Analiza i modelowanie wydajności obliczeń

Lab 7. Skalowalność obliczeń na pojedynczym serwerze / węźle obliczeniowym

Cel: Przetestowanie skalowalności wykonywania operacji przez rdzenie oraz uzyskiwania dostępu do pamięci dla pojedynczej maszyny z pamięcią wspólną

Wstęp

- Celem laboratorium jest przetestowanie skalowalności obliczeń na serwerze dla przykładowych programów o różnych charakterystykach:
  - *scalability_vector_flops* – program będący specjalną wielowątkową wersją programu *latency_throughput_vector_flops* z laboratorium nr 3 – przykład programu wielowątkowego, dla którego czas wykonania i wydajność są w pełni determinowane przez możliwości przetwarzania potoków zmiennoprzecinkowych w rdzeniach
  - *memory_scalability* – wielowątkowa wersja kodu *multiple_arrays.c* z laboratorium 5 – przykładu kodu, w którym wydajność i czas wykonania są determinowane przez szybkość pobierania danych z pamięci i jednocześnie już jednowątkowe obliczenia starają się w pełni wykorzystać możliwości współbieżnego działania sprzętu (procesora i układu pamięci), którego celem jest ukrywanie opóźnienia w dostępie do pamięci

Kroki realizowane w ramach laboratorium:

1. **Zadanie 1. Badanie skalowalności w przypadku programu intensywnie korzystającego z potoków przetwarzania operacji zmiennoprzecinkowych, o pomijalnym czasie dostępu do pamięci**
   1. Rozpakuj paczkę *node_scalability.tgz*, wejdź do katalogu *01_instruction_throughput*
   2. W pliku *scalability_vector_flops.c* znajduje się wersja mikrobenchmarku z lab_03 mierzącego wydajność wykonywania operacji arytmetycznych, przystosowana do pomiaru skalowalności obliczeń arytmetycznych (z pomijalnym czasem dostępów do pamięci). Kod wykorzystuje wątki POSIX-owe *pthreads* i tym razem każdy wątek wielokrotne wykonuje kod:

```
int my_first_index = LOCAL_SIZE*my_id;
for(k=0; k<UNIT_SIZE; k++){
  int ijk = my_first_index+j*UNIT_SIZE;
  array_a[ijk+k] = 1.00000001*array_a[ijk+k]+0.000001;
}
```

      Wartość parametru **`UNIT_SIZE`** to rozmiar fragmentu tablicy **`array_a`**, przyjęty jako optymalny (gwarantujący najwyższą wydajność) dla danej architektury rdzenia (z założenia rozmiar powinien pozwalać na zmieszczenie fragmentu tablicy całkowicie w dostępnych rejestrach wektorowych rdzenia – dla architektur Haswell i Broadwell firmy Intel dobrą wartością jest 48).

      Pętla po indeksie **`k`** jest wykonywana wielokrotnie (**`NUM_ITER`** razy), tak aby czas pobrania fragmentu tablicy **`array_a`** z pamięci do rejestrów wektorowych stał się pomijalnie mały w stosunku do czasu wykonywania operacji arytmetycznych (to założenie odbiega od warunków realizacji większości standardowych obliczeń praktycznych – jednak dzięki niemu możemy w analizie zaniedbać czas dostępu do pamięci DRAM).

      Po wielokrotnym wykonaniu pętli po indeksie **`k`**, każdy wątek przechodzi do następnej wartości indeksu **`j`**, czyli do następnego małego fragmentu tablicy **`array_a`**, na którym wielokrotnie wykonuje operacje w pętli po indeksie **`k`**. Zakres indeksów **`j`** jest

tak dobrany, aby w ostateczności, niezależnie od liczby wątków **NO_TH**, przetwarzanie w algorytmie objęło całą tablicę **array_a** o rozmiarze **UNIT_SIZE * MULT** (jak widać niezależnym od liczby wątków).

