Analiza i modelowanie wydajności obliczeń

Lab 10. Mnożenie macierz-macierz

Cel: Analiza wydajności implementacji algorytmu mnożenia macierz-macierz.

Wstęp.

Laboratorium obecne jest kontynuacją laboratorium 9 dotyczącego optymalizacji (głównie *cache blocking*) implementacji algorytmu mnożenia macierz-macierz. Tym razem implementacje uzyskane w ramach poprzedniego laboratorium poddane zostaną analizie za pomocą narzędzi mierzących wykorzystanie pamięci DRAM i pamięci podręcznych różnych poziomów.

Wyniki badań mają odpowiedzieć na pytanie jakie operacje wykonywane są w kodzie (czy operacje arytmetyczne są realizowane przez rozkazy wektorowe, a jeśli tak to jakie, jakie inne rozkazy są wykonywane, w jaki sposób realizowane są dostępy do pamięci – czy także za pomocą operacji wektorowych, a jeśli tak to jakich), a ponadto ile danych transferowanych jest pomiędzy rozmaitymi poziomami pamięci – DRAM, L3, L2, L1, rejestry.

To ostatnie badanie wymaga użycia narzędzi takich jak **`perf stat,`** liczniki sprzętowe lub symulacje **`valgrind`**. Na podstawie oszacowania rozmiaru transferowanych danych, na podstawie maksymalnych szybkości transferu badanych w poprzednich laboratoriach, obliczane są graniczne czasy obliczeń i wydajności w Gflop/s. Graniczne czasy obliczeń i wydajności, z jednej strony podają wielkości, których nie będzie można dla danej implementacji kodu przekroczyć, a z drugiej strony będą wskazywać, w których miejscach powinna być dokonana modyfikacja lub dalsza optymalizacja kodu.

Kroki:

1. Rozpakuj paczkę *matrix_multiplication_papi_gcc.tgz* w nowym katalogu np. *lab_10*.
2. Przejdź do katalogu *matrix_multiplication_gcc* i przeglądnij pliki: *mat_mul_driver.c, mat_mul_par_papi.c, papi_set_user_events_mth.c*
   a) pierwszy z nich, zawierający wyłącznie funkcję **`main`**, alokuje pamięć dla 3 tablic **`A, B i C`**, przechowujących macierze kwadratowe o rozmiarze *n*, a następnie inicjuje tablice zadanymi (dowolnymi) wartościami i wywołuje procedurę **`mat_mul_par_papi`** obliczającą iloczyn **`C = A*B`**. Plik może być kompilowany z opcją **`-DPAPI_TEST`** lub bez niej. Bez opcji **`PAPI_TEST`** powyższe operacje są jedynymi znaczącymi za względu na wydajność. Można łatwo obliczyć, że jest to kilka (zależnie od optymalizacji) operacji na pojedynczy element tablic **`A i B`** (w sumie liczba rzędu *O(n*n)* – bez znaczenia dla całości kodu) oraz zapis wszystkich 3 tablic (z czego zapis tablicy **`C`** może zostać pominięty na skutek optymalizacji). W sumie jest to *k*n*n* zapisów, gdzie *k* = 2 lub 3.
   b) w wariancie ze zdefiniowanym symbolem **`PAPI_TEST`** program dodatkowo inicjuje zliczanie zdarzeń sprzętowych (wywołując funkcję **`papi_driver_mth_init`**) i sprawdza wynik zwracany przez funkcję **`mat_mul_par_papi`**, wykonując dla dodatkowej tablicy **`D`**, poprawne pętle obliczeniowe **`D = A*B`**, i porównując **`D`** z **`C`**.
   c) w drugim z plików, *mat_mul_par_papi.c,* znajduje się kilka implementacji mnożenia macierz-macierz, odpowiadających wersjom z poprzedniego laboratorium (w miejsce implementacji nr 3 należy użyć własnego, poprawnego i zoptymalizowanego – co najmniej *cache blocking* – kodu opracowanego w ramach laboratorium 9). Funkcja **`mat_mul_par_papi`** jest zorganizowana w taki sposób, że na początku i końcu zawiera odpowiednie wywołania funkcji z interfejsu PAPI pozwalające na zliczanie zdarzeń sprzętowych (wywołania są objęte kompilacja warunkową na podstawie definicji symbolu **`PAPI_TEST`**), natomiast w środkowej części znajdują się implementacje mnożenia macierz-macierz. Wybór implementacji do wykonania odbywa się poprzez zdefiniowanie odpowiedniej wartości dla symbolu **`WERSJA`** na początku funkcji i kompilację warunkową. Wybrana implementacja przeprowadza obliczenia, dla których zliczane są zdarzenia sprzętowe (tylko w wersji kodu **`mat_mul_driver_papi.exe`**, wersja **`mat_mul_driver.exe`** jest kompilowana bez definiowania symbolu **`PAPI_TEST`** i służy testowaniu jako standardowo wykonywana funkcja). Wykonanie zadań podczas laboratorium będzie polegało na testowaniu kolejnych

