1. Hradlové sítě

Pomocí počítačů řešíme stále složitější a rozsáhlejší úlohy a potřebujeme k tomu čím dál víc výpočetního výkonu. Rychlost a kapacita počítačů zatím rostla exponenciálně, takže se zdá, že stačí chvíli počkat. Jenže tento růst se určitě někdy zastaví – například není jasné, jestli půjde vyrábět transistory menší než jeden atom.

Jak si tedy s obrovskými daty poradíme? Jedna z lákavých možností je prostě do výpočtu zapřáhnout více než jeden procesor. Ostatně, vícejádrové procesory, které dneska najdeme ve svých stolních počítačích, nejsou nic jiného než víceprocesorový systém na jednom čipu.

Nabízí se tedy obtížnou úlohu rozdělit na několik částí, nechat každý procesor (či jádro) počítat jednu z částí a nakonec jejich výsledky spojit dohromady. To se snadno řekne, ale s výjimkou triviálních úloh už obtížněji provede.

Pojďme se podívat na několik zajímavých paralelních algoritmů. Abychom se nemuseli zabývat detaily hardwaru konkrétního víceprocesorového počítače, zavedeme si poměrně abstraktní výpočetní model, totiž hradlové sítě. Tento model je daleko paralelnější než skutečný počítač, ale přesto se techniky, které si ukážeme, dají snadno využít i prakticky. Konec konců sama vnitřní architektura procesorů se našemu modelu velmi podobá.

1.1. Hradlové sítě

Hradlové sítě jsou tvořeny navzájem propojenými *hradly*. Každé hradlo přitom počítá nějakou (obecně libovolnou) funkci $\Sigma^k \to \Sigma$, kde Σ je konečná abeceda (stejná pro celou síť) a k přirozené číslo (počet vstupů hradla, jinak též jeho *arita*).

Příklad: Často studujeme hradla *booleovská* (pracující nad abecedou $\Sigma = \{0, 1\}$). Ta počítají jednotlivé logické funkce, mezi nejběžnější patří:

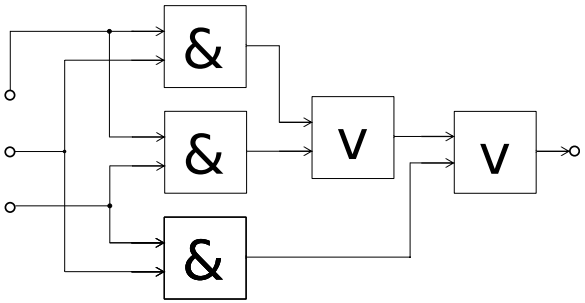
- nulární funkce: to jsou konstanty (FALSE $= 0$, TRUE $= 1$),
- unární funkce: identita a negace (NOT, $\neg$),
- binární funkce: logický součin (AND, $\&$), součet (OR, $\vee$), ...

Propojením hradel pak vznikne *hradlová síť*. Než vyřkneme formální definici, pojďme se podívat na příklad jedné takové sítě:

Naše síť má tři vstupy, pět booleovských hradel a jeden výstup. Na výstupu je přitom jednička právě tehdy, jsou-li jedničky přítomny na alespoň dvou vstupech. Jinými slovy vrací *majoritu* ze vstupů, tedy hodnotu, která převažuje.

Obecně každá hradlová síť má nějaké vstupy, hradla a výstupy. Každý vstup hradla je přitom připojen buďto na některý ze vstupů sítě nebo na výstup jiného hradla. Výstupy hradel mohou být propojeny na vstupy dalších hradel (mohou se větvit), nebo na výstupy sítě. Libovolně, jen nesmíme vytvářet cykly.

Nyní totéž formálněji:



Obr. 1.1: Hradlová síť – třívstupová verze funkce *majorita*

Definice: *Hradlová síť* je určena:

- *abecedou* Σ, což je nějaká konečná množina symbolů;
- po dvou disjunktními konečnými množinami I (*vstupy*), O (*výstupy*) a H (*hradla*);
- acyklickým orientovaným multigrafem (V, E), kde $V = I \cup O \cup H$;[1]
- zobrazením F, které každému hradlu $h \in H$ přiřadí nějakou funkci $F(h) : \Sigma^{a(h)} \to \Sigma$, což je funkce, kterou toto hradlo vykonává. Číslu $a(h)$ říkáme *arita* hradla h;
- zobrazením $z : E \to \mathbb{N}$, jež o hranách vedoucích do hradel říká, kolikátému argumentu funkce odpovídají;[2]

Přitom jsou splněny následující podmínky:

- Do vstupů nevedou žádné hrany.
- Z výstupů nevedou žádné hrany a do každého výstupu vede právě jedna hrana.
- Do každého hradla vede tolik hran, kolik je jeho arita.
- Všechny vstupy hradel jsou zapojeny. Tedy pro každé hradlo h a každý jeho vstup $j \in \{1, \ldots, a(h)\}$ existuje právě jedna hrana e, která vede do hradla h a $z(e) = j$.