[*w szczegółach: każdy wątek wykonuje operacje na* ***MULT/NO_TH*** *fragmentach tablicy* ***array_a****, gdzie* ***NO_TH*** *jako parametr określający liczbę wątków pthreads w programie jest zmienny przy badaniu skalowalności, a* ***MULT*** *jest dowolnie dobranym stałym parametrem, który musi tylko spełniać warunek bycia podzielnym przez każdą z uwzględnianych w eksperymencie wartości* ***NO_TH****). W ramach rozważanego algorytmu, każdy wątek w pojedynczej pętli po indeksie* ***k*** *pracuje na fragmentach o rozmiarze* ***UNIT_SIZE****, w związku z czym ostatecznie przetwarza część tablicy* ***array_a****, o rozmiarze* ***LOCAL_SIZE = UNIT_SIZE*MULT/NO_TH*** *i początkowym indeksie* ***LOCAL_SIZE*thread_id*** *.*]

Dzięki poczynionym założeniom, niezależnie od liczby wątków, w kodzie zawsze wykonywana jest liczba operacji:

global_nr_oper = 2 * NUM_ITER * UNIT_SIZE * MULT

(w przykładowym kodzie liczba **NUM_ITER** = 10000000, a **MULT** = 80).

3. Skompiluj i uruchom kod w dostarczonej konfiguracji (za pomocą *make* i pamiętając o zainicjowaniu środowiska kompilatora *icc*), z jednym wątkiem wykonującym obliczenia i sprawdź czy wynik wydajności jest zbliżony do maksymalnej wydajności jednowątkowej uzyskanej w lab_03 programem *latency_throughput_vector_flops*

   - kompilacja za pomocą *make* powoduje powstanie dwóch programów, standardowej wersji *scalability_vector_flops.exe* uruchamianej bezpośrednio po kompilacji oraz dodatkowego programu *scalability_vector_flops_papi.exe*, który korzystając z funkcji dostarczonych w adapterze biblioteki *papi* przystosowanym do obliczeń wielowątkowych (*papi_driver_mth*) podaje także wydajność w flop/takt (dla 1 wątku), celem sprawdzenia czy jest ona bliska maksymalnej dla rdzenia (w przypadku serwera HONORATA jest to 16 flop/takt)
   - kompilacja programu *scalability_vector_flops_papi.exe* wymaga dołączenia biblioteki *papi_driver_mth* , **przy pierwszym uruchomieniu kodu konieczne jest utworzenie biblioteki za pomocą: *make recreate_papi_lib_mth***

   [kod w obecnym laboratorium różni się w dwóch istotnych punktach od kodu z lab_03 – nie wykonuje rozgrzewki przed właściwymi obliczeniami, dla których mierzy się wewnętrzną wydajność, i nie synchronizuje działania wątków poza właściwymi obliczeniami. Dzięki temu **wydajność mierzona wewnętrznie** (dla właściwych obliczeń) może być niższa niż w lab_03, jednak **wydajność mierzona zewnętrznie** (uwzględniająca czas tworzenie wątków, inicjowania tablic i inne narzuty, widoczne w kodzie źródłowym) powinna być wyższa niż w lab_03 i zbliżona do obecnej wydajności wewnętrznej.]
   [innym czynnikiem mogącym zmieniać wyniki wydajnościowe może być **częstotliwość pracy rdzenia** – ***sposobem na uwzględnienie wpływu zmiennej częstotliwości pracy rdzeni jest użycie wersji z biblioteką PAPI* scalability_vector_flops_papi.exe**
   lub wykonanie obliczeń pod kontrolą* perf stat (uzyskując *przybliżoną częstotliwość pracy procesora):
   perf stat ./scalability_vector_flops.exe
   a następnie obliczenie wyniku w niezależnej od częstotliwości mierze:
   flop / takt = Gflop/s / GHz *]*

4. Zmieniając wartość parametru **NO_TH** dokonaj obliczeń dla rosnącej liczby wątków **(od 1 do co najmniej 16 lub liczby rdzeni mikroprocesora, jeśli jest mniejsza od 16)** i sprawdź skalowalność w sensie silnym