implementacji (dostarczonych lub własnych) i poddawanie ich analizie, między innymi przy użyciu liczników sprzętowych.

- pierwsza implementacja naiwnie implementuje wzór matematyczny w potrójnej pętli *ijk*
- druga implementacja poprawia pierwszą tak, aby dostępy do pamięci w najbardziej wewnętrznej pętli dotyczyły kolejnych elementów w tablicach przechowujących macierze, jest to osiągane przez zmianę kolejności pętli, prowadzącą do wersji *ikj*
- trzecia implementacja (własna) powinna dodać co najmniej optymalizację *cache blocking*, z rozmiarem bloku zdefiniowanym w pliku *sizes.h*
- dwie ostatnie wersje dodają do *cache blocking* ręcznie wykonaną optymalizację *register blocking* połączoną z wektoryzacją (wersja 4 z bloczkami 4x4x4 – jest to wersja *avx_1* z laboratorium 9, a wersja 5 z bloczkami 4x12x4 – wersja *avx_2* z laboratorium 9)

d) trzeci z plików, *papi_set_user_events_mth.c*, zawiera wybór zdarzeń zliczanych w trakcie wykonywania mnożenia macierz-macierz, zgodnie z przyjętą konwencją interfejsu z biblioteką PAPI. Jako domyślne ustawione są zdarzenia:

**`event 0: PAPI_REF_CYC`**

**`event 1: PAPI_TOT_CYC`**

**`event 2: MEM_UOPS_RETIRED.ALL_LOADS`**

**`event 3: L1D.REPLACEMENT`**

**`event 4: L2_LINES_IN.ALL`**

które powinny zliczać odpowiednio: liczbę taktów referencyjnych (z częstotliwością nominalną), liczbę taktów rzeczywistych (z częstotliwością rzeczywistą), liczbę rozkazów pobrania danych, liczbę podmian linii w L1 (związaną z transferem danych z L2 do L1 lub z L3 do L1), liczbę zaalokowanych linii L2 (związaną z transferem danych z L3 do L2 lub z L1 do L2)

[ oprócz wymienionych zdarzeń w pliku znajduje się wiele innych możliwych wyborów, opracowany interfejs umożliwia zdefiniowanie 6 zdarzeń, dla których zliczane będą liczby wystąpień, przy czym sprzętowo zrealizowane może być tylko zliczanie 4 zdarzeń użytkownika i dodatkowo 2 spośród domyślnie zliczanych każdorazowo przez rdzeń (jak np. całkowita liczba wykonanych rozkazów, której odpowiada predefiniowane zdarzenie PAPI: **`PAPI_TOT_INS`** i referencyjna liczba taktów **`PAPI_REF_CYC`** (jest ona używana do obliczenia częstotliwości pracy zegara, różni się od rzeczywistej liczby taktów **`PAPI_TOT_CYC`** proporcjonalnie tak jak częstotliwość rzeczywista różni się od częstotliwości nominalnej procesora) ]

[ zdarzenia uwzględnione w pliku *papi_set_user_events_mth.c*, poza wieloma dotyczącymi pamięci podręcznych L1 i L2, zawierają także zdarzenia dotyczące pamięci L3 i DRAM; problem z wykorzystaniem tych ostatnich zdarzeń polega na tym, że elementy sprzętowe, których dotyczą znajdują się poza rdzeniem dokonującym zliczania w swoich rejestrach, stopień złożoności tych elementów (zwłaszcza dla maszyn wieloprocesorowych z pamięcią ccNUMA) powoduje trudności z dokonywaniem pomiarów, a także ich późniejszą interpretacją; uwzględniając te zastrzeżenia, jako zdarzenia, które mogą pomóc w szacowaniu transferów pomiędzy L3 i DRAM mogą zostać wykorzystane:

**`MEM_LOAD_UOPS_RETIRED.L3_MISS`**

**`MEM_LOAD_UOPS_L3_MISS_RETIRED.LOCAL_DRAM`**

**`OFFCORE_RESPONSE_0:ANY_DATA:ANY_RFO:L3_MISS_LOCAL:SNP_ANY`** ]

[ zliczanie zdarzeń odbywa się indywidualnie dla każdego z wątków realizujących obliczenia, **interpretacja jest łatwiejsza w przypadku obliczeń jednowątkowych – takie obliczenia są podstawą zadań obowiązkowych laboratorium**, analiza wykonania wielowątkowego jest pozostawiona jako zadanie dodatkowe ]

3. Zbuduj bibliotekę interfejsu PAPI (**make recreate_papi_lib_mth**)
4. Program, łącznie z funkcją **`mat_mul_par_papi`** jest napisany w wersji równoległej OpenMP. Na potrzeby badań wydajności, w pierwszej kolejności analizowane jest wykonanie sekwencyjne, uzyskiwane przez ustawienie liczby wątków na 1 (**`export OMP_NUM_THREADS=1`**) lub zakomentowanie (tak jak w przesłanej paczce) dyrektyw zrównoleglenia (kod OpenMP różni się w pewnych miejscach wewnątrz funkcji **`mat_mul_par_papi`** od kodu sekwencyjnego)

5. Sprawdź poprawność instalacji poprzez uruchomienie programu poleceniem **`make`** (w dostarczanej wersji kodu jako domyślna uruchamiana jest najbardziej wydajna, dla kompilatora *gcc*, wersja *avx_2*)
6. Zanotuj wyniki raportowane przez narzędzie **`perf stat`** w liniach:

```
...      cycles:u                  #  ... GHz
...      instructions:u            #  ... insn per cycle
...      L1-dcache-loads:u         #  ... M/sec
...      L1-dcache-load-misses:u   #  ... of all L1-dcache hits
...      LLC-loads:u               #  ... M/sec
...      LLC-load-misses:u         #  ... of all LL-cache hits
```

   oraz przez funkcje z biblioteki PAPI, a także wydajność obliczeń w Gflops/s (obie wersje programu: **`mat_mul_driver_papi.exe`** i **`mat_mul_driver.exe`** mierzą wydajność funkcji **`mat_mul_par_papi`**, pierwsza z włączonym zliczaniem zdarzeń, druga bez)

   [ wyniki umieść w sprawozdaniu - można je porównać z innymi wersjami kodu badanymi szczegółowo w ramach laboratorium ]
7. **Zasadnicza część ćwiczenia polega na badaniu wydajności dla dwóch wersji implementacji mnożenia macierz-wektor: wersji *ikj* (druga implementacja w pliku *mat_mul_par_papi.c*) oraz wersji z optymalizacją *cache blocking* (własna wersja podstawiona w miejsce trzeciej implementacji w pliku *mat_mul_par_papi.c*)** - badanie pozostałych dwóch wersji (*ijk* oraz wersji z ręczną wektoryzacją) stanowi zadanie dodatkowe
8. Dla każdej z (dwóch) wersji implementacji mnożenia macierz-macierz należy przeprowadzić następujące kroki badania wydajności:
   a) ustalenie właściwej wersji implementacji w pliku *mat_mul_par_papi.c* za pomocą symbolu WERSJA (przykładowe oznaczenia: v2_ikj, v3_cb)
   b) otrzymanie wersji asemblera, np. za pomocą

```
gcc -S  -O3  -fopenmp -march=core-avx2 -D'NOMINAL_FREQUENCY=2.2'
mat_mul_par_papi.c -o mat_mul_par_v2_ikj.s
-I/home/students_wo/common_files/papi-6.0.0/src
-I../papi_driver_mth
```

