Algorytmy równoległe 2015 (zad. 1)

Michał Liszcz

2015-10-28

Contents

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:

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

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

2 Analiza problemu

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

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

Rozwiązania poszukujemy w obszarze Ω:

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

Zadane są następujące warunki brzegowe:

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

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

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

3 Metoda różnic skończonych

Poszukujemy rozwiązania numerycznego metodą różnic skończonych.

3.1 Dyskretyzacja dziedziny

W obszarze Ω wprowadzamy siatkę dyskretnych punktów:

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

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

Oznaczamy wartość p w punktach siatki:

$$p(t_k, x_n, y_m) = p^k_{n,m} \tag{8}$$

3.2 Dyskretyzacja równania

Operatory różniczkowe występujące w równaniu zastępujemy operatorami różnicowymi. Dla pochodnych pierwszego rzędu zapisujemy różnicę centralną (średnią z ilorazów różnicowyh "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].

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

Równanie (1) przyjmuje postać równania różnicowego:

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

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

$$p^{k+1}_{n,m} = 2p^k_{n,m} - p^{k-1}_{n,m} + (\Delta t)^2 c^2 (D_{xx} + D_{yy}) p^k_{n,m} \tag{11}$$

3.3 Warunki brzegowe

Równanie (4) prowadzi do następujących warunków brzegowych:

$$p^k_{0,m} = p^k_{N,m} = p^k_{n,0} = p^k_{n,M} = 0 \qquad \forall k, n, m \tag{12}$$

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

$$\begin{aligned} p^0_{n,m} &= P_{n,m} \\ D_t p^0_{n,m} &= S_{n.m} \end{aligned} \tag{13}$$

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:

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

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:

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

Przyjąłem nieznaczne początkowe zaburzenie na środku membrany:

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

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

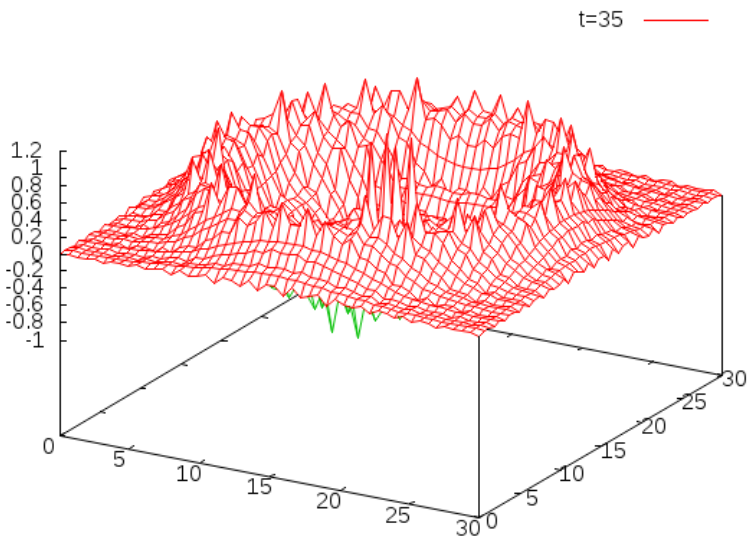

Figure 1: Wizualizacja fali rozchodzącej się w membranie.

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 prztwarzane będą równolegle.

Najmniejszym, niepodzielnym zadaniem jest obliczenie pojednycznego elementu z trójwymiarowej tablicy $p^k_{n,m}$. 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 $p^{k+1}_{n,m}$ należy znać wartości:

$$p^k_{n,m}, \qquad p^{k-1}_{n,m}, \qquad p^k_{n-1,m}, \qquad p^k_{n+1,m}, \qquad p^k_{n,m-1}, \qquad p^k_{n,m+1} \tag{17}$$