Poznámka: Někdy se hradlovým sítím také říká *kombinační obvody* a pokud pracují nad abecedou $\Sigma = \{0, 1\}$, pak *booleovské obvody*.

Definice: *Výpočet sítě* postupně přiřazuje hodnoty z abecedy Σ vrcholům grafu. Výpočet probíhá po *taktech.* V nultém taktu jsou definovány pouze hodnoty na vstupech sítě a v hradlech arity 0 (konstanty). V každém dalším taktu pak ohodnotíme vrcholy, jejichž všechny vstupní hrany vedou z vrcholů s již definovanou hodnotou.

[1] Proč potřebujeme multigraf? Například chceme-li výstup jednoho hradla připojit současně na více různých vstupů jiného hradla.

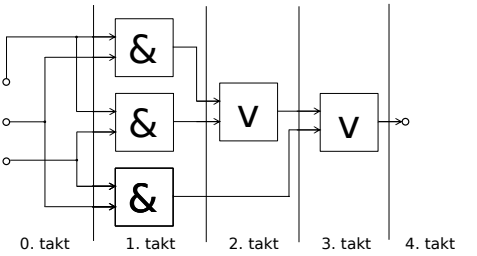
[2] Na hranách vedoucích do výstupů necháváme hodnotu této funkce nevyužitu.

Hodnotu hradla h přitom spočteme funkcí $F(h)$ z hodnot na jeho vstupech uspořádaných podle funkce z. Výstup sítě pouze zkopíruje hodnotu, která do něj po hraně přišla.

Jakmile budou po nějakém počtu taktů definované hodnoty všech výstupů, výpočet se zastaví a síť vydá výsledek – ohodnocení výstupů.

Podle průběhu výpočtu můžeme vrcholy sítě rozdělit do vrstev:

Definice: *i-tá vrstva S_i* obsahuje právě ty vrcholy v, pro které nejdelší z cest z libovolného vrcholu se vstupním stupněm 0 do v má délku rovnou právě i.



Obr. 1.2: Průběh výpočtu a rozdělení sítě na vrstvy

Pár pozorování o průběhu výpočtu:

- V i-té vrstvě jsou tedy právě ty vrcholy, které poprvé ohodnotíme v i-tém taktu výpočtu.
- Jelikož síť je acyklická, tak platí, že jakmile vrchol ohodnotíme, už se jeho ohodnocení nikdy nemůže změnit.
- Když se vydáme z libovolného vrcholu proti směru hran, po konečně mnoha krocích dojdeme do vrcholu s nulovým vstupním stupněm (vstupu sítě nebo konstantního hradla). Proto každý vrchol leží v nějaké vrstvě.
- Vrstvy jsou disjunktní, takže počet neprázdných vrstev je nutně konečný. V kombinaci s předchozím pozorováním dostaneme, že výpočet sítě se vždy zastaví.
- Navíc poslední neprázdná vrstva je právě ta, kde se výpočet zastaví – z každého dalšího vrcholu by totiž bylo možné po hranách dojít do vrcholu s výstupním stupněm 0 a jediné takové vrcholy jsou výstupy sítě. Ty by tedy musely také ležet v některé z následujících vrstev, což ovšem není možné, neboť výpočet se už zastavil.

To nás motivuje k následující definici:

Definice: *Časovou složitost* sítě definujeme jako počet jejích neprázdných vrstev. Podobně *prostorovou složitost* položíme rovnu počtu hradel v síti.

Poznámka o aritách: Kdybychom připustili hradla s libovolně vysokým počtem vstupů, mohli bychom jakýkoliv problém se vstupem délky n a výstupem délky ℓ vyřešit

v jedné vrstvě pomocí ℓ kusů n-vstupových hradel. Každému bychom prostě přiřadili funkci, která počítá příslušný bit výsledku ze všech bitů vstupu.

To však není ani realistické, ani pěkné. Jak z toho ven? Přidáme pravidlo, které omezí arity všech hradel nějakou pevnou konstantou, třeba dvojkou. Budeme tedy používat výhradně nulární, unární a binární hradla.

Poznamenejme ještě, že realistický model (byť s trochu jinými vlastnostmi) by vznikl také tehdy, kdybychom místo arity omezily typy funkcí, řekněme na AND, OR a NOT.

