Mateusz Denisiuk, nr indeksu 241273

Wrocław, dn. 13 czerwca 2019

CZ TN 13:15

Dokumentacja projektu

„Organizacja i Architektura Komputerów”

Temat 15: „Analiza struktury pamięci DDR na podstawie czasów dostępu”

PROWADZĄCY:

Dr inż. Tadeusz Tomczak

Spis treści

1 Założenia i cel projektu

Zgodnie z założeniami celem projektu jest wskazanie za pomocą mierzenia czasów dostępu pewnych cech struktury pamięci RAM w technologi DDR (Double Data Rate).

Zgodnie z ustaleniami poczynionymi w trakcie zajęć projektowych moja koncepcja sprawdzenia tych cech oparta jest o próbę wykazania regularnych wzrostów czasu dostępu w ściśle określonych warunkach.

Badania zostały przeprowadzone na platformie *Linux* zainstalowanej na komputerze wyposażonym w następujące moduły pamięci DDR4:

- CORSAIR CMSO8GX4M1A2133C15
  - gęstość 8 GB
  - częstotliwość zegara 1066 MHz
  - 1 rank
  - chipy DRAM x8
- Kingston KHX2400C14S4/4G
  - gęstość 4 GB
  - częstotliwość zegara 1200.5 MHz
  - 1 rank
  - chipy DRAM x8

Oba te moduły zgodnie ze specyfikacjami *JEDEC* odnośnie technologi DDR4 SDRAM posiadają 16 banków wewnętrznych podzielonych na 4 równe grupy operujące niezależnie.[1]

Repozytorium z programem znajduje się pod adresem:
https://github.com/mdenisiuk98/OiAK-Projekt-DDR/tree/master

2 Analiza teoretyczna działania pamięci DDR

Wykonane przeze mnie badania oparte są o przypuszczenia wynikające z zebranych wcześniej informacji na temat struktury i operacji zachodzących w modułach pamięci DDR4.

2.1 Podział na jednostki logiczne

Hierarchia jednostek logicznych pojedynczego modułu pamięci DDR przedstawia się następująco:

1. Rank
2. Bank
3. Rząd
4. Kolumna

Rank jest to zestaw chipów DRAM podłączonych do jednego zestawu szyn adresu i danych, natomiast bank jest podjednostką pojedynczego chipu DRAM. Każdy bank składa się z zestawu tzw. tablic, które zawierają pojedyncze komórki pamięci ustawione w rzędy i kolumny. Przykładowo w przypadku operacji odczytu każda tablica w banku dostarcza jeden bit danych adresowany poprzez ten sam rząd i kolumnę.

2.2 Grupowanie banków w technologii DDR4

Technologia DDR4 wprowadza dodatkowy podział pamięci na grupy banków.[1] Ponieważ oddzielne grupy banków wyposażone są w osobne kontrolery, komendy dla nich mogą być wydawane z mniejszym opóźnieniem niż komendy do różnych banków w poprzednich iteracjach DDR, co pozwala na znaczny wzrost prędkości transmisji danych. W przypadku wielokrotnego dostępu do tej samej grupy nałożone są jednak dodatkowe ograniczenia czasowe. Jest to jedno ze zjawisk, które staram się przetestować w tym projekcie.

2.3 Operacje dostępu

Poniższy diagram stanów przedstawia możliwe stany banku pamięci DDR4.

Rysunek 1: Diagram stanów pamięci DDR z dokumentacji modułów firmy Micron

Wszystkie pokazane na nim przejścia pomiędzy stanami powiązane są z różnymi ograniczeniami czasowymi. Na chwilę obecną skupię się jednak na procesie wykonania odczytu z pamięci. Jest to związane z faktem, że badanie odczytu pozwala na uzyskanie najbardziej

miarodajnych wyników. W przypadku wykonania zapisu dane przekazywane są do odpowiednich kontrolerów i zgłaszany jest koniec operacji przez co czasy zawsze będą mniej więcej takie same niezależnie od czynników takich jak adres, pod którym chcemy wykonać zapis.

Schemat operacji odczytu może wyglądać różnie w zależności od stanu początkowego banku, dla którego jest wykonywany. Odróżniam tutaj trzy możliwości ze stopniowo rosnącym czasem dostępu:

1. Bank jest aktywny oraz wykonywany jest odczyt z aktywnego rzędu.

2. Bank jest nieaktywny, więc należy wykonać komendę ACTIVATE.

3. Bank jest aktywny, lecz chcemy odczytać inny rząd. Należy zamknąć bank komendą PRECHARGE oraz ponownie go aktywować podając wymagany do odczytu rząd.

Innym czynnikiem, który postaram się zademonstrować eksperymentalnie będzie wpływ grup banków na odczyty sekwencyjne.

