených při vykonávání kódu funkce při jednom zavolání,
- **total ms/calls** – průměrný počet milisekund strávených v této funkci při jednom zavolání. Údaj zahrnuje i čas strávený ve funkcích volaných z této funkce.

Při zpracování výstupních dat pro vizualizaci profilovacích informací překladače byl použit výstup *grafu volání* ze souboru *out.txt*.

4.2.2 Graf volání

Graf volání testovaného programu v souboru *out.txt* je nejdůležitější část výstupu, která je načtena a zpracována samotným zásuvným modulem *CallGraph* a vizualizována programem *dot* do podoby obrázku grafu volání funkcí.

out - Notepad

File Edit Format View Help

Call graph (explanation follows)

granularity: each sample hit covers 4 byte(s) for 50.00% of 0.02 seconds

index	% time	self	children	called	name
		0.00	0.00	10/45	quicksort(int*, int, int) [4]
		0.02	0.00	35/45	bubbleSort(int*, int) [3]
[1]	100.0	0.02	0.00	45	vypiskKrokTrideniPosloupnosti() [1]

		0.00	0.02	1	<spontaneous>
[2]	100.0	0.00	0.02	1/1	main [2]
		0.00	0.00	1/1	bubbleSort(int*, int) [3]
		0.00	0.00	1/1	quicksort(int*, int, int) [4]
		0.00	0.00	1/1	vypisPosloupnost() [6]
		0.00	0.00	1/1	vypisSetridenouPosloupnost() [7]
		0.00	0.00	1/1	faktorial(int) [8]

[3]	77.8	0.00	0.02	1/1	main [2]
		0.00	0.02	1	bubbleSort(int*, int) [3]
		0.02	0.00	35/45	vypiskKrokTrideniPosloupnosti() [1]

		0.00	0.00	9	quicksort(int*, int, int) [4]
[4]	22.2	0.00	0.00	1/1	main [2]
		0.00	0.00	1+9	quicksort(int*, int, int) [4]
		0.00	0.00	10/45	vypiskKrokTrideniPosloupnosti() [1]
				9	quicksort(int*, int, int) [4]

[6]	0.0	0.00	0.00	1/1	main [2]
		0.00	0.00	1	vypisPosloupnost() [6]

[7]	0.0	0.00	0.00	1/1	main [2]
		0.00	0.00	1	vypisSetridenouPosloupnost() [7]

		0.00	0.00	16	faktorial(int) [8]
[8]	0.0	0.00	0.00	1/1	main [2]
				1+16	faktorial(int) [8]
				16	faktorial(int) [8]

Obrázek 2: Graf volání.

Základní popis parametrů je součástí výstupu souboru *out.txt*. Základní rozdělení vstupů (oblast zvýrazněná rámečkem) označuje jednotlivé funkce a jejich vztahy.

Rozdělení funkcí v jednotlivých vstupech:

- primární funkce, která má na levém okraji index v hranatých závorkách,
- volající funkce (rodiče), které zavolaly primární funkci,
- volané funkce (potomci), které primární funkce zavolala.

Uvedený popis dat v souboru *out.txt* poskytl základ pro programové řešení pluginu *CallGraph*.

Význam údajů pro primární funkce:

- **index** – unikátní číselný identifikátor přiřazený každé položce tabulky, jehož čísla jsou seřazena dle velikosti a index je uveden vedle jména všech funkcí,
- **%time** – procentuální vyjádření celkového času stráveného v primární funkci a jejich potomcích,
- **self** – celkový čas v sekundách strávený v primární funkci,

- **children** – celkový čas v sekundách strávený v potomcích primární funkce,
- **called** – počet volání funkce, jestliže funkce volá sama sebe, nejsou tato volání zahrnuta do této hodnoty a jsou uvedena za symbolem "+",
- **name** – jméno primární funkce následované indexem, jestliže je funkce součástí cyklu je číslo cyklu uvedeno mezi jménem funkce a indexem.

Význam údajů pro volající funkce (rodiče):

- **self** – celkový čas v sekundách strávený ve volající funkci,
- **children** – celkový čas v sekundách strávený v potomcích volající funkce,
- **called** – počet volání primární funkce rodičem, jestliže funkce volá sama sebe, nejsou tato volání zahrnuta do této hodnoty a jsou uvedena za symbolem "/",
- **name** – jméno rodiče následované indexem, jestliže je rodič součástí cyklu je číslo cyklu uvedeno mezi jménem funkce a indexem.

Význam údajů pro volané funkce (potomky):

- **self** – celkový čas v sekundách strávený ve volané funkci,
- **children** – celkový čas v sekundách strávený v potomcích volané funkce,
- **called** – počet volání potomka primární funkcí, jestliže funkce volá sama sebe, nejsou tato volání zahrnuta do této hodnoty a jsou uvedena za symbolem "/",
- **name** – jméno potomka následované indexem, jestliže je potomek součástí cyklu je číslo cyklu uvedeno mezi jménem funkce a indexem.

Pro sestavení grafu volání bylo v programovém řešení pluginu *CallGraph* využito údajů o primárních funkcích (uzel grafu) a volaných (hrana grafu) funkcích (potomcích). Tyto údaje postačili k sestavení popisných dat v jazyce DOT pro následné vygenerování grafu vzájemně se volajících funkcí.

Uvedený čas v sekundách je vypsaný na dvě desetinná místa. Vypsaná hodnota času 0.00 sekund je dána zaokrouhlením naměřených nízkých hodnot.

5 MĚŘENÍ VÝKONU APLIKACÍ

Při měření výkonu aplikací nebo algoritmů se v praxi setkáváme s několika faktory ovlivňující výkon nebo efektivnost programu nebo algoritmu. V následující kapitole budou představeny některé z metod měření výkonu aplikací nebo algoritmů.

5.1 Způsoby měření

Při měření výkonu aplikací nebo algoritmů je rozhodující rychlost vykonávání algoritmu, požadované hardwarové zdroje a za jak dlouho bude k dispozici výsledek. V závislosti na ostatních činnostech nás zajímá:

- paměťová náročnost,
- potřebný čas.

Mezi faktory ovlivňující výkon programu lze zahrnout následující:

- velikost a pořadí vstupních dat,
- efektivnost a vhodnost algoritmu funkcí,
- počet opakování funkcí v programu.

Efektivnost a vhodnost algoritmu funkcí a počet opakování funkcí programu, může záviset na velikosti vstupních dat – algoritmus, který je efektivní pro malou množinu vhodně uspořádaných vstupních prvků, může být při náročnějším zadání nevyhovující. Efektivnost a vhodnost algoritmu funkcí a počet opakování funkcí programu jsou měřeny, aby byly optimalizovány k dosažení lepších výsledků. Důležité je i správné vyhodnocení měření vedoucí k nalezení správného místa pro optimalizaci. Zvýšení efektivnosti algoritmu jedné funkce může snížit počet volání funkce jiné a naopak.

Graf volání funkcí získaný při profilování běhu aplikací v podobě, který generuje plugin *CallGraph*, provádí praktické měření výkonu aplikace nebo algoritmu.

Naměřené údaje v grafu volání funkcí:

- procentuálně vyjádřený celkový čas, který program strávil ve funkci a jeho potomcích,
- naměřený vlastní čas v sekundách, který program strávil v samotné funkci,
- počet volání funkce.

Získané údaje nám pomohou lépe pochopit chování programu na základě, kterých je možné určit nebo odhadnout vhodnost, rychlost některých použitých algoritmů.

Propojení uzlů a vzájemné volání funkcí:

- rekurzivní volání,
- cyklické volání.

Na základě naměřených hodnot a propojení uzlů je možné odhadnout asymptotickou složitost, která odpovídá náročnosti výpočtu v závislosti na velikosti vstupních dat.

Druhy asymptotické složitosti:

- konstantní složitost,
- lineární složitost,
- kvadratická složitost,
- exponenciální složitost,
- logaritmická složitost.

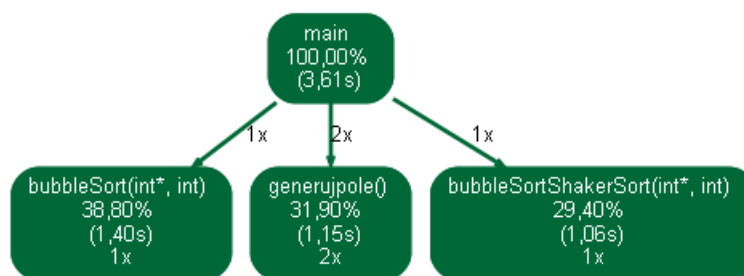
5.2 Cíl měření

Díky vizualizaci běhu aplikace pomocí grafu volání funkcí dostaneme její praktickou složitost na základě změřeného procentuálního času běhu programu a počtu volání jednotlivých funkcí. Přehledné grafické znázornění pomůže k snadnému odhalení potenciálního „úzkého hrdla aplikace“ a také k odhadnutí asymptotické složitosti aplikace.

Příklad grafu volání funkcí demonstruje čas strávený ve funkci při třídění 20000 prvků v poli pomocí algoritmů Bubble sort [12] a Shaker sort [13]. Oba uvedené algoritmy mají kvadratickou složitost. V případě algoritmu Shaker sort se jedná o vylepšený algoritmus Bubble sort s řazením prvků v obou směrech. Při velkém počtu prvků, je zřejmé, že rychlejší je vylepšený algoritmus Shaker sort, který potřeboval méně času pro setřídění prvků v poli.

Při optimalizaci programu je možné se zaměřit na následující možnosti:

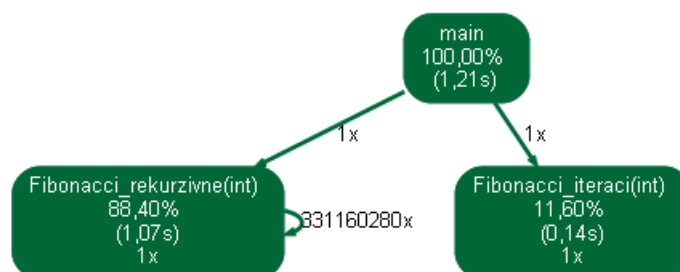
- znalost rozsahu řešeného problému,
- znalost složitosti algoritmu,
- optimalizace počtu volání funkcí.



