

Algorytmy równoległe 2015 (zad. 1)

Michał Liszcz

2015-10-21

Contents

1	Wstęp	2
2	Analiza problemu	2
3	Metoda różnic skończonych	2
3.1	Dyskretyzacja dziedziny	2
3.2	Dyskretyzacja równania	3
3.3	Warunki brzegowe	3
4	Algorytm sekwencyjny	4
5	Algorytm równoległy	4
5.1	Partitioning	5
5.2	Communication	5
5.3	Agglomeration	5
5.4	Mapping	5
5.5	Opis algorytmu	6
6	Analiza wyników	7
6.1	Metryki podstawowe - pomiar czasu	7
6.2	Metryki podstawowe - pomiar przyspieszenia i efektywności	9
6.3	Metryki skalowalne	10
6.4	Duża liczba procesorów	11

1 Wstęp

Zaproponować algorytm równoległy wyliczający kolejne położenia drgającej membrany rozpiętej na kwadracie o ustalonym boku. Boki membrany są sztywno zamocowane (warunki brzegowe). Należy ustalić położenie początkowe i prędkość $\left(\frac{\partial p}{\partial t}\right)_{t=0}$ (warunki początkowe).

Zastosować metodę różnicową do równania:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} - \frac{\rho}{T} \frac{\partial^2 p}{\partial t^2} = 0 \quad (1)$$

gdzie $p(x, y)$ - położenie punktu membrany, ρ - gęstość powierzchniowa, T - napięcie membrany.

2 Analiza problemu

Równanie (1) to klasyczne równanie falowe. Podstawiając $\frac{\rho}{T} := (c^2)^{-1}$, można zapisać je w standardowej postaci:

$$[\partial_{tt} - c^2 \nabla^2] p(t, x, y) = 0 \quad (2)$$

Rozwiązania poszukujemy w obszarze Ω :

$$\begin{aligned} \Omega &= [t_{\min}, t_{\max}] \times [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \\ W &= [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \end{aligned} \quad (3)$$

Zadane są następujące warunki brzegowe:

$$p(t, x, y) = 0 \quad \forall t \in [t_{\min}, t_{\max}], \forall (x, y) \in \partial W \quad (4)$$

Oraz warunki początkowe (membrana jest w pozycji $P(x, y)$ i porusza się z prędkością $S(x, y)$):

$$\left. \begin{aligned} p(0, x, y) &= P(x, y) \\ p_t(0, x, y) &= S(x, y) \end{aligned} \right\} \quad \forall (x, y) \in W \quad (5)$$

3 Metoda różnic skończonych

Poszukujemy rozwiązania numerycznego metodą różnic skończonych.

3.1 Dyskretyzacja dziedziny

W obszarze Ω wprowadzamy siatkę dyskretnych punktów:

$$\left. \begin{aligned} \Delta t &= \frac{t_{\max} - t_{\min}}{K} \\ \Delta x &= \frac{x_{\max} - x_{\min}}{N} \\ \Delta y &= \frac{y_{\max} - y_{\min}}{M} \end{aligned} \right\} \quad (6)$$

$$\left. \begin{aligned} t_k &= x_{\min} + k\Delta t, & k &= 0, 1, \dots, K \\ x_n &= x_{\min} + n\Delta x, & n &= 0, 1, \dots, N \\ y_m &= y_{\min} + m\Delta y, & m &= 0, 1, \dots, M \end{aligned} \right\} \quad (7)$$

Oznaczamy wartość p w punktach siatki:

$$p(t_k, x_n, y_m) = p_{n,m}^k \quad (8)$$

3.2 Dyskretyzacja równania

Operatory różniczkowe występujące w równaniu zastępujemy operatorami różnicowymi. Dla pochodnych pierwszego rzędu zapisujemy różnicę centralną (średnią z ilorazów różnicowych “w przód” i “w tył”), natomiast pochodne drugiego rzędu otrzymujemy po odjęciu stronami rozwinięć $p(x)$ w szereg Taylora wokół x_0 , kładąc w nich $x = x_0 \pm \Delta x$. Wyprowadzenia poniższych przybliżeń można znaleźć w literaturze [1].

