UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

DIRAC V PROSTREDÍ MICROSOFT WINDOWS: ADAPTÁCIA A VYSOKOVÝKONNÉ POČÍTANIE

DIPLOMOVÁ PRÁCA

4cc5e2f6-e6f0-484a-aa31-0e4cf906e17f

2015 **Bc. Ivan Hraško**

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

DIRAC V PROSTREDÍ MICROSOFT WINDOWS: ADAPTÁCIA A VYSOKOVÝKONNÉ POČÍTANIE

Diplomová práca

4cc5e2f6-e6f0-484a-aa31-0e4cf906e17f

Študijný program: Aplikovaná informatika (jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9 Aplikovaná informatika
Pracovisko (katedra, inštitút ...): KCH FPV – Katedra chémie
Vedúci diplomovej práce: doc. RNDr. Miroslav Iliaš, PhD.

Banská Bystrica 2015 **Bc. Ivan Hraško**

Univerzita Mateja Bela v Banskej Bystrici
Fakulta prírodných vied

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Ivan Hraško
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	9.2.9. aplikovaná informatika
Typ záverečnej práce:	Magisterská záverečná práca
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov: DIRAC v prostredí Microsoft Windows: adaptácia a vysokovýkonné počítanie

Anotácia: Vďaka systému CMake (www.cmake.org) má vedecký softvér DIRAC (diracprogram.org), používaný v operačných systémoch Linux, Unix a Mac, potenciál fungovať aj pod komerčným operačným systémom Microsoft Windows. Efektívne adaptovanie DIRACu pre MS Windows by rozšírilo možnosti vedeckého počítania aj pod týmto systémom. Znamená to programové zásahy do buildup systému DIRAC spolu s testovaním rôznych dostupných matematických knižníc i kompilátorov v rámci systému MS Windows.

Vedúci:	doc. RNDr. Miroslav Iliaš, PhD.
Katedra:	KCH FPV - Katedra chémie
Vedúci katedry:	doc. RNDr. Miroslav Medveď, PhD.
Dátum zadania:	26.03.2014
Dátum schválenia:	23.04.2015

prof. Ing. Miroslav Svítek, Dr.
garant študijného programu

ČESTNÉ VYHLÁSENIE

Čestne vyhlasujem, že diplomovú prácu som vypracoval samostatne pod odborným vedením vedúceho záverečnej práce doc. RNDr. Miroslava Iliaša, PhD. za použitia uvedených študijných materiálov a informačných zdrojov.

….....

Bc. Ivan Hraško

POĎAKOVANIE

Ďakujem za spoluprácu vedúcemu práce doc. RNDr. Miroslavovi Iliašovi, PhD. a všetkým ostatným vývojárom programu DIRAC.

ABSTRAKT

Bc. HRAŠKO, Ivan: DIRAC v prostredí Microsoft Windows: adaptácia a vysokovýkonné počítanie. [Diplomová práca]. Univerzita Mateja Bela v Banskej Bystrici. Fakulta prírodných vied. Katedra chémie. Vedúci práce: doc. RNDr. Miroslav Iliaš, PhD. Stupeň odbornej kvalifikácie: Magister v odbore Aplikovaná informatika. Banská Bystrica: FPV UMB, 2015. 79 s.

Témou a cieľom diplomovej práce je adaptovanie relativistického kvantovo-chemického programu DIRAC pre prostredie operačného systému Microsoft Windows. Adaptácia je vykonávaná prispôsobovaním a rozširovaním konfiguračných skriptov pre vykonanie kompilácie programu. Tieto zásahy sú urobené multiplatformovými nástrojmi CMake a Python. Súčasťou práce je i odskúšanie niektorých kompilátorov a matematických knižníc, ktoré by sa mohli s programom na systéme Windows využívať. Snahou je i zabezpečiť paralelný beh programu pomocou využitia niektorého z paralelných programovacích modelov, ktoré sú v DIRAC-u alebo v použitých knižniciach implementované.

Text práce sa delí do piatich kapitol. Prvá kapitola sa teoreticky zameriava na vysokovýkonné počítanie na paralelných architektúrach a paralelné programovacie modely. Druhá kapitola pojednáva o špecifikácii rozhrania pre zasielanie správ - MPI. Tu sú predstavené ciele, s ktorými špecifikácia vznikla, cieľové architektúry a aj niektoré z jej implementácií. Samostatná časť je venovaná i organizácii práce so zdrojovými kódmi a softvérovým nástrojom, ktoré boli využívané pri práci na programe DIRAC. Vo štvrtej kapitole je zosumarizovaná práca, ktorá bola na programe vykonaná. Posledná kapitola sa venuje priebežnej integrácii. Táto dôležitá činnosť pre zvýšenie kvality vývoja kódu DIRAC-u je zabezpečovaná pomocou služieb, ktoré ju automatizujú.

Výsledkom našej práce je vylepšenie použiteľnosti programu DIRAC v prostredí operačných systémov Microsoft Windows v 64-bitových verziách a stále i v starších, 32-bitových verziách. Podarilo sa odskúšať i paralelný beh programu s využitím paralelizmu založeného na modeli vlákien a sprevádzkovať DIRAC aj v prostredí Cygwin, kde existuje potenciál na využitie modelu zasielania správ.

Kľúčové slová
vedecký softvér DIRAC, CMake, buildup, MS Windows, vysokovýkonné počítanie

ABSTRACT

Bc. HRAŠKO, Ivan: DIRAC in the framework of Microsoft Windows: adaptation and high-performance computing. [Master's thesis]. Matej Bel University in Banská Bystrica. Faculty of Natural Sciences. Department of Chemistry. Supervisor: doc. RNDr. Miroslav Iliaš, PhD. Degree of Qualification: Master in Specialization Applied Computer Science. Banská Bystrica: FPV UMB, 2015, 79 p.

The topic and aim of this master's thesis is to adapt the DIRAC relativistic quantum chemistry program for the environment of the Microsoft Windows operating system. Adaptation is done through modifications and further extension of program's build-up configuration and associated utility scripts. Multi-platform tools CMake and Python are employed. Testing of compilers and mathematical libraries which could be used with the program on Windows is also part of the work. We tend to run the program in parallel by utilizing the parallel programming models implemented in DIRAC itself or in the linked libraries.

Thesis consists of five chapters. The first chapter theoretically focuses on high-performance computing, including parallel architectures and parallel programming models. The second chapter discusses message passing interface specification – MPI. Goals, target architectures and some of the implementations are presented. Work flow with source codes is described in the separate chapter together with listing of the software tools which were used during the work on DIRAC. This work that was done on the project is summarized in the fourth chapter. The last chapter deals with continuous integration. Smooth development of the DIRAC code is ensured and automatized by continuous integration services.

The output of our work is improved usability of DIRAC on both 64-bit and 32-bit Microsoft Windows operating systems. The program was successfully run in parallel by utilizing of so-called threads programming model. It is also possible to run DIRAC in the Cygwin environment with the potential to use message passing parallel programming model.

Keywords:
scientific software DIRAC, CMake, buildup, MS Windows, high-performance computing

PREDHOVOR

Vedecký program DIRAC, určený na relativistické kvantovo-chemické výpočty, bol vytvorený v prostredí operačných systémov Unix, Linux a Mac. Na týchto systémoch je aj najviac používaný, a to i s využitím možností paralelizácie na zvýšenie jeho výpočtového výkonu. Avšak, aj operačný systém Microsoft Windows má potenciál v oblasti vysokovýkonného počítania - klaudová platforma Microsoft Azure je toho príkladom. Adaptovaním softvéru pre iné operačné systémy sa otvárajú príležitosti k získaniu nových používateľov, či vedeckých grantov.

Adaptácia je vykonávaná pomocou praktickej úpravy konfiguračných skriptov programu DIRAC a testovania dostupných kompilátorov, matematických a iných knižníc za účelom jeho úspešnej kompilácie na operačnom systéme Windows.
Na projekte DIRAC pracuje množstvo vývojárov a je neustále rozširovaný, čo zvyšuje náročnosť adaptácie. Najmä, ak sú pridávané nové závislosti na externý softvér. Je preto potrebné pre Windows adaptovať aj novo pridávané funkcionality programu.
K odhaľovaniu chýb a problémov v programe napomáha kontrola pomocou priebežnej (kontinuálnej) integrácie. Existuje niekoľko služieb, ktoré je možné na tento účel využívať.
Cieľom je, aby bol program DIRAC čo najlepšie použiteľný v prostredí operačného systému Microsoft Windows. Jeho použiteľnosť je potrebné neustále vylepšovať. Značné úsilie je vynakladané na dosiahnutie paralelného vysokovýkonného počítania.

OBSAH

ZOZNAM ILUSTRÁCIÍ

ZOZNAM SKRATIEK A ZNAČIEK

CPU - Central Processing Unit, centrálna procesorová jednotka
CI - Continuous Integration, priebežná (kontinuálna) integrácia
YAML - YAML Ain't Markup Language, YAML nie je značkovací jazyk
HPC - High Performance Computing, vysokovýkonné počítanie
UMA - Uniform Memory Access, rovnaký čas prístupu do pamäte
NUMA - Non-Uniform Memory Access, rozdielny čas prístupu do pamäte
CC-UMA - Cache Coherent UMA, UMA s koherenciou vyrovnávacej pamäte
CC-NUMA - Cache Coherent NUMA, NUMA s koherenciou vyrovnávacej pamäte
SMP - Symmetric Multiprocessor, symetrický multiprocesor
SPMD - Single Program Multiple Data, jeden program a viacero údajov
MPMD - Multiple Program Multiple Data, viacero programov a viacero údajov
API - Application Programming Interface, rozhranie pre programovanie aplikácií
UTC - Coordinated Universal Time, koordinovaný svetový čas
MKL - (Intel) Math Kernel Library, (Intel) základná matematická knižnica
MPI - Message Passing Interface, rozhranie pre zasielanie správ
MPICH - MPI over CHameleon, MPI založené na systéme Chameleon
MIMD - Multiple Instruction Stream Multiple Data Stream, viacero prúdov inštrukcií a viacero prúdov údajov
GCC - The GNU Compiler Collection, GNU sada kompilátorov
GNU - GNU's Not Unix, GNU nie je Unix
MinGW - Minimalist GNU for Windows, minimalistický GNU pre Windows
MSYS - Minimal System, minimálny systém
BLAS - Basic Linear Algebra Subprograms, podprogramy pre základnú lineárnu algebru
LAPACK - Linear Algebra Package, balík lineárnej algebry
flag - prepínač/možnosť príkazového riadka (pre kompilátor)
napr. - napríklad
pozn. - poznámka
t.j. - to je
a pod. - a podobne
atď. - a tak ďalej
kap. - kapitola

ÚVOD

Témou diplomovej práce je adaptovanie kvantovo-chemického programu DIRAC[1] pre prostredie operačného systému Windows. Pod týmto sa myslí, aby bolo možné program úspešne skompilovať a vykonať testy. To v princípe znamená urobiť ho schopným vykonávať výpočty, ktoré od neho jeho používatelia očakávajú. Túto schopnosť programu je potrebné vzhľadom na jeho pokračujúci vývoj neustále udržiavať a vylepšovať.

K tomuto cieľu patrí i odskúšanie niektorých kompilátorov a matematických knižníc, ktoré by sa dali na systéme Windows používať. Ideálne by bolo, keby sa podarilo nájsť také prostriedky, ktoré by umožnili paralelný beh programu. Týmto by sa DIRAC podľa možností priblížil k cieľu vysokovýkonného počítania nielen na systémoch Unix a Linux, ale i na systémoch Windows.

Pri našej práci na týchto cieľoch chceme postupovať podľa možností čo najuniverzálnejším spôsobom. Takto by sa totiž mohol proces konfigurácie a kompilácie programu DIRAC našimi zásahmi nielen zachovať nepoškodený aj na iných platformách, ale došlo by i k jeho vylepšeniu a zjednodušeniu. K tomuto účelu sa javia ako vhodné nástroje CMake[2], prípadne Python[3].

Obsah tejto práce je rozdelený do piatich kapitol. Venujeme sa v nich teoretickému úvodu do vysokovýkonného paralelného počítania, vysvetleniu spôsobu našej práce so zdrojovými kódmi programu DIRAC a predstavujeme výsledky, ktoré sme dosiahli. V prípade, že sa niečo nepodarilo, hodnotíme stav vecí, aký je a snažíme sa vysvetliť príčinu, alebo aspoň vyhľadať možné spôsoby a nástroje, ktoré by pomohli k budúcemu riešeniu.

V prvej kapitole je uvedená definícia vysokovýkonného počítania a predstavenie architektúr paralelných počítačov. Taktiež sú predstavené aj programovacie modely používané pre programovanie na paralelných architektúrach. DIRAC niektoré z týchto modelov využíva, aby dosahoval vyšší výkon pri výpočtoch, ktoré vykonáva.

Druhá kapitola je venovaná rozhraniu pre zasielanie správ, ktoré patrí medzi programovacie modely spomenuté v prvej kapitole. Tu sa najprv venujeme tomu, s akými cieľmi bol tento štandard vyvíjaný a pre ktoré paralelné architektúry je vhodný. V tejto kapitole sa ešte v krátkosti zameriame na rôzne implementácie MPI[4], s ktorými sme sa počas práce na DIRAC-u stretli.

V tretej kapitole si rozoberieme organizáciu našej práce na takom rozsiahlom projekte, akým program DIRAC určite je. Ide hlavne o prácu s rôznymi vývojovými vetvami a repozitármi a ich administráciu. Taktiež spomenieme, aké softvérové nástroje

sme používali počas nášho adaptovania, vylepšovania a skúšania programu DIRAC[1]. Vymenujeme nami použité kompilátory, knižnice a verzie systémov, v ktorých sme s programom pracovali. V krátkosti bude vysvetlený i spôsob konfigurácie, kompilácie a otestovania programu.

Nasleduje štvrtá kapitola, ktorá sa už venuje konkrétnym činnostiam, ktoré sme uskutočnili. Táto kapitola je delená na niekoľko častí, z ktorých jednotlivé opisujú konkrétnu oblasť alebo oblasti práce vykonanej na projekte DIRAC.

Veľmi dôležitým aspektom pre rozsiahle projekty (ale nielen pre ne) je, aby sa vždy udržiavali v zdravom, relatívne funkčnom stave počas celej doby ich vývoja. Tento aspekt vývoja je zabezpečovaný pravidelným testovaním celého projektu. Vykonávanie tejto činnosti sa nazýva priebežná integrácia (*Continuous Integration , CI*). V súčasnosti existuje viacero služieb, ktoré nám môžu pomôcť tento proces zautomatizovať.

Priebežnej integrácii je venovaná piata kapitola, kde si predstavíme skripty pre niektoré zo služieb poskytujúcich priebežnú integráciu. Pokúsime sa nájsť a odskúšať i službu priebežnej integrácie pre Windows.

1. Vysokovýkonné počítanie

Vysokovýkonné počítanie (*High Performance Computing, HPC*) je snaha o využitie počítačových systémov (spájanie ich výkonu, napr. do klastrov) takým spôsobom, aby sa dosahoval čo najvyšší možný celkový výkon. Veľký výpočtový výkon je potrebný pre vykonávanie náročných výpočtov v rôznych odvetviach vedy (vrátane chémie), inžinierstva alebo obchodu. [5]

Pri tomto druhu výpočtov si silnú pozíciu udržiava programovací jazyk Fortran, ktorý je aj hlavným jazykom používaným v programe DIRAC[1]. [6][7]

K vysokovýkonnému počítaniu patria paralelné architektúry počítačov. Preto si ich ďalej predstavíme. Budeme sa venovať i programovacím modelom, ktoré sú pre ne určené.

1.1 Pamäťové architektúry paralelných počítačov

Rozlišujeme tri paralelné architektúry na základe parametra prístupu do pamäte. Ide o architektúry so spoločnou pamäťou (globálnym adresným priestorom), s distribuovanou pamäťou a architektúry, ktoré sú založené na hybridnom prístupe. [8]

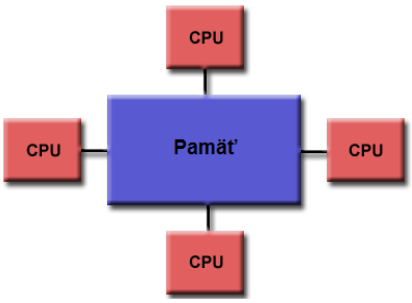
1.1.1 Spoločná pamäť

Paralelné počítače využívajúce spoločnú pamäť sa môžu medzi sebou navzájom líšiť. Majú ale vždy spoločné to, že všetky procesory majú možnosť pristupovať do všetkej pamäte, ktorá sa v takom systéme nachádza. Takto je dosiahnuté vytvorenie globálneho adresného priestoru, ktorý je dostupný pre všetky procesory. Jednotlivé procesory síce môžu pracovať nezávisle, ale delia sa o rovnaké pamäťové zdroje. To znamená, že pri tejto architektúre sú zmeny v pamäti spôsobené jedným procesorom viditeľné ostatným procesorom. Systémy tohto typu sa zvyknú klasifikovať ako UMA (*Uniform Memory Access*), alebo opačne ako NUMA (*Non-Uniform Memory Access*). Toto sa určuje v závislosti od času prístupu procesorov k údajom v pamäti. [8]

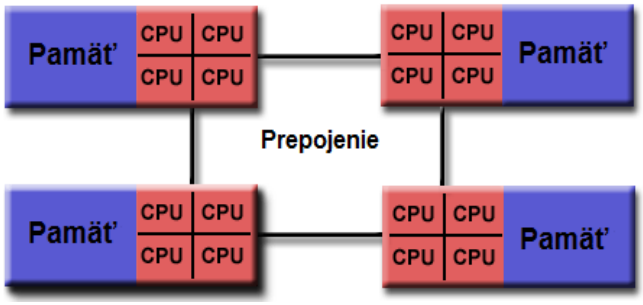
Paralelné počítače typu UMA (viď Obr. 1) zvyknú byť reprezentované najmä počítačmi SMP (*Symmetric Multiprocessor*). V tomto prípade sú všetky použité procesory rovnaké, majú i rovnaký spôsob a čas prístupu do pamäte. Niektoré z týchto paralelných systémov sa môžu dokonca označovať ako CC-UMA (*Cache Coherent UMA*). Koherencia vyrovnávacej pamäte znamená, že keď jeden z procesorov vykoná nejakú zmenu v spoločnej pamäti, všetky ostatné procesory o tom vedia. Táto vlastnosť je zabezpečovaná na úrovni hardvéru. [8]

Počítače označované ako NUMA (viď Obr. 2) sú bežne vytvárané spojením dvoch alebo viacerých SMP počítačov. Tento typ systému sa vyznačuje tým, že jeden SMP môže

priamo pristupovať do pamäte patriacej inému počítaču SMP. Týmto sa však nevyhneme tomu, že nie všetky procesory majú rovnaký čas prístupu ku všetkým pamätiam. Je to spôsobené tým, že prístup cez prepojenie medzi počítačmi (*bus*) je pomalší ako prístup do pamäte v rámci jedného SMP počítača. V prípade, ak je zabezpečená koherencia pamäte, nazývajú sa takéto paralelné počítače ako CC-NUMA (*Cache Coherent NUMA*). [8]



Obr. 1: Paralelný počítač UMA, prevzaté a upravené zo zdroja [8]



Obr. 2: Paralelný počítač NUMA, prevzaté a upravené zo zdroja [8]

Výhodou paralelných architektúr so spoločnou pamäťou je to, že poskytujú pre programátorov pre nich prívetivý (jednotný) pohľad na pamäť - globálny adresný priestor. Navyše výmena údajov medzi procesormi je rýchla a uniformná (rovnaká). [8]

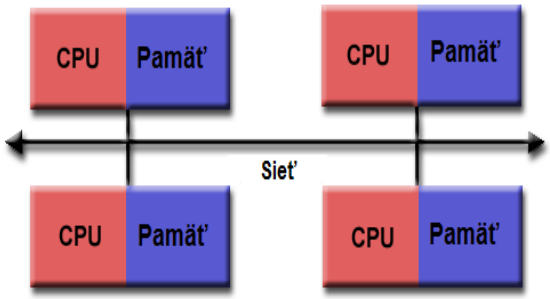
Ich nedostatky sa prejavujú predovšetkým v škálovateľnosti, pretože pridaním ďalších procesorov môže geometricky narásť množstvo údajov prenášaných medzi spoločnou pamäťou a procesormi. Navyše pre systémy s koherenciou vyrovnávacej pamäte môže geometricky narásť prevádzka spojená so zabezpečovaním tejto vlastnosti. Možnou nevýhodou je aj to, že programátor je zodpovedný za synchronizáciu, ktorá zabezpečuje správne pristupovanie do globálnej pamäte (poradie prístupu). [8]

1.1.2 Distribuovaná pamäť

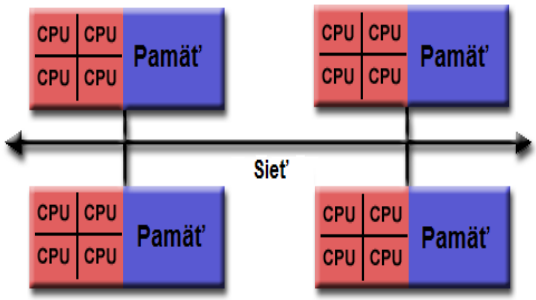
Systémy s distribuovanou pamäťou (viď Obr. 3) sa vyznačujú tým, že vyžadujú sieť na prepojenie procesorov medzi sebou. Pri tejto paralelnej architektúre procesory majú vlastnú lokálnu pamäť, ale nemapujú si pamäť iných procesorov. To znamená, že sa nevytvára globálny adresný priestor. Keďže každý procesor má vlastnú lokálnu pamäť, tak pracuje nezávisle od ostatných. Zmeny vykonané v lokálnej pamäti jedného procesora nemajú vplyv na pamäť iných procesorov. Neaplikuje sa tu princíp koherencie vyrovnávacej pamäte. Ak nastane situácia, že procesor potrebuje prístup k údajom iného procesoru, je to úloha, ktorú musí vyriešiť programátor. Je potrebné definovať, kedy a ako sa komunikácia uskutoční. Synchronizácia medzi procesmi je tiež ponechaná na riešenie programátorovi. Na zabezpečenie komunikácie sa môžu využívať rôzne typy sietí. [8]

Výhodou systémov s distribuovanou pamäťou je ich dobrá škálovateľnosť. Pridaním ďalších procesorov sa úmerne zvýši veľkosť pamäte. Každý procesor pristupuje nerušene do svojej lokálnej pamäte veľkou rýchlosťou. Pritom sa neprejavujú režijné náklady (spomalenie), ktoré by vznikli pokusmi o zabezpečovanie pamäťovej koherencie. Nespornou výhodou je aj nižšia finančná náročnosť týchto systémov. [8]

Nevýhodou paralelných systémov využívajúcich distribuovanú pamäť je zodpovednosť programátora za veľké množstvo detailov spojených s komunikáciou medzi jednotlivými procesormi. Ďalším rizikom je to, že môže byť problematické používať existujúce údajové štruktúry, ktoré boli pôvodne založené na využívaní globálnej pamäte. Vyskytujú sa aj nerovnaké prístupové časy do pamäte. To preto, lebo je rýchlejšie pre procesor pristupovať k svojej lokálnej pamäti, ako pristupovať cez sieť k pamäti iného procesora. [8]



Obr. 3: Paralelný počítač s distribuovanou pamäťou, prevzaté a upravené zo zdroja[8]



Obr. 4: Hybridný systém paralelného počítača, prevzaté a upravené zo zdroja [8]

1.1.3 Hybridné systémy

Najvýkonnejšie paralelné počítače spravidla využívajú kombináciu oboch vyššie spomenutých prístupov (viď Obr. 4). Skladajú sa z viacerých častí, ktoré využívajú model spoločnej pamäte. Tieto časti obsahujú niekoľko procesorov (prípadne grafických kariet), ktoré vedia iba o pamäti v tejto časti, ale nie v inej časti, na inom počítači. Na prepojenie častí so spoločnými pamäťami je použitá sieť. Prípadné prenosy údajov medzi týmito časťami sa teda musia uskutočniť cez ňu. Systémy s hybridnou architektúrou sa vyznačujú dobrou škálovateľnosťou. Na druhej strane však narastá náročnosť ich programovania. [8]

1.2 Paralelné programovacie modely

Existuje niekoľko programovacích modelov pre prostredie paralelných počítačov. Tieto sú vytvorené ako abstrakcie nad hardvérovými a pamäťovými architektúrami. To ale neznamená, že sú na hardvérové alebo pamäťové architektúry naviazané. Programovacie modely môžu byť teoreticky implementované nad ľubovoľným hardvérom. [8]

1.2.1 Spoločná pamäť (bez vlákien)

Pri použitý tohto modelu sa procesy delia o spoločný adresný priestor, v ktorom vykonávajú čítanie a zápis asynchrónne. Na kontrolovanie prístupu do pamäte sa využívajú zámky alebo semafory. Tento model relatívne zjednodušuje vývoj programov, ale môže dochádzať k znižovaniu výkonu paralelného systému. [8]

1.2.2 Vlákna

Tu je situácia taká, že jeden proces môže mať niekoľko vlákien, a tie môžu byť vykonávané paralelne. Platí, že každé vlákno má aj svoje lokálne údaje, ale delí sa o adresný priestor a zdroje rodičovského procesu. Využívaním spoločných zdrojov s rodičovským procesom dochádza k šetreniu zdrojov systému. Keďže vlákna komunikujú cez spoločnú pamäť, je nevyhnutné použitie synchronizačných prostriedkov. Treba si uvedomiť, že rodičovský proces skončí (môže skončiť), až keď skončí posledné jeho vlákno. Za paralelizmus programu je pri používaní vlákien zodpovedný programátor. Môže využiť OpenMP[9] direktívy alebo POSIX knižnicu. Systémy Windows používajú vlastnú implementáciu vlákien. [8]

1.2.3 Distribuovaná pamäť (zasielanie správ)

Na paralelnom počítači s distribuovanou pamäťou majú procesy vlastnú lokálnu pamäť na vykonávanie svojich výpočtov. V prípade, že je to potrebné, tak si údaje vymieňajú pomocou zasielania správ. Pre vykonanie zasielania správ je ale žiaduca kooperácia procesov. Dôvodom je, že ak nejaký proces správu posiela, je potrebné, aby ju nejaký proces prijal. Model zasielania správ je štandardizovaný v špecifikácii MPI[4]. [8]

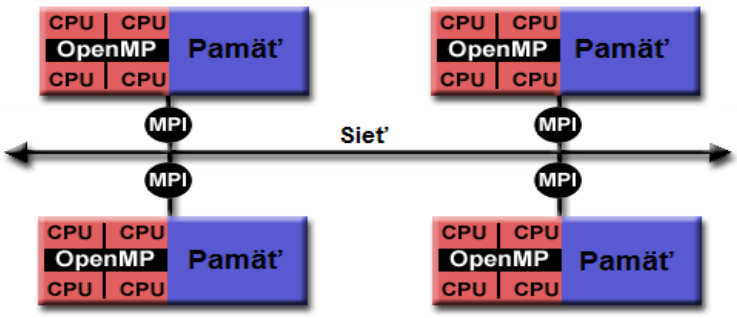
1.2.4 Dátový paralelizmus

Tento typ paralelizmu sa zameriava na množinu údajov (zvyknú sa organizovať ako pole, kocka). Spôsob práce je taký, že procesy pracujú paralelne na rovnakej dátovej štruktúre (množine údajov), ale každý proces pracuje na inej časti tejto štruktúry. Jednotlivé procesy vykonávajú na svojich častiach rovnaké činnosti. Čo sa týka rozdelenia dátovej štruktúry pre procesy, tak na architektúrach so spoločnou pamäťou sa štruktúra nachádza v globálnej pamäti, kde ju všetky procesy vidia. Na paralelných počítačoch s distribuovanou pamäťou sa štruktúra rozdelí na časti, ktoré sa potom rozpošlú procesom do lokálnej pamäte každého z nich. [8]

1.2.5 Hybridný model

Hybridný model (na Obr. 5) je založený na kombinovaní viacerých modelov z tých, ktoré boli doteraz uvedené. Častým príkladom hybridného modelu je spolupráca

OpenMP[9] a MPI[4]. Na serveroch prebiehajú výpočty vo vláknach pomocou OpenMP a medzi nimi sa posielajú správy po sieti s využitím MPI (viď kapitolu o MPI). [8]



Obr. 5: Hybridný model programovania paralelných počítačov, prevzaté a upravené zo zdroja [8]

1.2.6 SPMD (*Single Program Multiple Data*)

Ide o programovací model vyššej úrovne, ktorý je založený na využívaní predchádzajúcich modelov. Pri použití tohto modelu všetky procesy vykonávajú svoju vlastnú kópiu rovnakého programu. Avšak, to neznamená, že všetky musia vykonať celý program, pretože vo väčšine prípadov program obsahuje vetvenia. Týmto spôsobom každý proces vykoná iba jeho konkrétnu časť. Procesy zvyknú bežne spracovávať rozdielne množiny údajov. [8]

1.2.7 MPMD (*Multiple Program Multiple Data*)

MPMD je programovací model vyššej úrovne. V tomto modeli procesy môžu vykonávať rôzne programy súčasne, pričom spracovávajú rozdielne údaje. Tento model je menej bežný ako viac rozšírený už spomenutý model SPMD. [8]

2. MPI

V tejto kapitole predstavíme špecifikáciu rozhrania pre programovací model zasielania správ - MPI[4]. Priblížime si jej hlavné ciele a cieľové architektúry. V ďalších podkapitolách i niektoré implementácie. Budeme sa venovať tým, s ktorými sme sa nejakým spôsobom stretli pri našej práci. Program DIRAC[1] pracuje s MPI verzie 2. Ideálne je, keď použitá implementácia podporuje 64-bitový typ *integer* pre jazyk Fortran.

2.1 Prehľad a ciele

MPI (*Message-Passing Interface*) je špecifikácia rozhrania softvérovej knižnice, ktorá v paralelnom výpočtovom prostredí umožňuje komunikáciu paralelne vykonávaných procesov prostredníctvom využitia programovacieho modelu zasielania správ. [4]

Pri využitý tohto programovacieho modelu sa údaje presúvajú z adresného

priestoru jedného procesu (pamäte viditeľnej procesu) do adresného priestoru iného procesu. Vyžaduje sa vzájomná kooperácia oboch zúčastnených procesov. [4]

MPI je špecifikácia rozhrania knižnice, nie je implementáciou. Implementácií je dostupných niekoľko, viď kapitoly o Open MPI, Intel MPI, MPICH a Microsoft MPI. Rozhranie MPI nie je ani programovací jazyk, všetky ním definované operácie sú špecifikované ako metódy, funkcie alebo podprogramy. Podľa zvyklostí zaužívaných v programovacom jazyku, v ktorom je napísaná implementácia. Programovacie jazyky zvyčajne používané pre implementáciu MPI sú C, C++, Fortran-77 a Fortran-95. [4]

Pozn.: Implementácia Open MPI od verzie 1.7 podporuje aj programovací jazyk Java v takmer úplnej špecifikácii podľa MPI-3. [10]

Špecifikácia MPI vznikla za otvorenej spolupráce komunity dodávateľov paralelných systémov, počítačových vedcov a vývojárov paralelných aplikácií. Výhodou toho, že bola zadefinovaná je, že sa tým umožnila prenositeľnosť a jednoduché použitie programovacieho modelu zasielania správ. [4]

V prostredí s distribuovanou pamäťou, kde sa na komunikáciu využívajú vyššie funkcie a abstrakcie, ktoré sú postavené na funkciách zasielania správ na nižších úrovniach, sú výhody štandardizácie výrazné. [4]

Otvára sa tým možnosť pre dodávateľov paralelných systémov na efektívnu implementáciu presne a jasne špecifikovaných operácií. Pre niektoré z nich môžu byť dokonca schopní poskytnúť aj podporu na úrovni hardvéru. Týmto spôsobom je možné dosiahnuť zvýšenie škálovateľnosti týchto systémov. [4]

Hlavným cieľom MPI je vytvoriť praktický, prenositeľný, efektívny a prispôsobiteľný štandard so širokým uplatnením pri tvorbe paralelných programov využívajúcich model zasielania správ. [4]

Súhrnný zoznam cieľov podľa [4]:

- Navrhnúť API (*Application Programming Interface*)
- Umožniť efektívnu komunikáciu. Pri takejto komunikácii sa vyhnúť všade, kde je to možné kopírovaniu z pamäte do pamäte. Zároveň je potrebné umožniť súčasné prebiehanie výpočtov a komunikácie. Nezaťažovať komunikáciou koprocesor
- Umožniť používanie v heterogénnom prostredí
- Umožniť jednoduché implementovanie rozhrania v programovacích jazykoch C, C++, Fortran-77 a Fortran-95
- Zabezpečiť spoľahlivé komunikačné rozhranie, v ktorom sú zlyhania komunikácie ošetrené komunikačným subsystémom bez účasti používateľa

- Definovať rozhranie implementovateľné na rôznych platformách bez významných zásahov do ich základného komunikačného a systémového softvéru
- Sémantika rozhrania by mala zostať nezávislá od použitých programovacích jazykov
- Rozhranie by malo byť navrhnuté takým spôsobom, aby umožňovalo bezpečnú prácu s vláknami (*thread safe*)

2.2 Cieľové architektúry

Dôvodom značného rozšírenia programovacieho modelu zasielania správ je to, že je dobre prenositeľný medzi rôznymi architektúrami. [4]

Programy, ktoré sú na ňom založené, sú schopné pracovať na multiprocesoroch s distribuovanou pamäťou, sieťach pozostávajúcich z prepojených pracovných staníc, alebo i kombinácii oboch spomenutých prípadov. [4]

Nie je vylúčené ani použitie na architektúrach so spoločnou pamäťou, a to i vrátane architektúr viacjadrových procesorov, a na hybridných architektúrach. MPI je teda možné s úspechom implementovať na veľkom množstve systémov. Vrátane takých, ktoré pozostávajú z iných systémov prepojených vhodnou komunikačnou sieťou. Bez ohľadu na to, či už sú tieto podsystémy paralelné alebo nie. [4]

Rozhranie MPI je vhodné pre programy plne využívajúce paralelizáciu MIMD alebo viac obmedzenú techniku SPMD. MPI poskytuje množstvo funkcií pre dosiahnutie zvyšovania výkonu na škálovateľných paralelných systémoch so špecializovaným hardvérom zabezpečujúcim medziprocesorovú komunikáciu. [4]

2.3 Implementácie MPI

Počas našej práce na DIRAC-u[1] sme sa stretli s viacerými implementáciami MPI[4]. Jedná sa o Open MPI[11], Intel MPI[12], MPICH[13] a Microsoft MPI[14]. Teraz ich v krátkosti predstavíme.

