

Mnożenie liczb całkowitych poniżej $n \log n$

Ilustrowany przewodnik

Rohan Arun

na podstawie pracy OpenAI *Integer multiplication below $n \log n$* (23 września 2026)

Przekład na język polski: hipotures, wygenerowany z pomocą Claude (Anthropic)

Polskie wydanie, 12 października 2026

Streszczenie

Przekład przewodnika *Integer multiplication, illustrated* Rohana Aruna (<https://github.com/rohanarun/integer-multiplication-illustrated>), objaśniającego oryginalną argumentację z 24 rysunkami, przykładami obliczeniowymi i wyraźnym rozróżnieniem między twierdzeniami źródła, rachunkami poglądowymi i warunkowymi interfejsami dowodu. Według noty autora objaśnienia i rysunki przygotowano przy istotnej pomocy OpenAI Codex. To nie jest nowy dowód ani oficjalna rewizja pracy źródłowej. Wynik OpenAI nie przeszedł jeszcze niezależnej recenzji.

Licencja: Apache 2.0. Zmiany względem oryginału: przekład tekstu i etykiet rysunków; dodany słowniczek polsko-angielski. Wersja internetowa z widżetami interaktywnymi: <https://armum.eu/posts/mnozenie-liczb-calkowitych-1-problem-i-splot/>.

Spis treści

Zacznij od problemu, nie od symboli	3
Pytanie, twierdzenie i jak to czytać	4
Zobacz dokładnie, które iloczyny cyfr się sumują	6
Od mnożenia pisemnego do splotu	7
Transpozycja macierzy to fizyczne przestawienie	9
Dlaczego przesuwanie danych jest częścią matematyki	10
Wektory, rząd i rama, którą można narysować	11
Pożyczona pamięć robocza i zmiana reprezentacji	12
Jak znoszą się niechciane źródła	15
Dlaczego więcej wywołań może oznaczać mniej pracy	16
Jak skończona oszczędność rzędu staje się wykładnikiem	17

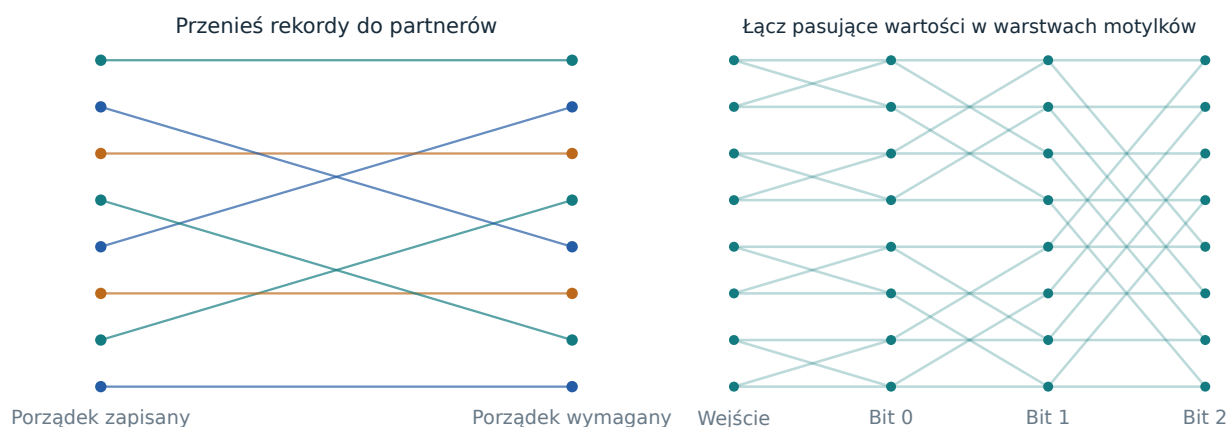
Co naprawdę zmienia transformata	18
Najmniejszy krok transformaty, zastosowany na wielu osiach	20
Pierwiastki z jedynki jako przesunięcia ze znakiem	21
Zmiana siatki bez sortowania wszystkiego	22
Mały błąd musi pozostać mały po przeskalowaniu	24
Dlaczego poprawienie jednej części może niczego nie zmienić	25
Dokładność i czas działania razem	26
Co właściwie mówi udane sprawdzenie	28
Co z tego wynika i co wciąż trzeba sprawdzić	29
Oznaczenia i mapa źródeł	30
Od tłumacza: słowniczek polsko-angielski	31
Źródła	33

Zacznij od problemu, nie od symboli

Podstawy Przy pierwszym czytaniu wystarczy zwykła arytmetyka.

Masz dwa długie ciągi cyfr dwójkowych i chcesz poznać ich dokładny iloczyn. Pytanie badawcze dotyczy tego, jak rośnie potrzebna praca, gdy ciągi stają się coraz dłuższe. Nie chodzi o to, czy mnożenie jest w ogóle możliwe, i nie jest to obietnica, że nowy program jutro prześcignie twój kalkulator.

Pomyśl o trzech rachunkach. Jeden płacisz za łączenie liczb, drugi za przesuwanie właściwych kawałków obok siebie, a trzeci za upewnienie się, że wynik końcowy jest dokładny. Praca proponuje sposób, by obniżyć dwa pierwsze rachunki jednocześnie, nadal płacąc trzeci. Ilustracje poniżej pokazują rzeczywiste ścieżki danych: lewy rysunek zmienia miejsce przechowywania rekordów, a prawy łączy pary rekordów łączonych na kolejnych poziomach transformaty.



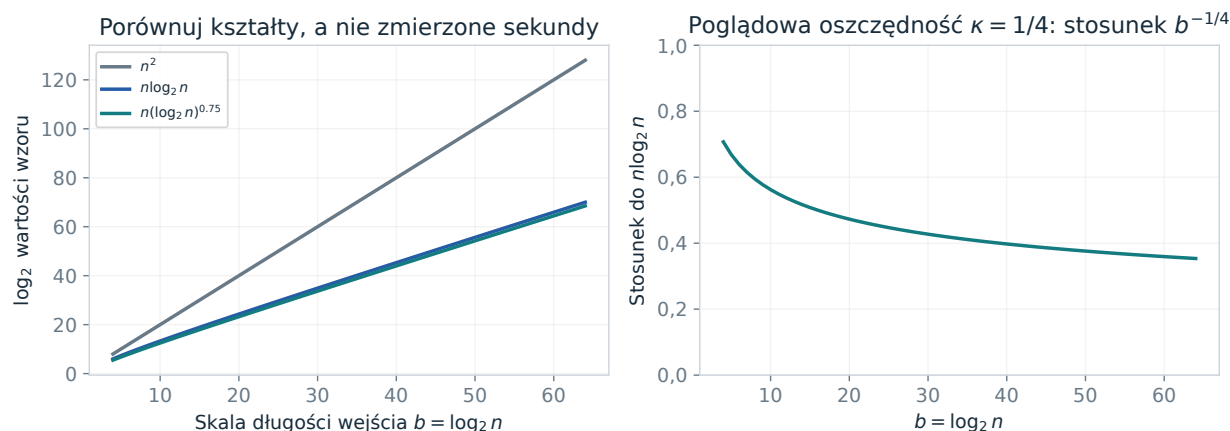
Rys. 1. Dwa odrębne zadania w szybkim mnożeniu. Po lewej: rekordy trzeba ustawić w użytecznej kolejności. Po prawej: ośmioelementowa sieć motylków na każdym poziomie łączy elementy różniące się jednym bitem adresu. To małe ilustracje struktury, a nie ogromny układ z pracy.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Czym jest bit? Bit to zero albo jedynka. W systemie dziesiętnym kolejne miejsca to jedności, dziesiątki, setki. W dwójkowym: jedynki, dwójki, czwórki, ósemki. Zatem dwójkowe 1011 oznacza $1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 11$. To wejście ma cztery bity. Praca oznacza przez n długość w bitach, a nie wartość liczby. Wiodące zera są dozwolone: 0011 też jest wejściem czterobitowym, choć jego wartość to trzy.

Co liczy logarytm? Dla potęg dwójki $\log_2 n$ liczy, ile podwojeń prowadzi od jedynki do n . Ponieważ ciąg 1, 2, 4, 8, 16 to cztery podwojenia, $\log_2 16 = 4$. Rozróżnienie między n a $\log_2 n$ jest kluczowe. Gdy ciąg bitów wydłuża się dwukrotnie, n się podwaja, a $\log_2 n$ rośnie zaledwie o jeden.

Co znaczy wzór na czas działania? Mnożenie szkolne zestawia każdą z mniej więcej n cyfr z mniej więcej n innymi, co daje około n^2 elementarnych iloczynów cyfr. Bardziej wyrafinowane algorytmy unikają liczenia wszystkich tych par osobno. Ograniczenie w rodzaju $O(n \log n)$ mówi, że powyżej pewnego progu praca wynosi co najwyżej pewną stałą razy $n \log n$. Nie określa ani tej stałej, ani progu, ani liczby sekund na konkretnym komputerze.



Rys. 2. Tempa wzrostu trzeba oglądać we właściwej skali. Po lewej: oś pionowa to logarytm wartości wzoru, więc wartości różniące się wykładniczo mieszczą się na jednym wykresie. Po prawej: podzielenie poglądowego, ulepszanego wzoru przez $n \log_2 n$ odsłania jego wolniejszy wzrost. Oszczędność $1/4$ jest celowo na tyle duża, żeby była widoczna; oryginalna praca używa 2^{-182} . Żaden panel nie jest pomiarem programu mnożącego.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Dwa sposoby czytania. Przy pierwszym czytaniu idź za przykładami, rysunkami i ramkami „Co zabrać dalej”. Budują one słownictwo, zanim użyje go techniczny wywód. Przy drugim czytaniu przeczytaj równania, a potem odpowiadające im oryginalne dowody w wydaniu rozszerzonym. Zrozumienie, dlaczego elementy do siebie pasują, to inny cel niż sprawdzenie każdego twierdzenia. Przewodnik służy pierwszemu celowi i wskazuje drogę do drugiego; nie zamienia pełnego dowodu w elementarną arytmetykę.

Szybki sprawdzian. Jeśli wejście rośnie z 16 do 32 bitów, n się podwaja, a $\log_2 n$ rośnie z czterech do pięciu. Wartość liczby wejściowej może być nieporównanie większa niż 32. Trzymanie tych trzech wielkości osobno zapobiega wielu późniejszym nieporozumieniom.

Co zabrać dalej. Praca próbuje zaoszczędzić odrobinę pracy na ogromnej liczbie skal. Żeby ją zrozumieć, śledź zarówno arytmetykę, jak i ruch danych.

Pytanie, twierdzenie i jak to czytać

Technika Zaczynj tutaj; znajomość analizy Fouriera nie jest potrzebna.

Mnożenie wydaje się sprawą zamkniętą, kiedy uczymy się go w szkole. Liczba elementarnych kroków potrzebnych do pomnożenia dwóch niezwykle długich ciągów dwójkowych to jednak zupełnie inne pytanie. Długość wejścia to liczba bitów, a nie wartość liczbową. Liczba o milionie bitów jest jednym wejściem długości miliona; to, że jej wartość może być bliska $2^{1000000}$, niczego w tej długości nie zmienia.

Omawiany tu manuskrypt to praca OpenAI *Integer multiplication below $n \log n$* z 23 września 2026 roku, w wersji ze źródłowego commitu `adc7f1241b42e322a6451854ab7e4b4c146bf78a`. Po-daje ona pesymistyczne ograniczenie

$$T(n) = O(n(\lg n)^{1-\kappa}), \quad \kappa = 2^{-182}.$$

Tutaj $\lg n = \max\{\lceil \log_2 n \rceil, 1\}$. Maszyna musi zwrócić dokładny iloczyn dla każdego wejścia, także dla wejść z wiodącymi zerami. Model maszyny używa ustalonego skończonego alfabetu i

ustalonej skończonej liczby jednowymiarowych taśm. Odczyt, zapis i przesuwanie danych kosztują czas.

Co to wydanie ustala, a czego nie. Oryginalny tekst formułuje swój wynik jako twierdzenie. To wydanie objaśnia argumentację, ale nie potwierdza niezależnie jej poprawności. Repozytorium społeczności traktuje swoje ulepszenia jako warunkowe, zależne od oryginalnej konstrukcji oraz zachowanych interfejsów implementacyjnych i analitycznych. To rozróżnienie jest ważne: sprawdzenie dokładnej skończonej sieci albo listy nierówności wymiernych nie dowodzi samo w sobie, że jedna maszyna taśmowa realizuje cały algorytm w deklarowanym czasie dla wszystkich rozmiarów wejścia. Twierdzenia oryginału pozostają w pełnym wydaniu tekstem źródłowym i przewodnik nie przedstawia ich jako nowych wyników. To niezależnie przygotowane wydanie objaśniające, a nie oficjalna rewizja ani rekomendacja OpenAI.

Dlaczego maleńki dodatni wykładnik ma znaczenie. Pomijając na chwilę stałe, podzielimy proponowane ograniczenie przez $n \log_2 n$. Iloraz to $(\log_2 n)^{-\kappa}$, który dąży do zera dla każdego ustalonego $\kappa > 0$. To rozróżnienie asymptotyczne, choćby κ było nie wiadomo jak małe. Nie jest to zapowiedź użytecznego przyspieszenia na komputerze. Nawet sprowadzenie tego wyidealizowanego stosunku do $1/2$ wymaga

$$(\log_2 n)^\kappa = 2, \quad \log_2 n = 2^{1/\kappa}, \quad n = 2^{2^{1/\kappa}}.$$

Przy $\kappa = 2^{-182}$ to niewyobrażalna skala. Nieznane stałe, progi i rzeczywista implementacja sprawiają, że praktyczne porównanie jest jeszcze trudniejsze. Większe κ oznacza lepszą oszczędność w wykładniku, a nie taką samą procentową poprawę zmierzonego czasu działania.

Argumentacja w jednym akapicie. Zamień mnożenie cyfr na splot. Za pomocą transformat zamień splot na mniejsze iloczyny punktowe. Spraw, żeby dwa kosztowne składniki stały się tańsze: przedstawianie adresów przechowywanych danych i wykonywanie wielu dwupunktowych kroków transformaty. Oba ulepszenia biorą się ze skończonych sieci liniowych, których pośrednie reprezentacje dobrano starannie. Udowodnij oszczędność rekurencyjną po wliczeniu każdej roli pomocniczej i każdej zmiany reprezentacji. Następnie pokaż, że układy danych, przepróbkowanie, precyzja i dokładne odtworzenie wyniku mieszczą się w tym samym końcowym ograniczeniu.

Droga przez wydanie rozszerzone. Nowe rozdziały podstaw wprowadzają każde potrzebne pojęcie na konkretnym przykładzie, zanim pojawi się ono w części technicznej. Najpierw przeczytaj przykład z cyframi i objaśnienie dotyczące taśm. Potem odtwarzanie rejestru pomocniczego, wspólne ramy i rekurencję – one wyjaśniają, skąd bierze się proponowana oszczędność. Przykłady z motylkami i wielomianami pokazują, co tę oszczędność zjada. Ostatnie rozdziały tłumaczą, dlaczego porządkowania i zaokrąglania nie można zostawić na koniec. W pełnym wydaniu każde dodane objaśnienie poprzedza oryginalny materiał techniczny, który oświeśla. Numeracja sekcji, równań, twierdzeń i rysunków oryginału jest zachowana.