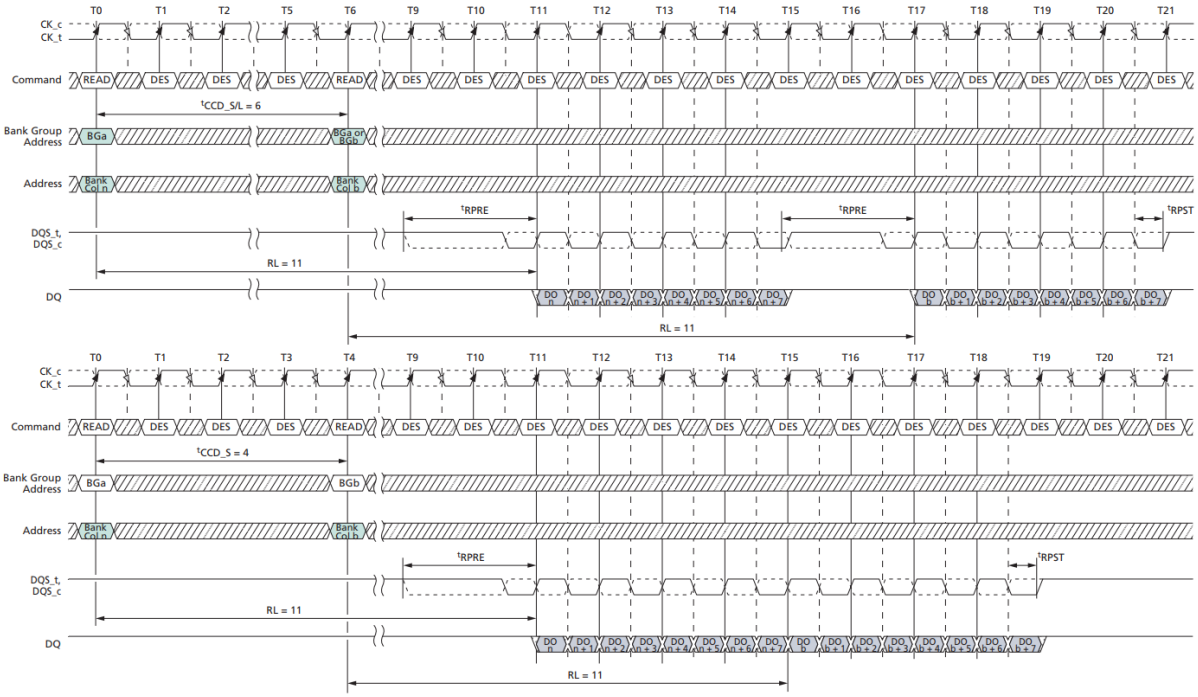
Przewiduję, że w przypadku badanego przeze mnie odczytu par adresów czas zamknięcia banków nie będzie wpływał na wyniki pomiarów, ponieważ program będzie przechodził dalej bezpośrednio po otrzymaniu wszystkich danych. To czy zamknięcie nastąpi zależeć będzie natomiast od wykonywanych poza sekwencją pomiarową dostępów. Jeśli będą one wykonywane do tych samych banków pozostaną one aktywne. W przeciwnym przypadku kontroler pamięci automatycznie wyda komendę PRECHARGE.

Ostatnią istotną częścią operacji odczytu jest działanie architektury 8n prefetch. W wyniku jej zastosowania przy każdym dostępie do pamięci tak naprawdę wykonywane są dwie operacje odczytu w innych grupach banków. W efekcie do pamięci podręcznej dostarczane jest nie 8, a 16 słów maszynowych. Jest to ilość zgodna z rozmiarów szyny pamięci podręcznej, która wynosi 64 bajty czyli 16 8-bajtowych słów.

2.4 Przewidywane czasy dostępu

W wymienionych powyżej przypadkach odróżnić będzie można warianty dostępów do różnych grup banków oraz tej samej grupy banków. Każdy z nich związany będzie z innymi ograniczeniami czasowymi zgodnymi ze standardem dla tego rodzaju pamięci DDR.

W przypadku,gdy oba banki w parze, dla której chcemy wykonać test mają już aktywne odpowiednie rzędy musimy wziąć pod uwagę trzy specyfikacje czasowe: RL (Read Latency), ${}^{t}CCD\_S$ (Short) oraz ${}^{t}CCD\_L$ (Long). Pierwszy z nich oznacza czas pomiędzy wydaniem polecenia i początkiem udostępniania danych do odczytu. Czasy ${}^{t}CCD$ to opóźnienia pomiędzy wydaniem komend odczytu dla dwóch banków. Jeśli znajdują się one w różnych grupach wynosi ono ${}^{t}CCD\_S$, a w przeciwnym wypadku ${}^{t}CCD\_L$. Różnica ta zademonstrowana jest na poniższym wykresie czasowym pochodzącym ze specyfikacji Micron DDR4 SDRAM.

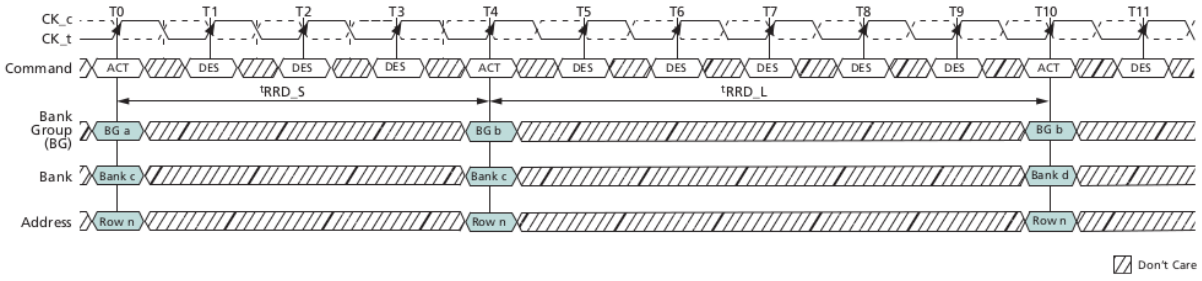


Rysunek 2: Porównanie czasów odczytu z tej samej i różnych grup banków

Z analizy wykresu wynika następujący zestaw wniosków:

- RL = 11 CK (cykli zegara)
- ${}^{t}CCD\_S = 4$ CK
- ${}^{t}CCD\_L = 6$ CK (dopuszczalna jest również wartość 5 CK)
- przy następujących po sobie dostępach do tej samej grupy banków występuje przerwa w dostawie danych równa różnicy pomiędzy opóźnieniami ${}^{t}CCD$