$$\left. \begin{aligned} \partial_t p(t_k, x_n, y_m) &\approx \frac{p_{n,m}^{k-1} - p_{n,m}^{k+1}}{2\Delta t} &:= D_t p_{n,m}^k \\ \partial_{tt} p(t_k, x_n, y_m) &\approx \frac{p_{n,m}^{k-1} - 2p_{n,m}^k + p_{n,m}^{k+1}}{(\Delta t)^2} &:= D_{tt} p_{n,m}^k \\ \partial_{xx} p(t_k, x_n, y_m) &\approx \frac{p_{n-1,m}^k - 2p_{n,m}^k + p_{n+1,m}^k}{(\Delta x)^2} &:= D_{xx} p_{n,m}^k \\ \partial_{yy} p(t_k, x_n, y_m) &\approx \frac{p_{n,m-1}^k - 2p_{n,m}^k + p_{n,m+1}^k}{(\Delta y)^2} &:= D_{yy} p_{n,m}^k \end{aligned} \right\} \quad (9)$$

Równanie (1) przyjmuje postać równania różnicowego:

$$\frac{p_{n,m}^{k-1} - 2p_{n,m}^k + p_{n,m}^{k+1}}{(\Delta t)^2} = c^2 \left(\frac{p_{n-1,m}^k - 2p_{n,m}^k + p_{n+1,m}^k}{(\Delta x)^2} + \frac{p_{n,m-1}^k - 2p_{n,m}^k + p_{n,m+1}^k}{(\Delta y)^2} \right) \quad (10)$$

Poszukujemy wartości p w chwili $k+1$, zakładając że znane jest całe rozwiązanie w chwilach poprzednich:

$$p_{n,m}^{k+1} = 2p_{n,m}^k - p_{n,m}^{k-1} + (\Delta t)^2 c^2 (D_{xx} + D_{yy}) p_{n,m}^k \quad (11)$$

3.3 Warunki brzegowe

Równanie (4) prowadzi do następujących warunków brzegowych:

$$p_{0,m}^k = p_{N,m}^k = p_{n,0}^k = p_{n,M}^k = 0 \quad \forall k, n, m \quad (12)$$

Warunki początkowe (5) są zadane przez odwzorowania P i S :

$$\begin{aligned} p_{n,m}^0 &= P_{n,m} \\ D_t p_{n,m}^0 &= S_{n,m} \end{aligned} \quad (13)$$

Drugie z powyższych równań rozpisujemy korzystając z definicji operatora D_t , kładziemy $k=0$ w (11), a następnie eliminujemy ujemny czas, łącząc ze sobą te dwa równania:

$$\begin{aligned} p_{n,m}^{-1} - p_{n,m}^1 &= 2\Delta t S_{n,m} \\ p_{n,m}^1 &= 2p_{n,m}^0 - p_{n,m}^{-1} + (\Delta t)^2 c^2 (D_{xx} + D_{yy}) p_{n,m}^0 \\ p_{n,m}^1 &= p_{n,m}^0 - \Delta t S_{n,m} + \frac{1}{2} (\Delta t)^2 c^2 (D_{xx} + D_{yy}) p_{n,m}^0 \end{aligned} \quad (14)$$

4 Algorytm sekwencyjny

Algorytm sekwencyjny operuje na trójwymiarowej tablicy liczb zawierającej wartości p w trójkach (k, n, m) .

W pierwszej kolejności ustawiane są wartości dla $k = 0$, zgodnie z (13). Następnie dla $k = 1$, przy użyciu (14). Pozostała część tablicy uzupełniana jest na podstawie zadanego równania różnicowego (11).

