Jakub Šoustar, xsoust02

Carry Look Ahead Parallel Binary Adder

Rozbor

Algoritmus sčítá dvě binární čísla. Oproti obyčejnému sekvenčnímu přístupu je výpočet výsledku urychlen pomocí předpočítání přenosů mezi řády. Nadále budeme pracovat s variantou algoritmu, která pracuje s $p=2\cdot n-1$ procesory, kde n je počet sčítaných bitů. Procesory tvoří perfektně vyvážený binární strom. Z tohoto důvodu je algoritmus schopný pracovat pouze s čísly tvořenými 2^k bity, pro nějaké $k\geq 0$. Strom je inicializován načtením po jednom bitu z obou sčítanců do listových uzlů (bity si odpovídají řádem; pokud je potřeba, jsou čísla zleva zarovnána nulami na potřebnou délku). Následně listové uzly spočtou své prvotní příznaky přenosu a z tohoto pole je stromem spočten *scan*. Z jeho výsledku jsou listové uzly schopny určit přenos z nižšího řádu a tedy i spočítat výsledek.

Protože algoritmus pracuje s perfektně vyváženým binárním stromem, je pro dvě n bitová čísla zapotřebí $p=2\cdot n-1$ procesorů (n listových a $n-1$ nelistových). Algoritmus se skládá z několika podčástí:

1. *up-sweep* – celkem provede $\log_2(n)$ kroků
2. *down-sweep* – celkem provede $\log_2(n)$ kroků
3. Dvakrát posun hodnot v listových procesorech a výpočty příznaků přenosů a hodnot. Všechny tyto operace mají konstantní časovou složitost a dále je zanedbáme.

Z toho získáváme tyto teoretické složitosti:

$$t(n)=2\cdot\log_2(n)=O(\log(n))$$
$$p(n)=2\cdot n-1=O(n)$$
$$c(n)=t(n)\cdot p(n)=O(n\cdot\log(n))$$

Implementace

V následujícím textu budou zmíněny výpočty zahrnující příznaky přenosu, pro detaily odkazujeme čtenáře na přednášky předmětu PRL. Algoritmus je implementován následujícím způsobem:

1. Kořenový procesor načte vstupní soubor. Je-li potřeba, doplní čísla zleva potřebným počtem nul tak, aby celkový počet bitů čísla byl mocninou dvou. Následně je každému listovému procesoru odeslán jeden bit z prvního i druhého čísla. První listový procesor obdrží nejvýznamnější bity atd. Listové procesory přijmou dva bity, jejichž součet budou počítat. V krajním případě sčítání dvou jedno-bitových čísel provede celý výpočet kořenový procesor a algoritmus se ukončí. Listové procesory spočtou prvotní příznak přenosu.
2. Fáze *up-sweep*:
   a. Je-li uzel nelistový, přijme příznaky přenosu od svých synů, které si uchová. Z těchto příznaků spočte vlastní příznak přenosu. Pokud je uzel kořenem, vypočtený příznak si uchová. Pokud není kořenem, tak vypočtený příznak odešle svému rodiči.
   b. Je-li uzel listový, odešle svůj prvotní příznak přenosu svému rodiči.
3. Fáze *down-sweep*:
   a. Je-li uzel nelistový, přijme příznak od svého rodiče (kořen použije neutrální příznak *propagate*). Uzel odešle svému levému synu příznak získaný z příznaku pravého syna a svého vlastního. Uzel odešle pravému synu svůj příznak.

