5. Paralelní sčítání, bitonické třídění *(zapsal: Petr Jankovský)*

Minule jsme si zavedli paralelní výpočetní model, ve kterém si nyní něco naprogramujeme ...

Sčítání binárních čísel

Mějme dvě čísla x a y zapsané ve dvojkové soustavě. Jejich číslice označme $x_{n-1} \dots x_0$ a $y_{n-1} \dots y_0$, kde i-tý řád má váhu 2^i. Nyní budeme chtít tato čísla sečíst.

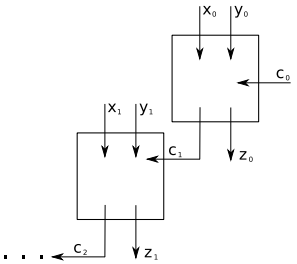
K tomuto účelu se ihned nabízí použít starý dobrý „školní algoritmus“, který funguje ve dvojkové soustavě stejně dobře jako v desítkové. Zkrátka sčítáme čísla zprava doleva. Vždy sečteme příslušné číslice pod sebou a přičteme přenos z nižšího řádu. Tím dostaneme jednu číslici výsledku a přenos do vyššího řádu. Formálně bychom to mohli zapsat třeba takto:

$$z_i = x_i \oplus y_i \oplus c_i,$$

kde z_i je i-tá číslice součtu, $\oplus$ značí operaci XOR (součet modulo 2) a c_i je *přenos* z $(i-1)$-ního řádu do i-tého. Přenos přitom nastane tehdy, pokud se nám potkají dvě jedničky pod sebou, nebo když se vyskytne alespoň jedna jednička a k tomu přenos z nižšího řádu. Jinými slovy tehdy, když ze tří xorovaných číslic jsou alespoň dvě jedničky (pomocí obvodu pro majoritu z minulé přednášky lehce zkonstruujeme):

$$\begin{aligned} c_0 &= 0, \\ c_{i+1} &= (x_i \ \& \ y_i) \vee (x_i \ \& \ c_i) \vee (y_i \ \& \ c_i). \end{aligned}$$

Takovéto sčítání sice perfektně funguje, nicméně je bohužel poměrně pomalé. Pokud bychom stavěli hradlovou síť podle tohoto předpisu, byla by složená z nějakých podsítí („krabiček“), které budou mít na vstupu x_i, y_i a c_i a jejich výstupem bude z_i a c_{i+1}.



Sčítání školním algoritmem.

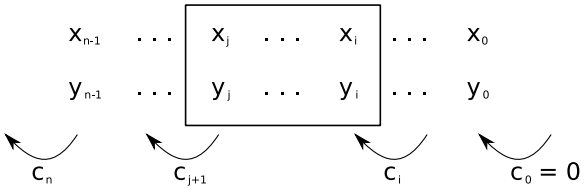
Všimněme si, že každá krabička závisí na výstupu té předcházející. Jednotlivé krabičky tedy musí určitě ležet na různých hladinách. Celkově bychom museli použít $\Theta(n)$ hladin a jelikož je každá krabička konstantně velká, také $\Theta(n)$ hradel. To dává lineární časovou i prostorovou složitost, čili oproti sekvenčnímu algoritmu jsme si nepomohli.

Zamysleme se nad tím, jak by se proces sčítání mohl zrychlit.

Přenosy v blocích

Jediné, co nás při sčítání brzdí, jsou přenosy mezi jednotlivými řády. Každý řád, aby vydal součet, musí počkat na to, až dopočítají všechny předcházející řády. Teprve pak se totiž dozví přenos. Kdybychom ovšem přenosy dokázali spočítat nějakým způsobem paralelně, máme vyhráno. Jakmile známe všechny přenosy, součet už zvládneme dopočítat na konstantně mnoho hladin – tedy v konstantním čase.

Podívejme se na libovolný *blok* v našem součtu. Tak budeme říkat číslům $x_j \dots x_i$ a $y_j \dots y_i$ v nějakém intervalu indexů $\langle i, j \rangle$. Přenos c_{j+1} vystupující z tohoto bloku závisí mimo hodnot sčítanců už pouze na přenosu c_i, který do bloku vstupuje.