Poznámka o uniformitě: Dodejme, že od běžných výpočetních modelů, jako je třeba RAM, se hradlové sítě liší jednou podstatnou vlastností – každá síť zpracovává výhradně vstupy jedné konkrétní velikosti. Chceme tedy najít nějaký obecný předpis, který pro každou velikost vstupu sestrojí příslušnou síť. Takovým výpočetním modelům se říká *neuniformní.*

A co myslíme oním předpisem pro sestrojení sítě? Bude to pro nás prostě jakýkoliv algoritmus (klasický, neparalelní) běžící v polynomiálním čase. (Kdybychom dovolili i pomalejší algoritmy, mohli bychom během konstrukce provádět nějaký náročný předvýpočet a jeho výsledek zabudovat do struktury sítě. To obvykle není žádoucí.)

Hledá se jednička

Abychom si nový výpočetní model osahali, zkusme nejprve sestrojit obvod, který zjistí, zda se mezi jeho n vstupy vyskytuje alespoň jedna jednička. To znamená vypočítat n-vstupovou funkci OR.

První řešení: Zapojíme hradla za sebe (sériově). Časová i prostorová složitost činí $\Theta(n)$. Zde ovšem vůbec nevyužíváme toho, že by mohlo počítat více hradel současně.

Druhé řešení: Hradla budeme spojovat do dvojic, pak výsledky těchto dvojic opět do dvojic a tak dále. Díky paralelnímu zapojení dosáhneme časové složitosti $\Theta(\log n)$, přičemž prostorová složitost zůstane lineární.

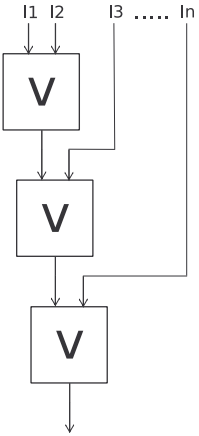
1.2. Sčítání binárních čísel

Zajímavější úlohou, jejíž paralelizace už nebude tak triviální, je obyčejné sčítání dvojkových čísel. Mějme dvě čísla x a y zapsané ve dvojkové soustavě. Jejich číslice označme $x_{n-1} \ldots x_0$ a $y_{n-1} \ldots y_0$, přičemž i-tý řád má váhu 2^i.

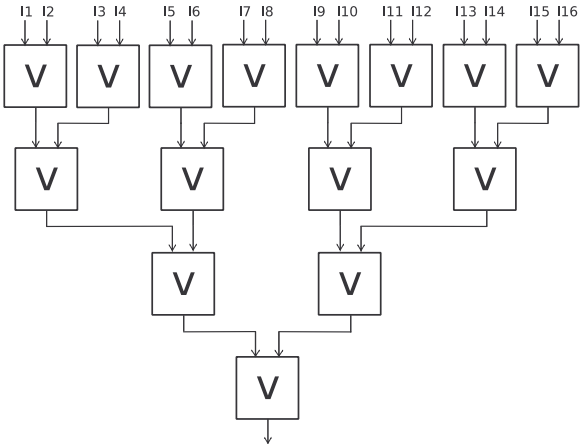
Ihned se nabízí použít starý dobrý „školní algoritmus sčítání pod sebou“. Ten funguje ve dvojkové soustavě stejně dobře jako v desítkové. Sčítáme čísla zprava doleva, vždy sečteme x_i s y_i a přičteme přenos z nižšího řádu. Tím dostaneme jednu číslici výsledku a přenos do vyššího řádu. Formálně bychom 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 do vyššího řádu nastane tehdy, pokud se nám



Obr. 1.3: Sériové řešení

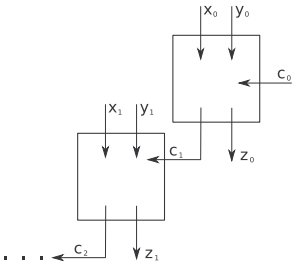


Obr. 1.4: Paralelní řešení

potkají dvě jedničky pod sebou, nebo když se vyskytne alespoň jedna jednička a k tomu přenos z nižšího řádu. To je tehdy, když mezi třemi xorovanými číslicemi jsou alespoň dvě jedničky – k tomu se nám hodí již známý obvod pro majoritu:

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

O tomto předpisu snadno dokážeme, že funguje (zkuste si to), nicméně pokud podle něj postavíme hradlovou síť, bude poměrně pomalá. Síť si můžeme představit tak, že je složena 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}:



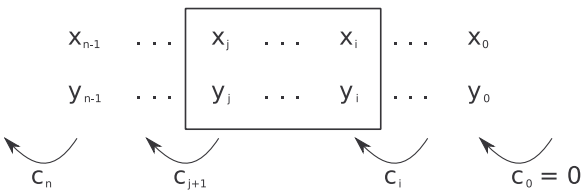
Obr. 1.5: Sčítání školním algoritmem

Každá krabička má sama o sobě konstantní hloubku, ovšem k výpočtu potřebuje přenos vypočítaný předcházející krabičkou. Jednotlivé krabičky proto musí ležet v různých vrstvách sítě. Vrstev tedy je $\Theta(n)$ a stejně tak hradel. Oproti sekvenčnímu algoritmu jsme si bohužel ani trochu nepomohli.