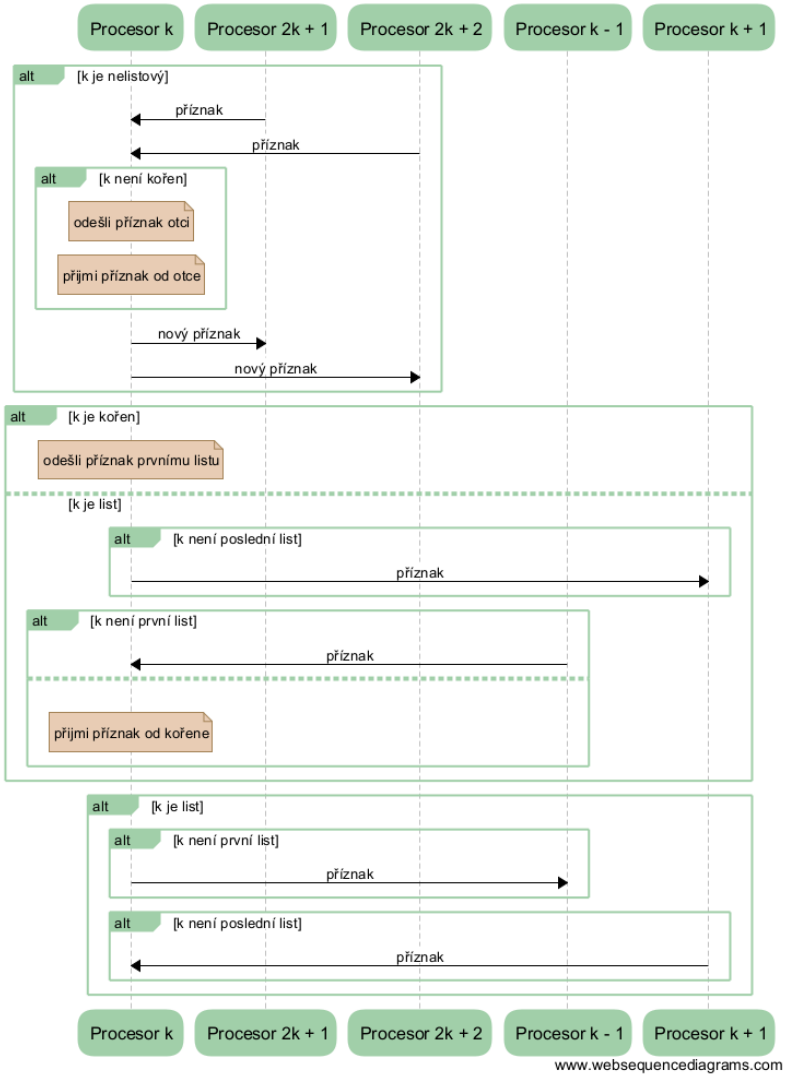
   b. Je-li uzel listový, přijme od rodiče příznak a uchová si jej.

4. Všechny listové uzly odešlou příznak svému pravému sourozenci. Kořen odešle příznak uchovaný z fáze *up-sweep* nejlevějšímu listu. Nejpravější list příznak neodesílá.

5. Všechny listové uzly odešlou příznak svému levému sourozenci. Nejlevější list hodnotu neodesílá. Nejpravější list nastaví svůj příznak na *stop* (až po odeslání původního). Všechny listy spočtou hodnotu *carry*. Ta je rovna jedné pokud je příznak *generate*, jinak je rovna nule. Všechny listy provedou výpočet výsledku a vypíší jej. Kořenový uzel vypíše informaci o případném přetečení na základě hodnoty příznaku získaného ve fázi *up-sweep*.

Komunikační protokol

Na následujícím diagramu je znázorněn komunikační protokol z pohledu procesoru s rankem k. Tento procesor, není-li listový, má dva synovské procesory s ranky $2 \cdot k + 1$ a $2 \cdot k + 2$ a dva sourozenecké procesory s ranky $k + 1$ a $k - 1$. Procesor a jeho dva synovské procesory dohromady tvoří strom, jehož kořenový uzel může být zároveň jedním ze synů jiného procesoru a oba synové mohou být kořeny dalších stromů. Z tohoto důvodu je komunikace ilustrována právě z pohledu kořene tohoto jednoduchého stromu. Na základě tohoto protokolu probíhá komunikace v libovolně větším stromu. Sourozenecký vztah uvažujeme pouze u listových procesorů a v těchto případech označujeme procesor s menším, resp. větším, rankem jako levého, resp. pravého, sourozence. Podobně jako u vztahu otec-syn, i v tomto případě může být procesor jak levým tak i pravým sourozencem jiných procesorů. Z diagramu jsou pro větší přehlednost vypuštěny veškeré nekomunikační operace (viz implementace).

