

Funkcje

poniedziałek, 1 kwietnia 2024 19:06

Funkcja:

- a) Z pozycji programisty funkcje są doskonałe. Z definicji można je wywołać z dowolnego punktu i nie psują kodu. (jeśli wszystko jest dobrze)
- b) Pamięć zajęta w mniejszym stopniu, ale
Wymaga:
 - Położenia argumentów na stosie
 - Zmiennych lokalnych
 - Adresu powrotu na stosie

Wszystko to trzeba zdejmować.

Więc różnie może być!

Narzut czasowy!

Trzeba jeszcze pamiętać o samej obsłudze: instrukcje **CALL** i **RET** oraz **operacjach na stosie służących do przekazania argumentów**, bo te instrukcje też muszą zostać wywołane dodatkowo do tego co chcemy policzyć.

Pierwsze programy w języku C z funkcjami traciły **połowę czasu** na samo wywoływanie funkcji i powroty z nich.

Wielobieżność (ang. *reentrancy*) – jest to cecha **funkcji**. Funkcja jest określana jako wielobieżna jeżeli wykonywanie jej może zostać przerwane (poprzez **przerwanie** lub wywołanie innej funkcji wewnątrz ciała funkcji), a potem może ona zostać (bezpiecznie) ponownie wywołana zanim poprzednie wywołanie zostanie zakończone. Po zakończeniu drugiego wywołania, można wrócić do przerwanej wywołania, a wykonywanie go może bezpiecznie kontynuować^[1]. Funkcje wielobieżne można więc wywoływać **rekurencyjnie**.

Dla funkcji, która **nie jest wielobieżna nie da się** zrobić czegoś takiego, że:

1. Zaczynamy wywoływać funkcję (nazwijmy to wywołaniem A)
2. Przerwywamy wywoływanie A
3. Wywołujemy teraz tę funkcję drugi raz (wywołanie B)
4. Kończymy wywołanie B
5. Wracamy do wywołania A i je kończymy.

Uwaga! Tomczak nazywa wielobieżność też **WIELOWEJŚCIOWOŚCIĄ!**

Największym problemem jest dostęp do danych, które są współdzielone pomiędzy różnymi uruchomieniami funkcji.

Jeśli funkcja korzysta tylko z własnych argumentów to **git**.

Jeśli operuje na zmiennych współdzielonych **nie git**.

Kiedy funkcja może zostać przerwana?

1. Funkcja działa i pojawia się zewnętrzne zdarzenie obsługiwane przez procesor w ten sposób, że ta funkcja znowu jest wywoływana.
2. (Ciekawszy) Programy wielowątkowe, szczególnie uruchamiane na maszynach wielordzeniowych: ta sama funkcja wywołuje się kilkakrotnie na kilku różnych fizycznych urządzeniach dzieląc dostęp do tych samych danych

Jak sobie radzić?

NIE STOSOWAĆ DANYCH WSPÓŁDZIELONYCH (nie zawsze się uda)

Wydajność

środa, 3 kwietnia 2024

21:38

- I. Wydajność - sposób ilościowego określania wykonanej pracy. Nas interesuje przede wszystkim **szybkość!**

Jeden z aspektów wydajności - (WAŻNY TERMIN!) Szybkość przetwarzania - parametry:

1. Opóźnienie (ang.latency) - czas potrzebny na wykonanie operacji, od momentu rozpoczęcia do zakończenia
2. Przepustowość (ang.throughput) - ilość pracy, którą można wykonać podczas jakiegoś czasu
SĄ ZWIĄZANE ZE SOBĄ PRAWEM LITTLE-A! (Wykład 6)

Prawo Little'a

$$n = T \cdot l$$

W stanie ustalonym (**stacjonarnym**) **średnia** liczba aktualnie wykonywanych operacji n jest iloczynem tempa napływania operacji T (przepustowości) i czasu trwania operacji (opóźnienia) l .

Jak to rozumieć:

1. Opóźnienie -----> szybkość pojazdu
2. Przepustowość --> pojemność pojazdu

"Żeby szybko dostać się z Wrocławia do Warszawy użyjemy motocykla, ale gdybyśmy mieli przewieźć 100 ton kamienia to już trudniej".

Parametr związany z opóźnieniem np. czas wykonania programu zmierzony od rozpoczęcia do zakończenia rozwiązywania konkretnego problemu. **Im krócej to zajmie tym większa szybkość przetwarzania**

CZAS WYKONANIA ODWROTNIE PROPORJONALNY DO SZYBKOŚCI PRZETWARZANIA

- II. Czas rzeczywisty/ ścienny - rzeczywisty czas, który upłynął (często nie odpowiada czasowi, który został faktycznie zużyty na wykonanie pracy)
 - III. Czas procesora - czas wykorzystany przez procesor na obliczenia związane z konkretnym zadaniem (komputery wykonują przeważnie wiele rzeczy naraz, a jeszcze częściej nie wykonują nic tylko czekają)
 - A) Czas użytkownika - czas wykorzystany na nasze obliczenia w naszym programie
 - B) Czas system - czas wykorzystany na funkcje systemowe
 - C) Czas Jałowy - czas, w którym procesor nic nie robi
I wiele innych np.
 - D) Steal time - czas związany z przełączaniem maszyn wirtualnych, zadania związane z obsługą samej wirtualizacji, przełączanie kontekstu
- Odpowiedź na pytanie z wykładu dotyczące przerwań i steal time:

Zewnętrzne przerwania to czas obok procesu, wydłużają czas rzeczywisty, ale nie czas procesora, znajdują się w różnicy między nimi.

Jak zmierzyć czas?

środa, 3 kwietnia 2024 22:11

Jak zmierzyć czas wykonania programu:

1. Za pomocą narzędzi w systemie, programy, które można uruchomić i zobaczyć jaki jest czas wykonania programu time, top, htop, ...

To co nas szczególnie interesuje to:

2. instrukcje pozwalające na pomiar czasu z dużą rozdzielczością
3. Sprzętowe liczniki w procesorze, które zliczają różne zdarzenia, które się zdarzają w procesorze np. cykl zegara, wykonania instrukcji, odstęp do pamięci, przerwanie, tych różnych zdarzeń są setki

RDTSC—Read Time-Stamp Counter

Opcode	Instruction	Description
0F 31	RDTSC	Read time-stamp counter into EDX:EAX

Description

Loads the current value of the processor's time-stamp counter into the EDX:EAX registers. The time-stamp counter is contained in a 64-bit MSR. The high-order 32 bits of the MSR are loaded into the EDX register, and the low-order 32 bits are loaded into the EAX register. The processor monotonically increments the time-stamp counter MSR every clock cycle and resets it to 0 whenever the processor is reset. See "Time Stamp Counter" in Chapter 15 of the IA-32 Intel Architecture Software Developer's Manual, Volume 3 for specific details of the time stamp counter behavior.

The RDTSC instruction **is not a serializing instruction**. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, **subsequent instructions may begin execution before the read operation** is performed.

Tutaj wszystko rozchodzi się o licznik TSC (Time Stamp Counter). Licznik zerowany podczas resetu procesora, z mniej więcej stałą częstotliwością zwiększany, odczytując go w pewnych momentach w czasie i obliczając różnicę tych dwóch odczytów można stwierdzić ile minęło cykli, które ten licznik zliczył. Dość dokładny, pozwala nawet mierzyć czas pojedynczych instrukcji jeśli trwają odpowiednio długo.

