MAGAZÍN
KATEDRY INFORMATIKY

číslo 7 | červen 2017 Univerzita Palackého v Olomouci

Úvodní slovo

Vážení čtenáři,

dostalo se k Vám již sedmé číslo Magazínu katedry informatiky, ve kterém bychom se i tentokrát s Vámi chtěli podělit o zajímavosti nejen z naší katedry, ale také ze světa informatiky obecně.

Nejdříve si připomeneme jubileum profesora Chajdy, který zásadním způsobem ovlivnil vznik a směrování olomoucké informatiky. Řada našich absolventů a studentů jej měla možnost poznat hlavně z pedagogické stránky, profesor Bělohlávek ve svém článku nabízí osobní, neméně zajímavý, pohled na tuto mimořádnou osobnost naší univerzity.

Většina z Vás se již jistě setkala s pojmem složitost algoritmu; pojem komunikační složitost ale tak známý není. Pokud Vás zajímá, co se pod tímto názvem skrývá, článek Petra Osičky Vám poskytne úvod do problematiky. Dále pro Vás máme stručné představení procesorů rodiny ARM, na které, zdá se, můžete narazit téměř na každém kroku, i když si to možná ani neuvědomujete.

Závěrem se věnujeme prvním držitelům Klirova stipendia, které naše katedra uděluje studentům za vynikající studijní výsledky.

Petr Krajča

redaktor Magazínu Katedry informatiky Univerzity Palackého v Olomouci

Obsah

Profesor Ivan Chajda

Radim Bělohlávek

V prosinci minulého roku oslavil neuvěřitelné sedmdesáté narozeniny Ivan Chajda, profesor katedry algebry a geometrie naší fakulty. Neuvěřitelné proto, že by mu sedmdesát let nikdo nehádal. Rád využívám příležitosti, kterou jeho nedávné kulaté narozeniny poskytují, abych pana profesora představil čtenářům Magazínu katedry informatiky. Toto představení má být také výrazem uznání, a sice uznání významu profesora Chajdy pro rozvoj informatických oborů na Univerzitě Palackého.

Prof. RNDr. Ivan Chajda, DrSc., se narodil 13. 12. 1946 v Přerově. Tam také prožil své dětství. Po absolvování gymnázia byl přijat na Přírodovědeckou fakultu Univerzity Palackého v Olomouci, kterou absolvoval v roce 1970. Vystudoval obor numerická matematika, který byl předchůdcem později zavedených informatických oborů. V roce 1975 získal hodnost CSc. (kandidát věd), v roce 1986 pak hodnost DrSc. (doktor věd), obě v oboru algebra a teorie čísel na Masarykově univerzitě v Brně. Od roku 1989 je profesorem v oboru algebra a teorie čísel. Je autorem 7 monografií a více než 460 prací publikovaných v recenzovaných mezinárodních časopisech. Pracuje především v oblasti abstraktní algebry, teorie svazů a souvisejících disciplín. Profesor Chajda vychoval mnoho studentů, někteří z nich jsou dnes docenty nebo profesory.

Profesor Chajda je výrazná, charismatická osobnost, která zásadním způsobem ovlivnila rozvoj matematických a informatických oborů na Univerzitě Palackého po přelomovém roce 1989. Na přírodovědeckou fakultu naší univerzity nastoupil krátce před rokem 1989, a to na základě pozvání tehdejšího vedoucího katedry algebry a geometrie a jejího pozdějšího dlouholetého vedoucího profesora Jiřího Rachůnka. Okolnosti tehdy a dnes byly v mnohém jiné. Pokud jde o situaci na poli vědy, o kterou mi v souvislosti s profesorem Chajdou jde, udála se velká změna. Dnes je zcela běžné, že olomoučtí matematici publikují v předních světových časopisech a že jejich práce jsou citovány. Tenkrát to byla výjimka. Kdybych měl jmenovat jednoho člověka, který se o tuto změnu zasloužil nejvíc, jmenoval bych právě profesora Chajdu. Po příchodu na univerzitu byl jediným matematikem s velkým doktorátem (DrSc.) a byl jedním z mála takových na celé univerzitě. Jeho seznam publikací byl mnohem delší, než měli ostatní, měl zkušenosti ze zahraničních konferencí a brzy se stal profesorem. Měl vizi a energii, jak tuto vizi uskutečňovat. Zdůrazňoval, že je to běh na dlouhou trať a systematicky, vytrvale, krůček po krůčku, postupoval. Výsledek se začal dostavovat a profesor Chajda je, alespoň myslím, v zásadě spokojený. Musím dodat „v zásadě", protože spokojený on není nikdy: „nesmíme usnout na vavřínech", jako bych teď slyšel.

Profesor Chajda ale významně ovlivnil i rozvoj informatiky jako oboru na naší univerzitě. Jako absolvent oboru, který byl předchůdcem dnešních informatických oborů, nastoupil po absolvování základní vojenské služby do přerovské Meopty, kde pracoval jako programátor a analytik. Napůl informatikem (a napůl, samozřejmě, matematikem) se cítí dodnes, aspoň mi to tak občas říká, a já to vnímám jako pocit zcela autentický. Je to člověk, který si prožil, co profese informatika obnáší. Ví například, že programovat neznamená jen znát programovací jazyk, ví, že profese programátora je tvořivá a že vyžaduje podobné schopnosti abstraktního a exaktního myšlení jako matematika.

Význam profesora Chajdy pro rozvoj našich informatických oborů vidím ve dvou rovinách. První, zcela konkrétní, představuje jeho působení jako pedagoga v infor-

matických studijních oborech. Druhou, poněkud hůře pojmenovatelnou, představuje jeho osobní příklad a mimopedagogické působení.

Pokud jde o působení Ivana Chajdy jako pedagoga v informatických oborech, dobře ho myslím charakterizují moje vzpomínky z roku 1989 a z let následujících, kdy jsem informatiku studoval. Jako studenti jsme jeho přednášky z algebry hltali. Byly náročné, kvalitní a měly šmrnc. Přednášel tehdy, i později, různé partie z abstraktní algebry, například teorii relací, uspořádaných množin a svazů, Booleových algeber, grup a okruhů. Tyto předměty utvářely schopnost exaktního myšlení a schopnost správné systémové analýzy absolventů olomoucké informatiky. Můj spolužák a dnes úspěšný informatik Stanislav Opichal mi jednou řekl: „Nejvíc mi během studia daly Chajdovy přednášky z algebry." To dnes říkám studentům, když se ptají, k čemu se učí teorii. V druhé etapě studia přednášel profesor Chajda dva důležité předměty. Jeden byl o teorii systémů, byl protkaný mnohými zajímavými příklady z praxe, a byl tak mezi studenty značně oblíbený. Druhý byl o univerzální algebře a přímo se do něho promítala vědecká činnost profesora Chajdy. Musím také zmínit semináře, které profesor Chajda pro zájemce z řad studentů vedl. Nebyly součástí výuky, prostě jsme se scházeli podle dohody a probírali různá zajímavá témata. Jedním z prvních byla tehdy poměrně nová metoda analýzy dat a reprezentace znalostí, tzv. formální konceptuální analýza. Dodnes slyším Ivanova slova o tom, že to je „jediný způsob, jak dostat inteligenci do počítače". Tato slova se zařadila mezi „Chajdovy kultovní hlášky" a později, když jsme sami začali brát i jiné rozumy, jsme se nad nimi dobromyslně usmívali. Zcela jednoznačně to byl ale tento vynikající seminář, který byl onou jiskrou, díky níž jsme se o konceptuální analýzu zajímali. Dnes tvoří formální konceptuální analýza a další metody analýzy relačních dat jeden z pilířů výzkumu na katedře informatiky.