2.3.1 Open MPI

Open MPI je *open source* implementácia špecifikácie MPI. Na projekte sa podieľa zoskupenie partnerov z oblasti akademickej sféry, výskumu a priemyslu. Táto spolupráca zabezpečuje, že Open MPI projekt je schopný vytvárať veľmi kvalitnú MPI knižnicu. [11]

Knižnica Open MPI je hlavne vyvíjaná na a pre 32 aj 64-bitové verzie operačných systémov Linux a (Mac) OS X. Windows bol podporovaný od verzie 1.3.3, ale jeho podpora bola odstránená vo verzii 1.8. Od tejto verzie už nie je viac podporovaný ani

operačný systém Solaris. Knižnica Open MPI by mala pracovať správne na väčšine POSIX-ových operačných systémov. [15]

Testovanie DIRAC-u[1] s využitím Open MPI je uskutočnené pomocou CI služieb CircleCI[16] a Codeship[17] (viď kapitolu o priebežnej integrácii).

2.3.2 Intel MPI

Knižnica Intel MPI (*Intel MPI Library 5.0*) implementuje špecifikáciu MPI-3 a je dostupná nielen pre operačné systémy Linux, ale aj pre systémy Windows [12]. Plánovali sme ju použiť s Intel[18] kompilátormi, viď kapitolu o problémoch s adaptáciou pre Intel.

Napriek tomu, že implementuje najnovšiu špecifikáciu, je stále možné ju využívať s programami, ktoré používajú iba MPI-1 alebo MPI-2. Existuje dokonca možnosť využívať jej behové prostredie aj pre programy, ktoré boli skompilované voči MPICH. A to bez potreby ich znovu skompilovať, viď [19]. [12]

Pri použití Intel MPI je možné spúšťať paralelné procesy v klastroch, ktoré sa skladajú zo serverov s rozličnými operačnými systémami (Linux a Windows). Nemal by byť problém ani pri spolupráci rozličných Intel procesorov, alebo pri použití procesorov od výrobcu AMD. [12]

2.3.3 MPICH

MPICH je vysokovýkonná implementácia MPI špecifikácie. Podporuje štandard MPI-1, MPI-2 i MPI-3. MPICH je šírená pod *open source* licenciou. [13][20]

Bola testovaná na viacerých platformách. Išlo o systémy Linux v 32-bitovej i 64-bitovej verzii, Mac OS/X (PowePC a Intel), Solaris (32-bitové aj 64-bitové) a Windows. Na Windows je však dostupná už iba cez Microsoft MPI [21]. [13][20]

Knižnica je dostupná vo viacerých Linux-ových distribúciách. Testujeme ju na CI službe Shippable[22] a doc. M. Iliaš na Semaphore[23]. Na systéme Windows už nie je podporovaná, podľa zdroja dokonca ani na Cygwin-e[24]. [13][21]

2.3.4 Microsoft MPI

Microsoft MPI (*MS MPI*) je implementácia MPI od Microsoft-u, ktorá umožňuje vývoj a spúšťanie MPI aplikácií na platforme Windows. [14]

Nám sa nepodarilo využiť MS MPI spolu s MinGW-w64[25] kompilátormi. Najmä s použitím kompilátora pre Fortran sme mali problémy.

Avšak, podarilo sa nám nájsť (síce trochu neskoro a ani inštalácia sa nepodarila) projekt, ktorý sa venuje adaptovaniu MinGW-w64 kompilátorov pre využitie s MS MPI. Jedná sa o projekt PToolsWin[26], ktorý je obsiahnutý aj v distribúcii HPC Linux[27].

Táto distribúcia by mala byť dostupná i na Microsoft Azure[28], alebo je možné si ju stiahnuť ako virtuálny obraz (*.ova*), napr. pre program Oracle VM VirtualBox[29].

Pre prípadné využitie MPI v klastroch založených na Microsoft HPC Pack viď aj Microsoft HPC Pack SDK[30].

3. Organizácia práce a použitý softvér

V tejto kapitole bude vysvetlený spôsob práce s vývojovými vetvami programu DIRAC[1]. Predstavíme si softvér, ktorý sme používali na jeho kompilovanie. Ďalej sa budeme v krátkosti venovať jazyku Python[3], ktorý bol pri práci na konfiguračných skriptoch a adaptácii spolu so systémom CMake[2] veľmi užitočný. Na konci kapitoly sa nachádza vysvetlenie spôsobu konfigurácie programu DIRAC na systéme Windows vrátane možností, ktorými je možné ovplyvňovať jeho prácu.

3.1 Práca so zdrojovými kódmi

Pri našej práci so zdrojovými kódmi DIRAC-u[1] sme využívali niekoľko služieb, ktoré umožňujú vytvoriť súkromné repozitáre. Konkrétne išlo o služby Bitbucket[31][32], GitHub[33][34] a službu GitLab[35][36]. Používali sme verzionovací nástroj Git[37][38]. Postupom času sme menili aj spôsob práce využívaním viacerých vývojových vetiev označovaných ako *branch* alebo *fork*.

Najprv sme pracovali iba s repozitárom *my-dirac*, ktorý bol umiestnený na Bitbucket. Tu boli zriadené dve vývojové vetvy *master* a *hrasko_dirac_devel*. Vetva *master* predstavovala kópiu originálneho DIRAC-u (v súčastnosti sa nachádza na GitLab). Vetva *hrasko_dirac_devel* bola určená pre našu prácu. Keď sme nejakú úlohu doriešili, alebo prípadne časť z nej, tak sme požiadali vedúceho práce doc. M. Iliaša, aby zmeny preniesol do originálu DIRAC-u. S originálom DIRAC-u sme udržiavali krok pomocou vetvy *master*, z ktorej sme si prenášali zmeny do *hrasko_dirac_devel*.

Netrvalo dlho a pribudli ďalšie vetvy určené pre prácu študentov. Vtedy sa zmenil spôsob organizácie našej práce s nimi. Vytvorila sa vetva *merges_from_students*, ktorej účelom bolo zozbierať zmeny z vývojových vetiev študentov, aby z nej mohli byť prenášané do originálneho DIRAC-u. Vetva *master* na Bitbucket bola určená na synchronizovanie študentských vetiev s originálom.

Prebiehalo to tak, že každý študent pracoval na svojej vetve, a keď niečo dokončil, vytvoril *pull request* do vetvy *merges_from_students*. Tu bola možnosť zmeny akceptovať, alebo ich aj odmietnuť. Keď boli zmeny akceptované (*merge*), tak ich vedúci práce príkazmi pre Git (*pull, push*) prenášal do originálneho DIRAC-u.

Pre použitie rôznych CI služieb (viď kapitolu o priebežnej integrácii) sme začali pracovať s DIRAC-om[1] i na GitHub[33], kde mal vedúci práce vytvorený súkromný repozitár s kódom DIRAC-u. Bolo nám umožnené urobiť si jeho *fork*. Spôsob práce sa tu po istých problémoch ustálil nasledovným spôsobom.

K vtedy už existujúcej vetve *master* bola dodatočne vytvorená vetva *merges_from_collaborators*. Následne bol postup práce taký, že sme priamo vykonávali zmeny vo svojom *fork-u* na vetve *merges_from_collaborators*. V prípade, že sme sa potrebovali zosynchronizovať s originálom, tak sme z vetvy *master* v repozitári vedúceho (*upstream*) stiahli zmeny do vetvy *master* v našom *fork-u*. Potom sme ďalej preniesli zmeny z nášho *master* do našej vývojovej vetvy *merges_from_collaborators*. Keď nastala potreba preniesť zmeny do originálneho DIRAC-u, tak sme vytvorili *pull request* z nášho *fork-u* vetvy *merges_from_collaborators* do repozitára doc. M. Iliaša taktiež do vetvy *merges_from_collaborators*. Akceptované zmeny boli prenesené do DIRAC-u na GitLab[35]. Vetva *master* v repozitári doc. M. Iliaša slúžila, podobne ako na Bitbucket[31], na synchronizáciu s originálom. Preto sme v prípade, že sme si chceli našu kópiu zosynchronizovať, sťahovali zmeny najprv z nej.

Práca na projekte PCMSolver[39] (viď kapitolu o adaptovaní PCMSolver) bola organizovaná tak, že na GitHub sme si vytvorili vlastný *fork* odvodený od originálneho kódu PCMSolver-a (je to otvorený projekt). V našej pracovnej kópii bol náš *fork* a jeho vetvy označené ako *origin* a originálny repozitár a jeho vetvy ako *upstream*. Túto kópiu sme udržiavali aktualizovanú s originálom (*upstream*) samostatne pomocou Git-u[37]. V prípade, že sme mali záujem o prenesenie našich zmien do originálneho projektu, vytvorili sme *pull request*. Tento mohol autor PCMSolver-a prijať alebo odmietnuť. Prijatím *pull request* požiadavky sa obsah zmien preniesol do originálneho repozitára.

Zo skúsenosti sme prišli na to, že niekedy je dobré mať vetvy nielen podľa študentov, ale i podľa účelu. Najmä pri konfigurovaní CI služieb, keď bolo potrebné urobiť veľké množstvo zmien, aby sme dosiahli požadovaný výsledok. Týmto spôsobom bola použitá aj vetva na službe Bitbucket *hrasko_dirac_devel*, ktorá bola po nastavení jednej z CI služieb vymazaná a nahradená vetvou *hrasko_dirac_devel_2*.

Na prácu s Intel kompilátormi[18] sme si vytvorili vetvu *intel_mpi*. Z nej sa povyberali zmeny (*cherry pick* [40]), ktoré by bolo dobé preniesť do originálu. A to i napriek tomu, že sa DIRAC nepodarilo úplne skompilovať. Niekto ďalší, kto bude adaptovať Intel kompilátory pre použitie s týmto programom na Windows, nebude musieť niektoré problémy riešiť znova. Viac informácií v kapitole o adaptovaní pre Intel.

3.2 Používaný softvér

Teraz si predstavíme operačné systémy, kompilátory a knižnice, ktoré sme používali na kompilovanie DIRAC-u[1].

Keďže hlavnou náplňou našej práce bolo zlepšiť adaptovanie DIRAC-u pre použitie na operačnom systéme Windows, väčšinu práce sme vykonali na ňom. Pracovali sme s jeho 32-bitovými aj so 64-bitovými verziami.

Mali sme k dispozícii 32-bitovú verziu systému Windows 7, na ktorom sme zo začiatku používali 32-bitové kompilátory MinGW[41] (tieto 64-bitové ani nie sú). BLAS[42] a LAPACK[43] knižnice sme využívali referenčné 32-bitové z Netlib[44].

Neskôr sa nám podarilo využiť aj kompilátory LLVM Clang[45]. Pre svoju funkčnosť však používajú hlavičkové súbory z MinGW, viď dokumentáciu na [46]. Ďalšie kompilátory, ktoré sme odskúšali boli TDM-GCC[47] a našli sme pre nás nový projekt MinGW-w64[25].

Až do pridania externého modulu PCMSolver[39] neboli problémy úspešne skompilovať DIRAC s kompilátormi MinGW alebo s kompilátormi TDM-GCC. To, že padalo veľa testov sme vyriešili, keď sme odstránili problém so správnym hľadaním báz na Windows, o čom sa zmieňujeme v inej časti práce. Avšak, kvôli ich neschopnosti (pravdepodobne chyba) skompilovať PCMSolver sme natrvalo prešli ku kompilátorom z MinGW-w64 projektu. Kompilátormi Clang sme PCMSolver ani neskúšali skompilovať Tieto nie sú priorita, pretože neobsahujú kompilátor pre hlavný jazyk Fortran.

Podarilo sa nám použiť paralelizovanú knižnicu OpenBLAS[48], ktorá poskytuje vysoko optimalizovanú verziu BLAS. Dokonca implementuje optimalizované verzie niektorých LAPACK funkcií (podľa skúseností asi iba na Windows). K tomuto nám pomohla vedomosť, že OpenBLAS existuje, keďže preň prof. H. J. Aa. Jensen[49] upravil vyhľadávanie matematických knižníc počas konfigurácie. Pre použitie na systéme Windows 7 sme si OpenBLAS museli skompilovať, aby vyhovoval architektúre procesora AMD K10[50]. Vlastnú knižnicu OpenBLAS je možné na Windows jednoducho skompilovať v prostredí MSYS[51].

Čo sa týka 64-bitových systémov, najprv sme pracovali na serveri „Urpín“ (pracoval s Windows 7), až pokiaľ nebol pre poruchu odstavený. Tam boli používané 64-bitové kompilátory TDM-GCC. Nie je nám známe, že by sa tam nachádzali nainštalované nejaké knižnice BLAS alebo LAPACK.

Neskôr sme si nainštalovali vlastný virtuálny počítač so systémom Windows Server 2012 R2 získaný z Microsoft DreamSpark[52]. Tu používame 64-bitové kompilátory z

MinGW-w64[25] a predkompilované 64-bitové referenčné BLAS[42] a LAPACK[43] knižnice z Netlib[44]. Tieto knižnice však nie je možné využívať bez zapnutej funkcie 64-bitového typu *integer* – program sa skompiluje, ale budú padať testy. Najprv sme i tu používali kompilátory z TDM-GCC[47], ale tie nedokázali skompilovať PCMSolver[39].

Posledný Windows počítač, s ktorým sme pracovali je virtuálny server Katedry chémie „Chémia“. Má nainštalovaný operačný systém Windows Server 2012. Na tomto počítači sa používajú 64-bitové verzie kompilátorov z MinGW-w64. Podarilo sa taktiež s úspechom využiť knižnicu OpenBLAS[48], ktorá pracuje paralelne na viacerých vláknach. Tu máme možnosť použiť i jej predkompilovanú verziu, pretože taká je pre architektúru Intel NEHALEM[53] nachádzajúcu sa na serveri dostupná.

Používame aj Cygwin[24], ktorý vytvára prostredie podobné Linux-u v rámci operačného systému Windows. V ňom sa nám podarilo program DIRAC[1] úspešne skompilovať s využitím Open MPI[11] knižnice na serveri „Chémia“ (existuje ale problém s viacerými testami). Cygwin používame aj na Windows 7 (32-bitový, iba sériovo) a Windows Server 2012 R2 (64-bitový, podľa potreby).

Pri práci sme sa nevyhli ani systému Linux. Novú funkcionalitu alebo opravu bolo potrebné odskúšať i na tomto systéme. Jeho použitie bolo nápomocné pri sledovaní, akým spôsobom prebieha konfigurácia DIRAC-u, alebo pri odskúšaní iných kompilátorov ako MinGW[41][47][25]. Iné kompilátory boli dostupné najmä na HPCC UMB[54]. Navyše nám známe CI služby všetky okrem jednej používajú Linux.

Pri našej práci sa ukázalo byť vhodné využívať na hľadanie informácií rôzne webové fóra a publikácie, z ktorých sme potom pokračovali k dokumentáciám a ďalším odkazom, ktoré nás doviedli k cieľu [55][56][57][58][59][60][61]. Na detailné sledovanie procesov na systéme Windows je vhodným nástrojom program Process Explorer[62].

3.3 Python

Pre dosiahnutie multiplatformovej adaptácie konfigurácie programu DIRAC[1] sa ukázali ako vhodné nástroje Python[3] a CMake[2][63]. Použitiu systému CMake sme sa venovali v bakalárskej práci[64].

Python je interpretovaný, interaktívny, objektovo orientovaný programovací jazyk, ktorý je dostupný pre veľké množstvo platforiem. Využívaný je v oblasti umenia, obchodu, vzdelávania, vedy, inžinierstva, pri vývoji softvéru a v iných ďalších odvetviach. Python pôvodne vytvoril Guido van Rossum. Momentálne je jeho vývoj zastrešený pod Python Software Foundation. Keďže ho môžeme používať na systémoch Linux, Windows,

Mac OS X a iných mnohých, stáva sa vhodným nástrojom na riešenie problémov, ktoré sa vyskytujú pri konfigurácii a kompilácii programu DIRAC[1]. [3][65]

V Python-e je napísaný konfiguračný skript (*setup*), ktorý spúšťa CMake[2] a odovzdáva mu potrebné parametre podľa možností zvolených používateľom. Využíva sa i pri nastavovaní a čítaní premenných prostredia pri behu DIRAC-u skriptom *pam* a pri nastavovaní použitých modulov v skripte *pamadm*. Svoje významné miesto má aj pri testovaní[66]. Pre DIRAC sa požaduje verzia minimálne 2.4 (na Windows 2.7). Python verzie 3.x nie je momentálne možné používať, ale Dr. R. Bast[67] už na tom pracuje.

My sme ho tiež použili pri viacerých príležitostiach. Najvýznamnejšie z nich boli adaptácia skriptu *doc/preprocess_sphinx*, zisťovanie verzií kompilátorov, naprogramovanie funkcionality v skripte *maintenance/get_failing.py* a adaptovanie kompilácie projektu PCMSolver[39] pre Windows. Pripomíname, že adaptácie urobené pomocou tohto jazyka sú univerzálne, teda ostala zachovaná (prípadne sa pridala) funkcionalita aj pre operačné systémy Linux, Mac OS X a iné.

Existuje množstvo zdrojov, kde je možné sa oboznámiť s používaním jazyka Python, viď [68][69][70][71][72]. Pre vytváranie dokumentácie DIRAC-u je potrebné mať nainštalované niektoré Python balíčky (viď DIRAC dokumentáciu na [1]). Na inštaláciu sme používali nástroj pip[73] a balíky sme vyhľadávali v zozname Python balíčkov[74].

3.4 Konfigurácia programu DIRAC na Windows

Povedzme si niečo o konfigurácii programu DIRAC[1], ktorú je potrebné vykonať pred jeho skompilovaním. Predpokladajme, že sa nachádzame priamo v adresári so zdrojovými súbormi.

Program pozostáva z viacerých modulov. To, ktoré z nich sa budú používať je možné nastavovať pomocou skriptu *pamadm*. Všetky jeho možnosti si môžeme zobraziť príkazom: *python pamadm --help*. My sme však vždy používali predvolené nastavenie.

Následná konfigurácia spočíva v spustení skriptu *setup*. Skript *setup* umožňuje ovládať všetky aspekty konfigurácie, ktoré by používateľa mohli zaujímať. Úplný zoznam možností, ktoré tento konfiguračný nástroj ponúka je možné si nechať vypísať na príkazovom riadku: *python setup --help*

Účelnosť tohto skriptu spočíva v tom, že tvorí akési rozhranie (sprievodcu) pre vytvorenie príkazu pre systém CMake[2]. Podľa zvolených možností, prípadne podľa preddefinovaných, sa postupne vytvára príkaz pre volanie CMake. Príkaz je automaticky spustený z tohto skriptu. Podľa toho, či prebehol príkaz s úspechom, alebo nie sa ešte

používateľovi zobrazí informácia, ako ďalej postupovať. Spracovanie konfigurácie systémom CMake[2] je ovládané súborom *CMakeLists.txt* a ďalšími súbormi, ktoré tento hlavný súbor volá alebo vkladá.

Po uskutočnení úspešnej konfigurácie sa pre vykonanie kompilácie presúvame do adresára s názvom *build*. Často tento adresár spomíname aj v ďalšom texte a vždy používame označenie *build*, a to i v prípade, že bol premenovaný. To sa dá tak, že sa ako posledný parameter pre skript *setup* napíše jeho používateľom želané meno. V adresári *build* nám stačí v tom prípade, že používame MinGW-w64[25] kompilátory a správne sme pre *setup* zadali parameter *--generator="MinGW Makefiles"* spustiť príkaz pre zavolanie programu make: *mingw32-make*

Po skompilovaní programu je ešte dobré overiť jeho funkčnosť spustením testov. Je to možné urobiť príkazom *ctest* (pre jeho parametre viď CMake dokumentáciu[63]). Zoznam testov sa nachádza v súbore *build/CTestTestfile.cmake*. Výsledky po uskutočnení testovania sa nachádzajú v adresári *build/Testing*.

Program DIRAC[1] je spúšťaný pomocou spúšťacieho skriptu, ktorý zabezpečuje vytvorenie prostredia pre jeho prácu (kopírovaním súborov, nastavovaním premenných prostredia). Ide o Python[3] skript *pam*, ktorý sa nachádza v *build* adresári. Využívaním rôznych jeho parametrov je možné ovplyvniť beh programu. Všetky parametre, ktoré skript *pam* ponúka je možné zobraziť: *python pam --help*

Pre ovládanie programu DIRAC je možné použiť i konfiguračný súbor v domovskom adresári používateľa (napr. *C:\Users\<používateľ>*). Súbor musí mať meno *.diracrc* a jeho obsah môže byť napr. aj takýto: *--mb=512*. Jedno nastavenie na jednom riadku. Konkrétne toto nastavenie určuje veľkosť pamäte, ktorú program využíva na 512 MB. Do konfiguračného súboru *.diracrc* sa zapisujú rovnaké parametre ako tie, ktoré akceptuje skript *pam*.

Potrebné softvérové nástroje, ktoré je nutné nainštalovať, aby DIRAC bolo možné skompilovať, príklad konfigurácie, kompilácie, otestovania a príklad prvého výpočtu je možné nájsť v aktuálnej DIRAC dokumentácii na jeho domovskej stránke[1] (odporúčané), alebo aj v CD prílohe tejto práce.

4. Práca vykonaná na projekte

Teraz predstavíme prácu, ktorú sme vykonali na programe DIRAC[1]. Vysvetlíme, ako boli jednotlivé úlohy vyriešené. Pri tomto však opisujeme i funkcionalitu, ktorá sa v projekte nachádzala už skôr. Je to preto, že niekedy naša úloha spočívala v upravení a

rozšírení niektorých častí (najmä) konfigurácie. Toto nie je možné bez toho, aby sa vedelo, ako sú jednotlivé veci riešené a na akom princípe pracujú. Ku konkrétnemu spôsobu riešenia sme prichádzali v spolupráci s vedúcim práce doc. M Iliašom a ďalšími vývojármi DIRAC-u vďaka ich námetom a postrehom (zadaniam).

4.1 Adaptácia generovania dokumentácie

Prvá vec, na ktorej sme začali na programe DIRAC[1] pracovať bola adaptácia generovania dokumentácie pre systém Windows. Dokumentácia (používateľská) je v DIRAC-u založená na *.rst* súboroch. Na ich spracovávanie sa využíva generátor Sphinx[75]. Najprv je potrebné spracovať určité kľúčové slová, ktoré sa v *.rst* súboroch nachádzajú pomocou regulárnych výrazov pre vytvorenie odkazov vo výslednej dokumentácii. Pre tento účel slúži skript *doc/preprocess_sphinx*. Jeho problémom bolo, že bol pôvodne napísaný v GNU bash[76]. Samozrejme, takéto riešenie nie je kompatibilné s natívnym prostredím systému Windows. Adaptácia spočívala v jeho prepísaní do jazyka Python[3], ktorý je na účely multiplatformovej adaptácie vhodný (viď kapitolu o Python-e). Išlo predovšetkým o napodobnenie príkazu sed[77].

Ako už bolo spomenuté, na tvorbu používateľskej dokumentácie sa využíva generátor Sphinx. Vytvorenie tejto dokumentácie je možné spustiť po uskutočnení úspešnej konfigurácie (*python setup ...*) s ľubovolnými nastaveniami v adresári *build* pomocou príkazu *mingw32-make html* (pozn.: treba ale zvoliť vhodný generátor, napr. *--generator="MinGW Makefiles"*). Po úspešnom zbehnutí príkazu sa výsledná dokumentácia nachádza v adresári *build/html* a spustíme ju otvorením stránky *index.html* v internetovom prehliadači.

Na generovanie dokumentácie pre vývojárov (programátorov) je použitý generátor doxygen[78]. Túto dokumentáciu vytvoríme príkazom *mingw32-make doxygen* v adresári *build*. V prípade úspešného vykonania príkazu sa vygenerovaná dokumentácia nachádza v adresári *build/_doxygen*. Máme možnosť využívať dokumentáciu vo formáte *html* (adresár *html*) alebo si môžeme vytvoriť *pdf* súbor (adresár *latex*). Súbor *pdf* vytvoríme napr. pomocou programu MiKTeX[79] spustením dávkového súboru *make.bat*. Toto sa nám podarilo úspešne odskúšať iba na 32-bitovom Windows 7, na 64-bitovom systéme Windows Server 2012 R2 sme neboli úspešní s inštaláciou zmieneného programu. Treba ale dodať, že existujú aj iné podobné programy pre využitie systému TeX[80]. Pre otvorenie *html* dokumentácie slúži súbor *index.html* a výsledný *pdf* súbor je *refman.pdf*. Máme k dispozícii ešte jeden typ dokumentácie – *html* prezentáciu.

Túto je možné si vytvoriť vykonaním príkazu *mingw32-make slides* v *build* adresári. Výsledná prezentácia sa potom nachádza v adresári *build/html_slides* a spustíme ju pomocou internetového prehliadača otvorením súboru *index.html*.

Nástroje, ktoré je potrebné nainštalovať, a spôsob ich inštalácie sa nachádzajú v dokumentácii na domovskej stránke DIRAC-u[1]. Prípadne v dokumentácii, ktorá je priložená na sprievodnom CD - táto je vygenerovaná na Windows.

4.2 Zisťovanie verzií kompilátorov

V DIRAC-u[1] nastala potreba včas (počas konfigurácie vykonávanej systémom CMake[2]) zistiť verzie použitých kompilátorov a v prípade, že niektorý z kompilátorov nespĺňa minimálnu požadovanú verziu, zobraziť o tom varovanie.

Dôvody na to sú zrejmé. Totižto pre každý s použitých programovacích jazykov v DIRAC-u (C, C++, Fortran) sa používa kompilátor a pre každý z nich je známa minimálna podporovaná verzia. Toto sa ďalej ešte rozlišuje podľa výrobcu kompilátora pre GNU[81]/MinGW[41][47][25], Intel[18], PGI[82], IBM XL[83], prípadne pre LLVM Clang[45] (tieto sú ale iba pre C a C++). Na operačnom systéme Windows sa nám podarilo s úspechom odskúšať iba MinGW kompilátory a LLVM Clang. Pokus o kompiláciu programu s Intel kompilátormi bol, ako je spomenuté v kapitole o problémoch s adaptovaním pre Intel, neúspešný.

Pre účely splnenia tohto zadania bolo potrebné vytvoriť súbory v adresári *cmake/compilers*, a to konkrétne: *FindCompilerVersion.cmake* (napísaný v CMake) a *GetCompilerVersion.py* (napísaný v Python-e[3]).

V prvom spomínanom súbore (*FindCompilerVersion.cmake*) sa vykonáva najprv zadefinovanie, aké sú požadované minimálne verzie jednotlivých kompilátorov podľa programovacích jazykov a ich výrobcov. Následne za zavolá (spustí) druhý zo spomínaných súborov (*GetCompilerVersion.py*), pričom sa mu zadávajú potrebné parametre pre jeho prácu. Ide o parametre:

- *CMAKE_C_COMPILER* – použitý kompilátor pre jazyk C
- *CMAKE_C_COMPILER_ID* – výrobca použitého C kompilátora
- *CMAKE_CXX_COMPILER* – použitý kompilátor pre jazyk C++
- *CMAKE_CXX_COMPILER_ID* – výrobca použitého C++ kompilátora
- *CMAKE_Fortran_COMPILER* – použitý kompilátor pre jazyk Fortran
- *CMAKE_Fortran_COMPILER_ID* – výrobca použitého Fortran kompilátora

- *CMAKE_SYSTEM_NAME* – názov systému, na ktorom pracujeme pre prípad, že sa vyskytujú rozdiely pri volaní kompilátorov na rôznych platformách

Pre bližší význam týchto premenných a ďalších použitých príkazov je, samozrejme, možné použiť dokumentáciu systému CMake[63].

Po úspešnom skončení skriptu v súbore *GetCompilerVersion.py* by sme mali dostať ako návratovú hodnotu verzie použitých kompilátor pre jazyky C, C++ a Fortran vo formáte CMake[2] premennej (zoznamu viacerých hodnôt oddelených pomocou bodkočiarky). Tieto verzie si ďalej môžeme spracovať do jednotlivých premenných (*PYTHON_C_VERSION*, *PYTHON_CXX_VERSION* a *PYTHON_Fortran_VERSION*) a následne pracovať s nimi podľa potreby.

Prvé, čo s týmito premennými potrebujeme urobiť, je zapísať ich hodnoty do súboru *COMPILERS_VERSIONS.txt.* Bude sa nachádzať v adresári *build*, aby si používateľ mohol jednoducho overiť, s akými verziami kompilátorov (a od akých výrobcov) si projekt práve skompiloval. Toto by mohlo byť užitočné, ak sa objavia chyby, ktoré sa vyskytujú pri nejakej konkrétnej verzii kompilátora. Po zapísaní hodnôt vyššie spomenutých premenných (získaných z Python-u[3]) nasleduje aj zápis hodnôt verzií kompilátorov, ktoré sa podarilo zistiť systému CMake.

Tu nastáva otázka: „Prečo si vytvárame vlastný skript na zisťovanie verzií kompilátorov, keď to už robí CMake?“ Odpoveď je, že CMake verzie kompilátorov v starších verziách (i také sa s DIRAC-om[1] používajú) nezisťuje vôbec. Napr. ešte vo verzii 2.8.7 by sme premenné typu *CMAKE_<LANG>_COMPILER_VERSION*, ktoré majú obsahovať verzie kompilátorov hľadali márne [84]. Dokonca aj pri súčasných verziách systému CMake nie je garantované, že nájde verziu pre každý kompilátor (viď dokumentáciu na [63]). Všimli sme si, že je takmer pravidlo, že verzie pre kompilátory jazykov C a C++ sú nájdené, ale verzia pre Fortran kompilátor nie je nájdená. Nevideli sme ani jeden prípad, a to ani na systéme Linux s použitím GNU[81], Intel[18] alebo PGI[82] kompilátorov, že by bola verzia pre Fortran kompilátor niekedy nájdená. To je celkom nepríjemná záležitosť, lebo pre DIRAC je práve Fortran najdôležitejší jazyk, a teda o zistenie verzie Fortran kompilátora máme o to väčší záujem.

Zisťovanie verzií kompilátorov neslúži iba na to, aby bol varovaný používateľ (alebo sa zastavil konfiguračný skript), keď verzia niektorého z nich nevyhovuje. Má to aj veľký informatívny význam, keď niekto skúma výsledky konfigurácie, kompilácie a testov, ktoré pochádzajú z iného systému, ku ktorému dotyčný nemá priamy prístup. Vtedy si nemôže verzie kompilátorov jednoducho sám zistiť ich spustením. Takáto situácia nastáva

najmä pri použití rôznych služieb CI, o ktorých sa zmieňujeme v inej časti práce (priebežná integrácia). Z toho dôvodu sú verzie kompilátorov spolu s názvom ich výrobcu pridávané i do výstupu konfigurácie.

Pokračujme s opisom obsahu CMake[2] súboru *FindCompilerVersion.cmake*. Tu sa totižto dostávame k tomu najdôležitejšiemu. K porovnaniu, či verzie aktuálne použitých kompilátorov vyhovujú. Vykonané je to dvomi spôsobmi z už spomenutého dôvodu, že CMake nie vždy sám nájde (všetky) verzie kompilátorov, takže niekedy máme iba verzie zistené pomocou nášho Python[3] skriptu.

V prípade, že máme k dispozícii aj verziu kompilátora zistenú CMake-om, tak ju najprv porovnávame s verziou, ktorú zistil skript v Python-e. V prípade, že nie sú zhodné, tak sa používateľovi zobrazí varovanie. Momentálne je to urobené takto, ale jednoduchou modifikáciou by sa dalo konfiguráciu aj okamžite ukončiť, čo však vývojármi DIRAC-u[1] nebolo želané. Nasleduje porovnanie verzie kompilátora (zistenej systémom CMake) s minimálnou požadovanou verziou. Zisťuje sa, či je aktuálna verzia rovná alebo väčšia ako minimálna požadovaná. V prípade, že nie je, tak je používateľovi zobrazené varovanie (opäť, keby bolo žiaduce, je možné toto správanie jednoducho modifikovať).

