Fakulta informačních technologií, Vysoké učení technické v Brně

Architektury výpočetních systémů (AVS 2021)
Projekt č. 1: Vektorizace kódu

Vojtěch Mrázek (mrazek@fit.vutbr.cz)

Termín odevzdání: 12. 11. 2021

Changelog

- 10. 10. 2021: oprava překlepu CCMake na CMake
- 18. 10. 2021: explicitní určení paralelizace (modře)

1. Úvod

Cílem tohoto projektu je zrychlit výpočet tzv. Mandelbrotovy množiny. Její základní výpočet vektorizovatelný téměř není, ovšem šikovným přeskládáním smyček a přidáním dalších parametrů je možné dosáhnout rozumného zrychlení. Profilovacím nástrojem bude Intel Advisor.
Projekt lze přeložit a spustit prakticky kdekoliv (vyžaduje CMake a kompilátor), budeme se ale spoléhat na optimalizační reporty poskytované Intel kompilátorem. Je žádoucí, aby cílový procesor disponoval podporou vektorového rozšíření AVX-512. Referenčním strojem je **výpočetní klastr Barbora s kompulátorem Intel** (viz A), na jehož výpočetním uzlu provádějte všechna měření do tohoto projektu. K ladění vašeho řešení můžete využít například i počítače v CVT (mimo serverů jako je Merlin, protože tam je omezená RAM na 1 GB) (viz B) nebo vlastní počítač, jestliže si nainstalujete nástroje od Intelu (jsou pro studenty zdarma). Pro prvotní ladění funkcionality algoritmu je možné využít i překladač GCC.

2. Mandelbrotova množina

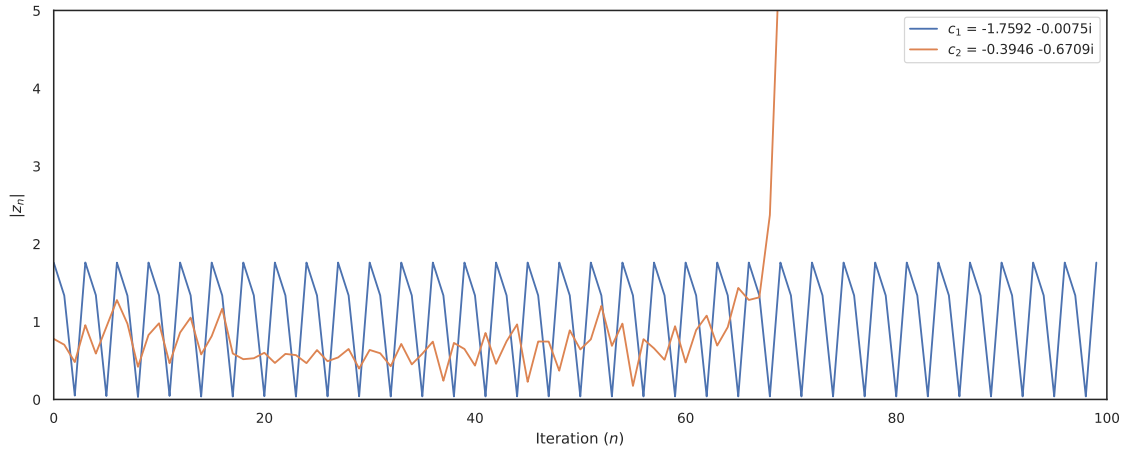
Vlastní implementace Mandelbrotovy množiny není cílem vaší práce. Přesto však je důležité získat představu o jejím fungování. Mandelbrotova množina[1] je množina bodů komplexní roviny získané rekurzivním procesem. Je to jeden z nejznámějších fraktálů. K jejímu určení se používá zobrazení, které každému komplexnímu číslu c přiřazuje posloupnost komplexních čísel z_n. Tato posloupnost je dána rekurizvním přepisem

$$z_0 = 0, z_{n+1} = z_n^2 + c$$

Mandelbrotovu množinu lze potom definovat jako množinu

$$M = \{c \in \mathbb{C} | c \to c^2 + c \to (c^2 + c)^2 + c \cdots \text{je omezena}\}.$$

Bylo dokázáno, že posloupnost jde k nekonečnu právě tehdy, kdy velikost nějakého členu překročí hodnotu 2.
Na základě tohoto iteračního výpočtu pro každý bod v prostoru (v našem případě pro reálnou část mezi -2 a 1 a pro imaginární mezi -1.5 a 1.5) určíme jednotlivé členy z_n pro $n \in (0, 100)$. Vybral jsem dvě počáteční hodnoty c a vypočítal pro ně absolutní hodnotu $|z_n|$ (obr. 2.1): hodnota c_1, pro kterou se pohybujeme v intervalu do 2, a hodnotu c_2, pro kterou dojde k překročení prahovové hodnoty 2 kolem 70. iterace a v blízké době se hodnota z_n přiblíží k nekonečnu.