Ta instrukcja jedynie przepisuje wartość TSC do rozszerzonego akumulatora EDX:EAX gdzie do EDX trafiają bardziej znaczące bity, a do EAX mniej znaczące.

- I. Instrukcja nieserializująca to taka instrukcja, która może się zacząć wykonywać w dowolnym momencie niezależnie od instrukcji, które się wokół niej znajdują.

Program składający się z sekwencji instrukcji - **instrukcje nie są wykonywane po kolei** tylko jak się dało. Procesor gwarantuje jednak, że **wynik tej sekwencji instrukcji będzie taki jakby były wykonywane po kolei**. Jakby były wykonywane po kolei **nie byłoby tak szybko**.

Więc jak sprawić, żeby TSC zmierzył dokładnie czas przed tymi, które chcemy zmierzyć i po? Będziemy potrzebować w miarę dowolnej funkcji serializującej.

Funkcje serializujące

środa, 3 kwietnia 2024 23:19

- I. Funkcja serializująca gwarantuje, że wszystkie poprzednie funkcje wykonają się przed jej wykonaniem. Ta szczególna cecha funkcji serializujących jest bardzo przydatna podczas mierzenia czasu wykonywania części programu.

Przykładowa funkcja serializująca CPUID:

CPUID—CPU Identification

Opcode	Instruction	Description
0F A2	CPUID	Returns processor identification and feature information to the EAX, EBX, ECX, and EDX registers, according to the input value entered initially in the EAX register

Description

Returns processor identification and feature information in the EAX, EBX, ECX, and EDX registers. The information returned is selected by entering a value in the EAX register before the instruction is executed. Table 3-7 shows the information returned, depending on the initial value loaded into the EAX register.

The ID flag (bit 21) in the EFLAGS register indicates support for the CPUID instruction. If a software procedure can set and clear this flag, the processor executing the procedure supports the CPUID instruction.

The information returned with the CPUID instruction is divided into two groups: basic information and extended function information. Basic information is returned by entering an input value of from 0 to 3 in the EAX register depending on the IA-32 processor type; extended function information is returned by entering an input value of from 80000000H to 80000004H. The extended function CPUID information was introduced in the Pentium 4 processor and is not available in earlier IA-32 processors. Table 3-8 shows the maximum input value that the processor recognizes for the CPUID instruction for basic information and for extended function information, for each family of IA-32 processors on which the CPUID instruction is implemented.

```
10  xor %eax, %eax          $ echo performance > /sys/firmware/acpi/platform_profile
11  cpuid                  $ cpupower -c all frequency-set --governor userspace
12  rdtsc                  $ cpupower -c all frequency-set -f 5000000
13  mov %eax, %esi          $ taskset -c 1 ./p
14  mov %edx, %edi          961775
15                          964825
16  mov $1000000, %ecx      513550
17  L: loop L              502575
18                          502625
19  xor %eax, %eax          502600
20  cpuid                  530300
21  rdtsc                  502550
22                          502625
23  sub %esi, %eax          502600
24  sbb %edi, %edx
```

7

Żeby otrzymać czas wykonania programu:

1. Wyzerowujemy akumulator (potrzebne do tego, żeby CPUID działało zawsze tak samo, cokolwiek to znaczy)
2. Wykonujemy CPUID
3. Zapisujemy wartość TSC w rejestrach (rdtsc)
4. Przenosimy wartości do innych rejestrów (tutaj: esi i edi)
5. Teraz badany fragment programu
6. Wyzerowujemy akumulator
7. Wykonujemy CPUID

8. Zapisujemy wartość TSC w rejestrach
9. Odejmujemy wartość eax od esi i edx od edi.esi

Pytanie z widowni: Dlaczego rdtsc nie jest samo w sobie funkcją serializującą?

Odpowiedź: Ta funkcja ma być prosta i szybka, dla różnych celów, jeśli chcemy wykorzystać ją do zmierzenia czasu, co jest głównym założeniem musimy stosować inne mechanizmy, jeśli chcielibyśmy tylko odczytać wartość z TSC możemy to jednak zrobić bez żadnych dodatkowych kosztów np. ile godzin upłynęło od uruchomienia komputera (wypowiedź Tomczaka sugeruje tutaj, że procesor resetuje się z każdym wyłączeniem/włączeniem komputera).

Wypowiedź Tomczaka sugeruje, że korzystanie z funkcji serializującej wiąże się z dodatkowymi kosztami co ma sens, bo w końcu uniemożliwiamy optymalne wykonywanie programu w celu upewnienia się, że zostaną wykonane wszystkie instrukcje, które chcemy.

UWAGA! Ta metoda pomiaru nie jest najlepsza, jest jednak stosunkowo prosta.

```
$ echo performance > /sys/firmware/acpi/platform_profile
$ cpupower -c all frequency-set --governor userspace
$ cpupower -c all frequency-set -f 5000000
$ taskset -c 1 ./p
961775
964825
513550
502575
502625
502600
530300
502550
502625
502600
```

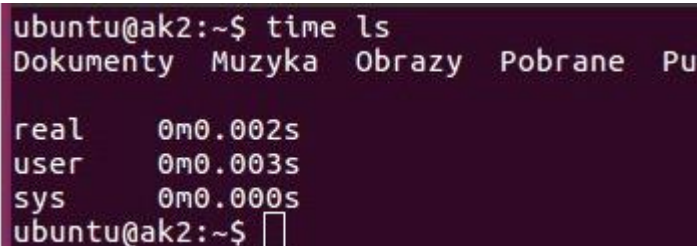
7

Pierwszych kilka odczytów jest coraz mniejszych składa się na to wiele powodów. Szczególnie pierwsze uruchomienia procesu są różne. Kiedy czasy się ustabilizowały to są rzędów pół miliona cykli. W tym przypadku wykonujemy milion iteracji pętli czyli 2 miliony instrukcji. Licznik w procesorze, na którym to było mierzone jest taktowany ze stałą częstotliwością 2,5 MHz. Jednak tutaj wcześniej częstotliwość pracy procesora była ustawiona na 5GHz, żeby ładniej wyszło. Został użyty program, który przywiązał proces do rdzenia procesora. No i to jest podstawowy szkielet do mierzenia czasu wykonania średnio długich fragmentów kodu.

Trzy kroki:

1. Ustawienie częstotliwości procesora, żeby się sama nie zmieniała.
2. Powiązanie danego procesu z konkretnym rdzeniem.
3. Zmierzenie.

