Badanie wpływu organizacji dostępu do pamięci globalnej na efektywność przetwarzania - Nvidia CUDA

Sebastian Maciejewski 132275 i Jan Techner 132332
grupa I1, zajęcia we wtorki o 15:10, tygodnie nieparzyste,
email: maciejewski.torun@gmail.com

11 czerwca 2019

1 Wstęp

Sprawozdanie dotyczy zadania w wariancie 5. - badanie wpływu organizacji dostępu do pamięci globalnej (dostęp łączony i dostęp niełączony) na efektywność przetwarzania.
Warianty kodu:
- grid wieloblokowy, obliczenia przy wykorzystaniu pamięci globalnej,
- grid wieloblokowy, obliczenia przy wykorzystaniu pamięci współdzielonej bloku wątków.

Każdy z tych wariantów był wykorzystywany w dwóch wersjach - z łączonym oraz niełączonym dostępem do pamięci. Zastosowaliśmy dwa rozmiary bloków - 8x8 i 16x16, a pomiary pobraliśmy dla instancji o różnych wielkościach (128x128, 256x256, 512x512, 864x864, 1024x1024).
Wszystkie pomiary zostały wykonane na komputerach laboratoryjnych w sali 2.7.6. na karcie Nvidia GTX 260.

1.1 Specyfikacja sprzętu i wykorzystanego oprogramowania

- karta graficzna Nvidia GTX 260
  - compute capability - 1.3
  - liczba multiprocesorów - 27
  - maksymalna liczba bloków na jeden multiprocesor - 8
  - maksymalna liczba wątków w bloku - 512
  - warp size - 32
  - half-warp size - 16
- Visual Studio 2013
- Nvidia Visual Profiler

2 Analiza przygotowania eksperymentu

2.1 Kod wykorzystywany przy pomiarach

Obliczenia z wykorzystaniem pamięci globalnej

W pierwszym wariancie kodu do obliczeń wykorzystujemy pamięć globalną - fragmenty pamięci zarezerwowane na każdą z 3 macierzy za pomocą cudaMalloc. Każdy z wątków oblicza jeden element wynikowy dodając wartości w swoim lokalnym rejestrze (C_local), po czym zapisuje wynik takiego przetwarzania do tablicy w pamięci globalnej.

Obliczenia z wykorzystaniem pamięci współdzielonej bloku wątków

W drugim wariancie wykorzystujemy współdzieloną pamięć bloku wątków. Pewien obszar pamięci został zaalokowany na współdzielony obszar pamięci bloku wątków za pomocą __shared__. Dwa takie obszary, o rozmiarze 8x8 lub 16x16 (w zależności od pomiaru) odpowiadają za przechowywanie fragmentów mnożonej macierzy. Dla każdego bloku wątków dane są pobierane do jego pamięci współdzielonej, następnie każdy z wątków wylicza na ich podstawie swój wynik.
Taka organizacja przetwarzania mogłaby prowadzić do problemów z np. nadpisywaniem danych, więc konieczna jest synchronizacja - aby 'poczekać' na zakończenie przetwarzania przez wszystkie wątki, wywoływana jest funkcja __syncthreads. Po zakończeniu przetwarzania, wynik jest zapisywany w pamięci globalnej.

2.1.1 Rodzaje dostępu do pamięci

Dostęp do pamięci w kartach graficznych Nvidia odbywa się za pomocą transakcji o rozmiarach 32, 64 lub 128 bitów. W karcie na której przeprowadzany był eksperyment (CC 1.3) takie dostępy są realizowane dla tzw. half-wrapów - 16 wątkowych grup.
Różnica między dostępem łączonym a niełączonym sprowadza się do tego, że dostępem łączonym nazywamy dostęp do pamięci tych wątków half-wrapu, które sąsiadują ze sobą w pamięci. Dostęp w przeciwnym wypadku (wątki nie siąsiadują ze sobą w pamięci) nazywany jest dostępem niełączonym.