Pokud jde o výše zmíněný druhý faktor, je dán tím, že na katedře informatiky působí několik lidí, které profesor Chajda ovlivnil, ať už přímo jako své žáky nebo nepřímo jako kolegy na fakultě. Ovlivnil je svým osobním příkladem a přístupem k životu. Vážíme si zejména jeho houževnatosti, vytrvalosti a systematičnosti, s jakými se věnuje všemu, co dělá, a inspiruje nás to. Z dnešního pohledu je například téměř neuvěřitelné a obdivuhodné, že profesor Chajda vlastně algebru pěstoval jako koníčka. Kandidátem věd (CSc.) a později i doktorem věd (DrSc.) se stal v období, kdy pracoval na plný úvazek v Meoptě. V tomto období také napsal kolem stovky prací z algebry, které byly uveřejněny v mezinárodních časopisech, každý týden jezdil na brněnské algebraické semináře profesora Novotného a jezdil na konference, na které si bral dovolenou. Podotýkám, že doktorem věd se stal v 39 letech, tedy ve výjimečně nízkém věku. Pro vědeckou práci tedy neměl komfort tehdejších, natož pak dnešních akademiků. Velmi mladý, ve 42 letech, se pak také stal profesorem, ale to už jako pedagog Univerzity Palackého. Jeho houževnatost a vytrvalost se projevuje i v jeho další životní vášni, totiž sportu. Dlouhé roky se věnoval cyklistice – sám závodil a později působil jako trenér mládeže. Později se cyklistice a s ní i běhu věnoval pro vlastní potěšení a tak je tomu dodnes. Bylo by ale chybou představit si, že si občas vyjede na kole nebo se občas proběhne. Dělá to s přístupem jemu vlastním: trénuje pravidelně a hodně a tréninky si přesně eviduje. Že to není žádné sportování „na pohodu" dokládá skutečnost, že běžel několik maratónů a všechny ve vynikajících časech kolem 2 hodin a 45 minut. Poslední prosincový den minulého roku se jako čerstvý sedmdesátník zúčastnil tradičního Silvestrovského běhu v Přerově. Trať 7,5 km uběhl při teplotě -7 stupňů v čase 30 minut a 52 sekund a ve výsledkové listině se před jeho datem narození 1946 nachází jen sedm účastníků, všichni narození mezi roky 1972 a 1980. Za posledních 10 let ujel na kole přes 250 000 kilometrů.

Kromě výše zmíněných vlastností musím uvést i otevřenost a přímočarost, s jakou se svými studenty a kolegy pan profesor vždy jednal. Jde ruku v ruce s jeho schopností udělat si legraci sám ze sebe, označit některé své publikace za „blbosti" a vůbec posuzovat akademický život realisticky a střízlivě, v kontextu jiných lidských aktivit. Stěžovat si na malou podporu vědy ze strany státu, natahovat ruku po dalších a dalších dotacích, to jsem u profesora Chajdy nikdy neviděl. Chtěl

bych taky zmínit jeho starostlivost a příkladnou péči o své studenty. Jako studenta pátého ročníku mě vzal na konferenci do Německa a celou cestu i účast mi z toho, z dnešního pohledu neskutečně skromného, balíčku peněz na vědu, které tenkrát měl, celou zaplatil. A stejně to dělal s ostatními svými studenty. Že se musíme rozhlížet kolem, nápady rozpracovávat, i když se zdá, že to nejde, výsledky pak publikovat a dobře prezentovat, tj. krátce řečeno „jak dělat vědu", jsme se taky učili od profesora Chajdy.

Před deseti lety jsem při příležitosti šedesátin profesora Chajdy psal podobný článek. (Musím přiznat, že psát po deseti letech tento článek bylo mnohem obtížnější. Nic zásadně nového jsem nenapsal, ale tak to je asi vždy, když píšete o člověku, který zůstává stejný.) V něm jsem nabídl vsadit se o deset tisíc korun, že i v sedmdesáti bude běhat deset kilometrů pod čtyřicet minut, a dvacet tisíc korun, že i v sedmdesáti bude psát práce, které inspirují ostatní. Obě sázky bych myslím vyhrál, tu druhou jednoznačně. Profesor Chajda zůstává člověkem, který inspiruje a pozvedává ostatní. Přeji mu, ať mu to ještě dlouhé roky vydrží!

Komunikační složitost

Petr Osička

Komunikační složitost je oblastí teorie výpočetní složitosti, která se zabývá následujícím scénářem a scénáři mu podobnými: dvě strany, říkejme jim Alice a Bob, spolu komunikují, aby tak vyřešily instanci nějakého algoritmického problému. Na začátku mají Alice i Bob každý pouze část vstupu, a tak k vyřešení problému spolu komunikovat musí. Zajímá nás, jaké množství komunikace (měřeno počtem zaslaných bitů) je k výpočtu potřeba. Podobně jako v ostatních partiích složitosti chceme znát omezení množství komunikace seshora, tedy množství komunikace v nejlepším (známém) algoritmu, a omezení zespoda, tedy množství komunikace, které je potřeba v každém algoritmu – komunikační složitost daného problému. Komunikační složitost je specifická v tom, že při hledání omezení zespoda byli informatici neobvykle úspěšní, mnohem úspěšnější než v případě časové a prostorové složitosti problémů.

Hledání mediánu

V problému nalezení mediánu mají Alice a Bob každý libovolnou sekvenci přirozených čísel menších než n. Cílem je nalézt medián sekvence, kterou obdržíme jejich spojením. Připomeňme, že pokud má sekvence k prvků je mediánem $\lceil \frac{k}{2} \rceil$-tý nejmenší prvek (pomocí $\lceil \cdot \rceil$ značíme zaokrouhlení na celá čísla nahoru). Triviální algoritmus, ve kterém Alice pošle všechna čísla ze své sekvence Bobovi, který je přidá do své sekvence, najde medián a pošle jej zpátky Alici, pro tuto chvíli zavrhneme jako nevyhovující, protože je při něm potřeba příliš komunikace. Komunikační složitost by v tomto případě byla $O(\log(n) \cdot m)$ bitů, kde m je počet čísel v Alicině sekvenci. Čísla, která jsou menší než n, lze totiž zakódovat pomocí $\lceil \log n \rceil$ bitů.