Blok součtu.

Z tohoto pohledu se můžeme na blok také dívat jako na nějakou funkci, která dostane jednobitový vstup a vydá jednobitový výstup. To je nám příjemné, neboť takových funkcí existují jenom čtyři typy:

1. $f(x) = 0$, (0)
2. $f(x) = 1$, (1)
3. $f(x) = x$, ($<$ – kopírování)
4. $f(x) = \neg x$.

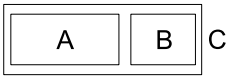
Jak se za chvíli ukáže, poslední případ, kdy by nějaký blok předával opačný přenos, než do něj vstupuje, navíc nikdy nemůže nastat. Pojďme si to rozmyslet. Jednobitové bloky se chovají velice jednoduše:

0	0	1	1
0	1	0	1
0	<	<	1

Tabulka triviálních bloků.

Z prvního bloku evidentně vždy vyleze 0, ať do něj vstoupí jakýkoli přenos. Poslední blok naopak sám o sobě přenos vytváří, ať již do něj vleze jakýkoliv. Bloky prostřední se chovají stejně, a to tak, že samy o sobě žádný přenos nevytvoří, ale pokud do nich nějaký přijde, tak také odejde.

Mějme nyní nějaký větší blok C složený ze dvou menších podbloků A a B, jejichž chování už známe. Z toho můžeme odvodit, jak se chová celý blok:



A \ B	0	1	<
0	0	0	0
1	1	1	1
<	0	1	<

Skládání chování bloků.

Pokud vyšší blok přenos pohlcuje, pak ať se už nižší blok chová jakkoli, složení těchto bloků musí vždy pohlcovat. V prvním řádku tabulky jsou tudíž nuly. Analogicky, pokud vyšší blok generuje přenos, tak ten nižší na tom nic nezmění. V druhém řádku tabulky jsou tedy samé jedničky. Zajímavější případ nastává, pokud vyšší blok kopíruje – tehdy záleží čistě na chování nižšího bloku.

Všimněme si, že skládání chování bloků je vlastně úplně obyčejné skládání funkcí. Nyní bychom mohli prohlásit, že budeme počítat nad tříprvkovou abecedou, a že celou tabulku dokážeme spočítat jedním jediným hradlem. Pojďmě si však rozmyslet, jak bychom takovouto věc popsali čistě binárně. Jak tedy tyto tři stavy popisovat pouze několika bity?

Evidentně nám k tomuto binárnímu zakódování tří stavů budou stačit bity dva. Označme si je jako p a q. Tato dvojice může nabývat hned čtyř možných hodnot, kterým přiřadíme tři možná chování bloku. Toto kódování můžeme zvolit zcela libovolně, ale pokud si ho zvolíme šikovně, ušetříme si dále práci při kompozici. Zvolme si tedy kódování takto:

- $(1, *) =<$,
- $(0, 0) = 0$,
- $(0, 1) = 1$

Tomu, že blok kopíruje, odpovídá dvojice $p = 1$; $q =$ *cokoliv*. V ostatních případech bude p nulové a q nám bude říkat, co je na výstupu příslušného bloku. Jinými slovy $p = 0$ znamená, že funkce je konstanta, přičemž q říká jaká; naproti tomu $p = 1$ znamená, že funkce je identita, ať už je q cokoli.

Pojďme si nyní ukázat, jak bude celé skládání bloků vypadat. Rozmysleme si, kdy je p celého dvojbloku jedničkové, tedy kdy celý dvojblok kopíruje. To nastane tehdy, pokud kopírují obě jeho části, a tedy $p = p_a \;\&\; p_b$. Dále q bude rovno jedničce, pokud $q = (\neg p_a \;\&\; q_a) \vee (p_a \;\&\; q_b)$.

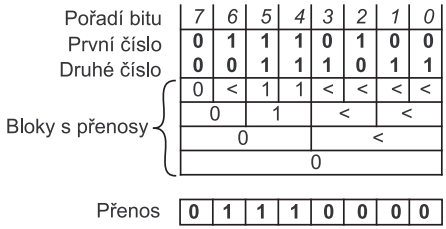
Skládání chování bloků lze tedy popsat buď ternárně – tabulkou, ale lze to i binárně výše uvedeným předpisem.