2.2 Istotne fragmenty kodu

Wariant G_L - dostęp łączony, pamięć globalna

```
1  template <int BLOCK_SIZE> __global__ void matrixMulCUDA(float *C, float *A,
2  float *B, int wA,
3  int wB) {
4  // Block index
5  int bx = blockIdx.x;
6  int by = blockIdx.y;
7
8  // Thread index
9  int tx = threadIdx.x;
10 int ty = threadIdx.y;
11
12 int row = by * blockDim.y + ty;
13 int col = bx * blockDim.x + tx;
14 float C_local = 0;
15
16 for (int k = 0; k < wA; k++) {
17   C_local += A[row*wA + k] * B[k *wA + col];
18 }
19
20 C[row*wA + col] = C_local;
21 }
```

Wariant G_NL - Dostęp niełączony, pamięć globalna

```
1  template <int BLOCK_SIZE> __global__ void matrixMulCUDA(float *C, float *A,
2  float *B, int wA,
3  int wB) {
4  // Block index
5  int bx = blockIdx.x;
6  int by = blockIdx.y;
7
8  // Thread index
9  int tx = threadIdx.x;
10 int ty = threadIdx.y;
11
12 int col= by * blockDim.y + ty;
13 int row = bx * blockDim.x + tx;
14 float C_local = 0;
15
16 for (int k = 0; k < wA; k++) {
17   C_local += A[row*wA + k] * B[k *wA + col];
18 }
19
20 C[row*wA + col] = C_local;
21 }
```

Wariant S_L - dostęp łączony, pamięć współdzielona bloku wątków

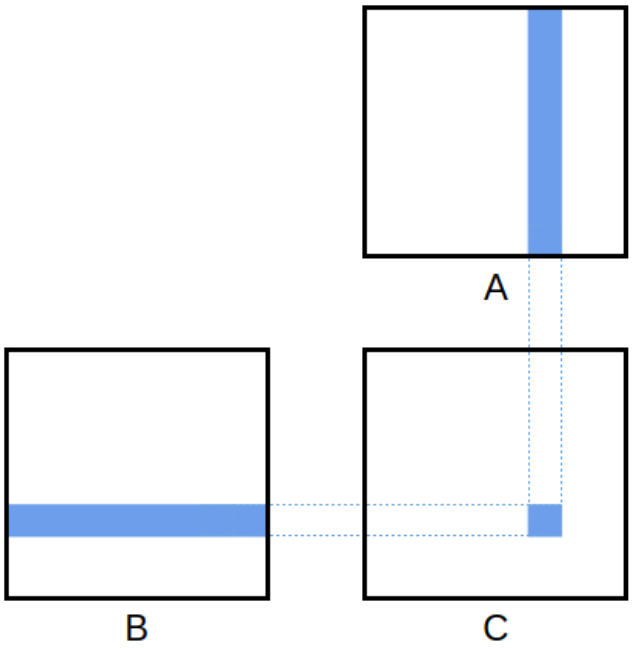
```
1  template <int BLOCK_SIZE> __global__ void matrixMulCUDA(float *C, float *A,
2  float *B, int wA,
3  int wB) {
4  // Block index
5  int bx = blockIdx.x;
6  int by = blockIdx.y;
7
8  // Thread index
9  int tx = threadIdx.x;
10 int ty = threadIdx.y;
11
12 int row = by * blockDim.y + ty;
13 int col = bx * blockDim.x + tx;
14 float C_local = 0;
15
16 __shared__ float matAhelp[BLOCK_SIZE][BLOCK_SIZE];
17 __shared__ float matBhelp[BLOCK_SIZE][BLOCK_SIZE];
18 for (int m = 0; m < wA / BLOCK_SIZE; ++m) {
19  matAhelp[tx][ty] = A[row * wA + m*BLOCK_SIZE + tx];
20  matBhelp[tx][ty] = B[(m*BLOCK_SIZE + ty)*wA + col];
21  __syncthreads();
22  for (int k = 0; k < BLOCK_SIZE; ++k)
23   C_local += matAhelp[tx][k] * matBhelp[k][ty];
24  __syncthreads();
25 }
26
27 C[row*wA + col] = C_local;
28 }
```