- bazując na doświadczeniu z poprzedniego laboratorium najlepiej uruchamiać program poleceniem *numactl* (numery konkretnych rdzeni najlepiej dobrać indywidualnie):

  numactl -C ...,...,...,...,... ./scalability_vector_flops.exe

  co pozwala na optymalne wykorzystanie rdzeni

- stwórz, na podstawie zwracanych przez program parametrów **wewnętrznego i zewnętrznego czasu wykonania kodu (*wallclock time*) dwie tabele** o następującym formacie (dla każdej liczby wątków należy obliczenia powtórzyć kilka razy i wybrać najkrótszy czas oraz najwyższą wydajność):

Liczba procesorów	1	2	4	8	16	...	...	itd.
Czas wykonania (wewn/zewn)								
Przyspieszenie obliczeń								
Efektywność zrównoleglenia								
Wydajność [Gflop/s]								
Częstotliwość [Ghz]								
Wydajność [flop/takt]								

[Uwaga: dwie tabele dotyczą dwóch pomiarów: wewnętrznego przez poszczególne wątki i zewnętrznego przez wątek nadrzędny (w modelu manager-worker). Pierwszy pomiar ma na celu wskazanie skalowalności bez narzutu systemowego, **jednak sposób obliczania wydajności polega na założeniu równoległości pracy wątków i jest realny tylko do liczby wątków równej liczbie rdzeni (nie należy jej przekraczać w tej tabeli).** Drugi pomiar, zewnętrzny, jest prawdziwym pomiarem wydajności z punktu widzenia użytkownika, jednak uwzględnienie narzutu systemowego zaburza obraz idealnego współdzielenia zasobów procesora i idealnej skalowalności, którą powinny uzyskać czyste obliczenia. Drugi pomiar podaje parametry wydajności dla dowolnej liczby wątków.

[program oprócz samego pomiaru czasu i obliczania wydajności w GFLOP/s oblicza także narzut systemowy – na uruchomienie i zarządzanie wątkami, synchronizację itp., Dla domyślnie ustawionych parametrów wykonania programu, w szczególności liczby powtórzeń obliczeń `NUM_ITER`, czas wykonania programu jest krótki (tak aby wiele osób mogło wykonywać obliczenia nie przeszkadzając sobie nawzajem) co powoduje jednak, że narzut jest znaczący. Jeśli narzut staje się zbyt duży (ponad 10%) można go zmniejszyć powtarzając obliczenia (w czasie kiedy serwer nie jest obciążony) zwiększając wartość parametru `NUM_ITER` np. 10 razy – należy wtedy do tabeli wpisać i przyjąć do obliczenia przyspieszenia czas wykonania odpowiednio pomniejszony, w tym wypadku 10 razy (wydajność w Gflop/s program powinien obliczyć prawidłowo – należy ją bezpośrednio wpisać do tabeli)]

5. Skonstruuj wykresy wydajności (w Gflop/s oraz flop/takt), przyspieszenia obliczeń i efektywności zrównoleglenia jako funkcji liczby wątków (wykres przyspieszenia obliczeń powinien w swoim charakterze odpowiadać odpowiednio przeskalowanemu wykresowi wydajności w GFLOP/s, ze względu na podstawową relację: wydajność jako odwrotność czasu wykonania lub bardziej szczegółowo: wydajność równoległa = przyspieszenie równoległe * wydajność sekwencyjna). Wyciągnij wnioski z charakteru wykresów i umieść je w sprawozdaniu (dla ułatwienia wyciągnięcia wniosków, zaznacz na wykresie przyspieszenia prostą przyspieszenia idealnego S(p)=p).