Obrázek 3: Porovnání rychlosti algoritmů třídění.

Z uvedeného grafu lze rychle porovnat velikosti uvedených celkových časů algoritmů vzhledem k funkci *main*, ze které je celkový 100 % čas (3,61 sekund) běhu program rozdělen na volané (potomky) funkce (algoritmy) třídění a generování pole. Algoritmus Bubble sort potřeboval 38,80 % a algoritmus Shaker sort potřeboval 29,40 % celkového času běhu programu. Zbývající čas byl potřebný pro funkci *generujpole()*, která byla v programu volána dvakrát. Údaje časů v kulatých závorkách představují naměřený spotřebovaný čas samotnou funkcí v sekundách zaokrouhlený na dvě desetinná místa.

V následujícím příkladu je cílem snížit počet volání funkce odstraněním rekurze v algoritmu součtu členů Fibonacciho posloupnosti [14].



Obrázek 4: Součet 40-ti členů Fibonacciho posloupnosti.

Odstranění rekurze v uvedeném algoritmu je řešeno s pomocí iterace. Způsob převedení rekurze do iterační podoby je založen na ukládání již vypočítaných předchozích hodnot, které jsou nutné pro další výpočet.

Z výsledného grafu volání funkcí a hodnot časů strávených ve funkcích, které měří obě varianty výpočtů je zřejmé zvýšení výkonu algoritmu v iterační podobě. V iterační podobě potřeboval algoritmus méně času pro získání výsledku.

6 VIZUALIZAČNÍ NÁSTROJE

Při vývoji práce byl použit vizualizační nástroj, který bude popsán v následujícím textu. Vizualizace profilovacích informací může být řešena různou formou výstupu. Klasickým zápisem může být tabulka s přehledem funkcí s jejich statistickými údaji. Takové řešení je sice přehledné, ale ke snadnějšímu pochopení chování programu není ideální. Optimální řešení, které nabízí programové řešení této práce je zobrazení grafu. Graf představuje uzly propojenými hranami, které znázorňují postupné volání jednotlivých funkcí běžícího programu.

6.1 GraphViz

Tento volně dostupný nástroj nabízí velké možnosti výstupu dle zvolení konkrétního výstupního programu z celého balíčku. Nástroj poskytuje možnost nastavení pro barvy, fonty, rozložení uzlů včetně velikosti, styly uzlů a čáry, hypertextové odkazy, definice vlastních tvarů prvků. V praxi jsou grafy obvykle generovány z externích dat nebo můžou být upravovány v textovém nebo grafickém editoru. [15]

6.1.1 Program DOT

V rámci diplomové práce byl použit program *dot*. Tento nástroj generuje z popisu grafu v jazyce DOT základní hierarchickou strukturu grafu. Program *dot* má široké možnosti nastavení výsledných formátů výstupu, například obrázků ve formátu *.png, *.jpg, *.pdf. Parametry programu se dají pohodlně řídit z příkazové řádky. Příklad příkazu, který načte zápis v jazyce DOT v souboru dot.txt a vytvoří obrázek ve formátu *.png může vypadat takto: `dot.exe -Tpng -o dot.png dot.txt`

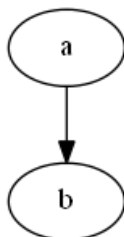
6.1.2 Jazyk DOT

Pro obecný popis grafu byl použit jazyk DOT. Jedná se o popisný kód grafu, uzlů a propojení hran. Kompletní zápis pomocí složité gramatiky je k dispozici na online odkazu [16]. Pro základní vysvětlení popisného kódu budou dostačovat uvedené příklady. Složitost zápisu uvedených příkladů je stupňována, tak aby byl ukázán základní princip popisného kódu jazyka DOT. Hlavní výhodou je jeho přehlednost, jednoduchost a rychlost s jakou se dá dosáhnout požadovaných výsledků.

Uvedený kód v jazyce DOT byl napsaný a spuštěný v editoru GVEEdit pro *GraphViz*. [15]

Příklad zápisu jednoduchého grafu s definicí hrany mezi uzly a, b:

```
digraph graf
{
a -> b;
}
```

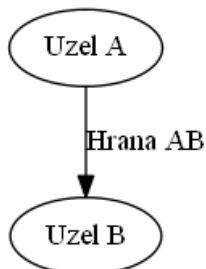


Obrázek 5: Jednoduchý graf.

Příklad zápisu včetně definice uzlů s jejich popisem a definice hran s jejich popisem:

```
digraph graf
{
/* definice uzlů */
a [label="Uzel A"];
b [label="Uzel B"];
/* definice hran */
a->b [label="Hrana AB"];
}
```

Ukázka výstupu:



Obrázek 6: Jednoduchý graf s popisem hrany.

Příklad zápisu použitého v souboru popisující graf volání funkcí, který generuje plugin *CallGraph*:

digraph

```
{
graph [ranksep="0.25", fontname=Arial, nodesep="0.125"];

node [label="\N", fontsize="10.00", fontname=Arial, style="filled, rounded", height=0,
width=0, shape=box, fontcolor=white];

edge [fontname=Arial];

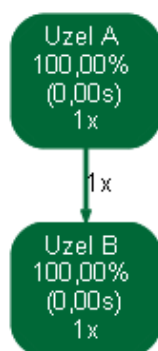
/* definice uzlu */

1 [label="Uzel A \n100,00% \n(0,00s)\n1x", fontcolor="white", color="#006837"];
2 [label="Uzel B \n100,00% \n(0,00s)\n1x", fontcolor="white", color="#006837"];

/* definice hran */

1-> 2 [color="#006837", label="1x", arrowsize="0.50", fontsize="10.00",
fontcolor="black", labeldistance="4.00", penwidth="2.00"];}
```

Ukázka výstupu:



Obrázek 7: Jednoduchý graf volání funkcí.

Klíčová slova jsou označena kurzívou, vlastnosti klíčových slov jsou v uvozovkách za rovnítkem.

Vysvětlení důležitých částí popisující graf zle rozdělit na popis definice grafu, uzlu a hrany.

- Popis definice grafu:

digraph	- definice grafu
{	- začátek definice grafu
graph [<i>ranksep</i> ="0.25",	- horizontální vzdálenost oddělující uzly
<i>fontname</i> =Arial,	- výchozí nastavení fontu grafu
<i>nodesep</i> ="0.125"];	- vertikální vzdálenost oddělující uzly
node [- začátek definice vlastností uzlu
<i>label</i> ="\N",	- výchozí popis uzlu
<i>fontsize</i> ="10.00",	- výchozí velikost fontu uzlu
<i>fontname</i> =Arial,	- výchozí nastavení fontu uzlu
<i>style</i> ="filled, rounded"	- styl vykreslení uzlu, vyplněný a zaoblený
<i>height</i> =0,	- výchozí velikost výšky uzlu
<i>width</i> =0,	- výchozí velikost šířky uzlu
<i>shape</i> =box,	- výchozí tvar uzlu
<i>fontcolor</i> =white	- výchozí barva písma uzlu
];	- konec definice vlastností uzlu
edge [
<i>fontname</i> =Arial	- výchozí nastavení fontu hrany
];	- konec definice vlastností hrany

- Popis uzlu:

1	- číslo uzlu
[- začátek definice uzlu
<i>label</i> ="Uzel A \n100,00% \n	
(0,00s)\n1x",	- vlastní text uzlu
<i>fontcolor</i> ="white",	- barva textu
<i>color</i> ="#006837",	- barva uzlu

]; - konec definice uzlu

- Popis hrany:

1-> 2 - hrana, definice propojení mezi uzly

[- začátek definice vlastností hrany

color="#006837", - barva hrany

label="1x", - vlastní text hrany

arrowsize="0.50", - velikost šipky

fontsize="10.00", - velikost fontu hrany

fontcolor="black", - barva textu

penwidth="2.00" - tloušťka čáry hrany

]; - konec definice vlastností hrany

} - konec definice grafu

II. PRAKTICKÁ ČÁST

7 UŽIVATELSKÁ DOKUMENTACE

Součástí této práce je uživatelská dokumentace, která pomůže s integrací do CodeLite. Dokumentace vysvětluje možnosti použití pluginu a popisuje jednotlivé nastavení údajů v dialogu.

7.1 Integrace do CodeLite

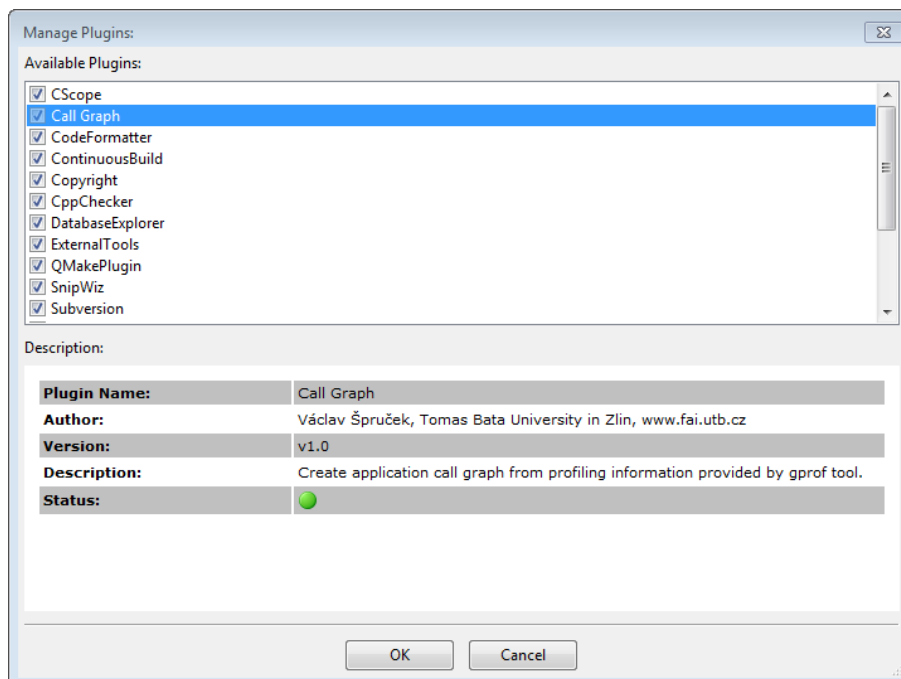
Instalaci zásuvného modulu a aktuální verzi CodeLite provede průvodce instalace. Průvodce byl vytvořen v programu *Inno Setup 5.4.3.*, který je volně dostupný na online odkazu [17]. Nástroj *Inno Setup* je možné použít pouze pro platformu Windows. Pro spuštění průvodce instalace zásuvného modulu a CodeLite s MinGW stačí spustit instalační soubor *codelite-3.5.5453-mingw4.4.1-CallGraph.exe*.

