Komparátor

Úkol 4-1: Navrhněte hradlovou konstrukci komparátoru, analyzujte velikost a hloubku sítě.
Na vstupu dostaneme dvě n-bitová čísla A, B, reprezentovaná jako bity

$$a_1, \ldots, a_n, b_1, \ldots, b_n$$

(nebo v opačném pořadí, chcete-li hezkou poziční notaci bitů)
Na výstupu chceme dvojici čísel X, Y t.ž. $X = min(A, B), Y = max(A, B)$, reprezentované stejným způsobem.
Pro konstrukci můžete používat libovolná hradla s (≤ 2)-bitovými vstupy a 1-bitovými výstupy (tedy kromě klasických and, or, xor a negace i jakákoliv jiná, pokud se hodí).
Analyzujte (asymptoticky) velikost i hloubku výsledné sítě. Obě hodnoty chceme (asymptoticky) optmální.

Řešení:

Řešení složíme ze dvou částí. V první části zjistíme, zda $A > B$. V druhé části potom čísla ze vstupu prohodíme nebo necháme v původní poloze.
Asi nejelegantnější řešení je vyřešit paralelizaci úlohy pomocí přístupu rozděl a panuj. Rozdělením čísel A, B na dvě poloviny A_L, A_R a B_L, B_R (L, R pro levou a pravou stranu, levá strana obsahuje vyšší bity, jako v desítkovém zápisu).
Potom platí, že $A > B$ pokud $A_L > B_L$ nebo ($A_L = B_L$ a zároveň $A_R > B_R$).
Nerovnost tedy můžeme vyhodnotit tak, že si nejprve spočítáme nerovnosti polovin (spolu s rovností) rekurzivně a výsledky složíme. Pro levou i pravou polovinu si pořídíme bit G_L, G_R indukující, zda příslušná polovina A je větší než polovina B; a další bit E_L, E_R indikující, zda si jsou levé nebo pravé poloviny rovny.
Po technické stránce bude hradlová síť vypadat následovně. V první vrstvě pro každou bitovou pozici vyhodnotíme:
bit G_i představující "$a_i > b_i$" jako $a_i \wedge \neg b_i$
bit E_i představující "$a_i = b_i$" jako $\neg(a_i \text{ XOR } b_i)$
Nyní navrhneme podsítě na vyhodnocování rekurze (spojovátka). Pro bity G_L, G_R, E_L, E_R na vstupu vyhodnotíme bity:
G jako $G_L \vee (E_L \wedge G_R)$ - dle rekurzivního popisu výše
E jako $E_L \wedge E_R$ - číslo je rovné pouze pokud se rovnají obě poloviny
Spojujeme dle binárního stromu, tedy jednou vrstvou spojovátek spojíme výsledky pro jednotlivé bity do výsledků pro dvojice, následně do čtveřic atd., až dostaneme výsledek celé úlohy po $\mathcal{O}(\log(n))$ vrstvách spojovátek. Pokud n není mocnina dvojky, můžeme kdykoliv do volných vstupů zapojit umělý vstup s bitem $E = 1$. Pro transparntnost chceme toto vkládání ideálně provádět na nejlevějších možných pozicích, ale části čísel navíc, které jsou rovny nemohou ovlivnit ostrou nerovnost ať už jsou vloženy kdekoliv. Alternativně spojovátko bez jednoho vstupu vůbec nepostavíme, a jediný vstup rovnou propojíme do další vrstvy.

Nakonec potřebujeme na výstup vydat správná čísla. Do každé bitové pozice obou výstupů navedeme příslušné bitové pozice vstupů, a použijeme signál z výsledku $A > B$ pro zablokování signálu ze špatného vstupu. Označme výsledek porovnání G.
$X_i = (a_i \wedge G) \vee (b_i \wedge \neg G)$ $Y_i = (a_i \wedge \neg G) \vee (b_i \wedge G)$
V této implementaci sice "technicky" zbytečně prohazujeme čísla pokud si jsou rovna, ale výsledek to nemění.
Spojování výsledků rekurze bylo provedeno binárním stromem, ten má logaritmickou hloubku a lineárně uzlů (spojovátek s konstantní velikostí). Výpočet hodnot pro bity, a kopírování na výstup probíhají v konstantní hloubce, za pomocí lineárního množství hradel. Celková hloubka je tedy $\mathcal{O}(\log(n))$ a velikost $\mathcal{O}(n)$.
Můžeme nahlédnout, že toto jsou optimální hodnoty. Pokud by hloubka výpočtu $A < B$ byla sublogaritmická, výsledek může záviset pouze na sublineráním množství vstupů. Protože síť je fixní, můžeme potom najít vstupní bity neovlivňující výpočet, a vstupy lišící se pouze na těchto pozicích budou špatně vyhodnoceny.
Sice nemusí platit, že každý obvod explicitně počítá hodnotu jako $A < B$, ale totéž tvrzení platí pro každý výstupní bit samostatně. Co se týče velikosti, protože každý výstupní bit musí záviset příslušných pozicích obou vstupních čísel, musí být spočítán alespoň jedním hradlem, což je $\Theta(n)$ hradel.
(v dalších řešeních už pouze ukazujeme, jak určit řídící bit "$A > B$", zbytek konstrukce je stejný)
Víceméně totožné konstrukce šlo získat i používáním jiných mezivýsledků, obzvláště oblíbená kombinace byly bity $A \geq B$ a $A \leq B$ nebo jejich ostré varianty a různé podivné kombinace. Složitost návrhu spojovátek potom byla v některých variantách podstatně větší.
Jistou konceptuální výhodu měla řešení, kde se pomocí návrhu spojovátek, která na vstupu brala dvojice bitů z obou čísel a na výstup 10 pro $A > B$ a 01 pro $A < B$. Výsledek jedné vrstvy spojovátek potom šlo chápat jako novou dvojici binárních čísel (tvořených levými resp. pravými bity všech spojovátek) poloviční délky. Nebylo potom třeba rozlišovat mezi porovnáváním a sléváním výsledků. Na druhou stranu, návrh takových spojovátek byl o něco komplikovanější.