Intuitivně tušíme, že pro zajištění lepší komunikační složitosti je potřeba výpočet lépe rozložit mezi Boba a Alici tak, aby Alice nemusela Bobovi posílat všechna čísla ze své sekvence. Docílit toho můžeme například tak, že necháme Alici i Boba společně provádět algoritmus půlení intervalů. Připomeňme, že v klasickém algoritmu půlení intervalů si pamatujeme interval, označme jej $[i, j]$, ve kterém se medián nachází. V každé iteraci algoritmu pak spočítáme střed k tohoto intervalu a porovnáme počet prvků, které jsou menší než k, s počtem prvků, které jsou větší než k. Na základě toho se potom můžeme rozhodnout, jestli dále pokračovat s intervalem $[i, k]$ nebo $[k, j]$, nebo jestli je už k hledaným mediánem. Pokud Alice i Bob spustí algoritmus půlení intervalů se stejnou sekvencí, mohou si číslo k spočítat každý sám. Protože však ani jeden z nich nemá celou sekvenci, jejíž medián mají spočítat, musí počet čísel menších než a počet čísel větších než k spočítat společně.

První algoritmus pro nalezení mediánu

1. Alice i Bob si zapamatují interval $[i, j]$, ve kterém se medián může nacházet. Na začátku algoritmu tento interval inicializujeme na $[1, n]$. (Ve skutečnosti stačí interval $[i, j]$, kde i je minimální a j je maximální číslo v obou sekvencích. Alice i Bob mohou najít maximum a minimum své sekvence a tato čísla si vzájemně poslat. Stačí k tomu $O(\log n)$ bitů.)

2. Dokud interval $[i, j]$ obsahuje alespoň dvě čísla (tj. dokud se i nerovná j), Alice a Bob opakují následující interakci:

   (a) Alice spočítá číslo $k = \lfloor \frac{i+j}{2} \rfloor$, dále počet prvků vlastní sekvence menších nebo rovných k, označme jej l_A, a počet čísel větších než k, označme jej g_A. Čísla l_A a g_A pošle Bobovi.

(b) Bob podobně jako Alice spočítá k a počty menších a větších prvků, označme je l_B a g_B. Protože od Alice dostal l_A a g_A, snadno spočítá kolik čísel z obou sekvencí je menších nebo rovno k, je to $l_A + l_B$, a kolik je větších než k, je to $g_A + g_B$. Podle toho dokáže určit, jestli pokračovat s intervalem $[i, k]$ nebo $[k, j]$, tj. pokud je víc čísel mensich rovno k, výpočet pokračuje s $[i, k]$, jinak s $[k, j]$. Tuto informaci pošle Alici.

3. V případě, že $i = j$, je mediánem číslo i.

Pro konkrétní sekvence čísel můžeme zachytit interakci mezi Alicí a Bobem pomocí tabulky. Řekněme tedy, že Alice má sekvenci $1, 1, 2, 2, 3$ a Bob sekvenci $2, 4, 7, 7$. Tabulka pak vypadá následovně.

interval	k	(l_A, g_A)	Bob $\rightarrow$ Alice
$[1, 7]$	4	$(5, 0)$	$[1, 4]$
$[1, 4]$	2	$(4, 1)$	$[1, 2]$
$[1, 2]$	1	$(1, 4)$	$[2, 2]$

Každý řádek tabulky odpovídá jednomu kolu interakce. Na prvním řádku můžeme vidět, že v Alicině sekvenci se nachází pět čísel menších než nebo rovných k a žádné číslo větší než k. Tuto informaci Alice pošle Bobovi. V Bobově sekvenci jsou dvě čísla menší rovna k a dvě čísla větší než k. Dohromady je to tedy sedm čísel menších nebo rovných k a dvě čísla větší než k. Bob potom pošle Alici informatici o tom, že příštím kole bude výpočet pokračovat s intervalem $[1, 4]$, jak vidíme v posledním sloupci tabulky.

Jaká je komunikační složitost tohoto algoritmu? Tedy kolik bitů komunikace v tomto algoritmu potřebujeme? Tento počet vyjádříme jako funkci závislou na n a na m, kde m je počet čísel v sekvenci Alice. Snadno vidíme, že algoritmus proběhne v $O(\log(n))$ kolech, protože v každém z nich prohledávaný interval půlíme. V jednom kole pošle Alice Bobovi dvě čísla, jejichž velikost je omezena m, to lze provést pomocí $O(\log(m))$ bitů. Informace, kterou zasílá Bob Alici se dá zachytit pomocí jednoho bitu (například 0 znamená dolní interval, 1 znamená horní interval). Celkem je tedy komunikační složitost algoritmu rovna $O(\log(m) \cdot \log(n))$.

Pro problém nalezení mediánu existuje, možná trochu překvapivě, algoritmus s komunikační složitostí $O(\log(n) + \log(m))$. Pro jednoduchost budeme předpokládat, že počet čísel v Alicině i Bobově sekvenci je stejnou mocninou dvou. Tento předpoklad není příliš silný, protože v opačném případě lze sekvence doplnit potřebným počtem čísel 1 a n (všímavý čtenář si jistě všimne, že to lze provést s komunikační složitostí $O(\log(m))$). V průběhu algoritmu si budou Alice a Bob udržovat sekvenci čísel, jedno z nichž je mediánem. Na začátku tyto sekvence nastavíme na vstupní sekvence. Poté opakujeme následující interakci.

Druhý algoritmus pro nalezení mediánu

1. Alice spočítá medián ze své sekvence, označme jej a, a pošle jej Bobovi. Bob spočítá medián ze své sekvence, označme jej b, a pošle jej Alici.

2. Na základě hodnot a a b proběhne následující:

   (a) Pokud $a = b$, pak jsme nalezli medián (Alice i Bob to ví).

   (b) Pokud $a < b$, Alice zredukuje svoji sekvenci na polovinu opakovaným mazáním nejmenšího čísla, Bob zredukuje svoji sekvenci na polovinu opakovaným mazáním největšího čísla.

   (c) Pokud $b < a$, Alice a Bob provedou to, co v předchozím bodu, pouze v opačném gardu.

3. Pokud už obsahují sekvence Alice a Boba každá pouze jedno číslo, je výpočet u konce. Mediánem je menší z obou čísel.

Pro případ, kdy má Alice sekvenci $1, 1, 2, 2$ a Bob sekvenci $3, 4, 7, 7$, zachycuje potřebnou komunikaci následující tabulka.