   [Na pojedynczym wykresie można umieścić dane z obu tabel, co prowadzi do czterech wykresów, dwie krzywe na każdym wykresie]

[Jeśli dane dla kolejnych liczb wątków (1,2,4,8,...) umieszczane są w równych odstępach na wykresie, odpowiednią do badania charakteru wykresów jest skal logarytmiczna na osi y]

[Na skalowalność niewątpliwie ma wpływ fakt, że w miarę rosnącej liczby wątków i pracujących rdzeni, system najczęściej zmniejsza częstotliwość pracy procesora. W celu zobrazowania tego zjawiska można umieścić na wykresie rosnącej wydajności obie krzywe w Gflop/s i w flop/takt – dobierając skale tak, żeby krzywe zaczynały się w tym samym punkcie]

[W sprawozdaniu umieść wnioski dotyczące skalowalności - malejącego czasu obliczeń (klasycznego przyspieszenia równoległego), rosnącej wydajności w Gflop/s i wydajności w flop/takt - w jakim zakresie można powiedzieć, że skalowalność jest liniowa?]

[Porównaj wartości w tabelach i charakter wykresu dla czasu wewnętrznego (czyste obliczenia bez narzutów) i zewnętrznego (z narzutami) - czy można oszacować jaka jest proporcja narzutu do czasu obliczeń dla rosnącej liczby wątków?]

2. Zadanie 2 - Badanie skalowalności w przypadku programu intensywnie (i w sposób zbliżony do optymalnego) korzystającego z dostępów do pamięci, o pomijalnym czasie wykonywania pozostałych operacji

1. Przejdź do katalogu *02_memory_throughput*
2. W pliku *memory_scalability.c* znajduje się wersja benchmarku wykorzystanego w lab_05 do pomiaru przepustowości pamięci różnych poziomów, także przystosowana do pomiarów skalowalności, tym razem z zastosowaniem standardu obliczeń wielowątkowych OpenMP.
   - W wersji dostarczonej w paczce *node_scalability.tgz* obliczenia (eksperymenty) przeprowadzane są kolejno dla 4 rozmiarów tablic oraz sekwencyjnej i wektorowej wersji kodu.
   - Rozmiary tablic można dobierać dowolnie np. jako typowe dla wielkości pamięci, kolejno, L1, L2, L3 i DRAM (np. 400, 3200, 25600, 409600), tak aby 5 tablic liczb typu double mieściło się w pamięci badanego poziomu, a znacząco przekraczało rozmiar pamięci niższego poziomu (dla L1 łączną liczbę zmiennych w rejestrach wektorowych procesora).
   - W przypadku każdego z poziomów należy uwzględnić, czy dana pamięć jest wspólna dla wielu rdzeni, czy odrębna. Jeśli jest wspólna, to dla obliczeń wielowątkowych w pamięci powinno się zmieścić nie 5 ale 5*NO_TH tablic, gdzie NO_TH jest liczbą wątków wykonujących obliczenia na jednym procesorze (stąd w przykładowym kodzie relatywnie niska wartość rozmiaru tablicy dla pamięci L3).
3. Dla każdego rozmiaru tablicy (zmienna *working_set_size = size_array[i_size]*) pojedynczy wątek wykonuje wielokrotnie pętlę

```
int jk=(fragments_per_thread*omp_get_thread_num()+i)*working_set_size;

for (j=0; j<working_set_size; j++){
   ijk = jk+j;
   array_a[ijk] = array_e[ijk]*array_b[ijk]+array_d[ijk]*array_c[ijk];
}
// kod uniemożliwiający kompilatorowi optymalizacje
// usuwające niektóre obliczenia
```

Liczba wykonań pętli jest tak dobrana, żeby dla każdego rozmiaru tablic wykonywana była ta sama liczba dostępów do elementów tablicy, niezależnie od liczby wątków.