Přenosy v blocích

Jak sčítání zrychlit? To, co nás při sčítání brzdí, jsou evidentně přenosy mezi jednotlivými řády. Kdybychom je dokázali spočítat rychleji, máme vyhráno – součet už získáme jednoduchým xorováním, které zvládneme paralelně v čase $\Theta(1)$. Uvažujme tedy nad způsobem, jak přenosy spočítat paralelně.

Podívejme se na libovolný *blok* v našem součtu. Tak budeme říkat číslům $x_j \ldots x_i$ a $y_j \ldots y_i$ 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.

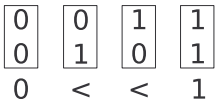


Obr. 1.6: Blok součtu

Pro konkrétní sčítance se tedy můžeme na blok dívat jako na nějakou funkci, která dostane jednobitový vstup (přenos zespoda) a vydá jednobitový výstup (přenos nahoru). To je nám milé, neboť takové funkce existují pouze čtyři:

$f(x) = 0$	konstantní **0**, blok *pohlcuje* přenos
$f(x) = 1$	konstantní **1**, blok *vytváří* přenos
$f(x) = x$	identita (značíme $<$), blok *kopíruje* přenos
$f(x) = \neg x$	negace, ukážeme, že u žádného bloku nenastane

Této funkci budeme říkat *chování* příslušného bloku.

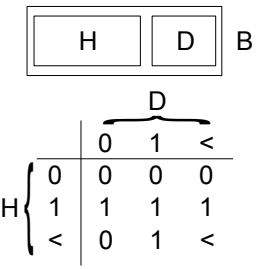


Obr. 1.7: Tabulka triviálních bloků

Jednobitové bloky se přitom chovají velice jednoduše:

Blok prvního druhu vždy předává nulový přenos, ať už do něj vstoupí jakýkoliv – přenos tedy pohlcuje. Poslední blok naopak sám o sobě přenos vytváří, ať dostane cokoliv. Oba bloky prostřední se chovají tak, že samy o sobě žádný přenos nevytvoří, ale pokud do nich nějaký přijde, tak také odejde.

Větší bloky můžeme rozdělit na části a podle chování částí určit, jak se chová celý blok. Mějme blok B složený z menších podbloků H (horní část) a D (dolní). Chování celku závisí na chování částí takto:



Obr. 1.8: Skládání chování bloků

Pokud vyšší blok přenos pohlcuje, pak ať se už nižší blok chová jakkoli, složení obou 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. Pojďme si přeci jen rozmyslet, jak bychom takovou operaci popsali čistě binárně.

Tři stavy můžeme zakódovat pomocí dvou bitů, říkejme jim třeba p a q. Dvojice (p, q) přitom může nabývat hned čtyř možných hodnot, my dvěma z nich přiřadíme stejný význam:

$$(1, *) = < \qquad (0, 0) = \mathbf{0} \qquad (0, 1) = \mathbf{1}.$$

Kdykoliv $p = 1$, blok kopíruje přenos. Naopak $p = 0$ odpovídá tomu, že blok posílá do vyššího řádu konstantní přenos, a q pak určuje, jaký. Kombinování bloků (skládání

funkcí) pak můžeme popsat následovně:

$$p_B = p_H \,\&\, p_D,$$
$$q_B = (\neg p_H \,\&\, q_H) \vee (p_H \,\&\, q_D).$$

A průchod přenosu blokem (dosazení hodnoty funkce) takto:

$$c_{j+1} = (p \,\&\, c_i) \vee (\neg p \,\&\, q).$$

Rozmyslete si, že tyto formule odpovídají výše uvedené tabulce.

Paralelní sčítání

Od popisu chování bloků je už jenom krůček k paralelnímu předpovídání přenosů, a tím i k paralelní sčítačce. Bez újmy na obecnosti budeme předpokládat, že počet bitů vstupních čísel je mocnina dvojky; jinak vstup doplníme zleva nulami.

Algoritmus bude rozdělen na dvě části:

První část spočítá chování všech *přirozených bloků* – tak budeme říkat blokům, jejichž velikost je mocnina dvojky a pozice je dělitelná velikostí (bloky téže velikosti se tedy nepřekrývají). Nejprve v konstantním čase stanoví chování bloků velikosti 1, ty pak spojí do dvojic, dvojice zase do dvojic atd., obecně v i-tém kroku spočte chování všech přirozených bloků velikosti 2^i.

