Wydział Elektroniki
Politechniki Wrocławskiej
Informatyka Techniczna

Wrocław, 17 czerwca 2021

ARCHITEKTURA KOMPUTERÓW 2

Projekt

Sprawozdanie z projektu

pt. „Dzielenie przez stałą z wykorzystaniem systemu RNS”

autorzy:
Jędrzej Piątek 249453
Jakub Górecki 184201

prowadzący:
Dr inż. Piotr Patronik

1. Wstęp teoretyczny

Implementacja dzielenia liczb całkowitych w hardware jest kosztowna w porównaniu do mnożenia. Nawet najlepsze algorytmy dzielenia cechuje znacznie większa złożoność czasowa niż algorytmy mnożenia, dodawania czy odejmowania. Publikacja „*Correctly rounded constant integer division via multiply-add*" pokazuje jednak, że w sytuacjach, gdy ma miejsce dzielenie dużej ilości liczb całkowitych przez stały dzielnik, dzielenie można i warto zastąpić szybszymi algorytmami mnożenia i dodawania (Drane, Cheung, Constantinides, 2012).

Zadanie polega na znalezieniu parametrów *a*, *b* oraz *k* takich, że:

$$Round\left(\frac{x}{d}\right) = \left\lfloor \frac{ax+b}{2^k} \right\rfloor \qquad x \in [0, 2^n - 1]$$

z założeniem, że *d* jest liczbą całkowitą nieparzystą większą niż 1, oraz *a* jest nieparzyste. W zależności od pożądanej metody zaokrąglania uzyskujemy różne warunki konieczne i wystarczające. Dla zaokrąglania *round towards zero* jest to:

$$-\frac{b+1}{\left\lfloor \frac{2^n}{d} \right\rfloor} < ad - 2^k < a - b \qquad dla \quad ad - 2^k < 0$$

$$ad - 2^k < \frac{a-b}{\left\lfloor \frac{2^n}{d} \right\rfloor} \qquad dla \quad ad - 2^k > 0$$

Warunek konieczny i wystarczający dla zaokrągleń *round to nearest*, *round to even*:

$$\frac{a(d-1) - 2b - 1}{2\lfloor (2^{n+1} + d - 3)/2d \rfloor} < ad - 2^k < a\left(\frac{d+1}{2}\right) - b \qquad dla \;\; ad - 2^k < 0$$

$$a\left(\frac{d-1}{2}\right) - b \le ad - 2^k < \frac{a(d+1) - 2b}{2\lfloor (2^{n+1} + d - 1)/2d \rfloor} \qquad dla \;\; ad - 2^k > 0$$

Dla *faithful rounding*:

$$\left\lfloor \frac{2^n}{d} \right\rfloor (2^k - ad) \le b < 2^k \qquad dla \;\; ad - 2^k < 0$$

$$\left\lfloor \frac{2^n}{d} \right\rfloor (ad - 2^k) < 2^k - b \qquad dla \quad ad - 2^k > 0$$

W celu zmniejszenia złożoności obliczeniowej algorytmu należy zminimalizować ilość iloczynów cząstkowych w *ax+b*, którego ilość zależy od rozmiaru tablicy iloczynów cząstkowych długości *n+k*. Minimalizując *k* otrzymamy przedział możliwych wartości *b*, spośród którego wybieramy *b* o najniższej wadze Hamminga, a następnie spośród nich ten o najmniejszej wartości. Waga Hamminga to w naszym przypadku suma jedynek w liczbie binarnej.

Optymalizując *k* oraz *b* otrzymujemy następujące wzory dla *round towards zero*:

Dla $ad - 2^k > 0$:

$$k_{opt}^{+} = min\left(k : \frac{2^k}{(-2^k) \bmod d} > d\left\lfloor\frac{2^n}{d}\right\rfloor - 1\right)$$

$$a_{opt}^{+} = \left\lceil\frac{2^{k_{opt}^{+}}}{d}\right\rceil$$

$$b_{opt}^{+} = 0$$

Dla $ad - 2^k < 0$:

$$k_{opt}^{-} = min\left(k : \frac{2^k}{2^k \bmod d} > d\left\lfloor\frac{2^n}{d}\right\rfloor - d + 1\right)$$

$$a_{opt}^{-} = \left\lfloor\frac{2^{k_{opt}^{-}}}{d}\right\rfloor$$

$$b_{opt}^{-} = \mathrm{minh}\left(\left(2^{k_{opt}^{-}} - a_{opt}^{-}d\right)\left\lfloor\frac{2^n}{d}\right\rfloor, 2^{k_{opt}^{-}} - a_{opt}^{-}(d-1) - 1\right)$$