V opačnom prípade, keď systém CMake nezistil verziu použitého kompilátora, a teda máme iba verziu zistenú naším skriptom v Python-e, sa mení to, že robíme iba jedno porovnanie. Porovnávame aktuálne zistenú verziu kompilátora s minimálnou požadovanou, či je zistená verzia použitého kompilátora rovná alebo väčšia od minimálnej. Ak verzia kompilátora nevyhovuje, používateľovi je zobrazené varovanie.

Tieto porovnávania sa robia pre všetky jazyky, ktoré využívame (C, C++, Fortran). V prípade, že všetky verzie kompilátorov vyhovujú minimálne požadovaným verziám, pokračuje sa nerušene (bez varovaní, bez zastavenia) v konfigurácii.

Pre úplnosť sa ešte pozrime na to, ako je riešený skript v súbore *GetCompilerVersion.py*. Tento skript, ako už bolo spomenuté, slúži na zistenie verzií kompilátorov a vznikol najmä z dôvodu, že CMake nevie nájsť verzie pre všetky kompilátory, ktoré používame.

Skript potrebuje k svojej činnosti isté parametre. Ide o premenné, ktoré sú mu odovzdávané systémom CMake, keď sa uskutočňuje jeho volanie v už spomenutom súbore *FindCompilerVersion.cmake*. V týchto premenných sú obsiahnuté názvy spolu s úplnými cestami k použitým C, C++ a Fortran kompilátorom. Ďalej reťazce obsahujúce názvy ich výrobcov (vydavateľov). Je využitá i premenná, ktorá obsahuje informáciu o tom, na akom operačnom systéme je konfigurácia práve vykonávaná.

Všetky informácie dostupné z parametrov sú potrebné k tomu, aby mohol byť z tohto skriptu spustený proces, ktorý vykoná príkaz kompilátora. Príkazom je názov kompilátora s celou cestou k nemu a s parametrom, ktorý slúži na zobrazenie jeho verzie (napr. *--version* alebo *-V*). Výstup uskutočneného príkazu (spracovávame štandardný výstup aj štandardný chybový výstup) je použitý na získanie reťazca, ktorý obsahuje verziu použitého kompilátora.

Spracovanie výstupu robíme za pomoci regulárnych výrazov. V prípade, že došlo k chybe alebo bol zistený neznámy výrobca kompilátoru, je za jeho verziu prehlásená verzia nula (0.0 alebo 0.0.0). Táto by nemala za žiadnych normálnych okolností prejsť porovnávaniami oproti požadovanej minimálnej verzii. Porovnávanie sa vykonáva v CMake[2] konfiguračnom súbore, z ktorého bol tento skript volaný. Návratová hodnota z nášho Python[3] skriptu pozostáva z verzií kompilátorov pre jazyky C, C++ a Fortran, ktoré sú oddelené pomocou bodkočiarky. Takýto formát výstupu zodpovedá premennej CMake s viacerými hodnotami, a teda je takto umožnené jeho ďalšie spracovanie jednoduchým spôsobom.

Zisťovanie verzií kompilátorov je zaradené do procesu konfigurácie vložením súboru *FindCompilerVersion.cmake* v CMake konfiguračnom súbore, ktorý sa nachádza v *cmake/compilers/ConfigCompilerFlags.cmake*.

4.3 Úprava nastavení kompilátorov pri spustení konfigurácie

Pôvodne sa možnosti pre kompilátory (*flags*) nastavovali iba automaticky podľa konfigurácie, ktorá bola napevno napísaná v súboroch *cmake/compilers/CFlags.cmake*, *cmake/compilers/CXXFlags.cmake* a *cmake/compilers/FortranFlags.cmake*.

Neexistoval teda jednoduchý spôsob, ako pri spustení skriptu *setup* nastaviť nejaké iné nastavenia pre kompilátory, ktoré by lepšie vyhovovali momentálnym potrebám.

Navyše možnosť urobiť niečo také môže pritom niekomu, kto chce experimentovať s nastaveniami kompilátorov veľmi chýbať. Keby ju mal, nemusel by potom prepisovať súbory, ale stačilo by mu, keby si zadal svoje želané nastavenia na príkazovom riadku pri spúšťaní konfigurácie.

Toto sa nám podarilo vylepšiť zavedením nových parametrov pre *setup* skript a rozšírením jeho funkčnosti. Pridali sme v ňom tieto parametre:

- *--custom-fc-flags* – pre zadanie vlastných možností pre kompilátor jazyka Fortran
- *--custom-cc-flags* – pre zadanie vlastných možností pre kompilátor jazyka C
- *--custom-cxx-flags* – pre zadanie vlastných možností pre kompilátor jazyka C++

Na uskutočnenie týchto zmien bolo potrebné vykonať relatívne jednoduché zásahy (pridania) do súboru *setup*, a aj do už vyššie spomenutých súborov v *cmake/compilers*.

V súbore *setup* sa pridala nová skupina argumentov, ktorá obsahuje tri argumenty pre nastavenie možností kompilátora každého použitého jazyka. Rozpoznávanie argumentov radi *ArgumentParser* (viď dokumentáciu pre Python[3] na [69]). Následne pri vytváraní príkazu, ktorý bude spustený systémom CMake[2] sa pridáva definícia *-DCUSTOM_C_FLAGS* pre nastavenia možností kompilátora jazyka C, definícia *-DCUSTOM_CXX_FLAGS* pre nastavenie možností kompilátora pre C++ a definícia *-DCUSTOM_Fortran_FLAGS* pre nastavenie možností Fortran kompilátora.

V súboroch nachádzajúcich sa v *cmake/compilers* bolo potrebné urobiť vetvenie pre prípad, keď sú zadané vlastné možnosti. Za takých okolností bolo potrebné „vymazať“ možnosti pre rôzne typy kompilácií (*debug*, *release*, *profile*) a vždy vo všetkých prípadoch používať iba používateľom explicitne zadané nastavenie.

Pre záujemcov môžeme ešte pripomenúť, že existuje a predtým aj existovala možnosť pridávania (nie úplného prepísania ako pri našej úprave) dodatočných nastavení pre kompilátory pomocou extra možností, ktorých autorom je Dr. R. Bast[67]. Nastavujú sa týmito parametrami:

- *--extra-fc-flags* – pre nastavenie extra možností pre Fortran kompilátor
- *--extra-cc-flags* – pre nastavenie extra možností pre C kompilátor
- *--extra-cxx-flags* – pre nastavenie extra možností pre C++ kompilátor

4.4 Skript na opätovné spustenie neúspešných testov

Množstvo vecí, ktorým sme sa na projekte DIRAC[1] venovali, sa týkalo testovania, ktoré je preň dôležité. Na účely testovania slúži aj skript v jazyku Python[3] *maintenance/get_failing.py*. Tento skript má za prvoradú úlohu umožniť vývojárom jednoduché spustenie posledne spadnutých (neúspešných) testov, čo im dovoľuje lepšie sa sústrediť na problematické časti programu.

Existujú dve možnosti jeho práce. Prvá je, že sa spustia testy, ktoré naposledy spadli lokálne na počítači (a v danej kompilácii programu – v konkrétnom adresári *build*), kde sa skript spúšťa. Od verzie 3 a vyššie je toto podporované aj priamo systémom CMake[2], konkrétnejšie jeho súčasťou CTest, pri využití možnosti na príkazovom riadku *--rerun-failed* (viď dokumentáciu[63]). Druhá možnosť je (táto je možno zaujímavejšia) spustiť u seba testy, ktoré skončili s chybou niekde inde.

To, že máme možnosť vedieť o testoch, ktoré sú problematické niekde inde, nám

umožňuje *cdash* web stránka. DIRAC[1] má svoju na [85]. Tu sú zverejňované výsledky (v niektorých prípadov aj z aktualizácie - *Update*) z konfigurácie - *Configure*, kompilácie - *Build* a z testovania - *Test* uskutočnených na viacerých serveroch. Napr. na Windows serveri Katedry chémie prebiehajú testy na natívnom Windows a v prostredí Cygwin[24]. Ďalej sú tu výsledky z CI služieb, o ktorých sa zmieňujeme v časti práce o priebežnej integrácii. Žiadna zo spomenutých častí (*Update*, *Configure*, *Build*, *Test*) nie je povinná a na zobrazenie v tabuľke s výsledkami stačí záznam aj z jednej z nich.

Pozrime sa na skript podrobnejšie. Prvotnú ideu na jeho napísanie mal Dr. R. Bast[67], ktorý na ňom začal pracovať, ale pre nedostatok času sa nakoniec ušla jeho realizácia (zmena spôsobu práce a rozšírenie) nám. Ako už bolo spomenuté, skript je schopný pracovať dvomi spôsobmi, a to lokálne alebo využívajúc informácie o neúspešných testoch z internetu. Jeho spustenie a zároveň výber možnosti práce lokálne alebo z internetu je uskutočnené príkazom zadaným z adresára *build*, takto:

```
python ..\maintenance\get_failing.py buildid|local "param1 param2 ..."|none
```

V prípade, že je zadané nastavenie *local*, vykonáva sa čítanie súboru *build/Testing/Temporary/LastTestsFailed.log*, ktorý v sebe obsahuje (ak je prítomný) na jednotlivých riadkoch informácie o posledne spadnutých testoch vo formáte *číslo:názov.*

Nás vždy zaujíma číslo testu, takže čítame súbor po riadkoch a zaznamenávame si čísla testov. Následne pridáme medzi čísla testov čiarku (aby vyhovovali požiadavkám CTest-u[2]), zobrazíme a v ďalšom príkaze aj spustíme CTest príkaz s nimi (prípadne aj so zadanými dodatočnými parametrami) pomocou Python[3] príkazu *subprocess.call()*. Pri tomto využívame CTest parameter *-I*, ktorý obsahujú aj verzie CMake staršie ako 3. Od verzie 3 a vyššie môžeme, ako bolo uvedené, používať pre podobnú funkcionalitu aj možnosť *--rerun-failed* [63]. Pre prehľadnosť skriptu je tento postup pre lokálne spracovanie zhrnutý vo funkcii *local()*.

Alternatíva je, že namiesto *local* je zadané nejaké *buildid* zo *cdash* webu[85]. To znamená, že sa budú načítavať informácie o neúspešných testoch z internetu pre zadané *buildid*. Na stiahnutie stránky s informáciami o problematických testoch využívame knižnicu napísanú v jazyku Python - Requests[86]. Túto je možné využívať až od Python verzie 2.6 a často ju treba dodatočne inštalovať, čo je ale v skripte ošetrené a používateľovi sa zobrazí informácia s návodom, ako má postupovať.

Teraz je spôsob práce skriptu taký, že zo stiahnutých údajov stránky vyhľadávame pomocou regulárnych výrazov záznamy o testoch a ukladáme si ich názvy. Keďže názvy testov nám pri spúšťaní CTest-u s požadovaným parameterom *-I* nepomôžu, stojíme pred

problémom, odkiaľ získať ich čísla, ktorým CTest[2] rozumie. Avšak, máme k dispozícii súbor s testami *build/CTestTestfile.cmake*, ktorý slúži pre CMake/CTest ako konfigurácia pre uskutočnenie testovania. Tu sú testy uvedené v poradí, v ktorom majú byť spustené, takže nám stačí tento súbor čítať po riadkoch a počítať si, koľko testov už bolo pridaných. Počítame výskyty príkazu *add_test()*, až pokiaľ nenájdeme riadok s príkazom takým, kde sa pridal test s menom, pre aké hľadáme poradové číslo. Toto uskutočníme pre všetky mená testov, ktoré sme zistili zo *cdash* stránky[85].

Potom si kvôli prehľadnosti získané čísla zoradíme a vytvoríme z nich raťazce (*str*) namiesto číslic (*int*). Následne máme možnosť pokračovať rovnakým spôsobom ako v prípade spracovávania lokálnych testov. Teda pridáme čiarky medzi čísla, zobrazíme používateľovi príkaz a spustíme ho ako podproces. Táto časť skriptu je definovaná vo funkcii *dashboard()*.

Ako bolo vyššie vidieť, skript požaduje dva argumenty, a to konkrétne (*buildid* alebo *local)* a (*"param1 param2 ..."* alebo *none)*. Prvý sme si už prebrali, keď sa použije *buildid* zo *cdash* webu, informácie o testoch sa načítajú z internetu pre daný *buildid*. Keď je zadané *local*, čítajú sa informácie z lokálneho súboru o lokálne spadnutých testoch. Druhý argument *"param1 param2 ..."|none* hovorí o tom, že máme možnosť (ale nemusíme, keď napíšeme iba *none*) zadávať CTest-u ďalšie parametre. Ako je i uvedené v zdrojovom kóde skriptu, môže ísť o parametre určujúce počet súčasne bežiacich testov, t.j. *-j* (*jobs*) alebo *-V* (*verbose*) pre detailný výpis počas testovania. Viď CMake/CTest dokumentáciu[63] pre ďalšie parametre. Užitočné by mohlo byť aj odoslanie výsledkov testovania na *cdash* web pomocou *-D ExperimentalTest -D ExperimentalSubmit* uvedených za sebou v úvodzovkách.

Skript má svoje limity. Do zoznamu testov nie je zahrnutý test pre *xcfun*. To platí pri využívaní funkcionality pre načítavanie testov z internetu, pretože *xcfun* sa nenachádza v súbore *build/CTestTestfile.cmake* uvedený v príkaze *add_test()*. Je uvedený v *subdirs()* a až v súbore *build/src/xcfun/test/CTestTestfile.cmake* je pre tento projekt pridaný test príkazom *add_test()*. Ojedinele sme si všimli, že sa môže stať, že niektoré testy zaznamenané na *cdash* webe sa nemusia nájsť lokálne (boli zapnuté aj iné testy alebo sa testy zmenili). Treba si uvedomiť, že nie je možné použiť možnosť *local*, keď predtým ešte nikdy neboli spustené žiadne testy (hľadaný súbor neexistuje) alebo v prípade, že sa vždy používala iba niektorá z testovacích možností *-D Experimental* a pod. Vtedy názov súboru *build/Testing/Temporary/LastTestsFailed.log* obsahuje aj časovú značku a nebude nájdený.

4.5 Zisťovanie dostupnosti xHost pre Intel (použitie najvyššej inštrukčnej sady)

Čo sa týka nastavení (*flags*) pre kompilátory, treba sa zmieniť aj o *xHost*. Táto možnosť je určená pre Intel[18] kompilátory, ale nie vždy je možné ju využiť. Slúži na ten účel, aby kompilátor generoval inštrukcie pre najvyššiu inštrukčnú sadu, ktorá je dostupná na procesore, na ktorom sa vykonáva kompilácia. Týmto očakávame vyššiu optimalizáciu a výkon programu. Ako sme ale spomenuli, nie všade je možné túto možnosť využívať. Na niektorých procesoroch síce kompilácia prebehne bez problémov, výsledný program však nie je možné úspešne spustiť.

Keďže sa predsa len ukázalo (na základe testovaní, ktoré uskutočnili iní vývojári), že by bolo prínosné používať *xHost* tam, kde je to možné, bolo potrebné vytvoriť nejaký systém rozhodovania. Taký, ktorý by automaticky pridával *xHost* medzi možnosti kompilátora. Samozrejme, na tých systémoch, kde je predpoklad, že výsledný program bude aj bezproblémovo schopný pracovať.

Či je bezpečné pridať túto možnosť sme sa rozhodli testovať pomocou kompilácie a spustenia jednoduchého testovacieho programu. Ide o to, že keď odskúšame skompilovať tento jednoduchý program a následne ho spustíme, sú iba dve možnosti. Buď sa nám ho spustiť podarí alebo nie. Kompilácia totiž prebieha aj na systémoch, ktoré možnosť *xHost* nepodporujú v poriadku.

Testovací program sa nachádza v súbore *cmake/compilers/test_xhost_flag.f90* a jeho obsah je nasledovný:

```
! test program to see whether it is possible to use Intel -xHost flag
program test
print *, 'Intel xHost flag is available on this computer.'
end program
```

Nič zvláštne tam nie je, ale v prípade, že ho skompilujeme s *xHost* nastavením a úspešne spustíme, dosiahneme to, že vypíše hlášku (namiesto chybovej): *Intel xHost flag is available on this computer.* Vtedy budeme vedieť, že na danom počítači je možné *xHost* nastavenie používať.

Overovanie je naprogramované nasledovne (všetky súbory, ktoré sa tohto týkajú sa nachádzajú v *cmake/compilers*):

V konfiguračnom súbore *ConfigCompilerFlags.cmake* sa vkladá súbor *ConfigXHostFlag.cmake*, čím sa zároveň aj zabezpečí vykonanie príkazov, ktoré sa v ňom nachádzajú. Vo vloženom súbore sa pomocou CMake[2] príkazu *execute_process()* vykonáva kompilácia testovacieho programu a jeho spustenie. Štandardný výstup aj

štandardný chybový výstup je zaznamenaný do premennej *_xhost_binary_output*. Potom sa už iba porovná obsah tejto premennej s očakávaným výstupom, a ak vyhovuje (*MATCHES*), nastaví sa premenná *XHOST_FLAG_AVAILABLE* na hodnotu *ON*. Týmto sa zapne pridávanie *xHost* nastavenia medzi možnosti kompilátorov pre C, C++ a Fortran.

Predpokladáme, že keď bolo úspešné testovanie *xHost* pre Fortran, znamená to, že aj pre C a C++ by mala byť táto možnosť pre Intel[18] kompilátory na práve používanom procesore dostupná.

Vlastné pridávanie alebo nepridávanie tejto možnosti je zabezpečené podľa hodnoty premennej *XHOST_FLAG_AVAILABLE* v súboroch určených pre nastavenia kompilátorov *CFlags.cmake*, *CXXFlags.cmake* a *FortranFlags.cmake*. Možnosť *xHost* sa pridáva iba pre profil kompilácie *release* (a prenesene do profilu *profile*), pretože práve pri tomto profile chceme dosiahnuť najvyššiu optimalizáciu.

Je potrebné ešte dodať, že toto testovanie dostupnosti *xHost* je už adaptované aj pre systémy Windows. Bolo odskúšané, dokonca aj možnosť *xHost* bola dostupná, a to na procesore AMD. Je tomu tak i napriek tomu, že na Windows sa kompilácia DIRAC-u[1] Intel kompilátormi nepodarila (problémy so zostavovaním, venujeme sa tomu v inej časti práce). Na Windows sa táto možnosť pridáva pomocou */QxHost* a na systémoch Linux pomocou *-xHost*.

Do riešenia tohto problému následne prispievali aj doc. M. Iliaš a Dr. R. Bast[67]. Aj napriek prínosu, ktorý táto možnosť má na podporovaných procesoroch sa s ňou vyskytli problémy a bolo potrebné deaktivovať jej automatickú kontrolu a pridávanie. Je ale možné si to znova zapnúť: *-D ENABLE_XHOST_FLAG_DETECTION=ON*. Dôvodom je, že na niektorých výpočtových klastroch pozostávajúcich z počítačov s rôznymi procesormi sa vyskytovali (boli hlásené) problémy.

Informácie o možnosti *xHost*, spôsobe jej pridávania, aké inštrukčné sady sa pri nej použijú a na akých procesoroch bude výsledok schopný pracovať podľa toho, na akom procesore prebehla kompilácia je možné nájsť na [87][88][89].

4.6 Oprava vyhľadávania báz na Windows

Vyšlo najavo, že na Windows 7 (32-bitový operačný systém) končí s neúspechom neobvykle veľa testov v porovnaní s inými systémami. Výsledky boli lepšie nielen na systémoch Linux, ale aj na Windows Server 2012 R2, ktorý je 64-bitový.

Nakoniec sme poslali výsledky z testovania na *cdash* web[85], aby mohli byť prezreté vedúcim práce doc. M. Iliašom, lebo takéto problémy by sa nemali vyskytovať ani

na 32-bitovom systéme. Ukázalo sa, že program má problémy pri svojom behu počas testov nájsť umiestnenia *basis* (umiestnenia adresárov *basis*, *basis_dalton*, *basis_ecp*). Tieto adresáre *basis* sú prekopírované zo zdrojových súborov do adresára *build*, kde ich program aj obvykle očakáva.

Pri skúmaní, čo by mohlo byť riešením tohto problému, sme pracovali najmä s konfiguračným súborom, ktorý DIRAC[1] používa. Súbor má názov *.diracrc*, a aby ho program vedel nájsť, mal by byť umiestnený v domovskej zložke používateľa (je/bola ale aj možnosť, že môže byť umiestnený v adresári *build*). V prostredí systému Windows je domovská zložka vo väčšine prípadov umiestnenie *C:\Users\<používateľ>*.

Experimentálne sa nám podarilo zistiť, že problém je v tom, že DIRAC nevie nájsť umiestnenia *basis* v tom prípade, keď sa nachádzajú na inej diskovej jednotke, ako sa nachádza pracovný adresár *scratch*. V tomto adresári sa vykonáva práca programu (sem je program skopírovaný a tu je spustený, to platí aj pri testovaní). Presne to bol prípad na Windows 7, kde sa nachádzali *basis* na jednotke *D* (*D:\Dirac\my-dirac\build*), ale pracovný adresár *scratch* sa nachádzal v domovskom adresári (kde sa zvykne nachádzať, pokiaľ si používateľ nenastaví inak) na jednotke *C*.

Príčinou tohto problému bolo najmä to, že v cestách, ktoré sa používajú na systéme Windows, sa nachádza dvojbodka na oddelenie označenia jednotky od zvyšku cesty. Avšak, dvojbodka je používaná ako oddeľovací znak ciest (celých ciest) na systémoch Linux. Takto tomu rozumel aj DIRAC. Z toho vyplýva, že program hľadal umiestnenia *basis*, ak mu bolo zadané, že sa nachádzajú v *D:\Dirac\my-dirac\build\basis* namiesto na jednom správnom na dvoch nesprávnych umiestneniach. Konkrétne prehľadával umiestnenia *D* a *\Dirac\my-dirac\build\basis* na pracovnej jednotke. Keďže pracovný adresár *scratch* sa nachádzal na jednotke *C*, tak na jednotke *C*. Čo nie je pravda, lebo *basis* sa nachádzali na jednotke *D*. Toto je dôvod vzniknutej chyby.

Je zaujímavé, že Fortran (podľa našich postrehov s MinGW[41][47][25] Fortran kompilátorom) predpokladá, že cesty odkazujú na umiestnenia, ktoré sa nachádzajú na tej istej jednotke, odkiaľ je program spustený. Neprekážajú mu ani rozdielne lomky v cestách. Na Windows sa môžu použiť také ako na systéme Linux, prípadne nevadí ani ich miešanie.

Vzniknutý problém sme sa rozhodli vyriešiť zmenou toho, ako sa v DIRAC-u uvažuje o jednotlivých cestách. Teda na systéme Windows nepoužívať na oddeľovanie ciest znak dvojbodka, ale použiť znak bodkočiarka. Takýmto spôsobom zvyknú byť oddeľované viaceré hodnoty na Windows aj v premenných prostredia. Súbor, ktorý bolo potrebné upraviť je *pam* (*pam.in*).

Tu sme zmenili spôsob vytvárania premennej *self.basdir* a aj spôsob, akým je rozložená na jednotlivé cesty (*split*), keď majú byť všetky vypísané na obrazovku používateľovi. V oboch prípadoch je na oddeľovanie použitý znak bodkočiarka v prípade, že sa pracuje na systéme Windows. Na iných systémoch zostala použitá dvojbodka.

Obsah *self.basdir* je skriptom exportovaný do premennej prostredia, ktorá je platná počas vykonávania programu. Jej názov je *BASDIR*. Exportovanú premennú *BASDIR*, samozrejme, program niekde číta. Z toho dôvodu bolo potrebné vykonať zmeny aj v niektorých Fortran zdrojových súboroch: *src/abacus/herbas.F*, *src/abacus/herrdn_dirac.F*, *src/ecp/recp_lnk.F*. Tu bolo potrebné urobiť spracovanie načítanej premennej *BASDIR* takým spôsobom, že sa budú cesty oddeľovať pomocou bodkočiarky na systéme Windows. Na systémoch Linux a iných sa zachovalo spracovanie pomocou dvojbodky.

Pri tej príležitosti sme upravili (v dotknutých podprogramoch), aby sa na Windows používala na ukončenie cesty lomka, ktorá je pre tento systém normálne bežná (t.j. \), i keď sme nepostrehli, že by s tým boli nejaké problémy. Využili sme premennú *SLASH*, ktorú sme si naplnili správnou lomkou podľa toho, na akom systéme sa pracuje.

Rozvetvenie spracovania bolo vykonané pomocou logickej premennej *WIN*, ktorej hodnota sa nastavila podľa definície pre preprocesor *SYS_WINDOWS*. Keď je definovaná *SYS_WINDOWS*, tak je hodnota premennej *.TRUE.*, inak je jej hodnota *.FALSE.*

Pri práci so zdrojovými kódmi v jazyku Fortran sme sa stretli s niektorými obmedzeniami. Napr. dĺžky názvu premennej, deklarácia premennej musí byť pred kódom, ktorý „už niečo robí“, odriadkovanie príkazov 6 medzier od okraja a pod.

Našťastie, sú na internete zdroje, ktoré pomôžu s riešením problémov aj v neznámom jazyku [90][91][92][93][94].

4.7 Adaptácia modulu PCMSolver

Po pridaní projektu PCMSolver[39] do DIRAC-u[1] a vyriešení problémov s jeho sťahovaním pomocou Git-u[37] ako externého modulu počas kompilácie (problémy s prihlasovacími právami), sme prišli na to, že DIRAC nie je možné skompilovať, keď je tento modul zapnutý.

Podarilo sa nám zistiť, že chyba bola zrejme v kompilátoroch, ktoré sme do tej doby používali. Používali sme kompilátory MinGW[41] na 32-bitovom a kompilátory TDM-GCC[47] na 64-bitovom systéme Windows. Ani na jednom systéme nám kompilácia nedokázala prejsť (pričom chybová hláška sa netýkala problémov so zdrojovými kódmi alebo s nedostupnými knižnicami a pod.). Kompilátory, s ktorými sme nemali žiadne

takéto problémy sme našli až v projekte MinGW-w64[25]. Tento projekt poskytuje kompilátory pre 32-bitové aj 64-bitové systémy Windows. Prešli sme na používanie týchto kompilátorov na všetkých systémoch.

Ani ich použitím sa však hneď PCMSolver[39] nepodarilo kompletne skompilovať. Tu sa už ale objavili náznaky toho, že sú problémy v konfigurácii, lebo boli hlásené nenájdené príkazy. Tie príkazy boli z prostredia Linux, takže bolo zrejmé, že bude potrebné adaptovať konfiguračné skripty pre systém Windows.

Ukázalo sa, že problémy nastávajú pri spájaní (*merge*) statických knižníc, ktoré bolo vyriešené pomocou príkazov z Linux-u (*ls*, *cat*, *tr*, *sed*, viď napr. [60]).

My sme sa pokúsili nahradiť tieto príkazy najprv príkazmi z Windows PowerShell-u[95] na systéme Windows, s čím sme neboli úspešní a navyše takéto riešenie by nebolo univerzálne pre všetky systémy. Z toho dôvodu sme nakoniec zvolili riešenie pomocou jazyka Python[3]. Jeho interpreter by mal byť dostupný na všetkých systémoch, na ktorých chcú používatelia pracovať či už s PCMSolver-om, alebo s DIRAC-om[1] (už len ich konfigurácia skriptom *setup* je napísaná v Python-e).

Súbor, ktorý bolo potrebné upraviť je *cmake/MergeStaticLibs.cmake*. Tu sa vykonávalo rozbalenie pôvodných statických knižníc a zaznamenanie ich obsahu (aké objektové súbory obsahovali) do súboru. Obsah tohto súboru bol vytváraný príkazom *ls*, ktorého výstup bol do neho presmerovaný (obsah bol dokonca zoradený abecedne, lebo príkaz *ls* sa tak zvykne správať). Fakticky išlo o rozbalenie knižníc do nejakého adresára (vykonané o jeden príkaz skôr, čo bolo v poriadku) a „vylistovanie“ jeho obsahu. Proces sa spúšťal pomocou CMake[2] príkazu *EXECUTE_PROCESS()*. My sme prepísali funkčnosť pôvodného Linux-ového príkazu do Python-u nasledovne:

```
import os; print('\\\\n'.join(sorted(os.listdir(os.curdir),key=str.lower)))
```

Tento predpis je možné spustiť pomocou interpretera jazyka Python, keď sa použije parameter *-c*. V CMake kóde to vypadá takto (skrátené):

```
EXECUTE_PROCESS(COMMAND ${PYTHON_EXECUTABLE} -c  \"import os; …
```

Týmto sme vyriešili prvý problém so statickými knižnicami.

To však nebola jediná potrebná úprava. Keďže sa knižnice rozbaľujú za účelom ich následného spojenia do jednej, tak bude ešte nevyhnutné vyriešiť ich spájanie. Táto činnosť je taktiež riadená zo súboru *cmake/MergeStaticLibs.cmake*. Spájanie knižníc (pridávanie objektových súborov z ostatných do jednej) sa vykonáva pomocou archivačného nástroja. Tento bol použitý i na ich rozbalenie. Príkladom je program *ar*[60] dostupný na systémoch Linux, ale s MinGW[41][47][25] kompilátormi aj na Windows.

V jeho použití by problém nebol. Ale v tom, že vstupy mu boli odovzdávané pomocou príkazov *cat*, *tr* a *sed* už áno. Príkaz *cat* bol použitý na čítanie súborov, ktoré obsahovali zoznamy s objektovými súbormi na pridanie (ich vytvorenie bolo už opísané vyššie). Príkaz *tr* na odstránenie ukončovacieho znaku riadku a *sed* na nahradenie prípadných výskytov __.*SYMDEF SORTED* (to sa týka iba niektorých systémov).

Tento problém sme vyriešili tak, že namiesto volania priamo archivačného nástroja zavoláme skript napísaný v Python-e[3], ktorý všetky spomenuté (tie čo sú potrebné) úkony vykoná univerzálnym spôsobom. Uvedený skript sa nachádza v súbore *cmake/MergeStaticLibs.py*. Jeho parametre, ktoré sú mu odovzdávané zo CMake[2] sú plná cesta k archivačnému programu, knižnica, ktorá sa má vytvoriť, a súbor so zoznamom objektových súborov, ktoré majú byť do knižnice pridané.

My sme súbor so zoznamom objektových súborov prečítali po riadkoch, skúsili sme odstrániť __.*SYMDEF SORTED* a vytvorili sme si pole s objektovými súbormi. Potom sme pre každý objektový súbor zavolali archivačný program, aby ho pridal do knižnice. Pri každom jednom takomto spracovaní (pridávaní objektových súborov, ktorých zoznam sa nachádza v jednom súbore) sa zobrazuje používateľovi správa o upravovanej knižnici a pridávaných objektových súboroch. Výsledná knižnica má názov *libpcm.a* (prípona *.a* sa používa na Linux-e a na systéme Windows, keď sa používa MinGW[41][47][25]).

Je dobré pripomenúť, že PCMSolver[39] priniesol so sebou aj nové požiadavky (závislosti) na nainštalovaný softvér (knižnice) na cieľovom systéme. Ide o knižnice Boost[96][97] pre C++ a kompresnú knižnicu zlib. V dokumentácii pre program DIRAC[1] sme pre používateľov vytvorili návod, ako si majú knižnice Boost skompilovať a nainštalovať pri použití MinGW-w64[25] kompilátorov. Knižnicu zlib sme už našli skompilovanú na internete, a to buď na jej domovskej stránke[98] alebo v MinGW-w64 projekte pre 64-bitové systémy Windows na [99]. O obidvoch možnostiach sú používatelia informovaní v dokumentácii (nachádza sa na [1] alebo i na priloženom CD).

Za istú pozornosť stojí aj to, že sme pridali definície pre systém Cygwin[24] (zadefinovali sme, že je Linux) do súboru *cmake/ConfigArchitecture.cmake*. Urobili sme to pri príležitosti pokusov s týmto prostredím, keď vyšlo najavo, že bez definícií sa nám nepodarí skompilovať DIRAC na Cygwin-e (viď kapitolu o Cygwin v časti o práci vykonanej na projekte). Momentálne je to síce tak, že PCMSolver tieto definície nepotrebuje, ale takto aspoň udrží krok so zvyškom DIRAC-u.

Hlavným vývojárom PCMSolver-a je Dr. Roberto di Remigio[100]. Zdrojový kód je šírený pod *open source* licenciou a dostupný na [39].

4.8 Výpis konfiguračných informácií

Ďalšou dôležitou úlohou, ktorú sme mali bolo, že sme zväčšili rozsah informácií, ktoré sú zobrazované používateľovi počas konfigurácie alebo spustenia programu. Tieto informácie vypovedajú o prostredí, v ktorom bol DIRAC[1] skompilovaný.

To znamená, že sa týkajú použitého operačného systému, použitých kompilátorov (a ich verzií), verzie systému CMake[2] a verzie interpretera jazyka Python[3]. Taktiež dôležité je vedieť aj o type procesora a nastaveniach konfigurácie. Napríklad, či je zapnuté využitie 64-bitového typu *integer*, či je zapnuté použitie MPI[4] (viď kapitolu o MPI), aké sú definície určené pre preprocesory kompilátorov a ktoré knižnice sú použité. Nechýba ani informácia o tom, kto program skompiloval (meno používateľa) a názov počítača, na ktorom sa kompilovanie programu uskutočnilo.