Druhá část pak dopočítá přenosy, a to tak, aby v i-tém kroku byly známy přenosy do řádů dělitelných $2^{\log n - i}$. V nultém kroku tedy pouze $c_0 = 0$ a c_n, který spočítá z c_0 pomocí chování bloku $\langle 0, n\rangle$. V prvním kroku pomocí bloku $\langle 0, n/2\rangle$ dopočítá $c_{n/2}$, v druhém pomocí $\langle 0, n/4\rangle$ spočítá $c_{n/4}$ a pomocí $\langle n/2, 3/4 \cdot n\rangle$ dostane $c_{3/4 \cdot n}$ atd. Obecně v i-tém kroku používá chování bloků velikosti $2^{\log n - i}$.

Sčítací síť tedy bude vypadat takto:

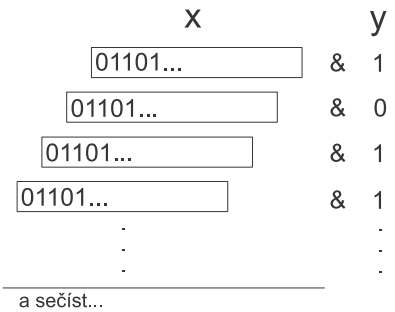
1. $\Theta(1)$ hladin výpočtu chování bloků velikosti 1.
2. $\Theta(\log n)$ hladin počítajících chování všech přirozených bloků.
3. $\Theta(\log n)$ hladin dopočítávajících přenosy „zahušťováním“.
4. $\Theta(1)$ hladin na samotné sečtení: $\forall i : z_i = x_i \oplus y_i \oplus c_i$.

Obr. 1.9: Výpočet přenosu

Algoritmus tedy pracuje v čase $\Theta(\log n)$. Využívá k tomu lineárně mnoho hradel: při výpočtu chování bloků na jednotlivých hladinách počet hradel exponenciálně klesá od n k 1, během zahušťování přenosů naopak exponenciálně stoupá od 1 k n. Obě geometrické řady se sečtou na $\Theta(n)$.

Paralelní násobení

Ještě si rozmysleme, jak rychle by bylo možné čísla násobit. Opět se inspirujeme školním algoritmem: pokud násobíme dvě n-ciferná čísla x a y, uvážíme všech n posunutí čísla x, každé z nich vynásobíme příslušnou číslicí v y a výsledky posčítáme.



Obr. 1.10: Školní násobení

Ve dvojkové soustavě je to ještě jednodušší: násobení jednou číslicí je prostý AND. Paralelně tedy vytvoříme všechna posunutí a spočítáme všechny ANDy. To vše stihneme za 1 takt výpočtu.

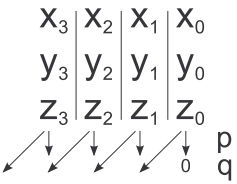
Zbývá sečíst n čísel, z nichž každé má $\Theta(n)$ bitů. Mohli bychom opět sáhnout po osvědčeném triku: sčítat dvojice čísel, pak dvojice těchto součtů, atd. Taková síť by měla tvar binárního stromu hloubky $\log n$, jehož každý vrchol by obsahoval jednu sčítačku, a na tu, jak víme, postačí $\Theta(\log n)$ hladin. Celý výpočet tedy běží v čase $\Theta(\log^2 n)$.

Jde to ale rychleji, použijeme-li jednoduchý, téměř kouzelnický trik. Sestrojíme *kompresor* – to bude obvod konstantní hloubky, který na vstupu dostane tři čísla a vypočte z nich dvě čísla mající stejný součet jako zadaná trojice.

K čemu je to dobré? Máme-li sečíst n čísel, v konstantním čase dokážeme tento úkol převést na sečtení $\lceil 2/3 \cdot n \rceil$ čísel, to pak opět v konstantním čase na sečtení $\lceil (2/3)^2 \cdot n \rceil$ čísel atd., až nám po $\lceil \log_{3/2} n \rceil = \Theta(\log n)$ krocích zbudou dvě čísla a ta sečteme klasickou sčítačkou. Zbývá vymyslet kompresor.

Konstrukce kompresoru: Označme vstupy kompresoru x, y a z a výstupy p a q. Pro každý řád i spočteme součet $x_i + y_i + z_i$. To je nějaké dvoubitové číslo, takže můžeme jeho nižší bit prohlásit za p_i a vyšší za q_{i+1}.

Jinými slovy všechna tři čísla jsme normálně sečetli, ale místo abychom přenosy posílali do vyššího řádu, vytvořili jsme z nich další číslo, které má být k výsledku časem přičteno.



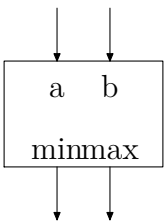
Obr. 1.11: Kompresor

Shrnutí: Naše síť pro paralelní násobení pracuje v čase $\Theta(\log n)$ – nejdříve v konstantním čase vytváříme mezivýsledky, pak použijeme $\Theta(\log n)$ hladin kompresorů konstantní hloubky a nakonec jednu sčítačku hloubky $\Theta(\log n)$. Jistou vadou na kráse ovšem je, že na to potřebujeme $\Theta(n^2)$ hradel.