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 $p^k_{n,m}$ będzie zbiór komórek o ustalonych indeksach n i m, natomiast k będzie dowolny:

$$E_{n,m} = \left\{ p^k_{n,m} : k = 0, 1, ..., K \right\} \tag{18}$$

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

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 ideksach 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:

$$Q_n = \{E_{n,m} : m = 0, 1, ..., M\} \tag{19}$$

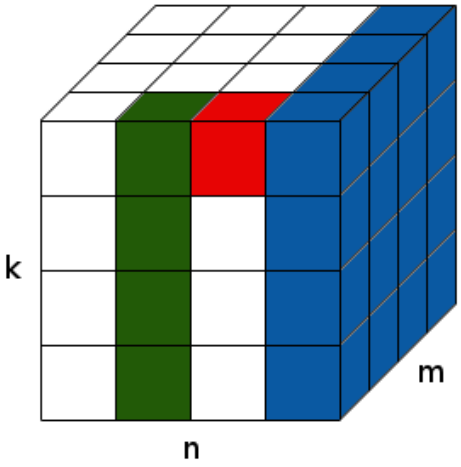

Figure 2: Podział siatki na zadania. Odpowiednimi kolorami oznaczono zadania: $p^k_{n,m}$, $E_{n,m}$, Q_n.

Przyjmujemy następujący podział zadań między procesory:

$$\begin{aligned} Z_1 &= \{Q_0, Q_1, ..., Q_{|Z_1|-1}\} \\ Z_2 &= \{Q_{|Z_1|+0}, Q_{|Z_1|+1}, ..., Q_{|Z_1|+|Z_2|-1}\} \\ &... \end{aligned} \tag{20}$$

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 komunikujacych 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.

Pomiary wykonałem z wykorzystaniem implementacji MPICH 3.0.4.

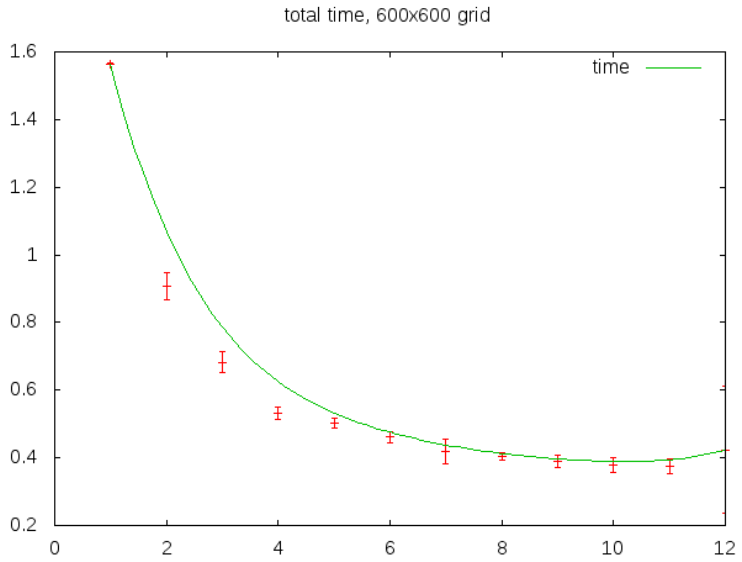

Figure 3: Czas wykonania programu w funkcji liczby procesorów dla siatki 100x600x600

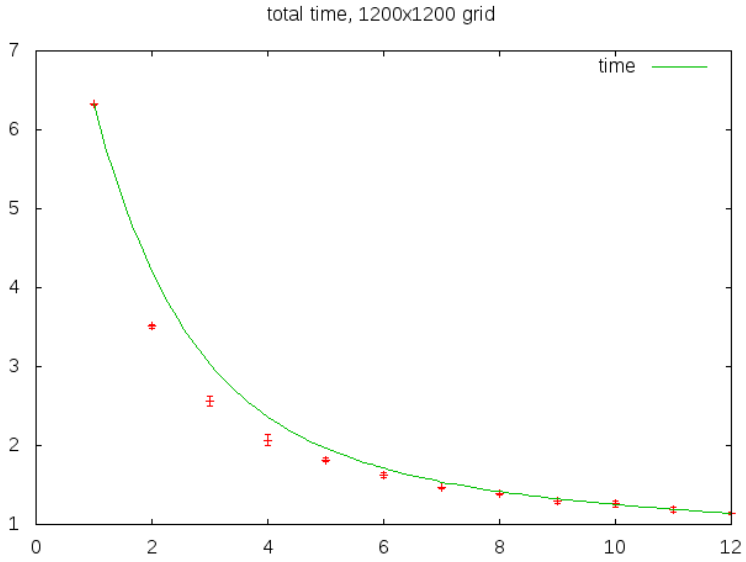

Figure 4: Czas wykonania programu w funkcji liczby procesorów dla siatki 100x1200x1200