W przypadku *round to even, faithful rounding:*

$$\begin{aligned}(k, a, b) = (k^{+} < k^{-}) \quad &? \quad \left(k^{+}, a^{+}, \mathrm{minh}\left(\Upsilon^{+}(k^{+}, a^{+})\right)\right)\\ &: \quad \left(k^{-}, a^{-}, \mathrm{minh}\left(\Upsilon^{-}(k^{-}, a^{-})\right)\right)\end{aligned}$$

Gdzie:

$$k^{+} = min\left(k : \frac{2^k}{(-2^k) \bmod d} > X^{+}\right)$$

$$k^{-} = min\left(k : \frac{2^k}{2^k \bmod d} > X^{-}\right)$$

$$a^{+} = \left\lceil\frac{2^{k^{+}}}{d}\right\rceil \qquad a^{-} = \left\lfloor\frac{2^{k^{-}}}{d}\right\rfloor$$

Definicje $X^{\pm}$ oraz $Y^{\pm}$ znajdują się w poniższej *tabeli 1*:

Tabela 1. Definicje $X^{\pm}$ oraz $Y^{\pm}$ dla RTZ, FR, RTE

	RTZ	FR
X^+	$d\lfloor 2^n/d \rfloor - 1$	$\lfloor 2^n/d \rfloor$
X^-	$d\lfloor 2^n/d \rfloor - d + 1$	$\lfloor 2^n/d \rfloor$
$Y^+(k,a)$	$(0,0)$	$(0,0)$
$Y^-(k,a)$	$((2^k - ad)\lfloor 2^n/d \rfloor,$ $2^k - a(d-1) - 1)$	$((2^k - ad)\lfloor 2^n/d \rfloor, 2^k - 1)$

	RTE
X^+	$d\lfloor (2^{n+1} - d - 1)/(2d) \rfloor - 1$
X^-	$d\lfloor (2^{n+1} - d - 3)/(2d) \rfloor + 1$
$Y^+(k,a)$	$(a(d-1)/2 + (2^k - ad),$ $a(d-1)/2 + (2^k - ad)\lfloor (2^{n+1} + d - 1)/(2d) \rfloor - 1)$
$Y^-(k,a)$	$(a(d-1)/2 + (2^k - ad)\lfloor (2^{n+1} + d - 3)/(2d) \rfloor,$ $a(d+1)/2 + 2^k - ad - 1)$

RNS (*Residue Number System*, system resztowy) – system liczbowy, która pozwala reprezentować liczby całkowite za pomocą wektora reszt z dzielenia względem ustalonego wektora modułów wzajemnie pierwszych. Reprezentacja taka jest jednoznaczna dla wszystkich liczb całkowitych ze zbioru $[0, M]$, gdzie M jest iloczynem wszystkich modułów.

Niech: $B = (m_1, m_2, \dots, m_n)$ – baza względnie pierwszych modułów, wtedy reprezentacją liczby X w systemie resztowym o bazie B jest $(x_1, x_2, \dots, x_n)$ gdzie $x_i = X \bmod m_i$ dla każdego $1 \le i \le n$

2. Cel projektu

2.1 Celem projektu była implementacja algorytmu dzielenia przez stałą opisanego w załączonej do tematu publikacji, z wykorzystaniem systemu RNS (system resztowy), jednak nie udało nam się tego systemu zaimplementować.

2.2 Po zaimplementowaniu algorytmu w C++, kolejne zadanie polegało na wyznaczeniu maksymalnego zakresu (n) dla danego k i każdej dostępnej wartości dzielnika, dokładnie rozpisane poniżej:

$$f(k) = max\left\{ n: \bigwedge_{d\in(3,2^{n-1})} \bigwedge_{x\in(2d+1,2^n)} \bigvee_{a,b} \left\lfloor \frac{x}{d} \right\rfloor = \left\lfloor \frac{ax+b}{2^k} \right\rfloor \right\}$$

$$k \in (16{,}42), \qquad d \bmod 2 \equiv 1, \qquad k, d, n, x, a, b \in \mathbb{Z}$$