      odnalezienie bloku podstawowej pętli obliczeniowej (zawierającej najprawdopodobniej pewną wersję operacji **`fma`** ), zapisanie kodu asemblera dla pojedynczej iteracji pętli
      - postać asemblera może wskazać potrzebę dokonania klasycznych optymalizacji - kompilator może ich, z niewiadomych przyczyn, nie przeprowadzić
      - w przypadku modyfikacji należy uzyskać nową wersję asemblera i porównać postać najbardziej wewnętrznej pętli algorytmu w starej i nowej wersji
   c) uruchomienie programu (w wersji zoptymalizowanej) za pomocą *make*
   d) zanotowanie wyników podawanych przez **`perf stat`** (wstępne uruchomienie *make*, a następne powtórzenia: ***numactl -C MY_CORE perf stat -d -d -d ./mat_mul_driver.exe*** *),* PAPI (uruchomienie ***numactl -C MY_CORE ./mat_mul_driver_papi.exe***), i **`valgrind`** (uruchomienie ***numactl -C MY_CORE valgrind --tool=cachegrind ./mat_mul_driver.exe*** )

      [ ***MY_CORE*** jest przydzielonym podczas zajęć numerem rdzenia mikroprocesora ]
      - ***w sprawozdaniu należy zamieścić wszystkie dane dotyczące wykonania:***
        - ***symbol implementacji***
        - ***rozmiar tablicy i liczba operacji [Gflop] z kodu źródłowego i wydruku***
        - ***czas wykonania i wydajność ( [Gflop/s] oraz [flop/cycle] ) z wydruku***
        - ***kod asemblera najbardziej wewnętrznej pętli algorytmu***
        - ***wyniki*** **perf stat** ***(jak w p.6 -*** **`cycles, instructions, L1-dcache-loads, L1-dcache-load-misses, LLC-loads,`** ***`LLC-load-misses`****)*
        - ***wyniki PAPI w formacie:***

```
PAPI_REF_CYC                 ....
PAPI_TOT_CYC                 ....
MEM_UOPS_RETIRED.ALL_LOADS   ....
L1D.REPLACEMENT              ....
```

**L2_LINES_IN.ALL ....**
**L3_MISS_LOCAL:SNP_ANY ....**

- ***wyniki* valgrind**

- niektóre z wyników oraz obliczone na ich podstawie pochodne parametry i miary należy umieścić w tabeli o strukturze zamieszczonej w p. 10
  - przed umieszczeniem danych w tabeli w sprawozdaniu należy zamieścić obliczenia miar pochodnych na podstawie danych powyżej (czasem przyjmując jedną z nich – kiedy różne narzędzia się różnią, lub wybierając wartość przybliżoną zbliżoną do danych pomiarowych – np. na podstawie kodu źródłowego)