[*Aby to osiągnąć wątki pracują na pewnej liczbie fragmentów (fragments_per_thread), każdy o rozmiarze working_set_size, które w sumie tworzą część tablicy o rozmiarze MULT_MEM_MAX*working_set_size/NO_TH (cała tablica ma rozmiar*

*MULT_MEM_MAX*working_set_size , który jest niezależny od liczby wątków, co odpowiada normalnej sytuacji zrównoleglenia zadań obliczeniowych).*]

4. Skompiluj i uruchom kod w dostarczonej konfiguracji (za pomocą *make*)

   [W wersji w paczce rozmiary wszystkich tablic dobrane są do pomiaru wydajności pamięci DRAM – tak powinno pozostać w obliczeniach w p. 5, 6, 7, 8]

   [Należy zapewnić, aby parametr MULT_MEM_MAX miał wartość podzielną przez każdą liczbę wątków planowanych do użycia w eksperymencie]

   1. Ustal wartość zmiennej środowiskowej standardu OpenMP **`OMP_NUM_THREADS`** na 1 (**`export OMP_NUM_THREADS=1`** dla bash i **`setenv OMP_NUM_THREADS 1`** dla tcsh).
   2. Uruchom kod i sprawdź czy wartości przepustowości pokrywają się z wynikami z lab_05 uzyskanymi programami *multiple_arrays_scalar.exe* i *multiple_arrays_vector.exe*

   [na serwerze HONORATA, ze względu na użycie specjalnej wersji alokacji pamięci (z wyrównaniem na odpowiedniej granicy), specyficznej dla kompilatora *icc* (funkcja *_mm_malloc*), wyniki mogą być nawet lepsze, niż w lab_05, gdzie stosowana była przenośna funkcja *posix_memalign*)]

5. Sterując wartością zmiennej **`OMP_NUM_THREADS`** przeprowadź obliczenia dla różnych liczb wątków, **od 1 do do co najmniej 16 lub liczby rdzeni mikroprocesora, jeśli jest mniejsza od 16**. Użyj dodatkowo zmiennej środowiskowej określającej sposób przypisania wątków do rdzeni: dla kompilatora *icc* – **`KMP_AFFINITY`** z opcjami **`compact`** lub ***`scatter`*** **`:`**

`export KMP_AFFINITY=`*`{`***`compact,scatter`***`}`*** dla bash i
`setenv KMP_AFFINITY=`*`{`***`compact,scatter`***`}`*** dla tcsh),

   [dla weryfikacji, że przypisanie pracuje w założony sposób można umieścić opcje jawnego wypisania przypisania wątków do rdzeni, np:

   `export KMP_AFFINITY=verbose,scatter ]`

   [ważną opcją **`KMP_AFFINITY`** jest także parametr bezpośrednio po typie przypisania, który określa poziom przydzielania kolejnych wątków:

   `export KMP_AFFINITY=verbose,compact,0`

   przydziela kolejne wątki do kolejnych procesorów logicznych, natomiast

   `export KMP_AFFINITY=verbose,compact,1`

   przydziela kolejne wątki do kolejnych rdzeni procesora.

   W efekcie, np. dla serwera HONORATA, dla pierwszej opcji 22 wątków jest przydzielane jednemu mikroporcesorowi, a dla drugiej obu mikroprocesorom. Podstawowy wariant badany w ramach laboratorium powinien korzystać z przydzielania kolejnym rdzeniom, natomiast wariant z przydzielaniem kolejnym procesorom logicznym można zanalizować w ramach zadań dodatkowych]

   1. *make* każdorazowo uruchamia program, jednak w celu uzyskania bardziej powtarzalnych wyników konieczne jest wykonywanie z przypisaniem do konkretnych rdzeni (za pomocą zmiennych środowiskowych jak np. *KMP_AFFINITY lub* pod kontrolą *numactl*)
   2. zawsze przydatne może być uruchomienie pod kontrolą *perf stat* i weryfikacja parametrów wykonania (czas, częstotliwość procesora itp.), np:

      `perf stat ./memory_scalability_scalar.exe`

      `perf stat ./memory_scalability_vector.exe`

6. **W pierwszym kroku dokonaj obliczeń i skonstruuj wykresy pod kątem przepustowości pamięci DRAM**
   - w dostarczonej wersji program wykonuje 4 krotnie obliczenia dla rozmiaru wybranego jako typowy dla badań pamięci DRAM (409600). Dzięki temu już po jednorazowym uruchomieniu powinno być możliwe uzyskanie wiarygodnych parametrów wydajnościowych.
   - zmiany zestawu rozmiarów tablic dokonuje się w liniach ok.70. Można wybrać ten sam rozmiar dla wszystkich tablic lub kilka rozmiarów do badania różnych poziomów pamięci).
7. **Wypełnij odpowiednie pozycje poniższej tabeli dla różnych liczb wątków, starając się uzyskać wiarygodne wyniki .**
   - **Zaobserwuj, że wydajność dla rozkazów skalarnych pobierania z pamięci i rozkazów wektorowych jest zbliżona. W związku z tym w tabeli umieść lepsze z nich (o wyższej wydajności).**
     - uwagę o ewentualnych różnicach w wydajności między opcjami vector i scalar można umieścić w sprawozdaniu

Liczba procesorów	1	2	4	8	16	...	...	itd.
- przypisanie (*compact)*								
Czas wykonania								
Przyspieszenie obliczeń								
Efektywność zrównoleglenia								
Przepustowość [GB/s]								
Częstotliwość [Ghz]								
Przepustowość [B/takt]								
- przypisanie (*scatter*)								
Czas wykonania								
Przyspieszenie obliczeń								
Efektywność zrównoleglenia								
Przepustowość [GB/s]								
Częstotliwość [Ghz]								
Przepustowość [B/takt]								

[***podobnie jak poprzednio program oprócz samego pomiaru czasu i obliczania wydajności w GB/s oblicza także narzut systemowy – na uruchomienie i zarządzanie wątkami, synchronizację itp., jeśli narzut staje się zbyt duży (ponad 10%) należy go zmniejszyć zwiększając liczbę powtórzeń pętli (zwiększając wartość parametru MULT, np. 10 razy) – należy wtedy do tabeli wpisać i przyjąć do obliczenia przyspieszenia czas odpowiednio pomniejszony – w tym wypadku 10 razy (wydajność w GB/s program powinien obliczyć prawidłowo – należy ją bezpośrednio wpisać do tabeli). Jako regułę można przyjąć, że obliczenia dla pojedynczego poziomu pamięci powinny trwać od 0.1 do 1.0 sekundy.***]

[**Na skalowalność ma jak zwykle wpływ fakt, że w miarę rosnącej liczby wątków i pracujących rdzeni, system najczęściej zmniejsza częstotliwość pracy**

procesora. W celu zobrazowania tego zjawiska należy dokonać obliczeń wydajności także w niezależnej od częstotliwości pracy mierze:

B / takt = GB/s / GHz]

8. Skonstruuj wykresy wydajności (dla obu miar GB/s i B/takt), przyspieszenia obliczeń i efektywności zrównoleglenia jako funkcji liczby wątków dla przepustowości pamięci DRAM (4 wykresy). Na każdym z wykresów umieść 2 krzywe – dla wariantów *compact* i *scatter.* Wykresy te powinny prowadzić do wniosków jak skaluje się wydajność dostępu do pamięci DRAM dla różnych sposobów przypisania wątków do rdzeni. (dla ułatwienia wyciągnięcia wniosków, zaznacz na wykresach przyspieszenia prostą przyspieszenia idealnego S(p)=p).

---------- 3.0 ----------

3. **Zadanie 3. Badanie skalowalności przy dostępie do pamięci podręcznych L1, L2, L3**
   1. Manipulując rozmiarami tablic w programie *memory_scalability.c* przeprowadź eksperymenty dla różnych poziomów pamięci podręcznej (L1, L2, L3) i wypełnij tabele podobne do tej dla pamięci DRAM, tyle że uwzględniając zamiast opcji *scatter i compact,* opcje użycia rozkazów skalarnych i wektorowych. Dla pamięci L3 sensowne jest porównanie wydajności dla wszystkich czterech opcji (*scatter+scalar, compact+scalar, scatter+vector i compact+vector*)
   2. Powtórz konstrukcję wykresów wydajności, przyspieszenia obliczeń i efektywności zrównoleglenia z p.2.8, tym razem jako funkcji liczby wątków dla przepustowości pamięci podręcznych L1, L2, L3. Zaobserwuj jak różni się wydajność dostępu dla rozkazów skalarnych i wektorowych. Wypełnij odrębną tabelę dla każdego poziomu pamięci i wariantu skalarnego oraz wektorowego (podział tabeli na warianty *compact* i *scatter* zachowaj tylko w przypadku jeśli czasy odpowiadające różnym wariantom różnią się). Na wykresach skalowalności umieść tyle krzywych ile jest interesujących, różniących się wynikami wariantów: *scalar-compact, scalar-scatter, vector-compact, vector-scatter* (2 lub 4 krzywe na wykres).