3. Przebieg prac

3.1 Część pierwsza

Prace nad projektem zaczęły się od dokładnego zapoznania się z dokumentem „*Correctly Rounded Constant Integer Division via Multiply-Add*”. Program został napisany w C++, a implementację algorytmów rozpoczęto zgodnie z kolejnością występowania ich w publikacji. Stworzono funkcje wyznaczające współczynniki *a* oraz *b* dla zadanych sposobów zaokrąglania, a następnie funkcje optymalizujące *b* względem wagi hamminga. Podczas tworzenia funkcji znajdującej optymalną wartość *k* okazało się, że część napisanego wcześniej kodu nie jest po wykonaniu tych optymalizacji potrzebna. Przykładem niech będzie wyznaczanie b w metodach RTZ i FR. Po minimalizacji k okazuje się, że b można przyjąć jako 0 i algorytm działa wtedy poprawnie.

Ogólny opis przebiegu działania algorytmu:

- Dla danego *n* i *d*, gdzie 2^n to górna granica naszego przedziału, a *d* to dzielnik, wyznaczamy minimalne dostępne *k*, zależne od wybranej metody zaokrąglania.

- Na podstawie *k* obliczamy *a* za wzoru: $a^+ = \left\lceil \frac{2^{k^+}}{d} \right\rceil$, $a^- = \left\lfloor \frac{2^{k^-}}{d} \right\rfloor$.

- Dalej na podstawie *a, k, n, d* oraz ‘znaku’ *a* stosujemy konkretną funkcję do obliczenia *b*.

- Po wyznaczeniu współczynników *a, b* można je przetestować zarówno pod względem poprawności jak i szybkości działania w porównaniu do standardowego dzielenia, co jednak nie stanowi już integralnej części algorytmu.

3.2 Część druga

Druga część projektu polegała na wyznaczaniu maksymalnego *n* dla wybranego *k*, tak jak opisano to w celach projektu.

Z racji na zmianę funkcji celu, samo działanie algorytmu także zostało zmodyfikowane. Do eksperymentów wybrano tylko metodę zaokrąglania w kierunku zera. Działanie algorytmu wyglądało następująco:

- Dla danego *k* i każdego dzielnika w aktualnym zakresie dozwolonym przez *n*, wyznaczano *a*, *b* i dla każdej liczby z przedziału [2d + 1, 2^n) sprawdzano czy wynik uzyskany w wyniku działania algorytmu pokrywa się z rzeczywistym wynikiem. Jeżeli dla całego przedziału wynik się zgadzał, zapisywano nowe *n* jako to maksymalne.

Z racji, że w tym algorytmie nie mamy czegoś takiego jak k^+, k^- , *a* musi być wyznaczane w minimalnie zmodyfikowany sposób:

$a = \left\lfloor \frac{2^k}{d} \right\rfloor$, jeżeli a mod 2 $\equiv$ 0 , to a = a +1. W praktyce to samo dzieje się w podstawowym algorytmie tylko w momencie wyznaczania k, przez co a jest potem zdeterminowane. Ta implementacja bezpośrednio wpływa nam na sposób wyliczania *b*, który zależy od znaku ad - 2^k. W trakcie eksperymentów przeprowadzanych w tej części, z racji na złą interpretację otrzymywanych wyników, zdecydowano się na lekką modyfikację działania algorytmu, co ku naszemu zdziwieniu pozwoliło na zwiększenie zakresu n (szczegóły wyników zostaną omówione we wnioskach). Modyfikacja była następująca:

- Jeżeli a $mod\ 2\ \equiv\ 1$, liczymy b wzorem dla ad - $2^k < 0$, jeżeli zakresy które wyjdą dla b okażą się sprzeczne, to znaczy*:* *b* > *bstart*, *b* < *bend*, *bend* < *bstart*, gdzie *bstart* i *bend* to początek i koniec dopuszczalnego zakresu dla *b*, to *a = a +1* i liczymy *b* wzorem dla *ad* - $2^k > 0$.

- Jeżeli a $mod\ 2\ \equiv\ 0$, *a = a -1*, liczymy b wzorem dla *ad* - $2^k < 0$, jeżeli zakresy są sprzeczne (jak wyżej), a = a +1, dalej liczymy b wzorem dla *ad* - $2^k < 0$, jeżeli ponownie zakresy są sprzeczne,
a = a+1 i liczymy b wzorem dla *ad* - $2^k > 0$.

