CA3DMM - raport

Opis rozwiązania

- Wyliczam (sprawdzając wszystkie możliwości) p_n, p_m, p_k minimalizujące wartość wyrażenia $p_n mk + np_m k + nmp_k$ (koszt komunikacji) takie, że $p_n | p_m$ lub $p_m | p_n$. Spośród wszystkich rozwiązań o tej samej wartości $p_n mk + np_m k + nmp_k$ wybieram takie, dla którego wartość $p_n p_m p_k$ (liczba aktywnych procesów) jest maksymalna.
- Procesy o rangach mniejszych równych $p_n p_m p_k$ kończą działanie. Pozostałe dzielą się na p_k grup rozmiaru $p_n p_m$ (plastry prostopadłościanu). Dla każdego plastra tworzony jest oddzielny komunikator.
- Plaster i-ty jest odpowiedzialny za wymnożenie i-tego przedziału kolumn macierzy A - A_i z i-tym przedziałem wierszy macierzy B - B_i. Niech $C_i = A_i B_i$ - wynik obliczeń i-tego plastra.
- Każdy plaster dzieli się na $s = \frac{max(p_n, p_m)}{min(p_n, p_m)}$ grup Cannona. Dla każdej grupy tworzony jest oddzielny komunikator. Dla odpowiadających sobie procesów (o tych samych współrzędnych (x, y) w grupie) w różnych grupach tworzony jest oddzielny komunikator.
- Procesy w plastrze dzielą się generowaniem macierzy A_i i B_i.
    - Jeśli $p_n < p_m$, to generowanie macierzy A_i jest podzielone równomiernie między wszystkie procesy i-tego plastra, natomiast generowaniem macierzy B_i zajmują się tylko procesy z pierwszej grupy Cannona, a następnie każdy proces z pierwszej grupy przekazuje swój wygenerowany kawałek macierzy B_i odpowiadającym sobie procesom w pozostałych grupach przy użyciu MPI_Broadcast.
    - Jeśli $p_n \geq p_m$, to generowanie macierzy B_i jest podzielone równomiernie między wszystkie procesy i-tego plastra, natomiast generowaniem macierzy A_i zajmują się tylko procesy z pierwszej grupy Cannona, a następnie każdy proces z pierwszej grupy przekazuje swój wygenerowany kawałek macierzy A_i odpowiadającym sobie procesom w pozostałych grupach przy użyciu MPI_Broadcast.
- Każda grupa Cannona wykonuje mnożenie posiadanych przez siebie fragmentów A_i i B_i (w zależności od tego, czy $p_n < p_m$ jest to albo fragment macierzy A_i i cała macierz B_i albo fragment macierzy A_i i cała macierz B_i).
    - Na początku fragmenty A_i i B_i są rozdystrybuowane między wszystkie procesy w grupie w taki sposób, że proces o współrzędnych w grupie (x, y) posiada blok (x, y) odpowiednio A_i i B_i.

- Przygotowanie do wykonania algorytmu Cannona polega na przesunięciu każdego bloku (x, y) fragmentu B_i o y pozycji w górę (cyklicznie) i każdego bloku (x, y) fragmentu A_i o x pozycji w lewo (cyklicznie). W tym celu proces o współrzędnych (x, y) w grupie komunikuje się z procesami o współrzędnych $(x - 1, y)$, $(x + 1, y)$, $(x, y - 1)$, $(x, y + 1)$ przy użyciu komunikatorów łączących procesy o tej samej współrzędnej x i y w obrębie tej samej grupy. Komunikacja odbywa się nieblokująco, przy użyciu MPI_Isend i MPI_Irecv, przed przejściem do wykonania algorytmu oczekujemy na zakończenie komunikacji przy użyciu MPI_Waitall.
- Fragment macierzy C_i, za którego wyliczenie jest odpowiedzialny proces jest wypełniany zerami.
- Następnie s razy wykonywany jest krok algorytmu Cannona:
  - Proces o współrzędnych (x, y) w grupie zleca wysłanie swojego fragmentu macierzy A_i procesowi o współrzędnych $(x, y - 1)$ oraz wysłanie swojego fragmentu macierzy B_i procesowi o współrzędnych $(x - 1, y)$ przy użyciu MPI_Isend.
  - Proces o współrzędnych (x, y) w grupie zleca odebranie nowego fragmentu macierzy A_i od procesu o współrzędnych $(x, y + 1)$ oraz odebranie nowego fragmentu macierzy B_i od procesu o współrzędnych $(x + 1, y)$ do pomocniczych buforów A', B'.
  - Proces wymnaża swoje fragmenty A_i i B_i i dodaje wynik do swojego fragmentu C_i.
  - Przed przejściem do następnego kroku oczekujemy na zakończenie komunikacji przy użyciu MPI_Waitall. Nowe fragmenty macierzy A_i i B_i potrzebne do kolejnego kroku znajdują się w pomocniczych buforach A', B'.
- Na koniec proces o współrzędnych (x, y) w plastrze zawiera blok (x, y) macierzy C_i obliczanej przez ten plaster.