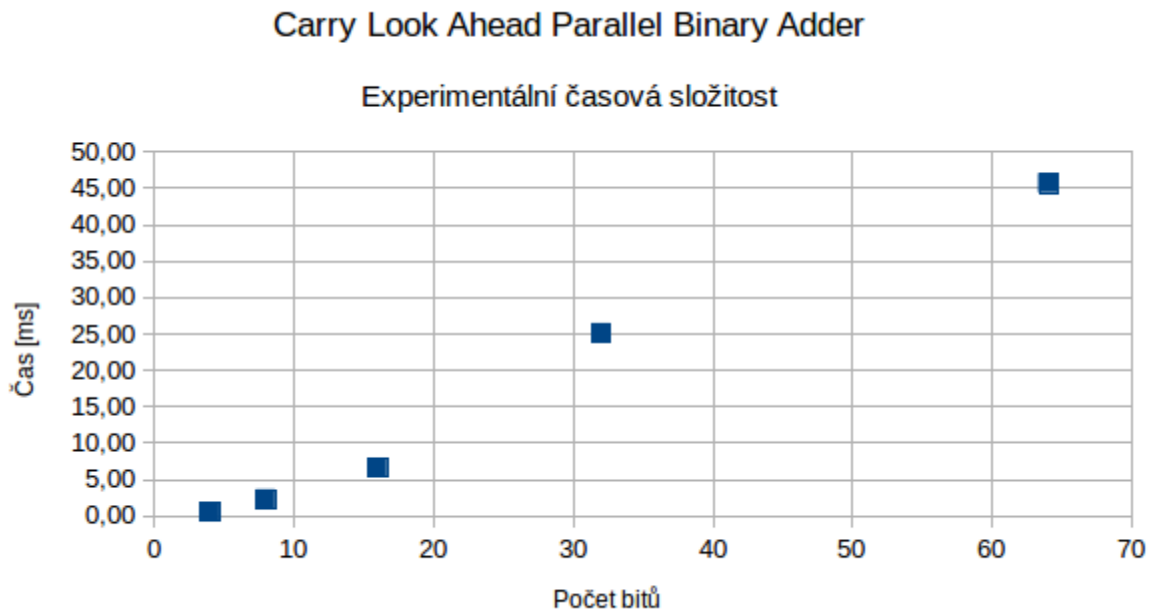


Jakub Šoustar, xsoust02

Experimenty

Časová složitost byla měřena pro dvě náhodně vygenerovaná čísla s počty bitů 4, 8, 16, 32 a 64. Pro každou velikost čísla bylo provedeno celkem 100 měření. V grafu jsou zaneseny průměry získaných hodnot. Měření bylo provedeno na osobním počítači, protože na školním serveru nebylo možné provést měření pro více jak osmibitová čísla. Pro vícebitová čísla se mi již nepovedlo hodnoty naměřit ani na mém počítači.

Naměřené hodnoty představují dobu běhu kořenového uzlu. Byla měřena pouze doba běhu hlavní části (fáze *up-sweep* a *down-sweep* a posun hodnot v listech) algoritmu. V průběhu měření byly z kódu pomocí direktiv preprocesoru odstraněny veškeré výpisy na standardní výstup, s výjimkou naměřeného času, který je vypisován na standardní chybový výstup. Měření se provádí s použitím funkcí `MPI_Barrier` a `MPI_Wtime` z knihovny Open MPI.



Závěr

Experimentálně naměřená složitost se od té teoretické odchyluje – jedná se spíše o složitost $O(n)$. Pro potvrzení by bylo vhodné provést další měření pro větší čísla v prostředí, které to umožní. Naměřené hodnoty také v průběhu měření vykazovaly značné odchylky (až tři řády) pro dvě po sobě jdoucí měření. Jejich příčinu se nepodařilo zjistit, ale pravděpodobně do jisté míry zkreslily výsledky měření. Odchylky od teoretické složitosti nejsou až tak překvapivé. Jakákoliv implementace algoritmu bude vždy navíc ovlivněna režií použité knihovny, hardwarem, na kterém program poběží (množství virtuálních procesorů na fyzický a s tím spojená režie pro přepínání kontextů atp.), operačním systémem, který plánuje běh jednotlivých procesorů a dalšími okolnostmi. S rostoucím počtem procesorů také poroste režie na jejich synchronizaci a vzájemné zasílání zpráv. Teoretický algoritmus tyto okolnosti nezohledňuje.

Nejnáročnější částí algoritmu jsou fáze *up-sweep* a *down-sweep,* při kterých je aktivně využito největší množství procesorů. Použitá verze algoritmu není nejefektivnější – pro algoritmus postačuje i n procesorů. Je možné a pravděpodobné, že implementace této verze by se svojí časovou složitostí více přiblížila té teoretické.

Nakonec je vhodné zmínit zbytečnost obou posunů příznaků v listových procesorech – tyto body byly vyžadovány zadáním, a to i přesto, že pro samotný výpočet jsou zcela nadbytečné.