Alice	Bob	a	b
$1, 1, 2, 2$	$3, 4, 7, 7$	1	4
$2, 2$	$3, 4$	2	3
2	3		

Protože se počet čísel v sekvencích Alice i Boba v každém kole zmenší na polovinu, proběhne $O(\log(m))$ kol. V každém kole pošlou Alice a Bob dohromady dvě čísla, každé rovno maximálně n. To lze provést s použitím $O(\log(n))$ bitů. Komunikační složitost je tedy

$O(\log(m)\log(n))$. Abychom se dostali ke slíbené složitosti, změníme bod 1 algoritmu. Alice a Bob si nemusí vyměnit čísla a a b, stačí vědet, které nich je větší (nebo zda-li se rovnají). Porovnání a a b lze provést po jednotlivých bitech, počínaje od nejvýznamnějšího (přičemž uvažujeme přímou binární reprezentaci čísel). Které z čísel je menší poznáme podle prvního bitu, ve kterém se liší. Dále si všimneme, že po zkrácení sekvencí jsou všechny jejich prvky z intervalu $[a, b]$, a proto se ve všech bitech vyšších než nejvyšší bit, ve kterém se a a b liší, rovnají. V dalších kolech tedy stačí srovnat nové mediány pouze na nižších bitech. Pro porovnání čísel a, b na jednom bitu si Alice a Bob musí vyměnit dva bity (jeden bit pošle Alice Bobovi, jeden bit Bob Alici). Kolik takových porovnání může během interakce proběhnout? Může se stát, že se čísla a a b vždy liší hned na prvním bitu, pak určitě proběhne $O(\log(m))$ kol, protože Alice i Bob po každé neshodě zkrátí své sekvence na polovinu. Na druhou stranu, a a b se mohou lišit až na nízkém bitu a může tedy být nutné téměř všechny jejich bity porovnat (příkladem, kdy se tak stane, je sekvence $1, 1, 1, 300$ pro Alici a $2, 2, 2, 2$ pro Boba). To vede k $O(\log(n))$ poslaným bitům.

Pro případ, kdy má Alice sekvenci $1, 1, 2, 2$ a Bob sekvenci $3, 4, 7, 7$, zachycuje potřebnou komunikaci následující tabulka (aktuálně porovnávaný bit, a tedy posílaný bit, je podtržen).

Alice	Bob	a	b
$1, 2, 2, 3$	$3, 4, 7, 7$	$2 = \underline{0}10$	$4 = \underline{1}00$
$2, 3$	$3, 4$	$2 = \underline{0}10$	$3 = \underline{0}11$
$2, 3$	$3, 4$	$2 = 0\underline{1}0$	$3 = 0\underline{1}1$
$2, 3$	$3, 4$	$2 = 01\underline{0}$	$3 = 01\underline{1}$
2	3		

Dolní omezení

V předchozí kapitole jsme na příkladu ukázali, jakým typem problémů a algoritmů se komunikační složitost zabývá, u každého algoritmu jsme určili jeho komunikační složitost. Hledali jsme horní omezení složitosti problému nalezení mediánu. V této kapitole se budeme zabývat hledáním dolních omezení.

Pro jednoduchost se budeme zaobírat pouze rozhodovacími problémy. Formálně je takový problém určen funkcí f zobrazující dva řetězce bitů (reprezentující vstup pro Alici a vstup pro Boba) na jeden bit (odpověď ano nebo ne). Podobně jako například u časové složitosti není omezení se na rozhodovací problémy nijak zásadní, problém jehož výsledkem je n-bitový objekt si totiž můžeme představit jako n problémů definovaných pomocí Booleovských funkcí, pro každý bit výsledku jeden. Vstupní řetězec, který má na začátku Alice budeme značit pomocí x, Bobův vstupní řetězec pomocí y (s případnými indexy). Mezi nejjednodušší rozhodovací problémy patří funkce $\mathrm{EQ}(x, y)$, která je rovna 1, právě když $x = y$; a funkce funkce $\mathrm{DIST}(x, y)$, která je rovna 1, právě když neexistuje bit, na kterém mají x i y najednou jedničku. U obou problémů předpokládáme, že x a y jsou stejně dlouhé, jejich délku budeme značit n. Intuitivně bychom řekli, že nalézt dolní omezení komunikační složitosti je pro oba problémy snadné. Například bychom řekli, že protože u problému EQ musíme porovnat x a y na všech bitech, je dolní omezení n. V tomto případě se naše intuice nemýlí. Nicméně i jednoduchá tvrzení je potřeba dokázat a mnohdy, i v našem případě, vedou formální úvahy o jednoduchých problémech k nalezení triku, který je aplikovatelný i ve složitějších případech.

V přechozím odstavci jsme si ujasnili, jakým typem problémů se budeme zabývat. Teď se podívejme na to, jak v naší situaci formalizovat pojem algoritmus. Formalizujeme ho pomocí pojmu *komunikační protokol,* což je soubor pravidel, který v závislosti na doposud proběhlé komunikaci a x a y:

- jednoznačně určí, jestli výpočet končí. Pokud ano, tak určí výsledek výpočtu.
- V případě, že výpočet nekončí, specifikuje zprávu, kterou pošle Alice Bobovi (nebo opačně, Alice a Bob se v posílání zpráv pravidelně střídají), obsah této zprávy závisí pouze na x (nebo y, v případě, že je odesilatelem Bob) a doposud proběhlé komunikaci.

Zmíněná pravidla ve skutečnosti formalizujeme pomocí funkcí. Řekněme, že Alice a Bob si doposud vyměnili řetězy $a_1, a_2, \ldots, a_{i-1}$, a že další zprávu bude posílat Alice. Zprávu, kterou Alice pošle, dostaneme aplikací funkce f_i na argumenty $a_1, a_2, \ldots, a_{i-1}$ a x. Obecně probíhá výpočet algoritmu následovně: Alice vypočítá

$a_1 = f_1(x)$ a pošle a_1 Bobovi, poté Bob spočítá $a_2 = f_2(a_1, y)$ a pošle a_2 Alici, Alice spočítá $a_3 = f_3(a_1, a_2, x)$ a pošle a_3 Bobovi atd. Na funkce f_i neklademe žádné další požadavky, mohou to být i nespočetné funkce. To znamená, že předpokládáme, že Alice i Bob mají neomezenou výpočetní sílu. Zajímá nás opravdu jenom počet bitů, které si Alice s Bobem pošlou.

Komunikační složitost protokolu P je nejvyšší počet bitů, které si během provádění protokolu Alice a Bob vymění, uvažujeme-li vstupy délky n (složitost vyjádříme jako funkci závislou na n). Komunikační složitostí funkce f, kterou označíme jako $D(f)$, je potom komunikační složitost nejlepšího protokolu, který tuto funkci počítá.

Při hledání dolního omezení komunikační složitosti funkce f můžeme využít kombinatorických vlastností, které jsou společné všem protokolům pro tuto funkci. Zafixujme číslo n a představme si tabulku, ve které řádky odpovídají částem vstupu, které má na začátku Alice, a sloupce částem vstupu, které má na začátku Bob. Ve vlastní tabulce je pak na místě odpovídajícím x a y hodnota $f(x, y)$. Pro $n = 2$ může taková tabulka vypadat následovně:

x/y	00	01	10	11
00	0	0	0	1
01	0	0	0	1
10	0	0	0	0
11	0	1	1	1

Z tabulky můžeme vyčíst například to, že $f(00, 00) = 0$ nebo $f(11, 01) = 1$.