W celu sprawdzenia poprawności rozwiązania, uruchomiłem program z następującymi parametrami:

$$\begin{aligned} t_{\min} &= x_{\min} = y_{\min} = 0 \\ t_{\max} &= x_{\max} = y_{\max} = 30 \\ K &= 100, \quad N, M = 30 \\ c &= 1 \end{aligned} \tag{15}$$

Przyjąłem nieznaczące początkowe zaburzenie na środku membrany:

$$\begin{aligned} P_{n,m} &= 5 \quad \forall (n, m) \in \{13, 14, 15, 16, 17\}^2 \\ P_{15,15} &= 7 \end{aligned} \tag{16}$$

W pozostałych punktach membrana jest w stanie równowagi: $P_{n,m} = 0$. W każdym punkcie membrana początkowo spoczywa ($S_{n,m} = 0 \quad \forall n, m$).

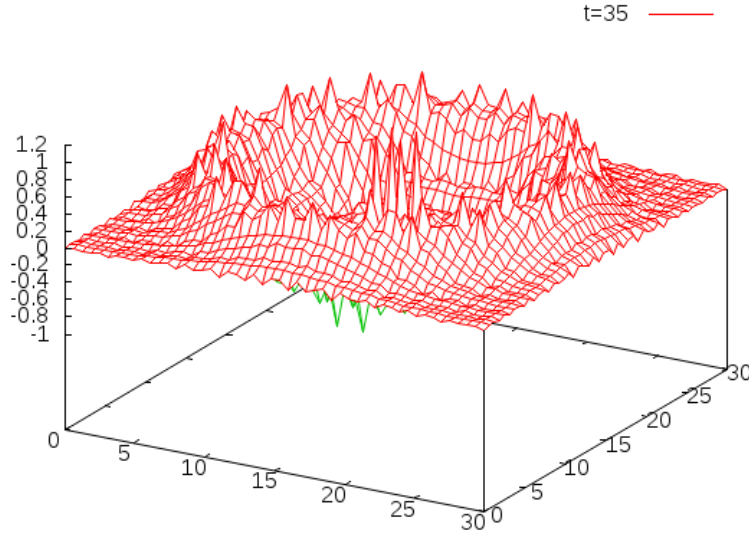


Figure 1: Wizualizacja fali rozchodzącej się w membranie.

Otrzymany wynik jest zgodny z przewidywaniami. Do sprawozdania dołączona jest animacja przedstawiająca propagację fali w czasie.

5 Algorytm równoległy

W dalszej części zostanie przedstawiony algorytm równoległy zgodny z metodologią PCAM.

Algorytm sekwencyjny uzupełniał trójwymiarową tablicę *warstwami*, kolejne iteracje były parametryzowane zmienną czasową (parametr k). W każdej iteracji generowana była dwuwymiarowa tablica reprezentująca wartości w punktach siatki w ustalonej chwili t_k . Można ją utożsamiać z obszarem W gdzie zdefiniowano problem (3). Kolejne punkty opisują próbę efektywnego zrównoleglenia tego algorytmu.

5.1 Partitioning

Ze względu na model problemu, najlepiej dokonać tutaj dekompozycji domenowej, poprzez podzielenie *danych* na porcje, które przetwarzane będą równolegle.

Najmniejszym, niepodzielnym zadaniem jest obliczenie pojedynczego elementu z trójwymiarowej tablicy $p_{n,m}^k$. Takich elementów jest $K \times N \times M$.

Dekompozycja funkcjonalna nie ma tutaj zastosowania - jest tylko jeden rodzaj operacji.

5.2 Communication

Stosując dekompozycję zaproponowaną w poprzednim punkcie, można łatwo określić wymagania dotyczące komunikacji. Siatka użyta do dyskretyzacji przestrzeni narzuca strukturę komunikacyjną. Jest to komunikacja lokalna - do obliczenia wartości komórki $p_{n,m}^{k+1}$ należy znać wartości:

$$p_{n,m}^k, \quad p_{n,m}^{k-1}, \quad p_{n-1,m}^k, \quad p_{n+1,m}^k, \quad p_{n,m-1}^k, \quad p_{n,m+1}^k \quad (17)$$

Daje to sześć wymian komunikatów dla każdej komórki.