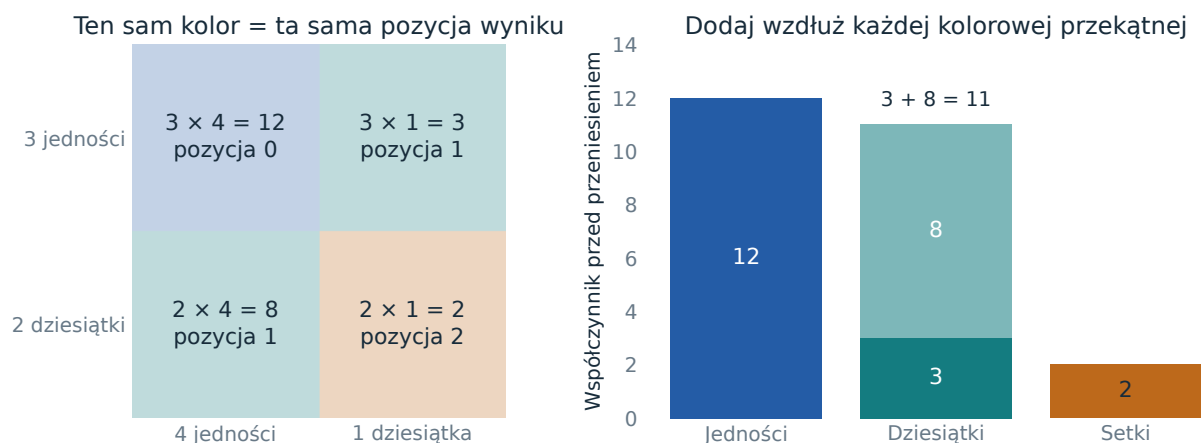
Co zabrać dalej. Główne pytanie brzmi: czy całe obliczenie, łącznie z ruchem danych i dokładnym odtworzeniem wyniku, może zaoszczędzić dodatnią potęgę $\log n$?

Źródło i atrybucja. Oryginalna praca i jej bibliografia są zachowane na licencji Apache 2.0 w dołączonym pakiecie źródłowym. Jej część historyczna przypisuje zasługi m.in. Schönhagemu i Strassenowi, Harveyowi i van der Hoevenowi, Nussbaumerowi i Quandalle'owi. Repozytorium Croc-Swap osobno zachowuje wkład społeczności. Nowe objaśnienia, rysunki i sprawdzenia przykładów w tym wydaniu przygotowano z pomocą OpenAI Codex 12 października 2026 roku. Źródło jest zamrożone; ten dokument nie jest bieżącym rankingiem rekordów.

Zobacz dokładnie, które iloczyny cyfr się sumują

Podstawy Splot bez nowej maszynerii.

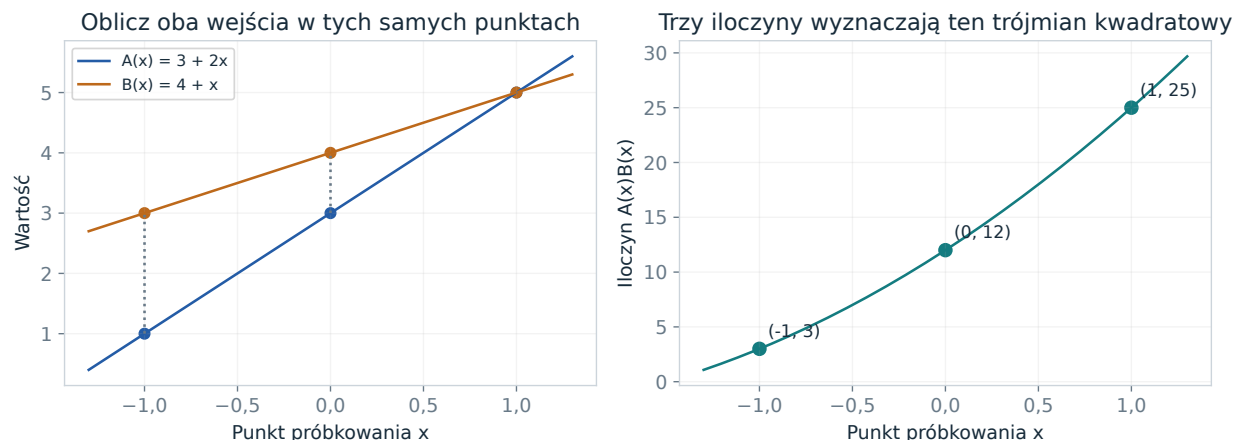
Każdy składnik mnożenia pisemnego ma swoje miejsce docelowe. W 23×14 iloczyn 3×4 trafia do jedności. Zarówno 3×1 , jak i 2×4 trafiają do dziesiątek, bo jeden czynnik w każdym z tych iloczynów jest cyfrą dziesiątek. Wreszcie 2×1 trafia do setek. Kolory przekątnych w siatce wskazują te miejsca docelowe.



Rys. 3. Splot to dodawanie wzdłuż przekątnych o tej samej pozycji. Dwie turkusowe komórki wnoszą 3 i 8 do tego samego współczynnika, co daje 11. Lista współczynników przed przeniesieniami to $[12, 11, 2]$. Komórka to konkretny iloczyn cyfr, a nie ogólny etap algorytmu. Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Ponumeruj pozycje cyfr od zera. Cyfra na pozycji i pomnożona przez cyfrę na pozycji j trafia na pozycję $i+j$, bo $10^i 10^j = 10^{i+j}$. Na tym polega cała reguła „dodawaj iloczyny o tej samej sumie pozycji”. Nazwa *splot* oznacza właśnie tę regułę. Działa ona tak samo w systemie dwójkowym i w każdej innej podstawie.

Po co przechodzić do wielomianu? Zastąp podstawę dziesięć zmienną x . Oba wejścia stają się wielomianami $A(x) = 3 + 2x$ i $B(x) = 4 + x$. Ich iloczyn jest kwadratowy: można go zapisać jako $C(x) = c_0 + c_1x + c_2x^2$ z trzema nieznanymi współczynnikami. Dla $x = 10$ musi dać iloczyn liczb całkowitych. Ale dziesięć to niejedyny przydatny punkt, w którym można go obliczyć. Łatwiejsze punkty próbkowania mogą ujawnić jego współczynniki.



Rys. 4. Najpierw policz wartości, potem odtwórz współczynniki. W punktach $x = -1, 0, 1$ iloczyny wartości wejść wynoszą 3, 12 i 25. Przez trzy różne punkty przechodzi dokładnie jeden wielomian stopnia co najwyżej drugiego. Krzywe to te konkretne wielomiany, a nie metafora transformaty. Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

W zerze $C(0) = c_0 = 3 \times 4 = 12$. W jedynce $C(1) = 5 \times 5 = 25$. W minus jedynce $C(-1) = 1 \times 3 = 3$. Zatem

$$c_0 = 12, \quad c_1 = \frac{25 - 3}{2} = 11, \quad c_2 = \frac{25 + 3}{2} - 12 = 2.$$

Odjęcie dwóch ostatnich równań wyodrębnia wyraz z nieparzystą potęgą, a ich dodanie – wyrazy z parzystymi potęgami. Obliczenie w dziesięciu daje teraz $12 + 11 \cdot 10 + 2 \cdot 100 = 322$.

Gdzie ten przykład się kończy. Ten mały przykład interpolacji to nie szybki algorytm z pracy. Pokazuje tylko, dlaczego schemat „oblicz wartości, pomnóż odpowiadające sobie wartości, odtwórz współczynniki” w ogóle może działać. Dla ogromnych wejść samo obliczanie wartości i odtwarzanie muszą być szybkie i dobrze kontrolowane. Transformaty Fouriera wybierają silnie ustrukturyzowane punkty próbkowania, dzięki czemu te kroki mogą dzielić się pracą. Kolejny rozdział podstaw (w części 2) wyjaśnia, jak wyglądają ich wzorce.

Co zabrać dalej. Splot grupuje iloczyny trafiające na to samo miejsce. Transformata daje inny sposób obliczenia tych samych współczynników, ale nie zmienia iloczynu.

Od mnożenia pisemnego do splotu

Technika Kompletny mały przykład.

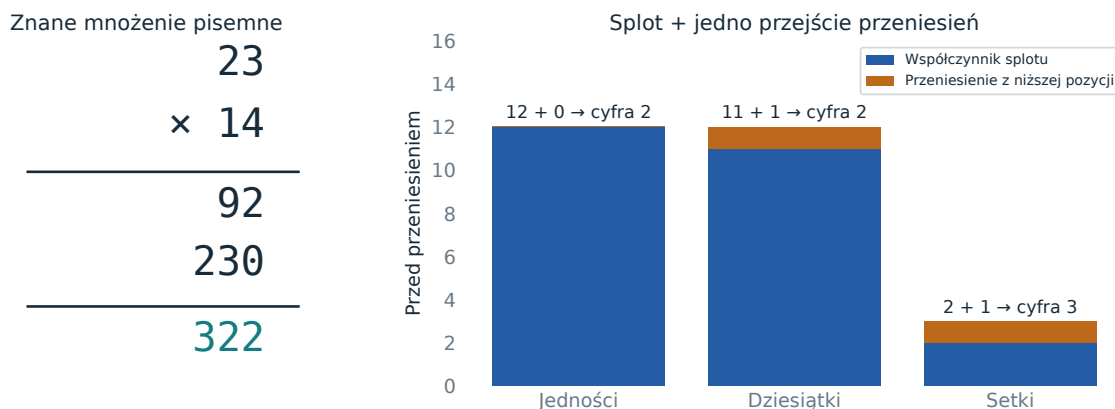
Zapisz liczbę całkowitą w podstawie B jako $a_0 + a_1B + \dots + a_{q-1}B^{q-1}$. Cyfry są wymienione od najmniej do najbardziej znaczącej. W objaśnieniu przyjmijmy $B = 10$; praca używa potęgi dwójki, żeby podział na cyfry był operacją dwójkową. Wprowadź zmienną formalną x i utwórz $A(x) = \sum_i a_i x^i$. Wtedy oryginalna liczba to dokładnie $A(B)$.

Dla dwóch wejść współczynniki iloczynu wielomianów spełniają

$$A(x)B(x) = \sum_k c_k x^k, \quad c_k = \sum_{i+j=k} a_i b_j.$$

Drugi wielomian wejściowy oznaczamy tu $B(x)$; samo B bez nawiasu wciąż oznacza podstawę. Suma definiująca c_k to właśnie *splot*. Grupuje iloczyny cyfr o tej samej wartości pozycyjnej. W

odróżnieniu od cyfr współczynniki c_k mogą przekraczać podstawę. Naprawiają to przeniesienia, wykonane już po obliczeniu współczynników.



Rys. 5. Jedno mnożenie, trzy reprezentacje. Lista współczynników $[12, 11, 2]$ odpowiada potęgom x^0, x^1, x^2 . Przeniesienie przechodzi z niższych pozycji na wyższe. Wynik to 322. Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Dla $23 \cdot 14$ współczynnik przy x wynosi $3 \cdot 1 + 2 \cdot 4 = 11$. Na pozycji zero liczba 12 zapisuje cyfrę 2 i przenosi 1. Pozycja pierwsza dostaje $11 + 1 = 12$, znowu zapisuje 2 i przenosi 1. Pozycja druga dostaje $2 + 1 = 3$. To zwykle mnożenie, tylko inaczej zorganizowane – na razie bez żadnych przybliżeń.

Skąd biorą się transformaty. Dla odpowiedniej transformaty F splot można odtworzyć z iloczynów odpowiadających sobie elementów transformat. Obliczenie dwóch wielomianów w wielu pierwiastkach z jedynki, pomnożenie tych wartości i transformacja odwrotna zastępują mnóstwo iloczynów krzyżowych ustrukturyzowanymi dodawaniami i iloczynami punktowymi. Zysk zależy od tego, czy transformaty da się tanio obliczyć i tanio ułożyć ich dane wejściowe. Transformaty w pracy są znormalizowane, więc znane czynniki skali trzeba przywrócić; hasło „transformuj, pomnóż, odwróć” te czynniki ukrywa.

Dlaczego potrzebne jest dopełnianie zerami. Splot cykliczny (kołowy) traktuje pozycje modulo długość S . Wkład na pozycji S zawija się wtedy na pozycję zero. Algebraicznie to wygodne, ale zepsułoby iloczyn liczb całkowitych. Dla stopnia co najwyżej $2q - 2$ wybranie $S > 2q - 2$ i wypełnienie nieużywanych miejsc zerami zapobiega niezerowemu zawinięciu. Oryginalne składanie wyniku dowodzi, że wybrany iloczyn wzdłuż osi o długościach pierwszych ma wystarczającą pojemność. Ogranicza też każdy współczynnik, zanim wybierze precyzję do jego przechowania.

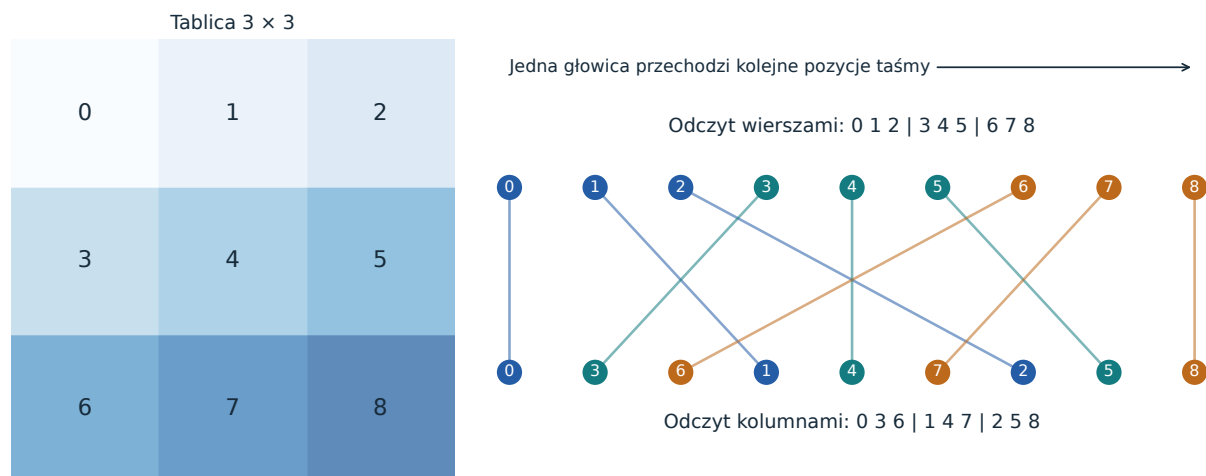
Co zabrać dalej. Trudne zadanie polega teraz na tym, żeby wszystkie współczynniki splotu policzyć tanio i na tyle dokładnie, by odtworzyć liczby całkowite, a potem raz wykonać przeniesienia.

Miejsca w oryginale: *00-introduction.tex*; *08-assembly.tex*, *eq:capacity*.

Transpozycja macierzy to fizyczne przestawienie

Podstawy Prześledź wszystkie dziewięć rekordów.

Odczytaj siatkę 3×3 wierszami: 0, 1, 2, 3, 4, 5, 6, 7, 8. Teraz odczytaj ją kolumnami: 0, 3, 6, 1, 4, 7, 2, 5, 8. Same rekordy się nie zmieniły. Zmieniło się ich położenie w strumieniu. Krzyżujące się ścieżki pokazują dokładną permutację, łącznie z rekordami, które zostają na miejscu.



Rys. 6. Geometria jednego konkretnego przeniesienia danych. Kolory oznaczają oryginalne wiersze, a każdy ponumerowany rekord jest połączony ze swoim miejscem docelowym. Strzałka u góry przedstawia głowicę taśmy przesuwaną się po kolejnych pozycjach. Głowica nie może teleportować się do następnego potrzebnego rekordu.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Program komputerowy może zapisać nowy wzór indeksowania w jednej linijce. To jednak nie dowodzi, że rekordy zostały fizycznie przestawione w jednym kroku. W modelu maszyny z pracy głowica przesuwa się w jednym kroku o co najwyżej jedną komórkę taśmy. Duży ruch danych trzeba więc zrealizować i za niego zapłacić.