Nastavenie (získanie) informácií, ktoré budú „zabudované“ do programu (zobrazované pri spustení *dirac.x.exe* z príkazového riadku) prebieha tak, že sa najprv nakonfiguruje súbor *cmake/binary-info/binary_info.py.in*. Konfiguráciou sa tu myslí to, že premenné uvedené v @@, ktoré sa v predmetnom súbore vyskytujú, budú CMake-om nahradené ich hodnotami. Z pôvodného súboru, ktorý sa nachádza v adresári obsahujúcom zdrojové kódy (*source*) sme takto získali nový súbor *binary_info.py.* Tento sa ale už nachádza v adresári, kde bude prebiehať kompilácia (*build*). Na konfiguráciu je použitý CMake príkaz *configure_file()*.

Nasleduje zadefinovanie úlohy (*target*), ktorú má vykonať riadiaci program pre kompiláciu (napr. *mingw32-make*), príkazom *add_custom_target()*. Ide o to, že súbor *binary_info.py* sa spustí interpreterom jazyka Python, a tak získame pomocou neho nový zdrojový súbor v jazyku Fortran - *binary_info.F90*. Je si to možné predstaviť takým spôsobom, že vykonaním súboru v Python-e sa zobrazia na štandardný výstup príkazy vo Fortran-e (ktoré sú v Python súbore/skripte „napísané“) a tieto sú presmerované do zdrojového súboru v jazyku Fortran. V tej chvíli môžeme súbor *binary_info.py* vymazať z adresára *build*. Fortran zdrojový súbor *binary_info.F90* sa použije pri kompilovaní programu, a tým bude do neho zahrnutý. Tu spomenutý postup je naprogramovaný v konfiguračnom súbore *cmake/binary-info/BinaryInfo.cmake*, ktorý bolo potrebné pri našej práci taktiež mierne upraviť.

Informácie, ktoré majú byť zobrazené počas konfigurácie DIRAC-u (*python setup ...*) sa nastavujú v súbore *cmake/ConfigInfo.cmake*. Súbor je vložený do spracovania systémom CMake v jeho hlavnom konfiguračnom súbore *CMakeLists.txt* (pozn.: účelom *setup* je nakoniec spustiť CMake s požadovanými parametrami).

V hlavnom CMake[2] konfiguračnom súbore však ešte predtým, než tak urobíme, musíme naplniť hodnotami niektoré premenné. Tieto premenné obsahujú informácie, ktoré máme v úmysle zobraziť používateľovi počas konfigurácie. Konkrétne ide o premenné *BLAS_LIBRARIES_INFO*, *LAPACK_LIBRARIES_INFO* a *EXPLICIT_LIBS_INFO*, ktoré sme zaviedli, aby sme pomocou nich mali možnosť podať informáciu o tom, aká knižnica BLAS[42]/LAPACK[43] je použitá (*builtin*, *explicit*), a aké knižnice (a či vôbec nejaké) boli zvolené explicitne - *EXPLICIT_LIBS*.

Informácie o BLAS/LAPACK knižniciach môžu nadobúdať hodnoty *builtin*, *explicit in <umiestnenie>* alebo *none*. Tieto premenné sa nastavujú iba v prípade, že sa pri konfigurácii nepodarilo nájsť (prípadne sme tomu zabránili ich manuálnym vybratím) BLAS alebo LAPACK knižnice automaticky. Priradené hodnoty sú určené v závislosti od hodnoty premennej *BLAS_LIBRARIES* alebo hodnoty premennej *LAPACK_LIBRARIES*. Do premennej *EXPLICIT_LIBS_INFO* sa nastavuje hodnota *none*, keď nemáme žiadnu knižnicu v premennej *EXPLICIT_LIBS* (čo sa stáva zrejme vždy, lebo táto premenná pravdepodobne nie je v DIRAC-u[1] momentálne využívaná).

Zavedením nových (*_INFO*) premenných nám išlo o to, aby bola aj v prípade, že premenná, o ktorej informujeme, nemá hodnotu (je prázdna), podaná správa používateľovi. Príkladom je *BLAS_LIBRARIES*. Keď sa podarí automaticky nájsť BLAS knižnicu na systéme, tak v tejto premennej bude ako hodnota celá cesta k nej. Ale v prípade nastavenia, že chceme využívať zabudovanú knižnicu (*builtin*) alebo si konkrétne (*explicit*) nejakú vyberieme, premenná je prázdna. Preto využijeme premennú *BLAS_LIBRARIES_INFO* na to, aby sme do nej nastavili hodnotu *none* alebo *explicit* (vrátane umiestnenia zvolenej knižnice).

V súbore *CMakeLists.txt* ďalej nastavujeme premennú *CONFIGURATION_TIME*, ktorá obsahuje čas spustenia konfigurácie v UTC. Čas zisťujeme pomocou jazyka Python[3]. Tento prístup je univerzálny na všetkých používaných systémoch a navyše uskutočniteľný i so staršími verziami systému CMake. Iná možnosť použiť CMake príkaz *string(TIMESTAMP)* bola zavedená až vo verzii 2.8.11 (viď dokumentáciu[63]). Až po získaní definícií pre preprocesory kompilátorov všetkých zdrojových súborov do premennej *_list_of_definitions* sa dostávame k okamihu, keď môžeme vložiť (a tým aj vykonať) spomínané súbory *BinaryInfo(.cmake)* a *ConfigInfo(.cmake)*.

V tomto poradí je to urobené preto, lebo *BinaryInfo.cmake* okrem toho, že zisťuje *USER_NAME* (meno používateľa) a *HOST_NAME* (názov hosťovského počítača) ešte vkladá (vykonáva) aj súbor *cmake/binary-info/ConfigGitRevision(.cmake)*.

Tento súbor zhromažďuje informácie o poslednej zmene (*commit*) v premennej *GIT_REVISION* (*hash* revízie), ďalej o jej autorovi v *GIT_LAST_COMMIT_AUTHOR*, dátume (*GIT_LAST_COMMIT_DATE*) a o pracovnej vetve (*GIT_BRANCH*). V adresári *build* sa nachádza súbor *GIT_STATUS_AT_BUILD*. Jeho naplnenie informáciami o stave pracovnej kópie zdrojových súborov (napr. či boli modifikované) pred začatím kompilácie je taktiež riadené z tohto miesta.

Snažili sme sa, aby informácie zobrazované používateľovi pri konfigurácii a pri spustení programu boli také isté. Vrátane usporiadania a spôsobu výpisu. Teda súbory *cmake/ConfigInfo.cmake* a *cmake/binary-info/binary_info.py.in* by mali byť podobné, čo sa týka obsahovej stránky informácií, ktoré svojím vykonaním poskytujú.

Pre potreby tejto úlohy sme adaptovali pomocou Python-u[3] vyhľadávanie pracovnej Git[37] vetvy v súbore *cmake/binary-info/ConfigGitRevision.cmake*, čím sme túto informáciu urobili dostupnou i na systémoch Windows. Ďalej sme tu zaviedli informácie o autorovi a dátume poslednej vykonanej zmeny (jedná sa o premenné *GIT_LAST_COMMIT_AUTHOR* a *GIT_LAST_COMMIT_DATE*). V tomto súbore sme taktiež namiesto pôvodného spôsobu overovania, či boli premenné naplnené nejakou hodnotou pomocou konštrukcie *if(NOT ${GIT_BRANCH} STREQUAL "")* použili lepší a spoľahlivejší spôsob *if(GIT_BRANCH)*. V prípade použitia prvého spôsobu sa stalo, že skript aj padol. Bolo to pri overovaní premennej *GIT_LAST_COMMIT,* ktorú sme si paradoxne zaviedli my v tom prípade, že v správe (*commit message*) bol znak pre nový riadok. Tým druhým spôsobom sme to vyriešili. K jeho objasneniu dodáme, že uvedený výraz bude vyhodnotený ako pravda vtedy, keď premenná *GIT_BRANCH* je definovaná a obsahuje hodnotu, ktorá je rozdielna od *0*, *OFF*, *FALSE*, alebo napr. aj iná ako prázdny reťazec a pod. (pre viac detailov viď CMake dokumentáciu[63]).

Do výpisu sme okrem autora a dátumu poslednej revízie pridali verziu CMake[2] a Python-u. Mená autorov môžu obsahovať aj iné ako ASCII znaky, čo je zabezpečené pomocou *# -*- coding: utf-8 -*-* v súbore *cmake/binary-info/binary_info.py.in* (viď [101]). Síce na príkazovom riadku vo Windows sa i napriek tomu také mená zobrazia zle, ale aspoň kvôli nim nebude s chybou ukončená kompilácia programu.

Na zisťovanie verzií kompilátorov sme použili skript, ktorý sme skôr na tento účel vytvorili v *cmake/compilers/GetCompilerVersion.py*. Pokladáme ho za viac univerzálny ako ten, ktorý sa používal pri výpise informácií o kompilátoroch predtým. Jeho rozboru sa venujeme v inej kapitole. Navyše výpis verzie kompilátora sme doplnili o vypísanie jeho výrobcu (vydavateľa, *vendor*).

Výpis informácií pri konfigurácii má značný význam pri zaznamenávaní údajov pre *cdash* web[85], pretože tak máme možnosť na webe vidieť všetky pre DIRAC[1] podstatné parametre. Inak by sme túto možnosť nemali. Význam zobrazených informácii sa zvyšuje pri využívaní rôznych CI služieb, ktorým sa venujeme v inej časti práce.

4.9 Problémy adaptácie Intel kompilátorov

Uskutočnili sme pokus o skompilovanie programu DIRAC[1] s využitím Intel[18] kompilátorov nachádzajúcich sa v produkte Intel Parallel Studio XE 2015 Update 3 Cluster Edition[102] na 64-bitovom Windows Server 2012 R2.

Príprava prostredia spočívala v nainštalovaní produktu Microsoft Visual Studio Professional 2013 with Update 3[103] a Intel Parallel Studio XE 2015 Update 3 Cluster Edition. Zmienené produkty sú k dispozícii aj v trial verzii, prípadne existujú verzie, ktoré môžu za zvýhodnených podmienok získať študenti, učitelia alebo vývojári *open source* softvéru (platí najmä pre Visual Studio) [52][104][105].

Po ich nainštalovaní sme pokračovali v konfigurácii prostredia pre prácu s kompilátormi pomocou príkazového riadku. Nasledovne:

a) Spustili sme v príkazovom riadku príkazy (viď dokumentáciu, ktorá sa zobrazuje na privítanie po nainštalovaní s prvými krokmi):

```
cd C:\Program Files (x86)\Intel\Parallel Studio XE 2015
psxevars.bat intel64
```

b) Pridali sme nasledujúce dve cesty do premennej prostredia *PATH*:

```
C:\Intel\Composer XE 2015\bin\intel64_mic
C:\Program Files (x86)\Intel\Composer XE\bin\intel64
```

c) Vytvorili sme si kópiu odkazu na príkazový riadok, otvorili jeho vlastnosti a do poľa *Target* sme zadali:

```
%comspec% /k ""C:\Program Files (x86)\Microsoft Visual Studio
12.0\VC\vcvarsall.bat" amd64 && "C:\Program Files (x86)\Intel\Composer XE
2015\bin\compilervars.bat" intel64 vs2013"
```

Po nastavení prostredia sme si úspešne odskúšali kompiláciu jednoduchých príkladov v jazykoch C a Fortran využívajúcich MPI[4]. Práve o podporu MPI zo strany Intel kompilátorov sme mali najväčší záujem. Také príklady je možné jednoducho nájsť na internete, prípadne v niektorých inštaláciách MPI implementácií (viď kapitolu o MPI).

Ešte uvedieme, že názov Intel C i C++ kompilátora[106] na Windows je *icl*, Fortran kompilátora[107] je *ifort*. Vhodný generátor je *"NMake Makefiles"*. Kompilácia a zostavenie programu sa následne spúšťa príkazom *nmake*. MPI kompilátory, v skutočnosti

dávkové súbory *.bat*, ktoré volajú kompilátory – konkrétne od Intel-u[18] alebo od Microsoft-u, majú pre C/C++/Fortran-90 názvy *mpicc.bat*, *mpicxx.bat*, *mpif90.bat*. MPI[4] programy sa spúšťajú napríklad takýmto príkazom: *mpiexec -np 4 program.exe*.

Pre názornosť ukážka spustenia konfigurácie a kompilácie DIRAC-u[1] (vo vopred pripravenom „vlastnom“ príkazovom riadku):

```
python setup --cc=icl --cxx=icl --fc=ifort --type=debug --int64 ^
--generator="NMake Makefiles" -D ENABLE_PCMSOLVER=OFF
cd build
nmake
```

Problémy s adaptáciou spočívajú najmä v tom, že v programe je celkom veľké množstvo kódu (podarilo sa nám ho objaviť najmä v *src/gp*) založeného na miešaní jazykov Fortran a C. Toto je ale uskutočnené spôsobom, keď sú mená funkcií v jazyku C, ktoré majú byť volané z Fortran-u systematicky definované s podčiarkovníkom na konci, napr. *void get_traceback_info_(...)* v súbore *src/gp/trace.c*. Toto je v poriadku na Linux-e, kde sa takáto konvencia[108] využíva aj pri Intel kompilátoroch, ale nie na iných systémoch. Na Linux-e to v princípe funguje tak, že vo Fortran-e sa zavolá:

```
call get_traceback_info (…)
```

ako v súbore *src/gp/memory_errorhandler.F90* a nájde sa správne funkcia:

```
get_traceback_info_(...)
```

ako je definovaná s podčiarkovníkom v súbore *src/gp/trace.c.* Pri Intel kompilátore na 64-bitovom Windows je to ale tak, že sa hľadá funkcia s názvom:

```
GET_TRACEBACK_INFO(...)
```

bez podčiarkovníka a veľkými písmenami, čo však splnené nie je. Preto už nie kompilátor, ktorý svoju úlohu splnil (zrejme on je ale zodpovedný za zmenu mien funkcií), ale zostavovací program - linker oznamuje chybu, že nedokáže nájsť požadované externé (*external*) symboly. V našom prípade sú to funkcie.

Problém volania funkcií z Fortran-u, ktoré sú napísané v jazyku C sme skúšali experimentálne riešiť týmito jednoduchými spôsobmi:

a) Meniť možnosti kompilácie pre Fortran zdrojové súbory, konkrétne použiť:

`/assume:underscore` alebo `/assume:nounderscore`

`/uppercase` alebo `/lowercase`

Výklad, čo tieto nastavenia znamenajú, je možné nájsť, napr. na [109]. Najpodstatnejšie ale je to, že kombinácia */assume:underscore /uppercase* slúži na to, čo chceme dosiahnuť. Teda to, aby sa ignorovalo, že C funkcie sú definované v zdrojových súboroch malými

písmenami (i keď sú vyhľadávané s veľkými), a aby sa predpokladalo, že v názvoch obsahujú podčiarkovník.

b) Pre funkcie, ktoré nie je možné nájsť sme skúsili vytvoriť pomocné funkcie, ktoré sú pomenované podľa Intel[18] - Windows konvencie[108] veľkými písmenami a bez podčiarkovníka, napr. takto pre funkciu *get_traceback_info_(...)*:

```
void GET_TRACEBACK_INFO(...)
{
      get_traceback_info_(...);
}
```

Tak, aby tieto volali funkcie, ktoré sa mali pôvodne zavolať, len ich zostavovací program nevedel nájsť.

c) Problémy nie sú ba pri volaní funkcií v jazyku C z Fortran-u, ale aj naopak, keď sa volajú funkcie v jazyku Fortran zo zdrojových kódov v jazyku C. Takto je to napr. v súbore *src/gp/gpc.c*, kde sa vo funkcii *void compoff_(...)* volá funkcia z Fortan-u *quit()*. Tá ale nemôže byť nájdená, pretože sa predpokladá existencia *QUIT()*. Ako rýchle riešenie sme použili jednoduchý trik (ktorý sa zdá už asi aj niekto pred nami skúšal):

```
#define quit QUIT
```

Napriek snahe sa nepodarilo DIRAC[1] úspešne skompilovať, pretože spomenuté spôsoby riešenia viedli ku konfliktným situáciám. Tie boli spôsobené tým, že keď sa pomocou nastavení kompilátora pre Fortran určilo, aby sa predpokladalo, že sú externé funkcie s podčiarkovníkom (my to chceme pre tie v C), tak sa to potom predpokladalo aj pre externé funkcie v jazyku Fortran. Avšak, toto následne opätovne spôsobilo problémy pri zostavovaní programu.

Tento problém sa dá síce riešiť konfiguráciou tak, že niektoré zdrojové súbory sa majú skompilovať bez týchto nastavení (v CMake[2] je to jednoduché). To sme ale potom zase tam, kde sme boli a máme problémy s externými funkciami v jazyku C. Pri našom pokuse o kompiláciu sme týmito spôsobmi riešenia došli k tomu, že z pôvodných 4 chýbajúcich externých symbolov sme sa síce posunuli ďalej v kompilácii (dostali sme sa k možnosti nájsť ďalšie problémy), ale skončili sme v situácii, keď ich chýbalo 41. To je veľké množstvo na riešenie pomocou triku s definovaním vlastných funkcií (podľa riešenia v bode b), čo nie je úplne dobrá praktika a zrejme by to prinieslo do kódu chaos.

Podľa našich zistení správny spôsob riešenia je, že sa urobí ISO_C_BINDING tak, ako je i odporúčané na stránkach Intel-u[110][111], alebo dokonca aj v dokumentácii k DIRAC-u[112]. Zrejme kódy sú staršieho dátumu a táto praktika pre korektnú spoluprácu

jazykov Fortran a C bola objavená neskôr. Treba dodať, že naše skúsenosti s programovaním v jazyku Fortran sú minimálne, a tak by toto bola príliš náročná úloha.

Ďalším problémom, s ktorým sme sa stretli pri pokuse o adaptovanie pre Intel[18] na Windows bolo, že Intel kompilátor nemá (pozn.: MinGW-w64[25] má) k dispozícii hlavičkový súbor *unistd.h* vkladaný v súbore *src/relccsd/ccunix.c*. Ten nebol potrebný, lebo na Linux-e sme skúšali kompiláciu aj bez neho a prebehla v poriadku. Bolo ale potrebné zadefinovať na Windows typ *off64_t* v tom istom súbore. Toto nie je nevyhnutné, keď sa používa MinGW-w64 kompilátor. Ten ho totiž už má definovaný.

Problémom je i to, že niektoré funkcie nie sú dostupné, napr. *snprintf(...)*. Riešenie by mohlo byť to, že sa použije funkcia *_snprintf(...)* podľa [113]. Iná alternatíva by bola napísať si vlastnú funkciu, ktorá by robila niečo podobné, napr. podľa [114].

Je potrebné spomenúť i to, že na Windows existujú Microsoft rozšírenia pre jazyky C a C++[115]. Tieto sa mierne vymykajú štandardom, a tak napr. znemožňujú použitie kľúčového slova *and* (alternatíva ku *&&*) bez vkladania dodatočného hlavičkového súboru Spomenuté kľúčové slovo sa tiež na jednom mieste v DIRAC-u[1] vyskytovalo. Toto je možné vyriešiť tak (bez ďalšieho hlavičkového súboru), že sa použije namiesto *and* kľúčové slovo *&&*. Existuje dokonca aj možnosť tieto rozšírenia vypnúť (možno by to bolo dobre, keďže DIRAC má byť multiplatformový), ale podľa dokumentácie potom existuje riziko, že pri istej konfigurácii by nastalo nepredvídateľné správanie [116].

Pre použitie matematických konštánt sa definuje *_USE_MATH_DEFINES*, viď [117]. Môžeme to urobiť dvomi spôsobmi, a to zadefinovať ju v každom zdrojovom súbore, kde sa používajú matematické konštanty (vkladá sa *math.h* alebo *cmath.h*). Toto je dobré urobiť pred vkladaných akýchkoľvek iných hlavičkových súborov. Prípadne, ako sme to urobili my pri svojom pokuse, pridať ju medzi definície v *CMakeLists.txt* príkazom *add_definitions(-D_USE_MATH_DEFINES)*.

Podarilo sa nám odhaliť (a vyriešiť) i jeden celkom zaujímavý rozdiel medzi MinGW[41][47][25] Fortran kompilátorom a Intel[107] Fortran kompilátorom na Windows. Rozdiel spočíva v tom, že MinGW Fortran kompilátor hľadá používateľské (nie systémové, ale tie uvedené v úvodzovkách) hlavičkové súbory na adresách relatívne k umiestneniu aktuálneho zdrojového súboru. Avšak, Intel Fortran ich takto nehľadá. Pravdepodobne skúša umiestnenia relatívne ku hlavnému zdrojovému adresáru. Pri adaptácii s týmto taktiež nastali problémy. Existuje jednoduché riešenie, a to pridať medzi možnosti Fortran kompilátora (*flags*) na systéme Windows túto možnosť: */assume:source_include* podľa zdroja [118].

Keby sa niekomu v budúcnosti podarilo prekonať spomínané problémy s miešaním kódu v C a vo Fortran-e, pravdepodobne by ešte čelil tomu, že by bolo potrebné prepísať možnosti pre kompilátory v module PCMSolver[39]. Taktiež by preň bolo potrebné skompilovať požadované knižnice Boost[96][97], a zrejme aj knižnicu zlib[98] použitím Intel[18] kompilátorov.

Je škoda, že sa momentálne nepodarilo adaptovať DIRAC[1] pre Intel kompilátory na Windows, lebo by sa tým otvorila cesta k vyššej optimalizácii kódu[107][106], využitiu MKL knižnice[119], a v neposlednom rade aj k natívnej implementácii MPI[4] vďaka možnosti použiť knižnicu Intel MPI[12].

Zmeny, ktoré boli neškodné pre inú funkcionalitu boli povyberané do hlavnej vetvy DIRAC-u. Ide najmä o prepísanie možností pre kompilátory v týchto súboroch:

cmake/compilers/CFlags.cmake

cmake/compilers/CXXFlags.cmake

cmake/compilers/FortranFlags.cmake.

Možnosti pre kompilátory bolo ešte potrebné prepísať aj pre podprojekt *xcfun*. Malá zmena bola potrebná aj v skripte *setup*, konkrétne ohľadom (budúceho) pridávania MKL nastavení: *mkl flag* (*command += ' -DMKL_FLAG="/Qmkl:%s"' % args.mkl*). Bolo prenesené aj definovanie matematických konštánt a použitie kľúčového slova *&&*.

Pri inštalácii Intel kompilátorov na Windows sme ocenili predošlé skúsenosti s nimi získané síce v Linuxovom prostredí, ale predsa na UMB HPCC[54].

4.10 Cygwin

Cygwin je distribúcia, ktorá umožňuje mnohým známym programom (nástroje GNU, BSD, X Server) z prostredia Unix-ových operačných systémov pracovať aj na operačnom systéme Windows. Je to zabezpečené Cygwin knižnicou (*cygwin1.dll*), ktorá poskytuje týmto programom možnosť POSIX systémových volaní a prostredie, ktoré potrebujú, aby mohli správne fungovať. [24][120]

Cygwin je možné v čase písania práce prevádzkovať na operačných systémoch Windows začínajúc verziou Windows XP SP3 a končiac Windows 8.1 alebo serverovou edíciou Windows Server 2012 R2. Podporované sú 32-bitové aj 64-bitové verzie týchto systémov. Takisto aj Cygwin je dostupný vo verzii 32-bitovej i 64-bitovej. [24][120]

Ak chceme v prostredí Cygwin používať program, ktorý je adaptovaný na systém Linux, musíme ho znova skompilovať zo zdrojových súborov na Cygwin-e. [24][120]

My sme použili Cygwin na skompilovanie DIRAC-u[1] na 32-bitovom Windows 7,

aj na 64-bitových systémoch Windows Server 2012 a Windows Server 2012 R2. Cygwin[24] nás začal zaujímať najmä preto, lebo preň je podporované Open MPI[11]. Naproti tomu na natívnom Windows Open MPI podporované už nie je [15].

Adaptovanie pre prostredie Cygwin bolo vykonané pridaním definície pre toto prostredie tak, aby sa pokladalo za Linux v súbore *cmake/ConfigArchitecture.cmake*. Bez tejto definície nebolo možné DIRAC[1] skompilovať. Na serveri „Chémia“ prebiehajú pravidelné testy na Cygwin-e. Je to zabezpečené pomocou nami vytvoreného testovacieho dávkového súboru (*.bat*), ktorý sa nachádza v *maintenance/cdash/cdash.cygwin.bat*. Avšak, pre chybu pravdepodobne mimo DIRAC-u je relatívne veľa testov neúspešných.

Súbor je síce je určený pre prácu v prostredí Cygwin, ale vlastne je vykonávaný v natívnom prostredí Windows. Cygwin príkazy voláme pomocou spúšťania programu *C:\cygwin64\bin\sh*(*.exe*), ktorý spustí Cygwin bash[76] a umožňuje nám vykonávať príkazy, ako keby sme boli v Cygwin-e. Parameter *--login* (potrebný je aj na to, aby bolo možné využívať všetky príkazy) nám slúži na prihlásenie sa a parameter *-c* na vykonanie príkazu uvedeného v úvodzovkách za ním.

Tento skript je prispôsobený pre použitie doc. M. Iliašom, ale jeho adaptácia pre iných používateľov je veľmi jednoduchá. Stačí hromadne nahradiť */home/milias* a príkaz na stiahnutie zdrojových kódov pre Git[37] (ak používame iný repozitár).

Pre úspešnosť vykonania sa predpokladá existencia umiestnenia v Cygwin prostredí */home/milias/test*. Tu sa bude vykonávať stiahnutie zdrojových súborov i kompilácia. Skontrolovať, ako prebehli jednotlivé kroky a pozrieť si informácie o priebehu testovania je možné v súbore */home/milias/cdash-testing.log*.

Pravidelné spustenie tohto dávkového súboru je možné naplánovať v Plánovači úloh systému Windows (*Task Scheduler*). Teraz si jeho obsah trochu priblížime.

Prvá vec, ktorá sa v skripte uskutočňuje je „vymazanie“ premennej prostredia z Windows *GIT_SSH*. Táto premenná je užitočná pre nastavenie PUTTY Pageant[121], ale spôsobuje konflikty s programom Git v Cygwin-e, čím znemožňuje pomocou neho úspešne stiahnuť zdrojové kódy DIRAC-u.

Ďalej vymazávame logovací súbor z predchádzajúceho spustenia testovania Parameter *-f* je použitý pre prípad, že by súbor ešte neexistoval. Nasleduje vymazanie starých zdrojových súborov vrátane skompilovaného programu. Robíme to preto, lebo je o niečo istejšie otestovať čisté zdrojové súbory tak, ako sú stiahnuté priamo z repozitára. Na riadku, ktorý nasleduje, si ich stiahneme nové. Hneď za tým stiahneme aj externé Git moduly. Potom nemusí byť týmto krokom zdržiavaná kompilácia programu.

Keď už máme pripravené zdrojové súbory, môžeme pokračovať s konfiguráciou DIRAC-u[1]. Nastavujeme, že sa vykoná 64-bitová kompilácia (s podporou 64-bitového typu *integer*). Bude sa využívať BLAS[42] z knižnice OpenBLAS[48] a program bude schopný pracovať paralelne s využitím Open MPI[11]. LAPACK[43] sa použije ten, ktorý je zabudovaný v DIRAC-u (*builtin*), lebo OpenBLAS na Cygwin-e[24] ho neobsahuje. *ZLIB_ROOT* nastavujeme preto, lebo konfigurácia je občas pomýlená a stáva sa jej, že chce použiť zlib[98] knižnicu prítomnú v natívnom Windows, čo ale nie je dobre. Pre zrýchlenie kompilácie sme predom skompilovali knižnice Boost[96]. To z toho dôvodu, lebo PCMSolver[39], ktorý ich požaduje, by si ich inak chcel skompilovať sám. Na Cygwin-e je táto jeho funkcionalita dostupná, čo je časovo náročné a zbytočné opakovať to pre každý test. Knižnice Boost, ktoré je možné nainštalovať do Cygwin-u konfigurácia PCMSolver-a nevie rozoznať (je to tak nastavené). Konfigurácii pomáhame s nájdením nami pripravených knižníc zadaním premenných *BOOST_INCLUDEDIR* a *BOOST_LIBRARYDIR*. Pri konfigurácii nastavujeme aj hodnotu pre premennú *CMAKE_LEGACY_CYGWIN_WIN32* na 0 (*false*). Znamená to, že sme uzrozumení s tým, že Cygwin nie je Windows, čo ale systém CMake[2] takto vyhodnocoval až do verzie 2.8.5 (viď CMake dokumentáciu[63]). Problémy by s týmto mohli nastať v tom prípade, keby bola niekde v CMake konfiguračných skriptoch použitá premenná *WIN32* s úmyslom, že Cygwin je Windows. Ale podľa našich zistení (vyhľadávania) nie je použitá ani takto, ani nijak inak či v DIRAC-u, alebo v jeho moduloch.

Po uskutočnení prvotnej konfigurácie je spustená experimentálna konfigurácia, kompilácia a testovanie so zaznamenávaním výsledkov a ich odoslaním na *cdash* web[85]. Toto je vykonané súhrnným príkazom *ctest -D Experimental*, ktorý vykoná všetky spomenuté kroky.

Je možné si všimnúť, že štandardný výstup a štandardný chybový výstup každého z príkazov je presmerovaný do logovacieho súboru pomocou príkazu *2>&1 | tee -a /home/milias/cdash-testing.log*. Príkaz *tee* je zaujímavý tým, že výstup bude nielen presmerovaný do súboru, ale stále i tak zobrazený na obrazovke. Aby bola v logovacom súbore zabezpečená prehľadnosť, je vždy pri vykonávaní ďalšej časti skriptu zobrazené na obrazovku (aj presmerované do súboru) hlásenie o tom, čo sa ide robiť. Využitý je príkaz *echo -e*. Parameter *-e* znamená, že sa zapne interpretácia *escape* sekvencií, napr. *\n*.

Je tu aj iný detail, a to ten, že používame *\"* napr. na to, aby sme mali pre príkaz *echo* hlášku, ktorú má zobraziť uvedenú v úvodzovkách, ale aby tieto úvodzovky neukončili reťazec určený ako príkaz pre *sh* program.

Pred spustením testovania si exportujeme premennú *DIRAC_MPI_COMMAND*, ktorá slúži na to, aby bol počas neho program vykonávaný paralelne. Má to svoju výhodu, že sa to robí takto, lebo keby bola premenná súčasťou prostredia neustále (napr. vo Windows premenných alebo v *.bashrc*), tak by boli verzie programu, ktoré neboli skompilované s podporou MPI[4], nefunkčné. Zároveň exportovanie premennej musí byť spojené s testovacím príkazom, aby mohlo byť v jednom príkaze pre zavolanie programu *sh* s parametrom *--login*. Keby bolo exportovanie premennej urobené v samostatnom (alebo v inom) príkaze, tak by premenná už v ďalších príkazoch (prihláseniach) nebola dostupná. Exportujeme aj premennú *OPENBLAS_NUM_THREADS*, pomocou ktorej sa určuje počet paralelných vlákien, na ktorých pracuje OpenBLAS[48].

Na konci súboru je momentálne nepoužívaná možnosť nezavrieť okno, v ktorom sa skript vykonával, ale počkať na stlačenie klávesy používateľom.

4.11 Ďalšia práca

Vykonali sme aj niektoré menšie úpravy, alebo sme skúšali niektorú funkcionalitu. Teraz tieto činnosti zhrnieme. Podrobnosti (Git log) k číslam zmien sú v prílohe.

Istý čas sme sa zaujímali o profilovanie [122]–[124]. Dopísali sme profilovacie možnosti aj pre C a C++ kompilátory (okrem Clang[45][46]), pretože predtým profilovanie bolo zapnuté iba pre jazyk Fortran. Odskúšali sme profilovanie na Windows s použitím MinGW-w64[25] kompilátorov. Úpravy boli potrebné v nasledujúcich konfiguračných CMake[2] súboroch pre nastavovanie kompilátorov *cmake/compilers/CFlags.cmake* a *cmake/compilers/CXXFlags.cmake*. Čísla zmien: 0cd2500, abf015a, 1de8ea1.

Podarilo sa nám vyriešiť problém s tým, že sa nedal priamo volať Git[37] na Windows z príkazového riadku, a teda ani využívať z DIRAC[1] konfigurácie. Bolo to spôsobené tým, že sa nesmel pridať adresár s Git binárnymi súbormi do *PATH*, pretože sa v tomto umiestnení nachádzal aj program *sh.exe*. Tento spôsobuje konflikty pri použití CMake (máme problém použiť generátor *"MinGW Makefiles"*). Vyriešili sme to tak, že sme pridali do *PATH* nie adresár *bin* ale *cmd*, odkiaľ sa tiež dá Git spustiť, ale v *cmd* už nie je program *sh.exe*. Upravili sme aj skript v *cmake/FindGit.cmake*, aby sa na Windows neuprednostňoval skript *git.cmd*, ale aby sa jednoducho hľadal *git.exe*, pretože raz sme s tým mali problémy. Čísla zmien: 85a30ea, bb7e95b.

Upravovali sme dokumentáciu, ktorá sa týkala využívania programu DIRAC na systéme Windows. Či už išlo o návody pre používateľov (ako si program skompilovať, využitie Cygwin[24]), alebo pre vývojárov (inštalácia Git, generovanie dokumentácie).

Často menené súbory dokumentácie sú:

doc/installation/windows.rst

doc/programmers/windows.rst.