Program time, mniejsza rozdzielczość:



```
ubuntu@ak2:~$ time ls
Dokumenty  Muzyka  Obrazy  Pobrane  Pu
real      0m0.002s
user      0m0.003s
sys       0m0.000s
ubuntu@ak2:~$
```

Czas użytkownika + czas systemu = czas rzeczywisty (czego tu w sumie nie widać)

Czym jest czas?

czwartek, 4 kwietnia 2024 00:29

Z tego wzoru:

$$\text{czas} = \frac{\text{instrukcje}}{\text{program}} \cdot \frac{\text{cykle zegara}}{\text{instrukcję}} \cdot \frac{\text{sekundy}}{\text{cykl zegara}}$$

Liczba instrukcji * czas wykonania poszczególnych instrukcji (tzn. liczba cykli zegara przeznaczonych na instrukcje * czas cyklu procesora)

Ten model uwzględnia to, że różne instrukcje mogą zajmować różną liczbę cykli, a każdej takiej instrukcji jest różna liczba w programie (operacje rejestrowe są bardzo szybkie, operacje na pamięci bardzo wolne):

Czas wykonania programu

$$t_p = T_{CPU} \cdot \sum_{i_t} N_{i_t} \cdot C_{i_t}$$

t_p - czas wykonania programu

T_{CPU} - czas cyklu procesora

N_{i_t} - liczba wykonań instrukcji typu i_t

C_{i_t} - średnia liczba cykli dla instrukcji typu i_t

Czas cyklu procesora

$$T_{CPU} = \frac{1}{f_{CPU}}$$

Odwrotność częstotliwości taktowania procesora (częstotliwość: jednostka herc, skrót **Hz** = 1 / s = s⁻¹), zależy od:

- Konstrukcji układów logicznych (organizacji komputera)
- Technologii wykonania
- Złożoności instrukcji
- Ustawień maszyny
- Ograniczeń termicznych (*ang. thermal constraints*)
- Temperatury otoczenia
- Ograniczeń poboru mocy (*ang. power constraints*)
- Liczby działających procesów
- Wydajności kodu (układu instrukcji)

Tomczak wspomina, że trzeba dobre 4,2J, żeby jeden gram wody podnieść o 1 stopień

Przedrostki SI

jotta	Y	10 ²⁴	1 000 000 000 000 000 000 000 000	kwadrylion
zetta	Z	10 ²¹	1 000 000 000 000 000 000 000	tryliard
eksa	E	10 ¹⁸	1 000 000 000 000 000 000	trylion
peta	P	10 ¹⁵	1 000 000 000 000 000	biliard
tera	T	10 ¹²	1 000 000 000 000	bilion
giga	G	10 ⁹	1 000 000 000	miliard
mega	M	10 ⁶	1 000 000	milion
kilo	K	10 ³	1 000	tysiąc
hekto	h	10 ²	100	sto
deka	da	10 ¹	10	dziesięć
		10 ⁰	1	jeden
decy	d	10 ⁻¹	0,1	jedna dziesiąta
centy	c	10 ⁻²	0,01	jedna setna
mili	m	10 ⁻³	0,001	jedna tysięczna
mikro	μ	10 ⁻⁶	0,000 001	jedna milionowa
nano	n	10 ⁻⁹	0,000 000 001	jedna miliardowa
piko	p	10 ⁻¹²	0,000 000 000 001	jedna bilionowa
femto	f	10 ⁻¹⁵	0,000 000 000 000 001	jedna biliardowa ¹⁴
atto	a	10 ⁻¹⁸	0,000 000 000 000 000 001	jedna trylionowa

Przedrostki dwójkowe

IEC		podstawa						SI	
nazwa	symbol	2	16		różnica	10		nazwa	symbol
kibi	Ki	2 ¹⁰	16 ^{2,5}	400 ₁₆	2,40%	1 024	> 10 ³	kilo	k
mebi	Mi	2 ²⁰	16 ⁵	10 000 ₁₆	4,86%	1 048 576	> 10 ⁶	mega	M
gibi	Gi	2 ³⁰	16 ^{7,5}	4000 000 ₁₆	7,37%	1 073 741 824	> 10 ⁹	giga	G
tebi	Ti	2 ⁴⁰	16 ¹⁰	100 000 000 ₁₆	9,95%	1 099 511 627 776	> 10 ¹²	tera	T
pebi	Pi	2 ⁵⁰	16 ^{12,5}	4 000 000 000 ₁₆	12,59%	1 125 899 906 842 624	> 10 ¹⁵	peta	P
eksbi	Ei	2 ⁶⁰	16 ¹⁵	1000 000 000 000 ₁₆	15,29%	1 152 921 504 606 846 976	> 10 ¹⁸	eksa	E
zebi	Zi	2 ⁷⁰	16 ^{17,5}	40 000 000 000 000 ₁₆	18,06%	1 180 591 620 717 411 303 424	> 10 ²¹	zetta	Z
jobi	Yi	2 ⁸⁰	16 ²⁰	1 000 000 000 000 000 ₁₆	20,89%	1 208 925 819 614 629 174 706 176	> 10 ²⁴	jotta	Y

JAK W PIEKARNIKU

sobota, 22 czerwca 2024

11:29



Pobór mocy

- Xeon w9-3495X
 - 19.08 cm², TDP 420 W Nominalna moc tracona
 - Powierzchniowa gęstość mocy: **22 W/cm²** 420/19,08 ≈ 22
 - 5.5 GHz: 1.88 kW, chłodzenie ciekłym azotem, **99 W/cm²** NAJLEPSZY WYNIK
- i9-14900KS: 2.57 cm², moc max. turbo 253 W, **98 W/cm²**
- Czajnik elektryczny: 2 kW, średnica ok. 14 cm, 10 W/cm²
- Palenisko kotła węglowego: 70-80 W/cm²
- Powierzchnia słońca: 6.3 kW/cm²
- Wzrost temperatury o 10 stopni dwukrotnie skraca trwałość

16

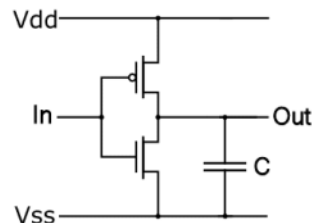
Jak widać procesor grzeje się to w tej chwili bardziej niż rozżażony węgiel w piecu

BARDZO WAŻNY WZÓR, SKĄD SIĘ BIERZE MOC TRACONA W UKŁADACH CYFROWYCH:

$$P = P_{dynamic} + P_{static}$$

$P_{dynamic}$ - moc dynamiczna (ang. *dynamic power*)

P_{static} - moc statyczna (ang. *static power*)



17

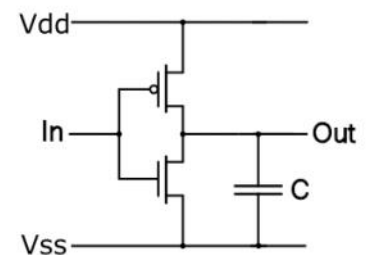
Widać dwa tranzystory, raz jeden się wyłącza, a drugi się włącza, a potem drugi się włącza i pierwszy się wyłącza.

Albo ten górny podłącza wyjście do zasilania, albo ten dolny do masy.

Wyjście tego układu jest zazwyczaj podłączone do wejścia następnego, a to są tranzystory CMOS.

Moc dynamiczna:

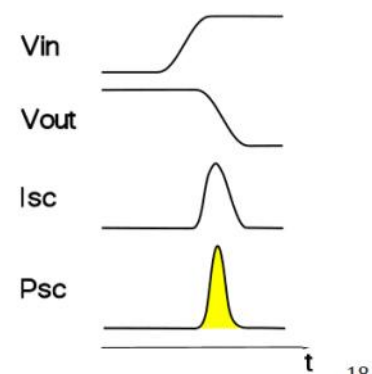
$$P_{dynamic} = \frac{C \cdot U^2 \cdot f}{2} + P_{sc}$$



P_{sc} - moc zwarcia (ang. *short-circuit power*)

W rzeczywistości:

$$P_{dynamic} \sim U^3 \sim U^4 \sim f^3$$



18

Skąd się bierze moc dynamiczna? Energia jest magazynowana w kondensatorze, a potem tracona w tranzystorze.

Z fizyki:

$CU^2/2$ energia magazynowana w kondensatorze.

To ładowanie następuje co cykl zegara pi razy oko i mamy moc traconą dynamiczną na przełączaniu.