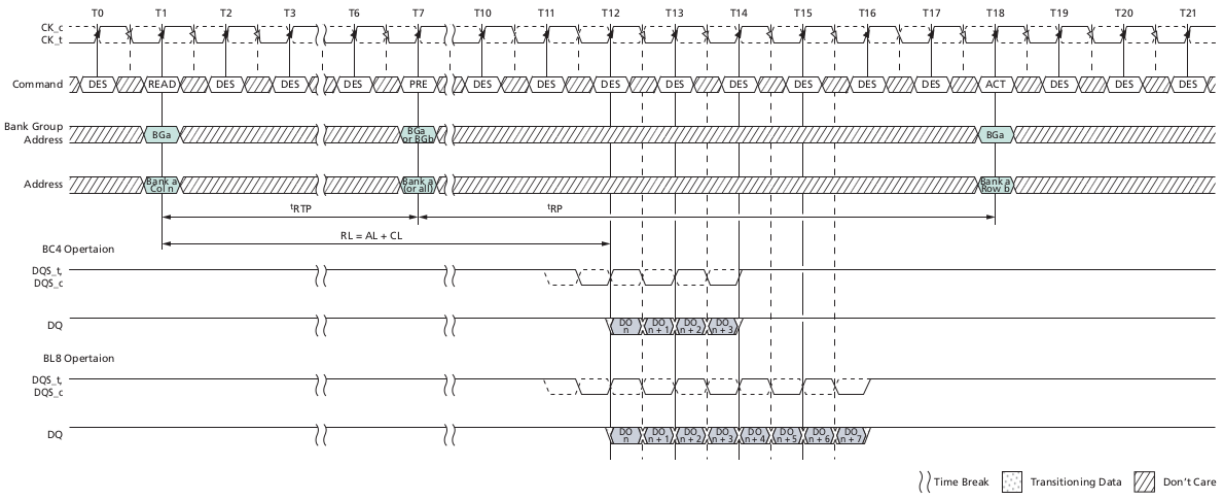
W przypadku aktywacji banków występuje taka sama różnica pomiędzy grupami banków jak dla odczytu. Opóźnienia pomiędzy komendami aktywacji są oznaczone symbolami ${}^{t}RRD\_S$ oraz ${}^{t}RRD\_L$, a różnica między nimi przedstawiona jest poniżej:



Rysunek 3: Porównanie opóźnień między komendami aktywacji

Dodatkowo musimy się liczyć z odstępem pomiędzy komendami aktywacji i odczytu ${}^{t}RCD = 16$ CK.

Ostanie ograniczenia czasowe spotykamy w przypadku, gdy wymagane jest zamknięcie banku i aktywacja innego rzędu. Po wydaniu komendy zamknięcia (PRECHARGE) musi upłynąć przynajmniej czas ${}^tRP$, co pokazano na wykresie czasowym poniżej:



Rysunek 4: Demonstracja czasu zamknięcia banku i ponownej jego aktywacji

Jak można odczytać obserwując cykle zegara od T7 do T18 ${}^tRP$ wynosi 11 CK. Jeśli jeden lub oba z dostępów w testowanej parze będzie wymagał takiej operacji musimy się liczyć z tym opóźnieniem. Pozostanie jednak ono takie same niezależnie czy będzie trzeba je wykonać dwa razy czy raz.

Dodatkowo należy pamiętać, że za każdym odczytem wykonywane są tak naprawdę dostępy do dwóch banków z różnych grup zgodnie z założeniami architektury 8n prefetch. Należy więc do każdego wyliczonego czasu dostępu dodać 8 cykli zegarowych w trakcie których przesyłane jest 16 słów maszynowych.

Zbierając wszystkie powyższe ograniczenia czasowe można wyliczyć zakładane czasy dostępów dla wymienionych wcześniej przypadków.

1. Oba banki aktywne bez konfliktu rzędów:

   - różne grupy - ${}^tCCD\_S$ + RL + 8 (czas dostarczenia danych) = 23
   - ta sama grupa - ${}^tCCD\_L$ + RL + 8 = 25 lub 24

2. Ówczesna aktywacja banku:

   - różne grupy - ${}^tRCD + {}^tCCD\_S + \mathrm{RL} + 8 = 39$
   - ta sama grupa - ${}^tCCD\_L + \mathrm{RL} + 8 = 41$ lub 40

3. Zamknięcie i aktywacja banku:

   - różne grupy - ${}^tRP + {}^tRCD + {}^tCCD\_S + \mathrm{RL} + 8 = 50$
   - ta sama grupa - ${}^tRP + {}^tCCD\_L + \mathrm{RL} + 8 = 52$ lub 51

Wszystkie z tych wyników należy również przeskalować. Wynika to z różnicy taktowań zegarów pamięci i procesora. W trakcie testów zegar procesora pracuje ze stałą częstotliwością 2800 MHz, a zegar pamięci z częstotliwością 1066 MHz. Przewidywane czasy przemnażamy więc przez współczynnik $\frac{2800}{1066} \approx 2,63$. Dodatkowo należy dodać pewien nadmiar wynikający z pomiarów przy użyciu rozkazu `rdtsc` języka *Assembly* oraz działania odpowiednich kontrolerów na procesorze (przykładowo tłumaczenie adresów wirtualnych na fizyczne). Pominę go jednak dla ostatecznych wyliczeń, gdyż powinien być praktycznie jednakowy dla każdego pomiaru.