Instalace pro platformu Linux je řešena formou standardních balíčkovacích systémů nebo manuálním překladem ze zdrojových kódů.

V další verzi CodeLite bude plugin *Call Graph* součástí instalace celého vývojového nástroje. Zdrojové kódy dynamické knihovny *Call Graph* budou součástí zdrojových kódů projektu CodeLite a budou zveřejněny společně se zdrojovými kódy celého projektu na online odkazu [18], stažení zdrojových kódů je možné provést programem SVN [19], který slouží pro správu verzí projektu CodeLite.

Pro kompletní integraci zásuvného modulu je potřeba v programu CodeLite povolit knihovnu ve správci pluginů. Dialog se otevře volbou v menu *Plugins - > Manage Plugins*.

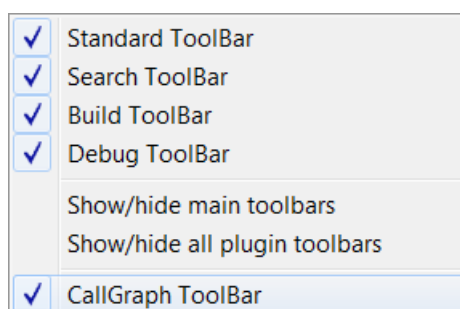
V dialogu je potřeba potvrdit volbu *Call Graph*, kde je základní popis pluginu, jméno autora a verze pluginu. Náhled dialogu zobrazuje Obrázek 8.



Obrázek 8: Dialog integrace pluginu.

Po potvrzení tlačítkem OK bude uživatel vyzván k restartování vývojového prostředí CodeLite.

Po opětovném otevření CodeLite bude potřeba povolit panel nástrojů pluginu volbou v menu *View -> Toolbars -> CallGraph ToolBar*.



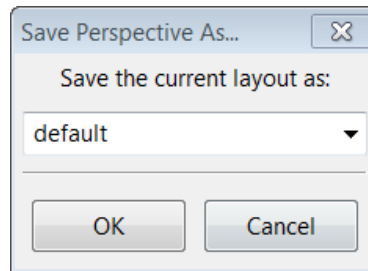
Obrázek 9: Výběr zobrazení panelu nástrojů.

Po povolení zobrazení panelu v menu se zobrazí v okně CodeLite nový panel nástrojů s jednou ikonou pro spuštění funkčnosti toho zásuvného modulu.



Obrázek 10: Základní panel pluginu Call Graph.

Změnu tohoto rozložení panelů nástrojů v okně vývojového prostředí CodeLite je výhodné uložit jako *Default* volbou v menu *Perspective - > Save Current Layout as...*

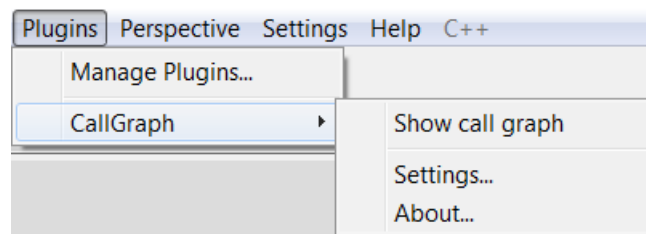


Obrázek 11: Uložení rozložení panelu jako default.

V případě problémů s rozmístěním panelů je možné toto nastavení rozložení panelů načíst volbou v menu *Perspective - > Restore Default Layout*.

7.2 Představení pluginu

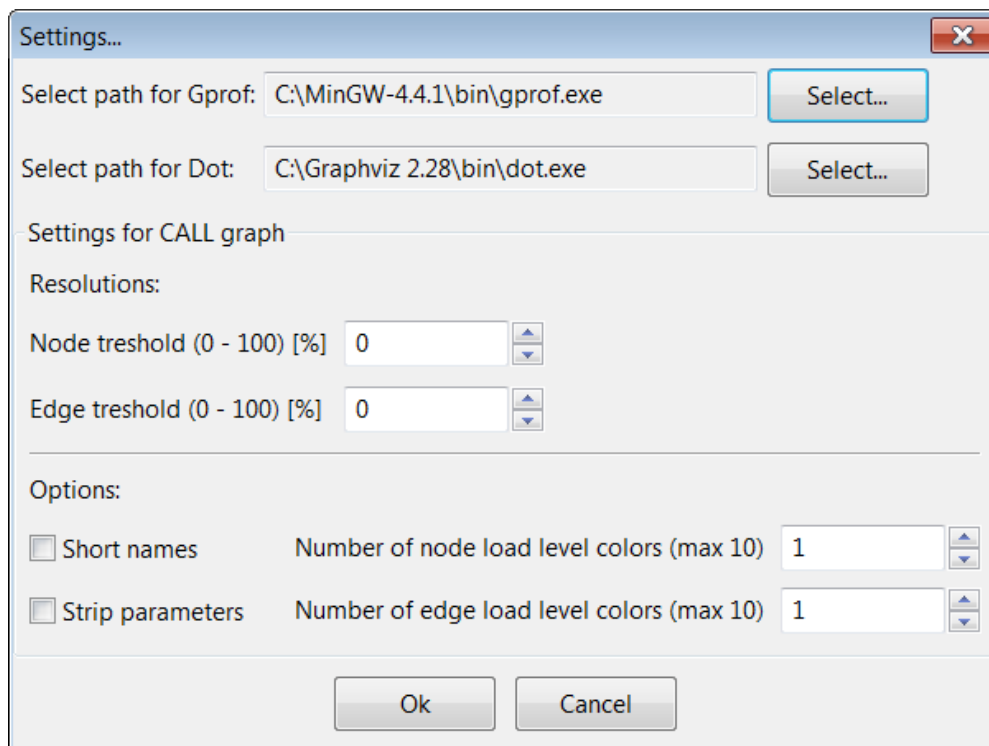
Rozhraní pluginu je přehledné a bez složitého nastavení. Uživatelské rozhraní pluginu tvoří dvě samostatná okna. První část má název dialogu *Settings...* a druhá část *About...*



Obrázek 12: Výběr rozhraní dialogu v menu CodeLite.

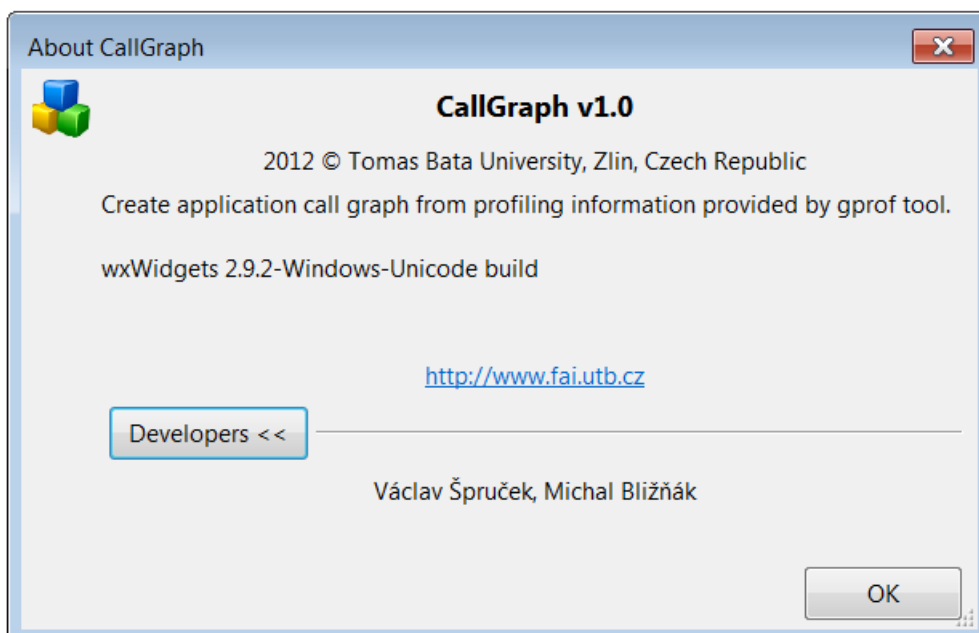
7.3 Uživatelské rozhraní

Dialog nastavení pluginu je zobrazen volbou v menu CodeLite výběrem možnosti *Plugins -> CallGraph -> Settings...*



Obrázek 13: Dialog základního nastavení pluginu.

Dialog *About CallGraph* je zobrazen volbou v menu *Plugins -> CallGraph -> About...*



Obrázek 14: Dialog o pluginu.