4. Wnioski

4.1 Podstawowy algorytm

Po pełnym zaimplementowaniu wszystkich elementów algorytmu opisanych w dokumencie, wszystko działa poprawnie dla każdego typu zaokrąglania. W implementacji w C++ nie jest widoczna znacząca poprawa w szybkości wykonywania algorytmu w porównaniu do standardowego dzielenia. Zakres działania algorytmu jest z góry ograniczony zakresem typu int. Z tego powodu maksymalna wartość *n* dla której algorytm powinien działać w ogólności to *n* = 16. Jeżeli przejdziemy na typ long long, granica ta przesunie się na 32 itd. Powód tej zależności dla przykładu inta opisano poniżej:

$\max number: \quad a(2^n-1)+b \;-\; największa\ liczba\ występująca\ w\ algorytmie$

$b < \frac{2^k}{d}$ - założenie dla wielkości b

$$\left\lfloor\frac{2^k}{d}\right\rfloor(2^n-1)+b < \frac{2^{n+k}-2^k}{d}+b = \frac{2^{n+k}}{2^{\lfloor\log_2 d\rfloor}} = 2^{n+k-\lfloor\log_2 d\rfloor} < 2^{32}$$

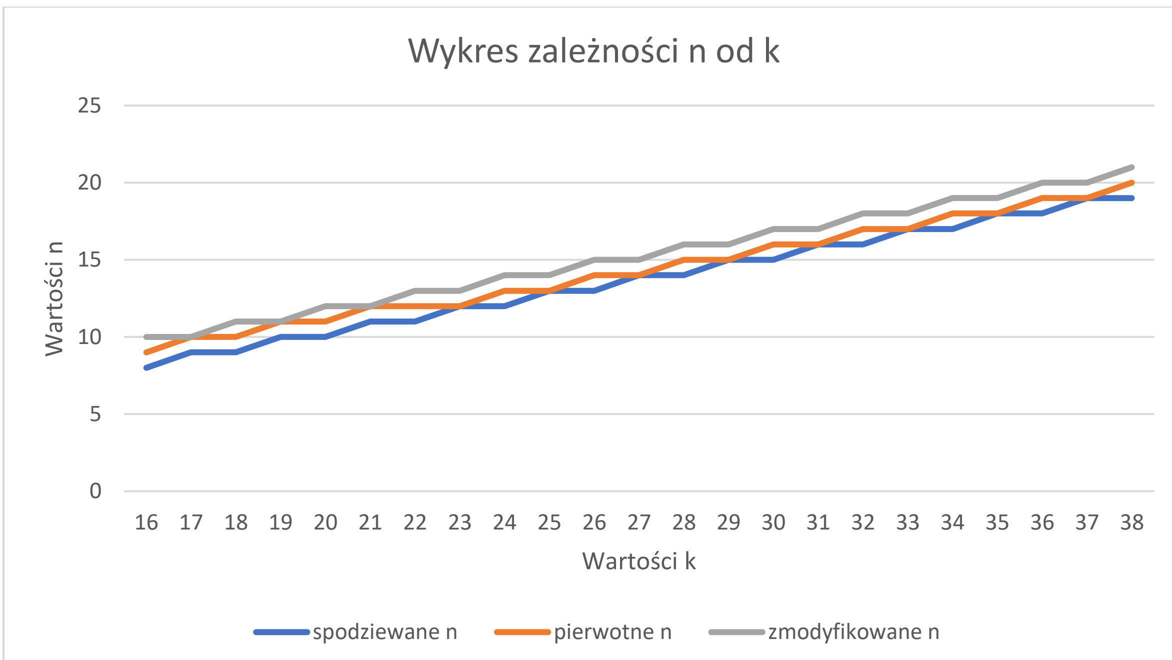
Z racji, że *k* nie jest dowolne, tylko wyznaczane przez algorytm - tylko dla niektórych wartości *d, k* może być wystarczająco małe, aby *n* było większe od 16.

4.2 Wyznaczanie maksymalnego zakresu w zależności od *k*.

Jak zostało wcześniej wspomniane do tych eksperymentów wykorzystano metodę zaokrąglania do zera. W trakcie przeprowadzania eksperymentów z racji na złą wizję spodziewanych wyników, zaczęto niepotrzebnie modyfikować algorytm co zostało opisane w sekcji 3. W wyniku tego powstały 2 różne zestawy danych oraz dane które spodziewano się otrzymać, zatem przedstawiono 3 wykresy.

