Dobrý den, rád bych vás seznámil se svojí diplomovou prací a popsal dosažené výsledky.

NEXT

V rámci prezentace představím pojem datově orientované programování, popíšu cíle práce a vlastní přínos, jaké jsem použil technologie k vypracování a představím ukázku dosažených výsledků práce.

NEXT

Datově orientované programování je programovací paradigma, které vzniklo díky identifikování nedostatků objektově orientovaného návrhu v určitých oblastech. Jak vyplývá z názvu, považuje data za to nejpodstatnější. Staví na tvrzení, že primární úkol aplikací je transformace dat z jedné formy do druhé. Nesnaží se hardware zatěžovat abstrakcemi v podobě modelů, ale snaží se ho respektovat a využívat co nejefektivněji. Právě na využívané optimalizace se soustředí tato práce. Optimalizace se týkají zejména správného využití vyrovnávacích pamětí, nepletení se do cesty instrukční pipeline procesoru a snaží se vytěžit maximum z paralelního potenciálu hardware. Pojem datově orientované programování byl poprvé použit v článku pro herní vývojáře. Z této sféry toto paradigma patrně pochází.

NEXT

Prvním bodem zadání je seznámení se s tímto paradigmatem. V teoretické části představuju definice a hlavní myšlenky tohoto programovacího stylu. Mezi ně patří například myšlenka nechat si napovědět daty, se kterými program pracuje a lépe tak pochopit řešený problém. Čím více kontextu máme, tím lépe můžeme o problému uvažovat. Zároveň můžeme zaznamenávat data o datech, které mohou do značné míry ovlivnit architekturu aplikace.

NEXT

Dalšímu bodu se věnuju jak v teoretické, tak v praktické části. Popisuju rozdíly v hlavních myšlenkách těchto směrů a také ukazuju různé způsoby řešení stejného problému.

NEXT

Efektivní využití hardware je jedním z hlavních témat tohoto paradigmata. V teoretické části dopodrobna popisuju, jak fungují části hardware, jako například vyrovnávací paměť nebo instrukční pipeline.

NEXT

V praktické části na ukázkových příkladech demonstruju, jak psát kód, který lépe využívá plného potenciálu hardware. Také je zde využito informací popsaných v teoretické části, které nám umožní uvažovat o dosaženém výkonu těchto příkladů.

NEXT

Výkon ukázkových příkladů je zhodnocen pomocí nástrojů zaznamenávajících dobu vykonávání programu a také profilovacích nástrojů pro získání detailnějších informací o aplikaci.

NEXT

Na základě naměřených výsledků je sestavena sada doporučení pro programátory, kteří by chtěli aplikovat principy tohoto způsobu programování.

NEXT

Vlastní přínos spočíval ve stanovení a implementaci ukázkových příkladů, které demonstrují různé aspekty tvorby programů. Každý program má různé varianty. Jedna varianta vždy trpí nějakým nedostatkem a další varianty tento nedostatek odstraňují. Odstranění tohoto nedostatku je následně potvrzeno pomocí nástrojů pro měření, které sledují nějakou významnou metriku. Ta je ve většině případů celková doba běhu programu. Výsledky měření jsou prezentovány v tabulkách a následná analýza je zachycena v textové a obrázkové podobě a opodstatňují jednotlivá doporučení.

NEXT

Pro implementaci byl použit programovací jazyk C++. Na žádost vedoucího práce také došlo k porovnání třech překladačů, a to GCC, Clang a MSVC. Jazyk byl zvolen kvůli jeho rozšíření v oblasti časově kritických aplikací a díky svému záměru lze celkem snadno uvažovat o převodu kódu z tohoto jazyka do jazyka symbolických adres. Pro získání informací o časování byla zvolena knihovna Google Benchmark. Jedná se o sofistikovaný nástroj poskytující užitečnou funckionalitu pro měření výkonu programů pomocí různých ukazatelů a také umožňuje uživatelům si stanovit, co je pro ně důležité sledovat. Nabízí různé možnosti exportování výsledků měření, což usnadnilo práci při vytváření tabulek pro diplomovou práci. Pro hlubší zkoumání charakteristik aplikace bylo využito profileru vTune. Jedná se o bezplatný nástroj vyvinut společností Intel a při studiu témata profilování jsem pochopil, že tento nástroj udává standard v této oblasti. Poskytuje vizuální a empirická data o charakteristikách profilovaného programu a usnadňuje identifikaci limitujících míst ve zdrojovém kódu. Podpůrným profilerem je Tracy, což je bezplatný open source hybridní profiler. Jeho výhoda by měla být malé zatížení profilovaného programu a tím pádem získání přesnějších výsledků.