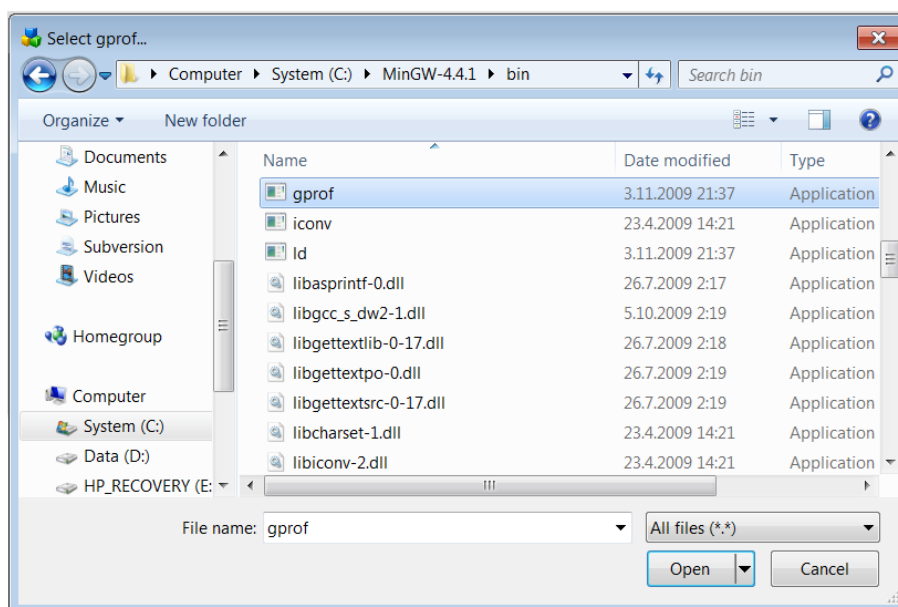
7.3.1 Popis nastavení dialogu

Dialog *Settings...* zobrazený na Obrázek 13 je rozdělen do tří částí:

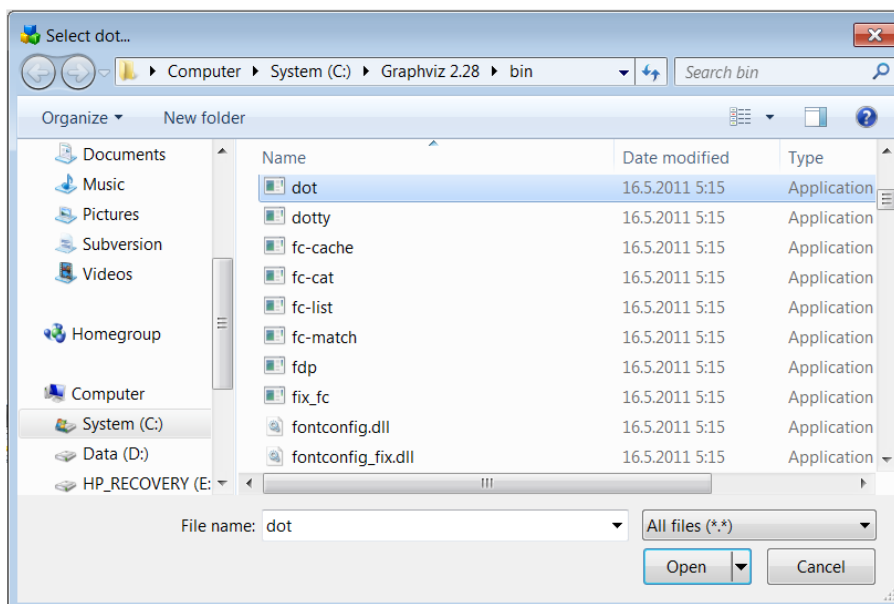
- výběr programu *gprof* a *dot*,
- nastavení rozlišení grafu dle vytížení uzlů a hran,
- nastavení zobrazení názvů uzlů a obarvení uzlů a hran.

První část slouží k výběru programu *gprof* a *dot*. Výběr jednotlivých programu je zobrazen na následujících obrázcích. Tento výběr je nutný na operačním systému Windows a to i v případě, že jsou oba programy nainstalovány. V systému Linux je možné tento krok přeskočit pokud jsou oba tyto programy nainstalovány, plugin si je najde v instalovaných programech sám.

Instalace těchto dvou programů je nutná pro správnou funkčnost pluginu. Stažení instalačního balíčku překladače *MinGW* je možné z online odkazu [20]. Stažení instalačního balíčku vizualizačního programu *GraphViz* je možné z online odkazu [15].



Obrázek 15: Výběr programu *gprof*.



Obrázek 16: Výběr programu dot.

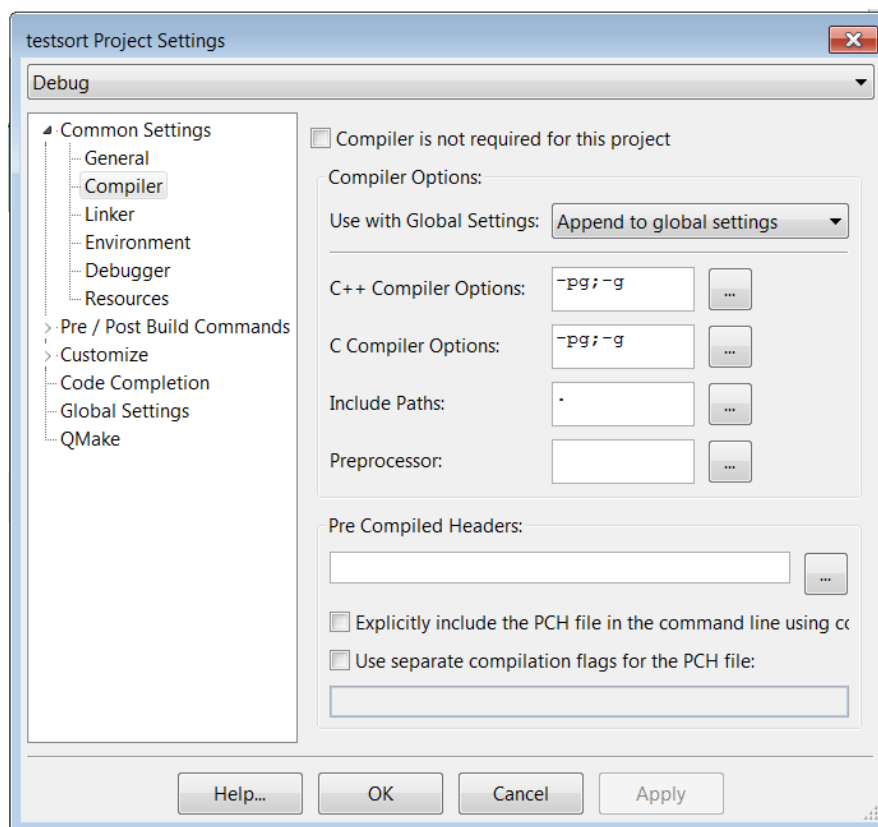
Ve druhé části nazvané *Resolution*, které nastavuje rozlišení grafu dle vytížení uzlů a hran. Toto nastavení filtruje zobrazení uzlů nebo propojení hran v grafu volání funkcí.

Ve třetí části nazvané *Options*, které nastavuje způsob zobrazení názvu uzlů a obarvení uzlů a hran určitým počtem barev dle vytížení uzlů nebo hran. Ukázky výstupu budou popsány v následující kapitole.

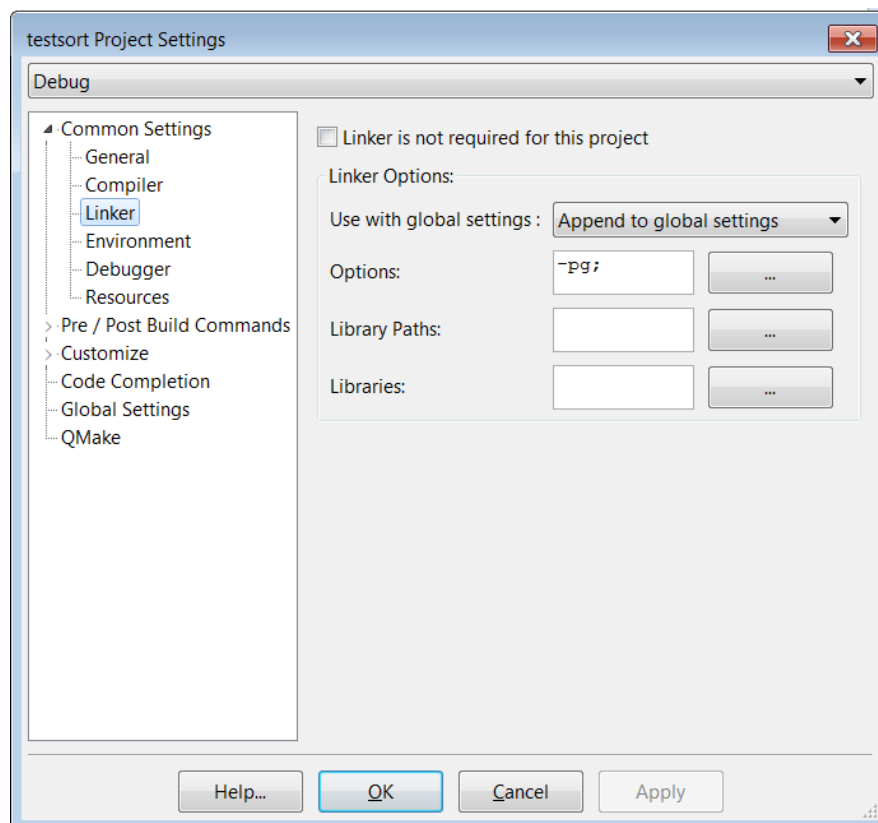
7.4 Práce s pluginem

Pro správnou funkčnost pluginu je potřeba nastavit cestu k programům *gprof* a *dot*. Program *gprof* je součástí instalace překladače *MinGW* a program *dot* je součástí instalace vizualizačního balíčku *GraphViz*.

Důležité je překládat a spouštět aktivní projekt s přepínačem *-pg*. Tento přepínač zajistí uložení profilovacích dat do souboru *gmon.out*. Nastavení přepínače je potřeba provést v dialogu *Project Settings* kontextovou volbou *Settings...* hlavního projektu a dle Obrázků 17 a 18 je důležité dopsat přepínač *-pg* pro *Compiler* i *Linker*.