Tabela 2. Dane do wykresu

k	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
spodziewane n	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	18	19	19
pierwotne n	9	10	10	11	11	12	12	12	13	13	14	14	15	15	16	16	17	17	18	18	19	19	20
Zmodyfikowane n	10	10	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	18	19	19	20	20	21



Wykres 1. Zależność n od k

Wartości n na wykresie ograniczone są poprzez k, które musi spełniać następującą nierówność:

$$\frac{2^k}{2^k \bmod d} > d\left\lfloor\frac{2^n}{d}\right\rfloor - d + 1$$

Problem wynikającym z tego ograniczenia jest brak możliwości znalezienia współczynnika *b* dla którego algorytm dzieliłby poprawnie. Możliwe, że w wyniku modyfikacji algorytmu którą opisano trochę wcześniej, zwiększono zakres przeszukiwań b, co dało nam poprawne wyniki w większej ilości przypadków.

4.3 Wyznaczanie dopuszczalnych wartości d w zależności od k, n

Prace w tym zakresie nie zostały dokończone, więc przedstawię tylko koncepcję. Tak jak zostało wcześniej opisane: maksymalne n, dla którego algorytm będzie działał w ogólności wynosi 16, jednakowoż nie jest to maksymalna wartość dla każdego dzielnika. Z racji na to, że dużo wielkości zależy tutaj od reszt z dzielenia, nie da się wyprowadzić tutaj prostego wzoru jawnego na dostępne *d* w zależności od *n* i *k*. Można jednak wyprowadzić kilka zależności. Tutaj ponownie naszym górnym ograniczeniem będzie zakres typu int, a dolnym ograniczeniem dopuszczalne wielkości k. Dla przypomnienia:

Górne ograniczenie: $2^{n+k-\lfloor \log_2 d \rfloor} < 2^{32}$

Dolne ograniczenie: $\frac{2^k}{2^k \bmod d} > d \left\lfloor \frac{2^n}{d} \right\rfloor - d + 1$

Dolne ograniczenie można sprowadzić do:
$2^k > 2^n - d \; dla \; 2^k \bmod d \; \equiv 1$,
$2^{k-1} > 2^n - d \; dla \; 2^k \bmod d \; \equiv 2$,
$2^{k-\lceil \log_2(2^k \bmod d) \rceil} > 2^n - d$, dla $2^k \bmod d > 2$

Powyższe wyprowadzanie sprowadzają się do tego, że jeżeli znajdziemy *d* wystarczająco duże, żeby spełniało górne ograniczenie, oraz odpowiednie dla drugiego ograniczenia, jesteśmy wstanie wykonywać dzielenie dla zakresu n>16.
W ramach znajdowania d zaprojektowałem następujący prosty algorytm:

- Założenia: n >16 (tylko wtedy jest sens to wykonywać), k>=n

Działanie: (Pierwsze 3 kroki to przypadek szczególny, kolejne to ogólny)
1. Zaczynamy od k = n
2. Szukamy wszystkich dzielników $2^n - 1$
3. Jeżeli znaleziony dzielnik z tej puli spełnia górne ograniczenie dla danego n i k, to jest on dopuszczalnym dzielnikiem.
4. Zwiększamy k o 1
5. Szukamy wszystkich dzielników dla liczb z zakresu $[2^n - 2^{k-n}, 2^n - 1]$
6. Jeżeli znaleziony dzielnik z tej puli spełnia górne ograniczenia, a także jest mniejszy od 2^{n-1}, to jest on dopuszczalnym dzielnikiem.
7. Powtarzaj punkty 4-6 aż do k = 31, lub innej sensownej granicy
8. Zmieniaj n aż uzyskasz wszystkie satysfakcjonujące wartości

Z racji na to, że teoretycznie im k jest mniejsze tym lepiej, jeżeli ten sam dzielnik pojawi się ponownie w danej iteracji (dla tego samego *n*), nie należy nadpisywać k (o ile jest to gdzieś zapisywane).

Dla algorytmu przeprowadzono tylko pojedyncze próby więc możliwe, że nie jest on perfekcyjny, jednakże pozwala on znaleźć dopuszczalne wartości dzielników dla n>16.

5. Literatura:

Biernat, J., 2005. Architektura Komputerów

Drane, T., Cheung W., Constantinides, G., 2012. Correctly rounded constant integer division via multiply-add