- Łatwo widać, że wynikowa macierz $C = AB$ będzie sumą macierzy C_i po wszystkich plastrach. Aby macierz C znalazła się w całości w pierwszym plastrze, wykonujemy MPI_Reduce używając komunikatorów łączących odpowiadające sobie procesy w różnych plastrach (o tych samych współrzędnych (x, y) w plastrze).
- Na koniec proces o współrzędnych (x, y) w pierwszym plastrze zawiera blok (x, y) macierzy C.
  - Jeśli użyta została opcja -ge ge_value, każdy proces w pierwszym plastrze zlicza ile jest elementów większych lub równych ge_value w jego fragmencie macierzy C, następnie wynik jest sumowany przy użyciu MPI_Reduce i wypisywany przez pierwszy proces.
  - Jeśli użyta została opcja -v, wszystkie procesy w pierwszym plastrze wysyłają przy użyciu MPI_Send kolejne wiersze swojego fragmentu macierzy C procesowi pierwszemu, który te wiersze w odpowiedniej kolejności odbiera i wypisuje.

Optymalizacje

- Oddzielne komunikatory dla:
    a. każdego plastra,
    b. każdej grupy Cannona,
    c. wierszy i kolumn w grupie Cannona,
    d. odpowiadających sobie procesów w różnych grupach Cannona w tym samym plastrze,
    e. odpowiadających sobie procesów w różnych plastrach.
- Użycie MPI_Broadcast i (d) do przekazania macierzy B_i lub A_i z pierwszej grupy Cannona do pozostałych.
- Komunikacja asynchroniczna w algorytmie Cannona: dodatkowy bufor na otrzymywanie nowego fragmentu macierzy A_i i B_i pozwala na nakładanie na siebie komunikacji i obliczeń.
- Użycie MPI_Reduce i (e) do uzyskania wynikowej macierzy C w pierwszym plastrze.
- Użycie MPI_Reduce i (a) do uzyskania wyniku do wypisania w pierwszym procesie w przypadku użycia opcji -ge ge_value.

Intensywność numeryczna

W mojej implementacji proces wykonuje $s \cdot \lceil \frac{m}{p_m} \rceil \cdot \lceil \frac{n}{p_n} \rceil \cdot \lceil \frac{d_i}{s} \rceil$ mnożeń, gdzie

- $s = min(p_n, p_m)$ - liczba kroków algorytmu Cannona,
- d_i- liczba kolumn macierzy A_i (wierszy macierzy B_i) wymnażanych przez plaster tego procesu, $d_k = \lceil \frac{k}{p_k} \rceil$ lub $d_k = \lfloor \frac{k}{p_k} \rfloor$, $\sum_{i=1}^{p_k} d_i = k$

i tyle samo dodawań.
Łącznie wykonywanych jest N mnożeń, gdzie $nmk \leq N < (n + p_n)(m + p_m)(k + p_k s)$ i tyle samo dodawań.
Każdy proces wysyła i otrzymuje $s \cdot \left(\lceil \frac{m}{p_m} \rceil \cdot \lceil \frac{d_i}{s} \rceil + \lceil \frac{n}{p_n} \rceil \cdot \lceil \frac{d_i}{s} \rceil \right) \cdot 8B$
Łącznie przesyłanych jest $M \cdot 8B$, gdzie
$p_m nk + p_n mk \leq M < p_m(n + p_n)(k + p_k s) + p_n(m + p_m)(k + p_k s)$
Jest to problem intensywny obliczeniowo.

Silne i słabe skalowanie

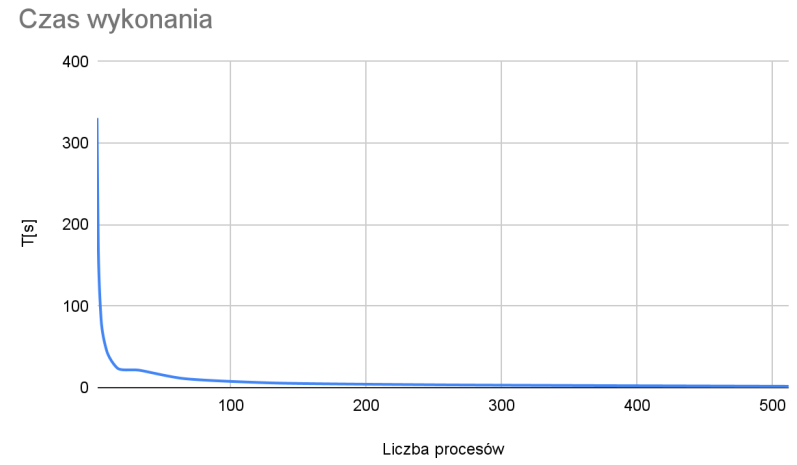
W obu przypadkach pomiary wykonane zostały dla 10 mnożeń, aby zamortyzować koszt inicjalizacji programu MPI, z opcją -g, ponieważ wypisywanie całej macierzy nie jest zoptymalizowane. Pomiary zostały wykonane na Okeanosie.