Testovacie skripty, ktoré používal doc. M. Iliaš na Windows si občas vyžadovali zásah z našej strany, aby sa upravila (alebo opravila) ich funkčnosť. Napríklad súbor *maintenance/cdash/cdash.WinS12-Miro.bat*.

Pri pokuse o použitie kompilátorov LLVM Clang[45] sa nám podarilo zistiť, že *setup* nereaguje na našu požiadavku použiť nami zadané C a C+ kompilátory. Preto sme tento skript (*setup*) upravili. Urobili sme vyberanie si kompilátorov používateľom univerzálnym spôsobom, pretože pôvodné riešenie bolo funkčné iba na niektorých systémoch [55]. Táto úprava dokonca priniesla zjednodušenie spracovania, lebo sa následne mohli zjednotiť vetvenia pre Windows a Linux. Čísla zmien: 263147c, f9510c2.

Keď sme našli chybu v CMake[2] konfiguračných skriptoch alebo v prípade, že sme boli o nej informovaní, tak sme ju opravili. Príkladom je oprava chyby v nastavovaní možností pre C kompilátor (*flags*) alebo vyriešenie problému, že sa zobrazovalo dvakrát za sebou pridanie definícii pri konfigurácii:

```
USE_BUILTIN_BLAS;USE_BUILTIN_LAPACK;USE_BUILTIN_BLAS;USE_BUILTIN_LAPACK
```

Vyskytovalo sa to v tom prípade, že boli BLAS[42]/LAPACK[43] knižnice vyhľadávané automaticky, ale neboli nájdené. Problém sa nachádzal v súbore *CMakeLists.txt*. V tom istom súbore sme vyriešili aj zisťovanie verzie interpretera Python[3] pre CMake staršej verzie ako 2.8.6 (viď CMake dokumentáciu[63]). Čísla zmien: fdca1f7, 9841260, 45d2b37.

Pomohli sme i pri opravovaní testov. Jednalo sa o testy *stex* a *xyz_input*. V prvom bolo potrebné na Windows použiť namiesto príkazu *mv*, ktorý bol volaný z Fortran zdrojových kódov príkaz *move*. Oprava sa urobila v súbore *src/prp/pamstex.F*. V prípade druhého testu sa prišlo na to, že je potrebné znížiť optimalizáciu jedného zdrojového súboru, keď sa kompilovanie vykonáva na Windows pri použitý 64-bitového typu *integer* kompilátormi MinGW-w64[25] pri zvolenom profile *release*. Oprava bola uskutočnená pomocou zmeny CMake konfigurácie v *CMakeLists.txt*. Idea opravy bola dohodnutá po tom, ako sme uskutočnili s vedúcim práce viacero testov. Čísla zmien: fd712e3, 769b38f.

Pokúsili sme sa adaptovať použitie *debugger-u*[125] na Windows pomocou Windows PowerShell-u[95], ale jeho správanie sa líši od Linux-ovej verzie, preto sme ho nakoniec urobili neinteraktívnym spôsobom. Môžu sa navyše vyskytnúť problémy, keď sa použitý *debugger* nachádza v ceste, kde sa nachádzajú medzery. Vykonaná zmena je v súbore *pam*(*.in*). Čísla zmien: 61dff48.

Pri spustení DIRAC-u[1] sa zobrazuje čas spustenia a názov počítača *hostname*. Adaptovali sme túto funkcionalitu i pre systémy Windows. So zisťovaním *hostname* boli však problémy na niektorých iných systémoch. Vtedy, keď na nich nebola dostupná premenná prostredia *HOSTNAME*. My sme preto upravili spôsob zisťovania názvu počítača tak, že keď nie je k dispozícii premenná prostredia, použije sa ešte Fortran funkcia (zisťovanie je vykonávané vo Fortran-e). Zmeny boli vykonané v súboroch: *src/gp/gptrygve.F* a *src/gp/time.F*. Čísla zmien: 37ae876, 66a1da3.

5. Priebežná integrácia

Projekt DIRAC[1] začal s využívaním CI služieb. Veľkým prínosom je to, že niektoré zo služieb testujú paralelné vykonávanie programu s využitím knižníc, ktoré implementujú špecifikáciu MPI[4] (viď kapitola o MPI). Všeobecne sa očakáva to, že využívanie týchto služieb pomôže s odhaľovaním chýb v kóde.

Priebežná integrácia (*Continuous Integration*) je praktika používaná pri vývoji softvéru. Jej princípom je to, že členovia tímu integrujú svoju prácu často (každý z nich minimálne raz denne – povedzme, že pri DIRAC-u to nie je nevyhnutné). Každá integrácia je overovaná automatizovanou kompiláciou programu vrátane uskutočnenia testov. Týmto spôsobom sa vlastne tím vývojárov snaží nájsť chyby v integrácii čo najskôr. Táto metóda pomáha vyvíjať súdržný softvér rýchlejším spôsobom. [126]

Vyžaduje sa ale, aby projekt obsahoval dostatok testov, ktoré môžu chyby nájsť. Ak sa nájde nejaký problém, mal by byť čo najskôr odstránený. Dôvodom je, že nahromadené chyby sa odstraňujú ťažšie. Navyše celý proces automatickej kompilácie a testovania by nemal trvať príliš dlho (čo je v prípade DIRAC-u niekedy ťažké dosiahnuť). [126]

Služby CI, ktoré sa momentálne naplno používajú sú Shippable[22][127], CircleCI[16][128], Codeship[17][129], Magnum CI[130][131], Wercker[132][133] a Semaphore[23][134]. My sme sa pričinili najmä o konfigurácie pre Shippable, CircleCI a Codeship, ktorých skripty v ďalšom texte aj opíšeme. Pridáme aj opis skriptu, ktorý bol experimentálne určený pre službu AppVeyor[135][136], ktorá ako jediná podporuje testovanie v prostredí operačného systému Windows. Autormi ďalších skriptov sú študent J. Bubniak alebo doc. M. Iliaš.

Pre systém Windows síce momentálne nemáme (vyhovujúcu) CI službu, ktorá by vykonávala kompilovanie a otestovanie DIRAC-u po každej zmene kódu (*commit*), ale namiesto toho na serveri Katedry chémie prebiehajú každú noc intenzívne testy v natívnom

prostredí Windows a taktiež v prostredí Cygwin[24] (viď kapitolu o Cygwin). V natívnom prostredí sa kompiluje a testuje DIRAC[1] použitím MinGW-w64[25] kompilátorov. Pre zabezpečenie paralelizmu sa využíva optimalizovaná matematická knižnica OpenBLAS[48], ktorá je schopná paralelne pracovať na viacerých vláknach. Neúspechom končí iba malé množstvo testov. V prostredí Cygwin je možné skompilovať DIRAC kompilátormi GNU[81] nielen s knižnicou OpenBLAS, ale i s využitím paralelizmu založeného na využívaní štandardu MPI[4]. Konkrétne sa v Cygwin-e využíva knižnica Open MPI[11]. Sú však isté problémy, ktoré by mohli byť spôsobené chybou v kompilátoroch alebo v Open MPI knižnici.

Niektoré zo služieb vykonávajú aj nasadenie (*deploy*) dokumentácie na server UMB. Toto sa uskutočňuje na službách Magnum CI[130], Semaphore[23] a Codeship[17] zásluhou študenta J. Bubniaka.

5.1 YAML

Vo väčšine prípadov CI služby využívajú pre svoju konfiguráciu súbory s príponou *.yml*. Tieto súbory sú napísané v jazyku YAML. Nie však všetky služby sú na tomto založené, napr. Codeship[17] je výnimka a takisto aj Semaphore[23]. Ich skripty napísané v Bash[76] majú v názvoch uvedenú skratku *yml* z toho dôvodu, aby sa vedelo, že tiež patria medzi súbory používané pri testovaní CI službami.

YAML je jazyk určený primárne pre serializáciu údajov. Bol navrhnutý tak, aby bol pre ľudí ľahko čitateľný a spolupracoval s modernými programovacími jazykmi (najmä s jazykmi Perl, Python, PHP, Ruby a Javascript). Je v ňom veľmi dobre zrealizovaná podpora pre použitie a veľmi často sa využíva: v konfiguračných a logovacích súboroch, na zasielanie správ medzi procesmi, na zdieľanie údajov pri použití viacerých programovacích jazykov, na perzistenciu objektov a na ladenie (*debugging*) komplexných dátových štruktúr. [137][138]

5.2 Shippable

Konfiguračný skript pre CI službu Shippable[22] sa nachádza v súbore *shippable.yml*. Slúži na to, aby daná služba vedela, aké kroky má vykonať pre uskutočnenie kompilácie a testovania programu DIRAC[1] podľa našich požiadaviek.

Tento skript je zaujímavý tým, že testuje paralelný beh programu pri využití MPI[4] implementácie MPICH[13]. Táto je síce dostupná iba vo verzii, v ktorej poskytuje 32-bitovú veľkosť typu *integer* (*i4*) pre Fortran, ale aj napriek tomu testy prechádzajú v

poriadku. Skúšali sme skompilovať vlastnú verziu MPICH[13] so 64-bitovou veľkosťou typu *integer* (*i8*) pre Fortran, ale bolo to neúspešné. Nepodarilo sa nám to preto, lebo MPICH niečo také ani nepodporuje. O tom sme sa dozvedeli už počas konfigurácie, ktorá bola následne ukončená.

Pozrime sa teraz na to, čo skript v sebe obsahuje. Na začiatku sa nachádza obmedzenie vývojových vetiev, pre ktoré sa bude testovanie vykonávať. Tento skript sa bude spúšťať iba pre hlavnú vetvu *master*.

Nasleduje nastavenie premennej prostredia *DIRAC_MPI_COMMAND*. Premenná je určená na definovanie MPI[4] príkazu, ktorý bude využívaný na spúšťanie DIRAC-u[1] počas jeho práce alebo testovania. Nastavuje sa i počet použitých procesorov.

Ďalej sme určili, aby Shippable[22] pripravilo prostredie pre jazyk Python[3]. To z toho dôvodu, že Python sa v DIRAC-u často používa. Je pomocou neho konfigurovaný, a dokonca aj niektoré časti jeho kompilácie sú na ňom závislé.

Pomocou značiek *git: submodules: true* je vyjadrené, že projekt používa podprojekty (konkrétne sú dva: PCMSolver[39] a unit-tests), ktoré sú sťahované pomocou programu Git[37]. Takto sa zabezpečí, že ich Shippable stiahne a inicializuje ešte pred spustením konfigurácie a kompilácie. Ďalej je skript rozdelený pomocou značiek *install*, *before_script* a *script* na tri časti.

V časti *install* prebieha inštalácia alebo aktualizácia balíkov, ktoré potrebujeme na prácu s projektom. Inštalujeme kompilátory programovacích jazykov C, C++ a Fortran. Ďalej balík s programom *make*. Nasleduje *cmake* na vykonanie konfigurácie a generovania *Makefile* súborov. Pre PCMSolver je dôležité, aby sa nainštalovala aj knižnica zlib[98], ktorá je ním požadovaná. Snažíme sa odinštalovať prípadné súčasti inštalácie Open MPI[11], pretože túto implementáciu nechceme používať a namiesto nej nainštalujeme implementáciu MPICH.

V ďalšej časti *before_script* nechávame zobraziť informácie o počte a type použitých procesorov (jadier), voľnom diskovom mieste a dostupnej operačnej pamäti. Najmä informácia o procesoroch môže byť celkom zaujímavá. Všetky vypísané informácie si je možné prezrieť vo výstupe spracovania skriptu na stránke Shippable.

Nasleduje časť označená ako *script*, kde sa už vykonáva samotná konfigurácia, kompilácia a testovanie. Spúšťame konfiguračné skripty DIRAC-u *pamadm* a *setup*. Nastavujeme názov pre adresár *build*, ďalej meno, pod ktorým budú výsledky zobrazené na *cdash* webe[85] - *BUILDNAME*, vyberáme, že chceme použiť 64-bitový typ *integer* a MPI. Hodnotu *DART_TESTING_TIMEOUT* nastavujeme na vysoké číslo, a to z toho

dôvodu, že nemáme záujem, aby boli procesy testu zastavované (*kill*) programom CTest (súčasť systému CMake[2]) predčasne. To by bolo aj zbytočné, lebo vykonanie skriptu je na Shippable[22] obmedzené na čas jednej hodiny. Urobili sme všetko preto, aby celý skript prebehol v danom limite. Môže sa však stať, že sa to občas nepodarí, vtedy je ešte možnosť spustiť celý proces testovania znova.

Po vykonaní základnej konfigurácie sa presúvame do premenovaného adresára *build* (*build_shippableci_mpi*). Tu spustíme testovací príkaz *ctest* pre experimentálnu konfiguráciu (*ExperimentalConfigure*). Vtedy sa informácie o konfigurácii zaznamenajú do výsledkov pre *cdash* web[85]. Nasleduje spustenie testovacích príkazov pre vykonanie kompilácie a zaznamenanie jej priebehu (*ExperimentalBuild*) a samotné vykonanie testov so zaznamenaním výsledkov (*ExperimentalTest)*. Výsledky z konfigurácie, kompilácie a testovania sú odoslané na *cdash* web *ctest* príkazom *ExperimentalSubmit*.

Výsledky budú na webe zobrazené v skupine „Miro“. Výber skupiny je ovládaný pomocou parametra *--track Miro*. Medzi tu už spomenutými príkazmi sa nachádza aj príkaz *ldd dirac.x*. Ním si nechávame pre overenie vypísať, aké zdieľané dynamické knižnice bude práve skompilovaný program DIRAC[1] využívať.

Použili sme malé triky na dosiahnutie požadovanej funkčnosti. Prvým je, že za testovacím príkazom (*ExperimentalTest*) nasleduje ešte || *true*. Takto si zabezpečíme, aby bol vykonaný príkaz vždy pokladaný za úspešne skončený, aj keď to tak nie je. Robíme to z toho dôvodu, že sa môže stať a často sa stáva, že niektoré testy skončia neúspešne. V tom prípade sa ale pokladá celý testovací príkaz *ctest* za neúspešný (nemá návratovú hodnotu nula). Shippable pri výskyte takéhoto príkazu ukončuje spracovanie skriptu, takže by sa nám nepodarilo odoslať výsledky testov na web. Týmto sa dosiahne aj to, že celý priebeh skriptu bude pokladaný za úspešný, teda dostaneme od Shippable zelený rámček na DIRAC[1] *readme* stránke. Druhým trikom je, že kompiláciu a testovanie spúšťame s využívaním všetkých dostupných jadier procesoru, čo je zabezpečené tak, že si ich počet zisťujeme zo systémového súboru */proc/cpuinfo*.

Na úplnom konci skriptu sa ešte nachádza vypnutie emailových oznámení o stave spracovania skriptu (či prebehol úspešne, alebo zlyhal, prípadne nestihol limit) pomocou značky *notifications: email: false*.

Pre časové obmedzenie sa na Shippable nevykonáva vytváranie (kompilácia) dokumentácie. To preto, lebo najmä inštalácia všetkých potrebných knižníc za istých okolností trvá veľmi dlho.

5.3 CircleCI

Skript pre CI službu CircleCI[16] je výnimočný z dôvodu, že tu využívame pre zabezpečenie paralelnej práce programu knižnicu Open MPI[11], ktorá podporuje 64-bitový typ *integer* (*i8*). Ďalej tým, že máme k dispozícii knižnicu ATLAS[139] s optimalizovanými funkciami BLAS[42]. Knižnicu Open MPI si musíme na dosiahnutie nášho cieľa (podpory 64-bitového typu *integer*) skompilovať vlastnú. Knižnicu ATLAS využívame tú, ktorá je na systéme už prítomná.

V tomto skripte, tak ako aj v iných, obmedzujeme jeho vykonávanie iba na vetvu *master*. Na CircleCI však existuje možnosť predsa len spustiť skript i na inej vetve, keďže po poslaní zmien na ňu (*commit*, *push*) sa zobrazí status aj pre túto vetvu pre dané zmeny. Status je „Not Run“ a my máme možnosť manuálne spustiť otestovanie aj v tejto vetve.

V opačnom prípade sa nám môže stať, že na vetve *master*, ktorá je automaticky testovaná, nemáme záujem niektoré zo zmien testovať, pretože nie sú až také významné (iba úprava dokumentácie a pod.). Teraz máme možnosť spracovanie nevykonať (preskočiť). Urobiť to môžeme takým spôsobom, že do správy, ktorá opisuje zmeny (*commit message*) pridáme „ci skip“. Vtedy bude status zobrazený na CircleCI stránke pre tieto zmeny „Skipped“. Opäť ale máme možnosť testovanie manuálne spustiť. Preskakovanie vykonávania testov pre niektoré zmeny podporujú aj iné CI služby, je však potrebné overiť si príslušné kľúčové slovo v dokumentáciách, ktoré sú pre ne určené, viď [128][127], [129][131][133][134][136].

Ďalej nasleduje nastavenie prostredia, v ktorom bude testovanie prebiehať. Testovacie prostredie ovplyvňujeme tým, že definujeme premenné prostredia. Premenná *DIRAC_MPI_COMMAND* slúži na špecifikovanie spúšťača, ktorý bude použitý pri paralelnom behu DIRAC-u[1]. V tejto premennej je nastavený i počet použitých procesorov. Premenné *PATH* a *LD_LIBRARY_PATH* sú prispôsobené pre správnu funkciu Open MPI knižnice. Do *PATH* pridávame umiestnenie binárnych súborov (*mpirun*, MPI[4] kompilátory) nami skompilovanej verzie Open MPI a do *LD_LIBRARY_PATH* umiestnenie jej knižníc. Premenné *FC*, *CC*, *CXX* sú využívané pre konfiguráciu, ako sa má naša MPI knižnica skompilovať. Definujú, ktoré kompilátory sa majú na tento účel použiť (GNU[81][140]: *gfortran*, *gcc*, *g++*). Pre kompilátory je potrebné nastaviť správne možnosti na príkazovom riadku (*flags*) tak, aby sme dosiahli želané výsledné knižnice. Na to nastavujeme premenné *FCFLAGS*, *CFLAGS* a *CXXFLAGS*. Najdôležitejšie je nastavenie pre Fortran kompilátor, kde okrem toho, že sa majú vytvoriť 64-bitové objekty (*-m64*), sa nastavuje, že sa má používať 64-bitový typ *integer* (*-fdefault-integer-8*).

Premenné *N_PROC* a *TEST_N_PROC* nastavujú počet jadier CPU pre kompilovanie a testovanie. Tieto dve premenné je potrebné nastavovať opatrne, lebo pri vysokom čísle sa môže stať, že nevystačí operačná pamäť vo virtuálnom stroji na CircleCI[16].

V časti skriptu, ktorá je označená značkou *checkout* sa vykonáva inicializácia a stiahnutie externých Git[37] modulov (PCMSolver[39] a unit-tests).

Preberieme si ešte zvyšné časti *dependencies* a *test*. V časti *dependencies* je vykonané stiahnutie, rozbalenie a skompilovanie Open MPI[11] knižnice verzie 1.8.4. Po skompilovaní sú všetky jej súčasti nainštalované na umiestnenie *$HOME/open_mpi*, kde odkazujú aj predom exportované premenné prostredia *PATH* a *LD_LIBRARY_PATH*. Overujeme, či naozaj vyhovuje typ *integer*, a to tak, že si nechávame vypísať nastavenie, ktoré je obsiahnuté v Open MPI a dostupné pomocou príkazu *ompi_info -a | grep 'Fort integer size'*. Očakávame výsledok rovný osem. Pre istotu sa v tejto časti ešte na začiatku pokúšame odinštalovať všetky súčasti prípadných MPI[4] implementácií, ktoré by sa mohli na systéme nachádzať (či už ide o Open MPI, alebo MPICH[13]).

V časti označenej ako *test* sa už venujeme samotnej konfigurácii, kompilovaniu a testovaniu DIRAC-u[1]. Avšak, najskôr si necháme vypísať niektoré zaujímavé informácie (budú zobrazené vo výstupe na CircleCI). Zaujíma nás počet jadier, ktoré budú použité na kompilovanie a testovanie, ďalej či *libblas.so.3gf* má správne nastavené alternatívy a ukazuje na nami želanú knižnicu ATLAS[139] nachádzajúcu sa v umiestnení */usr/lib/atlas-base/atlas/libblas.so.3gf*. V neposlednom rade aj informácia o počte jadier procesora, ktoré sú celkovo dostupné na testovacom stroji (*lscpu*), je užitočná. Spustením príkazu *echo -e "--mb=256\n--aw=256" > $HOME/.diracrc* nastavujeme pomocou konfiguračného súboru *.diracrc*, aby DIRAC používal viac pamäte. Je to kvôli tomu, aby sme sa vyhli chybám spôsobených jej nedostatkom. Nasleduje spustenie konfiguračných skriptov *pamadm* a *setup*. Tu nastavujeme názov pre adresár *build*, kde prebehne kompilácia programu. Meno, ktoré bude použité pre označenie výsledkov testov v tabuľke na *cdash* webe[85] - *BUILDNAME*. Časový limit pre testy, ktorý interne používa CTest[2] (*DART_TESTING_TIMEOUT*) má nastavenú vysokú hodnotu, pretože na CircleCI je limitov, ktorým musí test vyhovieť dostatok. Navyše doc. M. Iliaš referoval, že ukončovanie pomocou CTest nie je dobré. Vtedy vraj nie je zrejmé, prečo bol test neúspešný či pre chybu, alebo pre zastavenie po časovom limite. Nastavujeme, že sa má používať 64-bitový typ *integer* (*--int64*) a že je potrebné použiť MPI kompilátory a knižnice, lebo beh bude paralelný (*--mpi*). Používame matematickú knižnicu BLAS[42], ktorá je dostupná v ATLAS-e (*--blas*) a v DIRAC-u zabudovaný LAPACK[43] (*--lapack*).

Kompiláciu vykonávame príkazom *make* namiesto využitia testovacieho príkazu *ctest*. Takto síce prídeme o zaznamenanie výstupu kompilovania na *cdash* web[85], ale umožní nám to rozdeliť ďalšie testovacie príkazy, ktoré slúžia na vykonanie testov, do viacerých menších, a pritom mať výsledky z nich na *cdash* webe zhrnuté do jedného záznamu. Toto by sa nám inak docieliť nepodarilo.

Príkaz *ldd dirac.x* umožňuje skontrolovať, ktoré knižnice DIRAC[1] používa (bude používať) počas svojho behu. Nastavenia, ktoré boli zapísané do konfiguračného súboru *.diracrc* si môžeme zobraziť pomocou *pam --show*. Pri kompilovaní sme použili počet jadier procesora definovaný v premennej prostredia *N_PROC* a pri testovaní sme súčasne spustili *TEST_N_PROC* testov súčasne.

Čo sa týka zvláštností skriptu, tak ide o príkaz || *true* umiestňovaný za každým testovacím príkazom, ktorý spúšťa testy (*ExperimentalTest*). Týmto sme dosiahli to, že status celého spracovania tohto skriptu na CircleCI[16] bude označený ako „Success“. Tu sa síce nezastavuje beh skriptu po príkaze, ktorý skončí s neúspechom (iná návratová hodnota ako nula), čím sa CircleCI líši od iných služieb, napr. Shippable[22]. Ale aj napriek tomu, že sa tu skript vykonal celý, bol jeho priebeh ako jediný označený na DIRAC[1] *readme* stránke ako neúspešný. Nie je ale pravda, že by inde nepadali testy. Preto sme použili tento trik, aby sa vedelo, že prebehol v poriadku celý skript. O testoch, ktoré neprešli úspešne, sa dajú informácie získať zo *cdash* web stránky, tak ako je to aj pri iných službách CI, ktoré používame.

Druhá jeho zvláštnosť je, že každý príkaz je vždy spustený v „novom termináli“ a je nastavený do adresára so zdrojovými kódmi. To znamená, že použitie príkazu *cd* na zmenu adresára nestačí. Je to iba jeden príkaz, ktorý sa vykoná a príkaz nasledujúci za ním bude opäť spustený z východiskového adresára. Tento problém sa na CircleCI rieši tak, že za príkazom sa uvedie dvojbodka a v zložených zátvorkách sa napíše *pwd: build_circleci_mpi* (keď sa chceme prepnúť do *build* adresára). Tento spôsob riešenia poradila podpora CircleCI doc. M. Iliašovi. Na prekonanie tohto však existuje aj iný spôsob, a to taký, že môžeme za príkazom *cd* pospájať vykonanie ďalších príkazov, napr. použitím *&&*. Takto je to urobené pri kompilácii Open MPI[11] knižnice.

Pokladáme za potrebné zmieniť sa o časových obmedzeniach, ktoré na CircleCI platia pre vykonávanie skriptov. Stretli sme sa s dvomi typmi obmedzení.

Prvý je, že každý príkaz musí zobraziť nejaký výstup v priebehu desiatich minút od posledného. Tento limit nám, samozrejme, pri dlho trvajúcich výpočtoch, ktoré DIRAC vykonáva, robil problémy. Hodnota tohto časového obmedzenia sa však dá zmeniť

pomocou kľúčového slova *timeout*. Napr. *timeout: 2000* v zložených zátvorkách za dvojbodkou nasledujúcou po príkaze. Presne tam, kde je nastavený i pracovný adresár.

Ďalšie časové obmedzenie je, že každý príkaz má celkovo 120 minút na svoje dokončenie. Nepodarilo sa nám nájsť spôsob, ako tento limit zmeniť alebo inak prekonať, okrem rozdelenia dlho trvajúcich príkazov na menšie časti. To je dôvod, prečo sa v skripte pre CircleCI[16] rozdelil príkaz vykonávajúci testovanie (*ExperimentalTest*) do štyroch príkazov. Každý z nich vykonáva iba istú časť testov.

Pomoc, ako skompilovať knižnicu Open MPI[11] s podporou 64-bitového typu *integer* sme získali zo skriptu, ktorého autorom je Dr. R. Bast[67] a je dostupný na [141].

Z dôvodu konfliktov pri inštalácii medzi knižnicou BLAS[42] poskytovanou ATLAS-om[139] a (inou) knižnicou BLAS požadovanou Python[3] knižnicou matplotlib[142] bolo vynechané vytváranie (generovanie) dokumentácie. Avšak, na druhej strane sme získali testovací server, kde máme k dispozícii ATLAS.

Nastavenie notifikácií o stave vykonania skriptov je možné si upraviť v nastaveniach na CircleCI. Je možné ich aj úplne vypnúť.

Namiesto konfiguračného skriptu (*circle.yml*) je možné využiť aj webové rozhranie poskytované na stránke služby, čo je jednoduchšie. Pre účely DIRAC-u[1] je však asi lepšie, keď existuje niečo, čo je možné pridať do zdrojových kódov, aby to všetci videli.

5.4 Codeship

Pre CI službu Codeship[17] máme pripravené dva testovacie skripty. Jeden je určený pre testovanie sériového vykonávania výpočtov, druhý slúži na otestovanie ich paralelného vykonávania pri využitý Open MPI[11]. Po zavedení paralelného testovacieho skriptu sa ten pre sériové testovanie prakticky prestal používať, ale predstavíme obidva.

Prvý z nich, ktorý je určený pre testovanie sériovej práce programu, sa nachádza v súbore *maintenance/cdash/codeship.yml.sh*. Sú v ňom zapísané príkazy, ktoré skompilujú a otestujú DIRAC[1], a to dvakrát. Raz bez použitia 64-bitového typu *integer* a raz s jeho využitím. Preskočme komentáre, ktoré sa nachádzajú na začiatku súboru (budeme sa im venovať neskôr) a poďme ďalej na obsah skriptu.

Pre účely kompilovania a testovania si exportujeme premennú prostredia *N_PROC*, ktorú sme naplnili hodnotou obsahujúcou počet dostupných jadier procesora prečítaním systémového súboru */proc/cpuinfo*. Následne si hodnotu tejto premennej necháme vypísať na výstup pre prípad, že by sme si ju chceli overiť. Ďalej pokračujeme so zobrazením niektorých zaujímavejších parametrov o systéme, na ktorom je skript spustený. Konkrétne

nás zaujímajú štatistiky o procesoroch (počet jadier by mal byť tu i v premennej prostredia *N_PROC*, ktorú sme si exportovali zhodný), informácie o voľnom diskovom mieste a o veľkosti voľnej operačnej pamäte.

Po zobrazení informácií o systéme, Python[3] inštalátorom pip[73] inštalujeme knižnice potrebné k tomu, aby sme boli schopní vygenerovať dokumentáciu. Je dôležité pripomenúť, že i keď môžeme inštalovať Python knižnice, napr. pomocou pip, tak na Codeship[17] nie je možné inštalovať nové balíky do systému (*apt-get install*). V prípade, že nám nejaký chýba, je možnosť napísať správu na podporu Codeship a oni ho nainštalujú. Tak, ako to už pre nás urobili s balíkom programu *doxygen*.

Na ďalších riadkoch sa nachádzajú príkazy, ktoré majú za cieľ zistiť umiestnenie a verziu použitého programu Git[37], generátora Sphinx[75] a doxygen[78].

Nasleduje 32-bitová kompilácia DIRAC-u[1]. Prebieha pomerne štandardným postupom (podobne ako na Shippable[22] alebo na CircleCI[16]). Postupne je vykonaná konfigurácia skriptami *pamadm* a *setup*, čo je nasledované presunutím sa do *build* adresára. V ňom sa vykonajú všetky kroky *Experimental* testovania (okrem kroku *ExperimentalUpdate*), a to v poradí konfigurácia (*ExperimentalConfigure*), kompilovanie (*ExperimentalBuild*), testovanie (*ExperimentalTest*) a odoslanie výsledkov na *cdash* web[85] (*ExperimentalSubmit*).

Posledný krok, odoslanie výsledkov, je vynechaný v prípade, že pracujeme na inej než *master* vetve. Toto robíme preto, lebo na Codeship neexistuje žiadny spôsob, ako obmedziť testovanie iba na niektoré konkrétne vetvy. Takto aspoň nezaplníme *cdash* web výsledkami ďalších sériových testov.

Konfigurácia, kompilovanie a testovanie 64-bitovej verzie programu prebieha takmer rovnakým spôsobom. Rozdiel je, že tu musíme nastaviť, že sa má používať 64-bitová verzia typu *integer*. Urobené je to pomocou parametra *--int64* pri spustení konfiguračného skriptu *setup*.

Na konci tohto skriptu nasleduje ešte vytváranie dokumentácie. Keďže v DIRAC-u nájdeme tri druhy dokumentácie, tak tu máme tri príkazy. Príkaz *make html* pre vytvorenie *html* dokumentácie pre používateľov, *make slides* pre vytvorenie *html* prezentácie a *make doxygen* pre vytvorenie dokumentácie pre vývojárov.

Vytváranie posledne zmienenej dokumentácie sa tu nepodarí. Preto je aj príslušný príkaz obalený v príkaze *echo*, aby bola „pohltená“ jeho návratová hodnota, ale stále bol zobrazený výstup. Pozn.: Toto robíme z rovnakých dôvodov aj s testovacím príkazom *ctest* pre krok *ExperimentalTest*.

Problémy s vytváraním doxygen[78] dokumentácie sme objavili aj v prostredí 32-bitového Cygwin[24], kde sa nám ju nepodarilo vygenerovať s najnovšou dostupnou verziou doxygen (1.8.9). Pri použití staršej verzie (1.8.8) prebehlo všetko v poriadku. Generovanie dokumentácie s doxygen-om nie je témou našej práce.

Pozrime sa teraz aj na druhý skript, ktorý sa pre zmenu venuje paralelnému testovaniu. Využívame na Codeship[17] už nainštalovanú knižnicu Open MPI[11]. Táto verzia síce nepodporuje 64-bitový typ *integer*, ale i napriek tomu testy predchádzajú dobre. Skript sa nachádza v súbore *maintenance/cdash/codeship_open_mpi.yml.sh*.

Prvým príkazom exportujeme premennú prostredia *DIRAC_MPI_COMMAND*, ktorá zabezpečuje paralelný beh programu počas testovania tým, že definuje spúšťací MPI[4] program aj s počtom procesorov, ktoré sa majú použiť. Nasleduje výpis zaujímavých informácií o použitom systéme. Takisto ako v sériovom skripte o procesoroch (počte jadier), voľnom diskovom mieste a o veľkosti voľnej operačnej pamäte. Navyše je tu i zobrazenie informácii o inštalácii Open MPI, ktorá sa na tomto systéme nachádza. Informácie o nej sú zobrazené spustením programu *ompi_info*.

Taktiež, podobne ako v skripte, ktorý bol opísaný vyššie, nasleduje inštalácia Python[3] knižníc, ktoré sú potrebné pre vytvorenie dokumentácie. Ďalej sa nachádza zisťovanie umiestnenia a verzie programu Git[37], verzie generátora Sphinx[75] a verzie programu na generovanie dokumentácie doxygen[78].

Teraz vykonávame kompilovanie a testovanie programu DIRAC[1] iba raz. Zásadný rozdiel je v tom, že pri spustení konfiguračného skriptu *setup* nastavujeme, že sa má používať 64-bitový typ *integer* (*--int64*) pre jazyk Fortran a zároveň, že sa má program skompilovať s podporou MPI (*--mpi*).

Tu sme pokladali za užitočné vypísanie, s ktorými knižnicami je program linkovaný pomocou príkazu *ldd dirac.x*. Význam použitia || *echo "CTest end."* za testovacím príkazom je ten, že konečná návratová hodnota takto zloženého príkazu bude vždy nula. Nechceme, aby sa spracovanie skriptu zastavilo len preto, že niektoré z testov boli neúspešné. Kompilovanie je spustené za použitia maximálneho počtu jadier, pre vykonanie testov používame ich štvornásobne menší počet.