V tabulce můžeme nalézt několik obdélníků. Jako obdélník bereme kartézský součin $A \times B$, kde A je množina řádků a B je množina sloupců. Řekneme, že obdélník je monochromatický, pokud jsou v tabulce všechny hodnoty na místech odpovídajících tomuto obdélníku stejné. V následující tabulce je červeně zvýrazněn monochromatický obdélník daný řádky $00, 01$ a sloupci $00, 01, 10$.

x/y	00	01	10	11
00	0	0	0	1
01	0	0	0	1
10	0	0	0	0
11	0	1	1	1

Ačkoliv jsou řádky a sloupce obdélníku z přechozího příkladu vedle sebe, obecně to tak být nemusí.

Vztah mezi obdélníky a komunikačními protokoly zachycuje následující tvrzení, jehož důkaz může čtenář najít v materiálech, na které odkazujeme v referencích.

Tvrzení 1 *Množina všech dvojic (x, y), pro které si Alice a Bob vymění během provádění protokolu stejnou posloupnost zpráv, tvoří monochromatický obdélník.*

Zprávy, které si Alice a Bob posílají, tedy můžeme vnímat jako jakési zakódování toho, do kterého monochromatického obdélníku výsledek padne. Jak dlouhé takové zakódování musí být? Odpovědí na tuto otázku je, že tak dlouhé, abychom mohli rozlišit množinu monochromatických obdélníků obsažených v tabulce takovou, že každé místo v tabulce patří právě do jednoho obdélníku z této množiny. Takové množině říkáme *rozklad tabulky na monochromatické obdélníky*. Tabulku z předchozího příkladu můžeme rozložit například následovně (obdélníky odpovídají místům vyznačeným stejnou barvou).

x/y	00	01	10	11
00	0	0	0	1
01	0	0	0	1
10	0	0	0	0
11	0	1	1	1

Označme pomocí $C_f(n)$ minimální počet obdélníků, pomocí kterých lze tabulku odpovídající funkci f rozložit. Pak stačí obdélníky očíslovat, a posloupnost zpráv mezi Alicí a Bobem musí jednoznačně určit číslo daného obdélníku. Tedy platí následující tvrzení.

Tvrzení 2 *Pro každou funkci f platí, že $D(f) \geq \log_2 C_f(n)$.*

Pokud pro konkrétní funkci f dokážeme určit $C_f(n)$ nebo dolní odhad $C_f(n)$, pak jsme podle předchozího tvrzení nalezli i dolní omezení komunikační složitosti funkce f. Jedním ze způsobů, jak omezit $C_f(n)$ zespoda je nalezení množiny dvojic (x_i, y_i) pro $i = 1, 2, \ldots, k$ takových, že pro všechna i, j dvojice (x_i, y_i) a (x_j, y_j) nepatří do stejného monochromatického obdélníku (připomeňme, že x_i je vždy částí vstupu, kterou má Alice, a tedy řádkem tabulky, a y_i je částí vstupu, kterou má Bob, a tedy sloupcem tabulky). Potom je

jistě $C_f(n)$ rovno alespoň k. Množinou s požadovanou vlastností je tzv. *klamavá množina*. O množině $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\}$ řekneme, že je klamavá, pokud existuje $z \in \{0, 1\}$, pro které platí:

1. Pro každé $(x_i, y_i) \in S$ máme $f(x_i, y_i) = z$.
2. Pro všechny různé $(x_i, y_i), (x_j, y_j) \in S$ buď $f(x_i, y_j) \neq z$ nebo $f(x_j, y_i) \neq z$.

O tom, že prvky množiny S skutečně nemohou ležet ve stejném obdélníku se můžeme přesvědčit, když se podíváme na následující tabulku.

x/y	$\ldots$	y_i	$\ldots$	y_j	$\ldots$
$\vdots$					
x_j		?		z	
$\vdots$					
x_i		z		?	
$\vdots$					

Vidíme, že pokud mají být (x_i, y_i) a (x_j, y_j) ve stejném monochromatickém obdélníku, musí být na místech označených symbolem '?' hodnota z. Z bodu 2 definice klamavé množiny ale víme, že alespoň na jednom takovém místě se hodnota z nevyskytuje.

Pokud si představíme, jak vypadá tabulka funkce EQ, všimneme si, že v každém řádku a každém sloupci má právě jednu jedničku (pokud jsou řádky a sloupce v tabulce stejně uspořádány, leží jedničky na diagonále). Pro $n = 2$ vypadá tabulka následovně.

x/y	00	01	10	11
00	1	0	0	0
01	0	1	0	0
10	0	0	1	0
11	0	0	0	1

Klamavou množinu tvoří všechny dvojice (x, y) takové, že $f(x, y) = 1$. Je jich 2^n a tedy $C_{\mathsf{EQ}}(n) \geq 2^n$, odkud podle Tvrzení 2 plyne $D(\mathsf{EQ}) \geq n$.

Pro funkci DIST můžeme provést podobnou úvahu. Jako klamavou množinu vezmeme množinu $S = \{(x, \overline{x}) \mid x \text{ je řetězec délky } n\}$, kde $\overline{x}$ je řetězec, který dostaneme nahrazením všech jedniček v x nulami, a všech nul v x jedničkami. Skutečnost, že S je klamavá množina, pak plyne z toho, že pokud $x_1 \neq x_2$, tak mají buď x_1 a $\overline{x}_2$ nebo $\overline{x}_1$ a x_2 společnou jedničku. V opačném případě by totiž platilo, že $x_1 = x_2$ (toto tvrzení si čtenář snadno ověří). Zjevně také platí, že pro každé $(x, y) \in S$ dostaneme $f(x, y) = 1$. Protože S obsahuje 2^n prvků, můžeme podobně jako u funkce EQ usoudit, že $D(\mathsf{DIST}) \geq n$.

Aplikace

Komunikační složitost má řadu aplikací. Návrh protokolů pro konktrétní problémy má uplatnění zejména v oblasti distribuovaných a paralelních výpočtů. Výsledky týkající se dolních omezení komunikační složitosti mají uplatnění i v teoretické informatice. Zběžně uvedeme dva příklady. Pomocí komunikační složitosti lze nalézt dolní omezení počtu stavů, které musí konečný automat mít, aby mohl přijímat nějaký konkrétní jazyk. Například pro jazyk

$$L = \{xy \mid |x| = |y| = n, x \neq y\}$$

musí mít automat, který jej přijímá, alespoň 2^n stavů (použijeme dolní omezení pro funkci EQ, podrobnosti lze nalézt v literatuře uvedené v referencích). Druhou zajímavou aplikací je tvrzení o vztahu mezi časovou a paměťovou složitostí algoritmu (modelovaného pomocí Turinova stroje), který rozpoznává, zda-li je daný řetezec palindrom.