Nyní si tedy můžeme dopředu vypočítat chování bloku velikosti jedna, poté z nich skládáním bloků velikosti dva, dál velikosti čtyři, osm, atd ...

Paralelní sčítání

Paralelní algoritmus na sčítání už zkonstruujeme poměrně snadno. Bez újmy na obecnosti budeme předpokládat, že počet bitů vstupních čísel je mocnina dvojky, jinak si vstup doplníme nulami, což výsedný čas běhu algoritmu zhorší maximálně konstanta-krát.

1. Spočteme chování bloků velikosti 1. ($\mathcal{O}(1)$ hladin)
2. Postupně počítáme chování bloků[⟨1⟩] velikosti 2, 4, 8, …, 2^k. ($\mathcal{O}(\log n)$ hladin, na nichž se skládají bloky)
3. $c_0 \leftarrow 0$ (přenos do nejnižšího řádu je vždy 0)
4. Určíme c_n podle c_0 a chování (jediného) bloku velikosti n.
5. Postupně dopočítáme přenosy na hranicích dělitelných 2^k „zahušťováním“: jakmile víme c_{2^k}, můžeme dopočítat $c_{2^k+2^{k-1}}$ podle chování bloku $\langle 2^k, 2^k + 2^{k-1}\rangle$. ($\mathcal{O}(\log n)$ hladin, na nichž se dosazuje)
6. Sečteme: $\forall i : z_i = x_i \oplus y_i \oplus c_i$.



Výpočet přenosu.

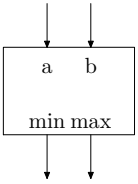
Algoritmus pracuje v čase $\mathcal{O}(\log n)$. Hradel je použito lineárně: na jednotlivých hladinách kroku 2 počet hradel exponenciálně klesá od n k 1, na hladinách kroku 5 exponenciálně stoupá od 1 k n, takže dohromady se sečte na $\Theta(n)$.

Třídění

Nyní se podíváme na druhý problém, a to na problém třídění. Již známe poměrně efektivní sekvenční algoritmy, které dovedou třídit v čase $\mathcal{O}(n \log n)$. Byli bychom jistě rádi, kdybychom to zvládli ještě rychleji. Pojďme se podívat, zda by nám v tom nepomohlo problém paralelizovat.

Budeme při tom pracovat ve výpočetním modelu, kterému se říká komparátorová síť. Ta je postavená z hradel, kterým se říká komparátory.

[⟨1⟩] myslíme „přirozeně zarovnané“ bloky, tedy takové, jejichž poloha je násobkem velikosti



Komparátor

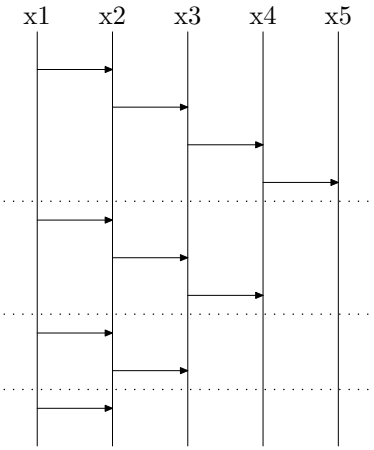
Definice: *Komparátorová síť* je kombinační obvod, jehož hradla jsou komparátory.

Komparátor umí porovnat dvě hodnoty a rozhodnout, která z nich je větší a která menší. Nevrací však booleovský výsledek jako běžné hradlo, ale má dva výstupy, přičemž na jednom vrátí menší ze vstupních hodnot a na druhém větší.