Záverom prebieha vytváranie dokumentácie príkazmi *make html*, *make slides* a *make doxygen*. Už sme sa zmienili vyššie, že *make doxygen* je problematický.

Ako sme sľúbili, ešte sa zmienime o komentároch, ktoré sa nachádzajú na začiatku obidvoch súborov. Týkajú sa najmä toho, ako by sa mali oba skripty spúšťať (nastaviť na Codeship). Sú k dispozícii dve možnosti.

Prvá možnosť je, že si obsah jedného z týchto dvoch skriptov skopírujeme, prípadne upravíme podľa vlastných potrieb, a vložíme do testovacích nastavení svojho projektu na Codeship[17]. Tento scenár je podľa nášho názoru o niečo vhodnejší pre účely ladenia. Jednotlivé príkazy tak, ako sú zapísané v skriptoch (v nastaveniach projektu) sa potom zobrazujú oddelene vo výstupe testovania, čo je prehľadnejšie. Je tu ešte jeden rozdiel oproti spôsobu spustenia, ktorý bude ešte len uvedený. V tomto prípade, keď jeden z príkazov skončí neúspechom, bude celé testovanie (spustenie skriptu) označené za neúspešné. Dokonca sa daným príkazom aj zastaví. Ak sa chceme tomu vyhnúť, musíme používať niektorý z trikov na zabránenie tomuto stavu, napr. môžeme uviesť príkaz do príkazu *echo*, dať za ním || *echo "Command end."*, atď.

Druhá možnosť spustenia skriptov je taká, že sa do testovacích nastavení na Codeship zadá tento jeden príkaz (prípadne dva):

```
chmod a+x maintenance/cdash/<skript>.yml.sh &&
./maintenance/cdash/<skript>.yml.sh
```

Použitím tohto spôsobu bude ale všetok výstup z príkazov, ktoré sú obsiahnuté v skripte, zhrnutý pod jeden príkaz (alebo druhý z nich). Toto nie je až také prehľadné ako bola prvá možnosť. Povedzme ale, že to prináša výhodu toho, že aj akýkoľvek neúspešný príkaz nemá vplyv na status celého testovania. Dokonca sa spracovanie pri ňom nezastaví. Pritom nemusíme používať žiadny zo skôr uvedených trikov, aby sme zastaveniu zabránili.

5.5 AppVeyor

Dlho sme hľadali CI službu, ktorá poskytuje testovanie v prostredí operačného systému Windows. Všetky predchádzajúce CI služby (Shippable[22], CircleCI[16], Codeship[17] a iné) vykonávajú testovanie na systémoch Linux. Nakoniec sme našli iba jednu takú, ale pre obmedzenia, ktoré na nej sú, sa nám nepodarilo urobiť testovanie podľa našich predstáv. Avšak, aspoň sme získali s ňou nejaké skúsenosti.

Jedná sa o službu AppVeyor[135]. Táto služba je však zadarmo iba pre *open source* projekty, čo DIRAC[1] nie je. Pre iné projekty je možné využiť skúšobnú dobu. Rozhodli sme sa, že najprv sa pokúsime na AppVeyor otestovať PCMSolver[39], ktorý je v DIRAC-u využívaný ako jeden z externých modulov. Tento projekt je *open source*, takže sme sa nemuseli zaujímať o plynutie skúšobnej doby CI služby.

Konfiguračný skript, ktorý sme používali je *appveyor.yml*. Nachádza sa medzi zdrojovými kódmi DIRAC-u v adresári *maintenance/cdash*. Táto služba má prívetivý spôsob konfigurácie. Máme možnosť najprv konfiguračné príkazy písať v používateľskom

prostredí, ktoré nám AppVeyor[135] poskytuje, a potom si môžeme z nich vygenerovať spomínaný konfiguračný skript. Tento postup je výhodný, pretože nemusíme neustále prispievať zmenami do projektu (*commit*) kvôli nastavovaniu CI servera.

Preberme si v krátkosti, čo sa nachádza v konfiguračnom skripte, ktorým sme skúšali testovať projekt PCMSolver[39].

Na jeho začiatku sa nachádza značka *version: 1.0.{build}*, ktorá slúži na určenie formátu, v akom sa budú označovať jednotlivé spustenia skriptu (*build version format*), napr. 1.0.1, 1.0.2 atď. Pre vybratie prostredia (verzie systému) je možné za touto značkou uviesť značku *os*, napr. *os: Windows Server 2012 R2*.

Nasleduje časť označená ako *init*. Tu inštalujeme softvér, ktorý budeme neskôr potrebovať. Značka *ps* znamená, že nasledujúce príkazy bude namiesto „klasického“ príkazového riadku - *cmd.exe* vykonávať Windows PowerShell[95]. Zmienený Windows PowerShell má v porovnaní s príkazovým riadkom vylepšené možnosti i viacej príkazov. Je v ňom možné nájsť aj mnoho takých príkazov, ktoré sa podobajú na príkazy, ktoré sú známe zo systémov Linux. Avšak, nie sú úplne také isté. Vykonávajú však podobnú činnosť, preto dostali aj rovnaké alternatívne meno (*alias*), i keď majú aj svoje vlastné.

Prvé príkazy v tejto časti sú *mkdir* pre vytvorenie adresára pre softvér, ktorý potrebujeme na systém nainštalovať a *cd*, aby sme sa do neho nastavili. Necháme si zobraziť hlášku o tom, čo ideme ďalej robiť - inštalovať kompilátory. Podobné hlášky používame aj na oddelenie nasledujúcich činností, čo zvyšuje prehľadnosť skriptu.

Kompilátory nainštalujeme 64-bitové z MinGW-w64[25], pretože s nimi sme nemali problém PCMSolver skompilovať. Pre vysvetlenie viď kapitolu o adaptácii modulu PCMSolver. Predinštalované 32-bitové kompilátory z MinGW[41] sú nám momentálne zbytočné. Na stiahnutie z internetu používame príkaz *wget*, čo je alias pre Windows PowerShell príkaz *Invoke-WebRequest*. Stiahnutý *.7z* archív si rozbalíme pomocou programu 7Zip[143], ktorý sa už nachádza v pracovnom prostredí.

Pokračujeme inštaláciou C++ knižníc Boost[96]. Kompilovanie Boost knižníc, ktoré sa nachádzajú v projekte PCMSolver nebolo adaptované pre Windows. Opäť si ich stiahneme a rozbalíme. Následne je ich ešte potrebné skompilovať a nainštalovať na niektoré zo štandardných umiestnení, ktoré sú systémom CMake[2] prehľadávané. Inštalovať ich môžeme napr. i do prednastaveného *C:\Boost*. Postup vykonania kompilácie a inštalovania knižníc Boost na systéme Windows sme opísali v používateľskej

dokumentácii DIRAC-u[1] na jeho domovskej stránke. Tu môžeme iba zhrnúť, že v princípe nám na to stačia dva príkazy. Prvý rozbalí Boost[96] kompilačný systém pre zvolené kompilátory (*bootstrap.bat*) a druhý vykoná kompilovanie a inštaláciu (*b2*).

Zostáva posledná knižnica, ktorá je potrebná pre PCMSolver[39], a to je knižnica zlib[98]. Stiahneme si už jej skompilovanú 64-bitovú verziu, rozbalíme a presunieme do *C:\Program Files*. Na tomto umiestnení by ju systém CMake[2] nemal mať problémy počas konfigurácie nájsť.

Posledným príkazom v časti *init* sa nastavíme do adresára, kde sa nachádzajú (budú do neho stiahnuté) zdrojové súbory PCMSolver-a.

V časti *environment* nastavujeme premennú prostredia *path*. Vlastne využívame už tú pôvodnú, ktorá tam bola (v predchádzajúcich spusteniach sme si ju nechali vypísať). Avšak, odstránime z nej umiestnenie kompilátorov MinGW[41] a inštalácie MSYS[51], lebo táto spôsobuje konflikty s *"MinGW Makefiles"* generátorom. Následne pridáme umiestnenia nových 64-bitových MinGW-w64[25] kompilátorov a knižnice zlib:

```
C:\software\mingw64\bin;C:\Program Files\zlib
```

Pokračujeme časťou konfiguračného skriptu označenou ako *build_script*. Tu nie je použitá značka *ps*, takže sa tu nepoužíva Windows PowerShell[95], ale príkazový riadok. V tejto časti spustíme konfiguráciu projektu PCMSolver. Parameter *--test* zabezpečuje dostupnosť testov na overenie funkčnosti skompilovaného programu. Po konfigurácii sa presunieme do *build* adresára, kde spustíme kompilovanie programu.

V časti *test_script* sme mali v pláne vykonať otestovanie v predošlom kroku skompilovaného programu príkazom *mingw32-make test* (to isté ako samostatný príkaz *ctest*). Tu sa taktiež ako v časti *build_script* pracuje v štandardnom príkazovom riadku.

Hlavným problémom testovania na AppVeyor[135] je to, že vykonanie skriptu je obmedzené na čas 40 minút. Je síce pravda, že i niektoré iné služby majú podobné limity (napr. Magnum CI[130] iba 30 minút), ale tam je aspoň k dispozícii viac procesorových jadier. Tu je k dispozícii iba jedno (za viac sa platí), čím sa vylučuje možnosť veci trochu zrýchliť. Najďalej sme sa dostali pri kompilovaní projektu PCMSolver na 70%. Pre tento neúspech sme sa rozhodli AppVeyor nateraz zanechať.

Uvidí sa, čo prinesie do budúcnosti. Možno v prípade, ak bude DIRAC *open source* by bolo prínosné urobiť aspoň jeho skompilovanie bez PCMSolver-a (ten má veľa

závislostí). Vtedy by sa dali použiť už predinštalované 32-bitové kompilátory MinGW[41], čo by tiež spracovanie zrýchlilo. Máme ale pochybnosti, či by bolo možné uskutočniť aj nejaké testy. Navyše by nebolo možné využívať 64-bitový typ *integer*. Ak by sa o to niekto pokúšal, veríme, že naše skúsenosti z tohto skriptu by mu boli nápomocné.

Na AppVeyor[135] je predinštalované celkom slušné množstvo softvéru. Vrátane tých nástrojov, ktoré aj my používame, keď pracujeme s programom DIRAC[1]. Môžeme tu nájsť napr. Git[37], Python[3], MinGW (kompilátory, ktoré nevyhovujú pre PCMSolver[39]), a dokonca aj systém CMake[2].

Na tejto CI službe sme mali možnosť na testovanie využívať prostredie operačného systému Windows Server 2012 R2. Zoznam nainštalovaných programov a možných prostredí spolu s ich verziami je možné nájsť v dokumentácii služby na [136].

Ešte predvedieme, ako sa uskutoční výpis počtu procesorov (jadier) a premennej *path* v prostredí Windows PowerShell[95] (v príkazovom riadku je to trochu jednoduchšie, napr. *echo %path%*):

```
echo $env:NUMBER_OF_PROCESSORS
echo $env:PATH
```

ZÁVER

Výsledkom našej práce na programe DIRAC[1] je zachovanie a vylepšenie jeho použiteľnosti na 32-bitových i 64-bitových systémoch Windows. Dokonca sa podarilo odskúšať paralelný „vysokovýkonný“ beh programu s využitím optimalizovanej matematickej knižnice OpenBLAS[48]. Tu sme takisto odskúšali DIRAC na 32-bitovom aj na 64-bitovom systéme Windows s príslušnými verziami knižníc.

Preskúmali sme možnosti i pre využívanie paralelizmu založeného na MPI[4]. Tu je však situácia komplikovaná. Implementácie Open MPI[11] a MPICH[13] už istú dobu neponúkajú podporu (nové verzie) pre systém Windows. Hľadali sme teda iné alternatívy.

Nástupcom MPICH na Windows je knižnica Microsoft MPI[14]. Tu nastal ale problém, že momentálne máme možnosť DIRAC na Windows skompilovať iba s voľne dostupnými kompilátormi MinGW-w64[25], alebo prípadne s kompilátormi LLVM Clang[45] (tieto sú iba pre jazyky C a C++). S MinGW-w64 však nie je jednoduché skompilovať ani cvičné príklady s využitím knižnice Microsoft MPI. Najmä s podporou pre Fortran kompilátor to vypadá byť beznádejné.

Ako sme sľubovali v úvode, tak sme aspoň hľadali možnosti, ktoré by mohli niekedy v budúcnosti pomôcť problém vyriešiť. Našli sme projekt PToolsWin[26], ktorý sa snaží o spoluprácu medzi špeciálne upravenými verziami MinGW-w64 kompilátorov a knižnicou Microsoft MPI.

Kvôli problémom MinGW-w64 kompilátorov s podporovanými a modernými MPI knižnicami na Windows sme skúmali možnosti adaptácie DIRAC-u pre iné kompilátory. Mali sme v pláne použiť kompilátory Intel[18]. Ich využitím by sa otvorila cesta k paralelizácii pomocou knižnice Intel MPI[12]. Nezanedbateľným prínosom by bola i možnosť použitia matematickej knižnice Intel MKL[119].

V tomto prípade sme ale narazili na problém, že adaptácia pre iné kompilátory by si vyžadovala relatívne rozsiahle zásahy do zdrojového kódu programu. Najmä ide o časti založené na miešaní jazykov Fortran a C (požaduje sa ISO_C_BINDING[110]). Takáto práca je mimo náš dosah, pretože si vyžaduje programátora v jazyku Fortran.

Natívna podpora pre MPI na Windows je nateraz teda otázkou budúcnosti, ale už teraz je možné využívať (momentálne obmedzene, zrejme kvôli chybe v kompilátore alebo v MPI knižnici) MPI v prostredí Cygwin[24]. Podarilo sa nám využiť pravdepodobne už dávnejšie nepoužívanú možnosť práce na systéme Windows aj v tomto prostredí.

Priebežná integrácia sa momentálne vykonáva iba na službách, ktoré sú určené pre systémy Linux (Shippable[22], CircleCI[16], Codeship[17] a iné). Podarilo sa nám však

nájsť aj službu, ktorá poskytuje testovanie na systéme Windows - AppVeyor[135]. S jej použitím je to ale kvôli jej obmedzeniam nateraz komplikované. Napriek tomu je na Windows zabezpečené priebežné testovanie programu DIRAC[1] pomocou pravidelne vykonávaných testov či už v natívnom prostredí Windows, alebo v prostredí Cygwin[24].

Pri všetkých uskutočnených testoch sa zobrazujú rozšírené informácie o prostredí, v ktorom sa testovanie uskutočnilo, čo napomáha riešeniu zistených problémov.

ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV

[1] DIRAC, a relativistic ab initio electronic structure program, Release DIRAC14 (2014), written by T. Saue, L. Visscher, H. J. Aa. Jensen, and R. Bast, with contributions from V. Bakken, K. G. Dyall, S. Dubillard, U. Ekström, E. Eliav, T. Enevoldsen, E. Faßhauer, T. Fleig, O. Fossgaard, A. S. P. Gomes, T. Helgaker, J. Henriksson, M. Iliaš, Ch. R. Jacob, S. Knecht, S. Komorovský, O. Kullie, C. V. Larsen, J. K. Lærdahl, Y. S. Lee, H. S. Nataraj, P. Norman, G. Olejniczak, J. Olsen, Y. C. Park, J. K. Pedersen, M. Pernpointner, R. di Remigio, K. Ruud, P. Sałek, B. Schimmelpfennig, J. Sikkema, A. J. Thorvaldsen, J. Thyssen, J. van Stralen, S. Villaume, O. Visser, T. Winther, and S. Yamamoto (see http://www.diracprogram.org).

[2] KITWARE, INC. *CMake* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://www.cmake.org/>.

[3] PYTHON SOFTWARE FOUNDATION. *Welcome to Python.org* [online]. [cit. 2015-05-21]. Dostupné na internete: <https://www.python.org/>.

[4] UNIVERSITY OF TENNESSEE. *MPI: A Message-Passing Interface Standard - mpi22-report.pdf* [online]. [cit. 2015-05-12]. Dostupné na internete: <http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf>.

[5] *What Is High Performance Computing?* [online]. [cit. 2015-05-30]. Dostupné na internete: <http://insidehpc.com/hpc-basic-training/what-is-hpc/>.

[6] *Fortran Still Going Strong - insideHPC* [online]. [cit. 2015-05-30]. Dostupné na internete: <http://insidehpc.com/2015/05/fortran-still-going-strong/>.

[7] ILIAŠ, M. 2011. DIRAC - SOFTVÉR BUDÚCNOSTI TEORETICKÉHO CHEMIKA. In *Acta Universitatis Matthiae Belii, Ser. Chem., No. 13 (2011) 58 – 62*.

[8] BARNEY, B. *Introduction to Parallel Computing* [online]. [cit. 2015-05-19]. Dostupné na internete: <https://computing.llnl.gov/tutorials/parallel_comp/>.

[9] OPENMP ARB. *The OpenMP® API specification for parallel programming* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://openmp.org/wp/>.

[10] INDIANA UNIVERSITY. *FAQ: Java* [online]. [cit. 2015-05-12]. Dostupné na internete: <https://www.open-mpi.org/faq/?category=java>.

[11] INDIANA UNIVERSITY. *Open MPI: Open Source High Performance Computing* [online]. [cit. 2015-05-12]. Dostupné na internete: <http://www.open-mpi.org/>.

[12] INTEL CORPORATION. *Intel® MPI Library* [online]. [cit. 2015-05-28]. Dostupné na internete: <https://software.intel.com/en-us/intel-mpi-library>.

[13] *MPICH | High-Performance Portable MPI* [online]. [cit. 2015-05-28]. Dostupné na internete: <https://www.mpich.org/>.

[14] MICROSOFT CORPORATION. *Microsoft MPI* [online]. [cit. 2015-05-28]. Dostupné na internete: <https://msdn.microsoft.com/enus/library/bb524831%28v=vs.85%29.aspx>.

[15] INDIANA UNIVERSITY. *FAQ: What kinds of systems / networks / run-time environments does Open MPI support?* [online]. [cit. 2015-05-12]. Dostupné na internete: <https://www.open-mpi.org/faq/?category=supported-systems>.

[16] CIRCLE. *Continuous Integration and Deployment - CircleCI* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://circleci.com/>.

[17] CODESHIP. *Continuous Integration and Delivery – The Codeship Blog | via @codeship* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://blog.codeship.com/>.

[18] INTEL CORPORATION. *Intel® Compilers* [online]. [cit. 2015-05-23]. Dostupné na internete: <https://software.intel.com/en-us/intel-compilers>.

[19] *MPICH ABI Compatibility Initiative | MPICH* [online]. [cit. 2015-05-28]. Dostupné na internete: <https://www.mpich.org/abi/>.

[20] *MPICH Overview | MPICH* [online]. [cit. 2015-05-28]. Dostupné na internete: <https://www.mpich.org/about/overview/>.

[21] *Why can't I build MPICH on Windows anymore?* [online]. [cit. 2015-05-28]. Dostupné na internete: <https://wiki.mpich.org/mpich/index.php/Frequently_Asked_Questions#Q:_Why_can.27t_I_build_MPICH_on_Windows_anymore.3F>.

[22] SHIPPABLE, INC. *Continuous Integration with Docker | Shippable* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://www.shippable.com/>.

[23] RENDERED TEXT. *Semaphore - Continuous Integration & Deployment* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://semaphoreci.com/>.

[24] *Cygwin* [online]. [cit. 2015-06-01]. Dostupné na internete: <https://www.cygwin.com/>.

[25] *Mingw-w64 - GCC for Windows 64 & 32 bits [mingw-w64]* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://mingw-w64.yaxm.org/doku.php>.

[26] PARATOOLS, INC. *PToolsWIN - ParaTools, Inc.* [online]. [cit. 2015-05-28]. Dostupné na internete: <http://www.paratools.com/PToolsWIN>.

[27] PARATOOLS, INC. *HPCLinux - ParaTools, Inc.* [online]. [cit. 2015-05-28]. Dostupné na internete: <http://www.paratools.com/HPCLinux/>.

[28] MICROSOFT CORPORATION. *Microsoft Azure: Cloud Computing Platform & Services* [online]. [cit. 2015-05-28]. Dostupné na internete: <http://azure.microsoft.com/en-us/>.

[29] ORACLE CORPORATION. *Oracle VM VirtualBox* [online]. [cit. 2015-05-28]. Dostupné na internete: <https://www.virtualbox.org/>.

[30] MICROSOFT CORPORATION. *Microsoft HPC Pack SDK* [online]. [cit. 2015-05-28]. Dostupné na internete: <https://msdn.microsoft.com/en-us/library/cc853440%28v=vs.85%29.aspx>.

[31] ATLASSIAN. *Git and Mercurial code management for teams* [online]. [cit. 2015-05-30]. Dostupné na internete: <https://bitbucket.org/>.

[32] ATLASSIAN. *Bitbucket 101 - Bitbucket - Atlassian Documentation* [online]. [cit. 2015-05-30]. Dostupné na internete: <https://confluence.atlassian.com/display/BITBUCKET/Bitbucket+101>.

[33] GITHUB, INC. *GitHub* [online]. [cit. 2015-05-30]. Dostupné na internete: <https://github.com/>.

[34] GITHUB, INC. *GitHub Help - GitHub Enterprise Documentation* [online]. [cit. 2015-05-30]. Dostupné na internete: <https://help.github.com/>.

[35] GITLAB, B.V. *Create, review and deploy code together | Better than GitHub | GitLab* [online]. [cit. 2015-05-30]. Dostupné na internete: <https://about.gitlab.com/>.

[36] GITLAB, B.V. *Documentation | GitLab* [online]. [cit. 2015-05-30]. Dostupné na internete: <https://about.gitlab.com/documentation/>.

[37] TORVALDS L. et al. *Git* [online]. [cit. 2015-05-30]. Dostupné na internete: <http://git-scm.com/>.

[38] TORVALDS L. et al. *Git - Documentation* [online]. [cit. 2015-05-30]. Dostupné na internete: <http://git-scm.com/documentation>.

[39] DI REMIGIO, R. *An API for the Polarizable Continuum Model* [online]. [cit. 2015-05-26]. Dostupné na internete: <https://github.com/PCMSolver/pcmsolver>.

[40] BUTCHER, M. *TechnoSophos: Git Cherry Picking: Move small code patches across branches* [online]. [cit. 2015-05-30]. Dostupné na internete: <http://technosophos.com/2009/12/04/git-cherry-picking-move-small-code-patches-across-branches.html>.

[41] *MinGW | Minimalist GNU for Windows* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://www.mingw.org/>.

[42] *BLAS (Basic Linear Algebra Subprograms)* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://www.netlib.org/blas/>.

[43] *LAPACK — Linear Algebra PACKage* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://www.netlib.org/lapack/>.

[44] UNIV. OF TENNESSEE – OAK RIDGE NATIONAL LABORATORY. *The Netlib* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://www.netlib.org/>.

[45] UNIVERSITY OF ILLINOIS et al. *"clang" C Language Family Frontend for LLVM* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://clang.llvm.org/>.

[46] UNIVERSITY OF ILLINOIS et al. *Clang Compiler User's Manual — Clang 3.7 documentation* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://clang.llvm.org/docs/UsersManual.html>.

[47] *TDM-GCC : News* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://tdm-gcc.tdragon.net/>.

[48] XIANYI ZHANG et al. *OpenBLAS : An optimized BLAS library* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://www.openblas.net/>.

[49] Hans J. Aa. Jensen (University of Southern Denmark, Odense) <http://findresearcher.sdu.dk/portal/en/person/hjj>.

[50] SHVETS, G. *AMD Athlon II Dual-Core Mobile P320 - AMP320SGR22GM* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://www.cpu-world.com/CPUs/K10/AMD-Athlon%20II%20Dual-Core%20Mobile%20P320%20-%20AMP320SGR22GM.html>.

[51] *MSYS | MinGW* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://mingw.org/wiki/MSYS>.

[52] MICROSOFT CORPORATION. *Microsoft DreamSpark* [online]. [cit. 2015-06-01]. Dostupné na internete: <https://www.dreamspark.com/>.

[53] SHVETS, G. *Intel Xeon E7-2860 - AT80615005781AB* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%20E7-2860%20AT80615005781AB.html>.

[54] UNIVERZITA MATEJA BELA. *High Performance Computig Center - HPCC UMB* [online]. [cit. 2015-06-01]. Dostupné na internete: <http://www.hpcc.umb.sk/>.

[55] KITWARE, INC. *CMake - KitwarePublic* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://www.cmake.org/Wiki/CMake>.

[56] *Hot Questions - Stack Exchange* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://stackexchange.com/>.

[57] MICROSOFT CORPORATION. *Microsoft TechNet: Informácie pre IT odborníkov* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://technet.microsoft.com/sk-sk/>.

[58] MICROSOFT CORPORATION. *MSDN-the microsoft developer network* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://msdn.microsoft.com/en-US/>.

[59] MICROSOFT CORPORATION. *Technická podpora spoločnosti Microsoft* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://support.microsoft.com/sk-sk>.

[60] IEEE. *The Open Group Base Specifications Issue 7, 2013 Edition* [online]. Dostupné na internete: <http://pubs.opengroup.org/onlinepubs/9699919799/>.

[61] *Gmane -- Mail To News And Back Again* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://gmane.org/>.

[62] RUSSINOVICH, M. *Process Explorer* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://technet.microsoft.com/en-us/sysinternals/bb896653.aspx>.

[63] KITWARE, INC. *Documentation | CMake* [online]. [cit. 2015-05-23]. Dostupné na internete: <http://www.cmake.org/documentation/>.

[64] HRAŠKO, I. 2013. *Použitie systému CMake pre konfiguráciu a kompilovanie softvéru* : bakalárska práca. Banská Bystrica : FPV UMB, 2013. 51 s.

[65] PYTHON SOFTWARE FOUNDATION. *General Python FAQ — Python 2.7.10 documentation* [online]. [cit. 2015-05-21]. Dostupné na internete: <https://docs.python.org/2/faq/general.html>.

[66] BAST, R. *Adding tests in DIRAC — runtest 1.0.0 documentation* [online]. [cit. 2015-05-21]. Dostupné na internete: <http://runtest.readthedocs.org/en/latest/doc/adding_tests/dirac.html>.

[67] Radovan Bast, PDC and Department of Theoretical Chemistry and Biology, KTH Stockholm <http://bast.fr/>.

[68] PYTHON SOFTWARE FOUNDATION. *Python 2.4.4 Documentation - 18 October 2006* [online]. [cit. 2015-06-02]. Dostupné na internete: <https://docs.python.org/release/2.4.4/>.

[69] PYTHON SOFTWARE FOUNDATION. *Python Documentation contents — Python 2.7.10 documentation* [online]. [cit. 2015-06-02]. Dostupné na internete: <https://docs.python.org/2/contents.html>.

[70] *Python tutorial* [online]. [cit. 2015-06-02]. Dostupné na internete: <http://www.tutorialspoint.com/python/index.htm>.

[71] HELLMANN, D. *Python Module of the Week - Python Module of the Week* [online]. [cit. 2015-06-02]. Dostupné na internete: <http://pymotw.com/2/>.

[72] *Google's Python Class | Google for Education | Google Developers* [online]. [cit. 2015-06-02]. Dostupné na internete: <https://developers.google.com/edu/python/>.

[73] PYPA. *Installation — pip 7.0.3 documentation* [online]. [cit. 2015-06-02]. Dostupné na internete: <https://pip.pypa.io/en/latest/index.html>.

[74] PYTHON SOFTWARE FOUNDATION. *PyPI - the Python Package Index : Python Package Index* [online]. [cit. 2015-06-02]. Dostupné na internete: <https://pypi.python.org/pypi>.

[75] BRANDL, G. et al. *Overview — Sphinx 1.3.1+ documentation* [online]. [cit. 2015-05-20]. Dostupné na internete: <http://sphinx-doc.org/>.

[76] FREE SOFTWARE FOUNDATION, INC. GNU Bash [online]. [cit. 2015-05-20]. Dostupné na internete: <https://www.gnu.org/software/bash/>.

[77] FREE SOFTWARE FOUNDATION, INC. *sed, a stream editor* [online]. [cit. 2015-05-20]. Dostupné na internete: <http://www.gnu.org/software/sed/manual/sed.html>.

[78] VAN HEESCH, D. *Doxygen: Main Page* [online]. [cit. 2015-05-20]. Dostupné na internete: <http://www.stack.nl/~dimitri/doxygen/index.html>.

[79] SCHENK, CH. *Home - MiKTeX Project Page* [online]. [cit. 2015-05-20]. Dostupné na internete: <http://miktex.org/>.

[80] *TeX Users Group (TUG) home page* [online]. [cit. 2015-05-20]. Dostupné na internete: <http://tug.org/>.

[81] FREE SOFTWARE FOUNDATION, INC. *GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF)* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://gcc.gnu.org/>.

[82] NVIDIA CORPORATION. *PGI Compilers & Tools* [online]. [cit. 2015-05-23]. Dostupné na internete: <http://www.pgroup.com/index.htm>.

[83] IBM CORPORATION. *IBM Overview of the IBM XL C/C++ and XL Fortran compilers - United States* [online]. 12. december 2014 [cit. 2015-05-23]. Dostupné na internete: <http://www-01.ibm.com/support/docview.wss?uid=swg27005175>.

[84] KITWARE, INC. *CMake - Cross Platform Make* [online]. [cit. 2015-05-23]. Dostupné na internete: <http://www.cmake.org/cmake/help/v2.8.7/cmake.html>.

[85] *CDash - DIRAC* [online]. [cit. 2015-05-23]. Dostupné na internete: <https://testboard.org/cdash/index.php?project=DIRAC>.

[86] REITZ, K. et al. *Requests: HTTP for Humans — Requests 2.7.0 documentation* [online]. [cit. 2015-05-23]. Dostupné na internete: <http://docs.python-requests.org/en/latest/>.

[87] INTEL CORPORATION. *Intel® C++ Compiler XE 13.1 User and Reference Guide* [online]. [cit. 2015-05-24]. Dostupné na internete: <https://software.intel.com/sites/products/documentation/doclib/iss/2013/compiler/cpp-lin/>.

[88] INTEL CORPORATION. *Intel® Compiler Options for Intel® SSE and Intel® AVX generation (SSE2, SSE3, SSSE3, ATOM_SSSE3, SSE4.1, SSE4.2, ATOM_SSE4.2, AVX, AVX2, AVX-512) and processor-specific optimizations | Intel® Developer Zone* [online]. [cit. 2015-05-24]. Dostupné na internete: <https://software.intel.com/en-us/articles/performance-tools-for-software-developers-intel-compiler-options-for-sse-generation-and-processor-specific-optimizations>.

[89] INTEL CORPORATION. *Runtime fatal error produced on application built with -xHost option* [online]. [cit. 2015-05-24]. Dostupné na internete: <https://software.intel.com/en-us/articles/runtime-fatal-error-produced-on-application-built-with-xhost-option>.

[90] PAGE, C. G. *Professional Programmer's Guide to Fortran77* [online]. [cit. 2015-05-26]. Dostupné na internete: <http://www.star.le.ac.uk/~cgp/prof77.html>.

[91] HILE, G. N. *GNU Fortran Notes* [online]. [cit. 2015-05-26]. Dostupné na internete: <http://www.math.hawaii.edu/~hile/fortran/fortmain.htm>.

[92] SHENE, C. K. *Fortran 90 Tutorial* [online]. [cit. 2015-05-26]. Dostupné na internete: <http://www.cs.mtu.edu/~shene/COURSES/cs201/NOTES/fortran.html>.

[93] STANFORD UNIVERSITY. *Fortran 77 Tutorial* [online]. [cit. 2015-05-26]. Dostupné na internete: <http://web.stanford.edu/class/me200c/tutorial_77/>.

[94] NICHOLSON, J. A. *FORTRAN Tutorial - Free Guide to Programming Fortran 90/95* [online]. [cit. 2015-05-26]. Dostupné na internete: <http://www.fortrantutorial.com/>.

[95] MICROSOFT CORPORATION. *Getting Started with Windows PowerShell* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://technet.microsoft.com/en-us/library/hh857337.aspx>.

[96] *Boost C++ Libraries* [online]. [cit. 2015-05-26]. Dostupné na internete: <http://www.boost.org/>.

[97] JAMES, A. *How to install the C++ Boost Libraries on Windows - Andres Jaimes* [online]. [cit. 2015-05-26]. Dostupné na internete: <http://andres.jaimes.net/718/how-to-install-the-c-boost-libraries-on-windows/>.

[98] GAILLY, J. L. - ADLER, M. *zlib Home Site* [online]. [cit. 2015-05-26]. Dostupné na internete: <http://www.zlib.net/>.

[99] *MinGW-w64 - for 32 and 64 bit Windows - Browse /External binary packages (Win64 hosted)/Binaries (64-bit) at SourceForge.net* [online]. [cit. 2015-05-26]. Dostupné na internete: <http://sourceforge.net/projects/mingw-w64/files/External binary packages %28Win64 hosted%29/Binaries %2864-bit%29/>.

[100] Roberto Di Remigio (Universitetet i Tromsø, Tromsø) <http://www.researchgate.net/profile/Roberto_Di_Remigio>.

[101] PYTHON SOFTWARE FOUNDATION. *PEP 0263 -- Defining Python Source Code Encodings | Python.org* [online]. [cit. 2015-05-27]. Dostupné na internete: <https://www.python.org/dev/peps/pep-0263/>.