Tvrzení 3 *Pokud má algoritmus rozpoznávající palindromy časovou složitost $T(n)$ a pamětovou složitost $S(n)$, pak platí $T(n) \cdot S(n) = \Omega(n^2)$.*

Použití komunikační složitosti pro získání předchozích výsledků je zajímavé zejména proto, že definice výpočetních problémů, kterými se výsledky zabývají, vůbec žádnou komunikaci explicitně nezmiňují. Přesto se zdá, že potřeba komunikace zespodu omezuje výpočetní složitost daných problémů.

Reference

(1) E. Kushilevitz, N. Nisan. Communication Complexity. Cambridge University Press, 1997.

(2) E. Kushilevitz. Communication Complexity. Dostupné `https://www.cs.bris.ac.uk/probtcs08/materials/kushilevitz.pdf`.

ARM: tak trochu jiné procesory

Petr Krajča

Při pohledu na osobní počítače posledních dvaceti let by se mohlo zdát, že postavení platformy Wintel (operačního systému Windows a procesorů rodiny Intel) je neotřesitelné. Nedávná statistika StatCounter, ale ukazuje, že všechno je trochu jinak. Podle ní již 37,93 procent zařízení přistupujících k webovým stránkám používá operační systém Android (v případě Windows je to 37,91 %). Tato statistika naznačuje ještě jednu zajímavou věc—čím dál tím větší zastoupení procesorů rodiny ARM. Dá se totiž předpokládat, že většina oněch zařízení s Androidem běží právě na architektuře ARM. Proto si ji v tomto příspěvku stručně představíme a ukážeme si některé zajímavé rysy třicetidvoubitových ARMů.

Zatímco společnosti Intel a AMD mezi sebou již řadu let vedou tu více, tu méně úspěšný souboj kdo představí lepší a výkonnější procesory, společnost ARM se se svými procesory se vydala jinou cestou a její procesory se dnes dají najít v celé řadě zařízení. Od řídících systémů, přes spotřební elektroniku, notebooky až po servery. I když srovnání společnosti ARM Holdings, která v současné době stojí za architekturou procesorů ARM, s Intelem nebo AMD není úplně korektní. Společnost ARM Holdings totiž ve skutečnosti procesory vůbec nevyrábí, a ani neprodává.

Hlavním artiklem této společnosti je licencování návrhů procesorů dalším výrobcům, kteří si doplní další moduly a nechají si na zakázku vyrobit vlastní fyzické procesory. Připojením například grafického čipu, modulů pro bezdrátovou komunikaci, atp. vznikají úplné a ucelené počítačové systémy, někdy též označované jako SoC (system-on-chip), které jsou schopné bez velkého množství dalších podpůrných obvodů obstarat funkcionalitu celého počítače. V tomto duchu si nechává například Apple nebo Samsung vyrábět procesory na míru pro své mobilní telefony či tablety. Společnosti, které nemají dostatečný kapitál na výrobu svých vlastních procesorů, mohou sáhnout po již hotových řešeních například od Nvidia nebo Qualcomm.

Umístění celého systému na jeden čip má své nesporné výhody z pohledu miniaturizace, kdy každý ušetřený kubický milimetr hraje významnou roli. Má ale také význam z pohledu spotřeby elektrické energie, což je místo, kde procesory ARM opět mají co nabídnout. Proto tyto procesory nachází uplatnění nejen v přenosných zařízeních, ale všude tam, kde spotřeba hraje významnou roli, tj. ve spotřební elektronice nebo v řídících jednotkách a systémech. Každé takové nasazení má své specifické požadavky, a proto jsou procesory ARM, resp. jejich instrukční sady, rozděleny do tří kategorií — A, R a M. Kategorie A jako aplikační, např. pro použití v mobilních telefonech, R jako real-time, určených např. pro řídící systémy kritické z pohledu bezpečnosti a přesnosti, M jako microcontroller, např. pro řídící systémy, spotřební elektroniku). Přičemž není bez zajímavosti, že některé konkrétní procesory jsou založeny na von Neumannově architektuře a jiné zase na harvardské.

Důraz na nízkou spotřebu je vidět i u vícejádrových ARMů. Zatímco u Intelu či AMD jsou jednotlivá procesorová jádra rovnocenná a stejně výkonná, v případě procesorů ARM lze často narazit na architekturu označova-

nou jako big.LITTLE, kdy konkrétní fyzický procesor obsahuje jádra výkonná (s větší spotřebou), která jsou aktivována operačním systémem v případě nutnosti provádět intenzivnější výpočty, a méně výkonná (s menší spotřebou), která jsou využívána, pokud zařízení nepotřebuje provádět intenzivní výpočty.

Variabilita, kterou architektura ARM přináší, má však také svou daň. Díky tomu, že se jednotlivé systémy s procesory ARM mezi jednotlivými výrobci často dost podstatně liší, je podpora na straně operačních systémů jen omezená. Většinou výrobci hardware chystají nebo upravují software na míru konkrétnímu zařízení a případná úprava software nebo upgrade operačního systému (bez podpory výrobce) nemusí být možná.

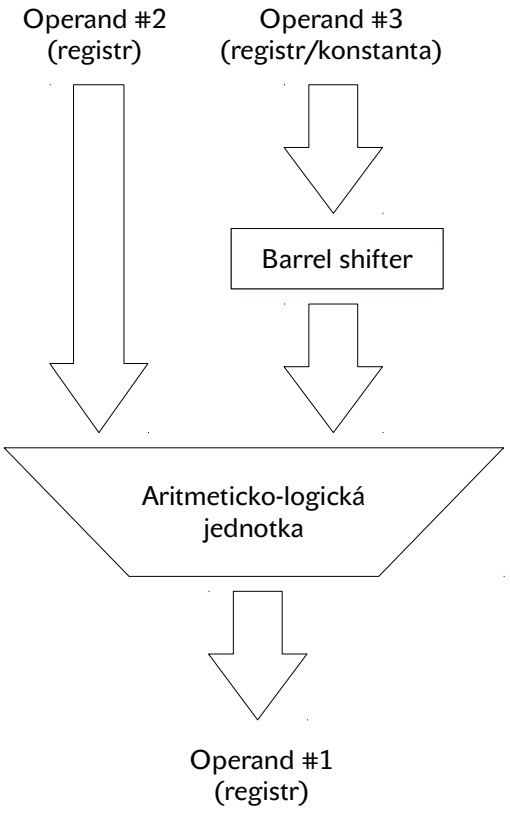
Instrukční sada ARMv7-A

Původní význam zkratky ARM byl *Acorn RISC Machine*, později byl upraven na *Advanced RISC Machine*. To naznačuje, že se jedná o architekturu RISC (Reduced Instruction Set Computer), tedy architekturu procesorů, která se snaží omezit množství instrukcí, které procesor má, na nejnutnější (nebo nejrozumnější) minimum. Díky tomu mohou být procesory efektivní při zpracování jednotlivých instrukcí a také levnější na výrobu. Autoři procesorů ARM při vytváření souboru instrukcí udělali několik zajímavých rozhodnutí, která stojí za povšimnutí.