Drugim elementem jest ten moment kiedy przełączamy stan na wyjściu.

W momencie kiedy jeden się nie włączył całkiem, a drugi się nie wyłączył całkiem:



W pewnym momencie płynie taka szpilka prądu zwarcowego, która jest krótka, ale wysoka:



i jak przemnożymy przez to napięcie to wychodzi nam moc zwarcowa:



Przy każdym przełączeniu mamy zwarcie. W tym momencie są techniki radzenia sobie z tym, ale z naszego punktu widzenia to może być ważne jeśli wykonywalibyśmy układy cyfrowe na płytkach drukowanych.

Prymitywne wytłumaczenie:

Te szpilki powodują, że te chwilowe pobory prądu są bardzo duże dlatego trzeba stosować odsprężanie czyli kondensatory "filtrujące", które są bardzo blisko wyprowadzeń układu scalonego, żeby w momencie szpilek pobierać prąd z kondensatora blisko, a nie wiadomo jak.

Żeby uzyskać większą częstotliwość taktowania musimy podnieść napięcie. Rzeczywista zależność jest w 3 lub 4 potęgę.

Nie jesteśmy na liniowej zależności wydajności od traconej mocy, ta zależność jest wielomianem trzeciego lub czwartego stopnia. Dlatego nie da się podnosić częstotliwości w nieskończoność.

Moc statyczna:

$$P_{static} = V \cdot I_{leak}$$

I_{leak} - prąd upływu (*ang. leakage current*)

$$I_{leak} \sim k \cdot e^{\frac{-q \cdot U_T}{a \cdot k_a \cdot T}}$$

napięcie progowe
(*ang. threshold voltage*)

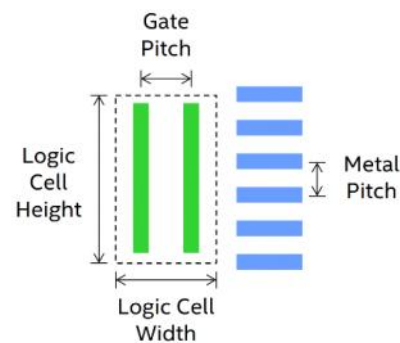
temperatura

Zależność wykładnicza. Im niższe napięcie tym prąd upływu jest większy. Im wyższa temperatura to ten prąd upływu jest większy. Im układ się bardziej grzeje tym się bardziej grzeje.

CIEKAWOSTKA:

Odległość pomiędzy atomami krzemu to ok. **0.24 nm**

Process	Gate pitch	Metal pitch	Year
7 nm	60 nm	40 nm	2018
5 nm	51 nm	30 nm	2020
3 nm	48 nm	24 nm	2022
2 nm	45 nm	20 nm	2024
1 nm	42 nm	16 nm	2026



W krzemie odległość między atomami to ćwierć nanometra.

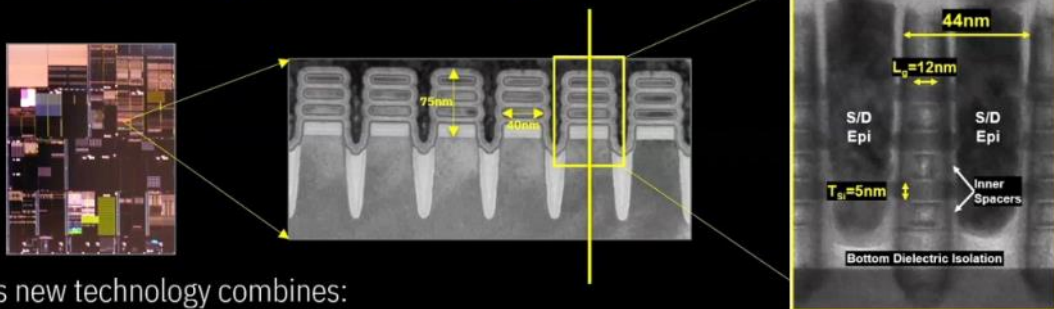
W tej chwili mówi się o technologiach na poziomie dwóch lub jednego nanometra. Trzech na pewno. Tych warstw atomowych jest na tyle mało, że one słabo izolują. Ile może nam zaizolować 100 warstw atomowych?

Z tego powodu prądy upływu są dość duże.

CIEKAWOSTKA:

Z produkcją procesorów jest różnie, ponad połowę się wyrzuca, bo jest za dużo defektów.

- IBM will announce a *new breakthrough* in semiconductor scaling, the world's first chip with 2nm technology.



- This new technology combines:
 - An industry-first *Bottom Dielectric Isolation* to enable 12nm gate length
 - A 2nd generation *Inner Spacer dry process* for precise gate control
 - EUV patterning* to produce variable Nanosheet widths from 15nm to 70nm
 - A novel *Multi-Vt scheme* for both SoC and HPC applications
- Expected to offer 45% performance improvement or 75% power reduction compared to 7nm

I NA TYM KOŃCZYMY PIERWSZY ELEMENT, KTÓRY SKŁADA SIĘ NA WYDAJNOŚĆ CZYLI CZĘSTOTLIWOŚĆ TAKTOWANIA

Liczba wykonań instrukcji

Zależy od:

- Użytego algorytmu
- Architektury listy rozkazów
- Języka programowania
- Jakości implementacji
- Jakości kompilatora lub interpretera
- Ustawień kompilacji
- Użytych bibliotek
- Danych wejściowych

22

Dobry kompilator i dobre biblioteki potrafią sobie poradzić z kiepskim kodem jeżeli się dobrze tego użyje. Napisanie bardzo szybkiego kodu jest jednak trudne i zależy w ogromnej mierze od programisty.

Kompilator znajduje braki zależności w kodzie i je poprawia, żeby było lepiej.

Powinniśmy testować wydajność kodu na danych, na których będzie zazwyczaj pracował program.

CPI PROSZĘ ZAPAMIĘTAĆ

sobota, 22 czerwca 2024 12:31

CPI - liczba cykli na instrukcje (cycles per instruction).
IPC (odwrotność) - ilość instrukcji na cykl

CPI (*ang. cycles per instruction*), zależy od:

- Organizacji procesora (struktury logicznej)
 - Typowy zakres: 0.1 do 10 na rdzeń
- Zależności pomiędzy instrukcjami sąsiadującymi w kodzie programu
- Czasów dostępu do pamięci:
 - Wzorców dostępu do pamięci
 - Rozmiaru przetwarzanych danych
 - Pamięci podręcznej
 - Zainstalowanych modułów pamięci
 - Kontrolera pamięci
 - Pamięci wirtualnej
- Migracji procesów

23

Oprócz konstrukcji procesora bardzo duży wpływ na czas wykonania instrukcji ma to jak te instrukcje ze sobą współpracują w kodzie. I od podsystemów pamięci: kilka wykładów.

Średnio z 10% instrukcji w kodzie programu to dostępy do pamięci, a one potrafią trwać dziesiątki setki lub tysiące razy dłużej niż inne instrukcje.

Dlatego to w wielu przypadkach jest czas decydujący o czasie wykonania pojedynczych instrukcji.

W procesorach wielordzeniowych w momencie kiedy proces jest przenoszony na inny rdzeń to okazuje się, że po przeniesieniu na inny rdzeń wracamy do momentu

```
$ taskset -c 1 ./p
961775
964825
513550
502575
502625
502600
530300
502550
502625
502600
```

Przenosimy się na te pierwsze odczyty i tu było łagodnie, bo tylko 2 razy wolniej natomiast jak będziemy mieli pecha to rząd albo 2 wielkości wolniej. To zależy od bardzo wielu czynników. Bardzo trudno zmierzyć bez zaburzeń.