Wariant S_NL - dostęp niełączony, pamięć współdzielona bloku wątków

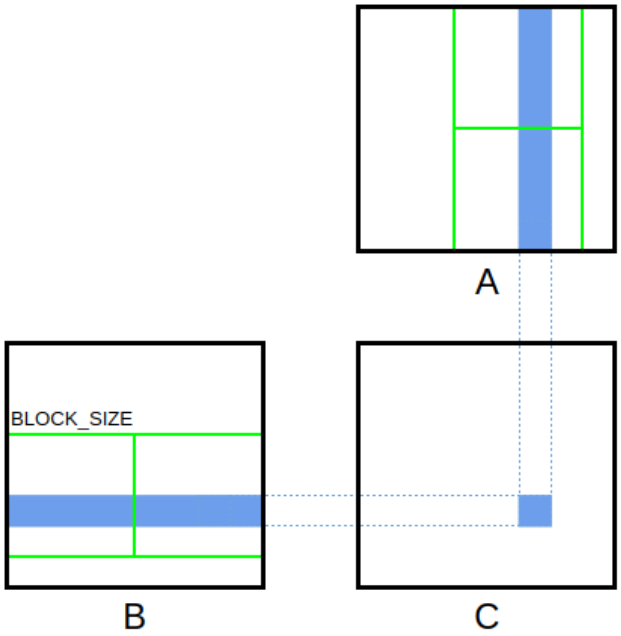
```
1  template <int BLOCK_SIZE> __global__ void matrixMulCUDA(float *C, float *A,
2  float *B, int wA,
3  int wB) {
4  // Block index
5  int bx = blockIdx.x;
6  int by = blockIdx.y;
7
8  // Thread index
9  int tx = threadIdx.x;
10 int ty = threadIdx.y;
11
12 int col = by * blockDim.y + ty;
13 int row = bx * blockDim.x + tx;
14 float C_local = 0;
15
16 __shared__ float matAhelp[BLOCK_SIZE][BLOCK_SIZE];
17 __shared__ float matBhelp[BLOCK_SIZE][BLOCK_SIZE];
18 for (int m = 0; m < wA / BLOCK_SIZE; ++m) {
19  matAhelp[tx][ty] = A[row * wA + m*BLOCK_SIZE + tx];
20  matBhelp[tx][ty] = B[(m*BLOCK_SIZE + ty)*wA + col];
21  __syncthreads();
22  for (int k = 0; k < BLOCK_SIZE; ++k)
23   C_local += matAhelp[tx][k] * matBhelp[k][ty];
24  __syncthreads();
25 }
26
27 C[row*wA + col] = C_local;
28 }
```

Wizualizacja przetwarzania

Na poniższych grafikach widać różnicę w procesie obliczania macierzy z wykorzystaniem pamięci globalnej i pamięci współdzielonej wątków.



Rysunek 1: Wykorzystanie pamięci globalnej



Rysunek 2: Wykorzystanie pamięci podręcznej, dostęp niełączony

W przypadku pamięci globalnej każdy wątek odczytuje jeden wiersz macierzy A i jedną kolumnę macierzy B, a następnie oblicza odpowiadający element macierzy C. Wszystkie dane odczytywane są bezpośrednio z pamięci globalnej, co powoduje bardzo dużą liczbę odwołań do pamięci globalnej.
W drugim przypadku, dla pamięci współdzielonej bloku wątków, każdy blok oblicza macierz będącą częścią macierzy wynikowej C o rozmiarze równym rozmiarowi bloku. Każdy wątek jest wówczas odpowiedzialny za obliczenie jednego elementu takiego wycinka macierzy. W taki sposób znacząco ograniczamy ilość dostępów do pamięci globalnej, ponieważ macierze A i B są odczytywane tylko $\frac{n}{BLOCK\_SIZE}$ razy, gdzie n to wielkość instancji a $BLOCK\_SIZE$ to wielokść bloku.

3 Analiza wyników eksperymentu pomiarowego

3.1 Miary efektywności

Prędkość obliczeń

$$\frac{ZL}{t} = \frac{2 * n^3}{t} \tag{1}$$

gdzie ZL to złożoność obliczeniowa, t to czas, a n to rozmiar instancji.

Przyspieszenie w stosunku do IKJ

$$\frac{t_{IKJ}}{CPR} \tag{2}$$

gdzie t_{IKJ} to czas przetwarzania IKJ a CPR to czas przetwarzania równoległego.

Przyspieszenie w funkcji wielkości instancji

$$\frac{CPI}{CP128} \tag{3}$$

gdzie CPI to czas przetwarzania dla instancji a $CPI128$ to czas przetwarzania dla instancji o rozmiarze 128, będący naszym punktem odniesienia - obliczamy przyspieszenie w stosunku do przetwarzania instancji o rozmiarze 128.