Podívejme se podrobněji na architekturu ARMv7, která představuje poslední vývojovou větev 32bitových procesorů rodiny ARM. Poznamenejme, že tuto architekturu stále podporují i nejnovější šedesátičtyřbitové procesory (ARMv8).

Mezi základní kameny procesoru patří registry, rychlá paměť pro uchovávání dat, se kterými se právě pracuje. Procesory ARM poskytují programátorovi 16 třicetidvoubitových registrů pojmenovaných R0 až R15. Některé z těchto registrů mají své specifické určení. Např. registr R15 je takzvaný *program counter* (PC), tzn. že ukazuje na následující instrukci, která má být provedena, R14 je zase *link register* (LR), registr, do nějž je uložena návratová adresa při volání podprogramu, registr R13 pro změnu ukazuje na vrchol programového zásobníku (SP, *stack pointer*). Další registry mají svou úlohu danou konvencí, například R0 až R3 slouží k předávání argumentů funkcím a procedurám, nebo třeba registr R0 slouží k uložení návratové hodnoty při návratu z volání podprogramu. Vedle toho má procesor ještě další, řídící registry, přičemž významnou roli hraje příznakový registr CPSR (*current program status register*). Do něj je možné uložit příznaky, s jakým výsledkem skončila předchozí aritmetická operace, např. zda-li byl její výsledek nula, jestli byl výsledek záporný, atp.

Jednotlivé instrukce procesoru jsou v paměti uloženy jako třicetidvoubitové hodnoty. Obrázek ukazuje, jak v binární podobě vypadá zakódování aritmetické instrukce typu `ADD Rd, Rn, imm`, tj. operace, která do prvního operandu (registr Rd), přiřadí hodnotu Rn + imm. Pro další operace by to bylo podobné.



Rozeberme si, co všechno je v těchto 32 bitech uloženo. Bity 26 a 27 nastavené na 0 v tomto případě udávají, že se jedná o aritmetickou operaci, bity 21 až 24 udávají o jakou aritmetickou operaci se jedná (např. 0100 – sčítání, 0010 – odčítání, 1100 – logický součet, 0000 – logický součin). Bit č. 25 udává, že se jedná o operaci, jejíž třetí operand bude přímá hodnota (konstanta). Bit č. 20 určuje, zda operace po svém provedení bude měnit příznaky v registru CPSR podle toho, jaké hodnoty nabyl výsledek. Bity 0 až 19 jsou vyčleněny pro jednotlivé operandy; pro každý použitý registr jsou vyčleněny čtyři bity, tj. bity 12 až 15 určují cílový registr, kam bude uložena výsledná hodnota; bity 16 až 19 udávají druhý operand dané aritmetické operace. Zbývá 12 bitů pro uložení konstanty.

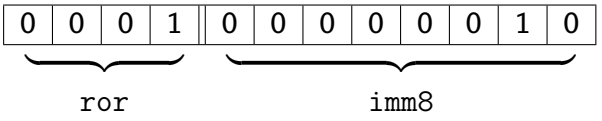
Těchto dvanáct bitů se pokusili tvůrci architektury ARM využít na maximum. Přímočarým řešením by bylo uložit konstantu do těchto dvanácti bitů přímo nebo ve formě dvojkového doplňku. To by nám dalo přípustné hodnoty z intervalu 0 až 4095, nebo -2048 až 2047. Tyto intervaly sice zahrnují běžně používané konstanty jako jsou 0, $\pm$ 1, $\pm$ 2, $\pm$ 4, 10, 16 atd., ale zahrnují i hodnoty, které v reálném programu příliš často nenajdete (např. 1337). Na druhou stranu taková často používaná konstanta 2147483648 (odpovídá 32bitovému číslu, jehož binární reprezentace má nejvyšší bit nastaven na jedna a zbylé na nula, což také odpovídá nejmenšímu číslu typu `int` ve formě dvojkového doplňku) se do těchto intervalů nevejde.

Proto při architektury ARM udělali její tvůrci netradiční rozhodnutí. Umístili před druhý vstup do aritmeticko-logické jednotky tzv. *barrel shifter*, jednotku, jež se stará o bitové posuny a rotace

To umožňuje na třetí operand aplikovat bitový posun nebo rotaci. Toho se elegantně využívá pro zakódování konstant do instrukce. Konstanta je uložena ve dvou složkách. Spodních 8 bitů je určeno pro celé neznamenkové číslo a horní čtyři bity reprezentují, opět neznamenkové číslo, dvojnásobek počtu bitových rotací vpravo, které se mají aplikovat na spodních 8 bitů. V pseudokódu by se to dalo vyjádřit jako:

```
imm32 = RotateRight(
            ZeroExtend32(imm8),
            ror * 2).
```

Již zmíněhou hodnotu 2147483648, tj. binárně 1000 0000 0000 0000 0000 0000 0000 0000, pak můžeme v tomto kódování vyjádřit jako:



V ukázkové instrukci jsme uvažovali jako třetí operand konstantu, avšak je možné použít i registr, tedy mít operaci typu `ADD Rd, Rn, Rm`. Díky tomu, že je před třetím operandem opět barrel shifter, je možné i na hodnotu Rm aplikovat bitový posun nebo rotaci (detaily již vynecháme).

Doposud jsme přehlíželi nejvyšší čtyři bity v zápisu instrukce. Ty udávají, za jaké podmínky má být instrukce provedena. Typicky je tam uložen příznak, že instrukce má být provedena vždy, avšak je možné určit, že se provedení instrukce má řídit podle nastavení příznaků v registru CPSR. Toho se využívá nejčastěji ve spojení s operací CMP (compare), která od sebe odečte své dva operandy a podle výsledku nastaví příznaky v registru CPSR. Praktické použití si můžeme ukázat na funkci signum,

která by v assembleru procesorů ARM mohla vypadat následovně.

```
; porovnej obsah registru R0 s 0
; a nastav CPSR
CMP R0, 0

; pokud je hodnota R0 větší než 0,
; přiřaď do R0 hodnotu 1
MOVGT R0, 1

; pokud je hodnota R0 menší než 0,
; přiřaď do R0 hodnotu -1
MVNLT R0, 1
```