Dlaczego obliczenie może pomóc w permutacji. Dla pary bitów trzy aktualizacje XOR zamieniają ich wartości:

$$x \leftarrow x \text{ XOR } y, \quad y \leftarrow y \text{ XOR } x, \quad x \leftarrow x \text{ XOR } y.$$

Dla stanu początkowego $(x, y) = (1, 0)$ kolejne stany to $(1, 0)$, $(1, 1)$, $(0, 1)$. XOR to dodawanie modulo dwa: równe bity dają zero, różne dają jedynkę. Wartości pośrednie są kombinacjami, a mimo to wynik końcowy jest czystą zamianą. Duża konstrukcja również używa pośrednich kombinacji liniowych, żeby uzyskać pożądaną operację na całej tablicy. Ta tożsamość dla dwóch bitów wyjaśnia jedynie, że jest to możliwe; nie dowodzi szybkości dużej konstrukcji.

Dwa znaczenia geometrii trzymaj osobno. Siatka przedstawia tu pozycje w tablicy. Podprzestrzenie i rzędy pojawiające się później przedstawiają algebraiczne kierunki operatorów działających na takich tablicach. Rysunek krótkiej fizycznej ścieżki nie jest sam w sobie świadectwem niskiego rzędu, a tożsamość niskiego rzędu nie jest sama w sobie ograniczeniem czasu na taśmie. Argumentacja potrzebuje precyzyjnego połączenia między nimi.

Co zabrać dalej. Kolejność przechowywania jest częścią problemu. Użyteczne przestawienie trzeba wykonać prawdziwymi krokami, a nie samą zmianą nazw indeksów.

Dlaczego przesuwanie danych jest częścią matematyki

Technika Do przeczytania przed sekcją o modelu maszyny.

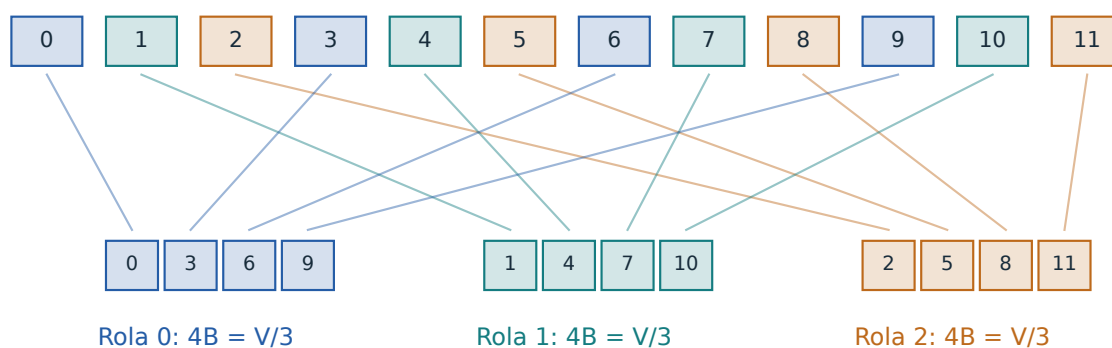
Wyobraź sobie długi rząd ponumerowanych pudełek i głowicę, która przesuwa się o jedno pudełko na krok. Numer to adres pojęciowy; maszyna nie dostaje darmowego dostępu swobodnego tylko dlatego, że na papierze możemy napisać $A[i, j]$. Tablica wielowymiarowa zostaje spłaszczona do jednego uporządkowanego strumienia. W porządku leksykograficznym najszybciej zmienia się ostatnia współrzędna. Dla tablicy 2×3 kolejność to

$$(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2).$$

Przejsięcie na odwrotną kolejność współrzędnych przestawia rekordy. Na tej maszynie koszt obejmuje doprowadzenie ich na te pozycje.

Rola to miejsce w układzie; wiersz to blok danych. Skończona sieć ma W ról, licząc zarówno role danych, jak i role pomocnicze. W węźle rekurencji kompletne wiersze są rozdzielane między nie cyklicznie. Wiersz zawiera cały sufix aktywnych pól adresu. Dzięki tej kompletności każda rola widzi ten sam lokalny zbiór adresów. Po odpowiednim dopełnieniu każda rola dostaje dokładnie ułamek $1/W$ logicznej objętości.

Całe wiersze są rozdawane cyklicznie; każdy zachowuje cały sufix adresu



12 wierszy $\times B$ bitów = łącznie V bitów. Wszystkie trzy strumienie ról mają zgodny kształt lokalny.

Rys. 7. Podział objętości, który napędza rekurencję. Każdy poglądowy wiersz ma B bitów. Dwanaście wierszy daje $V = 12B$; trzy strumienie ról mają po $4B$. Prawdziwa praca używa znacznie większych ustalonych sieci.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Dlaczego liczymy także role pomocnicze. Załóżmy, że liczylibyśmy tylko użyteczne wiersze wyniku, zapominając o pamięci roboczej. Konstrukcja mogłaby wtedy wyglądać na tanią tylko dlatego, że ukrywa dużą ilość danych tymczasowych. W w sieci liczy wszystkie role. Role pomocnicze mogą zawierać dowolne istniejące dane, bo układ skalarny przywraca ich zawartość. Nie zakładamy, że są darmową, wyzerowaną pamięcią. Późniejsza operacja na tablicy w ramie musi działać poprawnie także na tych rolach.

Objętość logiczna nie zwalnia z liczenia kopiowania. V oznacza objętość danych, którą normalizuje się wywołanie rekurencyjne. Fizyczne kopie, dopełnianie, aktualizacje liczników, powroty głowicy i sprzątanie wciąż zajmują czas. Są wliczane jako narzut. Gdy aktywne jest dziecko, harmonogram z pracy „parkuje” strumień rodzica i przywraca je po powrocie. Deklarowany rachunek wymaga, by nie przechodzić wielokrotnie przez wszystkich zaparkowanych przodków.

Ustalona maszyna wciąż może obsługiwać rosnące deskryptory. Długości wejścia, długości osi i stany wywołań są przechowywane jako dane. Liczba taśm i alfabet nie mogą rosnąć wraz z n . Dlatego dowód dotyczący skończonego układu jest tylko jednym składnikiem: trzeba go przenieść na jednolite operacje na strumieniach z jawnie policzonym kosztem obsługi tych deskryptorów.

Co zabrać dalej. Śledź zarówno liczbę wywołań rekurencyjnych, jak i ilość danych przetwarzanych przez każde wywołanie. Oszczędność w jednym może zniknąć, jeśli źle policzysz drugie.

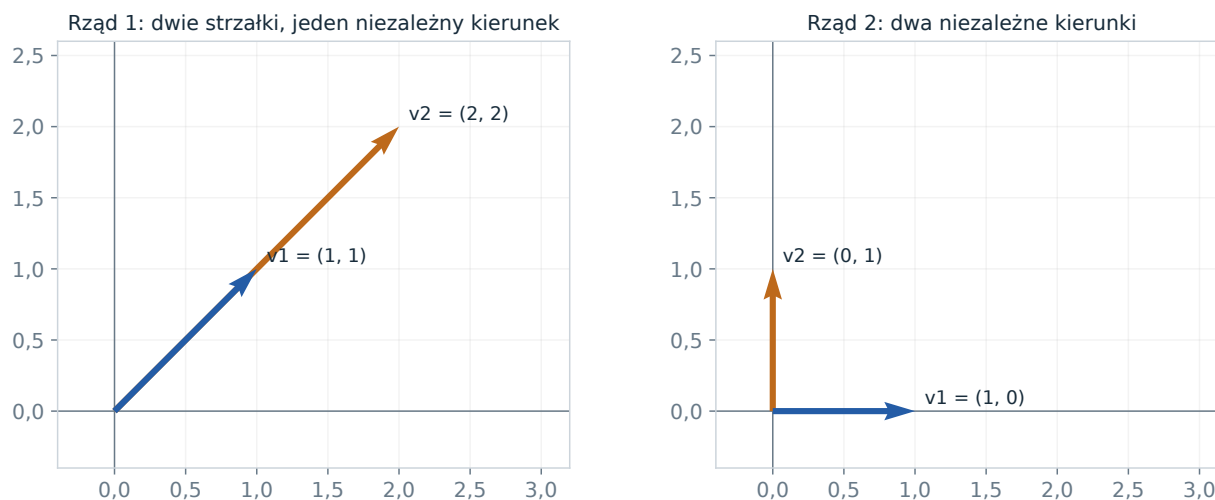
Miejsca w oryginale: `02-streams.tex`, `lem:elementary-streams`.

Wektory, rząd i rama, którą można narysować

Podstawy Geometria stojąca za technicznym słownictwem.

Wektor może być tak prosty jak para liczb, na przykład $(1, 2)$. Narysuj go jako strzałkę od początku układu do tego punktu. Dodawanie wektorów dodaje ich współrzędne. Mnożenie przez liczbę skaluje strzałkę. Podprzestrzeń to zbiór zamknięty na te działania: przykładem jest prosta przechodząca przez początek układu, a innym cała płaszczyzna.

Macierz to reguła łączenia współrzędnych wejścia we współrzędne wyjścia. Jej kolumny mówią, dokąd trafiają podstawowe kierunki wejściowe. Jeśli każde wyjście leży na jednej prostej, istnieje tylko jeden niezależny kierunek wyjścia, nawet gdy macierz ma wiele niezerowych elementów. Ta liczba niezależnych kierunków to jej rząd.



Rys. 8. Licz niezależne kierunki, a nie strzałki. Po lewej $(2, 2)$ to dwukrotność $(1, 1)$: obie strzałki leżą na jednej prostej. Po prawej żadna wielokrotność $(1, 0)$ nie daje $(0, 1)$: potrzebne są oba kierunki. To odpowiednio rząd jeden i rząd dwa.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

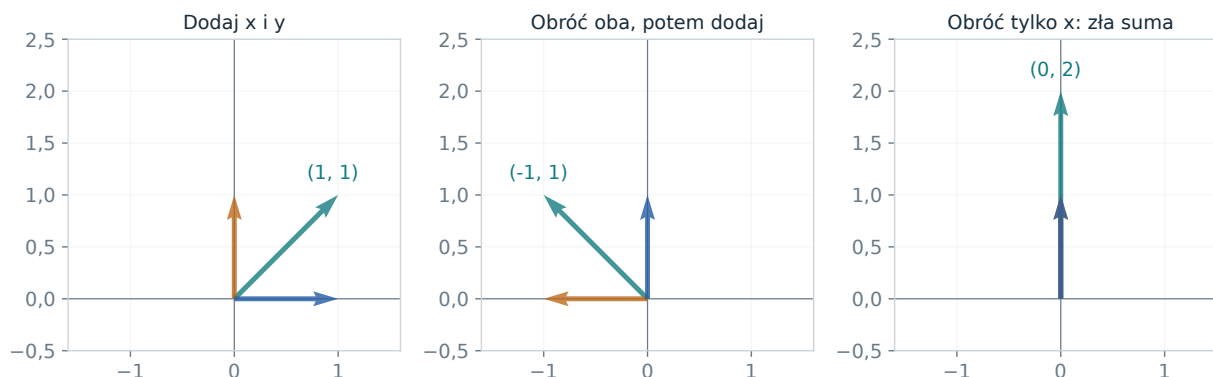
Macierze

$$\begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} \quad \text{i} \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

mają rzędy jeden i dwa. Pierwsza ma cztery niezerowe elementy, ale niższy rząd. Dla pierwszej macierzy wejście (s, t) daje $(s + 2t, s + 2t)$, zawsze na prostej przekątnej. Druga może dać każdy

punkt płaszczyzny. Dlatego liczenie elementów macierzy przeoczyłoby oszczędność, której szuka praca.

Rama (ang. *frame*) to spójna reguła reprezentacji. Niech D obraca każdy wektor o ćwierć obrotu przeciwnie do ruchu wskazówek zegara. Wartość logiczna $(1, 0)$ jest przechowywana jako $(0, 1)$. Wartość logiczna $(0, 1)$ jest przechowywana jako $(-1, 0)$. Suma przechowywanych wartości to $(-1, 1)$, czyli dokładnie obrót sumy logicznej $(1, 1)$. Oba wejścia używały tej samej reguły, więc dodawanie zachowało sens.



Rys. 9. Mieszanie reprezentacji zmienia wynik. Niebieski i złoty to dwa wejścia, turkusowy to ich suma. Obrócenie obu wejść daje obróconą poprawną sumę. Obrócenie tylko pierwszego daje $(0, 2)$, co nie jest zamierzonym obróconym wynikiem $(-1, 1)$. Nakładające się strzałki w prawym panelu celowo pokazują dwa równe kierunki.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

W pracy rama działa na całej tablicy indeksowanej adresami, a nie tylko na wektorze o dwóch liczbach. Obowiązuje to samo prawo liniowości. Późniejszy rysunek z kafelkami 2×2 pokazuje konkretną wersję tablicową: obróć siatkę adresów każdego wejścia, a potem dodaj odpowiadające sobie komórki. Otrzymasz obróconą sumę.

Dlaczego zmiana ramy może się opłacać. Jedna reprezentacja może sprawić, że tania będzie jedna bramka, a inna, że tania będzie następna. Algorytm może się między nimi przełączać, pod warunkiem że płaci za przełączenie i konsekwentnie zachowuje interpretację. Ramy na końcach wyznaczają zamierzoną operację końcową; ramy pośrednie dobiera się tak, żeby suma kosztów przejść była mała. Prawdziwy dowód używa bardziej wyrafinowanych operatorów niż obroty i jawnie sprawdza ich rzędy resztowe.

Co zabrać dalej. Rząd mierzy liczbę niezależnych kierunków. Rama to odwracalna reprezentacja. Wspólne ramy zachowują bramki liniowe; niepolicone zmiany ram ukrywałyby prawdziwą pracę.

Pożyczona pamięć robocza i zmiana reprezentacji

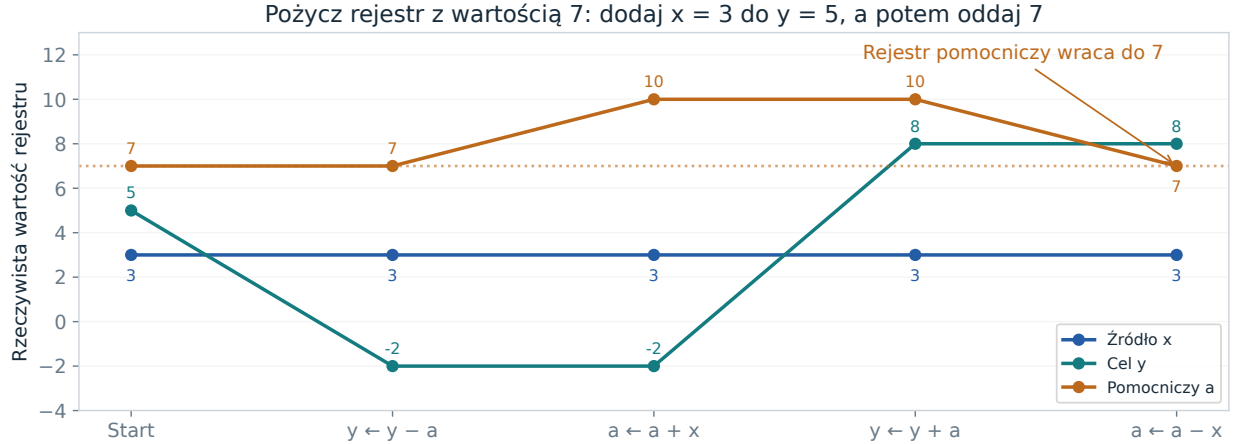
Technika Do przeczytania przed dowodem dotyczącym skończonej sieci.

