1 Sčítání čísel

Scéna: *Operandy*

Naším úkolem je sečíst dvě čísla napsaná v binární soustavě. Úkolem první scény je ukázat, jak se pracuje se sčítanci.

Oba sčítance mají 32 číslic; tento počet lze v rozumných mezích měnit v poli [**N=**] na jinou mocninu 2. Knoflíkem [**RND**] lze 1. nebo 2. sčítanci přiřadit jinou náhodnou hodnotu. V řadě případů také bude užitečné nastavit jeden sčítanec jako komplement druhého (0 proti 1 a 1 proti 0); to se provede odpovídajícími knoflíky [**KMPL**]. Nakonec lze hodnotu každé číslice změnit (z 1 na 0 a z 0 na 1) klepnutím myší.

Scéna: *Algoritmus ze základní školy*

Zdálo by se, že není třeba o sčítání podrobněji mluvit. Algoritmus se žáčci učí již na základní škole a zná jej každý gramotný člověk a způsob sám je velmi jednoduchý a na první pohled i rychlý.

Tato scéna umožňuje krokovat výpočet podle základní školy a ukazuje, že postup je ve skutečnosti pro větší čísla poměrně zdlouhavý (zkuste si jej pro 64 bitů). Jistě není třeba dodávat, že červená šipka označuje přenos do vyššího řádu a zelená tečka naopak naznačuje, že k přenosu nedošlo.

Při sčítání dvou čísel algoritmus postupuje zprava doleva a každý krok je vázán na výsledek předchozího (konkrétně na to, zda dojde k přenosu do vyššího řádu). Počet kroků je roven počtu číslic sčítanců.

Scéna: *Paralelní algoritmus ze základní školy*

V moderních procesorech je k dispozici dostatečné množství logických obvodů, takže (na rozdíl od lidského počtáře) mohou provádět velké množství operací současně a tím urychlovat výpočet. Při současné šířce slova v počítačích, kdy 64 bitů se stává standardem, je takové zrychlení opravdu žádoucí. Zdá se ale, že elementární algoritmus pro sčítání nelze paralelním prováděním operací nijak urychlit.

Je zřejmé, že nejobtížnější část výpočtu je určit, ve kterých sloupcích dochází k přenosu. Pokud to víme, dá se výsledný součet určit *jediným* paralelním krokem, kdy pro každý sloupec sečtene dvě číslice sloupce a přenos z nižšího řádu modulo 2, neboli výsledek ve sloupci je 1 právě když buď 1 nebo 3 z těchto hodnot jsou rovny 1.

Při podrobnějším pohledu na algoritmus pro sčítání je zřejmé, že pro některé sloupce umíme rozhodnout, zda z nich bude vycházet přenos do vyššího řádu, aniž bychom čekali na výsledek z řádů nižších.

- Jestliže ve sloupci oba sčítance mají hodnotu 1, pak z takového sloupce bude přenos vycházet *vždy*, bez ohledu na to, zda z nižších řádů přenos přijde nebo ne. Budeme říkat, že sloupec *generuje*.
- Jestliže ve sloupci oba sčítance mají hodnotu 0, pak z takového sloupce přenos *nikdy* nevyjde, i kdyby do něho vstoupil přenos z nižšího řádu. Budeme říkat, že sloupec *absorbuje*.
- Nakonec jestliže sloupec je komplementární, tj. v jednom sčítanci je 0 a v druhém 1, pak sloupec vytváří přenos pouze pokud vytváří přenos sloupec napravo od něj. Budeme říkat, že sloupec *přenáší*.

Povšimněte si, že sloupec zcela vpravo nemá co přenášet, protože napravo od něj již další sloupec není. U pravého sloupce je proto vždy možno rozhodnout o přenosu bez znalosti hodnot v jiných sloupcích.

Pro rozdělení sloupců do typů není třeba znát informaci v jiných sloupcích a proto je snadno provedeme pro všechny sloupce současně a nezávisle v jednom paralelním kroku.

Zkuste nyní algoritmus krokovat. V prvním kroku se u sloupce vpravo a u všech sloupců, které generují nebo absorbují, určí, zda z nich vychází přenos. Zbývající (přenášející) sloupce vytvoří "ostrůvky" nerozhodnutých sloupců, oddělené rozhodnutými sloupci.

Šířka těchto ostrůvků bývá výrazně menší, než je šířka celých čísel a dá se dokázat, předpokládajíce náhodné a navzájem nezávislé hodnoty číslic sčítanců, že generujících a absorbujícíh sloupců bývá okolo jedné poloviny všech sloupců (jednoduchý důkaz) a šířka největšího ostrůvku je obvykle řádově kolem logaritmu šířky slova (značně složitý důkaz).

Můžete si zkusit opakovaně generovat náhodné sčítance a provádět první krok algoritmu a sledovat, jak ostrůvky nerozhodnutých sloupců vypadají.

Nyní budeme pokračovat krokování. V druhém kroku je možno o přenosu rozhodnout v pravých sloupcích ostrůvků a v každém dalším kroku se ostrůvky postupně zmenšují zpravo doleva, o 1 sloupec v každém kroku. Současně se přitom zpracovává tolik sloupců, kolik je dosud nerozhodnutých ostrůvků a doba paralelně prováděného výpočtu je proto dána maximální šířkou ostrůvku a je proto obvykle výrazně menší, než u základní varianty sčítacího algoritmu.

Scéna: *Paralelní algoritmus - nejhorší případ*

Jak bylo řečeno v předchozí scéně, doba nutná k paralelnímu provedení sčítání podle vykládaného algoritmu je dána šířkou největšího ostrůvku a je tedy silně závislá na konkrétním tvaru sčítanců.

V nejlepším případě, pokud například sčítáme dvě stejná čísla, ostrůvky vůbec nevzniknou a rozhodnutí o všech přenosech se dostane v jednom kroku a i v průměrném případě je algoritmus velmi rychlý.

V nejhorším případě však bohužel je tento algoritmus stejně rychlý (či lépe řečeno pomalý) jako sériově prováděný algoritmus ze základní školy. Nastává to v případě, kdy všechny sloupce (s případnou výjimkou sloupce vpravo) jsou komplementární. Pak totiž vznikne jen jeden ostrůvek, který má šířku o 1 menší než je šířka sčítanců a proto výpočet probíhá úplně stejně jako v základním algorimu a paralelismus je zcela nevyužit.

Tento případ generuje tato scéna; zkuste si výpočet krokovat, ale myslím, že jej ani nedokončíte, protože trvá příliš dlouho a nic nového se při něm nedozvíte.

Scéna: *Bloky*

Nyní začneme vysvětlovat jiný algoritmus, obvykle označovaný jako carry look-ahead, který zaručeně i v nejhorším případě počítá velmi rychle, rychlostí srovnatelnou s rychlostí, kterou paralelizovaný elementární algoritmus počítá v obvyklém případě, a tedy podstatně rychleji než elementární algoritmus v základním provedení.