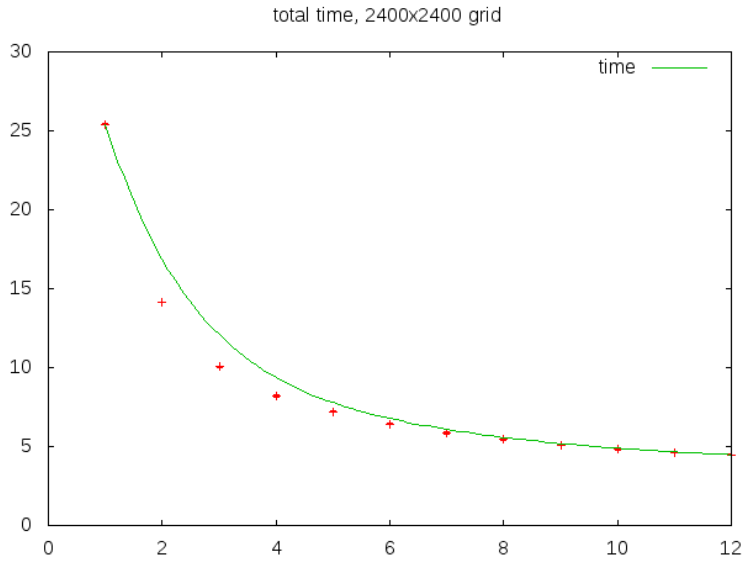

Figure 5: Czas wykonania programu w funkcji liczby procesorów dla siatki 100x2400x2400

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

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

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 $E(n,p)$:

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

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

Niepewności oszacowałem metodą różniczki zupełnej.