Ostatecznie otrzymujemy przewidywane czasy dostępów:

1. Oba banki aktywne bez konfliktu rzędów:

   - różne grupy - 60
   - ta sama grupa 66 lub 63

2. Ówczesna aktywacja banku:

   - różne grupy - 102
   - ta sama grupa - 108 lub 105

3. Zamknięcie i aktywacja banku:

   - różne grupy - 131
   - ta sama grupa - 137 lub 134

3 Metoda pomiarowa

Przyjęta przeze mnie koncepcja na zebranie pomiarów czasu dostępu w swojej ostatecznej formie jest zgodna z następującym zestawem założeń:

- program służący do zebrania pomiarów został napisany w języku `C++` z elementami *Assembly*, skompilowany kompilatorem *g++* oraz uruchomiony w środowisku *Linux* w architekturze *x86_64*
- pomiary przyjmują 5 parametrów:
  - pierwszy adres porównywanej pary
  - adres początkowy przedziału porównań
  - adres końcowy przedziału porównań
  - krok z jakim wykonywane są porównania
  - ilość testów dla pojedynczej pary
- adresy przyjmowane są w celu ułatwienia jako pary indeksów tablicy dwuwymiarowej
- pomiary wykonywane są dla każdej pary, z której pierwszy element pozostaje stały, a drugi jest wybierany z podanego zakresu z zadanym krokiem
- pomiary dla każdej pary wykonywane są zadaną ilość razy z uprzednim wyczyszczeniem pamięci podręcznej procesora (cache)
- wszystkie testowane adresy należą do i podawane są jako przesunięcie względem początku wcześniej alokowanego bloku pamięci

Ponadto udostępnione są dwa tryby testowania:

- czas pomiaru zbierany jest dla obu odczytów
- odczyt pierwszej zmiennej z pamięci wykonywany jest przed rozpoczęciem pomiaru czasu

3.1 Alokacja pamięci pod testy

Pamięć, do której wykonuję dostępy w testach alokowana jest statycznie w postaci tablicy dwuwymiarowej zmiennych typu `long long int` o rozmiarze 12582912x16. Wybrany przeze mnie typ jest zgodny rozmiarem z rozmiarem jednego słowa maszynowego w mojej pamięci DDR - 8 bajtów. Łącznie alokuję więc ok. 2 GB pamięci. Tak duży rozmiar jest jedynym sposobem w jaki mogę spróbować spowodować, że zmienne będą ułożone w sposób ciągły. Spodziewam się, że jakiś początkowy odsetek tablicy zostanie umieszczony fragmentarycznie w otwartych już stronach w pamięci, a adresy dalsze będą zajmować całe nowe strony.

3.2 Zapewnienie chybień do pamięci podręcznej

Czynnikiem możliwie wpływającym na na dokładność pomiarów jest stosowana w procesorach pamięć podręczna. Przy każdym dostępie do pamięci RAM odczytane z niej dane umieszczane są w pamięci podręcznej o znacznie niższych czasach dostępu i znacznie niższej pojemności (w moim przypadku niewiele ponad 6 MB). Aby tego uniknąć zaimplementowałem w moim programie funkcję `cacheFlush()` służącą do zapełnienia pamięci podręcznej innymi danymi. Zasada jej działania opiera się o alokację odpowiednio dużej tablicy dwuwymiarowej (96000x600 elementów typu `int`) i wykonywanie dostępów do niej w sposób wymuszający dodanie za każdym razem 64 bajtów danych do pamięci podręcznej. Dostępy te polegają na dwóch pętlach, z których zewnętrzna przechodzi po kolumnach, a wewnętrzna po wierszach. Przy odpowiedniej ilości takich dostępów wcześniej przechowywane przez procesor zmienne zostają zastąpione innymi.

3.3 Mierzenie czasu dostępu

Sekcja programu odpowiadająca za pomiary oparta jest o utworzenie wskaźników na testowaną parę adresów i przekazanie ich do sekcji kodu napisanej w języku *Assembly*, gdzie przy użyciu rozkazu `rdtsc` odczytywany jest rejestr powiązany z zegarem monotonicznym procesora `TSC`[3]. Rejestr ten przechowuje ilość cykli zegara monotonicznego, która upłynęła od czasu uruchomienia komputera. W wyniku tego wykonanie rozkazu `rdtsc` na początku i na końcu testowanej sekcji kodu oraz obliczenie różnicy pomiędzy zwróconymi wartościami pozwala na dokładny odczyt ile cykli zegara upłynęło podczas wykonania mierzonego fragmentu kodu. W programie korzystam z nieco zmodyfikowanej wersji tego rozkazu - `rdtscp`. Wariant ten zapewnia, że odczyt `TSC` nastąpi dopiero w momencie, w którym wykonają się wszystkie wcześniejsze instrukcje.