Základním prvkem pro nový algoritmus je blok. *Blok* je souvislý soubor sloupců, na displeji je označen žlutým obdélníkem. Šířku bloku je možno na displeji měnit tažením jeho levé nebo pravé strany myší. Polohu bloku lze měnit tažením bloku myší za libovolné jiné místo než je jeho pravý a levý okraj nebo okénko s číslicí. Je též možno požádat o vytvoření nového (náhodného) bloku.

U bloku nás pro dané hodnoty sčítanců bude zajímat, zda z jeho nejvyššího (tj. nejlevějšího) sloupce vychází přenos. Zda k tomu dojde však se budeme snažit určit pouze na základě těch číslic sčítanců, které jsou uvnitř bloku. To pochopitelně vždy nepůjde a proto budeme podobně jako už jsme to učinili výše, rozlišovat následující tři případy:

- blok *generuje*: některý sloupec bloku obsahuje dvě číslice 1 a všechny sloupce bloku, které leží nalevo od něho, jsou komplementární (obsahují jednu 1 a jednu 0); tento případ budeme označovat symbolem '+'
- blok *absorbuje*: některý sloupec bloku obsahuje dvě číslice 0 a všechny sloupce bloku, které leží nalevo od něho, jsou komplementární (obsahují jednu 1 a jednu 0); tento případ budeme označovat symbolem '-'
- blok *přenáší*: všechny sloupce bloku jsou komplementární; tento případ budeme označovat symbolem '<' (s výjimkou případu, kdy blok zahrnuje sloupec zcela vpravo, kdy budeme používat symbol '-'.

Je jasné, že tyto tři případy se navzájem vylučují a jeden z nich vždy nastává. V prvním případě, kdy blok generuje, ze sloupce obsahujícího dvě 1 vyjde přenos a je přenášen komplementárními sloupci až k levému okraji bloku a ven z něho, proto z levého sloupce bloku přenos vychází. V

druhém případě sloupec se dvěma nulami absorbuje případný přenos, který by do tohoto sloupce vstoupil z nižšího řádu a sloupce nalevo od něj samy o sobě přenos vytvořit nejsou schopny.

V nejzajímavějším třetím případě sloupce bloku samy přenos nevytvoří, ale pokud ze sloupce stojícího vpravo od bloku do bloku přenos vstoupí, sloupce bloku přenos přenesou celým blokem, až nakonec vystoupí z levého sloupce bloku dále. Jestliže však vpravo od bloku již žádný jiný sloupec neleží, nebude co přenášet, což vysvětluje výjimku v označování přenášejícího bloku.

Vyzkoušejte různé hodnoty sčítanců a šířku i polohu bloku a sledujte, která z možností '+', '-' a '<' nastává.

Scéna: *Jednosloupcový blok*

Jak již bylo naznačeno výše, výpočet hodnoty bloku tvořeného jedním sloupcem je jednoduchý: sloupec obsahující dvě 1 generuje, pokud obsahuje dvě 0, pak absorbuje a komplementární blok zcela vpravo negeneruje a jinak přenáší.

Ověřte si tyto možnosti pro různé hodnoty sčítanců a polohu bloku, který je možno myší táhnout jako tomu bylo v předchozí scéně (ale nelze měnit jeho šířku).

Scéna: *Široký blok*

Hodnotu bloku složeného z více než jednoho sloupce lze určit snadno podle definice: Postupujíce zleva doprava, najdeme první nekomplementární sloupec bloku. Obsahuje-li dvě 1, blok generuje, jinak absorbuje. Pokud je ale blok složen jen z kompementárních sloupců, pak přenáší nebo negeneruje podle toho, zda zasahuje až do pravého sloupce sčítanců.

Tento postup je sice jednoduchý, ale sekvenční a v nejhorším případě (komplementární operandy) by vedl ke stejně pomalému výpočtu jako předchozí dvě metody.

Budeme proto postupovat jinak:
rozdělíme blok *libovolným způsobem* na dva bloky obsahující oba alespoň jeden sloupec (stiskněte knoflík [**Krok**] v ovladači nazvaném [**Konstrukce**]),
určíme hodnoty podbloků a
spočteme hodnotu celého bloku následujícím způsobem:

- jestliže levý blok generuje, celý blok generuje také,
- jestliže levý blok absorbuje, celý blok absorbuje také,
- jestliže levý blok přenáší, celý blok má hodnotu rovnou hodnotě pravého bloku.

Pokud levý blok generuje nebo absorbuje, jeho rozhodující sloupec (nejlevější nekomplementární) je také rozhodujícím pro celý blok. Pokud levý blok přenáší, představuje pouze nedůležité přenášející sloupce pro pravý blok.

Hodnoty podbloků, které obsahují více sloupců se určí rekurzivně dalším dělením (používajíce opětovně knoflík [**Krok**] v ovladači [**Konstrukce**]).

Při paralelním výpočtu lze hodnoty levého i pravého bloku určovat *současně*.

Rekurzivní výpočet hodnoty bloku lze krokovat knoflíkem [**Krok**] v ovladači [**Výpočet**], který se objeví v okamžiku, kdy je rekurzivní konstrukce stromu výpočtu ukončena.

Knoflíky [**Zpět**] v obou ovladačích vrací zpět akci knoflíku [**Krok**] a knoflíky [**Dokonči**] a [**Zruš**] umožňují příslušnou akci zcela dokončit nebo vrátit do původního stavu. Knoflík [**Široký**] umožňuje volit mezi dvěma způsoby kreslení vrcholů stromu rekurze. V prvním je vrchol natažen mezi krajními sloupci bloku, který reprezentuje, v druhém má šířku jednoho sloupce. Druhá metoda bude užívána ve scénách znázorňujících úplnou sčítačku, kde bude zobrazováno několik stromů najednou a široké vrcholy by daly nepřehledné schéma.

Scéna: *Nejhorší dělení*

Pro správnost výpočtu hodnoty bloku je zcela lhostejné, jak jej dělíme do podbloků. Způsob dělení je ale podstatný pro počet paralelních kroků, které je nutno provést k ukončení výpočtu hodnoty. Tato scéna ukazuje nejhorší možný způsob dělení: od celého bloku se vždy oddělí jediný

sloupec vpravo nebo vlevo. Schéma výpočtu, které takto dostaneme pak má takovou hloubku, že se sotva vejde na obrazovku a vedlo by k algoritmu výpočetně srovnatelnému s algoritmem ze základní školy.

Zkuste si vytvořit široký blok a sledujte jeho strom výpočtu.

Scéna: *Optimální dělení*

Čtenář, který má alespoň základní znalosti v návrhu efektivních algoritmů, jistě ví nebo alespoň tuší, že pro rychlost paralelního výpočtu je optimální dělit blok na dva stejně velké podbloky poloviční velikosti (nebo na dva podbloky lišící se v počtu sloupců o 1 pokud blok sám má lichý počet sloupců a na přesné poloviny rozdělit nejde). Počet paralelních kroků pro výpočet hodnoty je pak $\lceil \log_2 w \rceil$, kde w je počet sloupců bloku a je to to nejlepší, co můžeme dosáhnout.