1.3. Třídící sítě

Ještě zkusíme paralelizovat jeden klasický problém, totiž třídění. Budeme k tomu používat *komparátorovou síť* – to je hradlová síť složená z *komparátorů.*

Jeden 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: na jednom z nich vrací menší ze vstupních hodnot a na druhém tu větší.



Obr. 1.12: Komparátor

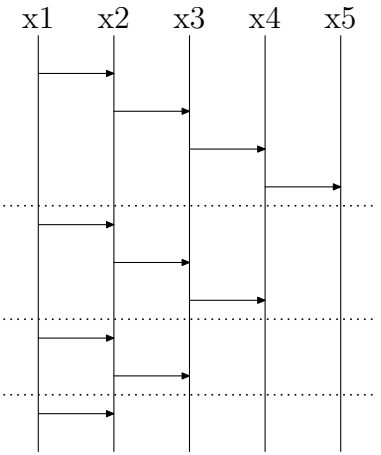
V našem formalismu hradlových sítí bychom mohli komparátor reprezentovat dvojicí hradel: jedno z nich by počítalo minimum, druhé maximum. Hodnoty, které třídíme, bychom přitom považovali za prvky abecedy.⟨3⟩

Ještě se dohodněme, že výstupy komparátorů se nikdy nebudou větvit. Každý z nich přivedeme na vstup jiného komparátoru nebo na výstup sítě. Větvení by nám ostatně k ničemu nebylo, protože na výstupu potřebujeme vydat stejný počet hodnot,

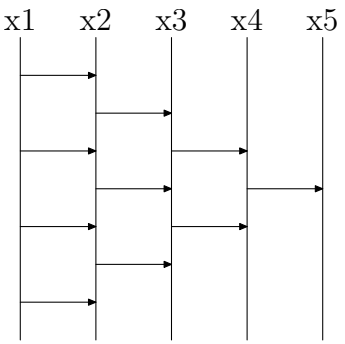
⟨3⟩ Komparátorovou síť můžeme také snadno přeložit na booleovský obvod. Každý prvek abecedy reprezentujeme číslem o $b = \lceil \log_2 \Sigma \rceil$ bitech. Způsobem podobným paralelní sčítačce lze z booleovských hradel sestrojit komparátor hloubky $\Theta(\log b)$. Zkuste to.

jako byl na vstupu, a nemáme žádné hradlo, kterým bychom mohli případných vícero větví sloučit opět do jedné.

Příklad: Zkusíme do řeči komparátorových sítí přeložit *bublinkové třídění*. Z něj získáme obvod na levém obrázku (šipky představují jednotlivé komparátory). Toto nakreslení ovšem poněkud klame – pokud síť necháme počítat, mnohá porovnání budou probíhat paralelně. Skutečný průběh výpočtu znázorňuje pravý obrázek, na němž jsme všechny operace prováděné současně znázornili vedle sebe. Ihned vidíme, že paralelní bublinkové třídění pracuje v čase $\Theta(n)$ a potřebuje kvadratický počet komparátorů.



Obr. 1.13: Bubblesort



Obr. 1.14: Skutečný průběh výpočtu

Nyní si předvedeme rychlejší třídící algoritmus. Půjdeme na něj jak se říká „od lesa“. Nejdříve vymyslíme síť, která bude umět třídit jenom něco – totiž bitonické

posloupnosti. Pak z ní teprve sestrojíme obecné třídění. Bez újmy na obecnosti přitom budeme předpokládat, že každé dva prvky na vstupu jsou navzájem různé a že velikost vstupu je mocnina dvojky.

Definice: Posloupnost $x_0, \ldots, x_{n-1}$ je *čistě bitonická,* pokud ji můžeme rozdělit na nějaké pozici $k \in \{0, \ldots, n-1\}$ tak, že prvky $x_0, \ldots, x_k$ tvoří rostoucí poslopnost, zatímco prvky $x_k, \ldots, x_{n-1}$ tvoří posloupnost klesající.

Definice: Posloupnost $x_0, \ldots, x_{n-1}$ je *bitonická*, jestliže ji lze získat rotací (cyklickým posunutím) nějaké čistě bitonické posloupnosti. Tedy pokud existuje $0 \leq j < n$ takové, že posloupnost $x_j, x_{(j+1) \bmod n}, \ldots, x_{(j+n-1) \bmod n}$ je čistě bitonická.

Definice: *Separátor řádu n* je komparátorová síť S_n se vstupy $x_0, \ldots, x_{n-1}$ a výstupy $y_0, \ldots, y_{n-1}$, která dostane-li na vstupu bitonickou posloupnost, vydá na výstup její permutaci s následujícími vlastnostmi:

- $y_0, \ldots, y_{n/2-1}$ a $y_{n/2}, \ldots, y_{n-1}$ jsou bitonické posloupnosti;
- $y_i < y_j$, kdykoliv $0 \leq i < n/2 \leq j < n$.