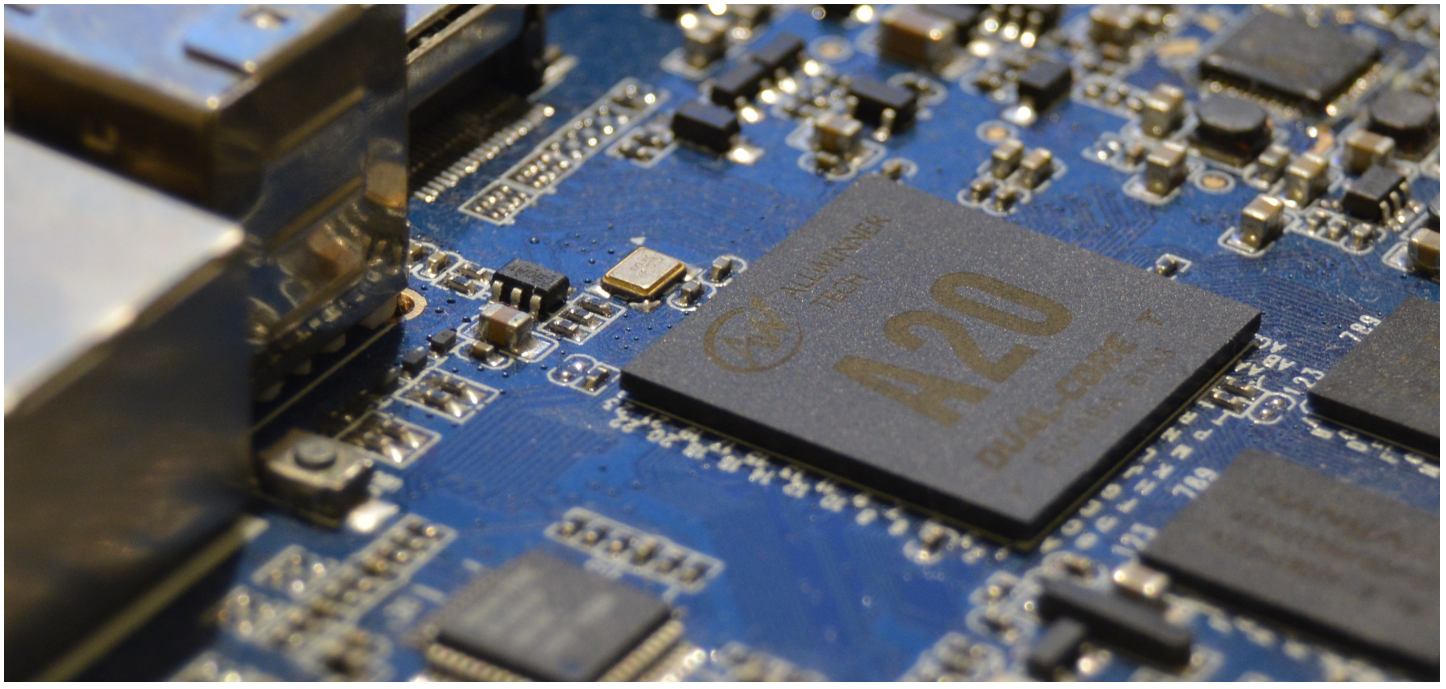
Všimněte si, že se tuto funkci podařilo implementovat bez podmíněných skoků.

Thumb

Podmíněné provádění instrukcí představuje užitečnou vlastnost, která najde použití při implementaci krátkých podmínek, ale z pohledu celého programu bude podmíněných instrukcí zlomek z celkového množství. Podíváme-li se na to z pohledu velikosti binárního kódu programu, celou jednu osminu kódu programu zabírají příznaky, za jakých podmínek má být instrukce provedena. A podobných neefektivit by šlo nalézt daleko více.

Z pohledu kapacit pamětí, které nabízí dnešní desktopové počítače nebo mobilní telefony, je to věc, kterou lze tolerovat. Ovšem v případě zařízení, která fungují jako řídící jednotky ve spotřební elektronice nebo podobných výrobcích, je takové plýtvání těžko obhajitelné, protože má přímý důsledek na finální cenu výrobku. Proto mají procesory rodiny ARM ještě jedno kódování instrukcí, označované jako Thumb, které kombinuje uložení instrukcí ve formě 16bitových a 32bitových hodnot. To umožňuje kód „zahustit“, tj. zvýšit poměr instrukcí na jednotku paměti. Toho využívají zejména procesorová jádra určená pro microtrollery (MCU). Zajímavé je, že procesory, které podporují oba formáty instrukcí, mohou mezi oběma formáty přecházet pomocí příznaku v instrukci skoku nebo volání podprogamu.

Náš výčet zajímavých vlastností procesorů rodiny ARMv7 není úplný, např. instrukce pro přístup k paměti by si určitě zasloužily také zvláštní kapitolu, o procesorech ARMv8 by se dal napsat celý samostatný článek. Formát našeho magazínu nám neumožňuje jít do hlubších detailů. Proto nezbývá než vám, čtenářům, doporučit pořízení nějakého jednodeskového počítače s ARMem, jako je třeba Raspberry PI, a nechat prozkoumávání vlastností těchto bezesporu zajímavých počítačů na vás.



Udělení Klirova stipendia

Eduard Bartl

V polovině dubna tohoto roku bylo na katedře informatiky poprvé uděleno Klirovo stipendium. Oceněno bylo celkem deset studentů, kteří dosáhli nejlepších studijních výsledků.

Foto: Martina Šaradínová, Žurnál UP

V jednom z předchozích čísel magazínu jsme zveřejnili záměr pravidelně oceňovat prospěchovým stipendiem nejlepší studenty informatických oborů na Přírodovědecké fakultě UP. Toto stipendium nese jméno profesora George J. Klira (1932 – 2016) – významného odborníka v oblasti počítačových a systémových věd českého původu, který úzce spolupracoval s některými členy katedry informatiky.

První předání Klirova stipendia proběhlo 14. dubna 2017. Oceněni byli dva nejlepší studenti v každém ročníku standardní doby studia napříč všemi informatickými obory. Jmenovitě se jedná o tyto studenty:

Ročník	Držitelé stipendia	
1.	Alena Svobodová	Tomáš Stropek
2.	Jana Lubojacká	Michal Václavek
3.	Tereza Šaratová	Martin Mazáč
4.	Lukáš Medelský	Tomáš Mikula
5.	Ondřej Zamec	Zdeněk Vídeňský

Každý z uvedených studentů byl za své vynikající studijní výsledky odměněn částkou 5 000 Kč. Na finančním zajištění stipendia se významně podíleli sponzoři Behaim IT Solutions, GMC Software a OLTIS Group, jejichž zástupci se také zúčastnili vyhlášení.

Klirovo stipendium za letní semestr aktuálního akademického roku bude uděleno v září 2017.

Redakce:
Petr Krajča, Petr Osička
Katedra informatiky PřF UP
17. listopadu 12
771 46 Olomouc

web: www.inf.upol.cz/magazin
email: magazin@inf.upol.cz
atom: http://www.inf.upol.cz/atom/magazin
telefon: 585634701
GPS: 49.591870, 17.262336