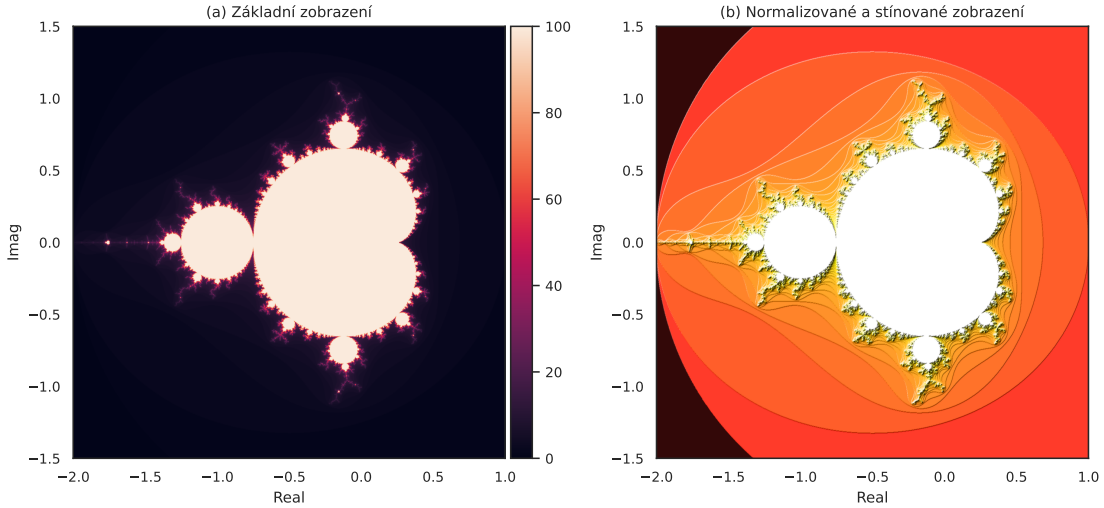


Obrázek 2.1: Průběh absolutní hodnoty $|z_n|$ pro dvě vybrané hodnoty c pro 100 iterací.

Pokud určíme, ve které iteraci došlo k překročení prahové hodnoty 2, dostáváme se k zobrazení na obrázku 2.2a. To můžeme pomocí logaritmické normalizace (power-norm) a stínování zdůraznit, jak je zobrazeno na obrázku 2.2b.
Vaším úkolem bude tedy vektorizovat algoritmus, který se skládá ze tří smyček (viz kód níže, který je i v souboru *RefMandelCalculator.cc*). První dvě smyčky iterují přes všechny

[1] `https://cs.wikipedia.org/wiki/Mandelbrotova_mno%C5%BEina`
`http://kmlinux.fjfi.cvut.cz/~pauspetr/html/skola/fraktaly/reserse.htm#_Toc73066115`

body v prostoru komplexních čísel, třetí (nejvíc zanořená, implementovaná ve funkci `mandelbrot`}) počítá iterativně, zda nedošlo k překročení prahové hodnoty ($|z_n| > 2 \Leftrightarrow im(z_n)^2 + re(z_n)^2 > 4$). Vzorce uvedené ve výpočtu odpovídají násobení komplexních čísel.