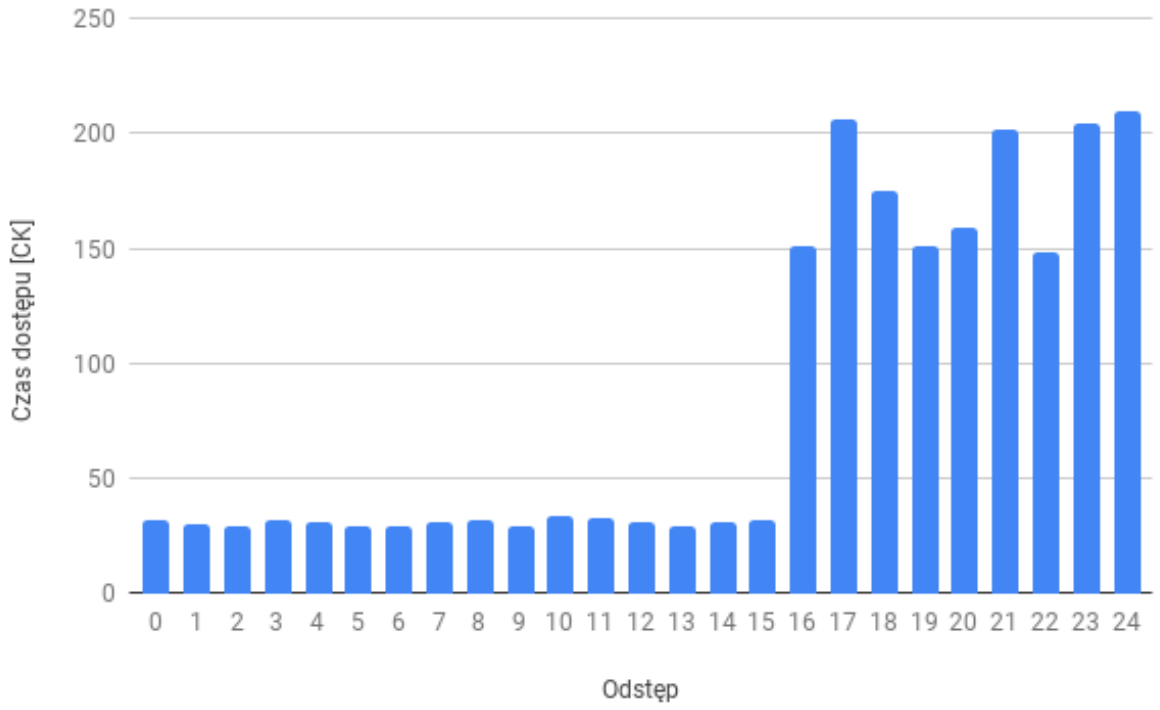
Fragment kodu *Assembly* został również opatrzony słowem kluczowym języka *C/C++* `volatile`. Jego celem jest zasygnalizowanie kompilatorowi, że poniższy fragment kodu assemblerowego nie może być usunięty lub zmodyfikowany, nawet jeśli pozornie jest nieoptymalny lub nie wpływa na resztę programu.[4] W efekcie upewniam się, że wszystkie rozkazy zostaną zachowane w stanie niezmienionym. Korzystam również z tak zwanych `clobberów` rejestrów oraz pamięci. W przypadku rejestrów oznacza to, że są one modyfikowane i kompilator nie może przyjąć, że znajdą się w nich oczekiwane wartości. W odniesieniu do pamięci `clobber` powoduje, że zapisy i odczyty pamięci nie są optymalizowane.[5] Korzystam z tego, ponieważ chcę osiągnąć wyniki z jak najmniejszą ingerencją.

3.4 Sposób zbierania wyników

Po zebraniu całej serii pomiarów dla jednej pary adresów zliczana jest średnia oraz wybierana jest najniższa wartość. Obie te dane wypisywane są wraz z odpowiadającym im przesunięciem względem adresu początkowego do pliku tekstowego. Zbieram również wyniki najniższe, ponieważ uważam, że najdokładniejszym wynik w danej serii jest ten najniższy, ponieważ przyczyną niedokładności jest wykonywanie przez procesor innych zadań w trakcie pomiaru i inne opóźnienia, co skutkować może jedynie zawyżeniem wyniku.

4 Analiza wyników pomiarów

Pierwszą odkrytą na podstawie wykonanych pomiarów cechą pamięci DDR jest działanie architektury prefetch. Zademonstruję je na przykładzie poniższego zestawu pomiarów:

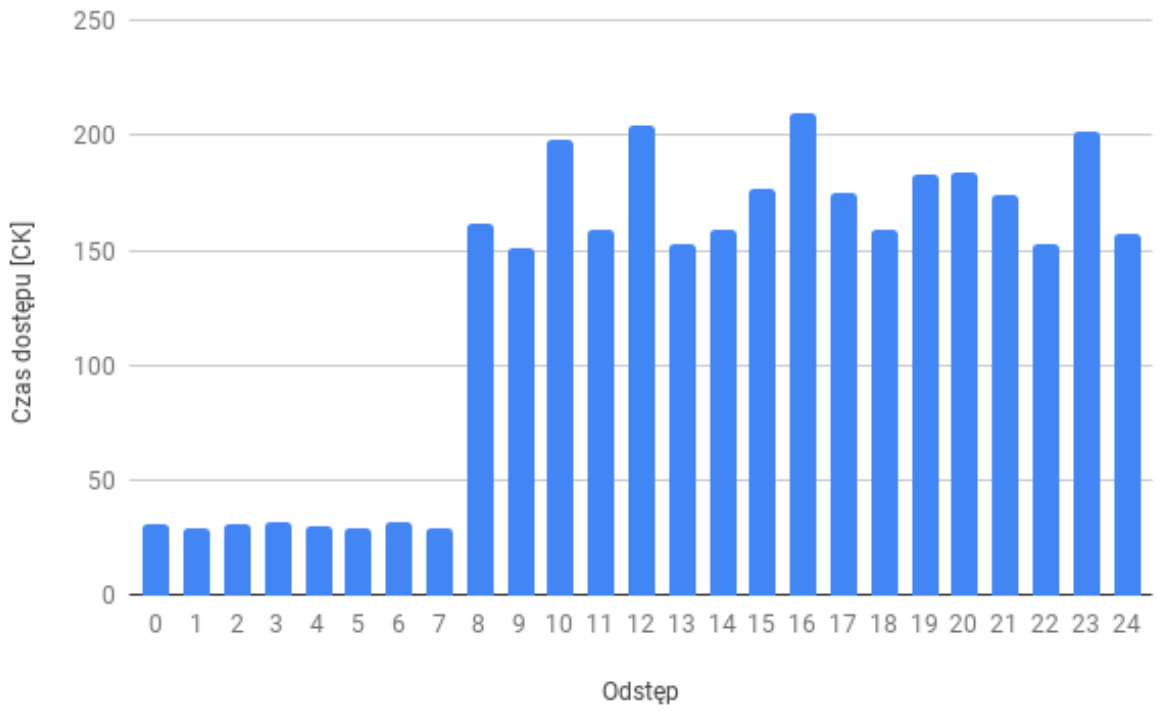


Rysunek 5: Demonstracja czasów dostępu z krokiem 1 zaczynając od elementu [a][0]