Zkuste si vytvořit široký blok a sledujte jeho strom výpočtu. Obměňujte blok i sčítance a sledujte konstrukci stromu výpočtu i odpovídající výpočet hodnot bloků.

Scéna: *Hraniční dělení*

Nyní popíšeme jiný způsob dělení, který je pro některé bloky mírně neoptimální, ale má výhody, které se stanou zřejmými později. V této scéně již je nutné předpokládat, že počet šířka operandů je mocninou dvojky (což v současné době v počítačové aritmetice nezpůsobuje prakticky žádná omezení; první mikroprocesory pracovaly se šířkou 8 bitů a záhy bily vystřídány 16-bitovými, nyní končí éra 32-bitových a tato kniha je už psána na 64-bitovém procesoru - vše jsou mocniny 2).

Blok rozdělíme nejprve podle hranice procházející polovinou šířky sčítanců (nikoli polovinou bloku), pak podél hranice procházející první nebo třetí čtvrtinou, pak 1., 3., 5. nebo 7. osminou atd. Pochopitelně jestliže některá hranice blokem neprochází (např. pokud je blok celý v dolní polovině čísel nebo ji přesahuje o jediný sloupec), pak může být odpovídající krok přeskočen.

Blok obsahující všechny sloupce je pochopitelně dělen optimálně na poloviny. Menší bloky sice mohou být děleny neoptimálně, ale počet úrovní dělení nebude nikdy větší než u všesloupcového bloku. Jelikož se všesloupcový blok v návrhu algoritmu bude vyskytovat, bude paralelní výpočetní složitost dána počtem úrovní rozkladu tohoto největšího bloku a optimalizací rozkladu bloků menších by se počet paralelních kroků výpočtu stejně nemohl zlepšit.

Dělení podle binárních hranic má ale výhodu, že všechny bloky jsou děleny v odpovídajících úrovních stejným způsobem, tedy podle stejných hranic, což se příznivě projeví na tvaru a jednoduchosti výsledného obvodu.

Scéna: *Nejvyšší sloupec*

V této scéně se budeme zabývat otázkou, zda z nejvyššího řádu, tedy ze sloupce úplně vlevo, vychází přenos. Obdobnou otázku budeme muset vyřešit i pro ostatní sloupce, pro levý sloupec je ale nejobtížnější, protože závisí na všech číslicích obou sčítanců.

Pro vyřešení tohoto problému stačí uvažovat blok, obsahující všechny sloupce, tedy prostírající se vlevo od sloupce, který zkoumáme, až ke sloupci, který leží úplně napravo (a tedy máme jistotu, že do něj zprava nemůže vstoupit přenos).

Tento všesloupcový blok nemá nikdy hodnotu '$<$' a už jsme rozebírali, že pokud generuje, pak z nejvyššího řádu vychází přenos, zatímco pokud absorbuje nebo přenáší, pak z nejvyššího řádu přenos nevychází. Naše otázka se tedy snadno zodpoví určením hodnoty bloku, kterou určujeme na základě dělení bloku podle binárních hranic jak bylo popsáno v předchozí scéně.

Knoflíky ovladače [**Konstrukce**] si nyní rozviňte obvod pro výpočet hodnoty bloku a knoflíky ovladače [**Výpočet**] si určujte hodnotu bloku i podbloků vzniklých dělením a tedy i existenci přenosu v nejvyšším řádu pro různé hodnoty sčítanců. Výpočet je krokován po jednotlivých paralelních krocích.

Nyní již je tedy patrna hlavní myšlenka algoritmu: současně s prováděním početních úkonů v pravé části schématu, která zahrnuje méně významné bity, se zpracují levé poloviny sčítanců. Pokud se zjistí, že levá polovina generuje nebo absorbuje, není třeba znát výsledek z pravé části. Nové na algoritmu je ale to, že pokud levá polovina přenáší, pak případný přenos z pravé poloviny sčítanců není nutné "postrkovat" levou částí sloupec po sloupci, jako to činí elementární algoritmus

ve své školní podobě vždy a v paralelizované podobě přinejmenším v nejhorším případě, ale přenos "přeskočí" z hranice v polovině sčítanců do nejlevějšího sloupce okamžitě, protože víme, že by k tomu při detailním počítání stejně došlo.

Je vidět, že výpočet je velmi rychlý - počet vrstev obvodu a tedy i počet paralelních kroků potřebných pro výpočet je $\lceil \log_2 n \rceil$, kde n je šířka sčítanců. Obzvláště důležité je to, že výpočet *nezávisí* na hodnotách sčítanců a je tedy takto rychlý i v nejhorším případě.

Scéna: *Obecný sloupec*

Podobně jako byl v předchozí scéně určen přenos v nejvyšším řádu, ukážeme si, jak provést určení přenosu v libovolném jiném sloupci. Výběr sloupce pro ilustraci metody provedeme kliknutím na malý bledý čtvereček nad sloupcem.

Jako v předchozí scéně stačí určit hodnotu bloku, určeného vlevo uvažovaným sloupcem a vpravo zasahující až na samé pravý okraj sčítanců. Takovýto blok nikdy nemůže dostat zprava přenos z nižšího řádu, protože vpravo od něj již nic není, a v naší klasifikaci nikdy nenabývá hodnoty '<'. Proto z jeho levého sloupce, tedy z našeho uvažovaného sloupce, vychází přenos právě když tento blok generuje.

Při konstrukci si povšimněte, že blok je dělen podle binárních hranic. Jak již bylo uvedeno, je pro nás dostačující aby počet vrstev při dělení bloku nepřekročil počet vrstev, vypočítávajících hodnotu všesloupcového bloku.

Scéna: *Překrývání*

Nyní již víme, jak zkonstruovat úplnou sčítačku: pro každý sloupec si sestrojíme obvod, určující zda z tohoto sloupce vychází přenos, způsobem popsaným v předchozích dvou scénách a pak z informace o přenosech určíme v jednom paralelním kroku hodnotu součtu. Všechny obvody pro výpočty přenosů pracují současně.

V této scéně se konečně dozvíte důvod pro dělení bloků podle binárních hranic. Na displeji je zobrazen obvod pro výpočet přenosu pro levý sloupec. Klepněte nyní na malý bledý čtvereček nad libovolným jiným sloupcem (výhodně nad sloupcem v levé části schématu). Tím se zobrazí obvod pro výpočet přenosu pro zvolený sloupec. Podle stavu konstrukce (knoflíky ovladače [**Konstrukce**]) budou obvody v různém stavu vývinu od prostého bloku po úplně dokončený obvod pro výpočet jeho hodnoty.