e) weryfikacja kolejnych parametrów:
- l**iczba taktów podczas wykonania**: **`perf stat`** dotyczy całego programu, natomiast PAPI tylko funkcji **`mat_mul_par_papi`**, jednak w wersji **`mat_mul_driver.exe`**, dla której podawane są wyniki **`perf stat`**, czas poza funkcją jest pomijalny
  - podzielenie liczby taktów rzeczywistych przez czas (raportowany na wydruku dla funkcji **`mat_mul_par_papi`**) daje w wyniku średnią częstotliwość pracy procesora (rdzenia) w trakcie wykonywania obliczeń, wartość tę można porównać z wartością obliczoną w programie na podstawie liczby taktów rzeczywistych i referencyjnych oraz nominalnej częstotliwości pracy rdzenia (możliwa różnica wynika z faktu, że w kodzie nie da się zmierzyć niezależnie i czasu wykonania (procedurami systemowymi), i wartości liczników sprzętowych (jeden z pomiarów musi być wewnątrz drugiego) – ***w sprawozdaniu zamieść odpowiednie obliczenia***
- **liczba wykonanych rozkazów**: liczba wykonanych rozkazów (*instructions) będzie* (z dużym przybliżeniem) równa liczbie rozkazów w bloku asemblera odpowiadającym najbardziej wewnętrznej pętli kodu pomnożonej przez wielokrotność wykonania bloku w programie. Biorąc liczbę rozkazów z **`perf stat`** lub **`valgrind (I refs)`** (obie powinny być bardzo zbliżone) oraz liczbę rozkazów w bloku asemblera oblicz liczbę wykonań bloku asemblera. Porównaj liczbę wykonań bloku asemblera z liczbą iteracji w najbardziej wewnętrznej pętli implementacji wynikającą z kodu źródłowego (n*n*n). Ewentualne różnice mogą wynikać z rozwinięcia najbardziej wewnętrznej pętli implementacji przez kompilator (*loop unrolling*). ***W sprawozdaniu zamieść odpowiednie obliczenia: liczby wykonań bloku asemblera oraz wnioski z porównania jej z teoretyczną wartością n*n*n***
  - w przypadku rozwinięcia pętli przez kompilator zaobserwuj, czy zostało ono jednocześnie wykorzystane do wektoryzacji. ***Jeśli tak, podaj w sprawozdaniu jakie wektorowe operacje wykonywane są w bloku asemblera***
  - dla rozkazów operacji arytmetycznych oblicz ilu standardowym matematycznym operacjom (mnożeniom i dodawaniom) odpowiada jedna operacja wektorowa – w połączeniu z liczbą rozkazów w bloku asemblera i liczbą wykonań bloku asemblera, oblicz całkowitą liczbę operacji zmiennoprzecinkowych (dodawań i mnożeń) w trakcie wykonania. ***W sprawozdaniu zamieść odpowiednie obliczenie liczby operacji zmiennoprzecinkowych oraz porównanie jej z wartością wynikającą z algorytmu i kodu źródłowego***
- **liczba rozkazów dostępu do danych** (**`perf stat`** i PAPI podają tylko liczbę odczytów, **`valgrind`** podaje niezależnie liczbę odczytów i liczbę zapisów): porównaj wartości raportowane przez programy profilujące z wartościami obliczonymi na podstawie postaci kodu asemblera (w konwencji AT&T używanej przez *gcc* pierwszym argumentem rozkazu jest źródło, a drugim cel) i liczby wykonań bloku asemblera (dostęp do danych realizowany jest jawnie przez rozkazy takie jak **`mov`** i **`broadcast`**, lub niejawnie – jeśli argumentem operacji arytmetycznej jest lokalizacja w pamięci). ***W sprawozdaniu zamieść odpowiednie obliczenia liczby operacji odczytu i zapisu oraz porównanie z wynikami profiler*ów**
  - w kodzie asemblera zaobserwuj ilu pobranym bajtom odpowiada każdy rozkaz dostępu (rozkazy kończące się na **`sd`** dotyczą 64 bitów, kończące się na **`pd`** 128, 256 lub 512 bitom, zależnie od typu rejestru z którym związana jest operacja). Oblicz ile (giga)bajtów danych pobieranych jest do rejestrów, a ile zapisywanych, w pojedynczym wykonaniu bloku asemblera oraz w trakcie całego wykonania obliczeń (zapisy i odczyty poza blokiem asemblera odpowiadającym najbardziej wewnętrznej

pętli implementacji stanowią pomijalnie mały udział). ***W sprawozdaniu zamieść odpowiednie obliczenia liczby odczytywanych do rejestrów i zapisywanych z rejestrów bajtów danych***

- **liczba chybień w `L1`** (podawana przez **`perf stat`** i **`valgrind`** **), liczba podmian linii w `L1`** (podawana przez PAPI - **`L1D.REPLACEMENT`**):
  - porównanie liczby chybień z liczbą operacji daje informacje, co który z dostępów kończy się chybieniem
    - przy parametrach pamięci podręcznych uzyskanych w poprzednich laboratoriach (m.in. szerokość linii 64 bajty) i rozmiarze danych podwójnej precyzji 8 bajtów, wczytanie pojedynczej, odpowiednio długiej, tablicy jednowymiarowej (a więc np. jednego wiersza macierzy przechowywanej wierszami) powinno powodować chybienie przy co ósmym (**skalarnym**) dostępie do pamięci (w przypadku dostępów wektorowych należy rozmiar linii podzielić przez liczbę bajtów w pojedynczej operacji dostępu)
  - liczba chybień w L1 (podmian linii) może służyć do obliczenia liczby bajtów transferowanych pomiędzy pamięcią L1 i pozostałymi poziomami pamięci – należy pomnożyć liczbę chybień przez rozmiar linii. ***W sprawozdaniu zamieść odpowiednie obliczenia liczby bajtów odczytywanych do L1 (analiza wyników*** **`valgrind`** ***wskazuje, że liczba danych zapisywanych z L1 jest, w analizowanych wersjach implementacji, pomijalnie mała)***
    - *w tabeli znajdują się osobno wiersze zakładające, że całość transferu odbywa się z L2 (tak byłoby w przypadku klasycznych* exclusive caches*) oraz że całość odbywa się z L3 (tak mogłoby być w szczególnych przypadkach dla mechanizmu* inclusive caches*, np.* victim caches*). Nie rozstrzygając, która z sytuacji ma miejsce (są jeszcze możliwe przypadki pośrednie, część podmian z L2, a część z L3, dokładna analiza wymagałaby badania większej liczby zdarzeń sprzętowych) należy w tabelce wypełnić oba wiersze. W sprawozdaniu, na podstawie analizy pozostałych zdarzeń sprzętowych, kodu źródłowego i kodu asemblera, można podjąć próbę zbadania jak wygląda rzeczywisty ruch danych pomiędzy poziomami pamięci podręcznej i jaki wpływ ma to na czas obliczeń i uzyskaną wydajność wykonania*