Obrázek 2.2: Znázornění iterace, ve které došlo k překročení prahové hodnoty $|z_n| > 2$.

```
template <typename T>
static inline int mandelbrot(T real, T imag, int limit) {
        T zReal = real;
        T zImag = imag;

        for (int i = 0; i < limit; ++i) {
                T r2 = zReal * zReal;
                T i2 = zImag * zImag;

                if (r2 + i2 > 4.0f)
                        return i;

                zImag = 2.0f * zReal * zImag + imag;
                zReal = r2 - i2 + real;
        }
        return limit;
}

int * calculateMandelbrot() {
        int *pdata = data;
        for (int i = 0; i < height; i++) {
                for (int j = 0; j < width; j++) {
                        float x = x_start + j * dx; // current real value
                        float y = y_start + i * dy; // current imaginary value
                        *(pdata++) = mandelbrot(x, y, limit);
                }
```

```
27          }
28          return data;
29  }
```

3. Struktura projektu, překlad a spuštění

Archiv zadání obsahuje:

- `MB-xlogin00.txt`: šablonu otázek, na které je nutné odpovědět
- `scripts/`: pomocné skripty pro evaluaci
- `src/`: adresář se zdrojovými kódy
- `evaluate.pbs`: skript pro cluster, který provede evaluaci
- `advisor.pbs`: skript pro cluster, který naměří hodnoty pro Intel Advisor

Zdrojový kód je rozdělen na pomocné soubory `main.cc`, který zpracovává argumenty příkazové řádky, soubory v `common`, které se starají o ukládání výsledků do NumPy pole, zpracování argumentů příkazové řádky a měření času. Jádro práce nalezneme ve složce `calculators`. Důležité jsou tři kalkulátory označené jako Ref (referenční bez vektorizace), Line (vektorizace po řádcích) a Batch (vektorizace po malých skupinách). Soubory calculators/LineMandelCalculator.* a calculators/BatchMandelCalculator.* jsou jádrem vaší práce.
Pro kompilaci a spuštění je možné použít CMake následujícím způsobem (pro kompilátor Intel)

```
ml intel-compilers/2021.1.2 CMake/3.21.2-GCCcore-9.3.0 # pouze na Barbore
cd Assignment
mkdir build && cd build
CC=icc CXX=icpc cmake ..
make -j
```

Poté je možné spustit vlastní aplikaci s různými parametry příkazové řádky:

```
$ ./mandelbrot -h
AVS Assignment 1 - Mandelbrot calculation using SIMD instructions
Usage:
  AVS: Mandelbrot [OPTION...] <OUTPUT>

  -s, --size arg        Base matrix size (default: 2048)
  -i, --iters arg       Number of iterations (default: 100)
  -c, --calculator arg  Calculator name [ref, batch, line] (default: ref)
      --batch           Run in silent/batch mode
  -h, --help            Print help
```

Můžete měnit základ velikosti matice s, kdy výsledná matice bude mít rozměr $3s \times 2s$, počet iterací algoritmu, kalkulátor (důležité pro vaše testování). Výsledek je možné uložit do komprimovaného NumPy pole (*.npz)[2]. S tímto polem pak pracují skripty pro testování. Prvním přístupem je vizualizace pole uloženého z běhu.

[2] `https://numpy.org/doc/stable/reference/generated/numpy.load.html`

```
$ ./mandelbrot -c ref -s 1024 -i 100 ref.npz
======================== Mandelbrot SIMD calculator ==========================
Calculator:        RefMandelCalculator
Base size:         1024
Matrix size:       3000x2000
Iteration limit:   100
Elapsed Time:      512 ms
$ python3 ../scripts/visualise.py cmp_batch.npz  --show --save img.png
```

Nebo můžete pole navzájem porovnat — pozor, vzhledem k tomu, že kompilátor může měnit pořadí floatových operací, tak nemusí výsledky odpovídat úplně přesně, ale může docházet k drobným odchylkám.

```
$ ./mandelbrot -c batch -s 1024 -i 100 batch.npz
======================== Mandelbrot SIMD calculator ==========================
Calculator:        BatchMandelCalculator
Base size:         1024
Matrix size:       3072x2048
Iteration limit:   100
Elapsed Time:      ??? ms
$ ./mandelbrot -c ref -s 1024 -i 100 ref.npz
======================== Mandelbrot SIMD calculator ==========================
Calculator:        RefMandelCalculator
Base size:         1024
Matrix size:       3072x2048
Iteration limit:   100
Elapsed Time:      ??? ms
$ python3 ../scripts/compare.py batch.npz ref.npz
[ok] Results are very close (eps = 0.007% )
```

Nebo můžete pustit automatický test všech tří implementací (vhodné na konci před odevzdáním).

```
$ bash ../scripts/compare.sh
RefMandelCalculator;512;1536;1024;100;???
LineMandelCalculator;512;1536;1024;100;???
BatchMandelCalculator;512;1536;1024;100;???
Reference vs line
[ok] Results are very close (eps = 0.005% )
Reference vs batch
[ok] Results are very close (eps = 0.005% )
Batch vs line
[ok] Results are same
Test passed
```

4. Postup práce

Vaším úkolem budou čtyři základní kroky, které se budou podílet na hodnocení (max. 10 bodů).