- Zmiana jednego parametru często powoduje pogorszenie (wielu) innych
- Czas wykonania jest sumą ważoną

$$t_{instr} = T_{CPU} \cdot \sum_{i_t} N_{i_t} \cdot C_{i_t}$$

- Znaczne skrócenie czasu wykonania wybranych klas instrukcji może spowodować nieznaczący wzrost szybkości całego programu
- Różne algorytmy wymagają różnych klas instrukcji

24

Żeby poprawić program trzeba zwracać na uwagę na wszystkie elementy w tym również na takie, których nie widać.

Polepszymy jedną rzecz: częstotliwość taktowania, to:

- a) trzeba uprościć instrukcje i musi być ich więcej
- b) instrukcje będą się wykonywały więcej cykli

Czas wykonywania programu jest sumą ważoną,, przyspieszymy jeden fragment programu o małym znaczeniu dla całości i może to mieć żaden wpływ.

Są algorytmy, które potrzebują przede wszystkim:

- a) Wykonywać obliczenia
- b) Przesyłać dane

- Czas wykonania zależy nie tylko od czasu wykonania instrukcji

$$t_{exec} = t_{instr} + t_{mem} + t_{IO} + t_{network} + \dots$$

- Czas wykonania nie musi być sumą czasów poszczególnych etapów

$$t_{exec} \geq \max(t_{instr}, t_{mem}, t_{IO}, t_{network}, \dots)$$

25

t_{mem} - oczekiwania na wykonanie operacji na pamięci

t_{IO} - oczekiwania na operacje wejścia wyjścia, dysku, karcie sieciowej, klawiaturze (jeśli źle napiszemy program)

$t_{network}$ - czasy na przesyłanie danych przez sieć (ping 10 ms czyli wieczność!)

Jak się da to różne operacje się na siebie nakładają.

UWAGA! To nie jest tak, że te czasy muszą się sumować!

W tej chwili jak się da to jak na coś czekamy możemy zrobić coś innego.

Dolnym ograniczeniem jest maksymalny czas z tych wszystkich operacji z tym, że to też nie jest takie proste. Może być tak, że na różnych etapach różne rzeczy są najwolniejsze

OŁTARZYK SAMOUWIELBIENIA

sobota, 22 czerwca 2024 12:51

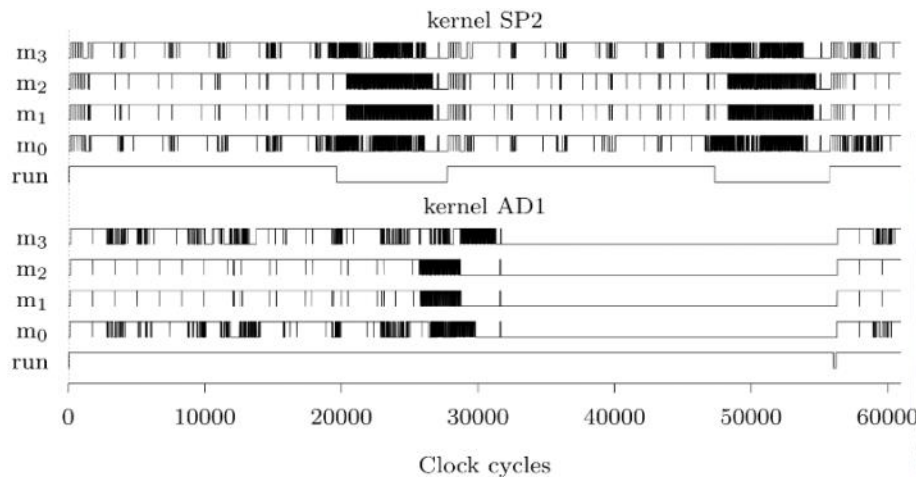


Table 1

The computational complexity of kernels after compiler optimisations. Column “savings” contains the ratio of the number of operations resulting directly from C++ implementation (before compiler optimisations) to the optimised one. Column “data” shows the number of transferred 64-bit words. The values from the last row were computed assuming that each kernel is launched only once per time step.

Kernel	FLOP	Savings	Data	FLOP/B
AD1	54 577	1.88	2064	3.3
D2	26 134	2.25	2469	1.3
P1	14 886	3.15	1227	1.5
SP2	16 441	2.86	1740	1.2
C	3450	4.20	405	1.1
All kernels	115 488	2.34	7905	1.8



26

System, który Tomczak niedawno robił w pewnej jednej firmie. On wykonuje obliczenia na układach FPGA.

Najważniejsze są wykresy na górze dla dwóch różnych funkcji.

Run - wykonywane obliczenia (góra), niewykonywane obliczenia(dół)

Górne wykresy m3 - m0 przesłania danych do poszczególnych elementów układu.

W górnej funkcji czasy przesyłania danych zajmują tak z 30% więcej czasu niż wykonywanie obliczeń chociaż się to na siebie nakłada.

Dolny algorytm szybko przesyła dane, a dwa razy dłużej trwają obliczenia.

Łączny czas wykonania był wypadkową wszystkich czasów.

Jedna funkcja się skończyła jak się zdążyły obliczenia wykonać.

Łączny czas wykonania jakiś tam wychodził.

Na FPGA nie da się zrobić jednak skomplikowanego algorytmu obliczeniowego.

Parametry Maszyny

sobota, 22 czerwca 2024 13:13

Czasem może nas interesować nie czas wykonania, ale przepustowość.

Jak mamy rozwiązać równanie to interesuje nas czas od uruchomienia programu do wygenerowania wyniku.

Jak gramy w grę nie interesuje nas ile ta gra będzie działała tylko ile mamy klatek na sekundę.

Oczywiście ta przepustowość jest związana z czasem przetwarzania jednej klatki.

Jednak patrząc całościowo interesuje nas ta liczba generowanych klatek na sekundę.

Jeżeli chodzi o parametry maszyny przede wszystkim podaje się przepustowość tej maszyny.

- Podawane w kategoriach przepustowości:

$$P_{i_t} = \frac{N_{i_t}}{t_p} = \frac{f_{CPU}}{C_{i_t}} \rightarrow t_p = \frac{N_{i_t}}{P_{i_t}}$$

- Różne parametry
 - Szybkość wykonywania obliczeń
 - Przepustowość pamięci
 - Tracona moc
 - Koszty (różne), np. cena, całkowity koszt posiadania²⁷ (*ang. total cost of ownership, TCO*), rozmiar · moc

Żeby dowiedzieć się ile będą trwały operacje należy podzielić liczbę operacji przez liczbę operacji na sekundę jakie maszyna wykonuje, żeby otrzymać czas wykonania operacji.

Szybkość wykonywania obliczeń - liczba operacji na sekundę

Przepustowość pamięci - liczba przesłanych danych na sekundę

*INNE:

Tracona moc

Koszty (różne)

Dziewczyny z superkomputerem



OMG! Dzięki superkomputerowi
Będę mogła zasymulować umysł
chłopaka, w którym się kocham
i dzięki temu lepiej go poznam
i będę miała więcej szans na podryw.

Świetnie kochana!

Chłopaki z superkomputerem



Spójrz, aerodynamikę homara
można poprawić aż o 25%
ucinając mu jedynie wąsy.