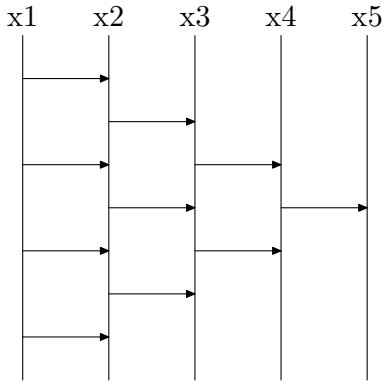
Výstupy komparátorů se nevětví. Nemůžeme tedy jeden výstup „rozdvojit" a připojit ho na dva vstupy. (Větvení by dokonce ani nemělo smysl, protože zatímco rozdvojit bychom mohli, sloučit už ne. Pokud tedy chceme, aby síť měla n vstupů i n výstupů, rozdvojení stejně nesmíme provést, i kdybychom jej měli povolené.)

Příklad: *Bubble sort*

Obrázek Bubble 1 ilustruje použití komparátorů pro třídění Bubble sortem. Šipky představují jednotlivé komparátory. Výpočet však ještě můžeme vylepšit.



Bubble 1



Bubble 2

Snažíme se výpočet co nejvíce paralelizovat (viz obrázek Bubble 2). Jak je vidět, komparátory na sebe nemusejí čekat. Tím můžeme výpočet urychlit a místo času $\Theta(n^2)$ docílit časové složitosti $\Theta(n)$. V obou případech je zachován kvadratický prostor.

Nyní si ukážeme ještě rychlejší třídící algoritmus. Půjdeme na něj však trochu „od lesa". Nejdříve vymyslíme síť, která bude umět třídit jenom něco - totiž bitonické posloupnosti.

Definice: Řekneme, že posloupnost $x_0, \ldots, x_{n-1}$ je *čistě bitonická* právě tehdy, když pro nějaké $x_k, k \in \{0, \ldots, n-1\}$ platí, že všechny prvky před ním (včetně jeho samotného) tvoří rostoucí poslopnost, kdežto prvky stojící za ním tvoří poslopnost klesající. Formálně zapsáno musí platit, že:

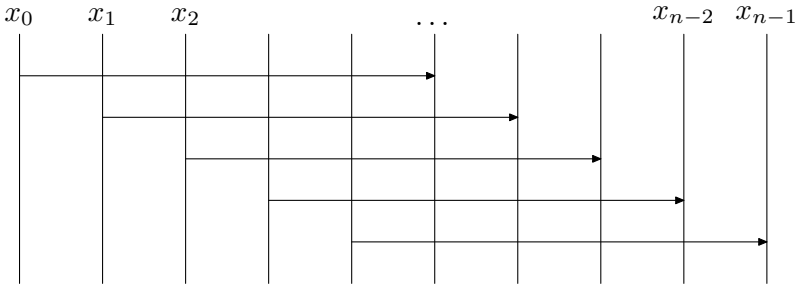
$$x_0 \leq x_1 \leq \ldots \leq x_k \geq x_{k+1} \geq \ldots \geq x_{n-1}.$$

Definice: Posloupnost $x_0 \ldots x_{n-1}$ je *bitonická*, právě když $\exists\ j \in \{0, \ldots, n-1\}$, pro které je rotace původní posloupnosti o j prvků, tedy posloupnost

$$x_j, x_{(j+1) \bmod n}, \ldots, x_{(j+n-1) \bmod n},$$

čistě bitonická.

Definice: *Separátor* S_n je síť, ve které jsou vždy i-tý a $(i + n/2)$-tý prvek vstupu (pro $i = 0, \ldots, n/2 - 1$) propojeny komparátorem. Minimum se pak stane i-tým, maximum $(i + n/2)$-tým prvkem výstupu.



$$(y_i, y_{i+n/2}) = CMP(x_i, x_{i+n/2})$$

Lemma: Pokud separátor dostane na vstupu bitonickou posloupnost, pak jeho výstup $y_0, \ldots, y_{n-1}$ splňuje:

(i) $y_0, \ldots, y_{n/2-1}$ a $y_{n/2}, \ldots, y_{n-1}$ jsou bitonické posloupnosti,

(ii) Pro všechna $i, j < n/2$ platí $y_i < y_{j+n/2}$.