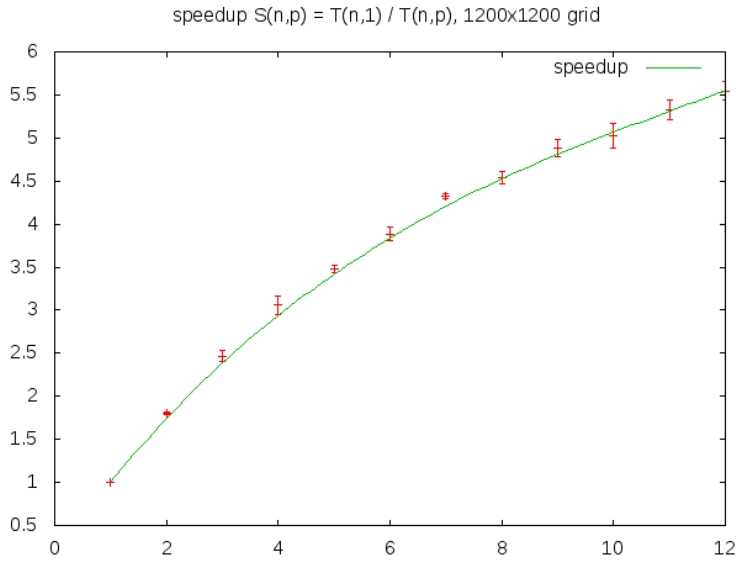

Figure 6: Przyspieszenie w funkcji liczby procesorów dla siatki 100x1200x1200

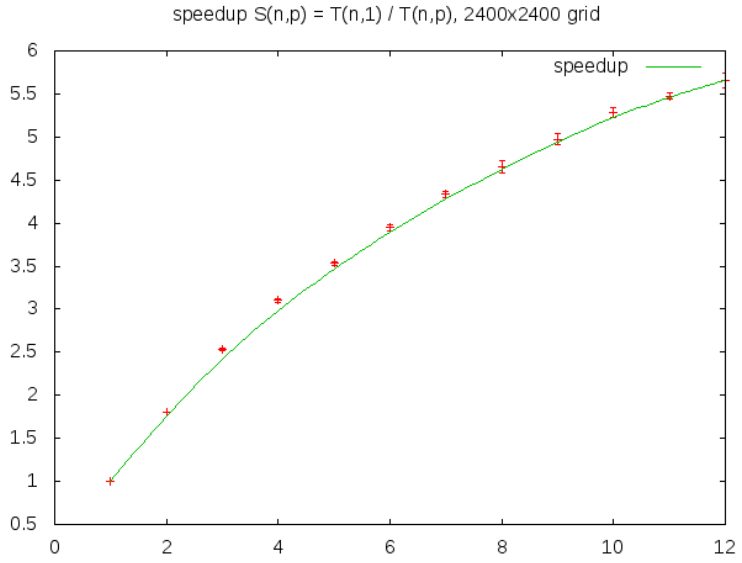

Figure 7: Przyspieszenie w funkcji liczby procesorów dla siatki 100x2400x2400

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 efekywność 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.

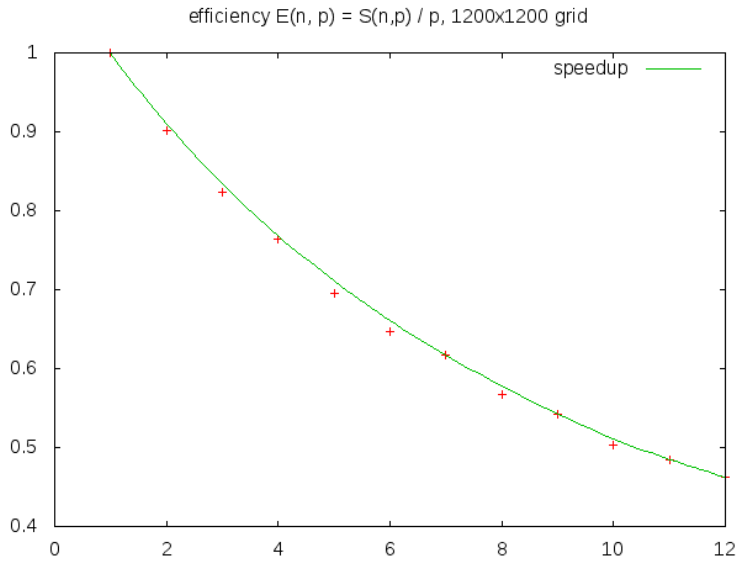

Figure 8: Efektywność w funkcji liczby procesorów dla siatki 100x1200x1200

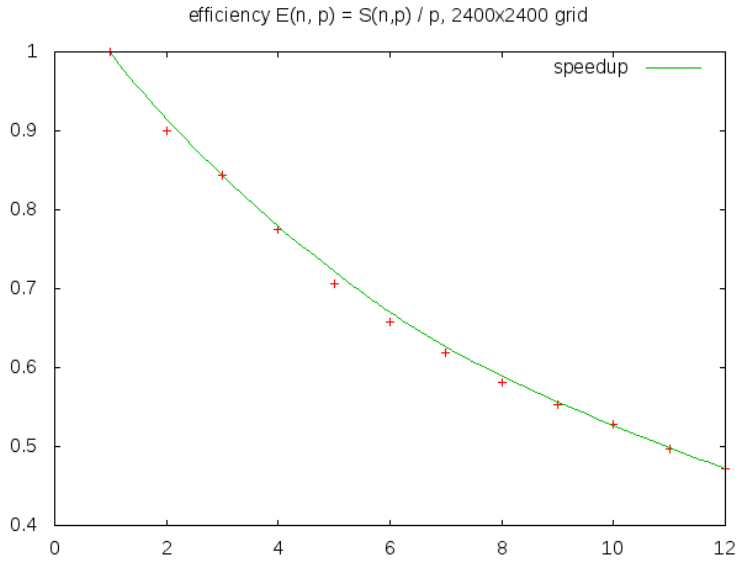

Figure 9: Efektywność w funkcji liczby procesorów dla siatki 100x2400x2400

6.3 Duża liczba procesorów

Testy dla większej ilości procesorów przeprowadziłem wykorzystując cztery węzły 12-procesorowe. Charakterystyki są zgodne z tymi przedstawionymi w poprzednim punkcie.