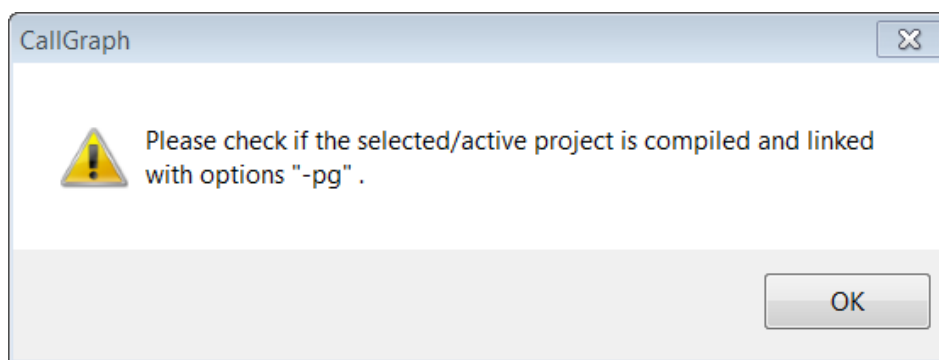


Obrázek 17: Doplnění pro Compiler.



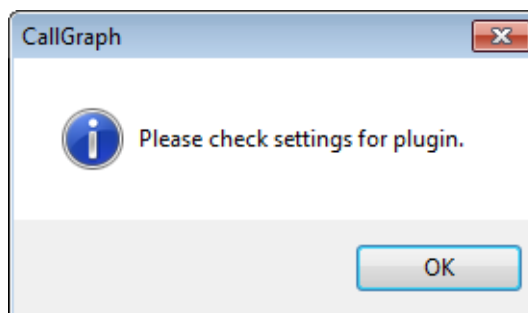
Obrázek 18: Doplnění pro Linker.

V opačném případě nebude soubor *gmon.out* v adresáři *Debug* aktivního projektu vytvořen. Tento soubor obsahuje profilovací informace od překladače GCC a je pluginem a programem *gprof* načten jeho obsah a vizualizován pomocí programu *dot*. Pokud není přepínač *-pg* nastaven bude uživatel upozorněn následujícím dialogem.



Obrázek 19: Dialog upozorňující na nastavení přepínače aktivního projektu.

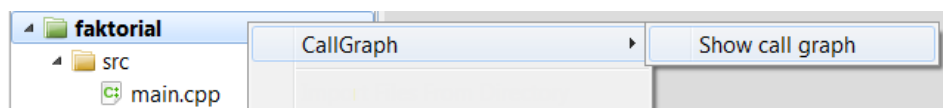
Pokud nebudou nastaveny cesty pro programy *gprof* a *dot* bude uživatel upozorněn následujícím dialogem.



Obrázek 20: Dialog upozornění na nastavení pluginu.

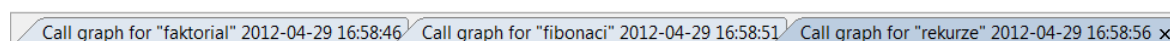
Po správném nastavení je možné spustit vizualizaci profilovacích informací aktivního nebo vybraného projektu třemi uvedenými způsoby:

- ikonou z panelu nástrojů (Obrázek 10),
- volbou v menu CodeLite -> *Plugins* -> *CallGraph* -> *Show Call Graph* (Obrázek 12),
- kontextovou nabídkou vybraného nebo aktivního projektu (Obrázek 21).



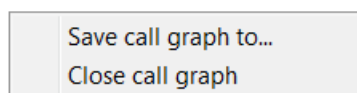
Obrázek 21: Kontextová nabídka vybraného nebo aktivního projektu.

Po provedení vizualizace profilovacích informací v prostředí CodeLite se zobrazí záložka s grafem volání funkcí pro aktivní nebo vybraný projekt včetně data a času zobrazení. Záložky s grafem volání funkcí jsou řazeny za sebou viz Obrázek 22.



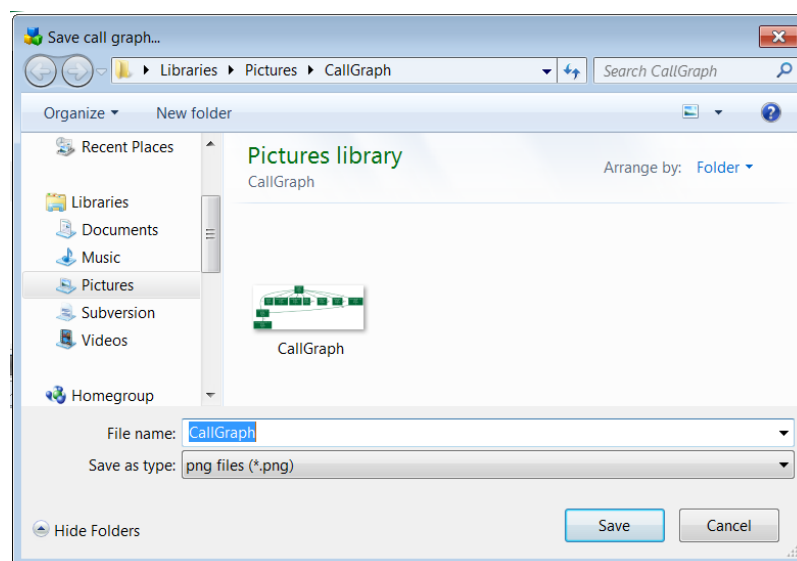
Obrázek 22: Nadpisy záložek příslušných grafů volání aktivních projektů.

Obrázek s grafem je možné pomocí kontextové nabídky (Obrázek 23) uložit volbou *Save call graph to...* a nebo zavřít aktivní záložku volbou *Close call graph*.



Obrázek 23: Kontextová nabídka záložky.

Uložení grafu volání funkcí jako obrázek s výchozím názvem souboru *CallGraph* je možné pouze ve formátu **.png*.



Obrázek 24: Uložení obrázku s grafem.

Následující upozornění se zobrazí při nastavení hodnoty filtrování uzlů tak, že budou vyfiltrovány všechny uzly a výsledný graf bude prázdný.

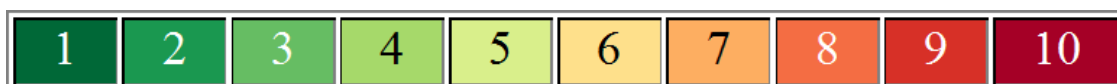
Call Graph is empty, please check settings of node resolutions for this plugin!

Obrázek 25: Prázdný graf.

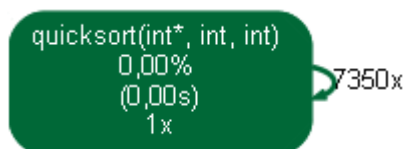
7.5 Ukázky výstupu

Obarvení hran a uzlů grafu je možné maximálně deseti barvami s přechodem od zelené k červené. Barvení zohledňuje vytížení uzlu a počet barev, které se mají použít pro obarvení grafu. Pro vytížení 0 % je uzel a hrana obarvena zelenou barvou. Při 100% vytížení uzlu a počtu barev dvě je tento uzel obarven červenou barvou. Pro více barev je základní interval 0 až 100 % rozdělen na menší intervaly, do kterých je přiřazeno vytížení uzlu. Minimální hodnota intervalu je 10 % pro jednu barvu. Obrázek 26 ukazuje přechod barvy od zelené k červené. Vzhledem ke zvolené barvě je i vhodně obarven text uzlu na výchozí bílou, která je pro lepší čitelnost ve světlejším odstínu změněna na černou.

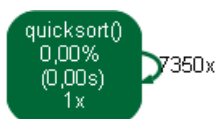
V nastavení dialogu pluginu (Obrázek 13) je volba pro úpravu názvu funkcí v uzlech grafu volání. K dispozici jsou tři možnosti, které jsou znázorněny na Obrázek 27 až Obrázek 29.



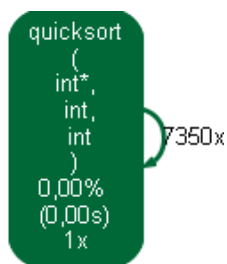
Obrázek 26: Barevné schéma obarvení uzlů a hran.



Obrázek 27: Výchozí nastavení zobrazení uzlu.



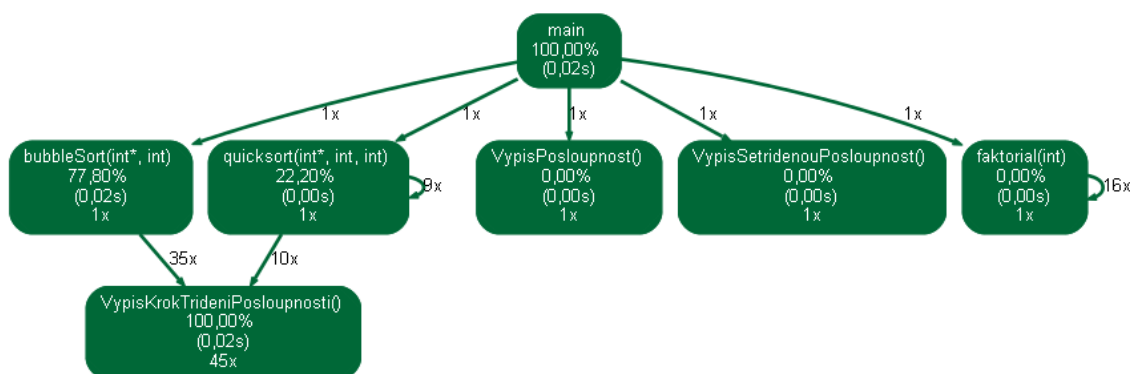
Obrázek 28: Funkce bez vnitřních parametrů.



Obrázek 29: Parametry funkce řazeny pod sebe.