Współpracują tu dwie idee, które na początku wydają się niezwiązane. Po pierwsze, obliczenie liniowe może pożyczyć rejestr, nie znając jego zawartości, pod warunkiem że usunie wpływ tej zawartości i ją przywróci. Po drugie, tablice mogą być reprezentowane w różnych układach współrzędnych w różnych miejscach układu. Koszt zmiany reprezentacji może być mniejszy niż koszt bezpośredniego wykonania pożądanej operacji na całej tablicy.

Przykład odtworzenia w czterech krokach. Niech a będzie dowolną wartością pomocniczą. Zaczynając od (x, y, a) , wykonaj

$$y \leftarrow y - a, \quad a \leftarrow a + x, \quad y \leftarrow y + a, \quad a \leftarrow a - x.$$

Stan końcowy to $(x, y+x, a)$. Początkowe a znosi się w rejestrze docelowym, a rejestr pomocniczy wraca dokładnie do wartości początkowej. To pogładowa tożsamość w pierścieniu z dodawaniem, a nie proponowana implementacja pełnej sieci. W charakterystyce dwa zarówno dodawanie, jak i odejmowanie stają się operacją XOR.



Rys. 10. Nieznane nie znaczy bezużyteczne. Na wykresie rejestry startują od $x = 3$, $y = 5$, $a = 7$. Rejestr docelowy kończy na ośmiu, a źródło i rejestr pomocniczy wracają do trzech i siedmiu. Tożsamość symboliczna powyżej wyjaśnia, dlaczego działa to dla dowolnych wartości początkowych, a nie tylko dla tego przebiegu.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Jak prawdziwa sieć wyodrębnia właściwe źródło. Oryginalna konstrukcja indeksuje wartości trójkami $S, T \subseteq [h]$. Zbieranie przez współrzędne wspólne dla trójek daje współczynnik zależny od $|S \cap T|$. Układ boczny znosi niechciane wyrazy krzyżowe. W wersji bitowej współczynnik centralny to $|S \cap T|$ modulo dwa: przecięcia rozmiaru zero i dwa już znikają, przecięcie rozmiaru jeden zostaje zniesione, a samoprzecięcie rozmiaru trzy zostawia jedynkę. Wypadkowa aktualizacja celu to więc $y_S \leftarrow y_S + x_S$. W wersji zespolonej współczynnik centralny to $(|S \cap T| - 1)/2$; wyrazy boczne znoszą różne od siebie przypadki przecięcia zero i dwa. W obu przypadkach harmonogram odtwarzania rejestrów pomocniczych usuwa ich stare wartości.

Trzy aktualizacje bloków dają

$$(x, y) \rightarrow (x, y + x) \rightarrow (-y, y + x) \rightarrow (-y, x).$$

To zamiana ze zmianą znaku; na bitach minus znika. Sama sieć nie wykonała jeszcze szybkiej transformaty. Dostarcza algebraicznego rusztowania.

Co znaczy rama. Niech g będzie tablicą logiczną w jednej roli. Rama to odwracalny operator liniowy D działający na elementach tej tablicy indeksowanych adresami. Przechowywana tablica to Dg . Bramka punktowa natomiast łączy różne role pod tym samym adresem. Dla bramki dodającej dwie role wspólna rama daje

$$Dg_1 + Dg_2 = D(g_1 + g_2).$$

To zwykła liniowość, ale wymaga *tej samej* ramy D przy każdym wystąpieniu tej bramki. Przy niezwiązanych ramach D_1, D_2 pożądana tożsamość nie zachodzi.

Wspólną ramą może być obrót siatki adresów o ćwierć obrotu

Tablica A	Tablica B	A + B
1	5	6
2	6	8
3	7	10
4	8	12
D(A)	D(B)	D(A + B) = D(A) + D(B)
2	6	8
4	8	12
1	5	6
3	7	10

Rys. 11. Diagram jest przemieniczny, bo rama jest wspólna. Każdy panel to prawdziwa tablica 2×2 . Dolny rząd obraca siatkę adresów górnego. Dodanie dwóch lewych dolnych tablic daje prawą dolną tablicę, czyli dokładnie obrót sumy z górnego rzędu.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Między wierzchołkami z ramami D_u, D_v zmiana to $D_v D_u^{-1}$. Prześledzenie tego niezmiennika przez cały układ daje zrealizowany operator

$$D_{\text{out}} S_{\Omega} D_{\text{in}}^{-1},$$

gdzie S_{Ω} to układ skalarny zastosowany punktowo. Jeśli układ skalarny jest permutacją ról ze znakami, każda ścieżka widzi tylko swoje ramy końcowe i swój znak skalarny. Ramy pośrednie znoszą się algebraicznie, ale ich fizyczne zmiany i tak trzeba opłacić. Tani dobór tych ram to właściwy problem projektowy.

Dlaczego geometria ma znaczenie. Praca przypisuje wierzchołkom podprzestrzenie. Porównywalne sąsiednie podprzestrzenie dają kierunki resztowe, których wymiary liczą mniejsze operacje. Niski rząd oznacza mniej niezależnych kierunków do zrealizowania. Macierz z wieloma niezerowymi elementami może mieć rząd jeden; na przykład $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ przenosi tylko jedną niezależną kombinację. Z kolei rzadka macierz jednostkowa ma pełny rząd. Rachunek zależy więc od rzeczywistych operatorów resztowych, ich baz i wkładów końcowych, a nie od liczby widocznych bramek.

Układ bitowy łączy skalary w $\mathbb{F}_2$, używając wymiernych etykiet podprzestrzeni dla operatorów adresowych. Układ zespolony łączy diadyczne skalary Gaussa, używając dwójkowych etykiet pod-

przestrzeni dla operatorów fazowych. To dwa różne zastosowania ciała; mylenie wartości skalarnych z etykietami adresów zaciemnia konstrukcję.

Co zabrać dalej. Odtwarzanie sprawia, że istniejące wiersze mogą służyć za pamięć roboczą. Wspólne ramy zamieniają skalarną zamianę w operację na tablicy. Rzędy resztowe wyceniają zmiany między tymi ramami. Potrzebne są wszystkie trzy własności.

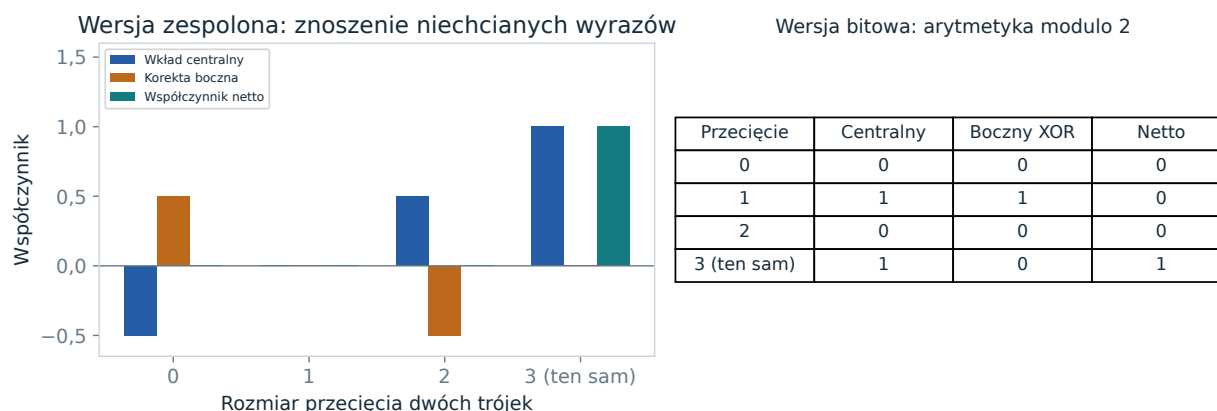
Miejsca w oryginale: 03-motifs.tex, lem:scalar-motifs; eq:common-frame-identity.

Jak znoszą się niechciane źródła

Podstawy Czytaj współczynniki, a nie tylko strzałki układu.

Układ źródłowy tymczasowo łączy wiele wartości. Wyzwanie polega na tym, żeby każdemu celowi zostawić dokładnie to źródło, które jest dla niego przeznaczone. Oryginalna konstrukcja oznacza źródło i cel zbiorami trzyelementowymi. Ich część wspólna może mieć rozmiar zero, jeden, dwa albo trzy. Dwa zbiory trzyelementowe mają trzy wspólne elementy tylko wtedy, gdy są tym samym zbiorem.

Na przykład $\{1, 2, 3\}$ i $\{3, 4, 5\}$ mają jeden wspólny element, a $\{1, 2, 3\}$ i $\{2, 3, 4\}$ – dwa. Centralna część konstrukcji zespolonej daje współczynnik $(q - 1)/2$, gdy przecięcie ma rozmiar q . Dla czterech możliwości to kolejno $-1/2, 0, 1/2, 1$. Wkłady boczne znoszą dwa niechciane przypadki niezerowe. Wykres pokazuje oba wkłady i ich sumę.



Rys. 12. Przetrwa tylko wkład własny. Współczynniki zespolone to zwykłe liczby ze znakiem. Konstrukcja bitowa używa arytmetyki modulo dwa, w której $1 + 1 = 0$; jej wkład boczny znosi przypadek przecięcia rozmiaru jeden. Obie dają współczynniki netto 0, 0, 0, 1.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Dla celu oznaczonego S każde inne źródło T ma więc zerowy współczynnik netto, a źródło S ma współczynnik jeden. Cel dostaje dokładnie swoją wartość, mimo że wewnątrz użyto wielu innych wartości. To mocniejsze niż sprawdzenie jednego wygodnego wejścia: opisuje wkład każdego źródła do wyniku.

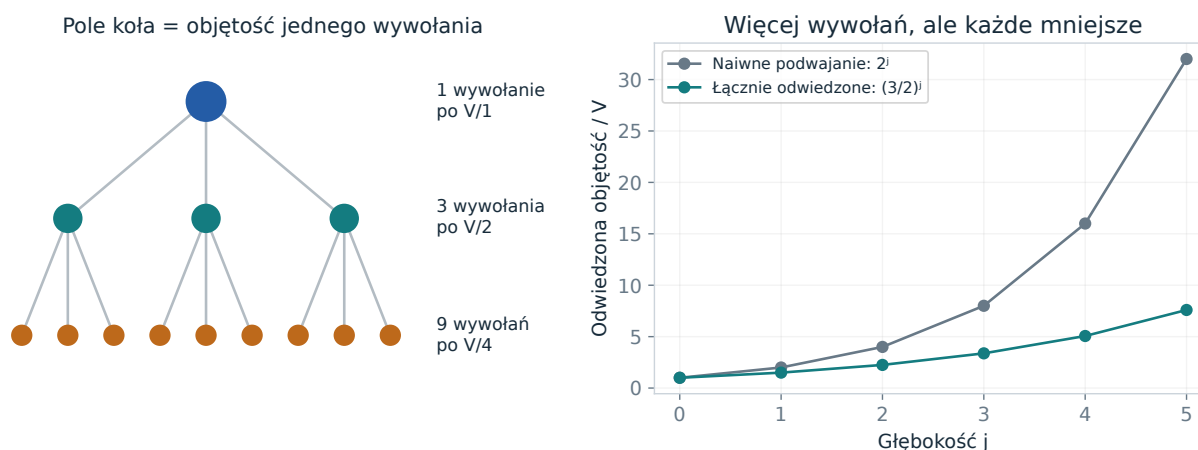
Gdzie wchodzi odtwarzanie pamięci roboczej. Tabela znoszenia dotyczy nowo wprowadzanych wartości źródłowych. Sama nie usuwa dowolnych wartości, które już były w rejestrach pomocniczych. Te wartości początkowe znosi i zwraca osobny harmonogram odtwarzania. Dowód w oryginale potrzebuje obu własności: poprawnego nowego wkładu i niezmiennych pożyczonych danych. Wykres przebiegu rejestrów (rys. 10) pokazuje tę drugą własność na jednym małym przykładzie.

Co zabrać dalej. Zatłoczona sieć może realizować prostą funkcję, bo niechciane współczynniki się znoszą. Dowód musi uwzględnić każdy wkład, także wartości, które początkowo zajmują pamięć roboczą.

Dlaczego więcej wywołań może oznaczać mniej pracy

Podstawy Licz rozmiar każdego zadania, a nie tylko liczbę zadań.

Przypuśćmy, że krok podwajający szerokość wykonuje trzy mniejsze wywołania, każde na połowie objętości danych. Trzy wywołania brzmią gorzej niż dwa, ale nie oznaczają trzech pełnowymiarowych zadań. Łączna odwiedzona objętość to $3 \cdot (V/2) = 1,5V$. Na następnym poziomie jest dziewięć wywołań po $V/4$, łącznie $2,25V$. Pola kół w drzewie odpowiadają tym malejącym objętościom.



Rys. 13. Mnożnik wzrostu to liczba wywołań razy względna objętość. To poglądowe drzewo używa $m = 2$, $W = 2$, $s = 3$. Pierwsze trzy poziomy odwiedzają V , $1,5V$ i $2,25V$. Prawy wykres porównuje to z podwajaniem na każdym poziomie. Wywołania mogą wykonywać się po kolei i ponownie używać pamięci; łączna odwiedzona objętość opisuje pracę, a nie szczytowe zużycie pamięci.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Po j poziomach jest 3^j wywołań, każde na objętości $V/2^j$. Iloczyn daje $V(3/2)^j$. Tymczasem czynnik szerokości to 2^j . Żeby wyrazić pracę jako potęgę szerokości, rozwiąż $2^q = 3/2$. Daje to $q = \log_2(3/2)$, około 0,585, czyli mniej niż jeden. Odpowiadający temu poglądowy koszt rośnie wolniej niż sama szerokość.

Co musi dodać prawdziwy dowód. Drzewo liczy tylko pokazane wkłady rekurencyjne. Prawdziwy algorytm w swoich węzłach także przegląda, kopiuje, układa i odtwarza rekordy. Musi ograniczyć ten narzut, obsłużyć szerokości, które nie pasują dokładnie do drzewa, i zachować jeden ustalony opis maszyny. Źródło rezerwuje w nierówności rekurencyjnej ostry zapas na te zadania. Jego liczbowa oszczędność jest znacznie mniejsza niż narysowana tutaj.

Pytanie kontrolne. A gdyby każde z trojga dzieci rzeczywiście przetwarzało całą objętość V ? Mnożnik wyniósłby trzy, a nie $3/2$, i ten argument nie pokonałby już czynnika dwa. Dlatego dzielenie przez W w pracy jest zobowiązaniem implementacyjnym, a nie kosmetyczną normalizacją.

Co zabrać dalej. Rekurencja liczy pracę ważoną. Wagą jest objętość danych każdego dziecka, łącznie ze skutkami wymaganego układu danych i pamięci roboczej.

Jak skończona oszczędność rzędu staje się wykładnikiem

Technika Do przeczytania przed zamianą adresów.