5.3 Agglomeration

Przedstawiony w poprzednich punktach sposób podziału zadań i wynikający z niego schemat komunikacji jest bardzo nieefektywny.

W typowych zastosowaniach ilość zadań będzie kilka rzędów wielkości większa od liczby procesorów. Pojedyncze zadanie jest bardzo proste - składa się z kilku operacji dodawania i mnożenia.

Należy pogrupować zadania tak, by były wykonywane w sposób najbardziej efektywny na maszynie wyposażonej w kilkanaście procesorów.

W pierwszej kolejności zakładamy, że dane będziemy dzielić względem *przestrzeni*, to znaczy, że najmniejszą porcją danych z trójwymiarowej tablicy $p_{n,m}^k$ będzie zbiór komórek o ustalonych indeksach n i m , natomiast k będzie dowolny:

$$E_{n,m} = \{p_{n,m}^k : k = 0, 1, \dots, K\} \quad (18)$$

Algorytm równoległy będzie działał iteracyjnie względem *czasu*, podzielonego na K iteracji. W każdej iteracji zostanie wyliczona wartość jednej komórki $p_{n,m}^k$.

Porcja danych $E_{n,m}$ to jednowymiarowa tablica. Daje to mniej zadań - $N \times M$. Dodatkowo, wyliczenie pojedynczej komórki z E wymaga już tylko czterech aktów komunikacji.

5.4 Mapping

Zadania $E_{n,m}$ należy przypisać do fizycznych procesorów, na których będą wykonywane. Zadania mają identyczny *rozmiar* - można więc podzielić je równo na Z procesorów, pamiętając o wymaganiach komunikacyjnych (17). Jeden procesor powinien obsługiwać zadania sąsiadujące ze sobą przestrzennie - o kolejnych indeksach n i m .

Optymalnym sposobem podziału jest przydzielenie pojedynczemu procesorowi kilku całych, sąsiednich *wierszy* (ciągły obszar pamięci) ze zbioru $\{E_{n,m}\}$.

Niech Q_n oznacza zbiór zadań $E_{n,m}$ w n -tym wierszu przestrzeni:

$$Q_n = \{E_{n,m} : m = 0, 1, \dots, M\} \quad (19)$$

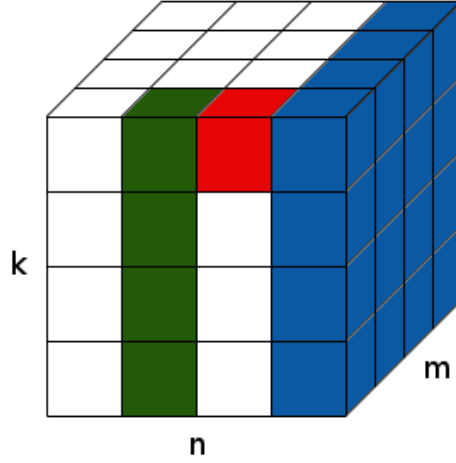


Figure 2: Podział siatki na zadania. Odpowiednimi kolorami oznaczono zadania: $p_{n,m}^k$, $E_{n,m}$, Q_n .

Przyjmujemy następujący podział zadań między procesory:

$$\begin{aligned} Z_1 &= \{Q_0, Q_1, \dots, Q_{|Z_1|-1}\} \\ Z_2 &= \{Q_{|Z_1|+0}, Q_{|Z_1|+1}, \dots, Q_{|Z_1|+|Z_2|-1}\} \\ &\dots \end{aligned} \quad (20)$$

W zależności od mocy obliczeniowej procesorów, podział może być dokonany na nierówne części. W testowanym przypadku każdy z procesorów otrzymał taką samą liczbę zadań.