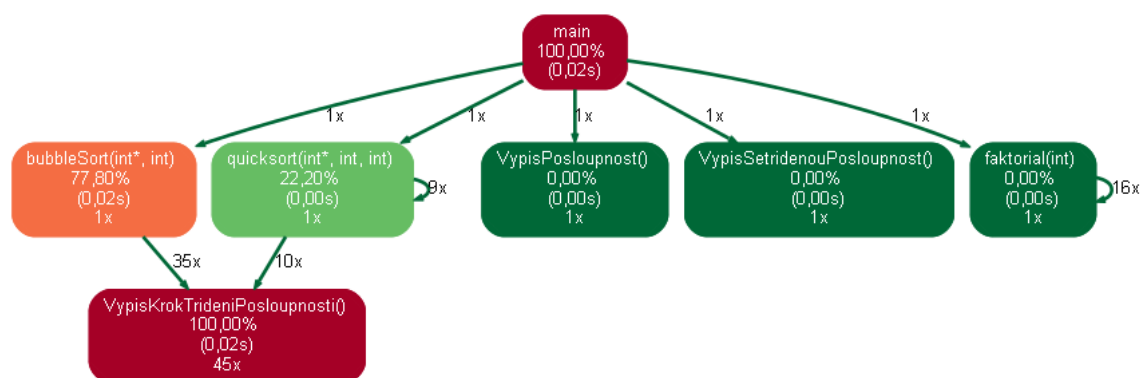
Význam hodnot v uzlu:

- název funkce,
- procentuálně vyjádřený celkový čas, který program strávil ve funkci a jeho potomcích,
- naměřený strávený čas v sekundách zaokrouhlený na dvě desetinná místa,
- počet volání funkce.



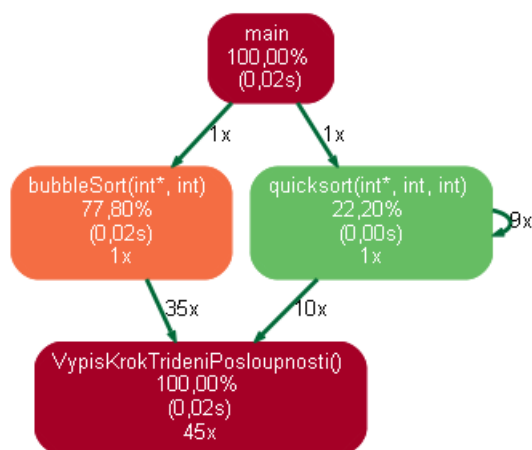
	Function name	Total time (%)	Self time (s)	Called
1	VypisKrokTrideniPosloupnosti()	100,00	0,02	45
2	main	100,00	0,02	1
3	bubbleSort(int*, int)	77,80	0,02	1
4	quicksort(int*, int, int)	22,20	0,00	10
5	VypisPosloupnost()	0,00	0,00	1
6	VypisSetridenouPosloupnost()	0,00	0,00	1
7	faktorial(int)	0,00	0,00	17

Obrázek 30: Ukázka grafu volání obarveným jednou barvou s tabulkou.



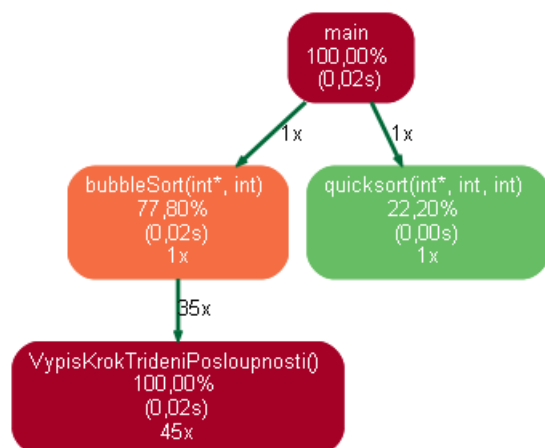
	Function name	Total time (%)	Self time (s)	Called
1	VypisKrokTrideniPosloupnosti()	100,00	0,02	45
2	main	100,00	0,02	1
3	bubbleSort(int*, int)	77,80	0,02	1
4	quicksort(int*, int, int)	22,20	0,00	10
5	VypisPosloupnost()	0,00	0,00	1
6	VypisSetridenouPosloupnost()	0,00	0,00	1

Obrázek 31: Ukázka obarveného grafu volání dle vytižení uzlů.



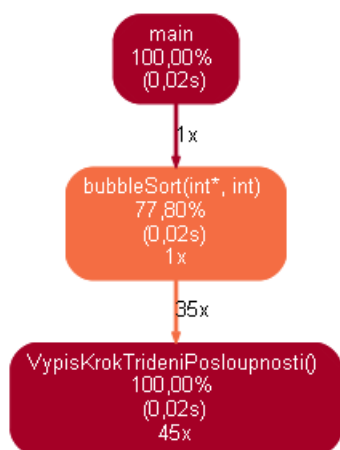
	Function name	Total time (%)	Self time (s)	Called
1	VypisKrokTrideniPosloupnosti()	100,00	0,02	45
2	main	100,00	0,02	1
3	bubbleSort(int*, int)	77,80	0,02	1
4	quicksort(int*, int, int)	22,20	0,00	10

Obrázek 32: Ukázka grafu volání s vyfiltrovanými uzly.



	Function name	Total time (%)	Self time (s)	Called
1	VypisKrokTrideniPosloupnosti()	100,00	0,02	45
2	main	100,00	0,02	1
3	bubbleSort(int*, int)	77,80	0,02	1
4	quicksort(int*, int, int)	22,20	0,00	10

Obrázek 33: Ukázka grafu volání s vyfiltrovanými uzly a hranami.



	Function name	Total time (%)	Self time (s)	Called
1	VypisKrokTrideniPosloupnosti()	100,00	0,02	45
2	main	100,00	0,02	1
3	bubbleSort(int*, int)	77,80	0,02	1

Obrázek 34: Ukázka grafu volání, který tak představuje „úzké hrdlo“ aplikace.

8 PROGRAMOVÁ DOKUMENTACE

Programová dokumentace byla vytvořena pomocí průvodce *Doxygenwizard* dokumentačního systému *Doxygen*. [21] Zvolený systém nabízí velké možnosti nastavení generování programové dokumentace na vhodně okomentovaných zdrojových kódech včetně hlavního menu, úvodní stránky, panelu pro vyhledávání, seznamu tříd, diagramů a další. Jako zvolený výstupní formát byl použit HTML a HTML HELP. Dokumentaci je možné otevřít přes stránku *index.html* ve webovém prohlížeči nebo v souboru nápovědy *CallGraph_programing_manual.chm* otevřít v Microsoft HTML Help editoru.

8.1 Struktura projektu

Projekt pluginu obsahuje zdrojové soubory (*.cpp) a hlavičkové soubory (*.h) v jazyku C a C++. Dále obsahuje soubory návrhu grafického rozhraní (*.fbp) vytvořených v grafickém návrhovém nástroji wxFormBuilder dostupného z online odkazu. [22]

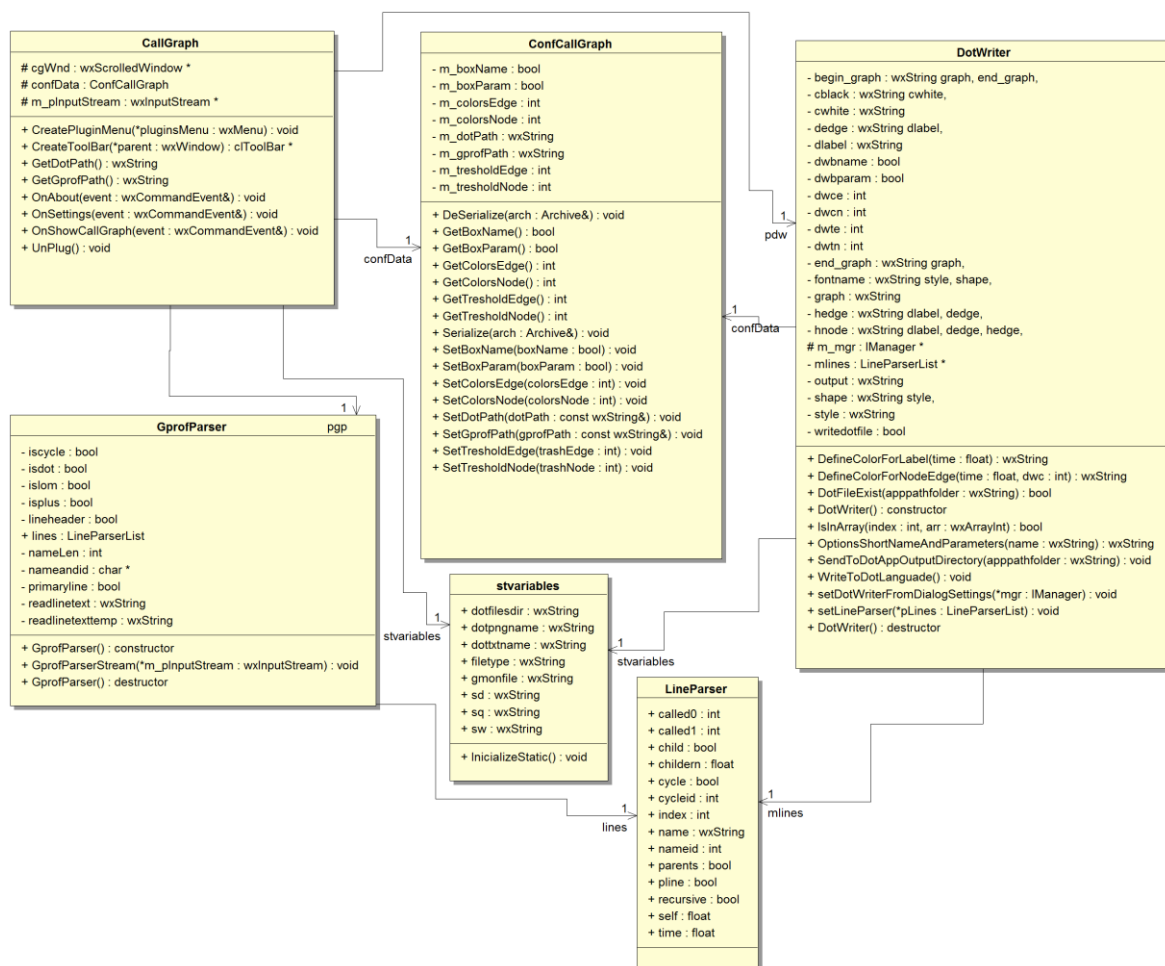
8.1.1 Popis tříd

Popis tříd je součástí programové dokumentace generované pomocí systému Doxygen. Je rozdělen na dvě skupiny:

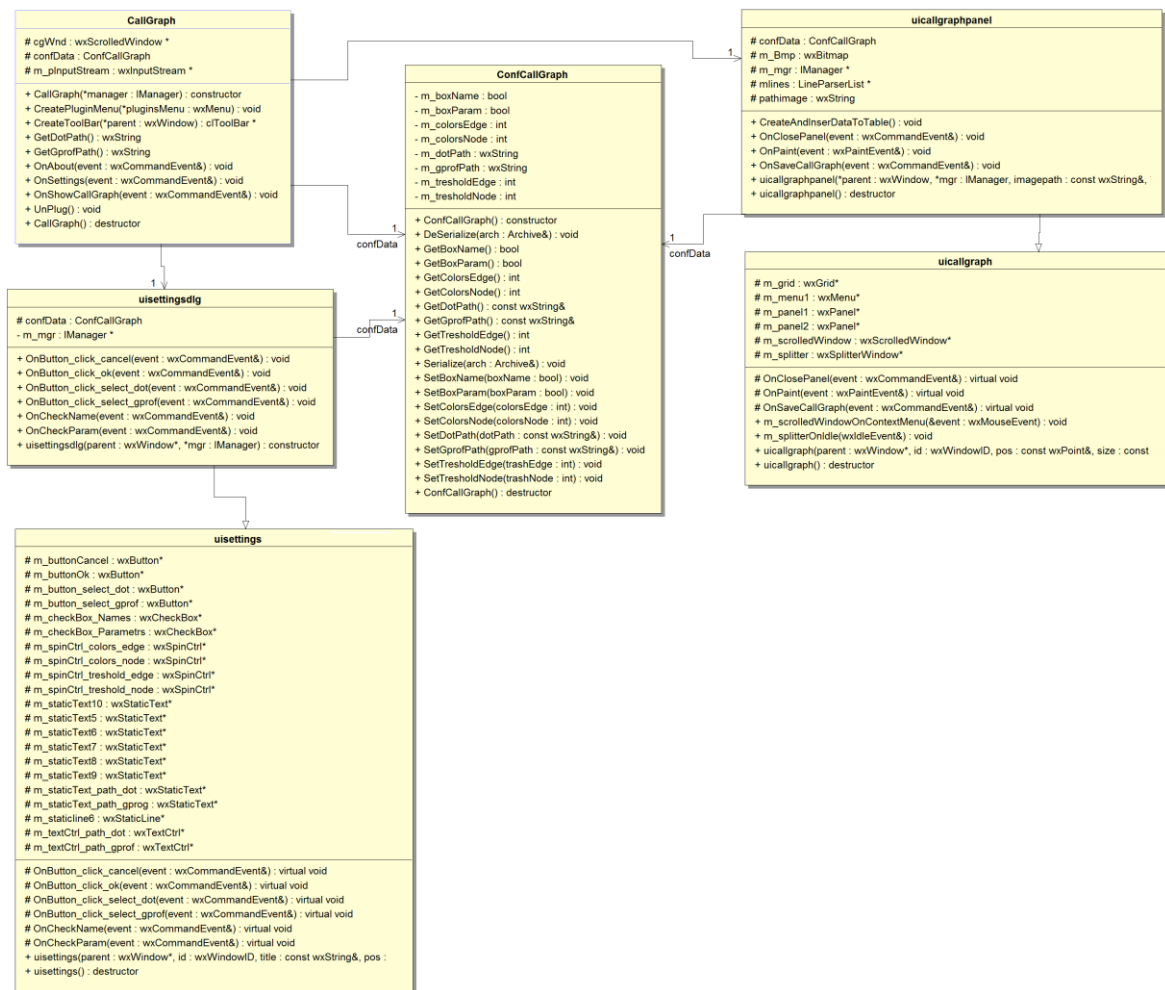
- Základní třídy:
 - CallGraph – definuje strukturu pluginu,
 - ConfCallGraph – definuje strukturu pro uložení nastavení dialogu,
 - DotWriter – načtená data převádí na zápis do jazyku DOT popisující graf,
 - GprofParser – ukládá proud dat od programu gprof do struktury LineParser,
 - LineParser – datová struktura pro uložení dat pro následné zpracování.
- Rozhraní pluginu:
 - uicallgraph – třída generovaná v návrhovém nástroji,
 - uicallgraphpanel – panel pro zobrazení výsledků,
 - uisettings – třída generovaná v návrhovém nástroji,
 - uisettingsdlg – dialog nastavení pluginu.

8.1.2 Diagram tříd

Diagram tříd byl vytvořen v nástroji CodeDesigner RAD dostupného z online odkazu [23]. Tento nástroj umožnil rychle a snadno vytvořit diagram tříd z načtených zdrojových kódů projektu *CallGraph*. Diagram bylo možné uložit jako obrázek ve formátu *.jpg, *.bmp, nebo *.png.



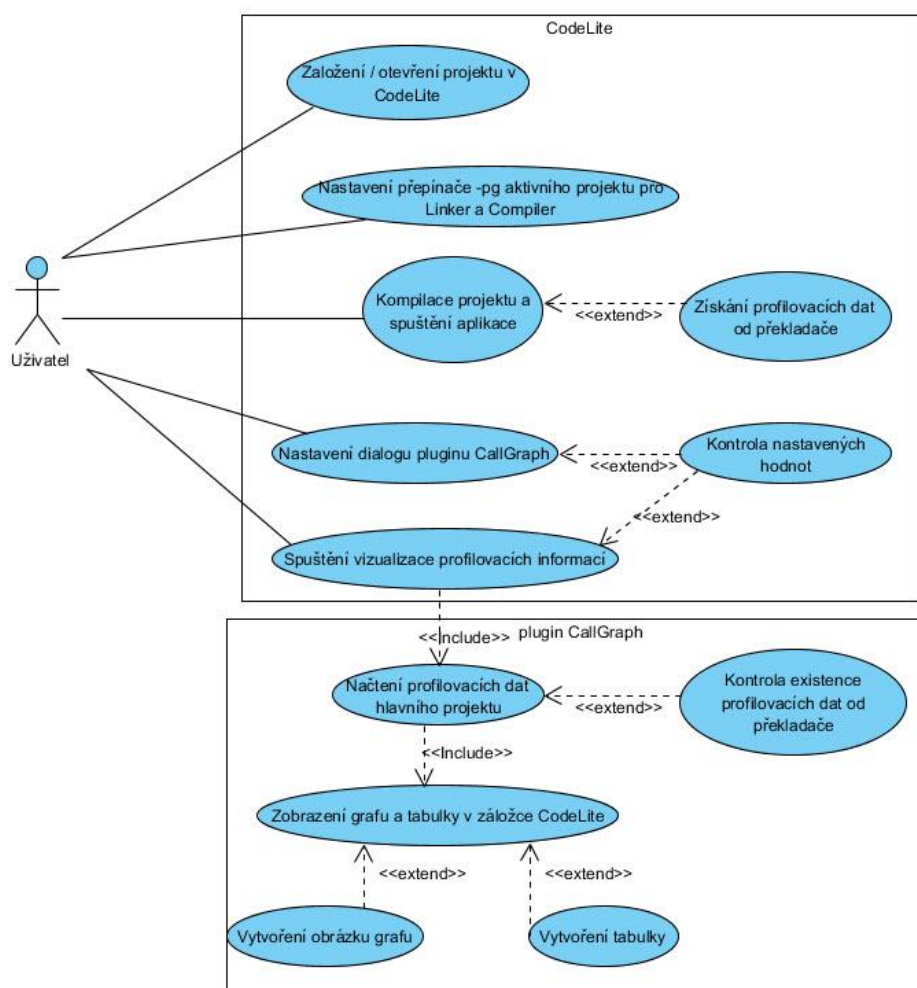
Obrázek 35: Diagram tříd jádra pluginu.



Obrázek 36: Diagram tříd rozhraní pluginu.

8.1.3 Příklad užití

Následující diagram reprezentuje způsob komunikace uživatele s vývojovým nástrojem CodeLite a s integrovaným pluginem *CallGraph*. Diagram ukazuje posloupnost akcí, které je potřeba provést v CodeLite a následně, které budou provedeny v pluginu *CallGraph*. K analýze chování systému byl použit modelovací jazyk UML.



Obrázek 37: Diagram případu užití.

8.1.4 Důležitý programový kód

Pro programové řešení jsou nejdůležitější následující části kódu, bez jejich implementace by nebylo možné dosáhnout požadované funkčnosti.

Pro získání proudu dat bylo potřeba spustit následující proces s příkazem parametrů v řetězci `cmdgprof` pro program `gprof`.

```

wxProcess gprofProcess;
gprofProcess.Redirect();
wxExecute(cmdgprof, wxEXEC_SYNC, &gprofProcess);
m_pInputStream = gprofProcess.GetInputStream();
    
```

Následně byl metodou `GprofParserStream` objektu `pgp` načten výstupní proud dat `m_pInputStream`.

```
GprofParser *pgp = new GprofParser();
if(m_pInputStream->CanRead())
{
    pgp->GprofParserStream(m_pInputStream);
}
```

Výstupní proud dat byl v objektu `pgp` převeden na text a po řádcích načítán jako řetězec do proměnné `readlinetext`.

```
wxTextInputStream text(*m_pInputStream);
wxString readlinetext = text.ReadLine();
```

Pro načtení dat ve třídě `GprofParser` byl použit příkaz `sscanf`. Načtené hodnoty byly ukládány do objektu `line` datové struktury `LineParser`.