1. Implementovat vektorizovanou verzi po řádcích a po sloupcích
2. Vyhodnotit obě implemenace v Intel Advisor
3. Vyhodnotit efektivitu implementací

4. Odpovědět na otázky v dokumentu `MB-xlogin00.txt`

V referenční implementaci jsou tři smyčky `for`. Vzhledem k datovým závislostem budete muset změnit pořadí smyček tak, aby počítání iterací bylo místo nejnižší úrovně o úroveň výš. Proto bude vhodné průběžně ukládat data do nějaké pomocné paměti. Pro korektní měření **všechny alokace paměti provádějte v konstruktoru kalulátoru**. Vektorizaci provádějte pomocí OpenMP SIMD pragem. Abyste se dostali na rozumnou efektivitu, je nutné nějakým způsobem pro celou vektorizovanou smyčku určit, zda výpočet ukončit ($\forall i : |z_n^i| > 2$ — pomůže vám redukce). Je očekávané, že u vektorizovaného kalkulátoru pouze jedna implementace bude efektivnější než referenční — tuto vlastnost pak vysvětlíte v odpovědích na otázky.
Druhým krokem bude vyhodnotit vektorizaci v nástroji Intel Advisor. Doporučujeme s tímto nástrojem pracovat průběžně. Při spouštění na clusteru Barbora využívejte PBS skript *advisor.pbs* (`qsub advisor.pbs`), který vám rychle spočítá výsledky, do složky `build_advisor/advisor_{calc}` uloží projekty, které si v Advisor gui zobrazíte. Pro připojení na cluster s podporou grafického rozhraní využijte `ssh -X -i  /id_rsa-dd-... dd-21-22-XXX@barbora.it4i.cz` nebo tunelování VNC. Práce s nástrojem Advisor a tunelování přes VNC je detailně popsána v příloze A. V Advisoru `ml Advisor; advixe-gui` otevřete projekt, zvolte možnost *Show my results*. Připomínáme, že login uzel neslouží k náročným výpočtům. Proto na něm také nesmíte spustit příkazy *Collect* nebo *Start survey analysis*, abyste nezatěžovali přihlašovací stroje. Výsledky si můžete stáhnout a analyzovat v lokální instalaci Advisoru, případně v CVT (viz příloha B).
Po analýze kódu můžete vyhodnotit celkovou efektivitu skriptem `qsub evaluate.pbs`. Tento skript spustí několik běhů projektu (paralelně tak, aby nebyl překročen počet jader) a vytvoří soubor `datalog.csv`. Z tohoto souboru vám skript

```
python3 scripts/plot_evaluate.py datalog.csv --save eval.png
```

vytvoří boxploty pro různá nastavení velikosti mřížky a počtu iterací. Tento graf potom odevzdáváte a závěry z něj shrnete v odpovědním formuláři.

4.1. Odevzdávané soubory

Ve výsledném archivu *xlogin00.zip* odevzdávejte pouze šest následujících souborů:

- *LineMandelCalculator.cc/h*
- *BatchMandelCalculator.cc/h*
- *eval.png*
- *MB-xlogin00.txt*

Do jiných souborů nesmíte zasahovat. Před odevzdáním doporučujeme ověřit funkčnost (viz výše).
Případné dotazy řešte prosím přes mail s Vojtěchem Mrázkem, nebo přes fórum ve WISu (ideální pro obecné dotazy, problémy s připojením, atd.).

A. Superpočítač Barbora

Superpočítač Barbora umístěný na VŠB v Ostravě je složen z celkem 192 uzlů, každý uzel je osazen dvěma 18jádrovými procesory Intel Cascade Lake 6240 a 192 GB RAM. Tyto procesory podporují tedy vektorové instrukce AVX-512. Pro připojení na superpočítač Brbora je potřeba mít vytvořený účet, se kterým je možné se připojit na jeden z dvojice tzv. čelních (login) uzlů – `barbora.it4i.cz` (round-robin DNS).
Login uzly **neslouží** ke spouštění náročných úloh, všechny experimenty je nutné provádět na výpočetních uzlech. Tento projekt sice není výpočetně náročný, přesto by aktivita jiných uživatelů na login uzlu mohla zkreslit měření výkonosti. Je však možné využít těchto uzlů k prohlížení získaných profilovacích dat a ke kompilaci.