Nyní již určitě vidíte, proč preferujeme dělení podle binárních hranic. Při tomto způsobu se obvody pro určování přenosu v různých sloupcích podstatně *překrývají*. Není tedy nutné budovat tyto obvody zcela odděleně, což by vedlo k velkému plýtvání hardwarem, ale překrývající se část vytvoříme pouze jednou a její výsledek pak rozvádíme na vstup více než jednoho následného hradla.

Klepáním do čtverečků nad sloupci je možno zobrazovat (a nebo naopak skrývat) libovolný sloupec a sledovat překrývání obvodů pro výpočet přenosů. Je vidět, že překrývání je opravdu velké, například všechny sloupce z levé části mají shodnou pravou část svých obvodů. Nakonec se zjistí, že počet hradel pro *všechny* sloupce není o moc větší než je obvod pro levý, nejsložitější, sloupec.

Je též možno zvolit nebo vypustit několik sloupců najednou: stiskněte myš v bledém čtverci nad některým sloupcem a tahněte myš do strany.

Scéna: *Kompletní sčítačka*

Tato scéna již jen shrnuje, co se v předchozí scéně objevilo při volbě všech sloupců a plně rozvinuté konstrukci, umožňuje ale krokování výpočtu pro různé hodnoty sčítanců.

Teprve na úplné sčítačce je vidět, že je její konstrukce vlastně jednoduchá, je-li formulována rekurzivně: Vytvoříme dva obvody pro výpočet přenosů pro poloviční šířku slova, mírně zobecněné možností přivádět kromě dvou sčítanců ještě zprava přenos za sousedního obvodu. Pravý podobvod již dává polovinu konečných výstupů, výstupy levého podobvodu se ještě všechny zkombinují s nejvýznamnějším výstupem pravé části.

Povšimněte si, že sčítačka má $\log_2 N$ vrstev a v důsledku velkého překrývání je v každé vrstvě přesně $N/2$ hradel, které kombinují třístavové hodnoty jednotlivých bloků.

Scéna: *Vrstvená sčítačka*

V kompletní sčítačce ukázané v předchozí scéně byly hradla (vrcholů stromů) uspořádána do vodorovných vrstev, ale jistá propojení mezi (žlutými) hradly přecházela přes několik vrstev. V této scéně je sčítačka doplněna dalšími hradly, vybarvenými růžově, která zaplňují "díry" v obvodu. Růžová hradla neprovádějí žádný výpočet a jsou to ve skutečnosti paměťové elementy, které si mohou zapamatovat jednu třístavovou hodnotu. V každém kroku (kliknutí knoflíku [**Krok**]) růžové hradlo přečte hodnotu vstupu a zapamatuje si jej. Výstup je vždy roven uchovávané hodnotě.

Růžové prvky jsou velmi podobné tomu, co se v elektronice nazývá klopný obvod typu D; jediný rozdíl je, že si klopný obvod pamatuje jediný bit. Třístavovou veličinu je ale možno uchovávat ve dvou bitech a proto se naše růžové prvky obvykle implementují dvojicí klopných obvodů typu D.

Zkuste si znovu počítat hodnoty přenosů; výpočet probíhá stejně jako v předchozí scéně, ale vstupní hodnoty přicházejí "just in time" a nemusí čekat jako se to stávalo v minulé scéně.

Je vidět, že postup mezivýsledků obvodem je velmi pravidelný, v každém kroku jedna nová vrstva určí své hodnoty. V následujících scénách uvidíme, proč je to užitečné.

Scéna: *Zapomínající sčítačka*

Jak bylo uvedeno v minulé scéně, tok informace v obvodu je velmi pravidelný. V každém kroku je aktivní jedna vrstva, která vypočte své hodnoty z hodnot v *předchozí* vrstvě. Protože nás mezivýsledky ve skutečnosti nezajímají a cílem je určit hodnoty poslední vrstvy, je možné mezivýsledky zapomenout, jakmile byly použity následující vrstvou.

Zkuste znovy provádět výpočet a uvidíte, že je dostatečné si uchovávat pouze mezivýsledky jediné vrstvy.

Scéna: *Zřetězená sčítačka*

V předchozí scéně jsme viděli, že stačí si pamatovat vždy pouze výsledky jediné vrstvy. Ty vrstvy, jejichž jejichž výsledky již byly použity, "zahálí" a je je možno využít k sčítání jiných dvojic sčítanců. Jakmile hodnoty pro jeden pár postoupí o jednu vrstvu dolů, je možné do obvodu vložit nový pár sčítanců, který postupuje obvodem se zpožděním jedné vrstvy.

V této scéně sčítáme plynulý proud dvojic sčítanců a každý pár má svoji barvu, která označuje jemu odpovídající mezivýsledky. Prvky obvodu mají svoji původní barvu pouze dokud k nim nedorazí první mezivýsledky, pak jsou již označovány barvou páru, který právě zpracovávají. Při krokování je jasně vidět, jak informace obvodem postupuje.

Sčítačka typu carry look-ahead je tedy velmi silný nástroj, kromě toho, že jeden pár v obvodu stráví pouze čas úměrný logaritmu počtu bitů a doba trvání výpočtu je zcela nezávislá na jejich hodnotách, mohou do obvodu sčítance vstupovat bezprostředně za sebou s odstupem, který na velikosti sčítanců nezávisí a je dán pouze výpočetním zpožděním použitých výpočetních prvků.

Scéna: *Animovaná sčítačka*

Tato scéna přináší inženýrský náhled na implementaci zřetězené sčítačky z předcházející scény. Růžové paměťové prvky jsou nezměněny, pouze jsou nakresleny poněkud menší. Na druhé straně každý žluté výpočetní hradlo je doplněno o jeden růžový paměťová prvek stejného typu jako růžové prvky popsané v předchozích scénách.

Žlutá výpočetní hradla pracují *asynchronně*: jakmile se změní vstupní hodnoty, odpovídajícím způsobem se ihned změní i výstup. Přesněji řečeno, změna se objeví na výstupu po jistém časovém okamžiku, zpoždění, které může být *různé* pro různá hradla.

Pokud bychom zpracovávali pouze jeden pár sčítanců, růžové paměťové prvky by nebyly třeba, ale rychlost postupu informace by byla v různých částech obvodu jiná. To by ale v případě zřetězeného zpracování více párů sčítanců vedlo k tomu, že by nebylo možné hodnoty odpovídající různým párům na výstupu obvodu odlišit.

Z tohoto důvodu musíme používat *synchronní* obvod a synchronizace je dosahována růžovými paměťovými prvky ve všech pozicích obvodu. Výstup paměťového obvodu je roven uchovávané hodnotě; po většinu doby prvek ignoruje vstupní hodnotu. Paměťové prvky však dostávají hodinový signál (jeho přivádění k prvkům není znázorněno). Jakmile paměťový prvek dostane hodinový

signál (v reálných systémech například vzestupná hrana hodinového signálu, u našeho appletu kliknutí knoflíku [**Krok**]), prvek přečte vstupní hodnotu a nahradí jí dříve uchovávanou hodnotu. Po této mžikové události se zapamatovaná hodnota dále nemění až do dalšího hodinového impulzu.