Například pro načtení řádku:

```
[4] 3.7 0.00 0.01 1 6 quicksort(int*, int, int) [4],
```

bylo potřeba sestavit následující příkaz:

```
sscanf((const char*)readlinetext.mb_str(conv), "[%d] %f %f %f %d %d\n%s", &line->index, &line->time, &line->self, &line->children, &line->called0, &line->called1, nameandid);
```

Kódování musí být nastaveno:

```
wxCSSConv conv(wxT("ISO-8859-1"));
```

Z údajů uložených v listu `lines` je objektu `pdw` list předán v metodě `setLineParser`.

```
DotWriter *pdw = new DotWriter();
pdw->setLineParser(&(pgp->lines));
```

Další metoda předá údaje o nastavení pluginu, podle kterých se bude generovat popisný kód grafu v jazyce DOT.

```
pdw->setDotWriterFromDialogSettings(m_mgr);
```

Po předání nastavení je možné generovat popisný kód grafu v jazyce DOT v metodě `WriteToDotLanguage` objektu `pdw`, která postupně prochází list `lines` s údaji

načtenými z profilovacího souboru a nejprve generuje popis uzlů v jazyce DOT a následně popis hran v jazyce DOT.

Po dokončení generování popisného kódu grafu v jazyce DOT je zavolána další metoda `SendToDotAppOutputDirectory` objektu `pdw`, které je předána cesta k adresáři aktivního nebo vybraného projektu v CodeLite v proměnné `projectPathActive`, kde bude následně vytvořen adresář `CallGraph` pro uložení popisných dat grafu v jazyce DOT do souboru `dot.txt`.

Funkce `DotFileExist` objektu `pdw` kontroluje existenci souboru `dot.txt` a při splnění je spuštěn proces s příkazem parametrů v řetězci `cmddot` pro program *dot*.

```
if (pdw->DotFileExist(projectPathActive))
{
    wxProcess cmddotProcess;
    cmddotProcess.Redirect();
    wxExecute(cmddot, wxEXEC_SYNC, &cmddotProcess);
}
```

Vygenerovaný obrázek *dot.png* je následně zobrazen v záložce CodeLite s vygenerovanou tabulkou.

8.2 Výstupní data

Výstupní data získaná vizualizací grafu jsou uložena v adresáři *CallGraph* aktivního projektu.

8.2.1 Popis souborů

Do adresáře jsou zapsány dva soubory:

- `dot.txt` - popisný soubor grafu v jazyce DOT, který je možné v případě potřeby dodatečně upravovat v editoru GVEdit pro *GraphViz*,
- `dot.png` – obrázek grafu vygenerovaný aplikací *dot* na základě popisu v souboru *dot.txt*.

ZÁVĚR

V teoretické části bylo popsáno profilování běhu aplikací pomocí různých profilovacích nástrojů v několika vývojových nástrojích. Dále je v práci popsán výstup profilovacího nástroje Gprof překladače MinGW. Výsledky získané měřením pomocí nástroje Gprof jsou použity k zobrazení v podobě grafu generovaného nástrojem Dot z vizualizačního software GraphViz.

Plugin CallGraph, který je výsledkem programového řešení této práce provádí generování a zobrazení grafů volání funkcí v IDE CodeLite. Výsledný obrázek grafu slouží k lepšímu pochopení struktury analyzovaného programového kódu a k návrhu vhodného způsobu jeho optimalizace nebo vylepšení algoritmu.

Rozšíření funkcionality IDE CodeLite o plugin CallGraph poskytuje vývojářům a programátorům užitečný nástroj pro efektivní tvorbu programového kódu a optimalizaci volání funkcí běžící aplikace.

ZÁVĚR V ANGLIČTINĚ

The theoretical part of this work is describing profiling of running applications by different profiling tools and the output of the profiling tool Gprof – interpreter of MinGW. The result data collected by Gprof are used for visualizing of graphs generated by Dot tool from the visualizing software GraphViz.

Plugin CallGraph implemented in the practical (programming) part of this work is designed for displaying call-graphs of functions in IDE CodeLite. Graphically represented call-graph of functions serves for better understanding and optimizing of the analyzed source code structure or algorithms.

Plugin CallGraph as an extension of IDE CodeLite gives the advantage of useful tool for effective programming and optimizing to all IDE CodeLite developers.

SEZNAM POUŽITÉ LITERATURY

- [1] *Microsoft Visual Studio 2010: Vývoj od návrhu až po nasazení* [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://www.microsoft.com/cze/msdn/vstudio/2010/>
- [2] Welcome to NetBeans. [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://netbeans.org/index.html>
- [3] Codelite Main: Home page. [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://www.codelite.org/>
- [4] *GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF)* [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://gcc.gnu.org/>
- [5] About - wxWidgets [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://wxwidgets.org/about/>
- [6] Profilování a optimalizace. MARTÍNEK, David. *Fakulta informačních technologií VUT v Brně* [online]. 2011 [cit. 2012-05-06]. Dostupné z: <http://www.fit.vutbr.cz/~martinek/clang/profiling.html#profiling>
- [7] *VTune™ Amplifier XE 2011 from Intel - Intel® Software Network* [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/>
- [8] *Very Sleepy* [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://www.codersnotes.com/sleepy>
- [9] *Valgrind Home* [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://valgrind.org/>
- [10] Profilování a optimalizace. MARTÍNEK, David. *Fakulta informačních technologií VUT v Brně* [online]. 2011 [cit. 2012-05-06]. Dostupné z: <http://www.fit.vutbr.cz/~martinek/clang/profiling.html#callgrind>
- [11] Profilování a optimalizace. MARTÍNEK, David. *Fakulta informačních technologií VUT v Brně* [online]. 2011 [cit. 2012-05-06]. Dostupné z: <http://www.fit.vutbr.cz/~martinek/clang/profiling.html#gccgprof>
- [12] Bubble sort [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://www.algoritmy.net/article/3/Bubble-sort>
- [13] Shaker sort [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://www.algoritmy.net/article/93/Shaker-sort>
- [14] Fibonacciho posloupnost [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://www.algoritmy.net/article/116/Fibonacciho-posloupnost>

- [15] Graphviz - Graph Visualization Software [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://graphviz.org/>
- [16] The DOT Language [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://www.graphviz.org/doc/info/lang.html>
- [17] Inno Setup [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://www.jrsoftware.org/isinfo.php>
- [18] Codelite / trunk [online]. 2012 [cit. 2012-05-06]. Dostupné z: <https://codelite.svn.sourceforge.net/svnroot/codelite/trunk>
- [19] Subversion.tigris.org [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://subversion.tigris.org/>
- [20] MinGW [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://www.mingw.org/>
- [21] Doxygen manual [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://www.doxygen.nl/manual.html>
- [22] wxFormbuilder[online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://wxformbuilder.org/>
- [23] Welcome to the CodeDesigner RAD [online]. 2012 [cit. 2012-05-06]. Dostupné z: <http://www.codedesigner.org/>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

DOT Plain text graph description language.

GCC GNU Compiler Collection.

GNU General Public License.

GraphViz Graph Visualization Software.

IDE Integrated Development Environment.

MinGW Minimalist GNU for Windows.

PNG Portable Network Graphics.

SEZNAM OBRÁZKŮ

Obrázek 1: Plošný profil.	17
Obrázek 2: Graf volání.	18
Obrázek 3: Porovnání rychlosti algoritmů třídění.	22
Obrázek 4: Součet 40-ti členů Fibonacciho posloupnosti.	22
Obrázek 5: Jednoduchý graf.	24
Obrázek 6: Jednoduchý graf s popisem hrany.	24
Obrázek 7: Jednoduchý graf volání funkcí.	25
Obrázek 8: Dialog integrace pluginu.	30
Obrázek 9: Výběr zobrazení panelu nástrojů.	30
Obrázek 10: Základní panel pluginu Call Graph.	30
Obrázek 11: Uložení rozložení panelu jako default.	31
Obrázek 12: Výběr rozhraní dialogu v menu CodeLite.	31
Obrázek 13: Dialog základního nastavení pluginu.	32
Obrázek 14: Dialog o pluginu.	32
Obrázek 15: Výběr programu gprof.	33
Obrázek 16: Výběr programu dot.	34
Obrázek 17: Doplnění pro Compiler.	35
Obrázek 18: Doplnění pro Linker.	35
Obrázek 19: Dialog upozorňující na nastavení přepínače aktivního projektu.	36
Obrázek 20: Dialog upozornění na nastavení pluginu.	36
Obrázek 21: Kontextová nabídka vybraného nebo aktivního projektu.	37
Obrázek 22: Nadpisy záložek příslušných grafů volání aktivních projektů.	37
Obrázek 23: Kontextová nabídka záložky.	37
Obrázek 24: Uložení obrázku s grafem.	37
Obrázek 25: Prázdný graf.	38
Obrázek 26: Barevné schéma obarvení uzlů a hran.	38
Obrázek 27: Výchozí nastavení zobrazení uzlu.	38
Obrázek 28: Funkce bez vnitřních parametrů.	38
Obrázek 29: Parametry funkce řazeny pod sebe.	39
Obrázek 30: Ukázka grafu volání obarveným jednou barvou s tabulkou.	39
Obrázek 31: Ukázka obarveného grafu volání dle vytížení uzlů.	40
Obrázek 32: Ukázka grafu volání s vyfiltrovanými uzly.	40

Obrázek 33: Ukázka grafu volání s vyfiltrovanými uzly a hranami.	41
Obrázek 34: Ukázka grafu volání, který tak představuje „úzké hrdlo“ aplikace.	41
Obrázek 35: Diagram tříd jádra pluginu.	43
Obrázek 36: Diagram tříd rozhraní pluginu.	44
Obrázek 37: Diagram případu užití.	45