A.1. Přístup na výpočetní klastr a moduly

Přístup k Barboře je možný výlučně prostřednictvím SSH s použitím Vašeho privátního klíče. V rámci předmětu jste po podepsání souhlasu s podmínkami užívání infrastruktury obdrželi přihlašovací údaje, které jste následně ve cvičení použili k získání tohoto klíče. Jestliže klíčem nedisponujete, je možné ho získat na stránce SCS training [3] (po přihlášení). **Klíč pečlivě uschovejte, budete jej potřebovat při každém přihlašování na klastr.**
V projektu budete využívat grafický nástroj Intel Advisor, který na dálku funguje nejlépe prostřednictvím vzdálené plochy. Na své pracovní stanici si nainstalujte VNC klienta s podporou tunelování přes SSH a postupujte dle návodu v dokumentaci IT4I [4].
Dle tohoto návodu výše může váš postup na klastru pro vytvoření interaktivní úlohy vypadat například následovně:

```
ssh -i ~/.ssh/id_rsa-dd-21-22-255 dd-21-22-255@barbora.it4i.cz
Enter passphrase for key '/home/vojta/.ssh/id_rsa-dd-21-22-255':
  ____             _
 |  _ \           | |
 | |_) | __ _ _ __| |__   ___  _ __ __ _
 |  _ < / _` | '__| '_ \ / _ \| '__/ _` |
 | |_) | (_| | |  | |_) | (_) | | | (_| |
 |____/ \__,_|_|  |_.__/ \___/|_|  \__,_|


  ...running on Red Hat Enterprise Linux 7.x

[dd-21-22-255@login1.barbora ~]$ vncpasswd
Password:
Verify:
Would you like to enter a view-only password (y/n)? n
A view-only password is not used
[dd-21-22-255@login1.barbora ~]$ ps aux | grep Xvnc | sed  -rn 's/(\s) .*Xvnc (\:[0-9]+) .*/\1 \2/p'
    # vyberu si neco volneho
[dd-21-22-255@login1.barbora ~]$ vncserver :3 -geometry 1600x900 -depth 32
xauth:  file /home/training/dd-21-22-255/.Xauthority does not exist

New 'login1.barbora.it4i.cz:3 (dd-21-22-255)' desktop is login1.barbora.it4i.cz:3
```

[3] https://scs.it4i.cz/training
[4] https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/vnc/

```
Creating default startup script /home/training/dd-21-22-255/.vnc/xstartup
Creating default config /home/training/dd-21-22-255/.vnc/config
Starting applications specified in /home/training/dd-21-22-255/.vnc/xstartup
Log file is /home/training/dd-21-22-255/.vnc/login1.barbora.it4i.cz:3.log

dd-21-22-255@login1.barbora ~]$ logout
Connection to barbora.it4i.cz closed.
```

Po skončení práce musíte vašeho klienta ručně ukončit! Jinak dojdou porty na uzlu: `vncserver -kill :3`.

Následně se je možné připojit pomocí VNC klienta. Ten se v příkladu výše musí připojit na `login1.barbora.it4i.cz` (viz hostname v konzoli, na tom konkrétním login uzlu byl puštěn VNC server, není možné použít přímo round-robin DNS) na port 5900 + číslo displeje, v příkladě výše 3, t.j. 5903. Připojení musí být prostřednictvím SSH tunelu, protože port 5903 není přístupný mimo síť it4i. Jestliže váš VNC klient nepodporuje tunelování přes SSH, vytvořte si jej dle zmíněného návodu (v GNU/Linuxu pomocí příkazu `ssh -L 5903:login1.barbora.it4i.cz:5903 dd-21-22-XXX@barbora.it4i.cz`, ve Windowsech např. prostřednictvím PuTTY). Pak VNC server připojíte k lokálnímu počítači na protunelovaný port.