Jak widać początkowa część wyników jest znacznie niższa niż reszta (oscyluje w okolicach 32 cykli zegara). Taka różnica wynika z prostego faktu, że po pierwszym dostępie wartość, którą próbujemy odczytać przy drugim znajduje się już w pamięci podręcznej. Jak zaprezentowano powyżej dzieje się tak dla szesnastu wartości zaczynając od adresu początkowego w danej parze. Wiedząc, że pojedynczy odczyt wydobywa osiem słów z banku, możemy więc wnioskować, że każda operacja odczytu wydobywa zestaw po osiem słów z dwóch banków z różnych grup.

Ważną obserwacją w tej kwestii jest również fakt, że każdy impuls odczytowy wydobywa 16 słów z wiersza, w którym znajduje się odczytywany adres pamięci. Ma to miejsce

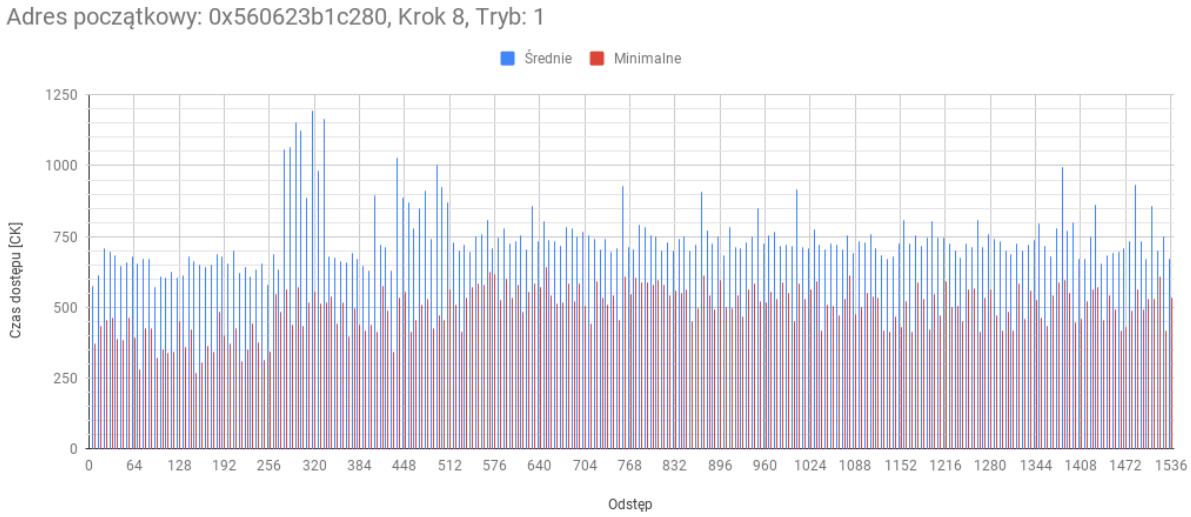
niezależnie od kolumny, którą odczytujemy.



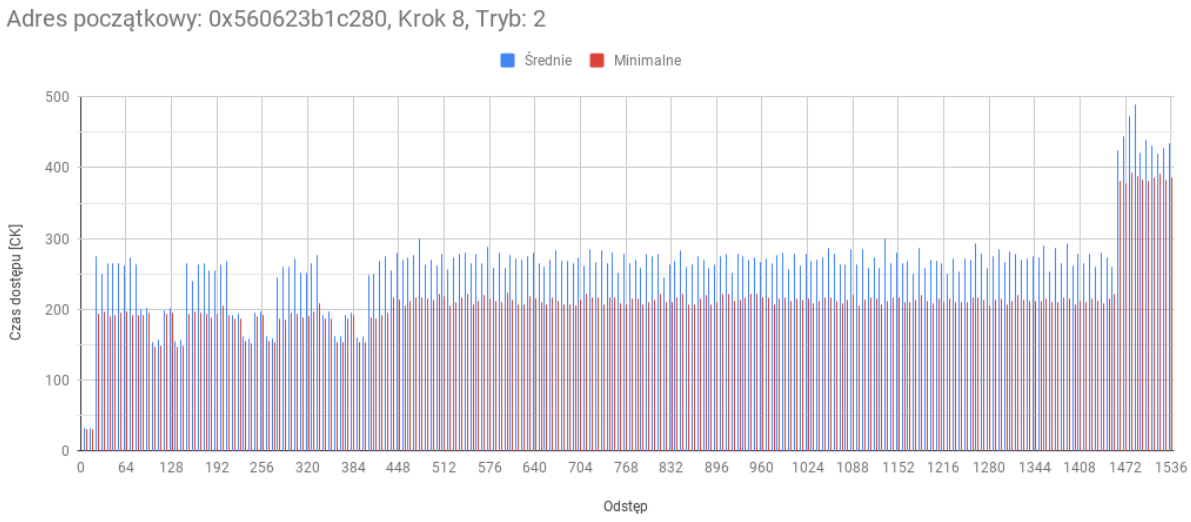
Rysunek 6: Demonstracja czasów dostępu z krokiem 1 zaczynając od elementu [a][8]

Pierwszym wnioskiem odnośnie struktury pamięci DDR jest więc fakt, że adresy w pamięci wirtualnej mapowane są do różnych banków w grupach po 8 słów maszynowych. Można też wyróżnić podział na zestawy 16 kolejnych adresów, które są odczytywane odczytując dowolny z nich.