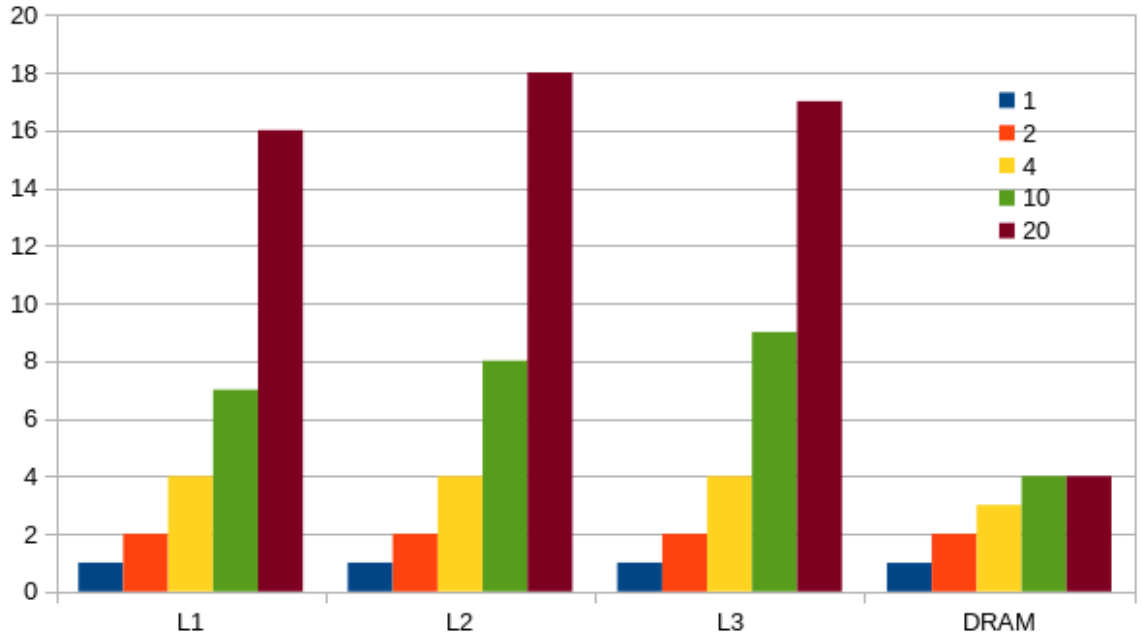
      [**Na skalowalność ma jak zwykle wpływ fakt, że w miarę rosnącej liczby wątków i pracujących rdzeni, system najczęściej zmniejsza częstotliwość pracy procesora. W celu zobrazowania tego zjawiska należy dokonać obliczeń wydajności także w niezależnej od częstotliwości pracy mierze:**

      B / takt = GB/s / GHz]
   3. Wyciągnij wnioski z charakteru wszystkich otrzymanych wykresów i umieść je w sprawozdaniu (dla ułatwienia wyciągnięcia wniosków, zaznacz na wykresach przyspieszenia prostą przyspieszenia idealnego S(p)=p).