- **liczba linii alokowanych w pamięci L2**
  - raportowana tylko przez PAPI (**`L2_LINES_IN.ALL`**), służąca do szacowania wielkości transferu z pamięci L3 (po pomnożeniu przez rozmiar linii)
    - *podobnie jak dla liczby podmian linii w L1 źródłem danych umieszczanych w L2 może być L1 lub L3. Dla uproszczenia zakładane jest, że podmiana linii w L2 związana jest z transferem z L3 (nawet w przypadku* victim cache *może tak być, jeśli źródłem danych dla L2 jest L1, ale linia z L1 jest przenoszona na skutek transferu z L3)*
      - *w szczególności ostatni mechanizm (*victim cache*) może powodować, że liczba podmian linii w L1 i w L2 jest zbliżona (każde chybienie w L1 jest chybieniem w L2, co powoduje transfer z L3 do L1 i transfer podmienianej linii z L1 do L2)*

- **liczba chybień w L3**
  - *kwestia chybień w L3 jest zjawiskiem najbardziej złożonym, ze względu na porównywalność rozmiaru wszystkich tablic (dla małych zadań, takich jak badane w ramach laboratorium) z pojemnością L3 oraz kwestie strategii podmiany linii w L3 (problem komplikuje się jeszcze bardziej przy obliczeniach równoległych, szczególnie dla maszyny złożonej z dwóch mikroprocesorów, jak serwer Honorata)*
  - *wartości zwracane przez* ***`perf stat`*** *i* ***`valgrind`*** *mogą znacznie się różnić (*`valgrind` *może raportować więcej chybień niż* ***`perf stat`****), także zawodne są wyniki PAPI dotyczące zdarzeń związanych z chybieniami w L3 (można włączyć ich zliczanie w pliku* papi_set_user_events_mth.c*, pamiętając o ograniczonej liczbie liczników sprzętowych w procesorze (rdzeniu) – nowe zdarzenia muszą zastąpić zdarzenia zliczane dotychczas).*
  - **dla obu zwracanych (przez `perf stat` i `valgrind`) liczb chybień można obliczyć liczbę bajtów transferowanych do L3.**

*W sprawozdaniu zamieść odpowiednie obliczenia liczby bajtów odczytywanych do L3 (liczba bajtów zapisywanych będzie zawsze mało znacząca dla czasu wykonania kodu, także dla większych tablic)*

9. Wypełnij poniższą tabelę danymi uzyskanymi z pomiarów i parametrami obliczonymi na ich podstawie:

| | rozmiar pobranych danych [GB] | maksymalna szybkość transferu danych (lab 5) [GB/s] | minimalny czas transferu [s] | maksymalna wydajność [Gflop/s] |
|---|---|---|---|---|
| symbol metody np. ikj, rozmiar tablicy: np. 1080 i liczba operacji [Gflop] | | | | |
| zdarzenie i odpowiadający typ transferu danych, np.: | | | | |
| L1 loads ( L1 -> rejestry ) | | | | |
| L1 misses ( L2 -> L1 ) | | | | |
| L1 misses ( L3 -> L1 ) | | | | |
| L2 misses (line in) ( L3 -> L2 ) | | | | |
| L3 (LLC) misses – perf ( DRAM -> L3 ) | | | | |
| L3 (LLC) misses – valgrind (DRAM -> L3) | | | | |
| symbol metody np. cb_108, rozmiar tablicy: np. 1080 i liczba operacji [Gflop] | | | | |
| itd. | | | | |

uwagi:

- przeliczenia wartości na miary kilo, mega, giga należy dokonywać z mnożnikami 1024 (np. Gflop = flop/1024/1024/1024 )
- rozmiar pobranych danych należy obliczać:
  - dla transferu do rejestrów: na podstawie asemblera – liczby i typu rozkazów dostępu do danych (łącznie odczytów i zapisów),
  - dla transferu do L1, L2, L3: na podstawie liczby chybień (podmian linii) i długości linii w B
  - dla transferu do L3: dla wyników **`perf stat`** tylko odczyty, dla wyników **`valgrind`** łącznie odczyty i zapisy
- maksymalne szybkości transferu danych należy wziąć z wyników laboratorium 5

  [ *dla serwera Honorata powinny być zbliżone do wartości z tabeli - podanych dla częstotliwości zbliżonej do nominalnej, 2.2 GHz* ]
- czas transferu danych (obliczony jako stosunek rozmiaru danych do maksymalnej wydajności) jest minimalnym czasem, jaki musi zająć transfer do rejestrów lub określonej pamięci w trakcie wykonania programu – czas wykonania całości programu, ze względu na współbieżne działanie sprzętu nie jest sumą czasów składowych, jednak nie może być krótszy niż najdłuższy z czasów składowych
- maksymalna wydajność w ostatniej kolumnie [Gflop/s] może być obliczona jako iloraz liczby operacji i czasu transferu
  - maksymalną wydajność należy rozumieć jako ograniczenie wydajności spowodowane konkretnym typem transferu danych, czyli sytuacją kiedy program wykonuje rozmaite operacje, ale czas wykonania wynika z czasu realizacji tego transferu
  - wydajności wynikające z transferu danych należy także porównać z maksymalną wydajnością potoków przetwarzania (laboratorium 3 – wydajność dla obliczeń wektorowych – takich jak w asemblerze)

- wydajność rzeczywista nie może być wyższa od żadnej z wydajności składowych, a w przypadku kiedy wykonanie różnych transferów nie jest w pełni współbieżne (czas jest większy od maksymalnego czasu poszczególnych transferów) będzie jeszcze niższa
- tabela zakłada maksymalne szybkości transferu danych (uzyskane w laboratorium 5), w przypadku różnych implementacji rzeczywiste przepustowości mogą być niższe (dotyczy to np. wariantu *ijk*, gdzie dostęp do tablicy jest realizowany ze skokiem **n** elementów, co powoduje nie tylko znacznie większy rozmiar przekazywanych danych (chybienie przy każdym dostępie do tablicy **B**, ale także prawdopodobnie niższą przepustowość transferu do pamięci każdego z poziomów)

**Na podstawie tabeli wyciągnij wnioski jakie są czynniki ograniczające wydajność wykonania programu – czy jest on ograniczany przez wydajność potoków przetwarzania, czy przez wydajność pamięci, a w przypadku pamięci, który poziom pamięci jest decydujący. Spróbuj w analizie uwzględnić niedokładności pomiarów i ewentualne odstępstwa od poczynionych założeń, spróbuj uszeregować różne czynniki ograniczające wydajność programu pod kątem ich znaczenia – dla różnych badanych implementacji mnożenia macierz-macierz. Swoje analizy umieść w sprawozdaniu.**

10. Po wykonaniu wszystkich pomiarów i obliczeń, dokonaj porównania obu wariantów implementacji mnożenia macierz-macierz, z optymalizacją *cache blocking* i bez niej – na które parametry optymalizacja ma wpływ, a na które nie? **Porównanie i wynikające z niego wnioski umieść w sprawozdaniu**

Dalsze kroki:

11. Powtórz powyższe kroki dla wersji zwektoryzowanych oraz błędnej implementacji (*ijk*) powodującej nadmierną liczbę chybień (dla badanego rozmiaru tablic będzie to dotyczyć tylko chybień w L1, dla tablic większych także w L3)
12. Po wykonaniu wszystkich pomiarów i obliczeń, dokonaj porównania badanych wariantów implementacji mnożenia macierz-macierz – w jaki sposób przeprowadzane optymalizacje wpływają na parametry wykonania programu? **Porównanie i wynikające z niego wnioski umieść w sprawozdaniu**
13. **Przeprowadź badanie dla wersji wielowątkowej (za pomocą `perf stat` i PAPI) – czy w tej wersji zmieniają się czynniki ograniczające wydajność programu?**