NEXT

Následně krátce představím jeden z ukázkových případů a jaký problém řeší. V objektově orientovaném návrhu jsou tvořeny třídy pro zaznamenání konceptů z reálného světa. V datové struktuře v obrázku vidíme 9 proměnných typu double. Je vytvořeno pole těchto prvků o počtu n a následně se na jednotlivých prvcích provádí nějaká operace.

NEXT

Kde je problém? Ve využití kapacity vyrovnávací paměti. Pokud jsou proměnné deklarovány v datové struktuře, bývají v operační paměti umístěny za sebou. Jestliže v tomto příkladě použijeme pouze 4 proměnné z 9, není je třeba načítat. Musíme si ale uvědomit, že při čtení dat z operační paměti nám vždy dorazí kus paměti o velikosti cache bloku, který je zpravidla 64 B velký.

NEXT

Jestliže proměnná datového typu double zabírá 8 B paměti, pak jeden prvek zabere 2 cache bloky a z těchto bloků vyčteme pouze 4 proměnné, a to a, b, c a result.

NEXT

Řešení tohoto problému je typická ukázka datově orientovaného přístupu. Spočívá ve stanovení, která data jsou odkazována častěji, než jiná. Díky tomu můžeme proměnné v datové struktuře rozdělit do dvou skupin. Jedna bývá označována jako hot, a druhá jako cold. Původní datová struktura nyní obsahuje dvě pole.

NEXT

Prostou změnou uspořádání dat v datové struktuře docílíme mnohem lepšího využití vyrovnávací paměti. Toto má za následek snížení počtu odkazů do paměti, které trvají z pohledu latence mnohem delší dobu, než odkázání do vyrovnávací paměti procesoru.

NEXT

Měření pomocí nástroje Google Benchmark nám ukáže vliv této optimalizace na dobu běhu programu. V závislosti na počtu prvků v poli je možné dosáhnout téměř trojnásobného zrychlení.

NEXT

Informace z aplikace vTune nám poskytnou dodatečný pohled na věc. Zde jsou zobrazeny údaje z jednotek monitorujících hardwarové události, konkrétně počet cache miss a počet zastavení instrukční pipeline kvůli cache miss. Cache miss znamená odkaz na data, která nebyla nalezena ve vyrovnávací paměti, nebo v dané úrovni vyrovnávací paměti. Při porovnání hodnot si můžeme všimnout, že příklad se separovanými daty tyto čísla zásadně redukuje.

NEXT

Ačkoliv se takové výkonové zlepšení může zdát nevýznamné, protože se bavíme o řádu milisekund, v závislosti na problémové oblasti se může jednat o zásadní výhru. Kupříkladu v oblasti videoher se to může projevit na nejvyšším dosažitelném počtu snímků za sekundu. Pokud bychom díky optimalizacím dokázali zdvojnásobit snímkovou frekvenci ze 30 na 60 snímků za sekundu, je to významný nárůst. Taková změna by mohla zapříčinit zvýšení prodeje dané hry, protože množství hráčů nepovažuje 30 snímků za sekundu jako použitelné. Dalším příkladem by bylo využití optimalizací pro dlouho trvající výpočty. Pomocí nich bychom mohli snížit celkovou dobu výpočtu z jednotek hodin na desítky minut, což by zapříčinilo lepší produktivitu. Rovněž mají optimalizované výpočty potenciál spořit energii, jelikož díky snížení počtu odkazů do operační paměti se nespotřebuje stejné množství energie, jako při odkazování do vyrovnávací paměti. Výkonového zlepšení bude dosaženo zejména tehdy, pokud budou optimalizace aplikovány tam, kde skutečně vyřeší nějaké omezení. Pokud například program tráví 80 % času v nějaké smyčce, je vhodné zaměřit optimalizace právě na toto místo, a ne na zbylých 20 %.