Zkuste si výpočet s použitím knoflíku [**Krok**]; v okamžiku hodinového impulzu (kliknutí) všechny paměťové prvky najednou nahradí původní uchovávanou hodnotu hodnotou, která byla na vstupu. Animace ukazuje, jak informace postupuje podél vodičů, připojujících výstupy paměťových prvků se vstupy následujících hradel, kterými mohou být výpočetní hradla nebo další paměťové prvky.

Jakmile informace dorazí na výpočetní hradlo, je zpracována a s jistým zpožděním se výsledek objeví na jeho výstupu (zpoždění jsou různá pro různá hradla - rozdíly jsou úmyslně zvýrazněny), což se projeví m.j. změnou barvy hradla.

Barvy hradel a signálů reflektují barvu odpovídajícího páru sčítanců, ke kterému se vztahují. Je tedy snadné sledovat postup informace obvodem.

Jistě je vám zřejmé, že hodinový kmitočet musí být takový, aby doba mezi dvěma po sobě následujícími hodinovými pulzy byla dostatečná pro přenos signálu mezi hradly a jeho zpracování. Náš applet je silně synchronizován a nepřijímá kliknutí, dokud neprovede všechny nutné operace, ale v případě reálného obvodu by příliš rychlé hodiny vedly k chybným výpočtům.

Scéna: *Číslování*

Tato scéna již není součástí vlastní procházky appletem, nicméně si hned ukážeme jistou magii čísel.

Sloupce sčítačky jsou očíslovány zprava doleva od 0 do $n-1$, kde n je bitová šířka sčítanců, v úvodním případě $n = 32$. Vrstvy hradel jsou očíslovány shora dolů od 0 do $\log_2 n - 1$ (pamatujte, že n je mocnina 2 a proto je $\log_2 n$ celé číslo). Na úvodním obrázku je $\log_2 32 = 5$. Vrstva operandů je očíslována jako -1.

Sčítačka je znázorněna v úplné podobě se žlutými výpočetními hradly se dvěma vstupy a vloženými růžovými paměťovým obvody. Hradla jsou očíslována čísly, které nemají žádnou souvislost s operandy nebo mezivýsledky, ale popisují typ hradla: 1 je žluté výpočetní hradlo a 0 je růžové paměťové hradlo.

Za této situace se podívejte na posloupnost 0 a 1 v hradlech libovolného sloupce a zjistíte, že představuje binární zápis čísla sloupce (s nejvýznamnějším bitem dole a nejméně významným nahoře ve vrstvě 0). Ve vrstvě očíslované i je číslice, která v binárním zápisu vystupuje u mocniny 2^i.

Stiskněte nyní myší na libovolné hradlo. Odpovídající obdélník zčervená a zčervenají také obdélníky hradel, na jejichž hodnotách závisí hodnota stisknutého hradla. Zčervenalé obdélníky ve vrstvě operandů (očíslované -1) vytvářejí interval, jehož levý konec je ve sloupci, ve kterém se nachází stisknuté hradlo, zatímco pravý konec je ve sloupci, jehož pořadové číslo se dostane tak, že ve sloupci obsahujícím stisknuté hradlo nahradíme nulou číslo v stisknutém hradlu a čísla ve všech obdélnících nad stisknutým hradlem.

Stiskněte např. hradlo v sloupci 29 ve vrstvě 3. Jelikož 29 je binárně 11101, pak nahrazením číslic ve vrstvách 3, 2, 1, 0 nulami dostaneme 10000 neboli dekadicky 16 a při stisknutí zmíněného hradla zčervenají operandy od sloupce 29 až po sloupec 16.

U žlutého hradla jsou oba vstupy připojeny na výstupy hradel v předchozí vrstvě. Jeden vstup je připojen na hradlo ve stejném sloupci, druhý na hradlo ve sloupci, jehož pořadové číslo se dostane tak, že se v sloupci se stisknutým hradlem nahradí nulou všechny číslice nad stisknutým hradlem, a pak se od vzniklého čísla odečte 1.

Stiskneme-li ještě jednou hradlo ve vrstvě 3 a sloupci 29 neboli binárně 11101, pak po náhradě nulou ve vrstvách 2, 1, 0 dostaneme 11000, neboli 24 a vidíme, že pravý vstup hradla je napojen skutečně do sloupce 23 = 24-1. Levý vstup totiž závisí na číslicích operandů od sloupce 29 do $11000_2 = 24$ a proto pravý vstup hradla zpracovává hodnoty od následujícího sloupce vpravo, tedy od sloupce 24-1=23.

2 Laboratoř

Booleovská proměnná nebo *bit* je proměnná, která má dvě možné hodnoty, 0 a 1.
Funkce $\vee$, $\wedge$ a $\oplus$ dvou booleovských proměnných jsou definovány takto

x	y	$x \vee y$	$x \wedge y$	$x \oplus y$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Ve zbývající části kapitoly budeme předpokládat, že k je kladné celé číslo a $n = 2^k$, takže $k = \log_2 n$.

Pokud x a y jsou celá nezáporná čísla $y \neq 0$, budeme jako x **div** y označovat výsledek celočíselného dělení čísla x číslem y, tedy největší celé nezáporné číslo q takové, že $qy \leq x$, zatímco x **mod** y je zbytek při celočíselném dělení čísla x číslem y, tedy číslo $x - y \cdot (x$ **div** $y)$. Pro programátory v C poznamenávám, že použité označení se používá v Pascalu, zatímco v C se používají symboly / a %.

Pro celá čísla i a j taková, že $0 \leq i < k$ a $0 \leq j < n$, označme $\xi(i,j) = (j$ **div** $2^i)$ **mod** 2.
Pro celá čísla i a j taková, že $-1 \leq i < k$ a $0 \leq j < n$, označme $\kappa(i,j) = (j - j$ **mod** $2^{i+1})$.

Nyní již můžeme formulovat sčítací algoritmus:

```
 1           Vstup:     Kladné celé číslo k a číslo n = 2^k;
 2                      posloupnosti bitů a_{n-1}...a_1a_0 a b_{n-1}...b_1b_0;
 3           Proměnné:  bity g_{i,j} a p_{i,j} pro i = -1,0,1,...,k-1, j = 0,1,...,n-1;
 4                      bity c_j pro j = 0,1,...,n-1;
 5           Výstup:    bity s_j pro j = 0,1,...,n;
 6           for j = 0,1,...,n-1 pardo
 7                begin g_{-1,j} ← a_j ∧ b_j; p_{-1,j} ← a_j ∨ b_j; end;
 8 AddFori   for i = 0,1,...,k-1 do
 9 AddForj        for j = 0,1,...,n-1 pardo
10                    if ξ(i,j) = 1
11                    then begin
12 AddG                    g_{i,j} ← g_{i-1,j} ∨ (p_{i-1,j} ∧ g_{i-1,κ(i,j)-1});
13 AddP                    p_{i,j} ← p_{i-1,j} ∧ p_{i-1,κ(i,j)-1};
14                         end;
15                    else begin
16                         g_{i,j} ← g_{i-1,j};
17                         p_{i,j} ← p_{i-1,j};
18                         end;
19          s_0 = a_0 ⊕ b_0; s_n = g_{k-1,n-1};
20 AddFors  for j = 1,...,n-1 pardo
21 AddSets       s_j = a_j ⊕ b_j ⊕ g_{k-1,j-1};
```

Sčítání binárně zapsaných čísel

Symbol **pardo** znamená, že tělo příslušného **for**-cyklu se provádí pro všechny hodnoty cyklové proměnné současně, například různými procesory nebo částmi obvodu implementujícího algoritmus. V našem případě se **for**-cyklus se záhlavím v řádce 9 provádí (nebo může provádět) současně pro všech n hodnot cyklové proměnné j, takže je zřejmé, že může být vykonán v konstantním čase nezávislém na délce n operandů.