W testach zmniejszyłem zakres czasowy do $K = 50$, natomiast równanie rozwiązywałem na większej przestrzennie siatce $N = M = 3600$.

Testy przeprowadziłem z wykorzystaniem implementacji OpenMPI 1.8.1.

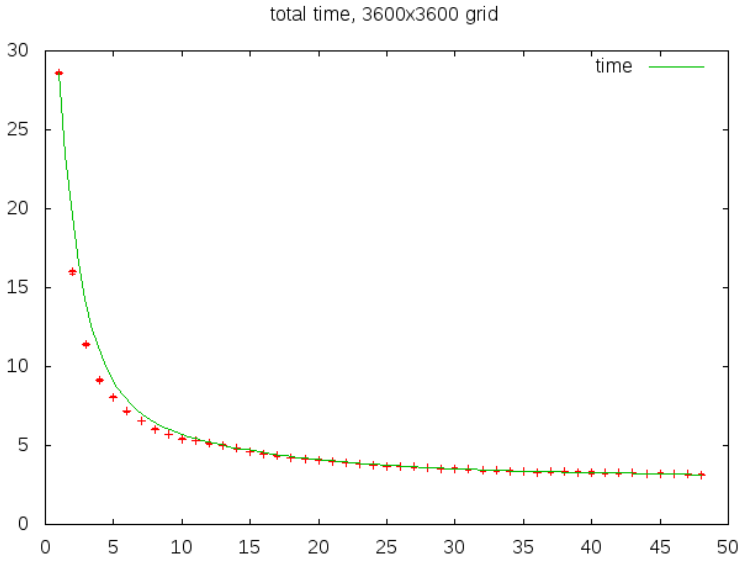

Figure 10: Całkowity czas w funkcji liczby procesorów dla siatki 50x3600x3600

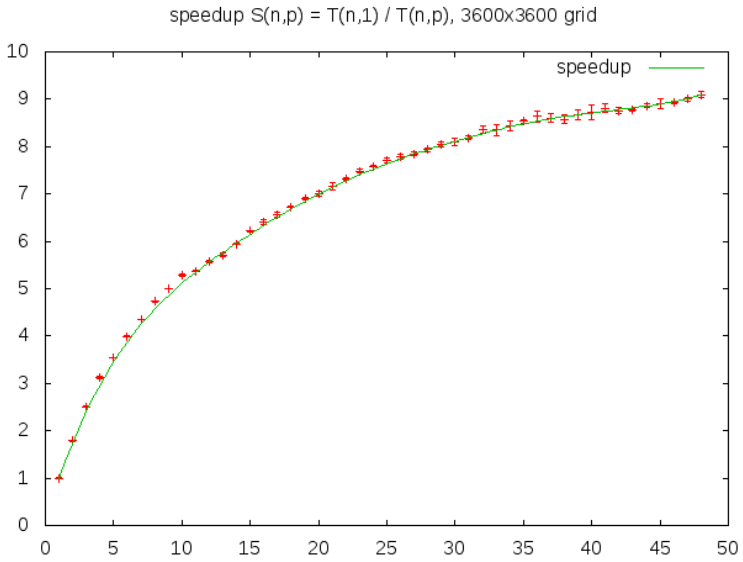

Figure 11: Przyspieszenie w funkcji liczby procesorów dla siatki 50x3600x3600

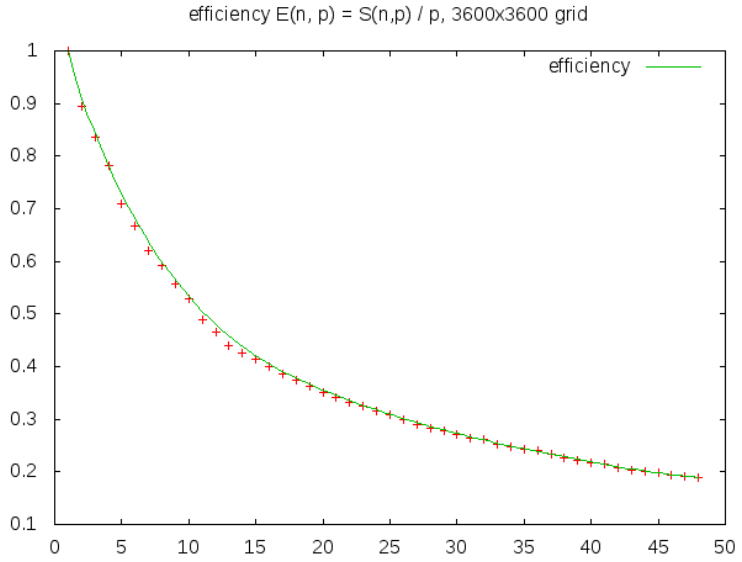

Figure 12: Efektywność w funkcji liczby procesorów dla siatki 50x3600x3600

7 Dyskusja wyników

Efektywność programu spada ze wzrostem liczby procesorów. Można próbować uzasadnić kształ krzywej otrzymanej przykładowo na rysunku 10 (zależność czasu przetwarzania T od liczby procesorów p).

Spróbujmy oszacować taki czas. Jest $K = 50$ identycznych iteracji - czas T będzie więc liniową funkcją K, z zerowym wyrazem wolnym. W każdej iteracji procesor operuje na danych o rozmiarze $N \cdot \frac{N}{p}$. Załóżmy że wypełnienie jednej komórki zajmuje mu czas t. Dodatkowo, w każdej iteracji procesor wymienia dane o rozmiarze $N \cdot 1$ z dwoma sąsiadami. Zajmuje to czas s. Całkowity czas T to zatem:

$$T(p) = K\left(\left[N\frac{N}{p}\right]t + [2N]\,s\right) \tag{24}$$

Mniej szczegółowe szacowanie zależności (24) przedstawiłem wcześniej, w równaniu (21).

Współczynniki t i s można otrzymać, aproksymując (24) z punktami pochodzącymi z pomiarów. Można do tego wykorzystać program komputerowy, na przykład Wolfram Mathematica.

```
K = 50;
N = 3600;
model = NonlinearModelFit[fitData, K( (N^2/p)t + 2 N s ), {t, s}, p]
```

Otrzymane współczynniki dopasowania (niepewności podane w standardowej notacji):

$$\begin{aligned} t &= 402,7(1,7)\cdot 10^{-10} \approx 0,04\mu\text{s} \\ s &= 754,9(5,5)\cdot 10^{-8} \approx 7,50\mu\text{s} \end{aligned} \tag{25}$$

Czas przesłania jednej komórki (typ *double*) w obie strony, jest około 200 razy większy od kilku operacji mnożenia i dodawania wykonanych na tej i sąsiednich komórkach. Należy dodatkowo zweryfikować wyznaczone wartości.

Dopasowana krzywa wraz z wartościami zmierzonymi przedstawiona jest na rysunku 13.

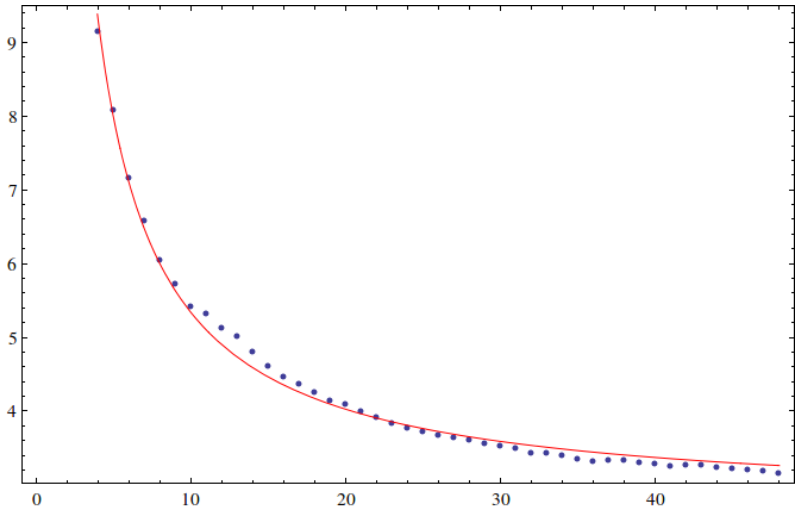

Figure 13: Krzywa dopasowana w programie Mathematica.

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.