Řešení dle definice porovnávání

Při zápisu v obecných základech platí, že $a_n, \ldots, a_0 > b_n, \ldots, b_0$ právě když $a_n > b_n$ nebo ($a_n = b_n$ a zároveň $a_{n-1}, \ldots, a_0 > b_{n-1}, \ldots, b_0$)
Tím dostáváme rekurzivní předpis. Dle tohoto předpisu můžeme přímo navrhnout obvod. Stejně jako u klasického sčítání s carry-bitem ale dostaneme linerání hloubku.
První možnost, jak tento problém vyřešit, je pokusit se výpočet paralelizovat rozdělením na části. Tím obvykle dostaneme konstrukci velmi podobnou přístupu přes rozděl a panuj.
Jiný možný přístup je uvědomit si, že výsledek porovnání $A > B$ určuje nejlevější bit, kde se vstupní čísla liší. Můžeme tedy tento bit hledat. Nejlevější lišící se bit ("rozhodující bit") je takový, který se liší (a_i XOR b_i) a zároveň všechny bity nalevo jsou shodné. Hledání opět můžeme realizovat konstrukcí

rozděl a panuj. Pokud ale nesklouzneme do tohoto směru, může konstrukce vypadat třeba i následovně:
Nejprve provedeme všechna porovnání $a_i = b_i$. Pomocí binárního stromu tedy můžeme určovat rovnost na podintervalech rostoucích délek. Pak pro každou pozici můžeme zjistit, zda příslušný bit je rozhodující pomocí z-AND-ování rovností nejvýše $\log(n)$ již spočítaných podintervalů (podobně jako u rychlé binární sčítačky) a z-AND-ováním $a_i! = b_i$. Celý proces zvládneme v logaritmické hloubce, a chytrým AND-ováním s pouze lineráním počtem hradel. Výsledkem bude vrstva hladel indikující pro všechny bitové pozice, zda se jedná o rozhodující bit. Následně můžeme na všech pozicích provést operaci "i je rozhodující" $\wedge a_i > b_i$ a všechny výsledky spojit jedním OR-ovacím stromem do finálního hradla indikujícího "$A > B$". Bez precizní výstavby tento postup vyžaduje kvadratický počet hradel.

Řešení sčítačkou

Alternativní work-around je využít sčítačky a odčítání přes dvojkový doplněk. Pro plný popis se odkažme na předměty stylu počítačových architektur. Zjednodušeně řečeno, pomocí scčítání můžeme i odčítat pokud nejprve invertujeme menšitel (a přičteme k němu 1). Nejvyšší carry-bit potom indikuje přetečení bitové reprezentace, a tím říká, zda výsledek je kladný nebo záporný. Protože inverzi a sčítání zvládneme v logaritmické hloubce a s pomocí lineárního množství hradel, jedná se o adekvátně dobré porovnání čísel.

Další řešení

Objevila se i řešení stylu vyhodnocování rekurzivního předpisu porovnávání, kde následně každá pozice výstupu obsahovala samostnou síť vyhodnocující z mezivýsledků pro pozice nalevo zda je vše rovno (a tedy je jedno co se provede se vstupy), nebo něco nalevo rozhodlo nerovnost. Vyhodnocení nejpravějšího hradla potom bylo ekvivalentní celé síti popsané v řešení pomocí vyhledávání rozhodujícího bitu.
Několik řešení se pokoušelo spolu s řídícími bity rovnou přenášet i data. Kromě mnohem vyšší složitosti takového vyhodnocování bez výjimky vedlo na sekvenční proces s lineární hloubkou.

Řešení s přímým tokem dat

Některá řešení se pokoušela, namísto porvnání čísel v prvním kroku a prohození v druhém, přímo o prohazování bitů za chodu. Tedy zleva doprava se prochází bity vstupních čísel, a případně rovnou prohazují. Pokud dojde k prohození nebo verifikaci aktuální pozice, posílá se dál řícídí signál, který zablokuje následující hradla a nedovolí jim použít jiný směr přehození.
Hlavním problémem těchto řešení byl obvykle návrh, který rovnou počítá hodnotu většího a menšího bitu (bez vyhodnocení "$a_i < b_i$"), což bylo dostatečně matoucí ke vzniku problémů. Taktéž lze nahlédnout, že výsledek každého prohození závisí na všech předchozích, jedná se tedy o naskrze sekvenční proces, a tedy implementace obvodem má nevyhnutelně lineární hloubku.