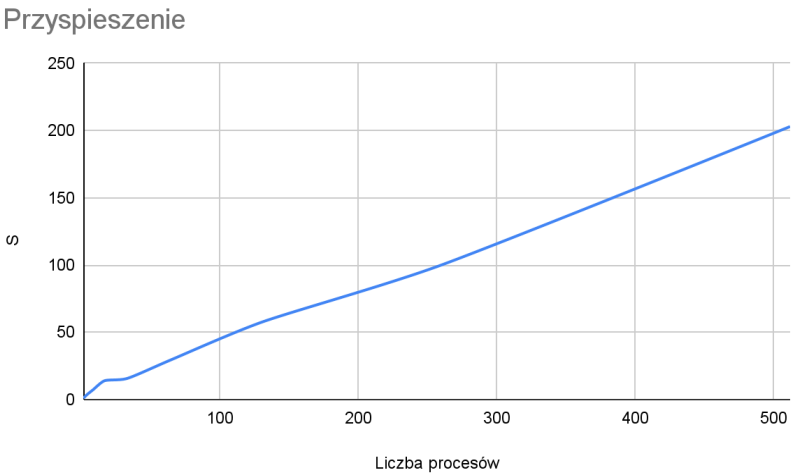
Silne skalowanie

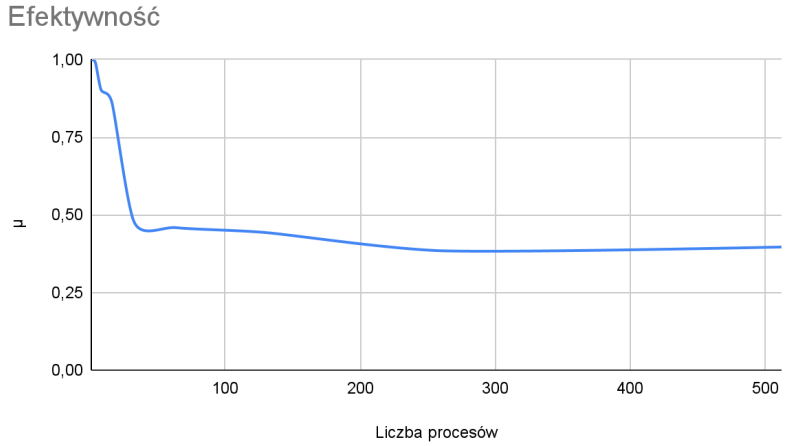
W przypadku silnego skalowania pomiar wykonywany jest dla instancji tej samej wielkości, dla różnej liczby procesów.

```
mpiexec -n <liczba procesów> ./ca3dmm 2000 2000 2000 -s
23,29,31,37,41,43,47,53,59,61 -g 8196
```

Liczba procesów	Czas [s]	Przyspieszenie	Efektywność
1	330,801	1	1
2	165,54	1,998314607	0,9991573034
4	83,497	3,961830964	0,990457741
8	45,717	7,235842247	0,9044802809
16	23,898	13,84220437	0,865137773
32	21,359	15,48766328	0,4839894775
64	11,268	29,35756124	0,4587118943
128	5,824	56,79962225	0,4437470488
256	3,356	98,57002384	0,3850391556
512	1,63	202,9453988	0,396377732







Przyspieszenie jest wprost proporcjonalne do liczby procesów, co było do przewidzenia, ponieważ problem jest intensywny obliczeniowo. Podwojenie liczby procesów powoduje dwukrotne zmniejszenie czasu wykonania.
Wraz ze wzrostem liczby procesów efektywność spada, co jest spowodowane rosnącymi kosztami komunikacji.

Słabe skalowanie

W przypadku słabego skalowania pomiar wykonywany jest dla instancji problemu wielkości wprost proporcjonalnej do liczby procesów ($nmk \sim P$).

```
mpiexec -n <liczba procesów> ./ca3dmm <n> <m> <k> -s
23,29,31,37,41,43,47,53,59,61 -g 8196
```

n	m	k	nmk	Liczba procesów	Czas [s]	nmk/P
1000	1000	1000	1000000000	1	41.441	1000000000
2000	1000	1000	2000000000	2	41.732	1000000000
2000	2000	1000	4000000000	4	41.851	1000000000
2000	2000	2000	8000000000	8	45.671	1000000000
4000	2000	2000	16000000000	16	47.025	1000000000
4000	4000	2000	32000000000	32	85.478	1000000000
4000	4000	4000	64000000000	64	89.772	1000000000
8000	4000	4000	128000000000	128	90.198	1000000000
8000	8000	4000	256000000000	256	90.169	1000000000
8000	8000	8000	512000000000	512	90.528	1000000000

Na początku problem skaluje się liniowo (czas wykonania dla 2 razy większej instancji z 2 razy większą liczbą procesorów jest porównywalny). Potem obserwujemy, podobnie jak w przypadku silnego skalowania, znaczny spadek wydajności między 16 a 32 procesami. Może to być spowodowane rosnącym kosztem komunikacji.