Konstantní čas vyžaduje i provedení **for**-cyklu se záhlavím v řádce 20, který může také být prováděn pro všechny hodnoty j v témže okamžiku.

Na druhé straně **for**-cyklus se záhlavím v řádce 8 se musí provádět postupně pro jednotlivé hodnoty i, protože při provádění jeho těla pro nějaké i již musí být známy. Celkově proto provedení algoritmu vyžaduje čas $O(\log n)$, přičemž se provede celkově $O(n \log n)$ booleovských operací.

Ve škole dokážeme, že posloupnost $s_n \ldots s_1 s_0$ představuje binární zápis součtu čísel s binárními zápisy $a_{n-1} \ldots a_1 a_0$ a $b_{n-1} \ldots b_1 b_0$.

Na uvedený program je možno se dívat nejenom jako na program pro výpočet součtu dvou binárně zapsaných čísel pomocí booleovských operací $\wedge$, $\vee$ a $\oplus$ na počítači s možností paralelismu, ale také jako na popis kombinačního booleovského obvodu, jak bylo znázorněno na obrázcích v procházce, který obsahuje obdélníkovou matici podobvodů uspořádaných do řádků $-1, 0, 1, \ldots, k$ a sloupců $n-1, \ldots, 1, 0$, kde podobvod v průsečíku i-tého řádku a j-tého sloupce má 2 výstupy označené $g_{i,j}$ a $p_{i,j}$ a buď 4 vstupy pokud $i \geq 0$ a $\xi(i,j) = 1$ nebo 2 vstupy jinak, a dále obsahují jeden řádek složený z $n+1$ podobvodů určujících hledané hodnoty s_i, přičemž vzorce uvedené v programu určují jednak způsob napojení vstupů podobvodů v jednom řádku na výstupy podobvodů předchozích řádků nebo na vstupní hodnoty $a_{n-1} \ldots a_1 a_0$ a $b_{n-1} \ldots b_1 b_0$, jednak funkci podobvodů, neboli způsob, jak se výstupy vypočtou na základě několika elementárních hradel **and**, **or** a $\oplus$.

3 Škola

Naším cílem je dokázat, že pokud se bity $s_n, s_{n-1}, \ldots, s_1, s_0$ vypočtou z bitů $a_{n-1}, \ldots, a_1, a_0$ a $b_{n-1}, \ldots, b_1, b_0$ způsobem popsaným v laboratoři, pak skutečně $s_n \ldots s_1 s_0$ je binární zápis součtu čísel $a_{n-1} \ldots a_1 a_0$ a $b_{n-1} \ldots b_1 b_0$, neboli

$$\sum_{i=0}^{n-1} a_i 2^i + \sum_{i=0}^{n-1} b_i 2^i = \sum_{i=0}^{n} s_i 2^i \qquad (S)$$

Běžný algoritmus sčítání popisuje následující věta, kterou zde nebudeme dokazovat

Tvrzení 1 *[Labeled: Addition] Nechť jsou dány posloupnosti bitů $a_{n-1}, \ldots, a_1, a_0$ a $b_{n-1}, \ldots, b_1, b_0$. Definujme bity $c_{n-1} \ldots, c_1, c_0, c_{-1}$ následujícím způsobem:*

$$c_{-1} = 0 \quad c_i = (a_i \wedge b_i) \vee (a_i \wedge c_{i-1}) \vee (b_i \wedge c_{i-1}) \text{ pro } i \geq 0.$$

Definujeme-li pak bity s_j, $j = 0, 1, \ldots, n$ tak, že $s_0 = a_0 \oplus b_j$, $s_j = a_j \oplus b_j \oplus c_{j-1}$ pro $j = 1, \ldots, n-1$ a $s_n = c_{n-1}$ pak jejich hodnoty splňují výše uvedenou rovnici (S).

Našim cílem je tedy dokázat, že hodnoty $g_{k-1,j}$ vypočítané v algoritmu z Laboratoře jsou stejné jako odpovídající hodnoty c_j z Tvrzení 1. Tvrzení dává návod, jak hodnoty c_j (a tedy i hodnoty s_j) snadno spočítat. Algoritmus z Laboratoře ale je možno provést velmi rychle, pokud je možné výpočet provádět paralelně na větším počtu procesorů, zatímco postup podle Tvrzení paralelizovatelný není.

Symbolem c_j budeme ve zbytku kapitoly vždy označovat proměnné určené v Tvrzení 1 a budeme je nazývat *přenosy*.

Při procházce jsme hodnotu bloku označovali jednou ze tří hodnot $+$, $-$ a $<$. V digitálních obvodech se takřka výhradně používají dvouhodnotové proměnné a proto pro popis hodnot bloku musíme (trochu nadbytečně) použít dvojice bitů (g, p), které je kódují následujícím způsobem:

Bit g	Bit p	Hodnota bloku	Význam
1	1	$+$	Blok generuje přenos
1	0	$+$	Blok generuje přenos
0	1	$<$	Blok přenáší přenos
0	0	$-$	Blok absorbuje přenos

Pro zjednodušení výkladu zavedeme nejprve operaci $\bullet$, definovanou takto: jsou-li (G_1, P_1) a (G_2, P_2) dvě dvojice bitů, pak položíme

$$(G_1, P_1) \bullet (G_2, P_2) = (G_1 \vee (P_1 \wedge G_2), P_1 \wedge P_2).$$

Použitím této operace je možno výše uvedenou definici $g_{i,j}$ a $p_{i,j}$ pro $i \geq 0$ a $\xi(i,j) = 1$ (řádek 12 a 13 algoritmu) přepsat takto:

$$(g_{i,j}, p_{i,j}) = (g_{i-1,j}, p_{i-1,j}) \bullet (g_{i-1,\kappa(i,j)-1}, p_{i-1,\kappa(i,j)-1}).$$

Nyní na bity $g_{i,j}$ a $p_{i,j}$ na čas zapomeneme a formálním způsobem zopakujeme výklad z procházky o tom, jak se z hodnot bloků počítají přenosy.