Miara stopnia łączenia dostępów do pamięci

Jest to miara obliczana w następujący sposób: dla każdego bloku pobierane będą dwie macierze (n^2), dodajemy do tego n^2 zapisów do pamięci globalnej i dzielimy przez sumę transakcji między pamięcią a multiprocesorem (odczytane z wyników z programu Visual Profiler).

Zajętość procesora

Wyliczana za pomocą kalkulatora zajętości SM miara - dla bloków o rozmiarze 8x8 osiąga 50%, zaś dla bloków o rozmiarze 16x16 osiąga 100%, czego można się było spodziewać. Pokazuje to, że pełnię możliwości tej karty wykorzystujemy wtedy, gdy bloki mają rozmiar 16x16.

CGMA - stosunek operacji do dostępów do pamięci

Jest to miara instensywności obliczeń. Zakładamy, że do obliczenia jednego elementu wynikowego potrzebujemy n danych z wiersza macierzy, n danych z kolumny macierzy oraz 1 operację zapisu do pamięci globalnej.
Warto zauważyć, że dla współdzielonej pamięci bloku wątków CGMA będzie bliski 1, ponieważ pomijamy operacje pobierania danych z pamięci globalnej do lokalnej.

GLD_EFFICIENCY

Jest to miara efektywności pobierania danych z pamięci globalnej obliczana w sposób następujący:

$$\frac{gld\_requests}{(gld\_32 + gld\_64 + gld\_128)/(2 * SM)} \tag{4}$$

gdzie SM to liczba multiprocesorów.

GST_EFFICIENCY

Jest to miara efektywności zapisu danych w pamięci globalnej. Oblicza się ją analogicznie do GLD_EFFICIENCY.

$$\frac{gst\_requests}{(gst\_32 + gst\_64 + gst\_128)/(2 * SM)} \tag{5}$$

gdzie SM to liczba multiprocesorów.

3.2 Wyniki pomiarów i obliczeń

V	BS	INST	T[ms]	GFLOPS	vs IKJ	ΔA	CMGA	JOIN	%	GLD_E	GST_E
G_L	8	128	0,17	25,32	12,07	1,00	0,0039	733,88	50%	-	141208
		256	1,10	30,63	2,74	6,61	0,0019	1512,50	50%	-	10904
		512	10,20	26,33	2,65	61,55	0,0010	3045,11	50%	-	591
		864	58,03	22,23	2,43	350,32	0,0006	5162,90	50%	-	59
		1024	87,08	24,66	5,87	525,70	0,0005	6121,00	50%	-	35
	16	128	0,12	33,95	16,19	1,00	0,0039	245,81	100%	-	282894
		256	1,03	32,58	2,91	8,34	0,0019	552,75	100%	-	17551
		512	7,80	34,41	3,46	63,14	0,0010	1136,81	100%	-	1141
		864	39,38	32,75	3,58	318,82	0,0006	1930,81	100%	-	133
		1024	71,83	29,90	7,11	581,49	0,0005	2286,30	100%	-	63
G_NL	8	128	0,41	10,11	4,82	1,00	0,0039	242,44	50%	-	77004
		256	3,47	9,66	0,86	8,37	0,0019	501,87	50%	-	4603
		512	44,66	6,01	0,60	107,67	0,0010	1012,71	50%	-	181
		864	204,65	6,30	0,69	493,37	0,0006	1718,62	50%	-	23
		1024	504,72	4,25	1,01	1216,81	0,0005	2037,98	50%	-	8
	16	128	0,81	5,15	2,45	1,00	0,0039	29,13	100%	-	41510
		256	6,52	5,15	0,46	8,00	0,0019	65,27	100%	-	2595
		512	75,20	3,57	0,36	92,30	0,0010	134,00	100%	-	112
		864	388,92	3,32	0,36	477,38	0,0006	227,41	100%	-	13
		1024	990,65	2,17	0,52	1215,98	0,0005	269,23	100%	-	4
S_L	8	128	0,13	33,10	15,78	1,00	1	3879,09	50%	-	1112210
		256	0,86	38,81	3,47	6,82	1	8493,26	50%	-	82978
		512	6,85	39,20	3,94	54,03	1	17661,66	50%	-	5254
		864	54,47	23,68	2,59	429,82	1	25701,09	50%	-	330
		1024	32,73	65,60	15,61	258,29	1	42639,10	50%	-	652
	16	128	0,24	17,17	8,19	1,00	1	2352,76	100%	-	2155375
		256	1,61	20,78	1,86	6,61	1	6582,70	100%	-	170279
		512	12,32	21,79	2,19	50,45	1	15496,47	100%	-	11433
		864	59,10	21,83	2,39	241,97	1	27996,80	100%	-	1426
		1024	98,48	21,81	5,19	403,22	1	33641,27	100%	-	724
S_NL	8	128	0,13	31,95	15,24	1,00	1	1429,14	50%	-	1726347
		256	0,87	38,68	3,46	6,61	1	3247,42	50%	-	132073
		512	6,86	39,14	3,94	52,25	1	6899,09	50%	-	8390
		864	32,77	39,36	4,30	249,63	1	11969,10	50%	-	1041
		1024	60,00	35,79	8,52	457,02	1	14265,87	50%	-	487
	16	128	0,25	16,64	7,93	1,00	1	297,64	100%	-	2118706
		256	1,65	20,39	1,82	6,53	1	810,56	100%	-	164427
		512	12,36	21,72	2,18	49,04	1	1871,40	100%	-	10961
		864	59,12	21,82	2,39	234,54	1	3348,45	100%	-	1358
		1024	98,54	21,79	5,19	390,94	1	4014,01	100%	-	688