[102] INTEL CORPORATION. *Intel® Parallel Studio XE 2015 | Intel® Developer Zone* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://software.intel.com/en-us/intel-parallel-studio-xe>.

[103] MICROSOFT CORPORATION. *Visual Studio – Microsoft Developer Tools* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://www.visualstudio.com/>.

[104] MICROSOFT CORPORATION. *Bezplatné nástroje pro vývojáře – Visual Studio Community 2013* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://www.visualstudio.com/products/visual-studio-community-vs>.

[105] INTEL CORPORATION. *Qualify for Free Software | Intel® Developer Zone* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://software.intel.com/en-us/qualify-for-free-software>.

[106] INTEL CORPORATION. *Intel® C and C++ Compilers | Intel® Developer Zone* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://software.intel.com/en-us/c-compilers>.

[107] INTEL CORPORATION. *Intel® Fortran Compilers* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://software.intel.com/en-us/fortran-compilers>.

[108] INTEL CORPORATION. *Naming Conventions | Intel® Developer Zone* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://software.intel.com/en-us/node/510864>.

[109] *ifort - invokes the Intel(R) Fortran Compiler* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://computing.llnl.gov/tutorials/linux_clusters/man/ifort.txt>.

[110] INTEL CORPORATION. *Standard Fortran and C Interoperability | Intel® Developer Zone* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://software.intel.com/en-us/node/510843>.

[111] INTEL CORPORATION. *User and Reference Guide for the Intel® Fortran Compiler 14.0 | Intel® Developer Zone* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://software.intel.com/en-us/compiler_14.0_ug_f>.

[112] DIRAC AUTHORS. *Programming rules — DIRAC 14.0 documentation* [online]. [cit. 2015-05-20]. Dostupné na internete: <http://diracprogram.org/doc/master/programmers/rules.html>.

[113] MICROSOFT CORPORATION. *_snprintf, _snprintf_l, _snwprintf, _snwprintf_l* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://msdn.microsoft.com/en-us/library/2ts7cx93.aspx>.

[114] *snprintf - C++ Reference* [online]. [cit. 2015-05-20]. Dostupné na internete: <http://www.cplusplus.com/reference/cstdio/snprintf/?kw=snprintf>.

[115] MICROSOFT CORPORATION. *Microsoft Extensions to C and C++* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://msdn.microsoft.com/en-us/library/34h23df8.aspx>.

[116] MICROSOFT CORPORATION. */Za, /Ze (Disable Language Extensions)* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://msdn.microsoft.com/en-us/library/0k0w269d.aspx>.

[117] MICROSOFT CORPORATION. *Math Constants* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://msdn.microsoft.com/en-us/library/4hwaceh6.aspx>.

[118] *preprocessor include path differences* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://software.intel.com/en-us/forums/topic/413785>.

[119] INTEL CORPORATION. *Intel® Math Kernel Library (Intel® MKL) | Intel® Developer Zone* [online]. [cit. 2015-05-20]. Dostupné na internete: <https://software.intel.com/en-us/intel-mkl>.

[120] *Cygwin FAQ* [online]. [cit. 2015-05-19]. Dostupné na internete: <https://cygwin.com/faq.html>.

[121] *Download PuTTY - a free SSH and telnet client for Windows* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://www.putty.org/>.

[122] HONEYFORD, M. *Speed your code with the GNU profiler* [online]. [cit. 2015-06-02]. Dostupné na internete: <http://www.ibm.com/developerworks/library/l-gnuprof.html>.

[123] PAPADOPOULOS, X. *Debugging and profiling your C/C++ programs using Free Software | Free Software on WordPress.com* [online]. [cit. 2015-06-02]. Dostupné na internete: <https://xpapad.wordpress.com/2009/05/18/debugging-and-profiling-your-cc-programs-using-free-software/>.

[124] SUBODH V. PACHGHARE. *Code Profiling in Linux Using Gprof - Open Source For You* [online]. [cit. 2015-06-02]. Dostupné na internete: <http://www.opensourceforu.com/2011/06/code-profiling-in-linux-using-gprof/>.

[125] DONAHOO, M. J. *How to Debug Using GDB* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://cs.baylor.edu/~donahoo/tools/gdb/tutorial.html>.

[126] FOWLER, M. *Continuous Integration* [online]. [cit. 2015-05-28]. Dostupné na internete: <http://www.martinfowler.com/articles/continuousIntegration.html>.

[127] SHIPPABLE, INC. *Learn About What Makes Shippable Great - Shippable Documentation* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://docs.shippable.com/>.

[128] CIRCLE. *What can we help you with? - CircleCI* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://circleci.com/docs>.

[129] CODESHIP. *Codeship Documentation* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://codeship.com/documentation>.

[130] MAGNUM CI. *Magnum CI - Hosted Continuous Integration and Continuous Deployment Platform* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://magnum-ci.com/>.

[131] MAGNUM CI. *Magnum CI - Hosted Continuous Integration Platform for Private Repositories* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://magnum-ci.com/docs>.

[132] WERCKER. *wercker* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://app.wercker.com/sessions/new>.

[133] WERCKER. *wercker - docs* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://devcenter.wercker.com/docs/>.

[134] RENDERED TEXT. *Semaphore Documentation* [online]. [cit. 2015-06-03]. Dostupné na internete: <https://semaphoreci.com/docs/>.

[135] APPVEYOR SYSTEMS INC. *Continuous Integration and Deployment service for Windows developers - Appveyor* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://www.appveyor.com/>.

[136] APPVEYOR SYSTEMS INC. *Getting started - Appveyor* [online]. [cit. 2015-06-03]. Dostupné na internete: <http://www.appveyor.com/docs>.

[137] EVANS, C. C. et al. *The Official YAML Web Site* [online]. [cit. 2015-05-28]. Dostupné na internete: <http://yaml.org/>.

[138] EVANS, C. C. et al. *YAML Ain't Markup Language (YAML™) Version 1.2* [online]. [cit. 2015-05-28]. Dostupné na internete: <http://www.yaml.org/spec/1.2/spec.html>.

[139] *Automatically Tuned Linear Algebra Software (ATLAS)* [online]. [cit. 2015-05-25]. Dostupné na internete: <http://math-atlas.sourceforge.net/>.

[140] FREE SOFTWARE FOUNDATION, INC. *GCC online documentation - GNU Project - Free Software Foundation (FSF)* [online]. [cit. 2015-06-04]. Dostupné na internete: <https://gcc.gnu.org/onlinedocs/>.

[141] BAST, R. *Installation walkthru for DIRAC14 with GNU and OpenMPI (64bit-integers) on Ubuntu 14.04 x86_64.* [online]. [cit. 2015-06-05]. Dostupné na internete: <https://gist.github.com/bast/81760961c5a8199e561a>.

[142] HUNTER, J. *matplotlib: python plotting — Matplotlib 1.4.3 documentation* [online]. [cit. 2015-06-04]. Dostupné na internete: <http://matplotlib.org/>.

[143] PAVLOV, I. *7-Zip* [online]. [cit. 2015-06-04]. Dostupné na internete: <http://www.7-zip.org/>.

PRÍLOHY

Vedúci práce a oponent majú prístup k zdrojovému kódu programu DIRAC. Tento program nemá otvorený zdrojový kód. V prípade záujmu je potrebné získať bezplatnú licenciu: http://diracprogram.org/doku.php

Program PCMSolver má otvorený zdrojový kód a je dostupný na GitHub: https://github.com/PCMSolver/pcmsolver

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

DIRAC V PROSTREDÍ MICROSOFT WINDOWS: ADAPTÁCIA A VYSOKOVÝKONNÉ POČÍTANIE

Používateľská príručka

Banská Bystrica 2015 **Bc. Ivan Hraško**

1. Natívna verzia pre Windows s OpenBLAS

a) **konfigurácia**

```
C:\Dirac\my-dirac> python setup --generator="MinGW Makefiles" ^
       --int64 ^
       --blas="C:/OpenBLAS/libopenblas.dll" ^
       --lapack="C:/OpenBLAS/libopenblas.dll" ^
       -D ZLIB_ROOT="C:\Program Files\zlib" ^
       -D BOOST_INCLUDEDIR="C:\Boost\include" ^
       -D BOOST_LIBRARYDIR="C:\Boost\lib" ^
       build
```

b) **kompilovanie**

```
C:\Dirac\my-dirac> cd build
C:\Dirac\my-dirac\build> mingw32-make
```

c) **rýchly test**

```
C:\Dirac\my-dirac\build> ctest -L short
```

d) **prvý výpočet**

```
C:\Dirac\my-dirac\build> python pam --mol=methanol.xyz --inp=hf.inp
```

e) **výsledok** je v súbore *hf_methanol.out*, pracovné súbory sú v archíve *hf_methanol.tgz*

2. Verzia v prostredí Cygwin s OpenBLAS

a) **konfigurácia**

```
/home/user/Dirac/my-dirac$ python setup -D CMAKE_LEGACY_CYGWIN_WIN32=0 \
       --int64 \
       --blas=/usr/bin/cygblas-0.dll \
       --lapack=builtin \
       -D ZLIB_ROOT=/usr/lib \
       build
```

b) **kompilovanie**

```
/home/user/Dirac/my-dirac$ cd build
/home/user/Dirac/my-dirac/build$ make
```

c) **rýchly test**

```
/home/user/Dirac/my-dirac/build$ ctest -L short
```

d) **prvý výpočet**

```
/home/user/Dirac/my-dirac/build$ python pam --mol=methanol.xyz --inp=hf.inp
```

e) **výsledok** je v súbore *hf_methanol.out*, pracovné súbory sú v archíve *hf_methanol.tgz*

Obsah súboru *hf.inp:*

```
**DIRAC
.WAVE FUNCTION
**WAVE FUNCTION
.SCF
**MOLECULE
*BASIS
.DEFAULT
 cc-pVDZ
*END OF INPUT
```

Obsah súboru *methanol.xyz:*

```
6
my first DIRAC calculation # anything can be in this line
C       0.000000000000       0.138569980000       0.355570700000
O       0.000000000000       0.187935770000      -1.074466460000
H       0.882876920000      -0.383123830000       0.697839450000
H      -0.882876940000      -0.383123830000       0.697839450000
H       0.000000000000       1.145042790000       0.750208830000
H       0.000000000000      -0.705300580000      -1.426986340000
```

Zobrazenie všetkých možností konfigurácie: *python setup --help*

Na 32-bitových systémoch nie je možné použiť 64-bitový typ *integer*, preto je potrebné vynechať *--int64* zo *setup* príkazu.

Najmä na 64-bitových natívnych systémoch si treba dať pozor na použitie správnej verzie OpenBLAS, lebo pre tieto systémy sú k dispozícii dve verzie. Jedna podporuje 32-bitový typ *integer* a druhá 64-bitový (potrebné, ak je zvolené *--int64*).

Profilovanie je možné uskutočniť tak, že sa pri konfigurácii zvolí *setup --type=profile*, program sa skompiluje a otestuje. Výpočet sa potom uskutoční takto:

```
python pam --mol=methanol.xyz --inp=hf.inp --keep_scratch
```

Následne sa vyhľadá umiestnenie, kde prebiehal výpočet (*scratch*), napr.:

```
C:\Users\<používateľ>\DIRAC_scratch_directory\<používateľ>\DIRAC_hf_methanol_4044
```

Tu sa spustí v príkazovom riadku (odporúčame výstup presmerovať do súboru):

```
gprof dirac.x.exe gmon.out > subor.txt
```

Príklad prevzatý z http://diracprogram.org/doc/master/tutorials/getting_started.html.

Pre viac informácií viď aktuálnu dokumentáciu na: http://diracprogram.org/doku.php (odporúčané) alebo dokumentáciu na CD.

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

DIRAC V PROSTREDÍ MICROSOFT WINDOWS: ADAPTÁCIA A VYSOKOVÝKONNÉ POČÍTANIE

Systémová dokumentácia

Banská Bystrica 2015 **Bc. Ivan Hraško**

1. Natívna verzia pre Windows s OpenBLAS (potrebný softvér)

- **CMake**
  - <http://www.cmake.org/>
- **MinGW-w64**
  - 32-bitová verzia: <http://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win32/Personal%20Builds/mingw-builds/>
  - 64-bitová verzia: <http://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win64/Personal%20Builds/mingw-builds/>
- **Python**
  - verzia minimálne 2.7 (ale nie 3)
  - pre 32-bitové systémy *x86* a pre 64-bitové *x86-64*
  - <https://www.python.org/>
- **Knižnica zlib**
  - 32-bitová verzia: <http://www.zlib.net/>
  - 64-bitová verzia: <http://sourceforge.net/projects/mingw-w64/files/External%20binary%20packages%20%28Win64%20hosted%29/Binaries%20%2864-bit%29/>
- **Boost**
  - potrebné komponenty: *system*, *filesystem*, *test*, *chrono*, *timer*
  - konfigurácia (*bootstrap*) pre *mingw*
  - kompilovanie (*b2*) s *gcc*
  - <http://www.boost.org/>
- **OpenBLAS**
  - pre 32-bitové systémy 32-bitovú verziu (*Win32*)
  - pre 64-bitové systémy 64-bitovú verziu (ďalej sa delí podľa podpory 64-bitového typu *integer* na *Win64-int32* a *Win64-int64*)
  - je potrebné overiť si mikroarchitektúru procesora (napr. NEHALEM), môže byť potrebné skompilovať si vlastnú knižnicu
  - <http://www.openblas.net/>
- **Git** (podľa potreby)
  - <http://git-scm.com/>

Pre kompatibilitu s „Používateľská príručka“ odporúčame nainštalovať:

- knižnicu OpenBLAS do *C:\OpenBLAS*
- knižnicu zlib do *C:\Program Files\zlib*
- Boost do *C:\Boost*

PATH

CMake, MinGW-w64 kompilátory, Python, knižnicu zlib a OpenBLAS (a prípadne Git) je potrebné pridať do PATH, napr:

C:\cmake-3.0.2-win32-x86\bin;C:\mingw64\bin;C:\Python27;C:\Python27\Scripts

C:\Program Files\zlib;C:\OpenBLAS;C:\Git\cmd

Upozornenie: voliteľné Unix nástroje dostupné v inštalácii Git sa nesmú nachádzať v premennej prostredia PATH.

OPENBLAS_NUM_THREADS

Na nastavenie počtu paralelných vlákien využívaných knižnicou OpenBLAS sa používa premenná prostredia OPENBLAS_NUM_THREADS, napr. s hodnotou 2 (až do maximálneho počtu, ktorý daná verzia podporuje). Viac informácií na: https://github.com/xianyi/OpenBLAS/wiki

Pre viac informácií (napr. aké nástroje je potrebné nainštalovať pre generovanie dokumentácie) viď aktuálnu dokumentáciu na: http://diracprogram.org/doku.php (odporúčané) alebo dokumentáciu na CD.

2. Verzia v prostredí Cygwin s OpenBLAS (potrebný softvér)

- **Cygwin**
  - <https://www.cygwin.com/>
- **Kompilátory**
  - **balíky:** *gcc-core*, *gcc-fortran*, *gcc-g++, gcc-ada*, *gcc-objc*, *gcc-objc++*
- **Make a CMake**
  - **balíky:** *make, cmake*
- **Python**
  - **balík:** *python*
- **Knižnica zlib**
  - **balíky:** *zlib0*, *zlib-devel*
- **OpenBLAS**
  - **balík:** *libopenblas*
- **Git** (podľa potreby)
  - **balík:** *git*

Na nastavenie počtu paralelných vlákien využívaných knižnocou OpenBLAS sa používa premenná prostredia OPENBLAS_NUM_THREADS. Pre viac informácií viď: https://github.com/xianyi/OpenBLAS/wiki

Pre viac informácií (napr. aké balíky je potrebné nainštalovať pre generovanie dokumentácie) a prípadné riešenie problémov, napr. ak sú v Cygwin nájdené knižnice z Windows, viď aktuálnu dokumentáciu na: http://diracprogram.org/doku.php (odporúčané) alebo dokumentáciu na CD.

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

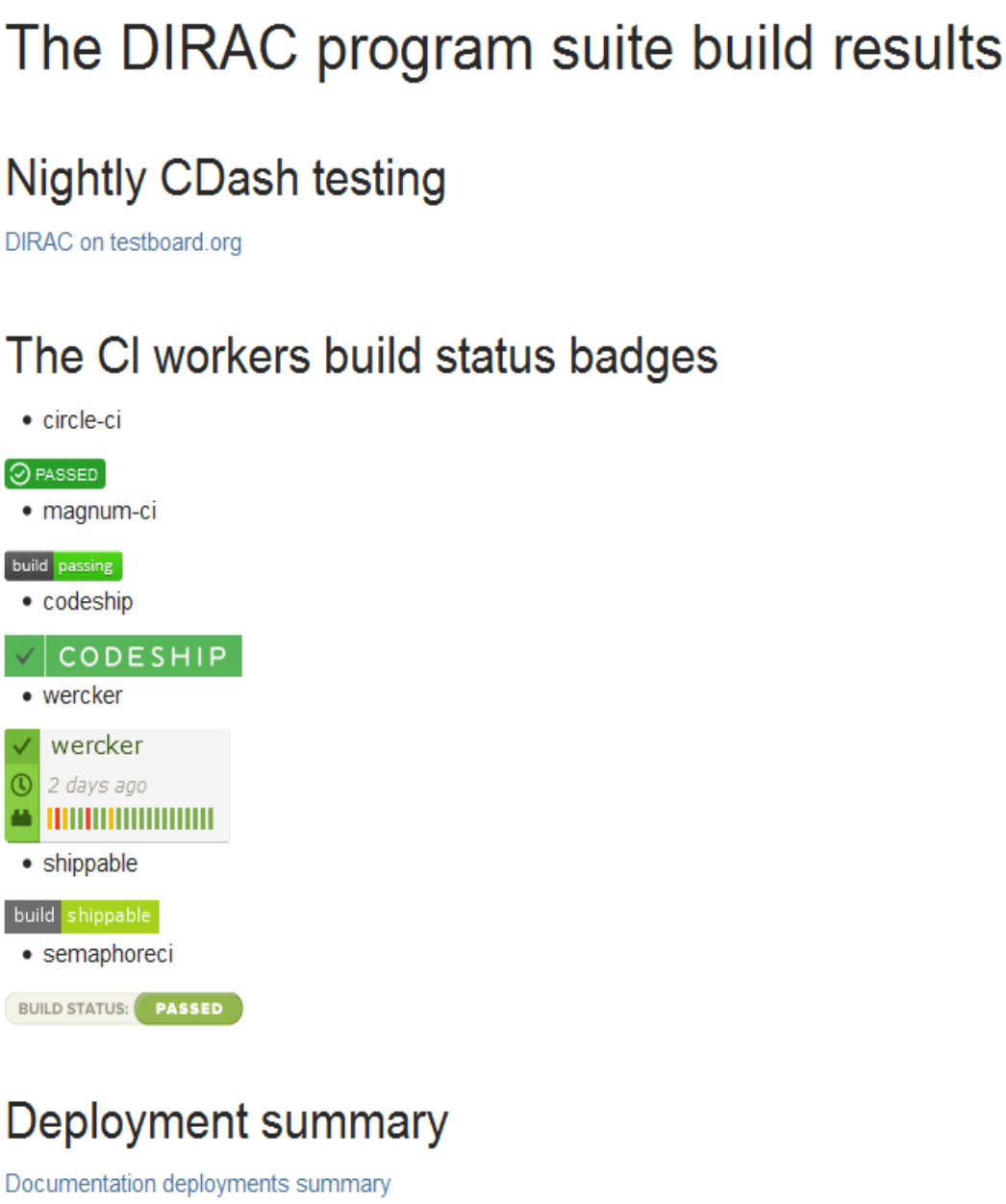
DIRAC V PROSTREDÍ MICROSOFT WINDOWS: ADAPTÁCIA A VYSOKOVÝKONNÉ POČÍTANIE

Obrazová príloha

Banská Bystrica 2015 **Bc. Ivan Hraško**

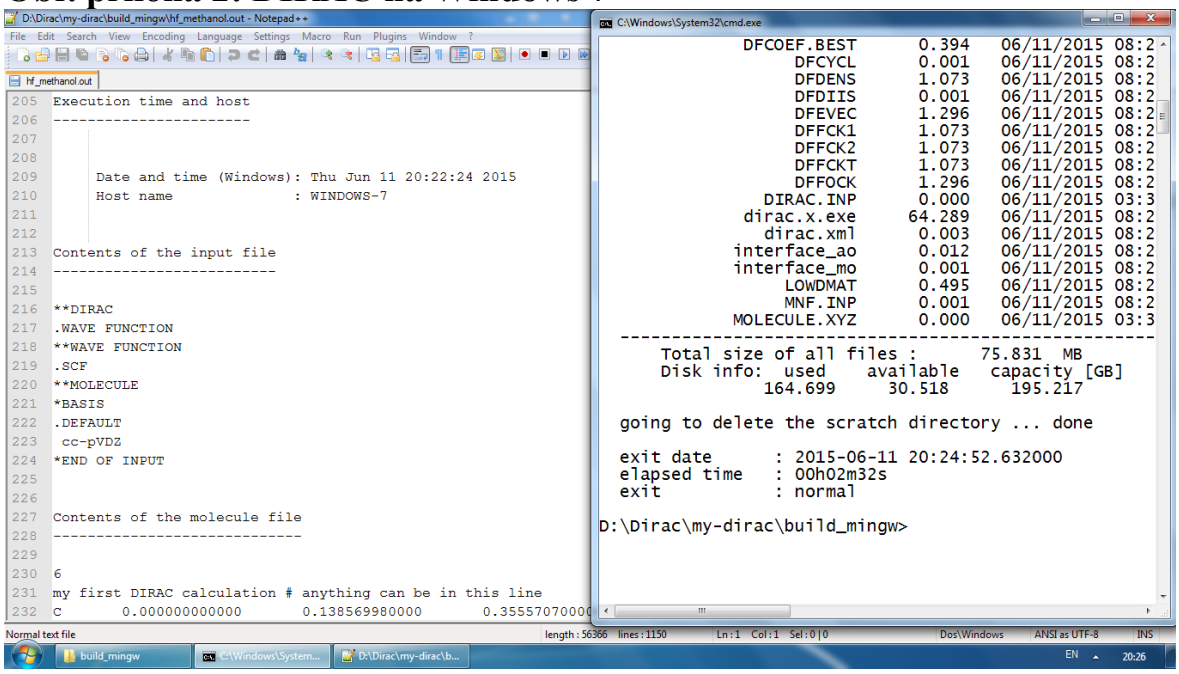
Zoznam obrazových príloh

Obr. príloha 1: DIRAC readme stránka



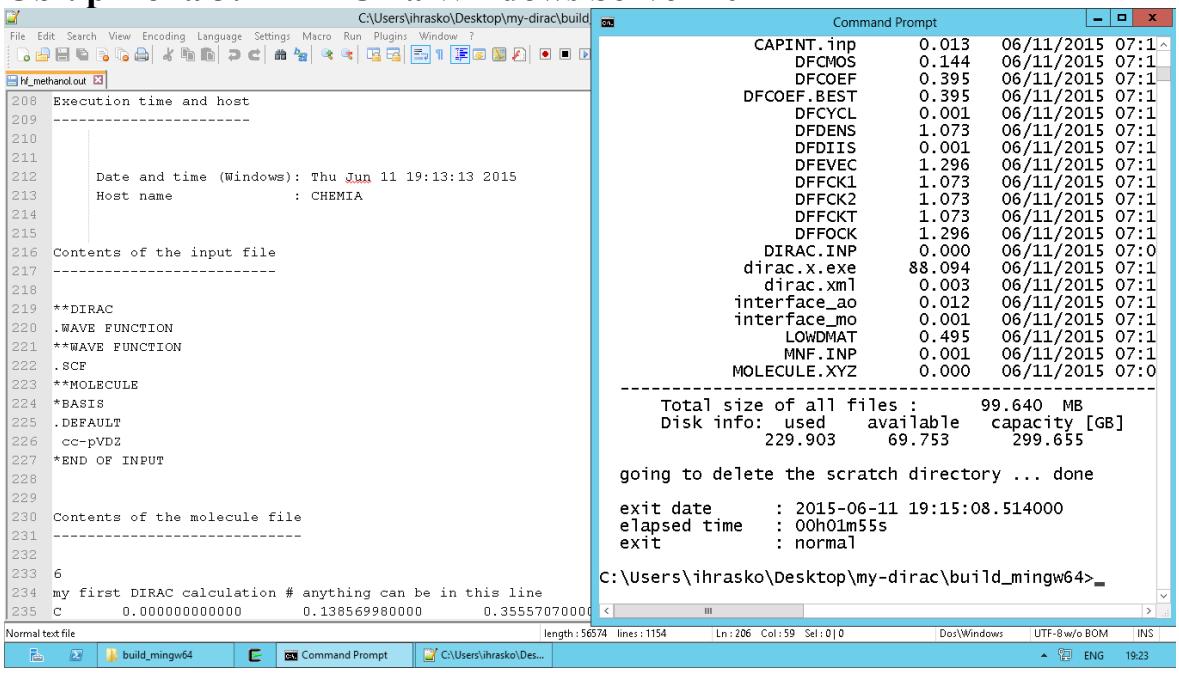
DIRAC „readme“ stránka zobrazená v repozitári na GitHub. Tu sa zobrazujú výsledky z používaných služieb priebežnej integrácie.

Obr. príloha 2: DIRAC na Windows 7



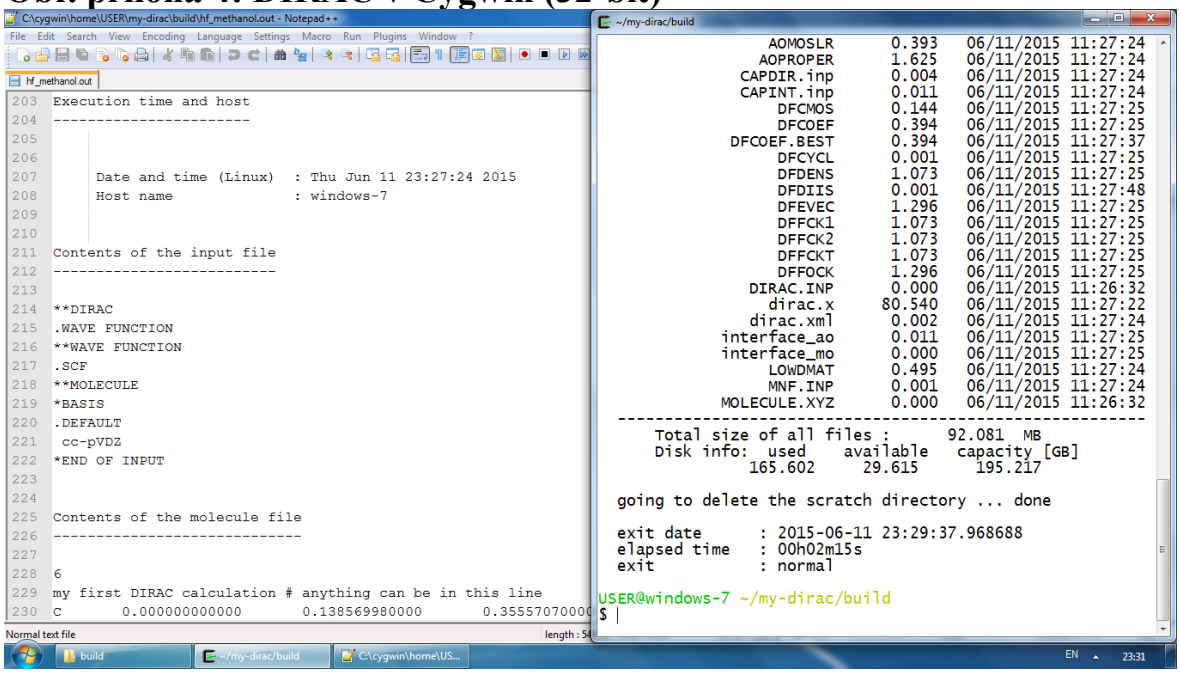
Program DIRAC úspešne vykonal výpočet uvedený v „Používateľská príručka“ na 32-bitovom systéme Windows 7.

Obr. príloha 3: DIRAC na Windows Server 2012



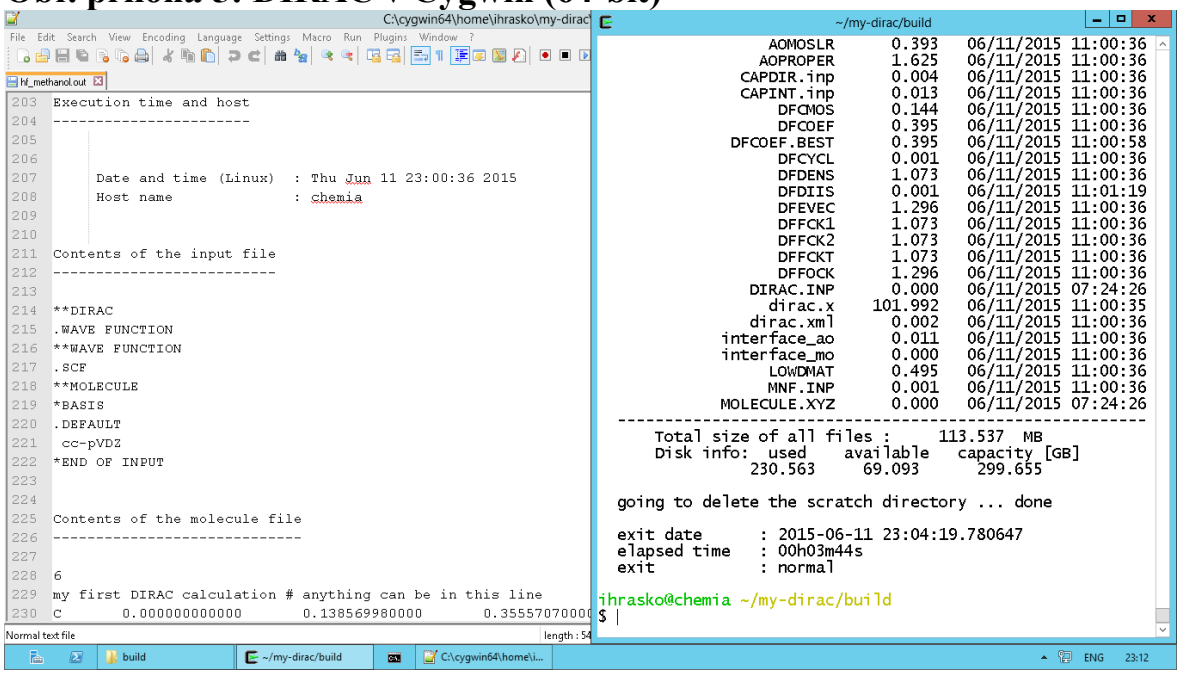
Program DIRAC úspešne vykonal výpočet uvedený v „Používateľská príručka“ na 64-bitovom systéme Windows Server 2012 - Server „Chémia“.

Obr. príloha 4: DIRAC v Cygwin (32-bit)



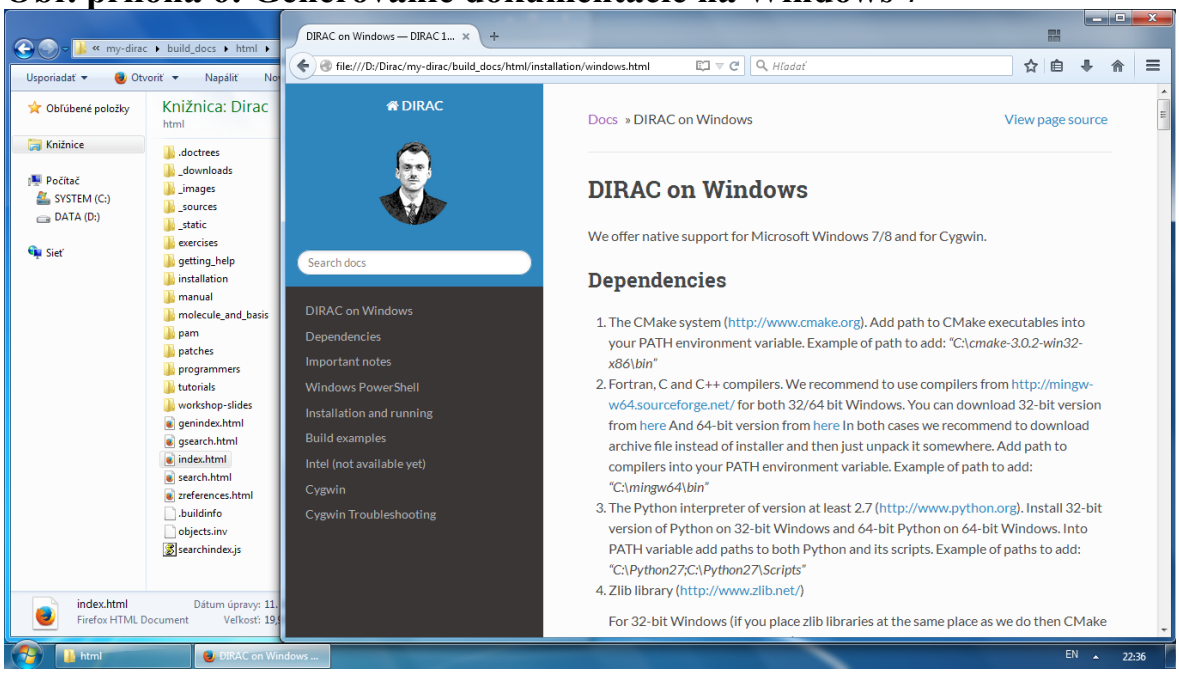
Program DIRAC úspešne vykonal výpočet uvedený v „Používateľská príručka“ v 32-bitovej verzii prostredia Cygwin.