Tak.

- Maksymalna liczba działań arytmetycznych wykonywanych w jednostce czasu
- Zależna od typu danych:
 - IPS (*ang. instructions per second*)
 - FLOPS (*ang. floating-point operations per second*)
- Różne definicje:
 - Szczytowa/maksymalna/teoretyczna moc obliczeniowa (*ang. peak computational power*)
 - Łatwa do określenia
 - Nieosiągalna w praktyce
 - Rzeczywista moc obliczeniowa (*ang. sustained computational power*)
 - Trudna do określenia, zazwyczaj mierzona dedykowanymi programami (*ang. benchmark*)

- Trudna do określenia, zazwyczaj mierzona dedykowanymi programami (*ang. benchmark*)
- Często znacznie niższa od teoretycznej

Będziemy dzielić moc obliczeniową na dwie klasy:

- a) IPS - liczba instrukcji na sekundę dla danych stałoprzecinkowych

Dla mikrokontrolerów podaje się moc obliczeniową w mipsach (milionach instrukcji na sekundę)

- b) FLOPS - liczba instrukcji na sekundę dla danych zmiennoprzecinkowych

Dla urządzeń przeznaczonych do obliczeń zmiennoprzecinkowych: kart graficznych, obecnych procesorów większej mocy

We flopsach (tera na przykład)

Podawane tylko dla maszyn obsługujących sprzętowo działania zmiennoprzecinkowe! One z definicji mają wbudowane jednostki zmiennoprzecinkowe różne, o różnych precyzjach nawet

Tu mówimy o mocy obliczeniowej dla:

- pojedynczej precyzji
- podwójnej precyzji
- potrójnej precyzji
- *połowiczna precyzja (dla sztucznej inteligencji)

W rzeczywistości nie jest tak prosto!

Najłatwiej mówić o szczytowej/maksymalnej mocy obliczeniowej.

Wynika ona z częstotliwości taktowania i liczby cykli na operację, nie da się jednak napisać takiego programu, żeby procesor od początku do końca z pełną mocą wszystko liczył dlatego wprowadza się parametr jak

rzeczywista moc obliczeniowa - ją się w zasadzie tylko mierzy specjalnymi kodami (benchmark). Mierzy się przede wszystkim operacje podstawowe algebraiczne. Od kopiowania wektorów do operacji na macierzach np. Mnożenie macierzy przez macierz, normalizacja kwadratowa (generalnie suma kwadratów spierwiastkowana).

Do tego dzisiaj się używa komputerów, do operacji na macierzach i wektorach, na dużych zbiorach danych, na których wielokrotnie wykonuje się praktycznie to samo.

Przykłady:

- Hewlett Packard Enterprise Frontier (OLCF-5):
 - ok. 2 (4) EFLOPS (FP64) maksymalnej mocy obliczeniowej
 - 1.1 EFLOPS (FP64) rzeczywistej mocy obliczeniowej (HPLinpack)
- Tesla H100:
 - 51 TFLOPS (FP32), 26 TFLOPS (FP64)
 - 205 TFLOPS (FP16)
- GTX 4090:
 - 83 TFLOPS (FP32), 1.3 TFLOPS (FP64)
 - 83 TFLOPS (FP16)
- AMD Ryzen Threadripper PRO 7995WX:
 - 11.5 TFLOPS (FP32), 5.8 TFLOPS (FP64)
- Ryzen 9 7945HX:
 - 3.8 TFLOPS (FP32), 1.9 TFLOPS (FP64)

30

Hewlett Packard Enterprise Frontier - najmocniejszy komputer na świecie.

Tutaj już są podane same rzeczywiste:

FP16 - połowiczna precyzja

FP32 - pojedyncza precyzja

FP64 - podwójna precyzja

Tesla H100 - akcelerator do obliczeń (karta graficzna bez wyjścia na monitor)

GTX 4090 - karta do gier, w OpenGL i w DirectX grafika jest w zasadzie na pojedynczej precyzji liczona

AMD Ryzen Threadripper PRO 7995WX - najmocniejszy procesor serwerowy

Ryzen 9 7945HX - dość mocny procesor laptopowy

Cały czas mówimy o wielkości milion milionów operacji na sekundę.

Ciekawostka, meteorolodzy mówią, że jak będą mieć system ZetaFlopowy () to będą mogli przewidzieć pogodę na dwa tygodnie. ;)

Taki żart bo jesteśmy już blisko końca.

Zrównoważenie maszyny

- Operacje (arytmetyczne) wymagają argumentów
- W wielu przypadkach szybkość przetwarzania jest ograniczona szybkością dostarczania danych
- Zrównoważenie maszyny (*ang. machine balance*): stosunek mocy obliczeniowej do przepustowości pamięci

31

Będziemy mogli wykonywać operacje jeśli nadążymy z dostarczaniem danych.

- I. Wprowadzamy nowy parametr, zrównoważenie maszyny (stosunek mocy obliczeniowej do przepustowości pamięci):

FLOP/B - Operacje na przesłany bajt:

Jeśli dostępną moc obliczeniową podzielimy przez przepustowość pamięci czyli rozmiar danych przesyłanych w ciągu sekundy :

- Hewlett Packard Enterprise Frontier (OLCF-5):
 - ok. 30 FLOP/B (FP64)
 - ok. 0.15 EB/s
- Tesla H100:
 - 25 FLOP/B (FP32), 13 FLOP/B (FP64)
 - 2 TB/s
- GTX 4090:
 - 83 FLOP/B (FP32), 1.3 FLOP/B (FP64)
 - 1 TB/s
- AMD Ryzen Threadripper PRO 7995WX:
 - 35 FLOP/B (FP32), 18 FLOP/B (FP64)
 - 0.33 TB/s
- Ryzen 9 7945HX:
 - 46 FLOP/B (FP32), 23 FLOP/B (FP64)
 - 0.083 TB/s

To dla normalnych maszyn wychodzi mniej więcej to samo!

Dla Tesli: $51 \text{ TFLOP/S} / 2\text{TB} = 25 \text{ FLOP/B}$

- Hewlett Packard Enterprise Frontier (OLCF-5):
 - ok. 2 (4) EFLOPS (FP64) maksymalnej mocy obliczeniowej
 - 1.1 EFLOPS (FP64) rzeczywistej mocy obliczeniowej (HPLinpack)
- Tesla H100:
 - 51 TFLOPS (FP32), 26 TFLOPS (FP64)
 - 205 TFLOPS (FP16)
- GTX 4090:
 - 83 TFLOPS (FP32), 1.3 TFLOPS (FP64)
 - 83 TFLOPS (FP16)
- AMD Ryzen Threadripper PRO 7995WX:
 - 11.5 TFLOPS (FP32), 5.8 TFLOPS (FP64)
- Ryzen 9 7945HX:
 - 3.8 TFLOPS (FP32), 1.9 TFLOPS (FP64)

30

Każdy algorytm wymaga określonej ilości działań i przestania określonej ilości pamięci, tę wielkość nazywamy intensywnością algorytmu.