Niech $C(V, f)$ będzie kosztem rekurencyjnym dla objętości V i parametru szerokości f . Jedną ze schematycznych rekurencji leżących u podstaw argumentu to

$$C(V, mf) \leq sC(V/W, f) + O(V).$$

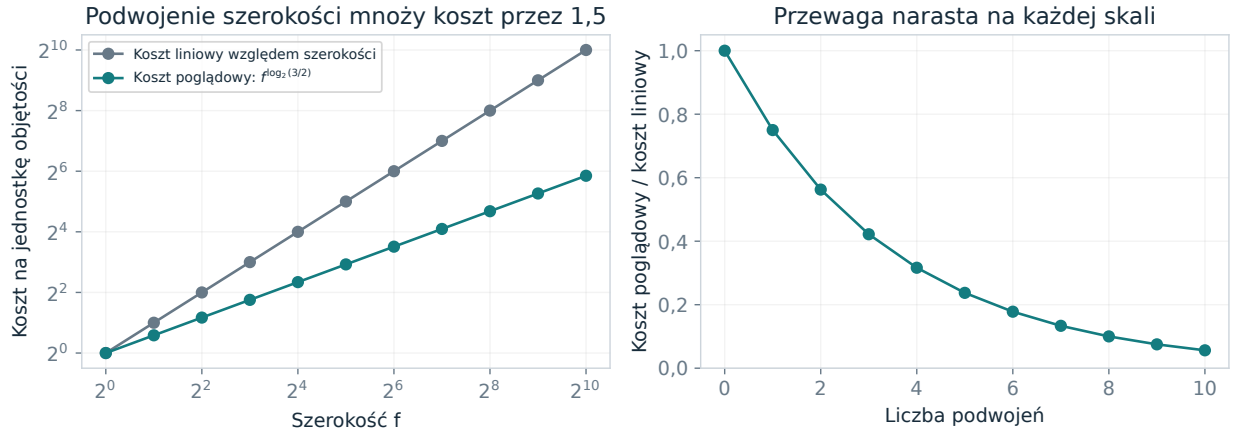
Pełna praca uwzględnia warunki dotyczące układu danych i deskryptorów; ten zapis wyodrębnia samo zasadnicze skalowanie. Rodzic wywołuje s mniejszych operacji, każdą na strumieniu roli o objętości V/W . Podziel przez V i oznacz przez $a(f)$ jednostajne ograniczenie kosztu na jednostkę objętości. Wtedy

$$a(mf) \leq (s/W)a(f) + O(1).$$

Jeśli $a(f)$ zachowuje się jak f^q , jego wkład rekurencyjny skaluje się o m^q . Kluczowe porównanie to więc

$$\frac{s}{W} < m^q < m, \quad q < 1.$$

Końcowy dowód używa nierówności ostrych, żeby można było wchłonąć narzut.



Rys. 14. Małe dzieci mogą przeważać dużą liczbę wywołań. Wykres przedstawia poglądową rekurencję z $m = 2$, $W = 2$, $s = 3$; nie jest to sieć z pracy ani pomiar czasu działania. Jej wykładnik rekurencyjny to $\log_2(3/2) \approx 0,585$.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Dlaczego $s < Wm$ rozstrzyga sprawę. Szerokość rosnąca m -krotnie normalnie kosztowałaby czynnik m . Wkład rekurencyjny rośnie natomiast o s/W . Jeśli $s = Wm$, to porównanie nie daje ostrej oszczędności w potęgę. Jeśli $s < Wm$, wykładnik mniejszy od jednośc może leżeć ostro powyżej $\log_m(s/W)$. Źródło konstruuje dwie takie nierówności: jedną dla zamiany adresów i jedną dla warstw numerycznych. Ich wymierne górne wykładniki nazywa τ i σ .

Małeńki deficyt wciąż daje dodatnią lukę. Zdefiniuj $\eta = (Wm - s)/(Wm)$. Algebra daje

$$\log_m(s/W) = 1 + \frac{\log(1 - \eta)}{\log m}.$$

Dla $0 < \eta < 1$ logarytm w liczniku jest ujemny. Gdy η jest małe, luka wynosi w przybliżeniu $\eta/\log m$. Przybliżenie pomaga intuicji; źródło używa dokładnych wymiernych deficytów i ostrych

nierówności wykładniczych, żeby poświadczyć wymierny wykładnik. Oryginalny wybór to $\tau = \sigma = 1 - 2^{-50}$, a nie wyraźnie duża oszczędność z wykresu pogładowego.

Dłaczego ścinanie łatwiej zbudować niż zamianę. Ścinanie adresów (ang. *address shear*) aktualizuje jeden blok kombinacją liniową innego: $(H, D) \mapsto (H + AD, D)$. Praca rozkłada wymierną macierz A rzędu a na dwa odwracalne czynniki trójkątne i częściową permutację z a elementami podstawowymi (ang. *pivots*). Aktualizacje trójkątne tanio podążają za porządkiem taśmy. Każdy element podstawowy wymaga jednej mniejszej zamiany. Liczba kosztownych wywołań to więc rząd a . Odpowiednia ustalona nieparzysta liczba pierwsza sprawia, że potrzebne wymierne współczynniki i odwrotności mają sens modulo jej potęgi. Przy przenoszeniu tożsamości wymiernej na adresy modularne nie można ignorować mianowników i odwracalności.

Mało efektowne warunki są niezbędne. Prawdziwe szerokości pól nie muszą być potęgami m . Liczby wierszy mogą nie dzielić się przez W . Niektóre układy zawierają między aktywnymi polami pola bierne. Kolejna sekcja źródła usuwa te ograniczenia i płaci za dopełnianie i sprzątanie. Sama pogładowa rekurencja nie dowodzi tych zobowiązań implementacyjnych.

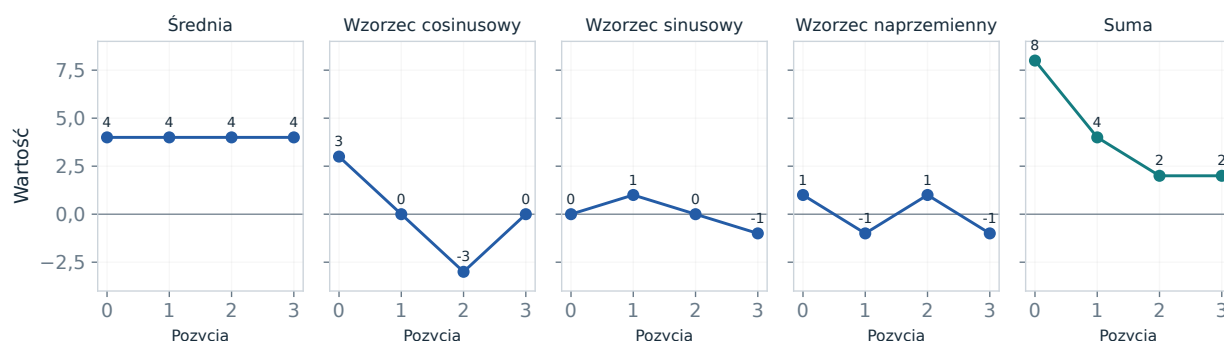
Co zabrać dalej. Skończone świadectwo dostarcza ostrego deficytu. Implementacja taśmowa musi zachować normalizację V/W , żeby ten deficyt zamienił się w oszczędność w wykładniku.

Miejsca w oryginale: *03-motifs.tex*, *eq:explicit-motif-exponents*; *04-swap.tex*.

Co naprawdę zmienia transformata

Podstawy Zastąp listę wartości wagami prostych wzorców.

Transformata to zmiana opisu. Weźmy cztery liczby $[8, 4, 2, 2]$. Zamiast wypisywać te wartości wprost, opisz je jako sumę czterech prostych wzorców. Rysunek pokazuje każdy wkład i wynik. Dodaj pierwszy punkt z każdego z czterech pierwszych wykresów: $4 + 3 + 0 + 1 = 8$. W drugim punkcie: $4 + 0 + 1 - 1 = 4$. Powtórz to dla pozostałych pozycji, a dostaniesz dwa i dwa.

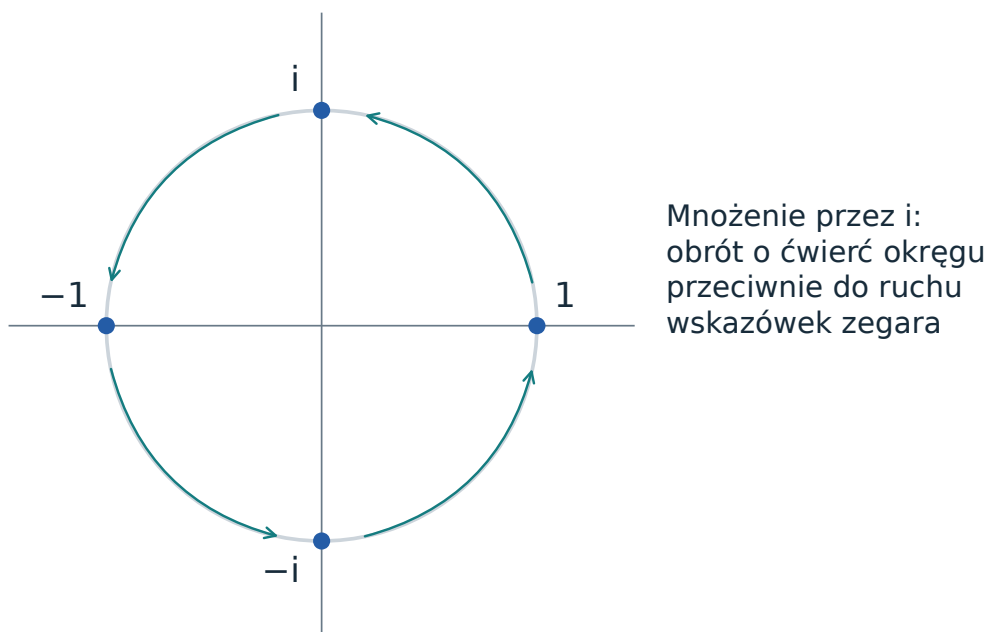


Rys. 15. Te same cztery liczby w opisie przez wzorce. Średnia $[4, 4, 4, 4]$, wkład cosinusowy $[3, 0, -3, 0]$, wkład sinusowy $[0, 1, 0, -1]$ i wkład naprzemienny $[1, -1, 1, -1]$ sumują się dokładnie do $[8, 4, 2, 2]$. Linie łączą cztery dyskretne próbki; to nie jest zmierzony dźwięk.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Dlaczego nazywać je częstotliwościami? Średnia nie oscyluje. Wzorce cosinusowy i sinusowy przechodzą jeden pełny cykl na czterech pozycjach, z różnymi fazami początkowymi. Wzorec naprzemienny zmienia znak w każdym kroku. W tej rzeczywistej bazie wzorców wagi wynoszą $4, 3, 1, 1$. To nie są cztery zespolone współczynniki Fouriera w konwencji pracy; to równoważny, elementarny sposób przedstawienia tych samych danych rzeczywistych.

Gdzie pojawiają się liczby zespolone. Liczba zespolona to para współrzędnych rzeczywistych zapisana jako $a + ib$. Dodawanie działa współrzędna po współrzędnej. Reguła $i^2 = -1$ sprawia, że mnożenie przez i obraca płaszczyznę o ćwierć okręgu: (a, b) przechodzi w $(-b, a)$. To wygodna księgowość dla wzorców oscylujących; nie znaczy, że liczby wejściowe albo ich iloczyn są urojone.



Rys. 16. Faza to położenie na okręgu. Mnożenie przez i przechodzi cyklicznie przez $1, i, -1, -i, 1$. Każdy punkt ma moduł jeden. Cztery obroty to najprostszy widoczny przykład pierwiastków z jedynki. Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Dla czterech próbek zwykle, nieznormalizowane współczynniki Fouriera ciągu $[8, 4, 2, 2]$ to $[16, 6 - 2i, 4, 6 + 2i]$ w konwencji z ujemnym wykładnikiem. Po podzieleniu przez cztery otrzymujemy wersję znormalizowaną $[4; 1, 5 - 0,5i; 1; 1, 5 + 0,5i]$. Średnia to cztery; sprzężone pary zespolone kodują razem wkłady cosinusowy i sinusowy. Nie musisz wyprowadzać tych zespolonych elementów, żeby śledzić resztę przewodnika, ale to rozróżnienie wyjaśnia, dlaczego czynników skali transformaty nie da się zgadnąć.

Dlaczego to pomaga w mnożeniu. Przesunięcie listy zmienia fazę każdej częstotliwości w prosty sposób. Splot składa się z ważonych przesunięć, więc w opisie częstotliwościowym staje się mnożeniem odpowiadających sobie składowych, z uwzględnieniem wybranej normalizacji. Trudna praca polega na sprawnej zmianie opisu i dokładnym jej odwróceniu. Pełna praca dodaje starannie dobrane układy danych i reprezentację wielomianową, żeby poradzić sobie z tymi kosztami.

Co znaczy tu iloczyn tensorowy. Najpierw zastosuj operację na dwóch elementach do każdej poziomej pary w tablicy 2×2 , a potem do każdej pionowej pary. Zastosowanie tej samej operacji osobno w obu kierunkach współrzędnych to jej działanie jako iloczynu tensorowego. Terminologia staje się zwięzła przy wielu osiach, ale znaczenie pozostaje to samo: „wykonaj tę operację w każdym wybranym kierunku, trzymając pozostałe ustalone”.

Co zabrać dalej. Transformata zmienia współrzędne, którymi opisujemy dane. Ustrukturyzowane wzorce mogą sprawić, że w tych współrzędnych operacja staje się tania.

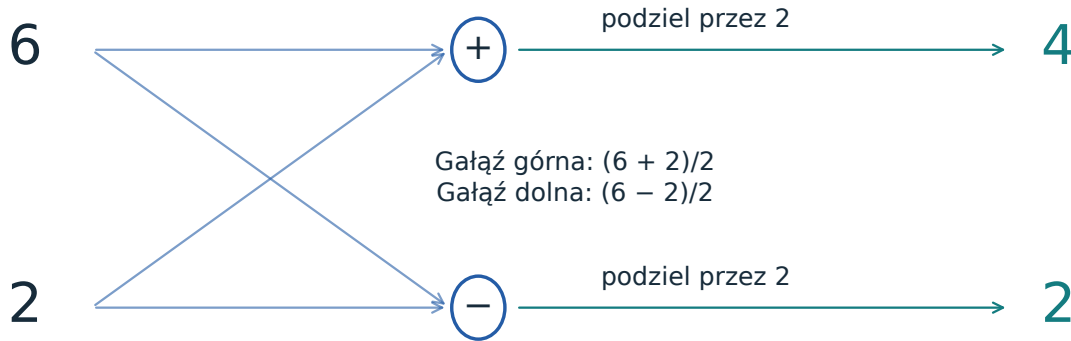
Najmniejszy krok transformaty, zastosowany na wielu osiach

Technika Do przeczytania przed jednoczesnymi warstwami motylków.

Dwupunktowa znormalizowana transformata Fouriera (najmniejszy „motylek” szybkiej transformacji Fouriera) przekształca

$$H_0(u, v) = \left(\frac{u+v}{2}, \frac{u-v}{2} \right).$$

Pierwsze wyjście zapisuje średnią, drugie różnicę. Dla $(u, v) = (6, 2)$ wyjścia to $(4, 2)$. Oryginał odzyskasz jako $u = 4 + 2$, $v = 4 - 2$. Czynniki $1/2$ ma znaczenie: to nie jest nieznormalizowana macierz Hadamarda ani macierz unitarna z czynnikiem $1/\sqrt{2}$.



Rys. 17. Prześledź dwie gałęzie arytmetyczne. Oba wejścia trafiają do obu operacji. Gałąź plus zwraca cztery, a gałąź minus dwa, po podzieleniu przez dwa. Wartości to przykład obliczeniowy, a nie pośrednie wartości z dużej sieci.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

W przykładzie z dwiema osiami zacznij od czterech elementów $[8, 4, 2, 2]$ w porządku adresów 00, 01, 10, 11. Zastosowanie H_0 do sąsiednich par daje $[6, 2, 2, 0]$. Zastosowanie go wzdłuż drugiego bitu, czyli do par pozycji zero i dwa oraz jeden i trzy, daje $[4, 1, 2, 1]$. To działanie $H_0 \otimes H_0$.