Jinak řečeno, separátor rozdělí bitonickou posloupnost na dvě poloviční a navíc jsou všechny prvky v první polovině menší než všechny v té druhé.

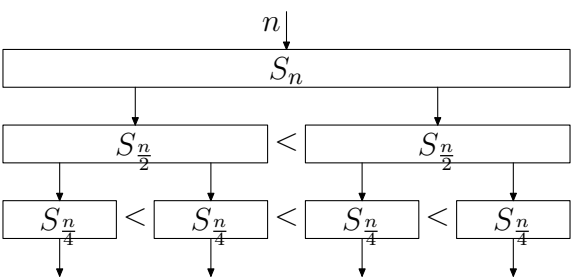
Lemma: Pro každé sudé n existuje separátor S_n konstantní hloubky, složený z $\Theta(n)$ komparátorů.

Důkaz tohoto lemmatu si necháme na konec kapitoly. Nejprve předvedeme, k čemu jsou separátory dobré.

Definice: *Bitonická třídička řádu n* je komparátorová síť B_n, která dostane-li na vstupu bitonickou posloupnost délky n, vydá ji setříděnou.

Lemma: Pro libovolné $n = 2^k$ existuje bitonická třídička B_n hloubky $\Theta(\log n)$ s $\Theta(n \log n)$ komparátory.

Důkaz: Konstrukce bitonické třídičky je snadná: nejprve separátorem S_n zadanou bitonickou posloupnost rozdělíme na dvě bitonické posloupnosti délky $n/2$, každou z nich pak separátorem $S_{n/2}$ na dvě části délky $n/4$, atd., až získáme jednoprvkové posloupnosti ve správném pořadí. Celkem použijeme $\log n$ hladin složených z n separátorů, každá hladina má přitom konstantní hloubku. □



Obr. 1.15: Bitonická třídička B_n

Bitonické třidičky nám nyní pomohou ke konstrukci třidičky na obecné posloupnosti. Ta bude založena na třídění sléváním – nejprve se tedy musíme naučit slít dvě setříděné posloupnosti do jedné.

Definice: *Slévačka řádu n* je komparátorová síť M_n s $2 \times n$ vstupy a n výstupy, která dostane-li dvě setříděné posloupnosti délky n, vydá posloupnost vzniklou jejich slitím.

Lemma: Pro $n = 2^k$ existuje slévačka M_n hloubky $\Theta(\log n)$ s $\Theta(n \log n)$ komparátory.

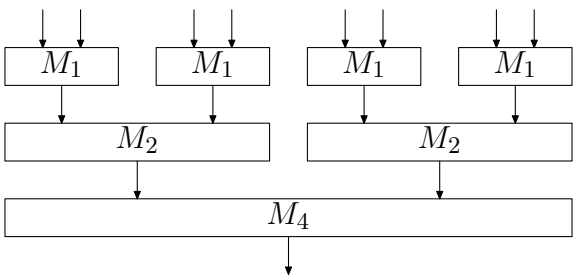
Důkaz: Stačí jednu vstupní posloupnost obrátit a „přilepit“ za tu druhou. Tím vznikne bitonická posloupnost, jíž setřídíme bitonickou třidičkou B_{2n}. □

Definice: *Třídící síť řádu n* je komparátorová síť T_n s n vstupy a n výstupy, která pro každý vstup vydá jeho setříděnou permutaci.

Lemma: Pro $n = 2^k$ existuje třídící síť T_n hloubky $\Theta(\log^2 n)$ složená z $\Theta(n \log^2 n)$ komparátorů.

Důkaz: Síť bude třídit sléváním. Vstup rozdělí na n jednoprvkových posloupností. Ty jsou jistě setříděné, takže je slévačkami M_1 můžeme slít do dvouprvkových setříděných posloupností. Na ty pak aplikujeme slévačky M_2, M_4, …, $M_{n/2}$, až všechny části slijeme do jedné, setříděné.