5.5 Opis algorytmu

W algorytmie równoległym każdy procesor będzie wykonywał w pętli następujące operacje:

1. pobranie od sąsiednich procesorów informacji o wartościach obliczonych w poprzednim kroku na brzegach ich obszarów,
2. przekazanie sąsiadom informacji o wartościach obliczonych w poprzednim kroku na brzegu swojego obszaru,
3. wyliczenie wartości dla aktualnego kroku na całym obsługiwanym obszarze.

Należy uważać na zakleszczenie w punktach 1. i 2.. Dwa procesory obsługujące sąsiadujące porcje danych mogą wzajemnie czekać na dane z poprzedniej iteracji sąsiedniego procesora. Istnieje kilka rozwiązań: można przykładowo wymusić ustaloną kolejność operacji 1. i 2. w komunikujących się procesorach, lub użyć komunikacji asynchronicznej.

W implementacji z wykorzystaniem MPI użyta zostanie funkcja `MPI_Sendrecv`, która w abstrakcyjny sposób ukrywa detale komunikacji asynchronicznej (`MPI_Isend` oraz `MPI_Irecv`).

6 Analiza wyników

Proponowany algorytm równoległy został przetestowany na klastrze Zeus w ACK Cyfronet. Testy obejmowały pomiary standardowych metryk programów równoległych: przyspieszenia, efektywności oraz oszacowania wpływu komunikacji na wydajność.

6.1 Metryki podstawowe - pomiar czasu

Pierwszy etap testów obejmował pomiar czasu, przyspieszenia i efektywności w wersji podstawowej. Testy przeprowadziłem osobno dla siatek o rozmiarach 100x300x300, 100x600x600, 100x1200x1200 oraz 100x2400x2400. W dalszej części sprawozdania n oznacza *rozmiar problemu* - w tym przypadku liczbę punktów siatki.

Liczbę procesorów zmieniałem w zakresie 1-12. *W przyszłości planowane są testy na klastrze o większej liczbie procesorów.*

Dla każdej ustalonej liczby procesorów p zmierzyłem czas $T(n, p)$ wykonania programu. Pomiar powtórzyłem czterokrotnie a następnie uśredniłem. Jako niepewność pomiaru $u(T(n, p))$ przyjąłem odchylenie standardowe średniej z uzyskanych wartości. Wyniki uzyskane dla dwóch pierwszych siatek to bardzo krótkie czasy i są obarczone dużymi niepewnościami pomiarowymi, dlatego pomijam je w dyskusji wyników.

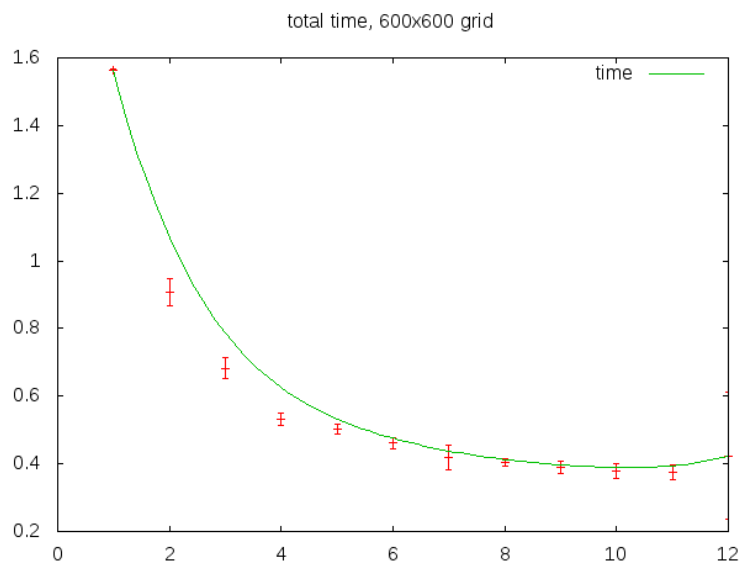


Figure 3: Czas wykonania programu w funkcji liczby procesorów dla siatki 100x600x600