Jeśli każde wejście ma moduł zespolony co najwyżej M , to każde wyjście również: $|u \pm v|/2 \leq (|u| + |v|)/2 \leq M$. Ukończona operacja jest więc kontrakcją w normie maksimum. Zastosowanie jej na kilku bitach adresu oznacza łączenie w pary rekordów różniących się wybranym bitem, dla każdego ustawienia pozostałych bitów. Naiwna implementacja wykonuje jedno przejście przez całą tablicę na każdą wybraną oś. Proponowany algorytm warstwowy organizuje te operacje wspólnie.

Dlaczego praca tymczasowo używa innego jądra. Zdefiniuj X jako zamianę dwóch elementów, $S = \text{diag}(1, i)$, $a = (1 + i)/2$, $b = (1 - i)/2$ oraz $C = aI + bX$. Przydatne tożsamości to

$$C^2 = X, \quad C^4 = I, \quad H_0 = bSCS.$$

Jądro C dobrze współpracuje z dwójkowymi ramami fazowymi, którymi wycenia się kierunki resztowe. Po ukończeniu jego warstwy tensorowej fazy diagonalne i znany diadyczny czynnik zespolony dają pożądaną warstwę H_0 . To klasyczne tożsamości liniowe; użycie macierzy fazowych nie wymaga komputera kwantowego.

Jedno zaokrąglenie na końcu, a nie po każdej bramce. Wartości wewnętrzne to dokładne diadyczne liczby Gaussa (ang. *Gaussian dyadics*): liczby, których część rzeczywista i urojona są liczbami całkowitymi podzielonymi przez potęgę dwójki. Wartości pomocnicze mogą rosnąć, nawet jeśli ukończona warstwa H_0 jest kontrakcją. Dowód potrzebuje więc bitów zapasu na wewnętrzne moduły i mianowniki. Wykonuje całą warstwę dokładnie i obcina dopiero na jej granicy. Sama kontrakcyjność nie uzasadniałaby odrzucania bitów po każdej aktualizacji pomocniczej.

Dlaczego wybrane bity tworzą wyjątkowe adresy. Wydajna aktualizacja upakowanych adresów może zgadzać się z pożądaną zmianą bazy dwójkowej na większości adresów, podczas gdy przeniesienia zaburzają mały, jawnie opisany zbiór. Praca wyodrębnia ten zbiór i naprawia go deterministycznie. Dowód potrzebuje zgodności poza nim, niezmienniczości zbioru wyjątkowego oraz ograniczenia kosztu wyjęcia, posortowania i ponownego wstawienia pełnych rekordów. „Rzadki” nie znaczą losowy ani pomijalny: każde wejście musi być obsłużone poprawnie.

Co zabrać dalej. Druga oszczędność grupuje arytmetykę z wielu osi. Musi przetrwać zmiany bazy, wzrost wartości pośrednich i dokładną korektę wyjątkowych rekordów.

Miejsca w oryginale: `05-layers.tex`, `eq:phase-C`; `eq:phase-basic-identities`.

Pierwiastki z jedynki jako przesunięcia ze znakiem

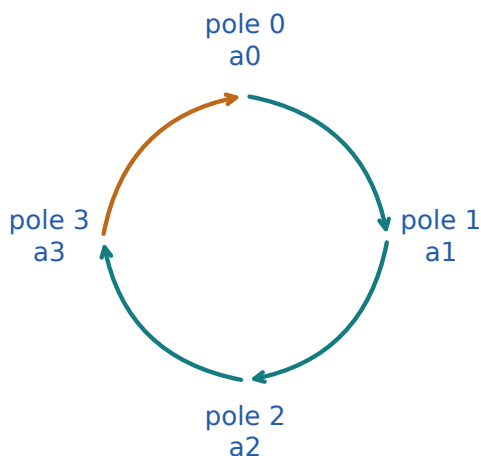
Technika Do przeczytania przed transformatami syntetycznymi i iloczynami w pierścieniu.

Transformata zwykle mnoży wartości przez pierwiastki z jedynki. W reprezentacji syntetycznej sama wartość jest krótkim wielomianem, przechowywanym jako lista współczynników. Pracujemy w

$$\mathcal{R}_r = \mathbb{C}[y]/(y^r + 1).$$

Oznacza to, że y^r zastępujemy przez -1 . Mnożenie przez y przesuwa listę współczynników i zmienia znak współczynnika, który się zawija.

zawinięcie: zmiana znaku a_3



Mnożenie przez y modulo $y^4 + 1$

$$\begin{aligned} [a_0, a_1, a_2, a_3] \\ \downarrow \\ [-a_3, a_0, a_1, a_2] \end{aligned}$$

Rys. 18. Operacja na pierwiastku staje się dokładnym przesunięciem danych. Dla $r = 4$ iloczyn $y(a_0 + a_1y + a_2y^2 + a_3y^3)$ redukuje się do $-a_3 + a_0y + a_1y^2 + a_2y^3$. Ta operacja nie wymaga żadnego przybliżonego mnożenia trygonometrycznego.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Dla potęgi dwójki t dzielącej $2r$ element $\omega_t = y^{2r/t}$ służy za pierwiastek transformaty. Jego potęgi działają jak permutacje współczynników ze znakami. Źródło dowodzi potrzebnego znoszenia się charakterów bezpośrednio. Ma to znaczenie, bo pierścień ilorazowy ma dzielniki zera: dzielenie przez element, który wygląda na niezerowy, nie jest automatycznie dozwolone.

Zachowaj kolejność, którą algorytm naturalnie produkuje. Rekurencyjna transformata Fouriera często zwraca częstotliwości w kolejności z odwróconymi bitami. Dla długości cztery ta kolejność to 0, 2, 1, 3. Jeśli obie przetransformowane dane wejściowe mają tę samą kolejność, odpowiadające sobie elementy nadal można mnożyć. Sortowanie każdej transformaty do ładniejszej kolejności dodałoby pracy, nie zmieniając matematycznego iloczynu. Transformata przeciwna jest napisana tak, by przyjmować kolejność, która rzeczywiście występuje.

Normalizacja jest częścią interfejsu. Obie transformaty w tym manuskrypcie zawierają czynnik $1/t$. W konsekwencji $F_t^+ F_t^- = I/t$, a nie I . Przy tej konwencji dla splotu cyklicznego długości t

$$F_t^+((F_t^- u) \odot (F_t^- v)) = \frac{u * v}{t^2}.$$

Jeden czynnik t pochodzi z normalizacji iloczynu w przód, a drugi ze znormalizowanej transformaty przeciwnej. Utrata któregoś zniszczyłaby dokładne odtworzenie. Wersje wielowymiarowe zastępują t iloczynem długości osi, z uwzględnieniem permutacji zachowanych w źródle.

Dlaczego mnożenie wielomianów nie czyni dowodu błędnym kołem. Elementy punktowe są teraz wielomianami, więc ich iloczyny wciąż trzeba obliczyć. Źródło pakuje ograniczone współczynniki ze znakiem w większe liczby całkowite z odstępem wystarczającym do odzyskania każdego współczynnika. Jawnie obsługuje wyodrębnianie wycelowane, znaki i zawijanie ujemno-cykliczne (ang. *negacyclic*). Do mnożenia tych krótszych liczb wywołuje wcześniej ustalony mnożnik $O(N \log N)$, a nie ulepszony algorytm, który próbuje uzasadnić. Długość wejścia N w tych wywołaniach różni się od pierwotnego n , a wszystkie wywołania są ujęte w końcowej tabeli kosztów.

Co zabrać dalej. Zmiana reprezentacji może zastąpić kosztowne skalarne mnożenia przez pierwiastki dokładnymi przesunięciami ze znakiem, ale układ danych i skalowanie wciąż kosztują.

Miejsca w oryginale: *06-transforms.tex*, *lem:synthetic-characters*; *lem:signed-ring-product*.

Zmiana siatki bez sortowania wszystkiego

Technika Do przeczytania przed przepróbkowaniem gaussowskim.

Naturalny układ splotu i wydajny układ transformaty syntetycznej nie mają tych samych długości osi. Redukcja mnożenia używa osi źródłowych o długościach parami względnie pierwszych; transformaty syntetyczne wolą potęgi dwójki. Przepróbkowanie gaussowskie (ang. *Gaussian resampling*) łączy je starannie kontrolowanymi odwzorowaniami liniowymi. To analityczny most, a nie zwykle zaokrąglanie indeksu do najbliższego sąsiada.

W jednym wymiarze źródło podaje tożsamość postaci

$$P_s F_s = 2^{2\alpha^2} B_0 P_t F_t A.$$

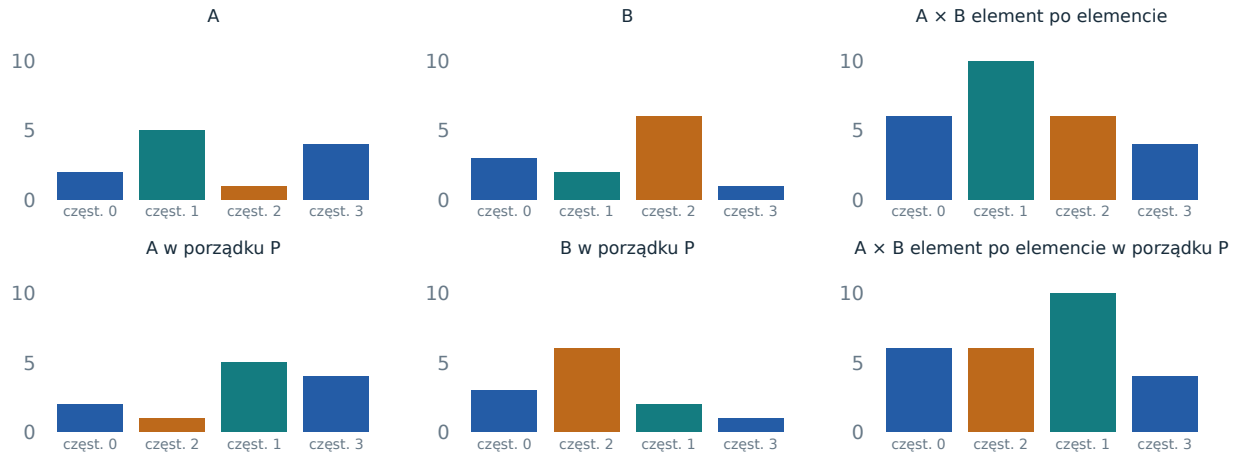
Tutaj A odwzorowuje siatkę źródłową długości s na dłuższą siatkę długości t , a B_0 odwzorowuje z powrotem. Operatory P to permutacje współrzędnych. Odwzorowania są realizowane w przybliżeniu z podaną precyzją, z jawnymi ograniczeniami błędów i czasu. Duży znany skalar jest uwzględniany przy odtwarzaniu. Nierówności wiążące s, t, α, p to założenia tego interfejsu, a nie ozdoba, którą można pominąć przy zmianie parametrów.

Dlaczego pozostawienie permutacji we wzorze oszczędza czas. Niech P przedstawia współrzędne. Jeśli dwie tablice używają tego samego przestawienia, to

$$(PA) \odot (PB) = P(A \odot B).$$

Odpowiadające sobie częstotliwości wciąż są poprawnie sparowane. Odbiorca, który zna tę kolejność, może ich użyć bezpośrednio. Ta prosta tożsamość wyjaśnia, skąd bierze się ta możliwość; źródło dowodzi bardziej szczegółowych tożsamości potrzebnych, by wszystkie jego permutacje źródłowe i docelowe były zgodne z transformatą przeciwną.

Przestaw oba wejścia tak samo, a każda częstotliwość nadal spotka swoją parę



Rys. 19. Spójność może usunąć zbędne przestawianie. Słupki pokazują tę samą permutację zastosowaną do obu konkretnych czynników i do ich iloczynu po elementach [6, 10, 6, 4]. Sam diagram nie dowodzi, że dowolna implementacja transformaty odwrotnej przyjmie tę kolejność. Musi to zapewnić jej interfejs. Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Gdzie pasuje tożsamość chirp. Tożsamość Bluesteina (algorytm Bluesteina, ang. *chirp-z*) przepisuje dwuliniowy wykładnik za pomocą $2jk = j^2 + k^2 - (j - k)^2$. Kwadratowe czynniki fazowe na wejściach i wyjściach zamieniają transformatę w spłot ze znanym sygnałem chirp. Oryginalny dowód trzyma odpowiednią permutację wewnątrz tego rachunku. Kolejne skręcenie fazy godzi spłot cykliczny z ujemnocykliczną relacją wielomianową $y^r = -1$. Każde skręcenie i jego usunięcie ma koszt zarówno algebraiczny, jak i precyzyjny.

Budżet błędów ma dwa różne rodzaje wzrostu. W złożeniu dokładnych kontrakcji istniejący błąd nie jest wzmacniany, więc lokalne błędy przybliżeń można dodawać. Cofnięcie znanej dużej normalizacji mnoży jednak błąd bezwzględny. Błędem byłoby więc wnioskować: „wszystkie transformaty są kontrakcjami, więc wynik końcowy jest dokładny”, nie śledząc przeskalowania. Źródło wybiera precyzję roboczą wystarczającą, by przetrwać zarówno skumulowany błąd lokalny, jak i końcowe wzmocnienie.

Co zabrać dalej. Układ danych jest częścią kontraktu algorytmu dotyczącego wejścia i wyjścia. Zachowanie użytecznej kolejności ma wartość tylko wtedy, gdy każdy kolejny krok ją akceptuje.

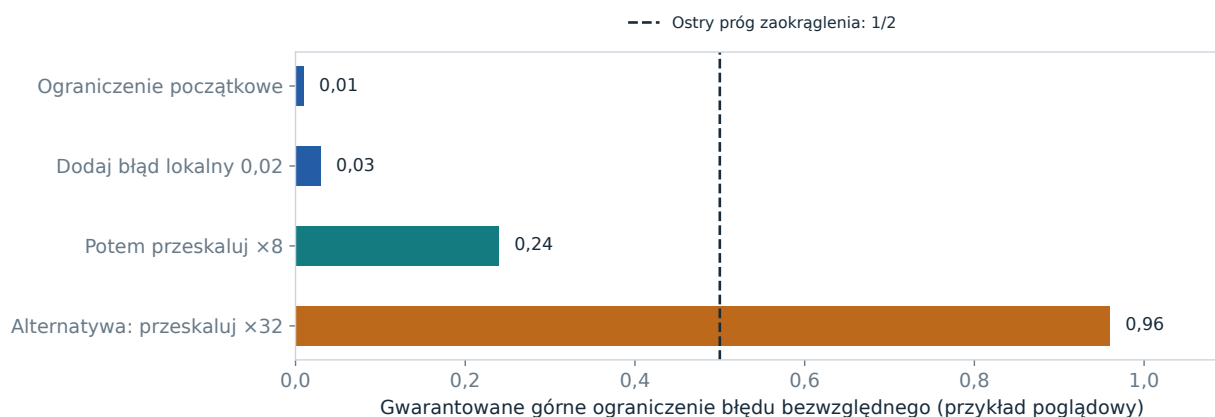
Miejsca w oryginale: *07-resampling.tex*, *lem:no-sort-resampling*; *08-assembly.tex*.

Mały błąd musi pozostać mały po przeskalowaniu

Podstawy Budżet liczbowy, który można sprawdzić ręcznie.