Następną cechą, którą starałem się przetestować jest sposób w jaki takie zestawy są rozmieszczane pomiędzy bankami. Kolejne testy zostały więc przeprowadzone testując pary adresów z krokiem równym 8. Ich wyniki zdają się przejawiać różne zależności zależnie od wybranego trybu.



Rysunek 7: Przykładowy pomiar puli adresów z krokiem 8 w trybie pierwszym



Rysunek 8: Przykładowy pomiar puli adresów z krokiem 8 w trybie drugim

W przypadku mierzenia czasu dla obu odczytów w sekwencji nie jestem w stanie zauważyć żadnej okresowości ani regularnie powtarzających się wzrostów. Stanowczo wzrasta również rozrzut wyników. Zupełnie inaczej dzieje się natomiast w przypadku drugiego trybu pomiarów. Wtedy to wyniki oscylują według kilku wartości progowych i znacznie

maleje różnica między wartościami średnimi i najniższymi w każdej serii. Przedstawione różnice zachodzą niezależenie od tego, który adres z pary jest odczytany jako pierwszy. Ustalone w wyniku badań progi dla trybu drugiego to:

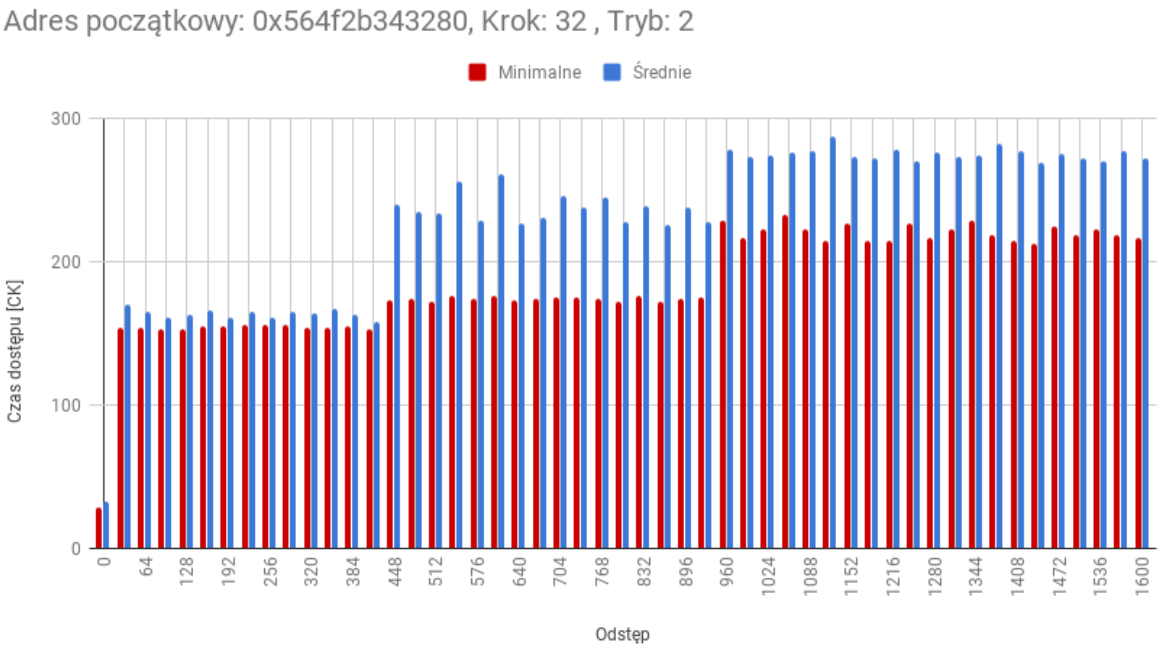
- 150 CK
- 200 CK
- 380 CK

Można zauważyć, że pomiędzy dwoma pierwszymi progami zachodzi mniej więcej taka sama różnica jak pomiędzy czasami dostępu wyliczonymi na podstawie wykresów czasowych dla przypadków, gdy odczyt wykonywany jest z otwartego banku i, gdy wcześniej wykonywana jest aktywacja. W takim wypadku obciążenie metody pomiarowej wynosiłoby ok. 90 CK. Dwa pierwsze progi występują w różnych sekwencjach zależnie od adresu początkowego pomiarów, a trzeci zaczyna występować dopiero po przekroczeniu pewnego odstępu granicznego, który również zdaje się zależeć od testowanej puli adresów. Przykładowo na powyższym wykresie dla wartości w początkowych możemy zauważyć sekwencję:

2 x 1 próg - 10 x 2 próg - 2 x 1 próg - 2 x 2 próg.

Zauważamy także odstęp graniczny 1456 (91 wierszy tablicy), po którym czasy oscylują wokół trzeciego progu.

Z kolei poniższy wykres z badań dla innej puli adresów oraz z krokiem 32 przedstawia wyniki cały czas będące w okolicy pierwszego progu, aż do odstępu granicznego 960 (60 wierszy tablicy), gdzie zaczynają one oscylować wokół drugiego progu.