Intensywność arytmetyczna

- Gęstość arytmetyczna, intensywność operacyjna
(*ang. arithmetic/operational/computational intensity*)

$$I = \frac{N_A}{S_{MEM}}$$

N_A - liczba wykonanych operacji (arytmetycznych)

S_{MEM} - rozmiar przesłanych danych
(*ang. memory traffic*)

33

Algorytm poglądowy, krótka pętla:

```

1 void vadd (double * vdst, const double * vsrc, unsigned long long N)
2 {
3     for (unsigned long long i=0 ; i < N ; i++)
4     {
5         vdst [i] = vdst [i] + vsrc [i] ;
6     }
7 }

```

$$I = \frac{1}{(2+1) \cdot 8} = \frac{1}{24} \text{ FLOP/B}$$

$$I = 0.0417 \text{ FLOP/B}$$

```

19 .L3:
20     fldl    (%eax)
21     faddl   (%edx)
22     addl    $8, %eax
23     addl    $8, %edx
24     fstpl   -8(%eax)
25     cmpl    %ecx, %eax
26     jne     .L3

```

- Odczyt,
- odczyt z dodawaniem,
- przesunięcie wskaźników,
- zapis wyniku (**Store Floating Point and Pop Long**), używana w asemblerze x86, służy do zapisywania wartości zmiennoprzecinkowej z górnej części stosu zmiennoprzecinkowego (ang. **Floating Point Unit**, FPU) do określonego miejsca w pamięci oraz usunięcia tej wartości ze stosu.
- obsługa pętli

Sumarycznie: dwa odczyty, jeden zapis, jedna operacja arytmetyczna.

Jedna operacja arytmetyczna/ dwa odczyty + 1 zapis * 8 bajtów, bo na typie double

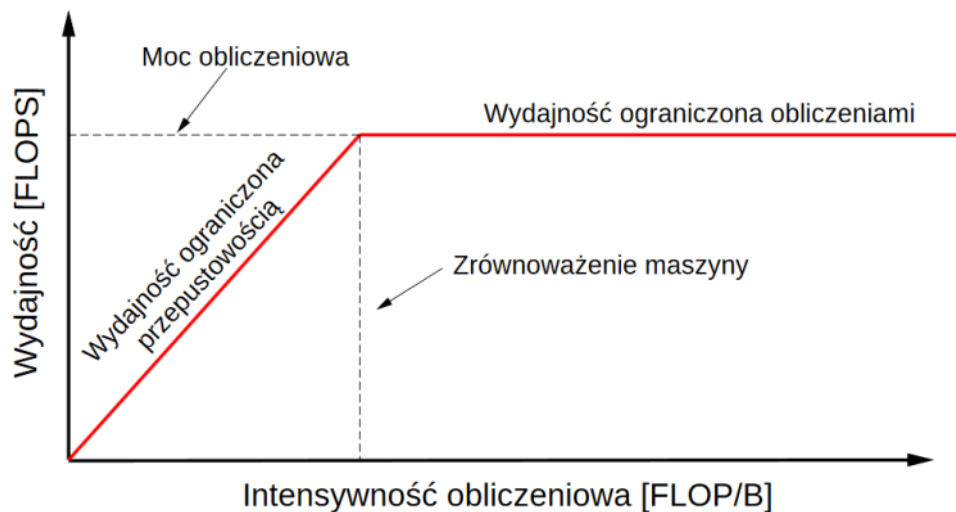
Ten algorytm ma intensywność arytmetyczną 1/24 FLOP/B czyli 0.0417

Dowolna z tych maszyn ma w dziesiątkach flopów na bajt zrównoważenie maszyny:

- Hewlett Packard Enterprise Frontier (OLCF-5):
 - ok. 30 FLOP/B (FP64)
 - ok. 0.15 EB/s
- Tesla H100:
 - 25 FLOP/B (FP32), 13 FLOP/B (FP64)
 - 2 TB/s
- GTX 4090:
 - 83 FLOP/B (FP32), 1.3 FLOP/B (FP64)
 - 1 TB/s
- AMD Ryzen Threadripper PRO 7995WX:
 - 35 FLOP/B (FP32), 18 FLOP/B (FP64)
 - 0.33 TB/s
- Ryzen 9 7945HX:
 - 46 FLOP/B (FP32), 23 FLOP/B (FP64)
 - 0.083 TB/s

Na każdy przesłany bajt potrafi wykonać 30 operacji zmiennoprzecinkowych, w tym algorytmie potrzebujemy przesłać tylko 1/12 operacji zmiennoprzecinkowych. Tutaj co 24 przesłane bajty wykonujemy jedną operację zmiennoprzecinkową.

Model „sufitowy” (ang. roofline)

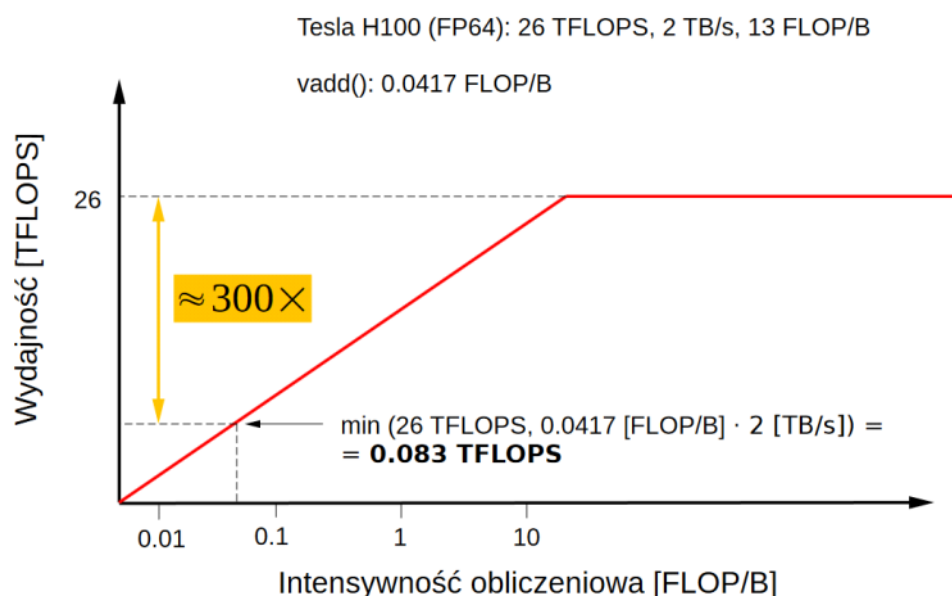


Okazuje się, że jeśli intensywność obliczeniowa będzie równa dokładnie zrównoważeniu maszyny to będzie wykonywał maksymalną ilość operacji zmiennoprzecinkowych zgodnie z maksymalną mocą obliczeniową.

Przed tym zrównoważeniem będzie ograniczony przepustowością, a po nim ograniczony maksymalną mocą obliczeniową.

Jeśli nasz algorytm wykonywałby $30 \cdot 24 = 720\,720$ operacji zmiennoprzecinkowych na każde 3 przesłane to by całkowicie wykorzystywał moc obliczeniową.

Przykład dla naszej tesli H100 (uwaga! To jest w skali logarytmicznej żeby było lepiej widać i żeby można było skalę lepiej oddać):



36

Nasz algorytm ma intensywność obliczeniową 4/100 flopa na bajt.

Obliczamy ile dla maksymalnej przepustowości (2TB/s) wykonamy działań zmiennoprzecinkowych. Na każdego przesłanego bajta wykonujemy 4/100 FLOP-a. Maksymalna wydajność algorytmu rozumiana jako przepustowość to 0,083GFLOPS-y, 84 tysięczne TFLOPS-a dla 26TFLOPS-ów maksymalnej mocy obliczeniowej!!!.