Wyobraź sobie przybliżone obliczenie, którego bieżący błąd bezwzględny wynosi co najwyżej 0,01. Przepuszczenie go przez dokładną kontrakcję nie powiększa tego błędu. Jeśli następne przybliżenie dodaje co najwyżej 0,02, nowe ograniczenie błędu to co najwyżej 0,03. To suma w najgorszym przypadku: nie zakładamy, że błędy się znoszą albo zachowują losowo.

Teraz przywróć czynnik skali osiem. Ograniczenie błędu staje się równe $8 \cdot 0,03 = 0,24$, czyli bezpiecznie poniżej jednej drugiej. Gdyby czynnik skali wynosił 32, ograniczenie wyniosłoby 0,96. Ta słabsza gwarancja nie uzasadnia zaokrąglenia do właściwej liczby całkowitej.



Rys. 20. Mały błąd wewnętrzny to nie to samo co błąd końcowy. Dwa ostatnie słupki to alternatywne przeskalowania tego samego ograniczenia 0,03, a nie kolejne operacje. Przekroczenie 1/2 oznacza, że to ograniczenie nie wystarcza do zagwarantowania dokładnego zaokrąglenia; nie dowodzi, że rzeczywisty wynik jest błędny.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Dowód w źródle ma większe i bardziej złożone normalizacje, ale zasada jest ta sama. Budżetuje błędy lokalne, ogranicza rozmiary wartości pośrednich i śledzi mnożenie błędu przy cofaniu skal. Potrzebuje też dość bitów na pośrednie dokładne wartości diadyczne, nawet gdy ukończona transformata jest kontrakcją.

Dlaczego nierówność musi być ostra. Jeśli prawdziwa liczba całkowita to 12, wynik leżący ostro między 11,5 a 12,5 zaokrągla się jednoznacznie do 12. Dla dokładnie 12,5 odległość do 12 i do 13 jest taka sama, więc dopuszczalny błąd „co najwyżej jedna druga” nie rozstrzyga, o którą liczbę chodziło. Praca dowodzi ostrego ograniczenia końcowego dla każdego współczynnika, a potem wykonuje dokładne przeniesienia.

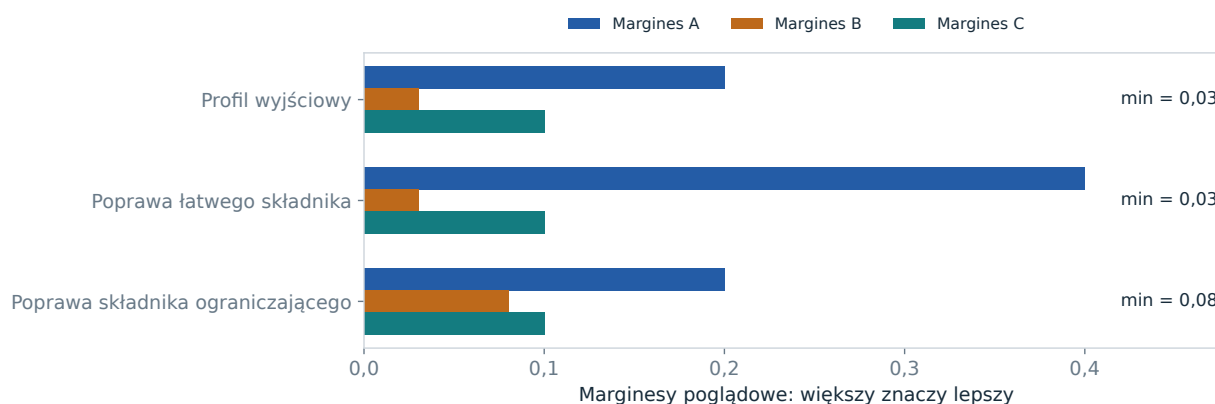
Co zabrać dalej. Przybliżenia dają się pogodzić z dokładnym wynikiem, jeśli udowodnimy, że błąd po pełnym przeskalowaniu jest mniejszy niż połowa odstępów między liczbami całkowitymi.

Dlaczego poprawienie jednej części może niczego nie zmienić

Podstawy Końcową oszczędność ogranicza najsłabszy margines.

Przypuśćmy, że trzy operacje mają znormalizowane koszty $p^{0,80}$, $p^{0,97}$ i $p^{0,90}$. Ich marginesy poniżej pełnej potęgi p wynoszą 0,20, 0,03 i 0,10. Dla dużych p najszybciej rośnie wyraz $p^{0,97}$ i to on decyduje o wykładniku sumy.

Spraw, by pierwsza operacja była znacznie tańsza, zmieniając $p^{0,80}$ na $p^{0,60}$. Jej margines rośnie do 0,40, ale sumą wciąż rządzi $p^{0,97}$. Zamiast tego popraw drugą operację do $p^{0,92}$. Teraz najmniejszy margines wynosi 0,08, więc łączny wykładnik też może się poprawić, z zastrzeżeniem pozostałego narzutu i wymagań ostrego zapasu.



Rys. 21. Lokalny zysk nie musi poprawić ogólnego wykładnika. Każdy wiersz to osobny profil poglądowy, a nie wynik historyczny. Większe marginesy są lepsze. Minimum pozostaje równe 0,03, gdy poprawia się łatwy wyraz, i rośnie do 0,08, gdy poprawia się wyraz ograniczający.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Końcowy rachunek w źródle ma siedem dominujących wyrazów zamiast trzech. Rysunek w kolejnym objaśnieniu technicznym pokazuje ich rzeczywiste oryginalne marginesy. Skala logarymiczna sprawia, że skrajnie małe wartości dodatnie stają się widoczne. Ten wykres odwraca konwencję wizualną: dalej w prawo oznacza mniejszy margines, bo oś to minus logarytm marginesu. Zawsze najpierw przeczytaj oś, a dopiero potem oceniaj, który słupek jest lepszy.

Dlaczego potrzebny jest odrobinę większy zapas. Koszt może też zawierać czynniki takie jak $(\log p)^2$. Taki czynnik rośnie wolniej niż dowolna ustalona dodatnia potęga p^p , ale nie jest stałą. Wybranie końcowej oszczędności ostro poniżej dostępnego marginesu zostawia miejsce na wchłonięcie tych czynników dla dostatecznie dużych p . Równość na granicy nie uzasadniałaby tego samego argumentu.

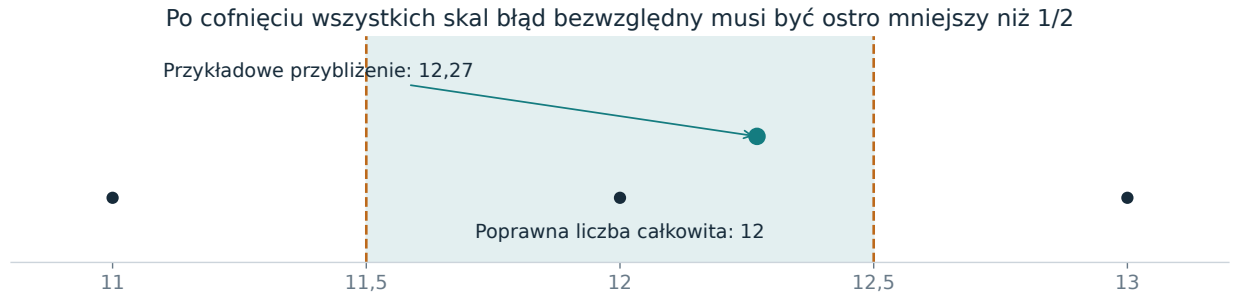
Co zabrać dalej. Śledź cały profil kosztów. Ulepszenie liczy się dla głównego wyniku tylko wtedy, gdy pozwalają na to wszystkie wyrazy ograniczające i interfejsy.

Dokładność i czas działania razem

Technika Do przeczytania przed końcowym dowodem składającym całość.

Praca ma dwie linie mety. Musi zwrócić dokładny iloczyn, a każda część obliczenia musi zmieścić się w deklarowanym ograniczeniu czasu. Te zobowiązania łączą się przez precyzję i objętość danych, ale żadne nie zastępuje drugiego.

Dokładność z ostrego przedziału błędu. Pożądane współczynniki spłotu c_j są liczbami całkowitymi. Jeśli obliczenie daje przybliżenie $\tilde{c}_j$ spełniające $|\tilde{c}_j - c_j| < 1/2$, zaokrąglenie do najbliższej liczby całkowitej odtwarza c_j jednoznacznie. Ograniczenie musi obowiązywać *po* przywróceniu wszystkich normalizacji transformat i skal cyfr. Remis dokładnie przy $1/2$ nie wystarcza. Po odtworzeniu współczynników przeniesienia i zapis dwójkowy to dokładne operacje na liczbach całkowitych.



Rys. 22. Przybliżona arytmetyka wewnątrz, dokładny wynik na wyjściu. Wartość 12,27 to przykład dydaktyczny. Dowód w źródle daje jednolite ograniczenie dla każdego odtwarzanego współczynnika; ten rysunek nie mierzy rzeczywistego błędu numerycznego.

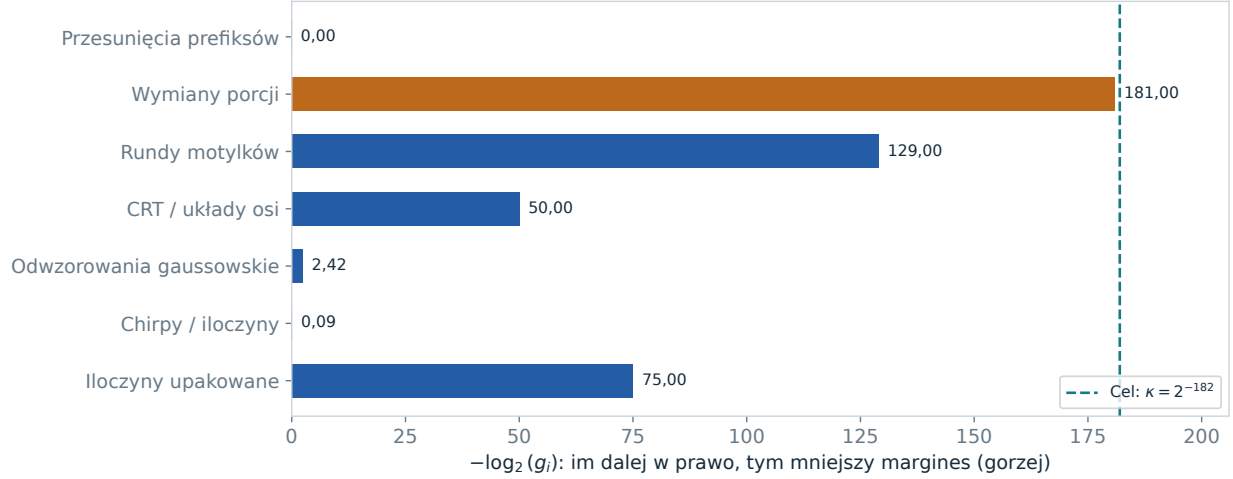
Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

W oryginalnym składaniu źródło używa szerokości cyfry $b = \lceil \log_2 n \rceil$ i precyzji roboczej $p = 6b$. W przybliżeniu jest n/b cyfr i $\Theta(p)$ przechowywanych bitów na współczynnik, co daje dopełnioną objętość $V = Tp = \Theta(n)$. Ta zależność zamienia koszty na przechowywany bit w ograniczenie wyrażone przez pierwotny rozmiar wejścia.

Siedem konkurujących rachunków. Końcowa tabela rozdziela operacje, które różnie skalują się z p . Zapisz znormalizowany wyraz jako p^{1-g_i} ; wtedy g_i to jego margines poniżej pełnej potęgi p . Przy $d = \Theta(p^\epsilon)$, $K = \Theta(p^{\epsilon c})$ i $\ell = \Theta(p^{1-\epsilon})$ marginesy wynoszą

$$\begin{aligned} g_1 &= 1 - \epsilon(1 + c), & g_2 &= \epsilon c(1 - \tau), \\ g_3 &= \epsilon(1 - \lambda'), & g_4 &= 1 - \tau - \epsilon(2 - \tau), \\ g_5 &= \frac{1}{4} - \delta - \frac{3}{2}\epsilon, & g_6 &= 1 - \delta - \epsilon, \\ g_7 &= \epsilon. \end{aligned}$$

Sumą rzadzi *najmniejszy* margines. Powiększenie marginesu, który już jest duży, może nie mieć żadnego wpływu na główny wykładnik.



Rys. 23. Oryginalny wybór parametrów ma wyraźne wąskie gardło. Te wartości wyliczono z dokładnych wymiennych parametrów oryginału, a nie oszacowano z pomiarów czasu. Oś pozioma to $-\log_2 g_i$: dłuższe słupki oznaczają mniejszą oszczędność. Wymiany porcji mają $g_2 = 2^{-181}$; wybrana oszczędność końcowa to $\kappa = 2^{-182}$.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Przeczytaj jeden wiersz do końca. Przy wymianach w układzie transformaty jest $O(p/K)$ zamian szerokości K . Każda kosztuje $O(VK^\tau)$. Łącznie daje to

$$O(VpK^{\tau-1}) = O(Vp^{1-\epsilon c(1-\tau)}).$$

To dokładnie drugi margines g_2 . Małe wielkości się mnożą: mały wykładnik wzrostu szerokości ϵc i mała oszczędność zamiany $1-\tau$ dają jeszcze mniejszy margines końcowy. To wyjaśnia, dlaczego imponujący lokalny pomysł może dać w całym algorytmie bardzo mały wykładnik asymptotyczny.

Dlaczego rezerwujemy ostry zapas. Niech $G = \min_i g_i$. Ustalone potęgi $\log p$ można wchłoniąć, jeśli $\kappa < G$, bo $(\log p)^C = o(p^{G-\kappa})$ dla ustalonego C . Oryginalny ostrożny wybór daje $G = 2\kappa$. Koszt końcowy mieści się wtedy w

$$O(Vp^{1-\kappa}) = O(n(\log n)^{1-\kappa}).$$

Ten argument działa od pewnego miejsca. Jeden ustalony próg obejmuje wszystkie potrzebne nierówności; poniżej niego maszyna używa mnożenia szkolnego. Powiększenie stałej w notacji O obejmuje skończony zbiór mniejszych długości. Ustanowienie pełnej jednolitej implementacji pozostaje częścią zobowiązań dowodowych.

Co próbują poprawić późniejsze ulepszenia społeczności. Niektóre ulepszenia zmieniają skończoną sieć albo cenę przejść między jej ramami; inne ponownie wykorzystują pamięć roboczą, zmieniają sposób łączenia etapów albo ulepszają wybór parametru zewnętrznego. Działają one na różnych poziomach. Nowy skończony profil wciąż trzeba związać z jego rzeczywistym układem, a potem przeprowadzić przez odpowiednią rekurencję i analizę kosztów. Oryginalny diagram siedmiu marginesów pomaga czytać ten zamrożony manuskrypt; nie twierdzi, że każda późniejsza konstrukcja używa tego samego rachunku ani że pokazuje najnowsze świadectwo społeczności.

Co zabrać dalej. Dokładne odtworzenie wymaga ostrego ograniczenia błędu; wykładnik wymaga ostrego marginesu w każdym dominującym koszcie. Oba muszą odnosić się do tego samego algorytmu.

Miejsca w oryginale: [08-assembly.tex](#), [eq:fixed-parameters](#); [eq:margin-list](#); [eq:final-error](#).

Co właściwie mówi udane sprawdzenie

Podstawy Trzy twierdzenia, których nie należy mylić.

„To działa” może znaczyć różne rzeczy. Układ może zwracać właściwe wartości na określonym skończonym zbiorze wejść. Fizyczny program może wykonywać ten układ z udowodnionym ograniczeniem zasobów. Twierdzenie może obejmować każdą długość wejścia, także rozmiary, których żaden eksperyment nie uruchomi. Każde z nich jest cenne, ale to różne stwierdzenia.