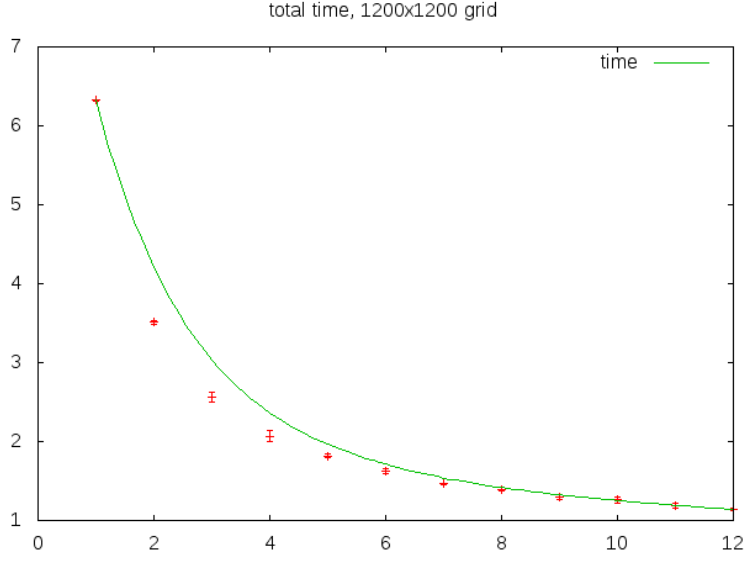


Figure 4: Czas wykonania programu w funkcji liczby procesorów dla siatki 100x1200x1200

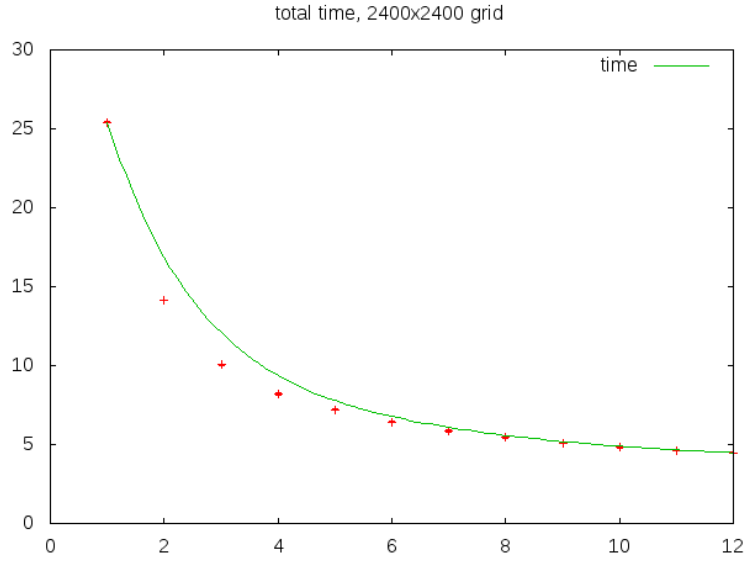


Figure 5: Czas wykonania programu w funkcji liczby procesorów dla siatki 100x2400x2400

Wykresy 3, 4 oraz 5 prezentują gładką krzywą, malejącą ze wzrostem liczby procesorów. Można spróbować “dopasować” do tych danych funkcję:

$$T(n, p) = \frac{A(n)}{p} + B(n) \quad (21)$$

Stała B wynika z istnienia części sekwencyjnej - wykresy nie zbiegają do 0.

Zmierzone wartości czasu wydają się być sensowne i zgodne z oczekiwaniami.