Povšimněte si především toho, že pokud hodnoty dvou sousedících bloků B_1 a B_2 jsou vyjádřeny pomocí dvojic bitů (G_1, P_1), respektive (G_2, P_2), pak $(G_1, P_1) \bullet (G_2, P_2)$ udává hodnotu, kterou jsme v procházce přiřazovali bloku složenému z obou bloků B_1 a B_2.

Platí totiž:
Pokud B_1 generuje, pak je $G_1 = 1$ a pak ale levý bit výsledné dvojice je roven $G_1 \vee \ldots = 1 \vee \ldots = 1$.
Pokud blok B_1 absorbuje, je $G_1 = P_1 = 0$ a tedy výsledná dvojice je rovna
$(0 \vee (0 \wedge G_2), 0 \wedge P_2) = (0, 0)$.
Nakonec jestliže B_1 přenáší, pak $G_1 = 0$ a $P_1 = 1$ a tedy výsledná dvojice je rovna
$(0 \vee (1 \wedge G_2), 1 \wedge P_2) = (G_2, P_2)$.

Nejprve dokážeme tvrzení, které vám jistě bylo intuitivně zřejmé během procházky

Tvrzení 2 *[Labeled: AdditionAssociativity] Operace $\bullet$ je asociativní.*

Důkaz: Používajíce asociativitu a distributivitu operací $\wedge$ a $\vee$ dostáváme

$$(G_1, P_1) \bullet ((G_2, P_2) \bullet (G_3, P_3)) = (G_1, P_1) \bullet (G_2 \vee P_2 \wedge G_3, P_2 \wedge P_3) =$$

$$= (G_1 \vee P_1 \wedge (G_2 \vee P_2 \wedge G_3), P_1 \wedge (P_2 \wedge P_3)) = (G_1 \vee (P_1 \wedge G_2 \vee P_1 \wedge (P_2 \wedge G_3))), (P_1 \wedge P_2) \wedge P_3) =$$

$$= ((G_1 \vee P_1 \wedge G_2) \vee P_1 \wedge (P_2 \wedge G_3)), (P_1 \wedge P_2) \wedge P_3) = ((G_1 \vee P_1 \wedge G_2) \vee (P_1 \wedge P_2) \wedge G_3), (P_1 \wedge P_2) \wedge P_3) =$$

$$= (G_1 \vee P_1 \wedge G_2, P_1 \wedge P_2) \bullet (G_3, P_3) = ((G_1, P_1) \bullet (G_2, P_2)) \bullet (G_3, P_3).$$

♣

Definujme nyní

$$\mathcal{B}_{i,i} = (a_i \wedge b_i, a_i \vee b_i) \quad \text{pro } 0 \leq i < n,$$

$$\mathcal{B}_{i,j} = \mathcal{B}_{i,i} \bullet \cdots \bullet \mathcal{B}_{j,j} \quad \text{pro } 0 \leq j < i < n.$$

$\mathcal{B}_{i,j}$ popisuje hodnotu bloku mezi i-tým a j-tým sloupcem sčítačky tak, jak byla popisována při procházce. $\mathcal{B}_{i,i}$ je definováno přesně tak, aby odpovídalo hodnotám jednosloupcových bloků, jak jsme je viděli během procházky. Hodnoty širších bloků pak byly určeny jako kombinace hodnot jednosloupcových bloků v nich obsažených za využití operace skládání bloků (tedy $\bullet$). V důsledku asociativity operace $\bullet$ není nutné specifikovat, jak se výrazy $\mathcal{B}_{*,*}$ spolu kombinují dohromady v definici výrazu $\mathcal{B}_{i,j}$ pro $j < i$.

Následující tvrzení říká formálním způsobem, že c_i, přenos v i-tém sloupci, je roven 1 právě tehdy, když blok začínající ve i-tém sloupci a končící zcela vpravo (sloupec 0) generuje přenos (tedy levý bit v $\mathcal{B}_{i,0}$ je 1).

Tvrzení 3 *[Labeled: AdditionCarry] Pro každé $i = 0, \ldots, n-1$ je přenos c_i roven prvnímu (t.j. levému) bitu dvojice $\mathcal{B}_{i,0}$.*

Důkaz: Indukcí podle i. Pro $i = 0$ je $c_0 = a_0 \wedge b_0$ na základě definice c_0 v Tvrzení 1, ale $a_0 \wedge b_0$ je také hodnota levého bitu $\mathcal{B}_{0,0}$, viz jeho definice.

Je-li $i > 0$ a tvrzení platí pro $i - 1$, to znamená $\mathcal{B}_{i-1,0} = (c_{i-1}, p)$ pro nějaké p, pak

$$\mathcal{B}_{i,0} = \mathcal{B}_{i,i} \bullet \mathcal{B}_{i-1,0} = (a_i \wedge b_i, a_i \vee b_i) \bullet (c_{i-1}, p) =$$

$$= (a_i \wedge b_i \vee (a_i \vee b_i) \wedge c_{i-1}), (a_i \vee b_i) \wedge p),$$

a přitom

$$a_i \wedge b_i \vee (a_i \vee b_i) \wedge c_{i-1}) =$$

$$= a_i \wedge b_i \vee a_i \wedge c_{i-1} \vee b_i \wedge c_{i-1} = c_i.$$

♣

Nyní se podíváme blíže na funkce ξ a κ definované v Laboratoři. Jejich podivně vyhlížející definice jsou ve skutečnosti velmi přirozeně vázány ke struktuře sčítačky.

Otevřte si poslední scénu procházky, věnovanou číslování. Předně si ověřte, že $\xi(i,j) = 1$ právě tehdy, když hradlo v řádku označeném číslem i a ve sloupci označeném j je žluté, ted kombinuje výstupy dvou hradel z předchozího řádku, zatímco $\xi(i,j) = 1$ právě tehdy, když je takové hradlo růžové a pouze předává dále výstup hradla nad ním.

Dále si klikněte na libovolné hradlo. Všechny bloky, na nichž závisí výstupní hodnota zvoleného hradla, zčervenají. Pokud se zvolí hradlo v řádku označeném číslem i a ve sloupci j, pak vidíme, že hodnota závisí na jednosloupcových blocích mezi sloupci s pořadovými čísly j (nalevo) a $\kappa(i,j)$ (napravo). Ověřte si, že zvláštní definice κ je ve skutečnosti vynucena touto vlastností.

Z toho také plyne, že jestliže zvolené hradlo H v řádku i a sloupci j má dva dvoubitové vstupy, pak levý z nich je napojen na hradlo v řádku $i-1$ a sloupci j, které odpovídá bloku pokrývajícímu sloupce mezi j a $\kappa(i-1,j)$. Blok, na který je tedy hradlo H napojeno svým pravým vstupem musí k bloku mezi j a $\kappa(i-1,j)$ zprava přiléhat a proto musí začínat v sloupci $\kappa(i-1,j)-1$. To vysvětluje zvláštní volbu indexů v řádcích 12 a 13 v programu.