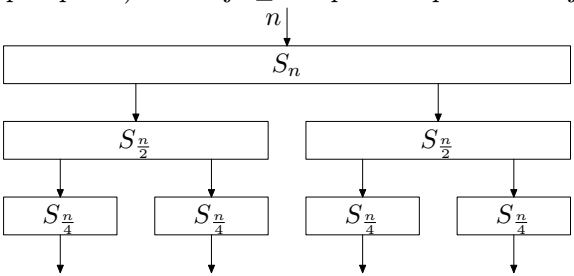
Separátor nám tedy jednu bitonickou posloupnost na vstupu rozdělí na dvě bitonické posloupnosti, přičemž navíc každý prvek první posloupnosti $(y_0, \ldots, y_{n/2-1})$ je menší nebo roven prvkům druhé posloupnosti $(y_{n/2}, \ldots, y_{n-1})$.

Než přistoupíme k důkazu lemmatu, ukažme si, k čemu se nám bude hodit.

Definice: *Bitonická třídička* B_n je obvod sestavený ze separátorů, který dostane-li na vstupu bitonickou posloupnost délky n (BÚNO konstruujeme třídičku pro $n = 2^k$), vydá setříděnou zadanou posloupnost délky n.

Třídička dostane na vstupu bitonickou posloupnost. Separátor S_n ji pak dle lemmatu rozdělí na dvě bitonické posloupnosti, kdy je každý prvek z první posloupnosti menší než libovolný prvek z druhé. Tyto poloviny pak další separátory rozdělí

na čtvrtiny, …, až na konci zbudou pouze jednoduché posloupnosti délky jedna (zjevně setříděné), které mezi sebou mají požadovanou nerovnost – tedy každá posloupnost (nebo spíše prvek) nalevo je $\leq$ než prvek napravo od něj.



Bitonická třídička B_n

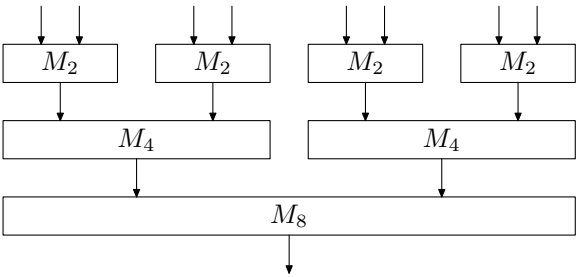
Jak je vidět, bitonická třídička nám libovolnou bitonickou posloupnost délky n setřídí na $\Theta(\log n)$ hladin.

Nyní se dá odvodit, že pokud umíme třídit bitonické posloupnosti, umíme setřídit všechno. Vzpomeňme si na třídění sléváním – Merge sort. To funguje tak, že začne s jednoprvkovými posloupnostmi, které jsou evidentně setříděné, a poté vždy dvě setříděné posloupnosti slévá do jedné. Kdybychom nyní uměli paralelně slévat, mohli bychom vytvořit i paralelní Merge sort. Jinými slovy, potřebujeme dvě rostoucí posloupnosti nějak efektivně slít do jedné setříděné. Uvědomme si, že to zvládneme jednoduše – stačí druhou z posloupností obrátit a „přilepit“ za první, čímž vznikne bitonická posloupnost, kterou poté můžeme setřídit naší bitonickou třídičkou.

Příklad: *Merge sort*

Bitonická třídička se tedy dá použít ke slévání setříděných posloupností. Ukažme si, jak s její pomocí sestavíme součástky *slévačky* M_n:

Setříděné posloupnosti $x_0, \ldots, x_{n-1}$ a $y_0, \ldots, y_{n-1}$ spojíme do jedné bitonické posloupnosti $x_0, \ldots, x_{n-1}, y_{n-1}, \ldots, y_0$. Z této posloupnosti vytvoříme pomocí bitonické třídičky B_{2n} setříděnou posloupnost. Vytvoříme tedy blok M_{2n}, který se ovšem sestává de facto pouze z bloku B_{2n}, jehož druhá polovina vstupů je zapojena v obráceném pořadí.



Paralelní MergeSort.