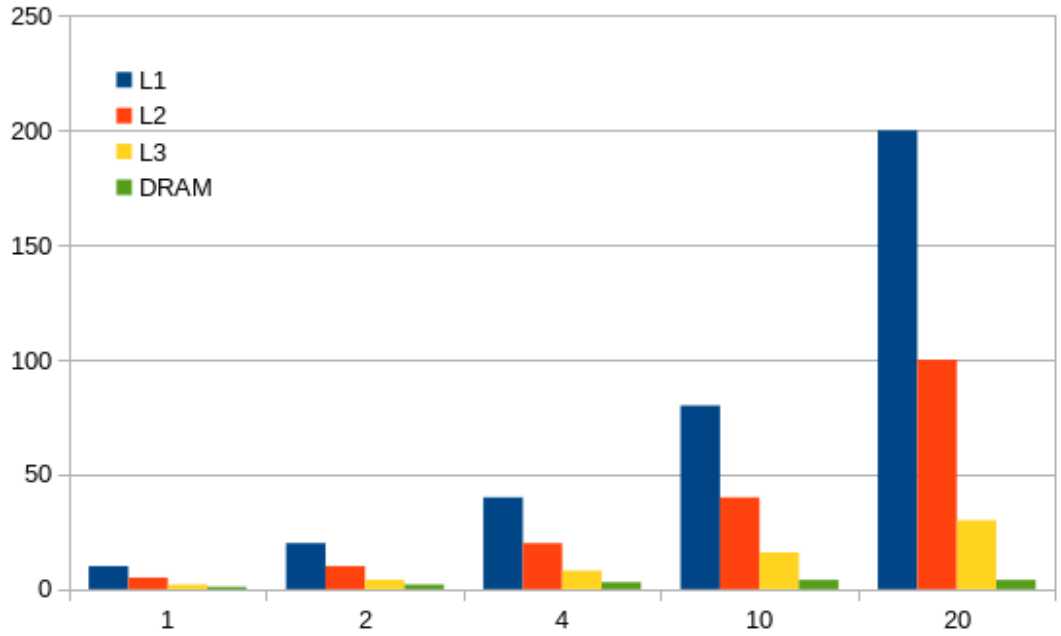
      [wykresy przyspieszenia obliczeń powinny w swoim charakterze odpowiadać odpowiednio przeskalowanym wykresom przepustowości w GB/s, ze względu na podstawową relację: wydajność – w tym wypadku określana przez przepustowość – jako odwrotność czasu wykonania]

---------- 4.5 ----------

4. Na podstawie wyników dla wszystkich poziomów pamięci skonstruuj jeden wykres przyspieszenia obliczeń na którym znajdzie się porównanie skalowalności dla wszystkich poziomów pamięci. Do konstrukcji użyj najlepszych wyników dla każdego poziomu i liczby wątków równej: 1, 2, 4, 8 i 16. Wykres powinien mieć postać jak poniżej (dane są fikcyjne, liczby wątków inne):



5. Dla tych samych przypadków (choć tym razem w dwóch wariantach skalarnym i wektorowym) skonstruuj 2 wykresy przepustowości dla określonych poziomów pamięci podręcznej. Każdy wykres (dla wariantu skalarnego i wektorowego) powinien mieć postać (dane poniżej jak zwykle fikcyjne, liczby wątków inne):



6. Ciekawym poznawczo rozszerzeniem powyższej analizy jest zastosowanie skali logarytmicznej dla osi y. Ze względu na rosnące dwukrotnie za każdym razem liczby wątków, na wykresie można będzie dostrzec do jakiego stopnia skalowalność jest liniowa
7. W sprawozdaniu proszę zapisać wnioski z całego eksperymentu (wszystkie wykresy), będące odpowiedziami na pytania: jak wiele możemy skorzystać wprowadzając do kodu lokalność odniesień (czyli korzystając jak najwięcej z pamięci podręcznej kolejnych poziomów), jak zysk zmienia się z liczbą wątków/rdzeni, na ile zysk zależy od wariantów implementacji i uruchomienia (wersje *scalar* i *vector, compact* i *scatter*)

Sprawozdanie:

1. Zrealizowane kroki, spostrzeżenia z analizy kodu (a także ewentualnie odpowiadającego kodu asemblera i uzyskanych wartości liczników sprzętowych), wykresy, opis wykresów, wnioski – zgodnie z regulaminem laboratoriów