6.2 Metryki podstawowe - pomiar przyspieszenia i efektywności

Wykorzystałem następujące definicje przyspieszenia $S(n, p)$ i efektywności $E(n, p)$:

$$S(n, p) = \frac{T(n, 1)}{T(n, p)} \quad (22)$$

$$E(n, p) = \frac{S(n, p)}{p} \quad (23)$$

Niepewności oszacowałem metodą różniczki zupełnej.

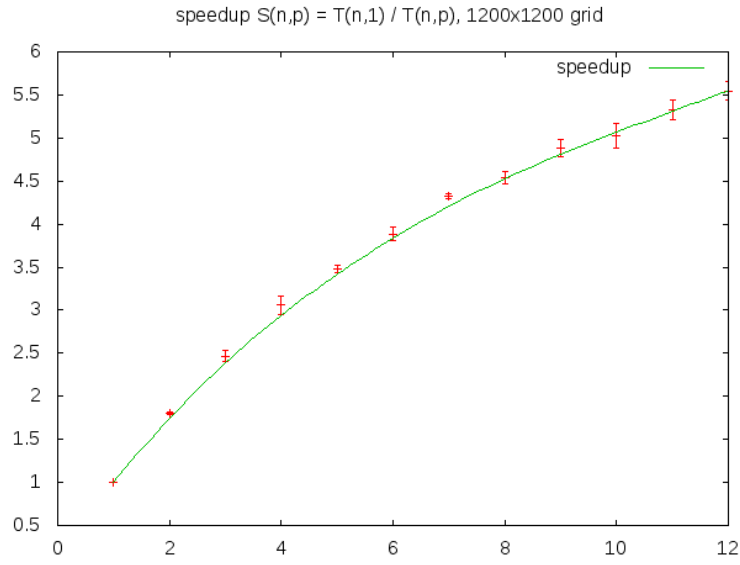


Figure 6: Przyspieszenie w funkcji liczby procesorów dla siatki 100x1200x1200

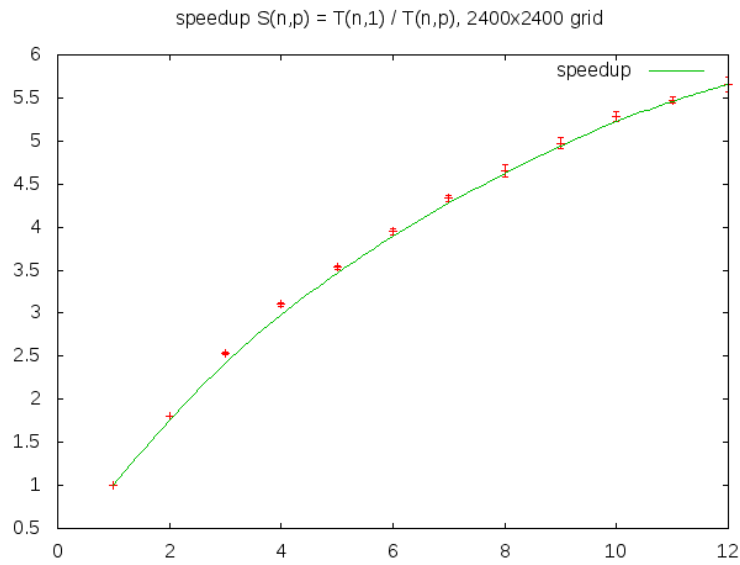


Figure 7: Przyspieszenie w funkcji liczby procesorów dla siatki 100x2400x2400

Z wykresów 6 i 7 widać, że udało się uzyskać przyspieszenie (około dwukrotne dla dwóch procesorów i pięciokrotne dla 12 procesorów).

Wykresy 8 i 9 przedstawiają obliczoną efektywność algorytmu. Efektywność maleje w przybliżeniu liniowo ze wzrostem liczby procesorów.

Dodatkowy komentarz wymaga przeprowadzenia testów dla większej liczby procesorów przy tych samych rozmiarach siatki, aby lepiej określić zachowanie funkcji przyspieszenia i efektywności dla dużych p .