Rysunek 9: Przykładowy pomiar puli adresów z krokiem 32

Jak widać w tym przypadku nie występują wyniki zbliżone do trzeciego progu czasowego. Może to wynikać z innego mapowania testowanego tutaj bloku pamięci lub z faktem, że odstęp graniczny dla tego zestawu adresów jest wyższy niż 1600.

5 Podsumowanie i wnioski

Ostatecznie uważam, że udało mi się pokazać działanie architektury prefetch oraz opisane w sekcji teoretycznej trzy możliwe warianty dostępu do pamięci DDR. Uważam też, że znacznie więcej informacji można wyciągnąć z wykorzystania drugiego trybu przygotowanego w ramach projektu programu. Głównym powodem tego przekonania jest znacznie mniejszy rozrzut wyników i zauważalne powtarzalności.

Potencjalną przyczyną mógłby być sposób rozmieszczenia danych w bankach. Jeśli znajdowałyby się one w dwóch bankach co osiem słów maszynowych na przemian każdy dostęp powodowałby aktywację obu tych banków w wyniku działania prefetcha. Gdyby bank pozostał aktywny do czasu następnego dostępu stale osiągany byłby pierwszy próg. Natomiast gdyby bank został zamknięty przykładowo w celu odświeżenia stale napotykalibyśmy drugi próg. Możliwe, że zachodzi taka sytuacja w wyniku działania kontrolerów pamięci. Wyjaśniłoby to także występowanie odstępu granicznego trzeciego progu, ponieważ w wyniku każdego dostępu aktywowane są dwa banki. Zakładam, że podane przy komendzie aktywacji adresy rzędów byłyby tymi samymi, w których znajdują się wartości z rzędu pierwszego adresu pary. Wtedy to niezależnie, który bank byłby otwierany w wyniku drugiego dostępu musiałoby zostać wykonane polecenia PRECHARGE oraz ACTIVATE z odpowiednim rzędem. Odstęp graniczny zależałby wtedy od adresu kolumny, w której leżałoby pierwsze 16 zmiennych z testowanej puli. W momencie, w którym kończy się kolumna i zaczyna następny rząd moglibyśmy zaobserwować pierwsze czasy bliskie trzeciego progu.

Taką hipotezę zdaje się również potwierdzać zmierzona przeze mnie zależność dotycząca architektury prefetch. W przypadku badań dostępów w wariancie w którym jeden z pary adresów odczytywany jest poza mierzoną sekwencją za każdym razem do pamięci podręcznej trafiały jedynie adresy umieszczone w jednym wierszu tablicy. Przykładowo jeśli niższy adres porównywanej pary będzie oznaczony indeksem [a][8], to tylko adresy z wiersza a pojawią się w pamięci podręcznej. Gdyby jednak dane były rozmieszczone w większej ilości banków spodziewałbym się, że za każdym razem będzie odczytywane będzie

16 kolejnych zmiennych z krokiem równym 8 (przykładowo prefetch wywołany pod adresem [a][8] spowoduje zebranie wartości od [a][8] do [a+1][8]).

Oczywiście należy wziąć pod uwagę fakt, że ze względu na ograniczenia czasowe i sprzętowe nie byłem w stanie przetestować całego bloku pamięci a jedynie pewne jego segmenty. Zachodzi więc możliwość, że testowane przeze mnie zależności mogą mieć miejsce jedynie dla niewielkich bloków pamięci, a reszta danych jest ulokowana w innych bankach. Uważam jednak, że gdyby alokować jeszcze większą część pamięci i przeprowadzić znacznie większą ilość testów możliwe byłoby wskazanie co najmniej par adresów, dla których zachodzą konflikty rzędów w bankach.

Na podstawie przedstawionych powyżej wniosków wyłania się powód pominięcia przeze mnie w analizie wpływu grupowania banków na czasy dostępu. Przede wszystkim zgodnie z przyjętą przeze mnie teorią odnośnie rozmieszczenia danych w pamięci każdy dostęp będzie wykonywany do tych samych dwóch grup. Natomiast w sytuacji, w której moje założenia byłyby błędne napotykamy inny problem: różnice w czasach dostępu są zbyt małe by testować je z zadowalającym poziomem ufności. Nawet w przypadku dłuższych sekwencji nie sądzę, że przekroczyłyby one rozbieżności wynikające z niepewności pomiarowej, a dodatkowo mogłaby jeszcze wzrosnąć losowość wyników.

Na zakończenie wymienię jeszcze możliwe usprawnienia programu i metody pomiarowej, które w moim odczuciu mogłyby wpłynąć korzystnie na dokładność wyników:

1. Przeniesienie programu w sferę jądra systemu (tzw. kernel-space) umożliwiłoby alokację pamięci pod testy ze stuprocentowym zapewnieniem ciągłości alokowanych adresów fizycznych oraz nadanie znacznie większego priorytetu programowi. Tym sposobem można by zmniejszyć czynnik losowy w pomiarach.

2. Oczywistym usprawnieniem byłoby wykonanie znacznie większej ilości pomiarów w każdej serii. Musiałem jednak ograniczyć się do 100 ze względu na czas jaki zajęłyby wtedy pomiary.

3. Pomocne mogłoby się okazać rozczytanie mapowania adresów wirtualnych na podstawie plików `maps` i `pagemap` udostępnianych przez system *Linux*.

Bibliografia

[1] DDR4 SDRAM JEDEC Standard

[2] 8Gb: x4, x8, x16 DDR4 SDRAM - Micron Technology, Inc.

[3] IA-32 Intel Architecture Software Developers Manual vol. 2 - Instruction Set Reference

[4] International Standard ISO/IEC 14882:2017(E) – Programming Language C++

[5] Using the GNU Compiler Collection 6.47.2 - Extended Asm