Nyní se pokusme odhadnout časovou složitost. Náš MergeSort bude mít řádově hloubku bloků $\log n$. V každém bloku M_n je navíc ukryta bitonická třídička s taktéž logaritmickou hloubkou. Celková hloubka tedy bude $\log 2 + \log 4 + \ldots + \log 2^k + \ldots + \log n$. Po sečtení nakonec dostáváme výslednou časovou složitost $\Theta(\log^2 n)$.

Dodejme ještě, že existuje i třídicí algoritmus, kterému stačí jen $\mathcal{O}(\log n)$ hladin. Jeho multiplikativní konstanta je však příliš veliká, takže je v praxi nepoužitelný.

Vraťme se nyní k důkazu lemmatu, který jsme na předminulé stránce vynechali.

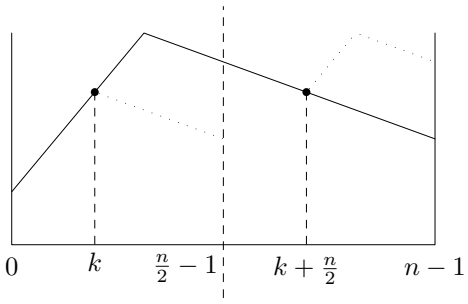
Důkaz Lemmatu:

(i) Nejprve nahlédneme, že lemma platí, je-li vstupem čistě bitonická posloupnost. Dále BÚNO předpokládejme, že vrchol posloupnosti je v první polovině (kdyby byl vrchol za polovinou, stačilo by zrcadlově obrátit posloupnost i komparátory a řešili bychom stejný problém). Nyní si definujme $k := \min j : x_j > x_{j+n/2}$. (Pokud by takové k neexistovalo, znamenalo by to, že vstupní posloupnost je monotónní. Separátor by tedy nic nedělal a pouze zkopíroval vstup na výstup, což jistě lemma splňuje.) Nyní si všiměme, že jakmile jednou začne platit, že prvek na levé straně je menší než na pravé, bude nám tato relace platit až do konce. Označme vrchol vstupní posloupnosti jako x_m. Pak k bude jistě menší než m a $k + n/2$ bude větší než m. Mezi k a m je tedy vstupní posloupnost neklesající, mezi $k + n/2$ a $n - 1$ nerostoucí.

Do pozice k tedy separátor bude pouze kopírovat vstup na výstup, od pozice k dál už jen prohazuje. Pro každé i, $(k \leq i \leq n/2 - 1)$ se prvky x_i a $x_{i+n/2}$ prohodí. Úsek mezi k a $n/2 - 1$ tedy nahradíme nerostoucí posloupností, první polovina výstupu tedy bude (dokonce čistě) bitonická. Podobně úsek $k+n/2$ až $n-1$ nahradíme čistě bitonickou posloupností. Obě poloviny tedy budou bitonické.

Dostaneme-li na vstupu obecnou bitonickou posloupnost, představíme si, že je to čistě bitonická posloupnost zrotovaná o r prvků (BÚNO doprava). Zjistíme, že v komparátorech se porovnávají tytéž prvky, jako kdyby zrotovaná nebyla. Výstup se od výstupu čistě bitonické posloupnosti zrotovaného o r bude lišit prohozením úseků x_0 až x_{r-1} a $x_{n/2}$ až $x_{n/2+r-1}$. Obě výstupní posloupnosti tedy budou zrotované o r prvků, ale na jejich bitoničnosti se nic nezmění.

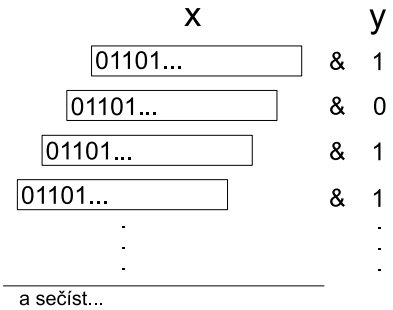
(ii) Z důkazu (i) pro čistě bitonickou posloupnost víme, že $y_0 \ldots y_{n/2-1}$ je čistě bitonická a bude rovna $x_0 \ldots x_{k-1}, x_{k+n/2} \ldots x_{n-1}$ pro vhodné k a navíc bude mít maximum v x_{k-1} nebo $x_k + n/2$. Mezi těmito body ovšem ve vstupní posloupnosti určitě neležel žádný x_i menší než $x_k - 1$ nebo $x_k + n/2$ (jak je vidět z obrázku) a posloupnost $x_k \ldots x_{k-1+n/2}$ je rovna $y_{n/2} \ldots y_{n-1}$. Pro obecné bitonické posloupnosti ukážeme stejně jako v (i). ♡