Nyní dokážeme následující dvě důležitá pomocné tvrzení o funkci κ. Přitom si uvědomíme, že pro libovolné celé nezáporné číslo x je x **div** $1 = x$ a tedy také x **mod** $1 = x - x$ **div** $1 = x - x = 0$.

Tvrzení 4 *[Labeled: AdditionKappa] Nechť i, j jsou celá čísla taková, že $0 \leq i < k-1$ a $0 \leq j < n$. Pak $\kappa(-1,j) = j$,*
je-li $\xi(i,j) = 1$, pak $\kappa(i,j) = \kappa(i-1,j) - 2^i$,
je-li $\xi(i,j) = 0$, pak $\kappa(i,j) = \kappa(i-1,j)$, a
$\kappa(k-1,j) = 0$.

Důkaz: Podle definice je $\kappa(-1,j) = j - j$ **mod** $2^0 = j - j$ **mod** $1 = j - 0 = j$.

Nechť nyní $i \geq 0$. Pro jistá celá čísla r, q taková, že $0 \leq r$ a $0 \leq q < 2^i$, a bit d platí $j = r2^{i+1} + d2^i + q$.

Pak ale

$$\xi(i,j) = (j \textbf{ div } 2^i) \textbf{ mod } 2 = (2r + d) \textbf{ mod } 2 = d,$$

$$j = r2^{i+1} + \xi(i,j)2^i + q,$$

a proto

$$j \textbf{ mod } 2^{i+1} = \xi(i,j)2^i + q, \quad \kappa(i,j) = j - j \textbf{ mod } 2^{i+1} = r2^{i+1},$$

$$j \textbf{ mod } 2^i = q, \quad \kappa(i-1,j) = j - j \textbf{ mod } 2^i = r2^{i+1} + \xi(i,j)2^i$$

a tedy

$$\kappa(i,j) = r2^{i+1} + \xi(i,j)2^i - \xi(i,j)2^i = \kappa(i-1,j) - \xi(i,j)2^i.$$

Nakonec $j < n = 2^k$, takže j **div** $2^k = 0$, j **mod** $2^k = j$ a $\kappa(k-1,j) = j - j$ **mod** $2^k = j - j = 0$.

♣

Tvrzení 5 *Nechť i, j jsou celá čísla taková, že $0 \leq i < k$, $0 \leq j < n$ a $\xi(i,j) = 1$. Pak také $\kappa(i-1, \kappa(i-1,j)-1) = \kappa(i,j)$.*

Poznámka: V řeči číslování z poslední scény Procházky toto tvrzení říká: je-li hradlo H v řádku i a sloupci j žluté, a označíme-li jako H' hradlo, na které je H napojeno svým pravým vstupem, a které leží v řádku $i-1$ a sloupci $\kappa(i,j)-1$, pak bloky odpovídající hradlům H a H' mají svůj pravý okraj ve stejnm sloupci. Ověřte si to ve scéně o číslování klepnutím na některé žluté hradlo (H) a pak na hradlo, na které je připojen jeho pravý vstup (H') a sledováním, který z červených boxů v hladině -1 je nejvíce napravo.
Důkaz: Označme $K = \kappa(i-1,j)$. Pak K je dělitelné číslem 2^i, viz definice $\kappa(i,j)$, ale $K-1$ je menší než K a proto největší číslo menší nebo rovné $K-1$ a dělitelné číslem 2^i je nejbližší nižší celočíselný násobek 2^i, tedy číslo $K-2^i$, a proto na základě Tvrzení 4 je

$$\kappa(i-1, \kappa(i-1,j)-1) = \kappa(i-1, K-1) = K - 2^i = \kappa(i-1,j) - 2^i = \kappa(i,j).$$

♣

Nyní se vrátíme k dvojicím bitů $(g_{i,j}, p_{i,j})$ definovaným programem v Laboratoři a ukážeme, že udávají hodnotu bloku, který začíná v j-tém sloupci a končí v $\kappa(i,j)$-tém sloupci. Tuto hodnotu jsme si označili jako $\mathcal{B}_{j,\kappa(i,j)}$.

Tvrzení 6 *[Labeled: AdditionPyramide] $(g_{i,j}, p_{i,j}) = \mathcal{B}_{j,\kappa(i,j)}$ pro $-1 \leq i < k$ a $0 \leq j < n$.*

Důkaz: Indukcí podle i. Podle Tvrzení 4 je $\kappa(-1,j) = j$ a podle definic $(g_{-1,j}, p_{-1,j})$ a $\mathcal{B}_{j,j}$ je

$$(g_{-1,j}, p_{-1,j}) = (a_j \wedge b_j, a_j \vee b_j) = \mathcal{B}_{j,j} = \mathcal{B}_{j,\kappa(-1,j)}.$$

Nechť nyní $0 \leq i < k$ a nechť tvrzení platí pro $i-1$ a libovolné j.
Jestliže $\xi(i,j) = 0$, pak

$$(g_{i,j}, p_{i,j}) = (g_{i-1,j}, p_{i-1,j}) = \mathcal{B}_{j,\kappa(i-1,j)} = \mathcal{B}_{j,\kappa(i,j)},$$

jestliže $\xi(i,j) = 1$, pak podle Tvrzení 5

$$(g_{i,j}, p_{i,j}) = (g_{i-1,j}, p_{i-1,j}) \bullet (g_{i-1,\kappa(i,j)-1}, p_{i-1,\kappa(i,j)-1}) =$$

$$= \mathcal{B}_{j,\kappa(i-1,j)} \bullet \mathcal{B}_{\kappa(i,j)-1,\kappa(i-1,\kappa(i,j)-1)} = \mathcal{B}_{j,\kappa(i-1,j)} \bullet \mathcal{B}_{\kappa(i,j)-1,\kappa(i,j)} = \mathcal{B}_{j,\kappa(i,j)},$$

neboť z definice $\mathcal{B}_{*,*}$ a asociativity $\bullet$ plyne pro libovoln $r \geq s > t$

$$\mathcal{B}_{r,s} \bullet \mathcal{B}_{s-1,t} = (\mathcal{B}_{r,r} \bullet \cdots \mathcal{B}_{s,s}) \bullet)(\mathcal{B}_{s-1,s-1} \bullet \mathcal{B}_{t,t}) = \mathcal{B}_{r,t}.$$

♣

Kombinace tohoto tvrzení a Tvrzení 3 ihned říká, že hodnoty proměnných $g_{k-1,j}$, definovaných v Laboratoři, dávají hledané přenosy c_j.

Tvrzení 7 *[Labeled: AdditionCarryFromBlock] $c_j = g_{k-1,j}$ pro každé celé číslo j takové, že $0 \leq j < n$.*

Z toho již okamžitě na základě Tvrzení 1 plyne

Věta 8 *Algoritmus uvedený v Laboratoři sečte dvě binární čísla s délkou binárního zápisu n v paralelním čase $O(\log n)$ s provedení $O(n \log n)$ booleovských operací.*