Tablica 1: Miary efektywności dla 6 pętli, przetwarzanie równolegle

Nazwy kolumn:

- V - wersja kodu
- BS - rozmiar bloku
- INST - rozmiar instancji (długość boku macierzy)
- T[ms] - czas przetwarzania w ms
- GFLOPS - prędkość przetwarzania
- vs IKJ - przyspieszenie w stosunku do IJK
- ΔA - zmiana przyspieszenia w funkcji wielkości instancji

- JOIN - miara stopnia łączenia dostępów do pamięci globalnej
- % - zajętość multiprocesora
- GLD_E - gld_efficiency
- GST_E - gst_efficiency

Nazwy wersji:

- G_L - wykorzystanie pamięci globalnej, dostęp łączony
- G_NL - wykorzystanie pamięci globalnej, dostęp niełączony
- S_L - wykorzystanie pamięci współdzielonej bloku wątków, dostęp łączony
- S_NL - wykorzystanie pamięci współdzielonej bloku wątków, dostęp niełączony

4 Wnioski

Przy analizie wyników przeprowadzonego eksperymentu można na pierwszy rzut oka zauważyć pewne prawidłowości. Dwie z nich to: zajętość procesora odpowiednio 50% dla bloków 8x8 i 100% dla bloków 16x16 oraz CGMA o wartości 1 dla wszystkich pomiarów dla kodu ze współdzieloną pamięcią bloku wątków. Obie obserwacje były łatwe do przewidzenia, bowiem wiążą się one z charakterystyką wykorzystanych miar efektywności - zajętość multiprocesora obliczyliśmy w kalkulatorze zajętości, a CGMA, jako miara instensywności obliczeń przyjmuje wartoci bardzo małe, gdy pobieramy każdorazowo dane z pamięci globalnej, lub bliskie (lub równe) 1, gdy korzystamy z pamięci współdzielonej bloku wątków, czego również można się było spodziewać.

Biorąc pod uwagę prędkość przetwarzania, można zauważyć, że pozostaje ona raczej stała. Maleje tylko dla dostępów niełączonych do pamięci globalnej w wariancie G_NL. Jest to najmniej korzystny wariant pod względem przetwarzania, ponieważ oprócz tego, że występuje bardzo dużo odwołań do pamięci globalnej, to nie występuje zjawisko łączenia dostępów do pamięci dla sąsiednich wątków w half-warpie. Dodatkowo można zauważyć, że prędkość przetwarzania jest wyższa dla rozmiaru bloku 8x8 niż dla rozmiaru bloku 16x16 dla wszystkich wariantów kodu oprócz wersji G_L, gdzie występują łączone dostępy do pamięci globalnej. Jest to zachowanie nieoczekiwane, ponieważ dla bloków 8x8 zajętość procesora wynosi tylko 50%, a w przypadku bloków o rozmiarze 16x16 zajętość wynosi 100%.