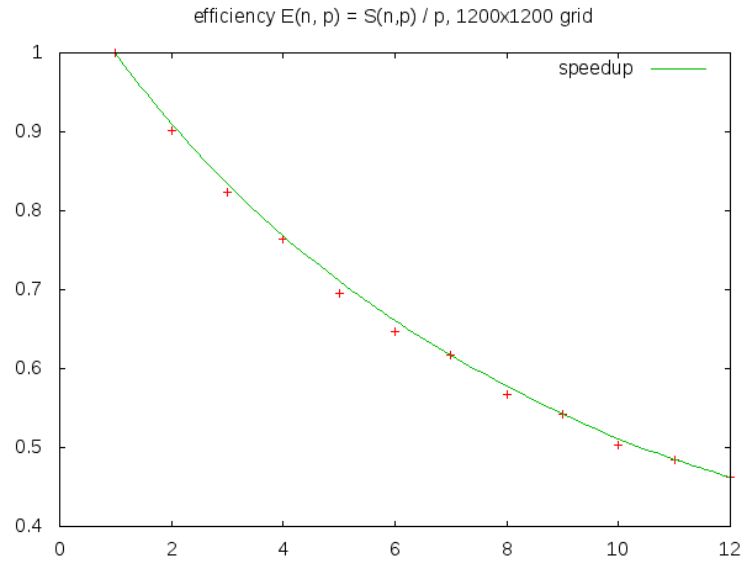


Figure 8: Efektywność w funkcji liczby procesorów dla siatki 100x1200x1200

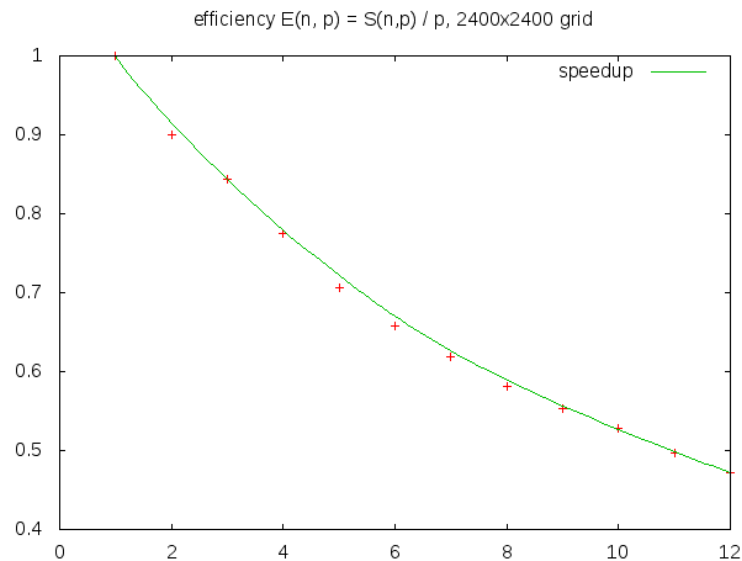


Figure 9: Efektywność w funkcji liczby procesorów dla siatki 100x2400x2400

6.3 Metryki skalowalne

uzupełnić

6.4 Duża liczba procesorów

uzupełnić

References

- [1] P. Frey, M. De Buchan, *The numerical simulation of complex PDE problems*, http://www.ann.jussieu.fr/frey/cours/UdC/ma691/ma691_ch6.pdf, 2008.
- [2] Hans Petter Langtangen, *Finite difference methods for wave motion*, http://hplgit.github.io/INF5620/doc/pub/main_wave.pdf, 2013.
- [3] Ian Foster, *Designing and Building Parallel Programs*, www.mcs.anl.gov/~itf/dbpp/.
- [4] Knut-Andreas Lie, *The Wave Equation in 1D and 2D*, <http://www.uio.no/studier/emner/matnat/ifi/INF2340/v05/foiler/sim04.pdf>, 2005.
- [5] https://en.wikipedia.org/wiki/Finite_difference.