Obr. príloha 5: DIRAC v Cygwin (64-bit)



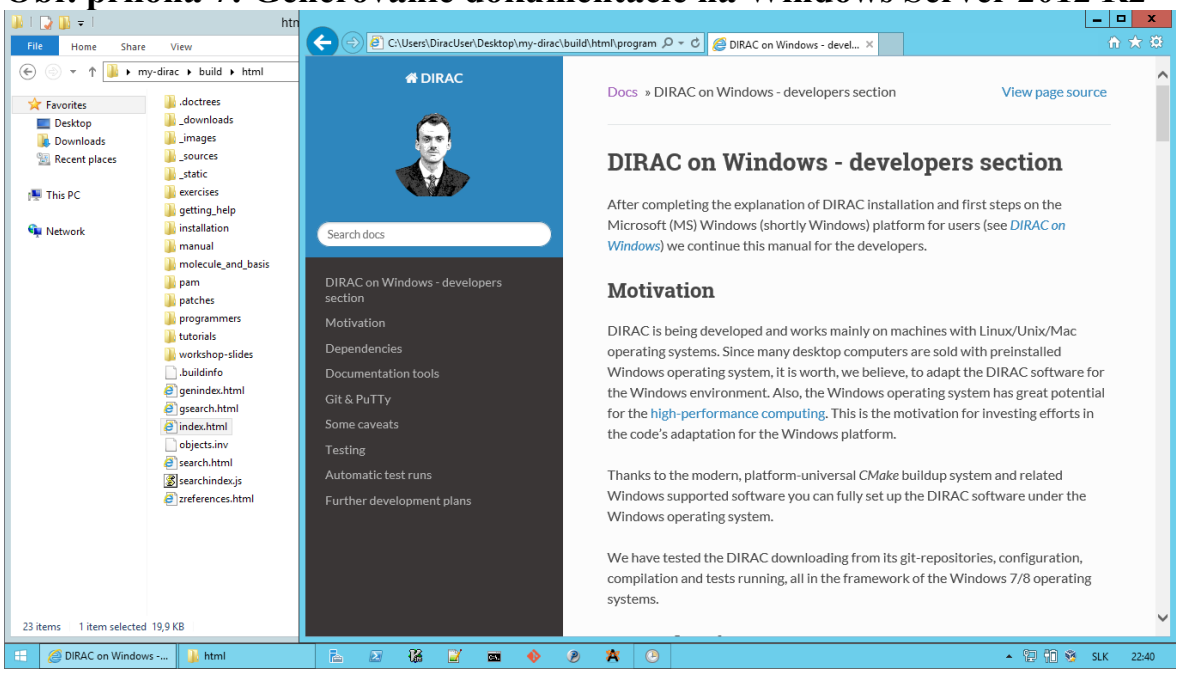
Program DIRAC úspešne vykonal výpočet uvedený v „Používateľská príručka“ v 64-bitovej verzii prostredia Cygwin - Server „Chémia“ (pozn.: je použitý profil *debug*).

Obr. príloha 6: Generovanie dokumentácie na Windows 7



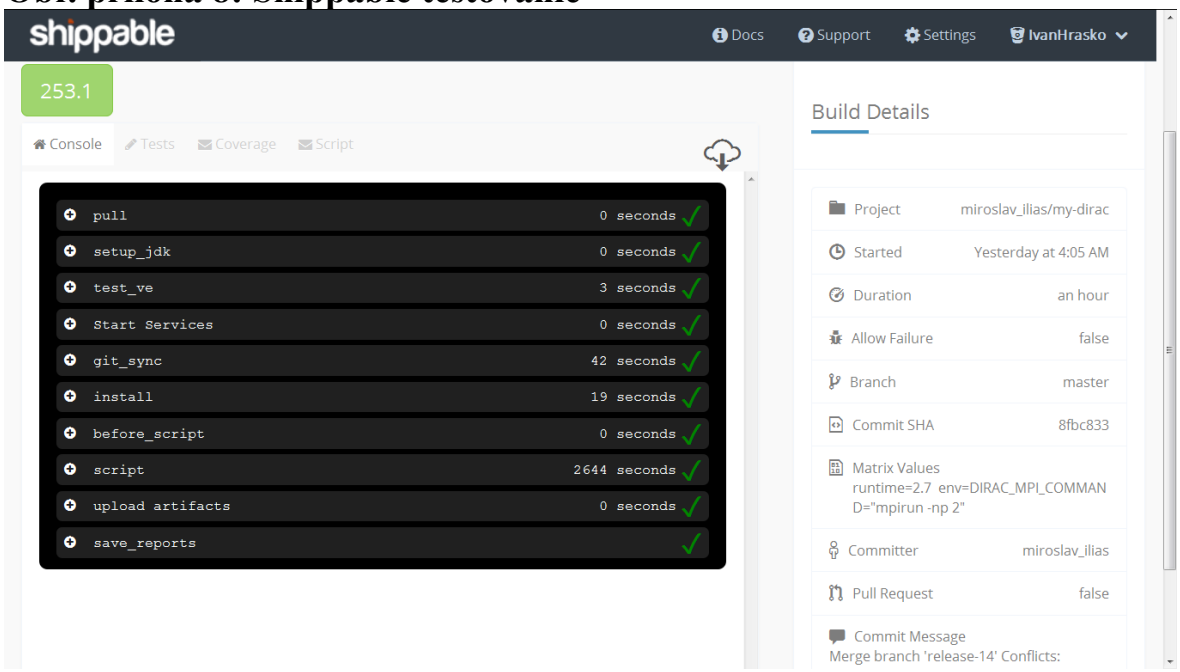
Vygenerovaná dokumentácia pre používateľov na 32-bitovom systéme Windows 7.

Obr. príloha 7: Generovanie dokumentácie na Windows Server 2012 R2



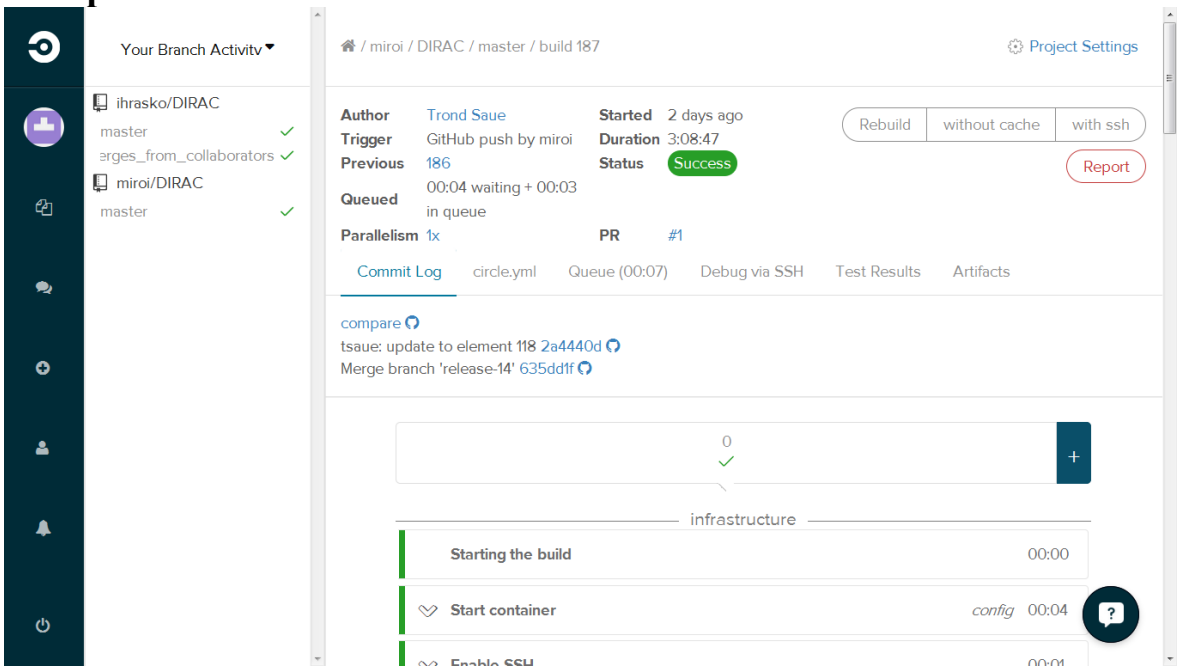
Vygenerovaná dokumentácia pre používateľov na 64-bitovom systéme Windows Server 2012 R2 - nainštalovaný vo VirtualBox.

Obr. príloha 8: Shippable testovanie



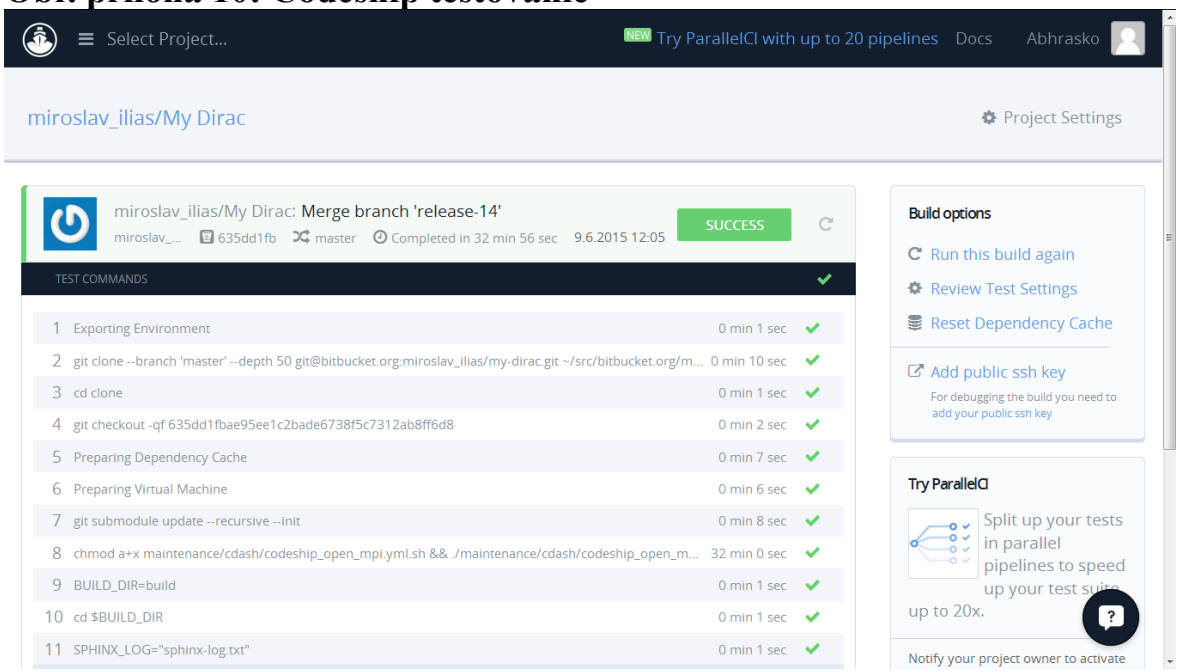
Pomocou služby priebežnej integrácie Shippable je program DIRAC pravidelne testovaný s využitím paralelizmu dostupného v knižnici MPICH.

Obr. príloha 9: CircleCI testovanie



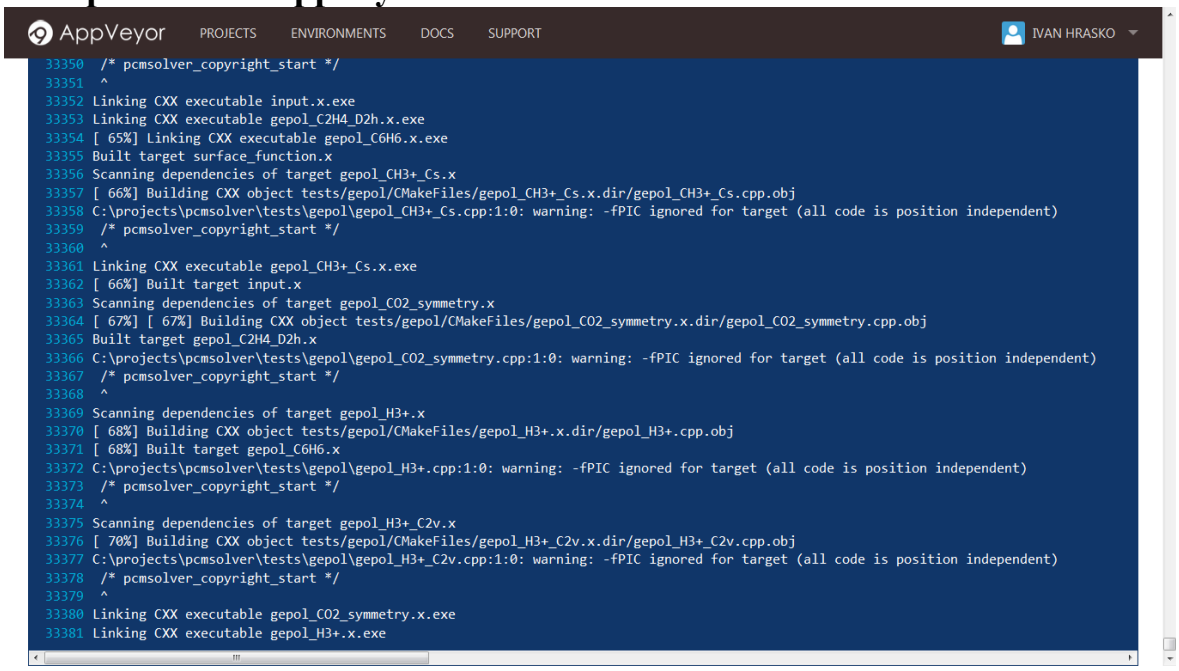
Pomocou služby priebežnej integrácie CircleCI je program DIRAC pravidelne testovaný s využitím matematickej knižnice ATLAS a paralelizácie dostupnej v knižnici Open MPI.

Obr. príloha 10: Codeship testovanie



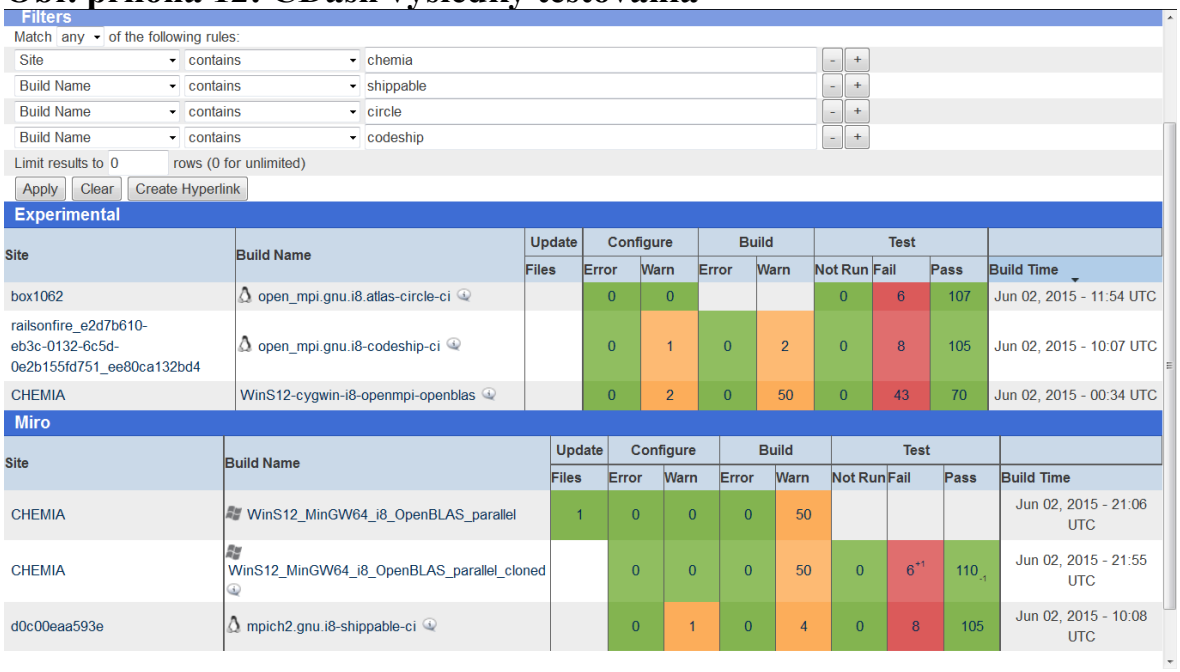
Pomocou služby priebežnej integrácie Codeship je program DIRAC pravidelne testovaný s využitím paralelizácie dostupnej v knižnici Open MPI.

Obr. príloha 11: AppVeyor testovanie



Jediná nám známa služba priebežnej integrácie, ktorá podporuje testovanie na systéme Windows je AppVeyor. Táto je však značne výkonnostne obmedzená, preto ju nevyužívame. Na obrázku je pokus o testovanie projektu PCMSolver (súčasť programu DIRAC). Pre obmedzenia sa kompilovanie programu nepodarilo dokončiť.

Obr. príloha 12: CDash výsledky testovania



Filters

Match any of the following rules:

Site	contains	chemia
Build Name	contains	shippable
Build Name	contains	circle
Build Name	contains	codeship

Limit results to 0 rows (0 for unlimited)

Apply Clear Create Hyperlink

Experimental

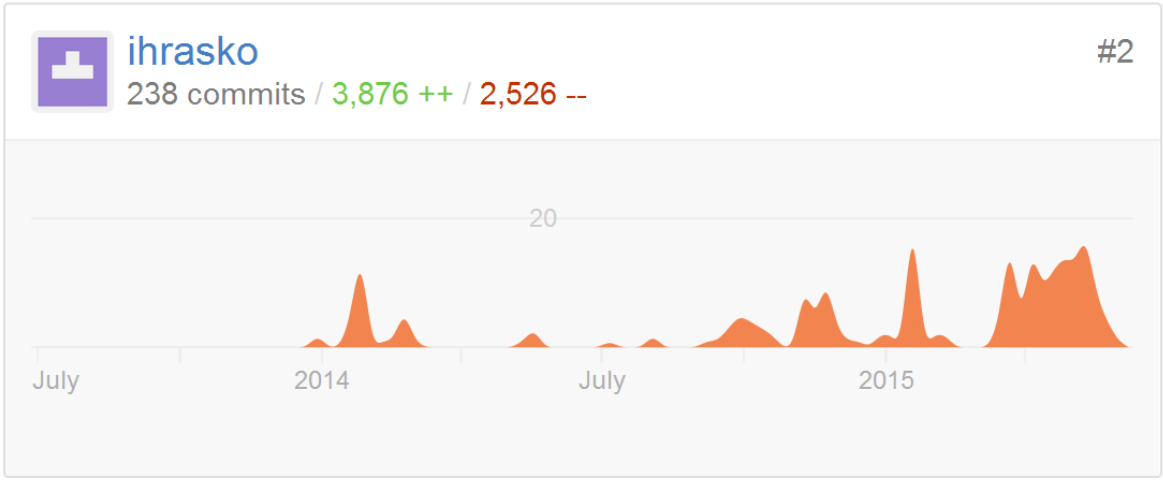
Site	Build Name	Update	Configure		Build		Test			
		Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	Build Time
box1062	open_mpi.gnu.i8.atlas-circle-ci		0	0			0	6	107	Jun 02, 2015 - 11:54 UTC
railsonfire_e2d7b610-eb3c-0132-6c5d-0e2b155fd751_ee80ca132bd4	open_mpi.gnu.i8-codeship-ci		0	1	0	2	0	8	105	Jun 02, 2015 - 10:07 UTC
CHEMIA	WinS12-cygwin-i8-openmpi-openblas		0	2	0	50	0	43	70	Jun 02, 2015 - 00:34 UTC

Miro

Site	Build Name	Update	Configure		Build		Test			
		Files	Error	Warn	Error	Warn	Not Run	Fail	Pass	Build Time
CHEMIA	WinS12_MinGW64_i8_OpenBLAS_parallel	1	0	0	0	50				Jun 02, 2015 - 21:06 UTC
CHEMIA	WinS12_MinGW64_i8_OpenBLAS_parallel_cloned		0	0	0	50	0	6 +1	110 -1	Jun 02, 2015 - 21:55 UTC
d0c00eaa593e	mpich2.gnu.i8-shippable-ci		0	1	0	4	0	8	105	Jun 02, 2015 - 10:08 UTC

Zobrazenie výsledkov testovania uskutočneného na serveri „Chémia“ a v službách priebežnej integrácie Shippable, CircleCI a Codeship na *cdash* web stránke určenej pre program DIRAC.

Obr. príloha 13: Git graf zmien do DIRAC



Zmeny do projektu DIRAC, ako boli zaznamenané na GitHub. Celý výpis sa nachádza v prílohe na CD.

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

DIRAC V PROSTREDÍ MICROSOFT WINDOWS: ADAPTÁCIA A VYSOKOVÝKONNÉ POČÍTANIE

Git log ku kapitole 4.11 Ďalšia práca

Banská Bystrica 2015 **Bc. Ivan Hraško**

Úplný Git log ku všetkým prácam vykonaným na programe DIRAC alebo PCMSolver je priložený na sprievodnom CD.

```
commit 0cd250042cf6d291a3faa2f1e88b440933b8bf1e
Author: IvanHrasko <***hrasko***@***.com>
Date: Mon Jul 7 22:55:31 2014 +0200
ihrasko:add profiling flags for C and C++
cmake/compilers/CFlags.cmake | 7 ++++++-
cmake/compilers/CXXFlags.cmake | 5 +++++
2 files changed, 11 insertions(+), 1 deletion(-)

commit abf015a8e73f61a902ac77280da9cf1ba40713c5
Author: IvanHrasko <***hrasko***@***.com>
Date: Sun Aug 3 18:38:14 2014 +0200
ihrasko: profiling flags for c and c++ are not tested (become commented)
cmake/compilers/CFlags.cmake | 10 +++++-----
cmake/compilers/CXXFlags.cmake | 10 +++++-----
2 files changed, 10 insertions(+), 10 deletions(-)

commit 1de8ea12b146fff8e2cb00653fb50c76b22ff19c
Author: IvanHrasko <***hrasko***@***.com>
Date: Sun Sep 28 20:28:02 2014 +0200
ihrasko: profiling flags for c and c++
cmake/compilers/CFlags.cmake | 29 ++++++++++++++++++-------------
cmake/compilers/CXXFlags.cmake | 26 ++++++++++++++++----------
cmake/compilers/FortranFlags.cmake | 22 ++++++++++++++++++------
3 files changed, 47 insertions(+), 30 deletions(-)

commit 85a30ea546e9abc5269d6dd98528a369cbf74886
Author: IvanHrasko <***hrasko***@***.com>
Date: Fri Jan 2 22:27:47 2015 +0100
ihrasko: we need git.exe (see programmers docs for windows how to install
git)
cmake/FindGit.cmake | 9 ---------
1 file changed, 9 deletions(-)

commit bb7e95b1cea5335510aa64577db44c2feb5ecd81
Author: IvanHrasko <***hrasko***@***.com>
Date: Tue Jan 6 21:03:36 2015 +0100
ihrasko: notes about FindGit.cmake
cmake/FindGit.cmake | 5 +++++
1 file changed, 5 insertions(+)

commit 263147c4a462957c1e0163a9e57acbea9bbefa53
Author: IvanHrasko <***hrasko***@***.com>
Date: Wed Nov 12 09:13:52 2014 +0100
ihrasko: specify custom compilers in setup
note: previous way was not working everywhere
see: http://www.cmake.org/Wiki/CMake_FAQ#How_do_I_use_a_different_compiler.3F
setup | 12 ++++++------
1 file changed, 6 insertions(+), 6 deletions(-)

commit f9510c2084203da13b709d37f0039e3032374dec
Author: IvanHrasko <***hrasko***@***.com>
Date: Wed Nov 12 10:34:33 2014 +0100
ihrasko: we dont need different processing for windows and linux anymore
setup | 6 +-----
1 file changed, 1 insertion(+), 5 deletions(-)
```

```
commit fdca1f7ee4f7a3352aa4e89d6aecf9cd87be80ab
Author: IvanHrasko <***hrasko***@***.com>
Date: Wed Nov 12 10:30:53 2014 +0100
ihrasko: bugfix
cmake/compilers/CFlags.cmake | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)

commit 98412602f40a36dbd47b46cd049907e4caa3962f
Author: IvanHrasko <***hrasko***@***.com>
Date: Fri Nov 28 18:49:29 2014 +0100
ihrasko: avoid to have definitions for builtin blas/lapack twice in
definitions when type is auto and (but) libary is not found (like this):
USE_BUILTIN_BLAS;USE_BUILTIN_LAPACK;USE_BUILTIN_BLAS;USE_BUILTIN_LAPACK
CMakeLists.txt | 4 ++--
1 file changed, 2 insertions(+), 2 deletions(-)

commit 45d2b375c8a386e28240d3911eea7241ead2972c
Author: IvanHrasko <***hrasko***@***.com>
Date: Sun Nov 30 18:02:48 2014 +0100
ihrasko: find python version string for older cmake
CMakeLists.txt | 13 +++++++++++++
1 file changed, 13 insertions(+)

commit fd712e3a309694087bf6cc423ba8fbb1bdf1fc61
Author: IvanHrasko <***hrasko***@***.com>
Date: Fri Nov 14 17:41:13 2014 +0100
ihrasko: fix for stex
src/prp/pamstex.F | 8 ++++++--
1 file changed, 6 insertions(+), 2 deletions(-)

commit 769b38f74d65bc58d8785a9bdc14603371c2182f
Author: IvanHrasko <***hrasko***@***.com>
Date: Fri Nov 21 22:03:57 2014 +0100
ihrasko: fix for gfortran on windows with 64bit integers
CMakeLists.txt | 14 +++++++++++++-
1 file changed, 13 insertions(+), 1 deletion(-)

commit 61dff4893dc3ab5916dc0bd92b967329b8c59602
Author: IvanHrasko <***hrasko***@***.com>
Date: Sat Nov 22 21:58:21 2014 +0100
ihrasko: non-interactive debugger for windows
pam | 7 ++++---
1 file changed, 4 insertions(+), 3 deletions(-)

commit 37ae87693e84437f566f6f498262a8ed7dc420d3
Author: IvanHrasko <***hrasko***@***.com>
Date: Mon Dec 15 22:04:19 2014 +0100
ihrasko: time and hostname for windows in program output
src/gp/gptrygve.F | 5 +++++
src/gp/time.F | 7 +++++--
2 files changed, 10 insertions(+), 2 deletions(-)

commit 66a1da3ec8f32ddcc307ad629d51f37f91fc2e31
Author: IvanHrasko <***hrasko***@***.com>
Date: Wed Jan 7 18:47:43 2015 +0100
ihrasko: hostname for program output first try to get from environment variable
if it is not set then from fortran function, if it is blank then not found
src/gp/time.F | 39 +++++++++++++++++++++------------------------
1 file changed, 21 insertions(+), 18 deletions(-)
```

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

DIRAC V PROSTREDÍ MICROSOFT WINDOWS: ADAPTÁCIA A VYSOKOVÝKONNÉ POČÍTANIE

Obsah sprievodného CD

Banská Bystrica 2015 **Bc. Ivan Hraško**

CD (obsah súborov):

DIRAC on Windows - developers section - DIRAC 14.0 documentation.pdf

- *html* dokumentácia pre vývojárov na Windows vytlačená z diracprogram.org

DIRAC on Windows - DIRAC 14.0 documentation.pdf

- *html* dokumentácia pre používateľov na Windows vytlačená z diracprogram.org

Git Log Dirac.txt

- úplný Git záznam o vykonaných zmenách v programe DIRAC

Git Log PCMSolver.txt

- úplný Git záznam o vykonaných zmenách v programe PCMSolver

index.html

- spustenie celej *html* dokumentácie (bola vygenerovaná na systéme Windows)

ivan_hrasko_diplomova_praca_dirac.pdf

- elektronická verzia diplomovej práce

V adresári testovanie sa nachádzajú súbory, v ktorých bol zaznamenaný priebeh kompilovania a testovania (prípadne generovania dokumentácie) programu DIRAC na 32-bitovom systéme Windows 7 (nainštalovaný 32-bitový Cygwin) a 64-bitových systémoch Windows Server 2012 (server „Chémia“) a Windows Server 2012 R2 (na oboch nainštalovaný 64-bitový Cygwin).

Počas testovania sa potvrdila dôležitosť potreby priebežnej integrácie i na systéme Windows, pretože sa objavili problémy s profilom *release* (vyššia optimalizácia), ktorý sme pri vývoji až tak často nepoužívali. Toto sa týka najmä problému s PCMSolver-om na Windows 7. Tu bolo použité riešenie, že sa PCMSolver vypol alebo sa použil profil *debug*. Kde nie je uvedené, bol zapnutý profil *release*. Používateľ by mohol taktiež použiť riešenie, že odskúša možnosti pre kompilátory pomocou funkcionality zadávania vlastných možností (*custom flags*), ktorú sme do konfigurácie programu DIRAC pridali (kap. 4.3). Všetky súbory sú textové s príponou *.log*, ktorú je možné zameniť za *.txt*.

testovanie\windows-12-chemia

- testovanie uskutočnené na Windows Server 2012 - server „Chémia“
- **cygwin-openblas.log**
  - skompilovanie a otestovanie programu v 64-bitovej verzii prostredia Cygwin vrátane modulu PCMSolver s využitím knižnice OpenBLAS
- **cygwin-openmpi-openblas.log**
  - skompilovanie a otestovanie programu v 64-bitovej verzii prostredia Cygwin vrátane modulu PCMSolver s využitím knižníc Open MPI a OpenBLAS
  - pravdepodobne pre chybu kompilátora alebo v knižnici Open MPI končí neúspešne relatívne veľa testov
- **mingw-nativne-openblas32.log**
  - skompilovanie a otestovanie programu v natívnom 64-bitovom prostredí Windows s využitím 32-bitovej knižnice OpenBLAS (vrátane modulu PCMSolver)
- **mingw-nativne-openblas64.log**
  - skompilovanie a otestovanie programu v natívnom 64-bitovom prostredí Windows s využitím 64-bitovej knižnice OpenBLAS (vrátane modulu PCMSolver)

testovanie\windows-12-r2-dokumentacia

- testovanie uskutočnené na Windows Server 2012 R2
- **cygwin.log**
  - vytváranie dokumentácie v 64-bitovej verzii prostredia Cygwin
- **nativne.log**
  - vytváranie dokumentácie v 64-bitovom natívnom prostredí Windows

testovanie\windows-7

- testovanie uskutočnené na Windows 7
- **clang-nativne-nopcm-openblas.log**
  - skompilovanie a otestovanie programu v natívnom 32-bitovom prostredí Windows s C a C++ kompilátormi LLVM Clang (a MinGW-w64 Fortran kompilátorom) s využitím knižnice OpenBLAS (bez modulu PCMSolver)
- **cygwin-openblas.log**
  - skompilovanie a otestovanie programu v 32-bitovej verzii prostredia Cygwin vrátane modulu PCMSolver s využitím knižnice OpenBLAS
- **dokumentacia-cygwin.log**
  - vytvorenie dokumentácie v 32-bitovej verzii prostredia Cygwin
- **dokumentacia-nativne.log**
  - vytvorenie dokumentácie v 32-bitovom natívnom prostredí Windows
- **mingw-nativne-nopcm-openblas.log**
  - skompilovanie a otestovanie programu v natívnom 32-bitovom prostredí Windows s kompilátormi MinGW-w64 s využitím knižnice OpenBLAS (bez modulu PCMSolver)
- **mingw-nativne-pcmsolver-netlib.log**
  - skompilovanie a otestovanie programu v natívnom 32-bitovom prostredí Windows s kompilátormi MinGW-w64 s využitím knižníc Netlib (vrátane modulu PCMSolver, zapnutý profil *debug*)
- **mingw-nativne-profile-nopcm-openblas.log**
  - skompilovanie a otestovanie programu v natívnom 32-bitovom prostredí Windows s kompilátormi MinGW-w64 s využitím knižnice OpenBLAS (bez PCMSolver), zapnutá možnosť pre profilovanie (viď „Používateľská príručka“)

CD (hierarchia súborov)

```
├────diracprogram.org
│       DIRAC on Windows - developers section - DIRAC 14.0 documentation.pdf
│       DIRAC on Windows - DIRAC 14.0 documentation.pdf
│
├────Git Log
│       Git Log Dirac.txt
│       Git Log PCMSolver.txt
│
├────html
...
│       index.html
...
├────PDF verzia
│       ivan_hrasko_diplomova_praca_dirac.pdf
│
└────testovanie
    ├────windows-12-chemia
    │       cygwin-openblas.log
    │       cygwin-openmpi-openblas.log
    │       mingw-nativne-openblas32.log
    │       mingw-nativne-openblas64.log
    │
    ├────windows-12-r2-dokumentacia
    │       cygwin.log
    │       nativne.log
    │
    └────windows-7
            clang-nativne-nopcm-openblas.log
            cygwin-openblas.log
            dokumentacia-cygwin.log
            dokumentacia-nativne.log
            mingw-nativne-nopcm-openblas.log
            mingw-nativne-pcmsolver-netlib.log
            mingw-nativne-profile-nopcm-openblas.log
```