Pomiary czasu przetwarzania dla różnych wersji kodu dały rezultaty zgodne z oczekiwaniami: czas działania dla wersji kodu, gdzie wykorzystane były pamięci współdzielone bloku wątków, były znacząco krótsze od tych z dostępem globalnym. Dla największych instancji z dostępem niełączonym czas przetwarzania jest 10 razy krótszy dla wersji z pamięcią współdzieloną. Wynika to z efektywnego zastosowania pamięci współdzielonych dla bloków wątków, które w znacznym stopniu ograniczają kosztowne czasowo odwołania do pamięci globalnej karty graficznej. Pewną anomalią są krótsze czasy przetwarzania dla wersji kodu G_L dla rozmiaru bloku 16x16 w porównaniu do wersji S_L (blok 16x16).

Wyniki przyspieszenia względem sekwencyjnego algorytmu mnożenia macierzy IKJ pokazują, że wykorzystanie karty graficzniej rzeczywiście przyspiesza obliczenia (nawet 15-krotnie dla najmniejszych instancji). Jednocześnie nie jest zaskoczeniem, że przyspieszenie wariantów S_L i S_NL, wykorzystujących pamięć współdzieloną, jest większe, niż tych z dostępem globalnym. Jedynym przypadkiem gdzie algorytm sekwencyjny uzyskuje lepsze rezultaty jest najmniej korzystny wariant G_NL, co jest potwiedzeniem wysokiego kosztu niełączonych odwołań wątków do pamięci.

Wpływ wykorzystania pamięci współdzielonej można również zaobserwować w wynikach obliczeń przyspieszenia względem rozmiaru instancji. Wyniki dla wariantów S_L i S_NL są zauważalnie lepsze - wraz ze wzrostem instancji czas przetwarzania nie wzrasta tak bardzo, jak w przypadku wariantów G_L i G_NL. Można zatem wysnuć wniosek, iż wykorzystanie współdzielonej pamięci ma szczególnie pozytywny wpływ na czas przetwarzania w przypadku, gdy rozmiar instancji rośnie.

Kolejna wyliczona przez nas miara, to stopień łączenia dostępów do pamięci globalnej, wyliczony dla wszystkich multiprocesorów na karcie graficznej.

Bierzemy tutaj pod uwagę wszystkie odwołania do pamięci globalnej, które składają się z: pobranie dwóch tablic o rozmiarach n x n przez kazdy w bloków oraz n x n zapisów wyników do macierzy wynikowej. W rzeczywistości

jednak każdy blok nie potrzebuje całych macierzy, a jedynie ich fragmenty o rozmiarach 8x8 lub 16x16, w zależności od rozmiaru bloku. Ilość transakcji multiprocesor-pamięć została odczytana z miar gst total (obliczonej jako suma gst_32, gst_64 i gst_128) oraz sumy gld_32, gld_64 oraz gld_128. Poprawność obliczonych wyników stoi pod znakiem zapytania, ze względu na zerowe wartości wszytskich miar określających ilość zapisów do pamięci globalnej (gld_*), otrzymanych z Visual Profilera. W swoich rozważaniach pominiemy interpretację otrzymanych wartości, ponieważ może ona nie mieć sensu ze względu na błędne wyniki oraz prowadzić do błędnych wniosków.

Ze względu na wyżej opisane, niezależne od nas błędy pomiarów, niemożliwe było obliczenie liczby odczytów danych z pamięci globalnej. Dlatego też miara GLD_E, określająca efektywność pobrań danych z pamięci globalnej, pozostaje niezdefiniowana.

Miara GST_E, obrazująca efektywność zapisów do pamięci, pokazuje, że wraz ze wzrostem rozmiaru instancji zapisy do pamięci są coraz mniej efektywne. Niemniej jednak, efektywność zapisu dla wariantów S_L i S_NL jest większa, niż dla wariantów z globalnym dostępem do pamięci.

5 Podsumowanie

Po przeprowadzeniu eksperymentu i dogłębnej analizie wyników nasuwają się następujące wnioski - przetwarzanie z wykorzystaniem pamięci współdzielonej bloku wątków jest na ogół znacznie bardziej efektywne niż obliczenia wykorzystujące bezpośrednie odwołania do pamięci globalnej. Dodatkowo znaczący wpływ ma takie ustawienie kolejności przetwarzania, żeby wykorzystywać mechanizm łączenia transakcji podczas dostępów do pamięci.