Trzy różne twierdzenia wymagają trzech różnych rodzajów uzasadnienia

Przykład algebraiczny	Konkretny program taśmowy	Dowód dla każdego rozmiaru
Wyniki się zgadzają, a rejestr pomocniczy wraca do startu	Ruch głowic, kopie, trasowanie i sprzątanie są wliczone w koszt	Każdy rozmiar wejścia spełnia ograniczenia kosztu i błędu
Skończony test sprawdza konkretny przypadek	Sama tożsamość skalarna nie wycenia przemieszczeń	Skończony przebieg nie obejmuje wszystkich n

Rys. 24. Odczytaj twierdzenie na poziomie, który wspierają jego dowody. Kolumny rozdzielają skończoną algebrę, fizyczną implementację i analizę dla wszystkich rozmiarów. To zobowiązania, które trzeba połączyć, a nie konkurencyjne sposoby dowodzenia tego samego.

Ilustracja: Rohan Arun (Apache 2.0), etykiety przetłumaczone.

Konkretny przykład. Wykres rejestrów (rys. 10 w części 2) pokazuje, że cztery aktualizacje działają, gdy rejestr pomocniczy zaczyna od siedmiu. Rachunek symboliczny dowodzi odtworzenia dla dowolnych wartości w podanym pierścieniu. Żadne z tych stwierdzeń nie mówi, ile ruchu na taśmie potrzebuje implementacja na ogromnej tablicy. To wymaga innego argumentu. Podobnie sprawdzenie siedmiu wymiennych marginesów weryfikuje arytmetykę w ramach modelu kosztów; nie ustanawia każdego założenia algorytmicznego tego modelu.

Dlaczego przewodnik oznacza przykłady poglądowe. Duże, widoczne różnice wykładników pomagają się uczyć, ale przedstawione jako zmierzone przyspieszenie zniekształcałyby pracę. Małe macierze i tablice mogą pokazać niezmiennik, nie będąc rzeczywistą skończoną konstrukcją. Podpisy mówią, które liczby pochodzą ze źródła, a które są przykładami dydaktycznymi. Nowe generatory rysunków sprawdzają pokazaną arytmetykę, żeby ilustracje objaśniały prawdziwe tożsamości, a nie tylko sugestywne obrazki.

Po lekturze przewodnika. Czytelnik powinien umieć opisać, dlaczego splot pomaga, dlaczego przesuwanie danych kosztuje czas, dlaczego wspólne ramy zachowują dodawanie, jak ważona rekurencja oszczędza w wykładniku i dlaczego odtworzenie wymaga ostrego budżetu błędów. Prześledzenie każdego szczegółu twierdzenia źródłowego wciąż wymaga dalszej nauki algebry liniowej, analizy Fouriera i dowodów złożonościowych. Wydanie rozszerzone umieszcza oryginalne argumenty bezpośrednio po materiale objaśniającym.

Co zabrać dalej. Zrozumienie architektury to istotny krok. Nie jest to jednak to samo co niezależne zweryfikowanie każdej części twierdzenia.

Co z tego wynika i co wciąż trzeba sprawdzić

Technika Do przeczytania przed końcowymi wnioskami i dodatkiem.

Dzielenie i pierwiastek kwadratowy. Szybkie mnożenie jest elementem składowym innych działań arytmetycznych. Iteracje w stylu Newtona poprawiają przybliżenia odwrotności albo pierwiastka kwadratowego przy geometrycznie rosnącej precyzji. Jeśli koszt przy precyzji n to $M(n)$, a mniejsze etapy kosztują $M(n/2), M(n/4), \dots$, to przy regularnym ograniczeniu z tego manuskryptu sumą rządzi największy etap. Przybliżone wyniki wciąż trzeba skorygować: dzielenie sprawdza iloraz i resztę, a całkowity pierwiastek kwadratowy używa dokładnych porównań kwadratów. Sama procedura przybliżająca nie jest jeszcze algorytmem dokładnego dzielenia ani części całkowitej pierwiastka.

Transpozycja macierzy. Praca powołuje się na zewnętrzną redukcję mnożenia do przestawienia macierzy z porządku wierszowego w kolumnowy. Uwzględnia całkowitą liczbę bitów macierzy, a nie tylko długość jej boku. To wynik o permutacji danych. Nie jest to twierdzenie o wykładniku mnożenia macierzy, które jest innym problemem. Redukcja i konwersje formatu wejścia zachowują własne założenia i koszty.

Dodatek o pogrupowanych prostokątach. Dodatek rozszerza operacje na upakowanych adresach na zbiór kompletnych bloków prostokątnych. „Prostokąt” oznacza tu iloczyn kartezjański przedziałów adresów, a nie geometryczne przybliżenie dowolnych danych. Pełne dane i informacje o miejscu docelowym muszą być wliczone w koszt, gdy wyjątkowe rekordy są wyjmowane i naprawiane. Ten materiał wspiera szczegóły układu danych, a nie wprowadza osobnego głównego ulepszenia.

Praktyczna lista kontrolna. Oceniając proponowane ulepszenie, zadaj te pytania po kolei:

1. Jaką dokładnie operację skalarną wykonuje wygenerowany układ przy dowolnych początkowych wartościach pomocniczych? Czy przywraca obiecaną pamięć roboczą?
2. Czy wszystkie wystąpienia każdej bramki mają wymaganą wspólną ramę? Czy uwzględniono wszystkie zmienione operatory, znaki, wkłady źródeł i końce?
3. Czy opłacony profil rzędów opisuje ten układ, a nie wersję złagodzoną albo schematyczną? Czy specjalizacje do ciał skończonych są uzasadnione?
4. Czy implementacja rekurencyjna naprawdę daje każdemu dziecku objętość V/W , z wliczonym trasowaniem, dopełnianiem, kopiowaniem i odtwarzaniem?
5. Czy kontrakty precyzji i przepróbkowania nadal obowiązują dla nowych układów danych?
6. Czy każdy końcowy margines kosztu i błąd odtworzenia mieszczą się ostro w wymaganych granicach przy tym samym wyborze parametrów?

Te pytania rozdzielają algebrę, implementację i analizę asymptotyczną. Sukces na jednym poziomie jest użytecznym świadectwem na tym poziomie; nie załatwia po cichu pozostałych.

Weryfikacja wykonana dla tego wydania. Zamrożone pliki oryginału sprawdzono z dostarczonym manifestem SHA-256. Ponownie obliczono przykład splotu cyfr, tożsamość odtwarzania w czterech krokach na 343 trójkach liczb całkowitych, przykład motylka na dwóch osiach i wszystkie siedem oryginalnych marginesów wymiernych. Drugie wydanie sprawdza też interpolację, rekonstrukcję wzorców częstotliwości, tożsamości ram dla wektorów i tablic, znoszenie się przecięć, zgodność permutacji i pogładowe budżety błędów. Rysunki są generowane z tych przykładów albo z jawnie oznaczonych tożsamości schematycznych. Kompilacja i przegląd wyrenderowanych stron

sprawdzają integralność i czytelność dokumentu. Żadne z tych działań nie stanowi nowego audytu pełnego twierdzenia o mnożeniu.

Co zabrać dalej. Przewodnik wyjaśnia architekturę dowodu i jego interfejsy. Szczegółowe argumenty zawiera materiał źródłowy; niezależna weryfikacja pełnego twierdzenia to osobne zadanie badawcze.

Oznaczenia i mapa źródeł

Technika Do korzystania podczas lektury części technicznych.

Symbol lub termin	Znaczenie w przewodniku i oryginalnej argumentacji
n	Długość każdej liczby wejściowej w bitach.
b	Szerokość cyfry w bitach w oryginalnym składaniu.
p	Precyzja robocza; oryginalnie $p = 6\lceil \log_2 n \rceil$.
V	Logiczna objętość przechowywanych bitów w węźle rekurencji.
W	Liczba wszystkich ról w ustalonej skończonej sieci, łącznie z pomocniczymi.
m	Ustalony wymiar sieci i mnożnik szerokości w rekurencji.
s	Liczba opłaconych mniejszych wywołań w omówieniu rekurencji; w przepróbkowaniu ta sama litera oznacza długość źródła.
d	Liczba osi transformaty, rosnąca powoli z precyzją.
K	Szerokość upakowanej porcji adresu; skaluje się jak $p^{\epsilon c}$.
r, ℓ	Długość wielomianu i jej logarytm dwójkowy.
T	Rozmiar dopełnionego pudełka transformaty; $Tp = \Theta(n)$. Nie mylić z funkcją czasu $T(n)$ ani z trójką oznaczoną T .
τ, σ	Wymierne górne wykładniki dla rekurencji bitowej i zespolonej.
λ'	Wykładnik w koszcie ukończonej warstwy jednoczesnej, po narzucie.
κ	Końcowa oszczędność w wykładniku $\log n$.
rola / wiersz	Ustalona pozycja przewodu w układzie / kompletny ciągły blok sufiksu.
rama	Odwracalny operator reprezentacji na tablicy indeksowanej adresami w jednej roli.

Symbol lub termin	Znaczenie w przewodniku i oryginalnej argumentacji
deficyt rzędu	Różnica między pełnym liniowym budżetem wywołań Wm a opłaconym budżetem s .
diadyczna liczba Gaussa	Liczba zespolona z mianownikami będącymi potęgami dwójki w obu składowych.
warunkowy	Wniosek zależy od zachowanych interfejsów, których pełnej niezależnej walidacji nie zapewnia skończone świadectwo.

Gdzie znaleźć poszczególne warstwy oryginału. Nazwy plików poniżej są zachowane w pakiecie źródłowym. Etykiety twierdzeń są stabilniejszymi odnośnikami niż numery stron.

Pytanie	Plik oryginału
Jaka maszyna jest liczona?	02-streams.tex
Co robi skończona sieć?	03-motifs.tex
Jak rząd wycenia przemieszczenia?	04-swap.tex
Jak grupuje się wiele motylków?	05-layers.tex
Jak wykonuje się transformaty i iloczyny w pierścieniu?	06-transforms.tex
Jak pasują rozmiary siatek i permutacje?	07-resampling.tex
Gdzie łączy się dokładność i wszystkie koszty?	08-assembly.tex
Jakie są dalsze konsekwencje?	09-exact-arithmetic.tex
	10-transposition.tex

Od tłumacza: słowniczek polsko-angielski

Dodatek tłumacza

Część terminów ma ustalone polskie odpowiedniki, ale wiele pochodzi z najnowszej literatury (Harvey i van der Hoeven, praca OpenAI) albo jest specyficzna dla tej jednej konstrukcji i nie ma polskiej nazwy. Kolumna „Status” rozróżnia te przypadki. „Ustalony” oznacza, że termin w tym znaczeniu znalazłem w polskich źródłach wymienionych pod tabelą (ogólnie znanych terminów, takich jak splot czy maszyna Turinga, nie dokumentuję osobno). „Przyjęty w przekładzie” oznacza, że polskiego odpowiednika nie znalazłem i zaproponowałem własny; przy pierwszym użyciu w tekście podany jest wtedy termin angielski.

Polski	Angielski (oryginał)	Status
splot; splot cykliczny (kołowy)	convolution; cyclic (circular) convolution	ustalony
szybka transformacja Fouriera	fast Fourier transform	ustalony
motylek, graf motylkowy	butterfly, butterfly graph	ustalony
kolejność z odwróconymi bitami	bit-reversed order	ustalony
pierwiastek z jedynki	root of unity	ustalony
logarytm iterowany $\log^*$	iterated logarithm	ustalony

Polski	Angielski (oryginał)	Status
kontrakcja (odwzorowanie zwężające)	contraction	ustalony
element podstawowy	pivot	ustalony
liczby całkowite Gaussa	Gaussian integers	ustalony
przepróbkowanie	resampling	ustalony
przeniesienie; podstawa systemu liczbowego	carry; radix	ustalony
rząd macierzy, podprzestrzeń, iloczyn tensorowy	rank, subspace, tensor product	ustalony
maszyna Turinga, taśma, głowica	Turing machine, tape, head	ustalony
przepróbkowanie gaussowskie	Gaussian resampling	przyjęty w przekładzie
splot / zawijanie	negacyclic convolution / wrapping	przyjęty w przekładzie
ujemnocykliczne chirp, algorytm Bluesteina	chirp, Bluestein's (chirp-z) algorithm	brak polskiego odpowiednika ^a
diadyczna liczba Gaussa	Gaussian dyadic	przyjęty w przekładzie
ścinanie adresów	address shear	przyjęty w przekładzie ^b
rama (reprezentacji)	frame	przyjęty w przekładzie ^c
rząd resztowy, deficyt rzędu	residual rank, rank deficit	przyjęty w przekładzie
rola, wiersz, porcja adresu	role, row, address chunk	przyjęty w przekładzie ^c
rejestr pomocniczy, pamięć robocza	scratch register, workspace	opisowy
margines, wąskie gardło	margin, bottleneck	opisowy
przykład poglądowy	toy example	opisowy
warunkowy (wynik)	conditional (result)	opisowy

^a Słowniki podają tylko potoczne „świergot” lub „ćwierkanie”.

^b „Ścinanie” jest ustalone w mechanice; jako przekształcenie liniowe pojawia się po polsku głównie w dokumentacji programów graficznych.

^c Terminy specyficzne dla tej jednej konstrukcji.

Źródła terminologii

- splot cykliczny (kołowy): Wikipedia, [Splot \(analiza matematyczna\)](#), sekcja „Splot cykliczny (kołowy)”
- szybka transformacja Fouriera, struktury motylkowe: Wikipedia, [Szybka transformacja Fouriera](#)
- motylek, graf motylkowy, kolejność z odwróconymi bitami: K. Snopek, [wykład o DFT i FFT](#), Politechnika Warszawska
- pierwiastek z jedyńki: Wikipedia, [Pierwiastek z jedyńki](#)
- logarytm iterowany: Wikipedia, [Logarytm iterowany](#)
- kontrakcja (odwzorowanie zwężające): zasada odwzorowań zwężających Banacha, np. [skrypt UG o metrycznej teorii punktów stałych](#)

- element podstawowy (pivot): P. F. Góra, [wykład z metod numerycznych](#), Uniwersytet Jagielloński
- liczby całkowite Gaussa: Wikipedia, [Liczby całkowite Gaussa](#)
- przepróbkowanie: polski opis patentowy [PL3053163T3](#) („przepróbkowanie sygnału audio”)

Uwaga o „transformacie”: polska Wikipedia rozróżnia *transformację* (przekształcenie) i *transformatę* (jego wynik). W tekście, zgodnie z częstym użyciem w fizyce i przetwarzaniu sygnałów, „transformata” oznacza czasem także samo przekształcenie.

Źródła

1. Rohan Arun, *Integer multiplication, illustrated*, <https://github.com/rohanarun/integer-multiplication-illustrated>, Apache 2.0.
2. OpenAI, *Integer multiplication below $n \log n$* , 23 września 2026, commit [adc7f1241b42e322a6451854ab7e4b4c146bf78a](https://github.com/openai/math/tree/adc7f1241b42e322a6451854ab7e4b4c146bf78a), <https://github.com/openai/math/tree/adc7f1241b42e322a6451854ab7e4b4c146bf78a/preprints/Integer-multiplication-below-n-log-n-September-23-2026>.
3. CrocSwap/integer-mult-bounds, <https://github.com/CrocSwap/integer-mult-bounds>; lista współtwórców: <https://github.com/CrocSwap/integer-mult-bounds/blob/main/CONTRIBUTORS.md>.
4. D. Harvey, J. van der Hoeven, *Integer multiplication in time $O(n \log n)$* , Annals of Mathematics 193 (2021).