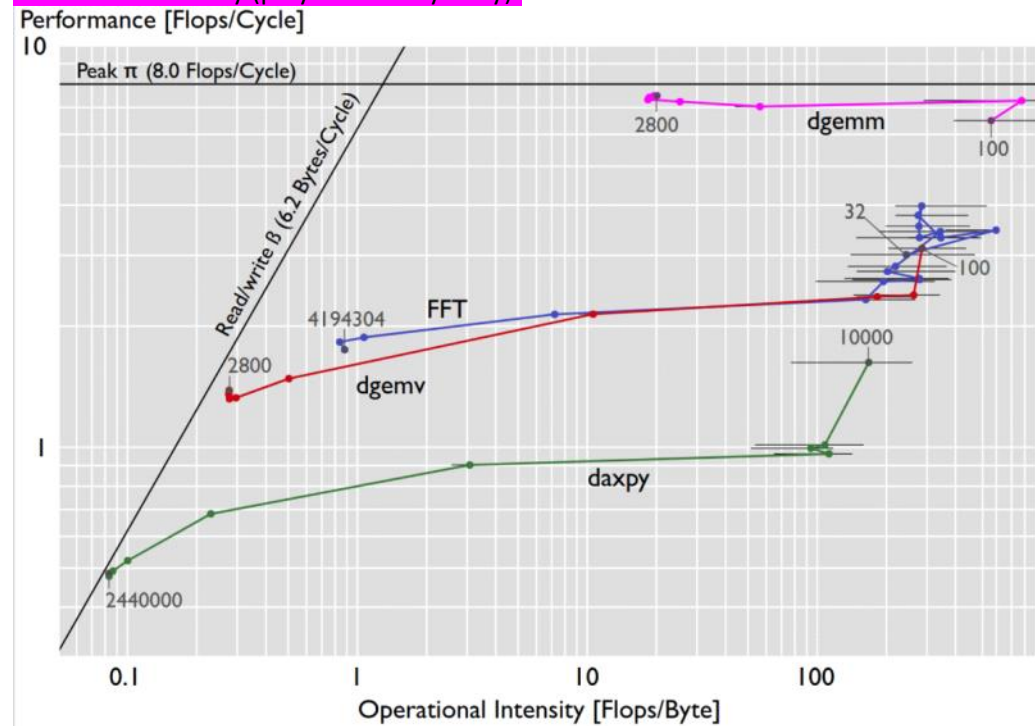
1/300 potencjału obliczeniowego tej karty graficznej.

To nie oznacza, że możemy pójść wyżej. To oznacza, że kupiliśmy za drogą maszynę!!!!

Ta przepustowość jest maksymalna teoretyczna, wyżej się nie da!!!!

Niektóre algorytmy są ograniczone mocą obliczeniową ale większość przepustowością pamięciową.

Mnożenie macierzy (przykład rzeczywisty):



Im większe dane tym algorytm działa wolniej, spada jego moc obliczeniowa.

Jeśli algorytm jest bardziej skomplikowany, ma wyższą intensywność obliczeniową, ale nigdy nie przekroczymy tej ukośnej linii.

Algorytmy, które faktycznie są ograniczone mocą obliczeniową a nie przepustowością do operacje na macierzach gęstych wymagające mnożenia tych macierzy przez siebie.

Dlaczego producenci skupiają się na mocy obliczeniowej, a nie przepustowości pamięci?

Bo taniej i łatwiej.

Tranzystor nie kosztuje nic.

Natomiast każda nóżka w układzie scalonym, każdy MHz na magistrali, każde zakłócenie, prąd, to wszystko kosztuje.

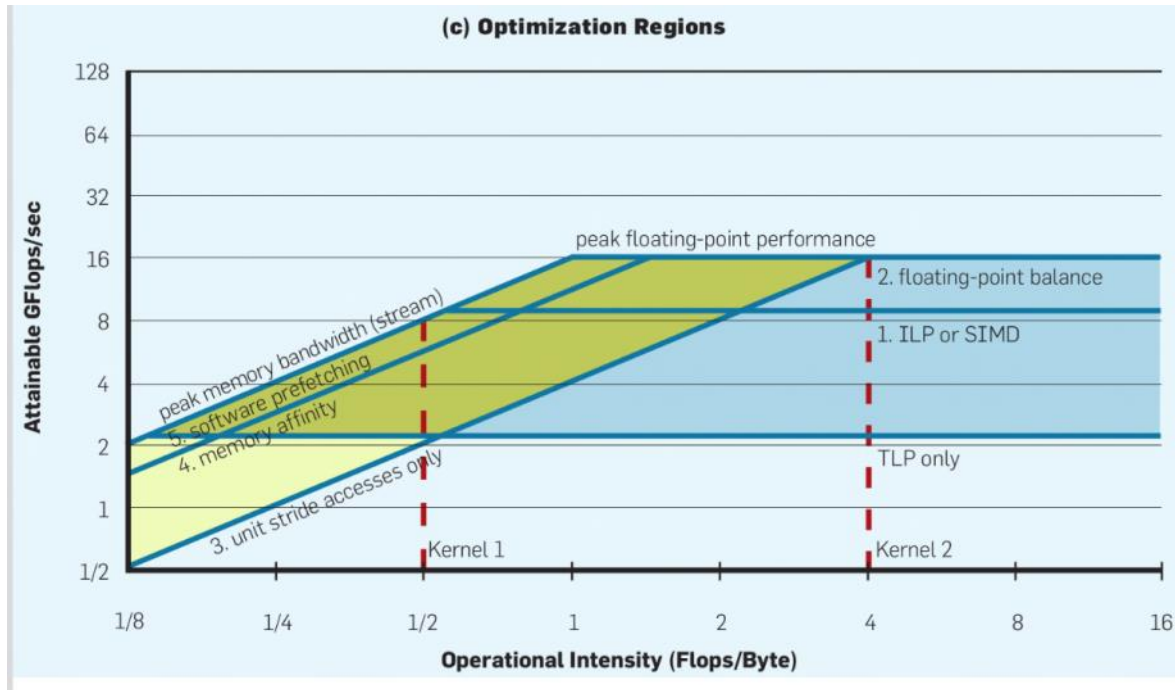
Ale przepustowość też rośnie.

Jeszcze trudniej niż przepustowość jest zwiększyć czas dostępu!!! W czasie oczekiwania na dane możemy jednak zrobić coś innego.

NIE DLA JUNIORA

sobota, 22 czerwca 2024 14:43

Niekoniecznie musi być tak prosto, że mamy jedno ograniczenie przepustowości i jedno mocy obliczeniowej.



Możemy wprowadzać różne ograniczenia wynikające z różnych sposobów pisania kodu.

"Jeśli Państwo piszecie kod zwykły sekwencyjny, używając najprostszych instrukcji uzyskacie znacznie niższe maksymalne moce obliczeniowe niż kod, który jest napisany z wykorzystaniem dedykowanych układów, które potrafią wykonywać wiele operacji naraz dla danych specjalnie ułożonych. Tylko trzeba mieć te dane specjalnie ułożone i umieć dopasować te działania, które my chcemy wykonać do działań, które udostępnia maszyny. A to skala logarytmiczna więc nawet jeśli widać niewielką przerwę to te zyski są wielokrotne, kilkukrotne"

Chemik Fizyk pisze kod, ktoś kto zna się na rzeczy w stosunkowo krótkim czasie czyni go 10,100 razy szybszym.

DZIĘKUJĘ BARDZO!