Po připojení ke vzdálené ploše můžete otevřít emulátor terminálu (menu Aplikace → Systémové nástroje → Terminál) a vytvořit interaktivní úlohu následujícím způsobem:

```
[dd-21-22-255@login1.barbora ~]$ qsub advisor.pbs
[dd-21-22-255@login1.barbora ~]$ qstat -u $USER
# pockat, az uloha dobehne
[dd-21-22-255@login1.barbora ~]$ ml Advisor
[dd-21-22-255@login1.barbora ~]$ advixe-gui

# nebo spustit interaktivni ulohu na uzlu
[dd-21-22-255@login1.barbora ~]$ xhost +
access control disabled, clients can connect from any host
[dd-21-22-255@login1.barbora ~]$ qsub -A DD-21-22 -q qexp     \
-l select=1:ncpus=36,walltime=1:00:00,vtune=2019_update8  \
-I -v DISPLAY=$(uname -n):$(echo $DISPLAY | cut -d ':' -f 2)
qsub: waiting for job 531414.isrv1 to start
qsub: job 531414.isrv1 ready

[dd-21-22-255@cn49.barbora ~]$
```

Příkaz `qsub` zadá požadavek na spuštění úlohy do fronty; jakmile bude v systému dostatek volných uzlů, dojde ke spuštění úlohy. Parametr `-A` určuje projekt, v rámci kterého máme alokované výpočetní hodiny (neměnit), `-q` určuje frontu, do které bude úloha zařazena (pokud budete na spuštění úlohy čekat příliš dlouho, můžete zkusit frontu `qprod`, ale preferujte `qexp`), parametr `-l` určuje zdroje, které budou úloze přiděleny (počet uzlů, počet procesorů, čas) a další možnosti (v našem případě načtení modulu pro profiler). Interaktivní úlohu pak získáte parametrem `-I`. Pomocí `-v` definujeme proměnnou prostředí `DISPLAY` pro připojení grafických aplikací. Více o spouštění úloh na superpočítačích IT4I naleznete v dokumentaci[5].

[5] `https://docs.it4i.cz/anselm/job-submission-and-execution/`

Nyní jste již v terminálu připojeni k výpočetnímu uzlu (viz nový hostname v terminálu) s tím, že by jste měli být schopní spustit grafickou aplikaci:

```
[dd-21-22-255@cn49.barbora ~]$ ml VTune/2019_update8 Advisor/2020_update3
[dd-21-22-255@cn49.barbora ~]$ amplxe-gui
[dd-21-22-255@cn49.barbora ~]$ advixe-gui
```

Software na superpočítači je dostupný pomocí tzv. *modulů*. Práci s nimi zajišťuje příkaz `ml` (*module load*). Tento příkaz bez parametrů vypíše aktuálně načtené moduly, a všechny moduly specifikované jako parametry se pokusí načíst. Příkaz `ml purge` je všechny odstraní.
Moduly je potřeba načíst po každém přihlášení nebo získání výpočetního uzlu (jsou implementovány proměnnými prostředí). V tomto projektu budou pro překlad (a spouštění) potřeba moduly `intel-compilers/2021.1.2` (kompilátor), `CMake/3.21.2-GCCcore-9.3.0` (překladový systém) a pro profilování modul `VTune/2019_update8 Advisor/2020_update3` (jak bylo již demonstrováno výše). **Jestliže jej potřebujete, modul profileru načtěte jako první!** Závisí na starších knihovnách, které mohou být přepsány novějšími po načtení kompilátoru. Pro profilování kvůli knihovnám musíte mít načtený i modul `intel-compilers/2021.1.2`, jinak se vaše přeložena aplikace nespustí.

B. Spouštění na počítačích CVT FIT

Projekt je možné spustit na počítačích CVT, jako jsou počítače v učebnách nebo server *merlin* — na něm ale nespustíte Intel Advisor nebo VTune, můžete na něm ale ladit funkčnost (Intel kompilátor je k dispozici). V učebnách s Linuxem (a Intel procesorem) je možné použít nástroje Intel Advisor a VTune napřímo.

```
$ source  /usr/local/share/intel/setvars.sh
:: initializing oneAPI environment ..
$ mkdir build
$ cd build/
$ CC=icc CXX=icpc cmake ..
-- The C compiler identification is Intel 20.2.4.20210910
-- The CXX compiler identification is Intel 20.2.4.20210910
-- ...
$ make
...
$ bash ../scripts/compare.sh
$ advixe-gui # pouze na pocitacich v ucebnach kvuli limitu RAM
$ vtune-gui  # take
```