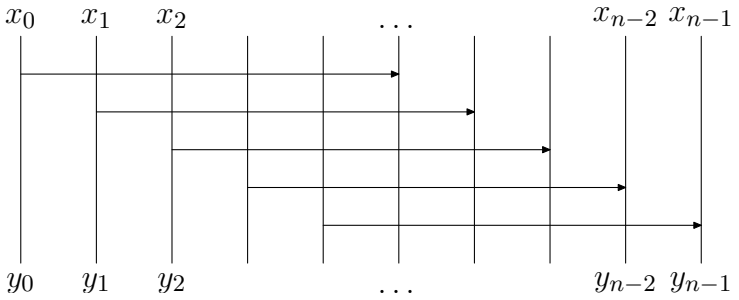
Celkem provedeme $\log n$ kroků slévání, i-tý z nich obsahuje slévačky M_{2^i} a ty, jak už víme, mají hloubku $\Theta(i)$. Celkový počet vrstev tedy činí $\Theta(1 + 2 + 3 + \ldots + \log n) = \Theta(\log^2 n)$. Každý krok přitom potřebuje $\Theta(n \log n)$ komparátorů, což dává celkem $\Theta(n \log^2 n)$ komparátorů. □



Obr. 1.16: Třidička T_8

Konstrukce separátoru: Zbývá dokázat, že existují separátory konstantní hloubky. Vypadají překvapivě jednoduše: pro $i = 0, \ldots, n/2 - 1$ zapojíme komparátor se vstupy x_i, $x_{i+n/2}$, jehož minimum přivedeme na y_i a maximum na $y_{i+n/2}$.

Proč separátor separuje? Nejprve předpokládejme, že vstupem je čistě bitonická posloupnost. Označme m polohu maxima této posloupnosti; maximum bez újmy na obecnosti leží v první polovině (jinak celý důkaz provedeme „zrcadlově“). Označme dále k nejmenší index, pro který komparátor zapojený mezi x_k a $x_{k+n/2}$ hodnoty prohodí, tedy $k = \min\{i \mid x_i > x_{i+n/2}\}$.

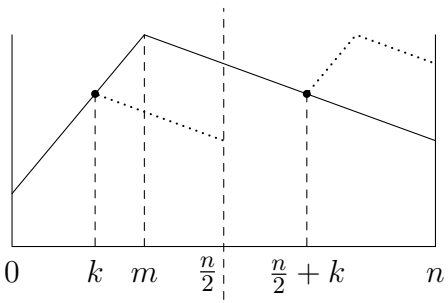


Obr. 1.17: Konstrukce separátoru

Jelikož maximum je jedinečné, musí platit $x_m > x_{m+n/2}$, takže k existuje a navíc $0 \leq k \leq m < n/2$. Také platí, že pro každé i mezi k a $n/2$ už komparátory musí prohazovat, protože od x_k je posloupnost až do konce klesající, a proto $x_i > x_{i+n/2}$.

Separátor se tedy chová velice jednoduše: první polovina výstupu vznikne slepením rostoucího úseku $x_0, \ldots, x_{k-1}$ s klesajícím úsekem $x_{n/2+k}, \ldots, x_{n-1}$; druhou polovinu tvoří spojení klesajícího úseku $x_{n/2}, \ldots, x_{n/2+k-1}$, rostoucího úseku $x_k, \ldots, x_m$ a klesajícího úseku $x_m, \ldots, x_{n/2-1}$.

Snadno tedy ověříme, že se separátor chová podle definice: První polovina je čistě bitonická a jelikož $x_{n/2-1} > x_{n/2}$, je druhá polovina bitonická (obvykle ne čistě). Navíc víme, že nejvyšším bodem první části je x_k a nejnižším bodem druhé části $x_{n/2+k} > x_k$, takže všechny prvky první části musí být menší než všechny v té druhé.



Obr. 1.18: Ilustrace činnosti separátoru

Doplňme, co se stane, pokud vstup není čistě bitonický. Zde využijeme toho, že separátor je symetrický, tudíž zrotujeme-li jeho vstup o p pozic, dostaneme o p pozic zrotované i obě poloviny výstupu. Podle definice ovšem pro každou bitonickou posloupnost existuje její rotace, která je čistě bitonická, a pro níž, jak už víme, separátor funguje. Takže pro nečistou bitonickou posloupnost musí vydat výsledek pouze zrotovaný, což ovšem na jeho správnosti nic nemění. □

Závěrem: Nalezli jsme paralelní třídící algoritmus o časové složitosti $\Theta(\log^2 n)$, který využívá $\Theta(n \log^2 n)$ komparátorů. Dodejme, že jsou známé i třídící sítě hloubky $\Theta(\log n)$, ale jejich konstrukce je mnohem komplikovanější a dává obrovské multiplikativní konstanty, které brání praktickému použití.

Pomocí dolního odhadu složitosti třídění (viz minulý semestr) navíc můžeme snadno dokázat, že logaritmický počet hladin je nejnižší možný. Máme-li totiž libovolnou třídící síť hloubky h, můžeme ji simulovat po hladinách a získat tak sekvenční třídící algoritmus. Jelikož na každé hladině může ležet nejvýše $n/2$ komparátorů, náš algoritmus provede maximálně $hn/2$ porovnání. My ovšem víme, že pro každý třídící algoritmus existují vstupy, na kterých porovná $\Omega(n \log n)$-krát. Proto $h = \Omega(\log n)$.