posloupnost prohozená separátorem

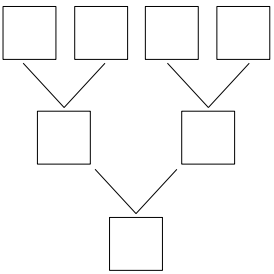
Paralelní násobení

Podobně jako u sčítání si vzpomeneme na školní algoritmus – tentokráte však pro násobení. To fungovalo tak, že jsme si jedno ze dvou binárních čísel na vstupu (říkejme mu třeba x) n-krát posouvali. Tam kde pak byly v čísle y jedničky, příslušné kopie x jsme sečetli. Jinými slovy tedy násobení umíme převést na nějaké posuny (ty lze realizovat pouze „předrátováním“ – nic nás nestojí), násobení jedním bitem (což je AND) a nakonec potřebujeme výsledných n čísel sečíst.



Školní sčítání.

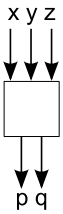
Jak nyní sečíst n n-bitových čísel..? Nabízí se využít osvědčený „stromeček“ – sčítat dvojice čísel, výsledky pak opět po dvojicích sečíst, až na konci vyjde jediný výsledek.



Stromeček

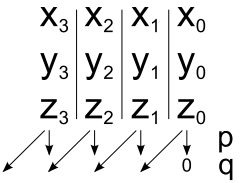
Toto řešení by však vedlo na časovou složitost $\Theta(\log^2 n)$. To je sice dle našich měřítek docela efektivní, ale překvapivě to jde ještě lépe – totiž na $\Theta(\log n)$ hladin. Této složitosti dosáhneme malým trikem.

Vymyslíme obvod konstantní hloubky, který na vstupu dostane tři čísla. Odpoví pak dvěma čísly takovými, že budou mít stejný součet jako původní tři čísla. Jinými slovy pomocí tohoto obvodu budeme umět sečtení tří čísel převést na sečtení dvou čísel.



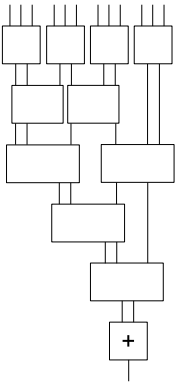
$$x + y + z = p + q$$

Všimněme si, že když sčítáme tři bity, může být přenos do vyššího řádu nula či jednička. Vezmeme si tedy bity x_i, y_i, z_i a ty sečteme. To nám dá dvoubitový výsledek, přičemž nižší bit z tohoto výsledku pošleme do čísla p, vyšší do čísla q.



Redukování sčítání

Toto zredukování sčítání nám nyní umožní opět stavit strom, byť o maličko složitější.



Složitější stromeček

Pokud jsme měli na začátku n čísel, po první redukci nám jich zbývá jen $2/3 \cdot n$ a obecně v k-té hladiné $(2/3)^k \cdot n$. Znamená to, že čísel nám ubývá exponenciálně, takže počet hladin bude logaritmický. Redukující obvod je při tom jen konstantně hluboký, takže celé redukování zvládneme v čase $\Theta(\log n)$. Na konci Složitějšího stromečku pak máme umístěnou jednu obyčejnou sčítačku, která zbývající dvě čísla sečte v logaritmickém čase.

Sečtení všech n čísel tedy zabere $\Theta(\log n)$ hladin.

Když se nyní vrátíme k násobení, zbývá nám vyřešit posouvání a ANDování. Uvědomme si, že to je plně paralelení a zvládneme ho za konstantně mnoho hladin. Celé násobení tedy zvládneme v logaritmickém čase.